Received July 6, 2020, accepted July 29, 2020, date of publication August 3, 2020, date of current version August 17, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3013721

Peak-Power-Aware Primary-Backup Technique for Efficient Fault-Tolerance in Multicore Embedded Systems

MOHSEN ANSARI[1], MOHAMMAD SALEHI[2], (Associate Member, IEEE), SEPIDEH SAFARI[1], ALIREZA EJLALI[1], AND MUHAMMAD SHAFIQUE[3,4], (Senior Member, IEEE)

[1]Department of Computer Engineering, Sharif University of Technology, Tehran 14588, Iran
[2]Department of Computer Engineering, University of Guilan, Rasht 45371-38791, Iran
[3]Institute of Computer Engineering, Vienna University of Technology (TU Wien), 1040 Wien, Austria
[4]Division of Engineering, New York University Abu Dhabi (NYU AD), Abu Dhabi, United Arab Emirates

Corresponding authors: Alireza Ejlali (ejlali@sharif.edu) and Mohsen Ansari (mansari@ce.sharif.edu)

This work was supported in part by the German Research Foundation (DFG) as part of the GetSURE Project in the scope of priority program Dependable Embedded Systems (SPP 1500).

ABSTRACT Multicore platforms offer great potential for task-level redundancy to achieve a degree of fault-tolerance/reliability in embedded systems by exploiting the idle cores. However, due to the Thermal Design Power (TDP) constraint, it may not be possible to simultaneously power-on all cores in a multicore chip at the full-throttle (e.g., in ARM's big.LITTLE architecture). Since TDP is the maximum sustainable power that a chip can dissipate safely (as per the specifications given by a chip vendor), violating TDP triggers a performance throttling mechanism (e.g., by lowering the operating voltage and frequency, or by power-gating with task migration) to avoid possible overheating problems. This can significantly affect the timeliness of the system, and hence, represents a serious challenge in using (off-the-shelf) multicore platforms in real-time embedded systems when exploiting it for full-scale reliability. That means only a few tasks can be afforded to run in a fully reliable mode under a given TDP constraint. In this article, at first, we study the power consumption of task-level redundancy running on multicore platforms. Then, to tackle the peak power problem, we propose a novel primary-backup scheme for power-aware scheduling of real-time tasks on core pairs in multicore systems. The proposed scheme aims at removing overlaps of peak power of concurrently executing tasks to keep the power consumption below the chip-level TDP constraint. This would facilitate higher reliability levels within a given power budget. To do this, considering the tasks' power profiles, we propose a task partitioning method along with maximum-peak-power-first (MPPF) and maximum-peak-power-last (MPPL) policies to schedule original and redundant copies of tasks, respectively. Our experiments show that our technique provides up to 50% (on average by 29.5%) peak power reduction compared to state-of-the-art schemes, while providing the same reliability level.

INDEX TERMS Peak power, primary-backup technique, embedded systems, fault-tolerance, thermal design power, real-time, efficiency, multicore platforms.

I. INTRODUCTION

Technology scaling allows integrating multiple cores onto a single chip for advanced embedded systems [1]–[4]. However, technology scaling also aggravates the reliability of on-chip systems. For instance, transient fault rate is increased due to lower voltages and shrinking transistor dimensions that lead to smaller critical charges [5], [6]. Transient faults typically resulted due to high-energy particle strikes in hardware that manifest as bit flips. Multicore embedded systems provide a great opportunity to employ fault-tolerant mechanisms against transient faults, such as redundant multithreading (RMT) [7], [8] and process level redundancy [9]. Task-level redundancy (e.g., RMT) is a well-established technique to achieve high reliability in multicore systems [10]–[12]. However, replicated executions significantly increase system

The associate editor coordinating the review of this manuscript and approving it for publication was Vyasa Sai.

power consumption that may exceed the chip Thermal Design Power (TDP) constraint which cannot be sustained for longer periods of execution. TDP is considered as the highest sustainable power that a chip can dissipate before being forced to exploit a performance throttling mechanism, e.g., Dynamic Thermal Management (DTM) [13], [14]. DTM is a system-level solution to tackle thermal hot spots and to keep the system below a safe operating temperature. However, it may reduce system performance and may lead to violating system timing constraints [13], [14]. When a chip violates its TDP constraint, DTM techniques automatically trigger task migration or throttle the voltage and frequency to prevent hardware damage, but it can also significantly reduce the system's performance [13], [15]. Therefore, DTM techniques, or at least their frequent triggers, may not be efficient for the systems that require satisfying strict timing constraints, e.g., real-time embedded systems [1], [13], [17], [18].

In this article, we demonstrate how task replication may increase peak power consumption and consequently may result in a chip TDP violation (see Section II.A). Then, we propose a peak-power-aware primary-backup technique scheme, which manages peak power consumption efficiently for task-level redundancy on multicore systems while considering the deadlines of different tasks (see Section IV). Our technique schedules real-time tasks on core pairs in a multicore system without violating the tasks' timing constraints. *This method aims at removing overlaps of peak power of concurrently executing tasks to keep the peak power consumption below the chip TDP.* To do this, considering the tasks' power profiles, at first, we partition the tasks into parts. We consider a part of a task a section of subsequent instructions such that different parts have different peak power values (see Section IV.A). Then, we use two different policies for scheduling the original and redundant copies of each task. We use the maximum-peak-power-first (MPPF) policy to schedule the original tasks, and for the redundant tasks, we use the maximum-peak-power-last (MPPL) policy. In this way, those parts of a task that consume higher power overlap with the parts of the other tasks that consume lower power. This leads to meet the timing and power constraints (see Section IV.A). In summary, our technique tries to spread the parts of tasks that consume high power over the entire available time interval before the tasks' deadline with the aim of keeping the total peak power below the chip TDP.

To evaluate our proposed technique, we compared it with state-of-the-art schemes for the worst-case and average-case execution conditions (Section V.B and Section V.C). Our experiments show that our technique provides up to 50% (on average by 29.5%) peak power reduction compared to the other schemes in the worst-case fault condition. Also, our technique provides up to 50% average power reduction in the average-case execution condition through canceling unnecessary execution when no fault occurs.

The rest of this article is organized as follows. In Section II we present the motivation and contributions of this work. Section III presents models and assumptions. Section IV presents our technique in details. The experimental results are presented and discussed in Section V. The related work is reviewed in Section VI. Finally, we conclude the paper in Section VII.

II. MOTIVATION AND CONTRIBUTIONS

A. MOTIVATIONAL ANALYSIS OF PEAK POWER PROFILES

Let us consider a 4-core embedded chip with 3W of TDP that executes two soft real-time tasks T_1 and T_2. For this experiment, we used the applications QSORT (T_1) and TIFF (T_2). We assume that the tasks arrive at time $t = 0$ and have a deadline $D = 4$ms. The execution time of T_1 and T_2 are 3.1ms and 2.4ms, respectively, and each task consumes about 0.8W of peak power throughout its execution (see details in V-B). We also assume that after finishing the task, the underlying core goes to deep sleep mode (i.e., power-gated) and consumes almost no power. Here, for simplicity of presentation and ease of discussion, we temporarily consider that the tasks' peak powers are equal to the tasks' average power. In the rest of this article, when we present our technique, we consider that the power consumption varies during a task's execution and, different tasks have different power profiles, similar to real-world scenarios.

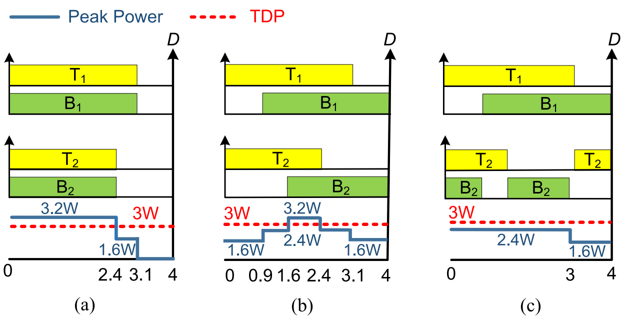


FIGURE 1. Motivational analysis of peak power problem in the primary-backup technique. a) Parallel execution of original and redundant tasks (hot standby sparing), b) Delayed execution of redundant tasks [16], c) Partitioned tasks execution (our scheme).

We exploit task-level redundancy to achieve fault-tolerance, and consider that any core can be dynamically coupled to another core to form a core pair (based on the technique presented in [19]). Each core pair executes an original task T_i and its redundant task B_i. For example, in Figure 1a, we have two core pairs that execute the tasks T_1 and T_2 and their redundant tasks B_1 and B_2. One way to execute a task in the *primary-backup* mode is to execute each original task and its redundant in parallel (i.e., hot standby sparing), as shown in Figure 1a. In this way, the tasks start simultaneously at $t = 0$, the tasks T_1 and B_1 finish at 3.1ms and the tasks T_2 and B_2 finish at 2.4ms. Here, the total peak power of the system is 3.2W during the time interval between 0 and 2.4ms, and hence, it violates the chip TDP of 3W.

Figure 1b shows another possible execution scenario for these tasks where the original tasks (T_1 and T_2) start as soon as possible and the redundant tasks (B_1 and B_2) start as late as possible, hoping that the original tasks will finish successfully and the redundant tasks will be dropped to avoid excessive power consumption. Such a method has been used in many previous works [16], [20]–[23]. For example, Haque *et al.* [23] propose exploiting the earliest-deadline-first (EDF) and earliest-deadline-late (EDL) policies to schedule the original and redundant tasks, respectively. These methods effectively reduce average power consumption by dropping unnecessary redundant executions when no fault occurs. However, since they do not consider peak power, they may result in violating TDP. This case is shown in Figure 1b wherein the time interval 1.6ms to 2.4ms all the four cores are active at the same time, and hence, the chip total power consumption is 3.2W, which is higher than the chip TDP (i.e., 3W).

Figure 1c shows a possible scheduling scenario for the tasks that do not violate the TDP constraint. In this scenario, T_2 is divided into two parts, and then, T_1 and the parts of T_2 are scheduled such that at any time instant at most three cores are active. Therefore, since each core consumes 0.8W, the total power is less than or equal to 2.4W (i.e., less than the chip TDP). In this article, we propose a scheme that considering tasks' power profiles, at first, partitions the tasks into parts with different peak power consumption, and then, schedules the tasks such that the total power consumption is kept below the chip TDP.

B. OUR NOVEL CONTRIBUTIONS AND CONCEPT OVERVIEW

In this article, we propose a peak-power-aware primary-backup technique that enables task-level redundancy to achieve fault tolerance in multicore embedded systems under timing and TDP constraints.

