FACTA UNIVERSITATIS
Series:**Automatic Control and Robotics** Vol. 15, N° 3, 2016, pp. 227 - 236
DOI: 10.22190/FUACR1603227D

DYNAMIC VOLTAGE AND FREQUENCY SCALING IN FAULT TOLERANT REAL-TIME SYSTEMS

UDC (004.413.4:007.52):621.22.01)

Sandra Đošić, Milica Jovanović, Igor Stojanović, Goran Lj. Đorđević

University of Niš, Faculty of Electronic Engineering, Department of Electronics, Niš, Republic of Serbia

Abstract. *In this paper, we analyze the faults tolerance capability of hard real-time systems under power consumption constraints. In particular, we are focused on real-time systems in which:1) the fault tolerance is achieved through time redundancy, which is used for re-executing tasks affected by transient faults, and 2)the dynamic voltage scaling used to manage power consumption. We proposea heuristic-based algorithm to find processor frequencies at which each real-time task should be executed so that: 1) the power consumption does not exceed the upper power limit; 2) the fault tolerance capability is maximized, and 3) all real-time requirements of individual tasks are met. Evaluations on synthetic task sets show that the proposed algorithm attains target power consumption level with minimum degradation in fault tolerance.*

Key words:*Real-time systems, power reduction, fault tolerance*

1. INTRODUCTION

Real-time systems (RTS) are nowadays widely used in numerous time critical applications, ranging from autopilot systems and space shuttles, to industrial process control, robots and smart automobiles. An inherent characteristic of such systems is that their requirement specification includes timing constraints in the form of task deadlines. Failure to meet the specified deadlines can lead to intolerable system degradation, and can, in some applications, result in a catastrophic loss of life or property. Due to the strictness of deadlines, hard real-time systems have to be designed in such a way as to provide an a priori guarantee that the timing constraints are met even under peak load conditions[1]. For achieving this design goal, a number of scheduling algorithms and formal feasibility analysis methods have been developed. Using these well-established traditional methods, designers can provide guarantees that tasks do always meet their timing requirements, when the system is running under a given scheduling policy, assuming

Received April 13, 2016
Corresponding author: Sandra M. Đošić
Faculty of Electronic Engineering, Aleksandra Medvedeva 14, 18000 Niš, Republic of Serbia
E-mail: sandra.djosic@elfak.ni.ac.rs

that the basic assumptions, e.g., task execution times and periodicity, are not violated at run-time [2].

Due to the catastrophic consequences of missing deadlines of some real-time tasks, fault tolerance is an essential component of such systems. While permanent faults are mostly related to the manufacturing process, transient faults mostly occur due to environmental conditions, such as high-energy particle hits originating from cosmic rays, capacitive coupling, electromagnetic interference, lighting or power fluctuations [3]. Several studies in the last two decades have indicated that transient faults are significantly more frequent than permanent faults [4]. It has been shown that transient faults are 30 times more frequent than permanent faults and that 83% of all faults were determined to be transient [5]. Transient faults have the feature that they occur and then disappear, so fault tolerance can be achieved running the task affected by a transient fault again (i.e. re-executing the task). It means that time redundancy can be used for achieving fault-tolerance by exploiting slack time in the system schedule to perform recovery executions [6], [7].

The slack time is also used by techniques for improving energy-efficiency [8]. Dynamic power-reduction techniques rely on runtime behavior to reduce power when systems are serving light workloads or are idle. This kind of techniques can be achieved in different ways, for example, *the dynamic voltage and frequency scaling* (DVFS) exploits the fact that the clock frequency of a processor is proportional to the supply voltage, while the amount of energy required for a given workload is proportional to the square of the processor's supply voltage. Thus, by lowering the supply voltage (and correspondingly the clock frequency), the power consumption of processor can be reduced quadratically [9]. Nowadays, DVFS is the most popular and widely used technique for reducing power consumption in power-sensitive applicationsand is supported by many commercial processors [8, 10, 11, 12].

