Lecture VII: Embedded System Design

TDDI11: Embedded Software

Dept. Computer and Information Science (IDA)
Linköpings universitet
Design challenge: optimizing design metrics

• Common metrics
  – **NRE cost** (Non-Recurring Engineering cost): The one-time monetary cost of designing the system
  – **Unit cost**: the monetary cost of manufacturing each copy of the system, excluding NRE cost
  – **Size**: the physical space required by the system
  – **Performance**: the execution time or throughput of the system
  – **Power**: the amount of power consumed by the system
  – **Flexibility**: the ability to change the functionality of the system without incurring heavy NRE cost
Design challenge: optimizing design metrics

- **Common metrics (continued)**
  - **Time-to-prototype**: the time needed to build a working version of the system
  - **Time-to-market**: the time required to develop a system to the point that it can be released and sold to customers
  - **Maintainability**: the ability to modify the system after its initial release
  - **Safety**: absence of catastrophic consequences on the user(s) and the environment.
Design metric competition:
improving one may worsen others
Design methodology is important

• The loss of the Mars climate observer (1999)
  • Most likely approached Mars too closely
  • One of the problems: requirement
    • Jet Propulsion Lab expected values in Newton
    • Contractor calculated in pound force
    • Trajectory adjustments unsuccessful
    • Was not caught by configuration process or manual inspection
Design methodology, design flow

• **Design methodology**: the process of creating a system
  – Goal: optimize competing design metrics
    • Time-to-market
    • Design cost
    • Manufacturing cost
    • Quality, etc.

• **Design flow**: sequence of steps in a design methodology.
  – May be partially or fully automated with compilers and CAD tools.
  – Use tools to transform, verify design.

• Design flow is one component of design methodology. Methodology also includes management, organization, etc.
Waterfall model

Early model for software development.
Spiral model

- More realistic, but can be complex, time to market?
Successive refinement:

• Several iterations, suited for when unfamiliar with domain application
Design flows for embedded systems

• Embedded systems need design of hardware and software

• Even if you don’t design hardware, you still need to select the correct boards, plug together several hardware components, and write code
Hierarchical design

- from a most abstract complete system design flow
- to more detailed design flow of components
- many involved teams: requirements, design, testing
- Communication is important!
"Concurrent" and "Over-the-wall" Engineering

- Large project, several teams
- Good communication between teams: e.g., components and design
- Eliminate “over-the-wall” approach
  - Cross-functional teams
  - Concurrent product realization
  - Information sharing
  - Integrated project management
Frequency of faults

[Jim Cooling 2003, cited from DeMarco78]
Requirement and Specification

• Requirement: informal descriptions of what customer wants:
  • Correctness, unambiguousness, completeness, verifiability, consistency, modifiability, traceability
• Specification: more detailed, precise descriptions
• Both Requirements and specification describe system behavior from outside
Requirement and Specification

• Describing embedded systems’ behavior
  – Can be extremely difficult
    • Complexity increasing with increasing IC capacity
    • Desired behavior often not fully understood in beginning

• English (or other natural language) common starting point
  – Precise description difficult to impossible
The 2002-Überlingen crash

Several factors, including ambiguities about relation between TCAS (Trafic Collision Avoidance System) and ATC (Air Trafic Control)
Models and languages

• How can we (precisely) capture behavior?
  – We may think of languages (C, C++), but computation model is the key

• Common computation models:
  – Sequential program model
    • Statements, rules for composing statements, semantics for executing them
  – State machine model
    • For control dominated systems, monitors control inputs, sets control outputs
  – Dataflow model
    • For data dominated systems, transforms input data streams into output streams
Models and languages

From “experience from specifying TCASII requirements using RSML”, Heimfahl and al, 1998.
Quality assurance

- Quality should be built in
- Quality standards (e.g., iso 9000)
- Verify specification and review designs
- Long lived bugs are more expensive
- The Therac-25 medical imaging system
  - Six known accidents, massive radiation overdose
  - Machine software developed in assembly by single programmer over several years
  - Some errors depended on typing speed of operators
  - Limited safety analysis, no mechanical backups, overly complex programs
Reuse: platforms

• A partial design:
  – for a particular type of system;
  – includes embedded processor(s);
  – may include embedded software;
  – customizable to a customer’s requirements:
    • software;
    • component changes.
Why platforms?

- Any given space has a limited number of good solutions to its basic problems.

- A platform captures the good solutions to the important design challenges in that space.

- A platform reuses architectures.
ALTERNATIVE TO PLATFORMS

• General-purpose architectures.
  • May require much more area to accomplish the same task.
  • Often much less energy-efficient.

• Reconfigurable systems.
  • Good for pieces of the system, but tough to compete with software for miscellaneous tasks.
Standards and platforms

• Many high-volume markets are standards-driven:
  – wireless;
  – multimedia;
  – networking.

• Standard defines the basic I/O requirements.
• Systems house chooses implementation of standards functions:
  – improved quality, lower power, etc.

• Product may be differentiated by added features:
  – cell phone user interface.

• Standards encourage platform-based design.
Platform vs. full-custom

- Platform has many fewer degrees of freedom:
  - harder to differentiate;
  - can analyze design characteristics.

- Full-custom:
  - extremely long design cycles;
  - may use less aggressive design styles if you can’t reuse some pieces.
Platforms and IP-based Design

- Platforms use IP:
  - CPUs;
  - memories;
  - I/O devices.
- Platforms are IP at the next level of abstraction.
ADVANTAGES OF PLATFORM-BASED DESIGN

• Fast time-to-market.
• Reuse system design---hardware, software.
• Allows chip to be customized to add value.
COSTS OF PLATFORM-BASED DESIGN

• Masks.
• NRE: design of the platform + customization.
• Design verification.
SOFTWARE AND HARDWARE REUSE

• Want to reuse as many hardware components as possible:
  • known performance, power.
• Want to use software libraries where possible.
• RTOS simplifies design of multi-tasking systems.