Introduction to Embedded Computing

Ref:
Chap. 1 of High-Performance Embedded Computing

Definition for Embedded Systems

Embedded systems (ES) = **information processing systems embedded into a larger product**

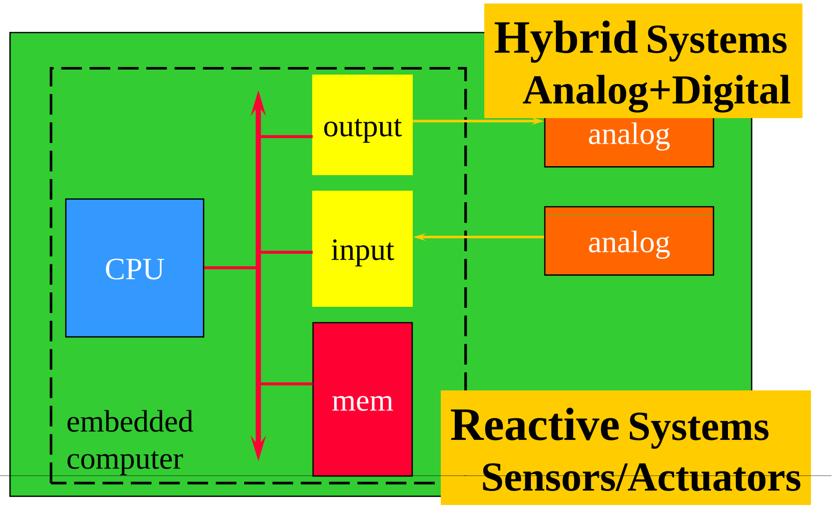
keyword: a specific function, embedded within a larger device, heterogeneous and reactive

Examples:

Main reason for buying is **not** information processing

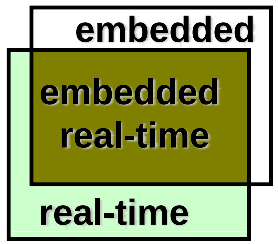
Source: P. Marwedel

Embedding a computer



Characteristics of Embedded Systems

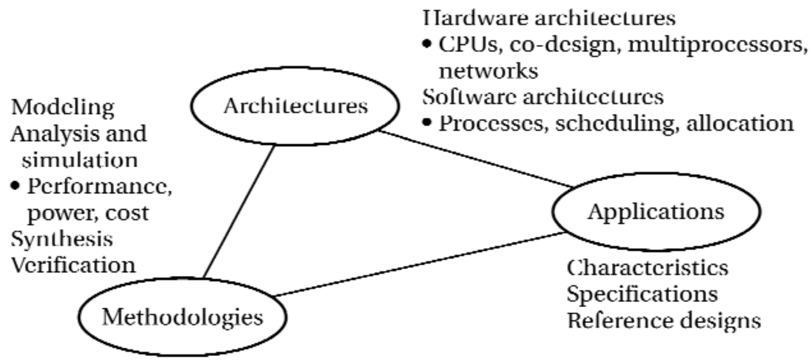
- Application specific
- Efficient
  - energy, code size, run-time, weight, cost
- Dependable
  - Reliability, maintainability, availability, safety, security
- Real-time constraints
  - Soft vs. hard
- Reactive - connected to physical environment
  - sensors & actuators
- Hybrid
  - Analog and digital
- Distributed
  - Composability, scalability, dependability
- Dedicated user interfaces



Designing embedded systems

- No one architecture (hardware or software) can meet the needs of all applications.
- We need to be able to design a system from the application:
  - Quickly and efficiently.
  - With reliable results.

Aspects of embedded system design



Architectures

- Both hardware and software architectures are important.
- The structure of the system determines cost, power, performance.
- Different application requirements lead us to different architectures.

Applications

- You can’t design the best embedded systems if you don’t know anything about your application.
- You can’t be an expert in everything.
  - But a little knowledge goes a long way.
- Domain expertise helps you make trade-offs:
  - Can the requirements be relaxed?
  - Can one requirement be traded for another?

Methodologies

- We must be able to reliably design systems:
  - Start from requirements/specification.
  - Build a system that is fast enough, doesn't burn too much energy, and is cheap enough.
  - Be able to finish it on time.
  - And know before we start how difficult the project will be.
- Invention lets us get around some key technical barriers.
- Methodology keeps us going.

