
Sepideh Safari, Mohsen Ansari, Ghazal Ershadi, and Shaahin Hessabi, member, IEEE

Abstract—Advancement of Cyber-Physical Systems has attracted attention to Mixed-Criticality Systems (MCSs), both in research and in industrial designs. As multicore platforms are becoming the dominant trend in MCSs, joint energy and reliability management is a crucial issue. In addition, providing guaranteed service level for low-criticality tasks in critical mode is of great importance. To address these problems, we propose “LETR-MC” scheme that simultaneously supports certification, energy management, fault-tolerance, and guaranteed service level in mixed-criticality multicore systems. In this paper, we exploit task-replication to not only satisfy reliability requirements, but also to improve the QoS of low-criticality tasks in overrun situation. Our proposed LETR-MC scheme determines the number of replicas, and reduces the execution time overlap between the primary tasks and replicas. Moreover, instead of ignoring low-criticality tasks or selectively executing them without any guaranteed service level in overrun mode, it mathematically explores the minimum achievable service guarantee for each low-criticality task in different execution modes, i.e. normal, fault-occurrence, overrun and critical operation modes. We develop novel unified demand bound functions (DBF), along with a DVFS method based on the proposed DBF analysis. Our experimental results show that LETR-MC provides up to 59% (24% on average) energy saving, and significantly improves the service levels of low-criticality tasks compared to the state-of-the-art schemes.

Index Terms—Task Replication, Energy Management, Guaranteed Service Level, DBF, Multicores, Mixed-Criticality Systems.

1 INTRODUCTION

WITH the advancement of Cyber Physical Systems, Mixed-Criticality Systems (MCSs) have recently become the subject of an important research area as the next generation of complex embedded and cyber physical systems [1]. MCSs integrate components with different levels of criticality onto a common platform to reduce cost, space, weight, heat generation and power consumption of the system [5-6]. The advent of MCSs poses significant new challenges on the system design since applications can interfere with each other on common resources. This point shows the importance of task scheduling and certification on the shared platform of MCSs. In addition to the certification issue, simultaneous energy and reliability management is another crucial aspect of designing MCSs. As the number of cores on a single chip continues to increase [8], [15], the chip power/energy consumption will increase exponentially. Dynamic Power Management (DPM) [35] and Dynamic Voltage and Frequency Scaling (DVFS) [9] are two popular techniques for energy management. However, scaling the supply voltage in DVFS can potentially degrade the system reliability due to the increasing transient fault rate in the current ever-decreasing technology feature sizes [7], [10]. Transient faults are usually mitigated through exploiting re-execution [16], [27-28] or replication [2], [11]. The former imposes time overhead, and affects the most critical parameter in MCSs, and the latter imposes power consumption overhead. Hence, choosing a suitable fault-tolerant technique for MCSs while reducing overall energy consumption is essential.

Another growing difficulty in the scheduling of MCSs is the quality-of-service (QoS) of low-criticality (LC) tasks in overrun situations. LC tasks have one designer-specified Worst-Case Execution Time (WCET) [16], [17], while high-criticality (HC) tasks have two instances of WCETs: $W_{LO}$ which is estimated by system designers, and $W_{HI}$ which is more pessimistic and estimated by certification authorities [6], [17-18]. The system starts in normal mode, and whenever an HC task exceeds its $W_{LO}$, the system switches to the overrun mode. HC tasks must be schedulable in both the normal and overrun modes, but the schedulability of LC tasks in overrun mode depends on the chosen scheduling scenario. Some scheduling algorithms discard all LC tasks [6], [16-17], while the others guarantee a minimum service level for LC tasks. Overall, presenting a scheduling algorithm that simultaneously supports timeliness, energy management, fault-tolerance and guaranteed service level for LC tasks is becoming increasingly challenging in the design of MCSs as technology advances to multicores.

In this paper, we exploit task replication to achieve the reliability target. The task replication approach schedules multiple copies of a task on different cores to tolerate a certain number of faults [10], [15], [17]. Our proposed LETR-MC scheme determines the proper number of replicas for...
each HC task to achieve the given reliability target. Then, it conservatively maps the task set into cores and schedules them to reduce the execution time overlap between the primary tasks and their replicas in an energy-saving manner while preserving certification constraints. In order to conquer service abruption problem in overrun situations, by addressing fault and overrun as two independent events, we consider different execution modes including normal, fault-occurrence, overrun and critical. Also, we modify the elastic mixed-criticality task model to let LC tasks have different guaranteed relaxed periods in each execution mode. In order to check the schedulability of the task set in each execution mode, we develop unified demand bound function (DBF) analysis. Also, we propose a DBF-based DVFS technique that assigns the proper task frequency, which considers timeliness and reliability constraints as well as QoS. By considering various execution modes and proposing the unified DBF schedulability test, the multicore system can tolerate fault occurrence, overrun and even both of them in distinct cores at the same time. The main contributions of this work are:

- Proposing a novel Low Energy Task Replication mechanism in Mixed-Criticality systems (called LETR-MC scheme) to support certification (timeliness), energy management, fault tolerance, and guaranteed service level simultaneously in multicore.
- Adapting task replication, not only as a fault tolerant technique but also to improve LC tasks’ QoS in the overrun situation.
- Considering different execution modes including normal, fault-occurrence, overrun, and critical, and mathematically guarantying an acceptable service level for LC tasks in each mode.
- Developing a new unified DBF-based schedulability test, under deadline, energy reduction, reliability requirements, and guaranteed relaxed periods considerations in different execution modes, and proposing a DBF-based DVFS technique, and exploiting DVFS along with DPM to save energy.

The remainder of this paper is organized as follows. In Section 2, we review related work. Section 3 presents models and assumptions. Section 4 presents our LETR-MC scheme in details. The experimental results are discussed in Section 5. Finally, we conclude the paper in Section 6.

2 RELATED WORK

Mixed-criticality systems were first introduced by Vestal [4]. The primitive research proposed different scheduling algorithms which are classified according to their policy to deal with LC tasks in the overrun situation. Some scheduling algorithms discard all LC tasks after entering the overrun mode, e.g. EDF-VD (Earliest Deadline First with Virtual Deadline) [6], [16-17], [21], [24]. Other ones degrade the QoS of LC tasks to mitigate the effect of service interruption [18], [23], [25-26]. Su et al. [18] have proposed ER-EDF scheduling in a single-core system, which increases the period of LC tasks in the overrun mode to reduce their execution frequency and competition with HC tasks. Su et al. [25] have improved the ER-EDF algorithm by focusing on the online behavior of the single-core system. Su et. al [26] have applied the ER-EDF scheduling to multicore. The reference [23] combines ER-EDF scheduling with virtual deadlines for the dual-criticality single-core system, and guarantees LC tasks service level in the overrun mode. The mentioned works do not consider fault-tolerance or energy management.

Other recent works explore the scheduling problem in the context of fault-tolerant MCSs without considering energy management. Works presented in [16], [27-29] use re-execution as their fault-tolerance technique. The references [16] and [27] wisely select the droppable LC tasks in the overrun mode. The references [28] and [29] immediately drop all LC tasks once either a transient fault or an overrun occurs. All of the above-mentioned references have considered a two-mode operation. Hence, the system switches to critical mode due to fault or overrun, and they do not guarantee an acceptable service level for LC tasks. The references [30] and [31] provide analysis techniques to bound the effects of task killing and service degradation on the safety and schedulability of the system. The reference [30] presents a method to convert the fault-tolerance problem into a standard scheduling problem in a single-core MCS. The reference [20] addresses fault occurrence and overrun with separate modes in a single-core and multiprocessors. However, it selectively chooses LC tasks to continue their execution in each mode.

Few works like [9], [32-34] cope with the energy management problem in MCSs, but they do not consider reliability requirements. Huang et al. [9] have proposed an optimal solution based on DVFS with the EDF-VD scheduling to minimize dynamic energy consumption for single-cores in normal mode, where tasks of the same criticality level share the same frequency. The reference [32] have extended the work in [9] to multicores, and HC tasks share the same frequency in overrun mode. Legout et al. [33] have developed an optimal solution for static energy reduction by applying DPM technique for single-core MCSs. Volp et al. [34] have considered an energy budget for multicore MCSs, and focus on energy utilization of HC tasks at the expense of sacrificing LC tasks. The reference [67] applies DVFS only on LC tasks with the cost of degrading their service level and only in the low-criticality mode of the system. Lia et. al in [36] and [37] have reduced the energy consumption of a single-core fault-tolerant MCS through resource demand analysis. However, they consider two-mode model, and drop all LC tasks in overrun situations; also the re-execution energy is not taken into account. All the mentioned works have considered two operating modes and have discarded all LC tasks or have selectively executed them in the overrun mode.

In this paper, we address the problem of scheduling mixed-criticality tasks on multicore, and reduce energy consumption while satisfying timeliness and reliability requirements, and at the same time, guarantee the acceptable service level for LC tasks in each mode.
3 Models and Preliminaries

In this section, we introduce the models and preliminaries which are used throughout the rest of the paper.

3.1 System and Application Model

In this paper, we consider an MCS with two different criticality levels, which are denoted as high-criticality and low-criticality levels. Also, we may choose any two criticalities out of the five criticalities levels in DO-178B standard [21]. There are \( n \) mixed-criticality tasks \( T = \{ t_1, t_2, ..., t_n \} \) that will be executed on a multicore platform. The cores are identical and DVFS enabled with a finite set of available frequencies, i.e. \( F = \{ f_1, ..., f_q \} \) where \( f_i = f_{\text{max}} \) and \( f_i = f_{\text{min}} \). These frequencies are normalized with respect to \( f_{\text{max}} \), i.e. \( f_{\text{max}} = 1 \).

