Embedded Systems

Theory

Christian Rossi

Academic Year 2024-2025

Abstract

The course delves into embedded systems, covering their characteristics, requirements, and constraints. It explores hardware architectures, including various types of software executors, communication methods, interfacing techniques, off-the-shelf components, and architectures suited for both prototyping and large-scale production.

In terms of software architectures, the course examines abstraction levels, real-time operating systems, complex networked systems, and the tools and methodologies used for code analysis, profiling, and optimization.

Students will also learn to analyze and optimize hardware/software architectures for embedded systems, focusing on managing design constraints and selecting appropriate architectures. Key topics include estimating and optimizing performance and power at various abstraction levels, project management, and designing for reuse.

Additionally, the course addresses run-time resource management and includes case studies to illustrate trade-offs based on application fields and system sizes.

Contents

CHAPTER 1

Introduction

1.1 Introduction

Embedded systems are characterized by their ubiquitous presence, low power consumption, high performance, and interconnected nature. These systems are commonly utilized in four main application contexts:

- *Public infrastructures*: safety is a primary concern to prevent potential attacks, and in some cases, latency is also crucial. Examples include highways, bridges, and airports.
- *Industrial systems*: reliability and safety are the predominant concerns. Key industries utilizing embedded systems include automotive, aerospace, and medical.
- *Private spaces*: this includes control systems for houses and offices.
- *Nomadic system*: these systems involve data collection related to the health and positions of animals and people, requiring both security and low latency.

In the future, embedded systems will evolve in several key ways:

- *Networked*: transitioning from isolated operations to interconnected, distributed solutions.
- *Secure*: addressing significant security challenges that impact both technical and economic viability.
- *Complex*: enhanced by advancements in nanotechnology and communication technologies.
- *Low power*: utilizing energy scavenging methods.
- *Thermal and power control*: implementing runtime resource management.

Usually, we process useful data locally and send the relevant data to the cloud less frequently. This approach conserves energy, thereby extending battery life. Due to time constraints, developing a device from scratch is often infeasible, so we typically collaborate with eco-alliance partners. The design process involves a multidisciplinary team, as it integrates multiple domains of expertise. The first prototype resulting from this design phase is called the Minimum Viable Product (MVP), which is the initial sellable version of the product.

1.1.1 Technological problems

Applications are expanding rapidly, pushing the need for mass-market compatibility and integrating into all aspects of life, which in turn drives up volumes. As technology evolves, the scale of integration grows, supported by new materials and programming paradigms. However, finding a balance in this progress is complicated by several factors. CMOS technology is reaching its physical limits, and the cost of developing new foundational technologies can often be unaffordable. Additionally, the power and energy demands create significant barriers, as does the exponential proliferation of data. Transitioning from invention to innovation is proving difficult, and the design methodologies in use today heavily exploit human effort—though this reliance on human ingenuity is, in some ways, a fortunate necessity.

Vertical applications Vertical applications requires various technologies to work ad desired such as sensors, computing units, storage, communication elments, and so on. Energy and power dissipation have become even more problematic with the introduction of the newest technology nodes, exacerbating existing challenges. Dependability, which encompasses security, safety, and privacy, is now a major concern. At the same time, the growing complexity of systems is reaching nearly unmanageable levels. Despite this, complexity continues to rise, driven by applications that build upon increasingly interconnected systems of systems.

1.2 Applications future

Future applications will be highly compute-intensive, requiring efficient hardware and software components across various domains, including embedded systems, mobile devices, and data centers. Key characteristics of these future applications include:

- *Compute-intensiveness*: they will demand significant computational resources, necessitating optimized hardware and software regardless of their application domain.
- *Connectivity*: these applications will be interconnected, either wired or wirelessly, and will often be online—globally interconnected through the Internet.
- *Physical entanglement*: they will be embedded within and capable of interacting with the physical world, not only observing but also controlling their environment. These systems will effectively merge into our everyday surroundings.
- *Intelligence*: these applications will possess the capability to interpret noisy, incomplete, analog, or remote data from the physical world, allowing for smarter interactions and decision-making.

