Proceedings of CONF-SEML 2026 Symposium: Learning and Decision Making in Multi Agent Software Systems
DOI: 10.54254/2755-2721/2026.BA31709

Research on Low-Power Optimization Methods and Software-Hardware Co-design Strategies for Embedded Systems

Wenji Shi

The Hong Kong Polytechnic University, Hong Kong, China
2368546848@qq.com

Abstract. As the clock frequency and integration density of embedded systems continue to rise, especially with the widespread use of mobile embedded devices such as smartphones and Personal Digital Assistants (PDAs), power consumption has become a critical challenge in system design. Besides, the slower slower advancement of battery technology compared to computational power makes system-level optimizations to reduce energy consumption essential for enhancing performance and increasing device longevity. Therefore, this paper explores methods for power consumption optimization in embedded systems, examining the potential for low-power solutions via the collaboration of hardware and software design. By reviewing recent research and investigating the findings of Tiwari et al., which involved simulations and typical embedded system setups, this study offers a comprehensive analysis of how to optimize power consumption from multiple angles. The results demonstrate that software-level optimizations, such as algorithm scheduling and instruction optimization, can greatly reduce system power usage. Furthermore, software-hardware co-design is identified as the key direction for the future development of low-power embedded systems, enabling high energy efficiency without compromising performance.

Keywords: Embedded system, Low-power optimization, Software-hardware co-design, Power model, Dynamic Voltage and Frequency Scaling (DVFS)

1. Introduction

Although low-power tech has become a hot topic for research with some progress made, there are still gaps in the existing research. At the hardware level, as semiconductors approach their physical limits, thus reducing power consumption becomes harder. On the software side, modeling software behavior and hardware power consumption is challenging due to their interdependence [1]. Besides, most previous studies stress optimizing hardware and software in isolation, neglecting to adequately explore or validate the potential of software-hardware co-optimization to achieve the maximum energy efficiency across the entire system [2]. Thus, this study seeks to explore how both software and hardware can work better together in embedded systems [2]. Through a literature review and case study, this paper summarizes low-power technologies in hardware (CMOS power models) and software (algorithm and source code levels). Furthermore, it analyzes Tiwari's research framework, uses the EMSIM power simulator to compare function codes and experimental data before and after optimization on the Linux platform, and evaluates the effectiveness of these strategies [1]. This full

DOI: 10.54254/2755-2721/2026.BA31709

evaluation offers design insights and practical approaches, emphasizing the need for future efforts to integrate software and hardware design. It highlights the need to address power issues through strategies such as dynamic power scaling, developing energy-efficient programs, and performing computations where minimal energy consumption is most effective.

2. Overview of low-power embedded systems

2.1. Definition and application of low-power embedded systems

Low-power embedded systems are characterized by their ability to minimize power consumption via design improvements, without compromising their functionality [3]. These systems are typically deployed in environments with stringent energy consumption regulations, particularly in portable devices such as smartphones, wearable timepieces, and compact home surveillance devices. Though computing power continues to advance rapidly, battery technology has not kept pace, thus making energy efficiency a critical factor in the design of embedded systems.

For example, the iPhone 17 Pro series features the Apple A19 Pro chip, which incorporates both high-performance cores and high-efficiency cores. To conserve power, the system automatically shifts to efficiency cores and disables unnecessary components when idle or performing background operations, ensuring the device lasts throughout the day with minimal usage. Similarly, many smart home sensors, like door and window magnetic sensors, can also work for a long time on one button cell battery because they mostly stay in a very low-power sleep state [4]. As technology advances, the integration of systems and processors' main frequencies continue to increase, leading to higher power consumption and hotter chips. Additionally, battery technology has advanced much more slowly than computing power; over the past 30 years, computer performance has increased by roughly four orders of magnitude, while battery capacity has only grown by 4 to 8 times. Therefore, reducing the power consumption is necessary for the system design [5]. This type of design has become a standard way to create embedded systems, and it directly affects how well the entire system performs.

