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

Sam Tennent

Synopsys

SYNOPSYS®
Silicon to Software™

accellera
SYSTEMS INITIATIVE

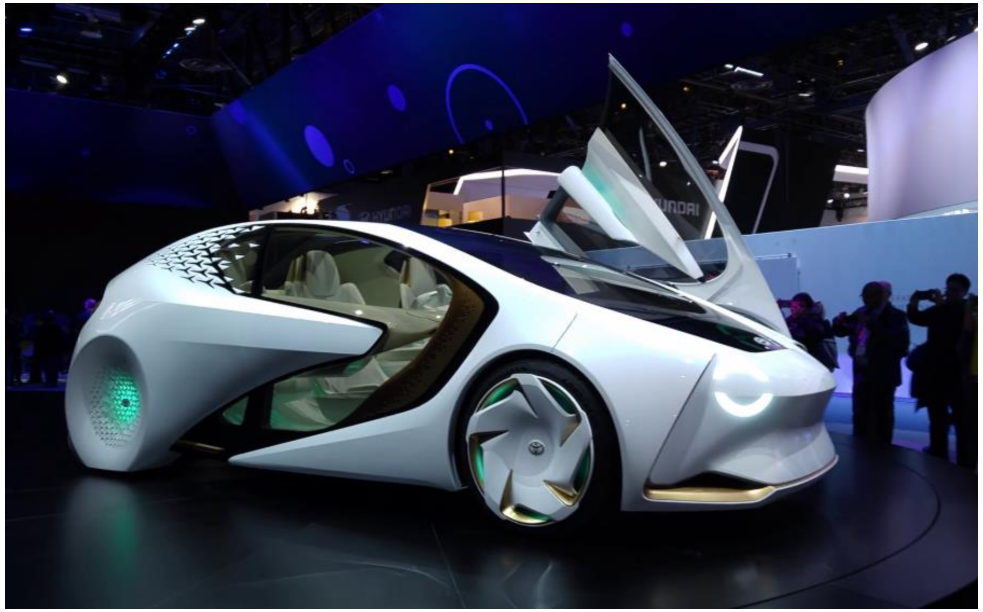
2018
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
EUROPE

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.

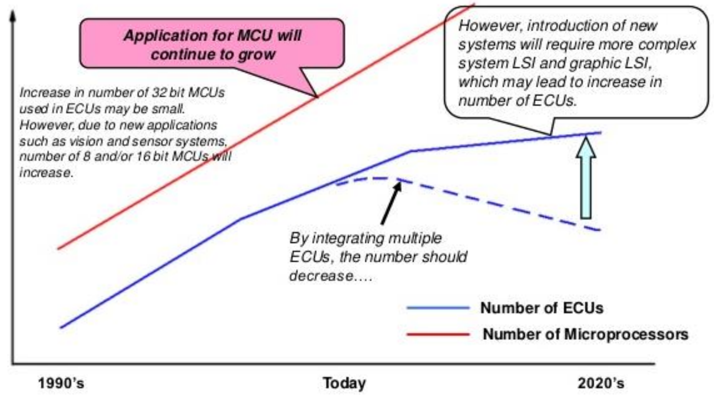
www.mckinsey.com



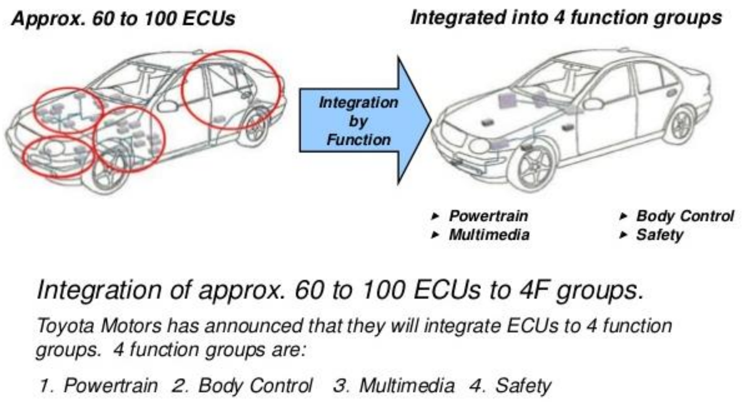
Less is More!

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
  - **Powertrain**, Chassis: Hybrid, Electric, Integrated need for timing critical responses.
- **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.



Nikkei Automotive Technology



Toyota Motor Company

SW Development and Test Challenges

Unprecedented Solutions Required

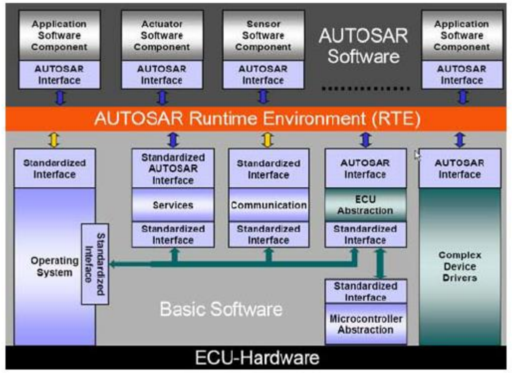
- **Software Development and Verification**
  - Debugging of high complexity multicore problems.
  - Difficult to expose and detect underlying problems when functionality is correct.
    - Software performance, driver setup errors……
- **System Integration with Tool and Hardware Test Ecosystem**
  - HIL 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

MCU/SOC Real-time Operating Systems

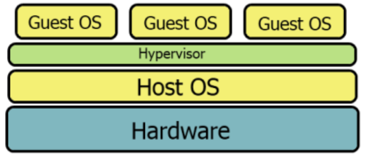
AUTOSAR + Linux – Demand Real Time Performance

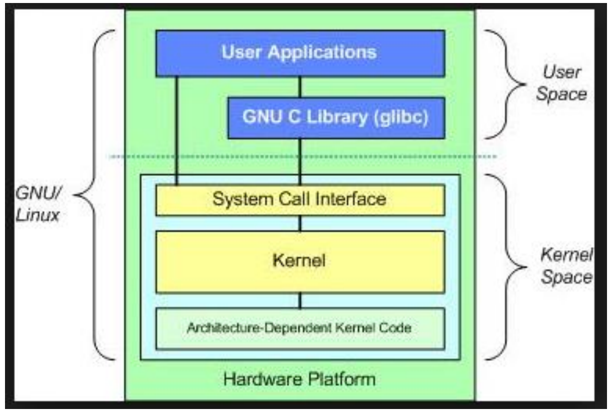
- **Automotive chips with defined core clusters and domains.**
  - Multiple OS on same silicon – Linux, AUTOSAR, Qnx etc.
  - Application specific domains with hypervisors.
- **Typical MCUs (or MCU domains) still need to..**
  - Meet strict timing requirements for sensor and I/O servicing.
- **AUTOSAR 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.**

Autosar.org



AUTOSAR





Linux

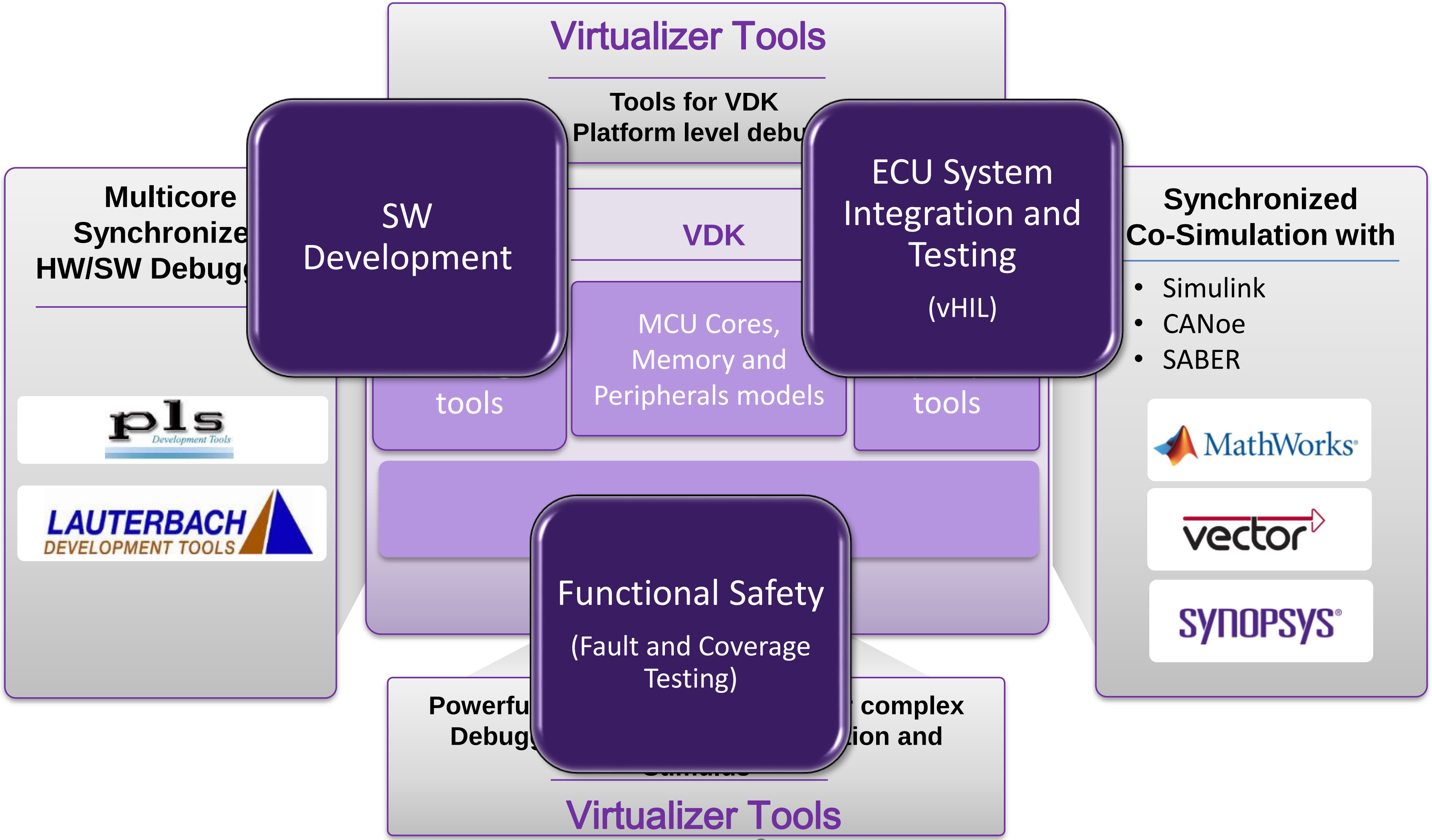
ECU OS Application Development Challenges