Tasks are independent, and do not share any resources other than the core. Also, similar to [49-52] we assume that the context switching overhead is negligible. Each task \( t_i \) has hextuple of parameters \( \{ \zeta, W_{\text{LO}}, W_{\text{HI}}, T_i, D_i, L_i \} \):

- \( \zeta \in \{ \text{LC}, \text{HC} \} \) denotes the criticality level of \( t_i \).
- \( W_{\text{LO}} \) is the designer-specified WCET for \( t_i \).
- \( W_{\text{HI}} \) is the CAS-specified WCET for \( t_i \).
- \( T_i \) is the period of \( t_i \) (minimum inter-arrival time).
- \( D_i \) is the deadline for the task. We assume \( D_i \leq T_i \).
- \( L_i \) is the task’s safety level according to DO-178B.

It should be noted that in dual-criticality systems, if \( \zeta = \text{LC} \), then \( W_{\text{HI}} = W_{\text{LO}} \), otherwise \( W_{\text{LO}} < W_{\text{HI}} \). Each task \( t_i \) generates a sequence of jobs (or task instances) with the period of \( T_i \). Table 1 shows different task models. In conventional (basic) mixed-criticality task model, both of the LC and HC tasks have one desired period \( T_i \). The Elastic Mixed-Criticality (E-MC) task model [18] defines a maximum period \( T_{\text{max}} \) for LC tasks in addition to their desired period. When a task is executed with the desired period, it has a maximum level of QoS. However, maximum period represents the minimum level of QoS. The period of LC tasks can be extended by 2 to 5 times [67]. In the E-MC model, the utilization of the system is defined as follows (which will also be used in this paper):

- Low-level utilization of HC tasks:
  \[
  U(\text{HC}, L) = \sum_{\tau_i \in \Gamma} \zeta_i^\text{HC} u_{i,\text{LO}} = \frac{W_{i,\text{LO}}}{T_i} \tag{1}
  \]
- High-level utilization of HC tasks:
  \[
  U(\text{HC}, H) = \sum_{\tau_i \in \Gamma} \zeta_i^\text{HC} u_{i,\text{HI}} = \frac{W_{i,\text{HI}}}{T_i} \tag{2}
  \]
- Low-level (desired) utilization of LC tasks:
  \[
  U(\text{LC}, L) = \sum_{\tau_i \in \Gamma} \zeta_i^\text{LC} u_{i,\text{LO}} = \frac{W_{i,\text{LO}}}{T_i} \tag{3}
  \]
- Minimum utilization of LC tasks:
  \[
  U(\text{LC}, \text{min}) = \sum_{\tau_i \in \Gamma} \zeta_i^\text{LC} u_{i,\text{min}} = \frac{W_{i,\text{LO}}}{T_{i,\text{max}}} \tag{4}
  \]

The Extended E-MC (E²MC) task model [23] assumes that an LC task can have a pair of small and large periods \( T_{i,\text{LO}} \) and \( T_{i,\text{HI}} \) which represent its service guarantee in the normal and overrun execution modes, respectively.

Our Modified E-MC (MEMC) task model: We present MEMC task model, where HC tasks have one period \( T_i \) while LC tasks, in addition to their desired period \( T_i \) have four other periods. Each of these periods reveals QoS of LC tasks according to a specific execution mode of HC tasks. \( T_{\text{NR}} \) and \( T_{\text{OV}} \) are the periods of LC tasks in the normal and overrun operation of HC tasks and fault-free scenario, respectively. These periods are bigger than or equal to the desired period, and show the minimum achievable release frequency for jobs of the LC tasks in each mode. Details of the binary search method to find the proper periods are clarified in Section 4.3. Table 2 shows the notation used for variables throughout this paper.

3.2 Fault Model and Reliability Analysis

As mixed-criticality embedded systems often control safety-critical applications, tolerating faults and achieving high reliability levels are of great importance. In MCSs, each criticality level has an important property, which is known as Probability of Failure per Hour (PFH). PFH represents the maximum probability of failure to which each task of that level must adapt. The avionics DO-178B standard defines five criticality levels from A with highest, to E with lowest criticality levels. Safety requirements of each criticality level are shown in Table 3 [2]. In this paper, we assume that LC tasks are chosen from D or E levels without any fault-tolerant provisions, and HC tasks are chosen from A, B, or C levels. Hence, each task \( t_i \) from HC task set must be guaranteed to be schedulable, even in presence of faults, to achieve a failure rate of at most \( \text{PFH} = \text{PFH}^\text{\zeta_{\text{HI}}} \). Faults can be categorized into transient and permanent faults. Transient faults are found more frequently than permanent faults [34, 39]. Hence, in this paper, we focus on transient faults. Nevertheless, we try to

Table 1. Timing parameters for tasks in different models

<table>
<thead>
<tr>
<th>( \zeta )</th>
<th>WCET</th>
<th>Basic MC task model</th>
<th>VD model</th>
<th>E-MC</th>
<th>E²MC</th>
<th>MEMC (proposed model)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HC</td>
<td>( W_{i,\text{LO}}, W_{i,\text{HI}} )</td>
<td>( T_i )</td>
<td>( D_i )</td>
<td>( D_{i,\text{LO}} \leq T_i )</td>
<td>( T_i )</td>
<td>( D_{i,\text{LO}} \leq T_i )</td>
</tr>
<tr>
<td>LC</td>
<td>( W_{i,\text{LO}} )</td>
<td>( T_i )</td>
<td>( D_i )</td>
<td>( T_{i,\text{max}} )</td>
<td>( T_{i,\text{VR}}, T_{i,\text{OV}}, T_{i,\text{CR}} )</td>
<td>( D_{i,\text{LO}} \leq T_i )</td>
</tr>
</tbody>
</table>

Table 2. Adopted notations

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Gamma = { t_1, t_2, ..., t_n } )</td>
<td>A set of ( n ) E-MC tasks</td>
</tr>
<tr>
<td>( M = { m_1, ..., m_k } )</td>
<td>A set of ( c ) identical cores</td>
</tr>
<tr>
<td>( F = { f_1, ..., f_q } )</td>
<td>Core frequency levels</td>
</tr>
<tr>
<td>( \zeta = { \text{LC}, \text{HC} } )</td>
<td>Task’s criticality levels</td>
</tr>
<tr>
<td>( W = { \text{LO}, \text{HI} } )</td>
<td>LO- and High-level WCET of task ( t_i )</td>
</tr>
<tr>
<td>( X = { \text{NR}, \text{OV}, \text{FR}, \text{CO} } )</td>
<td>System operation modes</td>
</tr>
<tr>
<td>( m_k )</td>
<td>Set of tasks on the core ( m_k ) including</td>
</tr>
<tr>
<td>( \Gamma_{m_k} )</td>
<td>HC, LC and replica tasks</td>
</tr>
<tr>
<td>( m_i(\zeta) )</td>
<td>Set of tasks with criticality level ( \zeta ) on the core ( m_i )</td>
</tr>
<tr>
<td>( \Gamma_{m_i}(B) )</td>
<td>Set of replica tasks on the core ( m_i )</td>
</tr>
<tr>
<td>( \text{PFH}(\zeta) )</td>
<td>Probability of failure per hour of tasks with criticality level ( \zeta )</td>
</tr>
</tbody>
</table>
provide provisions to tolerate permanent faults as well. Transient faults are typically modeled using a Poisson distribution with an average arrival rate \( \lambda \) which depends on the core frequency [34], [38]. The fault rate at frequency \( f_i \) is modeled as [39]:

\[
\lambda(f_i) = \lambda_0 10^{-d(f_i-f_{\min})}
\]

where \( \lambda_0 = 10^{-4} \) (faults/ms) is the transient fault rate at \( f_{\max} \), exponent \( d \) is a sensitivity factor parameter with typical values in the range 2-6 [14-15]. We choose \( d=2 \) similar to [15]. The reliability of a task is defined as the probability of executing the task successfully, in the absence of transient faults [35], [38]. The reliability of task \( \tau_i \) running at frequency \( f_i \) can be expressed as [39]:

\[
R(f_i) = e^{-\lambda(f_i) W_i f_i}
\]

where \( \lambda(f_i) \) is given by Eq. 5, and \( W_i \) is the execution time of the task \( \tau_i \). In our proposed method, similar to [2],[11-12], [15], and [39], the fault detection takes place as soon as the completion of each task instance. If a task instance completes earlier than its WCET, the fault detection mechanism takes place as soon as the completion of the task instance; otherwise, faults are detected at task’s low and high level WCETs. Therefore, at the end of execution of each task instance, an acceptance test (or, sanity check) [45-46] is conducted to check for the occurrence of transient faults. If the acceptance test indicates a fault occurrence, the faulty output is discarded and the task copy will be executed to determine the correct output. Otherwise, task copies will be cancelled as soon as successful completion of the corresponding primary tasks [38]. It should be noted that acceptance tests are not completely accurate. Sometimes a fault may remain undetected or the acceptance test may diagnose a correct outcome as a faulty one [11]. Therefore, the probability that the acceptance test will perform incorrectly is considered as a factor in the reliability computation of the task, which is known as the coverage factor of the acceptance test. Thus, the reliability of a task instance can be expressed as:

\[
R_i(f_i) = (1-\alpha) \times e^{-\lambda(f_i) W_i f_i}
\]

where \( \alpha \) is the probability of making an incorrect decision during the acceptance test. The reliability of HC tasks depends on their WCETs as follows:

\[
R^{LO}_{\tau_i}(f_i) = (1-\alpha) \times e^{-\lambda(f_i) W^{LO}_i f_i},
\]

\[
R^{HI}(f_i) = (1-\alpha) \times e^{-\lambda(f_i) W^{HI}_i f_i}
\]

Consequently, the probability of failure (PoF) of the task \( \tau_i \) based on its WCET \( W^{LO} \) and \( W^{HI} \) are as follows:

\[
PoF^{LO}_{\tau_i}(f_i) = 1 - R^{LO}_{\tau_i}(f_i)
\]

\[
PoF^{HI}(f_i) = 1 - R^{HI}(f_i)
\]

