computers

Review

DVFS and Its Architectural Simulation Models for Improving Energy Efficiency of Complex Embedded Systems in Early Design Phase

Parham Haririan

Department of Electrical Engineering (FB1), University of Bremen, Otto-Hahn-Allee 1, 28359 Bremen, Germany; haririan@uni-bremen.de

Received: 25 October 2019; Accepted: 2 January 2020; Published: 7 January 2020

Abstract: Dealing with resource constraints is an inevitable feature of embedded systems. Power and performance are the main concerns beside others. Pre-silicon analysis of power and performance in today's complex embedded designs is a big challenge. Although RTL (Register-Transfer Level) models are more precise and reliable, system-level modeling enables the power and performance analysis of complex and dense designs in the early design phase. Virtual prototypes of systems prepared through architectural simulation provide a means of evaluating non-existing systems with more flexibility and minimum cost. Efficient interplay between power and performance is a key feature within virtual platforms. This article focuses on dynamic voltage and frequency scaling (DVFS), which is a well-known system-level low-power design technique together with its more efficient implementations modeled through architectural simulation. With advent of new computing paradigms and modern application domains with strict resource demands, DVFS and its efficient hardware-managed solutions get even more highlighted. This is mainly because they can react faster to resource demands and thus reduce induced overhead. To that end, they entail an effective collaboration between software and hardware. A case review in the end wraps up the discussed topics.

Keywords: hardware-managed; embedded systems; architectural simulation; run-time power management; DVFS; hardware-software co-design

1. Introduction

The miniaturization trend in semiconductor technology has increased the number of transistors on a die. On the other hand, Moore's law has hit the power wall. Designers have innovated complex approaches to keep performance on its expected trend despite the power wall problem. However, these efforts have made the analysis of processors' power and performance a big challenge. As systems architectures get more complex, their power and performance analysis becomes more difficult. Success of a semiconductor product plan depends mainly on its power and performance quality. Violations of the specified constraints of these metrics might lead to expensive costs for suppliers. This gives impetus to designers to develop power management mechanisms that interplay with performance in a more efficient way. As a result, today, power management mechanisms (particularly DVFS) have been integrated into performance evaluation frameworks [1–3]. For decades, DVFS has been an unrivaled power management technique in the design of computing systems. Efforts to keep up with a transition to modern applications and techniques may raise the question of whether DVFS is still beneficial. Full realization of revolutionary computing paradigms such as quantum computing might change the rules and lead to a radical shift in energy-efficient design techniques, but within the current off-the-shelf technology, there is still room for improving the efficiency of DVFS to comply with today's demands. Section 2 provides a thorough review on low-power design techniques available for embedded systems with an emphasis on DVFS as a system-level architectural approach.

Power and performance analysis using system-level architectural methods provide a worthwhile possibility for designers to make design decisions in the early stages of design. It has a huge effect on the development time of semiconductor products. On the other hand, late detection of issues through more precise RTL and gate-level methods may lead to tremendous problems in the design process. Furthermore, RTL techniques can barely afford the highly escalated complexity of modern embedded systems [4]. Indeed, architectural simulators have been designed and developed to simplify the study of complicated systems in the early design phase. The design space can be easier explored at system level, and therefore, most of the design decisions can be made in the early stages of design. This prevents the distribution of bulk of issues to deeper abstraction levels in which detection and solving the issues are more costly. Exploiting system-level architectural techniques for high-level modeling of embedded systems is indeed a wise choice for systems architects, since it enables running benchmarks on a virtual prototype of the whole system as if they would run on real hardware. This leads to a quicker and cheaper analysis of results and solving design issues. System-level modeling and its methods are discussed in Section 3 of this article in more detail. That section contains the methods as well as available modern tools and frameworks for modeling embedded systems in general and hardware-managed mechanisms in particular.

Hardware-based low-power design techniques are interesting in that they can afford a prompt reaction to resource demands compared to software-based methods. They can fulfill task deadlines in a more efficient way than traditional software-based approaches [5]. The point is that low-power design mechanisms can be implemented in both software and hardware. As most of the embedded systems include timing constraints, hardware-assisted methods are of more interest for system architects. In particular, complex embedded systems with strict timing requirements (e.g., mixed-criticality systems [6]) can benefit even more from hardware-based solutions. For readers who are not familiar with mixed-criticality application domain, Section 4 presents a brief yet informative review highlighting the high potential for power savings in mixed-criticality systems using hardware-controlled low-power design techniques.

Section 5 puts the topics reviewed in previous sections together, using a case review. It reviews a case in which a virtual prototype of a hardware-assisted power management mechanism (addressed in Section 2) provides a means for analysis of the power and performance of an integrated system designed for handling embedded systems with strict timing demands (mixed-criticality systems reviewed in Section 4). The mixed-criticality application domain is an example of a complex application domain within the embedded systems world. The case review exhibits the necessity and usefulness of architectural simulation tools and techniques (reviewed in Section 3) in analysis of highly complex embedded systems and their non-functional requirements in the pre-silicon phase, well before manufacturing.

DVFS and its simulation models are a wide area of research, and many studies have been published about them. The topic can be reviewed at different levels of abstraction, including circuit level, gate level, and system level with different perspectives and even different simulation models and relevant tools specifically designed for them. The research area gets a bit narrower when we focus only on architectural/system level techniques for studying the power–performance trade-off to make design decisions in the early design stage.

Many researchers have shown interest in system-level techniques and have published many research works in recent years. However, when considering timing constraints besides power and performance, the research area and published works get still narrower. Indeed, the combination of DVFS and timing constraints with their hardware-based solutions modeled with architectural simulation yields a very narrow and particular area of research. Furthermore, power management for tasks with priorities together with their strict timing constraints (as in mixed-criticality systems) points to very few works that have been published. Examples are introduced later in corresponding sections. Mixed-criticality systems were first introduced in 2007 in their current conceptual meaning. This will be explained more with references in Section 4. There are a few fundamental works on this topic by itself that are introduced in that section. The combination of other topics such as power and performance issues, together with their hardware-assisted solutions to their induced overhead, make it still a unique research area that is highly demanded by modern industry (especially for automotive, aerospace, and medical applications).

Selection of cited references was not limited to any specific database. Instead, to the best of the authors' knowledge, most of the few main works and publications that were fundamental to other later works have been introduced and referenced in this manuscript. In addition, recent works that are mainly based on those fundamental works and add value to the topic have been introduced. A case study that refers to a very particular and unique work that combines DVFS, mixed criticality with strict timing requirements, and its hardware-managed solutions modeled with architectural simulation models wraps up the discussed topics in the end. The manuscript provides a useful comprehensive package for researchers who are interested in this field.

In a nutshell, the purpose of the manuscript is to study DVFS and its architectural simulation models with an emphasis on hardware-managed solutions to reduce dynamic power consumption of modern complex embedded systems with strict timing constraints, while retaining performance. In other words, the aim is to review early-stage mechanisms and tools with which modern embedded systems are capable of providing a better solution to resource demands including power, performance, and timeliness.

2. System-Level Low-Power Architectures

Power dissipation is not a challenge of only the modern complex architectures. It has been widely explored in the past decades. Since the time of early computing machines using vacuum tubes, researchers and engineers have been trying to find ways to decrease the power consumption of computing systems. Invention of transistors was the first big success toward their goal. Their effort continued all the time, and after several decades, further advances in semiconductor technology provided transistors with significantly fewer power dissipation. Today, embedded systems dissipate power in range of mW to a few Watts. With emergence of new computing paradigms such as edge computing and IoT (Internet-of-the-Things), energy efficiency of embedded devices has attracted even more attention. So, what are today's challenges for energy-efficient design of embedded systems?

2.1. Embedded Systems and Energy

Technology advancements have enhanced the performance of embedded systems. However, at the same time, processor clocks have hit the power wall. This has made the power management of embedded systems even more challenging in such a way that today, the concept of power-aware computing is more underlined than the past. In spite of general-purpose ordinary computing systems or supercomputers, embedded systems are designed and developed for a specific purpose. They cover a broad range of specific application domains, each having their own typical and particular requirements. As the requirements are very different, trade-offs should be considered regarding the design of such systems. Among many different requirements, the trade-off between power and performance is considered in this article.

Embedded systems are usually battery operated. That is, a limited amount of power budget is available to execute the tasks within their given timing limits. For instance, the power budget of general-purpose processors is in range of some hundred Watts [7]. Supercomputers spend from some tens of kW (top green supercomputers) up to a few MW of power [8]. This is while low-power mobile embedded systems have a power budget of around 1–2 Watts [7]. Even some ultra low-power embedded systems such as wearable computing systems need to consume only a few mW [9]. These examples highlight the necessity of power management in embedded systems. Compared to supercomputers in which performance dominates the importance of power consumption, energy is the primary concern for embedded systems. Lower energy consumption is advantageous in that it needs smaller power supplies, which in turn leads to reduced cost, area, and weight. However, significance of energy-efficient architectures is not limited to embedded systems. Supercomputer architectures also tend to exploit low-power embedded processors to achieve high performance while dissipating less energy and occupying less area [10,11].

