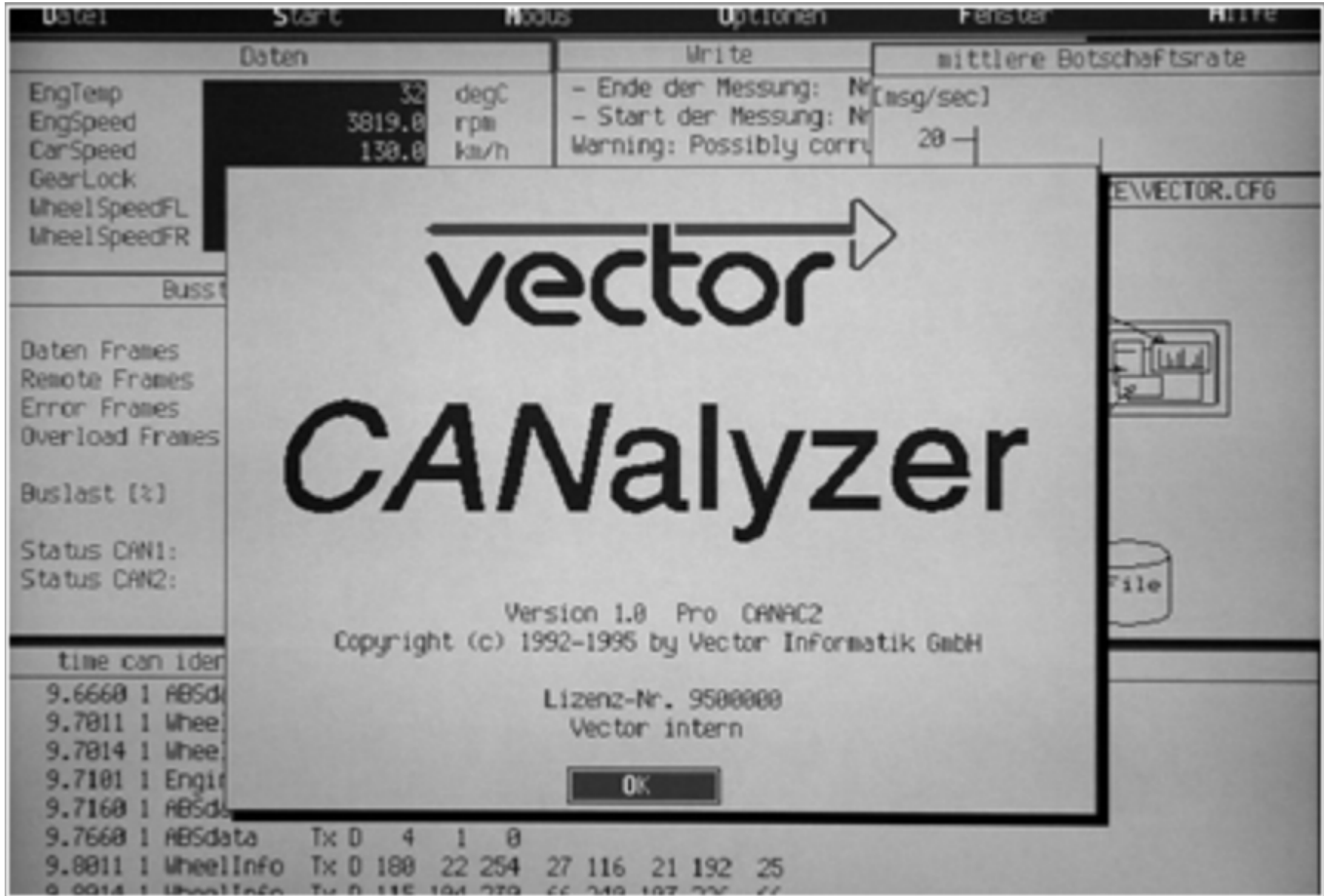


▶▶ Tools for CAN based networking
On the street, in the air, in the orbit



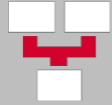
**Vector provides OEMs and suppliers
of automotive and related industries
a professional and open development platform of
tools, software components and services
for creating embedded systems.**

Vector: state of the art tools for (CAN) Networking



▶ 2013	25 th company anniversary, foundation of Vector Austria		
▶ 2012	Foundation of the office for Aerospace customers in Hamburg	CAN FD as part of ISO11898	▶ 2012
▶ 2010	Formation of the 2 Vector foundations, more than 1,000 employees		
▶ 2010	aquintos becomes part of the Vector Group		
▶ 2007-09	Vector Korea, UK, India and China	ARNIC 825 for avionics	▶ 2004
▶ 2005	More than 500 employees worldwide	J1939 for commercial vehicles	▶ 2003
▶ 2001/02	Foundation of Vector Consulting, Vector France and Scandinavia		
▶ 1999	More than 100 employees worldwide		
▶ 1997/98	Vector USA and Japan	CAN used in car interior (C-class)	▶ 1997
▶ 1996	Delivery of the first CANoe and CANape	CANopen / CAN in Automation (CiA)	▶ 1995
▶ 1992	Delivery of the first CANalyzer license	ISO 11898	▶ 1994
		CAN ECU in series production (S-class) <small>(ECU contained Vector embedded software)</small>	▶ 1990
▶ 1988	Foundation of Vector	Intel 82C526 in series	▶ 1988
		First silicon from Intel	▶ 1987
		CAN Spec completed, cooperation with Intel	▶ 1985
		Start of CAN development at Bosch	▶ 1983

Vector Application Areas and Product Examples



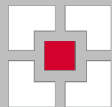
Development of Distributed Systems
PREEvision, Network Designer



ECU Software
MICROSAR, Customer Services



ECU Analysis, Simulation & Testing
CANoe, vTESTcenter, vTESTstudio, VT System, Logger



Diagnostics
CANdela, Indigo, vFlash



ECU Calibration
CANape, VX1000, vCDM

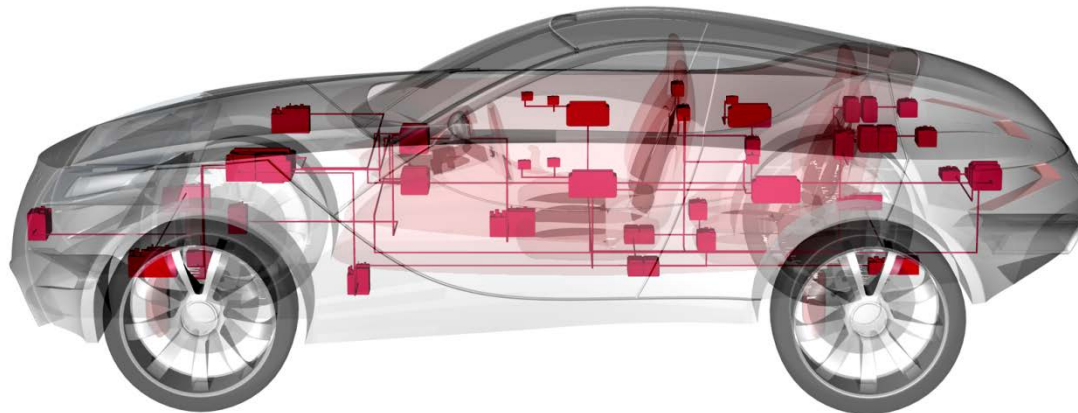


Consulting
Consulting Services, Engineering Services

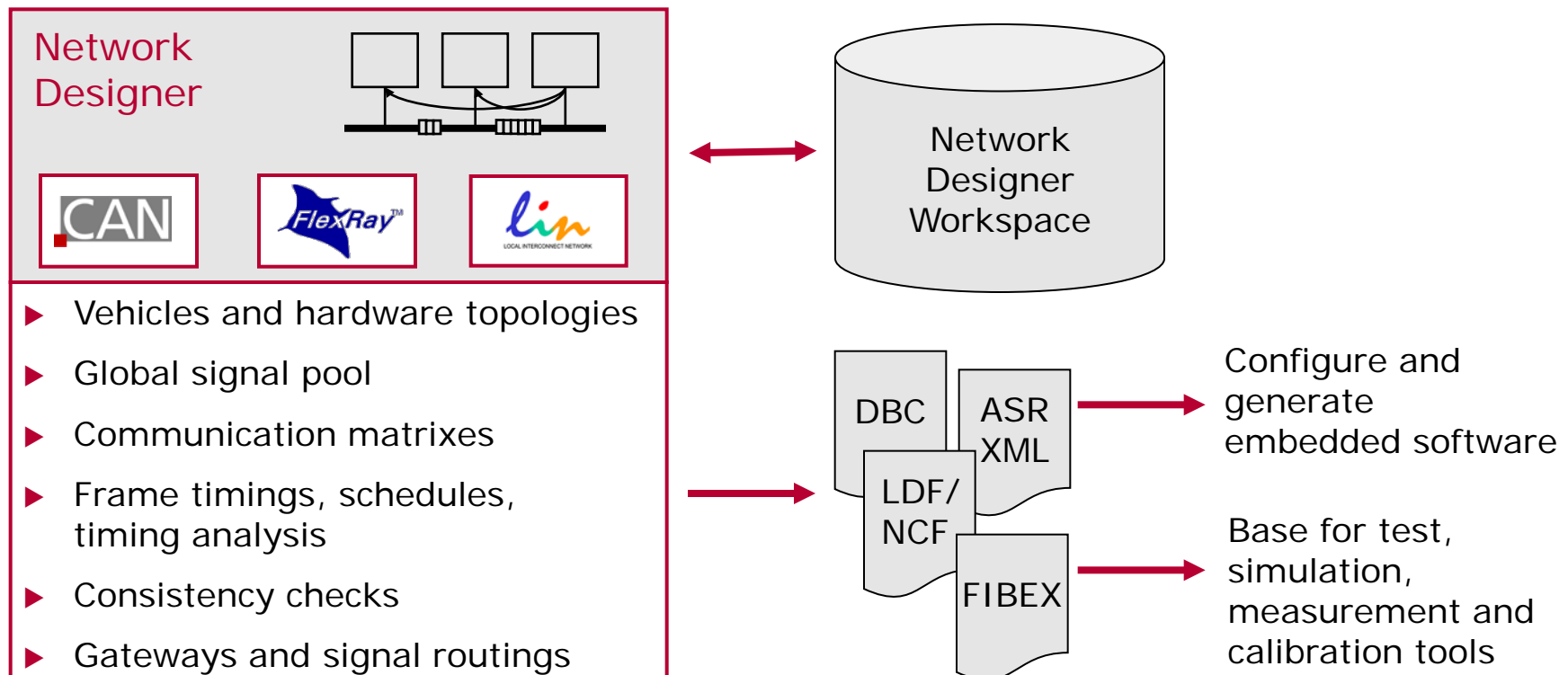
Vector offers solutions for ... >



- ▶ Partitioning of the complete system
- ▶ Definition of the network communication
- ▶ Realization of the single ECU functions



Network Designer supports the design of network architectures for distributed automotive systems



Development of Distributed Systems

Network Designer



Vehicle Project Explorer

Library of global reusable objects

CAN Network <Body_CAN> of <VectorMultibusVehicle> - Network Explorer

Signal	Message	Startbit	Length [Bit]	Transmitter	Receiver(s)
DriverDoor_LockRequest	DoorLockRequestData	0	1	InstrumentUnit	DriverDoorUnit
PassengerDoor_LockRequest	DoorLockRequestData	1	1	InstrumentUnit	PassengerDoorUnit
DriverDoor_WindowPosition	DriverDoorStatus	14	2	DriverDoorUnit	-
DriverDoor_SwitchStatus	DriverDoorStatus	10	2	DriverDoorUnit	-
DriverDoor_WindowCommand	DriverDoorStatus	12	2	DriverDoorUnit	-
DriverDoor_LockRequest	DriverDoorStatus	-	-	-	-
DriverDoor_LockStatus	DriverDoorStatus	-	-	-	-
EnableESP	ESPControlData	-	-	-	-
EngineRevolutions	GW_Engine	-	-	-	-
ESPActive	GW_ESP	-	-	-	-
PassengerDoor_WindowPosition	PassengerDoorStatus	-	-	-	-
PassengerDoor_LockStatus	PassengerDoorStatus	-	-	-	-

Common and bus type specific editors:

- ▶ Network Explorers
- ▶ Gateway Routings
- ▶ Schedule Editors

Gateway-Signals of Vehicle project <VectorMultibusVehicle> - 11 Gateway

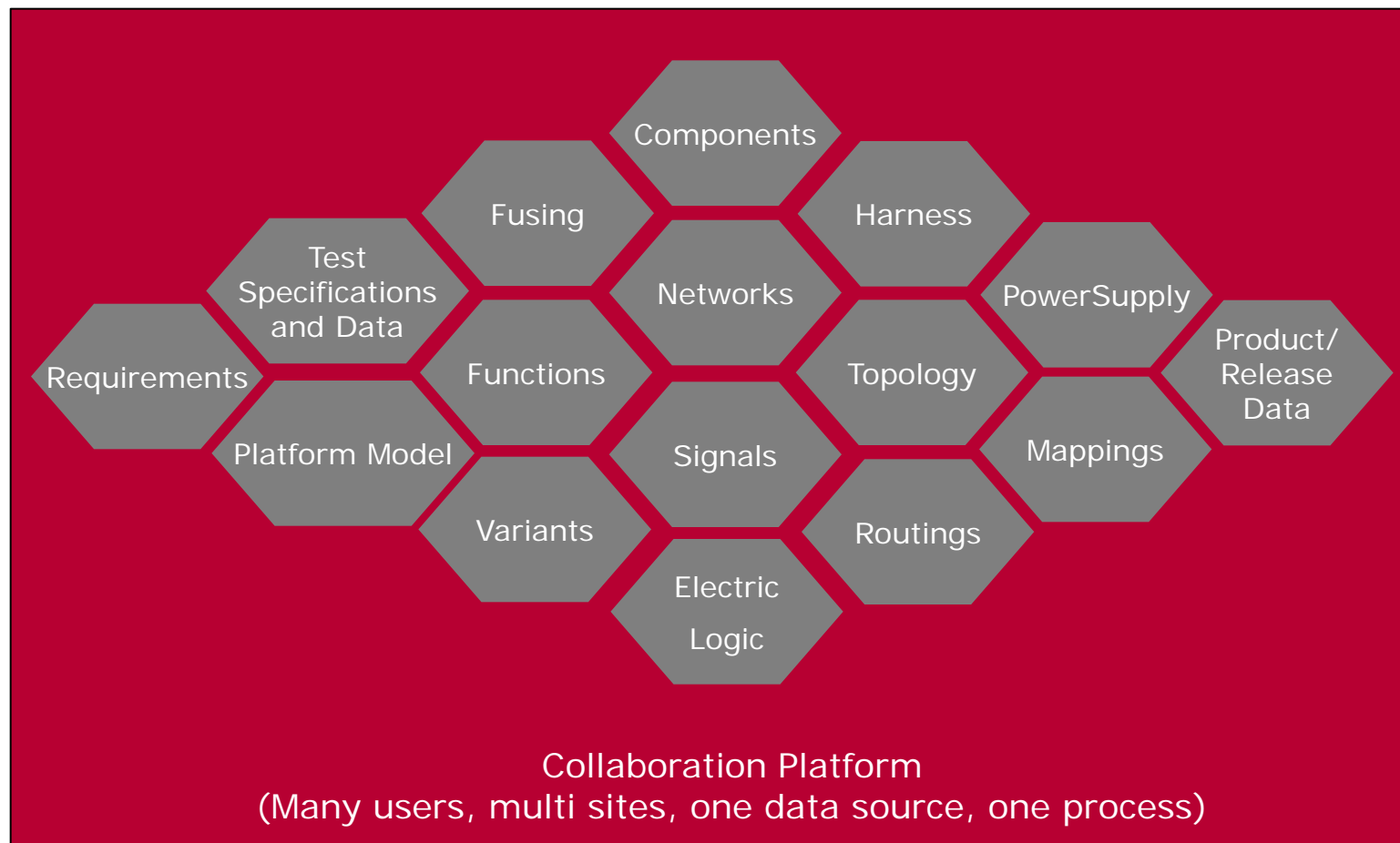
Src. Signal	Source Message	Source Network	Gateway
DriverDoor_LockRequest	DoorLockRequestData	Body_CAN	DriverDoorUr
DriverDoor_LockStatus	LockStatus	DriverDoor_LIN	DriverDoorUr
DriverDoor_SwitchStatus	FrontLeftSwitch	DriverDoor_LIN	DriverDoorUr
DriverDoor_WindowPosition	FrontLeftPosition	DriverDoor_LIN	DriverDoorUr
EnableESP	ESPControlData	Body_CAN	Gateway
EngineRevolutions	EngineData	PowerTrain_CAN	Gateway
ESPActive	ESPRealControl	Chassis FlexRay (rh: A)	Gateway

