The Journal of Systems and Software 85 (2012) 1386–1399

Contents lists available at SciVerse ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Quasi-static fault-tolerant scheduling schemes for energy-efficient hard real-time systems☆

Tongquan Wei[a,*,1], Piyush Mishra[b,1], Kaijie Wu[c,1], Junlong Zhou[a]

[a] CS Department of East China Normal University, Shanghai 200241, China
[b] GE Global Research, Niskayuna, NY 12309, USA
[c] ECE Department of University of Illinois, Chicago, IL 60607, USA

ARTICLE INFO

Article history:
Received 13 August 2011
Received in revised form 7 January 2012
Accepted 7 January 2012
Available online 28 January 2012

Keywords:
Energy-efficient dynamic scheduling
Dynamic voltage scaling (DVS)
Fault tolerance
Hard real-time embedded systems

ABSTRACT

This paper investigates fault tolerance and dynamic voltage scaling (DVS) in hard real-time systems. The authors present quasi-static task scheduling algorithms that consist of offline components and online components. The offline components are designed the way they enable the online components to achieve energy savings by using the dynamic slack due to variations in task execution times and uncertainties in fault occurrences. The proposed schemes utilize a fault model that considers the effects of voltage scaling on transient fault rate. Simulation results based on real-life task sets and processor data sheets show that the proposed scheduling schemes achieve energy savings of up to 50% over the state-of-art low-energy offline scheduling techniques and incur negligible runtime overheads. A hard real-time real-life test bed has been developed allowing the validation of the proposed algorithms.

1. Introduction

The number of faults in hardware, particularly the transient faults, has been rising continuously due to the increasing complexity of design, aggressive technology scaling, and extreme operating conditions. For example, the increasing integration level of transistors, reducing feature sizes, and lowering voltage levels are making the integrated circuits highly susceptible to radiation-induced bit-flips. In addition, high-energy particles, such as neutrons from cosmic radiation, are able to introduce transient faults in electronic systems (Normand, 1996). On the other hand, a growing number of complex safety critical applications operate under extreme conditions and demand ultra-reliability and high performance. For example, hard real-time systems deployed in navigation, process control, and system surveillance require the high fault tolerance without sacrificing the feasibility of task sets. The need for reliability is rising even in non-critical applications which are prone to operate in harsher environments but have lower expectancy of failures. Common examples include outdoor sensor networks and massive communication infrastructure deployed in fields which suffer frequent physical abuse and are often exposed to strong radiation.

Numerous fault-tolerance techniques have been proposed for real-time systems (Shin and Lee, 1984; Shin et al., 1987; Kwak et al., 2001; Axer et al., 2011; Huang et al., 2011). One of the typically used fault-tolerance techniques is online concurrent fault detection followed by a hardware-based checkpointing and rollback recovery mechanism. It allows processors to rollback to the previously known valid states to resume normal executions by exploiting the slack time available in task schedules. Traditionally, fault tolerance techniques aim to maximize the fault coverage and minimize the fault detection latency and associated redundancy costs (Pradhan, 1986). The costs are usually measured in terms of hardware, time, or information overhead and are of great significance in real-time embedded systems due to their severe resource constraints.

Owing to the fast-evolving application of real-time systems in battery-powered portable devices, energy has emerged as another important design constraint. Dynamic power management is an active area of research and several techniques have been proposed to minimize energy consumption at the system level (Benini et al., 2000). The energy efficiency is achieved by dynamically reconfiguring active system components and selectively turning off system components when they are idle. Dynamic voltage scaling (DVS) is a widely used system level power management technique that exploits technological advances in power supply circuits to reduce the energy consumption. It reduces the processor power consumption by dynamically scaling down the processor supply voltage at the cost of the increased execution times.

Fault tolerance and energy have been jointly investigated in the literature. On one hand, with the continuous shrinking of the

☆ The preliminary version of this manuscript appeared in ICCAD, 2006.

* Corresponding author.
E-mail address: tqwei@cs.ecnu.edu.cn (T. Wei).
[1] Member, IEEE.

0164-1212/$ – see front matter © 2012 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2012.01.020

feature size and reducing of voltage margins, it is expected that all digital computing systems will be remarkably vulnerable to transient faults (Ernst et al., 2004). Fault-tolerance in real-time systems is typically achieved by using some forms of redundancy. This redundancy causes extra power dissipations and hence necessitates energy efficient schemes for batter-powered real-time systems to reduce heat dissipation and extend operational lifetime. On the other hand, energy management through dynamic voltage scaling has adverse effects on system reliability. Scaling down the supply voltage results in an increase in the rate of transient faults (Zhu et al., 2004). Consequently, both fault-tolerance and energy efficiency have been the primary design goals for real-time systems, integrated in the design process at all levels for joint optimization with the system feasibility.

In the recent past, considerable attention has been paid to exploit the DVS technique to achieve energy savings in the presence of transient faults and a number of excellent energy-efficient fault-tolerance schemes have been designed for real-time embedded systems (Zhu et al., 2004; Zhang et al., 2003; Melhem et al., 2004; Wei et al., 2006; Zhang and Chakrabarty, 2006; Zhao et al., 2009, 2011; Iqbal et al., 2011). In this paper a systematic approach is proposed to derive energy-efficient fault-tolerant task schedules for hard real-time embedded systems by utilizing both the static and dynamic slack in the task schedules. Based on the observation that the probability of the single event upset (SEU)-induced faults remains low in the foreseeable future and fault-free condition will continue to dominate (Reed et al., 2006; Weulersse et al., 2006; Langley et al., 2003), one fundamental innovation has been introduced in the proposed energy-efficient fault-tolerance schemes. That is, unlike the traditional approach that focuses on designing offline energy-efficient fault-tolerance algorithms, the proposed scheme aims to achieve further round of energy savings by designing efficient offline algorithms that enable the adaptation of the offline schedules to the runtime behavior of fault occurrences.

1.1. Related work

Extensive research has been performed to investigate the energy efficiency of real-time systems from both offline and online perspective (Shin and Choi, 1999; Gruian, 2001; Pillai and Shin, 2001; Saewong and Rajkumar, 2003; Krishna and Lee, 2003; Mochocki et al., 2007; Huang et al., 2009, 2011; Perathoner et al., 2010). The power efficient version of fixed-priority preemptive scheduling such as rate monotonic scheduling was explored by Shin and Choi (1999). Power reduction is achieved by exploiting the slack time both inherent in system schedule and due to runtime variations in task execution time. Similarly, Gruian (2001) presented a scheduling policy for hard real-time tasks with fixed priorities assigned in a rate monotonic manner. The offline scheduling uses exact timing analysis to derive multiple voltage scaling factors for each task based on stochastic characteristics of task execution time. The online scheduling policy distributes available slack time on priority basis. Based on the voltage scaling algorithms proposed in Pillai and Shin (2001), four voltage scaling algorithms including Sys-Clock, PM-Clock, and DPM-Clock were proposed in Saewong and Rajkumar (2003) for different hardware which may have high or low voltage scaling overhead and different taskset characteristics. Of these algorithms, Sys-Clock assigns a single frequency to all tasks in a task set, PM-Clock assign multiple frequencies to tasks in a task set, and DPM-Clock dynamically adapts offline task schedule to runtime behaviors of task execution times. Krishna and Lee (2003) described a two phase heuristic for independent and periodic tasks. The heuristic has an offline component computing a voltage schedule based on worst case execution time, and an online component utilizing slack time due to variations in task execution time for further round of energy savings. In Mochocki et al. (2007), both offline and online scheduling schemes were proposed to handle the transition time and energy overhead of DVS processors. The offline scheme generates task schedule during design time based on a prior known task execution time while the online scheme effectively accommodates runtime variations of task execution time to achieve energy savings. Online algorithms were presented in Huang et al. (2009, 2011) to effectively reduce system energy consumption to handle event streams with hard real-time guarantees. The scheduling scheme adaptively controls the power mode of the processor to postpone the processing of arrival events as late as possible. Although energy efficiency in real-time systems were explored from both offline and online aspects in the above literature, fault tolerance which is an another important design constraints were not considered.

Fault-tolerance is another important design constraint in energy efficient real-time systems. Joint optimization of energy and fault-tolerance in real-time embedded systems has attracted considerable attention in the past decade (Zhang et al., 2003; Melhem et al., 2004; Zhang and Chakrabarty, 2006; Zhao et al., 2009, 2011; Iqbal et al., 2011; Wei et al., 2011). Melhem et al. (2004) proposed DVS techniques to exploit slacks in a task schedule to reduce energy consumption while tolerating faults during task execution. A task in the task schedule is assumed to be susceptible to at most one fault occurrence and the processor can scale its frequency in a continuous range. In Zhang et al. (2003) and Zhang and Chakrabarty (2006) a fixed priority offline scheduling scheme was proposed based on the rate monotonic scheduling to tolerate faults in hard real-time systems. Practical design issues such as checkpointing cost and voltage switching overhead are considered. Fault-tolerance scheduling techniques were developed in Zhao et al. (2009, 2011) to minimize the system-level energy consumption while still preserving the systems original reliability. Fault tolerance is achieved by reserving shared recovery blocks that can be used by any task at the runtime. In Iqbal et al. (2011), the authors presented a soft error aware energy efficient scheduling technique for soft real-time systems with stochastic task execution times. The task execution time estimation is modeled as a joint state-space model, the solution of which is found by an online Monte Carlo sampling based recursive technique.

An offline reliability-aware power management scheme is presented in Zhu et al. (2008) for real-time tasks with probabilistic execution times. The scheme puts aside just enough slack to guarantee the required reliability while leaving more slack for energy management to achieve better energy savings. In Pop et al. (2007), the authors addressed the scheduling and voltage scaling for hard real-time applications that have been statically mapped on heterogeneous distributed embedded systems. Tasks in a given task set are assumed to share a common deadline and the effect of voltage scaling on system reliability is taken into account. Shafik et al. (2010) examined the impact of application task mapping on the reliability of MPSoC. The number of transient faults is minimized without compromising the timeliness of the system. All these energy-aware fault-tolerance schemes, however, statically derive offline task schedules to guarantee hard timing constraints, hence are conservative and cannot utilize the dynamic slack due to variations in task execution times and uncertainties in fault occurrences for further energy savings.

