What is covered in this course

- Embedded computing:
  - information systems
  - control systems
  - communication/multimedia
- Computer science vs engineering approach to building s/w

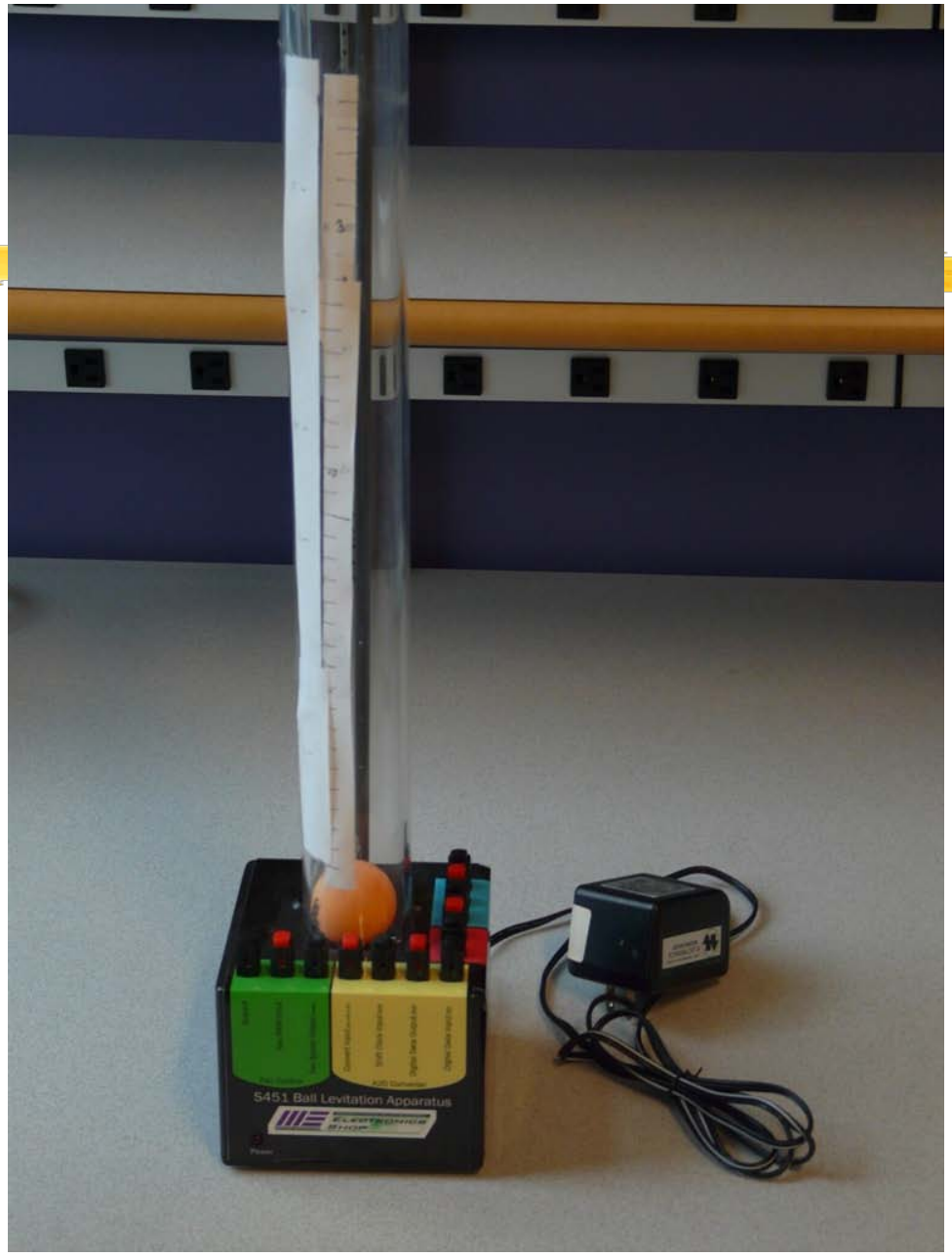
Topics

- Embedded systems design process
- Instruction sets for microprocessor architectures
- I/O devices and interfacing;
- Basic hardware and software platforms
- Embedded programming tips
- Optimizing code for execution time, power, and energy consumption