Since resource constraints are always a concern in design of embedded systems, system architects try to achieve design goals while having an eye on the constraints. In most of the cases, one of the constraints needs to be traded off for another.

A good example for considering trade-offs in design of embedded systems architecture is real-time systems where the execution of tasks has timing constraints as well. The real-time application domain is a typical domain that is addressed with embedded systems. Although performance is not the major challenge of embedded systems (except for compute-intensive applications), real-time embedded systems often require a predictable performance up to the point that the deadline requirements are met. From that point on, high performance is not a concern any more [8]. This way, the power management mechanism within the system architecture can trade performance for energy.

Although power is the main concern in design of embedded architectures, compute-intensive embedded systems with high computational requirements have pushed the embedded systems industry toward high-performance architectures. Today, the ever-increasing high-performance power-efficient requirements have made the design of modern embedded architectures even more challenging. Low power consumption and high performance are challenging design demands because their techniques and solutions come into conflict with each other; one needs higher processor clocks, which translates into higher power consumption, whereas the other needs lower processor clocks, which leads to performance degradation [8]. Thus, modern embedded systems require novel hardware and software solutions to afford the ever-increasing power and performance demands.

Traditional low-power design techniques focus on power optimization of system components separately. However, embedded systems often consist of complex interacting components that are integrated on the same platform. This also verifies that novel power management systems are required for designing embedded systems.

High-performance power-efficient design techniques can be applied at different abstraction levels in systems architecture including software, hardware, and middleware. Current software–hardware co-design techniques can also be leveraged through architectural approaches to meet the design requirements. In such efficient embedded systems, often software or middleware functions are managed with hardware to provide a better support to meet the application requirements. A good example is dynamic voltage and frequency scaling (DVFS), which is a well-known run-time mechanism to control the power consumption. DVFS handles the run-time scaling of voltage and frequency up or down whenever performance requirements change. Although not a new method, DVFS is still an interesting mechanism for low-power designers. With an appropriate processor selection for an embedded application, a 10% degradation of performance due to frequency down-scaling has been shown to yield up to around 35% less power consumption [12]. In modern modeling and design tools, either DVFS support is an originally integrated part or it is added as an afterthought due to its necessity. Section 3.4.1 provides comprehensive information in this regard. Conventionally, DVFS is implemented as a middleware functionality within the operating system. However, some application fields such as the real-time domain (in which timeliness is essential) encourage the use of hardware-assisted software solutions to better meet their strict timing requirements.

2.2. Sources of Power Dissipation

To better understand the energy-efficient design techniques for embedded system-on-chips, it is helpful to first understand the sources of power dissipation in embedded systems in general, and then review state-of-the-art energy management techniques that can be exploited or enhanced to reduce energy dissipation.

Two well-known categories of CMOS power consumption are dynamic power and static (leakage) power. Dynamic power consumption dominates the static leakage power dissipation and is calculated by the well-known CV^2f equation where supply voltage V has a significant impact on it, mainly because of its quadratic relationship with dynamic power. In addition, clock frequency f has a high impact on dynamic power, as it is the number of repetitions of voltage waveform per second.

In fact, there is a direct relation between them. That means that lower frequency leads to lower supply voltage and consequently much lower dynamic power consumption.

Alongside dynamic power dissipation, leakage power dissipation is another source of power dissipation in CMOS technology. Although leakage power is not negligible, in this article, the focus is on the design techniques to lessen the dominating dynamic power consumption of embedded System-on-Chips (SoCs).

2.3. Energy-Efficient Design Techniques

After we know the fundamental sources of power dissipation in digital system-on-chips, we can better understand energy-efficient design techniques. Today's embedded computing systems demand higher processing and communication capabilities that in turn require higher energy efficiency. To handle this challenge, power management techniques need to be applied to all design abstraction levels ranging from circuit level to architecture/system level. Let's interpret the same meaning when we read the phrases 'architectural' or 'system-level' views and techniques in the entire this article.

The focus of this article on designing energy-efficient embedded SoCs is only on architectural techniques. The rationale behind this, is that considering power issues in higher abstraction levels of top–down design process provides the possibility for designers to consider power budget as well as power management techniques and required design changes in the early design stage. This prevents late-stage higher costs and lets the designers have a better power–performance trade-off analysis to meet the desired goals for achieving a power-efficient design. The results will be useful later on, during the bottom–up implementation process.

In this article, architectural low-power design techniques are classified into the following categories:

A Microarchitectural techniques
  i.e., power saving in internal parts of processor(s) (e.g., cache, branch predictor, etc.)

B Architectural techniques, also known as system-level techniques
  (a) With run-time changes, also known as dynamic techniques
    i. DVFS
    ii. Turning off unused components, also known as gating techniques
  (b) Without run-time changes, also known as structural techniques
    i.e., power saving in design layout (e.g., heterogeneous architecture, 3D architecture, etc.)
  (c) Run-time techniques in conjunction with (fixed) structural techniques (e.g., dynamic heterogeneous architecture)

Both DVFS and turn-off (category B-a) are dynamic architectural methods. Between the two, the turn-off method takes time to shut down power and then takes still time to turn it on when it is again needed. For applications with critical timing constraints, this is not a solution to save energy.

Combined solutions (category B-c) consist of separate high-performance processing units and low-power processing units being selected at the run time of applications. An example of such innovative solutions is Arm's big.LITTLE technology [13]. In a high level of abstraction view, on the peak operating (or threshold) point of a power-efficient in-order 'LITTLE' core, the manager (or scheduler) handles the on-the-fly migration (or mapping) of task execution from the 'LITTLE' core to a performance-efficient out-of-order 'big' core with higher operating points (and back) whenever performance requirements change. The big.LITTLE architecture exploits and cooperates with DVFS and turn-off mechanisms in both of its migration-based and scheduler-based software models [14].

Host/PiM (Processing-in-Memory) configuration is another example of heterogeneous architectural low-power design techniques in which a PiM device (optimized for bandwidth-intensive workloads) works along with a host processing unit optimized for compute-intensive workloads. The heterogeneous host/PiM configuration also can benefit run-time approaches such as DVFS to offer improved energy efficiency. A recent study reports 7x improvement [15].

Although DVFS has been studied and used in designs for decades, attempts to design more efficient DVFS mechanisms would even strengthen the modern techniques and heterogeneous solutions in which harmonized cooperation with DVFS is a key.

This article focuses more specifically on DVFS based on a hardware-managed scheme to manage the power consumption of timing-constrained embedded systems. In addition, it reviews architectural simulation tools and mechanisms to handle design decisions and issues in the early design phase. Embedded SoCs usually incorporate several internal and external elements (processors, memories, buses, I/O, etc.) that affect their power consumption. Each of these elements needs their own specific tools and techniques for modeling and studying their power consumption. This manuscript focuses only on dynamically handling the voltage and frequency of the processing cores as the most power hungry elements within SoCs. Importance of the other techniques may not be neglected though.

2.3.1. DVFS

DVFS has been used in many processors as the main power management technique. Famous processor vendors such as Intel and AMD provided commercial products with DVFS capability. Examples are AMD's Bulldozer processor [16] or Intel's Multicore processors with Turbo Boost technology [17]. In addition, almost every smartphone uses DVFS mechanism to manage power consumption. The use of DVFS in real systems has empirically proven its practical importance and high impact on power and energy saving. Furthermore, the emergence of new computing paradigms and application domains even encourages the use of DVFS in modern embedded devices and systems more than the past [18–21].

Here, the focus is on architectural aspects in design and implementation of DVFS and the interplay between software and hardware to achieve energy-efficient embedded systems. Using the DVFS method, processors' supply voltage and operational frequency are adjusted to regulate their power and performance. With this energy minimization technique, voltage and frequency of processors can be dynamically changed at the run time of applications. The key feature of DVFS is that it leverages the power–performance trade-off [22]. This way, applications' temporal performance requirements can be exploited to save power consumption and reduce energy dissipation. DVFS benefits from those application phases, parts of code, or task periods in which lower performance does not hurt the proper functionality of the system. A typical example is the memory-bound applications or parts of programs [23]. Since processor(s) and memory are asynchronous to each other, usually the processor is stalled during memory operations. During this time, the processor's frequency can be scaled down to save energy without losing significant performance. The authors in [24] propose a DVFS mechanism to scale down the clock frequency in memory-bound parts of programs with minimum performance degradation.

In some embedded systems, missing task deadlines is tolerated. An example is the multimedia application domain where missing some of the deadlines can be acceptable, since it does not hurt the quality. The authors of [25] propose a method to save energy in multimedia embedded systems using DVFS. They propose an algorithm to drop some tasks to create slacks. DVFS uses these slacks to save power.