All major future applications will exhibit these traits to varying degrees. The ongoing integration of the digital and physical worlds (manifested through the Internet of Things (IoT) and Cyber-Physical Systems (CPS)) will be driven by advances in cognitive computing, big data analytics, and data mining.

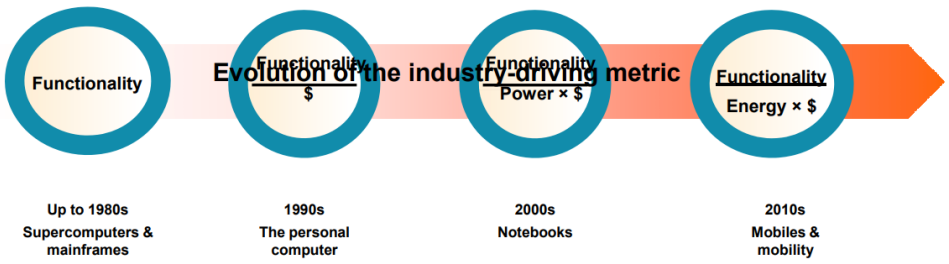


Figure 1.1: Industry requirements

The challenge is that optimizing for functionality, energy, and cost remains difficult:

$$\frac{\text{functionality}}{\text{energy} \times \text{cost}}$$

However, successfully optimizing this metric often leads to improvements across simpler metrics as well.

While the number of transistors in modern systems can continue to increase, a key challenge is that we cannot power all of them simultaneously. This limitation drives the need for innovative approaches to using extra transistors more efficiently. Techniques like multi-core and many-core processors, as well as domain-specific processors, are becoming essential. Additionally, heterogeneous processing combined with aggressive power management is crucial for optimizing performance across various tasks. As data generation continues to accelerate it becomes increasingly important to ensure that computation is carried out in the most efficient location. This efficiency is necessary to cope with the imbalance between the massive growth of data and the slowing progress of Moore's Law.

1.2.1 Transistors and cores

As transistors shrink in size, several challenges emerge. Process variation, physical failures, and aging mechanisms, such as negative bias temperature instability (NBTI), can degrade device performance over time. With very-large-scale integration (VLSI), packing more transistors into smaller spaces increases power density, which in turn creates thermal issues. Furthermore, traditional communication subsystems struggle to provide adequate power-performance trade-offs in these densely packed systems.

For instance, Network-on-Chip (NoC) designs can consume up to 30% of total chip power, and their performance plays a critical role in the efficiency of multicore architectures, which are increasingly necessary for improving system performance across various applications.

1.2.2 Silicon challenges

Despite silicon being a relatively good thermal conductor (about four times worse than copper), large chips can still experience significant temperature gradients, especially in high-performance CPUs. This creates hot spots that can affect chip reliability and performance. One practical solution involves dividing the chip into concentric rings and applying dynamic voltage and frequency scaling (DVFS) to each region. By fine-tuning voltage and frequency for each ring, it becomes possible to optimize performance while maintaining thermal balance, preventing excessive heat buildup in certain areas.

1.2.3 Frequency scaling

Modern systems address power management by partitioning chips into independent islands that operate at different voltage and frequency levels. This approach allows sections of the chip to be dynamically turned off when not in use, a process known as power gating. By adjusting power usage in this way, overall system energy efficiency is greatly improved without compromising performance when it is needed.

1.2.4 Cooling systems

Cooling systems are critical to managing the thermal output of modern computing hardware, and various methods are used depending on the specific requirements. Air cooling, while common, suffers from a low heat transfer coefficient (HTC), poor chip temperature uniformity, and requires large heat sinks and air ducts, particularly in data centers. It is also noisy and expensive to maintain. Water cooling offers an improvement, with better HTC, more uniform chip temperatures, smaller heat sinks, and fewer fans. Water cooling also allows for potential heat recovery, though it requires large pumps to function effectively.

Two-phase cooling systems present an even more efficient solution. They provide higher HTC, better chip temperature uniformity, and smaller pumps, along with isothermal coolant, which helps maintain consistent temperatures across the chip. This method excels at cooling hot spots and also allows for heat recovery. However, two-phase cooling systems suffer from low pump efficiency and reliability issues.

