Distributed Model Based Development for Car Electronics
Outline

• Background
• Methodology Paradigm Shift
Background
Automotive Supply Chain Spider Web

Tier 1

Tier 2

BOSCH

MAGNETI MARELLI

DELPHI

Infineon Technologies
# Distributed Car Systems Architectures

<table>
<thead>
<tr>
<th>Information Systems</th>
<th>Telematics</th>
<th>Fault Tolerant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Body Electronics</td>
<td>Body Functions</td>
<td>Fail Safe</td>
</tr>
<tr>
<td>System Electronics</td>
<td>Driving and Vehicle Dynamic Functions</td>
<td>Fault Functional</td>
</tr>
</tbody>
</table>

- Mobile Communications
- Distributed
- Fire Wall
- Theft warning
- Door Module
- Light Module
- ABS
- Network(ed)
- Redundant
- Safety Critical
- Steer by Wire
- Brake by Wire
- Shift by Wire
- Engine Management
- Access to WWW

**Protocols:**
- MOST Firewire
- CAN Lin
- CAN TTCAN
- FlexRay TTP

*Cadence Confidential*
Car Manufacturers

Product Specification & Architecture Definition (e.g., Protocols and Communication standards adoption)
System Partitioning and Subsystem Specification (to tier1)
Critical Software Development
System Integration

Cadence Confidential
Car Manufacturers: Challenges and Opportunities

• “More than 90% of the innovation in the car will come from Electronics” (Daimler-Chrysler)

• “More than > 20% of the cost of the car is by now due to Electronics” (BMW quote - PSA quote - ARM Quote)

• Trend:
  - Bring in house electronic competence
  - Keep in check the subsystem providers
    ▶ Plug-and-play among different suppliers
  - System integration increasingly complex
Tier 1 Subsystem Suppliers

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 position sensor
11. Brake switch

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

(Cadence Confidential)
Tier1 Subsystem Suppliers

Applications Platform layer

SW Platform layer

OSEK RTOS

Application Libraries

I/O drivers & handlers (configurable modules)

μControllers Library

HW layer

OSEK COM

CCP

KWP 2000

Transport

Platform Integration: “firmware” and “glue software”

Software Design: “Application”
Automotive Supply Chain: Platform & IP Providers

**Application Platform layer**

- OSEK RTOS
- Application Libraries
- Customer Libraries
- Application Programming Interface
- I/O drivers & handlers (configurable modules)
- μControllers Library

**SW Platform layer**

- SW Platform layer
- Hardware layer
- HW layer

**HW layer**

- NEC78K
- PPC
- HC12
- H8S26
- HC7755

“Software” platform: RTOS and communication layer

“Hardware” platform: Hardware and IO drivers

Cadence Confidential
Methodology Paradigm Shift
Development Cycle: **Yesterday**    **Today**    **Tomorrow**

- **Yesterday**
  - Requirements
  - Manual Validation
  - Algorithm
  - Manual Debugging
  - Target SW
  - ECU Testing
  - Physical Prototype

*Cadence Confidential*
Development Cycle: Yesterday

Yesterday:
- Requirements
  - Manual Validation
  - Manual Debugging
  - Manual Testing
- Algorithm
  - Manual Debugging
- Target SW
  - ECU Automatic
  - Physical Prototype

Today:
- Requirements
  - Manual Validation
  - Manual Debugging
  - Manual Testing
- Algorithm
  - Manual Debugging
- Target SW
  - ECU Automatic
  - Physical Prototype Integration
Development Cycle: Yesterday  Today
Tomorrow

Yesterday: Single ECU Model Based Development Process for Multi-ECU Systems!

Today:

Requirements

Manual Validation

Algorithm

Manual Debugging

Target SW

ECU Automatic Testing

Physical Prototype
Issues

• Expensive Design Process
  - Late Integration (vertical development – single ECU)
    ▶ Early Loops are performed based upon limited information
    ▶ Late Loops are performed with more information yet they are very expensive!
  - Late Detection of Functional and Architectural errors
  - Late Fault Injection/Analysis
  - Manual Definition of Communication Protocol Configuration (K-Matrix, Bus Schedule, etc.)

