Schedulability Analysis for Processors with Aging-Aware Autonomic Frequency Scaling

Alejandro Masrur, Philipp Kindt, Martin Becker
and Samarjit Chakraborty
Institute for Real-Time Computer Systems
TU Munich, Germany

Veit Kleeberger, Martin Barke
and Ulf Schlichtmann
Institute for Electronic Design Automation
TU Munich, Germany

***Abstract*—With the rapid progress in semiconductor technology and the shrinking of device geometries, the resulting processors are increasingly becoming prone to effects like aging and soft errors. As a processor ages, its electrical characteristics degrade, i.e., the switching times of its transistors increase. Hence, the processor cannot continue error-free operation at the same clock frequency and/or voltage for which it was originally designed. In order to mitigate such effects, recent research proposes to equip processors with special circuitry that automatically adapts its clock frequency in response to changes in its circuit-level timing properties (arising from changes in its electrical characteristics). From the point of view of tasks running on these processors, such *autonomic frequency scaling* (AFS) processors become slower as they gradually age. This leads to additional execution delay for tasks, which needs to be analyzed carefully, particularly in the context of hard real-time or safety-critical systems. Hence, for real-time systems based on AFS processors, the associated schedulability analysis should be *aging-aware* which is a relatively unexplored topic so far. In this paper we propose a schedulability analysis framework that accounts such aging-induced degradation and changes in timing properties of the processor, when designing hard real-time systems. In particular, we address the schedulability and task mapping problem by taking a *lifetime constraint* of the system into account. In other words, the system should be designed to be fully operational (i.e., meet all deadlines) till a given minimum period of time (i.e., its lifetime). The proposed framework is based on an aging model of the processor which we discuss in detail. In addition to studying the effects of aging on the schedulability of real-time tasks, we also discuss its impact on task mapping and resource dimensioning.**

I. Introduction

As device geometries continue to shrink with the advances in fabrication technology, the resulting processors are increasingly becoming susceptible to effects like aging. As a processor ages, the switching times of its constituent transistors increase because of which the processor is no longer able to sustain the clock frequency it was originally designed for. Hardware solutions to mitigate such effects consist of equipping processors with on-chip monitors or sensors [8], [5] that measure the timing margin available to circuits on the chip, or variations in their timing behavior arising from changes in their electrical characteristics. The output from such monitors is coupled with the clock generation circuit to adjust the clock frequency in response to changes in the signal propagation delay due to effects like aging. Such techniques have already been used in IBM's POWER7 architecture [9] in order to automatically adjust the processor's clock frequency and voltage level, with the aim of saving energy. The same technique is also applicable to cope with the effects of aging.

In such *autonomic frequency scaling* (AFS) processors, the execution times of tasks increase over time as the processor ages. For real-time or safety-critical applications running on such AFS processors, a natural question is: how to perform timing or schedulability analysis? Given a life-time constraint, i.e., the duration of time during which all deadlines have to be met, one obvious solution would be to perform schedulability analysis with task execution times at the end of this duration (i.e., using the *aged* execution times). However, for cost-sensitive domains such a naïve approach might be overly pessimistic and lead to resource overdimensioning.

As a concrete example, let us consider the automotive domain. Today, high-end cars have 50-100 electronic control units (ECUs) or processors to run a variety of hard real-time, safety-critical control applications. Currently, these ECUs are often low-cost processors or microcontrollers that run at 200-500 MHz. However, in order to reduce cost, weight and cabling requirements, there is an increasing effort on *ECU consolidation*, i.e., integrating multiple functionalities on fewer more powerful ECUs. In the near future, we will see ECU architectures evolve into powerful, multi-core processors that are currently found in the high-performance computing domain. As soon as that happens, such ECUs will be faced with the effects of aging, soft errors, etc., especially because they will be exposed to a wide variety of operating environments and temperatures, depending on where the car is deployed. This will be coupled with the fact that additional processor cooling mechanisms – as found in the mainstream computing domain – will probably be absent. Given that it is common for automotive OEMs to provide guarantees in the range of 15 years, it has to be ensured that tasks running on aging-aware *autonomic frequency scaling* or AFS ECUs continue to meet all deadlines over this entire time period. In this paper we propose an appropriate processor aging-aware schedulability analysis for such scenarios.

Processors with autonomic frequency scaling: In general, aging effects can be divided into two categories: degradation and destructive effects. Degradation effects may be classified into *Negative Bias Temperature Instability* (NBTI) and *Hot*

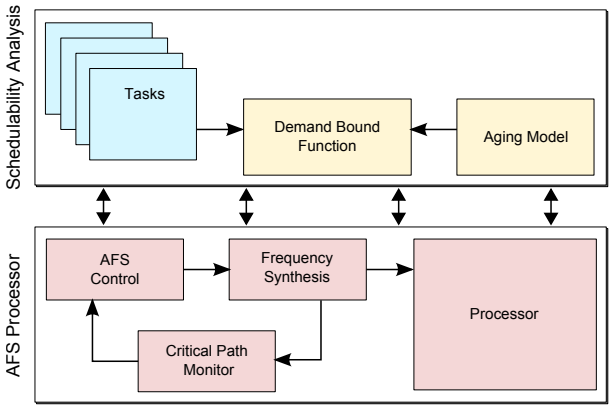


Fig. 1. Schematic view of an AFS processor

Carrier Injection (HCI). These cause a shift in a device's electrical characteristics, such as threshold voltage or on-current, which leads to prolonged switching times of transistors. On the other hand, destructive effects like *Time-Dependent Dielectric Breakdown* (TDDB) or *Electro Migration* (EM) cause the destruction of device parts and irreparable damage. In this paper, we are concerned with degradation effects, i.e., those that do not damage a processor but rather affect its timing behavior.

As a processor ages, its transistors gradually need more time for switching (due to degradation effects). Hence, as mentioned before, the processor cannot sustain the clock frequency it was originally designed for. While very large safety margins or *timing guardbands* can be used to mitigate such effects, they lead to poor resource utilization and hence more expensive processors, which are not acceptable in cost-sensitive domains like automotive architectures. Autonomic frequency scaling (AFS) processors use *critical path monitors* (CPM) which allow measuring the maximum delay degradation at runtime (see Fig. 1). Depending on the implementation, an AFS processor may become slower as it ages, in order to keep the processor operational. To reliably design real-time systems based on AFS processors, it is necessary to analyze the system's aging behavior and develop timing or schedulability analysis techniques that are aging-aware.

Our contributions: As discussed above, an AFS processor automatically reduces its clock frequency to account for additional circuit-level signal propagation delay incurred due to aging. This way, although the processor becomes slower, it avoids the occurrence of aging-dependent functional/computation errors. However, in the context of real-time systems, we need to analyze what the maximum slow-down is, to be able to guarantee deadlines all along the system's lifetime. For this purpose, we make use of the aging model from [14] to predict the processor's *worst-case* aging behavior within the desired lifetime t_{life}. This model returns the *maximum* aging-dependent delay D_{max} as a function of the *time under stress* t_{stress} (i.e., the time in which the processor is busy executing some workload).

