

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

1	Introduction	1
1.1	Introduction	1
1.1.1	Technological problems	2
1.2	Applications future	2
1.2.1	Transistors and cores	3
1.2.2	Silicon challenges	3
1.2.3	Frequency scaling	4
1.2.4	Cooling systems	4
1.3	Productivity and planning	4
1.3.1	Platform-Based Design	4
1.3.2	Technology Trends	5
2	Microprocessors	6
2.1	Introduction	6
2.2	Processors taxonomy	6
2.2.1	Selection process	7
2.3	General Purpose Processors	8
2.3.1	CISC GPP	8
2.3.2	RISC GPP	9
2.3.3	Superscalar GPP	9
2.3.4	CISC and RISC GPP	10
2.3.5	VLIW GPP	10
2.3.6	CISC and VLIW GPP	10
2.3.7	Comparison	11

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 elements, 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.

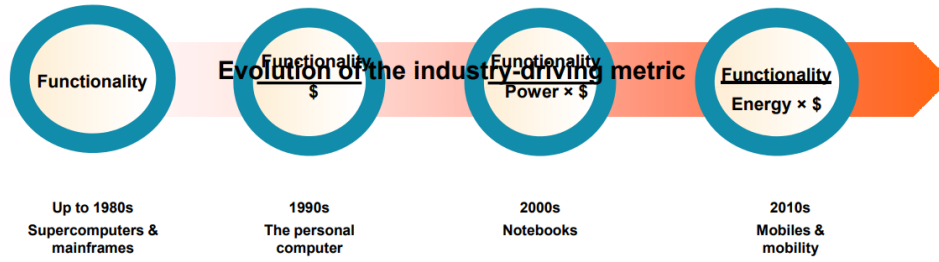


Figure 1.1: Industry requirements

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 are a part of the embedded systems applications implemented in software their algorithms the main features are:

- *Flexibility*: it is divided into maintainability and evolution of the application: software development is less complex, faster, verification is less critical and software designers are available in volume.
- *Time To Market* (TTM): time needed to have a product placeable to the customer.
- *Easy upgrade*.
- *Cost*: the cost depends on the volume. The design cost is one time only, and the production relies only on the cost of the silicon.

This approach is cost effective since usually we don't need all the power of a processor, but we use only a fraction of it. Hence, microprocessors keep only the needed functions and power, reducing the production cost and the consumption of material.

Typically an equivalent hardware solution has a better performance than software: microprocessors architecture is optimized for flexibility but it is generic, not specialized. In fact, performance is not only in computing speed, but also in energy, power, memory footprint, chip area, and so on. Usually, the software implementation is easier than the hardware one, but in case of new requirements it is more simple to modify the hardware than the software and the performance will remain better on hardware.

Designing a software solution requires a deep knowledge of the architecture of the microprocessors available on the market. The characteristics of the problems (algorithm) should drive the designer, beside non functional constraints (performance, cost, development time, power and so on). Before to select a specific processor, the initial step is the choice of the class of processor to be used and in which form it has to be purchased.

2.2 Processors taxonomy

The processors can be categorized into:

- *Application Specific Processors (ASP)*: tailored for specific classes of applications that works with a lot of operations per seconds.
- *General Purpose Processors (GPP)*: four season processor, not optimized for any specific applications. Versatile, it can be used for different type of applications.

The embedded systems multi-processors Typical solution is to use GPP for supervision and control of the activities of one or more ASPs

The processors can be:

- *Components Off The Shelf (COTS)*: just buy a chip off-the-shelf and mount in on a board (PCB) together with the interfaces to the rest of the systems
- *Intellectual Property (IP)*: the design (description) of the micro is purchased. Several abstraction levels:
 - *Soft-macro*: the micro is described in HDL at RT-level.
 - *Hard-macro*: description down to the level of layout.

Give the diffusion of Programmable Logic Devices (PLD), the used of IP is becoming more and more popular. Typically the suppliers of CPLD and/or FPGA make available one or more cores (even for free).

2.2.1 Selection process

The selection process is based on:

- *Class*: the main driver is the nature of the algorithm, the operations and data to be elaborated.
- *Form*: the main driver is the target architecture of the systems. But performance and power, in addition to royalties, can make the difference.
- *Performance*: ain metric is the average number of instructions per clock cycle (IPC/CPI). It is relative to the clock frequency, hence it need to be scaled to compare different architectures. MIPS is absolute measure of the throughput. It can be misleading when processors have different Instruction Sets (ISA). MFLOPS is the same but for Floating Points operations, for more specific architectures, like DSP or NP, metrics can change DSP, frequently it is used MMACS. NP, average number of processed packets in a time unit.
- *Power*: probably the most important driver for embedded systems. Average power or peak power, useful to roughly estimate the max power or the avg power of the overall systems. Frequently, especially at the beginning, it is used a joint measure between power and speed (related to performance). These information are very useful but must be properly used during the design process.