Zhang and Chakrabarty (2003) developed an online scheduling algorithm that combines checkpointing with DVS to tolerate faults in real-time uni-processor systems with periodic tasks. However, this scheme cannot handle hard real-time task scheduling. In Izosimov et al. (2008), the authors present an approach to the synthesis of fault-tolerant schedules for embedded applications with soft and hard real-time constraints. A set of task schedules is synthesized offline and, at run time, the scheduler selects the right schedule based on the fault occurrence and the actual task

execution times such that hard timing constraints are guaranteed and the overall processor utilization is maximized. However, the presented approach does not take energy into account.

In this paper efficient scheduling schemes are proposed to combine offline feasibility analysis and online voltage scaling for hard real-time systems based on the exact timing analysis of the rate monotonic algorithm (RMA). Two offline scheduling algorithms that enable the dynamic adaptation are proposed. One is the application level voltage scaling (A-DVS) algorithm where all the tasks run at the same processor speed. The other one is the task level voltage scaling (T-DVS) algorithm where the tasks run at their individual speeds. Instead of iteratively deriving the response time of each task for feasibility analysis, the exact timing analysis approach (Lehoczky et al., 1989) is used in the proposed algorithms for feasibility analysis. This strategy strikingly simplifies the adaptation of the proposed offline A-DVS and T-DVS algorithms to the runtime behavior of fault occurrences. The adaptation of the offline task schedules to the runtime behavior of fault occurrences is implemented by (1) pre-computing and saving in a lookup table the maximum slack requirements for the processor to dynamically slow down, (2) retrieving and comparing the stored slack time requirements with the generated cumulative slack in the runtime, and (3) dynamically scaling down processor speed when the generated slack time is equal to or greater than the stored slack requirements. Xtrem (Contreras et al., 2004), a SimpleScalar-based Intel XScale processor simulator, was used to evaluate the runtime overhead of the proposed scheduling schemes in addition to extensive simulation experiments. A hard real-time test bed has been designed and the proposed algorithms were also verified on the test bed.

1.2. Contributions and outline

The main contributions of this paper are summarized as follows:

- Quasi-static task scheduling algorithms consisting of offline components and online components are proposed. The offline components are designed the way they enables the online components to save energy in the runtime using slack due to uncertainties in fault occurrences.
- The proposed schemes are based on a fault model that considers the effect of DVS on transient fault rate. The worst case number of fault occurrences in a task at a certain voltage level is derived according to the given task level reliability goal. This strategy facilitates the design of systems with various reliability requirements.
- In addition to being verified under simulation environments, the proposed schemes also are implemented and validated on a real-life hard real-time test bed.

The rest of the paper is organized as follows. Section 2 introduces the system models. Section 3 describes the feasibility analysis for ECRMA-based fault-tolerance task scheduling, and proposes two offline task scheduling algorithms with different DVS granularity. Section 4 adapts the offline task schedules to the runtime behavior of task execution and fault occurrences. Section 5 presents the experimental results to demonstrate energy savings and runtime overhead. Section 6 describes the implementation of a hard real-time test bed for energy measurement and Section 7 concludes the paper.

2. System architecture and models

The focus of the study is a fixed-priority hard real-time embedded system comprising a DVS-capable uni-processor and a power-aware memory. It is assumed that the scheduler of the system is preemptive, such that, if required, the scheduler may suspend the current task and switch the system context to a new task according to its scheduling scheme.

2.1. Architecture and application model

Consider a task set Γ consisting of n independent periodic tasks $\{\tau_1, \tau_2, \ldots, \tau_n\}$. The timing characteristics of the task τ_i are defined as a tuple $\tau_i = \{T_i, D_i, C_i\}$, where T_i is the period, D_i is the deadline, and C_i is the worst case execution cycles. The hyper-period of the task set, denoted by T, is the lowest common multiple of all task periods $\{T_1, T_2, \ldots, T_n\}$.

It is assumed that tasks are arranged in the decreasing order of priorities according to the fixed priority rate monotonic algorithm (RMA) (Liu and Layland, 1973), that is, $T_1 < T_2 <, \ldots, < T_n$ is such that the period of task τ_i is smaller than the period of task τ_j for $i < j$. The processor is assumed to support L discrete frequency or voltage levels. Frequency levels and voltage levels are used interchangeably throughout this paper. Let f_i denote the operating frequency of task τ_i, where $i (1 < i < n)$ is the index of the task in the task set Γ. The operating frequency f_i of task τ_i can be expressed as a function of the processor voltage level l at which the task is running, that is, $f_i = f(l)$. Tasks in a given task set is assumed to be scheduled using RMA and the resulting schedule is feasible under fault-free conditions at a certain voltage level.

2.2. Fault and recovery model

It is assumed that a watchdog processor is used for timing checking. Faults are assumed to be detected using low-latency fault detection techniques such as the simultaneous multithreading scheme (Reinhardt and Mukherjee, 2000) such that the fault detection overhead is small enough to be accounted for in task execution time. Upon detecting faults, system is assumed to recover via backward recovery mechanism, where a set of checkpoints are inserted into a computing system for fault-tolerance. At each checkpoint, valid system states are copied and stored for error recovery. If one or more faults are detected during computation, the application rolls back to the immediate previous checkpoint, retrieves the stored system states, and resumes computation from the checkpoint.

The reliability of a task is defined to be the probability of completing the task successfully subject to faults (Zhu et al., 2004). Although reliability targets are typically given system-wide, it is a common practice to derive the per-task values (unit requirements) from system-wide values (system requirements). Assuming all tasks in a given task set share a common given reliability goal, then the task level reliability is maintained if all tasks in the task set finish their execution successfully under the given reliability target. Let R_i denote the reliability of task τ_i. R_i is derived as follows.

Let k_i denote the exact number of fault occurrences in task τ_i, f_i denote the operating frequency of task τ_i at the voltage level l $(1 \le l \le L)$, and O_{il} denote the optimal number of checkpoints for task τ_i at the voltage level l that minimizes the worst case response time of the task. Assuming checkpointing intervals are equal, the O_{il} is then given by

$$O_{il} = \left\lceil \sqrt{\frac{k_i C_i}{c_s f_i}} - 1 \right\rceil \quad \text{or} \quad \left\lfloor \sqrt{\frac{k_i C_i}{c_s f_i}} - 1 \right\rfloor,$$

where C_i is the execution cycles of task τ_i and c_s is the checkpointing overhead (Zhang and Chakrabarty, 2006). c_s is assumed to be

constant. For the sake of easy presentation, O_{il} is simply denoted by

$$O_{il} = \left\| \sqrt{\frac{k_i C_i}{c_s f_i}} - 1 \right\|. \quad (1)$$

Let OE_i be the overall execution time of an instance of task τ_i. Considering the checkpointing overhead and fault recovery overhead, the OE_i is derived as (Zhang and Chakrabarty, 2006)

$$OE_i = \frac{C_i}{f_i} + O_{il} \times c_s + \frac{k_i C_i}{f_i(O_{il}+1)} + 2k_i c_s. \quad (2)$$

Transient fault occurrences are typically modeled using the Poisson distribution. Let λ_l be the average fault arrival rate at the frequency level l. The λ_l can be derived using the equation $\lambda_l = \gamma \times e^{-\alpha f_i}$, where γ and α are constant parameters and f_i is the operating frequency of the task τ_i (Zhu et al., 2004). The probability of k_i fault occurrences during the execution of task τ_i at frequency f_i is thus given by

$$\frac{e^{-\lambda_l \times OE_i} \times (\lambda_l \times OE_i)^{k_i}}{k_i!},$$

and the reliability R_i of task τ_i is hence written as

$$R_i = \sum_{k_i} \frac{e^{-\lambda_l \times OE_i} \times (\lambda_l \times OE_i)^{k_i}}{k_i!}. \quad (3)$$

This definition of reliability considers the fault occurrences during the whole interval of OE_i. In other words, the fault occurrences during the recovery of a task from faults is taken into account.

Let R_g denote the task level reliability goal, then the reliability of task τ_i is maintained if the inequality

$$R_g \le R_i \quad (4)$$

holds. Since λ_l, OE_i, and k_i all are functions of the voltage level l of task τ_i, R_i is also a function of the voltage level l of task τ_i. For a given voltage level l ($1 \le l \le L$), λ_l and OE_i are known; thus, the worst case number of fault occurrences k_i at the voltage level l subject to target reliability R_g can be iteratively derived using the inequality (4).

2.3. Energy model

The power consumption of a CMOS device can be modeled as the sum of dynamic power consumption and static power consumption. The average dynamic power consumption is a function of the supply voltage and the operating frequency. Let p_d, V_{dd}, and f be the average dynamic power consumption, supply voltage, and operating frequency, respectively, then $p_d \propto {V_{dd}}^2 f$ holds (Weste and Eshraghian, 1992). Assuming processors use voltage scaling technique to scale frequency, the operating frequency is then approximately linear with the supply voltage (Weste and Eshraghian, 1992). As a result, the average dynamic power consumption can be estimated by a strictly increasing and convex function of the operating frequency, that is, $p_d \propto f^3$.

As technology advances towards deep sub-micro devices, the static power consumption due to leakage current and other current drawn continuously from the power supply has been increasing dramatically. It has been shown that the major contributors of the static power consumption are the subthreshold leakage current and the reverse bias junction current (Martin et al., 2002). Let p_s denote the static power consumption of a device, I_{subn} denote the subthreshold leakage current, and I_j denote the reverse bias junction current, then the static power consumption of the device is given by $p_s = V_{dd} I_{subn} + |V_{bs}| I_j$, where V_{bs} is the body bias voltage, and V_{dd} is the supply voltage (Jejurikar et al., 2004). For a certain generation of technology, the sub-threshold leakage current I_{subn} is the function of the supply voltage V_{dd}, and the V_{bs} and I_j are technology constants.

The total energy (E_{tot}) consumed by real-time tasks in a given task set is hence estimated by

$$E_{tot} = \sum_{i=1}^{n} OE_i \times \frac{T}{T_i} \times (p_d + p_s), \quad (5)$$

where OE_i is the overall execution time of task τ_i, as is given in Eq. (2), and T/T_i is the number of instances of the task τ_i in the hyper-period T.