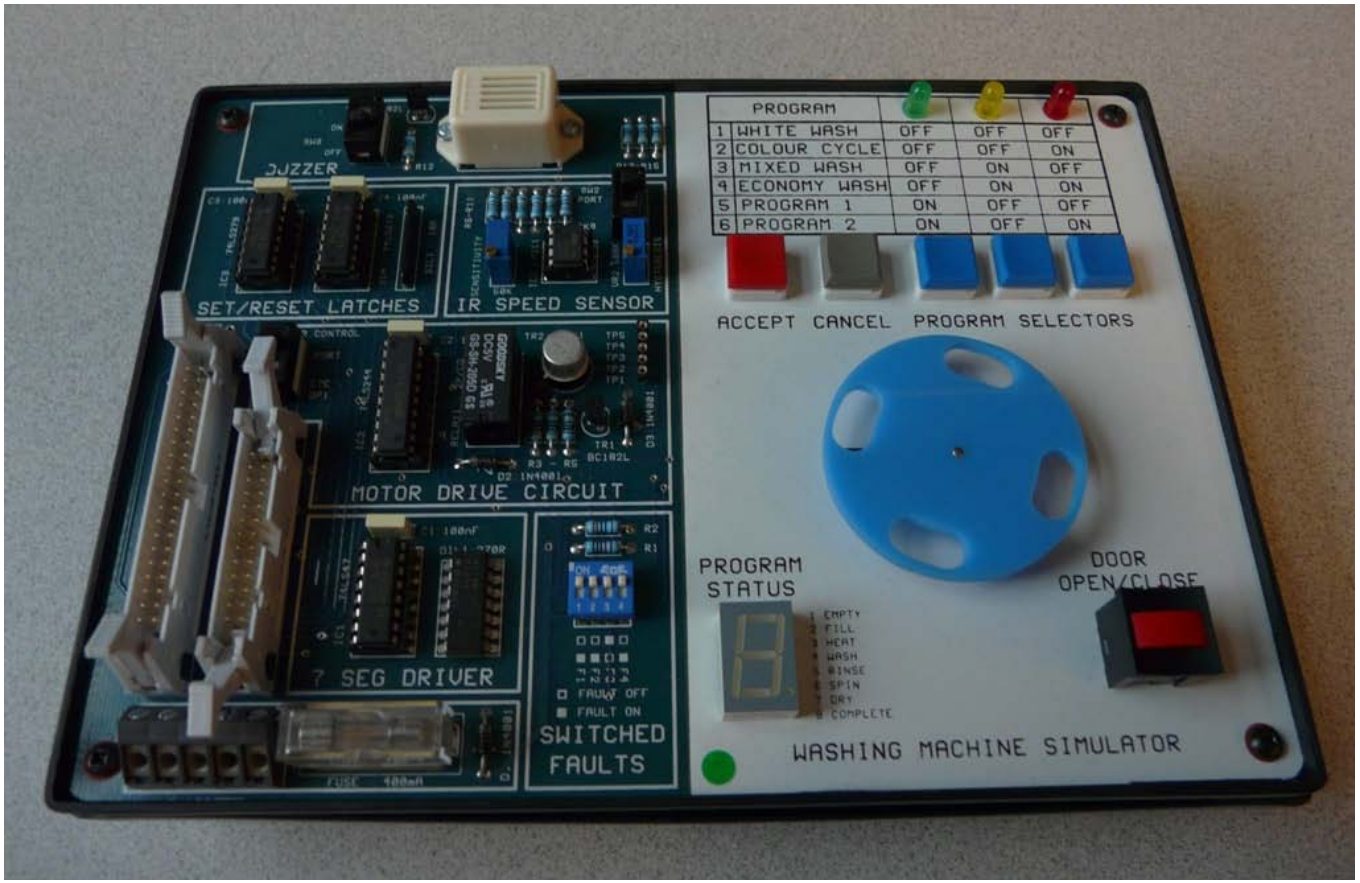
... Course structure

- **Practical issues related to computer based control systems: PID tuning, anti-aliasing filters, integrator saturation and windup, selection of sampling rates, etc.**
- **Case studies: Automobile seat-belt controller, model train controller, digital clock system, home heater controller, digital camera system, elevator controller.**
- **Projects: Team work, share experience among groups**

