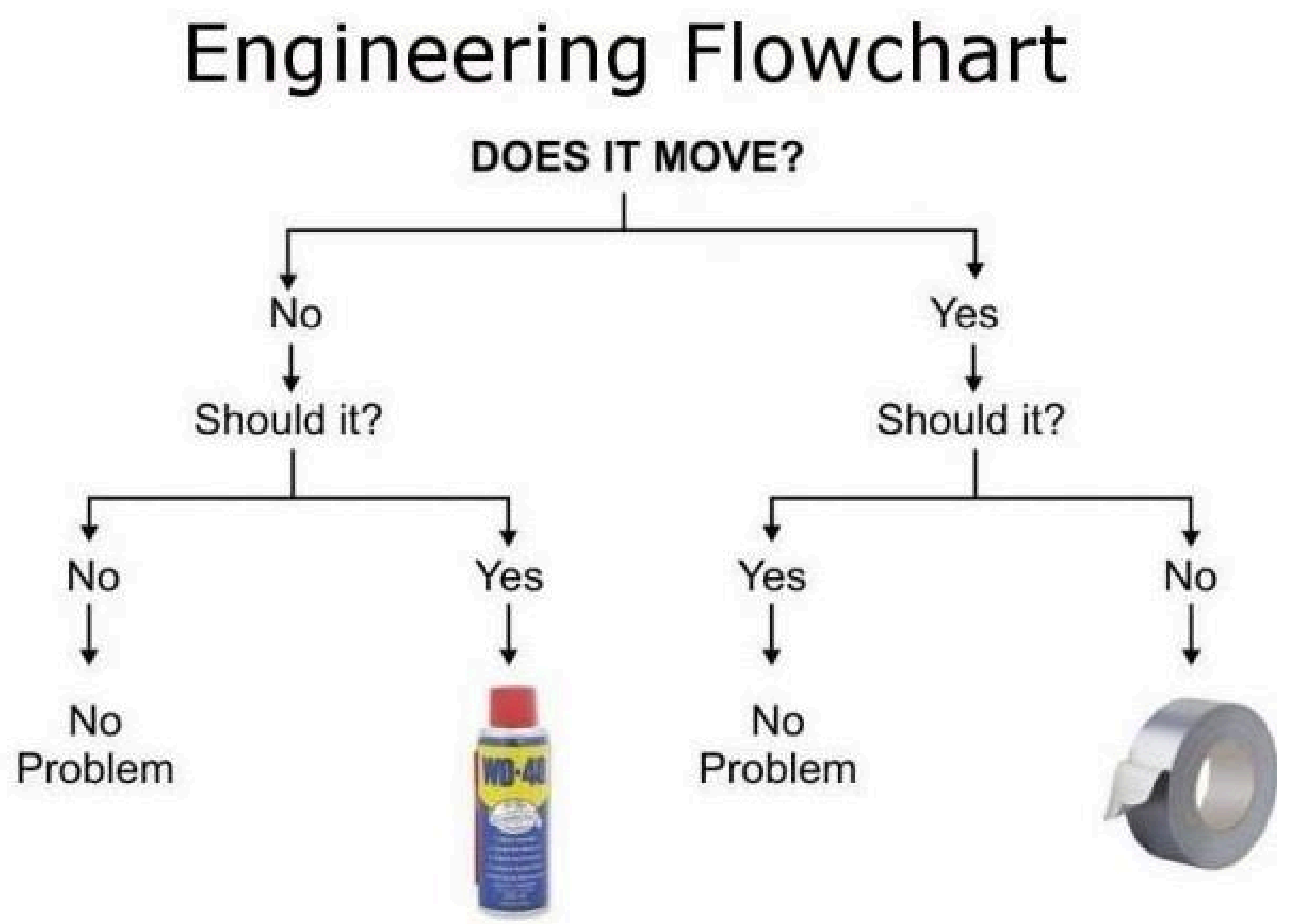
The Embedded System Design Process

WolfText - Chapter 1.3

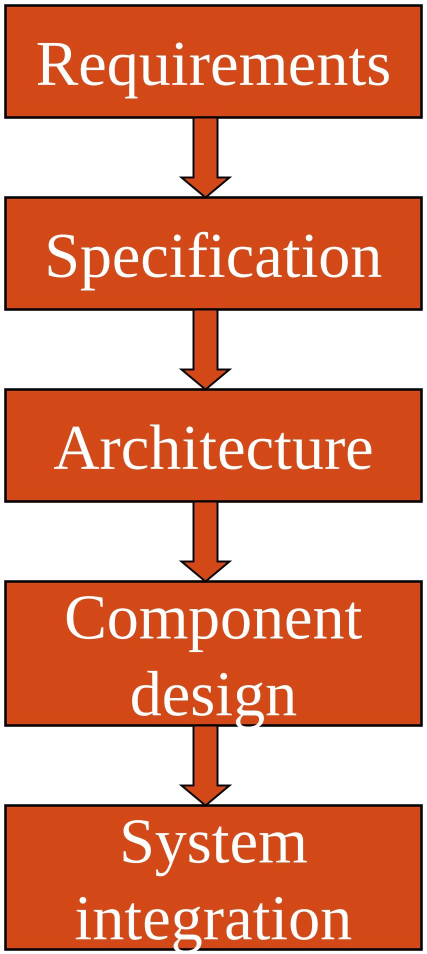
Design methodologies

- 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.

Design *methodologies* for complex embedded systems?



Levels of design abstraction



What does the customer want?

System functions/characteristics

Block diagram (HW vs. SW)

HW & SW module detailed design

Working system

Top-down vs. bottom-up

- Top-down design:
  - start from most abstract description;
  - work to most detailed.
- Bottom-up design:
  - work from small components to big system.
- **Real design often uses both techniques.**

Stepwise refinement

- At each level of abstraction, we must:
  - analyze the design to determine characteristics of the current state of the design;
  - refine the design to add detail.

Embedded system design constraints

- **Cost**
  - Competitive markets penalize products which don't deliver adequate value for the cost
- **Performance**
  - Perform required operations (throughput)
  - Meet real-time deadlines (latency)
- **Size and weight limits**
  - Mobile (aviation, automotive) and portable (e.g. handheld) systems
- **Power and energy limits**
  - Battery capacity
  - Cooling limits
- **Environment**
  - Temperatures may range from -40°C to 125°C, or even more

Impact of Constraints

- **Microcontrollers/SoCs** (rather than microprocessors)
  - Include peripherals to interface with other devices, respond efficiently
  - On-chip RAM, ROM reduce circuit board complexity and cost
- **Programming language**
  - Programmed in C rather than Java (smaller and faster code, so less expensive MCU)
  - Some performance-critical code may be in assembly language
  - Hierarchical design with SW libraries (math, I/O drivers, etc.)
- **Operating system**