Now, to account for aging in schedulability analysis, we can use our aging-model to obtain D_{max} considering a t_{stress} equal to t_{life} and compute the *speed* of the *aged* AFS processor after t_{life} of continuous use. Clearly, if the aged AFS processor can guarantee all deadlines, then the system is schedulable along its whole lifetime. However, even considering aging, the processor utilization is normally below 100% and, in most cases, t_{stress} is much less than t_{life} (i.e., the processor is not being constantly used during its lifetime, but rather it has some idle intervals). As a result, this first naïve approach leads to a pessimistic slow-down estimation and, hence, a more *expensive* design.

In this paper, we are concerned with a second, tighter analysis, which consists of integrating our aging model into the well-known *demand bound function* (DBF) for fixed-priorities [10]. We derive this way an aging-dependent DBF which can be used to perform a more accurate schedulability analysis for AFS processors. In particular, we can apply the proposed schedulability analysis to design AFS-based systems such that all timing constraints are guaranteed within a desired lifetime.

In addition, it is further possible to design task allocation algorithms that take aging into account. In this paper, we propose one such algorithm based on the well-known First-Fit heuristic – note that optimization algorithms such as branch and bound can also be used for this purpose. We show that an allocation algorithm based on the proposed approach leads to a more efficient design (i.e., one which requires fewer processors) than an allocation algorithm based on the naïve approach.

This paper is organized a follows: Section II gives an overview of related work, whereas Section III explains the principles behind AFS processors. Section IV introduces and discusses the models and the notation used along this paper. We also explain our aging model in detail before continuing with Section V where we derive and discuss our aging-aware schedulability framework. In Section VI we introduce a task mapping algorithm and in Section VII some experimental results will be summarized. Finally, Section VIII concludes the paper.

II. Related Work

The aging behavior of semiconductors is a major topic of research within the processor architecture community, but it has so far received relatively less attention from the software community. Notable exceptions to this are [7], [23], [16], [17].

There exists a large body of work that addresses aging effects at the hardware level. For example, Wu and Marculescu presented optimization techniques that allow synthesizing digital circuits that are less prone to aging degradation [23]. In [24], different process variations at a chip level are studied and characterized. In [16], [17], software techniques are presented to cope with problems that are posed by unreliable hardware. These include devising reliability-aware instruction sets, coupled with appropriate instruction scheduling using reliability-aware compilation techniques.

Recently, in [7], Huang et al. presented an allocation framework based on a heuristic that aims at maximizing the lifetime of an SoC (System-on-Chip). Our work here follows this line of research and also deals with the allocation issues that arise in aging devices. However, in contrast to recent research efforts, the approach presented here deals with the *schedulability* of real-time tasks and extends the state-of-the-art by considering hardware-dependent behavior in the schedulability and mapping problem, viz., aging effects.

III. Aging-Aware Autonomic Frequency Scaling

A processor ages with the time under stress (i.e., the time it is executing some workload). In general, aging effects can be divided into two categories: destructive and degradation effects. Destructive effects like *Time-Dependent Dielectric Breakdown* (TDDB) or *Electro Migration* (EM) cause the destruction of device parts and irreparable damage.

In this paper, we are concerned with degradation effects such as *Negative Bias Temperature Instability* (NBTI) and *Hot Carrier Injection* (HCI). These cause a shift of a device's electrical characteristics, e.g., threshold voltage or on-current, which results in elongated switching times of transistors [20], [4]. This leads to errors and, hence, the processor cannot continue to run at the same frequency.

There are a number of approaches that can be used to countervail aging degradation. In the following we discuss the most relevant such approaches:

- **Safety margins:** A processor's lifetime correct operation can be guaranteed by considering safety margins in, for example, supply voltage and/or clock frequency. These safety margins should take the worst-case aging conditions into consideration. In other words, if we know that a processor incurs a maximum aging-dependent delay D_{max}, we can reduce its clock frequency by an amount (i.e., a safety margin) that accounts for D_{max}. However, safety margins usually lead to a waste of resources (or energy) and, thus, a pessimistic design [9].
- **Supply voltage regulation:** Increasing the supply voltage V_{CC} allows reducing the delay of an aged circuit [19] and, therefore, helps to keep the clock frequency constant even though aging proceeds. An adaptive control circuit can automatically regulate the supply voltage to a currently needed value [4]. However, increasing the supply voltage strongly worsens the dynamic power consumption P_d [15]:
$$P_d \propto C_L V_{CC}^2 f, \tag{1}$$
where C_L is the total load capacitance in a chip and f is the operating frequency. In addition, the supply voltage is normally limited by maximum allowable input current, cooling constraints, temperature-dependent reliability issues, etc. [20]. This latter makes a supply voltage regulation be difficult to implement, in particular, as aging degradation becomes more relevant with shrinking device dimensions.
- **Clock frequency scaling:** Along a processor's lifetime, its clock frequency can be adapted to account for aging degradation [4]. In contrast to supply voltage regulation, where V_{CC} needs to be increased, the clock frequency needs to be *decreased* to account for aging. Hence, the power consumption goes down – see Eq. (1). However, as discussed previously, by reducing the clock frequency the processor becomes slower.

Clearly, all the above approaches have advantages and disadvantages. However, since scaling the clock frequency allows reducing power consumption, we believe this to be the most promising technique, particularly, in the context of embedded systems. Note that it is possible to combine supply voltage regulation with clock frequency scaling [9]. This leads to an intermediate solution with less power consumption than voltage regulation alone, but still more than what it is needed by sole frequency scaling.

In this section, we briefly describe existing technologies that allow for Autonomic Frequency Scaling (AFS) to compensate aging degradation. Here the purpose is to reduce the clock frequency; however, this should not be reduced more than necessary (otherwise, more processing speed is sacrificed than actually necessary). For this reason, we talk about finding the maximum allowable clock frequency, i.e., the maximum possible frequency that guarantees error-free operation.

To adjust the clock frequency to its maximum allowable value under current operating conditions (i.e., temperature, supply voltage drops, aging, etc.), a control circuit is required. We refer to this circuit as AFS control – see Fig. 1. Another circuit called Critical Path Monitor (CPM) allows measuring the difference between the current and the maximum allowable clock frequency. The output of the CPM is connected to the AFS control. The output of the AFS control is then led to a third circuit which allows synthesizing the clock frequency. The frequency synthesis allows quick, glitch-free, virtually continuous changes of the operating frequency [20].

The AFS control follows a desired strategy (i.e., control algorithm). In [6], [9], a power management strategy is implemented. However, in this paper, we assume that the AFS control adjusts the clock frequency to its maximum allowable value which then varies (i.e., decreases) as the processor ages.

1) Critical Path Monitor: As mentioned above, a circuit termed Critical Path Monitor (CPM) is used to detect the difference between the current and the maximum allowable clock frequency.

Fig. 2 shows the CPM circuit as proposed by [15]. The main idea behind CPM is as follows: A pulse-like signal is fed to a circuit that *replicates* the *critical path* within the processor, i.e., the electric network usually consisting of multiple cascaded stages with the longest propagation delay. Note that the critical path determines the maximum allowable clock frequency, since all its stages need to have enough time to *switch* within one clock cycle. The CPM circuit measures the critical path's delay and compares it with the current clock period to determine the difference between them.