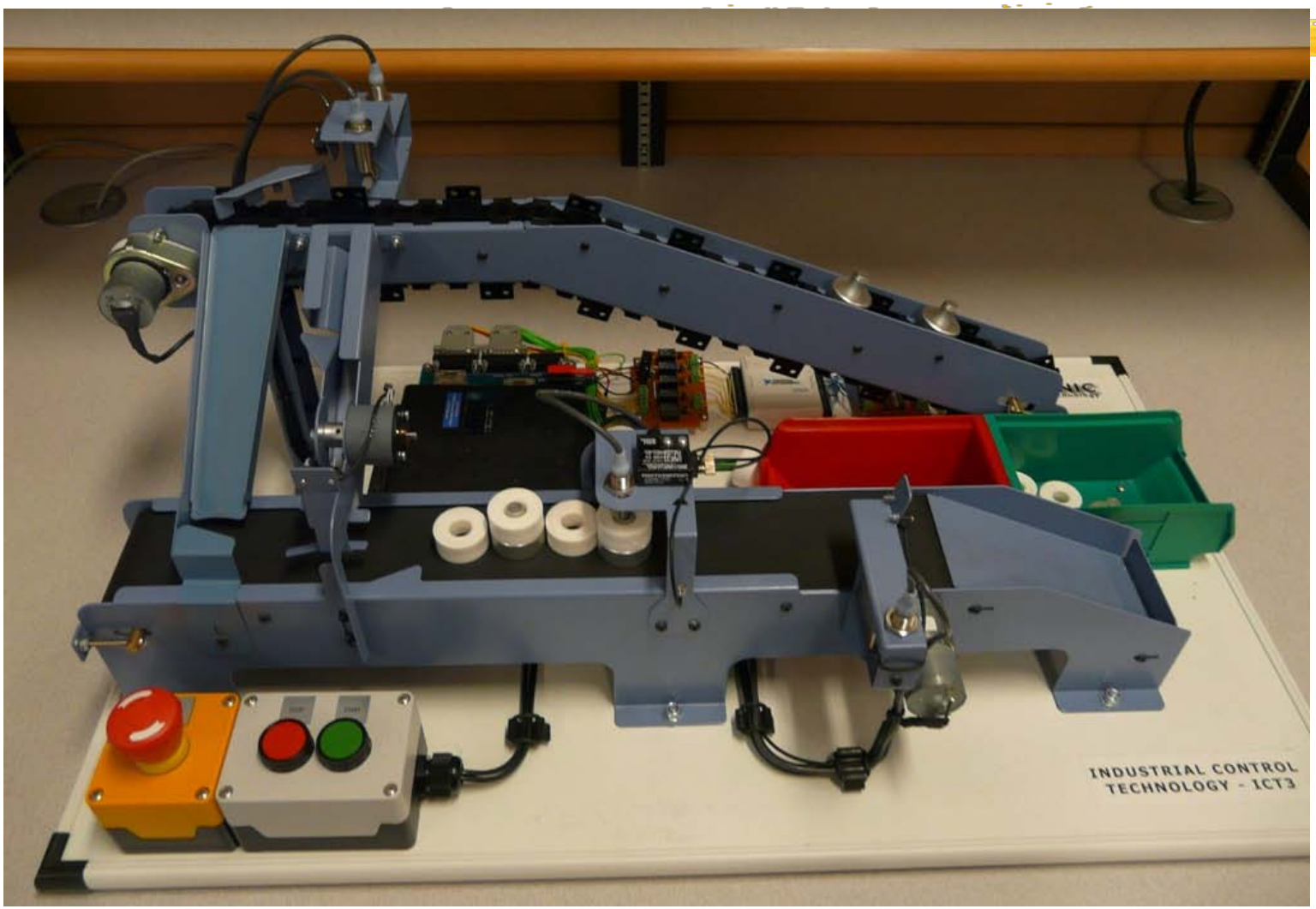
Air levitation system



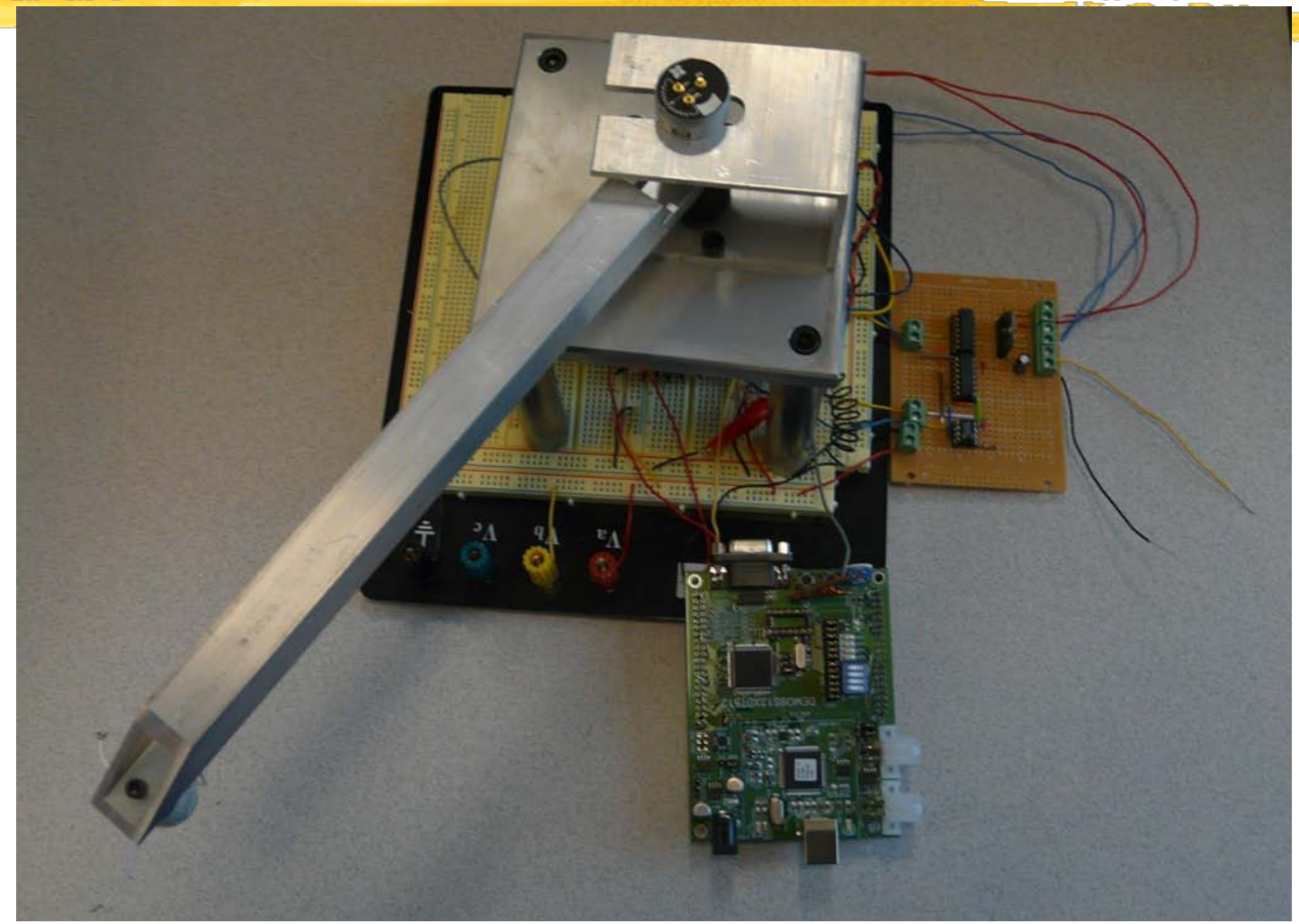
Washing machine simulator (Bytronic.com)



Industrial control trainer: Bytronic.com



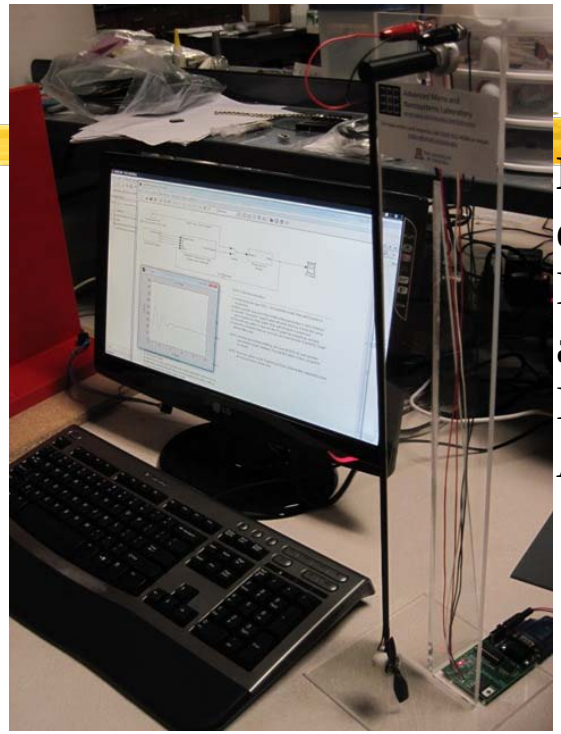
1-dof robot arm with a magnetic gripper (developed at SFU)



