CPS Architectures and Methodologies

Marilyn Wolf
Georgia Tech

Outline

- Examples from automotive and aircraft systems.
- Observations on architecture.
- Design methodologies.

Examples

Automotive.

Aerospace.

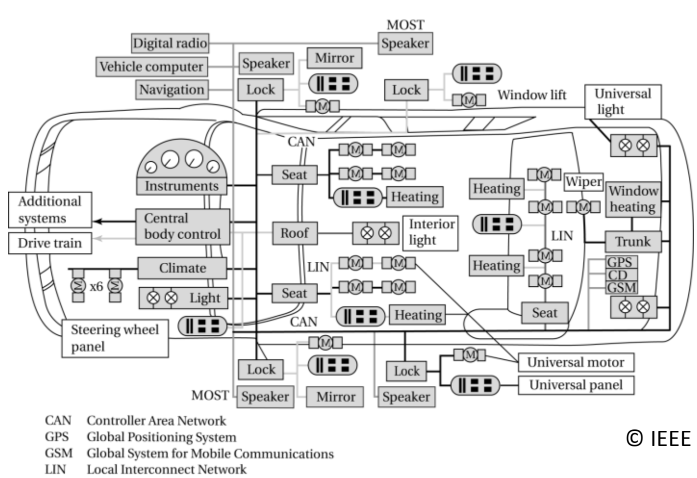
Automotive and aviation electronics

- Some functions are safety-critical.
- Must operate in real-time.
- Must fit within power budget (limited by generator).
- Must be lightweight to fit within vehicle weight budget.

Automotive electronics/avionics uses

- Operator vs. passenger: Passenger operations are less critical, more varied (TV, Internet, etc.).
- Control vs. instrumentation: Instruments report on the vehicle, control closes the loop.
  - Low-priority operations should not interfere with high-priority operations in the system: flight surfaces vs. instruments; instruments vs. passenger devices.

Automobiles as distributed embedded systems



© IEEE Computer Society

Architectural principles

System requirements

Networks and distributed control.

Cyber-side design issues.

Design goals

- Traditional software view of requirements:
  - Functional requirements: input/output relations.
  - Non-functional requirements: cost, performance, power, etc.
- Software view of requirements is not well-suited to control system requirements.
- Reliability and safety are first-tier requirements.
- Some project goals may be difficult to measure.

Design parameters

- Delay.
  - Latency.
  - Jitter.
- Bandwidth.
- Guarantees.
- Energy consumption.
  - Limited power available from generator.
  - Heat dissipation.
- Security.

Aspects of performance

- Embedded system performance can be measured in many ways:
  - Average vs. worst/best-case.
  - Throughput vs. latency.
  - Peak vs. sustained.
- Digital control systems are sampled.
  - Sample period determines deadline, latency.

Energy/power

- Energy consumption is important for battery life.
- Power consumption is important for heat generation or for generator-powered systems (vehicles).

Cost

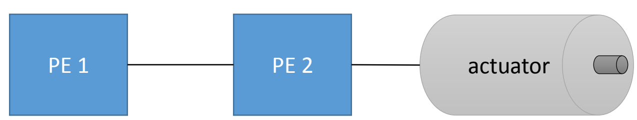
- Design cost must be paid off across all the systems.
  - Hardest in small-volume applications.
- Manufacturing cost is incurred for each device.
- Lifetime costs include software and hardware maintenance and upgrades.

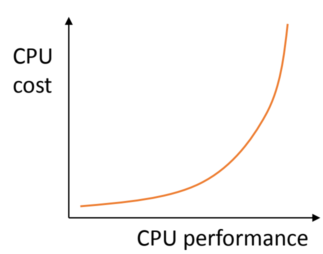
Other design attributes

- Design time must be reasonable. May need to finish by a certain date.
- System must be reliability; reliability requirements differ widely.
- Quality includes reliability and other aspects: usability, durability, *etc*.
- Systems that must be certified must use certifiable, documented design processes.

Why distributed control?

- Reduce closed loop delay by putting computation near physics.
- Improved cost/performance by reducing scheduling overhead.
  - Rate-monotonic scheduling requires unused cycles.
  - CPU cost is non-linear in performance.
- Control architecture drives cyber architecture?