In a nutshell, the main contributions of this article are:

- A peak power management scheme that is conducted at run time by managing peak-power overlaps between concurrently executing tasks.
- A scheduling algorithm for the primary-backup technique to enable task-level redundancy on multicore systems. This algorithm reduces peak power consumption through partitioning the tasks into parts without violating any real-time constraint.
- Employing two specific scheduling policies to manage concurrent task executions and peak power consumption in the worst-case fault scenario.
- An online technique to achieve further reduction in power consumption beyond what is provided by the proposed offline technique. It should be noted that in our scheme, at design time, the system is designed such that the chip TDP is met. At run time, we reduce average power consumption to alleviate effects of high power, e.g., aging and wear-out.

III. MODELS AND ASSUMPTIONS

In this Section, we present the models and preliminaries of the paper.

A. SYSTEM AND TASK MODEL

We consider a multicore system with m cores $C = \{C_1, C_2, \ldots, C_m\}$ similar to Intel SCC [24], [25]. In this article, we assume that the system consists of $k = m/2$ core pairs $CP = \{CP_1, CP_2, \ldots, CP_k\}$. The system executes a set of n non-preemptive frame-based soft real-time tasks $\Psi = \{T_1, T_2, \ldots, T_n\}$ where the tasks release at the same time and share a common deadline D. Note, this assumption is valid for frame-based tasks and cases where multiple tasks belong to one complex multi-tasked application (e.g., video streaming), and such an application model is adopted in many related works such as [13], [26]–[30]. We also assume that each task T_i has an execution time τ_i and all tasks have the same period of D. Examples of these systems are medical care devices, avionics systems, control of chemical reactions, and surveillance systems [22], and an example of a frame-based soft real-time application is MPEG Player (multimedia) [31], [32].

B. POWER MODEL AND POWER MANAGEMENT TECHNIQUE

We consider a system-level power model where total power comprises static and dynamic power (Eq. 1). The static power, (P_s), mainly consumed due to sub-threshold leakage current (I_{sub}) even when no computation is carried out. The dynamic power, (P_d), is dissipated due to activity of the digital circuits and depends on both supply voltage (V_{dd}) and operational frequency (f). According to [33]–[37], the total power of a core can be written as:

$$P = P_s + P_d = I_{sub}.V + C_{eff}.V_{dd}^2.f, \qquad (1)$$

where C_{eff} is the effective switched capacitance.

In this article, we use DPM to manage peak power consumption, i.e., whenever a core is temporarily idle, it goes to sleep mode to reduce dynamic power [38].

C. FAULT MODEL AND FAULT-TOLERANCE TECHNIQUE

High reliability as one of the main design objectives in many embedded systems is subjected to different types of faults [39]–[41]. The faults may occur at run time due to hardware defects, electromagnetic interference, cosmic ray radiations, etc. [5], [6], [42]. In this article, we consider transient faults that are a major reliability concern in current digital systems [6], [7], [41], [43]. Transient faults are usually modelled as a Poisson process with a fault rate that is given as a function of the supply voltage changes, as is used in many related works [20], [23]. In our evaluations in Section V, we assume that the transient fault rate is equal 10^{-7} faults per second [23], [44].

To achieve fault tolerance against transient faults, we deploy well-established mechanisms such as the primary-backup technique [19], [16], [20], [23]. To realize this technique in multicore systems, the dynamic core coupling (DCC) technique [19] can be used, to create a core pair. Each core pair includes two cores C1 and C2 where the original tasks are executed on C1 and redundant tasks are executed

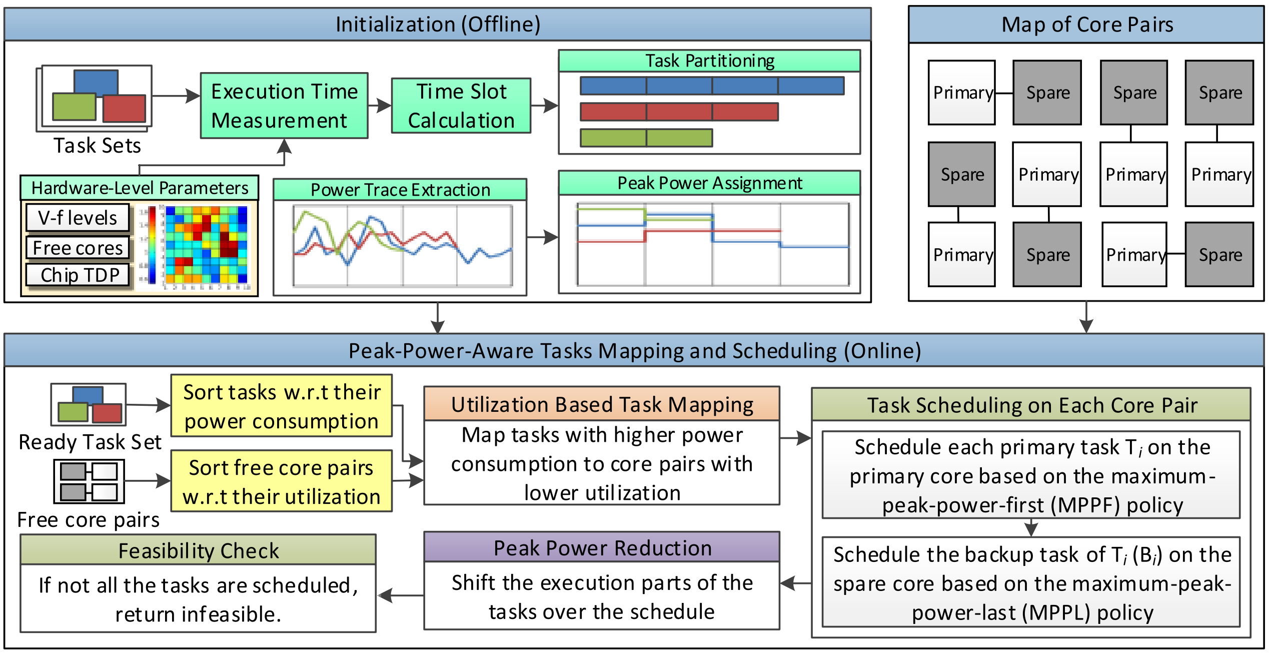

Initialization (Offline)
Task Sets
Execution Time Measurement
Time Slot Calculation
Task Partitioning
Hardware-Level Parameters
V-f levels
Free cores
Chip TDP
Power Trace Extraction
Peak Power Assignment
Map of Core Pairs
Primary
Spare
Peak-Power-Aware Tasks Mapping and Scheduling (Online)
Ready Task Set
Sort tasks w.r.t their power consumption
Free core pairs
Sort free core pairs w.r.t their utilization
Utilization Based Task Mapping
Map tasks with higher power consumption to core pairs with lower utilization
Task Scheduling on Each Core Pair
Schedule each primary task T_i on the primary core based on the maximum-peak-power-first (MPPF) policy
Schedule the backup task of T_i (B_i) on the spare core based on the maximum-peak-power-last (MPPL) policy
Feasibility Check
If not all the tasks are scheduled, return infeasible.
Peak Power Reduction
Shift the execution parts of the tasks over the schedule

FIGURE 2. Operational flow of our technique illustrating the offline and online steps.

on C2. This provides flexibility in establishing core pairs to execute original and redundant tasks. This has two main benefits: *i*) the system is able to tolerate permanent and transient faults and *ii*) hot spots can be effectively reduced by executing tasks with high peak power on physically distant cores.

The primary-backup technique requires a fault detection method. For this purpose, processing cores typically employ a low-cost hardware checker such as Meixner [45]. Argus provides low-cost, comprehensive, low power and high accuracy fault detection [20]. Argus provides run-time checking of control flow, computation, data flow and memory invariants. Meixner *et al.* in [45] have shown that checking these four invariants is sufficient for detecting all possible single errors. In this article, if during the execution of a task no fault has been detected, its results are supplied to the system and its backup task is cancelled. However, when a fault occurs, we ignore the faulty task and continue with its backup task and consider the results of the backup task as the correct results. Since the occurrence of a fault is indeterminate and considering the acceptance test for each part of tasks incurs the significant time and power overhead, the acceptance test is ran at task completion times. In addition, due to the data dependencies between the partitions of the tasks and transmitting the data of the mentioned partitions to the corresponding core for each task, partitions cannot be canceled when their corresponding part is completed. Therefore, when the first copy of a task completes successfully, the remaining parts of its corresponding copy are canceled.

IV. PEAK-POWER-AWARE PRIMARY-BACKUP TECHNIQUE

We propose a scheduling algorithm for frame-based task sets on multicore systems when the primary-backup technique is used for fault tolerance. Our proposed technique consists of the following steps:

Step-1: As shown in Figure 2, in the initialization step, at first, our proposed technique measures the tasks' execution times and extracts the tasks power profiles using offline profiling.

Step-2: Each task is divided into execution parts (task partitioning) and by the use of the power profile of each task, our technique determines peak power value for each part of the tasks (peak power assignment). To do this, for each part of a task, it considers the maximum instantaneous power as the peak power of the whole part. The task partitioning method consists in dividing a task into several subtasks that can be executed separately. Partitioning a task has both positive and negative effects: (*i*) positive effects: task partitioning might reduce physical interference between subtasks, enhance the exploitation of specialization, and increase efficiency; (*ii*) negative effects: task partitioning may incur overheads. As a result, whether task partitioning is useful or not has to be evaluated on a real-world task. For example, we have partitioned the application TIFF into two parts in Figure 1c such that the TDP constraint is met.

Step-3: At run time, when a task set becomes ready for execution, at first, to keep a balance between the power consumption and utilization of the cores, tasks are mapped with higher power consumption to core pairs with lower utilization (Utilization Based Task Mapping). Then, the tasks' ordering in the schedule of each core pair is determined considering the maximum-peak-power-first (MPPF) policy for the original tasks and the maximum-peak-power-last (MPPL) policy for their redundant tasks (Task Scheduling on each Core Pair). Finally, the parts of the tasks are spread over the schedule such that the peak power consumption is kept below the chip's

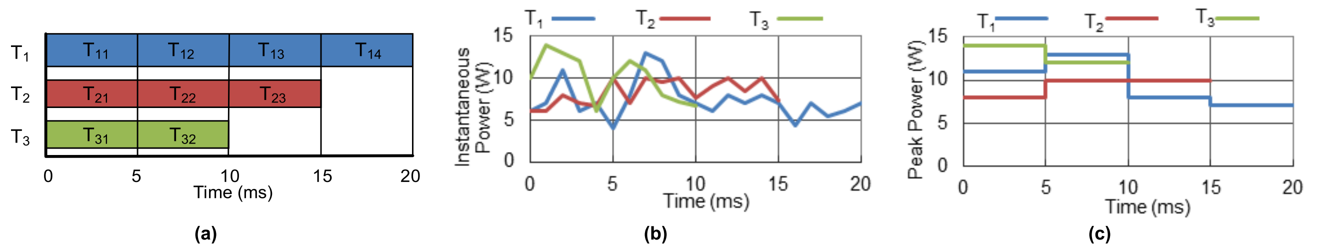


FIGURE 3. Offline part of our system. (A) Dividing the tasks into parts, (b) Power profiles of the tasks, (c) Peak power values for each part.