In multicore platforms, task replication is likely to become a quite viable option for reliability management. By scheduling multiple copies of the same task on multiple cores, the likelihood of successfully completing at least one of them increases significantly. If the PoF of an HC task (when executed with \( W^{HI} \)) meets the PoF target, it means that the mentioned task does not need any replicas. Otherwise, the scheduler should determine the proper number of replicas of task \( \tau_i \) to achieve the reliability targets. Note that each replica task, being a periodic task itself, generates a sequence of instances on the core where it is assigned. Hence, we define two extremes for the number of replica tasks. The minimum number of required replica tasks for each HC task \( (r_{lower\_bound}) \) is the case when all the replicas will be executed with \( W^{LO} \), and can be expressed as:

\[
PoF_{\tau_i} \geq PoF^{LO}_{\tau_i}(f_i), \quad PoF^{HI}(f_i)^{r_{lower\_bound}}
\]

\[
r_{lower\_bound} \geq \left[ \frac{\log(PoF_{\tau_i} / PoF^{LO}_{\tau_i}(f_i))}{\log(PoF^{HI}(f_i))} \right]\]

The maximum number of required replicas \( (r_{upper\_bound}) \) is the case when all the replicas will be executed with \( W^{HI} \).

\[
PoF_{\tau_i} \geq PoF^{HI}_{\tau_i}(f_i), \quad PoF^{HI}(f_i)^{r_{upper\_bound}}
\]

\[
r_{upper\_bound} \geq \left[ \frac{\log(PoF_{\tau_i} / PoF^{HI}_{\tau_i}(f_i))}{\log(PoF^{HI}(f_i))} \right]\]

Therefore, we define upper and lower bounds for the number of required replicas. Hence, we define the parameter \( K \), where \( \mathbf{K} \in \left[ r_{lower\_bound}, r_{upper\_bound} \right] \). When \( K=0 \), all the replica tasks will be executed with \( W^{LO} \), i.e., \( r \) is the minimum number of required replicas. Otherwise, increasing the value of \( K \) increases the number of replica tasks that will be executed with \( W^{HI} \). Therefore, by giving a certain PoF target, we can find the minimum number of replicas \( r \) for each task to achieve its reliability target as follows:

\[
PoF_{\tau_i} \geq PoF^{HI}_{\tau_i}(f_i), \quad PoF^{HI}(f_i)^{r^{K}} \quad PoF^{LO}_{\tau_i}(f_i)^{-K}
\]

\[
r^{K} \geq \left[ \frac{\log(PoF_{\tau_i} / PoF^{HI}_{\tau_i}(f_i)^{K})}{\log(PoF^{LO}_{\tau_i}(f_i))} \right]\]

Therefore, in task replication technique, it is sufficient to have at least one task copy execution that passes the acceptance test. Hence, the execution will be unsuccessful only if all copies of a task encounter faults. Therefore, in task replication, if a primary task and its replica(s) have an overlap execution part, as soon as a primary task completes successfully, the remaining parts of its replica(s) will be abandoned to avoid further energy consumption. Also, we consider that each core is capable of detecting faults [36-37]. In order to detect faults, processing cores typically employ a low-cost hardware checker like Argus [47]. Argus provides low-cost, comprehensive, low power and high accuracy fault detection. It uses run-time checking of control flow, computation, data flow, and memory invariants. Argus adds less than 17% to the core area, and less than 11% to the total chip area, including caches. Argus does not change the clock cycle time, and is applicable to many embedded applications as well as multicore chips. We consider the overhead of fault detection as a part of the task’s WCET [15], [34], [37].

### 3.3 Power and Energy Consumption Model

**Power Model:** System-level power model consists of static
and dynamic components [9], [36-37]. The static power, \( P_{\text{static}} \), is consumed even when no computation is carried out [38]. The dynamic power \( P_{\text{dynamic}} \) includes a frequency-independent \( (P_{\text{ind}}) \), and a frequency-dependent \( (P_{\text{dp}}) \) power consumption factor. \( P_{\text{ind}} \) is consumed by the peripheral modules in the active mode. Hence, the total power consumption of each core can be written as:

\[
P_{\text{total}} = P_{\text{static}} + P_{\text{dynamic}} = I_{\text{sat}} V_{\text{r}} + C_{\text{eff}} V_{\text{r}} f_{\text{i}} + P_{\text{ind}}
\]

where \( C_{\text{eff}} \), \( V_{\text{r}} \), and \( f_{\text{i}} \) are the effective switched capacitance, supply voltage, and operating frequency of the core during the execution of task \( \tau_i \), respectively. Also, we assume that \( P_{\text{ind}} \) is equal to 0.1 [34]. When DVFS is used, each task \( \tau_i \) is executed at a voltage \( V_{\text{r}} \), which is less than the maximum supply voltage \( V_{\text{max}} \). By considering a near-linear relationship between voltage and frequency [13], [15], [39] when a task \( \tau_i \) is executed at the scaled voltage \( V_{\text{r}} f_{\text{i}} V_{\text{max}} \), the operational frequency is \( f = \rho_i f_{\text{max}} \), where \( f \) is the operational frequency corresponding to \( V_{\text{r}} \) and \( f_{\text{max}} \) is the maximum operating frequency corresponding to \( V_{\text{max}} \). Therefore, the total power consumption, which is consumed to execute a task \( \tau_i \) is given by:

\[
P_{\text{total}} = I_{\text{sat}} \rho_i V_{\text{max}} + C_{\text{eff}} (\rho_i V_{\text{max}})^2 \rho_i f_{\text{max}}
\]

\[
P_{\text{total}} = \rho_i P_{\text{max}}^\text{static} + \rho_i^2 P_{\text{max}}^\text{dynamic}
\]

In this equation, \( \rho_i \) is in the range of \( \rho_{\text{min}} \) to \( \rho_{\text{max}} = 1 \). \( \rho_{\text{min}} = V_{\text{min}} / V_{\text{max}}. \) In this paper, in contrast to most of the previous works that consider \( P_{\text{static}} = 0 \) [9], [12], [34], [37], we assume that maximum static power is constant and is a portion of the maximum dynamic power, i.e., \( P_{\text{max}}^\text{static} = \rho_i P_{\text{max}}^\text{dynamic} \) [11], [67] where, \( \rho_i = 0.2 \) [11], [38]. Hence, the total power consumption can be re-written as follows:

\[
P_{\text{total}} = \rho_i \phi \rho_i P_{\text{max}}^\text{dynamic} + \rho_i^2 P_{\text{max}}^\text{dynamic}
\]

By scaling \( V_{\text{r}} f_{\text{i}} \) in addition to power, the execution time of the task, and its energy consumption will change.

**Energy Model:** Scaling down the \( V_{\text{r}} f_{\text{i}} \) levels increases the execution time of the task \( W_i = W_i / \rho_i \). The energy consumption of \( j \)-th job (single job) of a task \( \tau_i \) is as follows:

\[
E_{\tau_i} = P_{\text{dynamic}} (\phi \rho_i^2 + \rho_i^3) \times \frac{W_i}{\rho_i}
\]

Also, the normalized energy consumption \( NE \) of a single job of a task \( \tau_i \) is:

\[
NE_{\tau_i} = (\phi + \rho_i^2) \times W_{\tau_i}
\]

Since the tasks are periodic and they may have multiple jobs in any time interval, the total normalized energy consumption of a task in any time interval of a given length is the summation of all its jobs’ energies \( h_{\tau_i} \) in this duration.

\[
NE_{\tau_i} = \sum_{j=1}^{h_{\tau_i}} (\phi + \rho_i^2) \times W_{\tau_i}
\]

The normalized total energy consumption of each core is the sum of energy consumptions of tasks on that core \( (m_k) \): Set of tasks on the core \( m_k \) including HC, LC and replica tasks) as follows:

\[
NE_{m_k} = \sum_{\forall \tau_i \in m_k} \sum_{j=1}^{h_{\tau_i}} (\phi + \rho_i^2) \times W_{\tau_i}
\]

Finally, the normalized total energy consumption of the whole system is the sum of energy consumptions of tasks on all the cores as follows:

\[
NE_{\text{system}} = \sum_{k=1}^{M} NE_{m_k} = \sum_{k=1}^{M} \sum_{\forall \tau_i \in m_k} \sum_{j=1}^{h_{\tau_i}} (\phi + \rho_i^2) \times W_{\tau_i}
\]

### 3.4 System Operational Model

The overview of the execution model for each core is shown in Fig. 1. Similar to [20], we distinguish between the execution time overrun of tasks and fault occurrence. Therefore, we guarantee an achievable service level for LC tasks in each execution mode to improve their overall QoS, and also save more energy. The behavior of our execution model in each mode is defined by the following epochs:

**Epoch 1:** The system starts with a normal mode (called NR). Each core stays in NR mode until either overrun or fault occurs. Hence, in this mode, all tasks are executed once with \( W^{\text{OV}} \), and LC tasks are executed with \( T^{\text{NR}} \) period. \( T^{\text{NR}} \) represents the highest service level of LC tasks.

**Epoch 2:** If any HC task exceeds its low-level WCET, its designated core switches to the overrun mode (called OV), where HC tasks (on that core) are safely executed once, and must meet their deadlines assuming high-level WCETs. LC tasks of that core will be executed with \( T^{\text{OV}} \) period to guarantee their service levels. If an LC task does not complete in its low-level WCET, it will be terminated.

**Epoch 3:** If any HC task signals completion after running for its \( W^{\text{LO}} \) but encounters a fault, the system switches to the fault-occurrence mode (called FO), where the scheduler guarantees the execution of the sufficient number of replicas of a faulty HC task to satisfy its reliability requirement. Hence, the scheduler executes its replicas on the cores. Only the cores which host the replicas of the faulty tasks switch to the FO mode. In this mode, HC tasks and replicas are still executed with \( W^{\text{LO}} \). Also, the execution of the required replicas for HC tasks must be guaranteed to finish before the deadline. LC tasks will be executed with \( T^{\text{FO}} \) period.

**Epoch 4:** When a core is in OV mode, the fault can occur; similarly, when it is in FO mode, overrun can happen, and both of these conditions enter the designated core into the critical mode (called CR). In this state, HC tasks are executed with \( W^{\text{HI}} \), and replicas are executed completely to tolerate faults, and LC tasks are executed with their \( T^{\text{CR}} \) period which is the minimum guaranteed service level that can be maintained.