However, the case of using DVFS to save energy in real-time applications is tricky. Real-time embedded systems impose a strict stipulation on DVFS to protect timing constraints while adjusting their operational frequency and supply voltage. If a task deadline is close, DVFS can scale up the clock frequency to meet deadline; otherwise, a lower supply voltage and clock frequency would be enough to save energy while meeting deadline. The researchers in [26] propose a task mapping and scheduling algorithm for multicore embedded systems to minimize power consumption using DVFS while meeting deadlines. Majority of architectural techniques using DVFS for real-time systems concentrate their effort mainly into exploiting idle times (slacks).

2.3.2. Design Considerations When Using DVFS

Similar to other mechanisms, applying DVFS into designs requires several considerations. One of the important design considerations is the DVFS control policy. Scaling decisions are taken with DVFS governor (controller) based on program's behavior or system load. The DVFS mechanism tries to stretch the task execution times to eliminate idle periods via scaling by means of a specific control policy algorithm. This control policy can be implemented in either software or hardware. Implementation method of DVFS control policy exerts an influence on DVFS overhead.

DVFS is a very efficient energy-saving approach. However, its imposed overhead also needs to be considered by system architects. Beside the DVFS control policy, another source of DVFS overhead is its transition delay. The transition time is the delay between two levels of voltage or frequency during scaling. The time that the scaling process requires should be considered. Depending on the amount of delay, simple or complex implementation techniques are required. For applications with strict timing constraints, the transition delay might have a negative effect; it may increase the execution time of tasks and lead to deadline misses.

The researchers in [27] provide an architectural solution to solve the DVFS transition delay. They use on-chip voltage regulators to achieve a lower transition latency. They report possible voltage transition delay at a rate of tens of nanoseconds, which is a huge gain compared to microsecond-scaled off-chip regulators. They also report a possible efficiency drop in voltage conversion process that leads to higher energy losses, which is a disadvantage of their solution, but quicker scaling and fine-grained control of the voltage and frequency pair compensates the pitfall.

Another architectural design consideration regards the use of DVFS on multicore. There are two approaches to apply DVFS on multicore embedded systems. Architectural works on DVFS for multicore focus on either simultaneous scaling of all the cores (global DVFS) or scaling each individual core (per-core DVFS). Voltage and frequency of each on-chip core is separately scalable in the latter configuration. Some experiments [27,28] have shown that per-core DVFS provides higher improvement compared to the global DVFS scheme. It also [29] states that it maximizes the effectiveness of DVFS if each core has its own supply voltage and frequency.

2.3.3. Hardware-Managed DVFS

Success of DVFS in the research community as well as industry added more interest to put further efforts into research on more effective DVFS techniques. Almost all architectural works for improving DVFS management techniques are implemented as part of operating systems (OS-managed scheme). However, novel architectures exploit hardware-assisted mechanisms to manage resources [30–32]. DVFS does not limit any of the hardware techniques of SoC designers. It even makes it more attractive if DVFS management is assisted with hardware-based control. The reason is that hardware can react faster to power demands. A hardware-managed mechanism can make the idle intervals shorter, thus leading to more power saving. Moreover, hardware-dedicated solutions can better fulfill the task deadlines than the classical software-based solutions at the kernel level [5]. These are attractive features for systems with strict timing constraints. They entice system architects to choose hardware-based DVFS mechanisms for their designs.

It is important to note that in hardware-managed DVFS mechanisms, although hardware is emphasized, the role of software is not negligible. Hardware needs to be aware of the actual resource demands of the application(s) at run time to provide effective scaling decisions. In other words, hardware-based DVFS approaches entail an effective collaboration between software (applications) and hardware. In this fashion, applications inform hardware at run time about the resources that they need to complete the tasks. Then, hardware acts promptly to support the requested requirements. The result is a faster power management process with reduced latency that leads to greater energy efficiency and timeliness.

3. Modeling Hardware-Managed DVFS

Complexity of today's computing systems makes it necessary to exploit power and performance modeling techniques to produce accurate results and best drive the design project in the right direction. Embedded systems' design and development has a close relation to power and performance modeling and analysis. Designers need tools to evaluate their ideas and explore the design space of non-existing components or systems before building costly physical systems. Today, components with complex functionality have been integrated in modern computing systems. To reduce time and cost, it is beneficial to evaluate the design before manufacturing. A controlled environment that facilitates evaluation and debugging in the early design stage is very helpful for system-level designers.

In this section, an overview of system-level/architectural performance modeling techniques is presented. Integration of power management techniques to study and analyze the interplay between power and performance is a necessary feature in modern embedded systems. The section continues with an exploration of available modern integrated tools and their applications.

3.1. System-Level Performance Modeling Techniques

In design of embedded systems, performance can be sacrificed for power (and other features) up to a limit. So, a system designer considers energy efficiency and other necessary design features while having an eye on performance. Especially in case of battery-operated embedded applications, it is more important to consider the power and performance trade-offs.

Accurate modeling and performance analysis helps designers make right design decisions. For that, they need to design a baseline system first in order to examine their ideas and analyze the results. In order to make right decisions in systems design, it is necessary to have an accurate power and performance model of the baseline design. Choosing a modeling approach and applying proper benchmarks are the next steps. As there are several modeling approaches, an appropriate modeling method needs to be chosen. After modeling the baseline system in a way that best describes the features and requirements, proper benchmark applications are applied to the prepared model to produce the output results. In the end, the results should be analyzed for final deduction.

Analytical modeling, virtual prototyping (through simulation), and hardware prototyping are three major system-level performance modeling techniques that help system designers evaluate their design ideas [33,34].

A hardware prototype demonstrates a working implementation of ideas, but it obviously introduces a high cost for evaluating complex designs. Development of hardware prototypes is costly and also time-consuming.

Analytical modeling is a mathematical description of a process that does not consider implementation issues. With analytical modeling, system's behavior is modeled using mathematical equations. Analytical modeling employs less effort to model a complex system [35], but it does not refer to design decisions and implementation issues that a designer may encounter. It involves assumptions and simplifications that yield quick results. These assumptions reduce the accuracy of the model. Accuracy in this context means how close a model is to the real hardware that it represents. Analytical modeling does not give a clear understanding of what may happen in implementation of the system.

Preparing virtual prototypes through architectural simulation is a better alternative that provides a balance among all those determining design factors such as flexibility, extensibility, complexity and costs in pre-silicon design phase, long before manufacturing the system-on-chips. Through simulation, system can be modeled with more implementation details compared to analytical modeling. Although simulation takes longer, it acts as a virtual prototype of the system under design, and hence produces more accurate results to be analyzed. Table 1 exposes a comparison of system-level modeling techniques from various aspects.

Table 1. Comparison of system-level performance modeling techniques

Modeling | Aspect	**Accuracy**	**Speed**	**Complexity**	**Flexibility**	**Extensibility**	**Details**	**Cost**
Analytical modeling	low	high	low	high	high	low	low
Virtual prototyping	moderate	moderate	moderate	moderate	moderate	moderate	moderate
Hardware prototyping	high	low	high	low	low	high	high

Today, manufacturers try to reduce time to market and optimize products to higher levels of performance and power efficiency. Virtual prototypes are useful in that they facilitate pre-silicon power–performance analysis. That means, a system that does not yet exist can be studied and analyzed using virtual prototypes. This is a great possibility for designers, as they can quickly explore the design space without spending time and manufacturing costs that are needed to design hardware prototypes. It also reduces the time-to-market of products, which is an essential need for industry. Especially in case of MPSoCs with their high complexity, simulation plays a determining role in decreasing the development time and manufacturing costs.

Simulation results provide feedback by statically analyzing the results. This feedback can be exploited to optimize and improve the design. It makes it possible to modify or debug the design before fabricating a physical prototype. In addition to that, it makes early software development possible before the hardware is ready, thus significantly decreasing the system's development time.

To sum up, both analytical modeling and simulation are important performance modeling approaches in system-level analysis of embedded systems. Each approach has its advantages and disadvantages. Generally speaking, analytical modeling is theoretical, while simulation gives insights into practical implementations. Although analytical modeling can help explore the complex and huge design space quicker, eventually, architectural simulation is needed to face and solve implementation issues. Designers may combine the two modeling methods to analyze systems and benefit the good of the two worlds.

A good example to show how analytical modeling helps explore large and complex design spaces is modeling the power and performance of mixed-criticality multicore systems. This analysis can later be studied in more detail through architectural simulation.

3.2. Architectural Simulation

Architectural simulation solves the issues with both analytical modeling and hardware prototyping through providing a virtual prototype. In addition, it balances cost and accuracy.

Simulator is indeed a virtual model of systems architecture that mimics its behavior. Simulator is a very flexible tool to explore a variety of architectural design spaces. This is a valuable feature that enables designers to evaluate different ideas without having to make physical prototypes. In addition, it allows the designers to study the functionality and performance of non-existing systems through modeling new features before building real hardware.

