Information Processing Letters 116 (2016) 508–512

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

Generalizing fixed-priority scheduling for better schedulability in mixed-criticality systems

Yao Chen [a,*], Kang G. Shin [b], Huagang Xiong [a]

[a] School of Electronic and Information Engineering, Beihang University, Beijing, 100191, China
[b] Real-Time Computing Laboratory, EECS, The University of Michigan, Ann Arbor, MI 48109, USA

ARTICLE INFO

Article history:
Received 2 August 2015
Received in revised form 15 February 2016
Accepted 15 February 2016
Available online 27 February 2016
Communicated by Nathan Fisher

Keywords:
Real-time systems
Mixed-criticality
Fixed-priority scheduling
Schedulability analysis
Priority assignment algorithm

ABSTRACT

The design of mixed-criticality systems is often subject to mandatory certification and has been drawing considerable attention over the past few years. This letter studies fixed-priority scheduling of mixed-criticality systems on a uniprocessor platform but in a more general way, using different priority orderings in different execution phases and considering them collectively. Then a sufficient response-time analysis is developed and a new priority assignment scheme is proposed. This generalized approach has potential in better schedulability performance for mixed-criticality systems.

1. Introduction

There has been an increasing trend of integrating multiple functionalities of different criticality levels upon a shared hardware platform to address the growing demand for computing power cost-efficiently in safety-critical real-time systems. When certifying such mixed-criticality (MC) systems, the certification authorities and manufacturers mandate different assumptions about the worst-case system behavior, depending on the criticality of concerned functionalities. To simultaneously guarantee temporal correctness at all different levels of assurance, scheduling issues arising from these multiple certification requirements have been studied extensively [1–4].

As a preferred approach in industry due to its flexibility and ease of predictability, fixed-priority (FP) preemptive scheduling was firstly introduced into the MC scenario in Vestal's seminal work [1]. Schedulability analysis based on response-time is presented there and then improved by Baruah [2]. The Adaptive Mixed Criticality (AMC) scheme in [2] has been shown to be one of the most effective MC scheduling approaches and forms the basis of further related studies [5–7]. Note the above studies share a common assumption that enforces the same task priority ordering throughout the system's life.

As in general multi-mode systems [8], enabling change of priorities in the event of a mode-change has been studied in MC systems [3,9]. However, one characteristic of these schemes is that they do not distinguish the phase of mode transition from the steady new mode. Besides, the Priority May Change (PMC) approach [9] deals with behaviors of different criticality levels individually, ignoring the dependency. In this letter, the execution model is further relaxed that priorities can be re-assigned not only in the event of mode change but also when the mode transition ends. Based on this, we investigate FP scheduling of MC systems by considering different execution phases collectively.

* Corresponding author.
E-mail addresses: chenyao.kevin@gmail.com (Y. Chen), kgshin@umich.edu (K.G. Shin), hgxiong@buaa.edu.cn (H. Xiong).

http://dx.doi.org/10.1016/j.ipl.2016.02.009
0020-0190/

2. System model

Task model: An MC sporadic task τ_i is characterized by a four-tuple $\tau_i = (T_i, D_i, \xi_i, \vec{C}_i)$, where T_i denotes its period, D_i the relative deadline, ξ_i the criticality level, and $\vec{C}_i$ a vector of worst case execution time (WCET) estimations. In this letter constrained deadline is assumed and our attention will be restricted to dual-criticality systems, but the main principle can be scaled to an increased number of criticality levels with further efforts. Formally, for task τ_i, we assume $\xi_i \in \{LC, HC\}$ and $\vec{C}_i = \{C_i^L, C_i^H\}$ with $C_i^L \leq C_i^H$, where C_i^L (C_i^H) denotes low-criticality (*LC*) (high-criticality (*HC*)) WCET.

