

# Embedded systems & applications and system-on-chip design

Labros Bisdounis, Ph.D.



Project manager  
Research & Development Division  
INTRACOM TELECOM S.A.



Adjunct professor  
Department of Computer and  
Communication Engineering  
University of Thessaly

September 2007

# Motivation

---

- The heterogeneity of today's embedded systems faces developers and engineers with new problems when it comes to specifying, simulating, designing and optimising such complex systems.
- Implementations are typically comprised of software components, programmable or dedicated hardware components, communication and memory subsystems.
- The design of embedded systems is driven by cost vs. performance trade-offs. The optimization involves the simultaneous consideration of several incomparable and often competing objectives, such as cost, speed, power consumption, reliability etc.
- As a consequence, much effort and automated design tools are necessary in order to handle the complexity of today's embedded systems, and find the trade-off which is the most suitable for the market requirements.

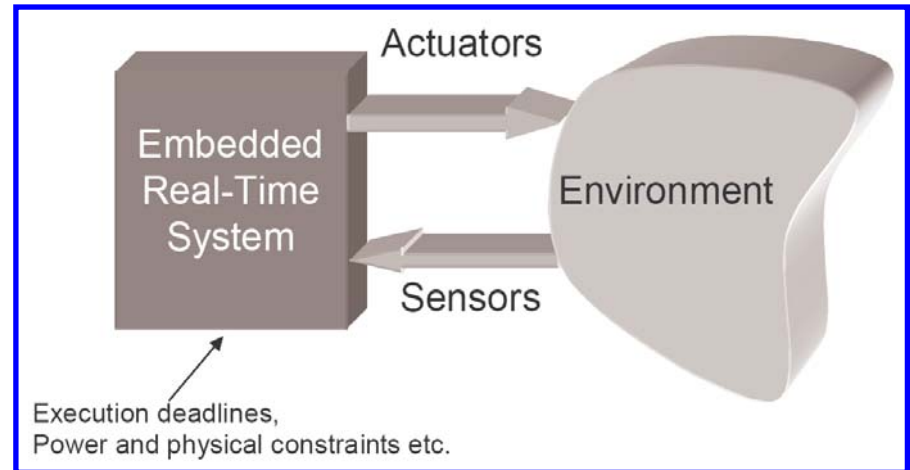
# What is an embedded system ?

---

- An embedded system is any device which **includes a programmable component** but itself is **not** intended to be a **general-purpose computer**.
- An embedded system:
  - ✓ is a collection of **programmable** components surrounded by **application-specific** hardware components and other **peripherals**.
  - ✓ **and interacts continuously** with its environment through sensors (with the general meaning).

# Main characteristics of an embedded system

- Dedicated and **application-specific** (not general purpose).
- Contains at least one **programmable component**.
- **Interacts continuously** with the environment.
- It is **real-time**: must meet external timing constraints (deadlines).
- Must **meet other constraints**: power consumption, physical constraints, cost, reliability, safety.



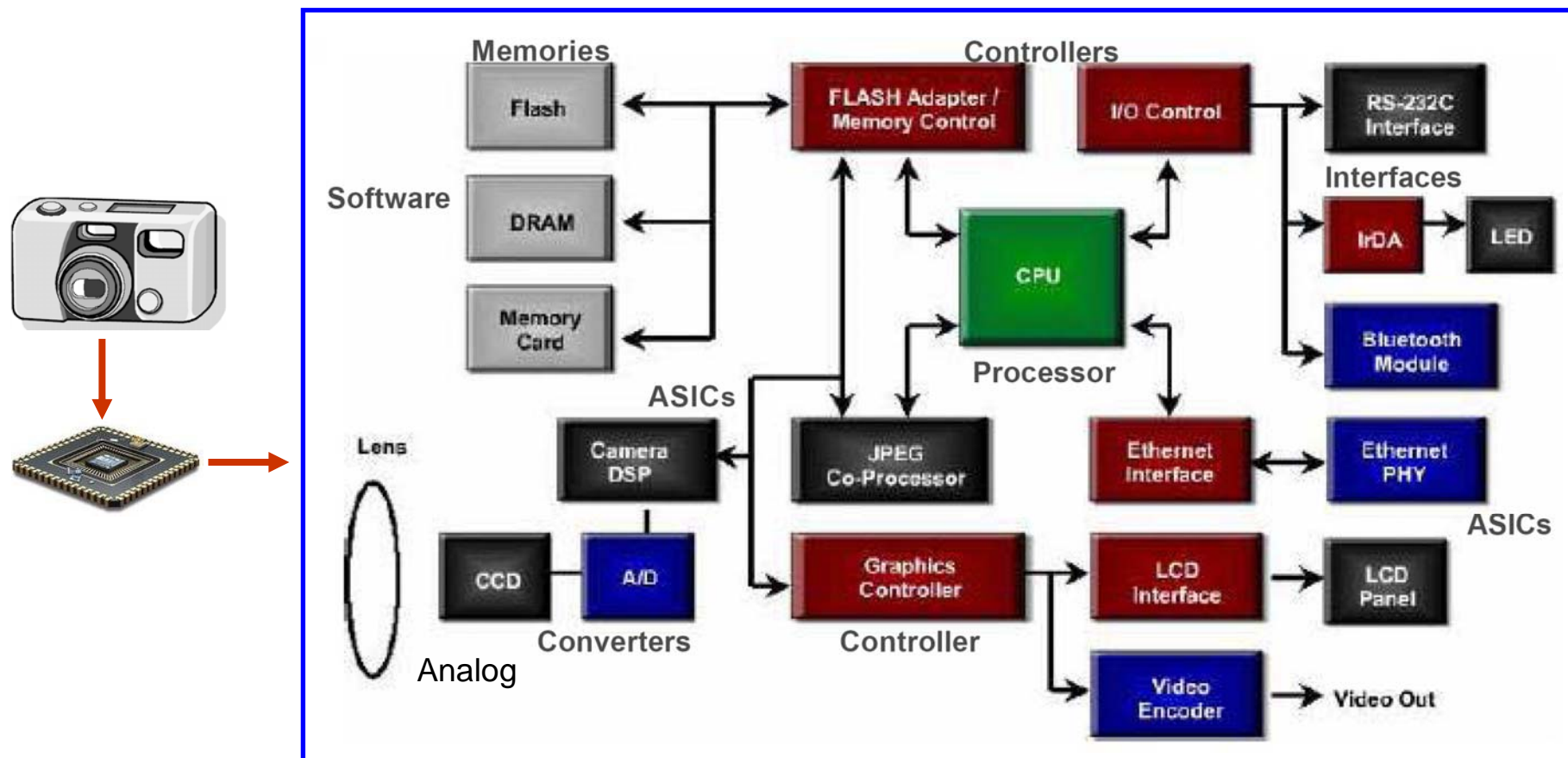
*Present and future of computing !*

# Components of an embedded system

---

- **Digital components**: processors, memories, controllers, buses, application specific circuits, peripherals (interface circuits).
- Embedded **software**.
- **Analog components**: sensors, actuators.
- **Converters**: A/D and D/A.

# An example: Digital camera



# Embedded software and digital components

---

- **Embedded software:** software running on embedded processors:
  - ✓ Application programs.
  - ✓ Real-time operating system.
  - ✓ Peripheral's drivers.
- **Digital (hardware) components:**
  - ✓ Programmable processors.
  - ✓ Dedicated hardware implementing critical and demanding tasks or tasks that are not suitable for software implementation.
  - ✓ Reconfigurable hardware (e.g. FPGAs).

# Computing elements in embedded systems

---

- **Processing:**
  - ✓ Used to transform data.
  - ✓ Implemented using programmable processors and/or custom hardware.
- **Storage:**
  - ✓ Used to maintain data.
  - ✓ Implemented using memory modules.
- **Communication:**
  - ✓ Used to transfer data between processors, custom hardware blocks, peripherals and memories within a system.
  - ✓ Implemented using buses in most cases.
- **Peripherals (interfaces) and controllers.**



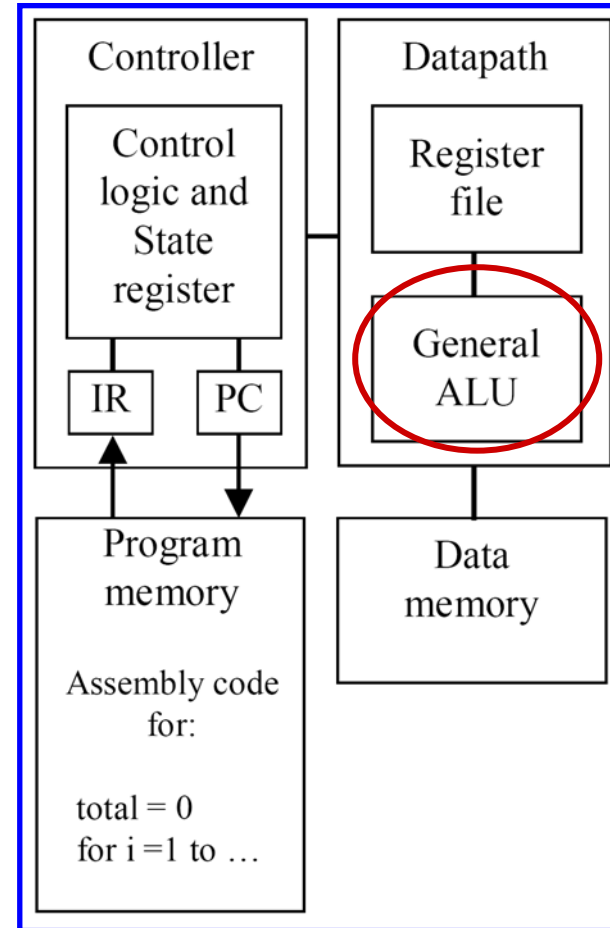
# Processors in embedded systems

---

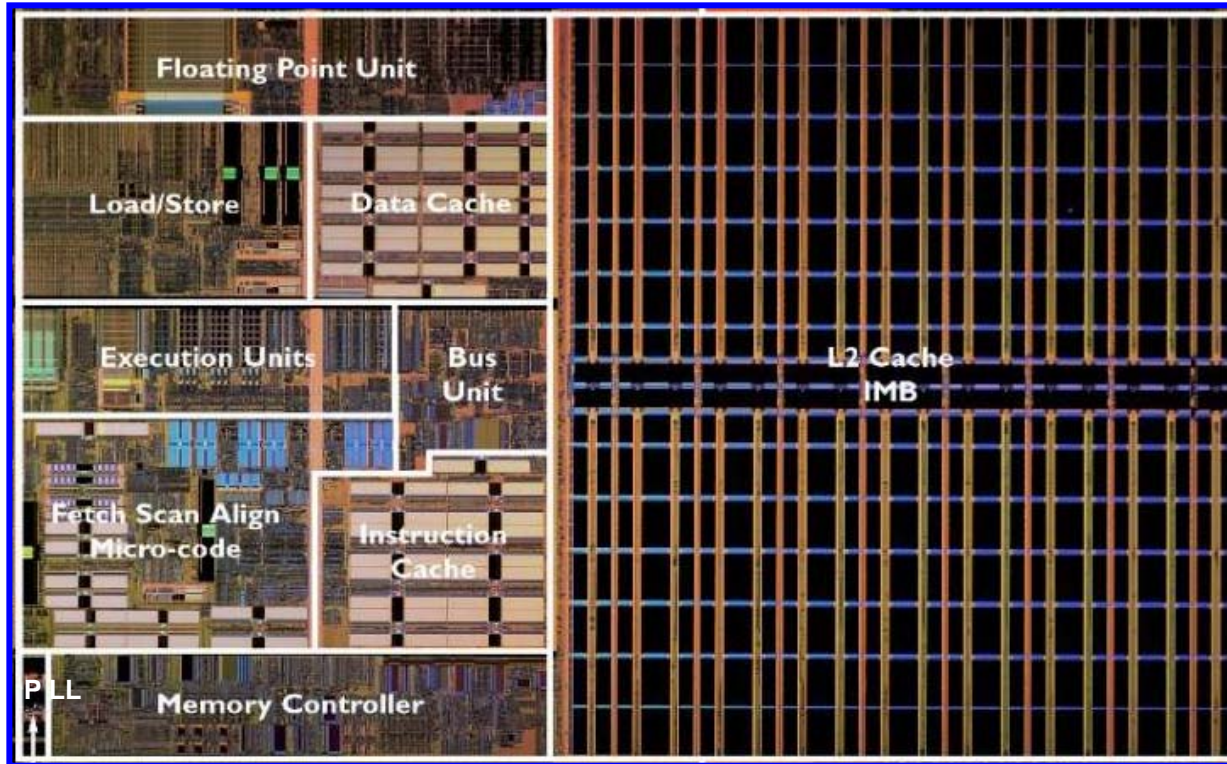
- There is need for energy, code size and run time efficiency.
- In general, processor in embedded systems is the device that runs a number of algorithms and contains control and datapath units.
- **General purpose (GP) processors:**
  - ✓ Perform a variety of computational tasks.
  - ✓ Flexible and low cost.
  - ✓ Slow and power hungry.
- **Application-specific processors (ASIPs):**
  - ✓ Tuned for an application domain, but programmable.
  - ✓ Fast and power efficient (compared to GP).
- **Application-specific circuits (ASICs):**
  - ✓ Customized hardware components for specific task.
  - ✓ Fast, power efficient, minimal area.
  - ✓ Inflexible and high cost.

# General-purpose processors (GP)

- Programmable devices used in a variety of applications.
- Contain program memory, general datapath with large register file and general ALU.
- Low time-to-market and NRE (non-recurring cost).
- High flexibility.

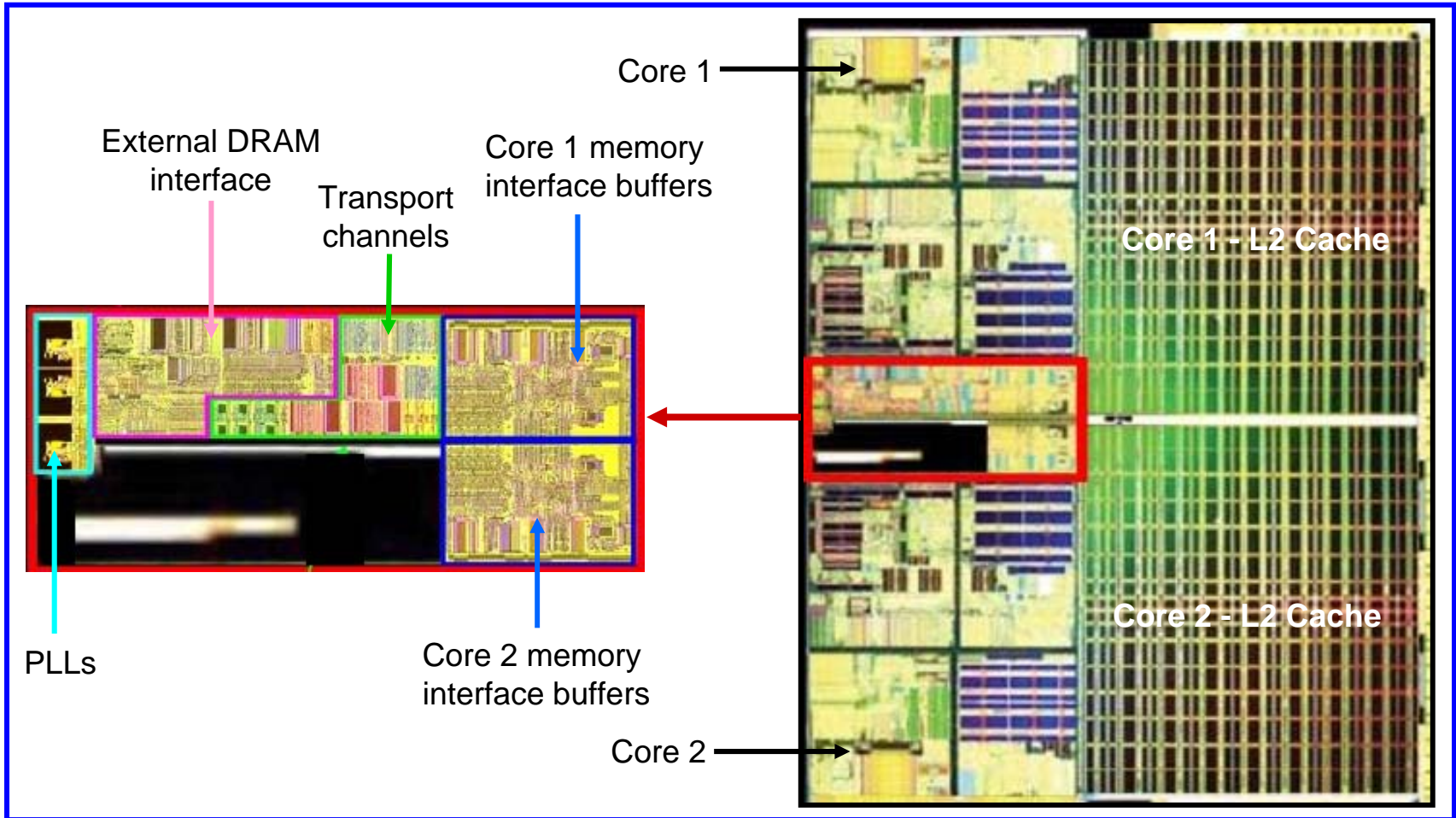


# GP example: AMD Athlon 64



- 64-bit CISC processor (technology: 130 nm, area: 193 sq.mm, transistors: 106 million)
- 16 64-bit and 16 128-bit registers
- 64KB 2-way associative L1 data and instruction cache memories
- 1MB 16-way associative L2 cache memory
- Integrated memory controller