Architectural simulator gets benchmarks as its input, executes on the simulated system, and generates performance results. A benchmark is a set of applications that runs on top of the simulated system to assess various architectural features of the system based on specific metrics.

Although simulation affects the time consumed and effort spent for both developing and evaluating design ideas, the (possibly) lacking features need to be added. A good example is integrating low-power design mechanisms within an architectural simulator. Since power consumption is a critical design constraint in today's complex integrated systems, building a simulation platform that unifies power and performance design aspects might entail considerable costs and time. However, to cope with the new challenges of modern intricate architectures, integrating run-time power management mechanisms such as DVFS within the architectural simulation tools has recently been a hot solution.

Individual simulators exploit different techniques to cover the simulation of systems architecture. The following section describes several aspects in which architectural simulators can be categorized.

3.3. Classification of Architectural Simulation Techniques

Architectural simulation tools can be categorized based on different perspectives that are reviewed in this section [33,34].

3.3.1. Modeling Scope

Simulation of a computing system can refer to either individual components or the entire system. Microarchitecture simulators provide a simplified model and focus on simulating the sole processor's microarchitecture and its internal components such as branch predictor, cache etc., ignoring other system components.

Full-system simulators cover a wide scope, modeling the entire computer system (system-level modeling including hardware, software, OS, and peripheral devices). Full-system simulators can boot and run an operating system, and simulate its interactions with workloads and the whole system. A simulated system using a full-system simulator appears to the user as a virtual prototype of a real target system. In other words, the user of a full-system simulator can suppose that applications are running on real hardware. Some widely used full-system simulators are Wind River Simics [36] which is a commercial product, and gem5 [37] which is used in academic research.

3.3.2. Modeling Details

Another category of simulators is based on the amount of details that they can provide. Functional (instruction-set) simulators model the functionality of processor instructions without microarchitectural details. An instruction-set (functional) simulator only models the execution of a program on a processor without timing information. In this model, input values are taken, instructions are simulated, and output values are generated. Since no detailed features are modeled with functional simulators, their simulation time is short. SPIM [38] is an example of a functional simulator.

In contrast to functional simulators, cycle-accurate (CA) simulators keep track of timing information and care about the microarchitectural details. Cycle-accurate simulators simulate the execution of a program on a cycle-by-cycle basis. Since cycle-accurate simulators provide detailed insight into the target architecture, they are slower than functional simulators. Cycle-accurate simulators provide performance results, which makes them suitable for performance analysis of different designs. Examples are gem5's FS mode and SystemC [39].

3.3.3. Execution of Program Instructions

An architectural simulator can either execute an unmodified program itself or traces of a program that are collected from its execution on a real processor.

The process of trace-driven simulation is two-fold; first, the simulator logs the actual data from a real experiment (or a reference simulation). Then, the simulator loads and runs the recorded data to produce performance results. During the logging phase, a benchmark application runs on a real processor, and the architectural state of the processor(s) including memory references, specific instructions, and system calls are logged into a so-called trace file. This requires a huge space for storage. In the next phase, the recorded trace files are given as input to the simulator. The trace-driven simulator loads the architectural state of the processor(s) and simulates the instructions. Trace-driven simulation results are reproducible, since the input trace file is fixed.

Execution-driven simulation is the actual simulation approach. An execution-driven simulator executes the instructions of a program one by one while updating the whole architectural status of the system components, which in turn makes it more accurate and also slow. In trace-driven simulation, this status update has already been logged during the recording phase, and the simulator does not spend time on it. Therefore, execution-driven simulation is usually slower, but does not need that much storage space as trace-driven simulation. In addition to that, input is fixed in trace-driven simulation, which leads to reproducible results. However, different programs without any modifications can be executed with an execution-driven simulator, making it more flexible and realistic.

Although the accuracy of execution-driven simulation is higher than trace-driven simulation, it takes more time to evaluate the same design. A trace-driven simulation needs bulk of storage to store the traces. Table 2 outlines a comparison of the two schemes from different aspects.

Table 2. Comparison of execution-driven and trace-driven simulation schemes

Aspects	Accuracy	Speed	Space	Realistic	Reproducible
Execution-driven	better	slow	small	more	no
Trace-driven	good	faster	big	less	yes

3.4. Simulation Platforms

Alongside commercial products, there are many free architectural simulators out there that are available for system-level research. Each of these simulation platforms covers different modeling aspects regarding simulation perspectives.

For using architectural simulators, one needs to consider the trade-off between accuracy and simulation speed. It is hard to find a simulator that supports both of these goals at the same platform, because in most of the cases, they conflict. Designers need to choose the right simulator(s) carefully depending on components or system that they simulate.

Architectural simulators do not possess all those capabilities that a researcher might need to evaluate new ideas. Sometimes, researchers try to couple different simulators to benefit the full interoperability between them. For instance, the researchers in [40] couple two simulation platforms—SystemC and gem5—that have been more popular in research in recent years. This way, their work enables more possibilities in design space exploration of embedded systems, as it utilizes the benefits of the both simulators to provide accuracy with higher simulation speed.

It is even very likely that a necessary feature is not available. In such a case, to fulfill the design goals and evaluate new ideas that need capabilities out of the box, researchers may need to either build a new simulator on an existing framework that lacks that feature, or extend an existing one to enable the required feature. For example, Flexus [41] is built on Micro Architectural Interface (MAI) of Simics to enable cycle-accurate simulations that Simics does not support. The works [1] and [3] added an in-situ power management support to gem5 as an afterthought. The work in [2] did the same for SystemC.

3.4.1. DVFS Support within Simulation Platforms

In the past, power management mechanisms were not available within performance simulation tools. As power aspects of computing systems are more stressed recently, researchers have tried to integrate power management mechanisms within the architectural simulators. Popular modern simulators either support DVFS originally, or researchers have extended them to support DVFS. The modular structure of architectural simulators makes it possible to apply needed modifications and add new features.

Table 3 exhibits different perspectives of various available architectural simulators with possibly their extended features. The simulators are listed in ascending order (1996–2013, date of their first launch or approval) and imply that recently, power management mechanisms (particularly DVFS) have been integrated into performance evaluation frameworks either natively or as an afterthought due to the recent demands.

Among available contemporary architectural simulators, ESESC [42] and Snipersim [43] originally support OS-based DVFS; ESESC is a fast execution-driven Arm simulator and is an enhanced version of SESC. Snipersim is a fast X86 simulator that trades simulation speed for accuracy. Its validation against Intel Core 2 provides average performance prediction errors of 25% at a simulation speed in range of some MIPS.

Table 3. Selected multicore architectural simulation platforms and their features

Perspective	**Workload**		**Modeling Details**		**Modeling Scope**		**DVFS Support**	
Simulator | Category	**Execution-Driven**	**Trace-Driven**	**Cycle-Accurate**	**Functional (Instr-Set)**	**Microarch (Bare-Bone)**	**Full-System**	**OS-Managed**	**Hardware-Managed**
SMTSIM ('96)	Y	N	Y	N	Y	N	N	N
SimpleScalar('02)	Y	Y	sim-outorder	sim-fast	Y	N	N	N
Simics ('02)	Y	Y	N *	Y	Y	Y	N	N
SESC ('05)	Y **	N	Y	N	Y	N	N	N
SPIM ('11)	Y	N	N	Y	Y	N	N	N
SystemC [39]	Y **	N	Y	Y	Y	N	Y [2]	N
gem5 [37]	Y	Y	FS mode	SE mode	SE mode	FS mode	Y [1]	Y [3]
Sniper [43]	Y	Y	Interval-based		Y	N	Y	N
ESESC [42]	Y	N	Y	Y	Y	N	Y	N

* Micro Architectural Interface (MAI) was designed but is not supported anymore. Flexus is built upon Simics' MAI to enable cycle-accurate simulations. ** Execution-driven with events.

gem5 is another architectural simulator that is also widely used in research. It supports full-system simulation and supports several ISAs (Instruction-Set Architecture). Very early versions of gem5 did not support any power and energy experiments. Recently, OS-based power and clock domains have been added to gem5 [1]. The work extends gem5 to support DVFS in a conventional kernel-based manner. It considers multiple clock domains to support per-core frequency adjustments. The approach in [1] enables experiments in power and energy-efficiency with gem5. However, it focuses on classical OS-managed DVFS mechanism using OS kernel utilities. Applications with hard timing constraints might need a hardware-controlled DVFS mechanism to fulfill their timing requirements in a more effective way.