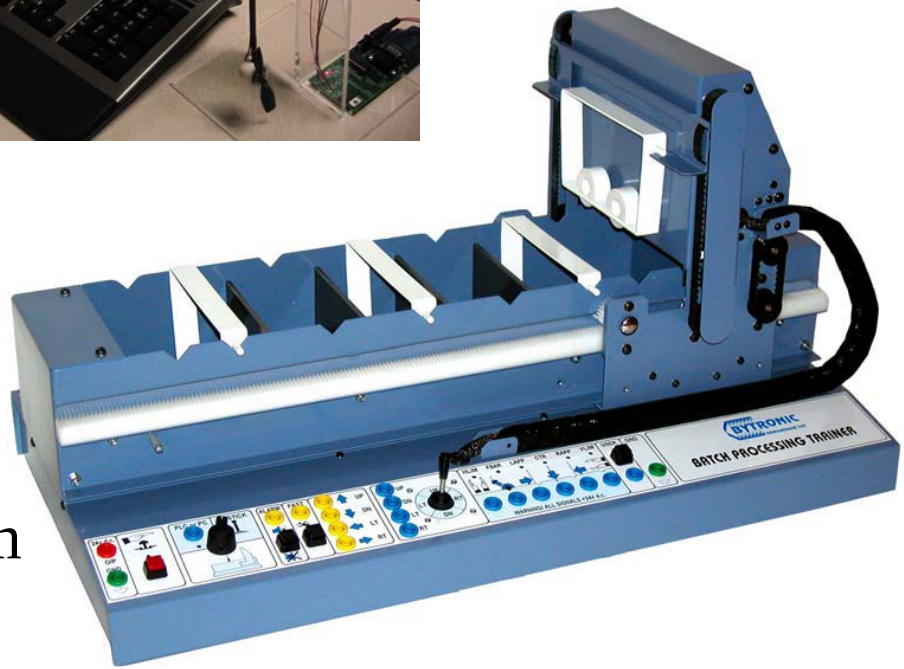
Other setups

- Motor pendulum setup
- Traffic light controller
- Batch processing unit

- ...



Motor pendulum developed by Dr. Enikov, Department of Aerospace and Mechanical Engineering, University of Arizona