  - Small system: typically no OS, but instead simple scheduler (or even just interrupts + main code (foreground/background system)
  - Complex system: If OS is used, likely to be a lean RTOS

Project Cost

- Total cost of a project involves **non-recurring** engineering (NRE), cost plus **recurring** (RE) cost, and number of units produced (K)

Project Cost = NRE + K*RE

- NRE includes design time, tools, facilities
- RE includes components, manufacturing, testing, and maintenance

What does “performance” actually mean?

- In general-purpose computing, performance often means average-case, may not be well-defined.
- In real-time systems, performance means meeting deadlines.
- Some systems require high throughput/bandwidth
- We need to analyze the system at several levels of abstraction to understand performance:
  - CPU.
  - Platform.
  - Multiprocessor.
  - Program.
  - Task.

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.
- A **real-time operating system** (RTOS) can manage scheduling of operations to satisfy critical timing constraints

The performance paradox

- Microprocessors generally use more logic circuits to implement a function than do custom logic circuits.
- But are microprocessors as fast as custom circuits?
  - aggressive VLSI technology;
  - heavily pipelined;
  - smart compilers;
  - re-use and improve efficient SW routines.

Execution Time = NI x CPI x Tclk

(#instructions) x (#clocks/instruction) x (clock period)

Power considerations

- Custom logic typical in low power devices.
- Modern microprocessors offer features to help control power consumption.
  - Turn off unnecessary logic/modules
  - Reduce memory accesses
  - Reduce external communication
  - Reduce clock rates (CMOS)
  - Provide "sleep modes"
  - Low-power electronic circuit design methods
- Software design techniques can also help reduce power consumption.