The researchers in [3] propose an approach for simulation of a hardware-managed DVFS mechanism in gem5. With this method, a custom hardware exclusively controls the scaling mechanism of DVFS, thus reducing part of its delay overhead. The method exploits run-time reconfiguration capabilities of gem5 to simulate the dynamic feature of DVFS. It applies extensions to gem5 to incorporate additional pseudo-instructions and utility functions to help simulate hardware-controlled voltage and frequency scaling. Instead of typical periodic checking of the OS status by DVFS governor in classical OS-managed DVFS, the proposed hardware-based DVFS technique utilizes application performance signals to make the hardware aware of the resource demands of applications. Performance signals are assigned to measure given applications' performance trend, providing a performance pattern, using their performance signal intervals. Thus, they help handle the performance demands of applications. To model application performance signals, the work in [3] uses an adapted version of the application heartbeats approach [44]. In their proposed method, the emitted signals inform the DVFS manager (which is modeled as custom hardware) about the application's status to better handle the performance constraints. In contrast to the work conducted by [1], operating system is not modified in this approach. From a bird's eye view, the solution consists of a hardware–software co-design mechanism by means of an architectural template in which the applications directly communicate with custom hardware to provide a high-performance energy-efficient design in a multicore environment. This mechanism considers the interplay between custom hardware components and embedded software. It aims to integrate the two aspects, to strike a balance between flexibility and efficiency within systems architecture. Figure 1 presents the simulation flow of this work.

The work validates its proposed DVFS technique with several experimental scenarios. It mainly aims to develop a simulation framework for design space exploration of timing-constrained energy-efficient systems by means of a hardware-based run-time DVFS management technique to further leverage the trade-off between power and performance.

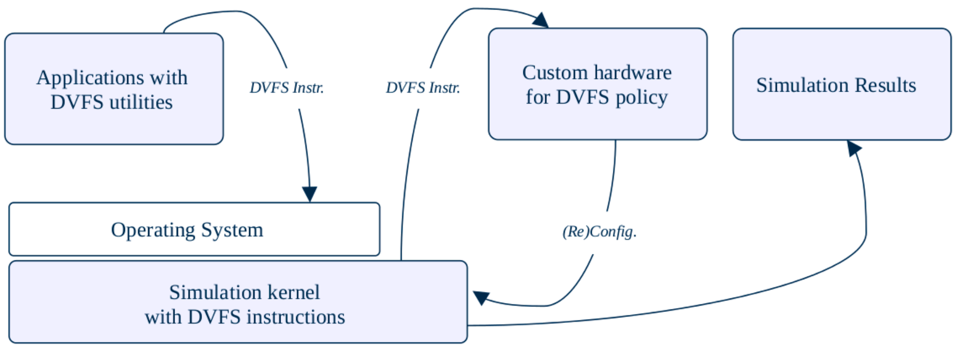
Applications with DVFS utilities
DVFS Instr.
DVFS Instr.
Custom hardware for DVFS policy
Simulation Results
Operating System
(Re)Config.
Simulation kernel with DVFS instructions

Figure 1. Hardware-assisted DVFS simulation mechanism proposed by [3].

The shrinking trend of sizes in CMOS technology has already added complexity to the analysis of power and performance. Furthermore, running real-time tasks with various criticality levels on a common platform imposes even more complexity. Section 5 reviews a case in which complex embedded systems—running mixed-criticality tasks—are explored via a virtual framework equipped with a run-time power management mechanism. The work in [45] exploits the simulation framework proposed by [3] and extends it to demonstrate a hardware-controlled run-time energy management technique for mixed-criticality tasks. It reports notable power savings using its method.

As the mixed-criticality concept is new and complex, first, a review is provided in the next section. Most of the works done on the mixed-criticality domain is on analytical and algorithmic modeling. After a review on the analytical models of mixed-criticality systems in the next section, Section 5 explains how applying the hardware-assisted DVFS mechanism to mixed-criticality applications through architectural simulation provides a means to study the energy-efficiency problem in this complicated application domain.

4. Mixed-Criticality Applications

This section presents a brief review on mixed-criticality applications and outlines their task models, scheduling algorithms, and research attempts that aimed to provide energy efficiency for mixed-criticality systems. Examples given in this section try to give a big picture of the problem of low-power design for mixed-criticality systems as well as their candidate solutions. The goal of this section is only to introduce the key points of the mixed-criticality systems concept and provide sufficient knowledge to understand the case review in Section 5.

It is easier to read and follow this section if one is already familiar with mixed-criticality systems and their features. Readers who intend to gain in-depth knowledge and comprehensive information on the topic may refer to [6] as well as other provided references.

4.1. Overview of the Mixed-Criticality Concept

In real-time embedded systems, a task that passes its deadline may produce failure. The possibility and level of this failure determine a criticality level that can be given to the task. The criticality level of a task determines the procedures needed to handle the situation.

To provide efficient use of resources, a complex real-time embedded system may assign several criticality levels to different tasks—with possibly shared resources—integrated into one system to provide assurance against failure [6]. Using components with different criticalities on the same computing platform is a complex work. The outcome is a so-called mixed-criticality system. Reference [6] summarizes the topic in this way: "Criticality is a designation of the level of assurance against failure needed for a system component. A mixed-criticality system is one that has two or more distinct levels."

In other words, integration of functions with different safety requirements into a common platform yields a mixed-criticality system. A mixed-criticality system has at least two distinct levels (for example, high-critical, low-critical, etc.). Some domain standards define up to five levels (e.g., Automotive ISO-26262, Aviation DO-178C) [6].

As the topic is new, not all standards and papers refer to the phrase with the same meaning. This article discriminates between the systems with completely isolated components with different criticality levels (multi-criticality) and the mixed-criticality systems in that the former do not share resources. In a mixed-criticality system, resources are supposed to be shared among components with different criticalities, but in an efficient way. However, it increases the complexity in this domain.

The high complexity of systems in fields such as aerospace, automotive, or railway introduce unfeasible development time and costs. To solve this, a rising trend is integrating functions with varying criticality levels within the same platform, which translates into increasing the utilization of mixed-criticality systems in modern industry. The main objective is to enhance the total weight, area, and power consumption. The challenge is to use shared resources in an efficient way with timing and safety guarantees. Modern standards implicitly impel designers to consider mixed-criticality issues within their platforms. Examples are AUTOSAR [46] and ARINC [47] standards for automotive and aerospace systems, respectively.

4.2. Research on Mixed Criticality

Research on the mixed-criticality domain has basically focused on providing timing guarantees for tasks with different criticality levels. This is mainly achieved by dropping non-critical tasks in favor of critical ones. There are recent attempts to improve QoS (Quality of Service) beside timing guarantees to keep non-critical tasks still running.

The underlying research question is how to share resource usage in an efficient way with timing and safety guarantees. This question gives rise to the design and implementation of specific run-time mechanisms in both software and hardware. Management of mixed-criticality systems is an intrinsically run-time problem because the information needed to make optimal decisions is only obtained at run time [48].

4.3. Mixed-Criticality Task Model

Although for a given non-critical real-time task, the worst-case execution time (WCET) is constant, it is very dependent on task criticality in a mixed-criticality system. There is a common model for mixed criticality systems in the literature [49–51] in which the WCET of each task in all existing criticality levels is modeled. WCET is usually a pessimistic estimation, and in this model, the WCET of a task with higher criticality is even more pessimistic, i.e., the pessimism in estimating the WCET depends on the criticality level of a task. On the other hand, the high pessimism of WCET estimations leads to the inefficient usage of resources at run time. To solve this problem, the mixed-criticality task model provides a dynamic guarantee for resource efficiency of mixed -criticality systems. The policy ensures that whenever a task exceeds its deadline, other tasks with a lower criticality level will be dropped at run time to guarantee the resource accesses of tasks with higher criticality level. This guarantee covers only those tasks with higher criticality level thereafter.

The work in [52] adopts the dual-criticality sporadic implicit-deadline task model [51,53] and extends it to multicore. Dual-criticality refers to two levels of criticality that are considered for each task: high (*HI*) and low (*LO*). Tasks with the criticality level *HI* are required for a safe system operation, while tasks with the criticality level *LO* are not. Using a mixed-criticality scheduler, the system needs to guarantee that the critical tasks meet their timing constraints in any scenario. For that, the worst-case execution time (WCET) of each task is modeled on both criticality levels (*HI-WCET*, *LO-WCET*).

A *sporadic* task model [54] consists of n tasks characterized by three parameters: an execution time C, a deadline D, and a separation time T for each task. Two successive arrivals of task requests need to be separated by a minimum of T time units. It also holds that $C \leq D$ and $C \leq T$ for each task. In *implicit-deadline* task model, the condition $D = T$ holds for every task.

State-of-the-art mixed-criticality schedulers [52,55] use a dynamic approach that consists of two operation modes. In low-criticality mode (*LO-mode*), the system guarantees a safe operation of all tasks (*LO* and *HI*), provided that they fulfill their (non-strictly) estimated WCETs: the so-called *LO-WCET*.