Certification requirements: Consider a system $\Gamma = \{\tau_i | 1 \leq i \leq n\}$ consisting of a set of independent MC tasks. During different runs the system could show different behaviors generally and its mandated temporal correctness differs depending on the concerned criticality level. For such MC system to be certified correct, the following timing requirements should be guaranteed:

- No job of any task τ_i can execute for more than $C_i^{\xi_i}$, otherwise the system is exhibiting *erroneous behavior*.
- As long as no job executes for more than C_i^L, the system is regarded as exhibiting *LC behavior*, and all jobs should meet their deadlines.
- If some *HC* job executes for C_i^L without completion, the system begins to exhibit *HC behavior* and from this instant of criticality change only *HC* jobs are required to meet their deadlines.

The *HC* jobs that are active (released but not yet completed) upon occurrence of the criticality change are referred as *carry-over jobs* [4]. Recall that *LC* jobs are not required to complete by their deadlines for *HC* system behaviors, which is an implication of the certification requirements. This strictness was then relaxed by Santy [10] which allowed *LC* tasks to execute after the criticality change and the system to change back to *LC* mode. And more related schemes can be found in a recent survey [11].

Scheduling strategy: As a special case of multi-mode systems [8], dual-criticality systems could go through three distinct phases: steady *LC* mode, mode transition period and steady *HC* mode. The mode transition period represents the time interval between the criticality mode change and the instant when all carry-over jobs have completed their execution. To favor accommodating the change of system load upon occurrence of the criticality change, in this letter we prefer to use different priority orderings in different phases and introduce a new strategy for task dispatching, called Generalized Fixed-Priority (GFP). Specially, each task τ_i has three unique-priority parameters:

- P_i^L: priority for jobs executed in the steady *LC* mode;
- P_i^T: priority for the carry-over job when the system exhibits *HC* behavior;
- P_i^H: priority for jobs released after the criticality change.

Starting from 1, assume the larger value represents the higher priority. Initially, the system starts in the steady *LC* mode and the scheduler selects the highest priority job for execution according to the ordering of P_i^L. When the criticality change occurs, the system switches to the mode transition period, *LC* jobs are discarded and *HC* jobs are scheduled according to the ordering of P_i^T and P_i^H, which may interleave with each other for different tasks, for example $P_i^T > P_j^T > P_j^H > P_i^H$. Finally in the steady *HC* mode, the scheduler selects the highest priority job according to the ordering of P_i^H.

Note that P_i^T is exclusive for the carry-over job, straddling the criticality change. The advantage is to mitigate the problem that some carry-over job may execute late in the steady *LC* mode and thus has to complete its remaining *HC* execution in a very short scheduling window after the criticality change. Since each job must have completed execution before the next release under the assumption of constrained deadline, we have $P_i^T > P_i^H$ for *HC* tasks. As for *LC* tasks, since they are prevented from executing after the criticality change, their priorities P_i^T and P_i^H are useless and ignored here.

3. Response time analysis

In this section a sufficient schedulability test is derived based on the same analysis framework as in [2,6], where response times in three distinct scenarios are studied.

3.1. Jobs finished in the steady LC mode

MC tasks behave exactly the same as traditional (non-MC) ones in the steady *LC* mode. Thus, the standard RTA method can be applied to derive τ_i's response time R_i^L:

$$R_i^L \leftarrow C_i^L + \sum\nolimits_{\forall \tau_k \in hp_L(i)} \lceil R_i^L / T_k \rceil C_k^L \tag{1}$$

where $hp_L(i) = \{\tau_k \in \Gamma | P_k^L > P_i^L\}$ denotes the set of tasks with higher priority than that of τ_i in the steady *LC* mode.

