

# **vVIRTUALtarget**

## Technical Reference

Version 1.0.6

Authors	Sebastian Lerch, Damian Philipp, Max-Ferdinand Suffel, Manuela Huber
Status	Released

## Document Information

### History

Author	Date	Version	Remarks
Sebastian Bauer, Damian Philipp, Max-Ferdinand Suffel, Manuela Huber	2015-10-09	1.0.0	Initial version
Sebastian Bauer	2015-10-21	1.0.1	Chapter "Project Migration" improved
Sebastian Bauer, Max-Ferdinand Suffel	2015-11-02	1.0.2	Chapter "User-Defined System Variables" added Chapter "Virtual MCU Mode Handling" added
Max-Ferdinand Suffel	2015-12-22	1.0.3	Updated Chapter "Virtual MCU Mode Handling".
Sebastian Bauer	2016-01-12	1.0.4	Updated document for R14
Sebastian Bauer	2016-07-21	1.0.5	Updated layout SPI added as unsupported feature Chapter on Post-Build Selectable added Further details in chapter "Workflow Overview" added
Sebastian Lerch	2017-07-13	1.0.6	Introduction updated Workflow description extended, in particular section on CANoe and Debugging

### Reference Documents

No.	Source	Title	Version
[1]	Vector	Technical References of vVIRTUALtarget MCAL modules Files: TechnicalReference_<Module>_VTT.pdf	see delivery
[2]	Vector	Startup Manual with vVIRTUALtarget File: Startup_<Program>_vVIRTUALtarget.pdf	see delivery
[3]	Vector	Technical Reference of VTTCntrl TechnicalReference_VttCntrl.pdf	see delivery
[4]	Vector	Technical Reference of vVIRTUALtarget OS (Gen7) File: TechnicalReference_Os.pdf	see delivery
[5]	Vector	Technical Reference of vVIRTUALtarget OS (Gen6) Files: MicrosarOS_CANoeEmuVTT.pdf TechnicalReference_Microsar_Os.pdf	see delivery
[6]	Vector	Technical Reference of MICROSAR Identity Manager TechnicalReference_IdentityManager.pdf	see delivery
[7]	Vector	VN8900 Product Information <a href="https://vector.com/vi_vn8900_de.html">https://vector.com/vi_vn8900_de.html</a>	

**Caution**

We have configured the programs in accordance with your specifications in the questionnaire. Whereas the programs do support other configurations than the one specified in your questionnaire, Vector's release of the programs delivered to your company is expressly restricted to the configuration you have specified in the questionnaire.

**Note**

All brand names in this documentation are either registered or non-registered trademarks of their respective owners.

## Contents

<b>1</b>	<b>Introduction.....</b>	<b>7</b>
1.1	Architecture Overview .....	8
1.2	Use Cases of vVIRTUALtarget.....	8
1.2.1	VTT Only.....	8
1.2.2	VTT Dual Target.....	8
1.3	Runtime Environment for vVIRTUALtarget.....	9
1.4	Required Tools and Licenses .....	9
1.5	Scope of this Document .....	9
<b>2</b>	<b>Functional Description .....</b>	<b>10</b>
2.1	Supported Features .....	10
2.2	Representable Features.....	11
2.3	Unsupported Features .....	11
<b>3</b>	<b>Workflow Overview.....</b>	<b>13</b>
3.1	Setting Up a VTT Project .....	14
3.2	Configuring the ECU .....	15
3.3	Generating Code.....	17
3.4	Generating the Solution .....	18
3.5	Execution and Test in CANoe.....	20
3.6	Debugging .....	22
<b>4</b>	<b>Integration Notes .....</b>	<b>24</b>
4.1	Code Integration in Dual Target Projects .....	24
4.2	Virtual MCU Mode Handling.....	24
4.3	User-Defined System Variables .....	26
<b>5</b>	<b>Configuration of Representable Features.....</b>	<b>27</b>
5.1	Hardware OS Counters (Gen6).....	27
5.2	High Resolution Timers (Gen6).....	29
5.3	Background Tasks.....	31
5.4	Handling Different MCU Clock Offsets .....	32
5.5	NvM Block Length Strict Check.....	32
<b>6</b>	<b>Project Migration .....</b>	<b>33</b>
<b>7</b>	<b>Post-Build Selectable .....</b>	<b>35</b>
7.1	Handling DLL names.....	35
7.2	Variant Selection in EcuM .....	35