With shrinking device dimensions, within-die and die-to-die variations are becoming more relevant. As a result, static timing analysis at circuit level does not suffice to determine

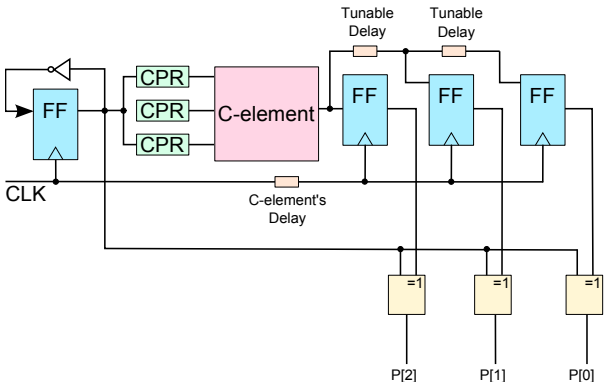


Fig. 2. Schematic view of the critical path monitor (CPM)

which electric network constitutes the critical path of a processor [15]. However, it is possible to identify all critical path *candidates*, i.e., a reduced subset of the processor's electric networks which might become critical at runtime. The CPM circuit then consists of replicas of each of these circuits, termed *Critical Path Replicas* (CPRs).

In Fig. 2, the flip-flop on the left-hand side toggles with every cycle of CLK (i.e., the clock signal) since its inverted output is fed back to its input. This toggling signal has to pass through each of the CPRs in parallel. Since the different CPRs may have different delays, these are connected to a special circuit called *C-element*. The C-element's output changes when all its inputs (coming from the different CPRs) have the same logic level (either high or low levels) – note that CPRs are designed such that they do not change the polarity (i.e., do not invert) their input signals. If the C-element's inputs have different logic values, its output remains unchanged. This way, the C-element toggles at the rate of the slowest CPR in the system.

The output of the C-element is further connected to a number of *detection* flipflops on the right-hand side of Fig. 2. Note that the clock signal attached to the detection flipflops is deferred by the C-element's delay. This way, it is possible to measure the CPR's delay without adding the extra delay incurred by the C-element. By means of tunable delays, the output signal of the C-element (i.e., the output signal of the slowest CPR) arrives to each of the detection flipflops at deferred points in time. Hence, by properly adjusting the tunable delays, the CPM circuit measures the difference between the clock and the delayed output signal of the slowest CPR. For this purpose, XOR gates are used to compare the output of the toggling flipflop on the left-hand side Fig. 2 with the outputs of the detection flipflops. The resulting $P[2]...P[0]$ quantifies the difference between the clock and the delayed output from the slowest CPR.

If the signal edge only arrives at the first detection flipflop (in one clock tick), $P[2]$ will be high, but $P[1]$ and $P[0]$ will be low. In this case, the clock frequency must be decreased. After the frequency has been decreased to a lower value, it will reach the second detection circuit. $P[2]$ and $P[1]$ will be high, but $P[0]$ is still low. If $P[0]$ and $P[1]$ and $P[2]$ are high, the frequency can be increased. In other words, the output signal of the CPM is a 3-bit code indicating the slack between the current and the maximum allowable clock frequency. This technique can be extended for more accurate frequency measurements. For instance, 12-bit CPMs are implemented in the POWER7 architecture [9].

2) Frequency synthesis: For implementing an autonomic frequency scaling, a frequency generator is required that is capable of changing the clock frequency in a seamlessly manner.

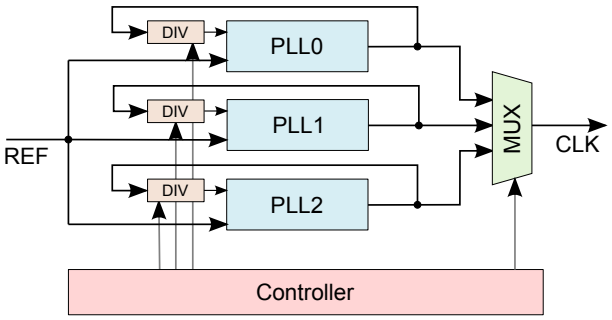


Fig. 3. Frequency synthesis circuit

Fig. 3 depicts a frequency synthesis circuit similar to the one described in [20]. The well-known frequency synthesis technique *Phase Lock Loop* (PLL) is used to allow for fast changes in the clock frequency. Three PLLs are locked to different multiples of a reference clock (denoted by REF).

A controller commands a set of clock dividers allowing the different PLLs to synthesize variations of REF, and a multiplexer that selects the output of one PLL. The controller implements a desired control strategy for changing the output frequency. For example, one PLL can be locked to the current clock frequency. The second PLL can then be locked to a slightly higher whereas the third one is locked to a slightly lower frequency. In case the clock frequency needs to be modified, either the higher- or the lower-frequency PLL is multiplexed to the output. (In our case, since the clock frequency needs to be reduced due to aging, normally the lower-frequency PLL is selected.) This allows quick frequency changes without needing to wait for a PLL to re-lock. Afterwards, the two other PLLs are re-locked to frequencies that are slightly below and slightly above the output frequency CLK.

The CPM and frequency synthesis circuits are connected by the AFS control block (see Fig. 1) to allow adaptive frequency controlling. The AFS control block implements a desired control strategy. As discussed above, such frequency control circuits have been already implemented, for example, in the POWER7 architecture [9].

IV. Models and Notation

In this section we discuss the different models used for performing a schedulability analysis on AFS processors. In particular, we introduce the task, the processor, and the aging model used in the paper. We further introduce the most

relevant notation, however, some notation will be discussed as it becomes necessary along the paper.

A. Task model

We denote by $\mathbf{T}$ a set of n real-time tasks running in the system. All these tasks are considered to be fully preemptive and independent of each other. Since tasks may be mapped to more than one processor, we use $\mathbf{T}_l$ to denote an arbitrary subset from $\mathbf{T}$ running on an AFS processor P_l. Each task T_i in $\mathbf{T}_l$ is characterized by:

- its relative deadline d_i,
- its worst-case execution time e_i, and
- its minimum inter-release time p_i.

In general, a real-time task T_i can be seen as an infinite sequence of jobs (or task instances). Jobs are released with at least a minimum possible separation p_i between them. In this paper, the relative deadlines d_i are assumed to be less than or equal to the respective minimum inter-release times p_i for all tasks. In addition, e_i must also be less than d_i. Otherwise, T_i alone is not schedulable, i.e., it will not be able to meet its deadline under any condition. Note that, since an AFS processor becomes slower along its lifetime, e_i stands for the worst-case execution time of a T_i at the processor's *unaged* speed s_0.

The ratio $u_i = \frac{e_i}{p_i}$ is known as task utilization and the sum of all u_i in $\mathbf{T}_l$ is the total utilization on the l-th processor considering again P_l's unaged speed: $U_l = \sum_{T_i} \frac{e_i}{p_i}$, where $T_i \in \mathbf{T}_l$.