New innovations, such as thermosyphon cooling, are emerging as promising alternatives, offering advanced methods to maintain chip performance while effectively managing heat dissipation.

1.3 Productivity and planning

The revenue model can be visualized simply as a triangle, where the product's life is represented by a span of 2W, peaking at W. The time of market entry defines the triangle, representing market penetration, with the area of the triangle corresponding to total revenue. Any delay in market entry results in a loss, which is the difference between the areas of the on-time and delayed triangles.

The productivity gap, however, reveals a more challenging situation. In theory, increasing the number of designers on a team should reduce project completion time. In practice, though, productivity per designer tends to decrease due to the complexities of team management and communication, a phenomenon famously referred to as the mythical man-month (Brooks 1975). At a certain point, adding more designers can even extend project timelines, a classic case of too many cooks spoiling the broth.

1.3.1 Platform-based design

To enhance productivity, the design methodology must support reuse, particularly at higher abstraction levels, and this should be backed by standardization. As integrated systems increasingly require both digital and non-digital functionalities, this dual trend is captured by the International Technology Roadmap for Semiconductors. It emphasizes the miniaturization of digital functions, often referred to as More Moore, and functional diversification, known as More-than-Moore.

1.3.2 Technology trends

Two significant trends in technology development are System-on-Chip (SoC) and System-in-Package (SiP). SoC focuses on full integration and achieving the lowest cost per transistor, while SiP focuses on lowering the cost per function for the entire system. These architectures are complementary rather than competitive, each requiring distinct industrial approaches and advanced research and design knowledge. SoC emphasizes miniaturization, whereas SiP centers on integrating multiple components, necessitating different manufacturing competencies for each.

MEMS Micro-Electro-Mechanical Systems (MEMS) involve the creation of 3D structures using integrated circuit fabrication technologies and specialized micromachining processes, typically on silicon or glass wafers. MEMS devices include transducers, microsensors, microactuators, and other mechanically functional microstructures. Applications range from microfluidics (valves, pumps, and flow channels) to microengines (gears, turbines, combustion engines). Integrated microsystems combine circuitry and transducers to perform tasks autonomously or with the assistance of a host computer. MEMS components bridge the gap between the electrical and non-electrical world, where sensors receive inputs from non-electronic events and actuators output to them.

Energy scavenging Energy scavenging involves capturing energy from objects with temperature gradients. Another source of scavenging is vibrations, such as self-winding watches, which generate around 5 microwatts on average when worn and up to 1 milliwatt when shaken vigorously.

Wireless sensor nodes Wireless sensor nodes are small, battery-powered devices that monitor local conditions. These devices typically have limited resources and form nodes within a wireless network that covers a region or object of interest. Wireless sensor nodes enable new applications by collecting, fusing, reasoning, and responding to sensor data. These applications can lead to smarter systems in fields ranging from environmental monitoring to industrial automation.

CHAPTER 2

Microprocessors

2.1 Introduction

Microprocessors play a crucial role in embedded systems by implementing software algorithms that govern their functionality. The key features of microprocessor-based applications include:

- *Flexibility*: this encompasses maintainability and the ability to evolve the application. Software development becomes less complex and faster, verification processes are less critical, and a larger pool of software designers is available.
- *Time To Market* (TTM): this refers to the duration required to bring a product to the customer, emphasizing the importance of efficient development cycles.
- *Easy upgrade*: software solutions can be updated and improved with minimal disruption.
- *Cost*: the overall cost is volume-dependent. Design costs are incurred only once, while production expenses are primarily linked to the cost of silicon.

This approach is cost-effective since embedded systems often do not require the full processing power of a microprocessor; they typically utilize only a fraction of it. Consequently, microprocessors retain only the necessary functions and capabilities, resulting in reduced production costs and material consumption.

While an equivalent hardware solution may generally provide better performance, microprocessor architectures are optimized for flexibility, making them more generic and less specialized. Performance encompasses various factors beyond computing speed, such as energy consumption, power efficiency, memory footprint, and chip area. While software implementations are typically easier to develop than hardware solutions, modifying hardware to meet new requirements is often simpler, with performance advantages remaining on the hardware side.