• Expensive Implementation
  - Architecture is chosen (CPUs, Buses, SW Scheduling, etc) _too early and validated too late_: overly conservative (for safety critical applications) implementation
Idea

• To extend the single ECU Model Based Development Process to....

• A Distributed Model Based Development Process based upon a Virtual Integration Platform!
Development Cycle: Yesterday Tomorrow

Yesterday

- Requirements
  - Manual Validation
  - Manual Debugging
  - Target SW
    - ECU Testing
    - Physical Prototype

Today: Single ECU Model Based Development Process for Multi-ECU Systems!

- Requirements
  - Tool Assisted Validation
  - Automatic Debugging
  - Target SW
    - ECU Automatic
    - Physical Prototype Integration
  - Testing
Development Cycle: Yesterday

Yesterday

Requirements

Manual Validation

Algorithm

Manual Debugging

Target SW

Automatic Testing

Physical Prototype

Today

Tool Assisted

Requirements

Model of Distributed Control Algorithm

Algorithm

Automatic Debugging

Target SW

Automatic EC

Testing

Physical Prototype

Tomorrow

Tool Assisted

Virtual Integration

Algorithm

Automatic Debugging

Target SW

Automatic EC

Testing

Physical Prototype

Tomorrow

Zero Time Validation
Development Cycle: Yesterday  Today  Tomorrow

Yesterday

Requirements
Manual Validation
Manual Debugging
Target SW
Physical Prototype

Algorithm

Requirements
Tool Assisted
Tool Assisted
Tool Assisted
Time Validation
Design Exploration

Virtual Prototype
Optimization

Algorithm
EC
Automatic
Debugging
Automatic
Debugging
EC
Automatic

Algorithm
EC
Automatic
Testing

Physical Prototype

Tomorrow

Tomorrow: Virtual Integration Platform for Distributed Model Based Development Process

Cadence Confidential

Automotive Tiger Team
Development Cycle: Yesterday  Today  Tomorrow

Yesterday

Requirements
Manual Validation
Tool Assisted Debugging
Automatic Target SW
Automatic Physical Prototype

Algorithm
Manual Debugging
Automatic Target SW
Automatic Physical Prototype

Requirements
Tool Assisted Tool Assisted Tool Assisted
Timed Validation Design Exploration

Virtual Prototype
Algorithm
Automatic Debugging
Automatic Target SW
Automatic Physical Prototype

Algorithm

Automotive Tiger Team
Solution: Virtual Integration Platform

- Cheaper Design Process
  - Early Integration (*horizontal* development – multi ECU)
  - Early Detection of Functional and Architectural errors
  - Early Fault Injection/Analysis
  - Semi-Automatic Definition of Communication Protocol Configuration (K-Matrix, Bus Schedule, Frame Packaging, etc.)

- Cheaper Implementation
  - Architectural Alternatives (CPUs, Buses, etc) *validated upfront* even if HW is not yet available: potential ECU optimizations!
Virtual Integration is key!

Functional Network Definition and Validation (Timed and un-Timed)

Communication Protocol Adoption and Validation

Safety Concept Proof via Fault Injection

ECU SW Scheduling Adoption and Validation

ECU Optimization/Derivative Design

now

ECU₁

ECU₂

ECUₙ

ECUₖ

tomorrow

integration

specification

system design

implementation

production & after sales

ECU SW Scheduling Adoption and Validation

Communication Protocol Adoption and Validation

Safety Concept Proof via Fault Injection

ECU Optimization/Derivative Design

now

Cadence Confidential

Source: BMW

Automotive Tiger Team
Added Values

• Early Communication Protocol Validation
  - No need for cycle accurate bus models (slow simulations)
  - No usage of statistical generated traffic
  - Automatic configuration of communication matrix and bus schedule
  - Early protocol adoption and validation via fast simulations at the transaction level (bus_granted, bus_idle, etc)
  - Early bus bottleneck analysis (bus traffic)
  - Early bus fault analysis

• Early ECU Optimization/Reduction
  - via SW distribution with automatic task configuration
  - Early ECU utilization analysis (multiple ECU’s at the same time!)