In the high-criticality mode (*HI-mode*), the system guarantees the safe operation of the *HI-tasks*, even considering the most pessimistic WCETs (*HI-WCET*). Normally, the system runs in *LO-mode*. In the rare case that the actual execution of a *HI-task* overruns its *LO-WCET*, the system terminates all the *LO-task*s and switches to the *HI-mode*. This dynamic approach with two modes decreases the hardware resources needed to guarantee the mixed-criticality schedulability but demands measuring the actual execution time of *HI-tasks* at run time.

According to the mixed-criticality task model above, the dynamic guarantee given to the tasks at their run-time is based on their estimated WCETs. The researchers in [52] introduce a *mode switch protocol* to specify this guarantee: The system is in low-criticality mode (*LO-mode*) when it starts running. In *LO-mode*, there is a guarantee that tasks that do not exceed their *LO-WCET* will meet their deadlines. In case a *HI-task* (task with high criticality) exceeds its *LO-WCET,* then the system switches immediately to *HI-mode.* In *HI-mode,* all *LO-tasks* are terminated, and there is also a guarantee that if *HI-tasks* do not exceed their *HI-WCETs*, they will meet their deadlines. It is possible further on, that the system switches back to *LO-mode* if all the tasks can meet their *LO-WCET* deadlines in *LO-mode*.

A mixed-criticality schedulable system is the one that can provide such a dynamic guarantee to all tasks at run time with a specific scheduling algorithm. Now, the question is if there is such a scheduling algorithm that makes a mixed-criticality system schedulable. The next section answers this question.

4.4. Mixed-Criticality Scheduling Algorithms

Researchers have published several papers on mixed-criticality scheduling algorithms. Among them, [56,57] are examples of the first works that use different scheduling algorithms to assess the usefulness and applicability of these algorithms on mixed-criticality systems. Mixed-criticality scheduling algorithms are still a hot topic of research, and researchers try to improve the schedulabilty and QoS of the algorithms [58]. In this manuscript, we are interested in those works that deal with power management issues. For example, the work in [52] combines its DVFS approach for mixed criticality with a scheduling technique called EDF-VD (Earliest Deadline First with Virtual Deadline) [55]. It is a mode-switched EDF scheduling method for mixed-criticality tasks. In this scheduling method, reservation time budgets are reserved for *HI-tasks* in the *LO-mode* by shortening the deadlines of *HI-tasks*. This makes *HI-tasks* finish earlier in the *LO-mode* so that more time until their actual deadlines is left. This way, time budgets are reserved for *HI-tasks*, and it ensures that they will still meet their deadlines even if they overrun. The core idea here is to prevent hassles in *HI-mode* with preparing the tasks early in *LO-mode*.

4.5. Mixed Criticality and Energy Efficiency

The main focus of researchers who extended scheduling techniques for mixed criticality is on providing real-time guarantees for tasks with different criticality levels. However, other design features such as power consumption still have room to be explored. For instance, modern automotive computers consist of more than 100 electronic components [52], which translates into high power consumption. In addition, many safety-critical devices such as medical heart pacemakers are battery-operated. So, low energy consumption will lead to a significant increase in device lifetime. With increased computing requirements and because of the battery-operated nature of mixed-criticality systems, energy minimization for such systems (especially using run-time DVFS techniques) is also becoming crucial. Conventional DVFS techniques do not consider the mixed-criticality concept. Using DVFS, the processor frequency can be reduced at run time to save energy in a mixed-criticality system. Two important aspects to consider for mixed-criticality applications are task deadlines and priorities. To meet the deadlines of critical tasks, DVFS can speed up the processor when the tasks overrun. Otherwise, if the tasks do not overrun, DVFS can help the system reduce the processor frequency, which in turn will save energy. Since task overrun is rare, this trend will reduce energy consumption for mixed-criticality systems [52].

Applying DVFS to mixed-criticality systems is not straightforward. This is mostly because of conflicts arising between energy minimization and timing guarantees required by mixed-criticality systems. By lowering the processor voltage and frequency, in fact, DVFS tries to stretch the task execution time to benefit from the whole available slack time and push the execution of tasks as much as possible toward the deadline. However, this is in contrast to the time reservation approach that needs to be followed to provide a deadline guarantee for critical tasks for the cases that they overrun.

The work of [59] allocates power budgets to different criticality levels and monitors its consumption. If a crucial task exceeds its power budget during its time window, the low-criticality tasks will get less power budget than their initially assigned budget. Energy consumption of mixed-criticality systems is also addressed by [60]. Its approach is based on trading energy usage with the missed deadlines of low-criticality tasks. It claims 17% reduction of energy usage with around 4% missed deadlines. The researchers in [52] introduce a method for power optimization in mixed-criticality systems with a focus on DVFS. They increase the speed of the processor using DVFS if *HI-criticality* tasks need more than their *LO-WCET* requirement. Hence, *LO-criticality* tasks are not suspended, but more energy is consumed. Their work exploits EDF-VD scheduling for providing energy efficiency to mixed-criticality systems.

All of the above works study the energy efficiency of mixed-criticality systems using analytical modeling. Section 3 explained that simulation provides more implementation details than analytical modeling. The work in [45] demonstrates how the hardware-managed DVFS simulation framework (Section 3) can be used for energy-efficient design exploration of mixed-criticality systems. The next section presents a summary of this work.

5. Case Review

Typically, high-criticality WCETs (*HI-WCETs*) are very pessimistic. This leads to a large potential for power optimization in mixed-criticality systems using DVFS. That is, in most cases, it is possible to decrease the core frequency (and hence core voltage and energy consumption) without violating the criticality requirements. Furthermore, a hardware-based management approach suits the mixed-criticality systems very well where fulfillment of task deadlines and high predictability are essential. The previous works on mixed-criticality systems for achieving energy efficiency using DVFS are all analytical works. Only three works were found in the literature so far, that integrate power management mechanisms (specifically DVFS) within virtual prototyping platforms that originally did not support DVFS [1–3]. One work—which is in fact an extension of [3]—studies DVFS for mixed-criticality systems using architectural simulation and is selected as the case review in this article [45]. Here, it is explained how the hardware-managed DVFS framework reviewed in Section 3 is used to demonstrate an energy-efficient mixed-criticality system in one step ahead of analytical modeling: architectural simulation.

Modularity and extensibility of the virtual template reviewed in Section 3 allows adding necessary components to that work. Its extension also uses heartbeat intervals, this time for indicating criticality signals of critical applications to model their fine-grained mixed-criticality constraints. Section 4.3 explained that guaranteeing the safe operation of *HI-tasks* entails the actual execution time of *HI-tasks* being measured at run time. Using run-time heartbeat intervals of critical applications (*HI-tasks*), the work ensures that presumed coarse-grained deadlines are guaranteed; an adapted version of the heartbeat framework used by this architecture measures the *HI-task* execution times and indeed fine-tunes the criticality requirements of *HI-tasks* as they contain the heartbeat infrastructure in this approach.

The assumptions considered by the reviewed case are important. They are as follows:

- Task model is dual-criticality sporadic implicit-deadline model (*HI-tasks* and *LO-tasks*) with two operation modes (*HI-mode, LO-mode*) (see Section 4.3).
- A software scheduler in collaboration with custom hardware utilizes a scheduling algorithm for mixed-criticality applications such as EDF-VD (see Section 4.4).
- *HI-tasks* are mixed-criticality schedulable in *HI-mode*.
- All tasks are mixed-criticality schedulable in *LO-mode*.

- In addition, the system is mixed-criticality schedulable. It means that using the mode switch protocol (see Section 4.3), the system is capable of providing a run-time guarantee for all tasks so that they do not miss their deadlines.

The baseline consists of a quad-core system on which several critical (*HI-tasks*) and non-critical tasks (*LO-tasks*) are running. Occasional addition of more *LO-tasks* during the run time of *HI-tasks* creates difficulties for the system. The challenge is that resource-hungry *LO-tasks* consume shared resources and therefore interfere with the execution of *HI-tasks* that are running on the same platform (mixed-criticality system). This leads to deviation from expected execution times of *HI-tasks* and consequently causes missed deadlines. Using a hardware-controlled DVFS approach, the work handles such a catastrophic situation and also saves energy.

Figure 2 depicts an architectural view of this work. The architecture is similar to [3] with further extension to support mixed-criticality tasks. In addition to that, it extends that work to support per-core DVFS. An on-chip run-time hardware-managed DVFS module as well as run-time health monitoring sub-systems accompany the baseline system. The work employs a hardware-assisted DVFS manager that determines the scaling direction during *LO-mode* operations in a way that *HI-tasks* and *LO-tasks* do not violate their *LO-WCETs*. Note that a violation in *LO-mode* leads to switch to *HI-mode* in which the DVFS manager plays no roles (mode switch protocol—see Section 4.3).