Designing a software solution necessitates a comprehensive understanding of the microprocessor architectures available in the market. The characteristics of the problem (algorithm) should guide the designer, alongside non-functional constraints such as performance, cost, development time, and power consumption. The initial step before selecting a specific processor is determining the appropriate class of processor and the form in which it will be acquired.

2.2 Processors taxonomy

Processors can be categorized into two main types:

- *Application Specific Processors* (ASP): these processors are tailored for specific classes of applications that require high performance, handling numerous operations per second.
- *General Purpose Processors* (GPP): known as versatile four-season processors, GPPs are not optimized for any particular application, making them suitable for various types of tasks.

In typical embedded systems, a multi-processor solution often employs GPPs for supervising and controlling the activities of one or more ASPs.

Processors availability Processors can further be classified as follows:

- *Components Off The Shelf* (COTS): these are standard chips purchased off-the-shelf and mounted onto a printed circuit board (PCB) along with the necessary interfaces to integrate with the rest of the system.
- *Intellectual Property* (IP): this involves purchasing the design specifications of a microprocessor. There are several abstraction levels in IP:
  - *Soft-macro*: the microprocessor is described using Hardware Description Language (HDL) at the register-transfer (RT) level.
  - *Hard-macro*: the description includes details down to the layout level.

With the increasing prevalence of Programmable Logic Devices (PLDs), the use of IP is becoming more popular. Suppliers of Complex Programmable Logic Devices (CPLDs) and/or Field Programmable Gate Arrays (FPGAs) often provide one or more cores, sometimes even for free.

2.2.1 Selection process

The selection process is based on:

- *Class*: the nature of the algorithm and the operations and data to be processed are the primary drivers for selecting a processor class.
- *Form*: the target architecture of the system plays a crucial role.
- *Performance*: a key metric for performance is the average number of instructions per clock cycle (IPC/CPI), which is relative to the clock frequency and should be scaled for comparison across different architectures. MIPS (Million Instructions Per Second) serves as an absolute measure of throughput but can be misleading when comparing processors with different Instruction Sets (ISAs). For floating-point operations, MFLOPS is commonly used, while specialized architectures like Digital Signal Processors (DSPs) often use MMACS (Million Multiply-Accumulate Operations Per Second), and Network Processors (NPs) typically measure the average number of processed packets per time unit.

- *Power*: power efficiency is perhaps the most critical driver for embedded systems. Both average power and peak power are important metrics for estimating the maximum or average power consumption of the overall system. Initially, a combined measure of power and speed (related to performance) is often used.

Other important considerations include:

- *Memory*: the bandwidth and size requirements of the application can impose hard constraints. In some cases, only internal memory may be available, making external memory utilization impractical. Achieving sufficient bandwidth may require integrated memory solutions, and for complex systems, addressing space can be a critical factor.

- *Peripherals*: embedded systems often process external signals and control physical devices. Designers can either choose an architecture focused on computation and design the rest of the system accordingly or opt for a single-chip solution that integrates computing, interfacing, and peripherals. Selecting the appropriate microprocessor can necessitate using a PCB or a System on Chip (SoC) that consolidates all peripherals. Using a microcontroller simplifies integration challenges, albeit with fewer alternatives available.

- *Software*: many embedded applications have limited legacy code and primarily utilize standard functions and libraries. The availability of libraries can simplify both the design and validation processes, making the development of software solutions feasible that might otherwise be impractical. Access to an operating system and SDK specific to the processor is also vital, serving as the foundation for software development. Differences among SDKs are significant, and factors such as the software compilation flow, code quality, and flexibility offered to the designer are important. The availability and reliability of analysis tools, including debuggers, as well as documentation and reference designs, contribute to the ecosystem surrounding the microprocessor.

- *Packaging*: for COTS processors, various packages are available, differing in size, pinout, and materials.

- *Certifications*: different certification standards exist, including consumer, industrial, aerospace, automotive, and military certifications. Some certifications are tailored for specific fields, such as MISRA rules for automotive code. The availability of a component with the appropriate certification can be a critical constraint for its adoption.