Further, in this paper, we consider a fixed-priority scheduling policy. That is, tasks are assigned priorities that remain unchanged at runtime. Clearly, independent of the scheduling policy, the processor utilization U_l must be less than one in order that $\mathbf{T}_l$ be schedulable on P_l. Otherwise, $\mathbf{T}_l$ is not schedulable on the unaged P_l.

B. Processor model

The processing platform in this paper consists of a set of identical AFS processors that we denote by $\mathbf{P}$. As already mentioned, the initial (i.e., unaged) speed of a processor in $\mathbf{P}$ will be denoted by s_0. Note that s_0 does not depend on l, since it is the same for all P_l in $\mathbf{P}$. A processor's speed is a factor indicating that it takes the processor $\frac{e_i}{s_0}$ time for executing e_i time units of computation demand. Without loss of generality, we assume that $s_0 = 1$ holds.

From our previous discussion, we know that the switching delays of transistors increase as aging progresses. In other words, we will need to wait longer for transistors to switch. An AFS processor reduces its clock frequency in order to avoid errors due to this increased propagation delay. However, by reducing the clock frequency, the speed of the AFS processor decreases.

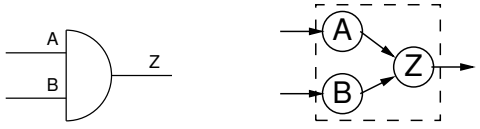
The aging-dependent speed of an AFS processor P_l is a function of t_{stress}, i.e., the time P_l is busy executing some workload. Hence, at a given point in time t_x, P_l's speed denoted by s_{lx} depends on P_l's *cumulative* workload in the time interval $(0, t_x]$. Clearly, if $t_x = 0$ holds, s_{lx} is equal to s_0. However, as time elapses, the cumulative workload executed by P_l increases and P_l starts aging. Its speed s_{lx} decreases for bigger t_x, i.e., it becomes less than 1 – note that $0 < s_{lx} \leq s_0$ holds. As a result, for any $t_x > 0$, P_l needs a longer time ($\frac{e_i}{s_{lx}} > \frac{e_i}{s_0}$) to execute e_i time units of computation demand.

C. Aging Model

Overall, the speed of any digital circuit, including processors, is directly proportional to the frequency at which it works. Of course, in the case of processors, the access delay to memory constitutes another speed limiting factor. Let us divide a T_i's worst-case execution time into memory access delay $D_{men,i}$ and execution time on the CPU $D_{cpu,i}$: $e_i = D_{men,i} + D_{cpu,i}$. Even if $D_{men,i}$ remains constant, $D_{cpu,i}$ and hence e_i vary with the clock frequency. If $D_{men,i} << D_{cpu,i}$ holds (e.g., data is retrieved from a high-end cache or T_i mainly works on data stored in the processor's registers), e_i is going to depend in a non-negligible manner on the clock frequency. As a result, a higher clock frequency yields shorter execution times, i.e., a faster processing time.

In general, in order to obtain the highest possible clock frequency for a digital circuit, the maximum signal propagation delay (between the primary inputs and outputs) needs to be computed. Since the propagation delay depends on aging effects, these have to be taken into account at all points in time.

To compute the aging-dependent propagation delay, we make use of the technique presented in [14]. This consists of translating the circuit's netlist into a directed acyclic graph, where every circuit element is represented by a set of nodes (or vertices v_j) and their corresponding edges (or lines l_i) – see Fig. 4.



(a) AND gate (b) Timing graph representation

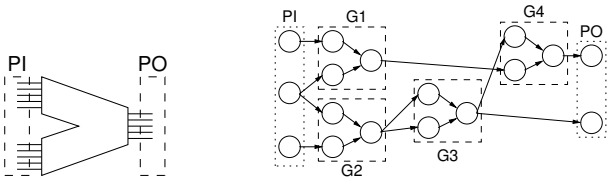
Fig. 4. A circuit element and its timing graph

Each pin of a circuit element is represented by a node (v_j) in the graph and the edges (l_i) between them represent all possible signal propagation paths inside the circuit element. The whole circuit graph is then built by connecting these nodes together (see Fig. 5).

The maximum circuit delay between the primary inputs and outputs is then computed by *topological traversal* of this graph [18]. It has been shown that such an approach is applicable to even extremely large circuits and provides an accurate estimate for a processor's clock frequency [21].

First, for all fan-in edges of a node, the corresponding edge delay D_{l_i} (e.g., a pin-to-pin delay) is added to the signal arrival time at the edge input $t_{l_i,\text{in}}$.

$$t_{l_i,\text{out}} = t_{l_i,\text{in}} + D_{l_i} \tag{2}$$



(a) Functional circuit unit

(b) Timing graph of circuit unit

Fig. 5. Functional circuit unit and timing graph representation consisting of primary inputs (PI), cells (G_i), and primary outputs (PO)

Afterwards, the resulting arrival time t_{v_j} of the signal at the node v_j is the maximum arrival time among all fan-in edges l_i.

$$t_{v_j} = \max_{\forall l_i} (t_{l_i,\text{out}}) \tag{3}$$

As the circuit ages every pin-to-pin delay of every cell varies with time. The aging effects *Negative Bias Temperature Instability* (NBTI) and *Hot Carrier Injection* (HCI) cause a drift of the single transistor parameters. Thus, the delay model of each cell has to incorporate these effects into the pin-to-pin delay computation. This can be done by a linear approximation. That is, the delay of the aged cell can be computed by the sum [14]:

$$D_{\text{age}} = D_0 + D_{\text{NBTI}} + D_{\text{HCI}} \tag{4}$$

The nominal unaged delay is denoted by D_0 and the impact of the different aging effects by D_{NBTI} and D_{HCI} respectively. The impact of the aging effects is given by an empirical technology-dependent degradation equation. For NBTI, this degradation equation can be represented as:

$$D_{\text{NBTI}} = f(T, V_{CC}, t_{stress}, W), \tag{5}$$

where T is the device temperature, V_{CC} the supply voltage, t_{stress} the time the circuit is constantly being used, and W the workload of the processor which is characterized by the data it processes. Similarly, the degradation equation for HCI depends on the same parameters.

With this approach, we can model the worst-case aging behavior of any digital circuit. In the case of a processor, we can characterize how different circuits age considering different sets of T, V_{CC}, t_{stress} and W.

As discussed previously, an AFS processor reduces its clock frequency as aging effects progress. For this, it makes use of a CPM circuit which measures aging degradation in the critical path – see Section III. Knowing the worst-case aging degradation of the critical path, allows us to predict the worst-case slow-down of the AFS processor. Again, it is difficult to determine the critical path of a processor by static analysis, however, it is possible to identify a reduced number of *critical path candidates*. That is, a number of circuits that might become critical at runtime limiting the maximum allowable clock frequency. Now, characterizing the worst-case aging of all critical path candidates allows us to model the worst-case behavior of the whole processor.

