JCSE Journal of Computing Science and Engineering

Regular Paper
Journal of Computing Science and Engineering,
Vol. 14, No. 1, March 2020, pp. 9-18

Real-Time Scheduling for Mixed-Criticality Systems in the Automotive Industry

Junghwan Lee*
Department of Battery Management System, Great Wall Motors, Hebei, China
roryjhlee@gmail.com

Myungjun Kim
Department of Computer Science, Chungbuk National University, Cheongju, Korea
mjkim@chungbuk.ac.kr

Abstract
Currently, mixed-criticality systems (MCSs) are rapidly adopted by the automotive industry, with the shift in electrical-electronic architecture from federated to integrated design to reduce developmental costs, pull in the development schedule, and easy reconfiguration of the system with service-oriented architecture. Several studies have been based on Vestal's original MCS model, in which the criticality modes are the same as criticality levels. However, the MCS model does not fit the automotive industry or the safety perspective. In this study, we identify the divergence of theory and automotive practice for real-time MCS. We also propose a generalized MCS model close to industry practice and a priority assignment algorithm along with schedulability analysis for both online and offline phases. Further, we present a practical example of memory partition and decomposition tasks based on AUTomotive Open System Architecture (AUTOSAR). The proposed design is currently being developed for battery management systems of electric and plug-in hybrid electric vehicles.

Category: Real-Time Systems

Keywords: Real-time scheduling; Mixed-criticality system; AUTOSAR

I. INTRODUCTION

The hard real-time (HRT) mixed-criticality system (MCS) is more general than the soft real-time (SRT) MCS, since architects usually do not intend to combine non-safety with safety subsystems during whole system design. Intuitively, we can imagine that a DVD player will not be integrated with a hybrid control unit, battery management system, energy management system, and motorcontrol unit (MCU), all of which are MCSs. Therefore, we can easily expect that MCSs will generally be required to be HRT rather than SRT systems. In his investigation, Vestal [1] also raised the following question: "How can we have a highly assured worst-case execution time for a piece of low-assurance software? Defects that may impact timing (e.g., infinite loops) are not assured to be absent to the degree required". This concern is the main reason why a scheduler handles an HRT MCS differently from a general HRT system. The question does not mean that a missed deadline is allowed because of an incorrectly estimated worst-case execution time (WCET). Instead, it means that tasks have different probabilities of failure according to their assurance levels, because software (SW) and hardware (HW) components

Open Access http://dx.doi.org/10.5626/JCSE.2020.14.1.9 http://jcse.kiise.org

Received 11 Feburuary 2020; **Accepted** 04 March 2020
*Corresponding Author

pISSN: 1976-4677 eISSN: 2093-8020

have different reliability and diagnostic coverage values based on the assurance levels. In fact, it is not easy to provide the rationale for a task with an additional estimated WCET related to assurance levels for a certification accessor in practice. Vestal's research and subsequent studies accommodate a missed deadline caused by a scheduling fault if the actual execution time (AET) exceeded the estimated WCET [1-10]. Unfortunately, this is not a general assumption in industry practice, since it is difficult to identify timing errors induced by incorrect WCET estimates or errors associated with the timer or scheduler, which may induce a common cause failure. Nevertheless, prior studies are invaluable and can be used in industry practice if an incorrectly estimated WCET can be separated from other defects due to timing errors. The method of setting an additional expected WCET as assurance level for tasks is similar to assurance-dependent development and assurance-dependent requirement. Prior works reported two rationales (R1 and R2) for a task carrying one more estimated WCET as follows:

(R1) A task has two extra job sets. Each job set corresponds to different critical modes according to the safety scenarios and each job set is run in different criticality modes; and

(R2) The confidence of estimated WCETs depends on the assurance level. An estimated WCET with a high assurance level is more precise than the one with a low assurance level.

For safety certification, presenting the evidence or rationale for a task with more than two WCETs is difficult, since certification assessors are not usually scheduling experts. At the least, safety and scheduling engineers, architects, and domain experts should cooperate in safety scenarios. Significant efforts are also needed to ensure that the test verifies and validates scenarios in every developmental phase, which is obviously difficult when considering development schedules, resources, and costs. Thus, a scientific method of assigning different WCETs to different safety integrity levels (SILs) is helpful for substantially reducing the required effort. Previous studies have investigated the probability of the WCET, which can be used to set the deadline tolerance for different SIL tasks [11-14]. The SIL is used in scheduling also because of the timing partition, where lower SIL tasks do not interfere with higher ones. However, when all the tasks are scheduled, the SIL inversion is not a challenge. Instead, we need to develop a method to detect timing errors, for e.g., when AET exceeds the estimated WCET, or the minimum arrival time of sporadic tasks and inter-arrival time of periodic tasks are not tolerable. The methods entail live monitoring, program flow monitoring, and execution budgets.

Scheduling errors stem from underestimated WCETs, SW modules, and random HW or systematic faults, such as miscalculations during schedulability tests. Thus, in this study, we use assurance levels and safety scenarios instead of criticality levels to avoid mixing SILs with safety scenarios.