TDP considering the worst-case fault scenario where all the original and redundant tasks are executed. This scenario, though pessimistic, guarantees that the maximum power consumed by the system is kept below the chip TDP. To achieve further power reduction, if during the execution of the first copy of a task no fault has occurred, the second copy of the task is not required, and then its execution is canceled.

In the following, we use an example to illustrate how our proposed scheme works.

A. AN ILLUSTRATIVE EXAMPLE TO DEMONSTRATE THE CONCEPT AND FUNCTIONING OF OUR TECHNIQUE

As an example, let us consider a frame-based task set consisting of three soft real-time tasks T_1, T_2 and T_3 with the execution time $\tau_1 = 20$ms, $\tau_2 = 15$ms and $\tau_3 = 10$ms that share a common deadline $D = 60$ms. In this example, we consider a dual-core chip with 20W of TDP where the cores (C1 and C2) constitute a core pair.

1) **Initialization Step (Design Time):** As shown in Figure 3, At first, our proposed technique calculates the task partitioning time slot PS as the greatest common divisor (GCD) of the execution time of the tasks, so we have: $PS = 5$ms. By the use of $PS = 5$ms, the tasks T_1, T_2 and T_3, are divided respectively into 4, 3 and 2 parts, as shown in Figure 3a (the j^{th} part of T_i is denoted by T_{ij}). Then, by the use of the tasks power profiles in Figure 3b, the peak power values for the parts of the tasks are determined (Figure 3c). For instance, based on the power profile of the first part of T_1 (T_{11}) between time $t = 0$ ms and 5ms in Figure 3b, the maximum instantaneous power of T_{11} (marked in Figure 3b) is considered as the peak power value for the whole part T_{11} in Figure 3c. In this step, like the original tasks T_1, T_2 and T_3, their redundant tasks B_1, B_2 and B_3 are also divided into parts. For example, like T_1, B_1 is divided into 4 parts, B_{11}, B_{12}, B_{13} and B_{14}. Also, the same peak power values in Figure 3c are used for the parts of the redundant tasks.
2) **Task Scheduling:** Based on the MPPF policy, T_3, T_1 and T_2 are respectively selected to be scheduled on the primary core from the beginning of the execution frame. Also, for the spare core, based on the MPPL policy, the redundant tasks B_3, B_1 and B_2 are respectively selected to be scheduled from the end of the execution frame. Accordingly, at first, the parts T_{31} and T_{32} of the first original task T_3 are placed at the beginning of the schedule of the primary core. Then, the parts B_{31} and B_{32} of B_3 (i.e., the redundant task for T_3) are placed at the end of the schedule of the spare core. For the next original task T_1, the parts T_{11}, T_{12}, T_{13} and T_{14} are placed in the time slots between 10ms and 30ms on the schedule of the primary core. For the redundant task of T_1 (B_1), its parts are placed in the time slots between 30ms and 50ms on the schedule of the spare core. For the last selected original task (T_2), its first part T_{21} is placed in the time slot [30ms, 35ms]. However, if the next part of T_2 (T_{22}) is placed in the next free time slot [35ms, 40ms], the chip TDP is violated (this condition is shown in Figure 4a). This is because, in this case, T_{22} is executed in parallel with B_{12} on the spare core, and hence, the total peak power is 23W that is higher than the TDP value (B_{12} consumes 13W and T_{22} consumes 10W of peak power).
To find a suitable time slot in which the peak power is less than or equal to 20W, our technique checks the next free time slots. Therefore, T_{22} is placed in the time slot [40ms, 45ms] where T_{22} is executed in parallel with B_{13} and the peak power is 18W (i.e., less than TDP). After placing T_{22}, the last part of T_2 (T_{23}) is placed in the next free time slot [45ms, 50ms]. For the redundant task of T_2 (B_2), the parts B_{23} and B_{22} are placed in the time slots [25ms, 30ms] and [20ms, 25ms] in the schedule of the spare core, respectively. This can be done because when B_{22} and B_{23} are executed in parallel with T_{13} and T_{14}, the peak power is less than 20W (as shown in Figure 4a). After placing B_{22} and B_{23}, the next free time slot is [15ms, 20ms]. However, the next part of B_2(B_{21}) cannot be placed there, because if B_{21} is executed in parallel with T_{12}, the peak power will be 21W (T_{12} consumes 13W and B_{21} consumes 8W of peak power), and hence, the chip TDP is violated. Therefore, in order to meet the TDP constraint, our technique checks the next free time slots to find a suitable time slot in which the peak power is less than or equal to 20W. Finally, B_{21} is placed in the time slot [10ms, 15ms] where B_{21} on the spare core is executed in parallel with T_{11} on the primary core and the peak

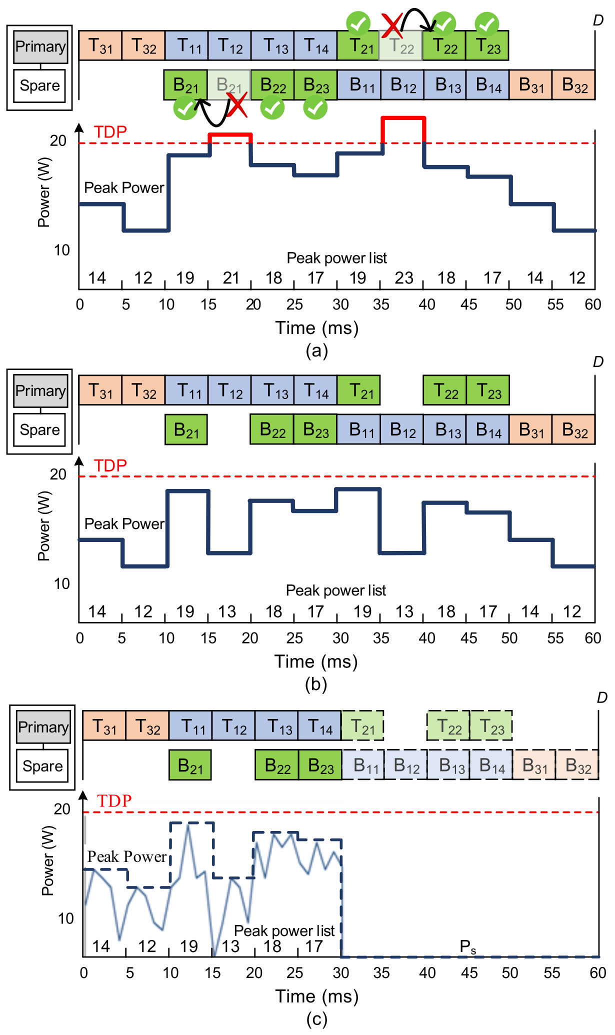


FIGURE 4. **Scheduling the tasks of the example in Section IV.A. a) Shifting the parts of B3 and B1 on C2, respectively, b) Final schedule for the worst-case fault scenario, c) Fault-free execution.**

power is 19W (i.e., less than the TDP). Figure 4b shows the final schedule where the peak power consumption of the system is kept below the TDP and even if both the original and redundant copies of all of the tasks are executed (i.e., under the worst-case fault scenario). As shown in this figure, this schedule meets the timing and TDP constraints.

3) **Tasks Execution (Run time):** Figure 4b shows the maximum power consumed by the system in the worst-case fault scenario where all the original and redundant tasks are executed completely. However, at run time, when during the execution of a task no fault occurs, i.e., the predominant execution scenario [50], the execution of the second copy of the task will be cancelled. Figure 4c shows the case where no fault occurs during the execution of the tasks. In Figure 4c, the dashed line shows the power values which is considered in our scheme, and the straight line shows the real power values, at run time. When T$_3$ on the primary core finishes successfully at $t = 10$ms, its redundant B$_3$ (B$_{31}$ and B$_{32}$) is dropped from the schedule of the spare core. Also, when T$_1$ finishes successfully at $t = 30$ms, all parts of its redundant B$_1$ will be canceled. As we explained earlier, in our system, a redundant task may finish before its original task finishes (even before it starts). No matter what copy of a task finishes first, after finishing the first copy of a task, if no fault has occurred, the second copy is dropped from the schedule to avoid extra power consumption. For instance, as shown in Figure 4c, when the redundant task B$_2$ finishes successfully at $t = 30$ms, our technique drops all parts of the original task T$_2$ from the schedule of the primary core.

B. FORMAL PROBLEM DEFINITION

We formulate the problem using integer linear programing (ILP), and use the following notation to demonstrate the system power consumption and task-to-core mapping [46]. In this formulation, n is the number of ready tasks in the task set, m is the number of execution parts of the tasks, p is the number of free core pairs and t is the number of time slots in each execution frame of the task set:

- The peak power consumption is represented by the matrix $PP \epsilon \mathbb{R}^{n \times m \times p \times t}$, in which each element PP_{ijkl} denotes the peak power consumption of the time slot l when the j^{th} part of the task T$_i$ (T$_{ij}$ or B$_{ij}$) is executed on the core k.
- The task-to-core-pair mapping is represented by the matrix $X \epsilon \{0,1\}^{n \times m \times p \times t}$. The j^{th} part of the task T$_i$ (T$_{ij}$ or B$_{ij}$) is mapped to the core k at the time slot l if and only if $X_{ijkl} = 1$.

The goal of our technique is to keep the total peak power consumption under a given TDP constraint and to meet tasks timing constraints (deadlines).

1) SATISFIABILITY GOAL

The peak power consumption of the chip (i.e., the sum of the peak power of all underlying cores in each time slot l) should be less than the chip TDP constraint:

$$\forall l : \sum_{i,j,k} X_{i,j,k,l} PP_{i,j,k,l} \leq PP_{TDP,chip}. \tag{2}$$

Also, peak power consumption of each core in each time slot l should be less than the core's TDP constraint (if given):

$$\forall l : X_{i,j,k,l} PP_{i,j,k,l} \leq PP_{TDP,l}. \tag{3}$$

Therefore, this is a 0-1 assignment problem and we have:

$$\forall i, j, k, l : X_{i,j,k,l} \in \{0, 1\}. \tag{4}$$

2) TIMING CONSTRAINT

All execution parts of all tasks in the task set have to finish before the deadline. In Eq. 5, FT_l is the finish time of the time slot l.

$$\forall i, j, k, l : X_{i,j,k,l} FT_l \leq D. \tag{5}$$

3) TASK TIME SLOT ORDERING

In order to guarantee that the order of time slots of a task is met, we check that the next part of each task $T_{i(j+1)}$ does not schedule before the previous parts of the selected task.

$$\forall i,j : t_{T_{ij}} < t_{T_{i(j+1)}}. \tag{6}$$

4) CORE ASSIGNMENT CONSTRAINT

Each execution part of each task can be only mapped to one core.

$$\forall i,j,l : \sum_{k} X_{i,j,k,l} = 1. \tag{7}$$

Since solving the above ILP problem and finding a schedule for a multicore system to optimally minimize peak power consumption is an NP-hard problem [1], [13], [47], we present a heuristic to provide a solution for peak power reduction (see Section IV.C).