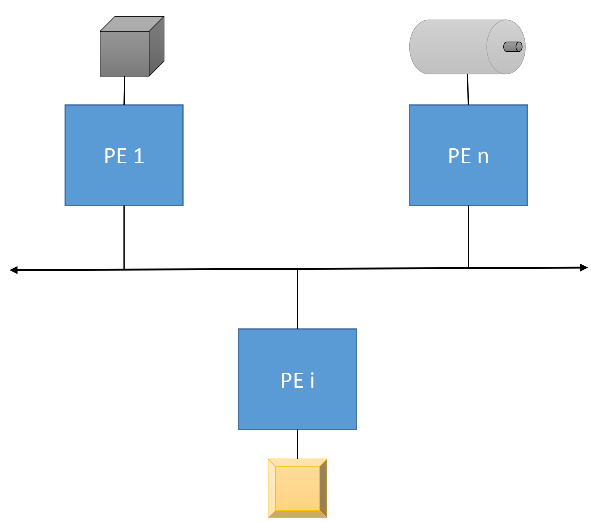


Cyber-oriented architectural aspects

- Network hardware architecture:
  - Bandwidth.
  - Scheduling.
- Hardware-dependent software (HDS), OS, and middleware:
  - Scheduling.
  - Latency.
  - Contention and performance effects.
- Application-level tasks:
  - Multi-criticality.
  - Security.

Bus-based control system

- Bus must provide real-time services.
  - Typically provided by TDMA, time-triggered architecture, etc.
- Generally based on message passing, not shared memory.
- Some bus standards provide redundancy and fault recovery methods.



CPS internetworking

- Cars and airplanes are internetworked---a network of heterogeneous networks.
  - Different networks for different cost/performance/guarantees points.
- Automotive networks:
  - Flex-ray for safety-critical, timing-critical functions.
  - CAN for less critical functions.
  - LIN for doors and other low-cost, low-bandwidth functions.
  - MOST for passenger entertainment.
- Internetworking support appears to be ad hoc and bridge-based.

The multi-criticality paradigm

- Complex systems are built from many tasks, some of which are more important than others (aviate, navigate, communicate):
  - Flight control.
  - Navigation.
  - Communication.
  - Sensors.
  - Mission planning.
- Must meet deadlines for all high-criticality tasks.
- Schedule lower-criticality tasks based on priorities and available resources.

Methodologies

Embedded system design methodologies.
Methodologies and standards.
Electronic system level (ESL) design methodologies.
System-on-chip vs. CPS.
CPS methodologies.

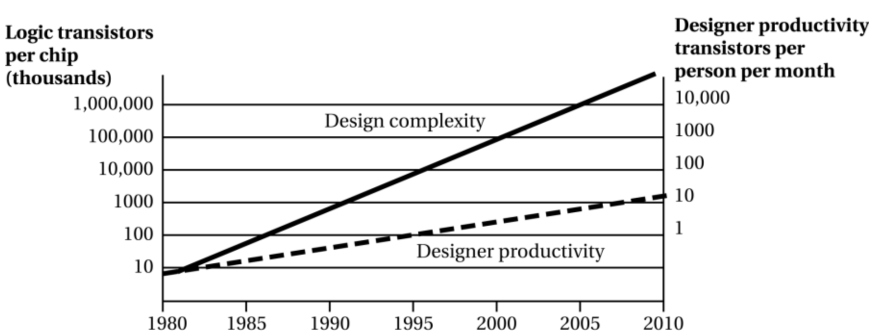
Design methodology

- Design methodology: a procedure for creating an implementation from a set of requirements.
- Methodology is important in embedded computing:
  - Must design many different systems.
  - We may use same/similar components in many different designs.
  - Design time, results must be predictable.

Embedded system design challenges

- Design space is large and irregular.
- We don't have synthesis tools for many steps.
- Can't simulate everything.
- May need to build special-purpose simulators quickly.
- Often need to start software development before hardware is finished.

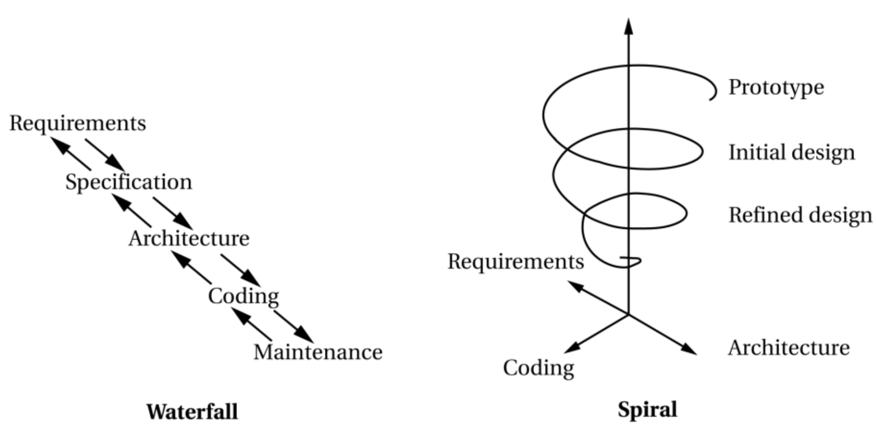
Design complexity vs. designer productivity