Fig. 1 shows the various operating modes that each core can operate independently of the others. As it is clear in Fig. 1, in all epochs, the scheduler exploits the released dynamic slacks to improve the periods of LC tasks at runtime.
(\(T_{\text{runtime}}\)), which is between their desired periods and the guaranteed ones. Also, we consider a local switching [31], i.e., any core switches from one mode to another one, independent of other cores. Therefore, it is not necessary to reduce the QoS of LC tasks on all cores at once. Also, in our proposed operational model, the system switches back from each of the OV, FO, or CR modes to the normal mode at the end of each hyper period.

4 PROPOSED METHOD

In this section, we explain each part of our proposed scheme in details.

4.1 System Overview

Fig. 2 illustrates the overall design flow of the proposed LETR-MC scheme. The scheduler receives different inputs from hardware and software levels, and schedules tasks in the offline and online phases. In the offline phase, the scheduler initially creates Information Table (IT), which will be used during system operation in different modes to find the best V-f level for each task to satisfy its reliability and timing constraints. IT consists of the task set, criticality and required PFH levels for tasks, V-f levels, the \(W^D\) and \(W^H\) of tasks, reliability, and the number of required replicas in each V-f level. After mapping all tasks, the scheduler calls the service guarantee exploration function for computing scaling factors to apply to LC tasks’ periods. We adapt the ER-POED scheduling for our proposed method. The offline scheduler applies partitioned scheduling to all tasks on each core. After scheduling all tasks, the service guarantee exploration is called again to update scaling factors, while checking for DBF-based schedulability. The last section of the offline phase is to apply DVFS by using static slack reclamation. Eventually, the final offline schedule is sent to the online manager for applying DVFS and DPM, or improving the service guarantees of LC tasks by using dynamic slacks.

4.2 Task Mapping

Algorithm 1 shows the pseudo-code of our proposed task mapping method. In line 1, IT is constructed and sent to the task mapping mechanism, where HC tasks \(\Gamma(HC)\) and their corresponding replicas \(\Gamma(B)\) are initially mapped to cores, and then LC tasks \(\Gamma(LC)\) are allocated. HC and replica tasks are sorted in decreasing order of utilization at the maximum V-f level, respectively (in lines 2-3). In each iteration, the first task in the queue is selected for mapping onto the cores (line 5). Since task replication has the potential to tolerate permanent faults in addition to transient faults, the scheduler tries to avoid assigning replicas of the same task on the same core. Therefore, in line 6, set of cores containing the primary or replica versions of the selected task is removed from the core set \(M\), and the remaining cores are put into the temporary core set \(TC\). However, if after removing the mentioned cores, the \(TC\) becomes empty, all cores are returned to the \(TC\). Selection of cores for mapping is based on Worst-Fit Decreasing (WFD) and First-Fit Decreasing (FFD) bin packing. In the Worst-Fit Decreasing bin packing cores are sorted in decreasing order by utilization (line 7), then the core with the lowest utilization among others is selected for mapping (line 8). In the First-Fit-Decreasing bin packing, cores are sorted in decreasing order by utilization (line 7), then the selected task is allocated into the core with the lowest capacity available.
(largest utilization), in which it can be feasibly allocated in line 8. WFD is the best from the energy-awareness perspective, due to its load-balancing behavior [48]. If there is a core for mapping, the schedulability condition is checked in line 13 as follows:

\[ U_{\tau_i \in \Gamma_{mk}(HC)}(HC, H) + U_{\tau_i \in \Gamma_{mk}(B)}(B, L) \leq 1 \]

(24)

If the selected core \( sc \) passes the schedulability test, the selected task is mapped to the core \( sc \) in line 14 and the next task is chosen for mapping. However, if the selected core cannot satisfy the schedulability constraint, it is removed from the core set and the algorithm goes to line 7. Removing the cores which contain the primary or replica versions of the selected task, and also removing the cores which cannot satisfy the schedulability constraint may make TC empty. In this case, the cores containing the primary or replica versions of the selected task are added to the TC, and selection of a core is performed based on this set in lines 17-29. After mapping all HC and replica tasks, the scheduler sorts LC tasks in a queue, based on their utilization in decreasing order in line 33, and selects the first task in the queue for mapping as shown in line 35. The core selection \( (sc) \) is also based on WFD and FFD lines 37-38. In each iteration of LC task mapping the scheduler checks the total utilization of each core in lines 39-40 as follows:

\[ U_{\tau_i \in \Gamma_{mk}(HC)}(HC, H) + U_{\tau_i \in \Gamma_{mk}(B)}(B, L) + U_{\tau_i \in \Gamma_{mk}(LC), (LC, L)} \leq U_{\text{bound}} \]

(25)

where \( U_{\text{bound}} \leq 1.5 \). If the schedulability test is passed, the selected task is mapped to the chosen core. Otherwise, the core \( sc \) is removed from TC, and the algorithm iterates in lines 42-47.

After mapping all tasks, due to the challenges associated with task migration including increased sensitivity to implementation complexity, tight power budgets, requirements on execution predictability, the lack of virtual memory support in many low-end MPSoCs, and high runtime overhead [56], [58], migration of task instances from a core to another one is not permitted. After mapping all tasks, the service guarantee exploration function is called for finding the proper scaling factors for periods of LC tasks in each core to extend their periods and reduce their utilization.

4.3 Service Guarantee and Period Assignment

As it is mentioned in Section 2, most mixed-criticality scheduling algorithms discard all LC tasks after entering the overrun mode, or selectively execute them [6], [9], [15-16], [19-20], [23], [26-28], [31-36]. In order to guarantee an acceptable service level for LC tasks, we consider different execution modes (NR, FO, OV, CR), and define four uniform scaling factors \([a, \beta, \theta, \gamma]\) for each LC task in each execution mode. These scaling factors indicate the period of LC tasks in each execution mode, and also represent how frequently LC tasks release their instances to guarantee the timeliness and QoS level. Similar scaling factors of LC tasks that run on the same core are equal to each other. Also, the relationships between these scaling factors and the desired periods \( T_i \) of LC tasks in each operation mode are as follows:

\[ T_i^{NR} = aT_i, \quad T_i^{OV} = \beta T_i, \quad T_i^{FO} = \theta T_i, \quad T_i^{CR} = \gamma T_i \]

(26)

The total utilization of a core after mapping LC tasks may be bigger than one. Therefore, \( T_i^{NR}, T_i^{OV}, T_i^{FO}, \) and \( T_i^{CR} \) may have a bigger value than \( T_i \). Differences between \( T_i \) and the other periods of LC tasks are illustrated in the example in Appendix. Based on the schedulability conditions, we first derive lower and upper bounds of these uniform scaling factors in each operation mode. The feasible periods, which satisfy scheduling constraints, will be found between the lower and upper bounds.

**Epoch 1:** Each core’s utilization in NR mode is defined as follows:

\[ U_{\Gamma_{mk}}^{NR}(HC, L) + U_{\Gamma_{mk}}^{NR}(LC, L) \leq 1 \]

(27)

By replacing the desired period of LC tasks with \( T_i = \alpha T_i \) in their utilization formula, we have:

\[ U_{\Gamma_{mk}}^{NR}(LC, L) = \sum_{\forall \tau_i \in \Gamma_{mk}(LC)} \frac{W_i^{LO}}{T_i} = \frac{1}{\alpha} \sum_{\forall \tau_i \in \Gamma_{mk}(LC)} \frac{W_i^{LO}}{T_i} \]

(28)

It should be noted that \( T_i \) is the original period of LC tasks that determines when the tasks are generated. \( T_i^{NR} \) is the period of LC tasks in the normal operation mode of the system. To ensure the schedulability of tasks in NR mode, by replacing Eq. 28 in Eq. 27, we need to have:

\[ U_{\Gamma_{mk}}^{NR}(HC, L) + \frac{1}{\alpha} U_{\Gamma_{mk}}^{NR}(LC, MC) \leq 1 \]

(29)

Hence, the lower bound for \( a \) can be found as:

\[ a_{LB} = \frac{U_{\Gamma_{mk}}^{NR}(LC, MC)}{1 - U_{\Gamma_{mk}}^{NR}(HC, L)} \]

(30)

Here, the utilization which arises from execution overlap of HC tasks and corresponding replicas is ignored. The point is that utilization-based approach does not consider scheduling of tasks (the start time and end time of tasks). Therefore, considering the utilization of all replicas in normal mode is pessimistic because in the actual case in the normal mode, replicas do not need to completely proceed, and only their overlap time with primary tasks will be executed. Therefore, at first, the service guarantee unit computes the scaling factors based on the utilization of tasks. Then, after scheduling the tasks, if it is necessary, the computed scaling factors will be updated through dbf analysis, as explained in Section 4.4.

**Epoch 2:** The utilization bound at OV mode is as follows:

\[ U_{\Gamma_{mk}}^{OV}(HC, L) + U_{\Gamma_{mk}}^{OV}(LC, L) \leq 1 \]

(31)

In the OV mode, LC tasks are executed with \( T_i^{OV} = \beta T_i \). By replacing the desired period of LC tasks with \( T_i^{OV} \) in their utilization formula (similar to computation of \( a \)) the lower bound for \( \beta \) can be found as:

\[ \beta_{LB} = \frac{U_{\Gamma_{mk}}^{OV}(LC, MC)}{1 - U_{\Gamma_{mk}}^{OV}(HC, H)} \]

(32)

**Epoch 3:** In the FO mode, replicas are executed completely due to fault occurrence in primary HC tasks. The utilization bound at this point is as follows:
Algorithm 2: Find feasible scaling factors $SF=[a, b, \theta, \gamma]$

**Inputs:** $\Gamma$, tasks to cores mapping, $M$, $a_{ib}$, $b_{ib}$, $\beta_{ib}$, $\beta_{ub}$, $\theta_{ub}$, $\gamma_{ub}$, and $\epsilon=0.001$. Execution modes $X$.