Bytronic.com

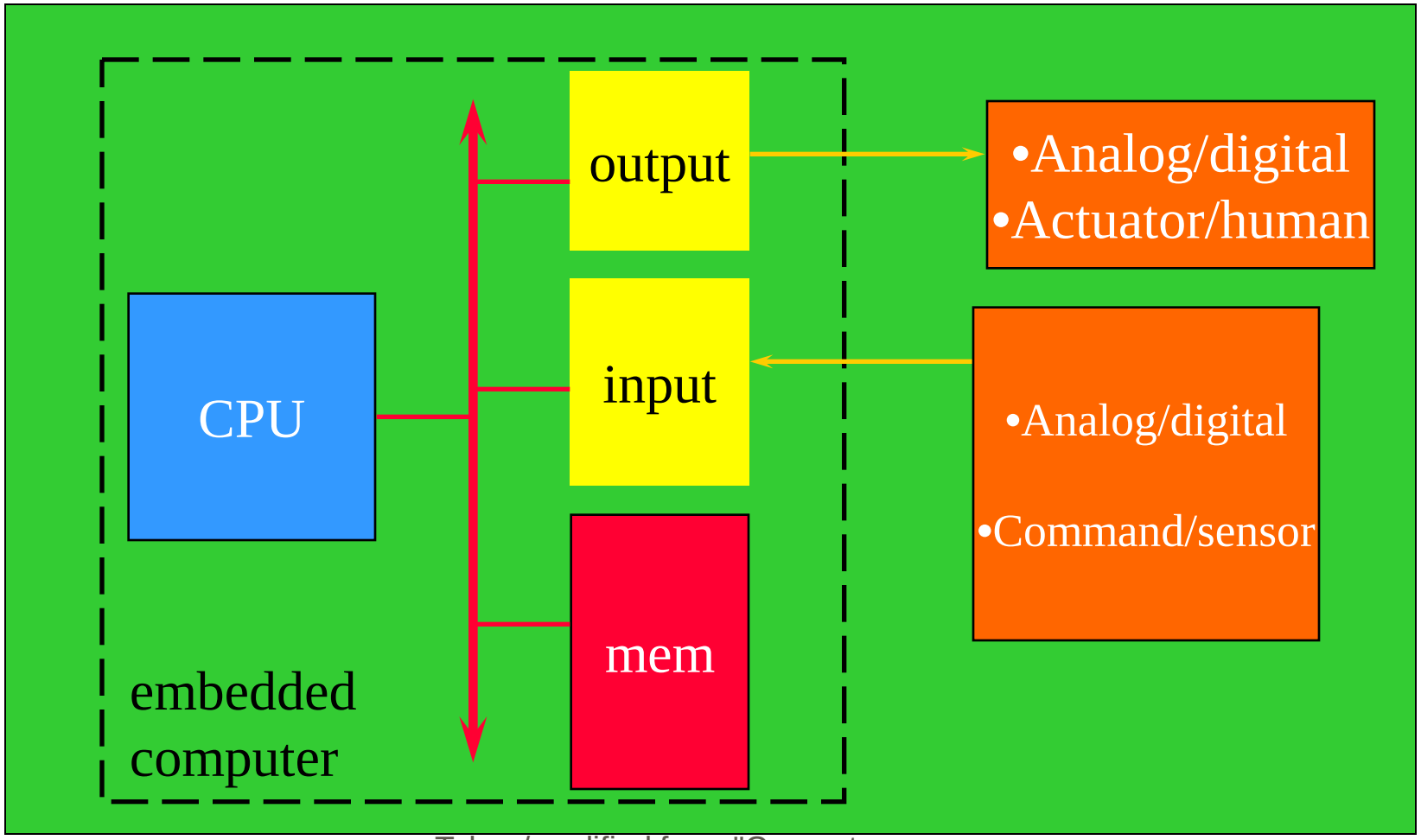
Introduction

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

Some definitions

- 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



Taken/modified from "Computers as Components ..", W. Wolf

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.
  - 4000 vacuum tubes in Whirlwind
- First microprocessor was Intel 4004 in early 1970's -> Used in a calculator

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/Microcomputer/Microcontroller

- **Microprocessor:** **CPU on a single chip**
- **Microcomputer:** **Microprocessor + I/O + Memory**
- **Microcontroller:** **includes CPU, I/O devices, on-board memory.**
- **Digital signal processor (DSP):** **microprocessor designed for digital signal processing.**
- **Typical embedded word sizes: 8-bit, 16-bit, 32-bit.**

Taken/modified from "Computers as Components ..", W. Wolf

Application examples

- Simple control: front panel of microwave oven, etc.
- Canon EOS Camera 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.

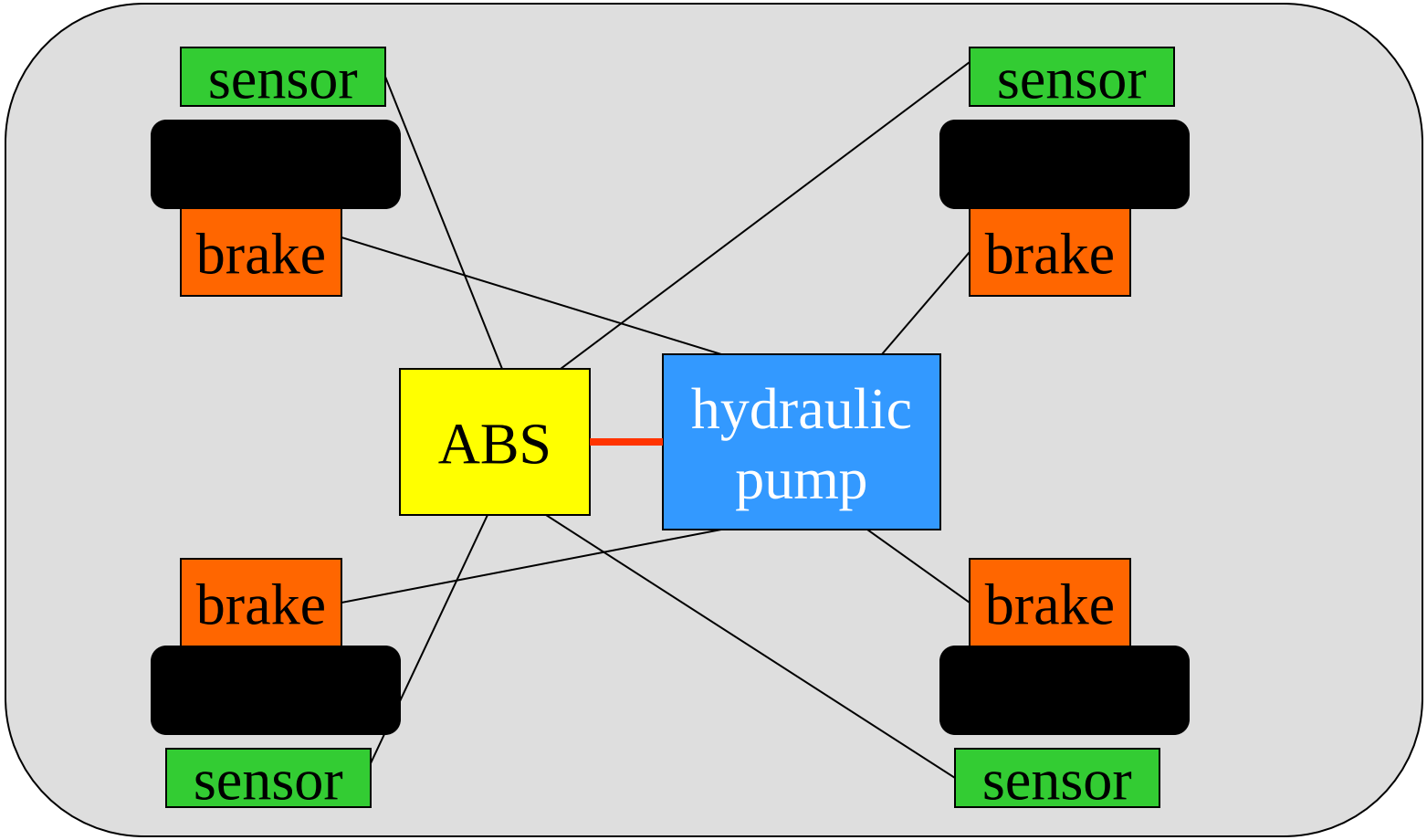
BMW 850i brake and stability control system

- Anti-lock brake system (ABS): pumps brakes to reduce skidding.
- Automatic stability control (ASC+T):
  - ASC+T: Controls throttle, ignition timing, automatic transmission
- ABS and ASC+T communicate.

