Evaluating and Exploiting Impacts of Dynamic Power Management Schemes on System Reliability

Liangzhen Lai
Electrical Engineering Department, UCLA
liangzhen@ucla.edu

Vikas Chandra
ARM Research
vikas.chandra@arm.com

Puneet Gupta
Electrical Engineering Department, UCLA
puneet@ee.ucla.edu

ABSTRACT

Hardware reliability has been a major concern for nano-scale computing systems. Different hardware design choices, application workloads and software management schemes can jointly affect the system's resilience. In this paper, we first develop a hardware evaluation platform based on an embedded/mobile development board and standard Linux kernel. We demonstrate the use of our platform to evaluate the system's power and radiation-induced soft error rate in presence of system power management schemes and with different application workloads and various hardware design configurations. We also propose system/cloud-based virtual sensing to capture varying ambient conditions for reliability evaluation. New reliability management policies are proposed and implemented in Linux kernel to exploit the flexibility in different existing power management schemes. We demonstrate that our policies can achieve the system reliability target under varying application workloads and ambient conditions. Experiments show that our policies are efficient and with less than 3% additional power overhead compared to the optimal schemes characterized offline.

Categories and Subject Descriptors

B.8.0 [**Hardware**]: PERFORMANCE AND RELIABILITY—*General*

Keywords

Power Management, Soft Error, Operating System, Linux Kernel

1. INTRODUCTION

Hardware reliability has been a major concern for nano-scale computing systems. Different design-time hardware implementation choices and run-time software management schemes, though may not be specifically designed for reliability purposes, can significantly affect the system's resilience. Therefore, it is necessary to analyze their effects under different application workloads and even exploit them to retain system resilience.

There can be two possible ways to implement system power management schemes to use system performance slack. Run-fast-then-stop (RFTS) completes the workload with nominal performance and then goes to certain sleep states for the remaining slack to reduce power. Just-in-time (JIT) tries to adjust the peak performance to elongate the runtime with lower power consumption. Both of them can be used to achieve power saving but can have different impacts on reliability.

There are two main questions to be answered:

- How are reliability-related phenomena affected by different power management mechanisms and hardware implementation choices?
- Can the system exploit different power management mechanisms to achieve its reliability target under varying ambient conditions and runtime workload?

In this work, we try to address the two issues in the context of radiation-induced soft error, as the hardware design choices and system power management schemes can jointly affect soft error rate (SER). For example, power gating, as one of RFTS schemes, can be implemented with or without state retention flip-flops (FFs) [16]. Power gating without retention FFs will dump its state into memory, which helps eliminate logic SER during gating. However, power gating with retention FFs will increase SER as FFs in retention mode are more vulnerable to soft errors.

Radiation-induced soft error also poses challenges of how to estimate SER, especially for mobile devices. It is difficult to monitor by hardware itself due to the unique ambient (i.e., location/altitude) dependence [14] and rare occurrence nature. The same type of device can be used by users living at different geographic locations. Even the same mobile device can be used at different locations over time. We observe that most mobile devices are typically equipped with various hardware sensors (barometers, GPS etc.) and Internet connection capabilities. Therefore, system-level or even cloud-based SER monitors can be implemented to capture the device location information and to assist the reliability management.

The key contributions of our work are the following:

- We implement a hardware evaluation platform based on BeagleBone Black development board and standard Linux kernel. We demonstrate the use of our platform for studying the system power and SER under different hardware design choices, application workloads and software management schemes.
- We propose a system/cloud-based virtual sensing approach to capture the varying location/altitude for SER estimate. This enables the reliability adaptation and management.
- We propose new reliability management policies to exploit the flexibility in existing Linux power management schemes. We show that our forbidden-state based

approaches are effective in achieving system reliability target with minimal power overhead compared to the optimal schemes characterized offline.

Previous work addressed either the hardware reliability modeling challenges [3, 11] or proposed new sophisticated power management algorithms [21, 25, 31]. They focused on a specific hardware model and did not consider the power and reliability impacts of wide varieties of hardware support for power management. Our evaluation platform focuses on studying the interactions between different hardware design choices and representative power management schemes in generic system software (i.e., Linux kernel). Our evaluation platform will be using the reliability models in some of the existing work. Our reliability management policies are proposed to exploit flexibility in different existing power management schemes rather than proposing new power management algorithms. Previous work [15, 18, 25] addressed the reliability management problem in the context of other reliability-related issues, including Negative-/Positive-biased Temperature Instability (N/PBTI), Electromigration (EM), etc. Our framework can be extended to consider these reliability issues as well.

The rest of the paper is organized as follows: Section 2 introduces some background information about radiation-induced soft error, hardware support for power management and power management schemes in Linux kernel. Section 3 describes the development of our evaluation platform and some results of using the platform to assess the system power and SER. Section 4 presents our approaches and results of power state management policies for retaining system reliability targets. Section 5 concludes the paper.

2. BACKGROUND

2.1 Radiation-Induced Soft Error

Radiation-induced soft error is caused by alpha particles or neutrons hitting silicon and flipping circuit logic states. The actual failure rate or failure-in-time (FIT) rate depends on the technology [17, 26], circuit structure (e.g., logic gate structure, size) [11], operating points (e.g., supply voltage) [7] and ambient conditions (e.g., altitude) [14].

A model for calculating altitude-dependent SER is proposed in [14] as:

$$SER = \int \sigma(E) \cdot (\frac{d\phi}{dE})dE \quad (1)$$

where $\sigma(E)$ is the bit fail cross section for particle energy E, $d\phi/dE$ is the fluence rate per unit energy, or differential flux. The main altitude dependence of $d\phi/dE$ is exponential attenuation based on the atmospheric depth d. This dependence can be represented as a factor $F_{alt}(d)$:

$$F_{alt}(d) = exp(\frac{d_{SL} - d}{L_n}) \quad (2)$$

where d_{SL} is the atmospheric depth at sea level, and L_n is the effective attenuation length. This altitude-dependence model is reported with less than 5% error compared to measured data across different geographic locations [14]. In this work, we will use pre-characterzied FIT rate data as baseline and apply the altitude dependence in Equation 2.

Various techniques have been proposed to detect and correct soft errors [19, 23, 24, 28]. Different architecture components can have different vulnerability to soft error [20], and the error propagation can become very complicated [8]. It is difficult to combine the SER of different components into one system failure rate. In this work, we assume that the system has two separate SER targets for the processor core and memory, since the factors we studied, such as supply voltage and location, affect the SER of either or both components. For core SER, only FF soft errors are considered, as combinational soft errors contribute a relatively small fraction [11]. The FFs include both architectural visible and non-visible (i.e., pipeline FFs, control FFs etc.) ones. For memory SER, we consider only soft errors in SRAM without ECC, as SRAM can be well-protected with ECC and physical interleaving.