7.3 Debugging of Variants..... 36

**8 Glossary and Abbreviations ..... 37**

8.1 Glossary ..... 37

8.2 Abbreviations ..... 37

**9 Contact..... 38**

## Illustrations

Figure 1-1	AUTOSAR 4.2 Architecture Overview .....	8
Figure 2-1	OS Published Information with Kernel Version .....	10
Figure 3-1	Workflow of vVIRTUALtarget basic .....	13
Figure 3-2	Selection of target type, case VTT Dual Target .....	14
Figure 3-3	Path to VTT project file in the project wizard .....	15
Figure 3-4	Exemplary set of modules in a VTT project .....	16
Figure 3-5	Overwriting the automatic synchronization using "Set user defined" .....	17
Figure 3-6	Generation dialog with Virtual Target enabled .....	17
Figure 3-7	Option to build VTT project after code generation .....	18
Figure 3-8	Available build steps of vVIRTUALtarget .....	19
Figure 3-9	Referencing a Node Layer DLL in a Node in CANoe .....	20
Figure 3-10	Channel Mapping: Bus name configured in DaVinci Configurator Pro must match the network name in CANoe .....	21
Figure 3-11	System Variables of a virtual ECU that are automatically registered in CANoe .....	21
Figure 3-12	Working Modes in CANoe .....	22
Figure 4-1	Hardware MCU module configured with modes NORMAL, SLEEP, RESET and POWER_OFF. Here, mode NORMAL is identified with mode umber 0, mode SLEEP is configured as mode number 1, mode RESET is configured as mode number 2 and mode POWER_OFF is configured as mode number 3 .....	24
Figure 4-2	All hardware MCU modes are synchronized to the virtual MCU module. In order to simulate the correct mode handling of the hardware MCU, for each hardware MCU mode a suitable virtual MCU mode has to be selected .....	25
Figure 5-1	Hardware timer configured for a hardware OS .....	27
Figure 5-2	VTT OS realizes additional timers as software timers .....	27
Figure 5-3	GTP channel configuration .....	28
Figure 5-4	OsCounters – Adjust the Tick Time .....	30
Figure 5-5	The OS Tick Time should be the greatest common divisor .....	31
Figure 5-6	DaVinci Developer - Add background runnable with background trigger .....	32
Figure 6-1	Alternative module definition dialog for switching module definitions .....	33
Figure 6-2	Button for editing the project settings .....	34

## Tables

Table 2-1	Representable features with limited support .....	11
Table 8-1	Glossary .....	37
Table 8-2	Abbreviations .....	37

# 1 Introduction

vVIRTUALtarget is the Vector solution for running and testing MICROSAR-based ECU software in a virtual environment independent of the real target hardware. The configuration workflow of virtual ECUs is highly integrated into the MICROSAR tool chain. CANoe is used as execution platform and test environment. With its rich test feature set and the possibility to connect and test virtual ECUs with real hardware CANoe is the ideal solution for all tasks during development and test. vVIRTUALtarget comes along with “virtual” OS and MCAL modules that are designed to seamlessly operate with CANoe.

vVIRTUALtarget projects can be used to test ECU software in an early development phase when the hardware is not yet available. Errors in ECU software can be eliminated early in development. In a later development phase the ECU configuration project can be easily migrated to a vVIRTUALtarget Dual-Target project which contains modules for both virtual and real target. The key aspect of the Dual Target approach is that configuration parameters have to be maintained only for the hardware MCAL modules. The configuration parameters of the virtual MCAL modules are automatically derived from the hardware MCAL modules. Non-MCAL configuration parameters are common for both targets. The “single-source principle” enables code generation for vVIRTUALtarget and hardware target from a single configuration project.

## 1.1 Architecture Overview

The following figure shows where the vVIRTUALtarget modules are located in the AUTOSAR architecture.

