Optimization Techniques for Multi-Core Integrated Modular Avionics

T. Venkata Mani[*], P. Neeraja, BHVSN Murthy[$], Anil Kumar Vuppala[!]
[#]*Research Centre Imarat, Hyderabad, India*
[$]*DIAT Pune, India*
[#]*International Institute of Information Technology, India*
**Email venkata.mani.rci@gov.in:*

doi: https://doi.org/10.37285/bsp.SACAD2025.74

Abstract: **Modern embedded system based on Systems-on-Chip (Systems) must deliver high performance under constrained power, memory, and hardware area budgets. These systems integrate heterogeneous processing units, including CPUs, DSPs, hardware accelerators, and multi-level memory hierarchies. However, poor coordination between hardware and software, inefficient task scheduling, memory contention, communication bottlenecks, and static power consumption degrade overall efficiency. This paper presents a comprehensive review of hardware - software optimization strategies across four complementary domains: power , memory, resource, and performance optimization. Architecture-level low-power design, cache-aware data placement, communication-aware scheduling, controlled retiming, and hardware-software partitioning is consolidated into a unified design methodology for energy-efficient embedded Systems. Recent trends such as energy-harvesting platforms, near-threshold computing, approximate computing, DMA-driven performance acceleration, and machine learning based power management are also discussed. This survey aims to provide embedded system designers with a holistic framework for co-optimizing hardware and software to achieve sustainable, high-performance System architectures for future intelligent edge environments**

Keywords- Embedded Systems, Hardware–Software Co-Design, Power Optimization, Memory Hierarchy Optimization, Resource Optimization, Performance Optimization, DVFS, Retiming, Cache Customization, Dynamic Power Management, Real-Time Scheduling.

I. Introduction

This Embedded Systems-on-Chip (Systems) serve as the computational foundation for modern applications including industrial automation, intelligent sensing, mobile devices, healthcare instrumentation, automotive systems and defense applications[10]. Their growing compute requirements, driven by edge AI and real-time workloads, contrast sharply with strict constraints on energy, memory[1,4,] silicon area, and cost. Unlike general-purpose processors, embedded systems must ensure:

- High throughput for real-time workloads
- Low power consumption for battery-based operation[9]
- Predictable latency for mission-critical reliability[7]
- Efficient communication and memory management

The challenge arises because advancements in software complexity outpace hardware scaling, leading to:

- Processor stalls due to memory bottlenecks[1]
- High leakage power from continuously active functional units
- Performance degradation from communication contention[13]
- Increased buffer and cache demand from pipelining
- Therefore, single-domain optimization is insufficient.
- Embedded Systems require co-optimization across hardware, software, and system levels.

II. Optimization Challenges in Embedded Systems

Previous research often treats power, memory, resource, and performance techniques as independent goals[10,15.] In contrast, this paper demonstrates that the Power, Memory, Resource and Performance are strongly interdependent. The Interdependencies based on optimization and the challenges encountered can be seen in the following table:

Optimization Action	Benefit	Challenges
Deep pipelines	Performance increases	Increase in Memory buffers leading to increased Static power[1,3]
Hardware acceleration	CPU load decreases	Increase in Area and communication bandwidth[12,4]
DVFS	Dynamic power reduces	Execution time if aggressive increases[9,15]
Memorybanking	Power and Latency reduces	Hardware area increases[1,3]

Table 1: Interdependencies based on optimization and the challenges encountered

III. Power Optimization in Embedded Systems

Before Power management includes controlling dynamic and static power components as given by

$$P_{dynamic} = \alpha C V^2 f$$

$$P_{static} = I_{leak} \cdot VP$$

Where C=capacitance, V= voltage, f=frequency of operation, Ileak = leakage current and VP = power consumed.