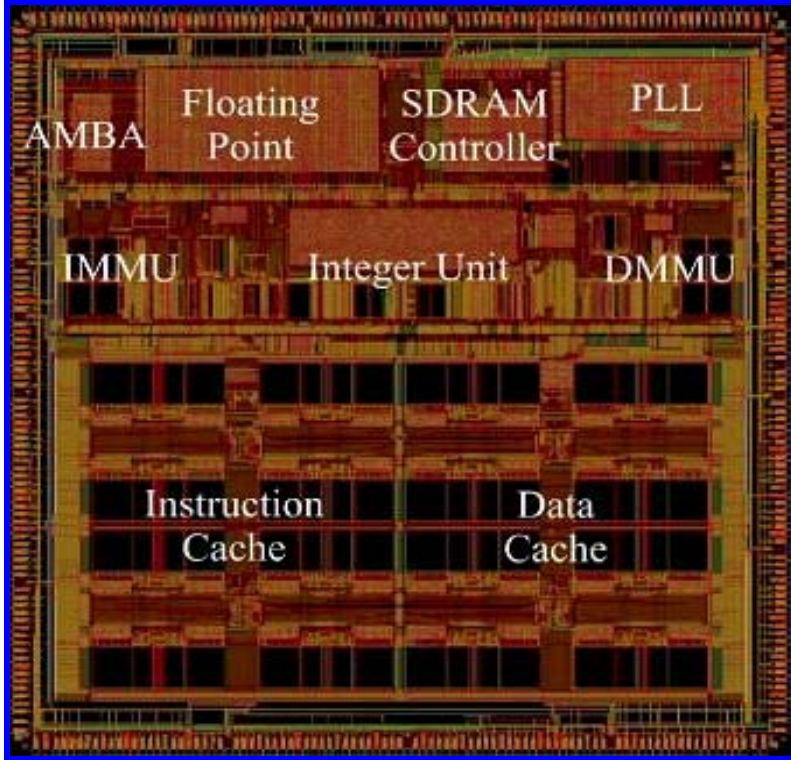
# GP example: Dual-core AMD Athlon 64



- Dual-core processor (technology: 90 nm, area: 199 sq.mm, transistors: 233 million)
- Discrete L1 and L2 cache structures for each core
- Memory interface module



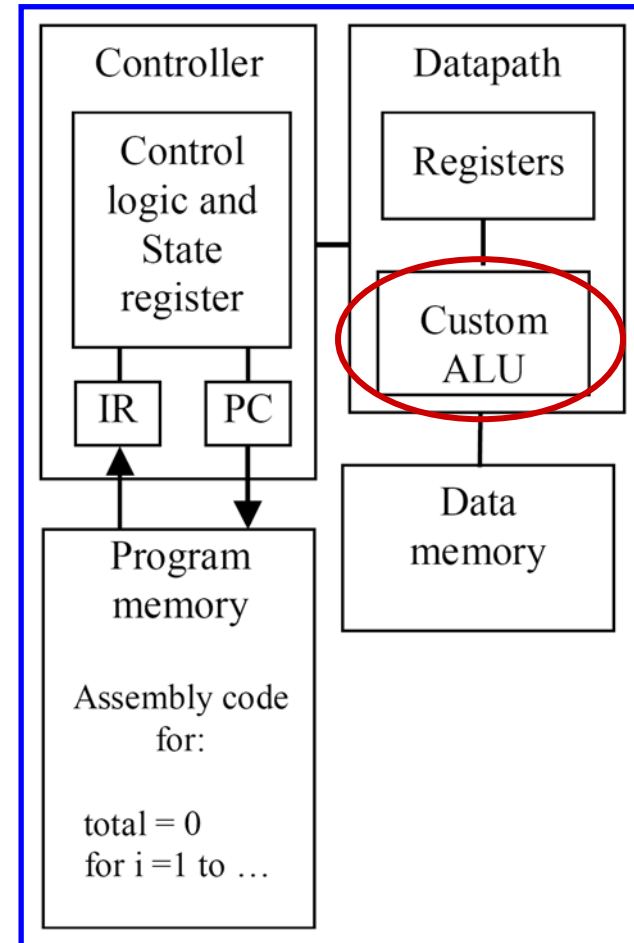
# GP example: ARM1020E processor



- 32-bit RISC processor with six-stage pipeline (technology: 130 nm technology, area: 10.3 sq.mm area).
- 64-bit instruction and data buses (two instructions are fetched per cycle).
- 16-bit compressed (Thumb) instruction set for better code density.
- Floating point co-processor.
- External memory controller and memory management units.
- 32KB 8-way associative caches instruction and data cache memories.

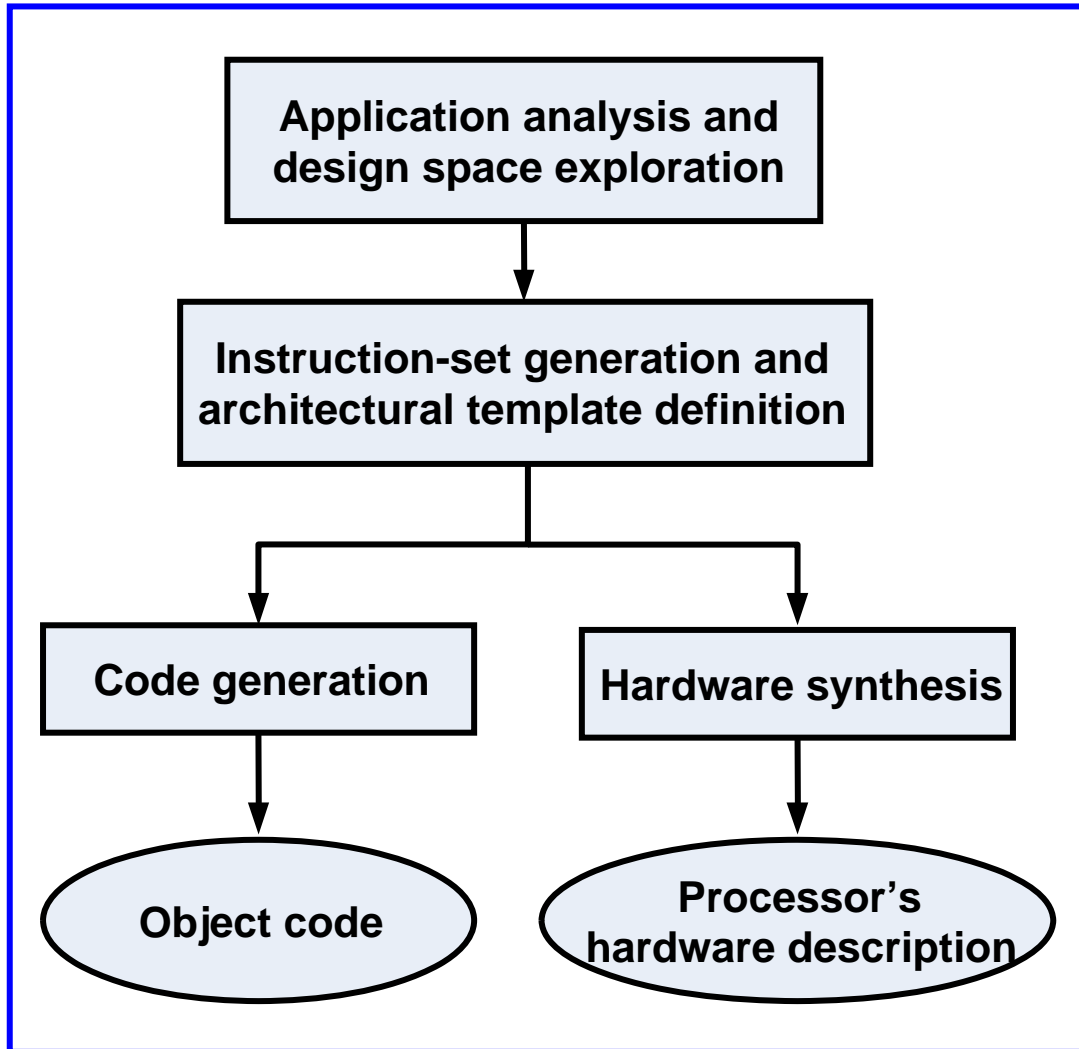
# Application-specific processors (ASIP)

- Programmable devices optimized for a particular application or family of applications having common characteristics.
- Contain program memory, optimized datapath and specific functional units.
- Use specific instruction set.
- High performance, small size and low power consumption.
- Usually they exhibit small flexibility.
- DSP, network processors, configurable processors etc.



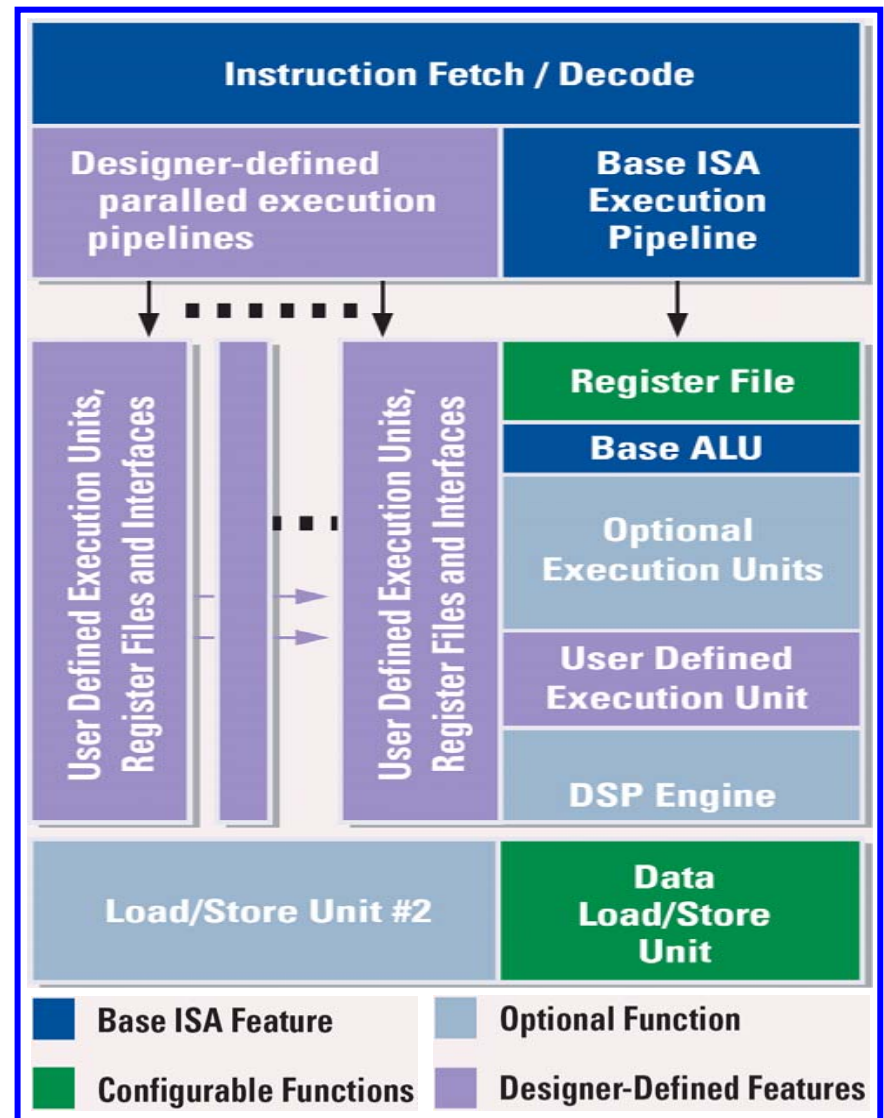
# ASIP design flow

---



# ASIP example: Tensilica Xtensa LX2 processor

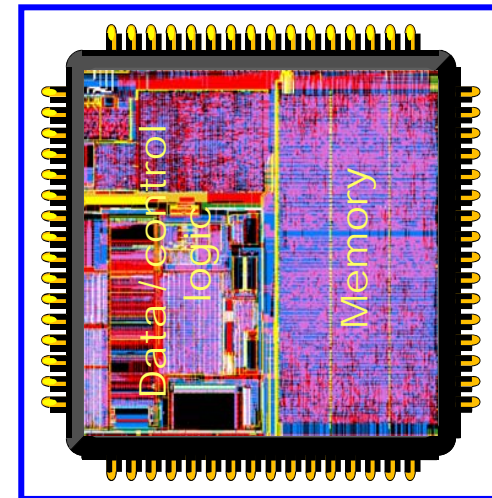
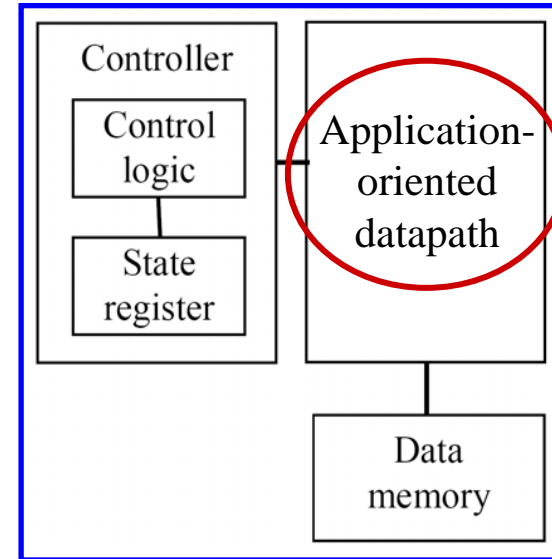
- **Configurable**, **extensible** and **synthesizable** processor for rapid implementation of complex SoC designs.
- Base architecture: 32-bit ALU, 68 registers, 80 instructions, compressed 16- or 24-bit instruction encoding.
- Selection and configuration of predefined processor functions (**configurability**).
- Optional predefined execution units: 32-bit multiplier, 16-bit MAC, DSP engine, floating point unit.
- Explorer to analyze the application and find options that will enhance performance.
- Specific language (Verilog-like) to describe new execution units, and processor generator to add them to the processor (**extensibility**).
- Designer-selectable 5- or 7-stage pipeline: option of adding two cycles for access memories with long access times.





# Application-specific circuits

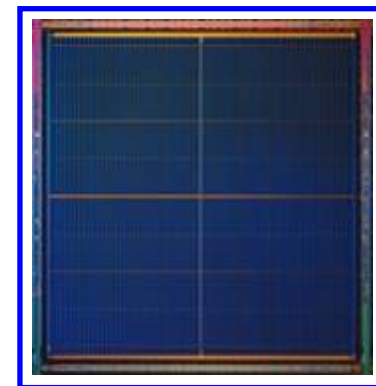
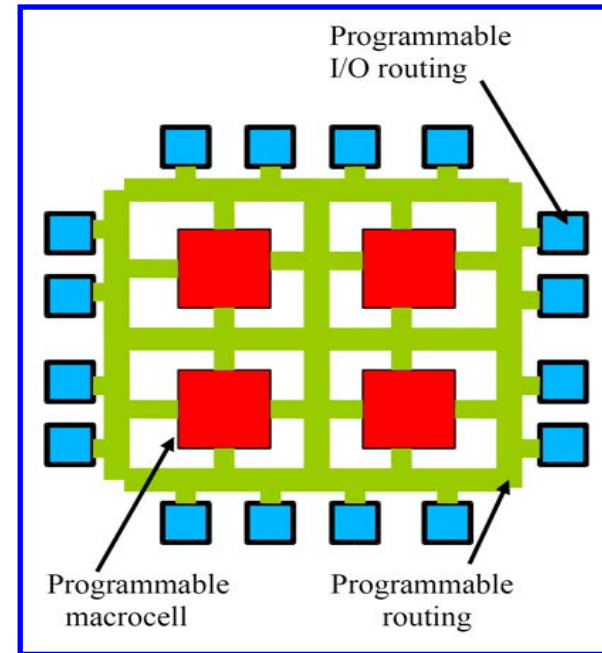
- Digital circuits designed to implement exactly one algorithm (application or part of application).
- Custom-designed circuits that are necessary if ultimate speed or energy efficiency is the goal (known as coprocessors or hardware accelerators).
- Contain only the components needed for the execution of a specific algorithm (no program memory is needed).
- Fast, low power consumption, small size, high cost for low volume.