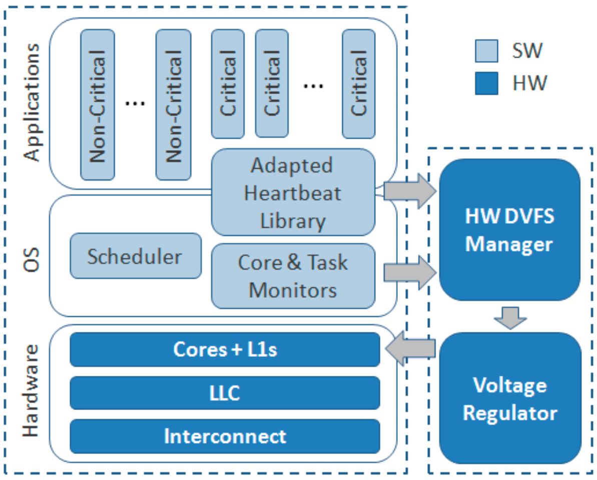

Figure 2. Hardware-managed DVFS architecture for mixed-criticality systems adopted from [45].

The health monitoring modules act as run-time sources of information beside the application heartbeats—which measure intervals of *HI-tasks*—to help the hardware-controlled DVFS manager make right scaling decisions. They provide mapping information of the critical tasks and also utilization of individual processing cores. Since the system is criticality schedulable, *HI-mode* operations are guaranteed to meet their corresponding deadlines, despite the interference of occasionally added *LO-tasks*.

The deciding module in this architecture is a policy algorithm that manages the scaling process. The work introduces a sample DVFS policy algorithm implemented as custom hardware to apply per-core DVFS to the baseline system in its architectural template. The DVFS manager (implemented as custom hardware) measures and checks if the heartbeat intervals of the *HI-tasks* meet mixed-criticality requirements. In addition, it considers the run-time information arriving from health monitoring modules to make the scaling decision. The work in [45] models its experimental mixed-criticality system through architectural simulation and reports 24.6% saving in power consumption using its proposed approach with no rise in heartbeat violations.

In a nutshell, although occasional addition of more heavy non real-time tasks during run time imposes timing impairments to the mixed-criticality system (chosen to model an intricate embedded system), using a hardware-controlled DVFS approach, the work is capable of handling such a situation to retain performance while besides, saving energy.

6. Conclusions

System-level architectural simulation enables SoC designers to have a prototype of the full systems under design with minimum cost and effort. It gives the possibility of running real applications with different requirements as if they run on real hardware. This is valuable in that, required changes can be detected in the early stage of design cycle rather than later on, during the costly verification phase.

A key feature that has recently attracted interest is integrating run-time power management mechanisms within the architectural simulation tools. This is to cope with the new challenges of modern embedded systems architectures. To reduce the induced overhead, hardware-assisted mechanisms provide a better solution due to faster reaction to resource demands as well as providing better timeliness for applications with strict timing requirements.

This article provides a review on system-level techniques, particularly DVFS, to model and design low-power embedded systems, with an emphasis on hardware-based mechanisms. A case review in the end gives insights into application of hardware-managed DVFS to the mixed-criticality application domain using system-level architectural simulation. The case review indicates the usefulness of system-level modeling of run-time methods through architectural simulation in power and performance analysis of next-generation complex embedded systems in the early design phase.

Funding: This research received no external funding.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Spiliopoulos, V.; Bagdia, A.; Hansson, A.; Aldworth, P.; Kaxiras, S. Introducing DVFS-management in a full-system simulator. In Proceedings of the IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems, San Francisco, CA, USA, 14–16 August 2013.
2. Lebreton, H.; Vivet, P. Power modeling in SystemC at transaction level, application to a dvfs architecture. In Proceedings of the 2008 IEEE Computer Society Annual Symposium on VLSI, Montpellier, France, 7–9 April 2008.
3. Haririan, P.; Garcia-Oritz, A. Non-Intrusive DVFS emulation in gem5 with application to Self-aware architectures. In Proceedings of the 9th Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), Montpellier, France, 26–28 May 2014.
4. Vece, G.B.; Conti, M.; Orcioni, S. Transaction-level power analysis of VLSI digital systems. In *Department of Information Engineering*; Università Politecnica delle Marche: Ancona, Italy, 2015.
5. Vetromille, M.; Ost, L.; Marcon, C.A.M.; Reif, C.; Hessel, F. RTOS Scheduler Implementation in Hardware and Software for Real Time Applications. In Proceedings of the 17th IEEE International Workshop on Rapid System Prototyping, Chania, Crete, Greece, 14–16 June 2006; doi:10.1109/RSP.2006.34.
6. Burns, A.; Davis, R. *Mixed Criticality Systems—A Review*, 12th ed.; University of York Tech. Rep.; University of York: York, UK, 2019.
7. Mittal, S. *A Survey of Techniques for Improving Energy Efficiency in Embedded Computing Systems*; Future Technologies Group Oak Ridge National Laboratory (ORNL): Oak Rdge, TN, USA, 2014.
8. Munir, A.; Ranka, S.; Gordon-Ross, A. High-Performance Energy-Efficient Multicore Embedded Computing. *IEEE Trans. Parallel Distrib. Syst.* 2012, **23**, 684–700.
9. Ghasemzadeh, H.; Jafari, R. Ultra low power signal processing in wearable monitoring systems: A tiered screening architecture with optimal bit resolution. *ACM Trans. Embedd. Comput. Syst. TECS* **2013**, *13*, 9, doi:10.1145/2501626.250163.

10. Rajovic, N.; Carpenter, P.M.; Gelado, I.; Puzovic, N.; Ramirez, A.; Valero, M. Supercomputing with commodity CPUs: Are mobile SoCs ready for HPC? In Proceedings of the International Conference on High. Performance Computing, Networking, Storage and Analysis (SC ’13), Denver, CO, USA, 17–21 November 2013; ACM: New York, NY, USA, 2013; p. 40, doi:10.1145/2503210.2503281.
11. Ahmad, I.; Ranka, S. *Handbook of Energy-Aware and Green Computing*; Taylor and Francis Group CRC Press: Boca Raton, FL, USA, 2011.
12. Gepner, P.; Fraser, D.; Kowalik, M.; Tylman, R. New Multi-Core Intel Xeon Processors Help Design Energy Efficient Solution for High Performance Computing. In Proceedings of the International Multiconference on Computer Science and Information Technology (IMCSIT), Mrągowo, Poland, 12–14 October 2009.
13. Processing Architecture for Power Efficiency and Performance. Available online: https://www.arm.com/why-arm/technologies/big-little (accessed on 1 November 2019).
14. Book: ARM Cortex-A Series, Programmer’s Guide for ARMv8-A, Version 1.0. Available online: https://developer.arm.com/docs/den0024/a/preface (accessed on 1 November 2019).
15. Scrbak, M.; Greathouse, J.L.; Jayasena, N.; Kavi, K. DVFS Space Exploration in Power Constrained Processing-in-Memory Systems. Architecture of Computing Systems. In *International Conference on Architecture of Computing Systems ARCS 2017*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2017; Volume 10172.
16. Available online: http://www.amd.com (accessed on 1 November 2019).
17. Available online: http://www.intel.com (accessed on 1 November 2019).
18. Mishra, S.K.; Parida, P.P.; Sahoo, S.; Sahoo, B.; Jena, S.K. Improving energy usage in cloud computing using DVFS. In *Progress in Advanced Computing and Intelligent Engineering. Advances in Intelligent Systems and Computing*; Saeed, K., Chaki, N., Pati, B., Bakshi, S., Mohapatra, D., Eds.; Springer: Singapore, 2018; Volume 563, doi:10.1007/978-981-10-6872-0_60.
19. Mao, Y.; Zhang, J.; Letaief, K.B. Dynamic Computation Offloading for Mobile-Edge Computing with Energy Harvesting Devices. *IEEE J. Sel. Areas Commun.* **2016**, *34*, 3590–3605.
20. Zhang, C.; Zhao, H.; Deng, S. A density-based offloading strategy for IoT devices in edge computing systems. *IEEE Access* **2018**, *6*, 73520–73530, doi:10.1109/ACCESS.2018.2882452.
21. Huang, P.; Kumar, P.; Giannopoulou, G.; Thiele, L. Run and be safe: Mixed-criticality scheduling with temporary processor speed up. In Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition, EDA Consortium, Grenoble, France, 9–13 March 2015.
22. Weiser, M.; Welch, B.; Demers, A.; Shenker, S. Scheduling for reduced CPU energy. In Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI), Monterey, CA, USA, 14–17 November 1994; pp. 13–23.
23. Contreras, G.; Martonosi, M. Live runtime phase monitoring and prediction on real systems with application to dynamic power management. In Proceedings of the IEEE/ACM Annual International Symposium on Microarchitecture, Orlando, FL, USA, 9–13 December 2006.
24. Choi, K.; Soma, R.; Pedram, M. Fine-grained dynamic voltage and frequency scaling for precise energy and performance trade-off based on the ratio of off-chip access to on-chip computation times. In Proceedings of the Conference on Design, Automation and Test in Europe, Paris, France, 16–20 February 2004; Volume 1, p. 10004.
25. Hua, S.; Qu, G.; Bhattacharyya, S.S. An energy reduction technique for multimedia application with tolerance to deadline misses. In Proceedings of the Design Automation Conference IEEE, Las Vegas, NV, USA, 2–6 June 2003.
26. Kianzad, V.; Bhattacharyya, S.S.; Qu, G. CASPER: An integrated energy-driven approach for task graph scheduling on distributed embedded systems. In Proceedings of the IEEE International Conference on Application-Specic Systems, Architecture Processors (ASAP), Samos, Greece, 23–25 July 2005.
27. Kim, W.; Gupta, M.S.; Wei, G.-Y.; Brooks, D. System level analysis of fast per-core DVFS using on-chip switching regulators. In Proceedings of the IEEE 14th International Symposium on High Performance Computer Architecture (HPCA), Salt Lake City, UT, USA, 16–20 February 2008.
28. Jayaseelan, R.; Mitra, T. A hybrid local-global approach for multi-core thermal management. In Proceedings of the IEEE/ACM International Conference Computer-Aided Design (ICCAD), San Jose, CA, USA, 2–5 November 2009.