3. Offline scheduling algorithms for fault-tolerant hard real-time systems

The rate monotonic algorithm (RMA), proposed by Liu and Layland (1973), is an optimal fixed priority algorithm that schedules periodic tasks by assigning higher priorities to tasks with shorter periods. The classical analysis of RMA yields a conservative bound on the processor utilization below which a system is guaranteed to meet all task deadlines. Lehoczky et al. (1989) showed that this conservative utilization bound of RMA can be relaxed based on the exact characterization of RMA (ECRMA) to derive both the necessary and sufficient conditions for the feasibility analysis of a schedule. This section first provides an overview of the ECRMA-based feasibility analysis of fault-tolerant task schedules, then presents the two proposed offline task scheduling algorithms of different granularity.

3.1. ECRMA-based feasibility analysis of fault-tolerant task schedules

The worst case behavior of RMA occurs when all tasks in a task set are instantiated simultaneously and are ready for execution immediately after initiation. This time instant is called critical instant. It has been shown that a schedule of independent periodic tasks is feasible if the first instance of each task is schedulable when it is instantiated at a critical instant. Lehoczky et al. (1989) showed that periodic tasks in a task set are schedulable for all task phasing if and only if at any time instance before the deadline of a task, the total demand for processor time by the task is equal to or less than the current available processor time. Specifically, let

$$W_i(t) = \sum_{j=1}^{i} OE_j \left\lceil \frac{t}{T_j} \right\rceil$$

denote the total demand of task τ_i for processor time over $[0, t]$, assuming 0 is the critical instant. The necessary and sufficient condition for the periodic task τ_i to be schedulable is given by

$$\min_t \left(\frac{W_i(t)}{t} \right) \le 1, \quad 0 < t < T_i,$$

and the entire task set is schedulable iff

$$\max_i \left(\min_t \left(\frac{W_i(t)}{t} \right) \right) \le 1, \quad 0 < t < T_i,\ 1 \le i \le n. \quad (6)$$

Further, it was shown that the schedulability test of each task needs to be performed only at a finite number of time instances called scheduling points. This is because the normalized demand of task τ_i on a processor, given by $W_i(t)/t$, is strictly decreasing except at those scheduling points. The set of scheduling points of task τ_i, S_i, is defined as multiples of T_g for $T_g \le T_i$, that is,

$$S_i = \left\{ h \times T_g | g = 1, 2, \ldots, i; h = 1, 2, \ldots, \left\lfloor \frac{T_i}{T_g} \right\rfloor \right\}.$$

Procedure A-DVS($\Gamma, low, high, k_i, c_s$)

1. $low = 1; high = L;$
2. **for** $\tau_i, 1 \le i \le n$ **do**
3. $S_i = \left\{h \times T_g | g = 1, 2, \cdots, i; h = 1, 2, \cdots, \left\lfloor \frac{T_i}{T_g} \right\rfloor\right\}$ {Derive S_i, the set of scheduling points of task τ_i}
4. **end for**
5. **while** $low \le high$ **do**
6. $mid = (low + high)/2$
7. **if** FCA(Γ, S, k_i, c_s, mid) **then**
8. $high = mid - 1$
9. **else if** (!FCA(Γ, S, k_i, c_s, mid)) && ($mid == L$) **then**
10. **print** "Infeasible Schedule"
11. **exit**(1) {Exit when infeasible}
12. **else**
13. $low = mid + 1$
14. **end if**
15. **end while**
16. **return** mid

Fig. 1. Application level voltage scaling algorithm (A-DVS).

Substitute the S_i and the $W_i(t)$ into Eq. (6), the necessary and sufficient condition for real-time tasks in a given task set to be feasible becomes

$$\max_i \left(\min_t \sum_{j=1}^{i} OE_j \frac{\lceil t/T_j \rceil}{t} \right) \le 1, \quad t \in S_i,\ 1 \le i \le n. \tag{7}$$

The proposed task scheduling techniques exploit the exact characterization of RMA to efficiently verify the feasibility of an offline schedule in the presence of faults and to dynamically adapt DVS policies to uncertainties of fault occurrences. As is shown in Eq. (7), the task execution time with fault recovery overhead, which is denoted by OE_j, is utilized in the feasibility analysis of the offline schedule. As a result, the generated feasible task schedules will be feasible when faults occur.

The proposed task scheduling techniques offer three advantages over the previous techniques: (1) higher tolerance to fault recovery overhead due to the relaxed constraints of the exact characterization of RMA, (2) low-cost ECRMA-based offline feasibility analysis schemes, and (3) efficient extension to the runtime reevaluation of DVS policies due to the relatively lower complexity of the offline feasibility analysis. In this paper, the scheduling point-based exact characterization of RMA provides a systematic approach to improve the feasibility of a schedule while tolerating faults for enhanced reliability and scaling voltage for energy efficiency. Two offline algorithms are proposed to integrate fault-tolerance and DVS policy evaluation by systematically searching for the energy-efficient fault-tolerant schedule for a given set of tasks, which are described in the next sections.

3.2. Application level voltage scaling (A-DVS)

The application level voltage scaling (A-DVS) is designed for the scenario where all tasks in a given task set run at the same processor speed. The A-DVS is suitable for systems in which frequent voltage and frequency scaling is inefficient. Fig. 1 shows A-DVS algorithm to derive energy efficient voltage setting subject to fault and feasibility constraints. Inputs to the algorithm are the task set Γ, the lowest voltage level (*low*) and highest voltage level (*high*) supported by the processor, the maximum number of faults each task should tolerate (k_i), and checkpoint overhead (c_s). Voltage levels *low* and *high* are initially set to 1 and *L*, respectively.

A-DVS starts by computing the set of scheduling points S_i of task τ_i, and then iteratively performs feasibility analysis using ECRMA to select the proper DVS strategy while tolerating k_i faults in each task instance. *S* denotes the array of S_i for $1 \le i \le n$. Lines 2 to 4 iteratively compute scheduling points of *n* tasks with the time complexity of $O(n^2R)$, where *n* is the number of tasks in the task set and *R* is the ratio of the largest period to the smallest period. Lines 5–15 iteratively search the energy efficient voltage level in the range from *low* to *high* based on the binary search algorithm. For a given voltage level *mid* = (*low* + *high*)/2, the feasibility checking subroutine FCA is called to test the feasibility of a schedule at the voltage level. If the schedule is found feasible, *high* is updated to (*mid* − 1) in line 8, else *low* is updated to (*mid* + 1) in line 13. This process continues until a feasible schedule is found for all tasks or the highest voltage level *L* is reached without satisfying Eq. (7), in which case the task set is deemed infeasible (lines 10 and 11). Line 16 returns the energy efficient voltage level of the feasible schedule. The time complexity of deriving *S*, the set of scheduling points of all tasks in the task set, is $O(n^2R)$, and the time complexity of the binary search algorithm is $O(\log_2 L)$ (Cormen et al., 2001). The overall time complexity of A-DVS algorithm depends on the complexity of FCA algorithm.

Note that the binary search based A-DVS algorithm is valid only if the energy consumption is monotonic with respect to frequency/voltage changes. When the processor static power consumption as well as context switching overhead is considered, the monotonicity does not hold. In this case, there exists a critical processor speed below which scaling down the processor speed will instead increase the energy consumption (Jejurikar et al., 2004). In other words, executing a task below the critical speed consumes more time and energy. Hence, in the binary search based A-DVS algorithm, the minimum voltage level *low* is initialized to the level corresponding to the processor critical speed.

Fig. 2 describes the feasibility checking algorithm FCA. It takes as inputs the task set (Γ), the array of S_i (S), the maximum number of faults each task should tolerate (k_i), checkpoint overhead (c_s), and the current voltage level (l).

The FCA algorithm uses a flag *Schedulable* which is reset to 0 whenever a task is found un-schedulable at the current common voltage level, and a buffer *Demand* which holds the total demand for processor time at scheduling points at the current voltage level. The algorithm operates in two phases. Phase 1, consisting of lines 3–6, derives the optimum number of checkpoints at the current voltage level and compute the worst case execution time OE_i of the current task τ_i. Phase 2, consisting of lines 7–18, verifies the schedulability of the current task using Eq. (7). It computes the total time demand of task τ_i at each scheduling point, check the schedulability of the task, and set the *Schedulable* flag accordingly. If task τ_i is found schedulable, the algorithm proceeds to the next task; else it returns 0 to A-DVS routine.

The time complexity of phase 1 and phase 2 of FCA algorithm is $O(n^2)$ and $O(n^2R)$, respectively, and the overall time complexity of FCA algorithm is $O(n^2R)$. Considering the time complexity of binary search algorithm, the overall time complexity of A-DVS algorithm is $O(n^2R \log_2 L)$ and the average time complexity per task is $O(nR \log_2 L)$. When compared to the application level technique proposed in Zhang and Chakrabarty (2006), which has the time complexity of $O(n^2RL)$, A-DVS incurs an order of magnitude lower cost. The relative lower-complexity of the A-DVS algorithm enables the adaptation of the offline task schedules to the runtime time behavior of task execution time and fault occurrences for further energy savings. The dynamic extension of A-DVS algorithm is explored in Section 4.1.

Procedure FCA(Γ, S, k_i, c_s, l)

1. $Schedulable = 0; Demand = 0; i = j = p = q = 1$
2. **for** $\tau_i, 1 \le i \le n$ **do**
3. **for** $\tau_j, j \le i$ **do**
4. $O_{jl} = \max\left(\left\|\sqrt{\frac{k_i C_j}{c_s f_j}} - 1\right\|, 0\right)$
5. $OE_j = \frac{C_j}{f_j} + O_{jl} \times c_s + \frac{k_i C_j}{f_j(O_{jl}+1)} + 2k_i c_s$
6. **end for**
7. **for** $s_{iq} \in S_i$ **do**
8. $Demand = 0$
9. **for** $\tau_p, 1 \le p \le i$ **do**
10. $Demand = Demand + OE_p \times \lceil s_{iq}/T_p \rceil$
11. **end for**
12. **if** $Demand \le s_{iq}$ **then**
13. $Schedulable = 1; q = 1$
14. **break**
15. **else**
16. $Schedulable = 0; q = q + 1$
17. **end if**
18. **end for**
19. **if** $Schedulable = 0$ **then**
20. **return** (0)
21. **else**
22. $i = i + 1$
23. **end if**
24. **end for**
25. **return** (1)

Fig. 2. Feasibility checking algorithm (FCA).