Global signals - 27 Signal(s)

Name	Length [Bit]	Value Type	Use
BreakIntervention_FL	8	Unsigned	Yes
BreakIntervention_FR	8	Unsigned	Yes
BreakIntervention_RL	8	Unsigned	Yes
BreakIntervention_RR	8	Unsigned	Yes
DiagnosticsBuffer	64	Unsigned	No
DriverDoor_Lock_ErrSig	1	Unsigned	No
DriverDoor_LockRequest	1	Unsigned	No

23 Dec 2009 11:14:02 - Workspace 'C:\Program Files\Vector Network Designer 2.2\Data\Multibus.ndw' successfully opened.
23 Dec 2009 11:23:53 - Workspace 'C:\Program Files\Vector Network Designer 2.2\Data\Multibus.ndw' saved successfully.

PREEVISION® One data model, one GUI, full traceability



Development of Distributed Systems

PREEvision Data Model



Requirements
(Links, Attributes)

Logical Architecture
(Domains, log. Functions, Activity Chains)

SW Architecture
(AUTOSAR, SW Components and Compositions)

SW Implementation
(Simulink „Gray Box“, Parameter Values, Basis SWCs)

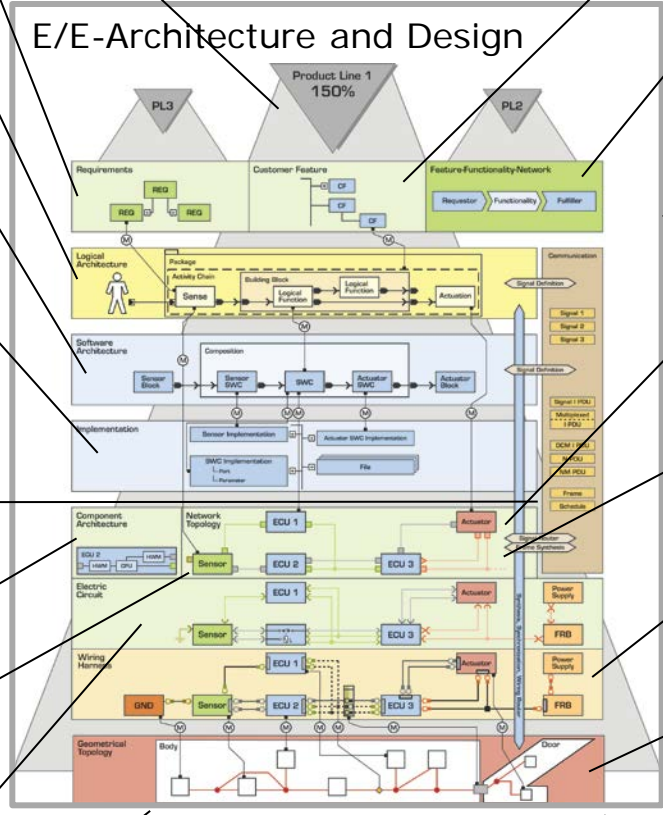
Communication
(Data Element, Signal, PDU, Frame, Bus Schedule, ...)

HW Components
(Internal Structure of ECUs, Sensors, Actuators, ...)

HW Network
(Bus Topologies & Conventional Communication Connections)

HW Schematics
(Electric Functions, Components, ...)

Product Lines
(Reuse, 150% Model, Variant Management)



Customer Features

Feature Function Network
(Use Cases)

Mapping
(Decoupling of Layers, Variant Management)

HW Power Concepts
(incl. Fusing Concepts, ...)

HW Grounding Concepts

HW Wiring Harness
(Cables, Connectors, Pins, ...)

HW Geometry
(Packaging, Environmental Requirements, ...)

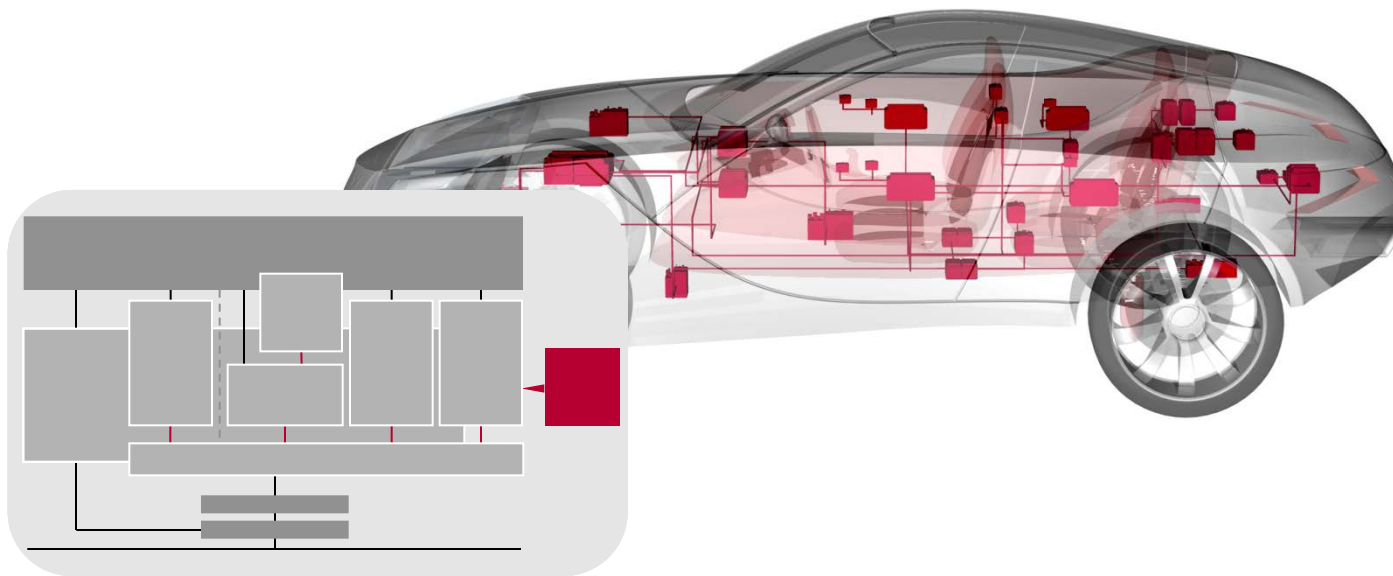
Consistency Checks

Reporting
(Document and HTML Generation)

Metrics
(Calculation of Quality Characteristics)

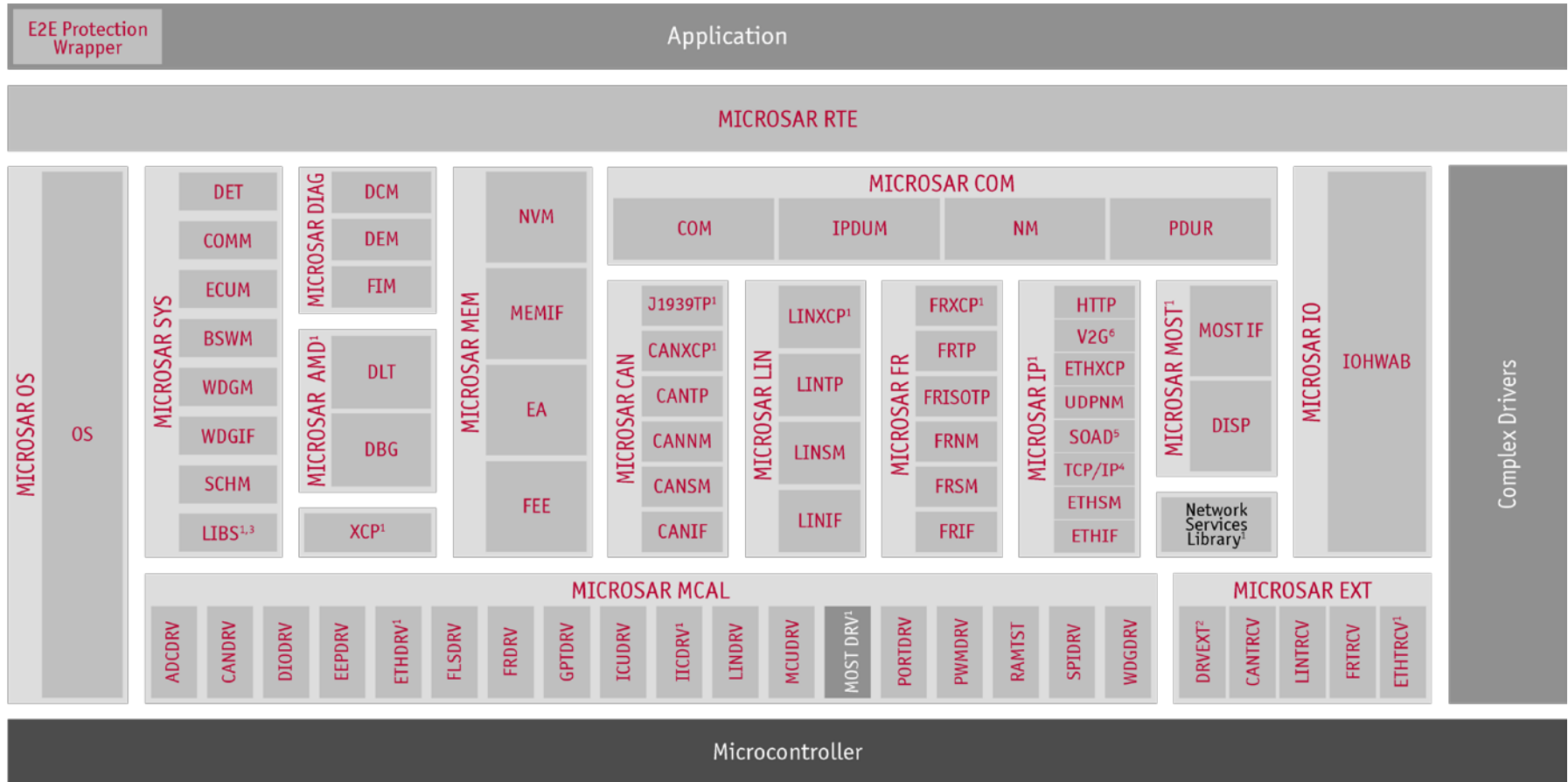


- ▶ Software components for the ECU communication
- ▶ Flash tools for CAN, LIN and FlexRay
- ▶ Real-time operating systems, hardware drivers for ECUs
- ▶ Engineering services, project specific developments for customers



ECU Software

Example Autosar Stack



Standard Software

Projects and Services

3rd Party Software

- ¹ Available extensions for AUTOSAR
- ² Includes EEPEXT, FLSEXT, and WDGEXT
- ³ Includes E2E, CPL and CRC
- ⁴ Includes IPv4/v6, ARP/NDP, ICMP, UDP, TCP, DNS, DHCP and TLS
- ⁵ Includes Diagnostics over IP (DoIP)
- ⁶ Includes SCC according to ISO 15118 and

Design und Generiertool für embedded Komponenten

Views for designing SWC and ECU Projects

ECU Project List

Library of global, reusable objects

Runnable Entity	Component	Order Index	HighPrioAppTask	InputProcTask	LowPrioAppTask
Periodically 10 msec		1	PDC_Manager_main	CentralLockingMaster_Fast	HornActuator_main
Periodically 20 msec		2	Horn_hmi_main		CentralLocking_HMI_main
Periodically 25 msec		3	CentralLockingMaster_Main		GloveBox_Main
Periodically 50 msec		4			
CentralLockingMaster_Main	CentralLockingM	5			
CentralLocking_HMI_main	CentralLocking_	6			
GloveBox_Main	GloveBox	7			

Component Type : CentralLockingMaster_T

Implementation

Runnable Entities: CentralLockingMaster_Fast, CentralLockingMaster_Main

Trigger: Periodically: 25 msec

On Data Reception:

Receiver Port Prototype	Data I
af_req_acc_read_lid_remot	af_re
af_req_access_keyless	req_a

Structure

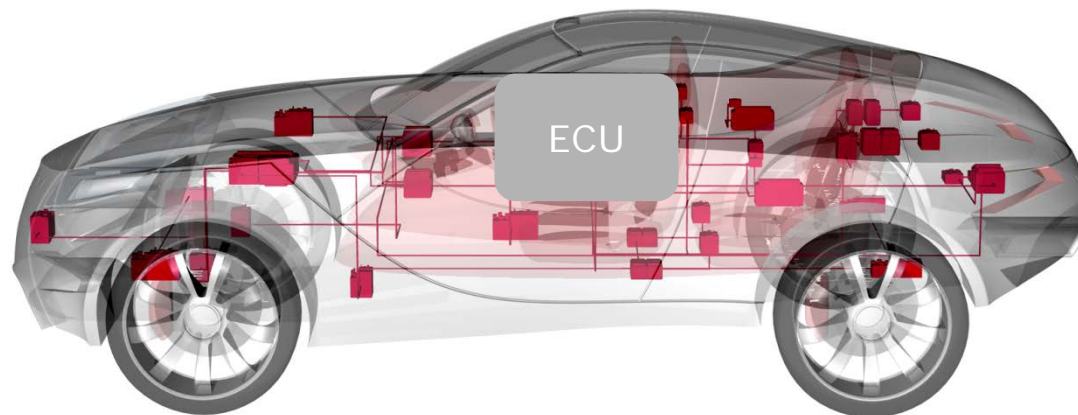
Sheet1

Name	Port Interface	Direction
af_vehicle_speed	af_vehicle_speed	R-Port
af_status_operation_mode	af_status_operation_mode	R-Port
af_status_crash	af_status_crash	R-Port
af_request_comfort_openclose	af_request_comfort_openclose	P-Port
af_req_locking_feedback	af_req_locking_feedback	P-Port