**Output:** Finding the best scaling factors.

**BEGIN**
1. for $i = 1$ to $M$ do // loop over all cores
   2. while $(\theta_{ub} - \beta_{ub} > \epsilon)$ do
      3. $\beta = \frac{(\theta_{ub} + \beta_{ub})}{2}$;
      4. while $(\theta_{ub} - \beta_{ub} > \epsilon)$ do
         5. $\theta = \frac{(\theta_{ub} + \beta_{ub})}{2}$;
      6. $\alpha_{ub,new} = max[\alpha, \theta]$;
      7. while $(\alpha_{ub,new} - \alpha_{ub} > \epsilon)$ do
         8. $\alpha = \frac{(\alpha_{ub,new} + \alpha_{ub})}{2}$;
      9. if ($T$ is schedulable in all execution modes $X$) then
         10. $\beta_{ub} = \theta_{ub} = \beta_{ub} = \alpha_{ub} = a$;
      11. else
         12. $\beta_{ub} = \beta, \theta_{ub} = \theta, \alpha_{ub} = a$;
      13. end if
   14. end while
   15. end while
   16. end while
   17. end for

**END**

$$U_{T_{\alpha}}^{FO} = U_{\alpha_{\mu}(HC)}(HC, L) + U_{\alpha_{\mu}(B)}(B, L) + U_{\alpha_{\mu}(LC)}(LC, L) \leq 1$$ (33)

In the fault-occurrence mode, the execution period of all the LC tasks on core $m_k$ is $T_{FO}$ where $T_{FO}=\theta T$. Hence, the lower bound for $\theta$ can be found as:

$$\theta_{LB} = \frac{1}{1-(U_{\alpha_{\mu}(HC)}(HC, L) + U_{\alpha_{\mu}(B)}(B, L))}$$ (34)

**Epoch 4:** In the CR mode, the utilization bound of each core is as follows:

$$U_{T_{\alpha}}^{FO} = U_{\alpha_{\mu}(HC)}(HC, H) + U_{\alpha_{\mu}(B)}(B, L) + U_{\alpha_{\mu}(LC)}(LC, L) \leq 1$$ (35)

In the FO mode, LC tasks execute with $T_{CR}=\gamma T$. Hence, the lower bound for $\gamma$ can be found as:

$$\gamma_{LB} = \frac{1}{1-(U_{\alpha_{\mu}(HC)}(HC, H) + U_{\alpha_{\mu}(B)}(B, L))}$$ (36)

We now define the upper bounds of these scaling factors. $\gamma_{LB}$ is the factor that makes the task set schedulable in a worst-case scenario [18], [23]. Suppose that, $T_{\gamma}\leq T_{FO}\leq T_{CR}$ and $T_{\gamma}\leq T_{VO}\leq T_{CR}$. Therefore, the upper bound for $\beta$ and $\theta$ are set as follows:

$$\beta_{ub} = \theta_{ub} = \gamma_{LB}$$ (37)

Also, assume that $T_{NO}^{NR} \leq T_{FO}$ and $T_{NO}^{NR} \leq T_{VO}^{NR}$. However, $T_{FO}$ and $T_{VO}$ are not equal, hence we choose the maximum of the two, as the upper bound for $a$:

$$\alpha_{ub} = max(\theta_{ub}, \beta_{ub})$$ (38)

Hence, the lower and upper bounds are guaranteed in the offline phase by considering task utilization. The pseudo code for finding proper scaling factors is described in Algorithm 2. The scaling factors are defined iteratively through binary search, according to [23], and $\epsilon$ shows the tolerable error. The code iterates for each core to find proper scaling factors for all LC tasks on that core. In line 9 of the algorithm, the scheduler checks whether the computed $a$, $b$, $\beta$, and $\theta$ are feasible based on the proposed scheduling (the effect of scheduling algorithm on scaling factors are described in Section 4.5). It should be noted that if the lower-bounds of these factors become less than 1, the algorithm sets them to 1. After finding the proper scaling factors, the scheduler updates Eq. 23 to set proper periods for LC tasks, and tune job’s release distances as follows:

$$T_i^{NR} = \frac{l}{1 \alpha T_i}, T_i^{OV} = \frac{l}{1 \beta T_i}, T_i^{FO} = \frac{l}{1 \theta T_i}, T_i^{CR} = \frac{l}{1 \gamma T_i}$$ (39)

### 4.4 Unified Demand Bound Functions Analysis

Demand Bound Functions (DBF) present an approach to analyze the schedulability of real-time workloads [22]. A mixed-criticality task set is schedulable if the maximum execution demand of all tasks is less than the resource supply in any time interval of a given length in each operation mode. The demand bound of a task in a given interval is defined as the sum of execution times of all jobs of tasks, which have both arrival times and deadlines in this interval. DBF-based test is effective but has high computational complexity, and can be applied to general mixed-criticality task sets. In the following, we develop a new DBF computation in each operation mode by considering timing, reliability, QoS, and energy. We compute DBF locally for each core, but we should also have a global glance at all cores because of replicas existence since execution or cancelation of replicas may have an effect on DBF computations of other cores. Also, in order to improve the accuracy of demand bound analysis, we use the unified DBF approach similar to [22], [40], which considers time intervals crossing the mode switch point (transition from one mode to another one) to link the system behaviors in different execution modes. We now derive task’s resource demands in all operation modes (epochs) to determine whether a given task set is schedulable under our proposed method.

**Epoch 1:** In the NR mode, the DVFS technique is applied to HC and LC tasks. Although replicas are canceled at the end of the successful completion of their corresponding primary tasks, their execution time overlaps with corresponding primary tasks should be considered in DBF computations. The overlap time of each replica job (overlap_time) is the difference between the completion time of the primary HC job and the start time of its corresponding replica job. The resource demands of tasks which are executed on core $m_k$ are computed as follows:

$$DBF_{\tau_{\beta}^{\alpha}(HC)}(\tau_i, l) = \max(\frac{l+T_i-D_i}{T_i}, 0) \cdot f_{\max} L_{WO}$$ (40)

$$DBF_{\tau_{\beta}^{\alpha}(B, L)}(\tau_i, l) = \sum_j overlap\_time_j$$ (41)

$$DBF_{\tau_{\beta}^{\alpha}(LC)}(\tau_i, l) = \max(\frac{l+T_i-D_i}{T_i}, 0) \cdot f_{\max} L_{WO}$$ (42)

A task set is schedulable in NR mode on core $m_k$ if and only if for $\forall i > 0$:

$$DBF_{\tau_{\beta}^{\alpha}(HC)}(\tau_i, l) + \sum_{\tau_{\beta}^{\alpha}(B, L)} DBF_{\tau_{\beta}^{\alpha}(HC)}(\tau_i, l) + \sum_{\tau_{\beta}^{\alpha}(LC)} DBF_{\tau_{\beta}^{\alpha}(HC)}(\tau_i, l) \leq l$$ (43)

**Epoch 2:** When an overrun occurs, three types of jobs may
exist, i.e. normal, overrun, and crossover [22-23]. Normal jobs have both release time and deadline before the mode switch point; these jobs execute with $W_{LO}$. Overrun jobs are released after the mode switch point. Jobs which are released before the mode switch point but have later deadlines are candidates as crossover jobs and their finish time (when they signal their completions) indicates their role. Assume that the switch point to the overrun mode is $t^o (t^o \leq t^f \leq t^o)$ where the length of time interval $[t^o, t^f]$ is $t^f - t^o$ and the time that the core spends in overrun mode is $t^f - t^o$. In this case, we define $x$ to show which tasks have crossover candidate jobs as follows:

$$
crossover = \begin{cases} 
  x = 0, & \text{if } t^o \mod T_i = 0 \\
  x = 1, & \text{if } t^o \mod T_i \neq 0
\end{cases}
$$

(44)

Now, if there is a task that has a crossover candidate job, i.e. $x=1$, we initially specify its job number $j$ that has release time before the mode switch point and the deadline after this point, and then check whether it is a crossover. Here, if a crossover candidate job $j$ signals completion before the mode switch point $t^o$, it is counted as a normal job (although its deadline is after $t^o$), if the released job starts after $t^o$ it is an overrun job. Otherwise, it is a crossover job:

$$
crossover = \begin{cases} 
  y = 0, & \text{if } \text{finish time}_{j} < t^o \\
  y = 1, & \text{if } \text{start time}_{j} < t^o < \text{finish time}_{j} \\
  y = 0, & \text{if } t^o < \text{start time}_{j}
\end{cases}
$$

(45)

If $x,y=1$, task $i$ has a crossover job and the scheduler saves its executed time before mode switch point in $Exc$. Note that, $n_i(l) = [(l + T_i - D_i)/T_i]$ is the number of jobs of the task $i$ in interval $l$. Also, $n_i(w)$ and $n_i(q)$ are the number of jobs of each task in normal and overrun modes, respectively. Therefore, $n_i(w) + n_i(q) = n_i(l)$. The number of overrun jobs for each task in duration $q$ is as follows:

$$
n_i(q) = \begin{cases} 
  \frac{(t^f - t^o) + T_i - D_i}{T_i}, & \text{if } x,y = 1 \\
  \frac{(t^f - t^o) + T_i - D_i}{T_i}, & \text{if } x,y = 0
\end{cases}
$$

(46)

Here, crossover job is considered as an overrun job. The crossover job has different execution time and frequency in the NR and OV modes. Hence, If $x,y=1$ the DBF requirement for crossover job (called $n_{OV}$) is computed as follows:

$$
n_{OV} = (Exc_i - \frac{f_{max}}{f_1}) + (W_{HI} - Exc_i)
$$

(47)

The DBF of HC and replica tasks in OV mode are as follows:

$$
DBF_{r_i \in \Gamma_m(6)}(\tau_i,l) = \begin{cases} 
  n_{i}(w_{r_i})\frac{f_{max}}{f_1} + n_i(q_{r_i})W_{HI}, & \text{if } x,y = 0 \\
  n_{i}(w_{r_i})\frac{f_{max}}{f_1} + n_i(q_{r_i})W_{HI} + n_i(q_{r_i} - 1)W_{HI}, & \text{if } x,y = 1
\end{cases}
$$