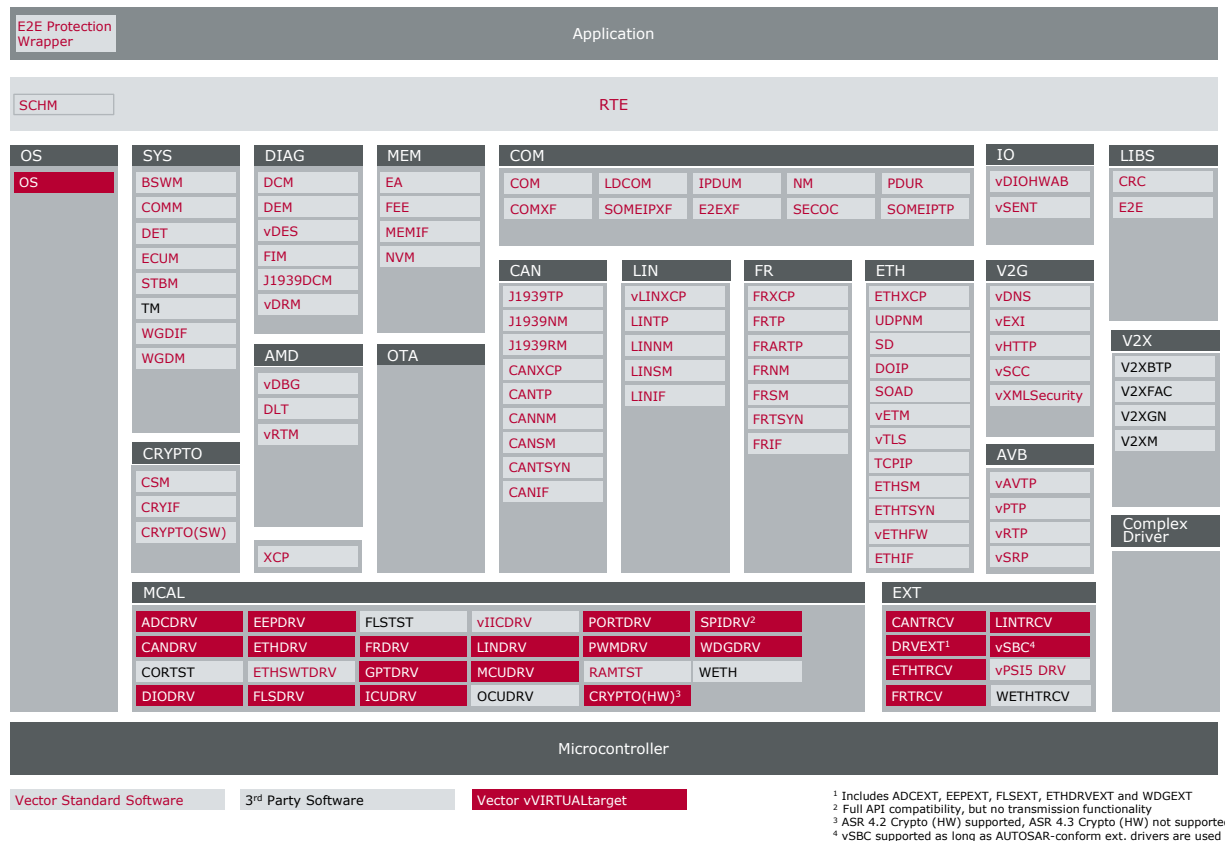


Figure 1-1 AUTOSAR 4.2 Architecture Overview

## 1.2 Use Cases of vVIRTUALtarget

### 1.2.1 VTT Only

A VTT Only delivery contains two sets of OS/MCAL modules: OS/MCAL modules as placeholders for hardware-specific modules and virtual OS/MCAL modules for vVIRTUALtarget. A VTT Only delivery may be used e.g. in the following scenarios:

- > Provide an early delivery even before the hardware-specific modules are available. Developers can start with the configuration of the MICROSAR BSW and test their ECU software on a PC. Later this project can be migrated to a project for a hardware-specific MICROSAR delivery (without vVIRTUALtarget) or a Dual-Target delivery (see 1.2.2) with both virtual and hardware-specific modules.
- > Prototyping and evaluation of the MICROSAR BSW and applications on a developer PC
- > Development of SWCs in a virtual environment

### 1.2.2 VTT Dual Target

A VTT Dual-Target delivery provides two sets of OS/MCAL modules: for the virtual environment and for a specific hardware. Therefore, the application can be executed on hardware as well as in the virtual environment.