2.2. Hierarchy and principles of low-power design

Low-power design is a crucial strategy for managing energy consumption in embedded systems, covering both hardware and software. At the hardware level, power consumption arises mainly from dynamic activity and static leakage in circuits. To reduce energy usage, hardware optimization focuses on selecting materials, improving manufacturing processes, and enhancing circuit design. For instance, DVFS technology adjusts the processor's voltage and frequency based on workload demands; multi-core heterogeneous architectures switch to low-power cores under light loads; and power gating technology shuts off power to specific parts of the circuit during idle periods to further conserve energy.

Software-wise, embedded systems depend on software drivers to execute instructions and access data, which directly affects the hardware's power consumption [6]. As shown in Figure 1, a typical embedded system setup experiences high interface power usage due to frequent peripheral wakeups and data transfers. Besides, the operating system's scheduling strategy determines how much time the processor spends working versus sleeping. And a microprocessor's dynamic power consumption is linked to the number of instructions executed, as well as the algorithms and methods used by software to access data, which consequently influences the hardware's state. Thus, improving both hardware and software simultaneously is essential for low-power design [7]. Early studies focused

DOI: 10.54254/2755-2721/2026.BA31709

mainly on boosting hardware efficiency, but it became clear that integrating software optimization with hardware optimization results in greater power savings, surpassing the benefits of hardware improvements alone. Hence, the combined optimization of both hardware and software is crucial for an effective low-power design.

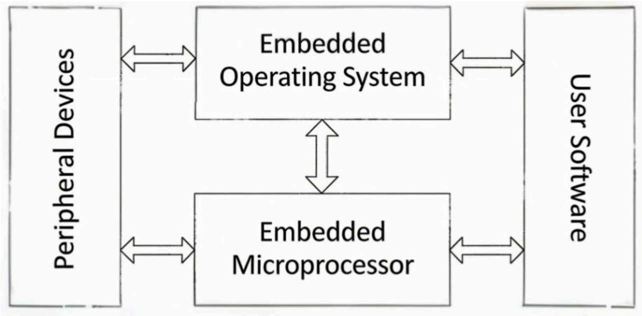

Figure 1. Typical embedded system structure

2.3. Technical challenges and trends in low-power design

Low-power design in embedded systems faces numerous technical challenges. From the hardware viewpoint, it is approaching physical limits, with power consumption constrained by process nodes, power management effectiveness, and device leakage current, leaving little potential for further reduction. On the software side, power modeling is complex, as software actions indirectly affect hardware power usage through instruction execution and data retrieval. These effects are limited by several aspects including processor architecture, cache structure, and memory access methods, which makes it hard to model them precisely and stops the creation of software-based low-power tech [1]. From the perspective of collaborative optimization, hardware-software co-optimization has yet to be realized. Existing designs mainly rely on single-layer optimization and fail to effectively use cross-layer power regulation mechanisms, thus not fully unlocking the system's energy-saving potential.

At present, several key trends are emerging in the development of low-power design. To tackle the energy efficiency challenges of general-purpose processors, heterogeneous computing has emerged as a popular solution. By combining computing units with different architectures, such as ARM's big.LITTLE, big-core/small-core architectures, or a combination of CPU, GPU, and NPU, the system assigns tasks to the most energy-efficient unit. For example, a smartphone offloads AI inference tasks to the NPU, resulting in significantly higher energy efficiency compared to the CPU. Besides, Domain-Specific Architectures (DSA), such as Google's TPU, offer specialized hardware optimized for specific algorithms (e.g., image processing and matrix operations), greatly boosting energy efficiency [8]. Furthermore, energy-aware architectures are gaining momentum, enabling dynamic power optimization at runtime through the integration of power monitoring units and OS-level energy-aware scheduling.

3. Power optimization and software-hardware synergy

3.1. Principles and methods of software-hardware co-design

The core idea of hardware-software co-optimization is to transcend the limitations of single-layer optimization by establishing a quantitative link between hardware power consumption and software execution. And this approach allows for comprehensive energy management across different layers,

DOI: 10.54254/2755-2721/2026.BA31709

with hardware acting as the physical platform where power is consumed, and software governing and influencing hardware behavior. The two are closely interconnected through instruction streams, data streams, and control streams. Common co-optimization techniques include cross-layer power modeling, as well as the abstraction and standardization of interfaces.