3.2. Carry-over job

Consider the carry-over job J_i^p in Fig. 1, released before the criticality change with span $s \in [0, R_i^L]$. For convenience of presentation, define task subset $hp_T^{xyz}(i)$ as

$$\{\tau_k \in \Gamma \backslash \tau_i | f(P_k^L, P_i^L) = x, f(P_k^T, P_i^T) = y, \\ f(P_k^H, P_i^T) = z\}$$

where x, y, z are binary variables, $f(u, v)$ is a binary function returning 1 if $u \geq v$, otherwise 0. Depending on the priority orderings, there are five possible situations for τ_k to interferer with J_i^p within the busy period $[t_0, f_i^p]$: $\tau_k \in hp_T^{xyz}(i)$ with $xyz \in \Omega = \{111, 110, 100, 011, 010\}$. Specially, $x = 1$ means τ_k has higher priority than τ_i in the steady *LC* mode and similar interpretation applies to y and z. Note that $hp_T^{000}(i)$ cannot interfere with τ_i and combinations $\{001, 101\}$ are ruled out due to constraint $P_k^T > P_k^H$.

3.2.1. Interference calculation

For each situation $xyz \in \Omega$, two kinds of interference of the higher priority task $\tau_k \in hp_T^{xyz}(i)$ are calculated:

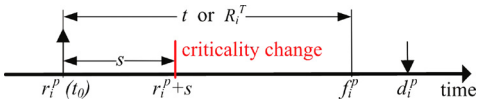

Fig. 1. The carry-over job.

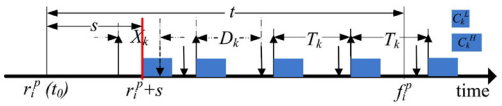

Fig. 2. Worst case pattern for task $\tau_k \in hp_T^{011}(i)$.

$I_{k,i}^{xyz}(s,t)$, an upper bound of the total interference within the entire busy period $[t_0, f_i^p]$; $Is_{k,i}^{xyz}(s,t)$, a lower bound of the interference within the front subinterval $[t_0, r_i^p + s]$ out of the total interference $I_{k,i}^{xyz}(s,t)$. Due to space limitations, we just show some intuition for their computation here and present detailed discussions in the supplementary file [13].

1) $\tau_k \in hp_T^{111}(i)$: τ_k always has higher priority than J_i^p and a sufficient value of its total interference has been derived in [2]. We reproduce it here with few modification:

$$I_{k,i}^{111}(s,t) = \min\left(\lceil t/T_k \rceil, \lceil (t-s+X_k)/T_k \rceil\right)(C_k^H - C_k^L) + \lceil t/T_k \rceil C_k^L \quad (2)$$

where X_k ($R_k^L \le X_k \le \zeta_k$) denotes a tighter relative deadline for *HC* task in the steady *LC* mode, with $\zeta_k = D_k - (C_k^H - C_k^L)$.

Disregarding the execution that the carry-over job may have finished in the *LC* mode, a safe lower bound for the interference within $[t_0, r_i^p + s]$ can be derived.

$$Is_{k,i}^{111}(s,t) = \max\left\{0, \lceil t/T_k \rceil - \lceil (t-s+X_k)/T_k \rceil\right\} C_k^L \quad (3)$$

2) $\tau_k \in hp_T^{110}(i)$: Jobs released within $[t_0, r_i^p + s]$ can interfere with J_i^p and the maximum number of such releases is $\lfloor s/T_k \rfloor + 1$. Among these, the carry-over job always has higher priority than J_i^p and can execute up to C_k^H while each of the others can execute up to C_k^L before the criticality change. Thus τ_k's total interference is bounded by

$$I_{k,i}^{110}(s,t) = \lfloor s/T_k \rfloor C_k^L + C_k^H \quad (4)$$

Given the carry-over job may start execution after the criticality change, we have

$$Is_{k,i}^{110}(s,t) = \lfloor s/T_k \rfloor C_k^L \quad (5)$$

3) $\tau_k \in hp_T^{100}(i)$: τ_k can only preempt J_i^p within $[t_0, r_i^p + s]$, where its interference is equivalent to the execution of traditional task with parameters (T_k, D_k, C_k^L).

