Design of Embedded Systems: Methodologies, Tools and Applications

Alberto Sangiovanni-Vincentelli
Dept. of EECS
University of California Berkeley

Disaggregation:
Electronic Systems Design Chain
Outline

- Automotive Applications
- Distributed System Design Methodology and Flow
- Platform-based Design
- UAV Control Example
- Metropolis
Automotive Supply Chain:
Car Manufacturers

- Product Specification & Architecture Definition
  (e.g., determination of Protocols and Communication standards)
- System Partitioning and Subsystem Specification
- Critical Software Development
- System Integration

Electronics for the Car: A Distributed System

Today more than 80 Microprocessors and millions of lines of code
Automotive Supply Chain:
Tier 1 Subsystem Providers

- Subsystem Partitioning
- Subsystem Integration
- Software Design: Control Algorithms, Data Processing
- Physical Implementation and Production

1. Transmission ECU
2. Actuation group
3. Engine ECU
4. DBW
5. Active shift display
6/7. Up/Down buttons
8. City mode button
9. Up/Down lever
10. Accelerator pedal
11. Brake switch

BOSCH

Automotive Supply Chain:
Tier 2 Platform & IP Providers

- "Software" platform: RTOS and communication layer
- "Hardware" platform: Hardware and IO drivers

WindRiver

SW Platform layer (> 60% of total SW)

Application Platform layer (≤ 10% of total SW)

µControllers Library

Application Programming Interface

I/O drivers & handlers (> 20 configurable modules)

OSEK RTOS

OSEK COM

Transport

KWP 2000

ST10

Nec78k HC08 HC12 H8S26 MB90

Chess/ISIS/MSI
**Complexity, Quality, Time-to-Market: TODAY**

<table>
<thead>
<tr>
<th></th>
<th>PWT UNIT</th>
<th>BODY GATEWAY</th>
<th>INSTRUMENT CLUSTER</th>
<th>TELEMATIC UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEMORY</td>
<td>256 KB</td>
<td>128 KB</td>
<td>184 KB</td>
<td>8 MB</td>
</tr>
<tr>
<td>LINES OF CODE</td>
<td>50,000</td>
<td>30,000</td>
<td>45,000</td>
<td>300,000</td>
</tr>
<tr>
<td>PRODUCTIVITY</td>
<td>6 LINES/DAY</td>
<td>10 LINES/DAY</td>
<td>6 LINES/DAY</td>
<td>10 LINES/DAY*</td>
</tr>
<tr>
<td>RESIDUAL DEFECT RATE @ END OF DEV</td>
<td>3000 PPM</td>
<td>2500 PPM</td>
<td>2000 PPM</td>
<td>1000 PPM</td>
</tr>
<tr>
<td>CHANGING RATE</td>
<td>3 YEARS</td>
<td>2 YEARS</td>
<td>1 YEAR</td>
<td>&lt; 1 YEAR</td>
</tr>
<tr>
<td>DEV. EFFORT</td>
<td>40 MAN-YEAR</td>
<td>12 MAN-YEAR</td>
<td>30 MAN-YEAR</td>
<td>200 MAN-YEAR</td>
</tr>
<tr>
<td>VALIDATION TIME</td>
<td>5 MONTHS</td>
<td>1 MONTH</td>
<td>2 MONTHS</td>
<td>2 MONTHS</td>
</tr>
<tr>
<td>TIME TO MARKET</td>
<td>24 MONTHS</td>
<td>18 MONTHS</td>
<td>12 MONTHS</td>
<td>&lt; 12 MONTHS</td>
</tr>
</tbody>
</table>

* C++ CODE

**Embedded Software Design: Our Take**

- Embedded Software Design must not be seen as a problem in isolation, it is an, albeit essential, aspect of **EMBEDDED SYSTEM DESIGN**
- Our vision is to change the way in which ESW is developed today by linking it:
  - Upwards in the abstraction layers to system functionality
  - Downwards in the programmable platforms that support it thus providing the means to verify whether the constraints posed on Embedded Systems are met.
Outline

- Automotive Applications
- Distributed System Design Methodology and Flow
- Platform-based Design
- UAV Control Example
- Metropolis

Virtual Integration is key for Distributed System Design

- Fct1 Spec & Sim.
- Fctn Spec & Sim.
- ECU1 Integration ECU
- ECUk Integration ECU

- ECU SW Scheduling Adoption and Validation
- Communication Protocol Adoption and Validation
- Safety Concept Proof via Fault Injection
- ECU Optimization/Derivative Design
- Functional Network Definition and Validation (Timed and un-Timed)

Source: BMW
Focus on Safety-Critical Real Time

- Most challenging problem
- Needs tight integration between algorithms and implementation
- Constraints include timing and fault tolerance
- Fault tolerance can be addressed at all levels of abstraction
Safety Critical Issues: Fault Analysis

Safety Concept Proof via Fault Injection (HW, SW, Bus...)