C. ALGORITHM AND DISCUSSION

The online part of our proposed scheme consists of two steps: *i*) partitioning ready task sets and mapping them to appropriate core pairs and *ii*) partitioning the tasks to execution parts and scheduling the parts of the tasks such that chip TDP and the deadline constraint of the task set are not violated. For task mapping, we select the core pair with the lowest utilization to execute the selected ready task. We do this to distribute the workload evenly between the cores. For task partitioning, the greatest common divisor (GCD) of the execution time of the tasks is considered as the partitioning time slot (PS) and, the tasks are scheduled based on the maximum-peak-power-first (MPPF) and maximum-peak-power-last (MPPL) policies. Finally, if it is required, the tasks execution parts are shifted over the schedule such that the peak power consumption is kept bellow the chip's TDP.

1) INITIALIZATION STEP (LINES 1-9)

Algorithm 1 shows the pseudo-code of the online task mapping and scheduling part of our technique. It gives the task partitioning and peak power values of the tasks' execution parts that are determined at design time and schedules a ready task set Ψ on a multicore system Φ. It gives the schedules $S_{primary}$ and S_{spare} for each core pair in Φ. At first, the tasks are sorted with respect to their maximum power consumption in line 1. The size of *PS* is the greatest common divisor (GCD) of the execution time of the tasks in line 2. Using *PS*, the execution frame (with the length of *D*) is divided into $h = D/PS$ slots in line 3. To determine the peak power consumption of the system in each time slot, we use an array including *h* slots with the initial value of 0 (i.e., the power consumption list *PL* in line 4). Next, the algorithm iterates until all of the tasks are scheduled (lines 5-38). In each iteration of the **while** loop, we select the task with the largest peak power value in the task list Ψ (line 6). In lines 7 and 8, we divide the original and redundant copies of the selected task (T_i and B_i) into parts with size *PS* (τ_i/PS is the number of execution parts).

Algorithm 1 The Online Part of Our Technique

Inputs: Set of ready tasks $\Psi = \{T_1, T_2, \ldots, T_n\}$ with the execution time $\{\tau_1, \tau_2, \ldots, \tau_n\}$ and a common deadline *D*, set of free core pairs $\Phi = \{CP_1, CP_2, \ldots, CP_k\}$, tasks' power profile, and chip TDP value.
Output: The task scheduling $S_{primary}$ and S_{spare} for the primary and spare cores in each core pair *CP*.

```
BEGIN:
1:  $\Psi.sort()$;                              //Sort tasks w.r.t the max. power
2:  $PS = GCD(\tau_i, 1 \le i \le n)$;          //Calculate the partitioning time slot
3:  $h = D/PS$;                                 //Total # of time slots in the frame
4:  $PL[1\ldots h] = \{0\}$;                    //Initialize the total power consumption list
5:  while($\Psi \neq \emptyset$) do
6:    $T_i = \Psi.remove()$;                    //Select the task with the max. power
7:    $T_i = \{T_{ij}, 1 \le j \le \tau_i/PS\}$   //Divide $T_i$ into parts with the size PS
8:    $B_i = \{B_{ij}, 1 \le j \le \tau_i/PS\}$   //Divide $B_i$ into parts with the size PS
9:    $\varphi = min_{utilization}\{CP_l, 1 \le l \le k\}$;
- -   //Scheduling $T_i$ on $\varphi.S_{primary}$ based on the MPPF policy
10:   $k = 1$;
11:   foreach part $T_{ij}$ starting from the first part do
12:     foreach free slot $s = k \rightarrow h$ in $\varphi.S_{primary}$ do
13:       if $PL[s]+peak\_power(T_{ij}) \le TDP$ then
14:         $\varphi.S_{primary}.add(s, T_{ij})$;
15:         $PL[s] = PL[s] + peak\_power(T_{ij})$;
16:         $k = s+1$;
17:         break;
18:       end if;
19:     end for;
20:     if $T_{ij}$ has not been placed then
21:       return infeasible;
22:     end if;
23:   end for;
- -   //Scheduling $B_i$ on $\varphi.S_{spare}$ based on the MPPL policy
24:   $k = h$;
25:   foreach part $B_{ij}$ starting from the last part do
26:     foreach free slot $s = k \rightarrow 1$ in $\varphi.S_{spare}$ do
27:       if $PL[s]+peak\_power(B_{ij}) \le TDP$ then
28:         $\varphi.S_{spare}.add(s, B_{ij})$;
29:         $PL[s] = PL[s] + peak\_power(B_{ij})$;
30:         $k = s-1$;
31:         break;
32:       end if;
33:     end for;
34:     if $B_{ij}$ has not been placed then
35:       return infeasible;
36:     end if;
37:   end for;
38: end while
END
```

To provide core usage efficiency, we schedule the selected task on the core pair with the lowest utilization (denoted by φ in line 9).

2) SCHEDULING THE ORIGINAL TASKS BASED ON MPPF (LINES 10-23)

The parts of T_i beginning from the first one, are placed in the time slots that come earlier in the schedule $\varphi.S_{primary}$ (lines 11-23). We use the variable *k* to determine the first time slot in which the current part of T$_i$ (T$_{ij}$) can be placed. When T$_{ij}$ is placed in a time slot s, the next part (T$_{ij+1}$) can be placed in a time slot starting from the next time slot $s+1$. Starting from k, we check free time slots one after another and place each part T$_{ij}$ in the first free time slot s ($s = k \rightarrow h$) in which the peak power consumption of T$_{ij}$ does not increase the total power consumption beyond the chip TDP ($PL[s]$+$peak\_power$(T$_{ij}$) $\leq TDP$). In this case, T$_{ij}$ is placed in the s^{th} time slot of $\varphi.S_{primary}$ in line 14. The power consumption list is updated in line 15 and the variable k is updated in line 16. If there is no such time slot, the algorithm is not feasible and returns in line 21.

3) SCHEDULING REDUNDANT TASKS BASED ON MPPL (LINES 24-37)

After scheduling the original tasks, we place the parts of the redundant tasks, beginning from the last part, in time slots that come later in the schedule $\varphi.S_{spare}$ (lines 25-37). To do this, the variable k determines the free time slots starting from the end of the execution frame (i.e., k is initialized to h in line 24). Therefore, when the current part of the redundant task B$_i$ (B$_{ij}$) is placed in a time slot s, the next part B$_{ij+1}$ can be placed in the next time slot starting from s-1. We check the free time slots one after another starting from k and place B$_{ij}$ on the first free time slot s ($s = k \rightarrow 1$) in which the peak power consumption of B$_{ij}$ does not increase the total power consumption beyond the chip TDP (i.e., $PL[s]$+$peak\_power$(B$_{ij}$) $\leq TDP$). In this case, B$_{ij}$ is placed in the time slot s of $\varphi.S_{spare}$ in line 28. The power consumption list is updated in line 29 and the variable k is updated in line 30. If B$_{ij}$ has not been placed in any time slot on $\varphi.S_{spare}$, the algorithm is infeasible and returns to line 35. Finally, it should be noted that Algorithm 1 can be applied to task graph applications such that the dependencies between the tasks should be considered for execution.

At run time, when a copy of a task that comes earlier in the schedule finishes (regardless of whether the task is original or redundant), our technique performs fault detection to check the correctness of the task execution. If no fault has occurred, the other copy of the task is cancelled to prevent consuming extra power. However, if the first copy of the task is faulty, we continue executing the second copy to achieve fault tolerance execution. Also, in our system, when a core is temporarily idle (i.e., there is no task for execution) the core is put into sleep mode to further reduce power consumption. It should be noted that the time complexity of our proposed algorithm depends on the partitioning size. If the partitioning size is very small, the number of context switches between partitions increases. By using the GCD of the tasks' durations, we determine the points in each program that can be used as possible partitioning points (but they are not necessarily used for partitioning). As the final discussion of this section, we discuss the time incurred to meet the TDP, timing and reliability constraints simultaneously. Since we shift some tasks to the next time slots to reduce peak power, we require more time slots for meeting the deadlines. Therefore, our proposed scheme incurs more time overhead as compared to other schemes that consider fewer constraints, e.g., the references [16] and [20].

D. ANALYSIS OF TIME COMPLEXITY

In the algorithm, n is the number of frame-based tasks, k is the number of free core pairs, m is the number of free cores, and h is the total partitioning time slots. In Algorithm 1, the ready tasks are sorted in O(n log (n)). The main While loop iterates for O($n \times m \times h$) times. The For loops iterates for O($n \times k \times h$) times. Finally, the algorithm schedules the tasks such that the TDP constraint is met. This step is also done in O($m \times n \times h$). Therefore, the order of the algorithm is max{O(n log (n)), O($m \times n \times h$), O($n \times k \times h$)}.

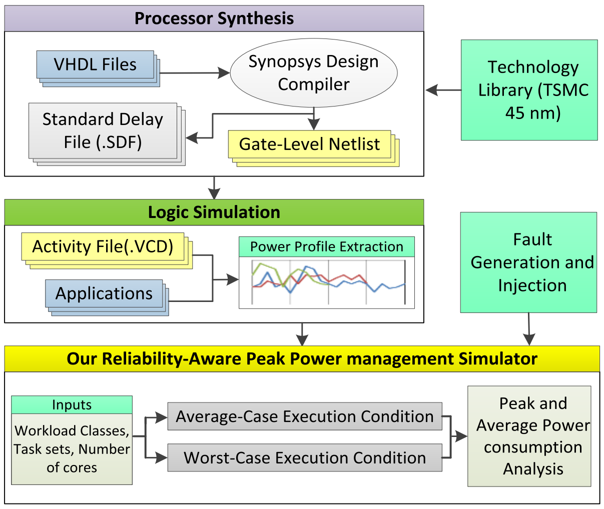

Processor Synthesis
VHDL Files
Synopsys Design Compiler
Standard Delay File (.SDF)
Gate-Level Netlist
Technology Library (TSMC 45 nm)
Logic Simulation
Activity File(.VCD)
Power Profile Extraction
Applications
Fault Generation and Injection
Our Reliability-Aware Peak Power management Simulator
Inputs
Workload Classes, Task sets, Number of cores
Average-Case Execution Condition
Worst-Case Execution Condition
Peak and Average Power consumption Analysis

FIGURE 5. Our tool flow for processor synthesis, scheduling simulation, and power and energy evaluation.