DVFS techniques are often used to reduce the total energy consumption in the RTS [10, 13, 14]. Kim et al.[10] compare several key DVFS algorithms proposed for hard real-time periodic task sets, analyze their energy efficiency, and discuss the performance differences quantitatively. However, the energy saving achieved by DVFS comes at the cost of increased execution time of real-time tasks, which may compromise the timing correctness of RTS. For example, in [11] execution time of a real-time task scales linearly with the processing speed, i.e., if the operating frequency is scaled the by a factor α, then the execution timeneeds to be scaled by factor $1/\alpha$.

Fault tolerance through time redundancy and energy management through frequency and voltage scaling are particularly discussed in the context of real-time systems. Also, these two problems were considered jointly [8, 11]. Using slack time is common to these two problems. Since slack time is a limited resource, it is obvious that more slack time for DVFS technique means less time for fault tolerance, and vice versa. Therefore, there is a tradeoff between low energy consumption and high fault-tolerance.In the context of real-time systems, this tradeoff was analyzed in several papers [15, 16, 17]. Qadi et al. [15] present a DVFS algorithm supporting the canonical sporadic real-time task model. The method, however, is designed only for the earliest deadline first priority discipline and it is not applicable to rate monotonic or any other. Melham et al. proposed a technique to exploit free slacks in task schedules to reduce energy consumption while tolerating faults based on DVFS [16]. They make several simplifying assumptions such as a task is susceptible to at most one fault occurrence and the processor can scale its frequency in a continuous range. Djosic et al. developed a heuristic-based fault tolerant

dynamic voltage and frequency scaling algorithm to find frequencies at which each task should be executed such that the energy consumed by the set of task is minimized [17].

The aim of this paper is to explore the tradeoff problem and to maximize fault tolerance under power consumption constraints while guaranteeing that each task can complete successfully before its deadline. We introduce new techniques that offer trade-offs between schedulability (ability of the system to accept more tasks) and reliability (ability of the system to tolerate more faults) in real-time systems. The technique we propose is applicable to a class of RTSs that can be modeled as a set of fixed-priority preemptive periodic tasks, with different periods. Several discrete supply voltages and processor's operating frequency levels are taken into consideration.

The remainder of the paper proceeds as follows. Section 2 introduces the system model. The proposed algorithm is presented in Section 3. Section 4 presents simulation results to validate our work. Conclusions are included in the Section 5.

2. System Model and Assumptions

Processor model. We considered uniprocessor hardware platform consisting of a DVFS-enabled processor. As the processing element of the system, we assume a CPU with a finite set of m discrete voltage/frequency (v/f) levels $(v_i, f_i), i = 1, \dots, m$, where v_i is the operating voltage, and f_i is operating frequency, which can be varied at run-time. Without loss of generality, we assume $v_i < v_{i+1}$ and $f_i < f_{i+1}$. It should be noted that v/f levels are considered as ordered pairs, meaning that voltage v_i corresponds to operating frequency f_i.

Task model. The real-time task set we consider consists of n periodic preemptive tasks, $\Gamma = \{\tau_1, \dots, \tau_n\}$. Each task τ_i has a period T_i, worst case execution time (WCET) C_i, deadline D_i and fixed priority p_i. Priorities of the tasks are assigned statically by using any priority assignment algorithm [18]. We assume that $D_i \leq T_i$, for $i = 1, 2, \dots, n$. The WCET corresponds to the longest time in which the task may finish its execution. For simplicity, we assume that the WCET of a task scales linearly with the processing speed [11, 19, 20]. So, if we scale the operating frequency by a factor α, then WCET has to be scaled by factor $1/\alpha$, i.e. $C_i(f_i) = (f_m/f_j) \cdot C_i(f_m)$ where $C_i(f_j)$ is WCET of task τ_i when executed at frequency f_j, and $C_i(f_m)$ is WCET of the same task when executed at the maximum frequency f_m.