(48)

$$
DBF_{r_i \in \Gamma_m(6)}(\tau_i,l) = \sum_{j=1}^{f_{max}} (overlap\_time_j)
$$

(49)

Jobs of LC tasks are initially released based on $T^{NR}$ period, and their frequency for applying DVFS is $f_{NR}$. After overrun, jobs of LC tasks on core $m_i$ are released according to $T^{OV}$, and the frequency scaling factor changes to $f_{OV}$ which may be different from their normal frequencies due to various slack reclamation in these two modes.

$$
DBF_{r_i \in \Gamma_m(6)}(\tau_i,l) = \begin{cases} 
  \frac{(t^f - t^o) + T_i^{NR} - D_i}{T_i^{NR}}, & \text{if } x,y = 0 \\
  \frac{(t^f - t^o) + T_i^{OV} - D_i}{T_i^{OV}}, & \text{if } x,y = 1
\end{cases}
$$

(50)

The DBF of each core in Epoch 2 is computed as follows:

$$
DBF_{r_i \in \Gamma_m(6)}(\tau_i,l) = \sum_{j=1}^{f_{max}} W_{LO}^{r_i}(\tau_i,l)
$$

(51)

**Epoch 3**: When a job of an HC task $r_i$ encounters a fault at time $t$ (where $t < t^f$), the replica(s) of the faulty job will be executed completely. Hence, the cores which contain its replicas are activated for updating their DBF computation. In this condition, all previous jobs of a faulty task are executed correctly and their replicas are dropped, and only their execution time overlap is calculated in DBF (Eq. 48).

$$
DBF_{r_i \in \Gamma_m(6)}(\tau_i,l) = \begin{cases} 
  \frac{(t^f - t^o) + T_i^{NR} - D_i}{T_i^{NR}}, & \text{if } x,y = 0 \\
  \frac{(t^f - t^o) + T_i^{OV} - D_i}{T_i^{OV}}, & \text{if } x,y = 1
\end{cases}
$$

(52)

$$
DBF_{r_i \in \Gamma_m(6)}(\tau_i,l) = \sum_{j=1}^{f_{max}} W_{LO}^{r_i}(\tau_i,l)
$$

(53)

$$
DBF_{r_i \in \Gamma_m(6)}(\tau_i,l) = \begin{cases} 
  (t^f - t^o) + T_i^{NR} - D_i, & \text{if } x,y = 0 \\
  (t^f - t^o) + T_i^{OV} - D_i, & \text{if } x,y = 1
\end{cases}
$$

(54)

A task set is schedulable in the FO mode on core $m_i$ if and only if for $\forall t \geq 0$:

$$
DBF_{r_i \in \Gamma_m(6)}(\tau_i,l) = \sum_{j=1}^{f_{max}} W_{LO}^{r_i}(\tau_i,l) + \sum_{j=1}^{f_{max}} W_{LO}^{r_i}(\tau_i,l) \leq l
$$

(55)

In DBF based approach, according to the online conditions, the scheduler can provide better service levels for LC tasks, and execute them more frequently than $T^{OV}$ or $T^{FO}$. Similar to [67], the scheduler introduces a set of early release points between the desired period and $T^{OV}$ or $T^{FO}$. It also chooses one of the early release points that passes the DBF test as a new release point of LC tasks ($T_{runtime}$ in Fig. 1).

**Epoch 4**: As soon as a core experiences both the fault and overrun situations at time $t$, all LC tasks on the core should be executed with $T^{CR}$. The scheduler considers two consecutive mode changes, i.e. from the NR to the OV and then to the CR (Fig. 3 (a)), and from the NR to the FO and then to the CR (Fig. 3 (b)). For example, in Fig. 3 (a) one of the jobs of the HC task $r_i$ on C2 overruns at the time $t^o$ and C2 switches to OV mode. Also, one of the jobs of $r_i$ encounters a fault in C1 and its replica in C2 ($B_{r_i}$) should be executed. Therefore, C2 enters the critical mode at switch point time $t^o$ and all remaining LC jobs on this core are executed with $T^{CR}$ period. Hence, DBF computations from switch point time $t^o$ to the end of the interval are as follows:
Algorithm 3: The task scheduling of our LETR-MC scheme

Inputs: $I$, $PFH$, Scaling factors, $M$, Available V-I levels, different execution modes $X$.

Output: The task scheduling on each core.

BEGIN:
1. Call(Algorithm 1); //Task mapping
2. Call(Algorithm 2); //Compute SF
3. for each core $m$ in $M$ do
4. while (all tasks in $T_{m}$ are not scheduled) do
5. -- //Event: A job of $t_{ij}$ is released at time $t$ on the core $m$
6. if $Q_{ASAP} \neq \emptyset$ & $Q_{ALAP} = \emptyset$ then
7. if $m_{ij}$ is busy then // $m_{ij}$ is executed
8. if $i_{ij}$ is HC or LC then // ASAP tasks
9. if $m_{ij}$ is LC or HC
10. if $d_{ij} > d_{ij}$ then // $d$: deadline of the job
11. ER-EDF($i_{ij}$);
12. else
13. ER-EDF($i_{ij}$);
14. else // $m_{ij}$ is replica
15. ER-EDF($i_{ij}$); //shift $m_{ij}$ toward its release time
16. end if
17. end if
18. else // core $m_{ij}$ is idle
19. ER-EDF($i_{ij}$);
20. end if
21. elseif ($Q_{ASAP} = \emptyset$ & $Q_{ALAP} = \emptyset$) then
22. Schedule $i_{ij}$ in free time slots from $d_{ij}$ to $r_{ij}$;
23. else // $Q_{ASAP} \neq \emptyset$ & $Q_{ALAP} \neq \emptyset$
24. $i_{ij} = Q_{ASAP}.select();$ //Sort jobs based on their $d$
25. $i_{ij} = Q_{ALAP}.select();$ // Select the first $J$ with earliest $d$
26. if $d_{ij} < d_{ij}$ then
27. goto line 6;
28. else
29. Schedule $i_{ij}$ in free time slots from $d_{ij}$ to $r_{ij}$;
30. end if
31. end if
32. end while
33. end for
34. Update $SF$; // Update $a$, $\beta$, and $\theta$
35. Check DBF; // for each core in each execution mode $X$
36. Apply offline DVFS; // SUF and DBF-based heuristics
END
By scheduling HC tasks as soon as possible and their corresponding replicas as late as possible, it is sufficient to complete only one copy of each task successfully. Hence, the other versions of that task will be cancelled immediately to avoid further energy consumption.

The offline DVFS is applied to the final schedule of normal execution mode based on our proposed method in Section 4-6. After applying DVFS, the scheduler updates execution time overlap between primary tasks and their corresponding replicas, and obtains new scaling factors if necessary.

4.6 Energy Minimization Problem

We define the energy minimization problem for a set of periodic mixed-criticality tasks on multicore platforms that exploits task replication technique. The number of required replicas and the frequency of tasks should be properly determined, while the timing constraints of HC tasks and replicas are met, and acceptable service levels for LC tasks are simultaneously guaranteed.

1) Formal Problem Modeling: The goal of reliability-aware energy minimization problem on mixed-criticality multicore is to minimize the total energy consumption in all operation modes (called X in Table 2). LC tasks are executed with $W^{LO}_{f}$ in all execution modes. However, HC tasks are executed with $W^{HI}_{f}$ in NR and FO modes, and with $W^{HI}$ in OV and CR modes. The objective can be written as:

Minimize: $\sum_{i=1}^{c} \sum_{j=1}^{n} \sum_{k=1}^{h_{ij}} (\phi \times (f_{ij} / f_{max})^{2}) \times W_{ij}^{X}$ \hspace{1cm} (60)

$X \in \{NR, OV, FO, CR\}$

Constraints of the energy minimization problem are as follows. Task’s frequency levels in each mode are in the range of the minimum and maximum core frequencies:

s.t. $\forall \tau_{i}, \ f_{i} \in [f_{min}, f_{max}]$ \hspace{1cm} (61)

The WCET of a task $\tau_{i}$ at frequency level $f_{i}$ in criticality level $X$ should not exceed the task timing constraint ($D_{i}$).

s.t. $\forall \tau_{i}, \ \forall X, \ \forall f_{i}, \ \frac{W_{i}^{X}}{f_{i}} \leq D_{i}$ \hspace{1cm} (62)

Total DBF of each core in each execution mode should be less than the given interval $l$.

s.t. $\forall X, \ \forall M, \ \forall l \geq 0, \ DBF_{j}^{X} \leq l$ \hspace{1cm} (63)

Each task’s reliability should meet the reliability target according to PFH in DO-178B standard, as follows:

s.t. $\forall \tau_{i}, \ R_{\tau_{i}} \geq R_{target}$ \hspace{1cm} (64)

Frequency scaling affects the WCET of a task and consequently its reliability and PoF. Therefore, the required number of replicas for each task ($r_{\tau}$) to meet its reliability target in frequency level $f_{i}$ should be considered in energy minimization. Also, the scheduler should decide about using slacks for energy reduction or improving the QoS of LC tasks. In addition, the task set should be schedulable in each execution mode with computed scaling factors ($a, \beta, \theta, \gamma$) even after applying DVFS.

s.t. $\forall a, \beta, \theta, \gamma: 1 \leq a \leq \beta, \theta \leq \gamma$ \hspace{1cm} (65)

Optimally solving the energy minimization of multicore systems without considering the reliability requirement is NP-hard [9], [11]. Therefore, solving this problem in fault-tolerant mixed-criticality multicore systems is also NP-hard; hence, we develop a heuristic to manage energy consumption.