II. THE DIVERGENCE OF THORY AND PRACTICE

A. Static Mixed-Criticality-No Monitoring

Vestal [1] 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 level and D is the lowest level. In his example, $T_1 = D_1 = 2$, $L_1 = B$, $C_{1B} = 1$, $C_{1A} = 2$ and $T_2 = D_2 = 4$, $L_2 = A$, $C_{2B} = 1$, $C_{2A} = 1$. He also mentioned that "deadline monotonic is not optimal since if task 1 is assigned the highest priority, task 2 sees a processor that already has 100% of its available level A time set aside for task 1. However, the system is feasible (as determined by the previous multi-criticality analysis algorithm) if task 2 is assigned the highest priority)" [1]. In Vestal's example, different WCETs are not set for R1; if WCETs are assumed to be set for R1, the task set is not schedulable, because the execution time of task 1 is 2 and that of task 2 is 1 in assurance level A. The total utilization exceeds 100%, which is not schedulable. Hence, we can conclude that the WCET is set for R2. In addition, tasks with a lower assurance level missing the deadline in high-criticality mode do not lead to system failure, while tasks with a lower level of assurance meeting the deadline in high-criticality mode may affect system performance in the SRT system.

Vestal [1] also stated that "for each task t_i we want to assure to level L_i that t_i never misses a deadline. This level of assurance is achieved when the analysis is based on computational times with the same level of assurance". Intuitively, we understand that measured WCETs for a task with a low level of assurance are less exact than measured WCETs for a task with a high level of assurance due to different test efforts for execution time measurement. The measured WCETs for tasks with a high level of assurance can be directly used for WCETs for both high and low assurance levels because it is the exact value that can be used in high-criticality mode. However, measured WCETs for low-assurance tasks cannot be directly used for high-assurance WCETs. A margin value is added to the original WCET for high-assurance WCET in low-assurance tasks. Hence, the difference between WCETs for low and high assurance levels in a task with a low assurance is larger than the gap between WCETs for low and high assurance levels in a high-assurance task. For example, C_{iA}, C_{iB}, C_{iC} and C_{iD} of level A assurance task t_i can be equivalent, since a WCET is measured at assurance level A. However, C_{jA} and C_{jB} of level B assurance task t_j can differ, since a WCET is measured at assurance level B, and the WCET of assurance level A is set to a more conservative value than that of assurance level B. C_{jB}, C_{jC}, and C_{jD} of level B assurance task t_j will again be equivalent. Hence, high-assurance tasks are likely to be deemed high priority, even in low-criticality mode, and thus approach criticality as priority assignments (CAPAs). Hence, if a set of level A and level B assurance tasks are not schedulable the B assurance tasks will be assigned a lower priority even though they have a smaller relative deadline. Further, if a fault from a high priority and low-criticality task occurs in a low-criticality mode without execution time monitoring, and as a result the execution time exceeds WCET, the high-criticality task may also miss the deadline. Hence, criticality inversion may result in high-criticality tasks missing deadlines.

B. Static Mixed Criticality

WCETs are set differently based on the criticality of a task, because executed jobs show different levels of criticality (as opposed to conservativity) of WCETs compared with static mixed criticality (SMC) [8]. The same priority assignment as in Audsley's approach and SMC response time analysis (RTA) are used for both SMC with no runtime monitoring (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 during execution monitoring, whereas jobs cannot be aborted in SMC-NO even if AET exceeds WCET since there is no execution monitoring. Vestal [1] designed algorithms for use in small computing power systems without execution time monitoring.

C. Adaptive Mixed Criticality

If AET is greater than $C_i(LO)$ for tasks with a low criticality level, the system criticality level is changed from low to high, and all low-criticality tasks are abandoned [8]. The main difference between adaptive mixed criticality (AMC) and SMC is that AMC drops all low-criticality tasks if AET is higher than $C_i(LO)$ for any task, while SMC only drops a task if AET is higher than $C_i(LO)$ for a low-criticality task. In the discussion regarding SMC-NO in Vestal's work, the main purpose of different WCETs is conservation of criticality. A conservative WCET can be used for a high-assurance task, which may reduce the test effort for determining the precise WCET. Vestal [1] 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 very low probability of the worst case occurring in all tasks simultaneously. Both high-assurance and low-assurance tasks run on high-criticality mode. According to Baruah et al. [8], in SMC and AMC, the main goal of different WCETs is executing different jobs according to criticality. Hence, the WCET of every job with a high assurance level may be tested, and low-assurance tasks will not be executed in high-criticality mode in AMC. However, executed jobs of high-assurance tasks may not differ between high-criticality and low-criticality modes in practice. Instead, the jobs required to run on high-criticality mode will be allocated to high-assurance tasks, and other jobs required to run on low-criticality mode will be allocated to low-assurance tasks. Baruah et al. [8] proposed RTA for the stable mode and the terms AMC-rtb and AMC-max for the changed criticality. However, generally, the mode of criticality is changed to high level for degradation or safe state of the system that is used to recalculate the deadline within the fault-tolerant time interval (FTTI) at the beginning of a timing fault in low-criticality mode in practice. AMC was extended to minimize stack size and multiple frequencies [15, 16].