Output: Processing Task Mapping of <FrontModule_Prj1>... Done
26 Apr 2007 15:06:42 - Result: 0 errors, 0 warnings

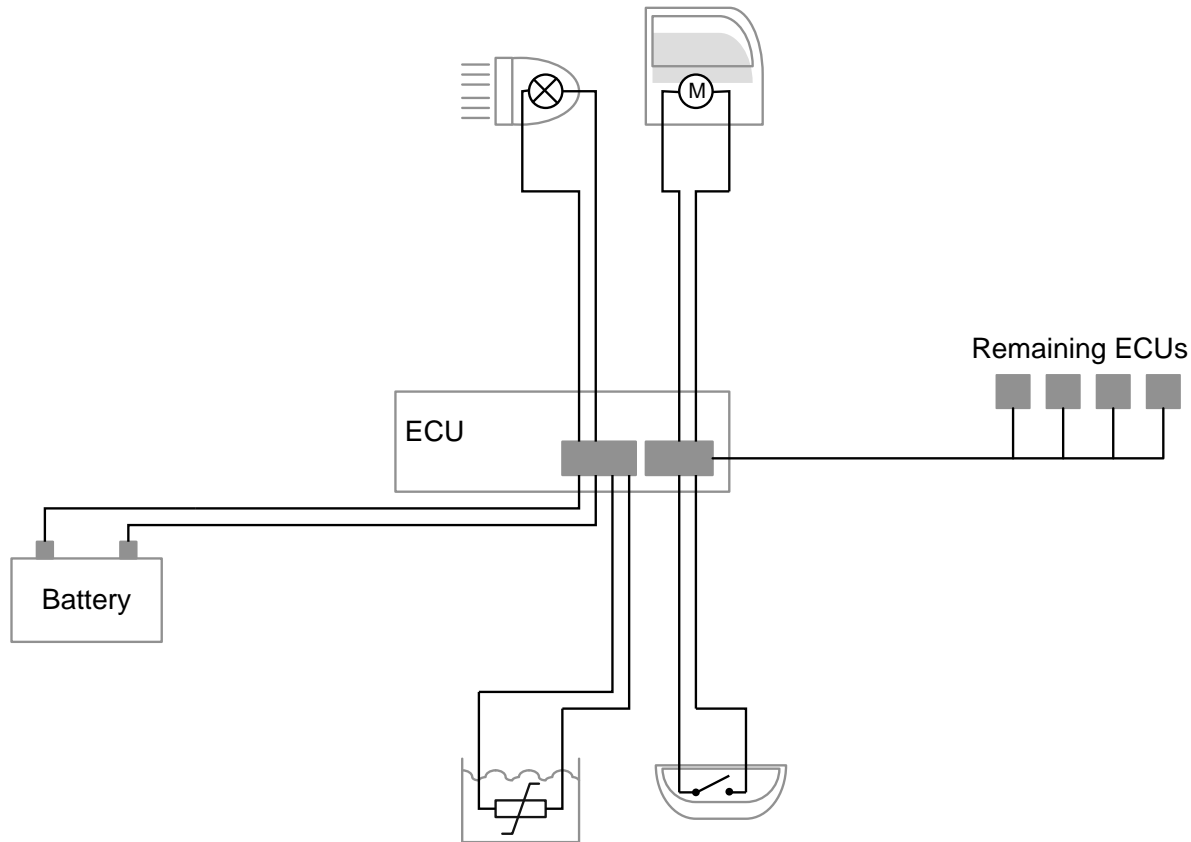


- ▶ Implementation of environments and scenarios for the test of ECUs and the bus communication during all development phases



What do I need for Testing an ECU?

ECU in its natural environment



ECU: Electronic Control Unit

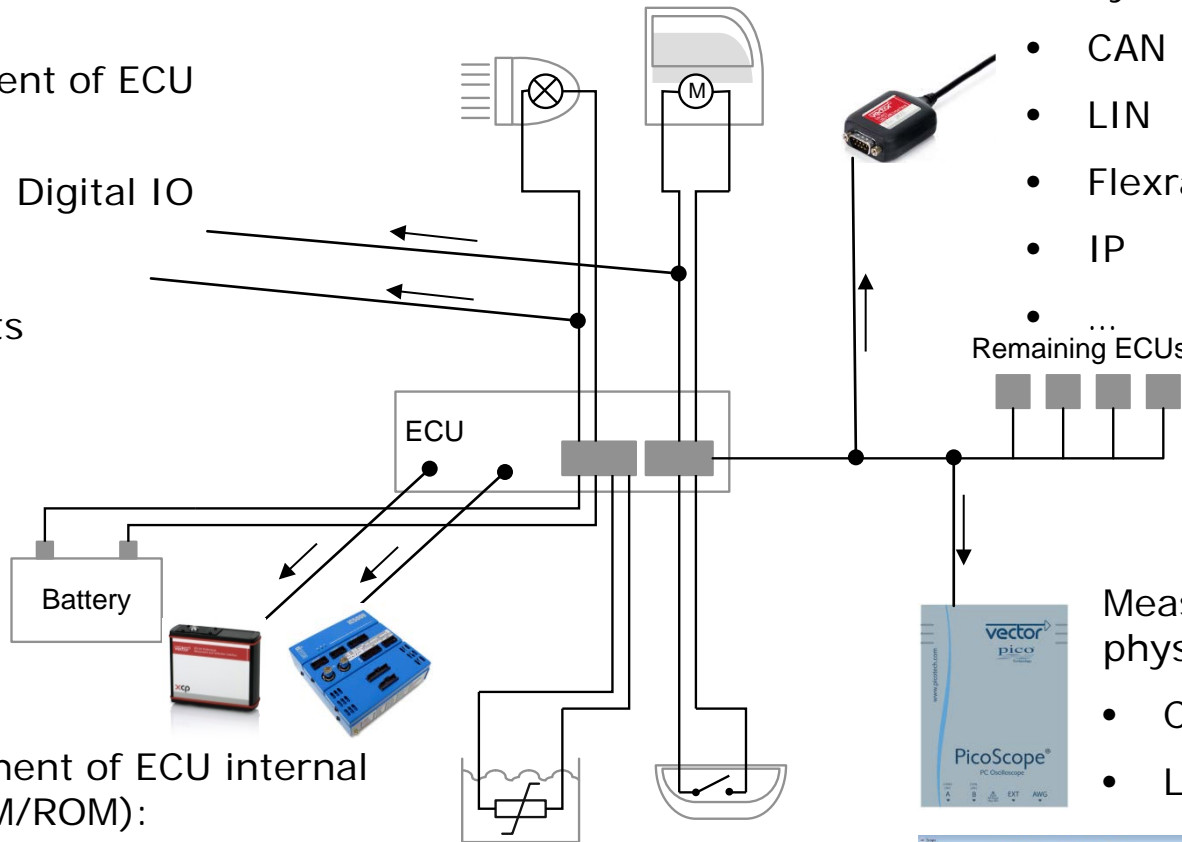
ECU test, simulation and analysis

Analysis / Data acquisition



Measurement of ECU outputs:

- Analog, Digital IO
- PWM
- Currents
- ...



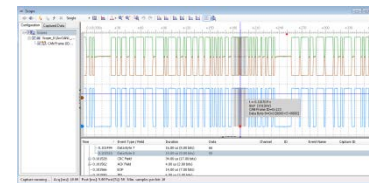
Analysis of bus data:

- CAN
- LIN
- Flexray
- IP
- ...

Remaining ECUs

Measurement of bus physics:

- CAN
- LIN



Measurement of ECU internal data (RAM/ROM):

- XCP via CAN, FR
- XCP via Vx Hardware
- XCP via iSystem Debugger

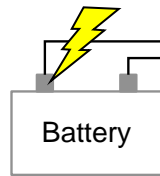
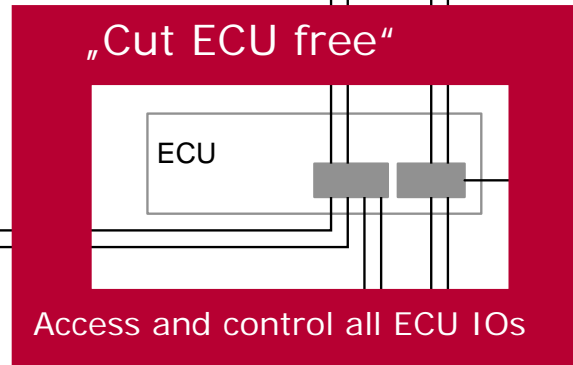
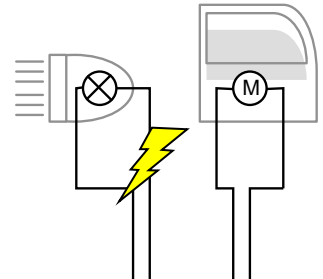
ECU test, simulation and analysis

Simulation of the ECU Environment



Stimulation of ECU inputs:

- Sensor simulation
- PWM
- Resistor
- ...



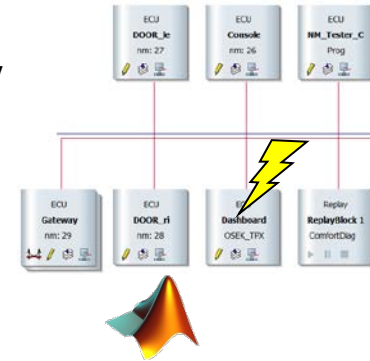
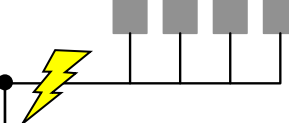
Diagnostics:

- Fault memory
- Coding
- Read/write ECU data
- ...

Remaining bus simulation:

- CAN
- LIN
- Flexray
- IP
- ...

Remaining ECUs



Fault injection:

- Broken wires,
- Short to Vbat, Gnd,
- communication errors (IL)
- Bus errors
- ...





Conclusion:

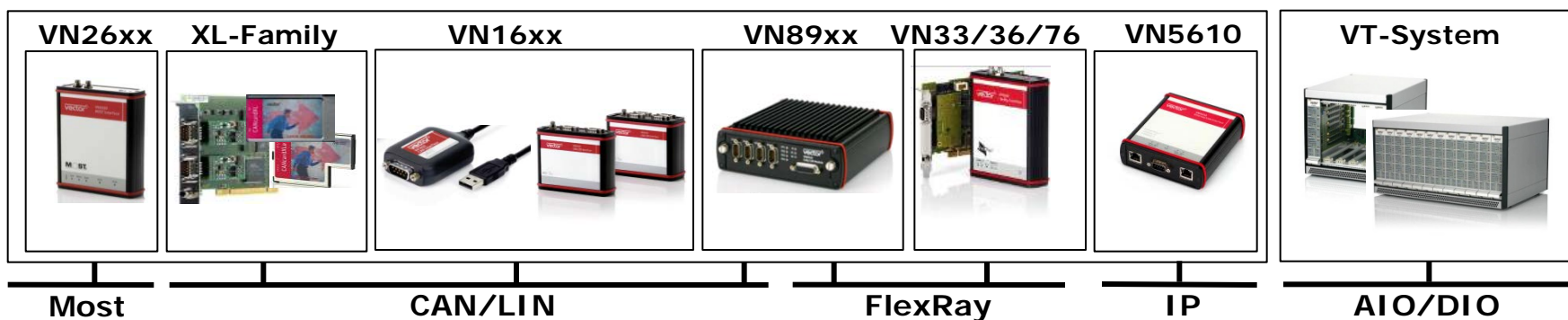
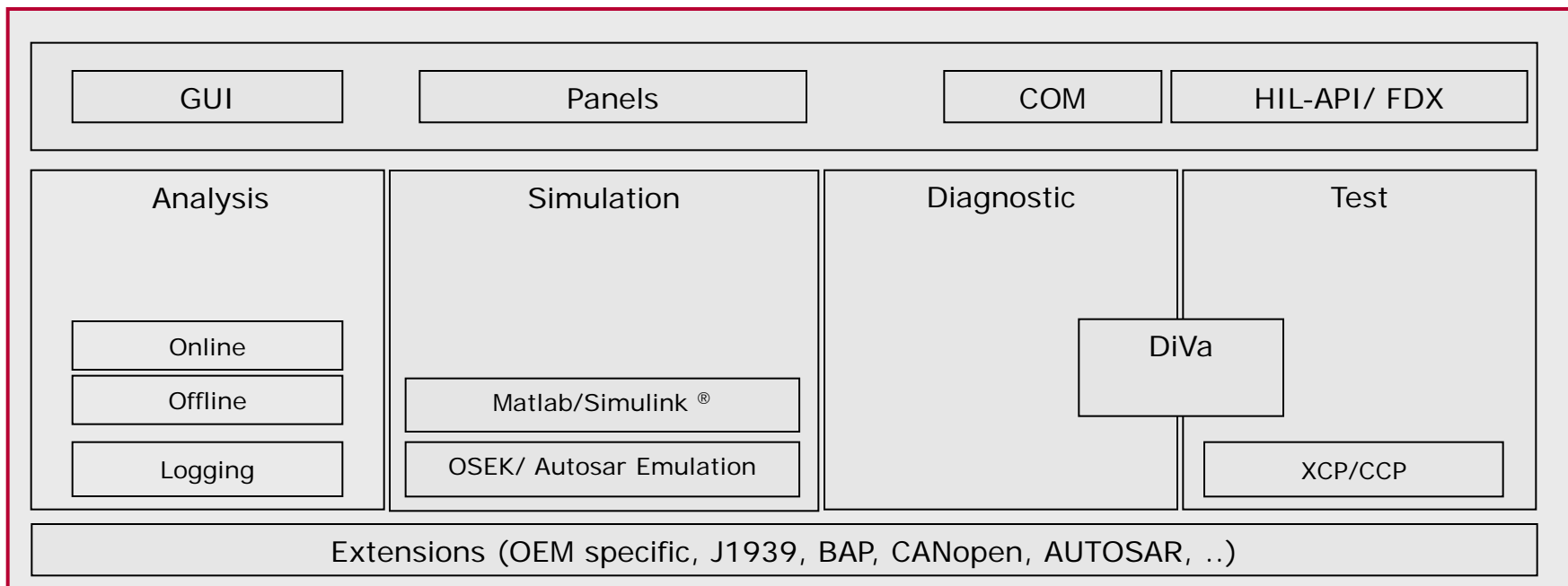
A powerful environment simulation which:

- ▶ provides access to all inputs and outputs of the SUT,
- ▶ is easy and flexible to setup and
- ▶ also provides interfaces to other systems

is the basis for a performant test setup.