VTT Dual-Target projects foster a single source principle. All BSW modules are configured for the hardware target, the configuration parameters of the virtual modules are automatically derived from the hardware modules. In the code generation dialog the user can select either the virtual target or the hardware target. The following use case is addressed with the Dual-Target principle:

- > VTT as a second target platform for prototyping or serial production projects to support tests of hardware-independent aspects of the ECU software

### 1.3 Runtime Environment for vVIRTUALtarget

To ease the setup and compilation of virtual ECUs vVIRTUALtarget comes along with a tool that generates automatically a Microsoft Visual Studio solution. This solution can be used to create a dynamic link library (DLL) comprising the MICROSAR basic software and the application software components (SWCs) that form the virtual ECU.

There are two runtime environments supported:

- > In **CANoe** the DLL can be loaded as a node layer DLL for a network node in the simulation setup. The virtual ECU can be stimulated, its behavior can be observed and tested with the complete functionality of CANoe.
- > **vVIRTUALtarget executor** provides a stand-alone runtime environment that executes the virtual ECU. A Test API in C# is generated for the user as an interface to the virtual ECU.

### 1.4 Required Tools and Licenses

- > **vVIRTUALtarget basic** which is part of the External Components Setup available for download on the Vector website
- > **DaVinci Configurator Pro** License with Option VTT
- > **CANoe** (with options as needed) or **VTT executor**
- > Microsoft Visual Studio 2010 or 2013<sup>1</sup>

### 1.5 Scope of this Document

This document aims at describing features and workflow of vVIRTUALtarget and providing guidelines for configuration and integration steps.



#### Caution

vVIRTUALtarget does not provide a complete and accurate emulation of your hardware ECU. For instance the runtime behavior of the virtual ECU might significantly differ from the execution on the hardware.

vVIRTUALtarget shall aid development and debugging of ECU projects. However, every ECU project must be tested on real hardware.

<sup>1</sup>Available for download on the Microsoft website.

## 2 Functional Description

This chapter lists the supported and unsupported features of VTT. If there is a workaround required, a reference to a detailed description is given.

To describe to which extent VTT supports specific features three classes are used in this section:

> **Supported Features**

Features that are fully functional in VTT with only minor deviations from the behavior on the hardware

> **Representable Features**

Features that are supported API-wise but the behavior may (considerably) deviate from hardware behavior. Representable features may e.g. not allow the simulation of a malfunction. Representable features may also require the implementation of a workaround in order to function.

> **Unsupported Features**

Features that are configured for the hardware target and cause errors in the case of VTT, either during configuration, code generation, compilation or runtime

The features mentioned below concerning the OS will depend on the OS version. To see which version you are using please check your OS configuration: if the container `OsPublishedInformation` can be created and the `OsKernelVersion` is 7 then it is an OS Gen7, otherwise OS Gen6.