Other important elements to be considered are:

- *Memory*: bandwidth and size of the application can be a hard constraint. In some case it could be impossible to use external memory only that internal to the micro is available. Maybe that some bandwidth can be achieved only by integrated memories. For complex systems, the crucial point could be the addressing space.

- *Peripherals*: an embedded systems typ elaborates external signals and control physical apparatus. It is possible to select and architecture oriented toward the computing, and to design the rest of the system, or to look for a single chip solution integrating computing, interfacing and some peripherals The choice of the micro require a PCB or a SoC integrating all the peripherals under the responsibility of the designer. Using a microcontroller, the designer get rid of the integration problem, paid in terms of reduction in the number of alternatives.
- *Software*: Many embedded applications have a limited legacy code, most of them exploit standard functions and libraries The availability of libraries simplifies both design and validation of the applications, making feasible in some cases the development of sw solutions otherwise impractical. The availability of an operating systems and SDK for the specific processor is important, too. It is the cornerstone of the software development, many differences exist among the various SDKs. Important aspects to be evaluated are the software compilation flow, the quality of the code generated and the flexibility and knobs offered to the designer Concerning the analysis tools, the richness, their accuracy and reliability are important. Debuggers should be carefully analyzed Documentation and reference design Community end ecosystem.
- *Packaging*: if the micro is a COTS, different packages are available: size, pinout, material (e.g. plastic, metal), certification (consumer, industrial, aerospace, and so on).
- *Certifications*: there exists several certification: Consumer, industrial, aerospace, automotive, military, and so on. Some are specifically tailored for narrow fields (e.g. MISRA rules for the code for automotive). The availability of a component with a proper certification could be a must constraint for its adoption.

2.3 General Pupurpose Processors

GPP - generic architecture valid for a range of application fields Common in PC/PDA For embedded systems low end when energy and performance are not crucial and the nature of the application is quite heterogeneous. Control and management of a slow interfacing with sensors, interactivity via GUI. GPPs can have different architectures:

- CISC: huge amount of complex instructions.
- RISC: few simple instructions

2.3.1 CISC GPP

Ideally, each Arith/Logic operation should be capable to access (read) data and write the result by exploiting any of the available addressing modes. Fully orthogonal architecture and Instruction Set. In the reality, the number of operation-addressing mode combinations has been limited. Usually one of the operand refers to the memory, while the other is a register with the role of both source and destination. Despite this simplification, CISC instructions are coded in variable length formats. More complex fetch and decode units and need of more speed when accessing the memory.

Architecture The ALU, is implementing a variety of the basic operations: Add/Sub, mul, div, logic operations, shift, comparison In some architectures the instructions can be vectorial, working on more registers at the same time, so this is a Single Instruction Multiple Data architecture. So many addressing mode and A/L operations, require a complex data-path that is hard to make fast.

To increase the throughput, the CISC processors implement complex pipeline (e.g. up to 23 stages) that are hard to manage Another strategy, still very costly in terms of hw resources, is based on altering the order of execution of the assembly instruction without modifying the semantics of the program. For a compiler, when considering so complex pipelines, it is hard to control the status of the execution of the instruction. The instructions could arrive to the decoding in an order that is not optimal w.r.t. the status of the pipelines at that time Modern CISCs have dedicated hw units to dynamically reschedule the instructions Out-of-order execution

CISC have remarkable performance but, over the time, energy efficiency and cost moved in favor of other architectures.

2.3.2 RISC GPP

Restricted Instruction Set Computer, when the architecture has few simple instructions. End of '80 the price of DRAMs fall down Size of the program is no longer a wall CISC operations can be decomposed into RISC instructions to simplify the architecture In the same period the integration scale raised up, dramatically Possibility to realize on a single chip complex architectures Pipeline and quantity of GP registers To exploit pipelining, the execution time of the instructions must be uniform. This is possible when instructions are simple

Main advantage is simple instructions set and simple architecture Instruction decoding is simplified Frequently RISC instruction have fixed lengths Easy balancing the stages of the pipeline and to increase clock Execution units (ALU and BPU) take advantage from simple inst RISC operates only on registers hence more speed and duration is predictable RISC processor tends to have many registers, so the presence of memory spill is reduced significantly

I/O, namely load/store from memory/registers and viceversa Each instruction requires to pre-load of the data and, eventually, to write the result in memory: load/store architecture Data transfer operation are crucial for performance Fortunately, the instructions have a limited number and simple of addressing modes Aggressive use of memory (cache) hierarchy to reduce the access time to data

Software development flow Considering the same high level source code, the number of RISC instructions will be higher that CISC RISC architectures has no hw units to solve pipeline conflicts But good predictability of execution allow the compiler to generate efficient code Power consumption: RISC are simple and inherently low power w.r.t. CISC

2.3.3 Superscalar GPP

There exists a lot of parallelism at the level of assembly instruction. It is superscalar an architecture having more than one executing units, hence more ALUs and or BPU.

