Developing & Testing Automotive Software on Multi-SoC ECU Architectures using Virtual Prototyping

Sam Tennent
Synopsys
Agenda

• Automotive Trend Towards *less* ECUs/*more* Integration
• MCU SoC RTOS and Applications
• Simple ADAS Demonstration
• Demonstration Video and Results.
• Summary
Consumers who arrived in Las Vegas for the 2017 Consumer Electronics Show—one of the premiere exhibitions of new technologies for the general public—might have wondered if they were at an auto show.

www.mckinsey.com
Less is More!

Less ECUs... More Integration

• Demands for more complex and more powerful features in Automotive is rapidly driving technology.
  - **Infotainment**: Connectivity, SWOTA and Security
  - **ADAS**: autonomous driving, sensor fusion.
  - **Gateway**: High bandwidth traffic routing and consolidation

• These accelerating demands are leading to:
  - Large compute core clusters.
  - Integrated MCU domains with Compute Clusters.
  - Integrated ‘smart’ communication gateways
  - Unprecedented challenges in Automotive Software Development.

© Accellera Systems Initiative
SW Development and Test Challenges

Unprecedented Solutions Required

• **Software Development and Verification**
  – Debugging of high complexity multicore problems.
  – Difficult to expose and detect underlying problems when functionality is correct.
    • Software performance, driver setup errors……

• **System Integration with Tool and Hardware Test Ecosystem**
  – HIL testing is on the critical path
  – Complex and costly system level verification.

• **Functional Safety Testing.**
  – More software, More features, More tests.
  – Need to cover more to increase Software quality (ISO26262)

• **Communications Verification.**
  – Large Scale High Bandwidth Multi-Protocol Verification.
  – CAN, LIN, ETHERNET, SPI, PCIe, FLEXRAY, I2C

© Accellera Systems Initiative 2018
MCU/SOC Real-time Operating Systems

AUTOSAR + Linux – Demand Real Time Performance

• Automotive chips with defined core clusters and domains.
  – Multiple OS on same silicon – Linux, AUTOSAR, Qnx etc.
  – Application specific domains with hypervisors.

• Typical MCUs (or MCU domains) still need to .
  – Meet strict timing requirements for sensor and I/O servicing.

• AUTOSAR is built on the OSEK/VDX OS specification
  – Predictable and precise scheduling.
  – Still the dominating choice for Timing and Safety Critical

• Mixed Linux and AUTOSAR clusters commonplace.

• Linux dominating choice for compute intensive apps.
ECU OS Application Development Challenges

OS Scheduling is typically not high visibility

- Multicore applications need careful mapping to maximize performance
  - How can we validate these requirements?
- High bandwidth multicore resource access is crucial for high performance
  - How can that be verified under a changing SW load?
- Changing Automotive standards and versions is ongoing.
  - Re-verification of AUTOSAR versions is a big effort.
- On-Chip, multi domain silicon (AUTOSAR + Linux etc) with hypervisors.
  - Requires integration tasks previously not seen on previous generation chips.
- High degrees of visibility into the OS is required to satisfy these challenges.
Virtualizer Development Kit

**VDK - More than just simulation models**

- **SW Development**
- **Functional Safety** (Fault and Coverage Testing)
- **ECU System Integration and Testing (vHIL)**

**Tools for VDK**
- Platform level debug
- Virtual MCU
- Interfaces to 3rd party tools for VDK
- Tools for VDK console, Platform level debug, SW analysis

**Synchronized Co-Simulation with**
- Simulink
- CANoe
- SABER

**Virtualizer Tools**
- Powerful Scripting Frameworks for complex Debugging, Analysis, Fault Injection and Stimulus
Embedded Software Code Coverage

Functional Safety and Coverage Based Fault Injection
SW Verification and Test Challenges

ISO26262 guidelines are challenging for large scale projects

• Software Quality is key for Automotive products.
• Challenges with Safety Management

