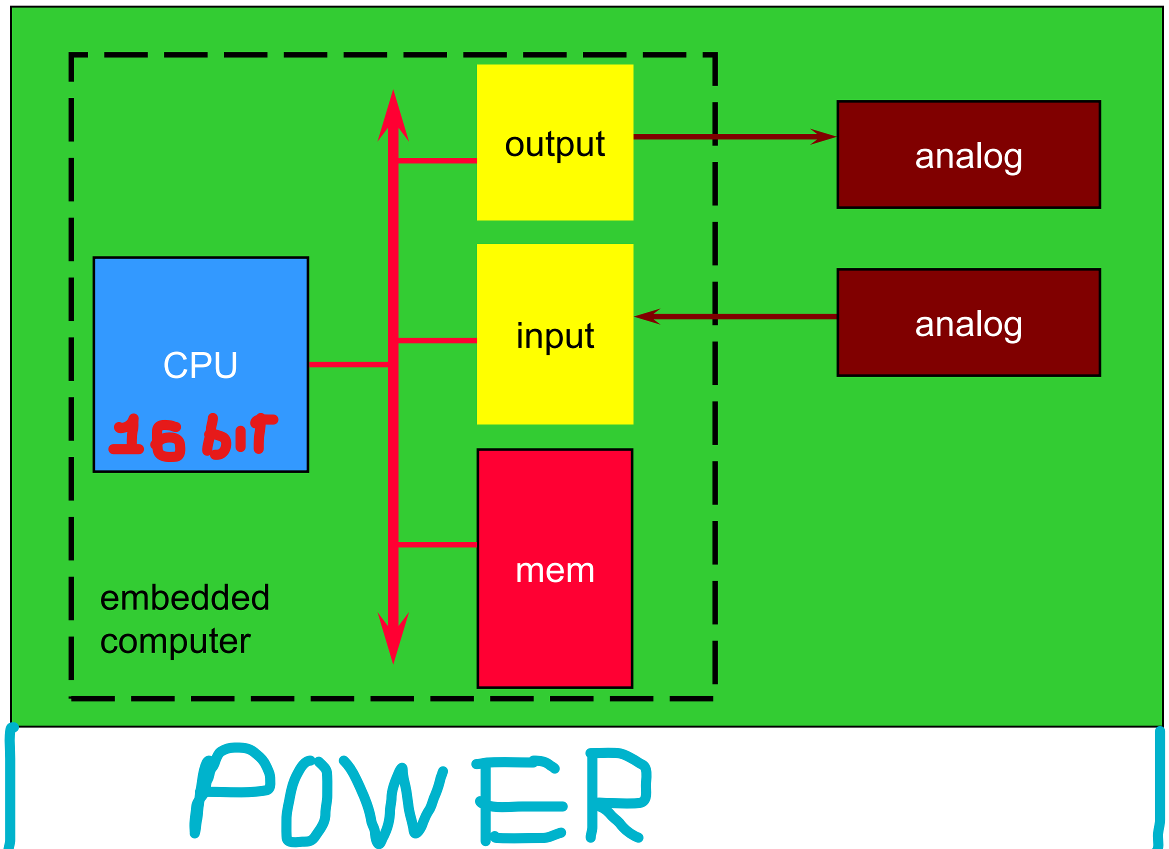
Introduction

- What are embedded computing systems?
- Challenges in embedded computing system design.
- Design methodologies.

Definition

- Embedded computing system: any device that includes a programmable computer but is not itself a general-purpose computer.
- Take advantage of application characteristics to optimize the design:
  - don’t need all the general-purpose bells and whistles.

Embedding a computer



Examples

- Cell phone.
- Printer.
- Automobile: engine, brakes, dash, etc.
- Airplane: engine, flight controls, nav/comm.
- Digital television.
- Household appliances.

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.
- HP-35 calculator used several chips to implement a microprocessor in 1972.

Early history, cont'd.

- Automobiles used microprocessor-based engine controllers starting in 1980.
  - Control fuel/air mixture, engine timing, etc.
  - Multiple modes of operation: warm-up, cruise, hill climbing, etc.
  - Provides lower emissions, better fuel efficiency.

Microprocessor varieties

- Microcontroller: includes I/O devices, on-board memory.
- Digital signal processor (DSP): microprocessor optimized for digital signal processing.
- Typical embedded word sizes: 8-bit, 16-bit, 32-bit.

Application examples

- Simple control: front panel of microwave oven, etc.
- Canon EOS 3 has three microprocessors.
  - 32-bit RISC CPU runs autofocus and eye control systems.
- Digital TV: programmable CPUs + hardwired logic.