• Early functional network validation in the time domain
  - Via ECU distributed architecture performance modeling (task scheduling, bus protocols…)
Added Values

• Distributed Function Safety Concept Proof via Fault Injection
  - *Early* and *Fast* in the design cycle
  - Repetitive Process: *easier* set-up
Distributed Architecture Layered Model in a Virtual Integration Platform
Automotive Supply Chain Design Roles

- Constraints
- Simple plant model
- Kernel benchmarks

System architect

System specification

Sub-function specifications

Algorithm+SW developer

Detailed plant models

Detailed ECU architectures

Abstract ECU architectures

Platform developer

ECU + RTOS + ... implementations

Sub-function implementations

System integrator

System implementation
System Architect

• Defines an overall system architecture (ECU, sensor, actuator, bus netlist), based upon specifications coming from Marketing

• Works at the most abstract level of the behavioral IP modeling, algorithm analysis, architectural design and performance analysis

• Uses existing designs and algorithm fragments (benchmarks) to analyze computation and communication requirements

• Distributes sub-functions to ECUs

• Defines the local sets of constraints based on global constraints (shape, weight, power, ins./sec, bit/sec, legal, …) performing top-level Design Space Exploration

• Considers error injection and fault tolerance
System Integrator

- Integrates the various sub-functions into a single global system functional network and the various ECU models into a single global architectural net-list
- May perform some re-mapping (function re-distribution and communication protocol adoption) for Design Space Exploration
- Validates the mapping using virtual models
- Exports to final ECU/RTOS and to protocol configuration and analysis tools
- Considers error injection & fault models
- System Integrator Design Role is similar to the System Architect Design Role: difference is only in type of models (full applications and detailed architectures vs. "computation kernels" and early functional and architectural models).
Platform designer

- Develops the ECUs, HW and SW platform elements
- Defines the ECU architecture (both implementation and performance models) starting from algorithm fragments
Algorithm Developer

• Models Behavioral/IP and analyzes algorithms
• Generates code from model automatically (not considering detailed optimization, but only estimated cost and performance)
• Performs algorithmic Design Space Exploration only, if performance estimation is available
Software Developer

- Develops and optimizes code from algorithmic (block diagram) specifications
- Uses software performance estimation
- May use functional prototyping for hardware-in-the-loop simulations
- Works concurrently with other SW developers
- Adds details to the models used by the algorithm developer
- Performs the mapping based on the platform design
- Feeds information back to algorithm designers
- May use rapid prototyping to improve model performance for optimization
- Performs Design Space Exploration at the ECU (or handful of ECUs) level

Algorithm developer and software developer today belong to different teams, but will tend to merge as model-based software generation expands.
Methodology

Design Roles

The Cadence Position: The Design Roles for Automotive Electronic System Design

- Constraints
- System overview
- System architecture
- Design of System Architecture
- Requirements
- Detailed part models
- ECU architecture
- Detailed ECU architectures
- System integration
- Validation of System Architecture
- System Implementation

Methodology

Business Protocol Configuration

Cadence Confidential
System Integrator
Tools / Design Flow

Design Roles

Methodology

Requirements

Tools
Tools / Design Flow

Requirements
Tools / Design Flow: Requirements
Distributed Function

ML

Algorithm

Algorithm

Algorithm

Bus Protocol Configuration
Tools / Design Flow: Requirements Distributed Function

ETAS AscetSD → Validation
Mathworks Simulink → Validation
C → Validation

ML

Bus Protocol Configuration
Tools / Design Flow: Architectural Modeling

- ETAS Ascet\textsuperscript{SD}
- Mathworks\textsuperscript{Simulink}
- \textsuperscript{ML}

- Automatic Import
- Zero Time Validation
- Model of Distributed Control Algorithm
- Zero Time Domain
- Architectural Models
- Bus Protocol Configuration

Cadence Confidential

Automotive Tiger Team
Tools / Design Flow: SW Distribution w/ Automatic Task Allocation

ETAS Ascet\(\text{SD}^{\text{TM}}\)
Mathworks\(\text{Simulink}\)
Cadence\(\text{\textregistered}\)

Timed Validation
Design Exploration

Virtual Prototype Time Domain

Bus Protocol Configuration
Tools / Design Flow: SW (re)-Distribution and Optimization
Tools / Design Flow: Protocol Configuration Exploration
Tools / Design Flow: Protocol Configuration Exploration
Tools / Design Flow: Protocol Configuration Exploration