Based on the Tiwari method, cross-layer power modeling provides a way to understand how software actions influence power consumption by creating power models derived from C-language instructions. This method begins by measuring the basic power usage of each individual instruction from a hardware perspective. It then integrates the transition costs between instructions, along with pipeline effects, to establish a connection between instruction groups and their corresponding power consumption. This approach proves valuable in several application scenarios, including comparing the power consumption of different implementation strategies, such as deciding whether to apply loop unrolling, during algorithm design; selecting the optimal instruction scheduling plan during compilation optimization; and predicting power consumption for various task sets during task scheduling. With this model, designers can assess the power usage of different software algorithms or hardware architectures early in the system design process, enabling co-design [9]. Interface abstraction and standardization play a key role in this process. The power management interfaces provided by the operating system allow software to control the hardware's power management unit through standardized calls, facilitating efficient dynamic power management. This ensures software portability and optimizes the use of the hardware's low-power features.

3.2. Hardware optimization technologies and paths

Hardware optimization seeks to minimize both static leakage power and dynamic switching power in circuits, primarily through the adoption of ultra-low-power FinFET technologies, cutting-edge nanoscale transistors, and various threshold voltage methods. For example, in IoT chip design, multi-threshold voltage technology allows the background control logic to use a high-threshold voltage library, reducing static power consumption by 50%-80% compared to core computing units. Transitioning from a 28nm planar process to a 16nm/12nm FinFET process can reduce overall chip power consumption by 30%-50% at the same performance level, thereby ensuring longer battery life for devices. Moreover, Dynamic Voltage and Frequency Scaling (DVFS) is another application of hardware-software co-optimization [5]. The hardware offers configurable voltage and frequency domains, while the software dynamically adjusts the processor's voltage and frequency according to real-time workload predictions, greatly reducing power consumption while preserving performance. Furthermore, near-threshold computing allows processors to operate at extremely low voltages, resulting in significant power savings. This is particularly beneficial for tasks requiring minimal performance, such as collecting sensor data from wearables or standby monitoring in smart homes. Low-power microcontrollers and power management ICs, which feature low-power modes, are often selected alongside low-power IP cores and components, providing multiple power states for software to optimize. For instance, in a smartwatch's "screen-off" state, it can enter a deep sleep mode where only the RTC clock remains active, reducing power consumption to the microwatt range. Similarly, wireless sensor nodes can turn off their RF modules and main processors while collecting data, using low-power timers to wake up, extending the device's battery life from a few months to several years.

DOI: 10.54254/2755-2721/2026.BA31709

3.3. Software optimization strategies and effects

By improving algorithm design and code performance, software optimization reduces the workload on the processor from the beginning. This involves choosing algorithms with lower complexity and better memory access, and optimizing code to reduce unnecessary instructions and branches. At the same time, the operating system, which acts as the central interface between hardware and software, can contribute to energy savings via intelligent scheduling strategies [10]. One common approach is to execute tasks in batches, allowing the processor to enter an idle state as quickly as possible. In addition, tasks can be allocated to cores with varying performance levels, as seen in the big.LITTLE architecture, depending on the task's needs. Besides, software can manage peripherals and storage devices by controlling their power states, shutting down idle devices, and reducing memory access frequency using memory compression techniques. By cutting down the energy consumption of non-core components, the overall system power usage is reduced [11].

Meanwhile, software-layer optimization techniques boost the efficiency of utilizing hardware's low-power states through a comprehensive power management system. While operating system and software-level strategies do not directly improve the chip's physical energy efficiency, they help minimize unnecessary energy waste at the system level by increasing hardware idle time, hence optimizing task execution, and precisely controlling peripheral devices. Specifically, smart task scheduling ensures that the CPU enters a deep sleep state immediately after completing its critical computations; efficient algorithms and code reduce the time the CPU spends working; and careful management of supporting components and clocks prevents excess energy from being wasted [3]. In quantitative terms, these optimizations have shown substantial reductions in energy consumption in practical applications. For instance, in GSM terminal systems, power consumption was reduced by about 50.86% through software-controlled low-power modes. When an adaptive surrogate model was implemented, system energy consumption was reduced by 2.5%, with reductions reaching up to 16.4% [12,13].