- ABS was introduced first → ASC+T interfaces with existing ABS module.

Taken/modified from "Computers as Components ..", W. Wolf

BMW 850i, cont'd.



Taken/modified from "Computers as Components ..", W. Wolf

Characteristics of embedded systems

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

Taken/modified from "Computers as Components ..", W. Wolf

Functional complexity

- Often have to run sophisticated algorithms or multiple algorithms.
  - Cell phone, laser printer, engine controls
- 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.
  - e.g., MULTIMEDIA→ audio (slow) & video (fast)

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.

Why use microprocessors?

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

The performance paradox

- Microprocessors use much more logic to implement a function than does custom logic.
  → overhead in fetch-decode-exec
- But microprocessors are often at least as fast:
  - heavily pipelined;
  - large design teams → fast/optimized cpu's
  - aggressive VLSI technology → use latest fabrication methods

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: Engg vs CompSci approach

- How much hardware do we need?
  - How big is the CPU? Memory?
- How do we meet timing deadlines?
  - Faster hardware or cleverer software?
  - Does a faster CPU always help the speed?
- How do we minimize power?
  - Turn off unnecessary logic? Reduce memory accesses?

Taken/modified from "Computers as Components ..", W. Wolf

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? → no keyboards/screens → difficult to see inside
  - Development platform? → e.g., host/target platforms

Taken/modified from "Computers as Components ..", W. Wolf

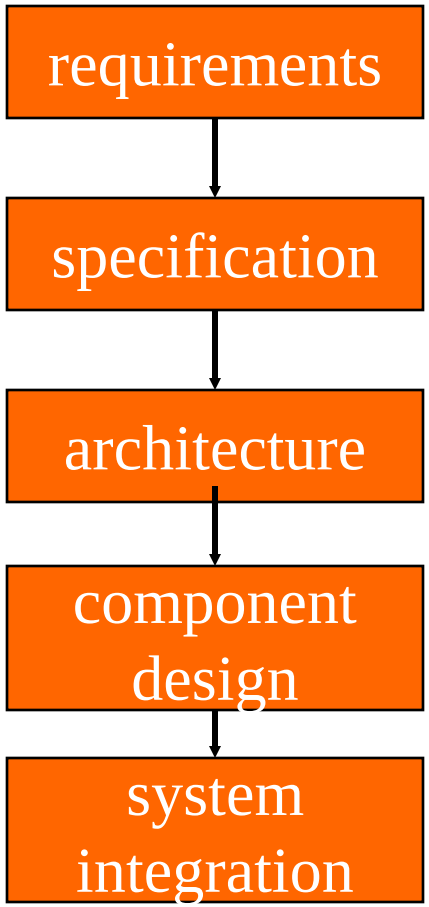
Design methodologies: Overview of design process

- A procedure for designing a system.
- Understanding your methodology helps you ensure you didn't skip anything.
- Most embedded systems are designed by TEAMS → Coordination

Design goals

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

Levels of abstraction



How it behaves

Structure in terms of large components

S/W and H/W modules

Taken/modified from "Computers as Components ..", W. Wolf

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.
  - verify the design to ensure it meets system goals, e.g., cost, speed, ...

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

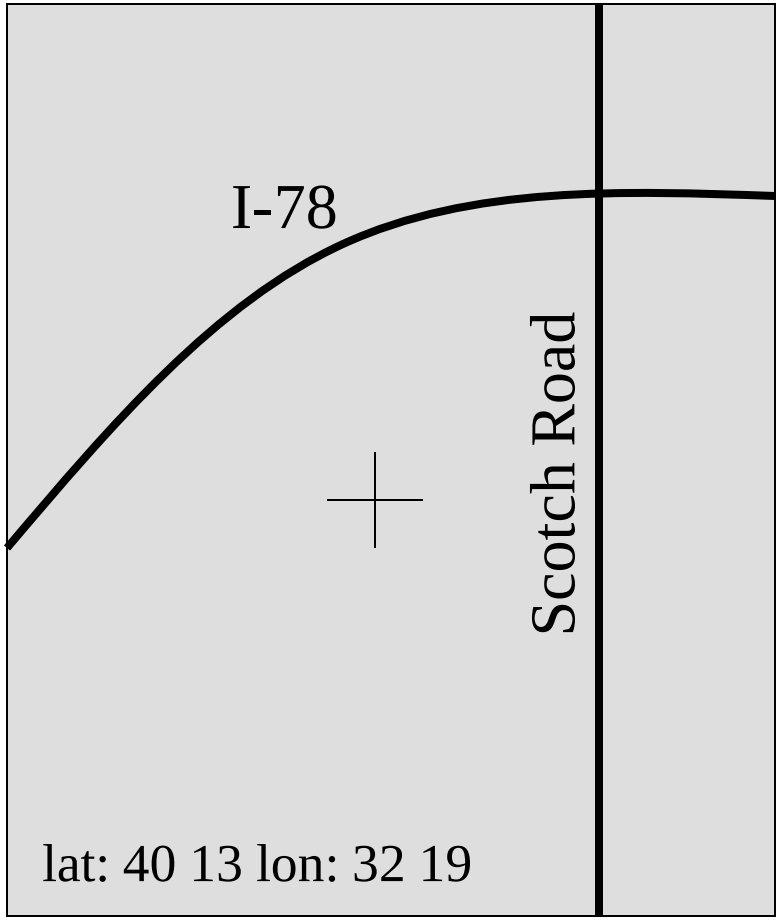
- Functional requirements:
  - output as a function of input.
- Non-functional requirements:
  - time required to compute output;
  - size, weight, etc.;
  - power consumption;
  - reliability;
  - etc.

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: 24 satellites orbiting the earth in 6 tracks
    - How many needed to locate an object?