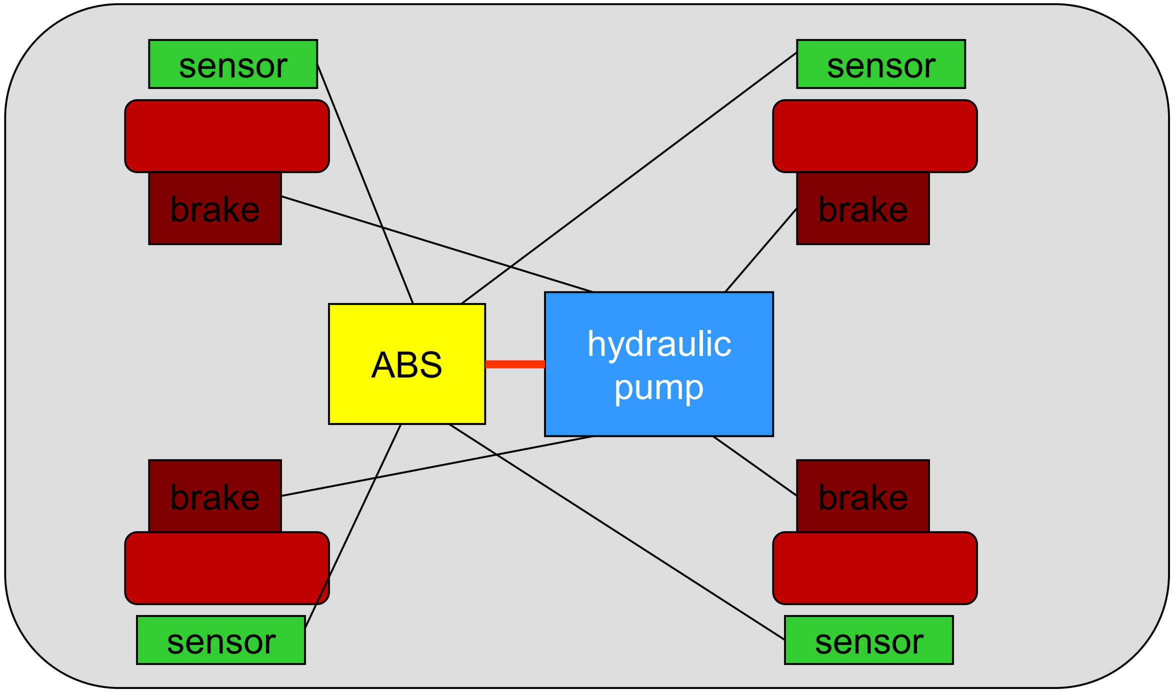
Automotive embedded systems

- Today's high-end automobile may have 100 microprocessors:
  - 4-bit microcontroller checks seat belt;
  - microcontrollers run dashboard devices;
  - 16/32-bit microprocessor controls engine.
- Low-end cars use 20+ microprocessors.

BMW 850i brake and stability control system

- Anti-lock brake system (ABS): pumps brakes to reduce skidding.
- Automatic stability control (ASC+T): controls engine to improve stability.
- ABS and ASC+T communicate.
  - ABS was introduced first---needed to interface to existing ABS module.

BMW 850i, cont'd.



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.

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.

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 window for calculator.

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;
  - large design teams;
  - aggressive VLSI technology.

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.

Cyber-physical systems

- A physical system that tightly interacts with a computer system.
- Computers replace mechanical controllers:
  - More accurate.
  - More sophisticated control.
- Engine controllers replace distributor, carburetor, etc.
  - Complex algorithms allow both greater fuel efficiency and lower emissions.

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.

Characterizing performance

- We need to analyze the system at several levels of abstraction to understand performance:
  - CPU.
  - Platform.
  - Program.
  - Task.
  - Multiprocessor.

Security, safety

- Security: system's ability to prevent malicious attacks.
- Integrity: maintenance of proper data values.
- Privacy: no unauthorized releases of data.
- Safety: no harmful releases of energy.
  - No crashes, accidents, etc.

Safe, secure systems

- Traditional security is oriented to IT and data security.
- But insecure embedded computers can create unsafe cyber-physical systems.
- 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.

Challenges in embedded system design

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

Cryptography

- Cryptography is the science of hiding information.
- Secret-key cryptography allows messages to be encoded and decoded.
    - Key must be kept secret or the message will not be secure.
- Public-key cryptography gives secret messages with an easier-to-use approach:
    - Sender uses secret key to encrypt message. Also provides a public key to others.
    - Receiver can decrypt message using public key. The sender's identity is established by the ability to use their public key.

Cryptography, cont'd.