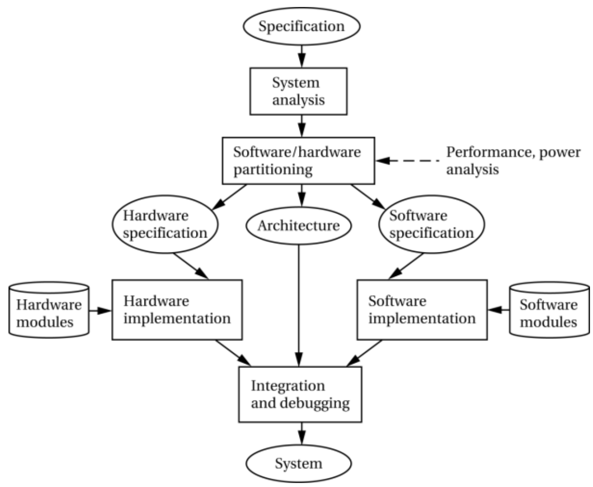
Basic design methodologies

- Figure out flow of decision-making.
- Determine when bottom-up information is generated.
- Determine when top-down decisions are made.

Waterfall and spiral models

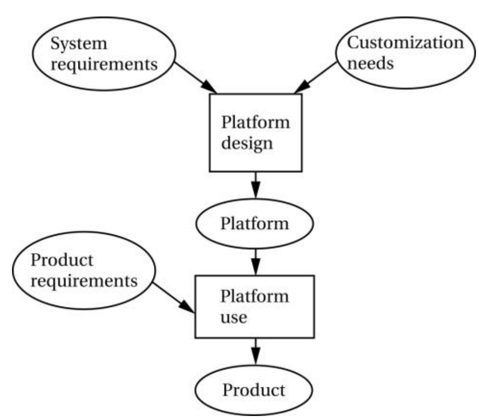


Hardware/software co-design flow



Platform-based design

- Platform includes hardware, supporting software.
- Two stage process:
  - Design the platform.
  - Use the platform.
- Platform can be reused to host many different systems.



Platform design

- Turn system requirements and software models into detailed requirements.
    - Use profiling and analysis tools to measure existing executable specifications.
- Explore the design space manually or automatically.
- Optimize the system architecture based on the results of simulation and other steps.
- Develop hardware abstraction layers and other software.

Programming platforms

- Programming environment must be customized to the platform:
    - Multiple CPUs.
    - Specialized memory.
    - Specialized I/O devices.
- Libraries are often used to glue together processors on platforms.
- Debugging environments are a particular challenge.

Standards-based design methodologies

- Standards enable large markets.
- Standards generally allow products to be differentiated.
  - Different implementations of operations, so long as I/O behavior is maintained.
  - User interface is often not standardized.
- Standard may dictate certain non-functional requirements (power consumption), implementation techniques.

Reference implementations

- Executable program that complies with the I/O behavior of the standard.
  - May be written in a variety of language.
- In some cases, the reference implementation is the most complete description of the standard.
- Reference implementation is often not well-suited to embedded system implementation:
  - Single process.
  - Infinite memory.
  - Non-real-time behavior.

Designing standards-based systems

- Design and implement system components that are not part of the standard.
- Perform platform-independent optimizations.
- Analyze optimized version of reference implementation.
- Design hardware platform.
- Optimize system software based on platform.
- Further optimize platform.
- Test for conformity to standard.

Design verification and validation

- Showing that the design is correct and fixing bugs often takes more time than initial design.
- Design correctness activities:
  - Testing exercises an implementation with stimuli and observed outputs.
  - Validation compares implementation to requirements or spec.
  - Verification compares the design at one level of abstraction to another.

V&V techniques

- Simulation uses software/hardware models to compute outputs from inputs.
  - Simulator-in-the-loop integrates a simulator of a physical plant with cyber controllers.
- Formal methods perform proofs: equivalence, properties, *etc.*
- Manual methods such as code reviews, walkthroughs, and inspections have been shown to catch many bugs.