2) Proposed Heuristics: Offline DVFS is applied to the normal execution mode of the system. In the offline phase, the scheduler uses static slacks to apply DVFS for energy reduction. At first, $f_{max}$ is assigned to all the tasks as their execution frequency. Then, selection of tasks for reducing the frequency and using the static slack in offline phase is done according to Smallest-Utilization-First (SUF), and the proposed DBF-based heuristics. Although executing a task under a lower frequency reduces energy consumption, the task takes longer time to complete, which may violate the timing constraints. Furthermore, lowering task’s execution frequency increases the system transient fault rate, which in turn increases the probability of executing all replicas, or increments the number of required replicas. The scheduler should consider timing constraints, reliability and QoS analysis simultaneous in energy management because a decision made in any one affects the others. Algorithm 4 shows the pseudo-code of offline DFVS and the interplay of energy, reliability, and schedulability analysis in energy management.

In SUF, HC and LC tasks are sorted in a queue, based on their utilization in increasing order in line 2. The scheduler selects the first task in the queue to scale down its frequency, in line 4. First, the scheduler extracts the amount of static slack times and finds the minimum frequency that the selected task can use it, in lines 5-6. If the schedulability constraints are met in DBF analysis of the normal mode, and if the reliability target is met, this frequency is assigned to the selected task in lines 8-9. If timing constraints in DBF analysis are not satisfied, the frequency is scaled one level up in lines 10-11. However, if the reliability target is not met, the scheduler checks whether more replicas can be properly mapped and scheduled. If it can, this frequency is set in lines 12-16. Otherwise, it scales frequency one level up in line 18. The scheduler repeats the previous steps until no task’s execution frequency can be scaled down, and updates the static slack in each iteration.

In the DBF-based heuristic, there might be more than one task that can be scaled down; the scheduler chooses the task which has more impact on total energy reduction. After extracting the amount of static slack times (line 24), while there is a task in a $\Gamma$(HL) and there is static slack, the algorithm iterates among all the tasks, except replicas, on each core. In each iteration, it selects one of the tasks (line 27). The scheduler receives the amount of static slacks, and finds task’s minimum acceptable execution frequency based on the QoS, reliability and timing constraints in equations 60-65 (line 28).

If the reliability of the selected task decreases after lowering down the frequency level, the scheduler tries to increase the number of replicas. If new replicas can be properly mapped and scheduled on multicore, the scheduler sets the execution frequency of the selected task, otherwise, it scales up the frequency level of the selected task, and algorithm goes on (lines 30-37). If lowering down
the frequency violates the schedulability in DBF analysis, the scheduler scales the frequency one level up and iterates the above steps in lines 38–39. If the reliability and certification constraints are met, the scheduler assigns the computed frequency to the selected task and computes the energy efficiency factor (EEF) for each task based on the minimum acceptable execution frequency (lines 41–42) as follows:

\[
EEF_i = \frac{NE_i(\text{before } \text{DVFS}) - NE_i(\text{after } \text{DVFS})}{NE_i(\text{before } \text{DVFS})}
\]  

After computing the EEF for all the tasks, the scheduler finds the task with maximum EEF, which decreases energy consumption the most, in line 46. Afterward, the scheduler sets the frequency of the selected task in line 47, updates static slacks in line 48, removes the selected task from the for loop in line 49, returns the frequency of all remaining tasks to the maximum value in line 50, and repeats the above steps for all remaining tasks. The above steps are repeated until the EEFs of all tasks become zero, or there are no more static slacks. During applying offline DVFS special care should be taken. One point is that the released slack from cancellation of replicas in the normal mode cannot be used for applying DVFS to HC tasks, because replicas may require to be executed, and only LC tasks can occupy their places. Whenever a fault or an overrun occurs, LC tasks can be extended and replicas are executed. The other point is that all the available static slacks cannot be used for applying DVFS to HC tasks. Regarding the overrun occurrence, only slack \( S = \text{Available slack}-(W_{HL, W^{(i)}}) \) can be used for applying DVFS to HC tasks because by applying DVFS, each task \( \tau_i \) is executed up to \( W_{LO}/f_{\tau_i} \) and overrun is detected at the end of this time. Therefore, by reserving some of the available static slack time for the overrun occasion, if an overrun occurs after applying DVFS, there is enough time for executing the remaining parts of the job. It should be noted that after applying DVFS, the execution time overlap between the primary tasks and their corresponding replicas will be updated and considered in DBF analysis, i.e., applying DVFS and computing the execution time overlap between primary tasks and replicas are performed simultaneously.

Our online energy manager uses dynamic slacks to apply DVFS and DPM during runtime for further energy savings. Dynamic slacks are released due to replica cancellation or early completion of tasks. In runtime, initially the amount of released dynamic slack is determined. If the idle time of the core is longer than \( \Delta_0 \), it is beneficial for the system to go into sleep mode to reduce energy consumption. Otherwise, online lightweight job-level DVFS is used for energy saving. It exploits greedy slack assignment, i.e., all the released slack from the current job will be used to further lower down the V-f level of the next job whenever it is possible to do so. In order to reduce the overhead of DVFS computations, inserting more replicas is not permitted. Therefore, the V-f level is lowered down to a level where no more replicas need to be inserted. Otherwise, the slack will be transferred to the next jobs or it may even not be used. DVFS transition overhead may be significant or negligible based on how often the DVFS is applied. In modern microprocessors, changing the DVFS setting is rather frequent in response to rapid changes in the application behavior. The DVFS transition delay overhead in high-end Intel Core2 Duo E6850 is between 9 to 62 us, for embedded Samsung Exynos 4210 processor based on ARM Cortex-A9

\[\text{Algorithm 4: Offline DVFS, interplay of energy, reliability, and certification}\]

**Inputs:** The task scheduling on each core, \( T \), PFH, Scaling factors, \( M \), Available V-f levels.

**Output:** Acceptable V-f level for each task.

**BEGIN:**

1. \( \Gamma(\text{HL}) = \Gamma(\text{HC})+\Gamma(\text{LC}); \) // HC & LC tasks set
2. \( \Gamma(\text{HL}).\text{sort}(); \) //Sort HC and replica tasks w.r.t utilization
3. for all tasks \( \Gamma(\text{HL}) \) do
4. \( \tau_i = \Gamma(\text{HL}).\text{select}(); \) //Select the task with smallest utilization
5. SS\( \leftarrow \text{Extract_StaticSlack}() \);
6. \( f_{\tau_i} \leftarrow \text{Determine_min_freq}(W, SS); \)
7. while (freq is not assigned) do
8. \( f_{\tau_i} \leftarrow \text{UP Scale}(f_{\tau_i}); \)
9. \( f_{\tau_i} \leftarrow \text{UP Scale}(f_{\tau_i}); \)
10. else
11. \( f_{\tau_i} \leftarrow \text{UP Scale}(f_{\tau_i}); \)
12. end if
13. \( f_{\tau_i} \leftarrow \text{determine_min_freq}(W, SS); \)
14. while (freq is not assigned) do
15. \( f_{\tau_i} \leftarrow \text{UP Scale}(f_{\tau_i}); \)
16. \( f_{\tau_i} \leftarrow \text{UP Scale}(f_{\tau_i}); \)
17. end if
18. end while
19. end for

20. \( \Gamma(\text{HL}) = \Gamma(\text{HC})+\Gamma(\text{LC}); \) // HC & LC tasks set
21. SS\( \leftarrow \text{Extract_StaticSlack}() \);
22. while \((\Gamma(\text{HL})\neq \emptyset) \) or \((\text{SS} \neq \emptyset)\) do
23. for all tasks \( \Gamma(\text{HL}) \) do
24. \( \tau_i = \Gamma(\text{HL}).\text{select}(); \)
25. \( f_{\tau_i} \leftarrow \text{Determine_min_freq}(W, SS); \)
26. while (freq is not assigned) do
27. \( f_{\tau_i} \leftarrow \text{UP Scale}(f_{\tau_i}); \)
28. \( f_{\tau_i} \leftarrow \text{UP Scale}(f_{\tau_i}); \)
29. end if
30. end while
31. end for
32. \( \text{END} \)
core is about 11 to 18 us, and this overhead for the TI MSP430 microcontroller that is used for ultralow-power embedded systems is about 10 to 145 us [66]. Also, by using ultra-fast voltage regulators, where \( V_{ul} \) switching is moved into the sub-micron regime like the Intel Haswell CPU, switching between voltage levels takes place in less than 1us [65]. Therefore, the overhead of applying DVFS is negligible and can be considered as a part of the task’s WCET. Energy consumption of the memory unit is out of the scope of this paper. However, there are data recomputation techniques [59-64] that can be used for energy reduction of the memory. Finally, in order to explain how our proposed method works, there is an illustrative example which is presented in the appendix.

5 RESULTS AND DISCUSSION
In this section, we perform extensive simulations to present the effectiveness of our proposed LETR-MC method from the perspective of energy saving, reliability and QoS in different operation modes.

5.1 Experimental Setup
Due to lack of benchmark packages for MCSs, similar to [18], [20], [22-23], [25-28], [30-37], [40-43], [67], we evaluate our proposed scheme using synthetic task sets. The UUnifast algorithm is used to generate utilization for \( n \) tasks \( T = \{ t_1, t_2, \ldots, t_n \} \) with total utilization equal to \( U \) [28]. The UUnifast algorithm is proposed by bini and buttazzo [44] to generate utilizations of a task set to study uniprocessor scheduling, which has the lowest complexity among all task generation algorithms. The generated utilizations for HC and LC tasks correspond to their high-level WCETs (i.e. \( u_i(H,C,H) \)) and low-level WCETs (i.e. \( u_i(L,C,L) \)), respectively. We define the \( \Phi \) factor which is the probability that the generated task is HC. The periods of tasks are randomly selected from the set \( T = \{10, 20, 40, 50, 100, 200, 400, 500, 1000 \} \) ms [20]. Hence, the \( W_{HI} \) for an HC task is computed according to \( W_{HI} = T_i u_i(H,C,H) \), and for LC tasks \( W_{HI} = W_{LO} \). However, for HC tasks \( W_{LO} = \mu W_{HI} \), where \( \mu \) is a random value in the range of \([0.3, 0.5]\). The PFH level of all HC tasks is selected from levels A, B, and C in DO-178B standard. Hence, the target reliability and the number of required replicas for each HC task is computed based on the selected PFH level. We consider a multicore platform, and the available frequencies for each core are set as \( F = \{0.6, 0.7, 0.8, 0.9, 1\} \). In the online phase, tasks

![Normalized energy consumption in WFD mapping with FC=100%](image)

![Normalized Energy Consumption in FFD mapping with FC=100%](image)

![Normalized energy consumption in WFD mapping with FC=99%](image)

![Normalized energy consumption in FFD mapping with FC=99%](image)

![Normalized energy consumption in WFD mapping with FC=98%](image)

![Normalized energy consumption in FFD mapping with FC=98%](image)

![Normalized energy consumption in WFD mapping with FC=95%](image)

![Normalized energy consumption in FFD mapping with FC=95%](image)

Fig. 4: Normalized energy consumption of different operation modes based on WFD and FFD mappings, with different fault coverage values.
are executed with actual execution time which varies between 70 to 100 percent of their WCET.

