Outline

- Challenges in Embedded Systems Design
- The Embedded System Design Process
- Formalisms for System Design
Challenges in Embedded Systems Design

- How much hardware do we need?
- How do we meet deadlines?
- How do we minimize power consumption?
- How do we design for upgradeability?
- Does it really work?

Difficulties in Design and Development

- Complex testing
  - run a real machine to have proper data
  - system must be tested in the embedded machine
- Limited observability and controllability
  - sometimes no keyboard or screen!
  - in real-time systems it’s not easy to stop the system to see what is going on
- Restricted development environments
  - much more limited than in PCs
  - usually compile code in PC and download it to embedded system
A design methodology is 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
- Better communication between team members
  - what they are supposed to do
  - what they should receive
  - when they have completed their assigned steps

Levels of abstraction

```
requirements

specification

architecture

component design

system integration
```
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

Design goals

- Performance
  - Overall speed, deadlines
- Functionality and user interface
- Manufacturing cost
- Power consumption
- Other requirements (physical size, etc.)
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 that it meets all system goals
  - cost, speed, ...
Requirements vs Specification

Consumers:
- are not embedded system designers
- see mostly users’ interactions
- most of the time have unrealistic expectations as to what can be done within their budgets
- have a different language

Translating from requirements to specification (from the consumer’s language to the designer’s)
- capturing a consistent set of requirements from the customer
- massaging those requirements into a more formal specification

Functional vs. non-functional requirements

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

- **Performance**
  - major consideration for the usability of the system and its ultimate cost
  - may be a combination of soft performance metrics and hard deadlines

- **Cost**
  - manufacturing costs (e.g. components, assembly)
  - nonrecurring engineering (NRE) costs (e.g. personnel, designing the system)

- **Physical size and weight**
  - depends on the application

- **Power consumption**
  - important not only in battery-powered systems
  - specified in terms of battery life

Validating the requirements

- Requires understanding what people want and how they communicate it

- User interface requirements can be refined by using a mock-up
  - may be executed on a PC

- Physical, nonfunctional models of devices can also help
  - better idea of size and weight
Sample requirements form (1/3)

- name
- purpose
- inputs and outputs
- functions
- performance
- manufacturing costs
- power
- physical size/weight

Sample requirements form (2/3)

- name
- purpose
  - one- or two-line description
- inputs and outputs
  - types of data: analog? digital? mechanical?...
  - data characteristics: periodic? occasional? how many bits?...
  - types of I/O devices: buttons? A/D converters? video displays?...
- functions
  - more detailed description of the system
  - when the system receives an input, what does it do?
  - how do interface inputs affect these functions?
  - how do different functions interact?
• performance
  • must be identified earlier to ensure that the system works properly
• manufacturing costs
  • cost has substantial influence on architecture
  • work with some idea of the cost range
• power
  • battery powered? plugged into a wall?
• physical size/weight
  • more or less flexibility in the components to use

Beyond the requirements form

• The requirements form should be the introductory of a longer document
• After writing the requirements you should check for internal inconsistency
  • forget to assign functions to an input/output?
  • considered all modes of operation?
  • unrealistic number of features into a battery-powered, low-cost machine?
Example: GPS moving map requirements

- Moving map obtains position from GPS
- Paints map from local database

GPS moving map needs (1/2)

- **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
GPS moving map needs (2/2)

- Cost
  - € 500 street price = approx. € 100 cost of goods sold
- 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 400x600
- functions: 5-receiver GPS; three resolutions displays current lat/lon
- performance: updates screen within 0.25 sec of movement
- manufacturing costs: ≈ 100€ cost-of-goods-sold
- power: 100mW
- physical size/weight: no more than 5cm x 15cm, 350g
Specification

Serves as the contract between the customer and the architects.

- 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

- Should be understandable enough
  - so that someone can verify that it meets system requirements and overall expectations of the customer
- Should be unambiguous

Problems of unclear specifications

- implementation of wrong functionality
- system architecture may be inadequate to meet the needs of the implementation
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
  - e.g. operating the GPS receiver

Specification vs architecture design

- The specification does not say how the system does things, only what the system does

- The purpose of the architecture is to describe how the system implements the functions

- The architecture is a plan for the overall structure of the system
  - it will be used later to design the components

The creation of the architecture is the 1st phase of the so called “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**

- Display
- Frame buffer
- Memory
- CPU
- GPS receiver
- Panel I/O

**GPS moving map software architecture**

- Position
- Database search
- Renderer
- Pixels
- User interface
- Timer
Designing hardware and software components

- Must spend time architecting the system before you start coding

- Some components are ready-made
  - CPU, memory chips, ...
  - some are software components

- Some can be modified from existing designs

- Others must be designed from scratch
  - at least you may have to design the board
  - custom programming

Identifying existing hardware and software components

- Much of the design process is composition of existing modules
  - Hardware boards and peripheral interfaces
  - Software modules
    - Operating System or run-time environment
    - Middleware (ex. Communications layers, RFID coding)
    - Applications

- Usually designers try to keep their usual development environment
  - Processors and low-level tools
  - Operating systems or run-time environments
System integration

- Put together the components (hardware blocks and software modules)
- Not as easy as it sounds...
  - 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
  - debugging facilities are limited

Formalisms for system modeling

- Need languages to describe systems
  - useful across several levels of abstraction
  - understandable within and between organizations
- Block diagrams are a start, but don’t cover everything
Object-oriented design

- Object-oriented (OO) design: A generalization of object-oriented programming
  - encourages design to be described as a number of interacting objects
  - some objects will correspond to real pieces of software
- Object = state + methods
  - State provides each object with its own identity
  - Methods provide an abstract interface to the object
- Unified Modeling Language (UML)
  - Encourages design by successive refinement and progressively adding detail to the design

Outline

- Challenges in Embedded Systems Design
- The Embedded System Design Process
- Formalisms for System Design

Next class

- Architecture of Embedded Systems