On the other hand, among all parts of a processor, the *arithmetic logic unit* (ALU) is probably the most complex one. The ALU's execution paths are among the longest in a processor (i.e., they involve the largest number of cascaded transistors). As a consequence, the ALU is certainly one critical path candidate. Clearly, there might be other circuits which can also become critical at runtime. However, for the sake of simplicity, we consider the ALU as the only critical path candidate. Note that the analysis of this paper holds valid for all other possible critical path circuits.

In line with this, we use in this paper the ALU's worst-case aging model to infer the aging degradation and, hence, the maximum allowable clock frequency of the whole processor. Fig. 6 shows such an aging model for the 9-bit ALU (in particular, c5315). Here the aging-dependent delay degradation of the ALU is depicted against t_{stress}, i.e., the time in which the ALU is under continuous use. This is obtained considering those values of T, V_{CC} and W that lead to the worst-case aging degradation.

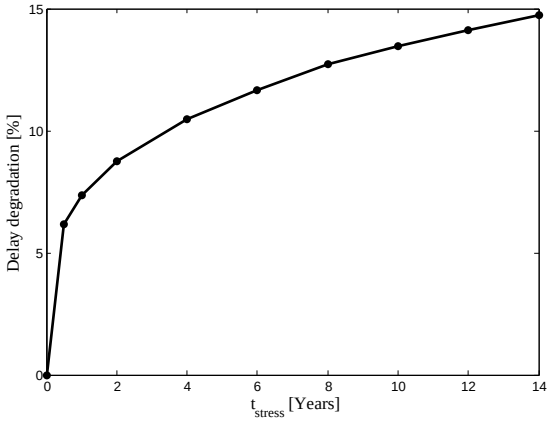


Fig. 6. Aging-dependent worst-case delay degradation for a processor considering its ALU as critical path

The curve of Fig. 6 gives the sum of $D_{\text{NBTI}} + D_{\text{HCI}}$. Further, using Eq. (4) the delay of the aging ALU can be determined. This delay is the inverse of frequency at which the ALU and, hence, the whole processor can then be operated without errorsw.

Finally, as stated in the previous sections, the processor ages with the time it is busy executing some workload – we denoted this time by t_{stress}. In general, if a processor's circuit is idle, circuit-level control techniques can be applied to put it into a non-aging state [22]. In this paper, we assume that this non-aging state is guaranteed at idle time.

V. Schedulability Analysis

In this section we present a technique for schedulability analysis that takes aging into account. As mentioned above, we consider a fixed-priority scheduling policy, i.e., where priorities are assigned once to tasks and do not change at runtime.

An example of a fixed-priority policy is Deadline Monotonic (DM) [12], where tasks are given priorities according to their deadlines d_i: the shorter d_i, the higher the priority of T_i.

We know that $\mathbf{T}_l$ is the set of real-time tasks running under fixed-priorities on a processor P_l. Traditionally, a schedulability test for $\mathbf{T}_l$ consists of computing the *worst-case response time* of every task $T_i \in \mathbf{T}_l$ [11], [1]. The worst-case response time of a T_i results from considering T_i's *critical instant* [13], i.e., that jobs of all higher-priority tasks are released simultaneously with T_i and with the minimum possible separation between them. This leads to the largest possible interference by higher-priority jobs and, hence, to T_i's longest (i.e., worst-case) response time. If the worst-case response time of every T_i in $\mathbf{T}_l$ is less than its corresponding d_i, then $\mathbf{T}_l$ is said to be schedulable on P_l.

As aging progresses, the speed of an AFS processor decreases. Thus, it takes the processor longer to finish a certain amount of execution or computation demand. From the point of view of the schedulability analysis, the execution demands of tasks become larger. As a result, even if $\mathbf{T}_l$ is schedulable on P_l at design time, it might be the case that some task T_i starts missing its deadline as the processor ages.

Of course, if we need to guarantee a given lifetime requirement t_{life}, we can use the worst-case aging model of Fig. 6 to compute the worst-case delay degradation D_{max} for a t_{stress} equal to t_{life}. However, since the processor utilization is normally below 100%, this approach (referred to as naïve approach in this paper) leads to a pessimistic schedulability analysis. This is because the workload (i.e., t_{stress}) in $(0, t_{life}]$ is usually less than t_{life}.

In this section we propose a tighter schedulability analysis which better integrates our aging model. This can be done by introducing the aging-dependent processor's speed s_{lx} into the demand bound function (DBF) of a $T_i \in \mathbf{T}_l$ – recall that s_{lx} denotes P_l's speed at time t_x. Clearly, the resulting DBF denoted by $h_i(t)$ allows computing the cumulative execution demand on the aged processor (after t_x time):

$$h_i(t) = \sum_{T_j \in HP(i)} \left\lceil \frac{t}{p_j} \right\rceil \frac{e_j}{s_{lx}}, \tag{6}$$

where $HP(i) \subset \mathbf{T}_l$ is the subset of tasks with higher priority than T_i together with T_i (e.g., for every T_j in $HP(i)$, $d_j \leq d_i$ must hold under the DM policy). Thus, $\sum_{T_j} \lceil \frac{t}{p_j} \rceil \frac{e_j}{s_{lx}}$ where $T_j \in HP(i)$ results in the maximum execution demand within $(0, t]$ (from the critical instant) produced by T_i and all its higher-priority tasks at a processor's speed of s_{lx}. Note that $h_i(t)$ assumes that s_{lx} remains constant within $(0, t]$. If the length of the interval $(0, t]$ is much less than the processor's age (i.e., $t << t_x$ holds), s_{lx} changes in a negligible manner. As a result, $h_i(t)$ is valid if $t << t_x$. This is normally the case since t_x is in the order of years whereas t is in the order of milliseconds.

From Eq. (6), T_i is schedulable if $h_i(d_i) \leq d_i$ holds. On the other hand, the minimum possible speed s_{lx} that allows meeting d_i is given by:

$$s_{lx} = \frac{\sum_{T_j \in HP(i)} \left\lceil \frac{d_i}{p_j} \right\rceil e_j}{d_i}. \tag{7}$$

Let us denote by D_{lx} the aging-dependent delay degradation that leads to the s_{lx} of Eq. (7) – note that $0 < D_{lx} < 1$ holds. While an unaged processor can execute e_i amount of computation in e_i time units, the aged processor needs $e_i(1 + D_{lx})$ time units for executing e_i. Hence, we can express s_{lx} as follows – recall that s_0 denotes the unaged speed of any P_l:

$$\frac{e_i}{s_{lx}} = \frac{e_i(1 + D_{lx})}{s_0}$$

$$s_{lx} = \frac{s_0}{1 + D_{lx}} \tag{8}$$

$$D_{lx} = \frac{s_0 - s_{lx}}{s_{lx}} \tag{9}$$

Now, we need to determine how much cumulative workload (i.e., what value of t_{stress} in Fig. 6) yields the aging-dependent delay degradation D_{lx}. For this purpose, since all tasks on P_l (the lower- and the higher-priority ones) generate workload, we need to consider the whole task set $\mathbf{T}_l$ instead of only $HP(i)$ in Eq. (6). We can use this DBF to compute P_l's total workload in the time interval $(0, t_x]$. This is denoted by $h_{|\mathbf{T}_l|}(t_x)$ and represents the value of t_{stress} that yields D_{lx} aging degradation.