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



Embedded Software Code Coverage

Functional Safety and Coverage Based Fault Injection

SW Verification and Test Challenges

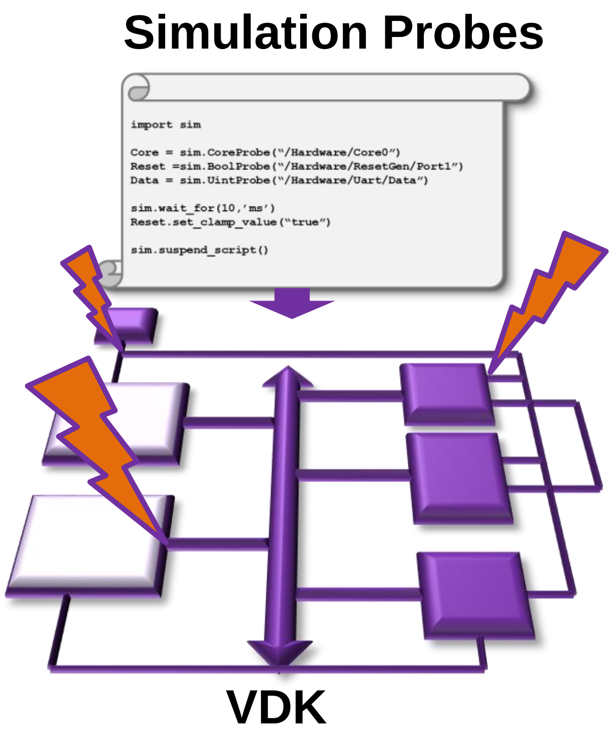
ISO26262 guidelines are challenging for large scale projects