Power model. We adopt the power model proposed in [21], where the power consumption of an active processor can be modeled as $P = P_s + h(P_{ind} + P_d)$, where P_s is the static power, P_{ind} is the frequency independent active power, and P_d is the frequency dependent active power. P_s can only be removed by powering off the whole system. P_{ind} is constant and corresponds to the power that is independent of CPU processing speed and it can be efficiently removed by putting systems into a sleep state. P_d includes processor's dynamic power and any power that depends on CPU speed. When there is computation in progress, the system is active and $h = 1$. Otherwise, when the system is turned off or in a power-saving sleep mode, $h = 0$. Since the DVFS technique enables energy management by varying the supply voltage and correspondingly the operating frequency, we only take into account the frequency-dependent active power, P_d, which is calculated by $P_d(f) = V^2(f) \cdot C_{ef}/f$, where V is the supply voltage, C_{ef} is the effective switching capacitance and f is the operating frequency. More details on processor power consumption can be found in [22].

Fault model. We assume that faults can occur during execution of any task. We consider transient faults and assume that the consequences of a fault can be eliminated by simple re-execution of the affected task. We suppose that the task affected by a transient fault is executed again at its original priority level and at its original operating frequency.

Feasibility analysis. The re-execution of the corrupted task must not violate timing constraints of any task in Γ. For checking the feasibility of faulttolerant real-time task set we use the response time analysis (RTA). RTA is an important technique for analyzing timing constraints of real-time systems [23, 24, 25]. It allows exact calculation of the worst-case response time R_i of task τ_i in real-time system scheduled under fixed priorities. RTA supposes that faults can occur during execution of any task and uses re-execution recovery for fault-tolerance.The RTA is based on the following recurrent equation:

$$R_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j + \left\lceil \frac{R_i^n}{T_F} \right\rceil \max_{j \in hp(i) \cup i} (C_j) \quad (1)$$

whereR_i is represented as the three-term sum. The first term is WCET of task τ_i. The second term expresses the interferencedue to preemption of task τ_i by the higher priority tasks τ_j from the set $\{\tau_j \in \Gamma | p_j > p_i\}$. The third term characterizes tolerance due to a possible fault in the system. The fault-tolerance capability is represented in the third term by a single parameter, T_F, which corresponds to the *fault tolerant interval*, i.e., the minimum time interval between two consecutive faults that the RTS can tolerate. There can be at most $[R_i^n/T_F]$ faults during the response time R_i of task τ_i. Since these faults could occur during the execution of task τ_i or any higher priority task which has preempted τ_i, each fault may extend the response time of task τ_i for $\max_{j \in hp(i) \cup i}(C_j)$. Hence, the third addend presents an extra time needed for task recovery (task re-execution) due to faults.

It should be noted that RTA equation (1) represents a recurrence relation sinceR_i appears on both sides. The calculation starts with $R_i^0 = C_i$, and ends with $R_i^{n+1} = R_i^n$. If during the calculation process we get that $R_i^{n+1} > D$, then task τ_i is infeasible and iteration process must be terminated.

3. ALGORITHM DESCRIPTION

In order to solve the tradeoff problem between fault-tolerance and energy consumption in DVFS-enabled hard RTS, we proposed a heuristic-based algorithm. The goal of the proposed algorithm is to maximize fault tolerance under power consumption constraints while guaranteeing that each task will never miss its deadline. The algorithm is applicable to a class of RTS that fits the system model specified in Section 2.

The proposed heuristic-based algorithm can find near-optimal solution within an acceptable computational time even for large real-time task set instances. The algorithm starts with assigning the maximum frequency to each task and then iteratively improves the frequency assignment, in terms of power consumption, by gradually decreasing the frequencies of selected tasks until no further improvement can be achieved. At each iteration the task is selected in a way to keep the task set feasible with minimal deterioration of the fault tolerant interval.

The pseudo-code of the algorithm is depicted in Fig. 1.

```
1    Assign maximum v/f level to all tasks in Γ;
2    if(rta(Γ) = false) then return FAILED;
3    if(pow(Γ) ≤ Pmax) then return SUCCESS;
4    for eachτi in Γ setkeyi = true;
5    while(there are unlocked tasks andpow(Γ)>Pmax){
6    TFmin = 0;
7    for each (unlocked task τi in Γ){
8        decrement v/f level of τi;
9    if(rta(Γ) = false)
10   keyi = false;
11   else{
12         TF = fti(Γ);
13         if(TF> TFmin){
14   k = i;
15   TFmin = TF;
16         }
17       }
18       increment v/f level of τi;
19     }
20     decrement v/f level of task τk;
21   if(τk is assigned with minimum v/f level) then
22   keyk = false;
23   }
24   if(pow(Γ) ≤ Pmax) then
25     return SUCCESS;
26   else
27   return FAILED;
```