V. RESULTS AND DISCUSSION

A. EXPERIMENTAL SETUP

Figure 5 shows our tool flow and simulation setup with processor synthesis, logic simulation, fault generation and injection, scheduling simulation, and power evaluation. To evaluate the efficiency of our technique, we developed a system-level simulator that is equipped with power and performance characteristics for the LEON3 processor [48] obtained through ASIC synthesis. To do this, we synthesized a VHDL implementation of LEON3 with Synopsys Design Compiler and a TSMC 45nm low-power standard cell library. The processor features direct-mapped, 4Kbyte/set and 32Byte/line instruction and data caches. The Memory Management Unit (MMU) includes 8-entry instruction and data TLB entries with a fast write buffer and 4kMMU page size.

To realize a wide range of competing real-world application scenarios, we selected several benchmark applications from different program groups of MiBench [49] including automotive, consumer, network, office, security

TABLE 1. The selected benchmark applications from different program groups of MiBench [49].

Program group / Bench. Name	Automotive	Consumer	Network	Office	Security	Telecomm
MiBench	qsort	jpeg	dijkstra	ispell	sha	adpcm
	susan	tiff	patricia	ghostscript	pgp	CRC32
	bitcount	mad		rsynth	rijndael	FFT
	basicmath	lame		sphinx	blowfish	gsm
				stringsearch		

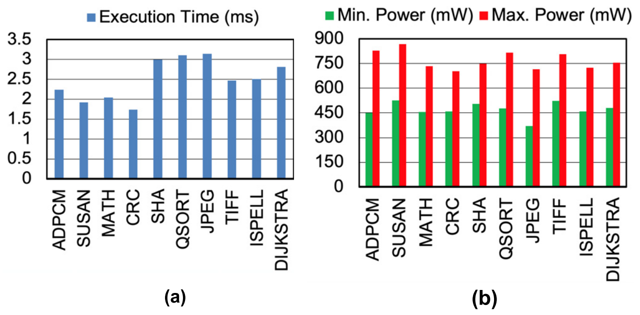


FIGURE 6. Characteristics of the benchmark applications. (a) Execution time, (b) Maximum and minimum power consumption.

and telecommunication. The MiBench Benchmark suite has been widely used in previous related works [1], [18], [50]. MiBench has different program groups and to have a fair analysis, we used applications from all of the different program groups including automotive, consumer, network, office, security, and telecommunication. We have shown the selected benchmark applications from different program groups of MiBench with the red color in Table 1. It should be noted that in embedded real-time systems, such features and timing constraints of the system are known at design time to ensure guarantees during their real-world operations [51], [52]. The worst-case execution time, the peak power consumption, and energy consumption of applications can be calculated at design time, as it has been demonstrated by various leading groups [2], [53]–[55], and industries related to automotive and industrial systems that strictly consider such worst-case analysis at design time. Therefore, we conducted experiments on various task sets including real-life embedded applications from the MiBench Benchmark suite [49]. The execution time and power consumption of the tasks were obtained through processor synthesis and gate-level simulations (the processor operates at 1.23Volt and 970MHz). In our experiments, we used the parameter core utilization (L) to consider different workloads in the simulations. It should be noted that this parameter determines the maximum (but not the absolute) core utilizations in each simulation case. The cores are set to different utilizations, but the utilization values can vary up to L. To study the effects of system workload on our scheme, we considered four core workload classes: $L = 0.6$, 0.7, 0.8 and 0.9. The workload values determine the cores' utilization in the experiment (e.g., for the workload $L = 0.6$ the utilization of each core is considered to be 0.6). For each workload value, we generated 100 task sets from the tasks in V-B and the average results were reported, similar to [50]. We also considered different chips with $m = 4$, 8 and 16 cores and with different TDP values. In addition to the power profiles of the tasks in V-B b that were derived for a given input data set, we generated several random power profiles for the tasks to realize a good coverage of real-world input data sets. To generate synthetic power profiles, in a power profile of a task, the instantaneous power consumption was randomly generated between the minimum and maximum power consumption of the task obtained from V-Bb. In this section, in the figures, we called our technique "RAPPM".

We compare our technique with the following state-of-the-art power management techniques:

- [16]-APM: A scheme that addresses average-power management in conjunction with fault-tolerance. This scheme executes original tasks as soon as possible and redundant tasks as late as possible, hoping that the original tasks finish successfully and the redundant tasks cancel to prevent extra power consumption (like that is shown in Figure 1b). We selected [16]-APM for comparison to highlight the important differences between peak-power and average-power management.
- ConvPB [41]: The conventional primary-backup scheme where each redundant task is executed in parallel with its original tasks (like that is shown in Figure 1a).

The comparisons are performed for two scenarios, i.e., (1) the worst-case fault condition when the system consumes the maximum possible power (Section V.B), and (2) the average-case execution scenario including both faulty and fault-free scenarios (Section V.C).

B. WORST-CASE EXECUTION CONDITION ANALYSIS

The worst-case fault condition where all redundant tasks are executed as well as original tasks, although pessimistic, determines the maximum power consumed by the system. Therefore, it can be considered as a good condition to compare peak power management techniques. Figure 7 shows

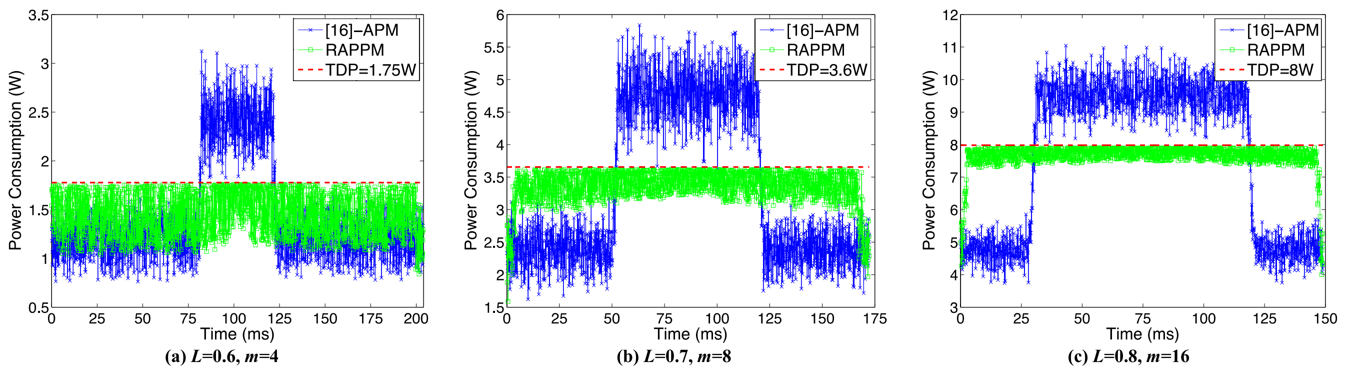


FIGURE 7. Power consumption profile for a frame-based task set in the worst-case execution condition.

the power consumption of our technique and the [16]-APM scheme for different numbers of cores ($m = 4$, 8 and 16) and core workloads ($L = 0.6$, 0.7 and 0.8). As Figure 7 shows, our technique reduces peak power by distributing the power consumption over the whole execution frame. This figure also determines the maximum peak power that is consumed by our technique (the dashed TDP line in Figure 7). This is the minimum possible value for the TDP constraint of the system that can be satisfied by our technique. For example, when our technique is used for a system with $m = 4$ cores under a core workload of $L = 0.6$ (Figure 7a), the chip TDP can be as low as 1.75W. However, the [16]-APM scheme misses this TDP constraint.

In Figure 7, for each system configuration we used only one random task set to provide insight into how our technique reduces peak power consumption. To provide a more detailed analysis, for each system configuration we used more task sets and the average results are shown in Figure 8. In this experiment, the minimum peak power consumption of our technique is considered as the chip TDP constraint and the peak power consumption of the schemes are normalized with respect to TDP (the normalized peak power of our technique is 1). The following observations can be made from Figure 8:

- The peak power consumption of our technique is far less than those of the [16]-APM and ConvPB schemes for all system configurations and core workloads. Our technique provides up to 39.1% and 50% peak power reduction compared to the [16]-APM and ConvPB schemes, respectively.
- When the number of cores increases (Figure 8a), the peak power reduction of our technique is higher than that of the other schemes. In this case our technique provides up to 43.8% and 44.7% peak power reduction compared to the [16]-APM and ConvPB schemes, respectively.
- When the core workload increases (Figure 8b), the peak power consumption of our technique increases and the difference between the three schemes reduced. However, our technique still outperforms the two schemes. In this case, our technique provides up to 44.75% and on average by 37.31% peak power reduction compared to [16]-APM and ConvPB.

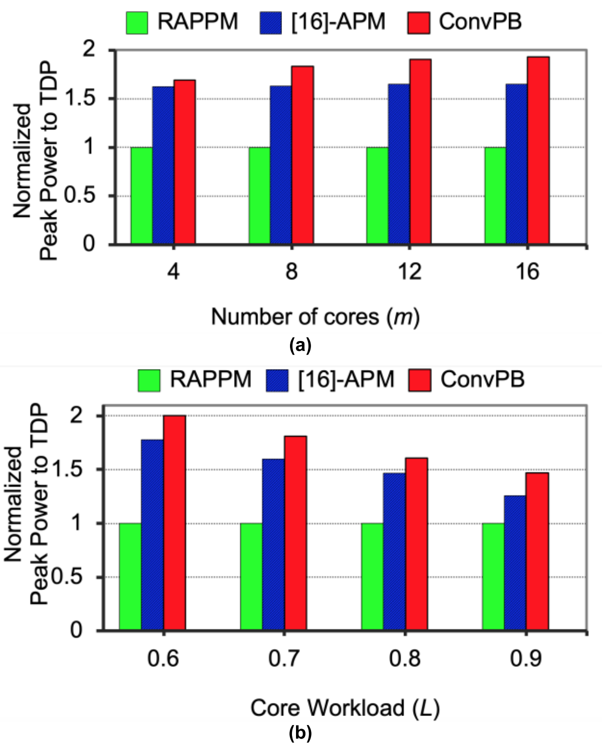


FIGURE 8. Normalized peak power to TDP in the worst-case fault condition. a) Under the core workload $L = 0.7$, b) For the number of cores $m = 8$.