D. Zero-Slack Scheduling

Zero-slack scheduling (ZSS) has been developed to resolve criticality inversion, which is the only challenge in an overload condition [17, 18]. Task t_i in ZSS consists of two WCETs for normal mode C_i^0 and critical mode C_i, and $C_i \leq C_i^0$ because the normal mode allows overload conditions, whereas the 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 a rate-monotonic (RM) or deadline-monotonic (DM) parameter. RM and DM are used for optimal fixed-priority scheduling. During runtime, admission control is required for mode switching. Admission control is used to calculate the slack time of higher criticality and lower priority tasks compared with a running task. If slack time for C_i^0 is not adequate in a task with higher criticality and lower priority, the mode is switched from normal to critical, and tasks are scheduled as CAPAs. In critical mode, the priority blocking of lower criticality tasks is increased, with C_i^0 of higher criticality tasks exhibiting a lower priority. Hence, the schedulability of ZSS is poor if the schedulability of CAPA is worse than that of DM or RM, and this disadvantage is more serious as the criticality level is increased. Another disadvantage is that the scheduling overhead is not minor since the slack time of higher criticality tasks is continuously calculated. De Niz and colleagues [17, 18] did not explain the procedure to set WCET in the original research for ZSS; however, they provided examples such as practices adopted in the automotive industry to set the task parameters. Nevertheless, it was not a general MCS model for the automotive industry since a task does not carry different criticality levels in a case scenario, because the task is generally partitioned not only for timing but also memory, as in ISO26262. ZSS has recently been extended with a dynamic budget [2, 3].

E. Earliest Deadline First-Virtual Deadline

Earliest deadline first with virtual deadline (EDF-VD) reduces the deadline for high-criticality tasks to resolve the criticality inversion challenge in an overload [4, 5]. To guarantee that high-criticality tasks meet the deadline in the offline phase, high-criticality tasks use a virtual deadline $\hat{d}_i$, which is calculated by $\hat{d}_i = xd_i$. The 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 high-criticality tasks is not used as a substitute for the original deadline. Hence, the algorithm guarantees that high-criticality tasks meet their deadlines. However, total utilization is decreased under overload, as total utilization is calculated in the low-criticality level due to a smaller deadline increase**.** Thus, the criticality is easily switched from low to a high mode. Similarly, EDF scheduling based on the demand-bound function was developed for MCS [6, 7]. These studies assumed that the relative deadlines of tasks can be freely altered if the deadlines are not beyond the true relative deadline specified by the system designer. Hence, they set the relative deadline smaller than the original relative deadline for lower-criticality mode. The effect of smaller deadline for lower-criticality mode is similar to that of the larger WCET in high-criticality mode. However, both the smaller deadline and the larger WCET decrease schedulability. In practice, adequate CPU resources are not available since computing power is related to cost. Generally, it is hard to make a feasible system even when the maximum relative deadline is specified by the system designer. The EDF-VD has recently been extended in many studies. Chen et al. [9, 10] have extended EDF-VD under the assumption that multiple-criticality modes exist and a fault from high-criticality tasks does not trigger a switch to high-criticality mode. Liu et al. [19] have extended the assumption of imprecise WCET of low-criticality tasks, and Guo et al. [20] have extended it to support a switch back from high to low-criticality mode according to the task completion rate.

III. MEMORY PARTITION AND LAYERS

A. Memory Partition

MCS require memory partitioning for freedom from interference. Lower Automotive Safety Integrity Level (ASIL) software components (SWCs) are not allowed to write data to the memory area of higher ASIL SWCs. In AUTOSAR, partitions can be made using application containers that include tasks. We provide full permission in supervisor mode to the highest SIL SWCs, which access all memory areas and registers. Other SWCs are required to set the accessible memory range in user mode. In Fig. 1, each core contains 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. Figs. 1 and 2 show tasks assigned to core 0 and core 1, respectively. The limitation of AUTOSAR is that a BSW stack is placed only on a single core. We allocated the BSW stack to core 0, as MPC5746 supports lock-step function for core 0. The number of tasks in the system application on

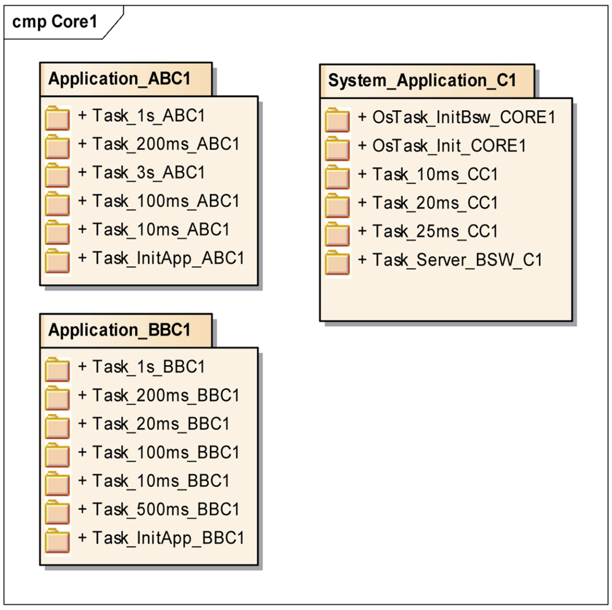


Fig. 1. Tasks on core 0.