Fig. 1 The pseudo-code of the proposed algorithm

The input parameters of the algorithm are:

- $\{(v_i, f_i)|i = 1, \ldots, m\}$, set of m discrete v/f levels, where $f_i < f_{i+1}$, and $m > 1$;
- $\Gamma = \{\tau_1, \ldots, \tau_n\}$, set of n real-time tasks, where each task τ_i is characterized by parameters: worst case execution time C_i, deadline D_i, period T_i and priority p_i;
- P_{max}, threshold power consumption level.

The algorithm assigns a voltage/frequency level to each task in Γ, so that:

- power consumption of the system is less than P_{max} and,
- faulttolerant interval (T_F) is maximized.

The algorithm returns a failure status if either the task set is not feasible or the algorithm is failed to reduce the power consumption below P_{max}.

The algorithm uses the following functions:

- *rta*(Γ) - performs RTA on the task set, and returns true if the task set Γ is feasible;
- *pow*(Γ) - calculates power consumption of the task set Γ;
- *fti*(Γ) - finds faulttolerant interval (T_F) by successively applying RTA starting from $T_F = 0$.

The algorithm starts with assigning the maximum v/f level (v_m, f_m) to each real-time task in step 1. Then, the feasibility of the task set is checked by employing RTA. If it turns out that the task set is not feasible when all tasks are executed at the maximum frequency, then algorithm exits immediately with appropriate error status (step 2). Otherwise, the algorithm continues by calculating the power consumption. If the system consumes less power than P_{max}, then the algorithm exits successfully by leaving each task with the maximum voltage/frequency level, step 3. Under such condition, the algorithm does not try to change v/f levels of tasks because the power consumption constraint is already satisfied, and the fault tolerant interval cannot be improved by decreasing the execution frequency.

Otherwise, if the power consumption is greater than P_{max}, the algorithm will try to reduce power consumption below P_{max} by selectively decreasing tasks v/f levels in a way to keep task set feasible with minimal deterioration of the fault tolerant interval. Algorithm associates to each task a logical variable, referred to as *key*, which if true indicates that changing task v/f level is allowed. Initially, keys of all tasks are set to *true*, step 4. At this point, we said that all tasks are *unlocked*. The outer loop of the algorithm (steps 5 – 23) repeats until there are unlocked tasks and power consumption is greater than P_{max}. At each iteration of the loop, the v/f level of a single task is decremented. The chosen task is one for which the voltage/frequency decrement yields the minimum increase of the fault tolerant interval among all unlocked tasks, provided that task set remains feasible. To find such task, the algorithm checks each unlocked task (for loop in steps 7 - 19). First, the v/f level of the unlocked task τ_i is temporarily decremented, step 8. For instance, if task τ_i is currently assigned with the v/flevel (v_j, f_j), its new temporary v/f level will be (v_{j-1}, f_{j-1}), step 8. For such setting, feasibility of task-set is tested using equation (1), step 9. If task set is not feasible, the v/f level of task τ_i is changed back to (v_j, f_j) and its *key* is set to *false*, which fixes its voltage/frequency setting, step 10. Otherwise, the algorithm computes the faulttolerant interval, step 12, then changes v/f level of task τ_i back to (v_j, f_j), and moves on the next unlocked task. After examining all currently unlocked tasks, the algorithm selects the one for which the largest fault tolerant interval has been computed, and then permanently decrements its v/f level, step 20. Additionally, the selected task is locked if its new v/f level is dropped to (v_1, f_1), step 22. A result of lowering the v/f level is a reduction of power consumption. Therefore, the algorithm returns at the beginning of the loop, recalculates the power consumption and compares it with the threshold value P_{max}, step 5. If the power consumption is reduced below P_{max}, the goal is reached, and algorithm exit successfully, step 25. The current voltage/frequency assignment to each task is the algorithm output. Otherwise, if the power consumption is still above P_{max}, and one or more tasks are still unlocked, the

algorithm enters the next iteration. If all tasks are locked, the algorithm finishes unsuccessfully without reaching the power constraint (step 27).