**Audio  
processing  
ASIC**

# Programmable devices

- Prefabricated digital devices that can be purchased and programmed by the designer/user.
- Programmable arrays of generic logic modules, programmed by the designer/user and not by the semiconductor foundry.
- PLDs, FPGAs.
- Alternative to ASICs with low NRE cost, and fast availability.
- Penalty on area, performance and power consumption.



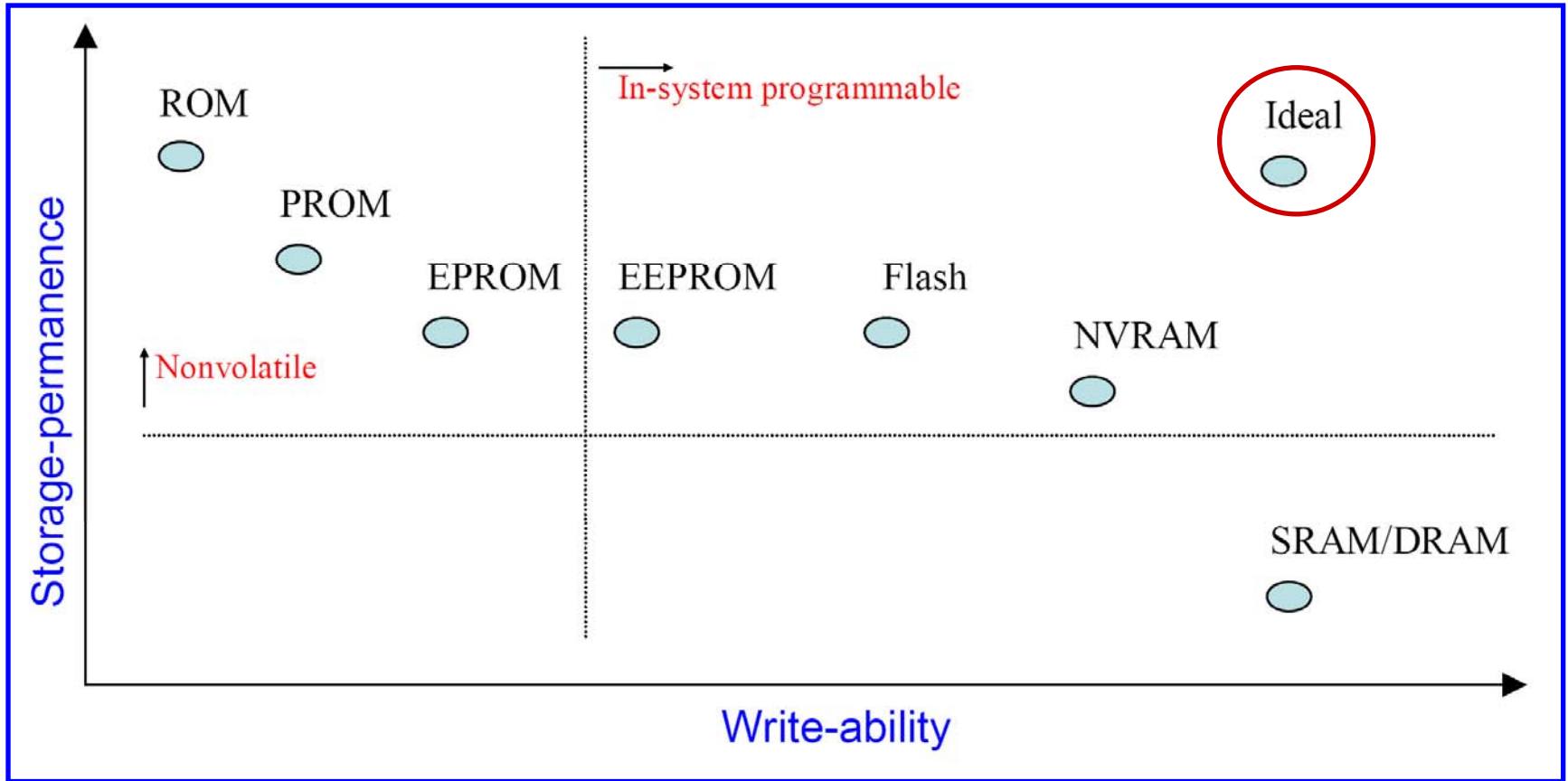
**Xilinx  
FPGA**

# Memory in embedded systems

---

- Memory is used in embedded systems for storage purposes and offers access capabilities (read and/or write).
- Main characteristics:
  - ✓ **Storage permanence**: ability of memory to hold stored bits after they are written.
  - ✓ **Write ability**: manner and speed a memory can be written.
- There are many different types of memories:
  - ✓ Random access: SRAM, DRAM
  - ✓ Read only: ROM, PROM
  - ✓ Erasable-programmable: EPROM, EEPROM, Flash
  - ✓ Combination of SRAM and EEPROM properties: NVRAM

# Comparison of memory types



# Communication in embedded systems

---

- Communication in an embedded system accounts for the transfer of data between processors, custom hardware blocks, peripherals and memories.
- Implemented using **buses**.
- Example: Common forms of communication are when a processor read or writes a memory or when a processor reads or writes a peripheral's register.
- Connectivity schemes:
  - ✓ **Serial** communication (USB, RS232 etc.): use single wire, high throughput for long distance communication, low cost).
  - ✓ **Parallel** communication (PCI, AMBA etc.): use multiple wires, high throughput for short distance communication, high cost).
  - ✓ **Wireless** communication (Infrared, RF).
- Each connectivity scheme has an associated **protocol** describing the rules for transferring data over it.

# Main issues in embedded systems communication

---

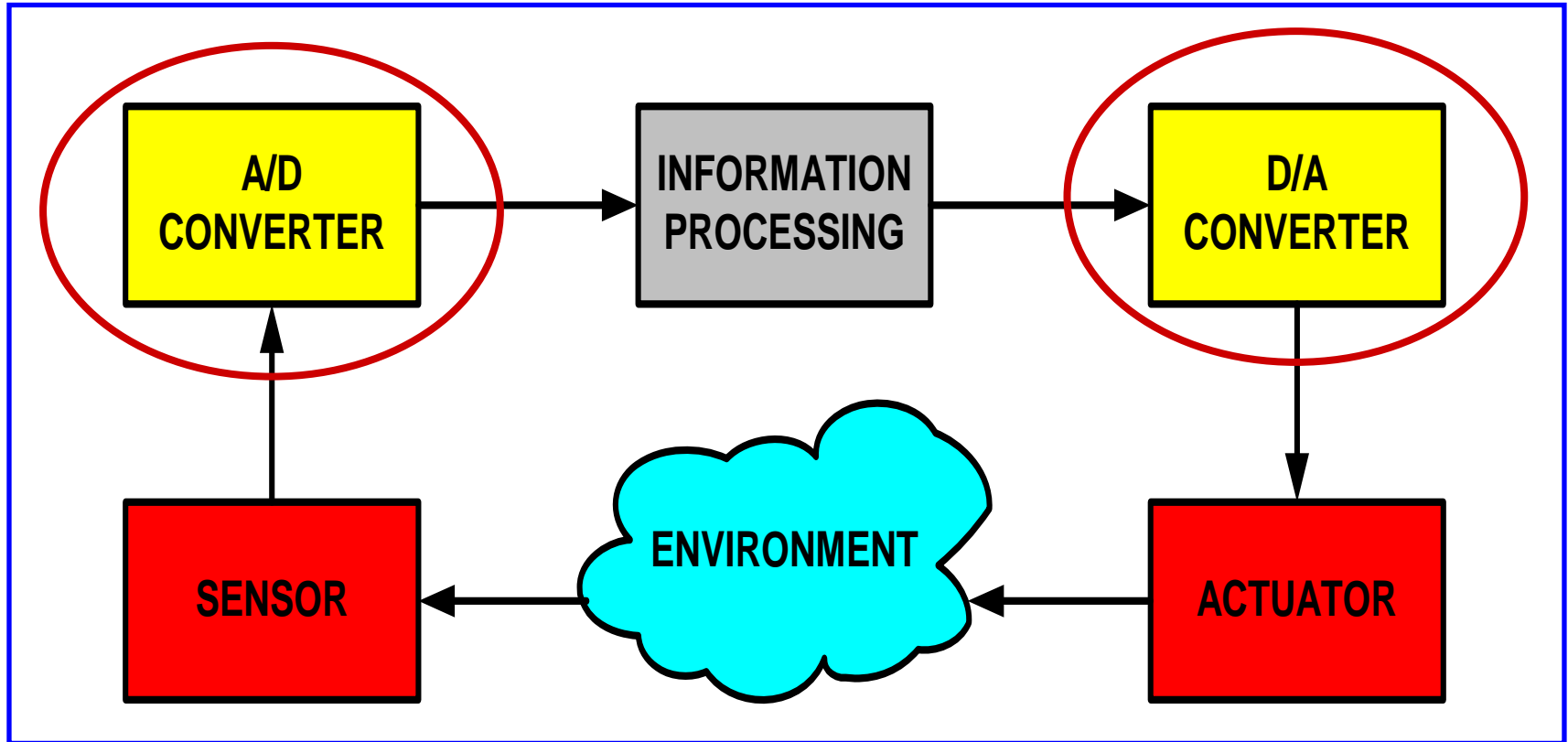
- The main issues regarding the communication of a processor with the peripherals through a system bus are:
  - ✓ The **addressing procedure**: how the system address map is used in order the processor to communicate with the memory and the peripherals.
  - ✓ The **interrupt-driven communication**: the processor accepts interrupt signals in order to read and process data from a peripheral.
  - ✓ The **direct memory access (DMA)** for transferring data between memories and peripherals, without going through the processor.
  - ✓ **Arbitration**: how to handle simultaneous servicing requests of peripherals.

# Peripherals and controllers in embedded systems

---

- Peripherals and controllers perform specific computation tasks.
- Custom single-purpose processing blocks:
  - ✓ Designed by us for a unique task.
  - ✓ Predesigned (by others) for a common task.
- Examples:
  - ✓ Timers, counters: to measure timed events or indicate that a maximum count reached.
  - ✓ UART - universal asynchronous receiver transmitter: takes parallel data and transmits serially, receives serial data and converts to parallel.
  - ✓ LCD interface: interface the system to a liquid crystal display.
  - ✓ External memory controller, PCI controller, USB interface, Ethernet or other network type interface.

# Converters in embedded systems



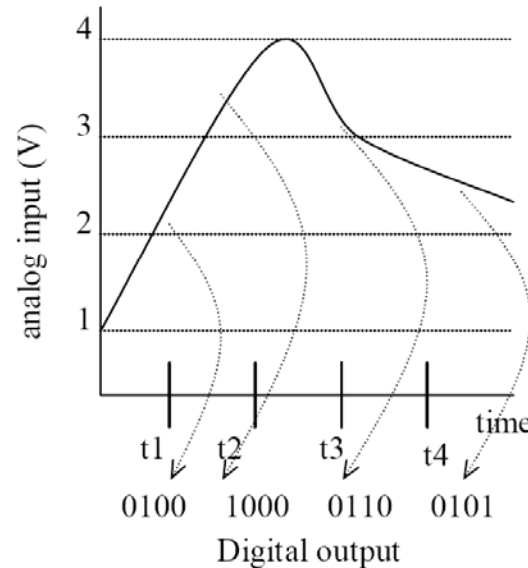


# Converters in embedded systems (cont'd)

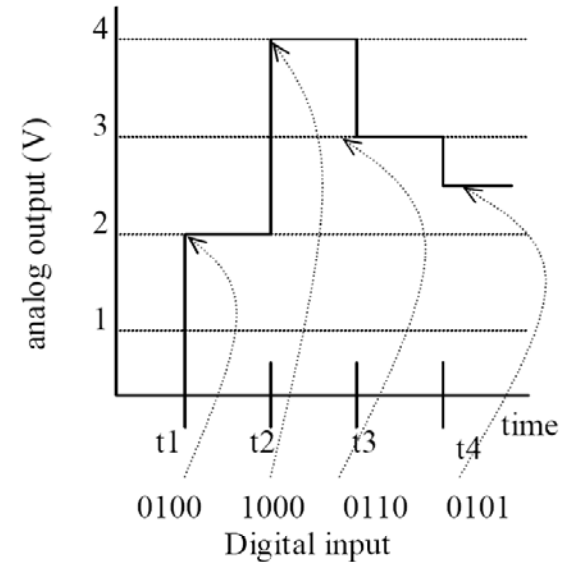
$V_{\max} = 7.5V$

7.5V	1111
7.0V	1110
6.5V	1101
6.0V	1100
5.5V	1011
5.0V	1010
4.5V	1001
4.0V	1000
3.5V	0111
3.0V	0110
2.5V	0101
2.0V	0100
1.5V	0011
1.0V	0010
0.5V	0001
0V	0000

proportionality



analog to digital



digital to analog

Main issues:

- **Sampling**: how often is the signal converted.
- **Quantization**: how many bits used to represent a sample.

# Analog components in embedded systems

- **Sensors**

- ✓ Capture physical stimulus (heat, light, sound, pressure, magnetism, mechanical motion).
- ✓ Typically, they generate a proportional electrical current.

- **Actuators**

- ✓ Convert a command to a physical stimulus (heat, light, sound, pressure, magnetism, mechanical motion).



Microphone



Megaphone



Laser diode,  
transistor



Antenna



DC motor



Accelerometer

# Sensors and actuators in embedded systems

---

- **Sensors** can be designed for virtually every physical stimulus. First, they capture the physical data and then they process them.
- Many physical effects are used for constructing sensors: generation of voltages in an electric field (law of induction), light-electric effect etc.
- Examples: heart monitoring sensors, car sensors (rain sensors for wiper control, proximity sensors), pressure sensors (touch pads and screens), audio sensors, motion sensors, thermal sensors (SARS detection through high fever) etc.
- **Actuators** produce output physical stimulus for various environments: motor control actuators (industrial applications), optical actuators (IR), thermal actuators, MEMS devices (Micro-Electro-Mechanical Systems) etc.
- **MEMS** technology regards the integration of mechanical elements and electronics on a common silicon substrate. Applications: biotechnology (DNA identification), communications (RF-MEMS), accelerometers (air-bags).

# Embedded applications

- **Automotive electronics** (airbag control, dashboard info, ABS, consumption control etc.).
- **Aircraft electronics** (guidance, flight control, air quality control, pressure control etc.).
- **Telecommunication systems** (mobile phones and network cards, mobile base stations etc.).
- **Medical systems** (diagnostic and monitoring systems, radiation systems).
- **Defence systems** (radars and safety communication systems, navigation systems like GPS etc.).
- **Consumer multimedia electronics** (cameras, game machines etc.).
- **Industrial process control systems**.
- **Robotics** (electro-mechanical systems).



# Embedded applications in every-day life

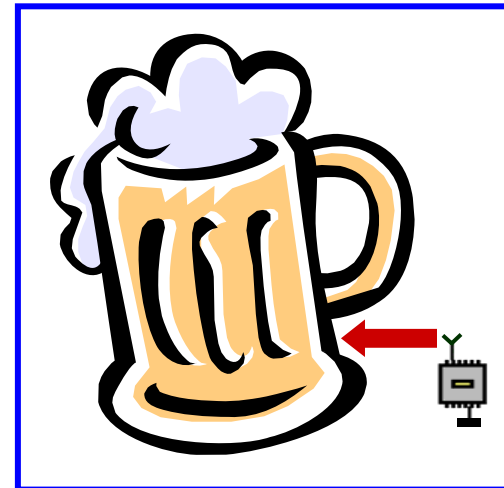
- **Mobile phone:**

- ✓ Multiprocessor system (8-32 bit processor for user interface, DSP, 32-bit processor for IR and Bluetooth ports)
- ✓ 8-100 MB memory, custom chips, integrated camera, megaphone, speaker etc.