Taken/modified from "Computers as Components ..", W. Wolf

GPS moving map needs (Requirements)

- Functionality: For automotive use. Show major roads and landmarks (highway driving).
- 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: $150 sreet price = approx. $30 cost of goods sold.

GPS moving map needs, cont'd.

- Physical size/weight: Should fit in hand
  - Requirements may not necessarily be specified in engineering units, e.g., Hand-size, ...
- 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 user-selectable resolutions; displays current lat/lon
performance	updates screen within 0.25 sec of movement
manufacturing cost	$30 cost-of-goods-sold
power	100 mW
physical size/weight	no more than 2”X6” 12 oz.

Taken/modified from "Computers as Components ..", W. Wolf

Specification

- A more precise description of the system:
  - should not (normally) imply a particular architecture;
  - provides input to the architecture design process.
- May include functional and non-functional elements.

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.
  - UML or other block diagrams may be used for description

Architecture design

- **Hardware components:**
  - **CPUs, peripherals, etc.**
- **Software components:**
  - **major programs and their operations.**
- **Must take into account functional and non-functional specifications.**
- **Errors can be very costly– experts perform this part, less experienced do detailed design**

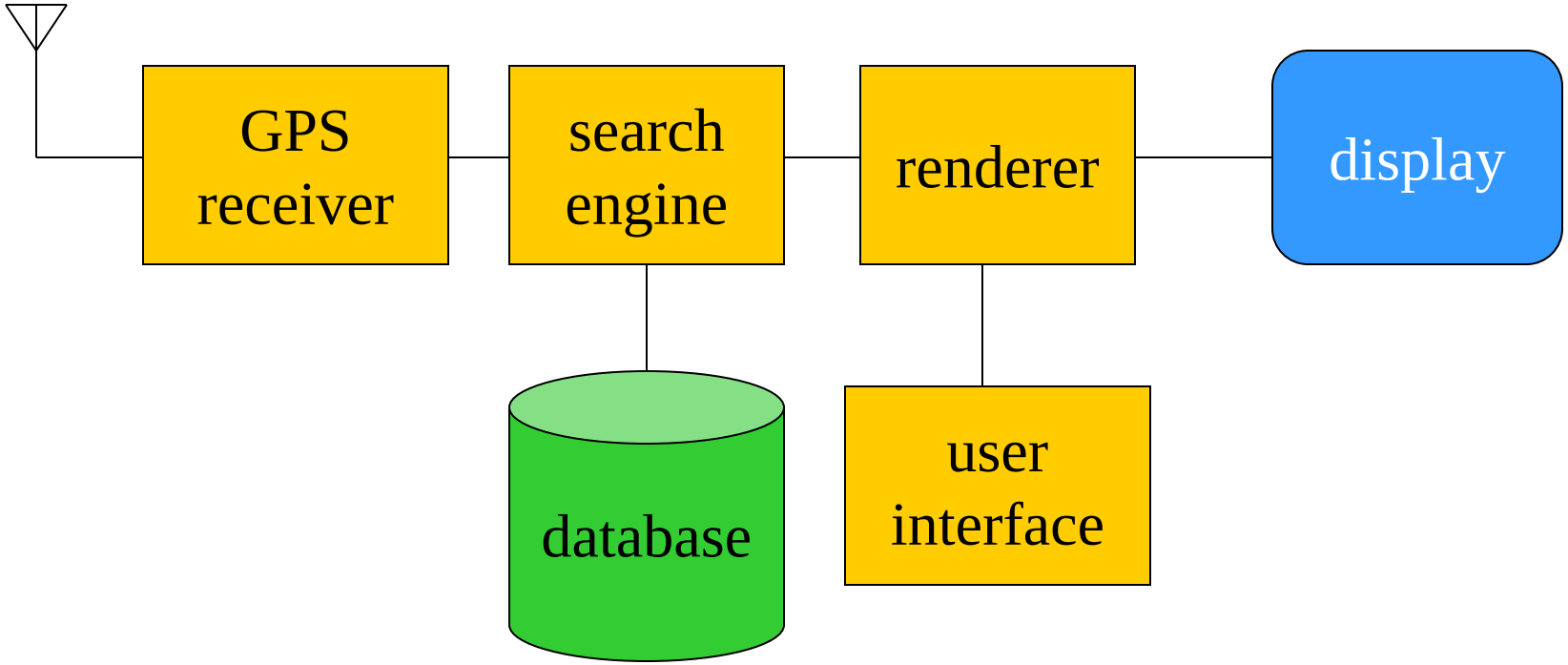
Taken/modified from "Computers as Components ..", W. Wolf