The various measures that can be adopted for power optimization at hardware level are shown in the following table:

Technique	Description	Benefit
DVFS	Scale voltage/frequency to workload	Quadratic reduction in dynamic power[9]
Power Gating	Disable idle modules completely	Eliminates leakage
Clock Gating	Turn off clock to unused logic	Lowers switching activity[15]
Efficient regulators	Switch-mode instead of LDO	Reduced power dissipation
Race-to sleep	Finish tasks fast, deep sleep	Overall energy reduces[15]

Table 2: Power Optimization measures

Some of the System level power optimization techniques include-

The system only enables transceiver during scheduled communication windows. This allows significant reduction in average energy consumption, especially for low-data-rate IoT sensors[10].

A. Adaptive Modulation and Coding

Dynamically adjusts modulation schemes (e.g., BPSK, QPSK,QAM) and coding rates according to channel quality. This Optimizes the trade-off between transmission energy, reliability, and latency[10,15].

B. Dynamic Power Mode Transitions

System firmware can transition components between run, standby, retention, and deep-sleep modes15. Techniques such as peripheral gating, clock throttling, and partial memory retention reduce idle power9. OS-level power governors orchestrate state transitions based on workload patterns. This achieves fine-grained system-wide energy savings without performance penalty.

C. Task-Level Power Management

Tasks are scheduled to minimize energy rather than maximize throughput. Background tasks are batched to reduce frequent wake-ups. Real-time tasks are aligned with DVFS settings to avoid peaks in power. This provides benefit of improved energy efficiency at the application level with minimal hardware overhead[12].

D. High-Level Synthesis (HLS)

Low power Resource binding maps operations to hardware units in an energy-efficient manner thereby reducing dynamic switching and lower leakage in unused hardware.[15] Voltage-Island-based scheduling enables aggressive energy reduction without compromising performance by mapping critical path operations to high-Vdd regions for timing closure, Slack-tolerant operations to low Vdd islands to reduce dynamic power ($\propto V^2$). HLS can shape the instantaneous power profile to avoid high-current spikes thereby reducing IR drops

IV. MEMORY OPTIMIZATION IN EMBEDDED SYSTEMS

Memory is a central performance and power bottleneck in embedded Systems.[1] Modern applications such as image/video processing, multimedia codecs, and machine learning workloads operate as iterative dataflow graphs (DFGs), where multiple processing elements concurrently access shared data. Due to limited on-chip memory and expensive off-chip DRAM transactions, scheduling and communication patterns strongly influence memory consumption, power utilization, and timing predictability.
An embedded System typically incorporates a layered memory hierarchy.

- Registers / Hardware Local Memory (HLM): Lowest latency and power, limited capacity1.
- Software Local Memory(SLM) / On-chip SRAM (OCM)- Used by processors and accelerators[4].
- Caches - Improve locality but incur tag-matching overhead and unpredictability.
- External DRAM / Flash MemoryHigh capacity but suffers from long latency and10–100× higher access energy[13].

Embedded platforms suffer from multiple architectural and scheduling constraints with key design challenges –

- Limited on-chip SRAM capacity
- Expensive off-chip DRAM access[1]
- Power-hungry communication interfaces
- Shared memory contention among CPU & accelerators[4]
- Thermal limits causing throttling
- Synchronization overhead in multi-core environments
- Memory Overhead Contributors such as parallel execution effects due to concurrent execution of multiple tasks[1] causing dataset overlap and increased peak memory occupancy and pipeline buffering[3] and retiming which requires additional buffers.
- Communication via Shared Memory- If tasks are partitioned poorly between HW and SW then data is written/read frequently and DRAM bandwidth becomes oversaturated[4]. Energy consumption increases due to bus toggling.

Memory and communication are the largest energy contributors, especially in:

- IoT edge devices
- Real-time platforms
- Mobile processors with always-on sensors

Since shared memory[15] often follows exclusive-read exclusive-write semantics, simultaneous hardware/software accesses introduce serialization and burst contention. Thus Scheduling and memory allocation decisions directly impact system-level efficiency.