- **Smart beer glass:**

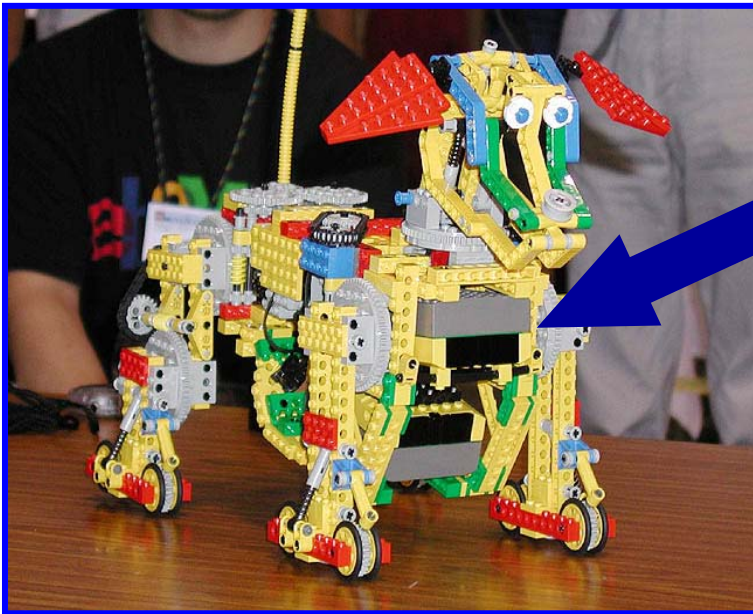
- ✓ Combines a fluid-level sensor with a simple 8-bit processor and an RF system with internal antenna. The system checks the fluid level & alerts the servers when close to empty.
- ✓ Integrates several technologies such as: radio transmission, sensor engineering, computer monitoring.



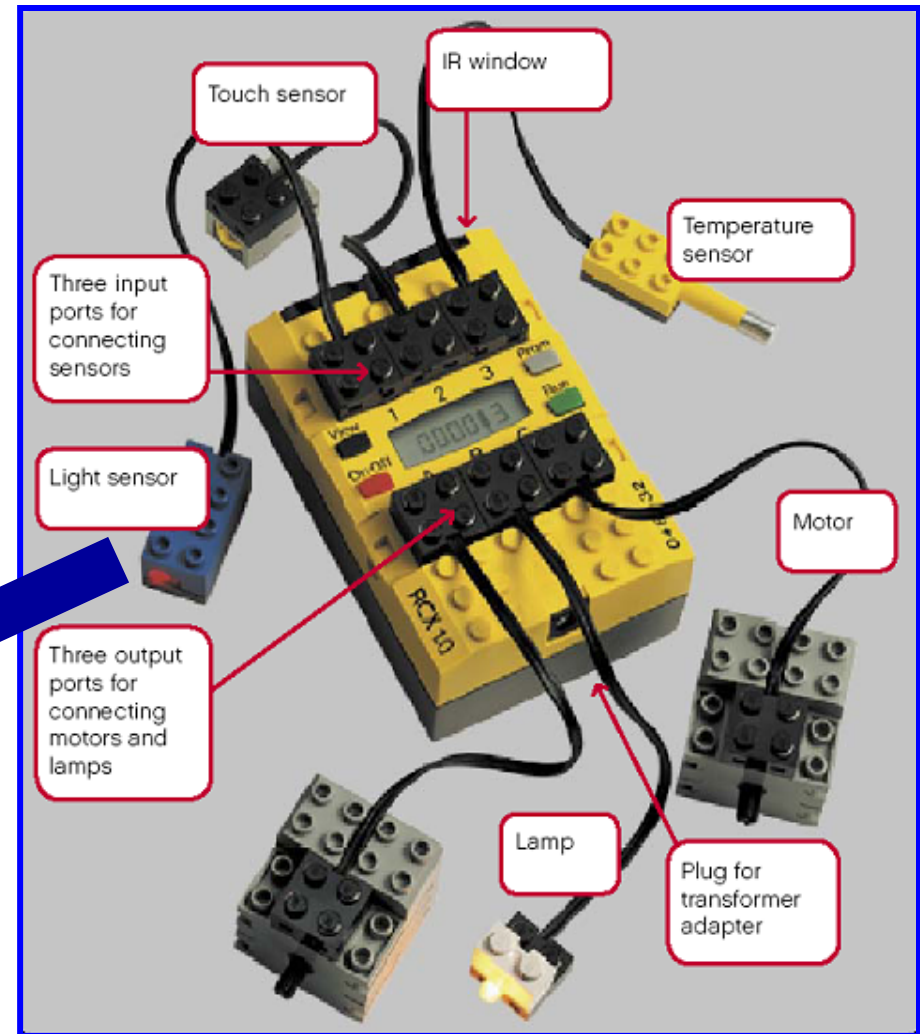


# Embedded applications for kids

- **Lego mindstorms robotics kit:** combines an 8-bit controller with 64 kB memory.
- Electronic circuits to interface the processor with the various sensors and motors.
- Good way to start learning embedded systems...

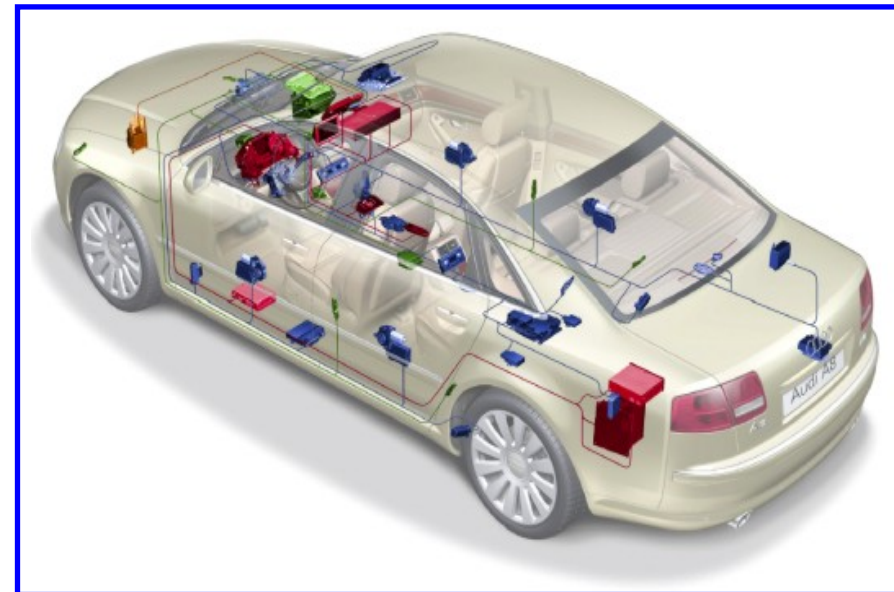


## Control unit



# Embedded applications for lotto winners

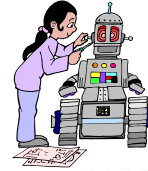
- 53 8-bit, 11 32-bit and 7 16-bit microprocessors (**71 in total!**).
- Multiple networks.
- Sensors and actuators distributed all over the vehicle.
- Windows CE operating system.
- Engine management: consumption, ignition, emission control etc.
- Instrumentation: data acquisition, display and processing.
- Safety and stability: airbags, ABS (anti-lock braking system), ESP (electronic stability control), efficient and automatic gearboxes etc.
- Entertainment and comfort: Radio-CD, A/C, television, GPS etc.



# Constraints in embedded applications



- Reactivity requirement.



- Timing constraints.



- Power dissipation constraints.



- Size and weight constraints.



- Cost constraints.



- Safety and security constraints.



- Reliability constraints.



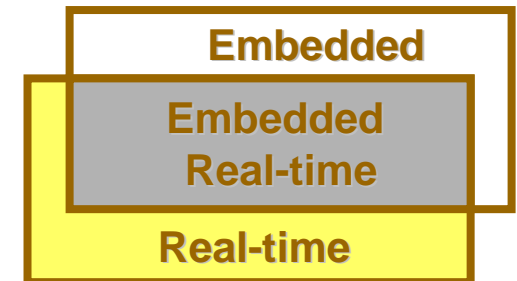
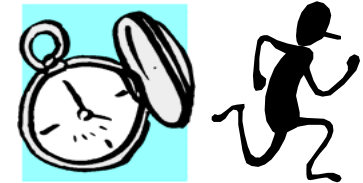
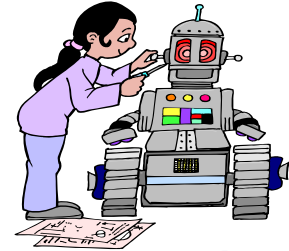
- Time-to-market constraints.





# Reactivity and timing constraints

- Reactivity requirement: embedded systems are in continual interaction with its physical environment through sensors and actuators, and execute at a rate determined by the environment.
- Timing constraints:
  - ✓ Most of the embedded systems have to perform in real-time which means that if data is not ready by a certain deadline (i.e. reaction of the system within a certain time interval dictated by the environment), the system fails.
  - ✓ A real-time constraint (deadline) is called hard, if not meeting that constraint could result in a failed operation of the system. All other time-constraints are called soft (if not meeting, the operation of the system will be degraded, but the system will not fail).
  - ✓ Embedded & real-time terms are almost synonymous.
  - ✓ Most embedded systems are real-time and most real-time systems are embedded.



# Power and size/weight constraints

- **Power consumption constraints:**

- ✓ High power dissipation needs strong power supply and expensive cooling system.
- ✓ High power consumption leads to short battery life time (very critical issue in mobile/portable applications).



- **Size and weight constraints:**

- ✓ Critical for mobile, portable devices (e.g. PDAs, mobile phones, cameras).
- ✓ Very critical for specific medical applications (e.g. pills with integrated camera and data acquisition system).



11 x 26 mm

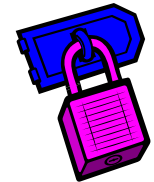
# Cost constraints

- **Cost constraints:** embedded systems are very often mass products in highly competitive markets and have to be shipped at a low cost (e.g. mobile phones market).
- We are mainly interesting in **manufacturing cost and design cost.**
- Main **cost factors:** design time and effort, type of used components (processors, memory, I/Os), technology (board-based, system-on-chip, type of manufacturing processes), testing time, power consumption.
- **Non-recurring engineering (NRE) costs** (design cost and prototypes development) are becoming very high, and because of that:
  - ✓ It is difficult to come out with low quantity products.
  - ✓ Implementation platforms are introduced, which are used for products of similar type.



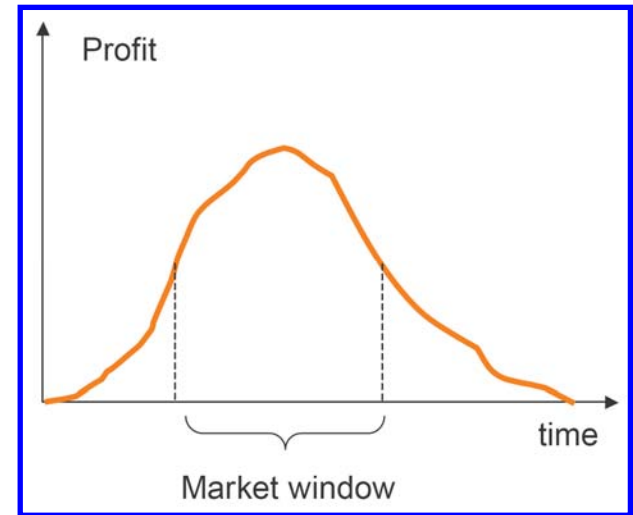
# Reliability and safety/security constraints

- **Reliability constraints:**
  - ✓ Reliability is the probability of an embedded system working correctly provided that it was working at  $t = 0$ .
  - ✓ Even perfectly designed systems can fail if the operation assumptions (workload, possible errors) turn out to be wrong. So, we have to be very carefully when define the operation assumptions for a specific application in a given environment.
- **Safety constraints:** embedded systems are often used in life critical applications (automotive electronics, nuclear plants, medical applications, defence applications etc.).
- **Security constraints:** embedded systems for communication applications must often support confidentiality and authenticity.
- In order to guarantee the above constraints during the design, exhaustive **verification** of the certain properties of the designed system must be performed, as well as synthesis and design based on **automatic design tools**.



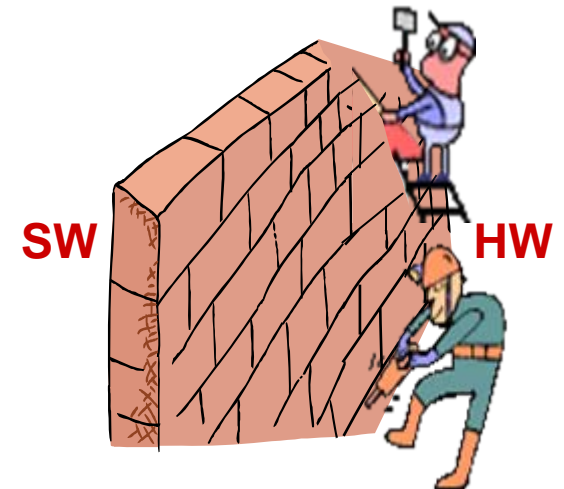
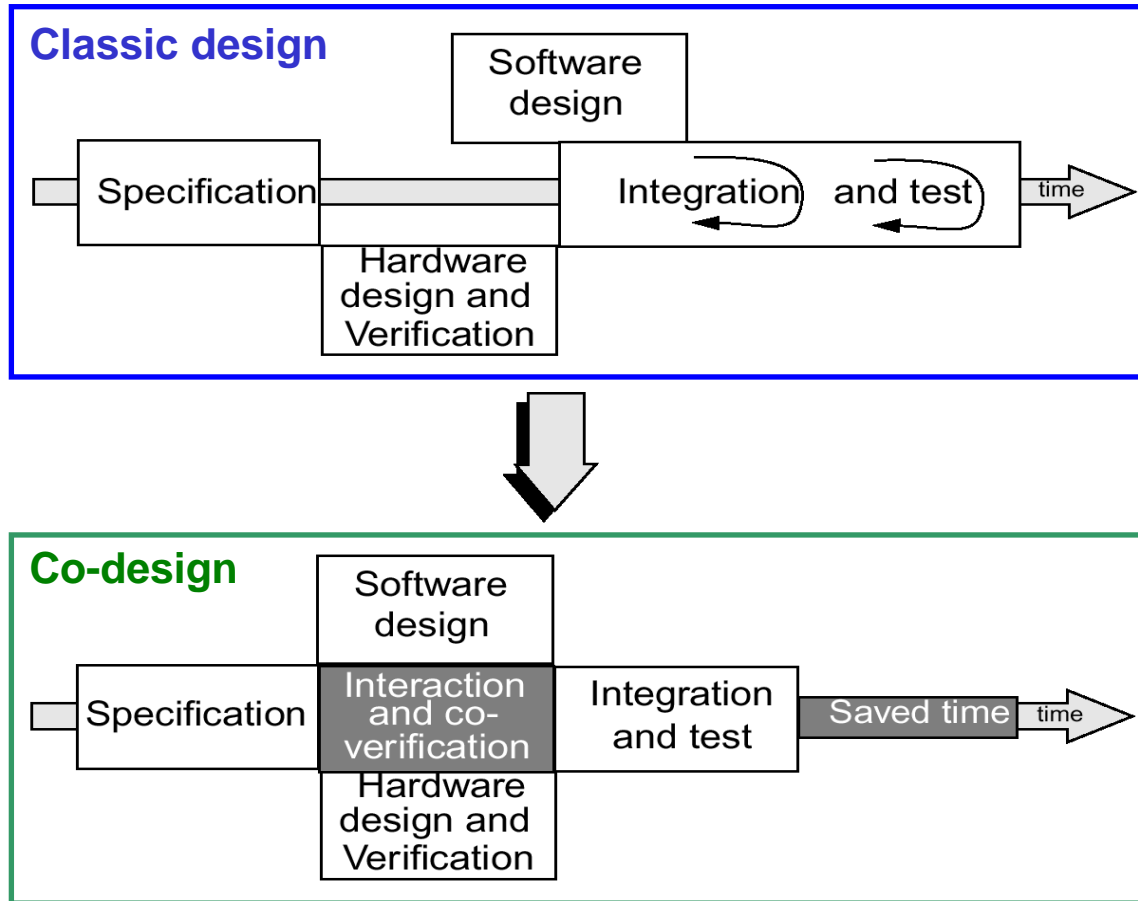
# Time-to-market constraints

- **Time-to-market constraints:** in highly competitive markets, it is critical to catch the **market window**: short delay may have catastrophic financial consequences, even if the quality of the product is excellent.
- **Development time has to be reduced** and some ways to achieve that are:
  - ✓ Efficient design methodologies.
  - ✓ Efficient design tools.
  - ✓ Reuse of previously designed and verified parts (hardware and software).
  - ✓ Use of existing hardware-software prototyping platforms.
  - ✓ Design team understanding **both software and hardware**.