29. Jevtić, R.; Le, H.P.; Blagojević, M.; Bailey, S.; Asanović, K.; Alon, E.; Nikolić, B. Per-core dvfs with switched-capacitor converters for energy efficiency in manycore processors. *IEEE Trans. Very Large Scale Integr. VLSI Syst.* **2014**, *23*, 723–730.
30. Robino, F.; Oberg, J. The HeartBeat model: A platform abstraction enabling fast prototyping of real-time applications on NoC-based MPSoC on FPGA. In Proceedings of the 8th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), Darmstadt, Germany, 10–12 July 2013; p. 1, doi:10.1109/ReCoSoC.2013.6581536.
31. Sironi, F.; Triverio, M.; Hoffmann, H.; Maggio, M.; Santambrogio, M.D. Self-aware adaptation in FPGA-based Systems. In Proceedings of the 2010 International Conference on Field Programmable Logic and Applications (FPL), Milano, Italy, 31 August–2 September 2010; Volume 187, p. 192.
32. Gregorek, D.; Garcia-Ortiz, A. The agamid design space exploration framework—Task-accurate simulation of hardware-enhanced run-time management for many-core. In *Design Automation for Embedded Systems*; Springer: Berlin/Heidelberg, Germany, 2018.
33. John, L.K.; Eeckhout, L. *Performance Evaluation and Benchmarking*; Taylor and Francis CRC Press: Boca Raton, FL, USA, 2005.
34. Guo, Q.; Chen, T.; Chen, Y.; Franchetti, F. Accelerating Architectural Simulation via Statistical Techniques: A Survey. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2016**, *35*, 433–446, doi:10.1109/TCAD.2015.2481796.
35. Van den Steen, S.; De Pestel, S.; Mechri, M.; Eyerman, S.; Carlson, T. Micro-architecture independent analytical processor performance and power modeling. In Proceedings of the 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Philadelphia, PA, USA, 29–31 March 2015; pp. 32–41, doi:10.1109/ISPASS.2015.7095782.
36. Magnusson, P.S. Simics: A full system simulation platform. *Computer* **2002**, *35*, 50–58, doi:10.1109/2.982916.
37. Binkert, N.; Beckmann, B.; Black, G.; Reinhardt, S.K.; Saidi, A.; Basu, A.; Hestness, J.; Hower, D.R.; Krishna, T.; Sardashti, S.; et al. The gem5 Simulator. *ACM SIGARCH Comput. Arch. News.* **2011**, *39*, 1–7.
38. Patterson, D.A.; Hennessy, J.L. *Computer Organization and Design: The Hardware/Software Interface*; Morgan Kaufmann: Burlington, MA, USA, 2011.
39. IEEE. *IEEE Design Automation Standards Committee: IEEE Std. 1666–2011, IEEE Standard for Standard SystemC Language Reference Manual (2011), Approval to Revised Standard*; IEEE: Piscataway, NJ, USA, 2011.
40. Menard, C.; Jung, M.; Castrillon, J.; Wehn, N. System simulation with gem5 and systemC: The keystone for full interoperability. In Proceedings of the IEEE International Conference on Embedded Computer Systems Architectures Modeling and Simulation (SAMOS), Pythagorion, Greece, 17–20 July 2017.
41. Available online: https://parsa.epfl.ch/simflex/flexus.html (accessed on 1 November 2019).
42. Ardestani, K.E.; Renau, J. ESESC: A Fast Multicore Simulator Using Time-Based Sampling. In Proceedings of the International Symposium on High Performance Computer Architecture (HPCA 19), Shenzhen, China, 23–27 February 2013.
43. Carlson, T.E.; Heirman, W.; Eeckhout, L. Sniper: Exploring the Level of Abstraction for Scalable and Accurate Parallel Multi-Core Simulation. In Proceedings of the SC11, Seattle, DC, USA, 12–18 November 2011.
44. Hoffmann, H.; Eastep, J.; Santambrogio, M.D.; Miller, J.E.; Agarwal, A. Application Heartbeats for Software Performance and Health. In Proceedings of the 15th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming, Bangalore, India, 9–14 January 2010.
45. Haririan, P.; Garcia-Ortiz, A. A framework for hardware-based DVFS management in multicore mixed-criticality systems. In Proceedings of the 10th International Symposium on Reconfigurable Communication-Centric Systems-on-Chip (ReCoSoC), Bremen, Germany, 29 June–1 July 2015.
46. Available online: http://www.autosar.org (accessed on 1 November 2019).
47. Available online: https://www.aviation-ia.com/content/arinc-standards (accessed on 1 November 2019).
48. Agrawal, K.; Baruah, S. Intractability Issues in Mixed-Criticality Scheduling. In Proceedings of the 30th Euromicro Conference on Real-Time Systems (ECRTS), Barcelona, Spain, 3–6 July 2018; doi:10.4230/LIPIcs.ECRTS.2018.11.
49. Li, H.; Baruah, S. Load-based schedulability analysis of certifiable mixed-criticality systems. In Proceedings of the EMSOFT, Scottsdale, AZ, USA, 24–29 October 2010; pp. 99–108.
50. Ekberg, P.; Yi, W. Bounding and shaping the demand of mixed-criticality sporadic tasks. In Proceedings of the ECRTS, Pisa, Italy, 11–13 July 2012; pp. 135–144.

51. Park, T.; Kim, S. Dynamic scheduling algorithm and its schedulability analysis for certifiable dual-criticality systems. In Proceedings of the EMSOFT, Taipei, Taiwan, 9–14 October 2011; pp. 253–262.
52. Huang, P.; Kumar, P.; Giannopoulou, G.; Thiele, L. Energy Efficient DVFS Scheduling for Mixed-Criticality Systems. In Proceedings of the ESWEEK, New Delhi, India, 12–17 October 2014.
53. Baruah, S.K.; Burns, A.; Davis, R.I. Response-time analysis for mixed criticality systems. In Proceedings of the RTSS, Vienna, Austria, 29 November–2 December 2011.
54. Baruah, S.K.; Goossens, J. Scheduling real-time tasks: Algorithms and complexity. In *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*; Leung, J.Y.T., ed.; CRC Press: Boca Raton, FL, USA, 2003; Chapter 28.
55. Baruah, S.; Bonifaci, V.; D'Angelo, G.; Li, H.; Marchetti-Spaccamela, A.; van der Ster, S.; Stougie, L. The Preemptive Uniprocessor Scheduling of Mixed-Criticality Implicit-Deadline Sporadic Task Systems. In Proceedings of the ECRTS, Pisa, Italy, 10–13 July 2012.
56. Vestal, S. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In Proceedings of the IEEE Real-Time Systems Symposium (RTSS), Tucson, AZ, USA, 3–6 December 2007; pp. 239–243.
57. Baruah, S.K.; Vestal, S. Schedulability analysis of sporadic tasks with multiple criticality specifications. In Proceedings of the ECRTS, Prague, Czech Republic, 2–4 July 2008; pp. 147–155.
58. Pathan, R.M. Improving the Schedulability and Quality of Service for Federated Scheduling of Parallel Mixed-Criticality Tasks on Multiprocessors. In Proceedings of the 30th Euromicro Conference on Real-Time Systems (ECRTS), Barcelona, Spain, 3–6 July 2018; doi:10.4230/LIPIcs.ECRTS.2018.12.
59. Broekaert, F.; Fritsch, A.; Sa, L.; Tverdyshev, S. Towards power-efficient mixed-critical systems. In Proceedings of the OSPERT, Paris, France, 9 July 2013.
60. Legout, V.; Jan, M.; Pautet, L. Mixed-criticality multiprocessor real-time systems: Energy consumption vs deadline misses. In Proceedings of the ReTiMiCS (RTCSA 2013), Taipei, Taiwan, 19–21 August 2013.