3.3. Task level voltage scaling (T-DVS)

The task level voltage scaling (T-DVS) offers higher energy savings and improves fault-tolerance at the cost of increased complexity. T-DVS algorithm shown in Fig. 3 is similar to the A-DVS except whenever a task, for example task τ_i, is found un-schedulable. In this case, the T-DVS repeatedly selects one task from among the tasks of equal and higher priorities and scales the voltage level of the task up by one level until task τ_i becomes schedulable. If the highest voltage level for all tasks of equal and higher priorities is reached and task τ_i is still un-schedulable, the task set is deemed to be infeasible. The task selected for voltage scaling satisfies two requirements: (1) its voltage level is lower than the highest processor supported voltage level and (2) the subsequent increase in energy consumption due to scaling up the voltage level of the selected task is minimal among all candidate tasks.

Inputs to T-DVS are a task set Γ which is assumed to be feasible at a certain voltage level, the maximum number of faults each task instance should tolerate (k_i), and checkpointing overhead (c_s). The parameters used to track the state of a schedule include f_i and $Level_i$ which denote the operating frequency and voltage level of task τ_i, respectively, and *min* which denotes the index of the task selected for voltage scaling. For the sake of easy presentation, *f* and *Level* are used to denote the arrays of operating frequencies and voltage levels of tasks in the task set, respectively. It is assumed that the operating frequency f_i of task τ_i is a function of the voltage level $Level_i$, that is, $f_i = f(Level_i)$.

Lines 2–4 initialize the operating frequency f_i and voltage level $Level_i$ of task τ_i to $f(1)$ and 1, respectively. Lines 5–7 iteratively compute the scheduling points of *n* tasks in the task set. The time complexity of the computation is in order of $O(n^2R)$, where *R* is the ratio of the largest period to the smallest period of tasks in the task

Procedure T-DVS(Γ, k_i, c_s)

1. $Schedulable = 0; i = 1$
2. **for** $\tau_i, 1 \le i \le n$ **do**
3. $f_i = f(1); Level_i = 1$
4. **end for**
5. **for** $\tau_i, 1 \le i \le n$ **do**
6. $S_i = \left\{h \times T_g | g = 1, 2, \cdots, i; h = 1, 2, \cdots, \left\lfloor \frac{T_i}{T_g} \right\rfloor\right\}$
{Derive S_i, the set of scheduling points for task τ_i}
7. **end for**
8. **for** $\tau_i, 1 \le i \le n$ **do**
9. $O_{il} = \max\left(\left\|\sqrt{\frac{k_i C_i}{c_s f_i}} - 1\right\|, 0\right)$
10. $OE_i = \frac{C_i}{f_i} + O_{il} \times c_s + \frac{k_i C_i}{f_i(O_{il}+1)} + 2k_i c_s$
11. $Schedulable = SCA(S, OE, i)$
{OE denotes the array of OE_i and S denotes the array of S_i}
12. **if** $Schedulable = 1$ **then**
13. $i = i + 1$
14. **else**
15. **while** !$Schedulable$ **do**
16. $min = MINIMUM(Level, OE, i)$
17. **if** $Level_{min} < L$ **then**
18. $Level_{min} = Level_{min} + 1$
19. $f_{min} = f(Level_{min})$
20. $O_{min,l} = \max\left(\left\|\sqrt{\frac{k_i C_{min}}{c_s f_{min}}} - 1\right\|, 0\right)$
21. $OE_{min} = \frac{C_{min}}{f_{min}} + O_{min,l} \times c_s + \frac{k_i C_{min}}{f_{min}(O_{min,l}+1)} + 2k_i c_s$
22. $Schedulable = SCA(S, OE, i)$
23. **else**
24. **print** "Infeasible Schedule"
25. **exit**(1)
26. **end if**
27. **end while**
28. **end if**
29. **end for**

Fig. 3. Task level voltage scaling algorithm (T-DVS).

set. Rest of the algorithm operates in 2 phases, which are iterated for each task in the task set. Phase 1, consisting of lines 9 and 10, derives the optimum number of checkpoints of task τ_i at the current voltage level and compute the worst case overall execution time OE_i for task τ_i. The array of the worst case overall execution time of tasks in a task set is denoted by *OE*. Unlike A-DVS, phase 1 of T-DVS takes constant time. Phase 2, consisting of lines 11–28, verifies the schedulability of task τ_i using Eq. (7) and performs voltage scaling at task level. Line 11 calls the schedulability checking subroutine (SCA) to verify the schedulability of task τ_i at the operating frequency of f_i. If task τ_i is found schedulable, the algorithm proceeds to the next task (line 13), else a task is selected for voltage scaling (lines 15–27). In line 16 the MINIMUM algorithm (Cormen et al., 2001) is called and the index *min* of the task selected for voltage scaling is returned. Task τ_{min} is the task that results in the minimal energy increase among all candidate tasks for voltage scaling.

If the voltage level $Level_{min}$ is found lower than *L*, then the operating frequency f_{min} of task τ_{min} is raised by one level in lines 18 and 19, the optimal number of checkpointing $O_{min,l}$ and the overall execution time OE_{min} of task τ_{min} are updated respectively in lines 20 and 21, and the schedulability of task τ_{min} is reevaluated based on the updated total time demand in line 22. Else if the voltage

Procedure SCA(S, OE, i)

1. $Schedulable = 0; Demand = 0; p = q = 1$
2. **for** $s_{iq} \in S_i$ **do**
3. $\quad Demand = 0$
4. $\quad$ **for** $\tau_p, 1 \le p \le i$ **do**
5. $\quad\quad Demand = Demand + OE_p \times \lceil s_{iq}/T_p \rceil$
6. $\quad$ **end for**
7. $\quad$ **if** $Demand \le s_{iq}$ **then**
8. $\quad\quad Schedulable = 1$
9. $\quad\quad$ **break**
10. $\quad$ **else**
11. $\quad\quad Schedulable = 0; q = q + 1$
12. $\quad$ **end if**
13. **end for**
14. **return** $Schedulable$

Fig. 4. Schedulability checking algorithm (SCA).

level $Level_{min}$ is found equal to the highest voltage level L, task τ_i is un-schedulable. This process continues until a feasible schedule is found for the task set or the highest voltage level is reached without satisfying Eq. (7), in which case the schedule of the task set is deemed infeasible.

The MINIMUM algorithm, shown in line 16 of Fig. 3, takes as inputs the array of voltage levels (*Level*), the array of overall execution time of tasks in the task set (*OE*), and index of the task whose schedulability needs to be tested (i). The algorithm is implemented by deriving the energy increase of each task due to voltage scaling and returning the index of the task that incurs the minimum energy increase. Index of any candidate task is returned if all tasks with equal or higher priorities reach the highest processor voltage level.

The schedulability checking algorithm (SCA) is described in Fig. 4. Inputs to SCA are the array of S_i (S), the array of the overall execution time of tasks in the task set (*OE*), and the index of the task whose schedulability needs to be tested (i). Line 1 initializes the flags *Schedulable* and *Demand* to 0. For each scheduling point of task τ_i, lines 3–6 compute the total time demand of task τ_i. Lines 7–12 check the schedulability of the task and set the *Schedulable* flag accordingly. The time complexity of SCA algorithm is $O(nR)$.

Unlike the task level feasibility analysis algorithm of Zhang and Chakrabarty (2006), T-DVS does not need to exhaustively explore all L^n possible combinations of tasks and voltage levels. The first feasible schedule generated by the algorithm is the desired task schedule and taken as the output. The time complexity of T-DVS algorithm is dominated by the complexity of feasibility analysis and voltage scaling. The feasibility analysis and voltage scaling involves $nL \times (nR + n)$ iterations for each task; thus, the time complexity of T-DVS per task is $O(n^2RL)$, which is one order of magnitude lower than that of previous techniques (Zhang and Chakrabarty, 2006). The online reevaluation of T-DVS algorithm is much simpler and takes constant time. The operation of the reevaluation is detailed in Section 4.2.

4. Online reevaluation of DVS policies

Offline scheduling assumes that all tasks exhibit the worst case execution time and all faults occur during the checkpointing. However, the runtime behavior of task execution and fault occurrences can vary significantly and the average case characteristics are considerably better than the worst case characteristics. Hence, the online reevaluation of DVS policies that adapts the offline schedules to the runtime characteristics of task execution and fault occurrences can save significant energy. The proposed offline algorithms, A-DVS and T-DVS, provide efficient mechanisms to exploit the slack generated in the runtime to slow down the processor to save energy. Note that A-DVS and T-DVS feasibility analysis guarantees that offline schedules meet all timing constraints. Therefore, dynamic DVS policies proposed in this section only need to ensure that the feasibility of the modified task schedules is preserved in the runtime.

4.1. Reevaluation of DVS policies at application level

For a given task set, the output of the A-DVS algorithm described in Section 3.2 is a voltage level l below which the input task set becomes infeasible. This implies that one or more tasks in the task set fail to satisfy Eq. (7) at the voltage level $(l-1)$. In the runtime, not all tasks execute up to their worst case execution times and not all faults occur during task executions. Hence, the slack generated in the runtime could be used to dynamically scale down the processor speed to save energy. It is assumed that tasks ready for execution are put into a ready queue. The online DVS policy manager runs a test to determine whether the cumulative slack is sufficient to slow down the processor for all the unexecuted lower priority tasks in the ready queue. The test compares the amount of time needed for all the unexecuted lower priority tasks in the ready queue to be feasible at $(l-1)$ or a lower voltage level with the available slack, as is discussed below.

Let *slk* denote the accumulated slack time and slk_i denote the slack time from task τ_i. The accumulated slack *slk* can be expressed as the sum of the slack from individual tasks. For instance, the accumulated slack can be written as

$$slk = \sum_{i=1}^{n} slk_i$$

for $1 \le i \le n$. The slack from individual tasks is updated regularly at several time instants. The slack slk_i from the task τ_i is initialized to 0 and updated at the end of the execution of the task to incorporate the generated slack. It is reset to 0 at the deadline of the task, indicating that the slack from the task is expired when a new instance of the task is released. When the accumulated slack is consumed by lower priority tasks, the slack from the task of the highest priority is consumed first. For example, if the priority of task τ_1 is higher than that of task τ_2 and the available accumulated slack is composed of slk_1 and slk_2, the slk_2 is not consumed until the slk_1 is used up when the accumulated slack is utilized to slow down the processor.

