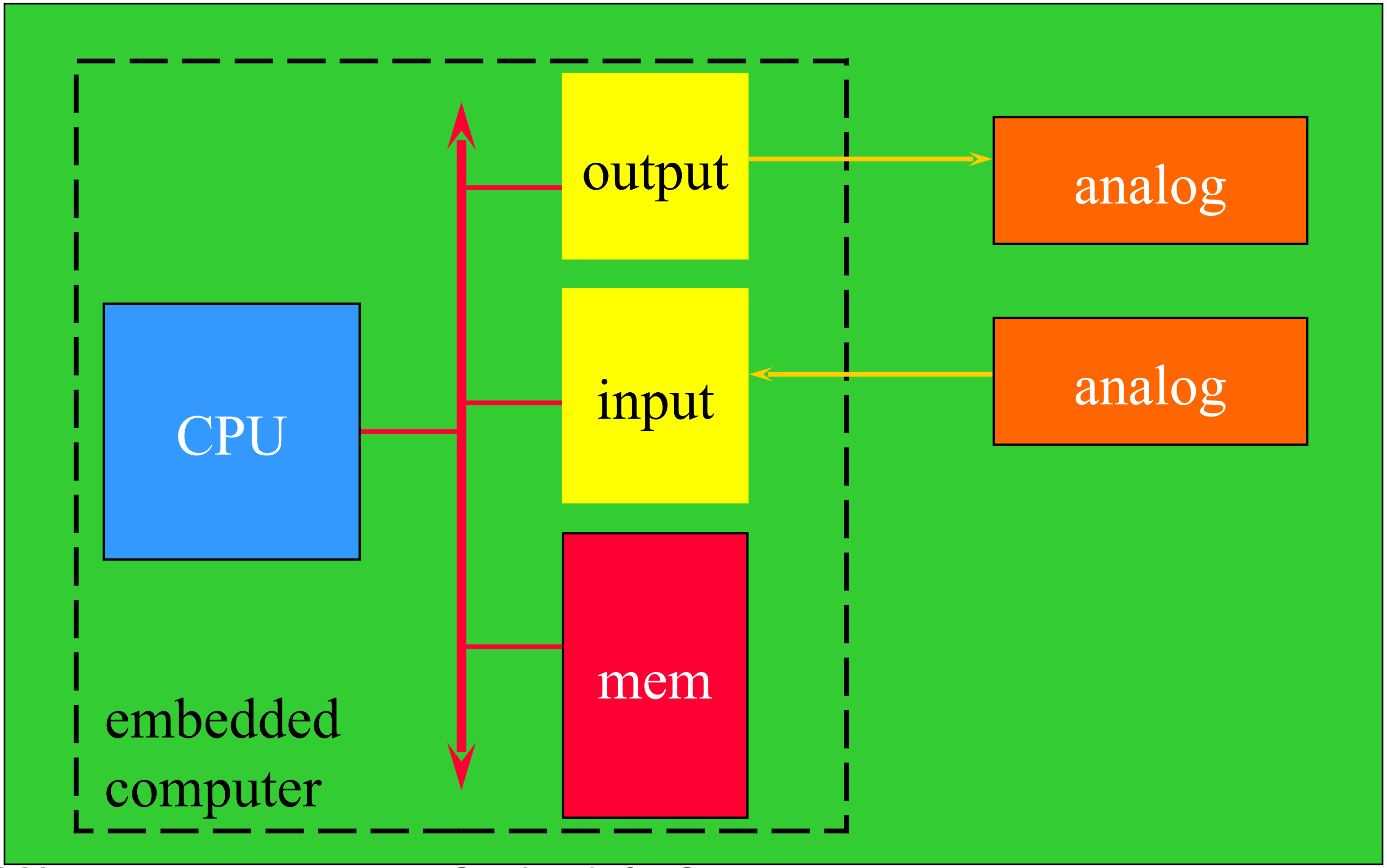
Introduction

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

Definition

- Embedded 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

- Personal digital assistant (PDA).
- Printer.
- Cell phone.
- Automobile: engine, brakes, dash, etc.
- Television.
- Household appliances.
- PC keyboard (scans keys).