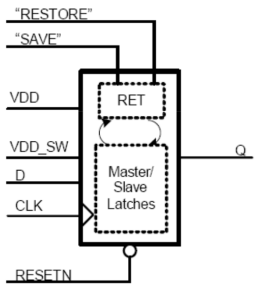


Figure 1: An example of state retention Flip-Flop [16]. The retention part is powered by a different supply rail than the main master/slave latches.

2.2 Hardware Support for Power Management

Most power management mechanisms are supported by some special hardware design techniques or configurations. Different design choices can result in different management efficiency (e.g., power saving and switching latency) as well as hardware reliability.

One example is the design of on-chip power delivery network in support for voltage scaling. In a typical design, memory other than L1 cache is designed to operate at a constant voltage and thus on a separate memory power rail. There are two possible ways to connect the power supply of L1 cache. One common way is to connect it to the same power rail as the processor core. This design can avoid the voltage level converter between the processor core and L1, but will also limit the lowest supply voltage which is usually determined by the SRAM V_{min}. The other way is to connect L1 to the separate power rail. This allows more aggressive voltage scaling but requires level converter and results in longer delay and higher design overhead.

Another example is the implementation of state retention for power gating. As mentioned earlier, the state retention can be realized by either retention FFs or dumping architecture states into memory. An example of retention FF is plotted in Fig. 1 [16]. The retention part is usually implemented by smaller transistors and powered by lower supply voltage than the master/slave latches on the main path. This makes the FF under retention mode more vulnerable to soft errors.

There can be a lot of different consideration for making these hardware design choices. The purpose of our work is studying the interactions between these design choices and system-level power management schemes rather than comparing them from reliability or power perspective. Therefore, we will make our implementation flexible for targeting these different hardware configurations.

2.3 Power Management in Linux Kernel

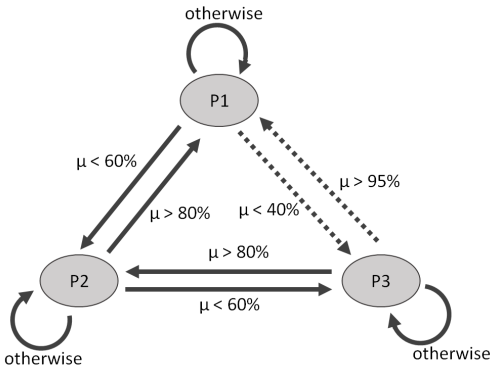


Figure 2: State transition diagram example based on CPU utilization μ for CPUFreq governors. Ondemand governor can switch between any two levels while conservative governor can only switch between neighboring levels. The switching threshold values are configurable in Linux kernel, and we use the default values for ondemand governor.

Different power management schemes can have substantially different impacts on SER. Previous work has proposed sophisticated scheduling and voltage scaling algorithms in the context of soft error for real-time systems [12, 21, 22, 27, 31]. However, most existing work does not consider the interactions between software management schemes and hardware implementation choices.

For evaluating hardware design choices, it is important to correctly account for the generic software system and workload behavior. In this work, we build our platform based on Linux kernel of version 3.12. The two power management schemes, JIT and RFTS, are implemented in Linux kernel as CPUFreq and CPUIdle modules respectively.

CPUFreq [9] (see left top of Fig. 4) is the module inside Linux kernel for CPU frequency scaling. Depending on the hardware driver and implementation, the supply voltage may or may not scale with the frequency, as voltage scaling does not need to be visible to the software. The CPUFreq policy is a data structure used to record information, such as current governor, operating frequency etc. The module is scheduled to run at a fixed period. At the beginning of each period, CPUFreq governor retrieves system utilization information from the kernel. Based on CPUFreq policy data and the frequency table, the governor will set a frequency target or range. An example state diagram of CPUFreq governor is shown in Fig. 2. The utilization threshold for state switching is configurable in Linux kernel. In this work, we use the default values of ondemand governor. Driver, which is the actual code talking to the hardware, will try to scale the frequency/voltage (i.e., P-states) to match the target specified by the governor. [1] The actual change done by the driver will be updated to CPUFreq policy and broadcasted through a system notifier.

CPUIdle [10] (see Fig. 3) is the module to support multiple CPU idle levels inside Linux kernel. Different available idle states (i.e., C-states) are pre-characterized with their corresponding power consumption and exit latency. The idle state selection is based on target residency, which is the minimum idle time to achieve a net energy saving compared to

[1]Latest version of the kernel deprecates this and requires the driver to set to the exact target frequency as specified.

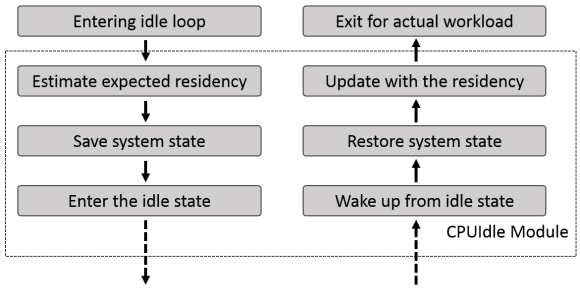


Figure 3: CPUIdle module flow chart

a lighter idle state. At runtime, CPUIdle module is called when the system completes the tasks and enters the idle loop. The module generates an estimate of the expected residency (i.e., sleep time) based on next scheduled event and other history information like past interrupts. The governor will select the idle state whose target residency is the largest but is still not larger than the expected residency. Corresponding driver function is used to prepare and enter the selected idle state. Upon wake up, the module will recover the system state and record the actual residency time.

3. RELIABILITY EVALUATION OF POWER MANAGEMENT SCHEMES

An evaluation platform is essential for studying system reliability under system-level power management with various hardware configurations and software applications. Some key questions can be answered by experiments using such an evaluation platform, e.g., how will software management schemes in conjunction with hardware design choices affect system reliability, what is the worst-case workload for system reliability. In this section, we describe the implementation of the evaluation platform. The base platform is introduced in Section 3.1. The targeting hardware configurations and corresponding power and SER models are explained in Section 3.2. The platform customization is described in Section 3.3. Some experimental results on system reliability evaluation are presented in Section 3.4.

3.1 Base Platform for Reliability Evaluation

The hardware evaluation platform has to be general enough for targeting different hardware configurations as well as fast enough for running system software. Therefore, implementing actual hardware for each of the configuration is impractical and simulation-based approaches are not feasible. Emulation-based tools [29] can meet both the speed and flexibility requirements, but lack accurate time accounting for frequency scaling and entering/leaving idle states.