2.3 General Pupropose Processors

General Purpose Processors (GPPs) are characterized by a generic architecture that is applicable across a wide range of fields. They are commonly found in PCs and Personal Digital Assistants (PDAs), and for embedded systems, they are suitable for low-end applications where energy efficiency and performance are not critical, and the application nature is quite heterogeneous. GPPs handle tasks such as controlling and managing slow interfaces with sensors and enabling interactivity through graphical user interfaces (GUIs). GPPs can be categorized into two main architectural styles:

- *Complex Instruction Set Computer* (CISC): features a large number of complex instructions.

- *Reduced Instruction Set Computer* (RISC): utilizes a smaller set of simple instructions.

2.3.1 Complex Instruction Set Computer

CISC architectures aim to allow each arithmetic or logic operation to access data and write results using any available addressing mode. They feature a fully orthogonal architecture and instruction set. However, in practice, the number of combinations of operations and addressing modes is often limited, typically with one operand referencing memory and the other being a register that acts as both source and destination. Despite the complexity, CISC instructions are encoded in variable-length formats, which necessitates more complex fetch and decode units and a higher speed for memory access.

The Arithmetic Logic Unit (ALU) implements various basic operations such as addition, subtraction, multiplication, division, logic operations, shifting, and comparisons. Some architectures even support vector instructions, enabling operations on multiple registers simultaneously, indicative of a Single Instruction Multiple Data (SIMD) architecture. The diversity in addressing modes and arithmetic/logical operations requires a complex data path that is challenging to optimize for speed. To enhance throughput, CISC processors often employ intricate pipelining techniques (e.g., up to 23 stages), which can be challenging to manage. Additionally, they may alter the order of execution for assembly instructions without changing the program's semantics. This complexity makes it difficult for compilers to predict instruction execution status, potentially leading to suboptimal instruction decoding. Modern CISC processors integrate hardware units that dynamically reschedule instructions, enabling out-of-order execution.

Although CISC architectures provide impressive performance, the focus on energy efficiency and cost has gradually shifted toward other architectures.

2.3.2 Reduced Instruction Set Computer

RISC architectures simplify instruction sets, utilizing a limited number of straightforward instructions. By the late 1980s, the declining prices of DRAMs and the increasing scale of integration allowed for more complex architectures on a single chip. CISC operations could be decomposed into multiple RISC instructions, streamlining architecture.

The primary advantages of RISC architectures include a simple instruction set and straightforward architecture, which simplify instruction decoding. RISC instructions typically have fixed lengths, facilitating balanced pipeline stages and higher clock speeds. Execution units, such as the ALU and Branch Processing Unit (BPU), benefit from simpler instructions, allowing RISC architectures to operate predominantly on registers, thus improving speed and predictability.

RISC employs a load/store architecture for input/output operations, which involves preloading data and writing results back to memory. While the limited number of addressing modes can aid performance, effective use of memory hierarchy is crucial for minimizing data access time.

When comparing RISC and CISC using the same high-level source code, RISC generally requires a higher number of instructions. RISC architectures do not have hardware mechanisms to resolve pipeline conflicts, but their predictable execution allows compilers to generate efficient code. Furthermore, RISC architectures typically exhibit lower power consumption due to their inherent simplicity compared to CISC architectures.

2.3.3 Superscalar

Superscalar architectures feature multiple execution units, enabling the simultaneous execution of more than one instruction per clock cycle. This parallelism enhances throughput but also

increases the complexity of control logic, impacting clock speed and power consumption. In these architectures, the microprocessor dynamically schedules instructions, relieving the compiler from managing code organization for specific hardware structures. However, a significant portion of the processor's resources is dedicated to implementing complex scheduling mechanisms and maintaining consistency.

2.3.4 Complex Instruction Set Computer and Reduced Instruction Set Computer

Some architectures combine out-of-order execution with superscalarity while maintaining compatibility with legacy code. The x86 instruction set serves as the de facto standard for CISC, where each instruction is decomposed into a series of simpler instructions that can be executed by an efficient core. This approach incurs additional hardware costs but merges the advantages of backward compatibility with the high performance achievable through a RISC core.