C. AVERAGE-CASE EXECUTION CONDITION ANALYSIS

We study the average-case execution condition where both faulty and fault-free execution scenarios were considered. To do this, we exploited a system-level fault injection where transient fault rate is equal 10^{-7} faults per second [16], [20], [44]. At first, we generate a fault vector that determines at which times faults occur. Then, based on the fault vector, we decide which task becomes faulty during the execution of a task set. Since transient faults are rare, the online part of our technique achieves further power reduction beyond what is achieved through the offline part of our technique at design-time. It cancels unnecessary task execution whenever the first copy of a task finishes successfully, resulting in a significant power reduction. Note

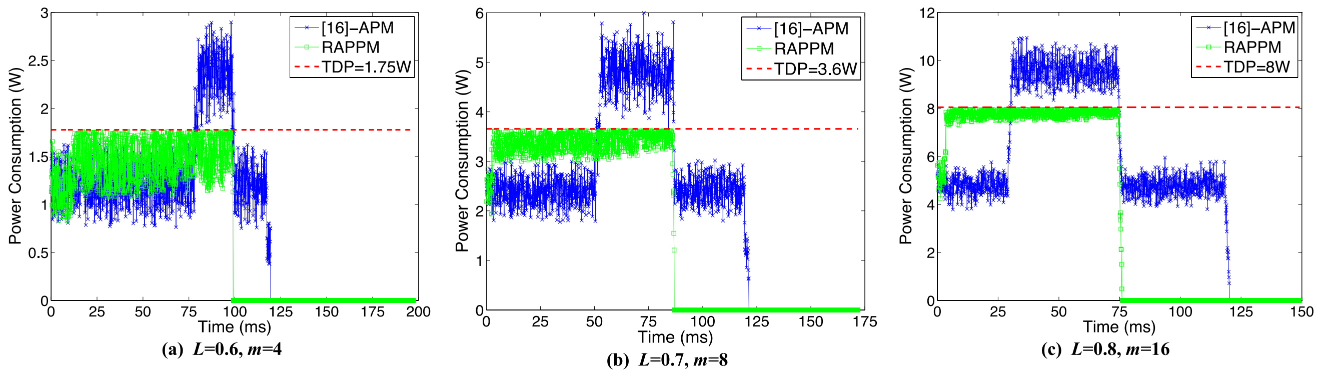


FIGURE 9. Power consumption profile for a frame-based task set in the average-case execution condition under the fault rate = 10^{-7}.

that the offline part of our technique guarantees meeting the TDP constraint. Nonetheless, reducing the average power consumption through the online part of our technique reduces the system energy consumption as well.

Figure 9 shows the execution of the task sets that were deployed in Figure 7, but in the average-case execution condition (where some tasks may become faulty). Recall that in Figure 7 the task sets were executed in the worst-case fault condition where it was considered that all tasks become faulty. As shown in Figure 9, both our technique and the [16]-APM schemes provide relatively less peak power consumption compared to the worst-case fault condition in Figure 7. This is because they cancel the second copy of the tasks that finish successfully. However, the power consumption of the [16]-APM scheme still increases beyond the TDP constraint even when no fault occurs. Also, as it can be seen from Figure 9, both schemes consume no power at the end of the execution frame. This is because, at the end of the execution frame, the first copy of the tasks may have already finished successfully (when no fault occurs) and the second copy of the tasks are cancelled. Therefore, considering that the fault rate is low, almost always at the end of each execution there is no task to be executed and the underlying cores go to sleep mode and consume no power. It should be noted that the deadline of each figure of Figure 9 is the last number of each figure (x-axis).

Figure 10 shows the average power consumption of our technique, [16]-APM and ConvPB schemes where the power consumption values are normalized with respect to the power consumption of our technique (i.e., the normalized power consumption of our technique is 1). As Figure 10 shows, our technique significantly reduces the average power consumption as compared to the [16]-APM and ConvPB schemes (our technique reduces the average power consumption to 30.4% and 49.7% of those of [16]-APM and ConvPB). Also, as Figure 10a shows, under a given core workload, the average power reduction of our technique compared to the other schemes does not change by increasing the number of cores. However, when the core workload increases, our technique provides higher power reduction as compared to [16]-APM (i.e., the normalized average power

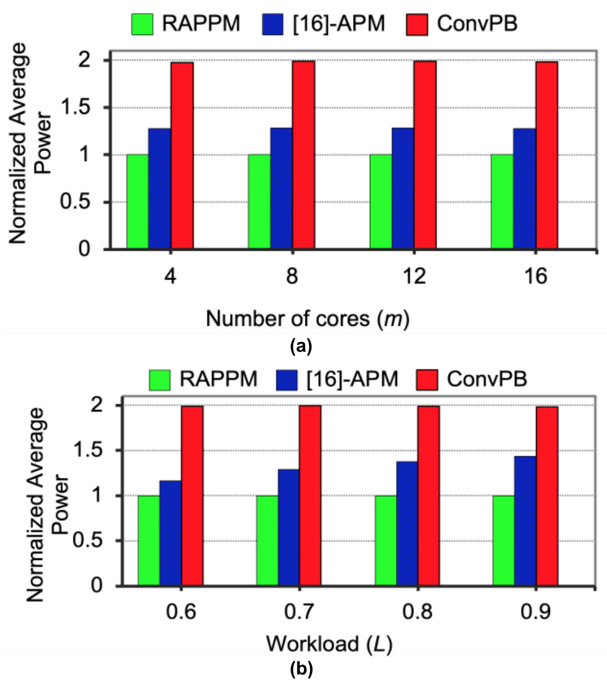


FIGURE 10. Normalized average power to our technique in the average-case execution condition. a) Under the core workload $L = 0.7$, b) For the number of cores $m = 8$.

of [16]-APM increases from 1.16 to 1.44 in Figure 10b). This is because our technique cancels more unused task copies compared to [16]-APM. Note that, in all cases (for all m and L values) the average power consumption of ConvPB is almost twice the average power consumption of our technique. This is because ConvPB always executes all original and redundant tasks, while our technique usually executes only one copy of each task, while providing the same fault tolerance capability. In our technique, the second copy of a task is executed only when a fault occurs during the execution of the first copy of the task. In this case, we also compared the energy consumption of our technique with [16]-APM. The experiments show that our technique completely outperforms [16]-APM from the energy consumption viewpoint. The our technique scheme provides on average respectively 14.3% (up to 28.1%) energy saving as compared to [16]-APM.

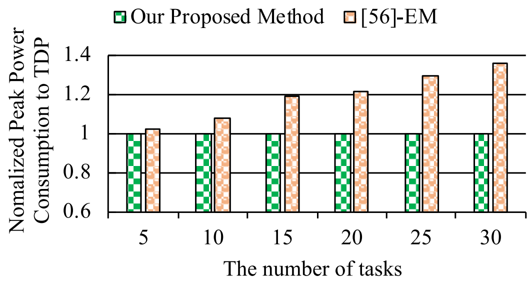


FIGURE 11. Normalized peak power to TDP in the worst-case fault condition comparing between our proposed technique and [56] -EM.

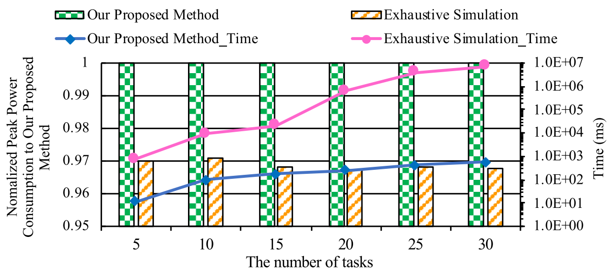


FIGURE 12. Accuracy and run-time efficiency comparison with the exhaustive simulation.

Recently, Roy *et al.* [56] have proposed a primary-backup technique that reduces the energy consumption for heterogeneous multicore systems. In order to provide a fair comparison between our proposed technique and [56]-EM, we apply our task and hardware model to this work and repeated the simulations. For these simulations, we have considered different numbers of tasks ranging from 5 to 30 tasks and compare our proposed technique and [56]-EM in the terms of peak power consumption. As shown in Figure 11, the peak power consumption of our technique is far less than [56]-EM for different number of tasks. Our technique provides up to 35.6% (on average by 19.3%) peak power reduction compared to [56]-EM.

Finally, we have compared our proposed technique to exhaustive simulation for reducing peak power consumption. For these simulations, we have considered different numbers of tasks ranging from 5 to 30 tasks. The experiments in Figure 12 show that our technique deviates by an average of 3.1% in the terms of power efficiency, but speeds up the peak power management decision time by a factor of up to 1230X.

VI. RELATED WORK

Many related works use dynamic voltage and frequency scaling (DVFS) [35], [44], [53], [57] to manage average power consumption [53], [54], [58]. In a multicore system where each core individually uses DVFS, the problem of minimizing the peak power consumption can be considered as the problem of minimizing the average power consumption [59]. However, due to scaling down the supply voltage, the rate of transient faults increases significantly [55], [60]. Moreover, due to chip-area cost and power consumption overhead of on-chip controllable power supplies, per-core DVFS may not be applicable for multicore systems featuring hundred cores [13]. Therefore, in this article, we use Dynamic Power Management (DPM) [61] to reduce the peak power consumption. DPM dynamically turns on/off system components to provide the requested services with a minimum number of active components [61]. In the following, we partition the previous related works into two parts: *i*) Power-Aware Reliability Management, and *ii*) Peak-Power Management.

A. POWER-AWARE RELIABILITY MANAGEMENT

Most of the previous researches focused on average power and energy consumption in fault-tolerant embedded systems [44], [50], [53], [55]. Also, some studies concentrated on thermal management in multicore systems [62], [63]. Fisher *et al.* [62] have proposed a global thermal-aware scheduling for sporadic tasks to minimize temperature. Jejurikar *et al.* [63] minimize energy consumption by considering the real-time constraints by using a deferment interval for each task. However, most of the studies in this field have focused on reducing the average power and energy consumption and have not considered peak power management.

B. PEAK-POWER MANAGEMENT

Some related works focused on minimizing the peak power consumption under real-time constraints [1], [13], [64]. Lee *et al.* [1] have proposed a new scheduling algorithm for real-time tasks to minimize chip-level power consumption, without relying on any extra hardware (e.g., DVFS controller). This work restricts the concurrent execution of tasks that are assigned to different cores. Lee *et al.* [64] has proposed a task scheduling that prevents the occurrence of the peak power consumption for task-graph models. The proposed algorithm in [64] schedules the tasks, considering data dependency information while reducing the peak power. As one of the most related work, Munawar *et al.* [13] have presented a scheme to minimize the peak power for frame-based and periodic tasks with real-time constraints on multicore systems. They schedule the sleep cycles for each active core to manage the peak power. These works try to minimize peak power and do not consider fault tolerance against transient faults. In this article, we consider peak power management for a fault-tolerant technique (the primary-backup technique) on multicore systems. Recently, Ansari *et al.* [18] proposed a peak-power-aware reliability management method that manages peak power overlaps between concurrently executing tasks such that the system reliability is preserved at an acceptable level while guaranteeing to keep the total power consumption of cores below the chip TDP and the power consumption of each underlying core below the core TDP constraint. It should be noted that the application model in [18] is task graph and hard real-time, while the model in this article is frame-based and soft real-time. It should be noted that in the context of real-time systems, this difference in application model is a fundamental switch from hard real-time to soft real-time, which is a shift from literature, e.g., from [31] to [32]. This difference in application model results in difference in almost all the other aspects including scheduling policies, energy management, experiments and experimental setup, etc. Another important difference is that the work in [18] considers the NMR technique for fault avoidance, while in this article we consider the primary-backup technique for fault tolerance, which is quite different from NMR and changes the problem fundamentally. Furthermore, unlike the work in [18], in this article we only exploit dynamic power management because per-core DVFS (used in [18]), due to chip-area cost and power consumption overhead of on-chip controllable power supplies, may not be applicable for multicore systems featuring hundred cores [13]. Generally, in this article, we exploit the primary-backup technique to achieve high reliability for multicore embedded systems and propose a scheme to keep the chip-level peak power consumption under its TDP constraint.