Specifications vs Architecture

- **Specifications:** Tell **WHAT** the system does

- **Architecture:** The next step; **HOW** the system does it

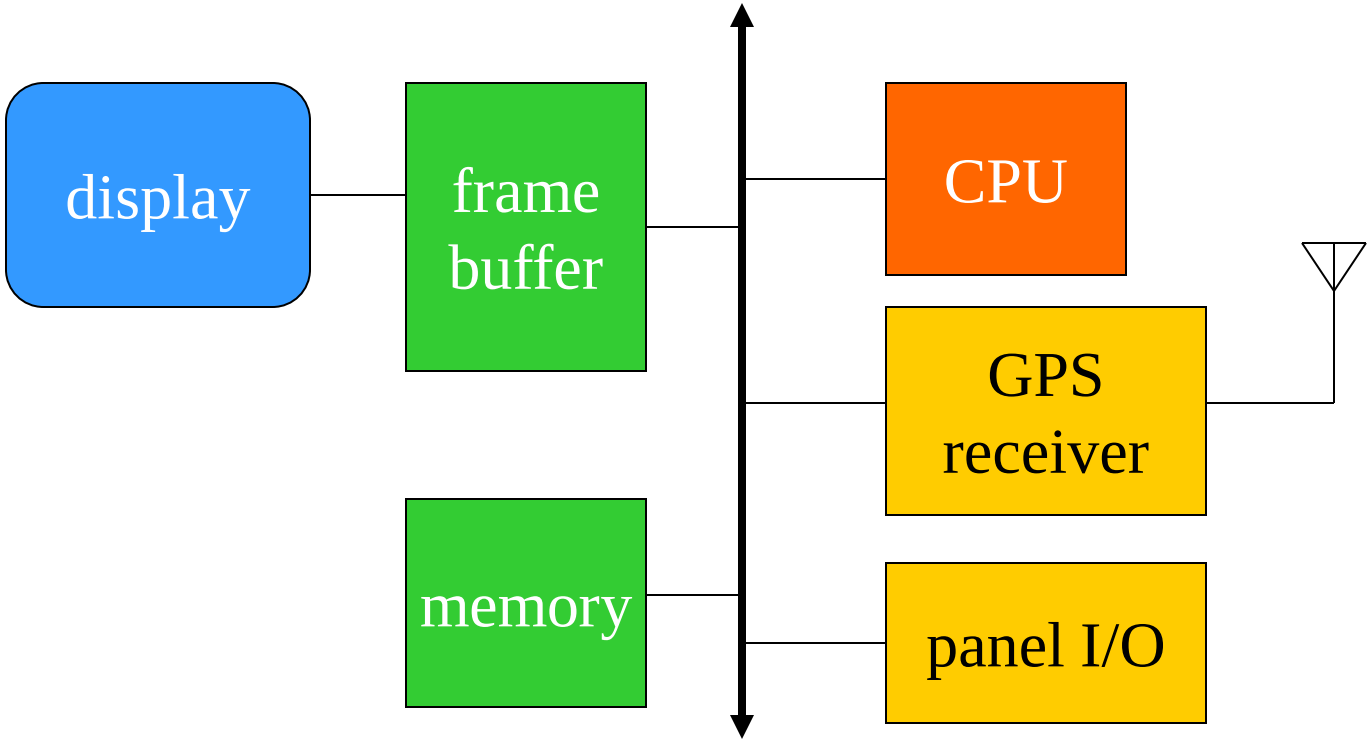
GPS moving map block diagram



Search and **rendering** can be done in parallel so that screen can be updated more fluidly

Taken/modified from "Computers as Components ..", W. Wolf

GPS moving map hardware architecture

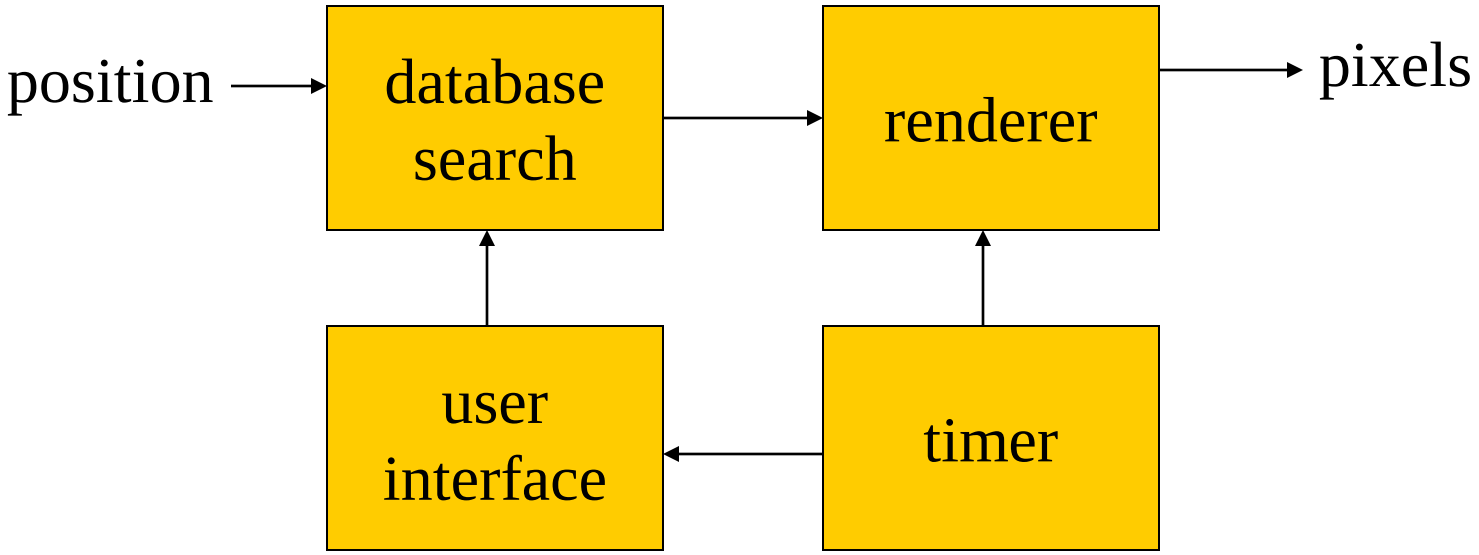


-**Frame buffer:** To display pixels

-**Memory:** program/data

Taken/modified from "Computers as Components ..", W. Wolf

GPS moving map software architecture



Taken/modified from "Computers as Components ..", W. Wolf

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.

Design tips

- Research the problem, requirements/specifications
- Obtain multiple solutions: At least 2
- Analyze solutions, choose the best one
- Work in a relaxed environment- not allways behind a computer
- Discuss solutions with a colleague

Good designs

- Well documented
- Testability

  - capability to test all requirements

- Modularity

  - each module should be small and simple
  - should be understood separately
  - hardware and application dependencies confined to a small number of modules

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.