2.3.5 Very Long Instruction Word

To address the challenges associated with hardware scheduling complexity, Intel developed a class of processors known as EPIC (Explicitly Parallel Instruction Computing) or VLIW (Very Long Instruction Word). VLIW supports explicit parallelism, allowing multiple instructions to be executed simultaneously under program control. A VLIW instruction consolidates multiple elementary instructions, typically of RISC nature, into a single wide word with a fixed structure.

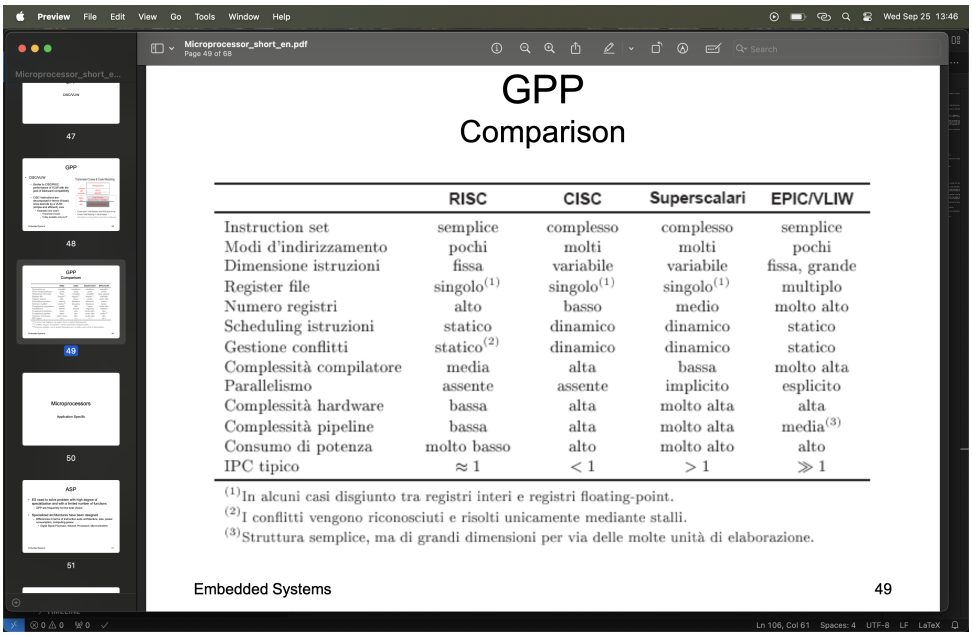
Once fetched from memory, the VLIW word is decoded in parallel, with individual RISC instructions dispatched to various execution units. This fixed structure shifts the complexity of scheduling onto the compiler, which must optimize algorithms to fit the specific architecture.

The goal of EPIC/VLIW is to transfer the burden of scheduling and code optimization to the compiler, resulting in simpler architectures, although with large sizes due to the number of execution units. Power consumption for VLIW architectures falls between that of simpler RISC and superscalar designs. Similar strategies have also been implemented in specialized architectures such as digital signal processors (DSPs) to achieve peak performance.

2.3.6 Complex Instruction Set Computer and Very Long Instruction Word

Similar to the CISC/RISC relationship, VLIW architectures aim for high performance while ensuring backward compatibility. CISC instructions are broken down into basic instructions that can be executed by a VLIW core, which is both simple and efficient.

2.3.7 Comparison



	RISC	CISC	Superscalari	EPIC/VLIW
Instruction set	semplice	complesso	complesso	semplice
Modi d'indirizzamento	pochi	molti	molti	pochi
Dimensione istruzioni	fissa	variabile	variabile	fissa, grande
Register file	singolo(1)	singolo(1)	singolo(1)	multiplo
Numero registri	alto	basso	medio	molto alto
Scheduling istruzioni	statico	dinamico	dinamico	statico
Gestione conflitti	statico(2)	dinamico	dinamico	statico
Complessità compilatore	media	alta	bassa	molto alta
Parallelismo	assente	assente	implicito	esplicito
Complessità hardware	bassa	alta	molto alta	alta
Complessità pipeline	bassa	alta	molto alta	media(3)
Consumo di potenza	molto basso	alto	molto alto	alto
IPC tipico	≈ 1	< 1	> 1	$\gg 1$

(1) In alcuni casi disgiunto tra registri interi e registri floating-point.
(2) I conflitti vengono riconosciuti e risolti unicamente mediante stalli.
(3) Struttura semplice, ma di grandi dimensioni per via delle molte unità di elaborazione.