Let us further assume that $D_k \leq D_{lx} < D_{k+1}$ holds where D_k and D_{k+1} are the degradation delays of two consecutive *markers* on the aging curve of Fig. 6. That is, D_{lx} falls onto the k-th segment of this aging curve between D_k and D_{k+1}. In addition, let us denote by t_k the t_{stress} corresponding to D_k and t_{k+1} the t_{stress} corresponding to D_{k+1} in Fig. 6. We can interpolate the value of $h_{|\mathbf{T}_l|}(t_x)$ in the following manner – note that $t_k \leq h_{|\mathbf{T}_l|}(t_x) < t_{k+1}$ must also hold:

$$D_{lx} - D_k = \alpha_k \left(h_{|\mathbf{T}_l|}(t_x) - t_k \right) \tag{10}$$

$$h_{|\mathbf{T}_l|}(t_x) = \frac{D_{lx} - D_k}{\alpha_k} + t_k, \tag{11}$$

where α_k is the slope of the k-th segment of the aging curve and is given by:

$$\alpha_k = \frac{D_{k+1} - D_k}{t_{k+1} - t_k}. \tag{12}$$

Now, similar as for Eq. (8), we can express s_{lx} as a function of s_k instead of s_0. Here, s_k denotes P_l's speed for $t_{stress} = t_k$ in Fig. 6:

$$\frac{1}{s_{lx}} = \frac{1 + D_{lx} - D_k}{s_k},$$

$$s_{lx} = \frac{s_k}{1 + D_{lx} - D_k}. \tag{13}$$

Replacing Eq. (10) in (13), we can find the expression of the demand bound function $h_{|\mathbf{T}_l|}(t_x)$ on the k-th segment of the aging curve:

$$h_{|\mathbf{T}_l|}(t_x) = \sum_{T_j \in \mathbf{T}_l} \left\lceil \frac{t_x}{p_j} \right\rceil \frac{e_j}{s_{lx}},$$
$$= \sum_{T_j \in \mathbf{T}_l} \left\lceil \frac{t_x}{p_j} \right\rceil \frac{e_j}{s_k}(1 + \alpha_k(h_{|\mathbf{T}_l|}(t_x) - t_k)).$$

Further, solving for $h_{|\mathbf{T}_l|}(t_x)$, we obtain:

$$h_{|\mathbf{T}_l|}(t_x) = \frac{(1 - \alpha_k t_k) \sum_{T_j \in \mathbf{T}_l} \left\lceil \frac{t_x}{p_j} \right\rceil \frac{e_j}{s_k}}{1 - \alpha_k \sum_{T_j \in \mathbf{T}_l} \left\lceil \frac{t_x}{p_j} \right\rceil \frac{e_j}{s_k}}. \quad (14)$$

The value of $h_{|\mathbf{T}_l|}(t_x)$ given by Eq. (11) produces an aging-dependent delay D_{lx} that leads to a speed s_{lx} as given in Eq. (7). With this s_{lx}, T_i can still meet its deadline. Any more additional workload than $h_{|\mathbf{T}_l|}(t_x)$ will make T_i miss its deadline. Considering this value of $h_{|\mathbf{T}_l|}(t_x)$ and that $\frac{t_x}{p_j} \leq \lceil \frac{t_x}{p_j} \rceil \leq \frac{t_x}{p_j} + 1$, we can find an upper and a lower bound for t_x, i.e., the time by which $h_{|\mathbf{T}_l|}(t_x)$ amount of cumulative workload is reached where $U_l = \sum_{T_j \in \mathbf{T}_l} \frac{e_j}{p_j}$:

$$t_x \leq \frac{h_{|\mathbf{T}_l|}(t_x) s_k}{U_l \left(1 + \alpha_k (h_{|\mathbf{T}_l|}(t_x) - t_k)\right)}, \quad (15)$$

$$t_x \geq \frac{h_{|\mathbf{T}_l|}(t_x) s_k}{U_l \left(1 + \alpha_k (h_{|\mathbf{T}_l|}(t_x) - t_k)\right)} - \frac{\sum_{T_j \in \mathbf{T}_l} e_j}{U_l}. \quad (16)$$

For the sake of our aging-aware schedulability analysis, we are interested in t_x's lower bound of Eq. (16). By t_x's lower bound, we can be sure that T_i will still be able to meet its deadline. Finally, note that t_x can also be computed in an exact manner by iterating in Eq. (14) from the lower (Eq. (16)) to the upper bound (Eq. (15)) of t_x until $h_{|\mathbf{T}_l|}(t_x)$ reaches the value given by Eq. (11).

VI. Task Mapping

In many practical cases, it is necessary to satisfy a given lifetime requirement for our system. That is, we need to guarantee that the system will be fully operational within a given time period equal to t_{life}. Now, with this lifetime requirement, we search for task mappings that allow reducing the number of processors and, hence, reducing the overall cost of our system. Clearly, the task mapping technique we use for this problem needs to be aging-aware.

The problem of finding the minimum number of processors such that all timing constraints can be guaranteed is known to be NP-hard in the strong sense. That is, finding an optimal solution requires exponential complexity. If we now add a lifetime requirement to this problem, it becomes even more harder to solve.

In this paper, we propose an allocation algorithm which is based on the well-known First Fit (FF) heuristic. FF leads to a number of processors that is acceptably close to the optimum and it further has polynomial complexity. Our algorithm (Alg. 1) first sorts the tasks in $\mathbf{T}$ in order of decreasing priority (i.e., in the case of DM, according to increasing values of d_i). Then, it iterates over the sorted $\mathbf{T}$ and tries to allocate tasks in the minimum possible number of processors while observing the given lifetime requirement.

Algorithm 1 Aging-aware task mapping

```
Require: Set of all real-time tasks T
Require: Lifetime requirement t_life for the system
 1: num_processors = 1
 2: Sort T according to decreasing priority
 3: for i = 1 to n do
 4:     for l = 1 to num_processors do
 5:         if Schedulable(T_i, processors(l)) then
 6:             if Lifetime≥ t_life then
 7:                 Allocate T_i to processors(l)
 8:             end if
 9:         else if l==num_processors then
10:             num_processors = num_processors + 1
11:             if Schedulable(T_i, processors(num_processors)) then
12:                 if Lifetime≥ t_life then
13:                     Allocate T_i to processors(num_processors)
14:                 else
15:                     Return (no solution found)
16:                 end if
17:             end if
18:         end if
19:     end for
20: end for
21: Return (number_processors)
```

The algorithm we propose starts with only one processor and allocates tasks to it as long as they are schedulable and the lifetime requirement is met (line 5 and 6). A T_i is schedulable on a processor if it can meet its timing requirement d_i. To test whether the lifetime requirement is met, we need to account for aging while mapping tasks to processors. Towards this, there are two possible ways of proceeding.