$$I_{k,i}^{100}(s,t) = \lfloor s/T_k \rfloor C_k^L + \min\left(C_k^L, s - \lfloor s/T_k \rfloor T_k\right) \text{ and } Is_{k,i}^{100}(s,t) = I_{k,i}^{100}(s,t) \quad (6)$$

4) $\tau_k \in hp_T^{011}(i)$: τ_k can only interfere with J_i^p after the criticality change within subinterval $[r_i^p + s, r_i^p + t]$. Fig. 2 shows an execution pattern of τ_k, where its first job is released $(X_k - C_k^L)$ before the criticality change and starts execution after the criticality change while all other jobs followed are released and executed as early as possible. One can verify that any leftward or rightward shift of the interval $[r_i^p + s, r_i^p + t]$ within the pattern of Fig. 2 does not increase the amount of τ_k's execution. Therefore, Fig. 2 depicts the worst-case scenario for τ_k to maximize its execution. Denote $w = t - s + X_k - C_k^L$ and then we have:

$$I_{k,i}^{011}(s,t) = \lfloor w/T_k \rfloor C_k^H + \min\left(C_k^H, w - \lfloor w/T_k \rfloor T_k\right) \text{ and } Is_{k,i}^{011}(s,t) = 0 \quad (7)$$

5) $\tau_k \in hp_T^{010}(i)$: Only the carry-over job of τ_k can be executed within the concerned busy period and this happens after the criticality change.

$$I_{k,i}^{010}(s,t) = C_k^H \text{ and } Is_{k,i}^{010}(s,t) = 0 \quad (8)$$

3.2.2. The response time test

Once the interference of higher-priority tasks is derived, the response time $R_i^T(s)$ for some given s can be formally constructed:

$$\begin{aligned} R_i^T(s) \leftarrow\ & C_i^H + \min\Big(s, \sum\nolimits_{\forall \tau_k \in hp_T(i)} Is_{k,i}^{xyz}(s, R_i^T(s))\Big) \\ & + \sum\nolimits_{\forall \tau_k \in hp_T(i)} \left(I_{k,i}^{xyz}(s, R_i^T(s)) - Is_{k,i}^{xyz}(s, R_i^T(s))\right) \end{aligned} \quad (9)$$

where $hp_T(i) = \bigcup_{\forall xyz \in \Omega} hp_T^{xyz}(i)$ denotes tasks having higher priority than τ_i during at least one of the three phases. The min function is introduced into the standard RTA framework to reduce pessimism of the analysis, which bounds the total interference within the subinterval $[t_0, r_i^p + s]$ by the subinterval length s.

By solving Eq. (9) for every possible s and taking the maximum obtained, we have:

$$R_i^T = \max_{0 \le s \le R_i^L} R_i^T(s) \quad (10)$$

3.3. Jobs released after the criticality change

Consider the scenario in Fig. 3, where J_i^p is released immediately after the criticality change. Note J_i^p can be preempted by carry-over jobs, which may get preempted by other tasks in the steady *LC* mode. Let $\hat{P}_i^L = \min\{P_k^L | \forall \tau_k \in \Gamma \backslash \tau_i, P_k^T > P_i^H\}$. To address both direct and indirect interferences, we extend the beginning of the busy period from r_i^p to an earlier time instant t_0, such that at any instant $t \in [t_0, r_i^p)$ the processor is busy executing tasks with priority higher or equal to $\hat{P}_i^L$. Define task subset $hp_H^{xyz}(i)$ as

$$\{\tau_k \in \Gamma \backslash \tau_i | f(P_k^L, \hat{P}_i^L) = x, \ f(P_k^T, P_i^H) = y, \ f(P_k^H, P_i^H) = z\}$$

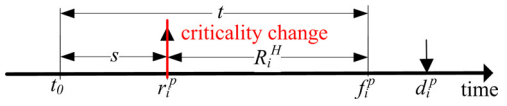

Fig. 3. The job released after criticality change.