In this work, we build the platform based on BeagleBone Black [6] board, on which the kernel and software are running. Power, reliability and different power management latencies are emulated online with a customized kernel module or by inserting dummy busy loops. The power and reliability models are pre-characterized and treated as input for the kernel module.

BeagleBone Black board is based on AM335x ARM Cortex-A8 processor [4] with 45nm process technology. The processor has separate L1 32KB data cache and 32KB instruction cache. L2 cache is of 256KB. The software stack is based on TI SDK7 [5]. Linux kernel is of version 3.12 with support for both CPUIdle and CPUFreq modules. Software benchmarks are used to represent different workloads, including

Table 1: Table of different hardware implementations

	Retention FF	L1 Rail	L1 ECC
Case I	No	Same as core	No
Case II	No	Separate	No
Case III	Yes	Same as core	No
Case IV	No	Same as core	Yes

web browsing, MPEG 4 decoding, FFPlay [13] and 3D image rendering. We also implement a synthetic benchmark in order to study the system behaviors under different workload intensity.

3.2 Hardware Configuration and Modeling

As discussed in Section 2.2, different hardware configurations can have different impacts on reliability. The hardware evaluation platform will be used to emulate systems with these configurations. The design configurations to be explored include the use of retention FFs for power gating, whether L1 cache is under the same power rail as the processor core, and whether L1 cache is protected with ECC. The hardware design configuration cases studied in this work are listed in Table 1.

Power and SER models for these configurations are essential for our evaluation platform. In this work, we derive the power and SER models based on a 45nm technology. We also use projected power and SER model for 28nm technology to study the impacts of technology scaling in Section 3.4.

BeagleBone Black is based on Cortex-A8 processor. Since we do not have the detailed design information, building the power and delay models at different supply voltages and idle states for the exact processor is not possible. In this work, the power and delay model is derived based on the synthesis, placement and routing results of a similar processor using commercial tools [1] with commercial 45nm process technology and libraries. The power and delay models should reflect the voltage-dependence of this process technology and libraries for mobile-class processors. As will be discussed later, the SER model is based on the FIT rate of individual elements (i.e., FFs and SRAM cells), which is independent of the processor model after normalization.

The power values are calculated using the same commercial tools [1] with the libraries. To model the power difference under voltage scaling, we also re-characterize the libraries at the supply voltages for all P-states in Table 2. The libraries include the retention FF cells, which are used to model the power when retention FFs are used. The power saving for clock gating (i.e., C2 state in Table 3) is derived by stripping off clock power. The memory power model is based on the memory compiler results for the same process technology. The memory compiler is configurable with options to include memory ECC and/or retention modes. The power difference is taken into account with respect to the corresponding targeting design configuration cases in Table 1. The power model projection for 28nm is done by repeat the same process with commercial 28nm technology and libraries, including memory compiler results.

As mentioned earlier in Section 2.1, it is difficult to model or compute the system failure rate based on SER of each individual component. So the processor core SER and memory SER are considered separately. In fact, even for individual component like processor core, the failure rate is extremely difficult to model due to masking effects at various levels,

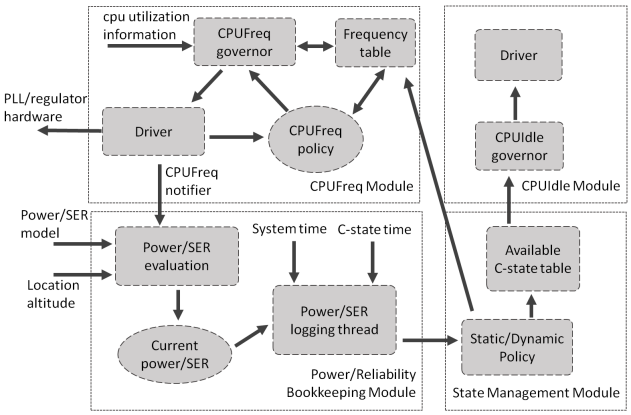


Figure 4: Overall block diagram

different visibility and vulnerability. However, the factors we are considering in this work, i.e., voltage and altitude, have similar effects on FFs or SRAM cells. Moreover, none of these factors will change the error propagation or behaviors other than increasing or decreasing the probability of getting bit-flips. In this work, we model SER of processor core or memory by the FIT rates of FFs or SRAM cells.

Sea-level processor core SER values are based on FIT rates in [11] for 45nm technology and only sequential elements (i.e., FFs) are considered. The core soft error rate is calculated based on the total number of FFs in the synthesized processor netlist. The SRAM SER values are also derived based on the results in [3, 11] and voltage dependence in [7]. The 28nm SER values are scaled based on the model in [7]. Altitude dependence of SER is modeled by Equation 1 based on the work in [14]. The parameters in Equation 1 are derived based on the design manual of the same 45nm technology.

3.3 Platform System Customization

The platform supports both CPUFreq and CPUIdle modules with five frequency levels and two idle states, Wait-For-Interrupt (WFI) and clock gating. To be able to evaluate more variety of power management schemes, two more idle states, including core power gating with and without memory retention mode, are added. Extra latency are emulated by inserting additional dummy busy loops. In this work, we use the ondemand CPUFreq governor.

A customized power/reliability bookkeeping module is developed for taking the system trace and keeping track of the system power and SER at runtime. The module structure is illustrated on the left bottom of Fig. 4. The module reads in the pre-characterized power and nominal SER model values for the processor core and cache. Based on these values and device location information, the module updates the system's current power consumption and SER for normal running and each C-state. Any frequency change notifier or location update will trigger the module and recalculate the power/SER values. A logging thread is running at a fixed period to calculate and record the accumulated power and SER, based on the corresponding time spent in normal running and each C-state.

The information for supported P-states and C-states are summarized in Table 2 and Table 3. The frequency switching delay shown in the driver is 0.3ms, which makes the CPUFreq module run every 300ms. But our actual measurement shows much longer latency for frequency level switching, which is mainly caused by the low bandwidth I2C com-

Table 2: Table of different CPUFreq states spec used

Frequency	Voltage	Latency
1000 MHz	0.90V	2.1ms
800 MHz	0.82V	2.1ms
720 MHz	0.80V	2.2ms
600 MHz	0.76V	2.2ms
300 MHz	0.72V	3.2ms

Table 3: Table of different CPUIdle states spec used

State	Description	Exit Latency	Target Residency
C1	WFI	68us	150us
C2	clock gating	130us	200us
C3	core power gating	530us/1060us[2]	800us/ 1450us
C4	core power gating with SRAM retention	650us/1180us[2]	1000us/ 1550us

munication between the processor and the off-chip regulator. Since only frequency change need to be visible to software stack, we also implement a customized version of the driver that bypasses the voltage changes and only does the frequency scaling. This gives our platform the capabilities to emulate systems with fast frequency switching latencies.