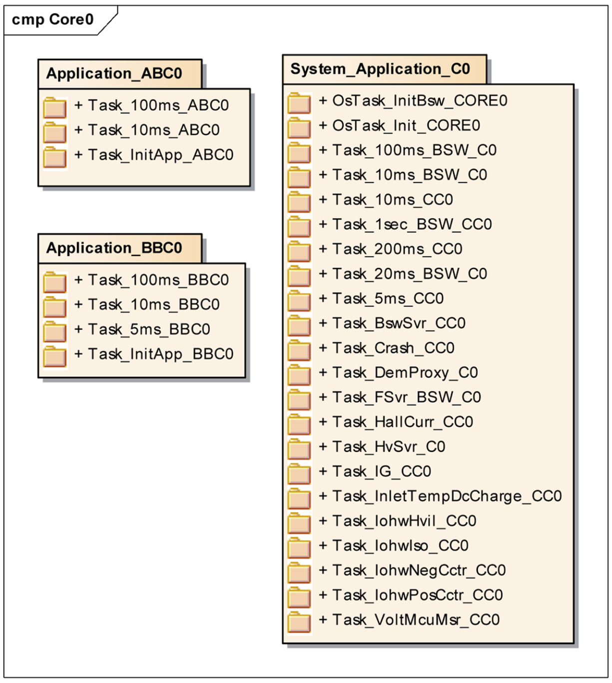


Fig. 2. Tasks on core 1.

core 0 is higher than on core 1 since all device drivers are allocated to core 0. Instead, we allocated all ASWs to core 1 except for the sensor/actuator components.

B. Layered Architecture

Fig. 3 displays four layers: application, sensor/actuator, BSW, and microcontroller abstraction (MCAL). AUTOSAR consists of application, BSW, and MCAL layers. We added a sensor/actuator (SA) layer to AUTOSAR to decouple BSW from ASW, which facilitates encapsulation of HW properties and decomposition of software components, as partitioning of BSW and MCAL is difficult, considering developmental efforts, performance, and modifiability. Thus, we allocated all device drivers and BSWs to the ASIL C partition. We minimized the complexity of the device drivers by removing logic code to control HW due to the exponential increase in the cost of development with ASIL. The development and test efforts for ASIL C are significantly greater than for ASIL A in ISO 26262. Therefore, the device drivers provide only basic HW peripheral functionality, such as analog-to-digital converters (ADC), digital input/output (DIO), and serial peripheral

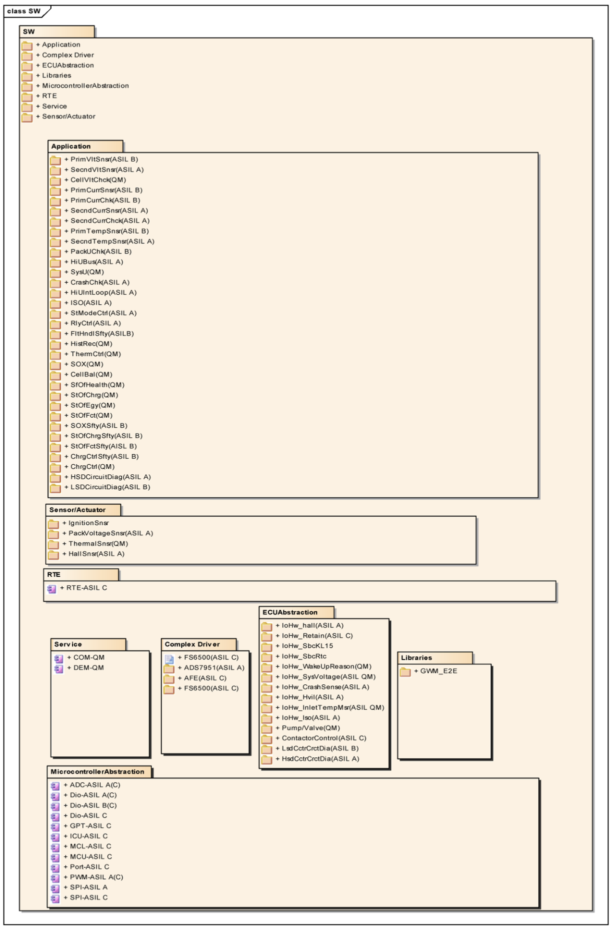


Fig. 3. Layered architecture.

interface (SPI) communication as server SWC, which is almost the same as the wrapper of MCAL. Sensor/actuator SWCs control the HW communication with BSW via client–server interface. Finally, we made the following design decisions:

Design decisions for the partitions and layers

D1. Do not create a QM partition. Instead, the QM component should be compliant with ASIL A.

D2. Create an ASIL C partition on both cores. The diagnostic function for the MCU will support both cores as ASIL C.

D3. MCAL, device drivers, and BSW stack are placed on the same partition as ASIL C.

D4. ASWs and SAs are placed on the ASIL A or ASIL B partition.