Define the execution time overflow as the additional time required by a task to be feasibly scheduled by each scheduling point at a certain voltage level. Let $ovfl_{il}$ denote the execution time overflow of task τ_i at the voltage level l. The $ovfl_{il}$ is set to 0 if task τ_i is schedulable at the voltage level l, else it is computed as the difference between the worst case response time R_{il} of task τ_i at the voltage level l and the deadline D_i of task τ_i, as is given below:

$$ovfl_{il} = \begin{cases} R_{il} - D_i & R_{il} \ge D_i \\ 0 & R_{il} < D_i \end{cases} \quad (8)$$

Consider an offline application level task schedule with the voltage level of l and assume in the runtime the execution of task τ_{i-1} is finished. The voltage level of the processor can be feasibly scaled down to $(l-1)$ if the accumulated slack time *slk* is greater than the sum of the execution time overflows of all the remaining unexecuted lower priority tasks in the ready queue. In other words,

Procedure D-ADVS(i, l)

1. $slk = \sum_{p=1}^{i-1} slk_p$; $ovfl_{sum}$ =0
2. $ovfl_{sum}$ = Overflow(i, l)
3. **while** $slk \geq ovfl_{sum}$ **do**
4. $l = l - 1$;
5. $slk = slk - ovfl_{sum}$
6. $ovfl_{sum}$ = Overflow(i, l)
7. **end while**
8. **return**

Fig. 5. Dynamic application level voltage scaling algorithm (D-ADVS).

Procedure Overflow (i, l)

1. update $ovfl_{sum} = 0$
2. **for** $i \leq j \leq n$ **do**
3. retrieve $ovfl_{j,(l-1)}$ from the lookup table
4. $ovfl_{sum} = ovfl_{sum} + ovfl_{j,(l-1)}$
5. **end for**
6. **return** $ovfl_{sum}$

Fig. 6. Runtime execution time overflow retrieval algorithm.

the processor can be scaled down to the voltage level $(l-1)$ if the inequality

$$slk = \sum_{p=1}^{i-1} slk_p \geq \sum_{p=i}^{n} ovfl_{p,(l-1)} \tag{9}$$

holds, assuming the tasks τ_p for $i \leq p \leq n$ are in the ready queue.

As is shown in Eq. (8), the straightforward approach to compute $ovfl_{il}$ requires iteratively estimating the response times R_{il} of task τ_i, hence it is highly computation intensive. An alternate simple yet efficient approach is proposed as follows. For the task τ_i, $ovfl_{il}$ is the minimum of the differences between t_2 and t_1, where t_2 is the total demand for processor time by the task at the voltage level l and t_1 is the scheduling points of the task. As a result, the execution time overflow $ovfl_{il}$ of task τ_i $(1 \leq i \leq n)$ at the voltage level l $(1 \leq l \leq L)$ is given by

$$ovfl_{il} = \max(\min_{t_1}(t_2 - t_1), 0) \quad t_1 \in S_i, \tag{10}$$

where t_2 is given by

$$t_2 = \sum_{p=1}^{i} OE_p \times \left\lceil \frac{s_{iq}}{T_p} \right\rceil,$$

S_i is the set of scheduling points for task τ_i, and s_{iq} is the qth scheduling point in S_i. OE_p is given in Eq. (2), and is re-written as

$$OE_p = \frac{C_p}{f_p} + O_{pl} \times c_s + \frac{k_i C_p}{f_p(O_{pl}+1)} + 2k_i c_s.$$

Since both t_1 and t_2 are computed during the offline feasibility analysis of the A-DVS algorithm, the execution time overflow given in Eq. (10) can be pre-computed during the offline feasibility analysis and stored in system memory to form a lookup table. The lookup table is implemented in software program as an array or associative array. The table can be pre-calculated and stored in static program storage or can be calculated as part of a program's initialization phase. In the runtime, the scheduler searches the lookup table at the end of each task execution to calculate the sum of the execution time overflows of all the remaining unexecuted lower priority tasks, compares the derived execution time overflows with the accumulated slack time, and determines if the processor can be feasibly scaled down according to Eq. (9). This strategy of dynamic scheduling significantly reduces the runtime computation overhead without compromising the feasibility performance of the task schedule.

As is shown in Eq. (10), the proposed scheme computes the execution time overflow based on the minimum of differences between t_2 and t_1, hence it provides a better opportunity to scale the processor frequency. Fig. 5 demonstrates the algorithms for the runtime reevaluation of DVS policies at the application level. Reevaluation of DVS polices is performed whenever a task instance finishes its execution. This strategy avoids incurring during the task execution any extra overhead that may cause the task to miss its deadline. Meanwhile, it ensures that the current accumulated slack time is checked sufficiently frequently such that the supply voltage can be scaled down opportunely to achieve energy savings if the accumulated slack time is large enough. The dynamic application level voltage scaling algorithm, D-ADVS, is shown in Fig. 5, where i is the index of the task τ_i to be executed, l is the voltage level of task τ_i, and $ovfl_{sum}$ is the sum of the overflows of unexecuted lower priority tasks in the ready queue. Line 1 updates the slk and initializes $ovfl_{sum}$ to 0. In line 2, subroutine **Overflow** is called to calculate the sum of the overflows of unexecuted lower priority tasks in the ready queue. Lines 3 to 6 iteratively scale down the processor voltage level if the accumulated slack time is large enough. Line 4 scales down the voltage level and operating frequency for all the remaining tasks by one level. Line 5 updates the slk by reducing $ovfl_{sum}$ and line 6 updates the $ovfl_{sum}$ by calling subroutine **Overflow** to explore the possibilities of further scaling down the processor voltage level. The **Overflow** subroutine is described in Fig. 6, where the notations i, l, and $ovfl_{sum}$ have the same definition as in the D-ADVS algorithm. The algorithm retrieves overflows of tasks τ_j $(i \leq j \leq n)$ at the voltage level $(l-1)$ from the lookup table stored in system memory in line 3, and calculates the sum of the overflows in line 4. Line 6 returns the $ovfl_{sum}$. The overall time complexity of the D-ADVS algorithm is $O(nL)$.

The example shown in Table 1 demonstrates the application level runtime DVS reevaluation before a certain scheduling point. Consider a task set of four periodic tasks running on a processor which supports 3 voltage levels, as shown in Table 1. Execution time overflows for each task at the voltage level 3 are 0, denoting a feasible schedule at the voltage level 3. At the voltage level 2, task τ_1 and τ_2 are schedulable while task τ_3 and τ_4 are not schedulable since $ovfl_{12} = ovfl_{22} = 0$ but $ovfl_{32} = 1$ and $ovfl_{42} = 2$. Therefore, the processor runs at the voltage level 3 to maintain the feasibility of the schedule. However, if the slack slk_2 generated in the runtime satisfies $slk_2 \geq (ovfl_{32} + ovfl_{42}) = 3$, the processor can be scaled down to level 2 without violating the feasibility of the schedule. The same procedure can be utilized iteratively at successive voltage levels and for successive slacks produced in the system.

4.2. Reevaluation of DVS policies at task level

The dynamic reevaluation of DVS strategies at the task level is similar to the reevaluation of DVS policies at the application level. During the offline scheduling, the T-DVS algorithm given

Table 1
Runtime DVS reevaluation at application level.

Tasks	Voltage level		
	1	2	3
τ_1	$ovfl_{11} = 0$	$ovfl_{12} = 0$	$ovfl_{13} = 0$
τ_2	$ovfl_{21} = 3$	$ovfl_{22} = 0$	$ovfl_{23} = 0$
τ_3	$ovfl_{31} = 4$	$ovfl_{32} = 1$	$ovfl_{33} = 0$
τ_4	$ovfl_{41} = 5$	$ovfl_{42} = 2$	$ovfl_{43} = 0$

Procedure D-TDVS(i, l)

1. $slk = \sum_{p=1}^{i-1} slk_p$;
2. retrieve $ovfl_{i,(l-1)}$ of task τ_i from lookup table
3. **while** $slk \geq ovfl_{i,(l-1)}$ **do**
4. $\quad l = l - 1$;
5. $\quad slk = slk - ovfl_{i,(l-1)}$
6. $\quad$ update $ovfl_{i,(l-1)}$ based on the lookup table
7. **end while**
8. **return**

Fig. 7. Dynamic task level voltage scaling algorithm (D-TDVS).

in Section 3.3 statically derives the optimum combination of frequency allocations for all the tasks and stores the information, such as the pre-computed execution time overflows and overall execution times of tasks, in a lookup table for dynamic adaptation. In the runtime, the frequency of each task is scaled individually without affecting the operating frequency of other tasks.

Fig. 7 shows the dynamic reevaluation of T-DVS scheduling, referred as to D-TDVS. Assume task τ_{i-1} finishes its execution and task τ_i is scheduled to execute at the voltage level l. The scheduler checks if the accumulated slack time *slk* from tasks τ_j $(1 \leq j \leq i-1)$ is large enough to scale down the frequency f_i of task τ_i by one or more levels. This can be verified by comparing the execution time overflow of task τ_i at lower voltage levels with the current available slack *slk*. If the available slack *slk* is greater than the execution time overflow of task τ_i at the voltage level $(l-1)$, the processor voltage is scaled down by one level and the current available slack is updated, else the slack continues to accumulate for tasks with lower priorities until it is updated when task τ_i is finished.

The runtime reevaluation of task level scheduling can be even simpler. That is, after task τ_{i-1} finishes its execution at the operating frequency f_{i-1} and the current slack time *slk* is ready, the new candidate operating frequency f_i' of task τ_i is derived as

$$f_i' = f_i \times \frac{OE_i}{slk + OE_i},$$

where OE_i is the overall execution time of task τ_i at the frequency f_i. Comparing f_i' with the lower frequencies supported by the processor determines whether or not the operating frequency f_i of task τ_i can be feasibly scaled down. This scheme takes constant time. It requires $n \times (L-1)$ extra memory space to store the overall execution time of each task for the adaptation of the offline task schedule to the runtime behavior of tasks and fault characteristics.

5. Simulation results and overhead analysis

The proposed scheduling schemes consist of offline components (A-DVS and T-DVS) and online components (D-ADVS and D-TDVS). Energy efficiency and fault tolerance capabilities of the proposed schemes were validated using extensive simulation experiments. The scheduling overhead of the online components of the proposed schemes was also evaluated using Xtrem (Contreras et al., 2004), a SimpleScalar-based Intel XScale processor simulator.