Note that in this article, we have solved the peak power problem using a primary-backup technique for a soft real-time non-preemptive frame-based task model. In the context of real-time systems, the difference in the objective function and application model is very fundamental. Indeed, different objective functions and application models result in difference in almost all other aspects including scheduling policies, power management, experiments, experimental setup, etc. To do this, in this article, we have proposed a peak-power-aware primary-backup technique that enables task-level redundancy to achieve fault tolerance in multicore embedded systems under timing and TDP constraints.

VII. CONCLUSION

We proposed a peak-power-aware primary-backup technique that manages peak power consumption for reliable task execution (i.e., using the primary-backup technique) on multicore systems through managing time overlaps between concurrently executing real-time tasks. To do this, we partition original and redundant tasks, and schedule them on core pairs in a multicore system under chip's TDP constraint without violating the tasks' timing constraints. For the offline part of our technique, we developed maximum-peak-power-first (MPPF) and maximum-peak-power last (MPPL) policies to minimize peak power overlap between original and redundant tasks. This part guarantees meeting the chip TDP in the worst-case fault condition. For the online part of our technique, we exploit a scheme that provides further power reduction in the average-case execution condition. It cancels the execution of the second copy of those tasks that have no fault in their first copy's execution. The experimental results show that our technique provides up to 50% peak power reduction compared to state-of-the-art schemes. In summary, our technique enables task-level redundancy to achieve reliability improvement in multicore systems under tight power consumption constraints.

REFERENCES

[1] J. Lee, B. Yun, and K. G. Shin, "Reducing peak power consumption inMulti-core systems without ViolatingReal-time constraints," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 4, pp. 1024–1033, Apr. 2014.

[2] A. Munir, S. Ranka, and A. Gordon-Ross, "High-performance energy-efficient multicore embedded computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 4, pp. 684–700, Apr. 2012.

[3] J. Henkel, V. Narayanan, S. Parameswaran, and J. Teich, "Run-time adaption for highly-complex multi-core systems," in *Proc. Int. Conf. Hardw./Softw. Codesign Syst. Synth. (CODES+ISSS)*, Sep. 2013, pp. 1–8.

[4] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel, "Mapping on multi/many-core systems: Survey of current and emerging trends," in *Proc. 50th Annu. Design Autom. Conf. DAC*, May 2013, pp. 1–10.

[5] Q. Han, M. Fan, and G. Quan, "Energy minimization for fault tolerant real-time applications on multiprocessor platforms using checkpointing," in *Proc. Int. Symp. Low Power Electron. Design (ISLPED)*, Sep. 2013, pp. 76–81.

[6] R. Melhem, D. Mosse, and E. Elnozahy, "The interplay of power management and fault recovery in real-time systems," *IEEE Trans. Comput.*, vol. 53, no. 2, pp. 217–231, Feb. 2004.

[7] S. K. Reinhardt and S. S. Mukherjee, "Transient fault detection via simultaneous multithreading," *ACM SIGARCH Comput. Archit. News*, vol. 28, no. 2, pp. 25–36, May 2000.

[8] S. S. Mukherjee, M. Kontz, and S. K. Reinhardt, "Detailed design and evaluation of redundant multi-threading alternatives," in *Proc. 29th Annu. Int. Symp. Comput. Archit.*, May 2002, pp. 99–110.

[9] A. Shye, T. Moseley, V. J. Reddi, J. Blomstedt, and D. A. Connors, "Using process-level redundancy to exploit multiple cores for transient fault tolerance," in *Proc. 37th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2007, pp. 297–306.

[10] S. Rehman, F. Kriebel, D. Sun, M. Shafique, and J. Henkel, "DTune: Leveraging reliable code generation for adaptive dependability tuning under process variation and aging-induced effects," in *Proc. 51st Annu. Design Autom. Conf. Design Autom. Conf. DAC*, 2014, pp. 1–6.

[11] F. Kriebel, S. Rehman, M. Shafique, and J. Henkel, "AgeOpt-RMT: Compiler-driven variation-aware aging optimization for redundant multithreading," in *Proc. 53rd Annu. Design Autom. Conf. DAC*, 2016, pp. 1–6.

[12] K.-H. Chen, J.-J. Chen, F. Kriebel, S. Rehman, M. Shafique, and J. Henkel, "Task mapping for redundant multithreading in multi-cores with reliability and performance heterogeneity," *IEEE Trans. Comput.*, vol. 65, no. 11, pp. 3441–3455, Nov. 2016.

[13] W. Munawar, H. Khdr, S. Pagani, M. Shafique, J.-J. Chen, and J. Henkel, "Peak power management for scheduling real-time tasks on heterogeneous many-core systems," in *Proc. 20th IEEE Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Dec. 2014, pp. 200–209.

[14] H. Khdr, T. Ebi, M. Shafique, H. Amrouch, and J. H. Karlsruhe, "MDTM: Multi-objective dynamic thermal management for on-chip systems," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2014, pp. 1–6.

[15] K. Ganesan and L. K. John, "MAximum multicore POwer (MAMPO): An automatic multithreaded synthetic power virus generation framework for multicore systems," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal. SC*, 2011, pp. 1–12.

[16] A. Ejlali, B. M. Al-Hashimi, and P. Eles, "Low-energy standby-sparing for hard real-time systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 31, no. 3, pp. 329–342, Mar. 2012.

[17] J. Saber-Latibari, M. Ansari, P. Gohari-Nazari, S. Yari-Karin, A. M. H. Monazzah, and A. Ejlali, "READY: Reliability-and deadline-aware power-budgeting for heterogeneous multi-core systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, early access, Jun. 18, 2020, doi: 10.1109/TCAD.2020.3003288.

[18] M. Ansari, S. Safari, A. Yeganeh-Khaksar, M. Salehi, and A. Ejlali, "Peak power management to meet thermal design power in fault-tolerant embedded systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 1, pp. 161–173, Jan. 2019.

[19] C. LaFrieda, E. Ipek, J. F. Martinez, and R. Manohar, "Utilizing dynamically coupled cores to form a resilient chip multiprocessor," in *Proc. 37th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2007, pp. 317–326.

[20] S. Safari, S. Hessabi, and G. Ershadi, "LESS-MICS: A low energy standby-sparing scheme for mixed-criticality systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, early access, Feb. 28, 2020, doi: 10.1109/TCAD.2020.2977063.

[21] O. S. Unsal and I. Koren, "System-level power-aware design techniques in real-time systems," *Proc. IEEE*, vol. 91, no. 7, pp. 1055–1069, Jul. 2003.

[22] M. K. Tavana, M. Salehi, and A. Ejlali, "Feedback-based energy management in a standby-sparing scheme for hard real-time systems," in *Proc. IEEE 32nd Real-Time Syst. Symp.*, Nov. 2011, pp. 349–356.

[23] M. A. Haque, H. Aydin, and D. Zhu, "Energy-aware standby-sparing technique for periodic real-time applications," in *Proc. IEEE 29th Int. Conf. Comput. Design (ICCD)*, Oct. 2011, pp. 190–197.

[24] J. Howard, S. Dighe, S. R. Vangal, G. Ruhl, N. Borkar, S. Jain, V. Erraguntla, M. Konow, M. Riepen, M. Gries, G. Droege, T. Lund-Larsen, S. Steibl, S. Borkar, V. K. De, and R. Van Der Wijngaart, "A 48-core IA-32 processor in 45 nm CMOS using on-die message-passing and DVFS for performance and power scaling," *IEEE J. Solid-State Circuits*, vol. 46, no. 1, pp. 173–183, Jan. 2011.

[25] Intel Corporation. (2009). *Single-Chip Cloud Computer (SCC)*. [Online]. Available: http://www.intel.com/content/www/us/ en/research/intel-labs-single-chip-cloud-overview-paper.html

[26] W.-M. Chen, T.-S. Cheng, P.-C. Hsiu, and T.-W. Kuo, "Value-based task scheduling for nonvolatile processor-based embedded devices," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, Nov. 2016, pp. 349–356.

[27] B. Zhao, H. Aydin, and D. Zhu, "Enhanced reliability-aware power management through shared recovery technique," in *Proc. Int. Conf. Comput.-Aided Design ICCAD*, Nov. 2009, pp. 63–70.

[28] D. Zhu, R. Melhem, and B. Childers, "Scheduling with dynamic voltage/speed adjustment using slack reclamation in multi-processor real-time systems," in *Proc. 22nd IEEE Real-Time Syst. Symp. (RTSS)*, Dec. 2001, pp. 84–94.

[29] C.-Y. Yang, J.-J. Chen, and T.-W. Kuo, "An approximation algorithm for energy-efficient scheduling on a chip multiprocessor," in *Proc. Design, Autom. Test Eur.*, Mar. 2005, pp. 468–473.

[30] A. Allavena and D. Mosse, "Scheduling of frame-based embedded systems with rechargeable batteries," in *Proc. Workshop on Power Management Real-Time Embedded Syst.*, 2001, pp. 1–8.

[31] G. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. New York, NY, USA: Springer, 2011.

[32] G. C. Buttazzo, G. Lipari, L. Abeni, and M. Caccamo, *Soft Real-Time Systems: Predictability Vs. Efficiency*. Springer, 2005.

[33] M. Ansari, S. Safari, F. R. Poursafaei, M. Salehi, and A. Ejlali, "AdDQ: Low-energy hardware replication for real-time systems through adaptive dual queue scheduling," *CSI J. Comput. Sci. Eng. (JCSE)*, vol. 15, no. 1, pp. 31–38, 2017.

[34] S. Yari-Karin, A. Sahraee, J. Saber-Latibari, M. Ansari, N. Rohbani, and A. Ejlali, "A comparative study of joint power and reliability management techniques in multicore embedded systems," in *Proc. CSI/CPSSI Int. Symp. Real-Time Embedded Syst. Technol. (RTEST)*, Jun. 2020, pp. 1–8.