IV. DEADLINE MONOTONIC MIXEDCRITICALITY (DM²C)

A. Decomposition of Software Components

The SWCs are typically decomposed to degrade SIL, as decomposition helps to reduce developmental efforts while maintaining the safety and integrity constant. The developmental cost increases exponentially as ASIL is changed from low to high [21, 22]. Supposing that we set the WCET of tasks as QM (quality management), ASIL A, ASIL B, ASIL C, and ASIL D in ISO26262, the ASIL D task is split into four ASIL A(D) tasks. The four ASIL A(D) tasks are deemed to be part of ASIL A. In previous studies, four ASIL A(D) tasks were simultaneously dropped in admission control, causing critical system failure. Hence, we cannot drop the tasks by referring only to ASIL. Another challenge is that there is no method to ascertain whether the fault stems from an underestimated WCET. Although we dropped some tasks for schedulability, the running task set could not be scheduled if the fault was due to a random HW or other systemic fault. Therefore, we must ascertain that the fault is due to an underestimated WCET if we drop lower SIL tasks as one of the fault reactions. However, we could not identify a method of discerning WCET faults from random HW or other systemic faults. Therefore, the scheduler was assigned to the highest ASIL as a common cause of failure. Hence, we cannot provide a rationale for dropping lower criticality tasks to safety assessors during the certification process in practice. Instead, we use execution budgets or deadline monitoring to prevent higher SIL tasks from not being schedulable due to lower SIL tasks.

DEFINITION 1. In implicit criticality mode (ICM), the number of criticality modes is equal to the number of assurance levels. The number of assurance levels is the same as the number of criticality modes, and a set of tasks with an $L_i \in L$ assurance level runs in a criticality mode $M_i \in M$, with the following parameters:

A set of criticality modes: $M = \{M_1, ..., M_n\}$,

A set of assurance levels: $L = \{L_1, ..., L_n\}$.

DEFINITION 2. In explicit criticality mode (ECM), the criticality modes are not equal to the assurance levels. The number of assurance levels differs from the number of criticality modes, and a set of tasks with different assurance levels $\in L$ can run in a criticality mode $M_i \in M$, where:

A set of criticality modes: $M = \{M_1, ..., M_n\}$,

A set of assurance levels: $L = \{L_1, ..., L_m\}$.

THEOREM 1. A set of tasks cannot be feasible under any scheduling algorithm if the set of tasks includes decomposed tasks and the property of criticality mode is an ICM.

Proof. The scheduling algorithm for MCS must guarantee tasks to run in a criticality mode as assurance level of tasks. L_1 tasks must run in M_1. A high assurance level task can be decomposed to two tasks whose assurance level will be lower than original assurance. If criticality mode is switched from lower level to high level then the scheduling algorithm guarantees high assurance level tasks to run and may drop lower assurance level tasks. However, original assurance level of lower assurance level tasks is high assurance level. Hence, the scheduling algorithm cannot guarantee high assurance level tasks to run in high level criticality mode, where:

A set of criticality modes: $M = \{M_1, ..., M_n\}$

A set of assurance levels: $L = \{L_1, ..., L_m\}$

For instance, according to ISO26262, four assurance levels include *A*, *B*, *C*, and *D*. In ICM, a scheduling algorithm on MCS guarantees *A*, *B*, *C*, and *D* assurance level tasks to run in *A*, *B*, *C*, and *D* criticality modes, respectively. A *D* assurance level task can be decomposed to two *B* assurance level tasks, and a *B* assurance level task can be decomposed to *A* assurance level task. Scheduling algorithms on MCS do not guarantee scheduling of tasks lower than criticality mode. If criticality is switched from *A* to *B*, the scheduling algorithm guarantees *B* assurance level tasks to run and drop *A* assurance level tasks.

B. Priority Assignment and Response Time Analysis

We assume that MCS is an ICM,

$L = \{L_1, ..., L_n\}$,

$t = (T_i, \vec{C}_i, L, D_i)$,

$C_i(L_n) \geq C_i(L_m) \geq C_i(L_2) \geq C_i(L_1)$,

and L_n is the highest criticality mode.

Response time analysis for DM^{2}C:

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

THEOREM 2. DM^{2}C is optimal in fixed-priority scheduling for MCS.

Proof. The DM is an optimal method in fixed-priority scheduling if the system is not an MCS. If the highest criticality tasks are not schedulable with DM, then they are not schedulable with any scheduling algorithm. The algorithm removes only tasks with lower criticality than those in the current criticality mode. Hence, the remaining tasks have lower criticality compared with the current criticality mode. At each stage of the criticality mode, the assigned priority for remaining tasks in DM is still optimal. If tasks are not schedulable, the lower-criticality tasks are removed again. Hence, it is still optimal. This logic is repeated until all the priorities are completely assigned to all tasks.

Algorithm 1. DM^{2}C priority assignment

CM: Criticality mode.
$DM(A)$: Arranges priority for a task set A as the DM priority assignment.
$DF(A, B)$: Deadline-floor function to make priorities for a task set A lower than the minimum priority of a task set B.