⚠️ • Specified Failures (handled)
  – Failures that can happen and have specified hardware behaviour.
  – Communications errors, power or clock tree faults

⚠️ • Unexpected Failures (not handled)
  – Failures not expected to happen and unspecified hardware behaviour.
  – Driver set-up errors, open/short circuit, transients, EMC

• Meeting these challenges with quantifiable SW coverage metrics is key to qualifying SW quality with safety critical systems.
Virtual Prototypes for Functional Safety

ISO26262 guidelines are challenging for large scale projects

• Virtual Prototypes provide a framework for advanced fault-injection (Simulation Probes)

• Simulation Probes used to influence the HW from outside the SW.

• Virtual Prototypes can be used to make testing more cost-effective through code-coverage measurements

• Fault Injection testing can be automated and measured during regressions

• May be used as testing evidence for certification
Code Coverage Overview

*Functional Safety Testing made more efficient*

- eSW Code coverage helps achieving cost-effective testing, i.e. same result with fewer tests (eliminate redundant tests)

- What to measure?
  - Function Coverage: Has each function in SW been called?
  - Call Coverage: Has each different function been covered once?
  - Statement Coverage: Has each statement in the SW been executed?
  - Branch Coverage: Has each possible branch been taken?
  - Decision coverage: Has every decision taken all possible outcomes at least once?
  - Condition coverage: Has each Boolean sub-expression evaluated both true and false?
  - Modified Condition/Decision Coverage (MCDC)

Supported natively in Virtualizer
Supported by tool integration (Tessy, T32)
Coverage Based Fault Injection Flow

*Functional Safety Testing made easier*

• Flow
  – **Gather** Coverage Metrics
  – Use **Fault Injection** and **Stimulus** Generation to fill the gaps
  – **Re-run** Coverage Metrics and **Re-evaluate**

• Can be used for ‘Hard to Test’ Scenarios
  – Signal Integrity Problems - Transients, “Stuck At” issues
  – Damaging Power based faults.
  – Test cascading effects applicable from driver to application.

• Highly complimentary flow to ISO26262 guidelines.
• Let Coverage metrics tell you what needs to be covered
• Use Simulation Probes to add Coverage and increase quality
ADAS Feature Demonstration

Simple ADAS Demonstration on System ECUs
(SoC + Gateway) + MCU
Simple ADAS Feature Demonstration

• Cruise Control (and the more involved Adaptive Cruise Control) is a good example of a simple ADAS feature.
  – **Cruise Control** Ability to maintain speed at a user defined level considering the effects of the environment.
  – **Adaptive Cruise Control**: Ability to maintain speed and distance from surrounding vehicles.

We will look at an example interaction of multiple ECUs in delivering this feature.
ADAS, Compute, Control and the ECU

High Bandwidth Sensors
Vision, Radar, MMI

Rest Of System
Engine Management
Sensors/Actuators

Highly Integrated Compute + Control

infotainment
COMPUTE
ADAS
chassis
MCU
Gateway
powertrain
System ECU Hardware Example

Simple ADAS System ECU Demonstration

• Automatic Gearbox SW implementation
  – Control the gearbox in ALL modes, based on current engine state

• Simple Cruise Control Override
  – Override the manual throttle but accept increments and decrements to the chosen speed.

• Adaptive Cruise Control
  – Test MCU responsiveness in a tracking scenario of forward vehicle.

• Simple Automatic Avoidance Measures
  – Emergency Stop, Safe Stop etc.