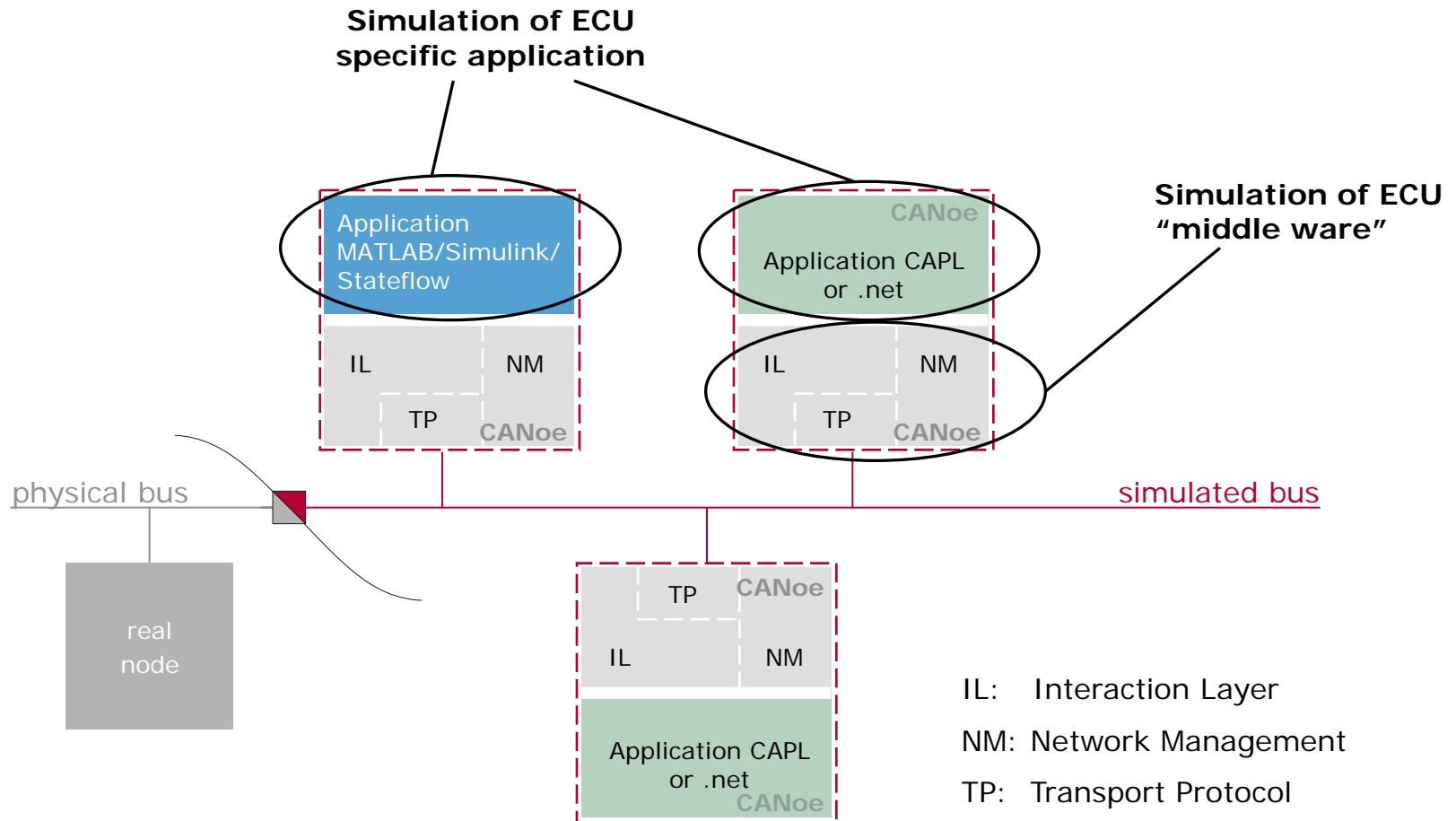
ECU test, simulation and analysis

CANoe



Creating a Remaining Bus Simulation with CANoe

What do I need for a bus simulation?



Details

Creating a Remaining Bus Simulation with CANoe



1. Database – define communication

The screenshot shows the Vector CANdb++ Editor interface. On the left, a tree view displays the project structure, including 'Network nodes', 'Messages', and 'Signals'. The 'Messages' folder is expanded, showing a list of messages, with 'Gateway_2 (0x111)' selected. The central pane displays a table of signals with columns: Name, Leng..., Byte Order, Value Type, Initial Value, Factor, and C. The 'Message 'Gateway_2 (0x111)'' dialog box is open, showing the following configuration:

- Name: Gateway_2
- Type: CAN Standard
- ID: 0x111
- DLC: 8
- Transmitter: Gateway
- Tx Method: not_used
- Cycle Time: 2

The dialog box also includes tabs for 'Layout', 'Attributes', and 'Comment', and buttons for 'OK', 'Abbrechen', 'Übernehmen', and 'Hilfe'.

Creating a Remaining Bus Simulation with CANoe



1. Database – define communication

The screenshot shows the Vector CANdb++ Editor interface. The main window displays a tree view of network nodes on the left and a list of signals in the center. A dialog box titled "Message 'Gateway_2 (0x111)'" is open, showing a bit index table for signal mapping.

Bit index	7	6	5	4	3	2	1	0
0	EngineTemp							
1	CarSpeed	14	13	12	11	10	9	8
2	CarSpeed	22	21	20	19	18	17	16
3	EngSpeed	30	29	28	27	26	25	24
4	EngSpeed	38	37	36	35	34	33	32
5	PetrolLevel	6	45	44	43	42	41	40
6	Voltage	54	53	52	51	50	49	48
7		63	62	61	60	59	58	Voltage

Creating a Remaining Bus Simulation with CANoe



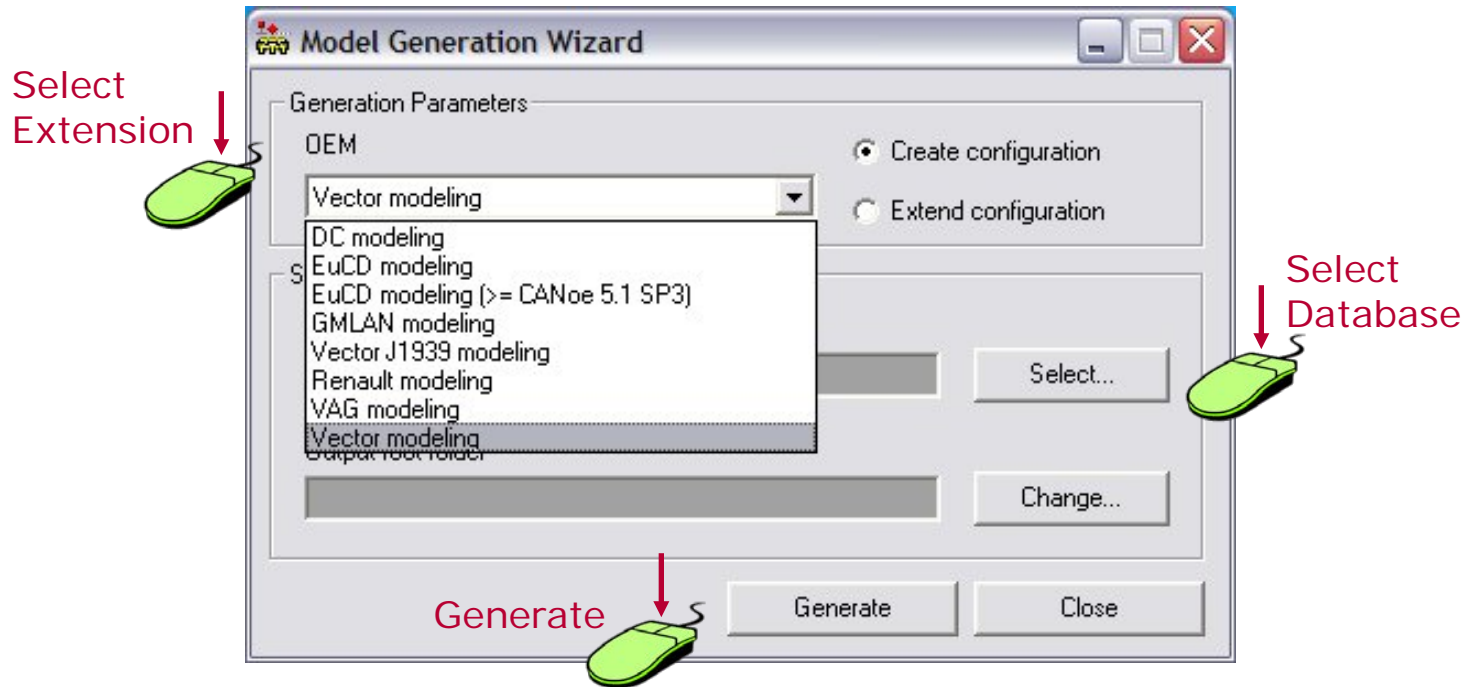
1. Database – define communication

The screenshot shows the Vector CANdb++ Editor interface. On the left, a tree view displays the project structure under 'Network nodes', including 'Messages' and 'Signals'. The main window shows a list of CAN messages with columns for Name, Length, Byte Order, Value Type, Initial Value, Factor, and C. A dialog box titled 'Message 'Gateway_2 (0x111)'' is open, showing the 'Attributes' tab. The 'Interaction Layer' section is expanded, and the 'GenMsgCycleTimeFast' attribute is selected, with a value of '100*' entered. The dialog also includes a 'Definition' field showing 'Integer [2 ... 100000] - 100*' and buttons for 'Read from DB...', 'Write to DB...', 'Reset', 'OK', 'Abbrechen', 'Übernehmen', and 'Hilfe'.

Name	Leng...	Byte Order	Value Type	Initial Value	Factor	C
Active	2	Intel	Unsigned	0	1	0
CarSpeed	16	Intel	Unsigned	0	0.5	0
data						
data						
data						
Data1						
Data2						
Data3						
Data4						
EngineRunning						
EngineTemp						
EngSpeed						
Gear						
Ig_15						
Ig_15R						
Light						
Mirror_I2d						
Mirror_I2l						
Mirror_I2r						
Mirror_I2u						
Mirror_r2d	1	Intel	Unsigned	0	1	0
Mirror_r2l	1	Intel	Unsigned	0	1	0
Mirror_r2r	1	Intel	Unsigned	0	1	0
Mirror_r2u	1	Intel	Unsigned	0	1	0
PetrolLevel	8	Intel	Unsigned	0	1	0

Creating a Remaining Bus Simulation with CANoe

2. Generate a basic bus simulation



For various OEMs there exist oem extension packages that implement the OEM specific basic software behavior.

With „OEM“ Vector nearly everything can be realized if no package is available.

New OEM Packages can be implemented.



DAIMLER



PORSCHE



VOLVO



Creating a Remaining Bus Simulation with CANoe

3. Result: complete basic simulation



Vector CANoe.NMEA 2000.LIN.MOST.FlexRay..J1587 - [Simulation Setup]

File View Start Mode Configuration Window Help

100 sym hex Real bus

Search for

Networks

- CAN Networks
 - Comfort
 - Nodes
 - Console
 - Dashboard
 - DOOR_le
 - DOOR_ri
 - Gateway
 - Generators
 - Interactive Generators
 - Replay blocks
 - Databases
 - Comfort
 - Env
 - Channels

MainPanel

KL 15

Comfort

- Console
- Dashboard
- DOOR_le
- Gateway
- DOOR_ri

PowerTrain

- Engine
- Gateway

Close all

Comfort_DashboardDsp

Gateway::Gateway 1

Gateway::Gateway 2

Gateway::Gateway_2 Hex CAN ID: 0x111

Signalname	Unit	Value
CarSpeed		0
EngSpeed		0
EngineTemp		0
PetrolLevel		0

Console::TP Console



- ▶ Digital/analog IOs of the ECU
- ▶ Equipment for fault injection
- ▶ ECU internal data (XCP)
- ▶ Scopes (triggered out of CANoe)
- ▶ Controllable Power supply
- ▶ External measurement devices (e.g. via GPIB)
- ▶ ...

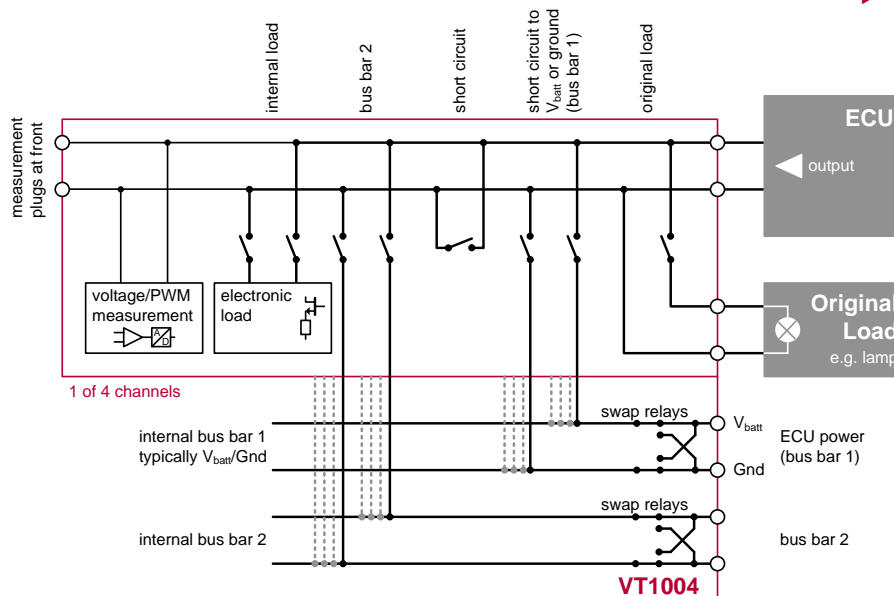
Creating a Remaining Bus Simulation with CANoe



4. Vector Test System (VTS)



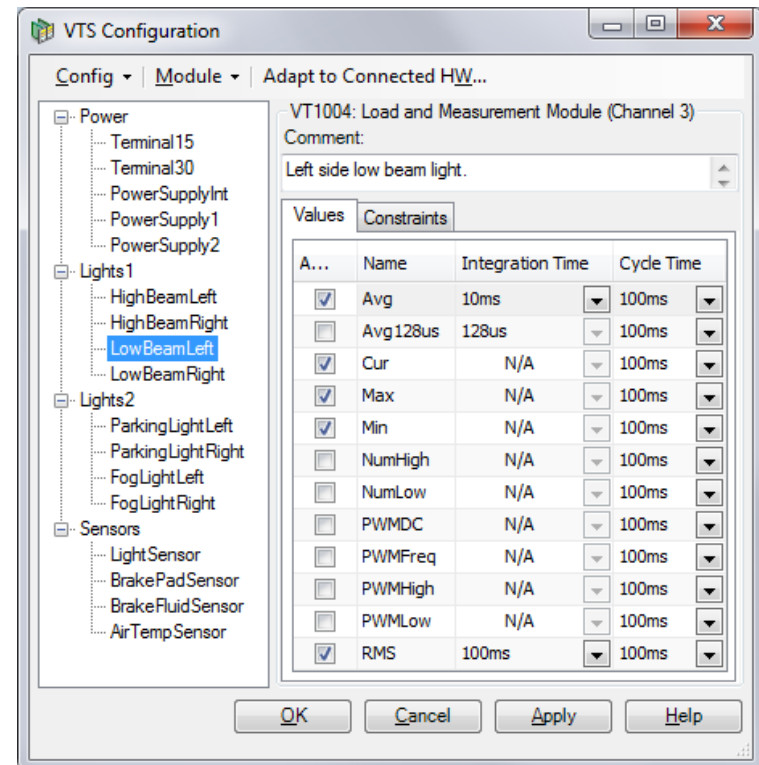
- ▶ All channels work parallel
- ▶ Designed for automotive ECUs
 - ▶ Automotive loads and sensors
 - ▶ Voltage range $\pm 32 \dots \pm 40 \text{ V}$
 - ▶ Currents up to 16 A
- ▶ Fully integrated in CANoe