$t^r = \emptyset$
$DM(t)$
while $RTA(t, t^r) = FALSE$
    foreach $\{t_k | \forall t_k \in t^r \wedge L(t_k) = CM\}$
        move t_k from t^r to t
    endfor
    while $RTA(t) = FALSE$
        if $\nexists t_k \in t \wedge L(t_k) < CM$ **then**
            return $\{FAIL, L_n\}$
        else
            $t_k \in t \wedge Max(D(t_k)) \wedge L(t_k) < CM$
            move t_k from t to t^r
        endif
    endwhile
    $CM \leftarrow L_{n \leftarrow n-1}$
    $DF(t^r, t)$
endwhile
return $\{SUCCESS, L_n\}$

V. SCHEDUABILITY ANALYSIS IN MCS

DEFINITION 3. $t_i \in t$, $t_i = (T_i, C_i, L_i, D_i)$ where t is a set of tasks, T_i is the period, C_i is the WCET, L_i is the criticality level, and D_i is the deadline. The WCET is based on the measurement of a single task. The test conditions are more stringent to ensure the reliability of a higher SIL. The WCET is derived from a normal distribution after the test performed in the single worst condition.

DEFINITION 4. A set of safety scenarios is $s = \{s_1, ..., s_n\}$, where s_1 is normal operation.

DEFINITION 5. A set of tasks must be run in safety scenario s_i as follows:

$$s_i \in s\ ,\ s_i = \{t_k | t_k \in t \wedge t_k\ must\ run\ in\ scenario s_i\}$$

Schedulability analysis during the design phase:

K = number of tasks in s_1,

$$\sum_{i=1 \wedge t_i \in s_1}^{n} \frac{C_i}{T_i} \leq k(2^{1/k} - 1)$$

The scheduling policy was designed using RM schedulability analysis under a normal operation scenario, and the WCET was derived from the test. The conservative WCET is set as a safety scenario. Although the system runs in normal operation, a few tasks will not run. For example, when a passive redundancy strategy is used for safety, the task runs only in a few high-criticality modes. In the ORTA for MCSs, the WCET is updated to AET if the AET over the WCET is detected by an execution budget or deadline monitoring during runtime. The system runs accurately if the ORTA is passed even though the AET is over the WCET, as this guarantees that all tasks are still schedulable under all safety scenarios. If

Table 1. Execution time in normal operation

Range	Total (sec)	Min (μs)	Max (μs)	Avg (μs)	Ratio (%)
Task_IohwLowSide_CORE0:0	179.875	58.287	685.636	282.300	2.82
OsTask_10ms_BSW_CORE0:0	673.740	0.018	2,502.000	137.575	10.57
OsTask_10ms_ASW_CORE0:0	51.335	0.018	335.502	80.566	0.81
OsTask_25ms_BSW_CORE0:0	1,894.000	0.018	3,804.000	314.944	29.74
IdleTask_OsCore_CORE0:0	1,383.000	0.022	4,598.000	304.066	21.71
Task_IohwHighSide_CORE0:0	88.051	41.200	316.640	138.189	1.38
OsTask_IohwServer_CORE0:0	887.573	0.018	889.138	149.756	13.93
OsTask_IohwSbc_CORE0:0	394.048	0.042	862.762	151.208	6.18
OsTask_5ms_BSW_CORE0:0	407.850	0.018	1,111.000	320.044	6.40
OsTask_5ms_ASW_CORE0:0	319.518	0.018	986.213	100.254	5.01
OsTask_20ms_BSW_CORE0:0	33.700	0.018	576.338	105.780	0.53
Task_100ms_ASILB_CORE0:0	5.847	0.018	650.951	53.141	0.09
Task_100ms_ASILA_CORE0:0	6.046	0.018	655.253	54.856	0.09
OsTask_100ms_ASW_CORE0:0	6.094	0.018	670.996	55.049	0.10
OsTask_NvmServer_CORE0:0	18.206	85.727	455.760	285.738	0.29
OsTask_IohwIso_CORE0:0	18.178	0.049	376.989	142.712	0.29
OsTask_200ms_ASW_CORE0:0	3.677	0.018	635.218	115.383	0.06
IdleTask_OsCore_CORE1:1	4,787.000	0.022	6,128.000	466.643	75.12
OsTask_10ms_ASW_CORE1:1	1,278.000	0.018	954.458	125.382	20.06
OsTask_10ms_BSW_CORE1:1	40.069	0.018	982.471	62.885	0.63
OsTask_20ms_ASW_CORE1:1	9.918	0.022	92.667	31.131	0.16
OsTask_100ms_ASW_CORE1:1	249.198	0.024	1,088.000	163.184	3.91
Task_100ms_ASILA_CORE1:1	3.600	0.018	254.142	56.507	0.06
OsTask_200ms_ASW_CORE1:1	2.649	0.018	253.762	83.149	0.04
Task_500ms_ASILB_CORE1:1	1.129	0.018	253.565	88.618	0.02

Total = total running time, Min = minimum execution time, Max = maximum execution time, Avg = average execution time, Ratio = running ratio of each task.

ORTA fails in a specific scenario, a reaction strategy can be developed or the system can be directly operated in a schedulable safety scenario. ORTA can be used in the production, verification, and validation phases to ensure incremental safety of the system by determining the optimal schedulable scenario and WCET. The AETs of tasks vary in each scenario, and the WCET of each task is derived from the AET in the verification and validation phases. Mixed-criticality systems are typically safety-criticality systems that require reliable execution of specific tasks rather than execution of multiple tasks. Further, actual scheduling is more complex since the AET differs in each operation scenario, as shown in Tables 1 and 2, which present measurements under three operational scenarios with the initial software version for sample A. For a more practical approach, we consider the operation and safety scenarios to set more precise WCETs.