- **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)**

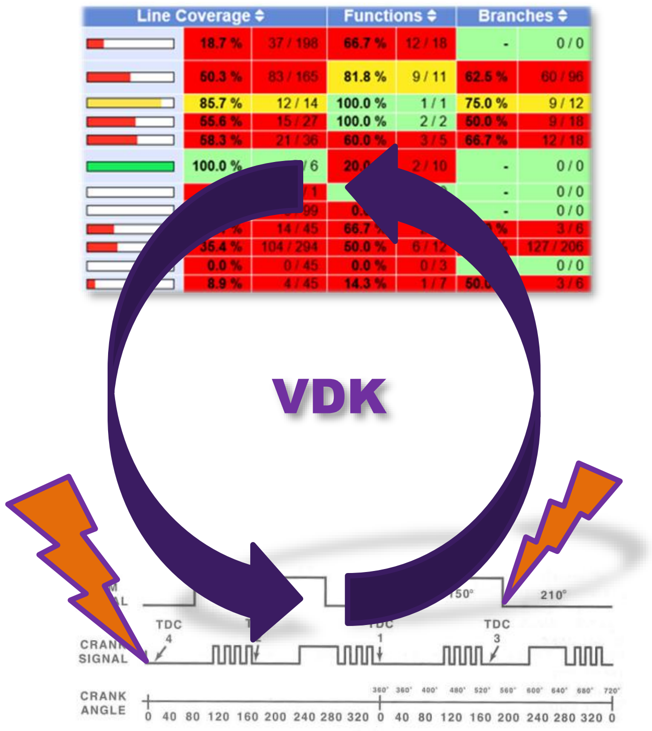
Supported natively in Virtualizer

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**

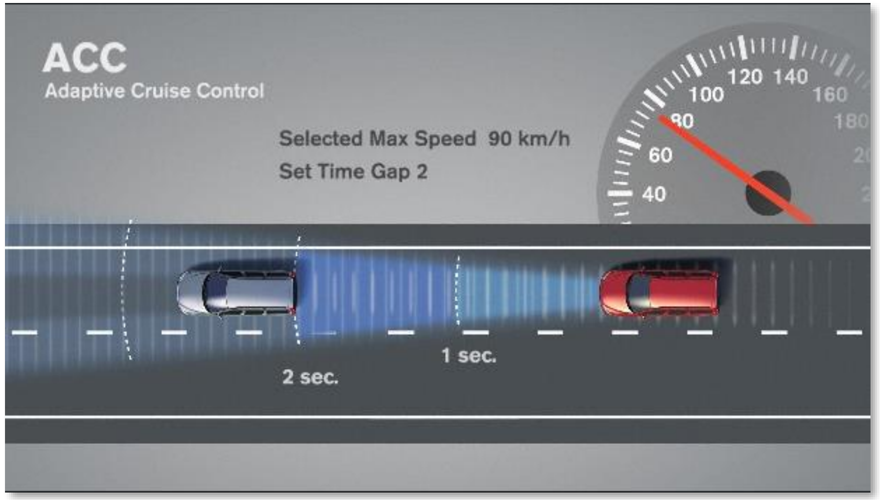


ADAS Feature Demonstration

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

Simple ADAS Feature Demonstration

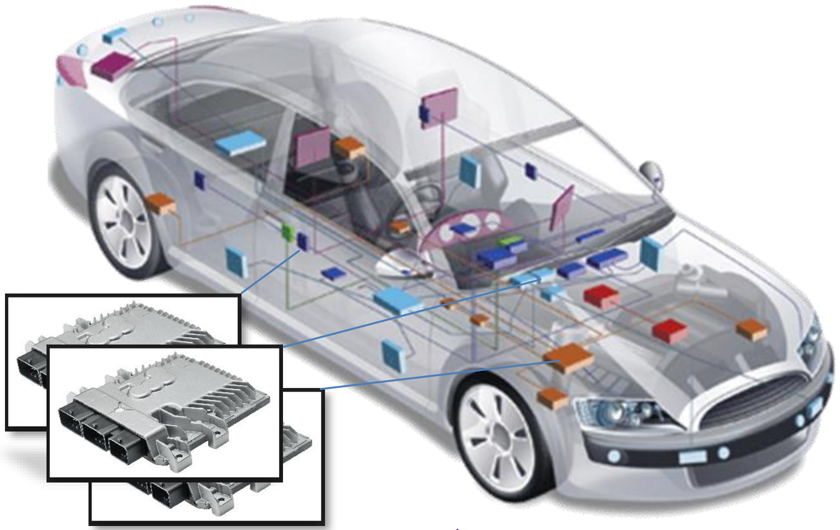
Driverless.global

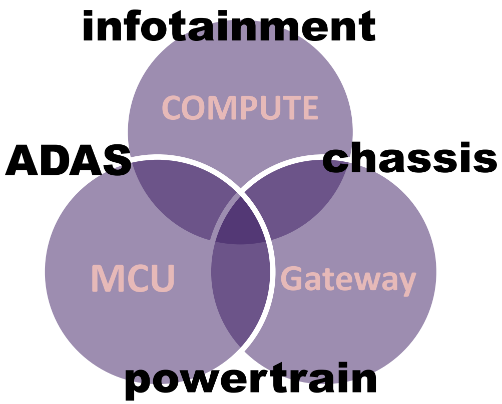


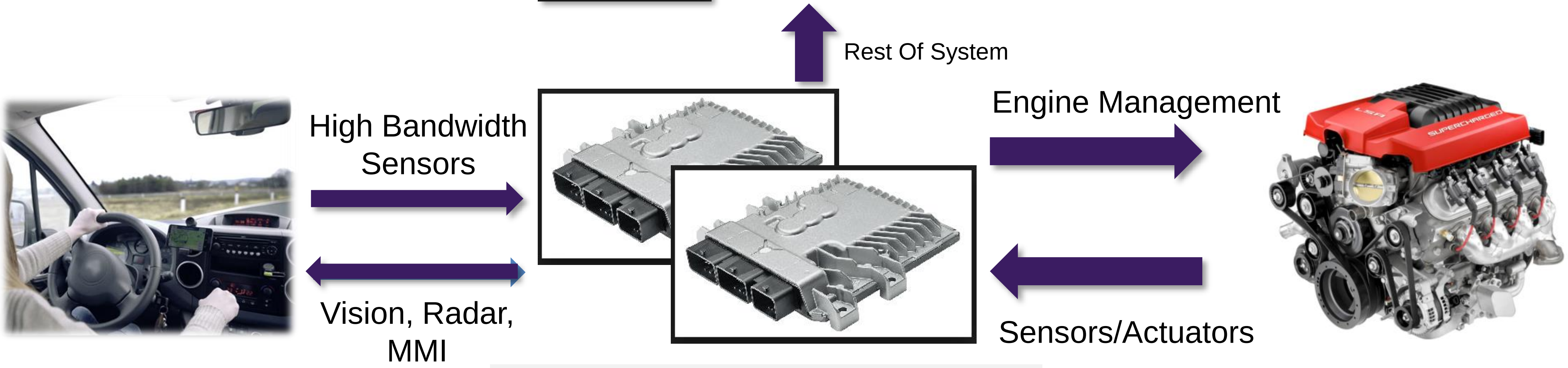
- **Cruise Control (and the more involved Adaptive Cruise Control) is a good example of a simple ADAS feature.**
  - **Cruise Control** Ability to maintain speed at a user defined level considering the effects of the environment.
  - **Adaptive Cruise Control**: Ability to maintain speed and distance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

We will look at an example interaction of multiple ECUs in delivering this feature.

ADAS, Compute, Control and the ECU



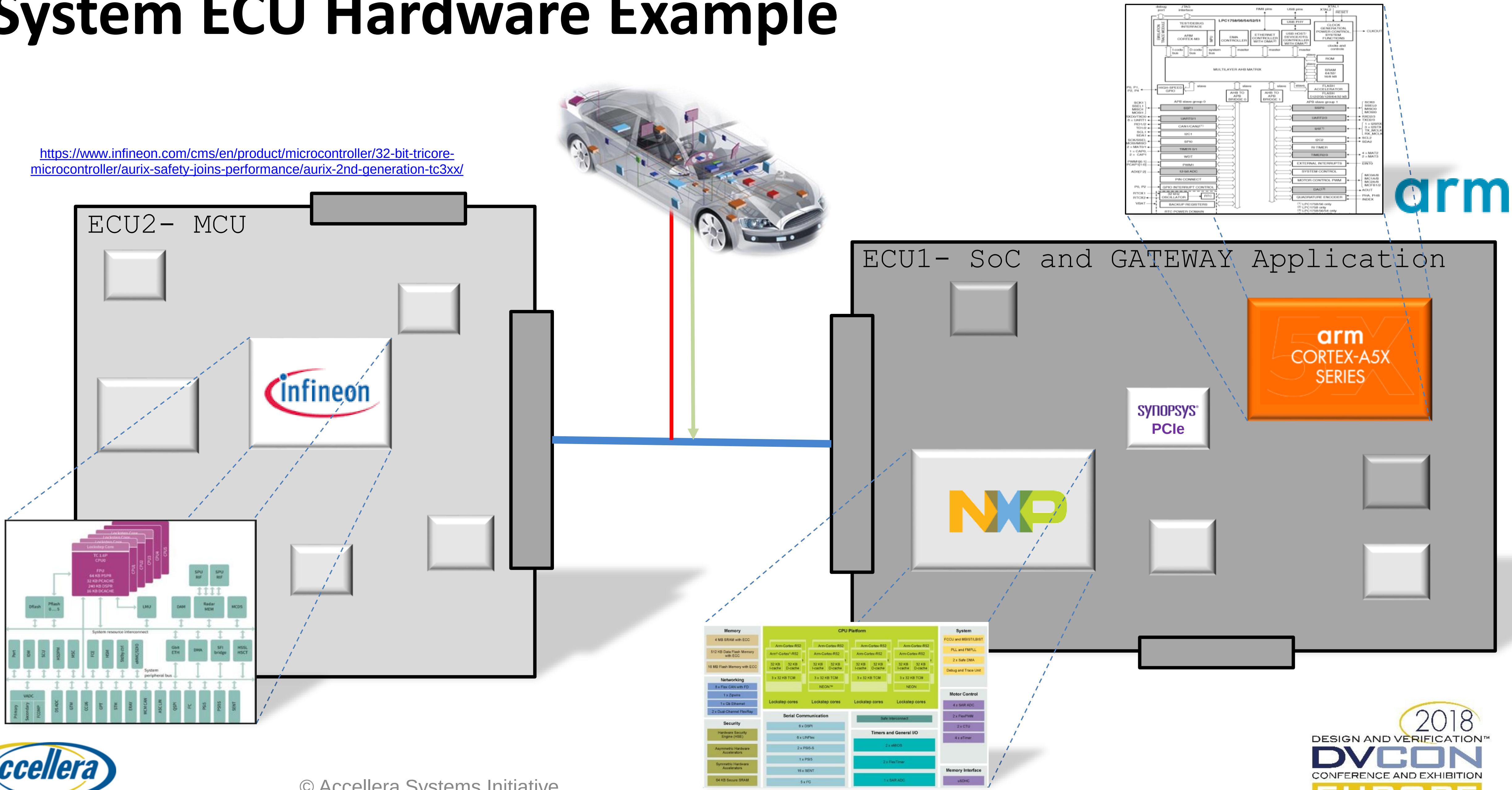




Highly Integrated Compute + Control

System ECU Hardware Example

https://www.infineon.com/cms/en/product/microcontroller/32-bit-tricore-microcontroller/aurix-safety-joins-performance/aurix-2nd-generation-tc3xx/