3.4 Results of Reliability Evaluation

The experiments are done with the hardware evaluation platform based on BeagleBone Black board (see Fig. 5). The device is assumed to be operated at sea level. The reliability bookkeeping thread is configured to generate a log entry every 100ms to limit the interference and have enough resolution for tracking the states.

The results of different hardware configuration cases with our synthetic workload are plotted in Fig. 6. The synthetic workload is implemented as a busy-loop that is scheduled every 20ms. The busy-loop is configurable with a tunable number of iterations to load the system with different workload intensity. 100% workload intensity means the number of iterations in the busy-loop is configured to fully load the system with no performance slack. 40% workload intensity means the number of iterations in the busy-loop is 40% of that for the 100% workload intensity case. For the ease of cross-comparison, all results in this section are normalized with respect to the maximum power/SER values from Case I study, which is about or equivalent to 0.26W power, 200 FF FIT and 500 SRAM FIT for per mega cells.

In all cases, the power increases with increased workload intensity. This is expected as higher workload intensity imposes higher performance demand on the system, which results in higher voltage state and less idle time. For systems without L1 memory ECC, i.e., case I, II and III, memory SER decreases with increased workload intensity, due to less time spent in retention mode and lowered supply voltage levels.

An interesting observation is that for systems without retention FFs, i.e., case I, II and IV, core SER is non-monotonic with respect to the workload intensity. Peak core SER occurs at medium (around 40% - 50%) workload intensity, where system is operating at low voltage level and with little slack for going into deep idle states. This suggests the potential of preferring an RFTS power management approach to reduce SER. This motivates the reliability management policies proposed in Section 4. For system with retention FF, i.e., case III, core SER is significantly higher than other cases and peaks at low workload intensity regions, as the FFs in retention mode are more vulnerable to soft errors. A potential software solution to this can be implementing a virtual idle state with all or part of CPU states dumped to RAM even in presence of retention FFs. This idle state can be used for low workload intensity scenarios, where sleep/wake-up latency requirements may be relaxed but core SER is the highest.

To study the impact of technology, the experiments with synthetic workload is repeated with the projected 28nm power and SER models. The results are plotted in Fig. 7. The trend of power and SER changes is very similar to the results in Fig. 6. Similar to the results reported in [26], we observe reduced SER of sequential elements at smaller technology. This reduced SER is primarily due to the smaller FIT rate for each individual sequential element, as a result of shrinked diffusion area and increased driving strength per diffusion area. In reality, this result may be different considering that the design at smaller technology is likely to have more logic gates and larger memory capacity.

[2]Exit latency and target residency for system with/without retention FFs

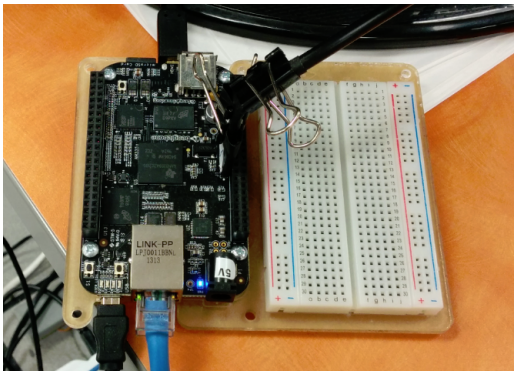


Figure 5: Photo of the hardware evaluation platform setup based on BeagleBone Black development board.

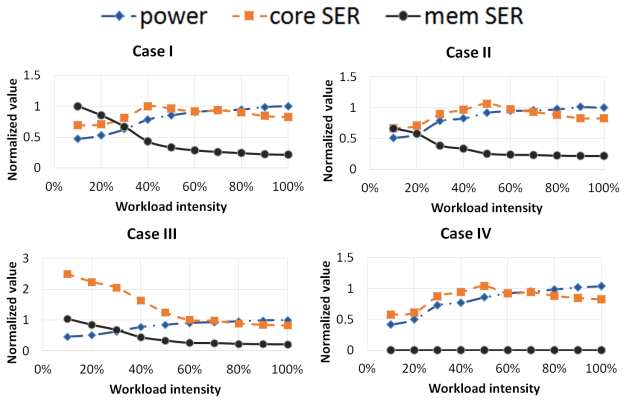


Figure 6: Power/SER Results of different hardware configuration cases with different software workload intensity at sea level. 100% workload intensity means the system is fully loaded and has no performance slack.

Table 4: Normalized power/core SER/memory SER results of different software benchmarks with different configuration cases at sea level

	Web browsing	MPEG 4 decoding	FFPlay	3D image rendering
Case I	0.35/0.57/1.05	0.58/0.83/0.77	0.36/0.80/0.51	0.54/0.85/1.01
Case II	0.39/0.59/0.65	0.57/0.77/0.45	0.37/0.80/0.39	0.58/0.85/0.62
Case III	0.34/2.30/1.04	0.61/1.93/0.70	0.35/1.57/0.52	0.52/2.42/1.01
Case IV	0.34/0.55/0	0.55/0.86/0	0.33/0.76/0	0.52/0.83/0

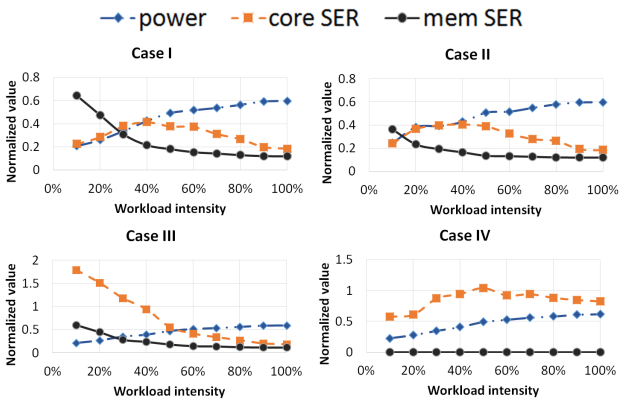


Figure 7: Power/SER Results of different hardware configuration cases with different software workload intensity for 28nm technology.