4. Vector Test System (VTS)

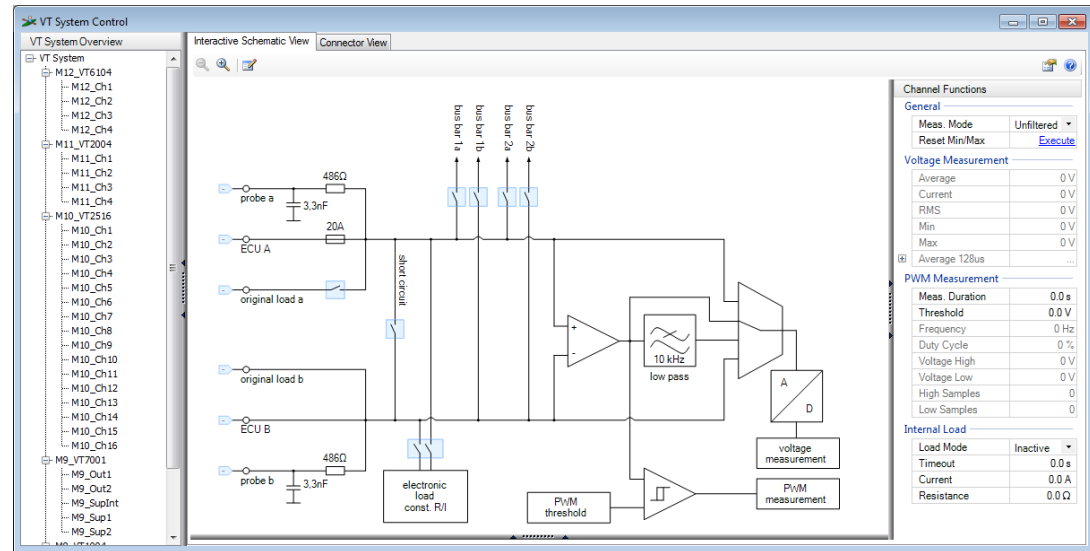
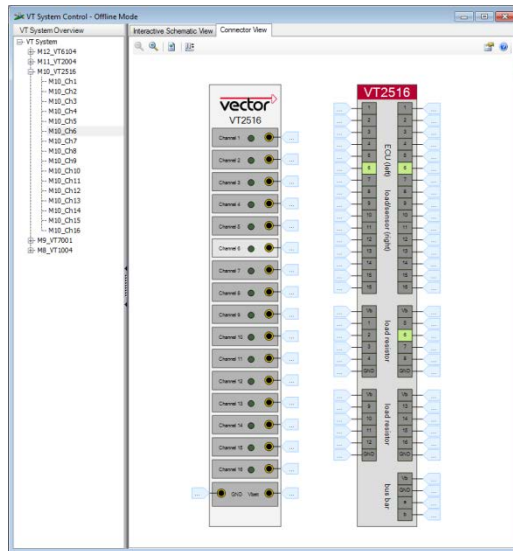
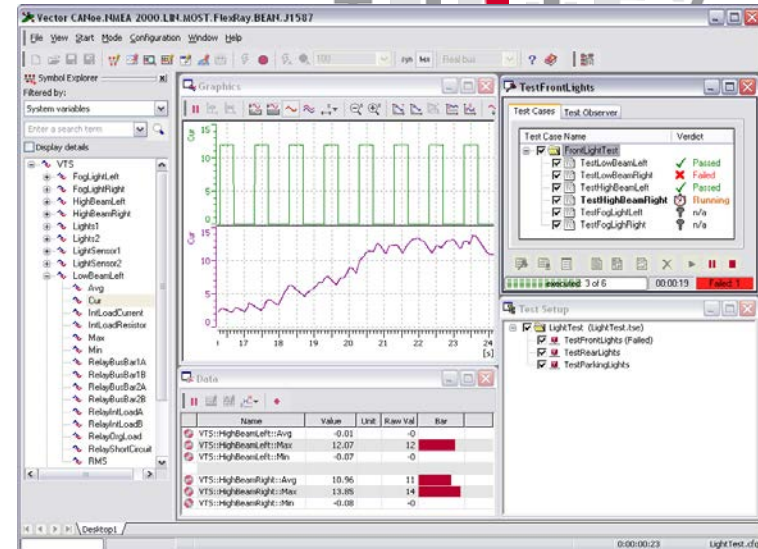
- ▶ VT System is easily configured via CANoe
 - ▶ Individual name can be assigned to each VT System channel
 - ▶ Used measurement signals are selected
 - ▶ Parameters like integration time (e.g. for average value calculation at the module) are configured
 - > Manually using the GUI
 - > Programmatically via CAPL and XML test modules
 - ▶ Constraints can be defined to protect the hardware (e.g. definition of the maximum output voltage)



Creating a Remaining Bus Simulation with CANoe

4. Vector Test System (VTS)

- ▶ Fully integrated into CANoe
- ▶ Direct access to all I/O signals in XML, CAPL, .NET and data visualization
- ▶ Interface to control all settings (CANoe 8.0)
- ▶ Automatic Documentation as GUI and PDF (CANoe 8.0)

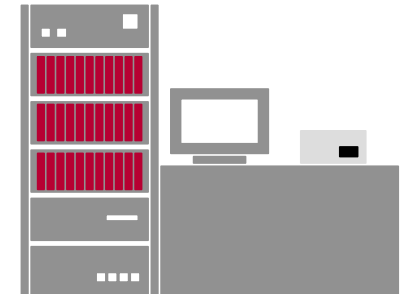
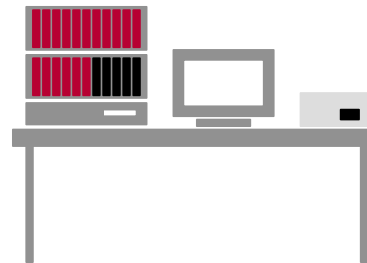


Creating a Remaining Bus Simulation with CANoe

4. Vector Test System (VTS)



- ▶ Modular and scaleable system



Creating a Remaining Bus Simulation with CANoe



4. Vector Test System (VTS)



Load Module
VT1004

Stimulation Modules
analog VT2004
digital VT2516



General Purpose Modules
analog VT2816
digital VT2848
Relais VT2820

Extension Module
VT7900



Real-Time Modules
ATOM VT6010
Core2Duo VT6050



Network Interface Module
VT6104



Power Module
VT7001



Creating a Remaining Bus Simulation with CANoe

4. Vector Test System (VTS)



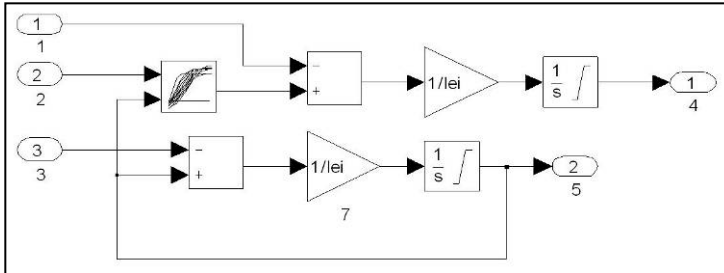
Module	Purpose	Fault Inject.	Channels
Load and Measurement Module VT1004	ECU actuator outputs	Yes	4
Stimulation Module VT2004	ECU sensor inputs	Yes	4
Digital Module VT2516	Digital ECU inputs and outputs	Yes	16
Power Supply Module VT7001	ECU power inputs (term. 15, 30)	Yes	2
Network Module VT6104	ECU networks (CAN/LIN)	Yes	4
Real-time Module VT6010	RT part execution (atom)	-	-
Real-time Module VT6050	RT part execution (core 2 duo)	-	-
General-Purpose Analog I/O Module VT2816 (FPGA)	Analog I/O for arbitrary use (user defined FPGA optionally)	No	in: 12 out: 4
General-Purpose Relay Module VT2820	Relays for arbitrary use	No	20
General-Purpose Digital I/O Module VT2848 (FPGA)	Digital I/O for arbitrary use (user defined FPGA optionally)	No	48
Extension Module VT7900	Base board for appl.-specific electronics	No	-

Creating a Remaining Bus Simulation with CANoe



5. Implement the application behavior

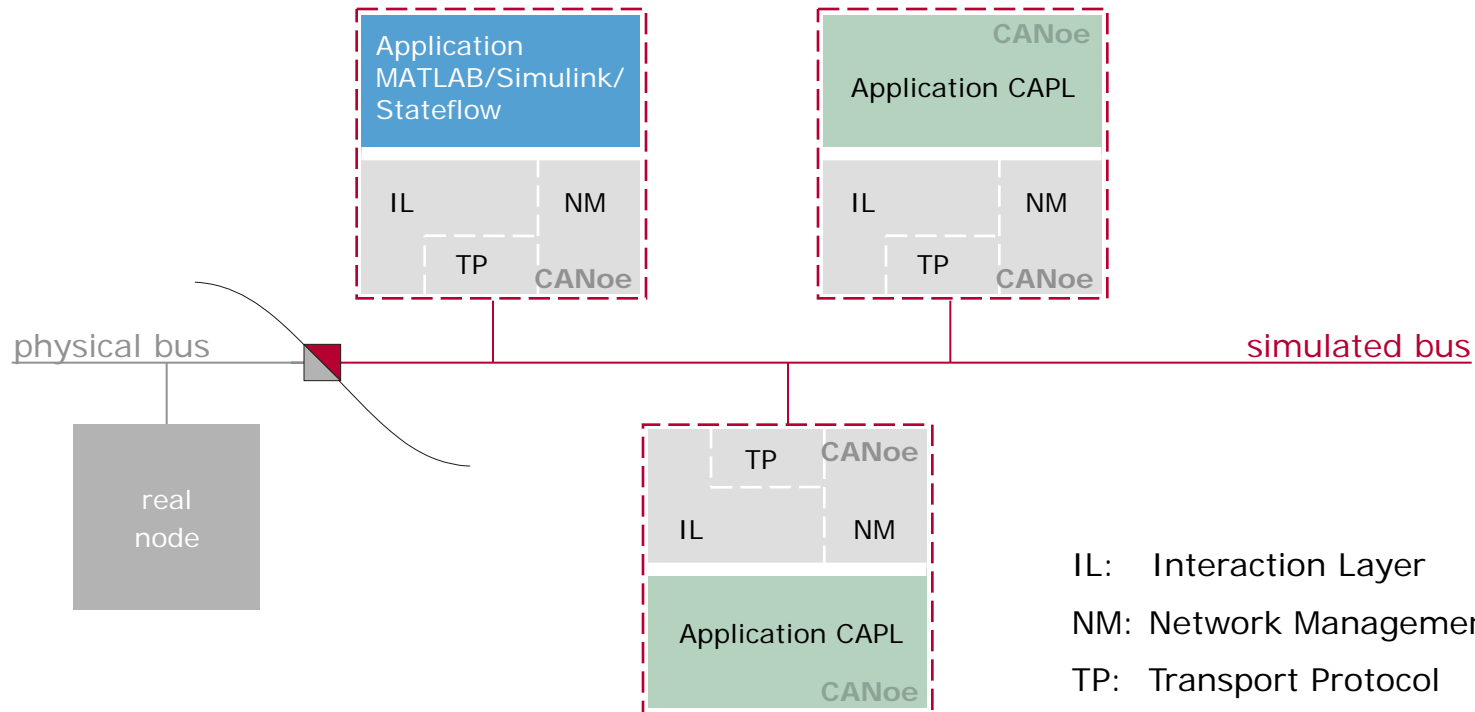
Application Model in MATLAB



Application Model in CAPL or .net

```

on timer tConsoleElementsDsp2
{
  // Shows the elements "Oil", "Water" & "Battery" in the Dashboard.
  // When the devices are ok the lights will disappear.
  if(gIg_15R)
  {
    @Env%NODE_NAME%EngOilDsp_ = 0;
    @Env%NODE_NAME%EngWaterDsp_ = 0;
    @Env%NODE_NAME%EngBatteryDsp_ = 0;
  }
}
    
```



IL: Interaction Layer
 NM: Network Management
 TP: Transport Protocol

Details

Creating a Remaining Bus Simulation with CANoe



5. CAPL Editor

Editing window, e.g. Message Console_1

```
if(this.WN_right_up)
{
    gMoveUp = gTrue;
    ReinitSleepTimer();
}
else gMoveUp = gFalse;
if(this.WN_right_down)
{
    gMoveDown = gTrue;
    ReinitSleepTimer();
}
else gMoveDown = gFalse;
```

CAPL Function explorer:
CANoe CAPL functions
Node layer functions

```
double deltaConsumedPetrol;
double distance;
int engPetrolLevel;

if (@EnvEngineRunningPowerTrain) {
    // calculation of actual remaining petrol
    distance = (double) (gSpeed * gPetrolLevelUpdateInterval);
    deltaConsumedPetrol = kPetrolConsumptionFactor *
        gActualPetrolLevel - deltaConsumedPetrol;
    if (gActualPetrolLevel < 0) {
        gActualPetrolLevel = 0;
    }
}

// ...

132 on timer tEngTempTimer
133 {
134     if (@EnvEngineRunningPowerTrain)
135     {
136         // engine is heating up
137         gEngTemp += kEngTempInc;
```

Creating a Remaining Bus Simulation with CANoe

5. CAPL/.net debugger



The screenshot shows the CANoe debugger interface. The main window displays a CAPL script for 'Engine.can' with the following code:

```
725 @sysvar::PowerTrain::GearTextDisplay = this;
726 }
727 /*@@end*/
728
729 /*@@sysvarUpdate:PowerTrain::EngineRunningPowerTrain:*/
730 on sysvar_update sysvar::PowerTrain::EngineRunningPowerTrain
731 {
732     $EngineData::PetrolLevel = 0;
733     $EngineData::EngTemp = 0;
734     $EngineData::EngSpeed = 0;
735     $EngineData::EngForce = 0;
736     $EngineData::EngPower = 0;
737     $EngineDataIEEE::EngSpeed = 0;
738     $EngineDataIEEE::EngForce = 0;
739
740     if(@this)
741     {
742         $EngineData::IdleRunning = EngineData.IdleRunning::Running;
743         setTimer(tEngTempTimer,kEngTempUpdateInterval);
744         setTimer(tSystemClockTimer,kDeltaT);
745         setTimer(tEngPetrol,kEngPetrolBaseTime);
746         NMSSetApplicationActivity(1);
747     }
748     else
749     {
750         $EngineData::IdleRunning = EngineData.IdleRunning::Idle;
751         NMSSetApplicationActivity(0);
752     }
753 }
```

The 'Watch' window on the left shows the following variables:

Name	Step Over	ta Type	Function
a		int	on key 'a'
a		int	MyFunc
b		int	MyFunc
this	1	sysvar	on sysvar u...
this		sysvar	on sysvar u...
array		int[10][10][...]	MyIncludeF...