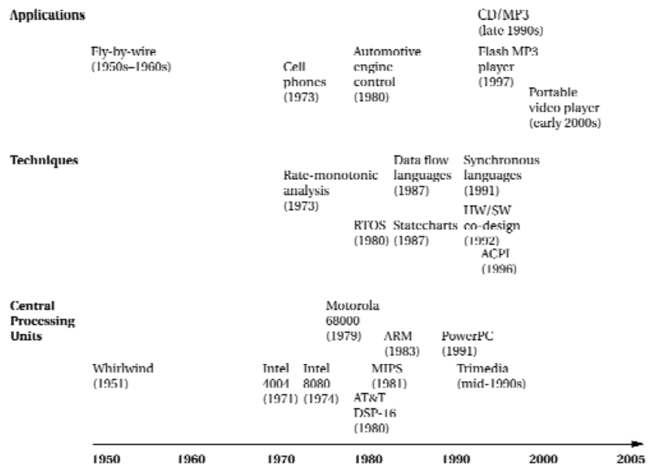
Modeling

- A key aspect of methodology is modeling.
  - Work with a simplified version of the object.
- Modeling helps us predict the consequences of design decisions.
- Models help us work faster (once we have the model).
- We can afford to use models if we can reuse them in several designs---methodology relies on and enables modeling.

Disciplines in embedded computing

- Core areas:
  - Real-time computing.
  - Hardware/software co-design.
- Closely related areas:
  - Computer architecture.
  - Software engineering.
  - Low-power design.
  - Operating systems.
  - Programming languages and compilers.
  - Networking.
  - Secure and reliable computing.
  - Signal processing … (applications!!)

History of embedded computing



Early history

- Late 1940's: MIT Whirlwind computer was designed for real-time operations.
  - Originally designed to control an aircraft simulator.
- First microprocessor was Intel 4004 in early 1970's.

Tied to advances in semiconductors

- A typical chip in near future
  - 50 square millimeters
  - 50 million transistors
  - 1-10 GHz, 100-1000 MOP/sq mm, 10-100 MIPS/mW
- Cost is almost independent of functionality
  - 10,000 units/wafer, 20K wafers/month
  - $5 per part
  - Processor, MEMS, Networking, Wireless, Memory
    - But it takes $20M to build one today, going to $50+M
- So there is a strong incentive to port your application, system, box to the "chip"

Source: RG UCSD

Trends in Embedded Systems

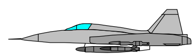
- Increasing code size
  - average code size: 16-64KB in 1992, 64K-512KB in 1996
  - migration from hand (assembly) coding to high-level languages
- Reuse of hardware and software components
  - processors (micro-controllers, DSPs)
  - software components (drivers)
- Increasing integration and system complexity
  - integration of RF, DSP, network interfaces
  - 32-bit processors, IO processors

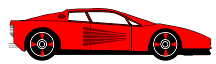
Structured design and composition methods are essential.

Source: RG UCSD

Embedded Systems: Applications

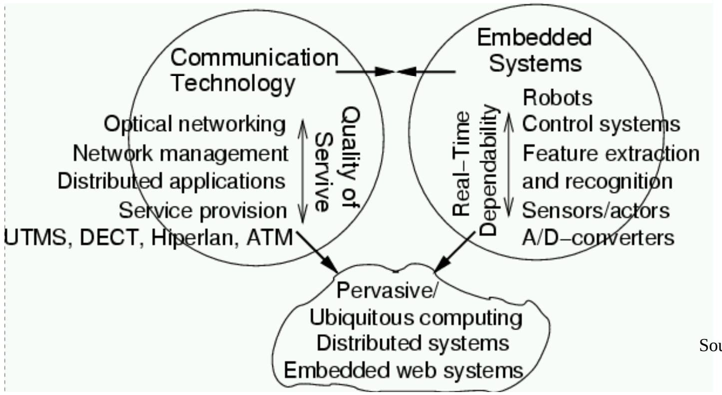
- Transportation
- Industrial process controllers
- Smart buildings
- Medical systems
- Military
- Security
- Robotics
- Computer/Communication products, e.g., printers, FAX machines, ...
- Emerging multimedia applications & consumer electronics
  - e.g., cellular phones, personal digital assistants, video-conferencing servers, interactive game boxes, TV set-top boxes, ...
  - Multimedia => Increasing computational demands, and
    - increased reliance on VLSI, HW/SW integration.