We also demonstrate the use of our platform for evaluating power/SER with real life benchmarks. Four software benchmarks are used, including web browsing, MPEG 4 decoding, FFPlay and 3D image rendering. We launch one benchmark at a time on the platform and record the average power and SER over a fixed period. The period starts 3 seconds before the beginning of the benchmark and lasts long enough for the benchmark to finish. This matches the running environments of typical embedded/mobile devices and includes the warm-up and cool-down period of the system, which is important for modeling the system-level power management schemes.

The power and SER results of running different software benchmarks are highlighted in Table 4. The numbers are in line with the results in Fig. 6. MPEG 4 decoding is the most demanding workload, therefore have the highest power consumption and lowest memory SER. The core SER is highest for 3D image rendering, which has slightly smaller workload intensity than MPEG 4 decoding. Comparing the results of web browsing and FFPlay, they have very similar power results. However, both core SER and memory SER are very different for these two benchmarks. Average core SER is much higher for FFPlay, and memory SER is much higher for web browsing. This is because the two benchmark have very different workload patterns. Workload for FFPlay is more steady, while web browsing behaves more like bursts of workload. As the results of these workload patterns, the power management behaves more like RFTS for web browsing and JIT for FFPlay. This further motivate our proposed reliability management policies, which will be described in Section 4.

4. EXPLOITING POWER MANAGEMENT SCHEMES FOR RELIABILITY

Since different power management schemes can all achieve power saving but have different impacts on system reliability, they can potentially be used to retain the system reliability target if ambient condition (e.g., location) changes. As prerequisite of such adaptation, a system/cloud-based virtual SER sensing is proposed in Section 4.1.

As mentioned earlier in Section 3.2, we consider the processor core SER and memory SER separately. In this work, we assume the system has specific SER target for the processor core and memory in order to retain its mean-time-between-failure (MTBF). For a real system, error injection studies can be performed to derive the relationship between soft errors (i.e., bit flips) and system failures. Based on the desired system MTBF, the SER target can be calculated.

The SER target can be a static constraint, i.e., specification sheet target, which means that the instantaneous SER at any time should be kept smaller than it. We propose a forbidden-state based policy for such static constraints in Section 4.2. Since typical soft error MTBF can be of days to months, restricting the instantaneous SER may be too pessimistic and unnecessary. We also consider the case when the constraint is dynamic, i.e., the system's overall SER within certain period should be kept smaller than the SER target. This period should be much shorter than the soft error MTBF (in days to months) but still significant longer than the power management operation time (in milliseconds). We propose a dynamic state enabling/disabling policy for such dynamic constraints in Section 4.3. Experiment results are described in Section 4.4. The effectiveness of our dynamic policy is evaluated in Section 4.5.

4.1 SER Estimation Mechanism

Soft error itself can result in different software/hardware symptoms and be detected by corresponding mechanism [28]. SER, however, is extremely difficult to measure or estimate due to soft error's rare occurrence nature and unique ambient dependence. Direct measurement of the error occurrence such as measuring memory ECC [30] errors or checkpoint recovery events may be possible. But it will require huge memory size or extremely long measurement time, as the soft error rate are typically less than one per mega devices per month.

Our proposed mechanism is motivated by the fact that most modern mobile devices are equipped with various types of sensors (e.g., GPS, barometer) and network connection capabilities. A system-level virtual altitude/location sensor can be implemented based on these sensors. SER estimation of current location can be made with altitude-dependence model [14] or a locally-stored look-up table. A cloud-based service can also be used to answer incoming queries of the measured flux rate data, solar activities and SER estimate from mobile devices with given location information.

In this work, we assume the system is equipped with the virtual altitude sensor. Whenever the virtual altitude sensor updates the altitude value, our reliability bookkeeping mod-

ule (see Fig. 4) will update the SER estimate for both core SER and memory SER based on the altitude-dependence model. The SER estimate is based on a pre-characterized reference SER value at sea level and scales with an exponential function of altitude dependence from Equation 2. The overhead of SER calculation is negligible. Alternatively, a local look-up table can be used to store the pre-characterized SER values at different altitudes.

4.2 Forbidden-State Based Policy

Different approaches can be used to affect the system power management choices and account for ambient condition changes. A good approach should be both effective and non-intrusive so that the impacts on the system behavior and performance is minimal and mostly predictable. Based on these considerations, we decide to preserve existing Linux power management modules and perform the reliability management by manipulating the available power states. In this way, our reliability management policy is isolated and independent from the power management schemes. This also improves the compatibility of our approaches with arbitrary power management governors.

Considering the structure of kernel power management schemes (see Fig. 4), we propose a forbidden-state based policy for enabling/disabling certain P-states and C-states, depending on system's current SER. The policy is implemented as a stand-alone module (see Fig. 4). For P-states, the module will change the frequency table directly so that the driver that decides the final frequency level will select only the available states. For C-states, a C-state table is maintained for CPUIdle governor to enter only the available idle states.

A static policy can be used for the case of static SER target. Whenever the reliability bookkeeping module updates the system SER, the state management module compares the SER values to the target SER and disables all states whose SER values are higher. This guarantees that the instantaneous SER at any possible power state can meet the reliability target.

4.3 Dynamic State Enabling/Disabling Policy

For dynamic SER target, the static policy may be too conservative and unnecessary. Therefore, we propose a policy which limits the average SER, SER_{avg}, within a given period of T through dynamically enabling/disabling the available P-states and C-states.

The policy is built based on our reliability bookkeeping module described in Section 3.3. The module keeps track of the accumulated SER at any time t as $SER_{acc}(t)$. So at the beginning of each period t_{start}, the accumulated SER target for the end of this period, SER_{end}, can calculated as:

$$SER_{end} = SER_{acc}(t_{start}) + SER_{avg} \cdot T \qquad (3)$$

At any time t, the remaining SER budget, SER_{budget}, until the end of this period t_{end} can be calculated as:

$$SER_{budget} = \frac{SER_{end} - SER_{acc}(t)}{t_{end} - t} \qquad (4)$$

The policy will use SER_{budget} instead of the actual SER target to determine which states should be enabled or disabled.