- Cryptographic hash function generates a message digest.
  - Short version of the message.
  - Two different messages are unlikely to generate the same key.
- Digital signature:
  - Sender signs the message or message digest using private key.
  - Receiver decrypts with public key.
- Digital signature plus encryption:
  - Sender signs the message or message digest with private key.
  - Sender encrypts the message with the receiver's public key.
  - Receiver decrypts with private key, then verifies signature using sender's public key.

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?

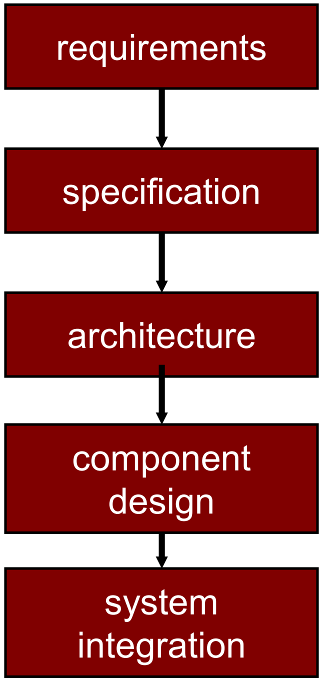
Design methodologies

- 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 goals

- Performance.
  - Overall speed, deadlines.
- Functionality and user interface.
- Manufacturing cost.
- Power consumption.
- Other requirements (physical size, etc.)

Levels of abstraction



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 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.

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

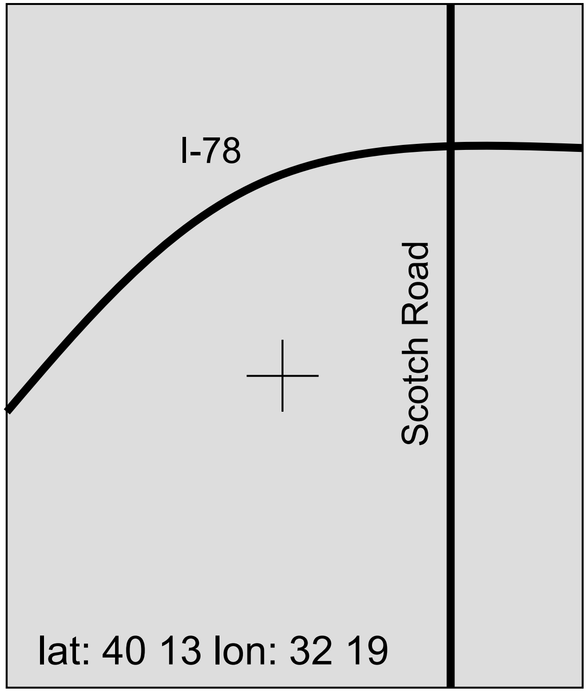
- Functional requirements:
  - output as a function of input.
- Non-functional requirements:
  - time required to compute output;
  - size, weight, etc.;
  - power consumption;
  - reliability;
  - etc.

Our requirements form

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

Example: GPS moving map requirements

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



GPS moving map needs

- 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: $120 street price = approx. $30 cost of goods sold.

GPS moving map needs, cont'd.

- Physical size/weight: Should fit in hand.
- Power consumption: Should run for 8 hours on four AA batteries.

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:
  - 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.