First, given t_{life}, the aging model of Fig. 6 can be applied to compute the worst-case aging-dependent delay after $t_{stress} = t_{life}$ time. With this worst-case delay, one can estimate the speed degradation of the processor at t_{life}. Then, when a task T_i is mapped to a processor, the allocation algorithm performs a schedulability test considering the degraded processor's speed $s_{t_{life}}$ (the resulting speed of the processor after $t_{stress} = t_{life}$ in Fig. 6). If the schedulability test is successful, this means that the system can also meet all deadlines at t_{life} and T_i can be mapped to it. (This schedulability test can be performed using Eq. (6), where s_{lx} is replaced by $s_{t_{life}}$.) If the schedulability test fails on a given processor, then T_i needs to be mapped to another processor. This is the approach based on the naïve schedulability analysis discussed before.

The problem with the naïve approach is its pessimism, which leads to a more expensive design. That is also the reason for considering a more elaborate technique based on the schedulability analysis presented in Section V. Our second approach proceeds as follows: When a task T_i is to be mapped to a given processor, the allocation algorithm also performs a schedulability test. This time, s_{lx} is computed (Eq. (7), i.e., the processor's speed by which T_i still meets its deadline. Then, D_{lx} is computed Eq. (9) and with this $h_{|\mathbf{T}_l|}(t_x)$ Eq. (11) can also be computed, i.e., the cumulate execution demand that produces the aging leading to s_{lx}. Then, we can compute t_x's lower bound (right-hand side of Eq. (16). If t_x's lower bound

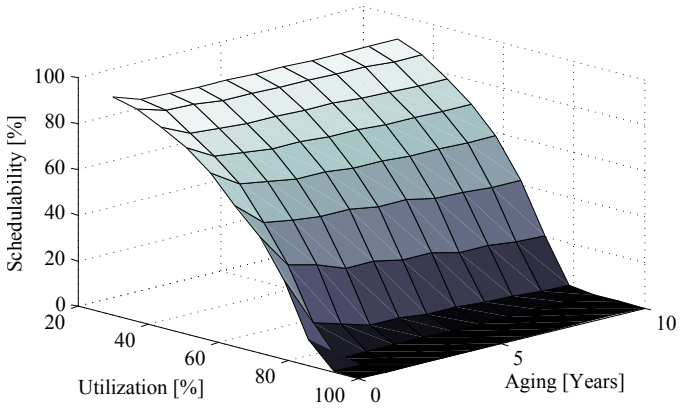


Fig. 7. Schedulability versus aging and utilization

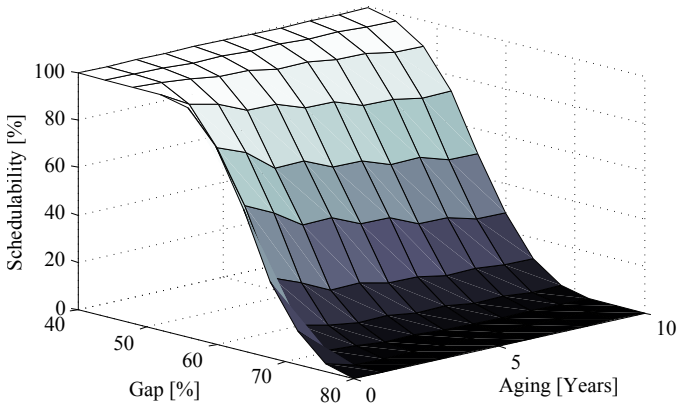


Fig. 8. Schedulability versus aging and gap

is greater than t_{life}, T_i can be mapped to the processor since the lifetime requirement is fulfilled. Otherwise, T_i needs to be sent to another processor where t_{life} can be met.

Our algorithm allocates all T_i onto one or more processors in the list of existing processors (line 4 to 9). If a T_i could not be scheduled on any of the exiting processors, it then adds a new processor to the list (line 10). The algorithm concludes when all T_i have been allocated and returns the number of processors that were necessary for accommodating all of them (line 21) while meeting the lifetime requirement. If any task T_i running alone on a processor is not capable of meeting the lifetime requirement, then the algorithm terminates with an error (line 15).

VII. Experimental Results

In this section we present and discuss some experimental results we conducted to illustrate the influence of aging on the schedulability.

Schedulability versus aging and utilization[1]**:** To obtain a schedulability curve with respect to aging and utilization, we generated 100,000 random sets of 10 tasks each. First, task utilizations u_i were obtained for a varying processor utilization [2], [3]. Then, we obtained periods p_i using a uniform distribution in $[0, 1]$ and computed $e_i = u_i p_i$. The relative deadlines d_i were also generated using a uniform distribution in $[e_i, p_i]$. Fig. 7 shows how the schedulability (i.e., the percentage of task sets that are schedulable) varies with aging and utilization of the processor. The higher the processor utilization the less the schedulability. In the same manner, as the processor gets aged, the percentage of schedulable task sets decreases. Note that the influence of aging becomes more relevant for high processor utilizations. With around 60% processor utilization, 70% of all task sets are schedulable when the system is *new*, whereas 60% are schedulable after 10 years. With 80% processor utilization, 30% of the task sets are schedulable in the new system, however, and only 2% are schedulable after 10 years. That is, with 60% processor utilization, the schedulability decreases by 10% after ten years of use. On the other hand, with 80% processor utilization, the schedulability decrease is around 30% after ten years.

[1]Corrected with respect to the version published at RTCSA'12.

TABLE I
Task set

$Task$	p_i (s)	d_i (s)	e_i (s)
T_1	0.0526	0.0055	0.0005
T_2	0.1073	0.0155	0.0077
T_3	0.3176	0.0257	0.0009
T_4	0.0771	0.0298	0.0161
T_5	0.3597	0.0301	0.0021

Schedulability versus aging and gap[1]**:** To analyze schedulability with respect to aging and *gap* (i.e., the difference between the period and the deadline of tasks), we again generated 100,000 random sets of 10 tasks each. First, task utilizations u_i were obtained for a constant processor utilization of 60% [2], [3]. Then, we computed periods p_i using a uniform distribution in $[0, 1]$ and computed $e_i = u_i p_i$. The relative deadlines d_i were generated according to a varying gap, i.e., $gap = p_i - d_i$. Fig. 8 illustrates schedulability of task sets with respect to the aging of the processor and the gap between periods and deadlines. Similar to the previous figure, the larger the gap, the less the schedulability of tasks. In addition, as the processor ages, the percentage of schedulable task sets decreases. Note that aging becomes more important for gaps between 50% and 70%. With a gap of 60%, 85% of the task sets are schedulable when the system is new, whereas only 33% of the task sets are schedulable after 5 years. That is, the schedulability decreases by around 50 in 5 years.

Task mapping and aging: Fig. 9 shows the performance of the two variants of our mapping algorithms (i.e., the naïve and the proposed approach) for the case of allocating the task set of Table I to processors. For a lifetime requirement of less than 3 years, both approaches are equally good. However, our proposed approach results in one processor less as the aging of processors progresses from 4 to 10 years.