4. Real-Time Task Sets and Simulation Results

This section presents simulation results that validate the performance of the algorithm for optimizing the power consumption and fault-tolerance of RTSs presented in Section 3.

4.1. Synthesized real-time task sets

In our simulation, we used a CPU with varying operating frequency range and varying number of discrete frequency levels within the range. The frequency range is normalized to the interval $(f_{min}, 1]$, where $0 < f_{min} < 1$ is the minimum operating frequency. The m available frequency levels $f_1, \dots, f_m$ are uniformly distributed in this range, with $f_1 = f_{min}$, $f_m = 1$, and $f_i, i = 2, \dots, m-1$ are determined as follows:

$$f_i = f_{min} + (i-1)\frac{1-f_{min}}{m-1} \tag{2}$$

Real-time task sets with different number of tasks$n$ and processor utilization U are generated randomly using *UUniFast* algorithm [26]. The utilization U of a task set is defined as the sum of utilizations of all the tasks belong to the set, that is:

$$U = \sum_{i=1}^{n} u_i = \sum_{i=1}^{n} \frac{C_i}{T_i} \tag{3}$$

During task set generation, the periodT_i of each task τ_i is chosen randomly from the range [1, 1000] with uniform distribution, and the task deadline is set to be equal to the task period,$T_i = D_i$. Tasks are assigned with priorities according to the earliest deadline first algorithm. Finally, the worst case execution time (on the maximum operating frequency f_m) of task τ_i is determined as $C_i = T_i \cdot u_i$.

4.2. Simulation results

We conducted simulation for 100 tasks set, each composed of 10 tasks. Since the main purpose of the proposed algorithm is tomaximize the fault-tolerance under power consumption constraints, the simulation results are presented from both the fault tolerance and energy consumption point of view. The power consumption performance is evaluated in terms of *normalized power reduction*(NPR), which is defined as the ratio of power consumption when tasks are executed at maximum frequency and the power consumption when tasks are executed on frequencies determined by the proposed algorithm, in percentage.We also define the *fault tolerance factor* (FTF) as the ratio between the maximum and achievedfault-tolerance capability. The maximum fault-tolerance capability is specified through fault tolerant interval when all tasks are executed at the maximum frequency. The achievedfault-tolerance capability is also specified through fault tolerant interval, butafter the proposed algorithm has been applied. It means thatthe achieved fault-tolerance capability is calculated for the given power consumption constraint.

Fig. 2 depicts the fault-tolerance factor versus normalized power reduction for different number of available frequency levels.

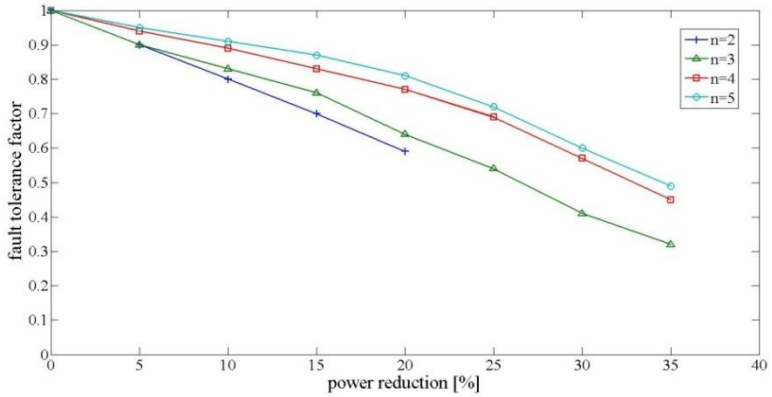

Fig. 2 Fault tolerance factor versus power reduction for different number of frequency levels

We used a subset of n, $n = 2, \dots, 5$ frequency levels according to the equation (2). As expected, the higher power savings, the lower fault tolerance factor, and vice versa. Also, these results show that the fault tolerance factor increases with the number of availablefrequency levels. For example, if NPR is equal to 15%, then the fault-tolerance factor reaches $FTF = 0.9$ when all available frequencies are utilized. The fault-tolerance factordrops to $FTF = 0.7$ when only two frequency levels are used.