# What is hardware-software co-design ?

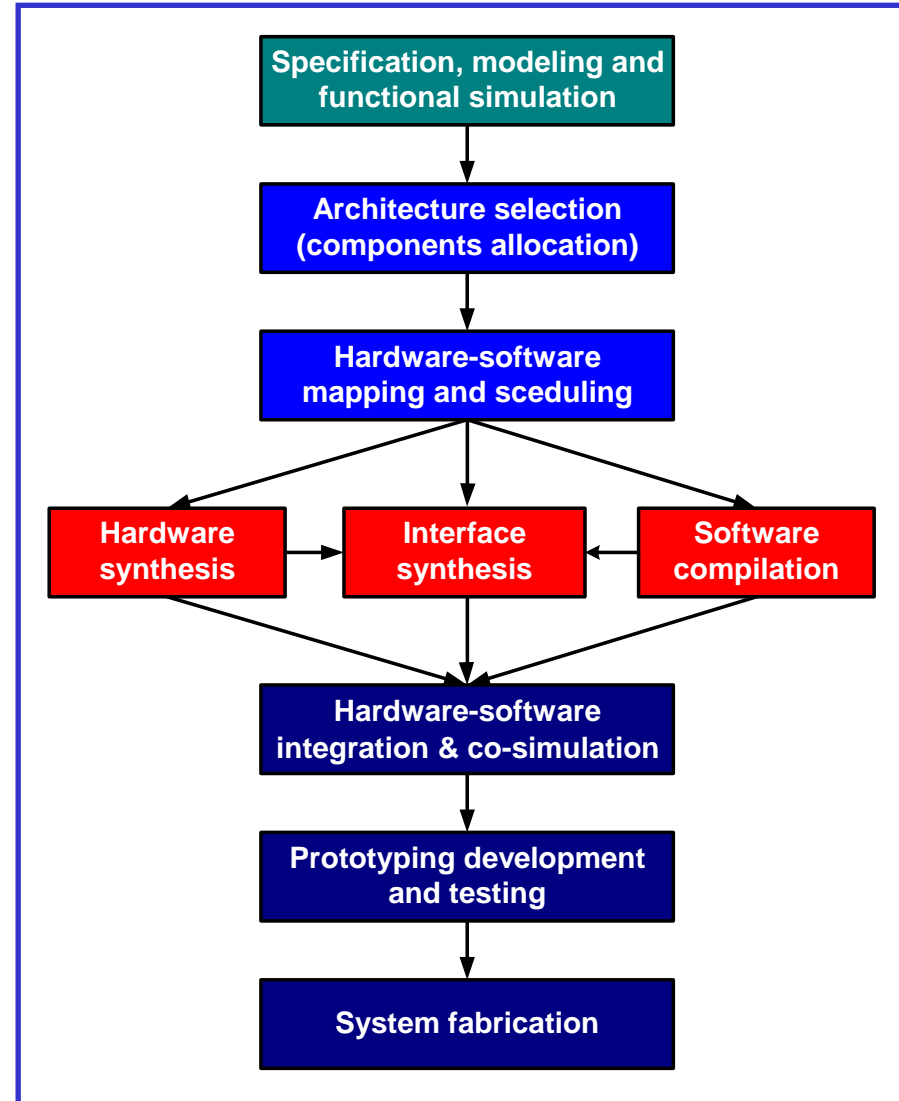
Co-design is the concurrent design of hardware and software components of a digital system.



*The wall between hardware and software must be torn down !*

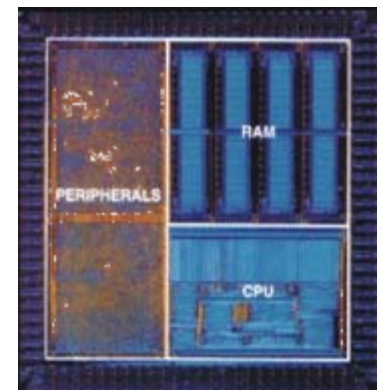
# Hardware-software co-design flow

- **Specification and modeling** (co-specification, functional co-simulation).
- **High-level co-synthesis:**
  - ✓ Architecture selection.
  - ✓ Components allocation (processing elements, storing elements and communication elements).
  - ✓ Tasks hardware-software mapping.
  - ✓ Scheduling of the several tasks.
- **Low-level co-synthesis:**
  - ✓ Hardware synthesis.
  - ✓ Software compilation and code generation.
  - ✓ Interface synthesis.
- **Integration, simulation, prototyping, fabrication.**
- All steps are supported by CAD tools.



# Difficulties in hardware-software co-design

- The difficulties in designing embedded systems are due to the fact that such systems has **high complexity**, **are dedicated towards a certain application**, and must be **efficient** in what concerns:
  - ✓ Run-time.
  - ✓ Power consumption.
  - ✓ Code-size (low memory requirements).
  - ✓ Cost (minimization of hardware resources).
  - ✓ Development time (time to market).
  - ✓ Size and weight.
- In order to achieve all the above, embedded systems have to be highly optimized.
- Both hardware and software aspects have to be considered simultaneously (co-design) in order to achieve:
  - ✓ A good solution by balancing hardware & software resources (flexibility).
  - ✓ Exploration of more design alternatives.
  - ✓ Design of systems-on-chip (optimized complex systems).



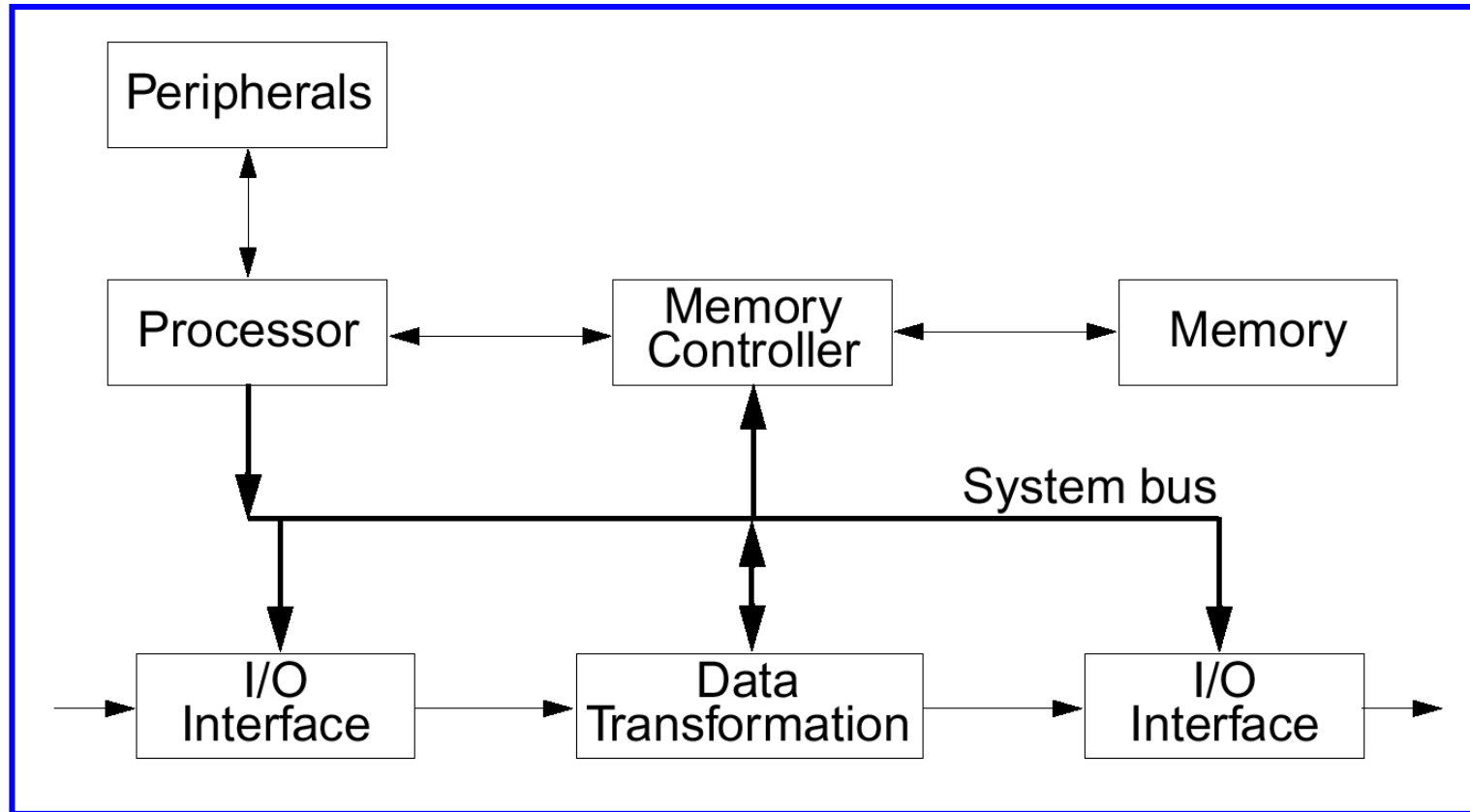


# What is a System-on-Chip (SoC) ?

---

- System-on-chip is an **integrated circuit** that implements most or all of the functions of a complete electronic system, which solves an embedded application.
- It is a **heterogeneous system**: may include hardware and software parts, control and data-processing functionality, digital and analog parts etc.
- Contains more than a single processor: memory modules, custom circuitry, I/O peripherals, A/D or D/A converters etc.

# A typical System-on-Chip

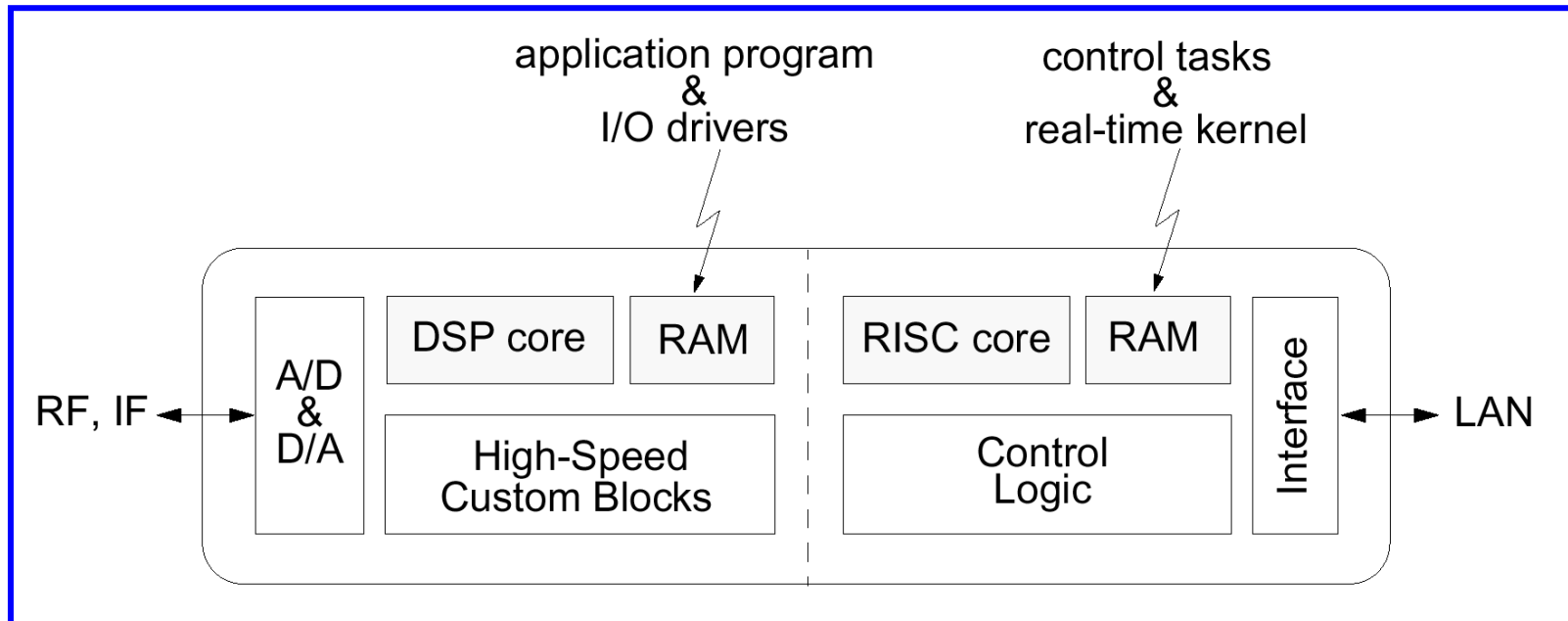


# Components of a typical System-on-Chip (cont'd)

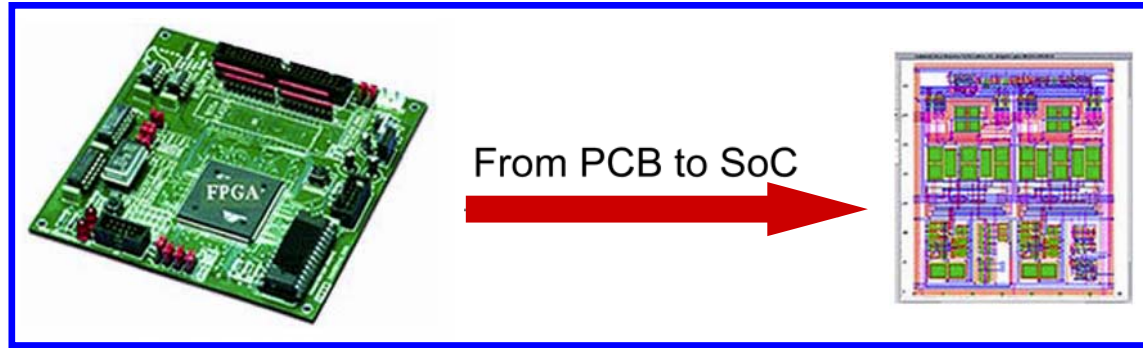
- Programmable processor cores:
  - ✓ Algorithms and protocols become increasingly complex, and this makes their implementation in hardware difficult.
  - ✓ Modern processors are fast as a result of their sophisticated design.
  - ✓ Upgrading the software implementation is easy (flexibility).
- Custom hardware is still quite useful:
  - ✓ High performance for time-critical task.
  - ✓ Low energy consumption.

# Typical System-on-Chip example

**Example:** SoC structure for the implementation of a typical wireless telecommunication application.



# Why do we need System-on-Chips ?



- System-on-chips now are technologically possible: today's chips can contain up to 100 million transistors (according to the Moores Law, approximately every 18 months the number of transistors on a single chip doubles).
- Higher performance: fast data transfer compared to board designs.
- Lower energy consumption: multi-component designs need additional drivers and interfaces for inter-component and inter-board connections.
- Reduced size: components connected on a PCB can now be integrated onto a single chip.
- Lower cost.
- Increased reliability and design security.

# Problems with System-on-Chip design

---

- Increased system complexity mainly due to the heterogeneity (analog along with digital parts, processors along with custom hardware, different memory types etc.), and due to the integrated nature.
- This creates **problems to the design phase** (expertise in different design areas is needed and the integration is a quite demanding task), and to the **technology** (different processes have to be incorporated onto a single die).
- Traditional hardware design methodologies do not work.
- Increased **verification** requirements.
- **Design productivity vs. time-to-market pressure.**
- Solutions to overcome complexity, low design productivity and time-to-market pressure is to use advanced SoC design methodologies and tools and mainly to re-use IP (Intellectual Property) blocks in SoC designs.

