MANTECH Publications
Journal of Research in VLSI Design Tools and Technology
Volume 3 Issue 2, May-August 2018

Hardware-Software Co-Design for Efficient System-on-Chips (SoCs)

Vimal Mehta[1], Subham Singh[2], Kunal Deshmukh[3]
Associate Professor, professor
Department of Hardware Acceleration and FPGA Research
Calicut University – Arts & Science Faculty, Kerala
Email: *Vimalmehta5f@gmail.com[1], subham.singghs@yahoo.com[2], kunal17deshmukh@rediffmail.com[3]*

Abstract

Hardware-Software Co-Design (HSCD) has emerged as a pivotal methodology in the development of high-performance and energy-efficient System-on-Chips (SoCs). With the increasing complexity of modern SoCs driven by multi-core processors, heterogeneous architectures, and AI workloads, traditional sequential design approaches often fall short in meeting performance, power, and area (PPA) constraints. HSCD allows concurrent optimization of hardware and software components, enabling early exploration of design trade-offs and accelerating time-to-market. This paper reviews the state-of-the-art in HSCD techniques, explores modeling and simulation approaches, discusses design automation tools, and presents real-world case studies demonstrating efficiency gains in SoCs. The challenges and future trends, including AI-assisted co-design and reconfigurable platforms, are also analyzed.

Keywords: *Hardware-Software Co-Design, System-on-Chip, Embedded Systems, Design Automation, Energy Efficiency, Heterogeneous Architectures, Performance Optimization.*

Page 84-97 © MANTECH PUBLICATIONS 2018. All Rights Reserved

Introduction

The exponential growth in computational demands of modern applications such as artificial intelligence (AI), high-performance computing (HPC), and multimedia processing has pushed SoCs to new limits. Traditional SoC design approaches, where hardware and software are developed sequentially, often result in suboptimal performance, higher power consumption, and longer development cycles. Hardware-Software Co-Design (HSCD) addresses these issues by enabling concurrent design and verification of hardware and software components.

HSCD allows designers to explore design trade-offs such as energy efficiency versus performance and implement architectures tailored to application-specific needs. By integrating tools, methodologies, and simulation frameworks, HSCD reduces development time and improves overall SoC efficiency.

2. Background

Hardware-Software Co-Design (HSCD) for System-on-Chips (SoCs) builds on decades of embedded system development and the evolution of integrated circuit technologies. Understanding the background requires examining both the historical trajectory of SoC design and the foundational principles of co-design.

2.1 Evolution of SoC Design

System-on-Chips (SoCs) have transformed the electronics industry by integrating multiple computational, memory, and peripheral components onto a single silicon die. This integration enables compact, high-performance, and energy-efficient devices suitable for applications ranging from smartphones to autonomous vehicles.

2.1.1 Early SoC Architectures

Initially, SoCs were designed around a single general-purpose processor (CPU), with application-specific hardware accelerators added to improve efficiency for specific tasks. Examples include digital signal processors (DSPs) for audio/video processing or network interface controllers for communication tasks. Early SoCs emphasized functionality over

Page 84-97 © MANTECH PUBLICATIONS 2018. All Rights Reserved

performance optimization, often resulting in over-provisioned hardware that consumed unnecessary power.

2.1.2 Growth of Complexity

As consumer demand for multifunctional and high-performance devices increased, SoCs evolved into heterogeneous architectures containing multiple cores, GPUs, memory hierarchies, and specialized accelerators for AI, security, and multimedia processing. This complexity introduced significant challenges in terms of **power consumption**, **silicon area**, **thermal dissipation**, and **design verification**. The sequential design paradigm—where hardware is designed first, followed by software development—proved inadequate to meet these multi-dimensional constraints.

2.1.3 Need for Co-Design

The rising complexity and performance demands made it clear that a holistic design approach was necessary. Hardware-Software Co-Design emerged as a methodology to address these challenges by:

- Allowing **early exploration of hardware/software trade-offs**, such as deciding whether an algorithm should be implemented in software or accelerated in hardware.
- Reducing **time-to-market** by enabling parallel development of software and hardware.
- Improving **system efficiency** by optimizing energy, latency, and throughput across both domains.

Example: In AI-enabled mobile SoCs, matrix multiplications for neural networks can be offloaded to a dedicated hardware accelerator, while control flow and scheduling remain in software. Such co-design ensures both energy efficiency and high throughput.