Safe, secure systems

- **Security**: system's ability to prevent malicious attacks.
  - Traditional security is oriented to IT and data security.
  - Insecure embedded computers can create unsafe cyber-physical systems.
  - Internet of Things presents special security challenges!
- **Safety**: no crashes, accidents, harmful releases of energy, etc.
  - We need to combine safety and security:
    - Identify security breaches that compromise safety.
  - Safety and security can't be bolted on---they must be **baked in**.
- **Integrity**: maintenance of proper data values.
- **Privacy**: no unauthorized releases of data.

Product development time

- Often designed by a small team of designers.
- Often constrained by tight deadlines.
  - 6 month market window is common.
  - Optimal sales windows (ex. calculators for back-to-school)
  - Optimal sales window for holiday "gadgets"
  - Longer lead times for control systems (automotive, aerospace, process control, etc.)
- **Hardware-software co-design** can shorten design cycle

Requirements

- Plain language description of what the user wants and expects to get.
- May be developed in several ways:
  - talking directly to customers;
  - talking to marketing representatives;
  - providing prototypes to users for comment.

Functional vs. non-functional requirements

- Functional requirements:
  - output as a function of input.
- Non-functional requirements:
  - time required to compute output;
  - size, weight, etc.;
  - power consumption (battery-powered?);
  - reliability;
  - low HW costs (CPU, memory) for mass production
  - etc.

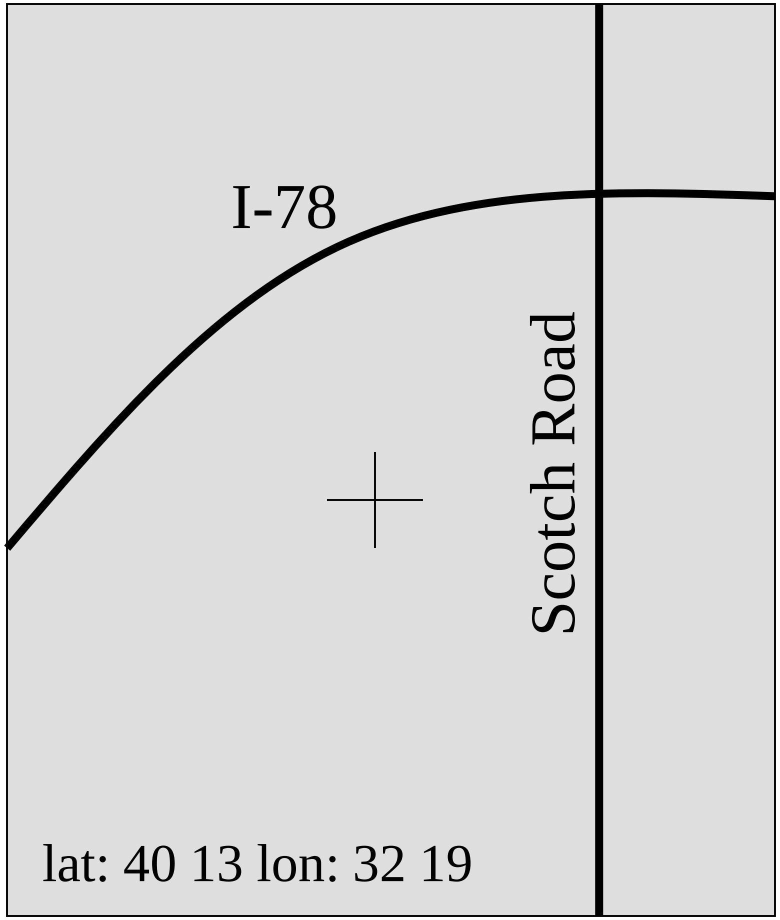
Sample requirements form

Use form to assist “interviewing” the customer.

name
purpose
inputs
outputs
functions
performance
manufacturing cost
power
physical size/weight

Example: GPS moving map

- Moving map obtains position from GPS, paints map from local database.