Figure 2.1: Comp

2.4 Application Specific Microprocessors

Embedded systems often require high specialization and a limited set of functions, making GPPs less suitable for certain applications. To address these specific needs, specialized architectures have been developed.

2.4.1 Digital Signal Processors

Digital Signal Processors (DSPs) are among the most widely used application-specific processors, designed specifically for numerical computing tasks. Despite variations among different manufacturers, DSPs share a common architecture optimized for handling operations like multiplication and accumulation, crucial for many algorithms.

A typical operation in DSPs is represented by the equation:

$$z_{t+1} = z_t + x \cdot y$$

This operation, known as Multiply-Accumulate (MAC), is essential for DSP performance, leading to architectures that optimize both addition and multiplication, often through a single three-operand instruction.

To achieve high efficiency, DSPs are designed to optimize the execution of loops, which are prevalent in many numerical algorithms. This is possible due to certain characteristics of loop structures, such as having small bodies—typically around ten assembly instructions—where the control loop variable remains unchanged. The update of this variable is straightforward, and the access patterns for vectors are generally regular. As a result, DSPs incorporate specific hardware enhancements to facilitate loop optimization, including circular buffers for storing loop bodies and dedicated registers for control loop variables, which allow for increments and decrements without burdening the ALU.

Another challenge for DSPs is memory access speed, which can become a bottleneck. To combat this, DSP architectures often include high-bandwidth interfaces and sophisticated memory hierarchies. This might consist of high-speed buses capable of transferring wide data words, unified cache systems for both data and instructions, and specialized caches, such as Harvard architecture, which maintains separate caches for instructions and data, or SHARC architecture, which can incorporate multiple Level 0 caches for different types of data.

In addition to their flexibility and computational power, many modern DSPs adopt a VLIW architecture. This design is particularly suited for numerical applications, where control is limited, and significant parallelism can be leveraged.

2.4.2 Network Processors

Network Processors (NPs) represent another class of specialized microprocessors tailored for processing network packets. These complex SoC architectures utilize high levels of parallelism to handle the demands of network applications, such as routers. Given the ever-evolving functionalities and protocols within the networking domain, designing an NP can be quite complex. Typical operations performed by NPs include buffering packets, modifying data-link headers, searching for specific fields in IP headers, and computing CRC codes. To maximize performance, NPs incorporate dedicated hardware for high-speed I/O interfaces, queue management, cryptographic cores, and multiple RISC cores.

As network applications frequently handle multiple independent data channels, dedicated hardware units, often called Packet Processors (PPs) or Channel Processors (CPs), are utilized to manage these flows. In cases where data streams are interdependent, RISC cores may be employed for supervision and high-level management. The nature of packet processing typically involves executing a series of straightforward routines on a large volume of data, often requiring programs to be written in assembly for specific PPs, although higher-level languages like C are becoming increasingly common.

2.4.3 Summary

Microcontroller Units (MCUs) constitute another category of application-specific microprocessors. These devices integrate peripherals and interfaces onto a single chip, making them ideal for applications that do not demand high computational power but do require careful management of hardware resources and development time. MCUs are often designed without interfaces to external memory, which results in smaller programs. The lack of memory interfaces can increase pin counts, prompting some architectures to employ multiplexing techniques to limit this increase.

MCUs typically provide a variety of peripheral interfaces, including I2C, SPI, CAN, JTAG, PWM, UART/USART, watchdog timers, and analog I/O capabilities. While programming for MCUs is primarily conducted in C, assembly language is sometimes utilized for specific tasks. The SDKs available for microcontrollers can vary significantly, ranging from basic C compilers to comprehensive frameworks that assist in performance analysis and configuration management.