2.2 Principles of Hardware-Software Co-Design

The essence of HSCD lies in the **collaborative and concurrent development** of hardware and software, focusing on holistic system-level optimization rather than isolated component performance. The key principles include:

Page 84-97 © MANTECH PUBLICATIONS 2018. All Rights Reserved

2.2.1 Concurrent Design

Rather than sequentially developing hardware and software, HSCD encourages **parallel design and verification**. For instance, while hardware engineers develop an accelerator for video encoding, software engineers can implement the driver and application logic simultaneously. This concurrent approach helps uncover integration issues early, preventing costly redesigns.

2.2.2 Early Validation

HSCD emphasizes **simulation and prototyping at early design stages**. High-level models and virtual prototypes allow designers to test system behavior before committing to silicon. For example, cycle-accurate simulations can predict bottlenecks in data transfer between a CPU and a hardware accelerator, enabling early optimization.

2.2.3 Partitioning

Partitioning refers to deciding **which functions are best implemented in hardware versus software**. The goal is to optimize metrics such as performance, power, area, and flexibility. Partitioning strategies include:

- **Profiling-based:** Identify computation-intensive software tasks suitable for hardware acceleration.
- **Graph-based:** Represent tasks as dataflow graphs, mapping them onto hardware/software based on dependencies and execution cost.
- **ML-assisted:** Use machine learning to predict optimal partitioning decisions based on historical designs.

2.2.4 Iterative Optimization

HSCD is inherently **iterative**, refining both hardware and software based on system-level metrics. Designers continually evaluate trade-offs between energy consumption, latency, and throughput. Iterative cycles may involve redesigning hardware accelerators, optimizing software kernels, or adjusting memory hierarchies to achieve the best balance.

Page 84-97 © MANTECH PUBLICATIONS 2018. All Rights Reserved

2.2.5 System-Level Perspective

A defining principle of HSCD is its **system-level orientation**. Unlike isolated hardware or software optimization, co-design focuses on the interaction between components. This includes communication delays, memory access patterns, and synchronization overheads. A system-level perspective ensures that optimizations in one domain do not negatively impact the other.

2.3 Illustrative Example

Consider a modern mobile SoC designed for video processing:

- **Hardware components:** GPU, DSP, video encoder/decoder, memory controller.
- **Software components:** Operating system, video player, driver stack, scheduling algorithms.

Through HSCD, designers can:

1. Move compute-intensive video decoding tasks to the DSP.
2. Keep video playback scheduling in software for flexibility.
3. Simulate memory bandwidth usage to avoid contention.
4. Iterate the design until achieving minimal power consumption while maintaining 60 FPS video playback.

This example highlights how co-design enables **efficient utilization of hardware resources** while maintaining software flexibility—a trade-off impossible in traditional sequential design.

3. Hardware-Software Co-Design Methodologies

Hardware-Software Co-Design (HSCD) methodologies define the strategies and workflows used to concurrently develop hardware and software components of a system-on-chip (SoC). Selecting the right methodology is critical, as it impacts performance, energy efficiency, design flexibility, and development time. Broadly, HSCD methodologies can be categorized into **Top-Down**, **Bottom-Up**, and **Partitioning-Based** approaches.

3.1 Top-Down Approach

The **Top-Down Approach** starts from **high-level system requirements** and progressively refines the design into hardware and software components. It is especially suited for complex SoCs with

Page 84-97 © MANTECH PUBLICATIONS 2018. All Rights Reserved

stringent performance, power, or area constraints, such as AI accelerators, automotive SoCs, or multimedia processors.

Workflow:

1. **System Specification:** Define system goals, functional requirements, performance targets, and power budgets.
2. **Architectural Design:** Identify system architecture, including CPU cores, accelerators, memory hierarchy, interconnects, and peripheral modules.
3. **Hardware-Software Partitioning:** Determine which tasks are implemented in hardware versus software.
4. **Implementation:** Develop hardware modules and software routines in parallel, following architectural specifications.
5. **Verification & Iteration:** Simulate the entire system to validate functional correctness, performance, and power consumption.

Advantages:

- Provides **high control over performance and energy optimization**.
- Ensures **system-level correctness** before hardware is fabricated.
- Facilitates **early identification of bottlenecks** through simulation.

Disadvantages:

- Requires **detailed upfront specifications**, which may limit flexibility.
- Longer initial design time compared to Bottom-Up methods.

Example:

In designing an AI inference SoC, designers first define the number of neural network layers to accelerate, memory bandwidth requirements, and latency targets. They then design custom matrix multiplication accelerators in hardware while implementing scheduling, data pre-processing, and control in software.