Embedded systems and ubiquitous computing

- Ubiquitous computing: Information anytime, anywhere.
- Embedded systems provide fundamental technology.

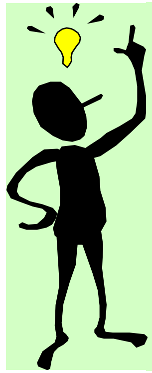


Source: P. Marwedel

Importance of Embedded Software

"... the New York Times has estimated that the average American comes into contact with about 60 micro-processors every day...." [Camposano, 1996]

Latest top-level BMWs contain over 100 micro-processors



Most of the functionality of embedded systems will be implemented in software!

Software complexity

- Exponential increase in software complexity
- In some areas code size is doubling every 9 months [ST Microelectronics, Medea Workshop, Fall 2003]
- *... > 70% of the development cost for complex systems such as automotive electronics and communication systems are due to software development* [A. Sangiovanni-Vincentelli, 1999]

Rob van Ommering, COPA Tutorial, as cited by: Gerrit Müller: Opportunities and challenges in embedded systems, Eindhoven Embedded Systems Institute, 2004

Source: Marwedel

More challenges for embedded SW

- Dynamic environments
- Capture the required behaviour!
- Validate specifications
- Efficient translation of specifications into implementations!
- How can we check that we meet real-time constraints?
- How do we validate embedded real-time software? (large volumes of data, testing may be safety-critical)

Source: Marwedel

Design goals

- Functional requirements: input/output relations.
- Non-functional requirements: cost, performance, power, etc.

Aspects of performance

- Embedded system performance can be measured in many ways:
  - Average vs. worst/best-case.
  - Throughput vs. latency.
  - Peak vs. sustained.

Energy/power

- Energy consumption is important for battery life.
- Power consumption is important for heat generation or for generator-powered systems (vehicles).

Cost

- Manufacturing cost must be paid off across all the systems.
  - Hardest in small-volume applications.
- Manufacturing cost is incurred for each device.
- Lifetime costs include software and hardware maintenance and upgrades.

Other design attributes

- Design time must be reasonable. May need to finish by a certain date.
- System must be reliable; reliability requirements differ widely.
- Quality includes reliability and other aspects: usability, durability, etc.

Example: PDA design

Why did they design it this way?

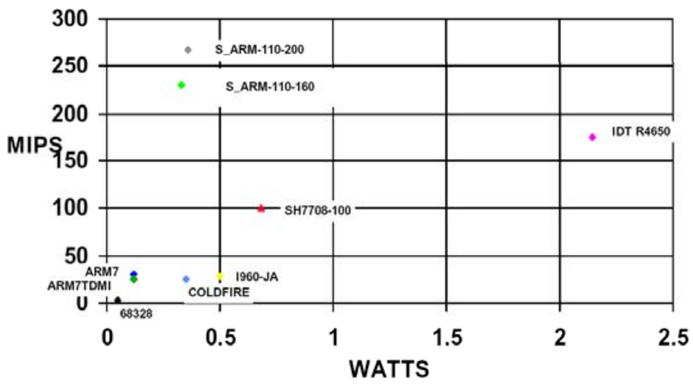
A 'Dragonball*' processor?