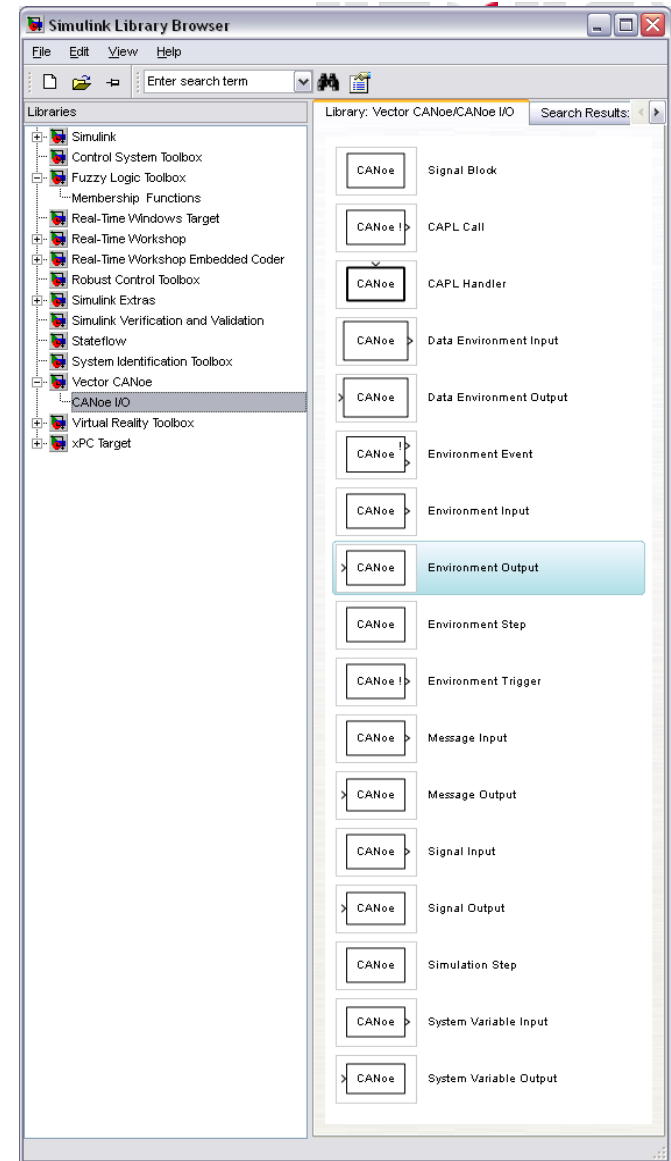
The 'Variables' window at the bottom shows the following variables:

Name	Value	Data Type	Watch	Function
engSpeed		float	<input type="checkbox"/>	CalcEngSpeed
engineIsRunning		int	<input type="checkbox"/>	CalcEngSpeed
gear		int	<input type="checkbox"/>	CalcEngSpeed
pedal		float	<input type="checkbox"/>	CalcEngSpeed
ratio		float	<input type="checkbox"/>	CalcEngSpeed
speed		float	<input type="checkbox"/>	CalcEngSpeed
connHandle		long	<input type="checkbox"/>	CanTp_ErrorInd
error		long	<input type="checkbox"/>	CanTp_ErrorInd

Creating a Remaining Bus Simulation with CANoe

5. I/O Blockset in Matlab/Simulink

- ▶ I/O blocks to interface Simulink with...
- ▶ Bus signals of any type (CAN, LIN, FlexRay) for reading and writing*
- ▶ Environment and system variable inputs and outputs
- ▶ CAPL functions (called from Simulink)
- ▶ Inputs for subsystems triggered by CAPL functions
- ▶ CANoe slave mode driver (simulation step)
- ▶ Connect the Simulink model to “the real world”



Creating a Remaining Bus Simulation with CANoe



5. Example: Model browser

The screenshot displays the Vector CANoe software interface. The main window shows a Simulink model for a car simulation. On the left, there are two data tables. The top table lists simulation parameters:

Name	Value	Unit	Raw Value	Bar
InitialCondition	1000.00		1000	
LowerSaturationLimit	600.00		600	
UpperSaturationLimit	6000.00		6000	

The bottom table lists simulation variables:

Name	Value
CarSpeed [mp/h]	
EngSpeed [r/min]	
Gear	

The main workspace shows a Simulink model with various blocks like 'Product', 'Impeller torque', 'Engine', 'Transmission', and 'Vehicle'. Annotations with arrows point from text boxes to specific elements in the model and data tables.

Text Box 1: Browse the model during model execution. Watch internal model execution.

Text Box 2: Watch or modify simulink parameters in CANoe windows like graphic window or also in CAPL Programs.

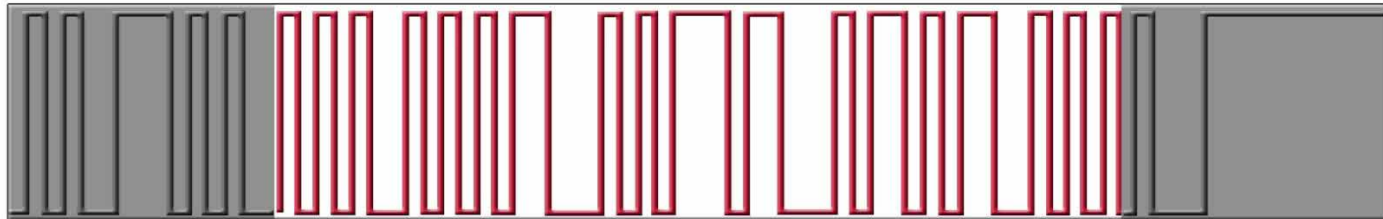
Text Box 3: For adding it to the window simply drag&drop

Details

ECU test, simulation and analysis

Overview: CANalyzer/CANoe

The image displays a comprehensive view of the Vector CANalyzer and CANoe software environment. On the left, the CANoe NMEA2000 LIN.MOST.FlexRay.J1587 simulation setup is visible, showing a network of ECUs including Gateway, DOOR_Le, Console, DOOR_r, and Dashboard connected to a Bus Comfort CAN 1. The central part of the interface shows a detailed trace of CAN messages, with a 'Trace Power Train' window highlighting specific data like engine speed and gear information. A 'Graphic' window provides a visual representation of engine speed (rpm) and gear changes over time. Below the trace, a 'Statistic' window shows key performance indicators: Busload [%] at 0.36, Std. Data [fr/s] at 21, and Std. Data [total] at 4393. At the bottom, a 'Dashboard' window displays a virtual instrument cluster with gauges for speed, RPM, and temperature.



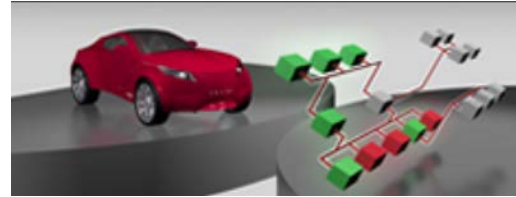
Classic CAN...

ECU test, simulation and analysis

CAN FD Support by Vector Tools

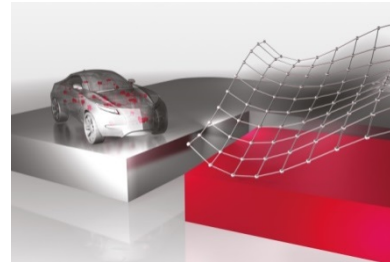
▶ CANoe/CANalyzer

- ▶ Full support since version 8.1 (Aug. 2013)



▶ CANape

- ▶ Full support since version 12.0 (Aug. 2013)



VN1611

1 LIN/K-Line

+ 1 CAN/CAN FD

VN1610

2 CAN/CAN FD



VN1630

2 CAN/CAN FD or LIN/K-Line (as piggy)

+ 2 CAN/CAN FD

VN1640

4 CAN/CAN FD or LIN/K-Line (as piggy)

- ▶ Other „VN“ Interfaces will follow in 2014

Test Data Management

Vector vTESTcenter /
PREEvision. vTESTcenter

- ▶ Configuration Management for test implementations and test parameters
- ▶ Scalable Team Collaboration Platform
- ▶ Traceability from test reports to requirements and vice versa
- ▶ Analysis of test results
- ▶ Management of testing projects

Test Design and Authoring Tool

vTESTstudio – Vector Test Studio (formerly ITE)

- ▶ Test programming (CAPL, C/C++, C#)
- ▶ Interactive test design: Test Table Editor "style of Test Automation Editor"
- ▶ Test Diagram Editor
- ▶ Definition of parameters and curves

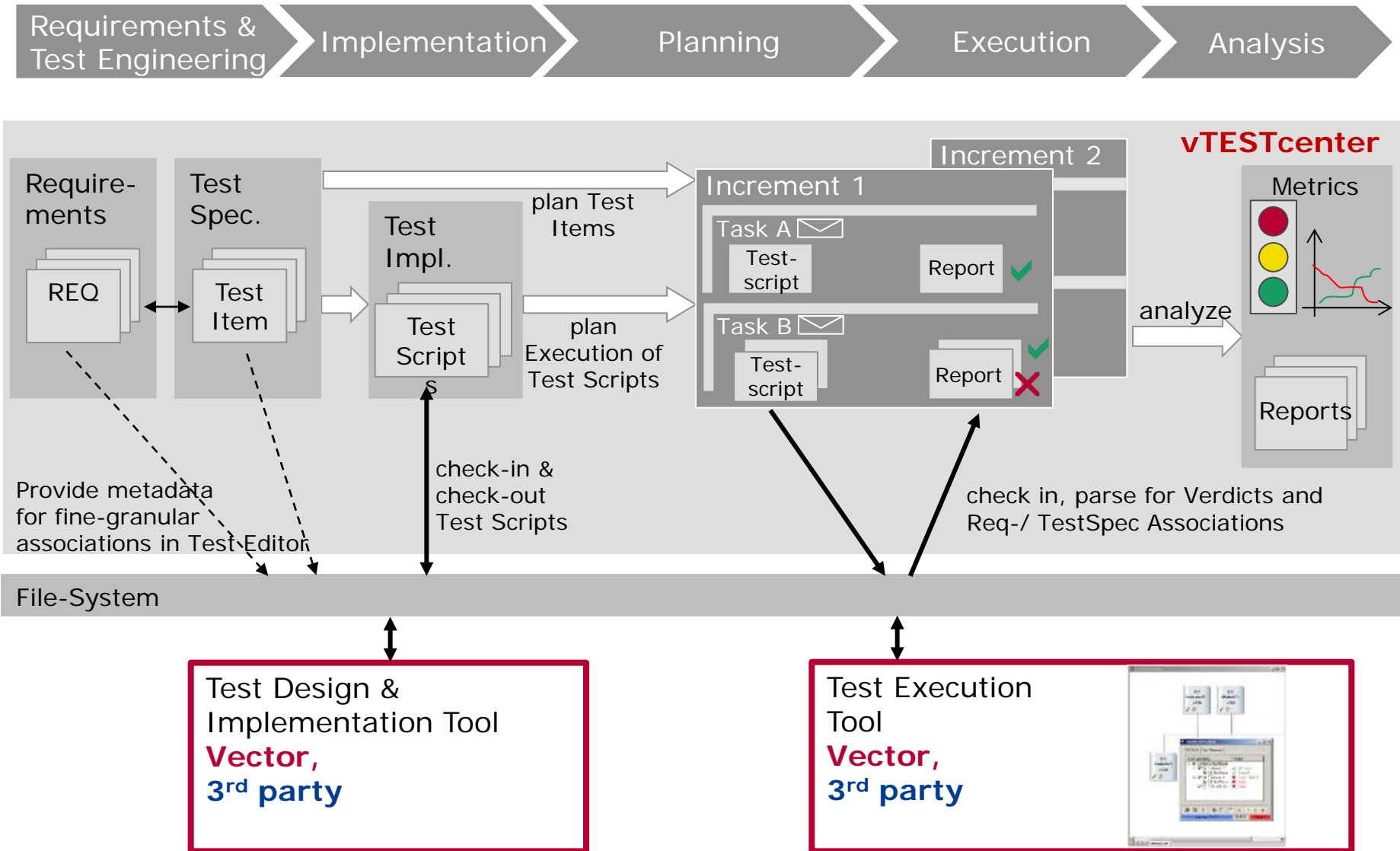
HIL Test Bench (applicable for SIL, MIL, ... as well)

CANoe + VT Modules + Bus Interfaces

- ▶ Real-time execution engine for tests and experiments
- ▶ Access to System under Test via bus systems and protocols (I/Os, diagnostics, XCP, DebugInterface, ...)
- ▶ Huge set of specific test functions
- ▶ Test reporting

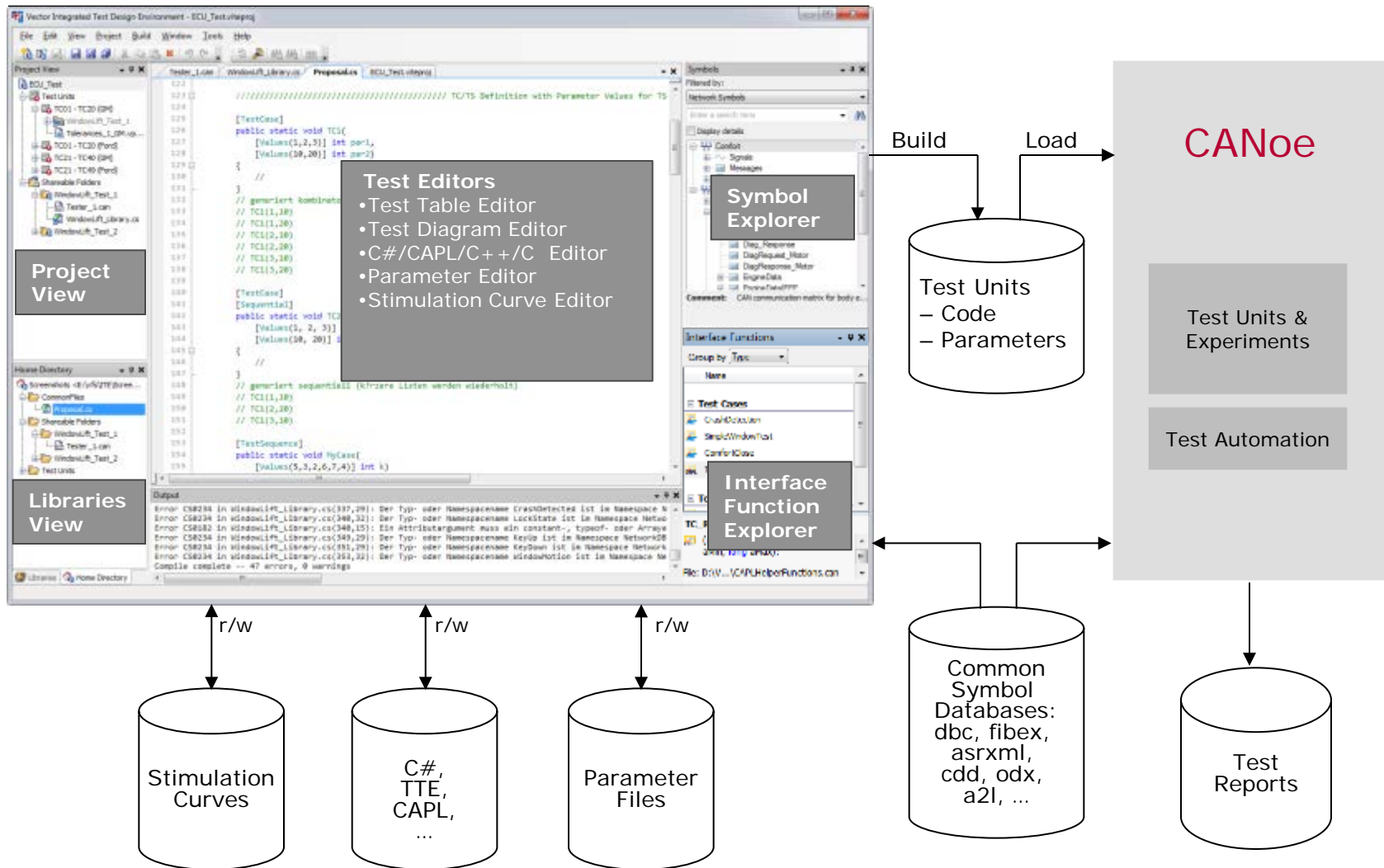
ECU test, simulation and analysis

Vector Testing Solution: vTESTcenter



ECU test, simulation and analysis

Vector Testing Solution: vTESTstudio



ECU test, simulation and analysis

Vector Testing Solution: vTESTstudio

- ▶ Easily define test sequences without programming knowledge
- ▶ Comfortable support of test step parameterization by drag & drop
- ▶ Direct calls to CAPL, C/C++ and C# test cases and functions possible

The screenshot displays the vTESTstudio interface for a test named 'LockingSystemTester.vtt'. The interface is divided into several panes:

- Test Execution Tree:** Shows a hierarchical view of the test structure, including 'Central Locking System Test', 'Test static requirements of the SUT', and 'Test velocity dependent requirements of the central locking system'.
- Command List:** A table listing test steps and their descriptions. The selected command is 'Request to lock the car. Let's see if the car will be locked (engine running)'.
- Stimulation (in):** A table defining input signals for the test.
- Expected (out):** A table defining expected output signals.
- Output:** A pane showing the compilation status: 'Compile complete -- 0 errors, 0 warnings'.