5.2 Experimental Results and Discussions

We evaluated the energy consumption, reliability and QoS of our proposed method in different operation modes in a quad-core platform based on WFD and FFD mapping, and the results are shown in Fig. 4 and Fig. 5, respectively. Utilization U was varied from 0.5 to 4.5 with steps of 0.5, and in each utilization point, 50 task sets were synthetically generated. The results are reported as the averages of 100 repetitions of the experiment at each utilization point. The 

$P^{HI}$ factor is equal to 0.5. HC, and LC tasks are selected from levels B and D, respectively. Therefore, the average energy consumption and QoS are computed with preserving the target reliability in all experiments. Also, in Fig. 4 and Fig. 5 the energy consumption and QoS of our proposed method for different fault coverage values (FC) are reported. In these figures, the energy consumption and QoS of each operation mode are computed based on the worst-case scenario of that mode, i.e. the energy consumption of OV and FO modes are reported where all the HC tasks overrun, or all HC tasks encounter faults and all replicas are executed completely with maximum frequency, respectively. In addition to the worst-case scenario, the energy consumption is analyzed based on the actual case (Real_WF and Real_FF). In the actual-case, faults are injected into the system with Poisson distribution, and HC tasks overrun randomly. In Fig. 4, the normalized energy consumption before applying offline DVFS (X_Base, where X represents the operation mode including NR, OV, FO, and CR), after applying offline DVFS technique based on DBF and SUF (X_OffDVFS), and after applying online DVFS (X_OnDVFS) are shown. By increasing the utilization, the energy consumption is increased, and the DBF-based DVFS lowers the energy more than the SUF one. Since transient faults and overrun are rare in nature, LETR-MC achieves further energy reduction at runtime beyond what is achieved through the offline part of LETR-MC at design-time. Also, decreasing the fault coverage value reduces the reliability of the system according to equations 6-8. Hence, the schedulability of the task sets will be decreased, because more replica tasks are needed to schedule to achieve the given reliability target. Therefore, the lower fault coverage value leads to lower schedulability. Fig. 5. represents the QoS of our proposed LETR-MC method in different operation modes with WFD and FFD mappings and different fault coverage values. The QoS is computed based on the fraction of the number of remaining schedulable LC jobs in each operation mode to the total original number of jobs based on the desired period of LC tasks. As it is clear, by increasing the utilization, QoS is decreased to keep the system schedulable. Also, due to the load balanced mapping in the WFD, it out performs FFD in all operation modes, i.e. WFD can preserve higher percentage of LC jobs in each operation mode in comparison with FFD mapping. Meanwhile, the normal operation mode with WFD mapping (NR_WF) has the highest QoS, and the critical operation mode with FFD mapping (CR_FF) has the lowest. The results of actual-case scenario (Real_WF and Real_FF) is close to the QoS of Normal mode (NR_WF and NR_FF). Also, lower fault coverage leads to lower schedulability and also lower QoS in both the WFD and FFD mapping, because replica tasks reserve the execution time of the cores to satisfy the given reliability target.

Fig. 6 illustrates both energy consumption and QoS of the LETR-MC method in different operation modes with WFD and FFD mappings and fault coverage 100%. In this figure, the energy consumption is normalized to that of the scenario where all tasks are executed at $f_{max}$. HC tasks are executed with $W^{HI}$, and the jobs of LC tasks are released based on their desired period. This figure shows that in each utilization point, the energy consumption of the FFD mapping is close to WFD one. However, the QoS of FFD in that utilization point is less than the WFD one. Hence, FFD mapping consumes less energy at the expense of lowering down the number of executable LC jobs.

We compared our proposed method from the QoS point of view with [27] in Fig. 7. The reference [27] proposes an offline algorithm which improves the QoS of EDF-VD algorithm by trying to save as much LC tasks as possible in OV mode. Without considering any PFH level for tasks, it assigns one re-execution to each task. Therefore, all HC and LC tasks and their re-executions must be schedulable in normal mode. However, in overrun mode, HC tasks and their re-executions must be schedulable, and for LC tasks the algorithm tries to find schedulable set of LC tasks based on the scaling factor in EDF-VD scheduler. Afterward, it
tries to reserve re-executions for LC tasks that are schedulable in overrun mode. In this set of experiments, the utilization of a single-core platform is varied from 0.05 to 1 with steps of 0.05. In each utilization point 50 task sets are synthetically generated and the experiments are repeated 100 times. The $P_{\text{HL}}$ factor is equal to 0.4. In the first scenario, we consider that all HC and LC tasks need one re-execution. Therefore, the proposed method in [27] finds the number of LC jobs that can be schedulable in overrun mode. As it is shown in Fig. 7, this method is not schedulable after utilization point 0.65. In the second scenario, we modified their proposed method considering that LC tasks do not need re-execution. Hence, the schedulability of their method is improved. However, LETR-MC is schedulable in all utilization points, also it can preserve higher number of jobs than [27] in overrun mode. It should be noted that the reference [27] considers two operation modes, i.e. normal and overrun. However, in this set of experiments, we reported the QoS of LETR-MC in FO and CR mode in addition to NR and OV modes. By considering different operation modes and proposing MEMC task model, the LETR-MC method can provide higher guaranteed QoS.

We compared the energy consumption of LETR-MC with HSFA algorithm in [36-37] (Fig. 8). In these set of experiments, each task set task has 10 tasks and the experiments are repeated 100 times and the average of these repetitions are reported. In Fig. 8 (a), the high utilization of HC tasks is equal to 0.3 (U(HC,H)=0.3), and the utilization of LC tasks H(LC,L) varies from 0.3 to 0.7 with steps of 0.1 in a single core platform. By increasing the utilization of LC tasks, all methods consume more energy, since in higher utilizations there is less slack time to use for energy saving. In Fig 8 (a) and (b), the LETR-MC method consumes lower energy than HSFA, since it lowers down the number of LC jobs to make the system schedulable. Also, the DBF-based DVFS lowers the energy more than the SUF one. It should be noted that HSFA does not have an online manager, and dynamic slacks are not used for energy saving. However, the LETR-MC method can reduce the energy the most in the online phase. The faulty mode of HSFA algorithm has the highest energy consumption. However, in the FO and OV modes of LETR-MC method, due to the lower number of LC jobs, the energy consumption is less than the worst-case scenario where all HC tasks are executed with $W^{\text{HL}}$ and all replicas are executed completely at $f_{\text{max}}$. HSFA uses EDF-VD algorithm and it drops all LC tasks after entering the overrun mode. Hence, this method consumes the least energy in the OV mode. However, LETR-MC executes guaranteed service level of LC jobs in OV mode, which leads in more energy consumption than HSFA. However, by applying the online DVFS, we can mitigate the energy consumption.

6. Conclusion

In this paper, we proposed the LETR-MC scheme that concurrently considers certification, fault-tolerance, energy reduction, and QoS. We used task replication to tolerate faults, and improve QoS of LC tasks. The number of required replicas for each HC task is computed through the proposed formulas. Then, our proposed scheduling algorithm reduces the execution time overlap between the primary tasks and replicas to save more energy by dropping the remaining parts of replicas at the end of correct execution of their primaries. Also, through the presented service guarantee exploration algorithm, we theoretically guarantee an acceptable service level for LC tasks in different operation modes of the system, i.e. normal, overrun, fault-occurrence, and critical. In order to check the schedulability of the proposed method, we analyzed the resource demands of mixed-criticality tasks with the deadline and reliability constraints, energy reduction and QoS guarantee. Finally, we showed that energy consumption can be reduced in the offline and online phases by exploiting static and dynamic slacks, respectively, while preserving the guaranteed service level for LC tasks.

REFERENCES


sentation techniques for scheduling systems. Ejlali, An Elastic mixed


Sepideh Safari received the M.Sc. degree in computer engineering from Sharif University of Technology, Tehran, Iran, in 2016. She is currently working toward the PhD degree in computer engineering at Sharif University of Technology. Her research interests include, energy management in fault-tolerant mixed-criticality embedded systems.

Mohsen Ansari received the M.Sc. degree in computer engineering from Sharif University of Technology, Tehran, Iran, in 2016. He is currently working toward the PhD degree in computer engineering at Sharif University, from 2016 until now. His research interests include low-power design of embedded systems and multi-/many-core systems with a focus on dependability/reliability.

Ghazal Ershadi is currently working toward the B.Sc. degree in computer engineering at Sharif University of Technology. Her research interests include low-power design of cyber physical systems, energy management in fault-tolerant multicore embedded systems.

Shaahin Hessabi received the BS and MS degrees in electrical engineering from Sharif University of Technology, Tehran, Iran, in 1986 and 1990, respectively, and the PhD degree in electrical and computer engineering from the University of Waterloo, Ontario, Canada. He joined Sharif University of Technology, in 1996. He has published more than 100 refereed papers in the related areas. His research interests include cyber-physical systems, reconfigurable and heterogeneous architectures, NoC, and SoC.