ETAS Ascet\wedge SD → \text{Automatic Import TIP}

Mathworks\wedge Simulink → \text{Automatic Import RTW, dSpace TL}

C\wedge C

Message Packaging in Frames

Frame 1

Frame 2

FlexRay

Bus Protocol Configuration

Cadence Confidential

Automotive Tiger Team
Tools / Design Flow: Fault Injection/Analysis
Tools in the Design Flow

Requirements

Validated Subsystem Virtual Prototype
Tools/Design Flow: Distributed System Aware ECU SW RT Validation

ETAS AcselSD
Mathworks Simulink
cC
Validated SW Distribution and Chosen Protocol Configuration

Frame 1
Frame 1

dSace TL
ETAS ASCET SD

Automatic Import
Automatic Import
Automatic Import

Plant Model

Bus Protocol Configuration

Cadence Confidential

Automotive Tiger Team
Tools/Design Flow: Distributed System Aware ECU SW RT Validation
Tools/Design Flow: *Distributed System Aware ECU SW RT Validation*

- ETAS Ascet®SD
- Mathworks® Simulink
- Cadence®

**Automatic Import**
- ETAS Ascet®SD
- Mathworks® Simulink
- Cadence®

**Frame 1**
- FlexRay
- Validated SW Distribution and Chosen Protocol Configuration

**Automatic Export of SW Task Configuration**
- ETAS ASCET®SD
- Mathworks® Simulink

**Plant I/O**
- SW Debugger
- Cycle Accurate Timing Estimator

**Cadence Confidential**
Tools/Design Flow: *Distributed System Aware ECU SW RT Validation*

- **ETAS Ascet/SD**: Automatic Import (TIP)
- **Mathworks Simulink**: Automatic Import (RTW, dSpace TL)
- **ML**: Export of RT Constraints
- **dSpace TL**: Automatic Code Generation of Instrumented Application SW
- **ETAS ASCET/SD**: SW Validation vs. RT Constraints, Device Driver Development and validation
- **SW Debugger**: Cycle Accurate Timing Estimator
- **Plant Model**: RestBus Traffic Dynamic or Static Model

**Cadence Confidential**
Tools/Design Flow: Distributed System Aware ECU SW RT Validation
Tools/Design Flow: Distributed System Aware ECU SW RT Validated

ETAS Ascet\textregistered SD
Automatic Import TIP
Plant Model

Mathworks\textregistered Simulink
Automatic Import RTW\textregistered .dace TL

Validated SW Distribution and Chosen Protocol Configuration

ML

EC
Target Code
Validated vs.\textregistered RT
Constraints
Device drivers
Developed and Validated

EC
Target Code
Validated vs.\textregistered RT
Constraints
Device drivers
Developed and Validated

Cadence Confidential

Automotive Tiger Team
Tools/Design Flow: Protocol Configuration Validated

- ETAS Ascet®SD
- Mathworks® Simulink
- C++

Plant Model

FlexRay

Validated SW Distribution and Chosen Protocol Configuration

Bus Protocol Configuration Validated

Excerpt of Comm Layer to ECs
CP and Bus controllers

EC
Target Code Validated vs. RT Requirements
Device drivers Developed and Validated

Cadence Confidential

Automotive Tiger Team
Tools/Design Flow

Requirements

Subsystem Validated Virtual Prototype

Subsystem Physical Prototype Ready For Testing
ECU Application SW Development: 2-Step New Flow

Application SW RT Constraint validation with usage of off-line bus trace

Plant Model for EC\□ sensors □ actuators Mathworks □ Simulink

CP □ Instruction Set Simulator

Peripherals Models

SW Debugger

Bus Traffic Static Model □ Input Trace from off-line Simulation □
Added Value

- Application SW is developed taking into account close-to-real (not statistical) bus traffic to the ECU (static model generated off-line via simulation)
- Models the rest of the distributed platform using high level and fast processor models (VPM)
- SW debugger/ISS provide same programming model as target: good trade-off between accuracy and speed (30MIPS – 100 MIPS)
- Code for ECU3 can be generated from Mathworks/Simulink specification automatically (via RTW, dSPACE TargetLink) for target
- Code for the simulation models for the rest of the platform still generated automatically from Mathworks/Simulink and/or ASCET-SD