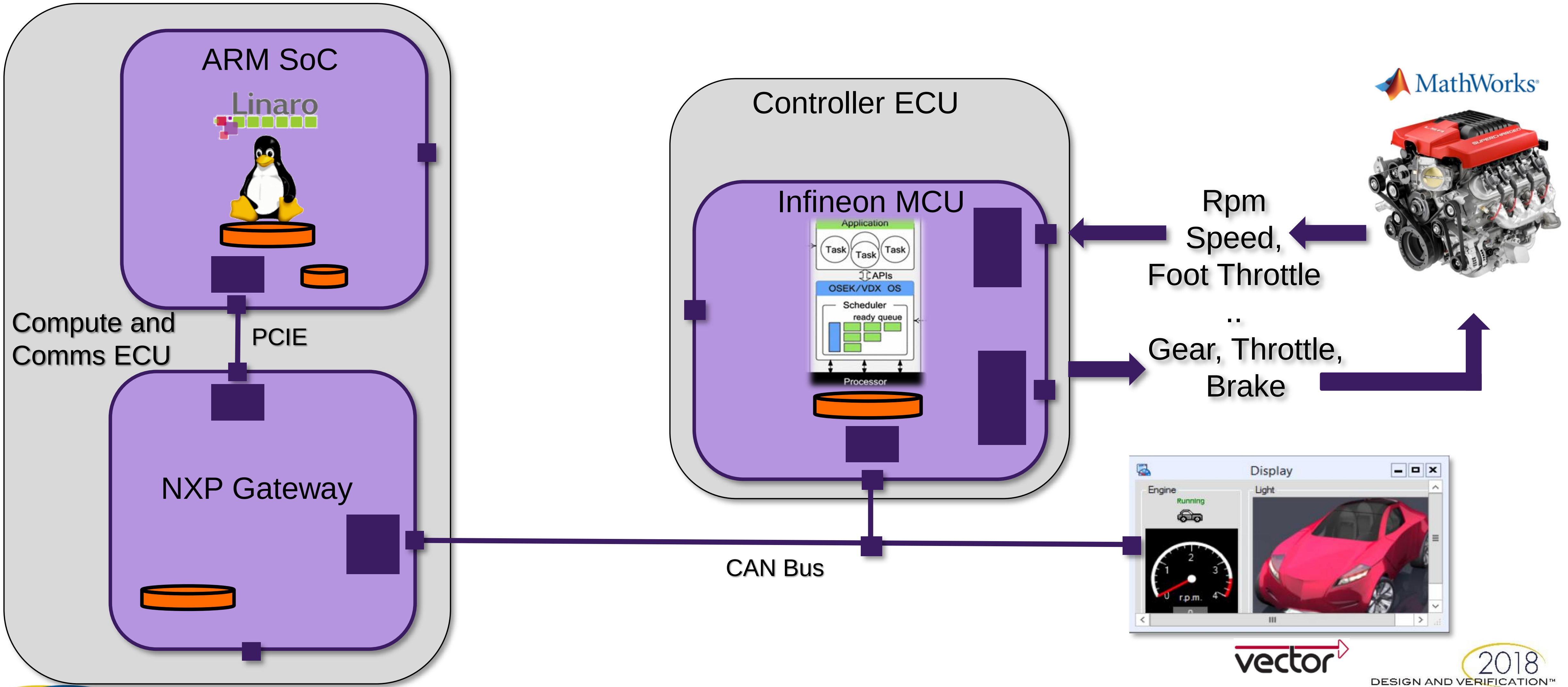


2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

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

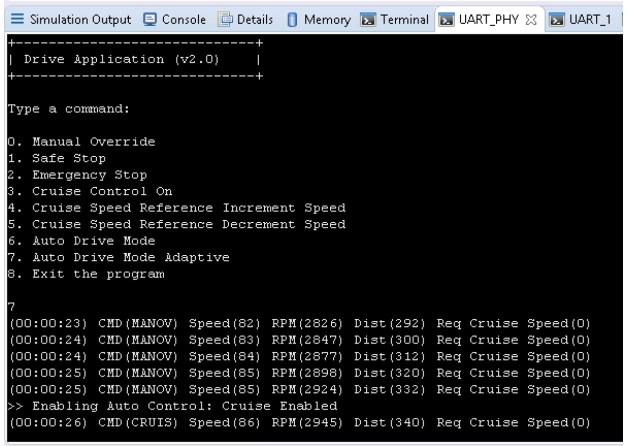


SoC DriveApp

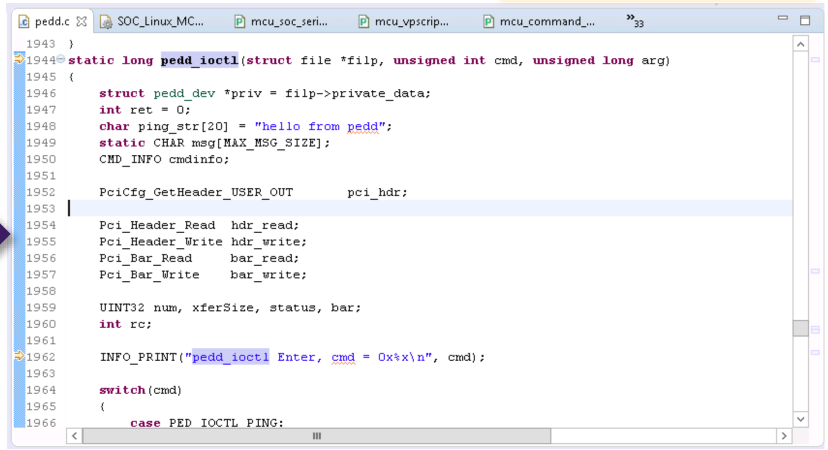
Communication with Gateway

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

PCIE Driver



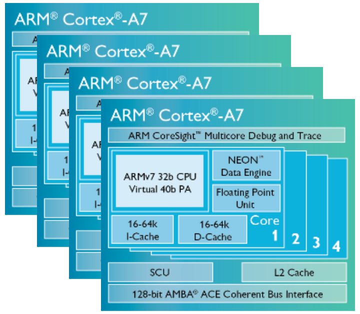
DriveApp



PCIE Root Complex

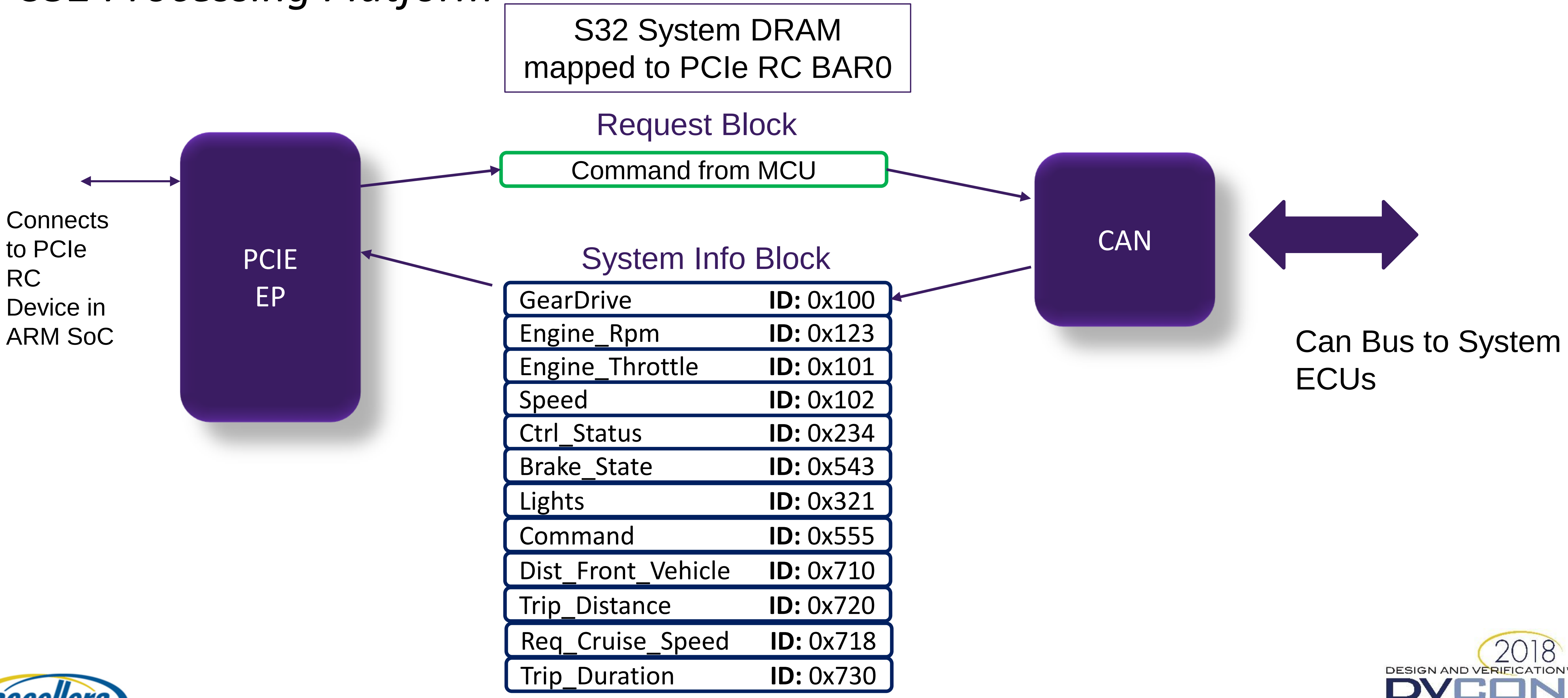
Connects to PCIe EP Device in Gateway

ARMv7 SP CPU_SS



Gateway Application

S32 Processing Platform



Gateway Software Tasks

Task Func.	OS Task Schedule	Description
OS_TASK_10MS	10 ms	Update PCIe System Data Structure with CAN data
OS_TASK_20MS	20 ms	Get Trip Speed , Trip Time and Trip Distance
OS_TASK_50MS	50 ms	Send CAN messages
OS_TASK_100MS	100 ms	Get the command data from the MCU
OS_TASK_200MS	200 ms	Calculate Trip Distance Travelled

MCU Multicore AUTOSAR Task Mapping

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

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

Basic Tasks and Background Tasks not shown.

MCU CAN BUS System Interface Spec.