We all wanted StrongARMs

*The Dragonball used in the early Palm Pilots is a Motorola 68328

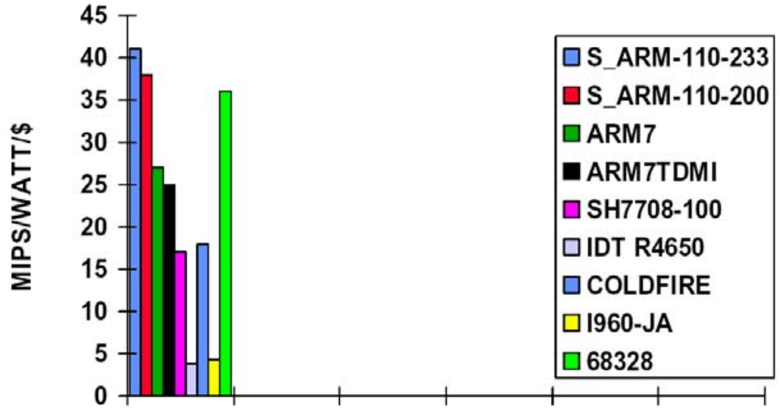
Source: MS HPL

MIPS vs. Watts



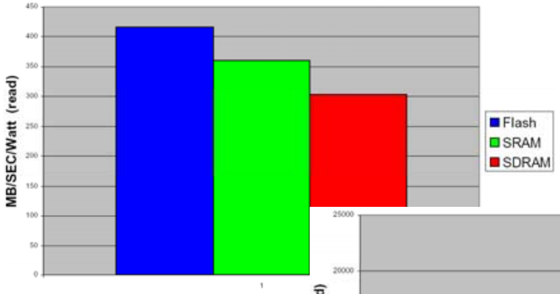
Source: MS HPL

MIPS/W/$

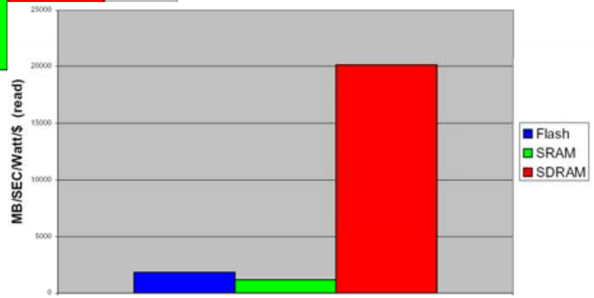


Source: MS HPL

Bandwidth vs. Watt and $

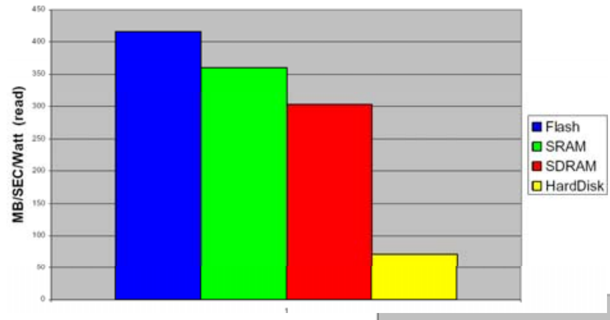


This is why PDAs use SDRAM

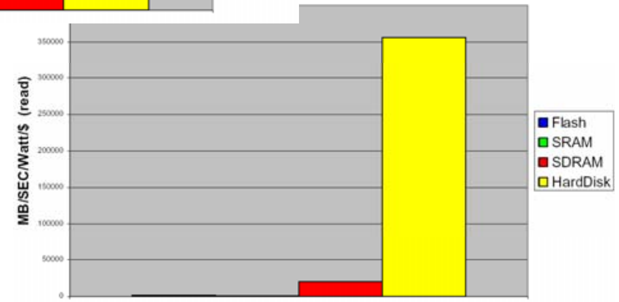


Source: MS HPL

BW/W/$ with hard disk

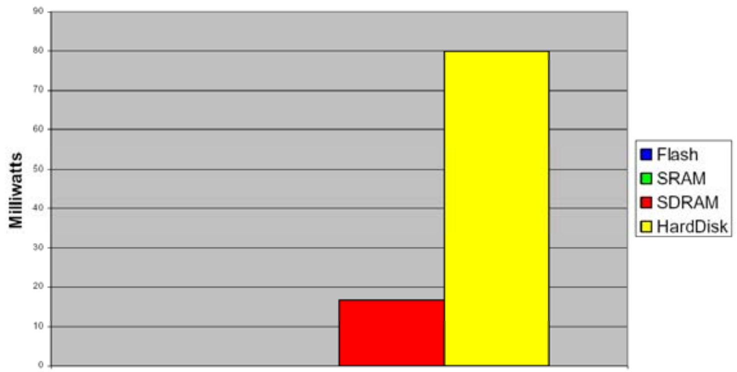


Why IPOD used hard disk



Source: MS HPL

Standby power



Here is why cell phone battery lasts longest, PDA shorter and IPOD only a few hours

Source: MS HPL

Design methodology

- Design methodology: a procedure for creating an implementation from a set of requirements.
- Methodology is important in embedded computing:
  - Must design many different systems.
  - We may use same/similar components in many different designs.
  - Design time, results must be predictable.