DRAFTS: Distributed Real-time Applications
Fault Tolerant Scheduling

- Automatic (off-line) synthesis of fault tolerant schedules for periodic algorithms on a distributed architecture
- Automatic (off-line) verification that all intended faults are covered

Long-term goals:
- Design Methodology for Safety Critical Distributed Systems
- Manage the design complexity of modern Drive-By-Wire applications

C. Pinello, UCB, T. Demmeler and J. Ehret, BMW
DRAFTS Strategy

- Identify critical functionality and possible faults
- Replicate critical functionality to withstand faults
- Exploit architecture redundancy to speed-up execution (in absence of faults)
- Functional Verification that all intended faults are covered

Outline

- Automotive Applications
- Distributed System Design Methodology and Flow
  - Platform-based Design
- UAV Control Example
- Metropolis
Platforms: Evolution

In general, a platform is an abstraction layer that covers a number of possible refinements into a lower level. The platform representation is a library of components including interconnects from which the lower level refinement can choose.
Principles of Platform methodology: Meet-in-the-Middle

- **Top-Down:**
  - Define a set of abstraction layers
  - From specifications at a given level, select a solution (controls, components) in terms of *components (Platforms)* of the following layer and propagate constraints

- **Bottom-Up:**
  - Platform components (e.g., micro-controller, RTOS, communication primitives) at a given level are abstracted to a higher level by their functionality and a set of parameters that help guiding the solution selection process. The selection process is equivalent to a covering problem if a common semantic domain is used.

Outline

- Automotive Applications
- Distributed System Design Methodology and Flow
- Platform-based Design
- UAV Control Example
- Metropolis
Platform-Based Design of Unmanned Aerial Vehicles (source: J. Liebman)

I. Platform-Based Design
II. UAV System
III. Synchronous Embedded Control

UAV System: Sensor Overview

- **Goal**: basic autonomous flight
- **Need**: UAV with allowable payload
- **Need**: combination of GPS and Inertial Navigation System (INS)
- **GPS** (senses using triangulation)
  - Outputs *accurate* position data
  - Available at *low rate* & has jamming
- **INS** (senses using accelerometer and rotation sensor)
  - Outputs estimated position with *unbounded drift* over time
  - Available at *high rate*
- Fusion of GPS & INS provides needed high rate and accuracy
**UAV System: Sensor Configurations**

- Sensors may differ in:
  - Data formats, initialization schemes (usually requiring some bit level coding), rates, accuracies, data communication schemes, and even data types
  - Differing Communication schemes requires the most custom written code per sensor

![Diagram showing Software Request and Software configurations]

**Platform Based Design for UAVs**

- **Goal**
  - Abstract details of sensors, actuators, and vehicle hardware from control applications

- **How?**
  - Synchronous Embedded Programming Language (i.e. Giotto)
  - Platform
Platform Based Design for UAVs

• Device Platform
  - *Isolates* details of sensor/actuators from embedded control programs
  - *Communicates* with each sensor/actuator according to its own data format, context, and timing requirements
  - *Presents* an API to embedded control programs for accessing sensors/actuators

• Language Platform
  - *Provides* an environment in which synchronous control programs can be scheduled and run
  - *Assumes* the use of generic data formats for sensors/actuators made possible by the Device Platform

Outline

• Automotive Applications
• Distributed System Design Methodology and Flow
• Platform-based Design
• UAV Control Example
• Metropolis
Metropolis Framework

Application-specific methodologies
- Multi-media, wireless communication, mechanical controls, processors

Meta-model Library
- Models of computation

Infrastructure
- Metropolis meta-model
  - language
  - modeling mechanisms
  - Meta-model compiler

Meta-model Library
- Architecture platforms

Tools
- Simulator
- QSS
- PIG
- STARS
- SPIN
...

Metropolis Project: main participants

- UC Berkeley (USA): methodologies, modeling, formal methods
- Cadence Berkeley Labs (USA): methodologies, modeling, formal methods
- Politecnico di Torino (Italy): modeling, formal methods
- Universitat Politecnica de Catalunya (Spain): modeling, formal methods
- Philips Research (Netherlands): methodologies (multi-media)
- Nokia (USA, Finland): methodologies (wireless communication)
- BWRC (USA): methodologies (wireless communication)
- BMW (USA): methodologies (fault-tolerant automotive controls)
- Intel (USA): methodologies (microprocessors)
- STMicroelectronics (France, Italy): methodologies (wireless platforms)
- Cypress (USA): methodologies (network processors, pSOC, all projects)
Metropolis meta-model

Concurrent specification with a formal execution semantics:

- **Computation**: \( f : X \rightarrow Z \)
  - proc

  Key difference with respect to UML, SystemC, …!!!

- **Communication**: state evaluation and manipulation
  - medium: defines states and methods

- **Coordination**: constraints over concurrent actions
  - quantity: annotation of each event (time, energy, memory, …)
  - logic: relates events and quantities, defines axioms on quantities
  - quantity-manager: algorithm to realize annotation subject to relational constraints

Metropolis Meta-Model

- Must describe objects at different levels of abstraction
  - Do not commit to the semantics of any particular model of computation
- Define a set of "building blocks"
  - specifications with many useful MoCs can be described using the building blocks
  - Processes, communication media and schedulers separate computation, communication and coordination
Supporting Theory

- Provide a semantic foundations for integrating different models of computation
  - Independent of the design language
  - Not just specific to the Metropolis meta-model
- Maximize flexibility for using different levels of abstraction
  - For different parts of the design
  - At different stages of the design process
  - For different kinds of analysis
- Support many forms of abstraction
  - Model of computation (model of time, synchronization, etc.)
  - Scoping
  - Structure (hierarchy)

Concluding Remarks

- Applications are critical to drive research and to test quality of results
- Safety-critical Real Time emphasis
- Rigorous methodology for distributed systems
- General framework to express designs at all levels of hierarchy and to support integration of foreign tools and designs
Embedded Software: Today

Embedded Software: Future?