The overall structure of the state management module is kept the same with this dynamic policy. Only an update request mechanism is added to calculate SER_{budget} on demand. Whenever the CPUFreq or CPUIdle module access

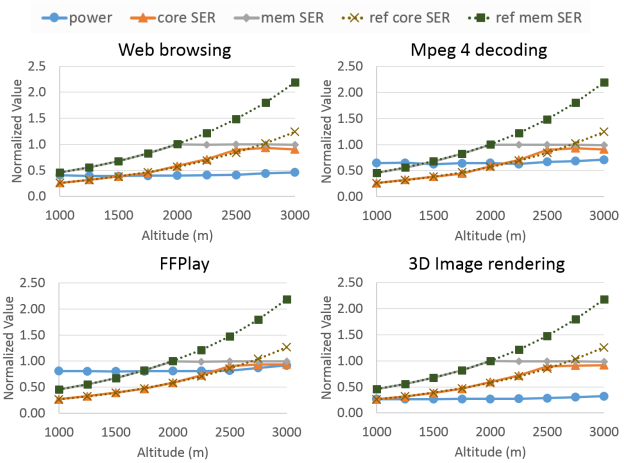


Figure 8: Power/SER Results of different software benchmarks on case I with dynamic policy.

the state table, a request is initiated to update current power management states. The state management module can also initiate a request for reliability bookkeeping module to update the SER values. Though we are working on a single-core platform, the module and policies are compatible for multi-core systems. For multi-core system, scheduling can also be a strong knob with techniques like workload aggregation or distribution in trading-off JIT vs. RFTS.

4.4 Experimental Evaluation of Reliability Management Policies

All experiments are run with the hardware evaluation platform described in Section 3. The dynamic policy is configured to account for overall SER over a period of one second, i.e., $T = 1s$ for Equation 3. The altitude values are fed to the system through a script. Corresponding SER values for processor core and memory are calculated based on the model in [14]. The reliability target is calculated based on each hardware configuration's worst SER states at an altitude of 2000m. In this section, if not otherwise mentioned, we report the maximum accumulative SER value over the period of one second and normalize it with respect to the SER targets, i.e., an SER value 1 means we are right at the SER target.

The implementation of the policies is verified by the reliability bookkeeping module. Both the static policy and dynamic policies can effectively control the SER with increasing altitude value. Some of the results with dynamic policies are plotted in Fig. 8, 9, 10, 11. For all benchmarks, our dynamic policy is able to retain both core SER and memory SER under the targets. In Fig. 8, the core SER reaches the target at around 2000m altitude, while the memory reach the target at 2500m altitude. The reason is that the worst-case workload patterns for core SER and memory SER are different (as suggested in Fig. 6). In Fig. 10, the core SER and memory SER are in the same pace because their dependence on workload is similar (see Fig. 6).

For all cases, the power overhead is small except for some large altitude cases. This is because extensive use of higher voltage states and shallower idle states are required once the gap between reference SER values and SER targets becomes large. This can also be explained with the limited dynamic range of SER in all possible power states. The study on the effectiveness of our dynamic policy against optimal power

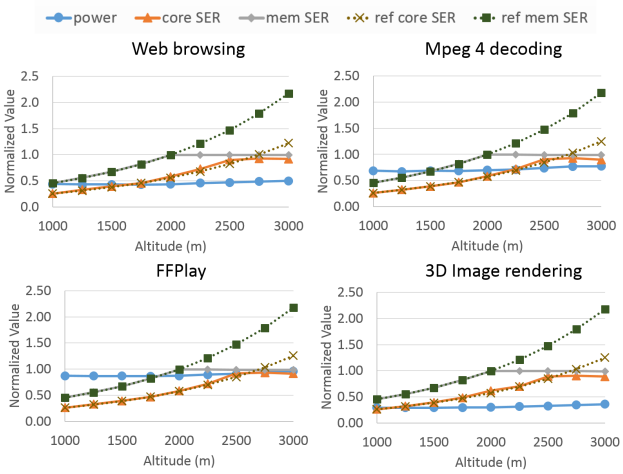


Figure 9: Power/SER Results of different software benchmarks on case II with dynamic policy.

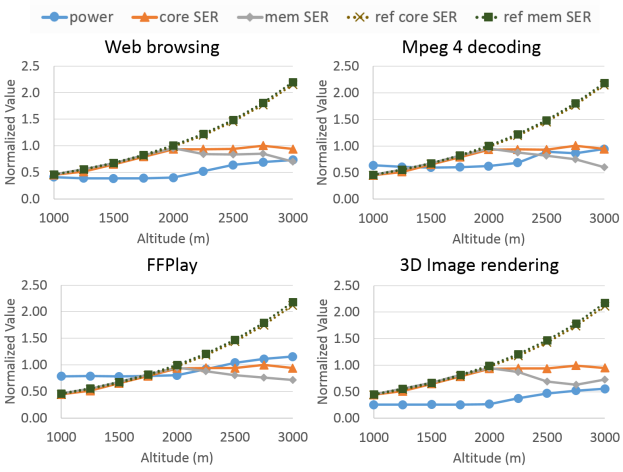


Figure 10: Power/SER Results of different software benchmarks on case III with dynamic policy.

state selection schemes will be described in Section 4.5.

The other way to study the policy behavior is to analyze how different power states are enabled/disabled. An example of web browsing benchmark is shown in Fig. 12. A free-running trace at sea level (see Fig. 12(a)) is used as a baseline for comparison. For case I at 2500m altitude (see Fig. 12(b)), where memory SER dominates, the policy retain SER target by disabling low voltage states, resulting in entering higher frequency states at around $t = 5s$ in Fig. 12(b). For case III at 2500 altitude (see Fig. 12(c)), where core SER dominates, the policy retain SER target by disabling deep sleep state so that the FFs spend less time in retention mode. This results in static frequency but varying SER (due to enabling/disabling the deepest idle state) at around $t = 5s$ in Fig. 12(c).

The system performance is also evaluated through quality metrics such as runtime and frame miss rate. Since our policies only disable lower frequency levels or long exit latency idle states, system performance with our policies is always the same or better. The overhead of our reliability bookkeeping module and state management module is negligible compared to the latency involved in changing power states.

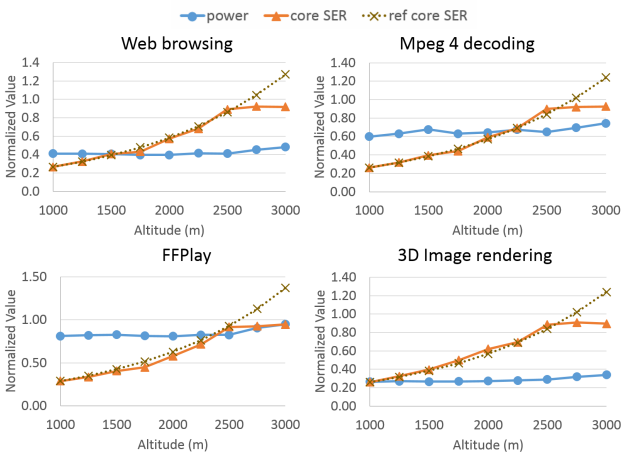


Figure 11: Power/SER Results of different software benchmarks on case IV with dynamic policy.