Command	Caption
<i>Central Locking System Test</i>	
<i>Test static requirements of the SUT</i>	
Test Case	Lock statically
Set	Initialize signals of central locking system
Set	Initialize signals of window system
State Change	Request to lock the car. Let's see if the car will be locked (engine running).
State Change	Request to lock the car. Let's see if the car will be locked (engine not running).
<Add command>	
Test Sequence	Statically open/close the window
Set	Initialize all signals
Test Case	Statically open the window
State Change	Start opening the window. Check if the window is really opening.
State Change	Stop opening the window. Check if the window is halted.
Set	Reset all signals
Test Case	Statically close the window
State Change	Start closing the window. Check if the window is really closing.
State Change	Stop opening the window. Check if the window is halted.
Set	Reset all signals
<Add command>	
Test Case	Unlock statically
Set	Initialize all signals
State Change	Request to unlock the car. Let's see if the car will be unlocked.
<Add command>	
<i>Test velocity dependent requirements of the central locking system</i>	
Test Case	Lock by increasing velocity
<Add command>	

Stimulation (in)		
LockRequest	=	Request lock
CrashDetected	=	0
EngineRunning	=	0
<Add signal here...>		
wait	500	

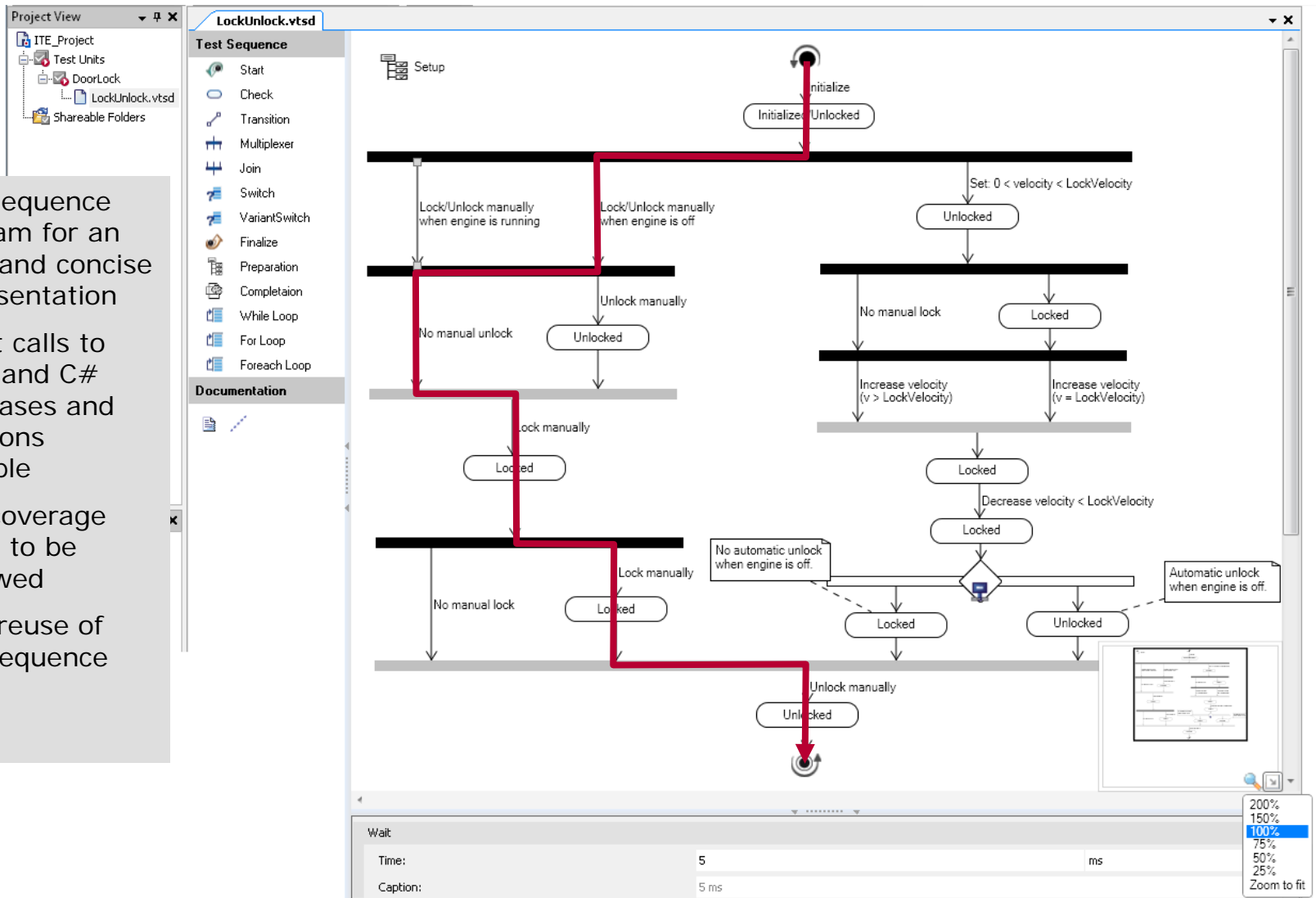
Expected (out)		
LockState	==	Locked
<Add signal here...>		
resettime		
caption	Request to lock the car. Let's see if the car ...	

Output
Compile complete -- 0 errors, 0 warnings

ECU test, simulation and analysis

Vector Testing Solution: vTESTstudio

- ▶ Test sequence diagram for an clear and concise representation
- ▶ Direct calls to CAPL and C# test cases and functions possible
- ▶ Test coverage easily to be reviewed
- ▶ Easy reuse of test sequence parts



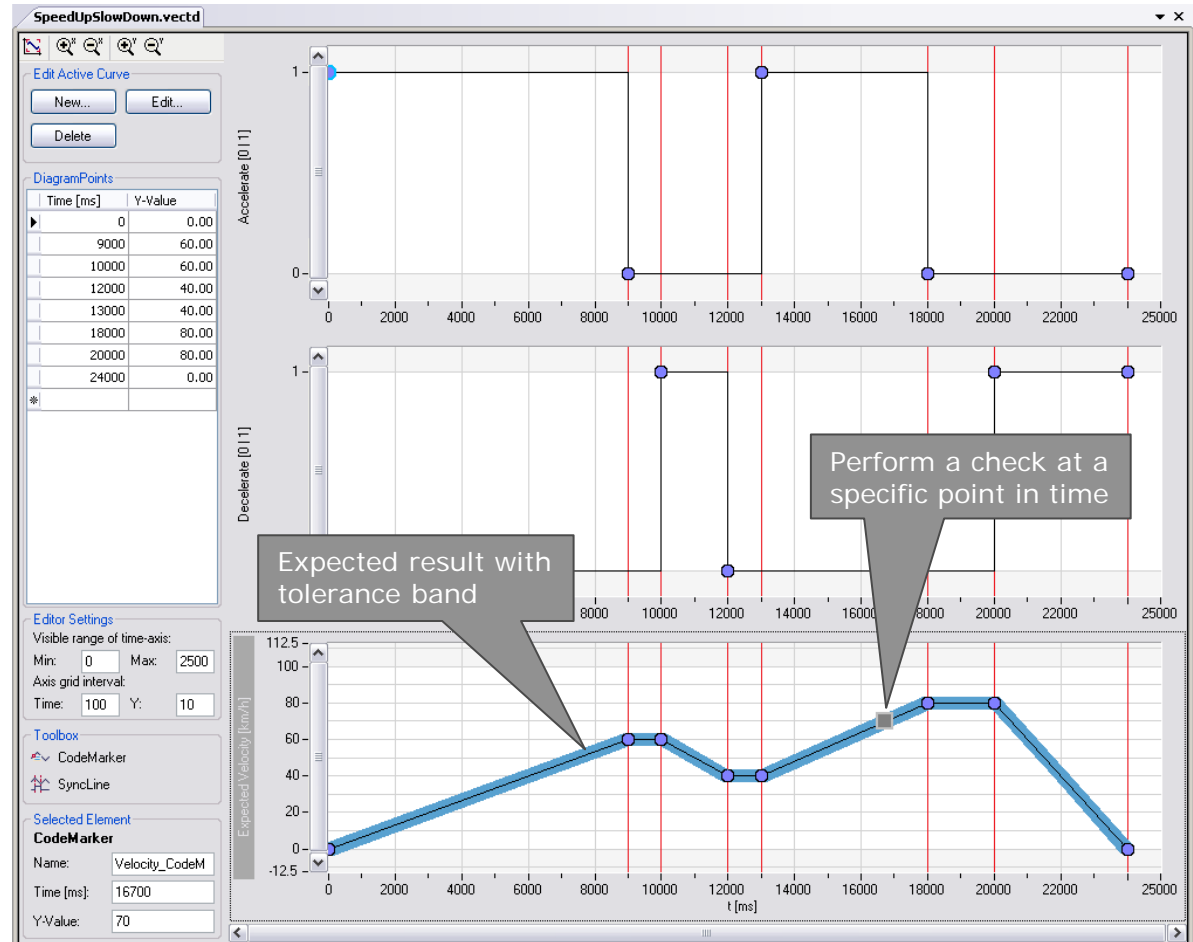
ECU test, simulation and analysis

Vector Testing Solution: vTESTstudio

- ▶ Test automation with Timing Diagrams for expected SUT behavior *

Features:

- ▶ Design of time synchronized test by integrating checks into wave forms at specific points
- ▶ Define wave forms of expected results with tolerances
- ▶ Interactive definition of expected results directly based on measured values



* Planned for 2014

ECU test, simulation and analysis

Vector Testing Solution: vTESTstudio

- ▶ Quick definition of a large number of test cases to increase test coverage
- ▶ Combinations
 - ▶ Sequential
 - ▶ All possible permutations
- ▶ Value definitions
 - ▶ Lists of values
 - ▶ Value ranges
- ▶ Direct use of parameters from parameter files as test case parameters

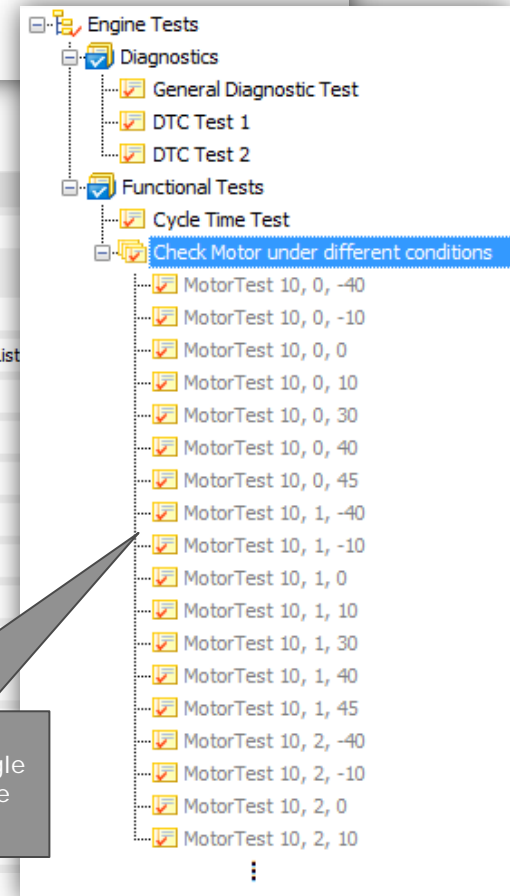
```
public static void MotorTest (  
    [Values(10, 11, 12, 13)] int voltage,  
    [Range(0, 127, 1)] uint dtc,  
    [Values(SysPars.TemperatureList.AttrUsageId)] int temperature)  
{  
    //...  
}
```

Parameter access in coding and table-style editor

Parameter Values			
Combinatorics:	Combinatorial		
Name:	voltage	dtc	temperature
Value Source:	Value List	Range	Parameter List
Range/List:		[min=0, max=127, i...	TemperatureList
	10	0	-40
	11	1	-10
	12	2	0
	13	3	10
	Add...	4	30
		5	40
		6	45
		7	

Use property...: CtrlVar Dependencies
Caption: Check Motor

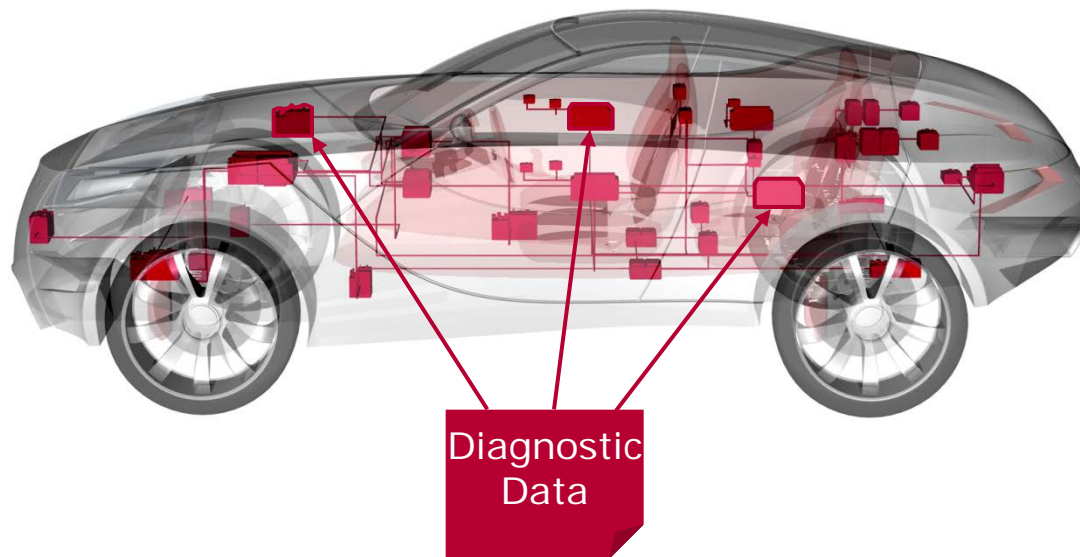
Generated test execution tree, loaded in CANoe. Single generated test cases can be started individually.