The parallelism offered by the execute stage allows to deliver more instruction per clock cycle. The complexity of the control logic increases as the impact on clock speed and power consumption. Moreover, it is the microprocessor to define dynamically the scheduling of the instructions. Pros: compiler has not to work on organizing the code for a specific hardware

structure Cons: a relevant part of the processor is reserved to implement complex scheduling mechanisms and to manage the consistency

2.3.4 CISC and RISC GPP

Approach combining out-of-order execution and superscalarity with the goal to remain compatible with code developed in the past. Instruction set x86 is the standard de-facto for CISC. Each CISC instruction is decomposed in a set of simple instructions easy to be executed by a simple and efficient core. There is a hardware cost, but this solution joins the benefit of backward compatibility with the high-performance achievable by a RISC core.

2.3.5 VLIW GPP

To overcome the limits related to the complexity of the hardware scheduling of the instructions, Intel developed in the past a new class of processors named EPIC (or VLIW). Itanium has been the first commercialized. These architectures support explicit parallelism: possibility to execute more instructions at the same time under the control of the program. A VLIW instruction is the packing of more elementary instructions, typically RISC, in a single wide word, having a fixed structure.

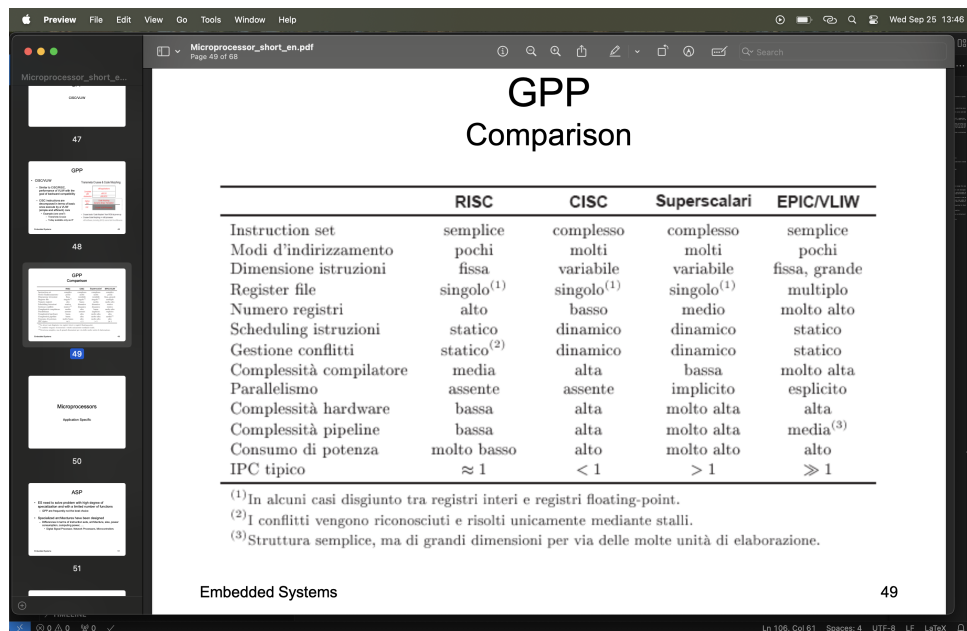
The VLIW word, once fetched from the memory, is decoded in parallel. At the end of the decoding, the single RISC instructions corresponding to the various slots, are dispatched to the different execution units. This fixed structure moves to the compiler the task to perform a complex scheduling. The algorithms to be optimized should be well structured to fit well a specific architecture.

EPIC/VLIW has the goal to move the complexity of the scheduling and code optimization towards the compiler. The architectures are simple, even if of large size given the number of execution units. Power consumption is in between simple and superscalar RISC. Similar solutions, allowing to achieve peak performance, have been adopted also in non general purpose architecture like the digital signal processors (DSPs).

2.3.6 CISC and VLIW GPP

Similar to CISC/RISC, performance of VLIW with the goal of backward compatibility. CISC instructions are decomposed in terms of basic ones executed by a VLIW (simple and efficient) core.

2.3.7 Comparison



**GPP
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.

Embedded Systems 49

Figure 2.1: Comp