A methodology of methodologies

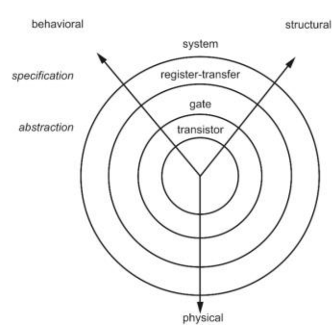
- Embedded systems include both hardware and software.
  - HW, SW have their own design methodologies.
- Embedded system methodologies control the overall process, HW/SW integration, etc.
  - Must take into account the good and bad points of hardware and software design methodologies used.

Joint algorithm and architecture development

- Some algorithm design is necessarily performed before platform design.
- Algorithm development can be informed by platform architecture design.
  - Performance/power/cost trade-offs.
  - Design trends over several generations.

Electronic system-level design

- ESL for mixed hardware/software systems.
- Often driven from SystemC or Matlab.
- Concentrates on refinement of abstract system to HW, SW components.

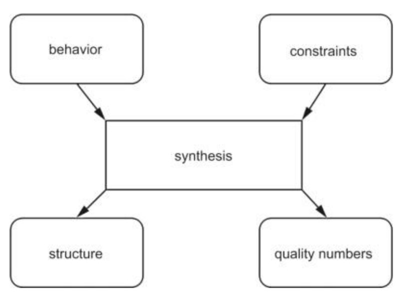


ESL tools

- Daedalus for multimedia MPSoCs:
  - Application modeled as Kahn process.
  - Design space exploration based on high-level models of major HW components.
- SCE uses three-level design hierarchy:
  - Specification is set of behaviors and abstract communication channels.
  - Transaction-level model maps onto platform architecture.
  - Implementation model is cycle-accurate.

X-chart

- X-chart extends Gajski-Kuhn Y-chart:
  - Input to synthesis is behavior and constraints.
  - Synthesis produces structure and quality metrics.



CPS and SoC: Differences

- Locked vs. evolving design:
  - SoC is locked at tapeout.
  - Many networked CPS are long-lived and evolve.
- Mission criticality:
  - Reliability a recent trend in SoC.
  - Many advanced CPS that motivate research are mission or safety critical.
- Self-containment:
  - Many CPS designs are constrained by their physical plant.
- Specs:
  - SoC specs are lower level (SystemC).
  - CPS specs are higher level (ADSL, step response, etc.).

CPS and SoC: Similarities

- Real-time.
- Software intensive.
- Complex functional specs, demanding non-functional specs.
- Networking:
  - SoCs have internal heterogeneous networks, often synthesized.
  - CPS use heterogeneous networks, many COTS.

SoC techniques and CPS

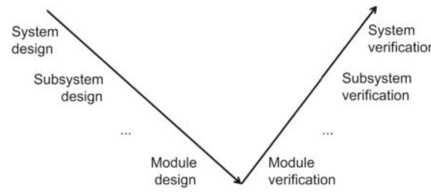
- Specification languages.
- Modeling:
  - Transaction-level modeling.
  - Power and thermal models.
  - Network models.
- Platform-based design.

CPS arch and model-based design

- Model-based design is primarily top-down.
- SoC world has a lot of experience with bottom-up design.
- Adapting the requirements effectively requires bottom-up design information.

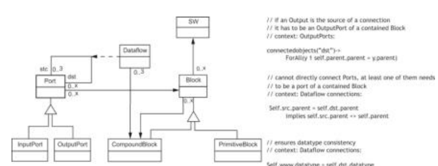
CPS design methodologies

- CPS requires deep design hierarchies, complex verification methodology.
- V-chart: design by top-down refinement, verify bottom-up



Karsai et al. Model-Integrated Computing

- Methodology and toolset for model-based design.
- Domain-specific modeling language (DSML) or meta-language allows deisgner to work directly in application domain.
- Composition:
  - Abstraction and hierarchy.
  - Modularization.
  - Interfaces and connecdtion.
  - Aspect-oriented programming.
  - Non-local interactions.



[Kar03] © 2003 IEEE

Model-based design in specific domains

- Tariq et al. DSML for irrigation networks using GME.
  - Components include channels, pools, gates, meters, physical links and communication.
  - Saint Venant's equation describes movement of water in the irrigation network.
- AADL is an SAE standard language for model-based engineering motivated by aerospace.
  - Threads, processes, data for software.
  - Processors, memories, and comm for hardware.