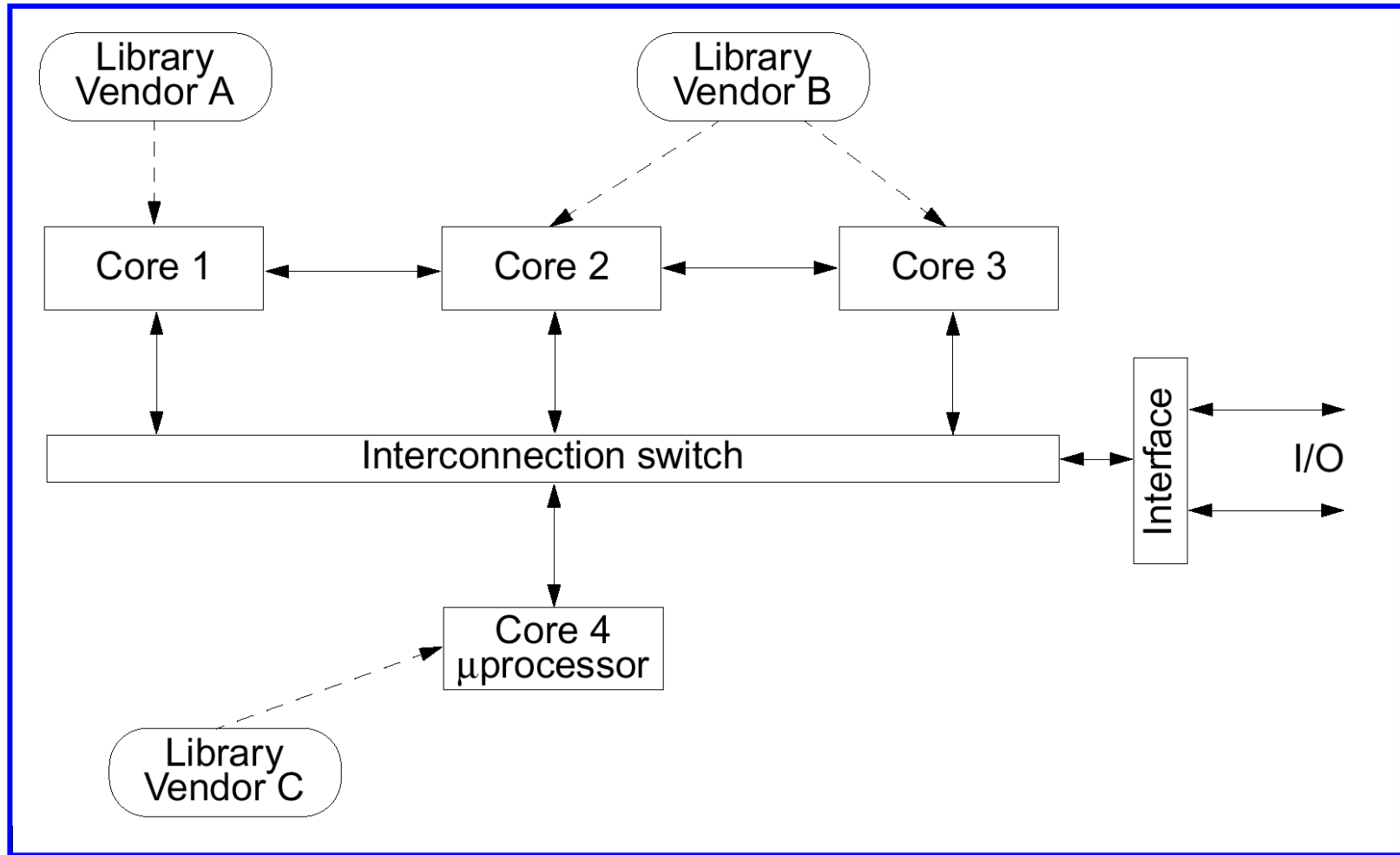
# IP-based System-on-Chip design

---

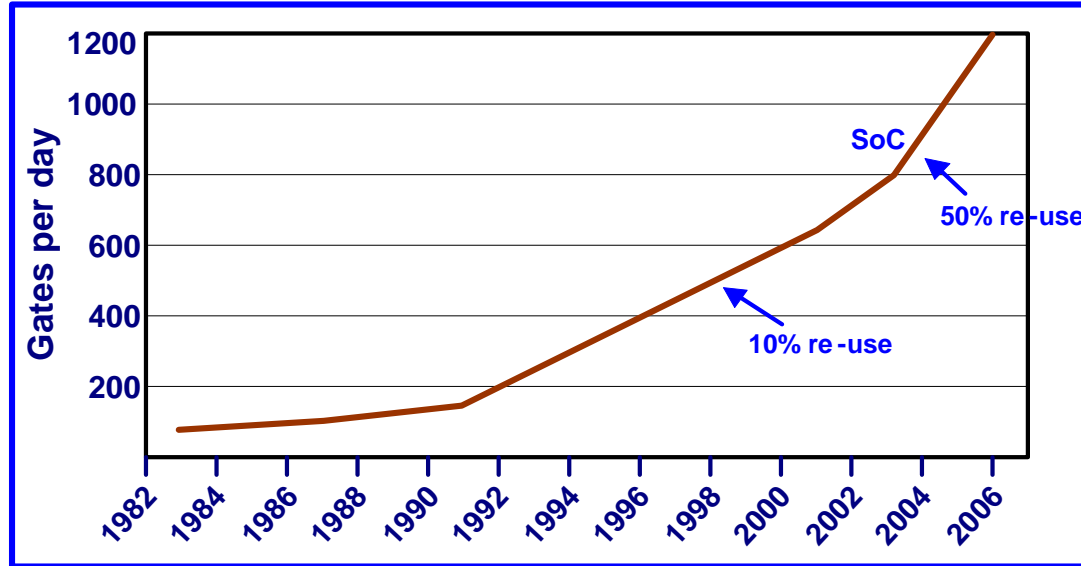
- IP-based design is the process of composing a new system by reusing existing components.
- Possible IP blocks to be used:
  - ✓ Microprocessors (ARM, MIPS, PowerPC, SPARC etc.)
  - ✓ Interfaces (USB, PCI, UART etc.)
  - ✓ Encoder and decoders (JPEG, MPEG, Viterbi etc.)
  - ✓ Memories (SRAM, Flash etc.)
  - ✓ Microcontrollers (HC11 etc.)
  - ✓ DSPs (TI, Oak etc.)
  - ✓ Transformers (FFT, IFFT etc.)
  - ✓ Networking blocks (Ethernet, ATM etc.)
  - ✓ Encryption blocks (DES, AES etc.).
- The increasing need of SoCs is forcing design houses and vendors to develop high-quality IP blocks (a new industry has been developed !).



# IP-based System-on-Chip design (cont'd)



# System-on-Chip design productivity and cost



## Cost of 50M transistor SoC

- Gates: 12.5 M
- Productivity (gates/day): 1200
- Total engineer months: ~ 520
- Engineer cost per month: ~ 8.5 K€
- Total personnel cost: ~ 4.5 M€
- Additional NRE (masks, CAD): ~ 4 M€
- Total cost: ~ 8.5 M€

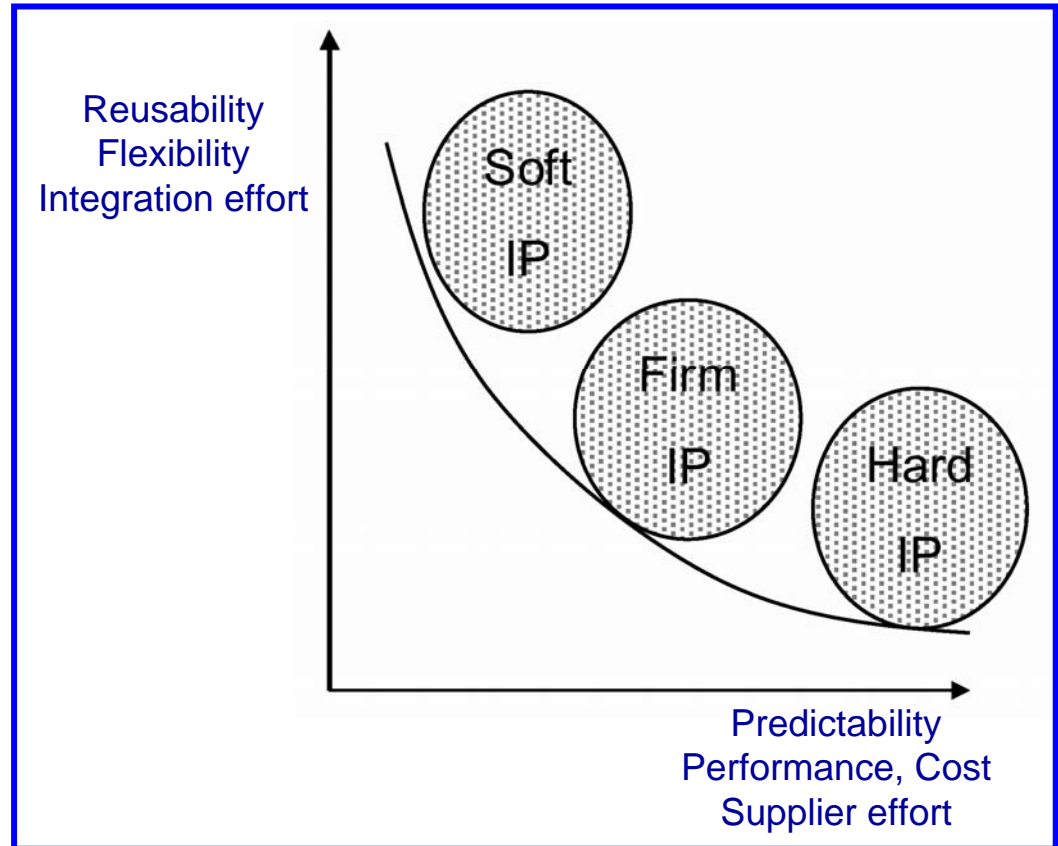
# Main issues in IP block design

---

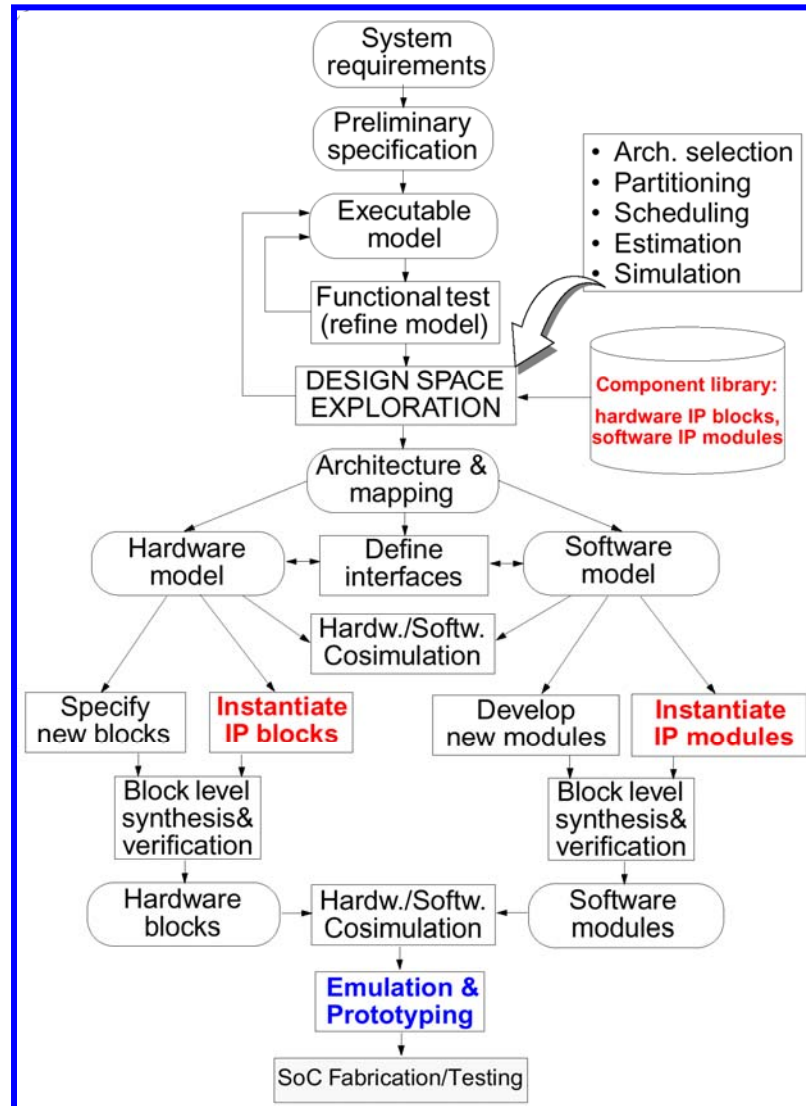
- How to specify an IP block for a reuse library: functionality, timing information, interface properties, achieved speed, power consumption etc.
- Much effort has to be allocated for:
  - ✓ Specification, simulation, estimation and exploration.
  - ✓ Integration (interfaces definition and implementation).
  - ✓ Verification and testing for many operating conditions and inputs.

# Types of IP blocks

- **Hard IP blocks:**  
Fully designed, placed and routed by the supplier. It is offered as a completely validated layout with definite timing characteristics and offers fast integration but low flexibility.
- **Firm IP blocks:**  
Technology-mapped gate-level netlist and offers flexibility during place and route, but with lower predictability.
- **Soft IP blocks:**  
Synthesizable RTL or behavioral descriptions. Require much effort for integration/verification, but offers maximal flexibility.



# System-on-Chip design flow

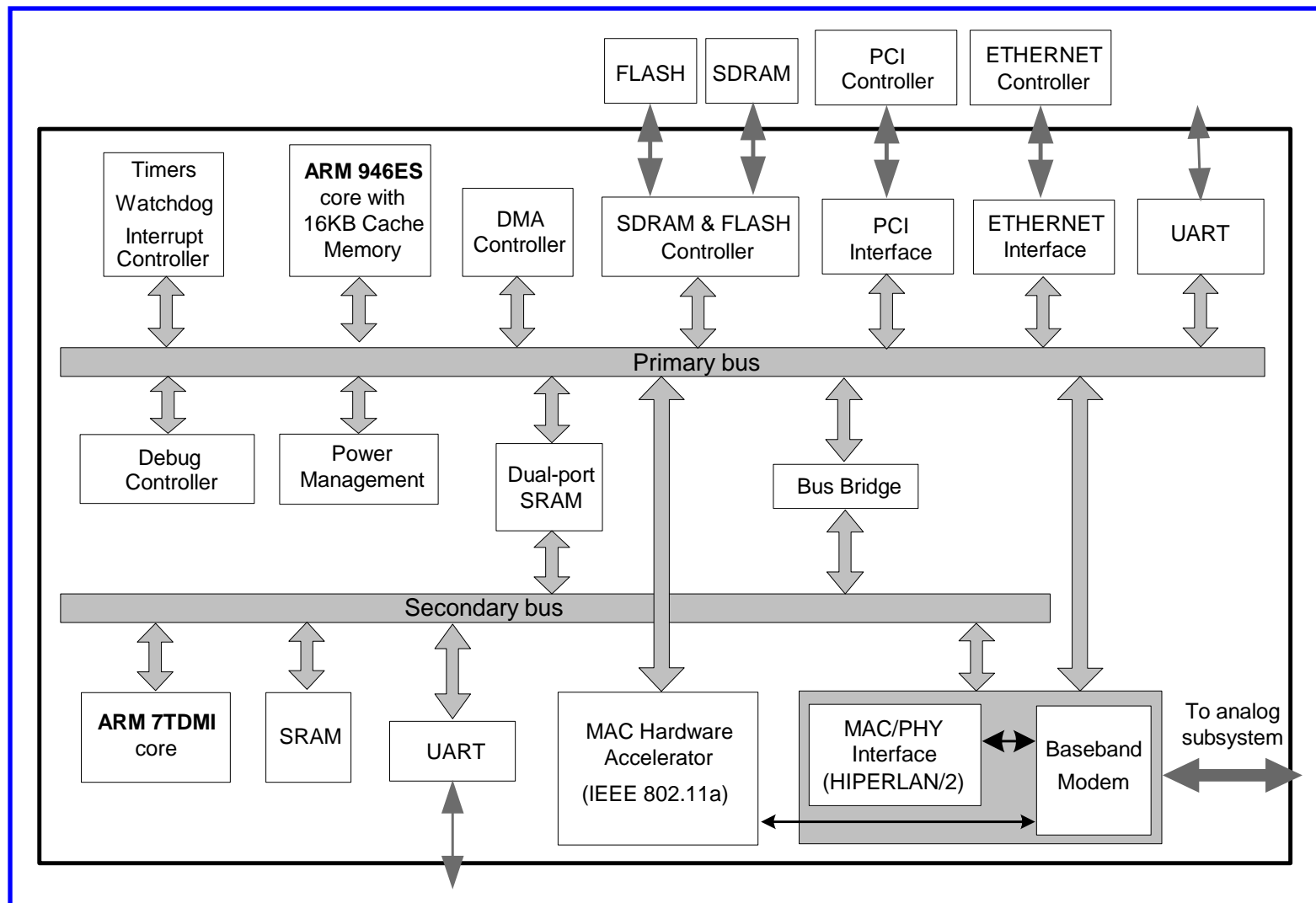


# Example: Wireless LAN System-on-Chip

---

- Flexible architecture to implement the digital part of the physical layer functionality of two wireless LAN standards (5 GHz band): HIPERLAN/2, IEEE 802.11a.
- Implements the operations of CL and DLC of the HIPERLAN/2 and the lower-MAC layer of the IEEE 802.11a standard.
- Contains two embedded microprocessors:
  - ✓ ARM946E-S for the implementation of the high layers of the HIPERLAN/2 standard.
  - ✓ ARM7TDMI for the implementation of the lower-MAC layer of the HIPERLAN/2 standard and for controlling the baseband modem (transmitter/receiver) of the system.
- Also, includes a MAC hardware accelerator (custom block) that implements critical functionality of the MAC layer of the IEEE 802.11a.
- Various peripherals: test and debug controller, power controller, Ethernet and PCI interfaces, SDRAM controller, DMA controller, UARTs.