Design methodologies

1. Understanding your methodology helps you ensure you didn't skip anything.

- Compilers, software engineering tools, computer-aided design (CAD) tools, etc., can be used to:
  - help automate methodology steps;
  - keep track of the methodology itself.

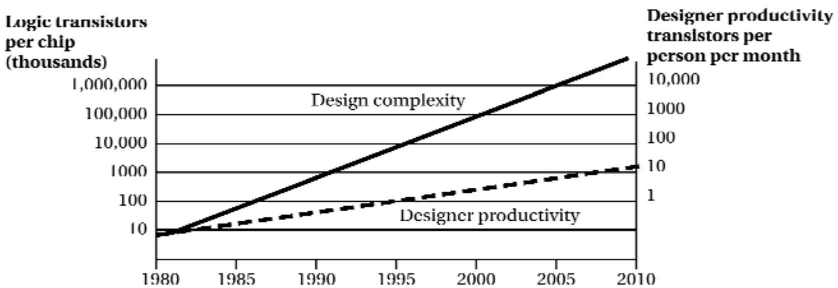
2. Members can work together more easily.

Embedded system design challenges

- Design space is large and irregular.
- We don't have synthesis tools for many steps.
- Can't simulate everything.
- May need to build special-purpose simulators quickly.
- Often need to start software development before hardware is finished.

Design complexity vs. designer productivity

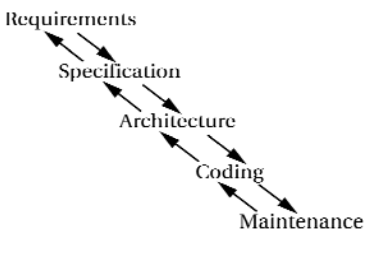
58% vs. 21%



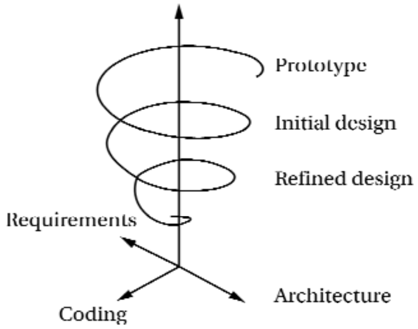
Basic design methodologies

- Figure out flow of decision-making.
- Determine when bottom-up information is generated.
- Determine when top-down decisions are made.

Waterfall and spiral models

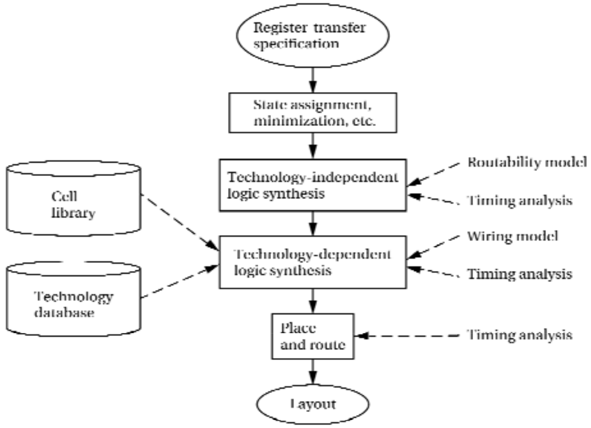


Waterfall



Spiral

Hardware design flow

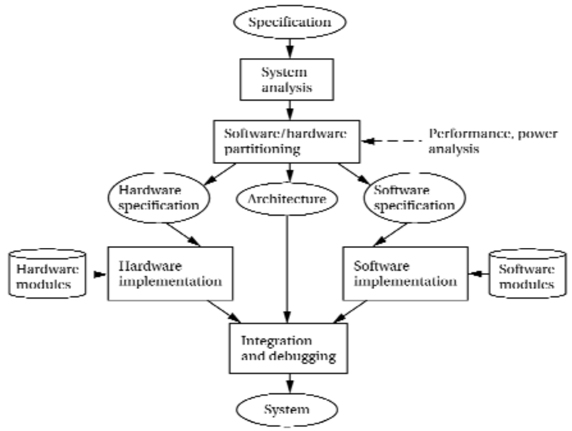


What is HW/SW Codesign?