MCU_GearDrive **ID:** 0x100

Reserved	GearDrive
63	8 … 0

MCU_Engine_Rpm **ID:** 0x123

Reserved	Rpm
63	16 … 0

MCU_Engine_Throttle **ID:** 0x101

Reserved		Foot Throttle	Throttle
63	48	32	16 … 0

MCU_Speed **ID:** 0x102

Reserved	Speed
63	16 … 0

MCU_Ctrl_Status **ID:** 0x234

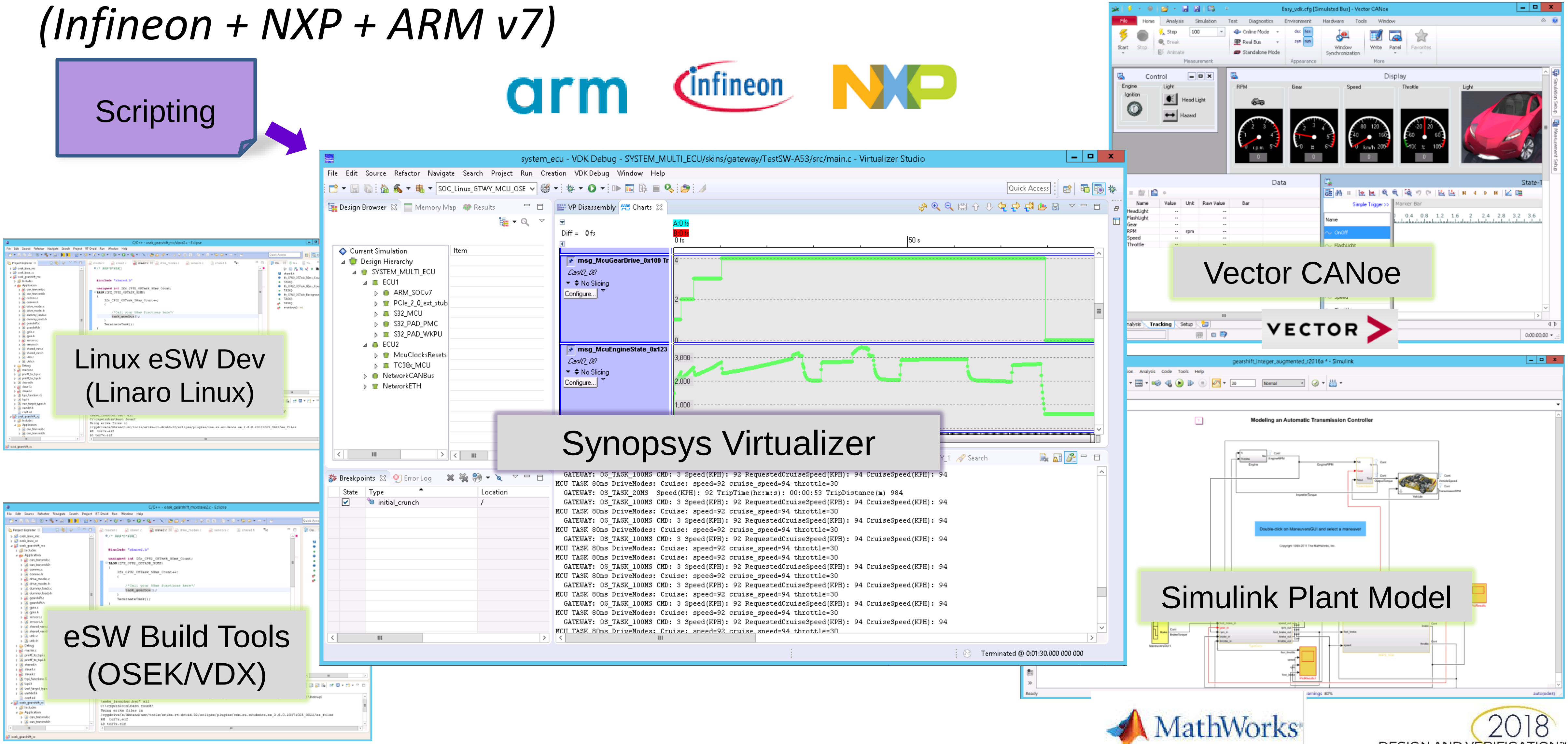
Reserved	Requested Cruise Speed	Autopilot	LastDrive Mode	DriveMode
63		17	16	8 … 0

MCU_Brake_State **ID:** 0x543

Reserved	Reserved	Foot BrakeTorque	BrakeTorque
63	48	32	16 … 0

Virtualizer VDK vHIL

(Infineon + NXP + ARM v7)

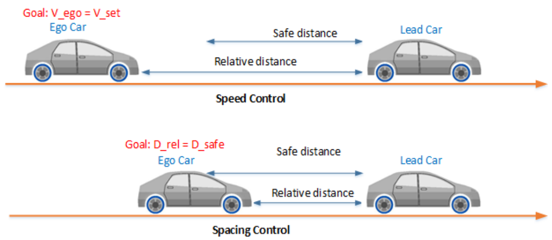


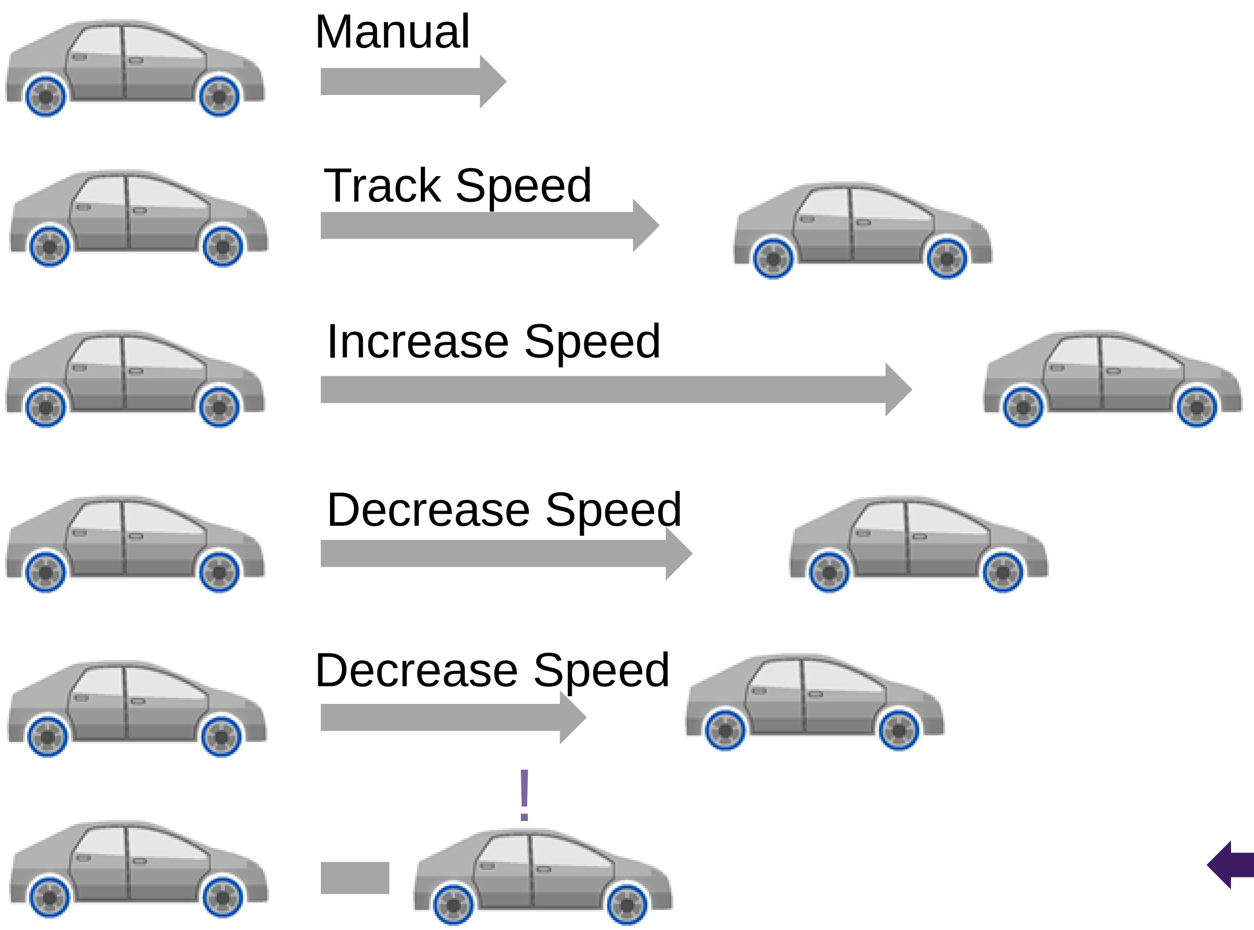
Demonstration

- **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**

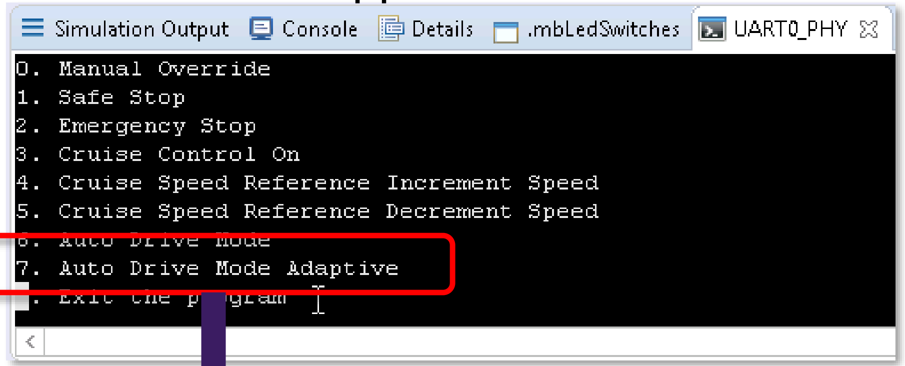
Demonstration Profile

Adaptive Cruise Control Demo Profile.