The small power overhead seen in the results is partly due to the fact that voltage scaling is not a strong knob for this technology. We also observe this from the generated libraries at different supply voltages. For other high subthreshold swing devices like FinFET, the circuits may scale better with voltage and thus have different results.

4.5 Policy Effectiveness Evaluation

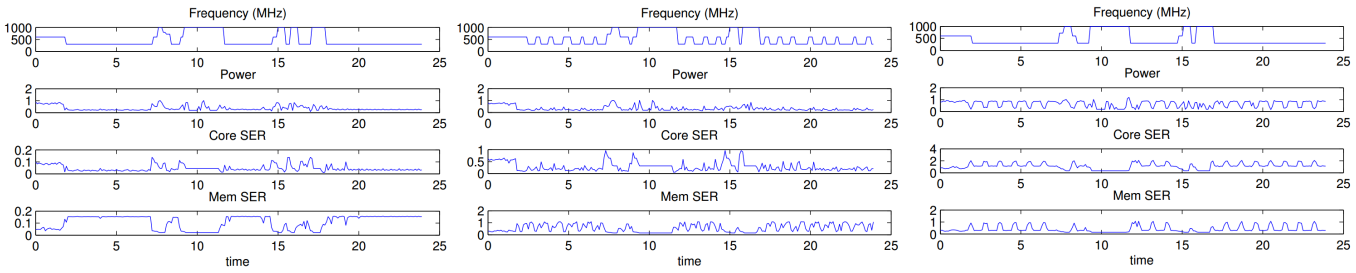
Our dynamic policy adopts a pessimistic approach by disabling all possible power states that will violate the reliability target if the states last for the entire remaining time, i.e., $t_{end} - t$ in Equation 4. This can result in left-over SER budget and unnecessary power overhead. To study the effectiveness of our policies, we also implement a trace-recording module, which can be enabled to store the all power state switching information. With these traces, we can characterize the optimal power state selection scheme offline and compare with our policies. Since our purpose is evaluating the effectiveness of our policies rather than evaluating the built-in Linux governors, we will use the power states suggested by the Linux governors as baseline. Only the same or higher power states, i.e., higher voltage states or shallower idle states are considered so that we won't have any performance impacts.

Considering the fact that P-state selection will change the idle time length and corresponding C-state selection contexts, we will implement two separate schemes for P-state and C-state selection.

To find the optimal C-state selection, we formulate the problem as an integer linear programming (ILP) problem as in Equation 5:

$$\begin{aligned} \text{minimize:} \quad & \sum_{n=1}^{N} (T_n \times \mathbf{P_n}^{\,t} \cdot \mathbf{S_n}) \\ \text{subject to:} \quad & \forall\, n : \mathbf{S_n}^{\,t} \cdot \mathbf{1} \geq 1 \\ & \sum_{n=1}^{N} (T_n \times \mathbf{R_n}^{\,t} \cdot \mathbf{S_n}) \leq K \\ & \forall\, n : \mathbf{S_n}^{\,t} \cdot \mathbf{L_n} \leq 0 \\ & \forall\, n : \mathbf{S_n} \geq \mathbf{0} \end{aligned} \tag{5}$$

The optimization objective is to minimize the energy overhead, where n is the index for each C-state selection, N is



(a) sea level (b) 2500m altitude Case I (c) 2500m altitude Case III

Figure 12: An example of web browsing benchmark execution with different altitude and hardware cases. For case I, P-states are exploited by the policy while C-states are used for case III. Note some y-axis scale for the figures are different.

the total number of C-state selections. T_n is the actual resident time for the C-state selection. Since we have four idle states here, $\mathbf{P_n}$ is a 4x1 vector corresponding to the power of each C-state. The power can be different for different n because the corresponding P-state can be different. $\mathbf{S_n}$ is selection vector, i.e., 4x1 integer vector for each C-state selection. The i-th entry in $\mathbf{C_n}$ equals 1 means the i-th C-state is selected. The first constraint ensures that one state is selected[3]. K is the SER budget for these idle time selection. The second constraint is to make sure that SER budget will not be exceeded. $\mathbf{R_n}$ is a 4x1 vector corresponding to the SER of each C-state. This SER can also be different for different n because of the corresponding P-state. $\mathbf{L_n}$ is a 4x1 vector representing the selection of built-in Linux governor. If the i-th C-state is selected by Linux governor, all entries including and after the i-th entry are 1 and 0 otherwise. So the third and fourth constraints are used to limit the available state selection.

For P-state selection, we will first calculate the total active time until next P-state switching time according to the operating frequency, then scale the idle time spent in each C-state accordingly. This actually is likely to be optimistic because the optimal scheme will prefer a lower voltage state compared to our policies, which can result in less active time within the scheduling ticks and thus entering more into the shallower idle states.

For each P-state, we can first compute the new active time and idle time of each C-state. The we can characterize the energy and SER impacts for all P-states. To find the optimal P-state selection, we can also formulate the problem as an Integer Linear Programming (ILP) problem in Equation 6.

$$\begin{aligned} \text{minimize:} \quad & \sum_{m=1}^{M} (\mathbf{E_m}^{\,t} \cdot \mathbf{S_m}) \\ \text{subject to:} \quad & \forall\, m : \mathbf{S_m}^{\,t} \cdot \mathbf{1} \geq 1 \\ & \sum_{m=1}^{M} (\mathbf{R_m}^{\,t} \cdot \mathbf{S_m}) \leq K \\ & \forall\, m : \mathbf{S_m}^{\,t} \cdot \mathbf{L_m} \leq 0 \\ & \forall\, m : \mathbf{S_m} \geq \mathbf{0} \end{aligned} \tag{6}$$

where m is the index for each P-state selection, M is the total number of P-state selections. Since we have five P-states, $\mathbf{E_m}$ and $\mathbf{R_m}$ are 5x1 vectors and corresponding to the pre-characterized energy and SER difference. Similar to C-state formulation, S_m is the 5x1 selection vector for P-states. K is the SER budget.