According to the definition of $\hat{P}_i^L$, the task with direct interference must belong to either $hp_H^{111}(i)$ or $hp_H^{110}(i)$, while the task with indirect interference must have higher priority in the *LC* mode than some carry-over job with direct interference and belongs to $hp_H^{100}(i)$. In summary, three possible situations exist for tasks to execute within the extended busy period and denote $hp_H(i) = \bigcup_{\forall xyz \in \Lambda} hp_H^{xyz}(i)$ with $\Lambda = \{111, 110, 100\}$. It's worth noting the interference terms $I_{k,i}^{xyz}(s,t)$ and $Is_{k,i}^{xyz}(s,t)$ derived in Section 3.2 also apply here, except that $\hat{P}_i^L$ is used instead of P_i^L. Thus, $R_i^H(s)$ can be obtained by

$$\begin{aligned} & R_i^H(s) + s \\ & \leftarrow C_i^H + \min\Big(s, \sum\nolimits_{\forall \tau_k \in hp_H(i)} Is_{k,i}^{xyz}(s, R_i^H(s)+s)\Big) \\ & \quad + \sum\nolimits_{\forall \tau_k \in hp_H(i)} \big(I_{k,i}^{xyz}(s, R_i^H(s)+s) \\ & \quad - Is_{k,i}^{xyz}(s, R_i^H(s)+s)\big) \end{aligned} \tag{11}$$

From the above assumption of busy period extension, we can achieve a bound for all possible values of s that need to be checked, denoted as Θ_i, by employing the standard RTA method to $hp_H(i)$ in the *LC* mode with initial iterative value $\sum_{\forall \tau_k \in hp_H(i)} C_k^L$:

$$\Theta_i \leftarrow \sum\nolimits_{\forall \tau_k \in hp_H(i)} \lceil \Theta_k / T_k \rceil C_k^L \tag{12}$$

Then R_i^H can be obtained by taking the maximum value $R_i^H(s)$ for all possible s.

$$R_i^H = \max_{0 \le s \le \Theta_i} R_i^H(s) \tag{13}$$

Moreover, if J_i^p is released φ after the criticality change but still during the mode transition period, its response time will reduce by φ since the busy period length $R_i^H(s)+s$ remains unchanged for fixed s. And if φ is large enough such that J_i^p is released in the steady *HC* mode, then the response time according to the standard RTA is exactly the case here but with $s = 0$. Therefore, we can conclude that Fig. 3 depicts the worst case situation for jobs released after the criticality change.

4. Priority assignment

As for the priority assignment, another dimension of FP scheduling, Audsley's Optimal Priority Assignment (OPA) algorithm [12], which was originally designed for traditional tasks, has been adapted to MC scenarios [2,6,9]. In this section, we exploit the idea behind Audsley's OPA and propose a new scheme called Heuristic Priority Assignment (HPA), with details summarized in Algorithm 1.

Initially, only one priority ordering is assumed and the OPA is employed to find feasible solutions (lines 2–5). If this fails, meaning no task can be assigned the current lowest priorities in all three phases, then different priority orderings are assumed and the ordering of P_i^L is considered first. In lines 6–7, the HPA identifies the *HC* task with the minimal ratio R_i^L/ζ_i among all candidates deemed schedulable in the steady *LC* mode and assigns it with the current lowest *LC* priority *pl*. The idea behind this decision is that smaller *LC* response time relative to the deadline makes the task more likely to be schedulable upon occurrence of the criticality change. Once the complete ordering of P_i^L is determined, the HPA continues with P_i^T and P_i^H (lines 12–20), starting with P_i^H due to the constraint $P_i^T > P_i^H$. It returns "unschedulable" if no *HC* task can be assigned the current lowest priority level, neither for *LC* (line 9) nor *HC* (line 18) behavior.

Algorithm 1 Heuristic priority assignment.