GPS moving map requirements

- **Functionality:** For automotive use. Show major roads and landmarks.
- **User interface:** At least 400 x 600 pixel screen. Three buttons max. Pop-up menu.
- **Performance:** Map should scroll smoothly. No more than 1 sec power-up. Lock onto GPS within 15 seconds.
- **Cost:** $200 street price.
- **Physical size/weight:** Should fit in dashboard.
- **Power consumption:** Current draw comparable to CD player.

GPS moving map requirements form

name	GPS moving map
purpose	consumer-grade moving map for driving
inputs	power button, two control buttons
outputs	back-lit LCD 400 X 600
functions	5-receiver GPS; three resolutions; displays current lat/lon
performance	updates screen within 0.25 sec of movement
manufacturing cost	$100 cost-of-goods-sold
power	100 mW
physical size/weight	no more than 2” X 6”, 12 oz.

Specification

- A more precise description of the system:
  - "What will the system do?" (functions, data, etc.)
  - should not imply a particular architecture;
  - provides input to the architecture design process.
- May include functional and non-functional elements.
- May be "executable" or may be in mathematical form for proofs.
- Often developed with tools, such as UML

"Contract" between customer & architects

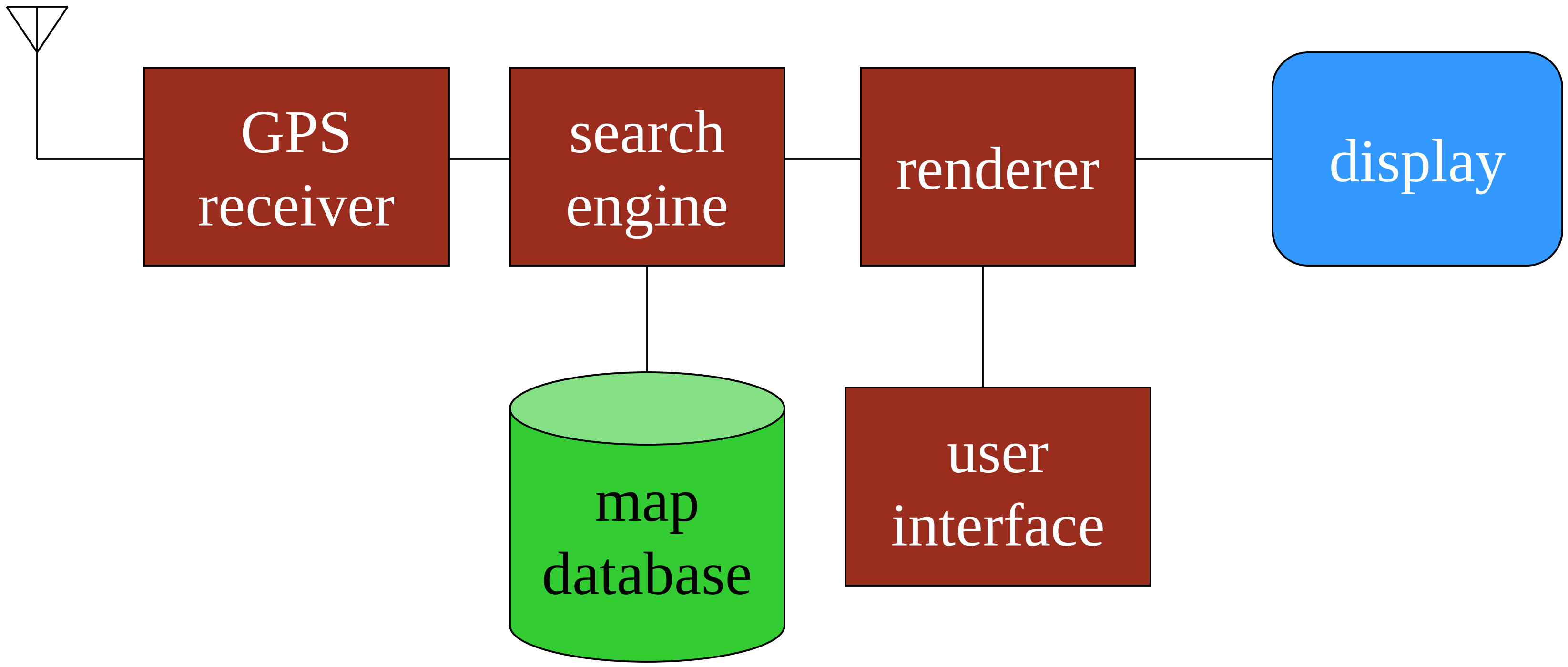
GPS moving map specification

- Should include:
  - what is received from GPS (format, rate, ...);
  - map data;
  - user interface;
  - operations required to satisfy user requests;
  - background operations needed to keep the system running.