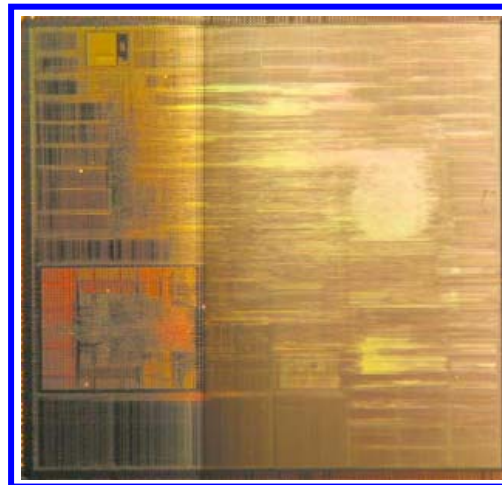
# Example: Wireless LAN System-on-Chip (cont'd)







# Example: Wireless LAN System-on-Chip (cont'd)

Process technology	0.18 $\mu\text{m}$ CMOS
Supply voltage	1.8 V (core), 3.3 V (I/O pads)
Operating frequency	80 MHz (some modem's blocks in 40 MHz)
Average power consumption	554 mW
Equivalent gates count	4,400,000
Transistors count	> 17,500,000
Pins count	456 (about 130 are for testing/debugging purposes)
Packaging	BGA 35mm x 35mm
Chip area	9.408mm x 9.408mm $\approx$ 88.5 sq. mm
Core area	8.576 mm x 8.576 mm $\approx$ 73.5 sq. mm
Area occupied by logic	43 sq. mm
Area occupied by memories	30.5 sq. mm
Total length of interconnections	47 m (six metal layers)



Design	
Fabrication	

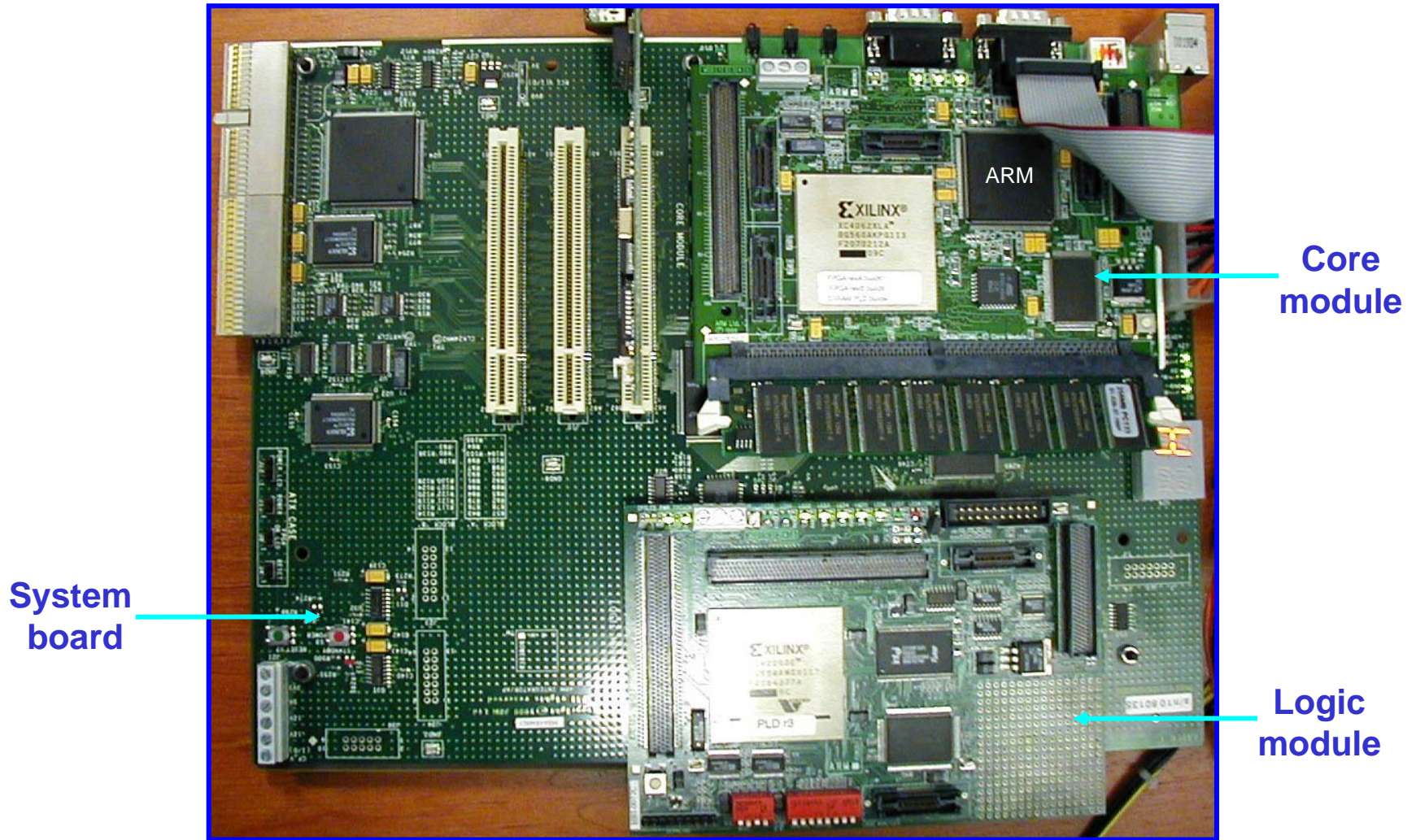


# Prototyping platforms

---

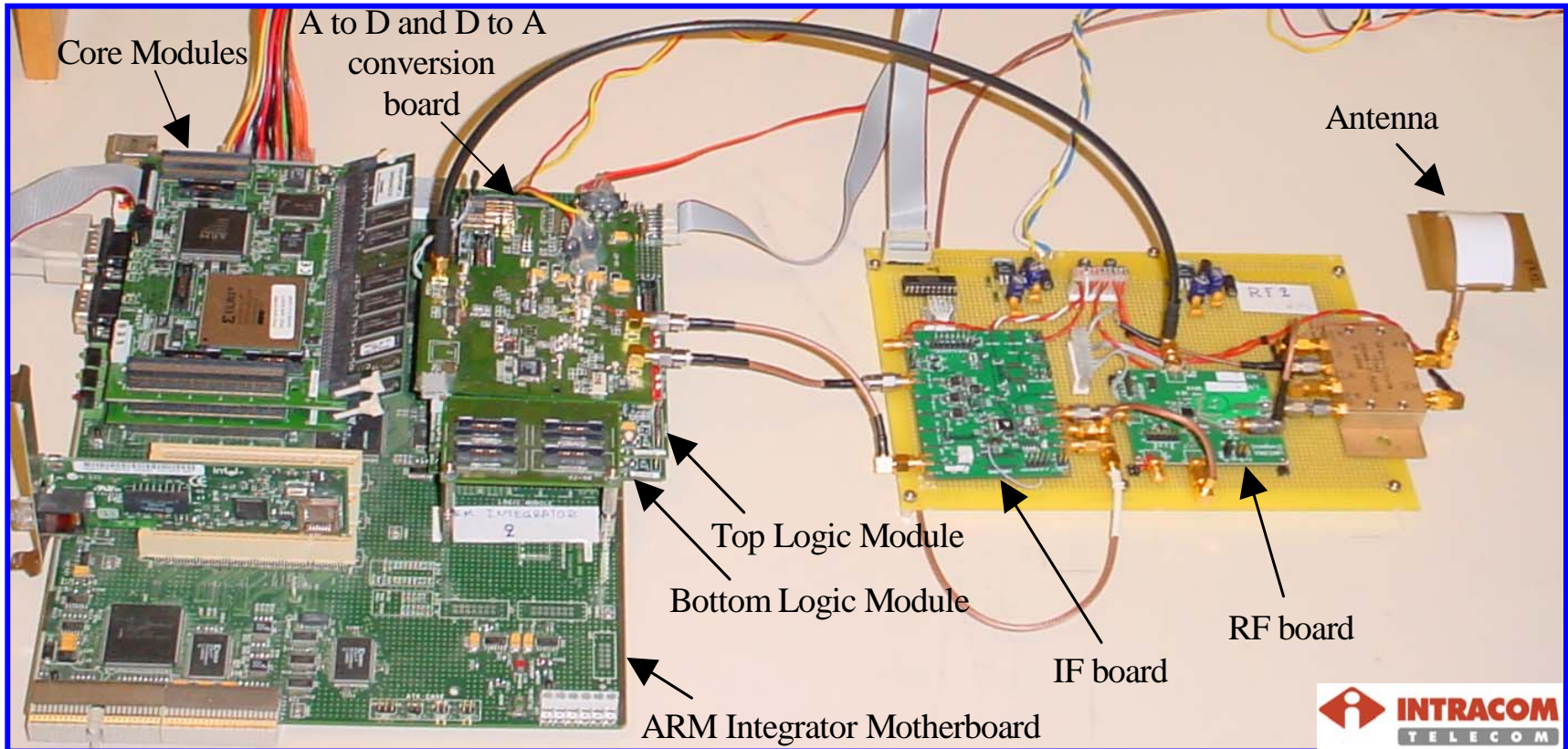
- Prototyping platforms are used after the simulation (or as an alternative to the simulation) in order to prototype the SoC design into an FPGA-based board, before its fabrication.
- Main characteristics of prototyping platforms:
  - ✓ Offer an accurate representation of the design since they are actual implementations and not just simulations.
  - ✓ Faster than simulations: they can reproduce a problem after several seconds of execution, while in HDL simulation environments things are much slower.
  - ✓ However, they are not exact replicas of the final SoC, and they cannot run at the same frequency with the real silicon SoC.
  - ✓ The design can be mapped relatively quickly (hours or days).
  - ✓ Debugging support is usually included.
  - ✓ However, tasks such as design partitioning (to the available FPGAs), clock tree routing, bus handling and memory mapping are complex and difficult.
  - ✓ Can be expensive for large designs, and sometimes they can lead to resource bottleneck (a group have to wait for another group to finish using the platform).

# Example: ARM integrator platform





# Wireless LAN System-on-Chip prototyping



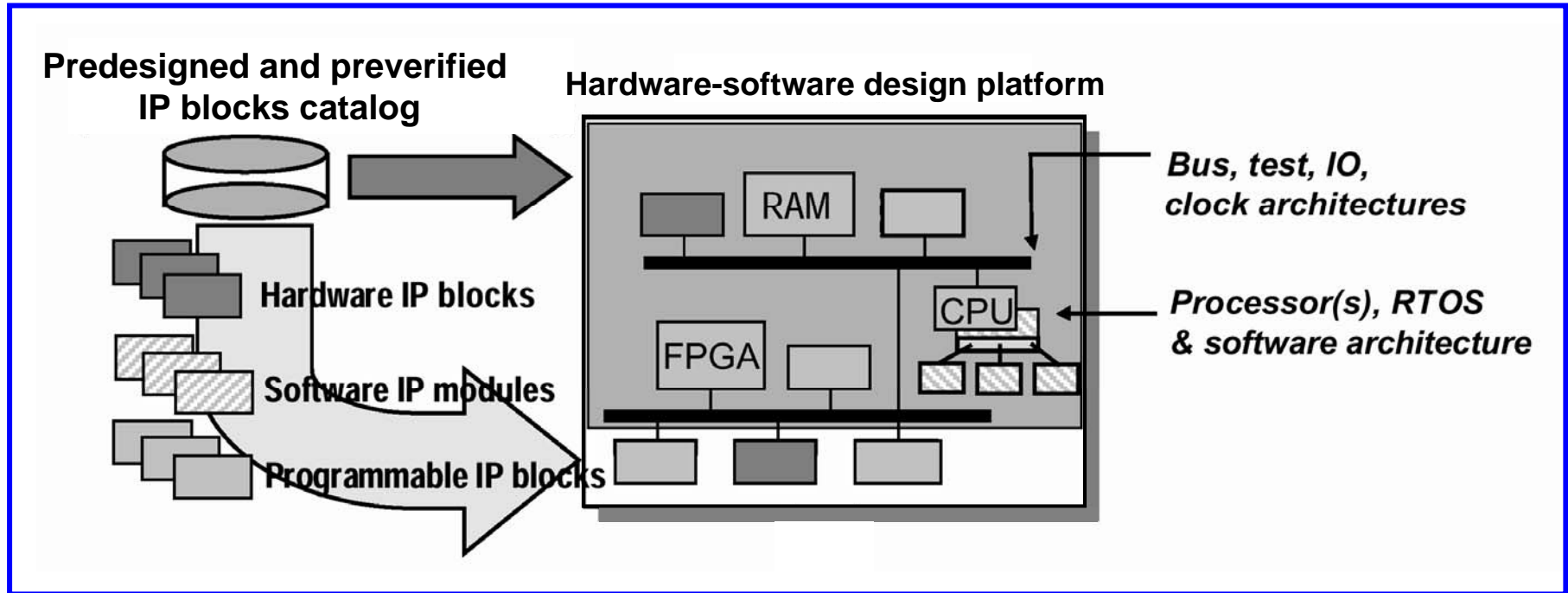
- Two ARM7TDMI core modules (one implementing the upper layers of the protocol and the second controlling the baseband modem and implementing the lower MAC protocol layer).
- Two logic modules with XILINX Virtex E 2000 FPGAs implementing the baseband modem functionality. The average FPGA utilization was 87%.

# IC devices with embedded reconfigurable resources

---

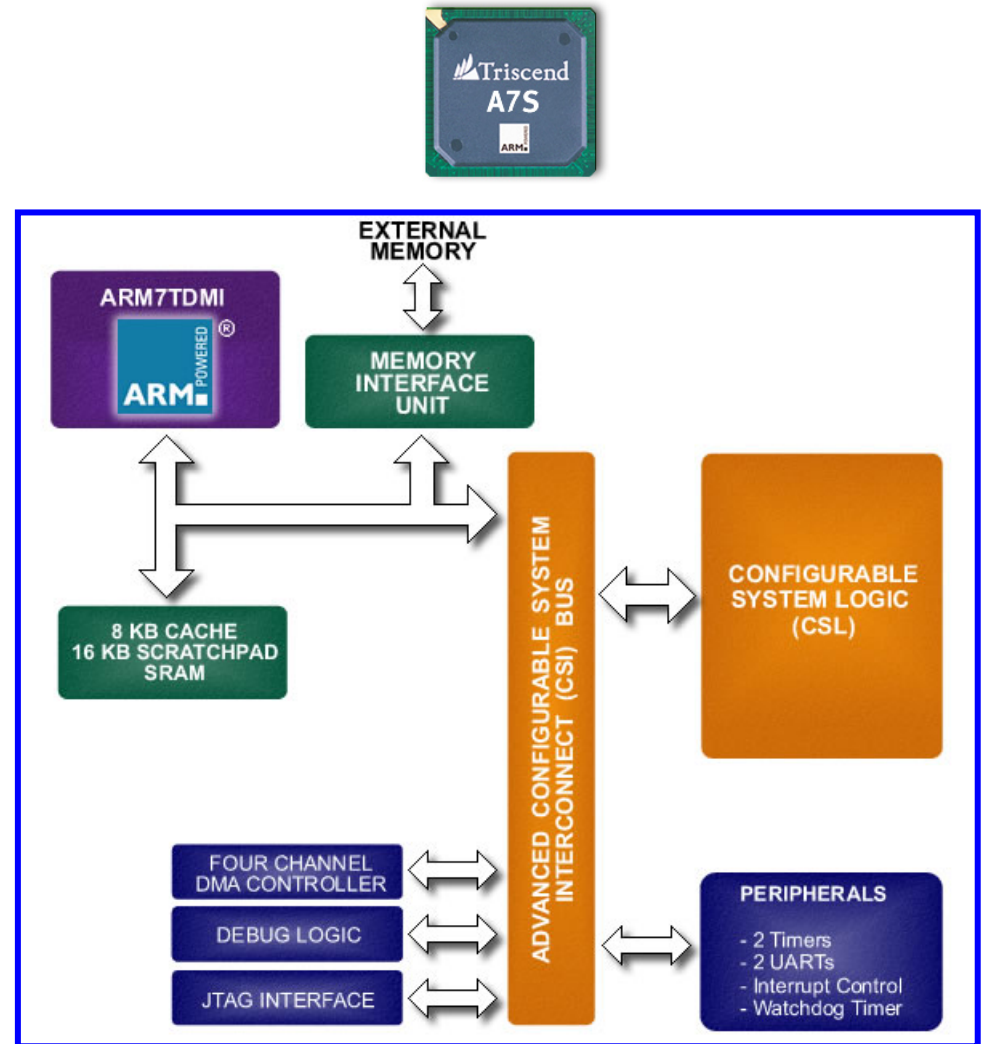
- Custom SoC designs have very high cost, making them practical only when we have production of a very large number of chips (millions).
- IC devices with embedded reconfigurable devices (also called **hardware-software design platforms**) have quite less cost, providing developers with an alternative solution to custom SoCs and standard processors with external peripherals.
- Hardware-software design platform is a stable SoC architecture for a target application or family of applications that is based on a **combination of a programmable CPU and a reconfigurable array of data-path units**.
- It can be extended and customized relatively fast and easy.
- The use of such platforms increases the **productivity** and the **success probability**, and **reduces the design time**.
- Derivative designs (implementing similar applications) can be easily created by using software or hardware modifications.
- Diverse applications each requires a different platform (it is difficult to use the same platform for a telecom application where the control functionality is dominant and for a multimedia application where the data-processing tasks are dominant).