Algorithm 2. Online response time analysis (ORTA)

E_m: $Actual\ execution\ time\ of\ t_m$
FS: $A\ set\ of\ scenarios\ failed\ in\ RTA$
if ($E_m > C_m$), **then**
  $update\ E_m \rightarrow C_m$
  foreach $s_k \in s$
    if $\exists t_m \in s_k$ **then**
      foreach $t_i \in s_k$
        $R_i = C_i + \sum_{t_j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$
        if $R_i > D_i$, **then**
          $FS \leftarrow s_k$
        break
    endif
      endfor
    endif
  endfor
endif

Table 2. Execution time in a plug-out high-power connector during runtime

Range	Total (ms)	Min (μs)	Max (μs)	Avg (μs)	Ratio (%)
IdleTask_OsCore_CORE0:0	65,010.000	0.022	4,598.000	307.086	18.67
task_IohwHighSide_CORE0:0	4.598.000	53.431	302.711	132.095	1.32
OsTask_IohwServer_CORE0:0	47,742.000	0.042	821.440	147.733	13.71
OsTask_IohwSbc_CORE0:0	20,475.000	0.042	543.333	143.692	5.88
OsTask_5ms_BSW_CORE0:0	18,783.000	0.018	1,051.000	269.782	5.40
OsTask_10ms_BSW_CORE0:0	36,929.000	0.018	2,928.000	144.831	10.61
OsTask_5ms_ASW_CORE0:0	16,770.000	0.018	979.093	92.381	4.82
OsTask_25ms_BSW_CORE0:0	121,454.000	0.018	3,871.000	325.687	34.89
Task_IohwLowSide_CORE0:0	9,654.000	58.464	665.951	277.324	2.77
OsTask_10ms_ASW_CORE0:0	2,222.000	0.018	329.836	63.839	0.64
OsTask_20ms_BSW_CORE0:0	1,312.000	0.018	575.458	75.353	0.38
OsTask_NvmServer_CORE0:0	1,424.000	0.042	608.293	409.193	0.41
OsTask_IohwIso_CORE0:0	859.504	0.049	213.442	123.456	0.25
OsTask_100ms_ASW_CORE0:0	235.431	0.018	668.796	66.356	0.07
Task_100ms_ASILB_CORE0:0	224.857	0.018	668.511	63.162	0.06
Task_100ms_ASILA_CORE0:0	213.366	0.018	668.631	59.934	0.06
OsTask_200ms_ASW_CORE0:0	224.431	0.018	650.373	112.779	0.06
OsTask_10ms_ASW_CORE1:1	70,569.000	0.036	781.093	126.701	20.27
IdleTask_OsCore_CORE1:1	263,999.000	0.022	6,164.000	471.044	75.83
OsTask_10ms_BSW_CORE1:1	2,305.000	0.018	965.067	66.208	0.66
OsTask_20ms_ASW_CORE1:1	509.797	0.022	92.307	29.290	0.15
Task_100ms_ASILA_CORE1:1	186.407	0.018	269.805	53.550	0.05
OsTask_100ms_ASW_CORE1:1	10,336.000	0.024	490.727	123.714	2.97
OsTask_200ms_ASW_CORE1:1	164.948	0.018	273.107	94.743	0.05
Task_500ms_ASILB_CORE1:1	60.003	0.018	253.831	86.211	0.02

Total = total running time, Min = minimum execution time, Max = maximum execution time, Avg = average execution time, Ratio = running ratio of each task.

VI. CONCLUSIONS

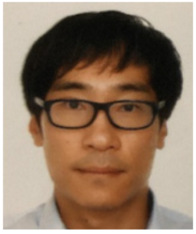
We presented the divergence of theory and practice for a closer automotive industry perspective. In addition, we proposed a priority assignment algorithm assuming ICM MCS for offline phases and a schedulability analysis for both online and offline phases assuming ECM MCS. The offline algorithm can be used in the verification and validation phases to automatically determine the WCET, and thereby reduce the test efforts.

AKNOWLEDGMENTS

This work was supported by Great Wall Motors under a BMS platform project on the Electric Vehicle and Plug-In Hybrid.