3.2 Bottom-Up Approach

The **Bottom-Up Approach** begins with **existing hardware IPs (Intellectual Property cores) and software modules**, integrating them into a complete system. This method is faster and often

Page 84-97 © MANTECH PUBLICATIONS 2018. All Rights Reserved

used in applications where time-to-market is critical or where reliable pre-validated IPs are available.

Workflow:

1. **IP/Module Selection:** Identify existing hardware blocks (e.g., CPU cores, DSPs, GPU cores) and software libraries.
2. **Integration:** Connect hardware IPs through interconnects and adapt software to interface with the selected hardware.
3. **Optimization:** Adjust parameters, memory allocation, and software scheduling to meet system requirements.
4. **Testing & Validation:** Validate the integrated system through simulations and prototypes.

Advantages:

- Faster development cycle due to reuse of existing modules.
- Lower design risk since IPs are already verified.

Disadvantages:

- **Limited flexibility for optimization**, as existing IPs may not perfectly match system requirements.
- May require additional **hardware-software adapters** to resolve interface mismatches.

Example:

In multimedia SoCs, designers may use an off-the-shelf video decoding IP and integrate it with a custom GPU. Software is adjusted to schedule video frames efficiently across hardware modules, reducing latency while reusing verified IPs.

3.3 Partitioning Strategies

Partitioning is a **core activity in HSCD**, as it determines which functionalities are implemented in hardware versus software. Effective partitioning is essential for balancing performance, power consumption, area, and cost.

3.3.1 Profiling-Based Partitioning

- **Method:** Software execution is profiled to identify computation-intensive tasks.
- **Hardware Mapping:** These tasks are offloaded to hardware accelerators for performance improvement.

Page 84-97 © MANTECH PUBLICATIONS 2018. All Rights Reserved

- **Example:** Profiling a DSP application may reveal that Fast Fourier Transform (FFT) consumes 60% of CPU cycles. This function is then implemented as a hardware accelerator to reduce latency and energy consumption.

3.3.2 Graph-Based Partitioning

- **Method:** The system is modeled as a task dependency graph (nodes = tasks, edges = data dependencies).
- **Hardware-Software Split:** Tasks are mapped to hardware or software based on computational intensity, data communication costs, and execution constraints.
- **Example:** In a video encoder SoC, filtering tasks (highly parallelizable) are assigned to hardware, while entropy coding (sequential, control-heavy) remains in software.

3.3.3 Machine Learning-Assisted Partitioning

- **Method:** Historical design data and performance metrics are fed into ML models that predict optimal partitions.
- **Advantages:** Handles complex systems where manual analysis is impractical. Can adapt to dynamic workload changes in runtime-adaptive SoCs.
- **Example:** An AI accelerator designer trains a model to determine which layers of neural networks benefit most from hardware acceleration under different memory and bandwidth constraints.

4. Modeling and Simulation

In Hardware-Software Co-Design (HSCD), **modeling and simulation** play a critical role in reducing design risks and shortening development time. Before committing to silicon fabrication, designers must verify the functional correctness, performance, and power efficiency of SoCs. Modeling and simulation frameworks provide the **virtual environment** where hardware and software components can be tested concurrently, allowing for early detection of design flaws, performance bottlenecks, and integration issues.

Page 84-97 © MANTECH PUBLICATIONS 2018. All Rights Reserved

4.1 High-Level Modeling

High-Level Modeling refers to describing hardware at an abstract, algorithmic level, typically using high-level programming languages such as C, C++, or SystemC. This abstraction enables **early design exploration** without needing to write detailed RTL (Register Transfer Level) code.

Key Advantages:

- **Faster Development:** Designers can implement and test hardware functionality without dealing with low-level timing or gate-level details.
- **Design Space Exploration:** Multiple architectural alternatives can be evaluated quickly for performance, area, and power.
- **Early Verification:** Algorithms can be validated before committing to hardware, reducing costly redesigns.

High-Level Synthesis (HLS):

HLS tools automatically translate high-level descriptions into RTL, bridging the gap between algorithm development and hardware implementation.

Example Tools:

- **Xilinx Vivado HLS:** Converts C/C++ code to FPGA-ready hardware modules.
- **Cadence Stratus HLS:** Generates cycle-accurate RTL for ASIC or FPGA designs.
- **Intel FPGA HLS Compiler:** Optimizes high-level designs for Intel FPGA architectures.

Example Application:

Consider an AI accelerator for convolutional neural networks (CNNs). Using HLS, designers can describe convolution operations in C++ and quickly evaluate multiple hardware architectures, such as pipelined or parallel MAC (Multiply-Accumulate) units, before generating RTL for FPGA or ASIC implementation. This allows **rapid exploration of trade-offs** between throughput, area, and energy consumption.