# Hardware-software design platforms concept



# Hardware-software design platforms examples

- **Triscend A7S** is a 32-bit reconfigurable system-on-chip.
- Combines onto a single chip:
  - ✓ A 32-bit ARM7TDMI embedded processor core.
  - ✓ A flexible Configurable System Logic (CSL) matrix.
  - ✓ A robust memory subsystem.
  - ✓ A high-performance custom internal bus.
  - ✓ Other system peripheral functions.
- **Altera Excalibur** platform is similar and combines an ARM9 with a PLD device.

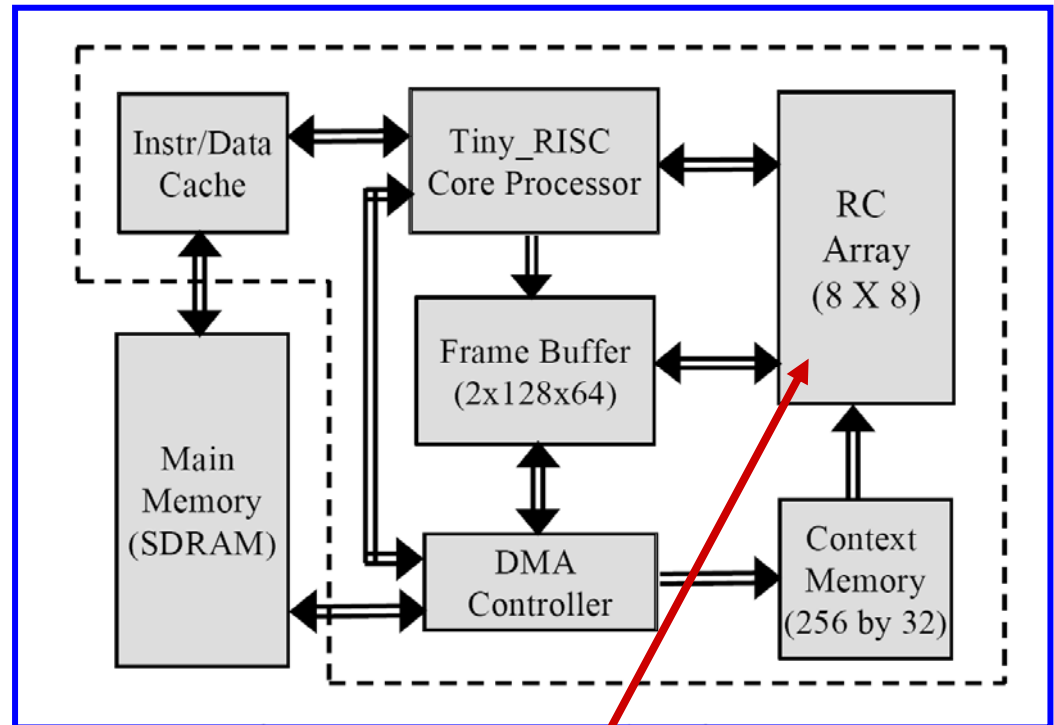




# Hardware-software design platforms examples (cont'd)

- **MorphoSys** platform contains a Tiny RISC processor that is a 4-stage pipeline, MIPS-like RISC machine with 16 32-bit registers, 32-bit ALU/shift unit and on-chip data cache memory
- The reconfigurable array consists of an 8x8 matrix of Reconfigurable Cells (RC).
- Each RC comprises an ALU-Multiplier, a shift unit, input multiplexers, and a register file with five 16-bit registers.

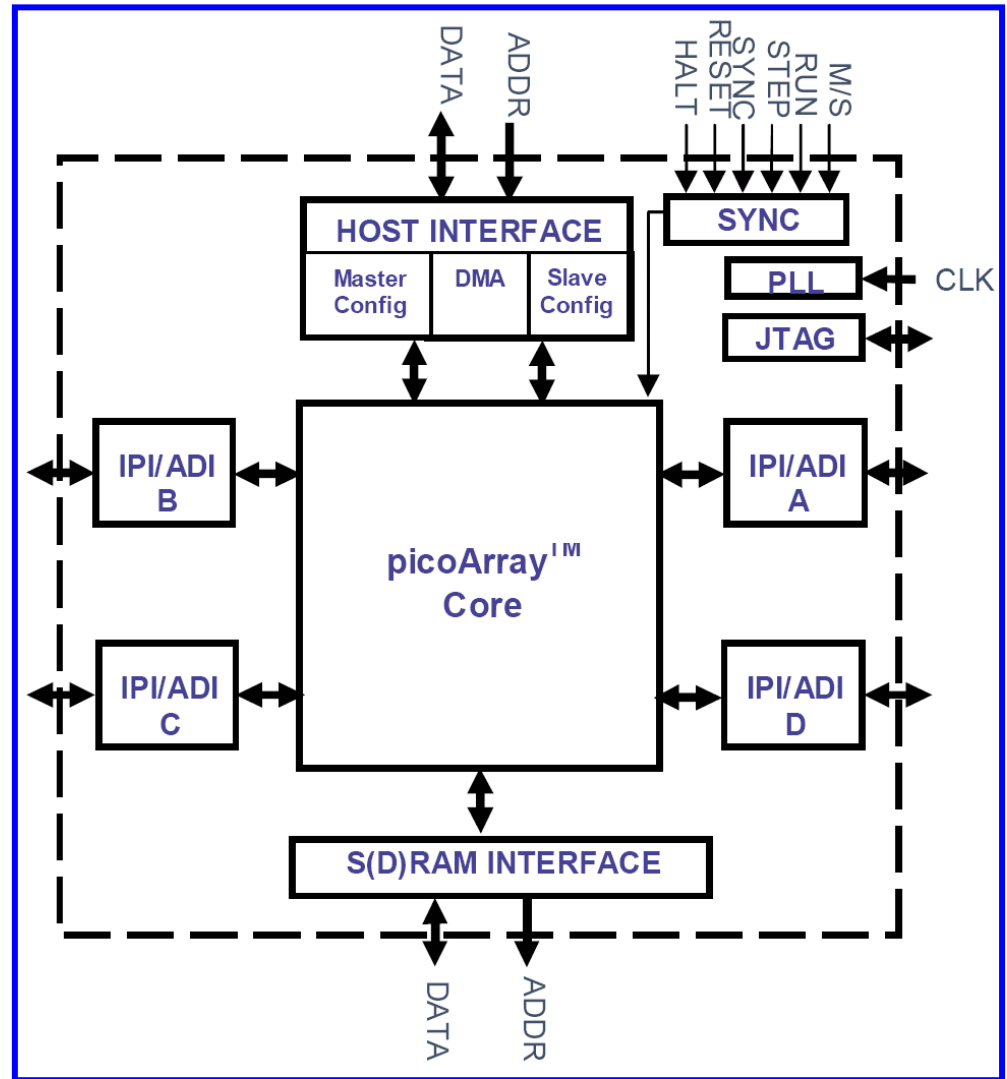
University of California at Irvine



It is also available as autonomous IP block (embedded reconfigurable core) with programmable capabilities that can be embedded in several SoCs and ASICs.

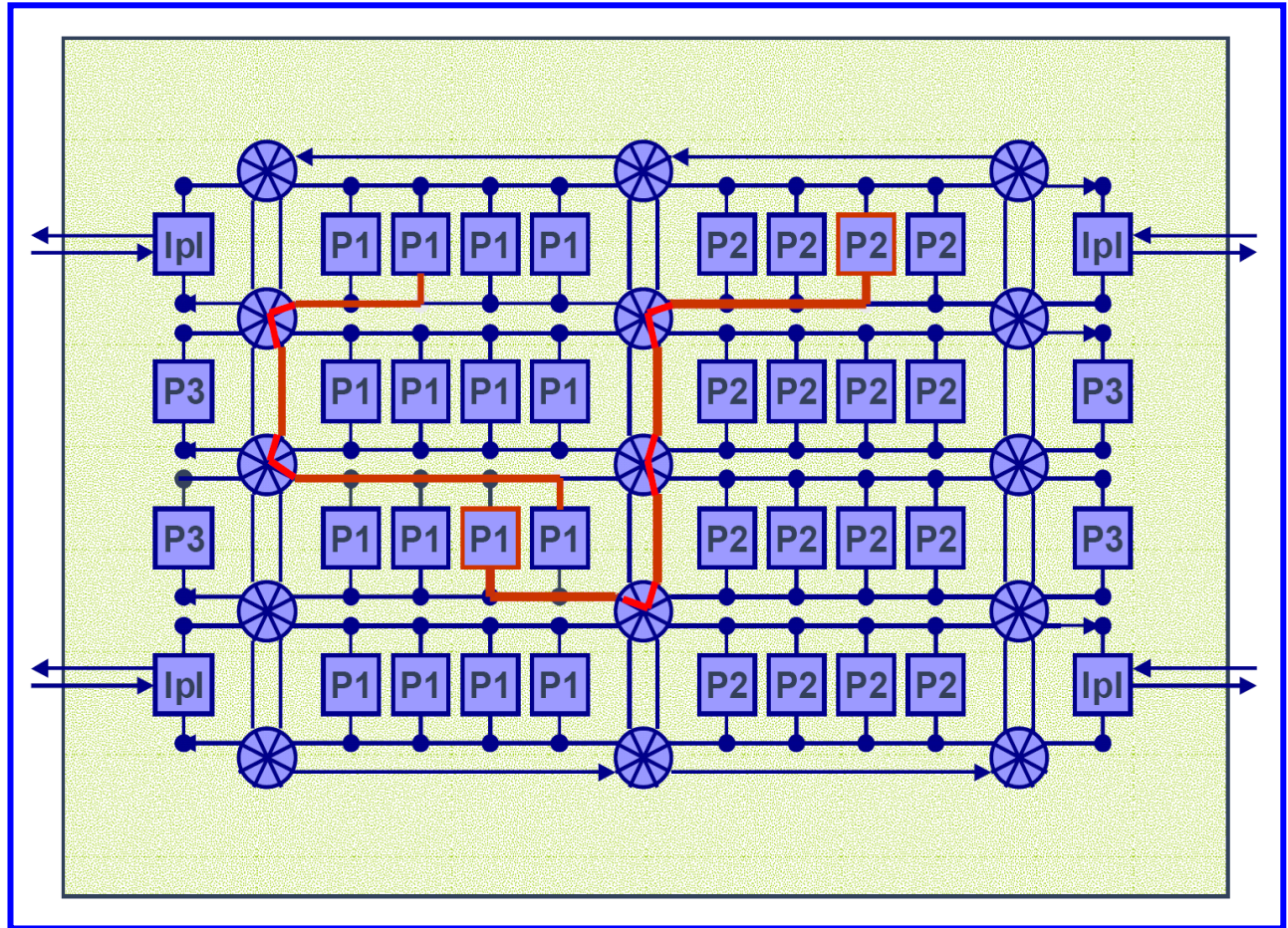
# Hardware-software design platforms examples (cont'd)

**picoChip** platform consists of configurable array of processors (picoArray) and peripherals (external microprocessor interface, external memory interface, interfaces allowing multiple arrays to be connected together).



# Hardware-software design platforms examples (cont'd)

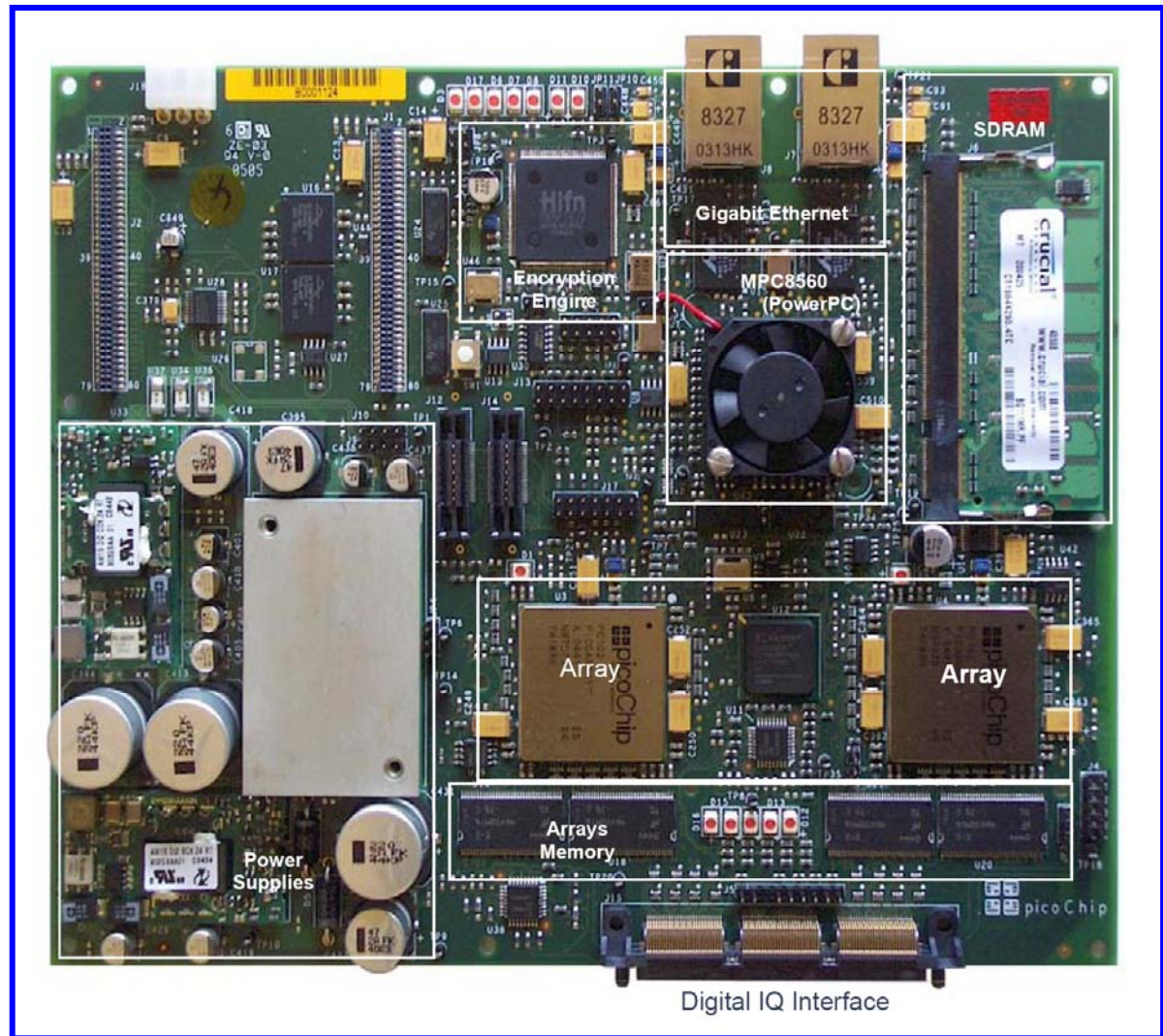
The array contains 322 elements; 308 processors (16-bit architecture with 3-way LIW and local memory) and 14 co-processors, all connected by programmable interconnect modules.





# Hardware-software design platforms examples (cont'd)

- The flexible nature of the picoArray technology allows the implementation of several communications standards (IEEE 802.11 - wireless LAN, IEEE802.16 - outdoor wireless).
- The physical layer is implemented on the arrays, while the MAC layer of the standards is implemented on a PowerPC external processor.
- An encryption engine implementing basic standards is also available.



# Conclusions

---

- Embedded systems are dedicated and application-specific, contains at least one programmable component, requires continuous interaction with the environment in real time and must meet several constraints.
- This nature of today's embedded systems faces developers and engineers with new problems when it comes to specifying, simulating, designing and optimising such complex systems.
- Implementations are typically comprised of general purpose or application-specific programmable components, dedicated processing components, communication and memory modules.
- The design of embedded systems is driven by several constraints: performance, power consumption, size and weight, cost, safety and security, reliability, time to market.
- Optimization involves the simultaneous consideration of these incomparable and often competing objectives.
- As a consequence, much design effort and advanced CAD tools are necessary in order to handle the complexity of today's embedded systems & applications.

# Conclusions (cont'd)

---

- With the current technology and in the context of increasingly complex applications and strong market pressure, system-on-chip is a natural approach for several embedded applications.
- Programmable components provide the necessary flexibility and custom hardware is needed for time-critical tasks implementation and for low power consumption.
- The real bottleneck in SoC design is productivity.
- Solutions are the IP-based design (blocks reuse), and the improvement of existing design methodologies and tools.
- Prototyping platforms are used after the co-simulation (or as an alternative to the simulation) in order to prototype and test the SoC design into an FPGA-based board, before its fabrication.
- As alternatives to FPGAs and custom SoCs, the designers today can use IC devices with embedded reconfigurable resources (hardware-software design platforms) as well as embedded reconfigurable cores.