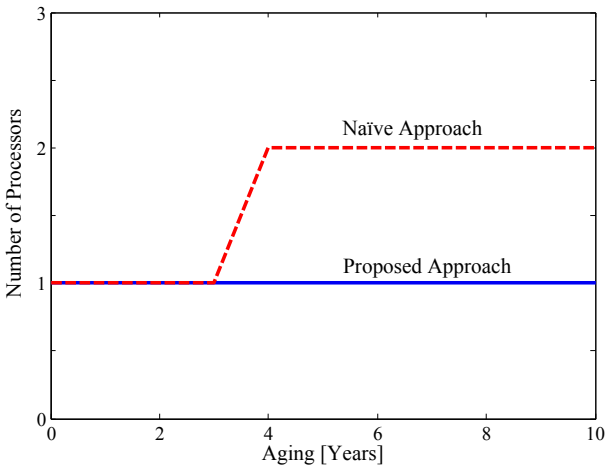


Fig. 9. Mapping results for task set of Table I

VIII. Concluding Remarks

As a processor ages, its transistors degrade and start requiring more time for switching. As a result, the error-free operation of the processor cannot be guaranteed at the same frequency (and/or supply voltage).

New upcoming processors such as IBM's POWER7 are able to reduce their clock frequency such that aging-dependent errors can be avoided. However, such technique, termed *automatic frequency scaling* (AFS), leads to a processing slowdown, which makes new design and analysis methods be necessary, in particular, for embedded and real-time systems.

In this paper, we have studied this problem in detail and extended the state-of-the-art in the area of real-time systems by presenting a schedulability analysis framework for such AFS-based processors.

Towards this, we introduced an aging model for processors and used it to make predictions about how the system ages and how this aging affects the schedulability of real-time tasks. We further presented a task mapping algorithm, which we showed to be beneficial in case a system needs to be designed taking lifetime constraints into account.

The main purpose of this work has been also to demonstrate the necessity of considering aging and other hardware-related issues when analyzing the schedulability of a system of real-time tasks. As future work, we plan to refine our aging model to allow for more accurate prediction of a processor's aging behavior.

Acknowledgments: This work was supported in parts by the German Research Foundation (DFG) as part of the priority program "Dependable Embedded Systems" (SPP 1500 - spp1500.itec.kit.edu).

References

[1] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.

[2] E. Bini and G. Buttazzo. Biasing effects in schedulability measures. In *Euromicro Conference on Real-Time Systems*, June 2004.

[3] E. Bini and G. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005.

[4] K. Bowman, J. Tschanz, C. Wilkerson, S.-L. Lu, T. Karnik, V. De, and S. Borkar. Circuit techniques for dynamic variation tolerance. In *Design Automation Conference (DAC)*, July 2009.

[5] A. Drake, R. Senger, H. Deogun, G. Carpenter, S. Ghiasi, T. Nguyen, N. James, M. Floyd, and V. Pokala. A distributed critical-path timing monitor for a 65nm high-performance microprocessor. In *International Solid-State Circuits Conference (ISSCC)*, Feb. 2007.

[6] M. Floyd, M. Allen-Ware, K. Rajamani, B. Brock, C. Lefurgy, A. Drake, L. Pesantez, T. Gloekler, J. Tierno, P. Bose, and A. Buyuktosunoglu. Introducing the adaptive energy management features of the POWER7 chip. *IEEE Micro*, 31(2):67–75, 2011.

[7] L. Huang, F. Yuan, and Q. Xu. Lifetime reliability-aware task allocation and scheduling for MPSoC platforms. In *Design, Automation, and Test in Europe (DATE)*, Apr. 2009.

[8] J. Keane, T.-H. Kim, X. Wang, and C. Kim. On-chip reliability monitors for measuring circuit degradation. *Microelectronics Reliability*, 50(8):1039–1053, 2010.

[9] C. Lefurgy, A. Drake, M. Floyd, M. Allen-Ware, B. Brock, J. Tierno, and J. Carter. Active management of timing guardband to save energy in POWER7. In *International Symposium on Microarchitecture (MICRO)*, Dec. 2011.

[10] J. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Real-Time Systems Symposium (RTSS)*, Dec. 1990.

[11] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: exact characterization and average case behavior. In *Real-Time Systems Symposium (RTSS)*, Dec. 1989.

[12] J. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2(4):237–250, 1982.

[13] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in hard real-time environments. *Journal of the Association for Computing Machinery*, 20(1):40–61, 1973.

[14] D. Lorenz, M. Barke, and U. Schlichtmann. Aging analysis at gate and macro cell level. In *International Conference on Computer-Aided Design (ICCAD)*, Nov. 2010.

[15] J. Park and J. Abraham. A fast, accurate and simple critical path monitor for improving energy-delay product in DVS systems. In *International Symposium on Low Power Electronics and Design (ISLPED)*, Aug. 2011.

[16] S. Rehman, M. Shafique, F. Kriebel, and J. Henkel. Reliable software for unreliable hardware: embedded code generation aiming at reliability. In *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, Oct. 2011.

[17] S. Rehman, M. Shafique, F. Kriebel, and J. Henkel. RAISE: Reliability-aware instruction scheduling for unreliable hardware. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan. 2012.

[18] S. S. Sapatnekar. Static timing analysis. In L. Scheffer, L. Lavagno, and G. Martin, editors, *EDA for IC implementation, circuit design, and process technology*. Taylor and Francis, 2006.

[19] V. Stojanovic, D. Markovic, B. Nikolic, M. Horowitz, and R. Brodersen. Energy-delay tradeoffs in combinational logic using gate sizing and supply voltage optimization. In *European Solid-State Circuits Conference (ESSCIRC)*, Sept. 2002.

[20] J. Tschanz, N.-S. Kim, S. Dighe, J. Howard, G. Ruhl, S. Vanga, S. Narendra, Y. Hoskote, H. Wilson, C. Lam, M. Shuman, C. Tokunaga, D. Somasekhar, S. Tang, D. Finan, T. Karnik, N. Borkar, N. Kurd, and V. De. Adaptive frequency and biasing techniques for tolerance to dynamic temperature-voltage variations and aging. In *International Solid-State Circuits Conference (ISSCC)*, Feb. 2007.

[21] C. Visweswariah, K. Ravindran, K. Kalafala, S. Walker, S. Narayan, D. Beece, J. Piaget, N. Venkateswaran, and J. Hemmet. First-order incremental block-based statistical timing analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(10), 2006.

[22] Y. Wang, X. Chen, W. Wang, V. Balakrishnan, Y. Cao, Y. Xie, and H. Yang. On the efficacy of input vector control to mitigate NBTI effects and leakage power. In *International Symposium on Quality Electronic Design (ISQED)*, Mar. 2009.

[23] K.-C. Wu and D. Marculescu. Aging-aware timing analysis and optimization considering path sensitization. In *Design, Automation, and Test in Europe (DATE)*, Mar. 2011.

[24] L. Zhang, L. Bai, R. Dick, L. Shang, and R. Joseph. Process variation characterization of chip-level multiprocessors. In *Design Automation Conference (DAC)*, July 2009.