Real-life task set benchmarks from Kim et al. (1996) and Locke et al. (1991) were used to compare the performance and energy characteristics of the proposed schemes with those of the scheduling schemes presented in Zhang and Chakrabarty (2006). Similar to the approach in Zhang and Chakrabarty (2006), the worst case number of fault occurrences in a task instance is assumed to be a fixed number k, which is essentially the number of fault occurrences of the longest task at the lowest frequency level. This strategy of selecting k ensures a fair comparison between the proposed schemes and the benchmarking schemes in Zhang and Chakrabarty (2006). Since the proposed schemes utilize the slack due to uncertainties in fault occurrences for energy savings, whether or not using a fixed number of fault occurrences does not affect the effectiveness of the proposed schemes. Two DVS-capable processors, Transmeta Crusoe supporting 5 voltage and frequency levels (Transmeta Corporation) and Intel XScale PXA260 supporting 3 voltage and frequency levels (Intel Corporation), were used for estimating the energy consumption. The discrete frequencies, supply voltages, and TDP power of the two processors are listed in Table 2.

Table 2
Processor frequencies, supply voltages and power (in Transmeta Corporation; Intel Corporation).

Processors	Characteristics		
	Frequency (MHz)	Supply voltage (V)	Power (mW)
Intel XScale PXA260	200	1.0	178
	300	1.1	283
	400	1.3	411
Transmeta Crusoe	300	1.2	1300
	400	1.225	1900
	533	1.35	3000
	600	1.5	4200
	667	1.6	5300

5.1. Energy characteristics

Two real-life task sets, Inertial Navigation System (INS) (Locke et al., 1991) and Computer Numerical Control (CNC) (Kim et al., 1996), were utilized to benchmark the energy consumption of the proposed scheduling algorithms. The characteristics of the benchmarking task sets are shown in Table 3, where the worst case execution time of a task is assumed to correspond to the maximum processor speed. It is assumed that both checkpointing and data retrieval take 0.4 ms, and the energy overhead of both checkpointing and data retrieval is 160 μJ. It is also assumed that online energy savings are achieved by only using the slack due to variations in fault occurrences.

The online and offline components are integral parts of the proposed application and task level scheduling schemes. Energy savings are achieved by primarily using online components that are enabled by the offline components. Energy values are obtained by multiplying processor power consumption and task execution time and considering checkpointing overhead and DVS transition overhead. Simulation results are reported for both application and task level techniques and compared with the JFTC, JFTA and JFTT techniques presented in Zhang and Chakrabarty (2006). JFTC, JFTA and JFTT refer to offline constant frequency, application level voltage scaling, and task level voltage scaling schemes, respectively.

The online component is essentially a greedy heuristic since it scales down the processor speed to save energy once the available slack time is large enough. An exhaustive search based online scheme is then developed to investigate the optimality of the proposed scheme. In the exhaustive search based online scheme, all possible adaptations of offline task schedules due to the slack generated in the runtime are produced and the energy consumption of each adapted task schedule is computed. The adapted task schedule with the minimum energy consumption is deemed to be optimal. The optimal adapted task schedule, referred to as *Optimal*, is compared with the proposed scheme in energy consumption. In the presented results, E_{13} denotes the percentage of $(E_1 - E_3)/E_1 \times 100\,\%$, E_{23} denotes the percentage of $(E_2 - E_3)/E_2 \times 100\,\%$, E_{43} denotes the percentage of $(E_4 - E_3)/E_4 \times 100\,\%$, and *NF* denotes an infeasible schedule.

Table 4 shows that the proposed application level scheduling scheme saves 22–52% energy over JFTC and 22–50% energy over

Table 3
INS and CNC real-life task sets.

Task index	Period (μs)		Deadline (μs)		WCET (μs)	
	INS	CNC	INS	CNC	INS	CNC
1	2500	2400	2500	2400	1180	35
2	40,000	2400	40,000	2400	4280	40
3	625,000	4800	625,000	4800	10,280	80
4	1,000,000	4800	1,000,000	4800	20,280	720
5	1,000,000	2400	1,000,000	2400	100,280	165
6	1,250,000	2400	1,250,000	2400	25,000	165
7		9600		4000		570
8		7800		4000		570

Table 4
Energy consumption of application level voltage scaling on Transmeta Crusoe processor.

Task set	k	JFTC (Zhang and Chakrabarty, 2006) E_1 (mJ)	JFTA (Zhang and Chakrabarty, 2006) E_2 (mJ)	Proposed E_3 (mJ)	Optimal E_4 (mJ)	E_{13} (%)	E_{23} (%)	E_{43} (%)
CNC	1	18.1	14.6	10.1	9.47	44.2	30.8	−6.21
	2	24.3	21.2	12.9	12.51	46.9	39.2	−2.97
	3	29.8	26.6	15.1	14.21	49.3	43.2	−5.89
	4	34.9	33.6	16.7	15.90	52.1	50.3	−4.85
	5	*NF*	*NF*	18.6	16.99	–	–	−8.65
INS	1	6050.7	5467.2	3986.0	3986.0	34.1	27.1	0.00%
	2	6735.1	6735.1	3986.0	3986.0	22.1	22.1	0.00
	3	7300.2	7300.2	5617.5	5617.5	23.1	23.1	0.00
	4	*NF*	*NF*	5935.8	5919.8	–	–	−0.27

Table 5
Energy consumption of task level voltage scaling on Transmeta Crusoe processor.

Task set	k	JFTC (Zhang and Chakrabarty, 2006) E_1 (mJ)	JFTT (Zhang and Chakrabarty, 2006) E_2 (mJ)	Proposed E_3 (mJ)	E_{13} (%)	E_{23} (%)
CNC	1	18.1	14.9	10.0	44.8	32.9
	2	24.3	21.1	12.9	46.9	38.9
	3	29.8	26.7	13.4	55.0	49.8
	4	34.9	34.1	14.5	58.5	57.5
	5	*NF*	*NF*	13.1	–	–
INS	1	6050.7	5457.6	4087.9	32.4	25.1
	2	6735.1	6222.1	5070.4	24.7	18.5
	3	7300.2	7284.1	5637.4	22.8	22.6
	4	*NF*	*NF*	5961.6	–	–

JFTA on Crusoe processors. Table 4 also shows that the discrepancy in energy consumption of the proposed application level scheduling scheme and the exhaustive search based optimal scheme is up to 8% for CNC task set. The discrepancy is relatively small for INS task set. This is because execution times of tasks in INS task set are relatively long and much more slack is needed to scale down the processor speed. Due to space limitation, the results on the comparison of the proposed scheme and the exhaustive search based optimal method is reported only for INS task set on Transmeta Crusoe processor.

Table 5 shows that the proposed task level scheduling scheme saves 22–58% energy over JFTC and 18–57% energy over JFTT on Crusoe processors. Similarly, the energy savings of the proposed application and task level schemes over the benchmarking schemes on XScale processors are shown in Tables 6 and 7. It can be drawn that the proposed scheduling schemes achieve significant energy savings when compared to the benchmarking scheduling schemes. This is primarily due to the runtime slack from uncertainties of fault occurrences. It also can be drawn (e.g. for $k = 5$ for CNC and $k = 4$ for INS) that the proposed techniques have higher fault tolerance

Table 6
Energy consumption of application level voltage scaling on Intel XScale processor.

Task set	k	JFTC (Zhang and Chakrabarty, 2006) E_1 (mJ)	JFTA (Zhang and Chakrabarty, 2006) E_2 (mJ)	Proposed E_3 (mJ)	E_{13} (%)	E_{23} (%)
CNC	1	7.6	8.2	3.6	52.6	56.1
	2	12.8	13.8	7.1	44.5	48.6
	3	17.6	18.8	9.0	48.9	52.1
	4	22.2	22.2	10.5	52.7	52.7
	5	*NF*	*NF*	12.9	–	–
INS	1	1326.2	1326.2	923.8	30.3	30.3
	2	1853.6	1853.6	1248.4	32.6	32.6
	3	2298.2	2298.2	1510.6	34.3	34.3
	4	*NF*	*NF*	1758.8	–	–

Table 7
Energy consumption of task level voltage scaling on Intel XScale processor.

Task set	k	JFTC (Zhang and Chakrabarty, 2006) E_1(mJ)	JFTT (Zhang and Chakrabarty, 2006) E_2 (mJ)	Proposed E_3 (mJ)	E_{13} (%)	E_{23}(%)
CNC	1	7.6	9.1	3.6	52.6	60.4
	2	12.8	14.5	7.5	41.4	48.3
	3	17.6	18.5	9.5	46.0	48.6
	4	22.2	22.5	11.0	50.5	51.1
	5	*NF*	*NF*	10.8	–	–
INS	1	1326.2	1327.5	932.5	30.3	30.4
	2	1853.6	1855.9	1292.6	30.3	30.4
	3	2298.2	2299.0	1496.5	34.9	34.9
	4	*NF*	*NF*	1726.5	–	–

capabilities due to relaxed utilization constraints of the proposed ECRMA-based scheduling approach.

5.2. Runtime overhead analysis

The proposed quasi-static scheduling scheme comprises two components. One is the offline scheduling component and the other is the runtime counterpart of the offline component. Since it is the overhead of the runtime component that has adverse impact on system timeliness, the complexity of the runtime component of the quasi-static scheme is evaluated in this section using Xtrem (Contreras et al., 2004), a high level functional power simulator tailored for Intel Xscale Technology-based systems.

Xtrem is a powerful infrastructure capable of providing power and cycle level estimation for C-based applications targeted to Intel Xscale core. It is based on the widely used SimpleScalar-Arm architecture simulator and is used for verifying the time complexity of the proposed schemes and estimating the runtime impact of dynamic DVS policies on the schedule feasibility. Due to space limitation, only the overhead of the application level scheduling scheme is analyzed in this section.