4. Conclusion

This paper explores methods to advance low-power technologies for embedded systems, especially through the combined efforts of hardware and software. The results show that the key strategy for surpassing today's energy-efficiency limitations and improving the system's energy conservation is through hardware-software co-design. By creating cross-layer power models and forming a deep connection between hardware and software scheduling methods, the system's energy can be utilized most efficiently while still meeting the necessary performance criteria. Besides, the integration of heterogeneous computing, near-threshold computing, and energy-aware architecture are advancing smarter low-power design. As technology advances, power management has shifted from reactive to predictive, thereby allowing for real-time adjustments as systems operate. This shift presents new opportunities for dynamic optimization. However, a key challenge remains in effectively balancing hardware-software co-optimization in real-world applications and adapting strategies to different contexts. Future research could focus on refining the cross-layer optimization model, improving power prediction accuracy, and enhancing scheduling methods under conditions like heterogeneous and near-threshold computing. Meanwhile, Intelligent Energy Efficiency Management will also be a key focus, using AI and other technologies for proactive optimization to reduce energy waste. Besides, application-specific optimizations are essential, as wearable devices and smart home apps have unique requirements, requiring customized hardware-software co-optimizations.

DOI: 10.54254/2755-2721/2026.BA31709

References

[1] Tiwari, V., Malik, S., & Wolfe, A. (1994). Power analysis of embedded software: A first step towards software power minimization. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2(4), 123-132.

[2] Pan, Y., & Li, W. (2010). Research on methods for reducing power consumption in embedded system design. Journal of Baicheng Normal University, 24(6), 33-35.

[3] Liu, W. (2012). Research and application of low power consumption technology in embedded systems. Zhengzhou University.

[4] Liu, X., & Liu, S. C. (2011). The current status and development trends of embedded system applications. Science and Technology Information, (2), 247-249.

[5] Mu, Y.F. (2022). Research on low power consumption and reliability technology of embedded systems. Electronic Testing, 36(9), 132-134.

[6] Han, B. A. (2014). Exploration of low power consumption design issues in embedded systems. Electronic Technology & Software Engineering, (1), 224.

[7] Halambi, A., Shrivastava, A., Biswas, P., & Dutt, N. (2002). An efficient compiler technique for code size reduction using reduced bit-width ISAs. IEEE Automation and Test in Europe Conference and Exhibition, 3(4), 402-408.

[8] Kawami, E., & Yasushi, S. (2008). Adaptation to structural changes using a model-based embedded intelligent control system. Proceedings of the Fuzzy Systems Symposium of the Japan Society for Fuzzy Theory and Intelligent Informatics, 24, FF2-1.

[9] Shimada, T., Sakai, T., & Kitamura, T. (2005). Construction and evaluation of a motion detection system using embedded processors. Reports of the Information Processing Society of Japan, Computer Architecture Research Group, 80, 31-36.

[10] Lee, M. T. C., Tiwari, V., Malik, S., & Fujita, M. (1997). Power analysis and minimization techniques for embedded DSP software. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 5(1), 123-135.

[11] Sundaresan, K., & Mahapatra, N. R. (2003). Code compression techniques for embedded systems and their effectiveness. IEEE Computer Society Annual Symposium, 12(5), 262-263.

[12] Song, Y., Guo, B., & Shen, Y. (2011). Implementation scheme of embedded system low-power software. Microcontroller and Embedded System Applications, 11(9), 4-7.

[13] Ni, Y., Du, X., Ye, P., Xiao, R., Yuan, Y., & Li, W. (2019). Frequent pattern mining assisted energy consumption evolutionary optimization approach based on surrogate model at GCC compile time. Swarm and Evolutionary Computation, 50, 100574.