4.2 Virtual Prototyping

Virtual Prototyping involves creating a **software-based simulation of the SoC** that mimics both functional and temporal behavior. Virtual prototypes can be at various levels of abstraction:

- **Functional Simulation:** Validates the logical behavior of the system. Timing details are abstracted to allow faster simulation.

Page 84-97 © MANTECH PUBLICATIONS 2018. All Rights Reserved

- **Cycle-Accurate Simulation:** Models timing at the clock-cycle level, capturing communication delays, memory contention, and bus arbitration.

Key Benefits:

- Detects **integration issues** between hardware and software before physical fabrication.
- Allows **performance profiling** to identify bottlenecks, such as memory latency or inefficient hardware-software interfaces.
- Enables **early software development**, as software engineers can test code on the virtual platform without waiting for hardware availability.

Example Tools:

- **Synopsys Platform Architect:** Allows system-level modeling and architectural exploration.
- **Mentor Graphics Vista:** Offers virtual prototyping for embedded systems.
- **Gem5:** Widely used for academic research in multi-core and heterogeneous SoC simulation.

Illustrative Example:

In a multimedia SoC, a virtual prototype can simulate a video decoding pipeline. Functional simulation ensures that the output video matches expectations, while cycle-accurate simulation identifies that memory bandwidth is insufficient, causing frame drops. Designers can then adjust memory architecture or scheduling strategies before hardware fabrication.

4.3 Hardware-in-the-Loop (HIL)

Hardware-in-the-Loop (HIL) simulation combines **real hardware components** with a **simulated environment** to validate real-time interactions. HIL is particularly important for **embedded systems** that interact with dynamic physical processes, such as automotive, robotics, or IoT devices.

Key Features:

- Real hardware executes tasks while the surrounding environment is simulated in software.
- Enables **real-time validation**, detecting timing, communication, and control issues that pure software simulation may miss.
- Provides a bridge between prototyping and full system deployment.

Page 84-97 © MANTECH PUBLICATIONS 2018. All Rights Reserved

Example Application:

In an autonomous vehicle SoC:

- The hardware acceleration unit for object detection is physically present.
- The software controlling vehicle motion is simulated.
- The HIL system simulates sensor inputs, traffic scenarios, and environmental conditions.

HIL allows designers to test **edge cases and safety-critical scenarios** without deploying the system in a real vehicle, significantly reducing risk.

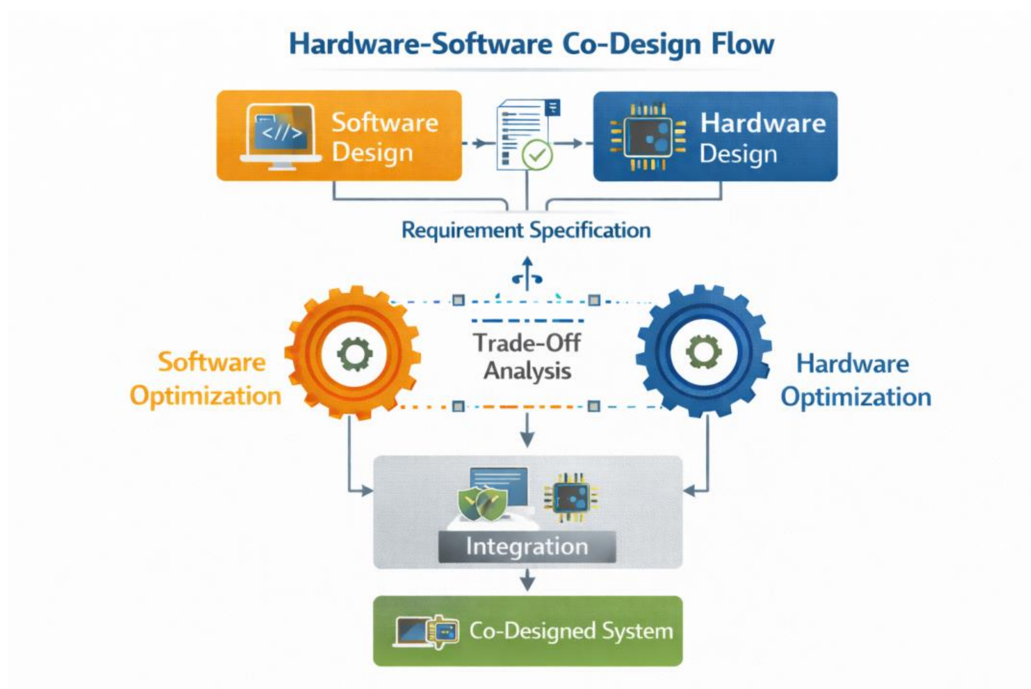

Figure 1: Hardware-Software Co-Design Flow

5. Design Automation Tools

Several tools facilitate HSCD:

- **Xilinx Vivado HLS**: For FPGA-based co-design.
- **Cadence Stratus HLS**: Supports high-level hardware synthesis and verification.
- **Synopsys Platform Architect**: Provides architecture exploration and power/performance modeling.