Fig. 3 depicts the fault-tolerance factor versus normalized power reduction for different processor utilization U, which varies from $U = 0.1$ to $U = 0.5$.

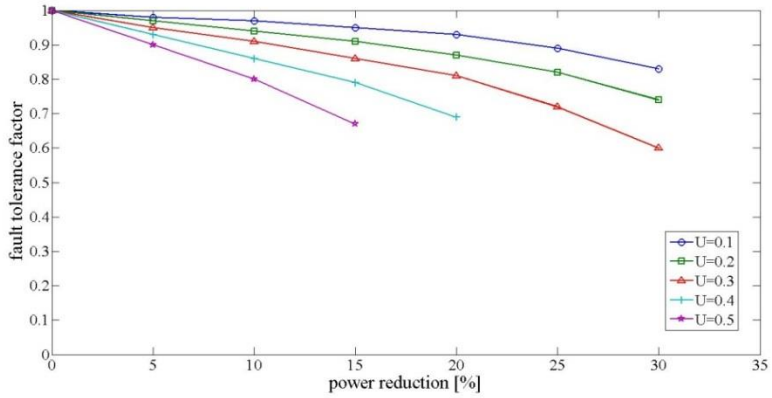

Fig. 3 Fault tolerance factor versus power reduction for different processor utilization

The fault-tolerance factor decreases with increasing power reduction, and vice versa. Also, the fault-tolerance factor increases with increasing processor utilization. For example, if NPR=15% fault tolerance factor varies from $FTF = 0.65$ to $FTF = 0.95$, when the processor utilization is $U = 0.5$ and $U = 0.1$, respectively.

5. CONCLUSION

High fault-tolerance against transient faults and low power consumption are key objectives in the design of real-time systems. Several effective energy saving techniques and mature fault-tolerance techniques are available to achieve these objectives. However, careful considerations should be taken in order to achieve both objectives simultaneously, since the usage of the fault-tolerance techniques increases the energy consumption and vice versa. Recognizing that the problem of assigning frequencies to real-time tasks for simultaneous optimization of fault-tolerance and power consumption is NP-hard, we realized and presented in the paper a heuristic-based algorithm to maximize the fault-tolerance under power consumption constraint.The algorithm can be used in the early stages of RTS design process as a tool for rapidly exploring the tradeoff between the power consumption and system's ability to tolerate transient faults.

REFERENCES

[1] J. W. S. Liu, *Real-Time Systems*, Prentice-Hall, NJ, 2000.

[2] P. Brucker,*Scheduling Algorithms*, Springer Science & Business Media, 2004.

[3] R. Johnson, S. Diehl-Nagle and J. Hauser, "Simulation approach for modeling single event upsets on advanced CMOS SRAMS, " IEEE Transactions on Nuclear Science, vol. 32, pp. 4122-4127, 1985.

[4] S. Ghosh, R. Melhem, D. Mosse, "Fault-Tolerant Rate-Monotonic Scheduling," Journal of Real-Time Systems, vol. 15, pp. 149-181, 1998.

[5] J. Srinivasan, S. V. Adve, P. Bose, J. A. Rivers, "The Impact of Technology Scaling on Lifetime Reliability," *Proceedings of the International Conference on Dependable Systems and Networks*, pp. 177–186, 2004.

[6] S. Đošić, M. Jevtić,"Analysis of transient fault tolerance in hard real-time systems with time redundancy," Facta Universitatis, Series: Automatic control and robotics, vol. 8, pp. 149-163, 2009.

[7] S. Đošić, M. Jevtić, M. Damnjanović,"Analysis ofpossibilitiesto overcome thetransientfaultsin real-time systems with time redundancy,"*Proceedings of XLVI International scientific conference on information, communication and energy systems and technologies ICEST 2011*, pp. 417-420, 2011.