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 1970's.
  - 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.
  - Analog TV: channel selection, etc.
  - Digital TV: programmable CPUs + hardwired logic.

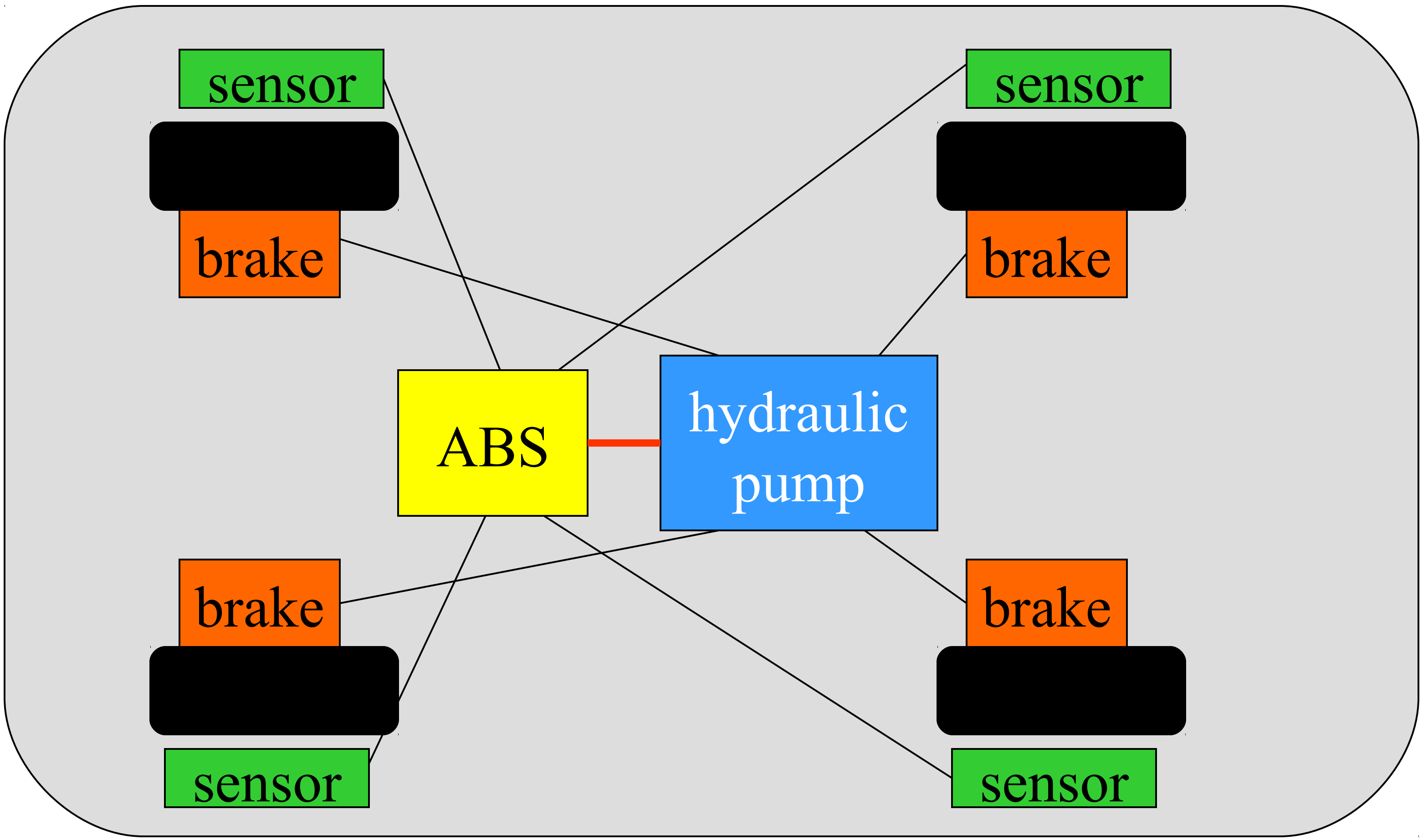
Automotive embedded systems

- 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.

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 is a clear winner for low power devices.
- Modern microprocessors offer features to help control power consumption.
- Software design techniques can help reduce power consumption.

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?

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?