The proposed dynamic D-ADVS scheme reduces runtime overhead by utilizing the pre-computed execution time overflow that is stored in system memory. The D-ADVS proceeds in three steps: (1) adding up the overflows of unexecuted tasks, (2) comparing the current available slack time with the sum of the overflows to make a decision on voltage scaling, and (3) updating the accumulated slack time. The D-ADVS is called whenever a task instance finishes its execution, which ensures that the feasibility of the offline schedule is maintained and the runtime generated slack time is utilized opportunely for energy savings. The D-ADVS scheme involves three primitive operations: addition, multiplication and division. Each operation takes 7 cycles on Xtrem. The experiment results show that the overall overhead of the proposed D-ADVS is 821 cycles to execute the first instances of all tasks in the CNC task set. Since the D-ADVS is called at the end of the execution of each task instance and repeats its three steps over each unexecuted task in the task set, it incurs the worst case overhead at its first call. Similarly, the D-ADVS incurs the best case overhead at its last call. It is shown in the experiment results that the average cycles for the worst case overhead is 190 and the average cycles for the best case overhead is 81, which translates to 0.48 μs and 0.20 μs, respectively, at the operating frequency of 400 MHz. This time overhead is negligible when compared to the execution times of tasks in the CNC task set.

6. Implementation of scheduling algorithms on a real-life test bed

This section presents the implementation and validation of the proposed energy-efficient task scheduling schemes for hard real-time embedded systems. A real-life test bed has been developed to accurately benchmark the energy savings of the proposed scheduling schemes. The test bed comprises a dual-core Intel T2500 processor with dynamic voltage scaling capability and runs the Linux Fedora 8 based hard real-time scheduling. A detailed description of the implementation was presented in Wei et al. (2011).

6.1. Implementation process and energy measurement method

The implementation of an energy-efficient task scheduling algorithm requires that the hardware and software platform on which the algorithm is to be implemented supports hard real-time scheduling, fixed priority and preemptive scheduling, and dynamic voltage scaling. A mini-ITX Express motherboard from Radisys Corporation (2002) was selected as the hardware platform due to its support of DVS and ease with energy measurement. The embedded motherboard mainly comprises an Intel Core Duo T2500 processor and 512M DDR2 memory module. T2500 is a DVS-capable processor with 2MB L2 cache. The Linux Fedora 8 operating system, one of the most widely used operating systems, was chosen as the software platform for the implementation. It supports two fixed priority and preemptive real-time scheduling policies: First-In-First-Out (FIFO) and Round-Robin (RR). The FIFO policy was selected as the basis for implementing RMA-based scheduling schemes. The rate monotonic scheduling scheme is utilized to assigned priorities to tasks with different periods while FIFO is used to break ties when tasks have the same priorities. The Linux FIFO policy provides a simple yet efficient approach to implementing fixed-priority preemptive scheduling algorithms.

The implementation of the RMA-based task scheduling algorithms consists of two major steps: task generation and task scheduling. In the first step, Dhrystone-based independent and periodic tasks were generated and task characteristics were derived. In the second step, the generated tasks were scheduled using scheduling algorithms such as the JFTA (Zhang and Chakrabarty, 2006) and the proposed A-DVS.

Dhrystone, a synthetic computing benchmark program developed to be representative of system programming (Weicker, 1984), was selected and modified to generate independent and periodic tasks. Dhrystone was written in C language code, has small size, and is portable to a large number of platforms and processor architectures. These characteristics make Dhrystone a popular benchmark in embedded applications due to its small memory requirements. The output of Dhrystone benchmark program is the number of iterations of the Dhrystone main code in unit time, which is derived by dividing a predefined number of iterations of the Dhrystone main code by the corresponding execution time. Tasks with different execution times are generated by varying the number of iterations of the Dhrystone main code. Ten tasks the execution times of which range from 1 ms to 1000 ms were generated using the Dhrystone benchmark program. Task utilizations were generated based on the Beta distribution of probability and were limited to being less than ln 2, the asymptotic bound of RMA (Wei et al., 2008). Task periods

Table 8
Energy consumptions of CPU and system board under JFTA and the proposed quasi-static application level scheduling scheme.

Mini-ITX motherboard	k	JFTA (Zhang and Chakrabarty, 2006) E_J (J)	Proposed E_A (J)	E_{AJ} (J)
Core	0	312.1	312.1	0
	1	396.7	318.5	19.7
	2	408.4	318.5	22.0
	3	418.1	325.6	22.1
	4	436.2	330.7	24.2
	5	*NF*	341.2	–
System board	0	489.12	489.12	0
	1	488.56	490.46	−0.38
	2	488.72	490.82	−0.45
	3	489.1	491.4	−0.46
	4	491.6	490.4	0.25
	5	*NF*	491.0	–

were hence derived as the ratio of the task execution times to the task utilizations.

Since the ATX 4-pin connector on the motherboard exclusively provides 12 V voltage to the voltage regulator module (VRM) of CPU and the ATX 20-pin connector on the motherboard provides 12 V, 5 V, and 3.3 V voltages to components on system board, the energy consumption of CPU and system board can be approximated by the energy delivered from ATX power supply connectors. This delivered energy can be derived by multiplying the currents flowing through ATX connectors by the voltages at the ATX connectors. Considering the fact that a VRM can achieve energy efficiency of up to 95% and difficulties to directly measure energy consumption of an onboard device, this strategy for energy estimation can be used to sketch energy consumption of CPU and system board. More specifically, measurement of energy consumption is accomplished by using a DAQ system and Tektronix A622 AC/DC current probe. The DAQ system is comprised of an NI PCI-6040E DAQ, NI BNC-2110 connector block, and a host computer with LabView, as is shown in Fig. 8.

6.2. Numerical results

The offline scheduling algorithms presented in Zhang and Chakrabarty (2006) and the proposed quasi-static scheduling schemes were implemented on the test bed. Due to space limitation, only the results for the JFTA (Zhang and Chakrabarty, 2006) and the proposed application level scheduling scheme (the A-DVS combined with its dynamic counterpart D-ADVS) were reported. One core of the Core Duo processor Intel T2500 was disabled in the implementation. 10 generated Dhrystone-based tasks were scheduled to execute on the test bed using the JFTA and the proposed application level scheme, respectively. Table 8 shows the energy consumptions of the core and system board, respectively. $E_{JA} = (E_J - E_A)/E_J \times 100\,\%$ denotes the energy savings of the proposed scheme when compared to the benchmarking scheme JFTA, where E_J and E_A represent energy consumptions of the task set under JFTA and the proposed application level scheme, respectively. *NF* denotes that the tasks in the task set cannot be feasibly scheduled.

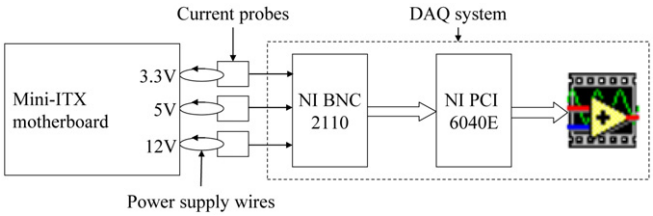

Fig. 8. DAQ system and current probes for energy measurement.

The energy consumption of the proposed quasi-static application scheme is verified and compared with that of the JFTA. Table 8 shows that as compared to the proposed scheme, the JFTA consumes the same core energy in the absence of fault occurrences and consumes about 20% more core energy in the presence of fault occurrences. This is because JFTA is an offline scheduling algorithm that considers the worst case fault occurrences to maintain the schedule feasibility. On the contrary, the proposed scheme is a quasi-static scheduling scheme the offline components of which enables the online components to save energy by utilizing uncertainties in fault occurrences.

The energy consumptions of the system board excluding the processor are close for the two scheduling algorithms under different fault arrival rates. For example, the difference between the JFTA and the proposed scheme in energy consumption of the system board is less than 0.5% with the number of fault occurrences ranging from 0 to 5, as is shown in Table 8. Furthermore, the energy consumption of the system board when it is idle is 479.8 J, which is about 10 J less as compared to the energy consumption of the system board under the load of the Dhrystone-based task set.

There are three possible reasons for the relative stableness in the energy consumption of the system board. First, the Dhrystone is a CPU-intensive benchmark program and it does not intensively exercise the system board, especially the memory module, to store and load data. Second, the JFTA and the proposed scheduling algorithms are also CPU-intensive and their impact on the system board energy consumption is small. Finally, the total size of the instructions of the schedulers and the Dhrystone-based tasks in the form of an executable file is about 20 k. This file could be readily fit in the 2MB L2 cache of the Intel T2500 processor, which reduces the memory access overheads.

7. Conclusion

This paper presents efficient quasi-static scheduling schemes that combine the offline feasibility analysis of RMA schedules and the online voltage scaling for hard real-time systems. The proposed schemes captures the effects of voltage scaling on system reliability, and achieve energy savings by utilizing both the static slack in offline task schedules and dynamic slack due to variations in task execution time and uncertainties of fault occurrences. The extension of the offline algorithms is enabled by pre-computing and saving in a lookup table the maximum slack requirements for the processor to slow down in the runtime. The stored slack time requirements are retrieved and compared at runtime with the accumulated slack. The online voltage scaling is performed whenever the generated slack time is equal to or greater than the stored slack requirements. Simulation results based on two real-life test sets and processor data sheets show that the proposed techniques achieve energy savings of up to 50% over the

state-of-the-art schemes. A SimpleScalar-based Intel XScale processor simulator, Xtrem, was used to evaluate the proposed scheduling schemes in addition to extensive simulation experiments. A hard real-time test bed has been designed and the proposed algorithms were also verified on the test bed.

Acknowledgments

This work was supported in part by the Fundamental Research Funds for the Central Universities of China under the Grant No. 78220021.

References

Axer, P., Sebastian, M., Ernst, R., 2011. Reliability analysis for mpsocs with mixed-critical, hard real-time constraints. In: The International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), pp. 149–158.

Benini, L., Bogliolo, A., Micheli, G., 2000. A survey of design techniques for system-level dynamic power management. IEEE Transactions on VLSI Systems 8 (3), 299–316.

Contreras, G., Martonosi, M., Peng, J., Ju, R., Lueh, G., Xtrem, 2004. A power simulator for the intel xscale core. In: ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems.

Cormen, T., Leiserson, C., Rivest, R., Stein, C., 2001. Introduction to Algorithm. The MIT Press.

Ernst, D., Das, S., Lee, S., Blaauw, D., Austin, T., Mudge, T., Kim, N.S., Razor, K.F., 2004. Circuit-level correction of timing errors for low-power operation. IEEE Micro 24 (6), 10–20.

Gruian, F., 2001. Hard real-time scheduling for low-energy using stochastic data and dvs processors. In: Proceedings of the International Symposium on Low Power Electronics and Design, pp. 46–51.