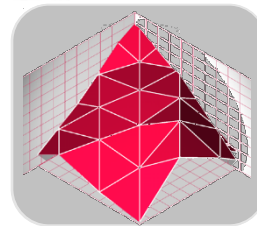
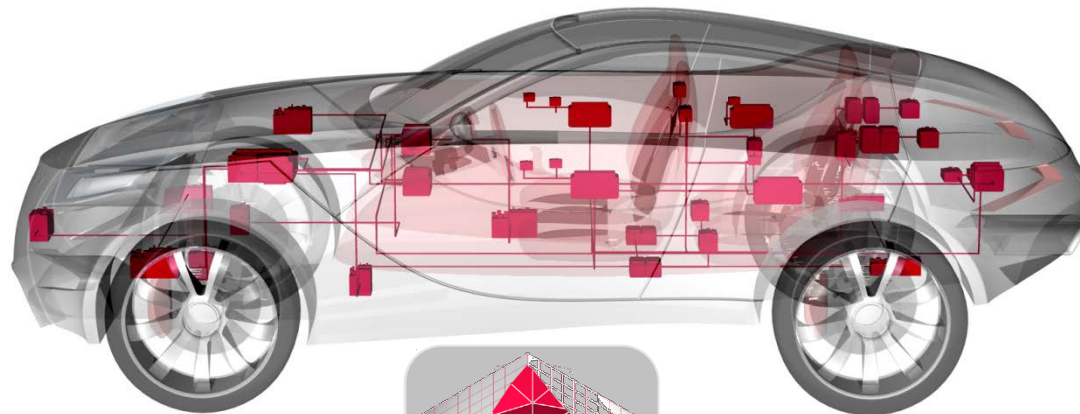
Tools for the complete diagnostic development process:

- ▶ Specification
- ▶ Integration
- ▶ Validation and test
- ▶ ODX data exchange and migration





- ▶ Tools for acquiring measurement data and changing parameters in the ECU during runtime
- ▶ Goal is the optimization of the ECU algorithms



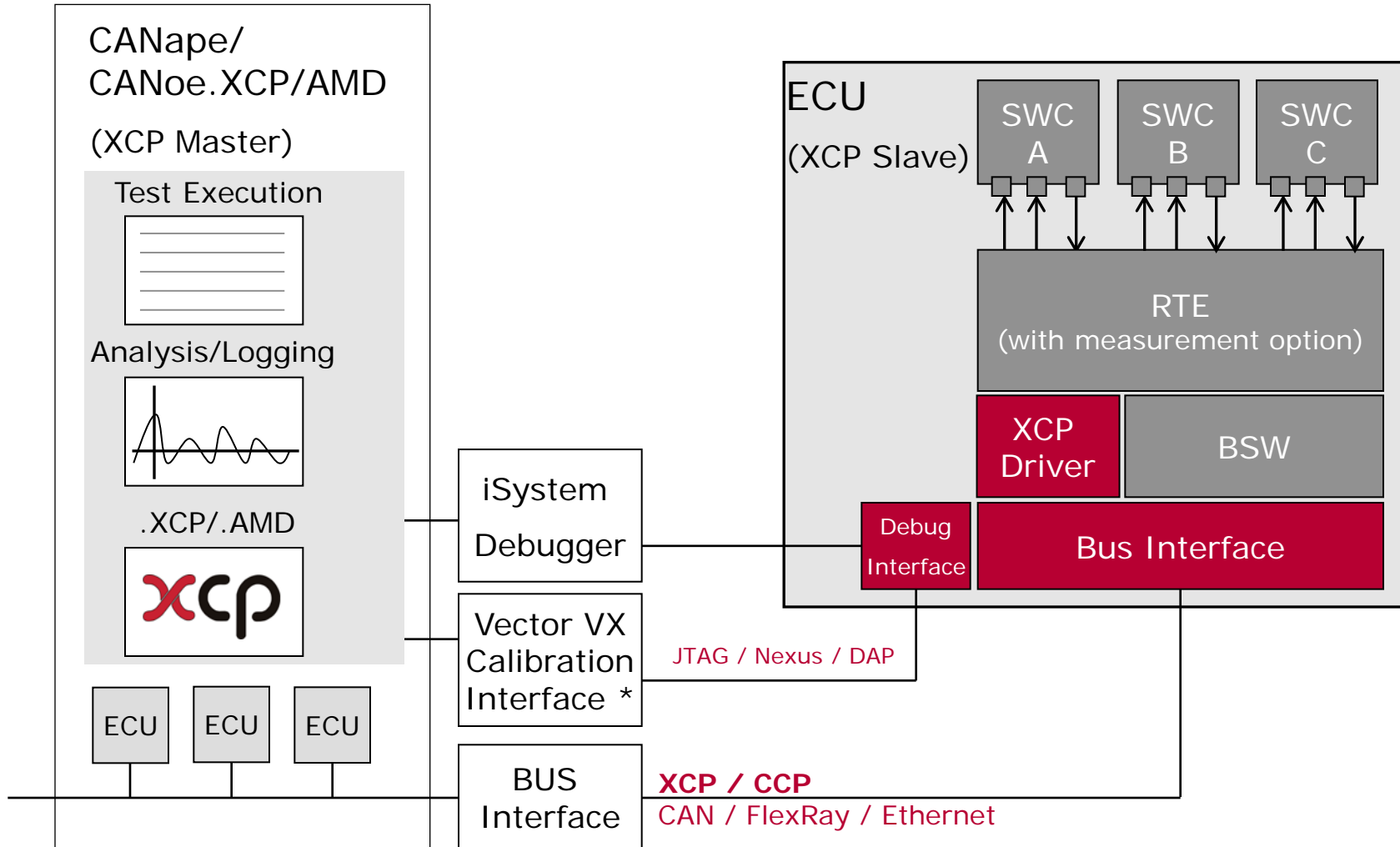
Calibration data

ECU Calibration

XCP access to ECU internal Data



- ▶ Access to ECU via bus or calibration interface (XCP, CCP, VX)



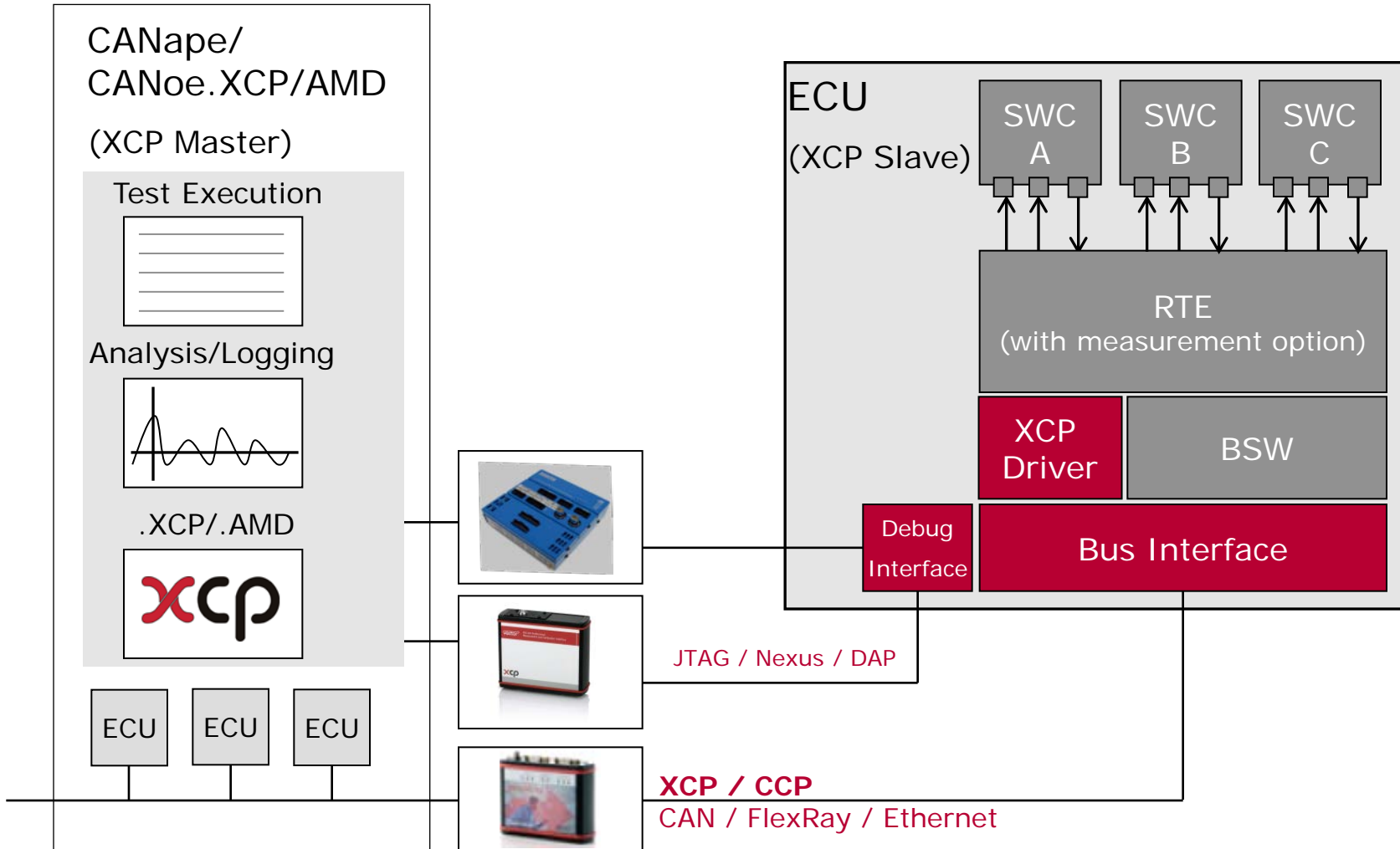
* Up to 30 MB/s of measurement data possible

ECU Calibration

XCP access to ECU internal Data

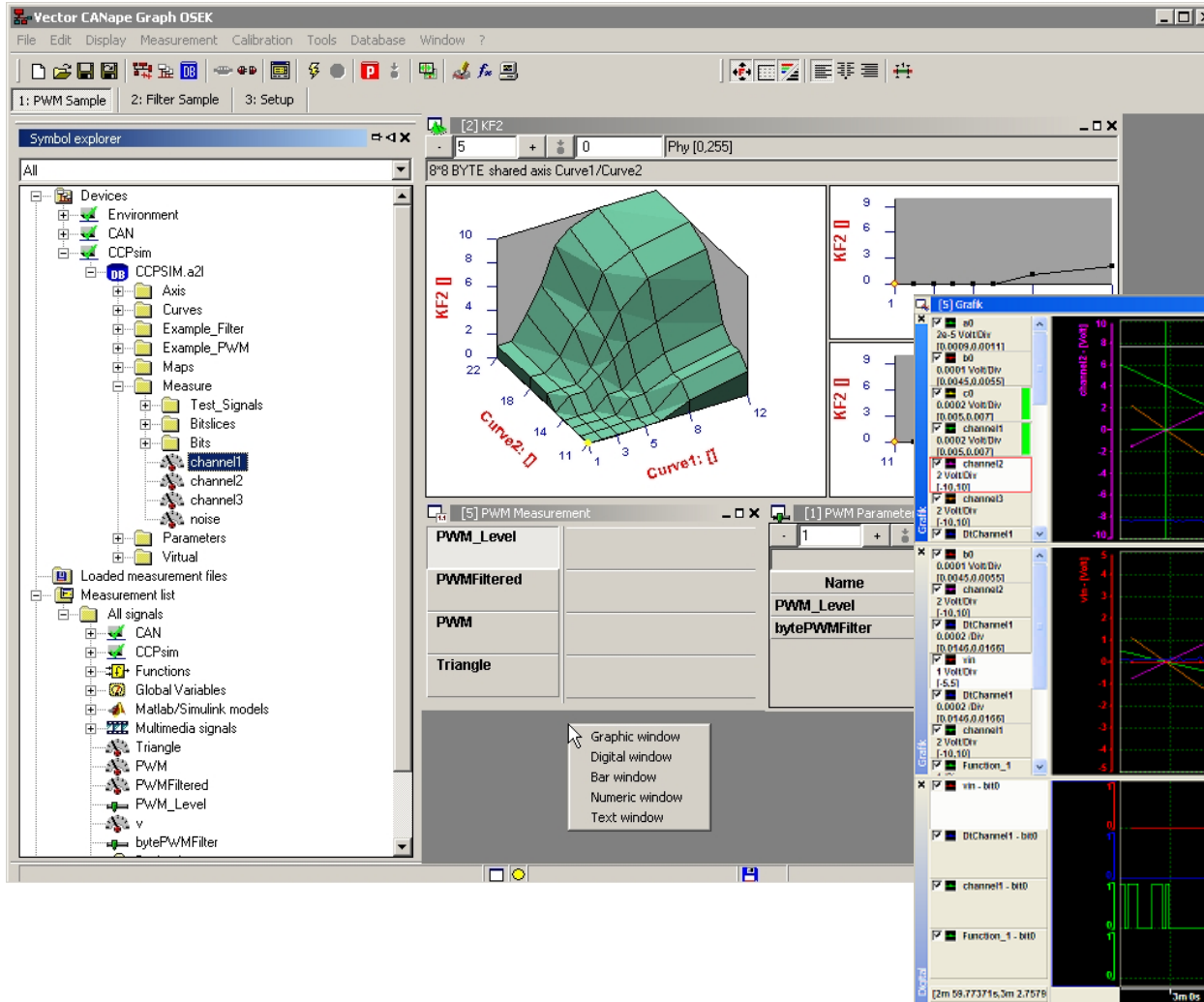


- ▶ Access to ECU via bus or calibration interface (XCP, CCP, VX)





Calibration



Measurement

Vector offers you complete solutions – comprising of products, services, and training - for the methods and standards used in each application area.

Solutions for:

- ▶ AUTOSAR
- ▶ FlexRay
- ▶ CAN
- ▶ LIN
- ▶ MOST
- ▶ IP
- ▶ ...
- ▶ Aerospace
- ▶ E-mobility
- ▶ Open protocols
- ▶ Flashing
- ▶ Logging
- ▶ Safety
- ▶ Special ECUs
- ▶ ...



Thank you for your attention.

For detailed information about Vector
and our portfolio please go to: www.vector.com

thomas.drexler@vector.com

Vector Informatik GmbH
Ingersheimer Str. 24
70499 Stuttgart