Page 84-97 © MANTECH PUBLICATIONS 2018. All Rights Reserved

- **Open-Source Alternatives**: Tools like Chisel and MyHDL enable academic and prototyping designs.

Table 1: Popular HSCD Tools and Their Features

Tool Name	Platform	Key Features	Typical Use Case
Xilinx Vivado HLS	FPGA	High-level synthesis, IP integration	FPGA accelerators
Cadence Stratus HLS	ASIC/FPGA	Cycle-accurate synthesis, power analysis	Custom SoC design
Synopsys Platform Architect	ASIC	Architecture exploration, simulation	Multi-core SoCs
Chisel	FPGA/ASIC	Open-source hardware construction	Academic prototyping
MyHDL	FPGA/ASIC	Python-based HDL	Rapid prototyping

6. Case Studies

6.1 AI Accelerator SoCs

AI workloads demand high computational throughput and low latency. HSCD enables moving matrix multiplications and convolutional operations to hardware accelerators while managing control and scheduling in software. Studies show up to 3× energy efficiency improvement over CPU-only implementations.

6.2 Multimedia SoCs

Video and audio processing pipelines benefit from co-design, with hardware handling filtering, compression, and encoding while software manages memory buffering and user interface. Co-design reduced latency by 40% in real-time streaming applications.

7. Challenges in Hardware-Software Co-Design

- **Toolchain Integration**: Interoperability between software simulation and hardware synthesis tools remains challenging.

Page 84-97 © MANTECH PUBLICATIONS 2018. All Rights Reserved

- **Design Complexity**: Heterogeneous SoCs require careful coordination of multiple IP blocks.
- **Verification Overhead**: Co-verification of hardware and software is computationally intensive.
- **Power and Thermal Management**: Optimizing energy without compromising performance is non-trivial.

8. Future Trends

8.1 AI-Assisted Co-Design

Machine learning can predict optimal hardware-software partitions, simulate performance, and suggest architectural improvements automatically.

8.2 Reconfigurable Architectures

Dynamic reconfigurable SoCs enable adaptive allocation of hardware accelerators to match software workloads, improving energy efficiency and flexibility.

8.3 Open-Source Co-Design Platforms

Emerging open-source frameworks allow collaborative development, facilitating rapid prototyping and academic research.

Conclusion

Hardware-Software Co-Design is a transformative methodology for developing efficient SoCs. By enabling concurrent design, early validation, and intelligent partitioning, HSCD addresses the performance, energy, and area challenges of modern embedded systems. The integration of advanced simulation tools, high-level synthesis, and AI-assisted methodologies further enhances design efficiency. While challenges remain, particularly in toolchain integration and verification complexity, ongoing research promises faster, more flexible, and energy-efficient SoCs.

References

1. Wolf, W. *Hardware-Software Co-Design of Embedded Systems*. IEEE Press, 2019.
2. Marwedel, P. *Embedded System Design: Modeling, Synthesis, and Co-Design*. Springer, 2021.

Page 84-97 © MANTECH PUBLICATIONS 2018. All Rights Reserved

3. De Micheli, G. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, 2017.
4. Xilinx. *Vivado HLS User Guide*. Xilinx Inc., 2022.
5. Cadence. *Stratus HLS Documentation*. Cadence Design Systems, 2022.
6. Synopsys. *Platform Architect: Architecture Exploration Guide*. Synopsys Inc., 2021.
7. Zhang, C., et al. "High-Level Synthesis for SoC Design: A Survey." *IEEE Trans. VLSI*, 2020.
8. Chen, Y., et al. "Efficient AI Accelerator Design via Co-Design." *Journal of Embedded Systems*, 2021.
9. Kumar, R., et al. "Multimedia SoCs: Hardware-Software Integration." *Microelectronics Journal*, 2020.
10. Lee, H., et al. "Machine Learning-Assisted Hardware-Software Partitioning." *ACM Trans. Embedded Computing Systems*, 2022.

Page 84-97 © MANTECH PUBLICATIONS 2018. All Rights Reserved