Huang, K., Santinelli, L., Chen, J., Thiele, L., Buttazzo, G., 2009. Adaptive dynamic power management for hard real-time systems. In: Proceedings of the IEEE Real-Time Systems Symposium.

Huang, J., Blech, J., Raabe, A., 2011. Analysis and optimization of fault-tolerant task scheduling on multiprocessor embedded systems. In: The International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), pp. 247–256.

Huang, K., Santinelli, L., Chen, J., Thiele, L., Buttazzo, G., 2011. Applying real-time interface and calculus for dynamic power management in hard real-time systems. Real-Time Systems Journal 47 (2), 163–193.

Intel Corporation. Intel PXA26x processor family electrical, mechanical, and thermal specification datasheet. Available from: http://developer.intel.com.

Iqbal, N., Siddique, M., Henkel, J., 2011. Seal: Soft error aware low power scheduling by monte carlo state space under the influence of stochastic spatial and temporal dependencies. In: Proceedings of the ICCAD, pp. 134–139.

Izosimov, V., Pop, P., Eles, P., Peng, Z., 2008. Scheduling of fault-tolerant embedded systems with soft and hard timing constraints. In: Proceedings of the DATE.

Jejurikar, R., Pereira, C., Gupta, R., 2004. Leakage aware dynamic voltage scaling for real-time embedded systems. In: Proceedings of the DAC, pp. 275–280.

Kim, N., Ryu, M., Hong, S., Saksena, M., Choi, C., Shin, H., 1996. Visual assessment of a real-time system design: a case study on a CNC controller. In: Proceedings of the RTSS.

Krishna, C., Lee, Y., 2003. Voltage-clock-scaling adaptive scheduling techniques for low power in hard real-time systems. IEEE Transactions on Computers 52 (12), 1586–1593.

Kwak, S., Choi, B., Kim, B., 2001. An optimal checkpointing strategy for real-time control systems under transient faults. IEEE Transactions on Reliability 50 (3), 293–301.

Langley, T., Koga, R., Morris, T., 2003. Single-event effects test results of 512mb sdrams. IEEE Radiation Effects Data Workshop July, 98–101.

Lehoczky, J., Sha, L., Ding, Y., 1989. The rate monotonic scheduling algorithm: exact characterization and average case behavior. In: IEEE Real-Time Systems Symposium.

Liu, C., Layland, J., 1973. Scheduling algorithms for multiprogramming in a hard real time environment. Journal of the ACM 20 (1), 46–61.

Locke, D., Vogel, D., Mesler, T., 1991. Building a predictable avionics platform in ada: a case study. In: The Proceedings of Real-Time Systems Symposium.

Martin, S., Flautner, K., Mudge, T., Blaauw, D., 2002. Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads. In: Proceedings of the ICCAD.

Melhem, R., Mosse, D., Elnozahy, E., 2004. The interplay of power management and fault recovery in real-time systems. IEEE Transactions on Computers 53 (2), 217–231.

Mochocki, B., Hu, X., Quan, G., 2007. Transition-overhead-aware voltage scheduling for fixed-priority real-time systems. ACM Transactions on Design Automation of Electronic Systems 12 (2).

Normand, E., 1996. Single event upset at ground level. IEEE Transactions on Nuclear Science 43 (6), 2742–2750.

Perathoner, S., Chen, J., Lampka, K., Stoimenov, N., Thiele, L., 2010. Combining optimistic and pessimistic dvs scheduling: an adaptive scheme and analysis. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 131–138.

Pillai, P., Shin, K., 2001. Real-time dynamic voltage scaling for low-power embedded operating systems. In: Proceedings of the ACM Symposium on Operating Systems Principle.

Pop, P., Poulsen, K., Izosimov, V., Eles, P., 2007. Scheduling and voltage scaling for energy/reliability trade-offs in fault-tolerant time-triggered embedded systems. In: Proceedings of the 5th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis.

Pradhan, D., 1986. Fault Tolerance Computing: Theory and Techniques. Prentice Hall.

Radisys Corporation, 2002. Endura TP945GM motherboard. Available from: http://www.radisys.com.

Reed, R., Kinnison, J., Pickel, J., Buchner, S., Marshall, P., Kniffin, S., LaBel, K., 2006. Single-event effects ground testing and on-orbit rate prediction methods: the past, present, and future. IEEE Transactions on Nuclear Science 50 (3), 622–634.

Reinhardt, S., Mukherjee, S., 2000. Transient fault detection via simultaneous multithreading. In: ACM SIGARCH Computer Architecture News.

Saewong, S., Rajkumar, R., 2003. Practical voltage-scaling for fixed-priority rt-systems. In: Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium.

Shafik, R., Al-Hashimi, B., Chakrabarty, K., 2010. Soft error-aware design optimization of low power and time-constrained embedded systems. In: The Proceedings of the DATE.

Shin, Y., Choi, K., 1999. Power conscious fixed priority scheduling for hard real-time systems. In: Proceedings of the DAC.

Shin, K., Lee, Y., 1984. Error detection process-model, design and its impact on computer performance. IEEE Transactions on Computers C-33 (6), 529–540.

Shin, K., Lin, T., Lee, Y., 1987. Optimal checkpointing of real-time tasks. IEEE Transactions on Computers C-36 (11), 1328–1341.

Transmeta Corporation. Transmeta longrun power management - dynamic power management for crusoe processors. Available from: http://www.transmeta.com.

Wei, T., Mishra, P., Wu, K., Liang, H., 2006. Online task-scheduling for fault-tolerant low-energy real-time systems. In: Proceedings of the International Conference on Computer-Aided Design (ICCAD).

Wei, T., Mishra, P., Wu, K., Liang, H., 2008. Fixed-priority allocation and scheduling for energy-efficient fault-tolerance in hard real-time multiprocessor systems. IEEE Transactions on Parallel and Distributed Systems 19 (11), 1511–1526.

Wei, T., Chen, X., Mishra, P., 2011. Design of a hard real-time multi-core testbed for energy measurement. Microelectronics Journal of Elsevier 42 (10), 1176–1185.

Wei, T., Chen, X., Hu, S., 2011. Reliability-driven energy-efficient task scheduling for multiprocessor real-time systems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 30 (10), 1569–1573.

Weicker, R., 1984. Dhrystone: a synthetic systems programming benchmark. Communications of the ACM 27 (10), 1013–1030.

Weste, N., Eshraghian, K., 1992. Principles of CMOS VLSI Design: A System Perspective. Addison-Wesley Publishing Company.

Weulersse, C., Hubert, G., Forget, G., Buard, N., Carriere, T., Heins, P., Palau, J., Saigne, F., Gaillard, R., 2006. Dasie analytical version: a predictive tool for neutrons, protons and heavy ions induced seu cross section. IEEE Transactions on Nuclear Science 53 (4), 1876–1882.

Zhang, Y., Chakrabarty, K., 2003. Energy-aware adaptive checkpointing in embedded real-time systems. In: Proceedings of the DATE.

Zhang, Y., Chakrabarty, K., 2006. A unified approach for fault tolerance and dynamic power management in fixed-priority real-time embedded systems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 25 (1), 111–125.

Zhang, Y., Chakrabarty, K., Swaminathan, V., 2003. Energy-aware fault tolerance in fixed-priority real-time embedded systems. In: Proceedings of the ICCAD.

Zhao, B., Aydin, H., Zhu, D., 2009. Enhanced reliability-aware power management through shared recovery technique. In: Proceedings of the ICCAD, pp. 63–70.

Zhao, B., Aydin, H., Zhu, D., 2011. Generalized reliability-oriented energy management for real-time embedded applications. In: Proceedings of the DAC, pp. 381–386.

Zhu, D., Melhem, R., Mosse, D., 2004. The effects of energy management on reliability in real-time embedded systems. In: Proceedings of the International Conference on Computer-Aided Design.

Zhu, D., Aydin, H., Chen, J., 2008. Optimistic reliability aware energy management for real-time tasks with probabilistic execution times. In: Proceedings of the Real-Time Systems Symposium.

Tongquan Wei received the B.E. degree in Electronics Engineering from Dalian University of Technology, China, the M.S. degree in Computer Engineering from University of Missouri-Rolla (Now Missouri University of Science and Technology), and the Ph.D. degree in Electrical Engineering from Michigan Technological University. He is currently an associate professor in the Computer Science and Technology Department of East China Normal University, Shanghai, China. His research focuses on power and fault tolerance management in real-time embedded systems. He is a member of the IEEE.

Dr. Piyush Mishra is a Lead Engineer in the Supervisory Controls and System Integration lab at GE Global Research, Niskayuna NY. He has published more than 18 research papers in refereed journals and conference proceedings. He serves as Vice-Chair on ACM SIGDA Technical Committee on FPGA and Reconfigurable Computing and as a reviewer for a number of prestigious publications. His research interests include digital hardware design, real-time simulation/emulation, reconfigurable computing, cyber-security, and energy-efficient designs.
Dr. Mishra joined the GEGR after working at the Electrical and Electronics Engineering department at Michigan Technological University where she served as Assistant Professor from 2004–2008. During his tenure there, he led research groups on energy-efficient reliability schemes, served on thesis committee of 4 Ph.D. and 7 master students and taught graduate and undergraduate courses.

Dr. Mishra received a bachelor's degree in electrical and electronics engineering from Birla Institute of Technology in India and Ph.D. in electrical engineering from Polytechnic Institute of NYU in Brooklyn, NY.

Kaijie Wu received the B.E. degree from Xidian University, Xi'an, China, in 1996, the M.S. degree from the University of Science and Technology of China, Hefei, China, in 1999, and the Ph.D. degree in electrical engineering from Polytechnic University (Now Polytechnic Institute of New York University), Brooklyn, New York, in 2004. He is currently an assistant professor in the Department of Electrical and Computer Engineering, University of Illinois at Chicago. His research interests include computer-aided design (CAD) of radiation-hardened VLSI System, countermeasures for side-channel cryptanalysis for crypto devices, and robust and fault-tolerant real-time systems. He is the recipient of the 2004 EDAA Outstanding Dissertation Award for "new directions in circuit and system test." He is a member of the IEEE.

Junlong Zhou received the B.E. degree in Software Engineering from Jiangxi University of Agriculture, China. He is now working towards his M.S. degree in the area of energy efficient fault tolerance real-time systems in Computer Science and Technology Department at East China Normal University.