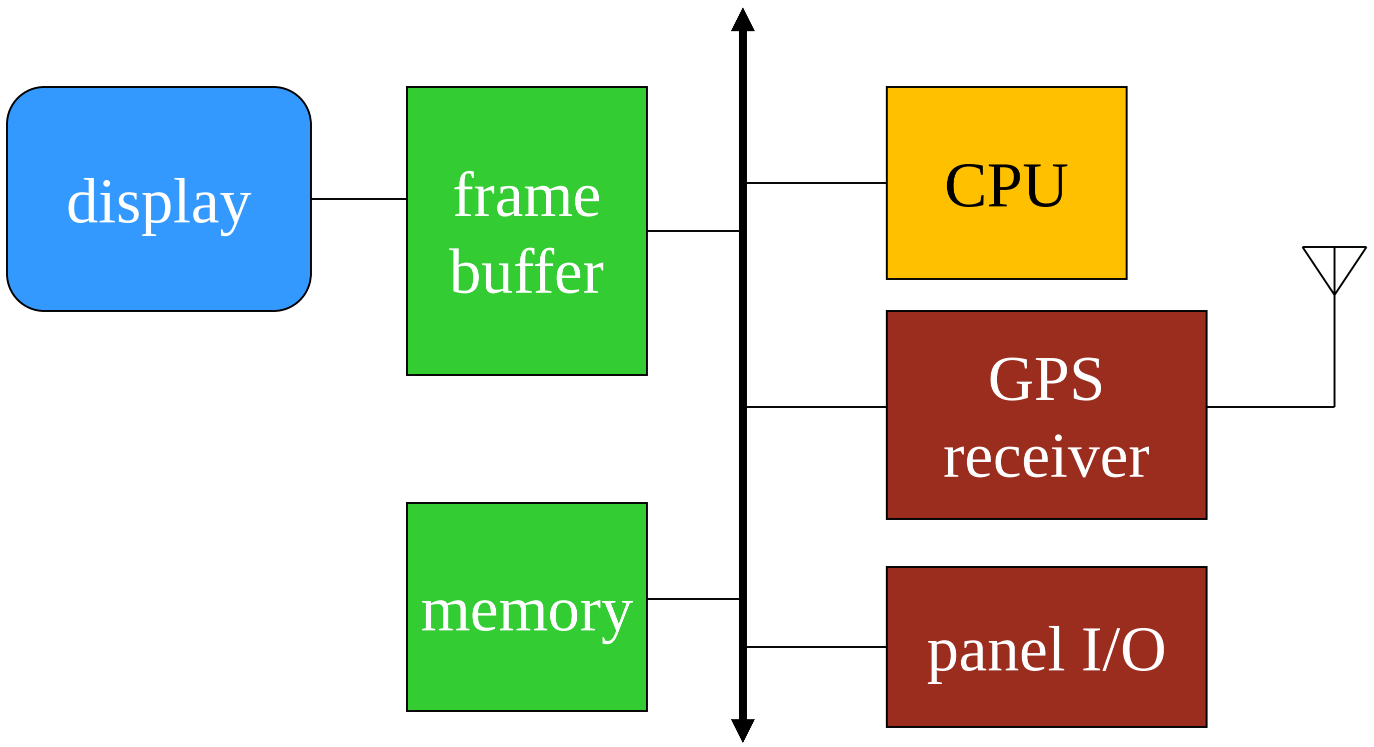
Architecture design

- What major components go to satisfying the specification?
- Hardware components:
  - CPUs, peripherals, etc.
- Software components:
  - major programs and their operations.
  - major data structures
- Evaluate hardware vs. software tradeoffs
- Must take into account functional and non-functional specifications.