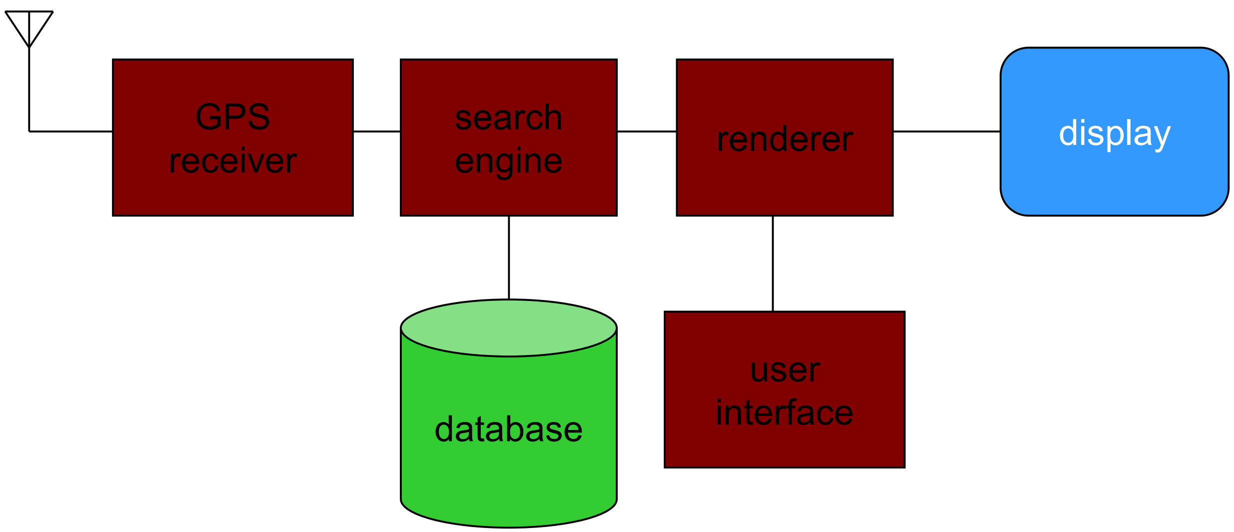
GPS specification

- Should include:
  - What is received from GPS;
  - map data;
  - user interface;
  - operations required to satisfy user requests;
  - background operations needed to keep the system running.

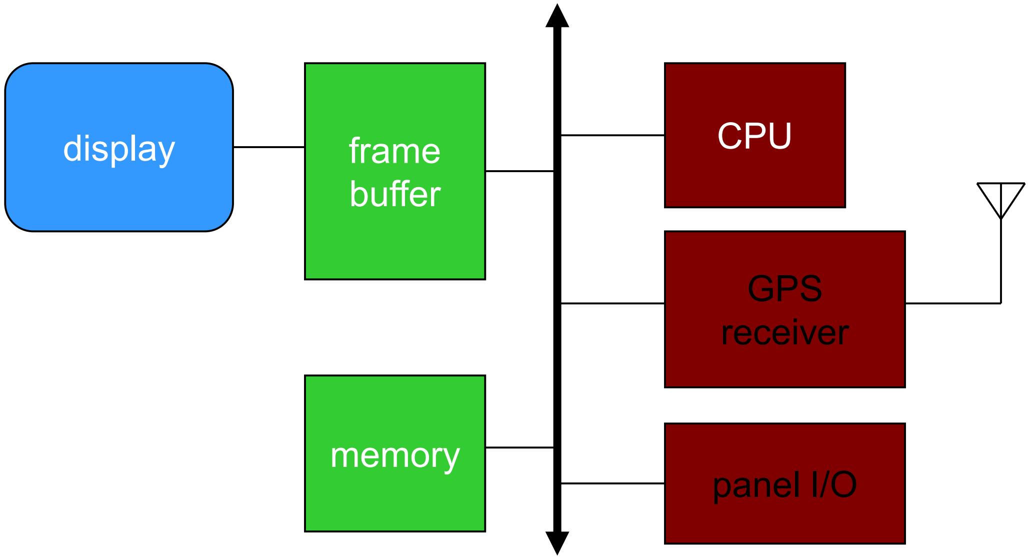
Architecture design

- What major components go satisfying the specification?
- Hardware components:
  - CPUs, peripherals, etc.
- Software components:
  - major programs and their operations.
- Must take into account functional and non-functional specifications.

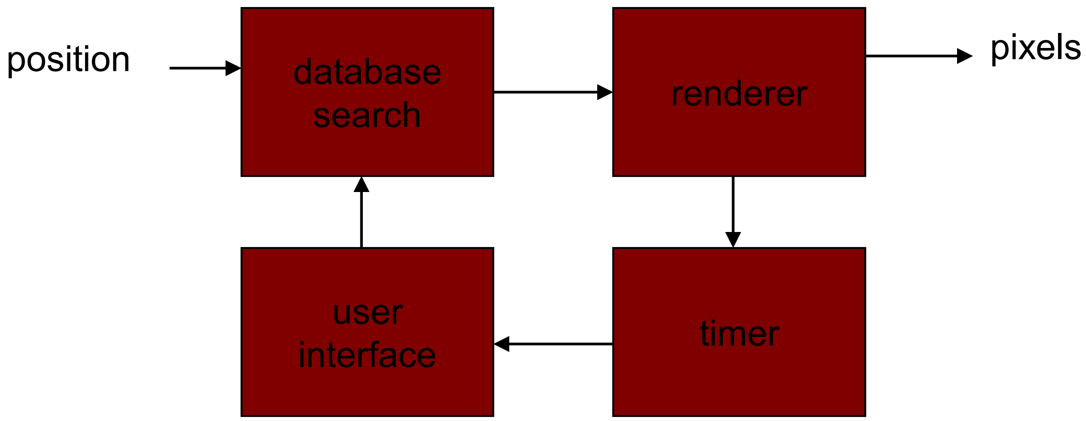
GPS moving map block diagram



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.
- 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.
- Have a plan for integrating components to uncover bugs quickly, test as much functionality as early as possible.

Summary

- Embedded computers are all around us.
  - Many systems have complex embedded hardware and software.
- Embedded systems pose many design challenges: design time, deadlines, power, etc.
- Design methodologies help us manage the design process.