SoC "DriveApp" Console Interface

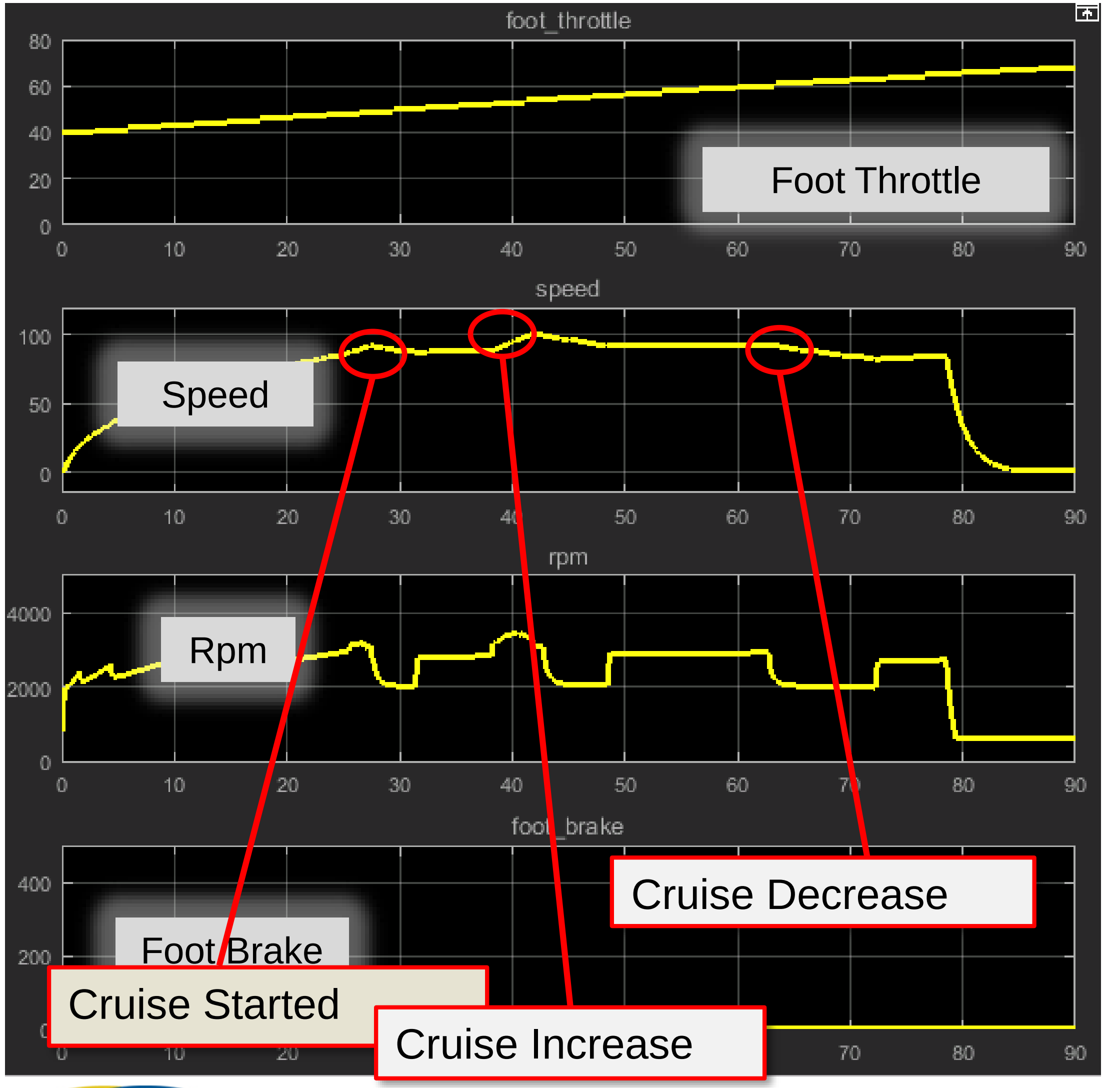


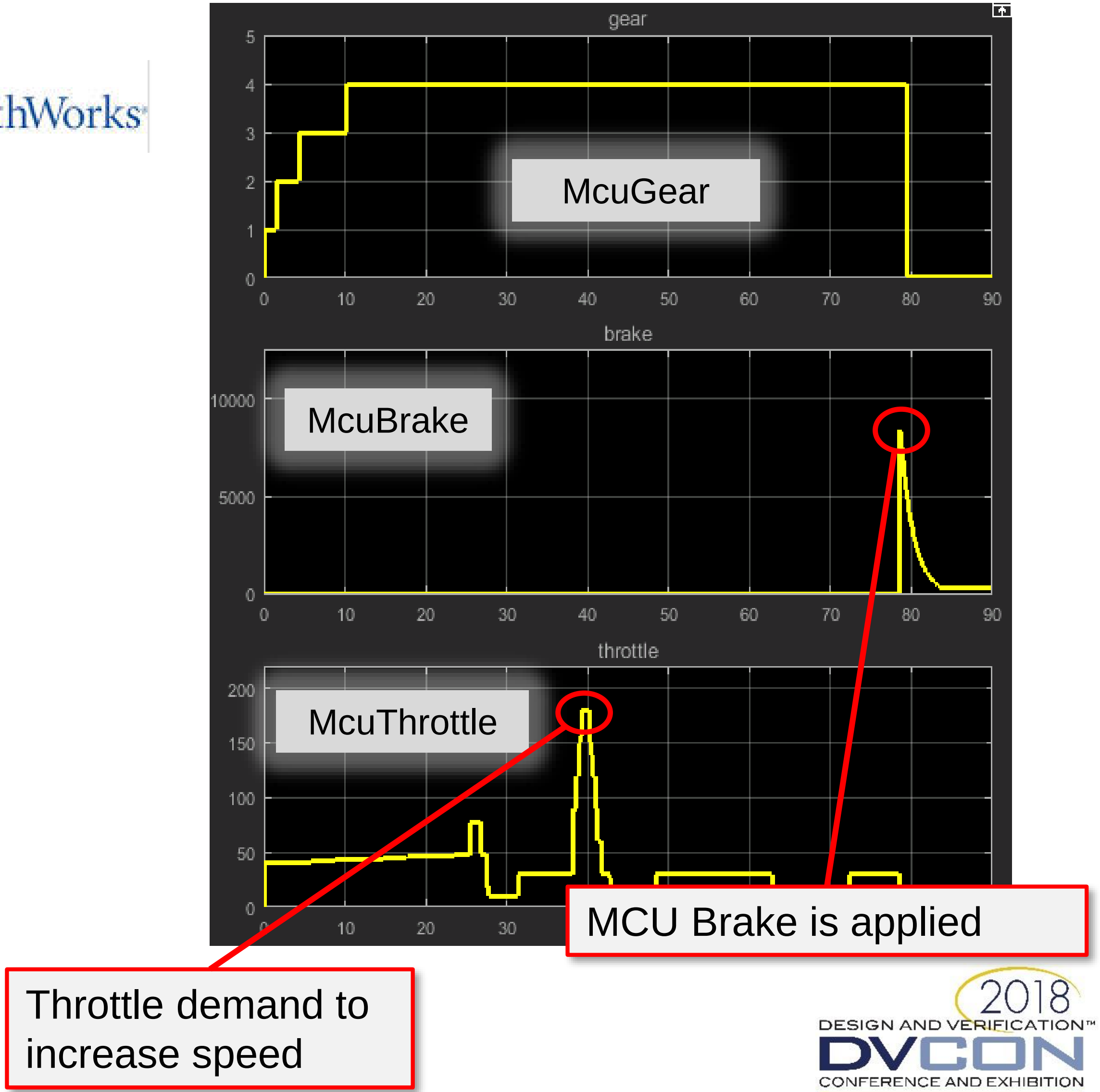
7. Auto Drive Mode Adaptive will duplicate the scenario pictured.

Video One

System Overview & Run Application

Demonstration Test Profile Scenario Summary.

