Characteristics of embedded systems

- Sophisticated functionality.
- Real-time operation.
- Low manufacturing cost.
- Low power.
- Designed to tight deadlines by small teams.

Functional complexity

- Often have to run sophisticated algorithms or multiple algorithms.
  - Cell phone, laser printer.
- Often provide sophisticated user interfaces.
  - GPS

Real-time operation

- Must finish operations by deadlines.
  - Hard real time: missing deadline causes failure.
  - Soft real time: missing deadline results in degraded performance.
- Many systems are multi-rate: must handle operations at widely varying rates.
  - Video: high rate (SD: 8-bitx720x480 @30Hz)
  - Audio: low rate (CD: 16-bit samples@44.1kHz)
  - Compression

Non-functional requirements

- Many embedded systems are mass-market items that must have low manufacturing costs.
  - Limited memory, microprocessor power, etc.
- Power consumption is critical in battery-powered devices.
  - Excessive power consumption increases system cost even in wall-powered devices.

Design teams

- Often designed by a small team of designers.
- Often must meet tight deadlines.
  - 6 month market window is common.
  - Can't miss back-to-school or Christmas window.

Why use microprocessors?

- Alternatives: field-programmable gate arrays (FPGAs), custom logic, etc.
- Microprocessors are often very efficient: can use same logic to perform many different functions.
- Microprocessors simplify the design of families of products.

The performance paradox

- Microprocessors use much more logic to implement a function than does custom logic.
- But microprocessors are often at least as fast:
  - heavily pipelined- more logic
  - large design teams – more optimized
  - aggressive VLSI technology – more expensive

Power

- Custom logic uses less power, but CPUs have advantages:
  - Modern microprocessors offer features to help control power consumption.
  - Software design techniques can help reduce power consumption.
- Heterogeneous systems: some custom logic for well-defined functions, CPUs+software for everything else.

Platforms

- Embedded computing platform: hardware architecture + associated software.
- Many platforms are multiprocessors.
- Examples:
  - Single-chip multiprocessors for cell phone baseband.
  - Automotive network + processors.

The physics of software

- Computing is a physical act.
  - Software doesn't do anything without hardware.
- Executing software consumes energy, requires time.
- To understand the dynamics of software (time, energy), we need to characterize the platform on which the software runs.

What does “performance” mean?

- In general-purpose computing, performance often means average-case, may not be well-defined.
- In real-time systems, performance means meeting deadlines.
  - Missing the deadline by even a little is bad.
  - Finishing ahead of the deadline may not help.

Challenges in embedded system design

- How much hardware do we need?
  - How big is the CPU? How many?
  - Memory?
- How do we meet our deadlines?
  - Faster hardware or cleverer software?
- How do we minimize power?
  - Turn off unnecessary logic? Reduce memory accesses?

Challenges in embedded system design

- How do we design for upgradability?
  - by changing software
  - use the same HW platform for several product generations and for several different versions of products
- Does it really work?
  - Is the specification correct?
  - Does the implementation meet the spec?
  - How do we test for real-time characteristics?
  - How do we test on real data?

Challenges, etc.

- Why embedded system design is more difficult?
  - Complex testing: must run a real machine to generate the proper data. We cannot separate the testing of an embedded computer form the machine in which it is embedded.
  - Limited observability and controllability
  - Development tools are more limited

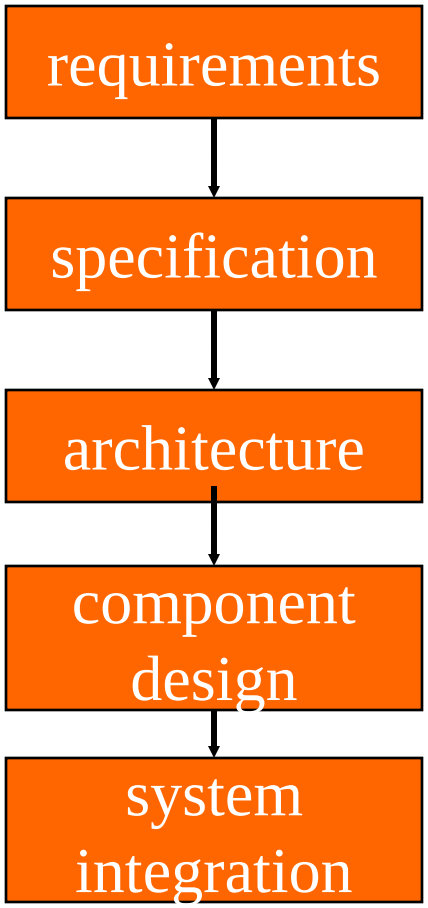
Characterizing performance

- Meeting the deadline is important
- We need to analyze the system at several levels of abstraction to understand performance:
  - CPU: a pipelined processor with caches.
  - Platform: bus and I/O
  - Program: only a part of it in the cache.
  - Task: several programs run in CPU.
  - Multiprocessor: inter-process communication

Design methodologies

- A procedure for designing a system.
- 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.

Levels of abstraction



informal

formal, more precise

how to implement function, interconnected components