[3]An upper bound here is unnecessary due to the minimization and the fact that $\mathbf{P_n}$ and T_n are non-negative

Table 5: The additional power of running the benchmarks with our policies compared to the optimal P-state and C-state selection schemes

Benchmark	P-state optimal		C-state optimal	
	Mean	Max	Mean	Max
Web browsing	0.08%	0.42%	0.72%	2.18%
MPEG 4 decoding	0.14%	0.73%	0.63%	2.47%
FFPlay	0.18%	1.31%	0.58%	2.51%
3D Image rendering	0.14%	1.17%	0.53%	1.89%

The formulated ILP problems are solved by LP_solve [2]. The comparison results for software benchmarks running at altitude of 3000m are summarized in Table 5. Over all cases, our dynamic policy is within 1.5% with the optimal P-state selection scheme and within 3% with the optimal C-state selection scheme. The difference is smaller for P-state because the overhead of selecting higher voltage state can be compensated by spending more time in idle state, i.e., JIT vs. RFTS. But selecting shallower idle state will not change the utilization and thus the P-state selection. This also supports our claim that different power management schemes (i.e., JIT and RFTS) can be exploited for achieving system reliability targets.

5. CONCLUSION

In this paper, we developed a hardware evaluation platform to assess system reliability under different system power management schemes. We also propose a system/cloud-based virtual sensing for reliability evaluation. We propose two reliability management policies based on the already existing power management schemes. Experimental results with real life benchmarks show that our policy is effective and achieves system reliability target with minimal power overhead, compared to the optimal schemes characterized offline. Future work will try to include more reliability issues. The kernel module code is available for download at https://github.com/nanocad-lab/JIT-RFTS.

Acknowledgment

This work is supported in part by NSF Variability Expedition grant CCF-1029030. The authors would like to thank Prof. Mehdi Tahoori and Mojtaba Ebrahimi, Karlsruhe Institute of Technology, for their generous help in getting the FIT rate data. The authors would also like to thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of the paper.

6. REFERENCES

[1] Cadence encounter digital implementation system. http://www.cadence.com/products/di/edi_system/pages/default.aspx.

[2] lp_solve. http://lpsolve.sourceforge.net/.

[3] D. Alexandrescu. A comprehensive soft error analysis methodology for socs/asics memory instances. In *On-Line Testing Symposium (IOLTS), 2011 IEEE 17th International*, pages 175–176. IEEE, 2011.

[4] am335x. http://www.ti.com/product/am3358.

[5] Am335xsdk. http://software-dl.ti.com/sitara_linux/esd/AM335xSDK/-latest/index_FDS.html.

[6] Beaglebone black. http://beagleboard.org/Products/BeagleBone+Black.

[7] V. Chandra and R. Aitken. Impact of technology and voltage scaling on the soft error susceptibility in nanoscale cmos. In *Defect and Fault Tolerance of VLSI Systems, 2008. DFTVS'08. IEEE International Symposium on*, pages 114–122. IEEE, 2008.

[8] H. Cho et al. Quantitative evaluation of soft error injection techniques for robust system design. In *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*, pages 1–10. IEEE, 2013.

[9] cpufreq. https://www.kernel.org/doc/Documentation/cpu-freq/.

[10] cpuidle. https://www.kernel.org/doc/Documentation/cpuidle/.

[11] M. Ebrahimi et al. Comprehensive analysis of alpha and neutron particle-induced soft errors in an embedded processor at nanoscales. In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, pages 1–6. IEEE, 2014.

[12] M. Fan, Q. Han, S. Liu, and G. Quan. On-line reliability-aware dynamic power management for real-time systems. In *Quality Electronic Design (ISQED), 2015 16th International Symposium on*, pages 361–365. IEEE, 2015.

[13] ffmpeg. https://www.ffmpeg.org/.

[14] M. Gordon et al. Measurement of the flux and energy spectrum of cosmic-ray induced neutrons on the ground. *Nuclear Science, IEEE Transactions on*, 51(6):3427–3434, 2004.

[15] E. Karl et al. Reliability modeling and management in dynamic microprocessor-based systems. In *Proceedings of the 43rd annual Design Automation Conference*, pages 1057–1060. ACM, 2006.

[16] M. Keating et al. *Low Power Methodology Manual: For System on Chip Design*. Springer, 2007.

[17] N. Mahatme et al. Analysis of soft error rates in combinational and sequential logic and implications of hardening for advanced technologies. In *Reliability Physics Symposium (IRPS), 2010 IEEE International*, pages 1031–1035. IEEE, 2010.

[18] P. Mercati et al. A linux-governor based dynamic reliability manager for android mobile devices. In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, pages 1–4. IEEE, 2014.

[19] S. Mitra et al. Built-in soft error resilience for robust system design. In *Integrated Circuit Design and Technology, 2007. ICICDT '07. IEEE International Conference on*, pages 1–6, May 2007.

[20] S. S. Mukherjee et al. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, page 29. IEEE Computer Society, 2003.

[21] P. Pop et al. 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*, pages 233–238. ACM, 2007.

[22] X. Qi, D. Zhu, and H. Aydin. Global scheduling based reliability-aware power management for multiprocessor real-time systems. *Real-Time Systems*, 47(2):109–142, 2011.

[23] S. Rehman, K.-H. Chen, F. Kriebel, A. Toma, M. Shafique, J.-J. Chen, and J. Henkel. Cross-layer software dependability on unreliable hardware. *Computers, IEEE Transactions on*, PP(99):1–1, 2015.

[24] S. Rehman, F. Kriebel, D. Sun, M. Shafique, and J. Henkel. dtune: Leveraging reliable code generation for adaptive dependability tuning under process variation and aging-induced effects. In *Proceedings of the 51st Annual Design Automation Conference*, pages 1–6. ACM, 2014.

[25] T. S. Rosing, K. Mihic, and G. De Micheli. Power and reliability management of socs. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 15(4):391–403, 2007.

[26] P. Shivakumar et al. Modeling the effect of technology trends on the soft error rate of combinational logic. In *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*, pages 389–398. IEEE, 2002.

[27] R. Sridharan and R. Mahapatra. Reliability aware power management for dual-processor real-time embedded systems. In *Proceedings of the 47th Design Automation Conference*, pages 819–824. ACM, 2010.

[28] N. J. Wang et al. Restore: Symptom-based soft error detection in microprocessors. *Dependable and Secure Computing, IEEE Transactions on*, 3(3):188–201, 2006.

[29] L. Wanner et al. Varemu: an emulation testbed for variability-aware software. In *Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, page 27. IEEE Press, 2013.

[30] H. Zhang et al. Guard: Guaranteed reliability in dynamically reconfigurable systems. In *Proceedings of the The 51st Annual Design Automation Conference on Design Automation Conference*, pages 1–6. ACM, 2014.

[31] D. Zhu. Reliability-aware dynamic energy management in dependable embedded real-time systems. In *Real-Time and Embedded Technology and Applications Symposium, 2006. Proceedings of the 12th IEEE*, pages 397–407. IEEE, 2006.