• SoC running Linux with Simple Drive Cycle Test Application
• Gateway MCU coordinating CAN/PCIE communication with SoC and MCU.
• MCU running AUTOSAR variant.
System vHIL Simulation

- ARM SoC
- Linaro
- NXP Gateway
- PCIE
- Controller ECU
- Infineon MCU
- CAN Bus
- Compute and Comms ECU
- Gear, Throttle, Brake
- Rpm Speed, Foot Throttle

© Accellera Systems Initiative
SoC DriveApp

Communication with Gateway

• Simple Linux Application running on ARMv7 CPU_SS – DriveApp
• Communicates with Gateway MCU through PCIE in the ARMv7 design (Linux Console App)
Gateway Application

S32 Processing Platform

S32 System DRAM mapped to PCIe RC BAR0

Request Block
Command from MCU

System Info Block
- GearDrive ID: 0x100
- Engine_Rpm ID: 0x123
- Engine_Throttle ID: 0x101
- Speed ID: 0x102
- Ctrl_Status ID: 0x234
- Brake_State ID: 0x543
- Lights ID: 0x321
- Command ID: 0x555
- Dist_Front_Vehicle ID: 0x710
- Trip_Distance ID: 0x720
- Req_Cruise_Speed ID: 0x718
- Trip_Duration ID: 0x730

Can Bus to System ECUs

Connects to PCIe RC Device in ARM SoC
## Gateway Software Tasks

<table>
<thead>
<tr>
<th>Task Func.</th>
<th>OS Task Schedule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS_TASK_10MS</td>
<td>10 ms</td>
<td>Update PCIe System Data Structure with CAN data</td>
</tr>
<tr>
<td>OS_TASK_20MS</td>
<td>20 ms</td>
<td>Get Trip Speed, Trip Time and Trip Distance</td>
</tr>
<tr>
<td>OS_TASK_50MS</td>
<td>50 ms</td>
<td>Send CAN messages</td>
</tr>
<tr>
<td>OS_TASK_100MS</td>
<td>100 ms</td>
<td>Get the command data from the MCU</td>
</tr>
<tr>
<td>OS_TASK_200MS</td>
<td>200 ms</td>
<td>Calculate Trip Distance Travelled</td>
</tr>
</tbody>
</table>
## MCU Multicore AUTOSAR Task Mapping

<table>
<thead>
<tr>
<th>Task Func.</th>
<th>OS Task Schedule</th>
<th>CPU ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_sensors</td>
<td>10 ms</td>
<td>CPU0</td>
<td>Reading Sensor Interface Data</td>
</tr>
<tr>
<td>task_gearbox</td>
<td>50 ms</td>
<td>CPU2</td>
<td>Controls Gearbox based on Engine State</td>
</tr>
<tr>
<td>task_drive_modes</td>
<td>80 ms</td>
<td>CPU2</td>
<td>Processes User Modes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Modes are Manual, SafeStop, EmergencyStop, Cruise Control.</td>
</tr>
<tr>
<td>task_transmit_can</td>
<td>20 ms</td>
<td>CPU1</td>
<td>Writing Engine State to CAN bus</td>
</tr>
<tr>
<td>task_output_drive</td>
<td>30 ms</td>
<td>CPU0</td>
<td>Updating MCU Engine state to the System</td>
</tr>
<tr>
<td>task_comms</td>
<td>100 ms</td>
<td>CPU1</td>
<td>Reading user commands Sent to the MCU</td>
</tr>
</tbody>
</table>

**CPU0 = Master  CPU1 = Slave  CPU2 = Slave**

Basic Tasks and Background Tasks not shown.
## MCU CAN BUS System Interface Spec.

<table>
<thead>
<tr>
<th>Message Name</th>
<th>ID</th>
<th>Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCU_GearDrive</td>
<td>0x100</td>
<td>Reserved, GearDrive</td>
</tr>
<tr>
<td>MCU_Engine_Rpm</td>
<td>0x123</td>
<td>Reserved, Rpm</td>
</tr>
<tr>
<td>MCU_Engine_Throttle</td>
<td>0x101</td>
<td>Reserved, Throttle, Foot Throttle</td>
</tr>
<tr>
<td>MCU_Speed</td>
<td>0x102</td>
<td>Reserved, Speed</td>
</tr>
<tr>
<td>MCU_Ctrl_Status</td>
<td>0x234</td>
<td>Reserved, Requested Cruise Speed, Autopilot, LastDrive Mode, DriveMode</td>
</tr>
<tr>
<td>MCU_Brake_State</td>
<td>0x543</td>
<td>Reserved, Foot BrakeTorque, BrakeTorque</td>
</tr>
</tbody>
</table>
Virtualizer VDK vHIL
(Infineon + NXP + ARM v7)