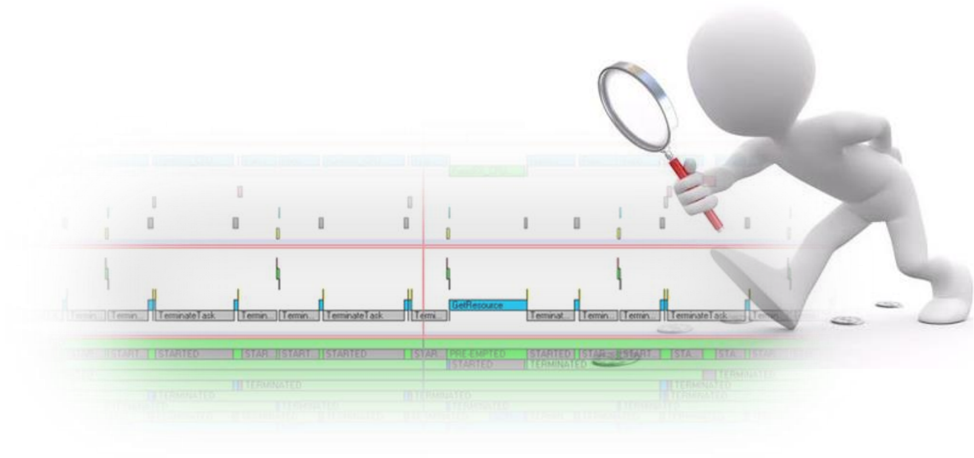




MCU Focused Results Analysis

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

- **Software Tracing**
  - Function Tracing
  - AUTOSAR
    - Task Visualisation
    - System Calls
    - Errors



Video Two

Software and Hardware Analysis of MCU operation

AUTOSAR Instrumentation Summary

OSEK visualization example with Simulation Probes Scripts

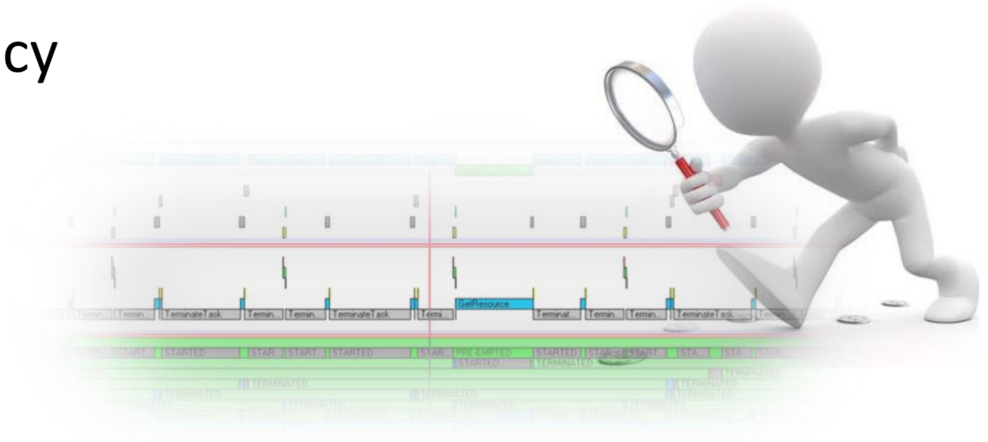
and more.. ISR2 tracking, Task Stack utilization, custom debugging and application analysis, …….

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

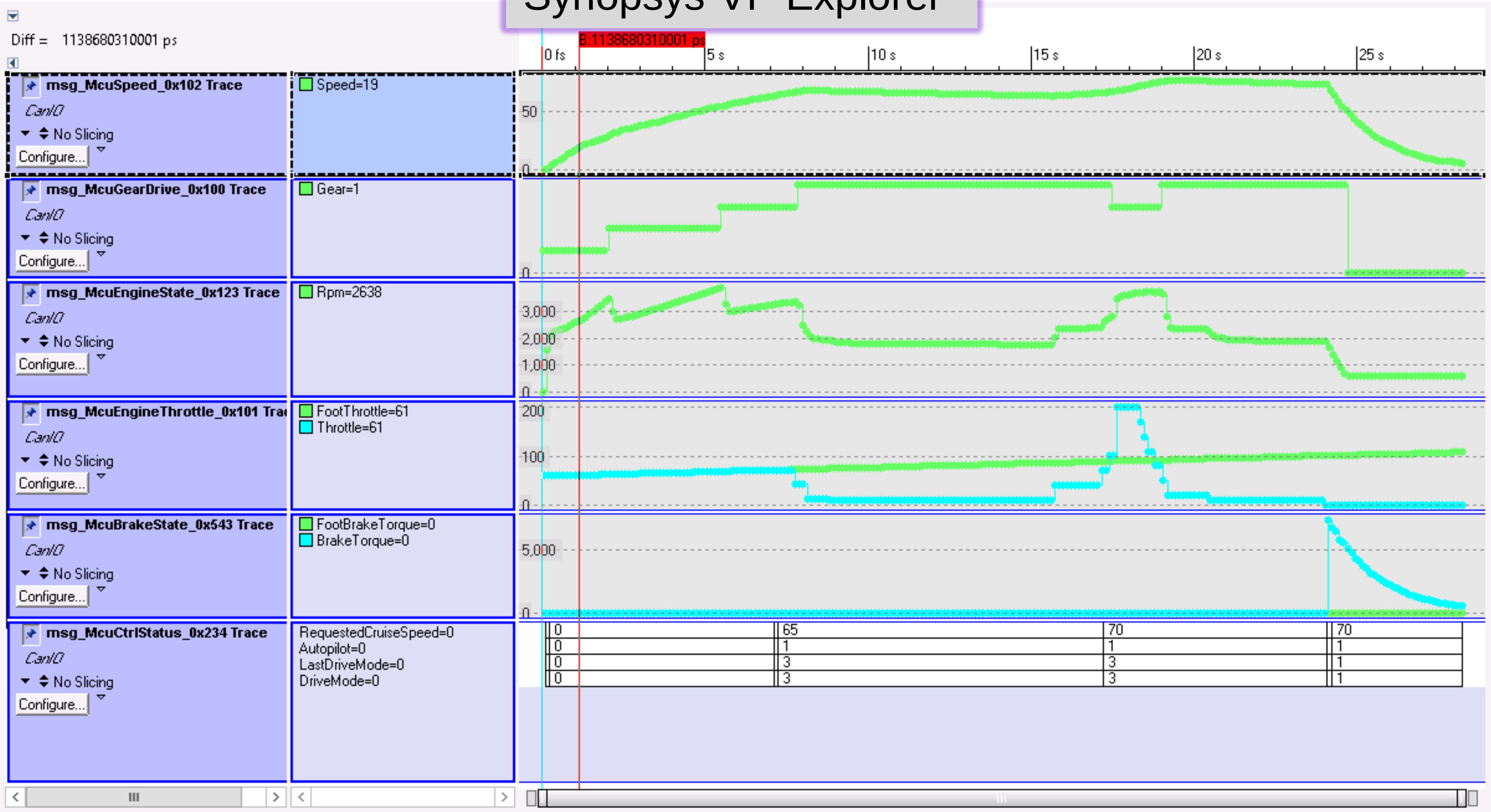


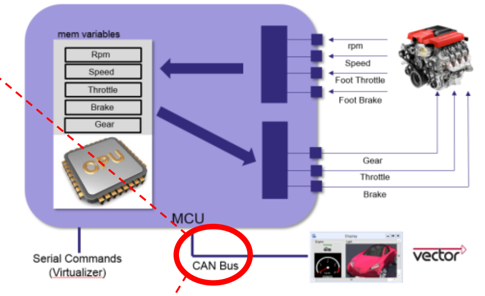
Video Three

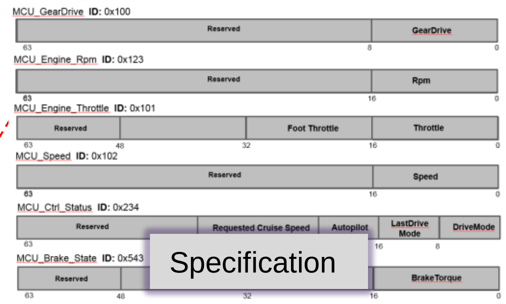
Software and Hardware Analysis of SoC operation