- Memory Bandwidth Optimization- Effective scheduling ensures bandwidth constraints are never violated. Techniques integrated:
  a) Loop tiling / blocking to exploit locality and reduce memory transactions[1].
  b) Data reuse via scalar replacement
  c) Stream buffers to minimize repeated reads
  d) Banked memory for parallel access[3]
  e) Burst-access scheduling to minimize row activations in DRAM.

  These methods prevent pipeline stalls due to communication starvation and also lowers per-access energy and achieves reduced overall memory power.

- Address Generation Optimization- Address generation in Systems consumes non-trivial power/latency. Optimizations include:
  a) Pointer arithmetic simplification[1]
  b) Induction variable substitution
  c) Strength reduction (shifts vs. multiplication)

These methods reduce pipeline latency and code size, enhancing predictability.

V. Resource Optimization in Embedded systems

Resource optimization aims to achieve maximum efficiency from limited computational elements, memory subsystems, and bandwidth resources. Modern Systems incorporate heterogeneous processing elements such as CPUs, DSPs, programmable accelerators, and shared communication channels. Each must operate under constrained silicon area, energy budget, and predictable latency constraints. The following play a major role in resource optimization-

- Hardware–Software Partitioning- Hardware–Software Partitioning strategy depends on
  a) Execution time profiling
  b) Task-level parallelism
  c) Communication intensity
  d) Data path complexity

  Hardware–software co-design is pivotal to minimizing execution latency and resource overuse. The Hardware acceleration improves throughput and determinism whereasSoftware execution offers flexibility and low design cost. The key partitioning principle for optimum Hardware– software co-design[4] involves tasks providing high speed-up with low silicon area mapped to hardware, while tasks with high area demand remaining in software.

- Communication Resource-Shared communication channels (buses, NoCs, DMA paths) create non-trivial delays if scheduling is not carefully designed. Uncontrolled access results in:
  a) Bandwidth saturation
  b) Increased wait states
  c) Pipeline stalls
  d) Degraded real-time guarantees

An efficient scheduler must:

a) Serialize conflicting accesses
b) Enforce exclusive access to shared memory
c) Cluster task execution to minimize switching overhead

- Memory Resource- Memory size and bandwidth limitations are always modeled as hard constraints. The scheduler evaluates worst-case memory occupancy dynamically to ensure no pipeline stage exceeds available storage[4].
- Architecture Adaptability - The optimization framework must support:
  a) Custom accelerators for compute-heavy blocks[12]
  b) Variable memory allocations to match algorithm behavior
  c) Dynamic resource binding depending on workload scenario

This makes the System scalable for large data computations, sensor fusion, and wireless signal-processing applications. Thus resource optimization using efficient HW–SW partitioning, memory and bandwidth allocation gives the benefit of Lower system cost and area.

VI. Performance Optimization in Embedded Systems

Performance in embedded systems depends not only on clock speed or instruction count but largely on how efficiently hardware resources are orchestrated**.**

- Software-Level Optimization
  a) Compiler-Assisted Optimization-Utilizing optimized compilation strategies[9] (O2/O3 flags, link-time optimization) leads to:
     - Reduced instruction count
     - Enhanced ILP (instruction-level parallelism)
     - Faster loop execution (unrolling, pipelining)
  b) Algorithmic Refinement- Performance gain is vastly influential by algorithm design. Efficient algorithms outperform hardware upgrades in many cases.