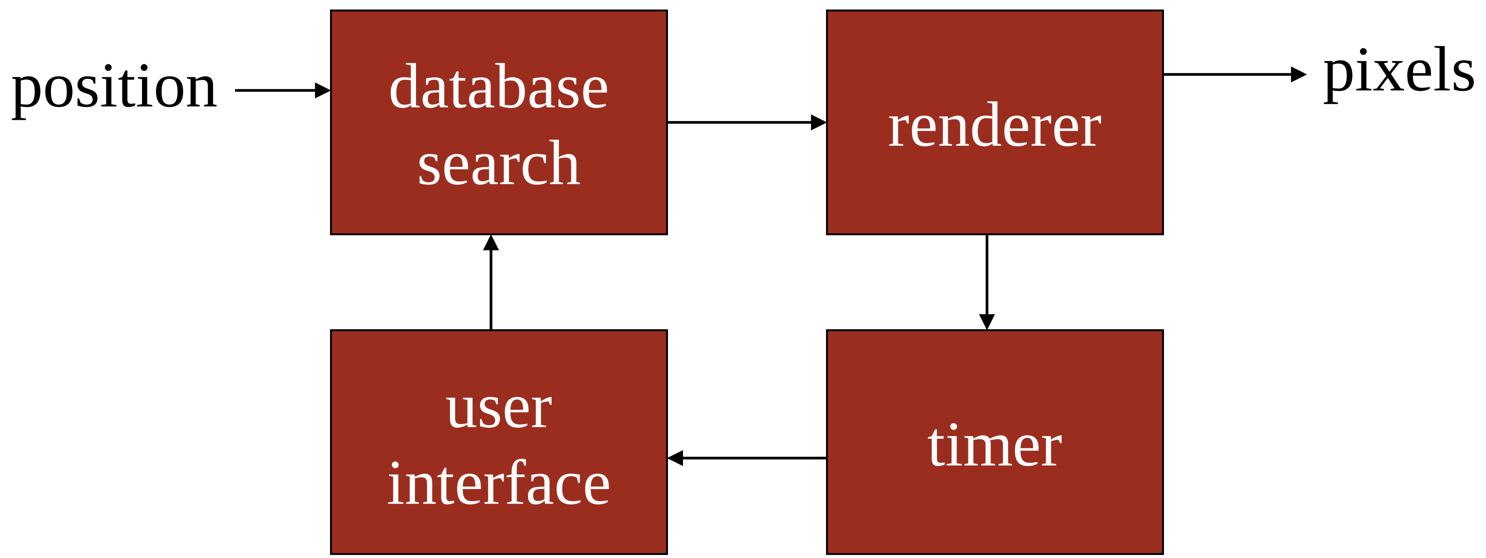
GPS moving map block diagram



GPS moving map hardware architecture



GPS moving map software architecture



Designing hardware and software components

- Must spend time architecting the system before you start coding or designing circuits.
- Some components are ready-made, some can be modified from existing designs, others must be designed from scratch.

System integration

- Put together the components.
  - Many bugs appear only at this stage.
  - Interfaces must be well designed
- Have a plan for integrating components to uncover bugs quickly, test as much functionality as early as possible.
  - **Test to each specification**

Challenges, etc.

- 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?
- How do we work on the system?
  - Observability, controllability?
  - What is our development platform?

Challenges in embedded system design

- How much hardware do we need?
  - CPU computing power? Memory?
  - What peripheral functions?
    - Implement in HW or SW?
- How do we meet timing constraints?
  - Faster hardware or cleverer software?
  - Real-time operating system or custom design?
- How do we minimize power consumption?
- How do we optimize cost?
- How do we ensure system security/reliability?
- How do we meet our time-to-market deadline?

Summary

- Embedded systems are all around us.
- Chip designers are now system designers.
  - Must deal with hardware and software.
- Today's applications are complex.
  - Reference implementations must be optimized, extended.
- Platforms present challenges for:
  - Hardware designers---characterization, optimization.
  - Software designers---performance/power evaluation, debugging.
- Design methodologies help us manage the design process and complexity.