1: $\Gamma_U = \Gamma$, $\Gamma_U^H = \Gamma^H$ and $Ph = 1$;
2: **for** each priority level *pl*, from 1 to $\|\Gamma\|$ **do**
3: **if** there exist task $\tau_i \in \Gamma_U$ deemed always schedulable with the current lowest priorities **then**
4: Assign $P_i^L = pl$, $\Gamma_U = (\Gamma_U \backslash \tau_i)$;
5: Assign $P_i^T = Ph + 1$, $P_i^H = Ph$, $Ph = Ph + 2$, $\Gamma_U^H = (\Gamma_U^H \backslash \tau_i)$ when $L_i = HC$;
6: **else if** there exist set of tasks $\Delta \subseteq \Gamma_U^H$, each of which is deemed schedulable in the steady *LC* mode ($R_i^L \le \zeta_i$) when assigned the current lowest *LC* priority ($P_i^L = pl$) **then**
7: Find $\tau_i \in \Delta$ s.t. $R_i^L/\zeta_i = \min\{R_k^L/\zeta_k | \forall \tau_k \in \Delta\}$ and assign $P_i^L = pl$, $\Gamma_U = (\Gamma_U \backslash \tau_i)$;
8: **else**
9: **return** unschedulable;
10: **end if**
11: **end for**
12: **for** each priority level *ph*, from *Ph* to $2\|\Gamma^H\|$ **do**
13: **if** there exist task $\tau_i \in \Gamma_U^H$ with P_i^H unassigned but deemed schedulable in the steady *HC* mode ($R_i^H \le D_i$) when assigned the current lowest *HC* priority ($P_i^H = ph$) **then**
14: Assign $P_i^H = ph$;
15: **else if** there exist task $\tau_i \in \Gamma_U^H$ with P_i^T unassigned but deemed schedulable in the mode transition period ($R_i^T \le D_i$) when assigned the current lowest *HC* priority ($P_i^T = ph$) **then**
16: Assign $P_i^T = ph$, $\Gamma_U^H = (\Gamma_U^H \backslash \tau_i)$;
17: **else**
18: **return** unschedulable;
19: **end if**
20: **end for**
21: **return** schedulable;

5. Illustrative example and evaluation

Although the above achievements are made under the assumption of three priority orderings, they apply as well if we consider the case of two priority orderings (one for *LC* behaviors and the other for *HC* behaviors) or set further constraint by enforcing merely one priority ordering like common assumption. For the latter, the HPA reduces to the OPA and only interference terms from $hp_H^{100}(i) \bigcup hp_H^{111}(i)$ need to be addressed for schedulability analysis, which is the same as Baruah's AMC-max scheme [2].

Consider the MC task system $\Gamma = \{\tau_i | 1 \le i \le 3\}$ in Table 1, scheduled on a uniprocessor with FP. It can be verified that none of the three tasks can be assigned the lowest priority throughout the system's life. In other words, the system is deemed as unschedulable according to the original OPA. However, the HPA can find feasible prior-

Table 1
Example of priority assignment.

Task	ξ	T	D	C^L	C^H
τ_1	LC	80	56	34	34
τ_2	HC	66	60	22	44
τ_3	HC	76	75	8	16

Task	P^L	P^T	P^H
τ_1	2	0	0
τ_2	3	3	2
τ_3	1	4	1

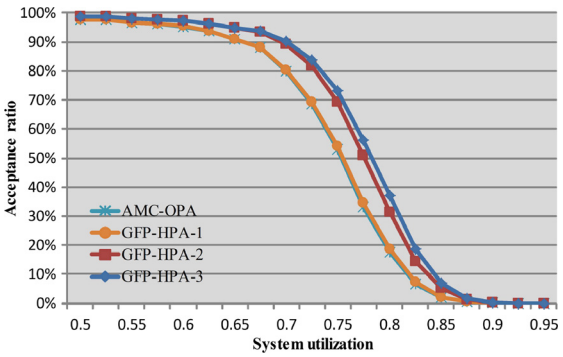

Fig. 4. Schedulability performance comparison.

ity assignment as shown in Column 7, 8 and 9, if different priority orderings are employed. Moreover, considering that the OPA is included in the HPA as described in Algorithm 1, we can come to an obvious conclusion that the HPA strictly dominates the OPA.