Technique	Benefit
Fixed-point arithmetic	Tremendous speed up in DSP
Precomputation tables	Eliminates redundant operations
Reduced branching	Improves pipeline predictability[13]

Table 3: Algorithm-Level Optimization Techniques

- Hardware-Level Optimization
  a) Hardware Acceleration- Mapping specific functions— FFT, ML inference, encryption to accelerators:
     - Eliminates software load
     - Reduces CPU interrupt overhead

- Improves real-time determinism

b) Cache-Conscious Data Layout[13]- Data placed to maximize locality:
- Reduces cache thrashing
- Allows smoother pipeline flow
- Avoids expensive stall cycles

- RTOS and Interrupt-Level Performance- RTOS tuning ensures critical tasks always meet real-time constraints. Real-time performance rests upon:
a) Priority-based task scheduling - deadlines dictate resource allocation[16]
b) Low interrupt latency - reducing response time[12]
c)Balanced context switching- limiting kernel overhead[7]
- DMA-Driven Communication Acceleration- A common performance issue is interrupt-driven sampling with high peripheral latency overwhelms CPU capacity[1]. The Original bottlenecks are-
  - Multiple interrupts per ADC conversion
  - Software-triggered SPI transactions
  - Kernel-to-user context switching overhead

a) Optimized DMA-based solution:
- ADC End-of-Conversion triggers DMA
- SPI clock + data read executed autonomously
- CPU only processes bulk data post-transfer

b) Benefits include
- CPU load reduced drastically
- No lost samples
- Throughput greatly increased
- Fully deterministic timing under real-time OS

This proves that performance optimization demands a system-level viewpoint -understanding not only the CPU but every peripheral interaction. Performance optimization using CPU/DMA acceleration, RTOS tuning, algorithm refinement gives the benefit of Higher throughput with real-time guarantee.

- Performance–Power Trade-Offs

a) Many performance-boosting techniques increase power consumption. DVFS mitigates this by dynamically adjusting frequency:

Workload	DVFS Policy	Result
Heavy compute	High frequency	Meet deadlines[9]
Idle/low activity	Low frequency + deep sleep	Save power

Table 4: Workload-Aware DVFS Power Management

b) Strong Interdependency- Performance enhancement (e.g., pipelining[1] →parallelism) requires:
- More buffering → more memory → more leakage
- Increased bandwidth → higher communication energy

Resource & performance optimizations must be harmonized with power & memory constraints for embedded Systems to remain feasible.

VII. Hardware–Software Framework Co-Optimization

Optimizing embedded Systems cannot be approached as a collection of independent tactics. The design must holistically coordinate:

- Power consumption
- Memory hierarchy and usage
- Communication and hardware resources
- Execution timing and real-time performance

a) Unified Optimization Perspective- Improving performance almost always demands- more parallelism, more buffer memory and higher bandwidth which causes increased energy and silicon cost**.** The real objective is not maximizing performance — it is achieving the right performance within power, memory, bandwidth, and area constraints[12]. Co-optimization ensures:
- Higher throughput within available hardware
- Reliable timing under real workloads
- Controlled energy dissipation
- Balanced memory pressure energy dissipation
- Balanced memory pressure

VIII. Emerging Trends and Future Directions

Modern embedded systems focus on edge intelligence — local computation under low-energy constraints. Future design priorities include:

a) Near-Threshold Computing[15]- Operating transistors near threshold voltage (0.3–0.5V):
- Up to 10× power reduction
- Reduced frequency → suited to IoT and wearables

b) Approximate Computing- Applicable when perfect accuracy not required such as Deep learning inference and Image/audio processing[9]. This reduces hardware complexity, switching activity, power consumption

c) ML-Based Power & Performance Management Machine learning predicts[15]:
- Workload patterns
- Optimal frequency scaling
- Sleep/wake timing
- Moves from reactive control **to** predictive optimization

d) RISC-V for Custom Low-Power Systems[3]- Open ISA allows:
   - Application-specific instruction extensions
   - Tight accelerator integration
   - Transparent power control

IX. CONCLUSION

Embedded Systems operate under strict energy, memory, and hardware limits while needing high performance and reliable real-time behavior. This paper unified four key optimization domains — power, memory, resource, and performance — into a cohesive hardware–software codesign strategy.