Demo CAN Bus Message Visualization Summary

Synopsys VP Explorer







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.

```
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.

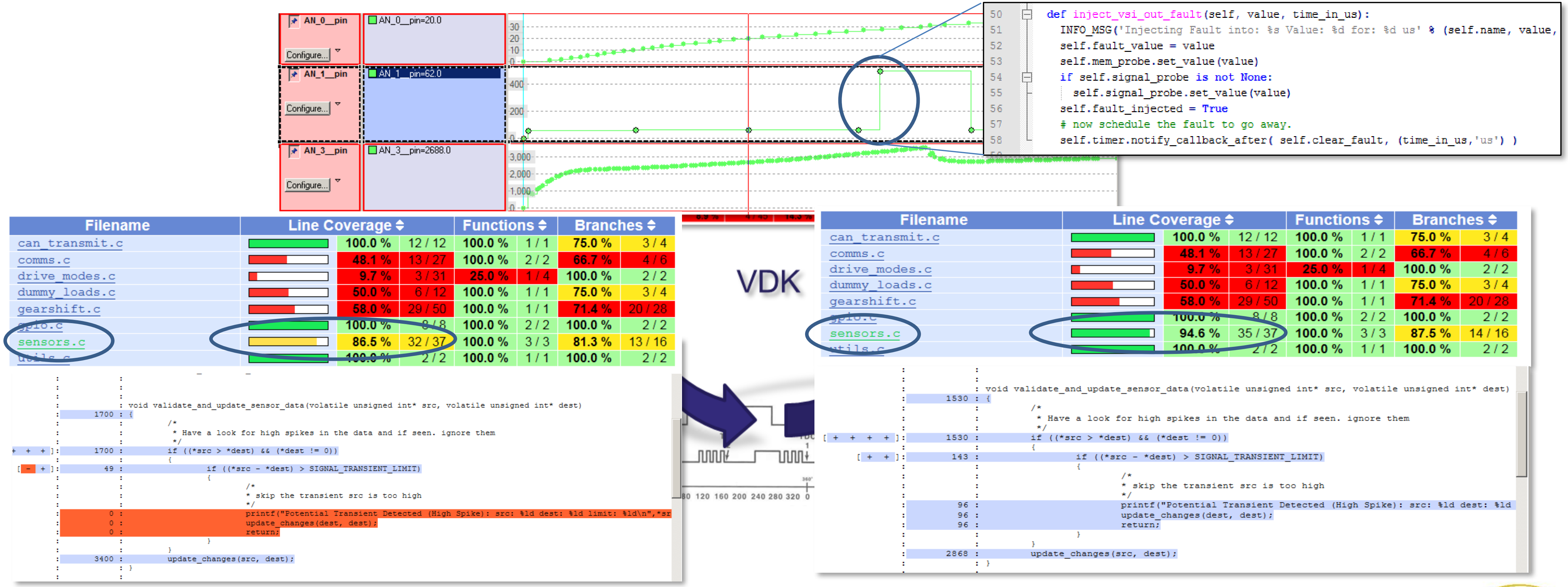
Filename	Line Coverage			Functions		Branches	
can_transmit.c		100.0 %	12 / 12	100.0 %	1 / 1	75.0 %	3 / 4
comms.c		48.1 %	13 / 27	100.0 %	2 / 2	66.7 %	4 / 6
drive_modes.c		9.7 %	3 / 31	25.0 %	1 / 4	100.0 %	2 / 2
dummy_loads.c		50.0 %	6 / 12	100.0 %	1 / 1	75.0 %	3 / 4
gearshift.c		58.0 %	29 / 50	100.0 %	1 / 1	71.4 %	20 / 28
gpio.c		100.0 %	8 / 8	100.0 %	2 / 2	100.0 %	2 / 2
sensors.c		86.5 %	32 / 37	100.0 %	3 / 3	81.3 %	13 / 16
utils.c		100.0 %	2 / 2	100.0 %	1 / 1	100.0 %	2 / 2

/vast_projects/snug2018/VDK_TC38x_v1.0.1/simulation/coverage/drive_c/work/vast_projects/snug2018/VDK_TC38x_v1.0.1/skins/osek/osek_gearshift_sc/Application/sensors.c.gcov.html

```
          :       :  */
          :       : #define SIGNAL_TRANSIENT_LIMIT 300
          :       :
          :       :
          :       :
          :       : void validate_and_update_sensor_data(volatile unsigned int* src, volatile unsigned int* dest)
          :  1700 : {
          :       :         /*
          :       :          * Have a look for high spikes in the data and if seen. ignore them
          :       :          */
+ + + ]   :  1700 :         if ((*src > *dest) && (*dest != 0))
          :       :         {
[ - + ]   :    49 :                 if ((*src - *dest) > SIGNAL_TRANSIENT_LIMIT)
          :       :                 {
          :       :                         /*
          :       :                         * skip the transient src is too high
          :       :                         */
          :     0 :                         printf("Potential Transient Detected (High Spike): src: %ld dest: %ld limit: %ld\n",*sr
          :     0 :                         update_changes(dest, dest);
          :     0 :                         return;
          :       :                 }
          :       :         }
          :  3400 :         update_changes(src, dest);
          :       : }
          :       :
```

Coverage Based Fault Injection

- **Using Simulation Probes we inject a transient into the MCU and re-evaluate.**



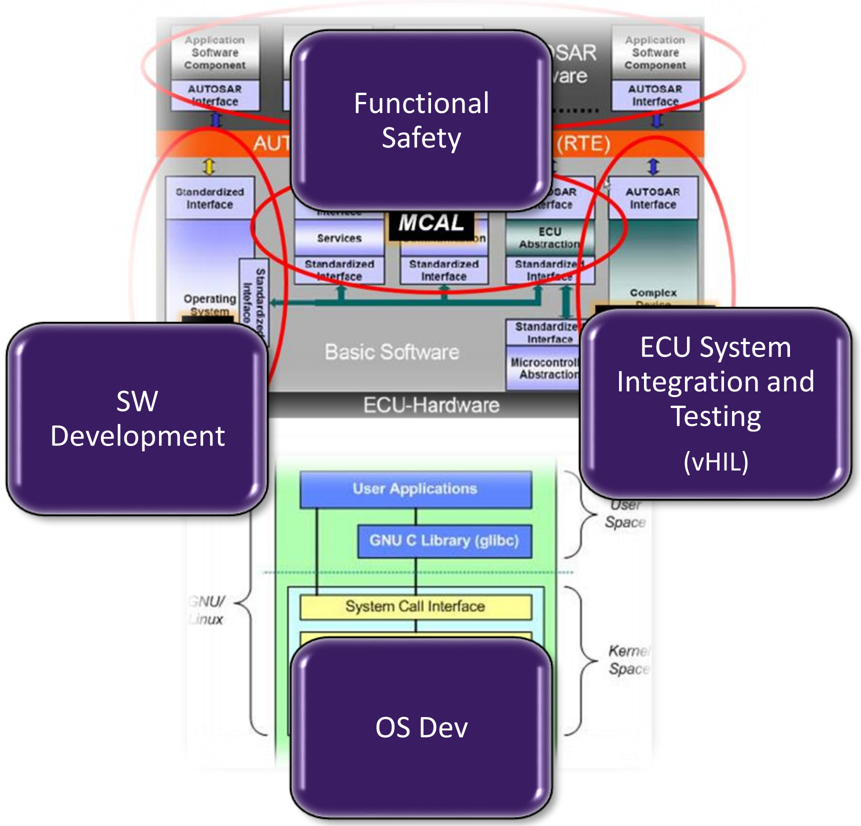
Filename	Line Coverage		Functions		Branches	
can_transmit.c	100.0 %	12 / 12	100.0 %	1 / 1	75.0 %	3 / 4
comms.c	48.1 %	13 / 27	100.0 %	2 / 2	66.7 %	4 / 6
drive_modes.c	9.7 %	3 / 31	25.0 %	1 / 4	100.0 %	2 / 2
dummy_loads.c	50.0 %	6 / 12	100.0 %	1 / 1	75.0 %	3 / 4
gearshift.c	58.0 %	29 / 50	100.0 %	1 / 1	71.4 %	20 / 28
gpio.c	100.0 %	8 / 8	100.0 %	2 / 2	100.0 %	2 / 2
sensors.c	86.5 %	32 / 37	100.0 %	3 / 3	81.3 %	13 / 16
utils.c	100.0 %	2 / 2	100.0 %	1 / 1	100.0 %	2 / 2

Filename	Line Coverage		Functions		Branches	
can_transmit.c	100.0 %	12 / 12	100.0 %	1 / 1	75.0 %	3 / 4
comms.c	48.1 %	13 / 27	100.0 %	2 / 2	66.7 %	4 / 6
drive_modes.c	9.7 %	3 / 31	25.0 %	1 / 4	100.0 %	2 / 2
dummy_loads.c	50.0 %	6 / 12	100.0 %	1 / 1	75.0 %	3 / 4
gearshift.c	58.0 %	29 / 50	100.0 %	1 / 1	71.4 %	20 / 28
gpio.c	100.0 %	8 / 8	100.0 %	2 / 2	100.0 %	2 / 2
sensors.c	94.6 %	35 / 37	100.0 %	3 / 3	87.5 %	14 / 16
utils.c	100.0 %	2 / 2	100.0 %	1 / 1	100.0 %	2 / 2

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:**
  - Reducing Product Time-to-Market.
  - Accelerating Development.
  - Increasing Product Quality and Functional Safety.

SW Development

ECU System Integration and Testing

Functional Safety Testing

Questions