- Traditional Design
  - SW and HW partitioning is decided at an early stage, and designs proceed separately from then onward.
- CAD today addresses synthesis problems at a purely hardware level:
  - efficient techniques for data-path and control synthesis down to silicon.
- ECS use diverse (commodity) components
  - uP, DSP cores, network and bus interfaces, etc.
- Codesign
  - A flexible design strategy, wherein the HW/SW designs proceed in parallel, with feedback and interaction occurring between the two as the design progresses.
  - Final HW/SW partition/allocation is made after evaluating trade-offs and performance of options.
  - Seek delayed (and even dynamic) partitioning capabilities.

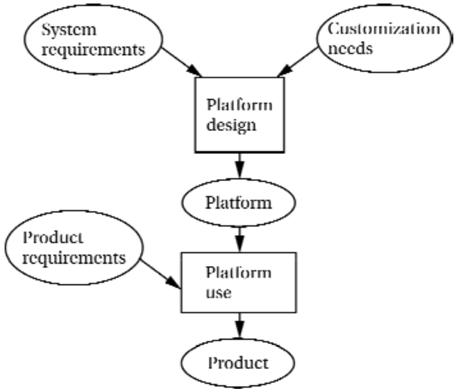
Source: RG UCSD

Hardware/software co-design flow



Platform-based design

- Platform includes hardware, supporting software.
- Two stage process:
  - Design the platform.
  - Use the platform.
- Platform can be reused to host many different systems.



Platform design

- Turn system requirements and software models into detailed requirements.
  - Use profiling and analysis tools to measure existing executable specifications.
- Explore the design space manually or automatically.
- Optimize the system architecture based on the results of simulation and other steps.
- Develop hardware abstraction layers and other software.

Programming platforms

- Programming environment must be customized to the platform:
  - Multiple CPUs.
  - Specialized memory.
  - Specialized I/O devices.
- Libraries are often used to glue together processors on platforms.
- Debugging environments are a particular challenge.

Standards-based design methodologies

- Standards enable large markets.
- Standards generally allow products to be differentiated.
  - Different implementations of operations, so long as I/O behavior is maintained.
  - User interface is often not standardized.

Reference implementations

- ⌘ Executable program that complies with the I/O behavior of the standard.
  - May be written in a variety of language.
- ⌘ In some cases, the reference implementation is the most complete description of the standard.
- ⌘ Reference implementation is often not well-suited to embedded system implementation:
  - Single process.
  - Infinite memory.
  - Non-real-time behavior.

Designing standards-based systems

- ⌘ Design and implement system components that are not part of the standard.
- ⌘ Perform platform-independent optimizations.
- ⌘ Analyze optimized version of reference implementation.
- ⌘ Design hardware platform.
- ⌘ Optimize system software based on platform.
- ⌘ Further optimize platform.
- ⌘ Test for conformity to standard.

H.264/AVC

- ⌘ Implements video coding for a wide range of applications:
  - Broadcast and videoconferencing.
  - Cell phone-sized screens to HDTV.
- ⌘ Video codec reference implementation contains 120,000 lines of C code.

Design verification and validation

- ⌘ Testing exercises an implementation by supplying inputs and testing outputs.
- ⌘ Validation compares the implementation to a specification or requirements.
- ⌘ Verification may be performed at any design stage; compares design at one level of abstraction to another.

A methodology of methodologies

- Embedded systems include both hardware and software.
  - HW, SW have their own design methodologies.
- Embedded system methodologies control the overall process, HW/SW integration, etc.
  - Must take into account the good and bad points of hardware and software design methodologies used.

Useful methodologies

- Software performance analysis.
- Architectural optimization.
- Hardware/software co-design.
- Network design.
- Software verification.
- Software tool generation.

Models of Computation

- Structural models.
- Finite-state machines.
- Turing machines.
- Petri nets.
- Control flow graphs.
- Data flow models.
- Task graphs.
- Control flow models.