[35] M. Salehi and A. Ejlali, "A hardware platform for evaluating low-energy multiprocessor embedded systems based on COTS devices," *IEEE Trans. Ind. Electron.*, vol. 62, no. 2, pp. 1262–1269, Feb. 2015.

[36] M. Ansari, M. Pasandideh, J. Saber-Latibari, and A. Ejlali, "Meeting thermal safe power in fault-tolerant heterogeneous embedded systems," *IEEE Embedded Syst. Lett.*, vol. 12, no. 1, pp. 29–32, Mar. 2020.

[37] M. Ansari, J. Saber-Latibari, M. Pasandideh, and A. Ejlali, "Simultaneous management of peak-power and reliability in heterogeneous multicore embedded systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 3, pp. 623–633, Mar. 2020.

[38] M. Ansari, A. Yeganeh-Khaksar, S. Safari, and A. Ejlali, "Peak-Power-Aware energy management for periodic real-time applications," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 4, pp. 779–788, Apr. 2020.

[39] S. Safari, M. Ansari, G. Ershadi, and S. Hessabi, "On the scheduling of energy-aware fault-tolerant mixed-criticality multicore systems with service guarantee exploration," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 10, pp. 2338–2354, Oct. 2019.

[40] S. Safari, M. Ansari, M. Salehi, and A. Ejlali, "Energy- budget-aware reliability management in multi-core embedded systems with hybrid energy source," *CSI J. Comput. Sci. Eng. (JCSE)*, vol. 15, no. 2, pp. 31–43, 2018.

[41] D. Pradhan, *Fault-Tolerant Computer System Design*. Upper Saddle River, NJ, USA: Prentice-Hall, 1996.

[42] Z. Shirmohammadi, M. Ansari, S. K. Abharian, S. Safari, and S. G. Miremadi, "PAM: A packet manipulation mechanism for mitigating crosstalk faults in NoCs," in *Proc. IEEE Int. Conf. Comput. Inf. Technol.; Ubiquitous Comput. Commun.; Dependable, Autonomic Secure Comput.; Pervasive Intell. Comput.*, Oct. 2015, pp. 1895–1902.

[43] Castillo, McConnel, and Siewiorek, "Derivation and calibration of a transient error reliability model," *IEEE Trans. Comput.*, vol. C-31, no. 7, pp. 658–671, Jul. 1982.

[44] F. R. Poursafaei, S. Safari, M. Ansari, M. Salehi, and A. Ejlali, "Offline replication and online energy management for hard real-time multicore systems," in *Proc. CSI Symp. Real-Time Embedded Syst. Technol. (RTEST)*, Oct. 2015, pp. 1–7.

[45] A. Meixner, M. E. Bauer, and D. Sorin, "Argus: low-cost, comprehensive error detection in simple cores," in *Proc. 40th Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, Dec. 2007, pp. 210–222.

[46] J. Chen, L. K. John, and D. Kaseridis, "Modeling program resource demand using inherent program characteristics," in *Proc. ACM SIGMETRICS Joint Int. Conf. Meas. Model. Comput. Syst. SIGMETRICS*, 2011, pp. 1–12.

[47] S. Pagani, H. Khdr, J.-J. Chen, M. Shafique, M. Li, and J. Henkel, "Thermal safe power (TSP): Efficient power budgeting for heterogeneous manycore systems in dark silicon," *IEEE Trans. Comput.*, vol. 66, no. 1, pp. 147–162, Jan. 2017.

[48] *LEON3 Multiprocessing CPU Core*. [Online]. Available: http://www.gaisler.com/doc/leon3_product_sheet.pdf

[49] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *Proc. 4th Annu. IEEE Int. Workshop Workload Characterization (WWC)*, Dec. 2001, pp. 3–14.

[50] M. Salehi, A. Ejlali, and B. M. Al-Hashimi, "Two-phase low-energy N-modular redundancy for hard real-time multi-core systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 4, pp. 1024–1033, Apr. 2015.

[51] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Springer, 2011, p. 378.

[52] P. Marwedel, *Embedded System Design*. Springer, 2006.

[53] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *Proc. 18th ACM Symp. Operating Syst. Princ. (SOSP)*, 2001, pp. 89–102.

[54] J.-J. Chen, S. Wang, and L. Thiele, "Proactive speed scheduling for real-time tasks under thermal constraints," in *Proc. 15th IEEE Real-Time Embedded Technol. Appl. Symp.*, Apr. 2009, pp. 141–150.

[55] D. Zhu, R. Melhem, and D. Mosse, "The effects of energy management on reliability in real-time embedded systems," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, 2004, pp. 35–40.

[56] A. Roy, H. Aydin, and D. Zhu, "Energy-aware standby-sparing on heterogeneous multicore systems," in *Proc. 54th Annu. Design Autom. Conf.*, Jun. 2017, pp. 1–6.

[57] T. D. Burd, T. A. Pering, A. J. Stratakos, and R. W. Brodersen, "A dynamic voltage scaled microprocessor system," *IEEE J. Solid-State Circuits*, vol. 35, no. 11, pp. 1571–1580, Nov. 2000.

[58] N. Fisher, J.-J. Chen, S. Wang, and L. Thiele, "Thermal-aware global real-time scheduling on multicore systems," in *Proc. 15th IEEE Real-Time Embedded Technol. Appl. Symp.*, Apr. 2009, pp. 131–140.

[59] M. A. Al Faruque, J. Jahn, T. Ebi, and J. Henkel, "Runtime thermal management using software agents for Multi- and many-core architectures," *IEEE Des. Test. Comput. Comput.*, vol. 27, no. 6, pp. 58–68, Nov. 2010.

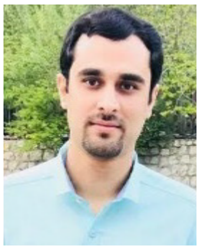
[60] D. Ernst, S. Das, S. Lee, D. Blaauw, T. Austin, T. Mudge, N. S. Kim, and K. Flautner, "Razor: Circuit-level correction of timing errors for low-power operation," *IEEE Micro*, vol. 24, no. 6, pp. 10–20, Nov. 2004.

[61] L. Benini, A. Bogliolo, and G. De Micheli, "A survey of design techniques for system-level dynamic power management," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 8, no. 3, pp. 299–316, Jun. 2000.

[62] N. Fisher, J.-J. Chen, S. Wang, and L. Thiele, "Thermal-aware global real-time scheduling and analysis on multicore systems," *J. Syst. Archit.*, vol. 57, no. 5, pp. 547–560, May 2011.

[63] R. Jejurikar, C. Pereira, and R. Gupta, "Leakage aware dynamic voltage scaling for real-time embedded systems," in *Proc. 41st Annu. Conf. Design Autom. (DAC)*, 2004, pp. 275–280.

[64] B. Lee, J. Kim, Y. Jeung, and J. Chong, "Peak power reduction methodology for multi-core systems," in *Proc. Int. SoC Design Conf.*, Nov. 2010, pp. 233–235.



MOHSEN ANSARI received the M.Sc. degree in computer engineering from the Sharif University of Technology, Tehran, Iran, in 2016, where he is currently pursuing the Ph.D. degree in computer engineering. He is currently a Visiting Researcher with the Chair for Embedded Systems (CES), Karlsruhe Institute of Technology (KIT), Germany. He is also a member of Embedded Systems Research Laboratory (ESR-LAB), Department of Computer Engineering, Sharif University of Technology. His research interests include low-power design of embedded systems and multi-/many-core systems with a focus on dependability/reliability.

MOHAMMAD SALEHI (Associate Member, IEEE) received the Ph.D. degree in computer engineering from the Sharif University of Technology, Tehran, Iran, in 2016. From 2014 to 2015, he was a Visiting Researcher with the Chair for Embedded Systems (CES), Karlsruhe Institute of Technology (KIT), Germany. He is currently an Assistant Professor in computer engineering with the University of Guilan, Rasht, Iran. His research interests include design of low-power, reliable and real-time embedded systems with a focus on dependability and energy efficiency in cyber-physical systems, and the Internet of Things (IoT).

SEPIDEH SAFARI received the degree in engineering from the Sharif University of Technology, Tehran, Iran, in 2016, where she is currently pursuing the Ph.D. degree in computer engineering. She is also a Visiting Researcher with the Chair for Embedded Systems (CES), Karlsruhe Institute of Technology (KIT), Germany. Her research interests include low-power design of cyber-physical systems, energy management in fault-tolerant embedded systems, and multi-/many-core systems with a focus on dependability/reliability.

ALIREZA EJLALI received the Ph.D. degree in computer engineering from the Sharif University of Technology, Tehran, Iran, in 2006. From 2005 to 2006, he was a Visiting Researcher with the Electronic Systems Design Group, University of Southampton, Southampton, U.K. In 2006, he joined the Department of Computer Engineering, Sharif University of Technology, as a Faculty Member, where he was the Director of the Computer Architecture Group, from 2011 to 2015. He is currently an Associate Professor in computer engineering with the Sharif University of Technology. His research interests include low power design, real-time embedded systems, and fault-tolerant embedded systems.

MUHAMMAD SHAFIQUE (Senior Member, IEEE) received the Ph.D. degree in computer science from the Karlsruhe Institute of Technology (KIT), Germany, in 2011. Afterwards, he established and led a highly recognized research group at KIT for several years as well as conducted impactful research and development activities in Pakistan. In October 2016, he joined the Faculty of Informatics, Institute of Computer Engineering, Technische Universitat Wien (TU Wien), Vienna, Austria, as a Full Professor in computer architecture and robust, energy-efficient technologies. Since September 2020, he has been with the Division of Engineering, New York University Abu Dhabi (NYU AD), United Arab Emirates. His research interests include brain-inspired computing, AI and machine learning hardware and system-level design, energy-efficient systems, robust computing, hardware security, emerging technologies, FPGAs, MPSoCs, and embedded systems. His research has a special focus on cross-layer analysis, modeling, design, and optimization of computing and memory systems. The researched technologies and tools are deployed in application use cases from the Internet of Things (IoT), smart cyber-physical systems (CPSs), and ICT for development (ICT4D) domains.

Dr. Shafique received the 2015 ACM/SIGDA Outstanding New Faculty Award, the AI 2000 Chip Technology Most Influential Scholar Award in 2020, six gold medals, and several best paper awards and nominations at prestigious conferences. He has given several keynotes, invited talks, and tutorials, as well as organized many special sessions at premier venues. He has served as the PC Chair, the Track Chair, and a PC Member for several prestigious IEEE/ACM conferences. He holds one U.S. patent has (co-)authored six books, more than ten book chapters, and over 200 articles in premier journals and conferences.