Linux eSW Dev (Linaro Linux)
eSW Build Tools (OSEK/VDX)
Vector CANoe
Simulink Plant Model
Synopsys Virtualizer
Scripting

© Accellera Systems Initiative 25
Demonstration

• **SoC Functionality**
  – Application Behaviour
  – Drive Cycle Mode Testing
  – eSW Debugging

• **Gateway Functionality**
  – Communications Visualisation
  – eSW Debugging

• **MCU Functionality**
  – Ecosystem Tools Connectivity (Simulink/Vector CANoe)
  – AUTOSAR OS visualization.

• **Results, Analysis and Debugging**
Demonstration Profile

Adaptive Cruise Control Demo Profile.

Manual

Track Speed

Increase Speed

Decrease Speed

Decrease Speed

SoC “DriveApp” Console Interface

7. Auto Drive Mode Adaptive will duplicate the scenario pictured.
Video One

System Overview & Run Application
Demonstration Test Profile Scenario Summary.

Foot Throttle

Speed

Rpm

Foot Brake

Cruise Decrease

Cruise Started

Cruise Increase

McuGear

McuBrake

McuThrottle

MCU Brake is applied

Throttle demand to increase speed
MCU Focused Results Analysis

• **Hardware Tracing**
  – MCU interface Boundary Traces.
    • Analog sampled inputs
    • CAN Bus Data Verification

• **Software Tracing**
  – Function Tracing
  – AUTOSAR
    • Task Visualisation
    • System Calls
    • Errors

© Accellera Systems Initiative
Video Two

Software and Hardware Analysis of MCU operation
AUTOSAR Instrumentation Summary
OSEK visualization example with Simulation Probes Scripts

Task Switching
OS Services
Task States
OS Errors
Task Summaries

and more.. ISR2 tracking, Task Stack utilization, custom debugging and application analysis, ....

Selectable and customizable visualization
SOC - MCU System Visibility

“End to End” Debugging

• SOC System Visibility
  – Tracking Activity from SW Function to Hardware Command
  – Debugging the Serial Link Driver in Linux
  – System Visibility and System Latency
Video Three

Software and Hardware Analysis of SoC operation
Demo CAN Bus Message Visualization Summary

Can Bus Packet Specification input, allows message and fields to be extracted and visualized
Coverage Based Fault Injection

*Practical walkthrough.*

- Coverage Metrics and Scripting can be used to uncover missing test coverage.

- The file `sensors.c`, has some large transient signal detection and correction logic.

```c
if (NewSample >> OldSample) {
    discard NewSample;
} else {
    store NewSample;
}
```

- We are **not covering** the code which handles such transients.

- Generating and testing such code can be difficult on the real hardware.
Coverage Based Fault Injection

• Using Simulation Probes we inject a transient into the MCU and re-evaluate.
Demonstration Summary

Four Main Use Case Overviews

• Development and Verification
  – Virtualizer Software Analysis Tools
  – Debugging with 3rdparty HLL debuggers

• Application Verification
  – SoC application with vHIL (Simulink, CANoe)

• OS
  – Porting and Verification
  – Driver development and Testing
  – Visualization and Debugging.

• Functional Safety Testing
  – Coverage Based Fault Injection
  – SW Test and Quality Metrics
Session Summary

• In the accelerating Automotive Industry
  – Software is key.
  – Software will always be on the Critical Path.

• Complexity growth in HW architecture and SW is exponential.

• For early development or post silicon use Cases
  – New challenges are pushing the boundaries of traditional approaches.

• Virtual Prototyping and VDKs can play a big part in:
  – Accelerating Development.
  – Increasing Product Quality and Functional Safety.
Questions