REFERENCES

1. S. Vestal, “Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance,” in *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, Tucson, AZ, 2007, pp. 239-243.
2. X. Gu and A. Easwaran, “Dynamic budget management with service guarantees for mixed-criticality systems,” in *Proceedings of 2016 IEEE Real-Time Systems Symposium (RTSS)*, Porto, Portugal, 2016, pp. 47-56.
3. X. Gu and A. Easwaran, “Dynamic budget management and budget reclamation for mixed-criticality systems,” *Real-Time Systems*, vol. 55, pp. 552-597, 2019.
4. S. Baruah, V. Bonifaci, G. d’Angelo, A. Marchetti-Spaccamela, S. van der Ster, and L. Stougie, “Mixed-criticality scheduling of sporadic task systems,” in *Algorithms - ESA 2011*. Heidelberg: Springer, 2011, pp. 555-566.
5. S. Baruah, V. Bonifaci, G. d’Angelo, H. Li, A. Marchetti-Spaccamela, S. Van Der Ster, and L. Stougie, “Preemptive uniprocessor scheduling of mixed-criticality sporadic task systems,” *Journal of the ACM*, vol. 62, no. 2, article no. 14, 2015.
6. P. Ekberg and W. Yi, “Bounding and Shaping the Demand of Mixed-Criticality Sporadic Tasks,” in *Proceedings of the 24th Euromicro Conference on Real-Time Systems*, Pisa, Italy, 2012, pp. 135-144, 2012.
7. P. Ekberg and W. Yi, “Bounding and shaping the demand of generalized mixed-criticality sporadic task systems,” *Real-Time Systems*, vol. 50, pp. 48-86, 2014.
8. S. Baruah, A. Burns, and R. I. Davis, “Response-time analysis for mixed criticality systems,” in *Proceedings of 2011 IEEE 32nd Real-Time Systems Symposium (RTSS)*, Vienna, Austria, 2011, pp. 34-43.
9. G. Chen, N. Guan, B. Hu, and W. Yi, “EDF-VD scheduling of flexible mixed-criticality system with multiple-shot transitions, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2393-2403, 2018.
10. G. Chen, N. Guan, D. Liu, Q. He, K. Huang, T. Stefanov, and W. Yi, “Utilization-based scheduling of flexible mixed-criticality real-time tasks,” *IEEE Transactions on Computers*, vol. 67, no. 4, pp. 543-558, 2017.
11. L. Kosmidis, E. Quinones, J. Abella, T. Vardanega, I. Broster, and F. J. Cazorla, “Measurement-based probabilistic timing analysis and its impact on processor architecture,” in *Proceedings of 2014 17th Euromicro Conference on Digital System Design*, Verona, Italy, 2014, pp. 401-410.
12. L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzetti, E. Quinones, and F. J. Cazorla, “Measurement-based probabilistic timing analysis for multi-path programs,” in *Proceedings of 2012 24th Euromicro Conference on Real-Time Systems,* Pisa, Italy, 2012, pp. 91-101.
13. C. Chen, J. Panerati, I. Hafnaoui, and G. Beltrame, “Static probabilistic timing analysis with a permanent fault detection mechanism,” in *Proceedings of 2017 12th IEEE International Symposium on Industrial Embedded Systems (SIES)*, Toulouse, France, 2017, pp. 1-10.
14. L. Santinelli, F. Guet, and J. Morio, “Revising measurement-based probabilistic timing analysis,” in *Proceedings of 2017 IEEE Real-Time Embedded Technology and Applications Symposium (RTAS)*, Pittsburg, PA, 2017, pp. 199-208.
15. Q. Zhao, Z. Gu, H. Zeng, and N. Zheng, “Schedulability analysis and stack size minimization with preemption thresholds and mixed-criticality scheduling,” *Journal of Systems Architecture*, vol. 83, pp. 57-74, 2018.
16. S. Baruah, “Schedulability analysis of mixed-criticality systems with multiple frequency specifications,” in *Proceedings of 2016 International Conference on Embedded Software (EMSOFT)*, Pittsburgh, PA, 2016, pp. 1-10.
17. D. de Niz and L. T. Phan, “Partitioned scheduling of multi-modal mixed-criticality real-time systems on multiprocessor platforms,” in *Proceedings of 2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Berlin, Germany, 2014, pp. 111-122.
18. D. de Niz, L. Wrage, A. Rowe, and R. Rajkumar, “Utility-based resource overbooking for cyber-physical systems,” in *Proceedings of 2013 IEEE 19th International Conference on Embedded and Real-Time Computing Systems and Applications*, Taipei, Taiwan, 2013, pp. 217-226.
19. D. Liu, N, Guan, J. Spasic, G. Chen, S. Liu, T. Stefanov, and W. Yi, “Scheduling analysis of imprecise mixed-criticality real-time tasks,” *IEEE Transactions on Computers*, vol. 67, no. 7, pp. 975-991, 2018.
20. Z. Guo, K. Yang, S. Vaidhun, S. Arefin, S. K. Das, and H. Xiong, “Uniprocessor mixed-criticality scheduling with graceful degradation by completion rate,” in *Proceedings of 2018 IEEE Real-Time Systems Symposium (RTSS)*, Nashville, TN, 2018, pp. 373-383.
21. A. Frigerio, B. Vermeulen, and K. Goossens, “A generic method for a bottom-up ASIL decomposition,” in *Computer Safety, Reliability, and Security*. Cham: Springer, 2018, pp. 12-26.
22. A. Frigerio, B. Vermeulen, and K. Goossens, “Component-level ASIL decomposition for automotive architectures,” in *Proceedings of 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, Portland, OR, 2019, pp. 62-69.



Junghwan Lee https://orcid.org/0000-0001-7728-9488

Junghwan Lee received the M.S. degree in computer science from Chungbuk National University, Cheongju, South Korea in 2001. He is currently working in the Department of Battery Management System of Great Wall Motors, Hebei, China, as an architect and pursuing a 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 https://orcid.org/0000-0002-9651-6161

Myungjun Kim received the M.S. degree in computer science from 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 a professor in the Department of Computer Science at Chungbuk National University, Cheongju, South Korea. His research interests include real-time systems and distributed computing systems.