To demonstrate the effectiveness of our generalized strategy GFP-HPA-x (the proposed GFP scheme combined with the HPA algorithm, where $x \in \{1, 2, 3\}$ denotes the number of priority orderings employed), an empirical investigation is conducted by comparing it with existing AMC-OPA (Baruah's AMC-max scheme combined with Audsley's OPA) in terms of acceptance ratio. And comparative results are presented in Fig. 4, with 10 000 task sets generated at each utilization level. As shown, GFP-HPA-x always outperforms AMC-OPA in schedulability and the improvement becomes more significant as x increases. Due to space limitation, we refer readers to Appendix II in the supplementary file [13] for full task set generation and more detailed discussions.

6. Conclusion

In this letter, we have proposed a generalized fixed-priority preemptive scheduling approach for mixed-criticality systems on a uniprocessor platform, allowing different priority orderings in different phases. We also derive a sufficient schedulability analysis to provide safe guarantees on certification requirements and provide an effective priority assignment scheme based on the popular OPA algorithm. As demonstrated through experiments, this generalized approach provides more choices in priority assignment, making it more flexible and thus conductive to better schedulability performance.

References

[1] S. Vestal, Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance, in: IEEE 28th Real-Time Systems Symposium, 2007, pp. 239–243.
[2] S.K. Baruah, A. Burns, R. Davis, Response-time analysis for mixed criticality systems, in: IEEE 32nd Real-Time Systems Symposium, 2011, pp. 33–43.
[3] S. Baruah, V. Bonifaci D'Angelo, et al., The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems, in: 24th Euromicro Conference on Real-Time Systems, 2012, pp. 145–154.
[4] E. Pontus, Y. Wang, Bounding and shaping the demand of mixed-criticality sporadic tasks, in: 24th Euromicro Conference on Real-Time Systems, 2012, pp. 135–144.
[5] T. Fleming, A. Burns, Extending mixed criticality scheduling, in: IEEE 34nd Workshop on Mixed Criticality Systems, Real-Time Systems Symposium, 2013, pp. 7–12.
[6] R. Pathan, Schedulability analysis of mixed-criticality systems on multiprocessors, in: 24th Euromicro Conference on Real-Time Systems, 2012, pp. 309–320.
[7] A. Burns, R. Davis, Adaptive mixed criticality scheduling with deferred preemption, in: IEEE 35nd Real-Time Systems Symposium, 2014, pp. 21–30.
[8] J. Real, A. Crespo, Mode change protocols for real-time systems: a survey and a new proposal, Real-Time Syst. 26 (2) (2004) 161–197.
[9] S. Baruah, A. Burns, R.I. Davis, An extended fixed priority scheme for mixed criticality systems, in: Proc. ReTiMiCS, RTCSA, 2013, pp. 18–24.
[10] F. Santy, L. George, P. Thierry, et al., Relaxing mixed-criticality scheduling strictness for task sets scheduled with FP, in: 24th Euromicro Conference on Real-Time Systems, 2012, pp. 155–165.
[11] A. Burns, R. Davis, Mixed criticality systems-a review, Tech. Rep., University of York, UK, 2013.
[12] N.C. Audsley, On priority assignment in fixed priority scheduling, Inf. Process. Lett. 79 (1) (2001) 39–44.
[13] Y. Chen, K.G. Shin, H. Xiong, Supplement of "Generalizing fixed-priority scheduling for better schedulability in mixed-criticality systems", https://kabru.eecs.umich.edu/wordpress/wp-content/uploads/supplementary_YaoChen2016.pdf.