Key contributions of this unified approach include:

- Reduced dependency on off-chip memory and system buses
- Minimized leakage and dynamic energy consumption
- Improved parallel task execution without resource contention
- Deterministic real-time guarantees through optimized scheduling
- Scalable accelerator integration for compute intensive workloads

Future embedded computing must continue evolving toward intelligent, context-aware co-optimization, leveraging predictive resource management and advanced architectures like RISC-V to meet the demands of edge AI and pervasive IoT environments.

ACKNOWLEDGMENT

The authors are extremely grateful to Shri. Anindya Biswas, DS & Director RCI and Sri Vijaya Shankar OS & Group director, DECS for his continuous support and encouragement.

REFERENCES

1. G. Eason, B. Panda PR, Catthoor F, Dutt ND, Danckaert K, Brockmeyer E, Kulkarni C, et al. Data and memory optimization techniques for embedded systems. ACM Trans Des Autom Electron Syst. 2001;6(2):149-206. doi:10.1145/375977.375978
2. Izosimov V, Pop PP, Peng Z. Synthesis of fault-tolerant schedules with transparency/performance trade-offs for distributed embedded systems. In: Proceedings of the Design, Automation and Test in Europe Conference (DATE). Munich; 2006. doi:10.1109/DATE.2006.244067
3. Grun P, Dutt N, Nicolau A. Access pattern based local memory customization for low power embedded systems. In: IEEE Int. Conf. Proceedings. IEEE; 2001. p. 778.
4. Ge Z, Lim HB, Wong WF. Memory hierarchy hardware–software co-design in embedded systems. Dept. of Computer Science, National University of Singapore; 2007.
5. Karakatic S, Podgorelec V, Hericko M. Optimization of combinational logic circuits with genetic programming. ElektronElektrotech. 2013;19(7):86-9. doi:10.5755/j01.eee.19.7.5169
6. Fohler G. Design optimization of time- and costconstrained fault-tolerant embedded systems for safety-critical applications. Univ. of Kaiserslautern; 2000. doi: 10.1109/DATE.2005.116
7. Laprie J, Arlat J, Beounes C, Kanoun K. Definition and analysis of hardware- and software-fault-tolerant architectures. Computer. 1990;23(7):39-51. doi:10.1109/2.56851
8. Austin TM, Bertacco V, Blaauw D, Mudge TN. Opportunities and challenges for better-than-worstcase design. In: ASP-DAC 2005: Asia and South Pacific Design Automation Conference. IEEE; 2005. doi:10.1109/ASPDAC.2005.1466113
9. Su CL, Tsui CY, Despain AM. Low power architecture design and compilation techniques for highperformance processors. ACAL Technical Report ACAL-TR-94-01; 1994. doi: 10.1109/CMPCON.1994.282878
10. Hardware/software co-design in multiprocessor embedded systems and IoT. J TheorApplInf Technol. 2024;102(2):719-21.
11. Ayari R. Optimization and mining methods for effective real-time embedded systems. ÉcolePolytechnique de Montréal; 2018.
12. Chatha KS, Vemuri R. Hardware–software partitioning and pipelined scheduling of transformative applications. IEEE Trans VLSI Syst. 2002;10(3):193-204.
13. Bahar RI, Albera G, Manne S. Power and performance tradeoffs using various caching strategies. In: ProcIntSymp Low Power Electron Des (ISLPED). ACM; 1998. doi:10.1145/285930.285982
14. Qasim S. Analyzing optimization strategies and techniques for power management in embedded systems. Int J IndManufSyst Eng. 2021;6(2):35-42. doi:10.11648/j.ijimse.20210602.12
15. Benini L, De Micheli G. System-level power optimization: techniques and tools. ACM Trans Des Autom Electron Syst. 2000;5(2):115-92.
16. Åkerholm M, Möller A, Hansson H, Nolin M. Towards a dependable component technology for embedded system applications. Mälardalen Univ.;(n.d.).doi:10.1109/WORDS.2005.52