[8] R. M. Santos, J. Santos, J. D. Orozco, "Power saving and fault-tolerance in real-time critical embedded system," Journal of system Architecture, vol. 55, pp. 90-101, 2009.

[9] T. D. Burd, T. A. Pering, A. J. Stratakos, R. W. Brodersen, "A Dynamic Voltage Scaled Microprocessor System," IEEE J. Solid-State Circuits, vol. 35, pp. 1571-1580, 2000.

[10] K. Woonseok, S. Dongkun, Y. Han-Saem, K. Jihong, M. Sang, "Performance Comparison of Dynamic Voltage Scaling Algorithms for Hard Real-Time Systems,"*Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium RTAS'02*, pp. 219 – 228, 2002.

[11] A. S. Ahmadian, M. Hosseingholi, A. Ejlali, "A Control-Theoretic Energy Management for Fault-Tolerant Hard Real-Time Systems," IEEE International Conference on Computer Design, pp. 173-178, 2010.

[12] Intel, Intel PXA270 Processor Electrical, Mechanical and Thermal Specification Data sheet, 2005. [Online]. Available: http://www.phytec.com/pdf/datasheets/PXA270_DS.pdf

[13] E. L. Sueur, G. Heiser, "Dynamic voltage and frequency scaling: The laws of diminishing returns," *Proceedings of the 2010 Workshop on Power Aware Computing and Systems HotPower'10*, pp. 1-5, 2010.

[14] F. Gruian, "Hard real-time scheduling for low-energy using stochastic data and dvs processors," *Proceedings of the 2001 International Symposium on Low Power Electronics and Design ISPLED*, pp. 46-51, 2001.

[15] A. Qadi, S. Goddard, "A dynamic voltage scaling algorithm for sporadic tasks," *Proceedings of the 24th International Real-Time Systems Symposium*, pp. 52–62, 2003.

[16] R. Melhem, D. Mosse, E. Elnozahy, "The interplay of power management and fault recovery in real-time systems," IEEE Transactions on Computers, vol. 53, pp. 217-231, 2004.

[17] S. Djosic, M. Jevtic,"Dynamic Voltage and Frequency Scaling Algorithm for Fault-Tolerant Real-Time Systems,"Microelectronics Reliability,Elsevier Ltd., vol.53, pp. 1036-1042, 2013.

[18] F. Cottet, J. Delacroix, Z. Mammeri, *Scheduling in Real-Time Systems*, John Wiley & Sons, 2002.

[19] W. Chedid, C. Yu, *Survey on Power Management Techniques for Energy Efficient Computer Systems*, Laboratory Rep., Mobile Computing Research Lab., Cleveland State University, Cleveland, OH, 2002.

[20] F. Xia, Y.C. Tian, Y. Sun, J. Dong,"Control-theoretic dynamic voltage scaling for embedded controllers," IET Computers & Digital Techniques, vol. 2, pp. 377 – 385, 2008.
[21] D. Zhu, R. Melhem, D. Mosse, "The Effects of Energy Management on Reliability in Real-Time Embedded Systems," *Proceedings of the International Conference on Computer Aided Design ICCAD*, pp. 35-40, 2004.
[22] [Online]. Available: http://systems.ihp-microelectronics.com/uploads/downloads/Diss_Panic.pd
[23] A. Burns, R.I. Davis, S. Punnekkat, "Feasibility Analysis of Fault-Tolerant Real-Time Task Sets," *Proceedings of the Euromicro Real-Time Systems Workshop*, pp. 29-33, 1996.
[24] G. Lima, A. Burns, "An Optimal Fixed-Priority Assignment Algorithm for Supporting Fault-Tolerant Hard Real-Time Systems", IEEE Transaction on Computers, vol. 52, pp. 1332-1346, 2003.
[25] R. Davis, A. Zabos, A. Burns, "Efficient Exact Schedulability Tests for Fixed Priority Real-time Systems," IEEE Transactions on Computers, vol. 57, pp. 1261 – 1276, 2008.
[26] E. Bini,G. C. Buttazzo,"Measuring the Performance of Schedulability Tests", Real-Time Systems, vol. 30, pp. 129–154, 2005.