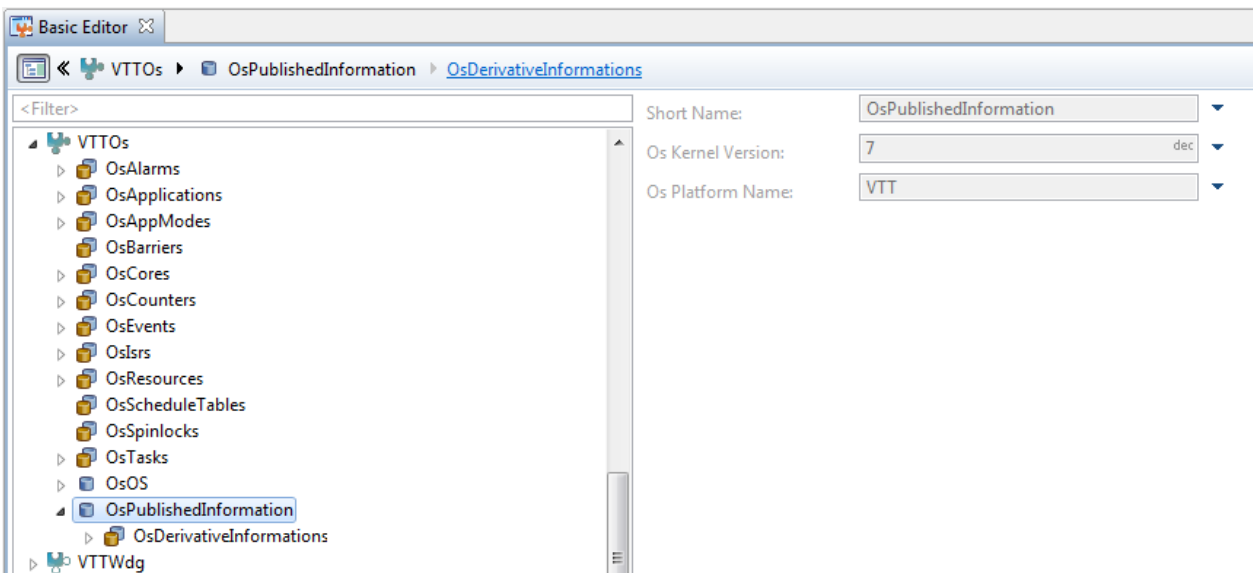


Figure 2-1 OS Published Information with Kernel Version

### 2.1 Supported Features

The supported features of the VTT modules are listed in the Technical References of the modules (see references).

## 2.2 Representable Features

The following table gives an overview of features that are currently only partially supported by VTT.

BSW Module	Feature	Limitation	Workaround
OS Gen6	Hardware Counters	Only one single hardware counter is supported. Further (software) counters must be incremented manually.	See 5.1
OS Gen6	High Resolution Timer	High Resolution Timers can be emulated by manual configuration.	See 5.2
OS Gen6 and Gen7	Background Tasks	In background tasks the simulation time must be manually advanced. Long-running tasks that do not advance simulation time are detected by the runtime kernel and the simulation is stopped.	See 5.3
MCU	Clock Offset	Offset mismatches between hardware MCU and virtual MCU must be handled by adapting code.	See 5.4

Table 2-1 Representable features with limited support

## 2.3 Unsupported Features

### > Post-Build Loadable

Post-build loadable can be selected as implementation variant in VTT modules, however, the post-build time flash process is not supported and will have to be tested on the hardware.

### > Precision Time Protocol

As VTT does not provide an accurate timing simulation, time synchronization protocols are not supported.

### > Ethernet Hardware Time Stamping

Ethernet Hardware Time Stamping is not supported in VTT.

### > Partial Networking

Transceiver drivers have limited functionality; in particular, VTT does not support simulation of partial networking for selective wake-up of an ECU.

### > SPI Communication

VTT does not support SPI transmission functionality, i.e. the VTT SPI driver only serves as a stub.

### > Multiple Instances of Drivers

VTT cannot be instantiated multiple times in a single configuration project.

### > AUTOSAR 4.3 Crypto Driver Hardware Support

VTT does not support a Crypto driver (HW) according to AUTOSAR 4.3.

The following unsupported features are only relevant for the VTT OS Gen6.

- > **OS Scalability Classes SC2 – SC4**  
VTT OS Gen6 only supports Scalability Class 1.

- > **MultiCore OS**  
VTT OS Gen6 does not support Multi-Core OS configurations.

The following unsupported features are only relevant for the VTT OS Gen7.

- > **Memory Protection (SC3)**  
Memory protection (SC3) can be configured in the OS. However, the actual protection mechanism is not executed.

### 3 Workflow Overview

This section gives an overview of the workflow with vVIRTUALtarget. Figure 3-1 shows the involved tools and artifacts. The first column in the figure with DaVinci Developer and DaVinci Configurator Pro does not differ from the usual workflow for an ECU project for hardware. The second and third columns illustrate the VTT-specific workflow for building and running a virtual ECU. The essential steps of the workflow are described in the following in more detail.



#### Note

The figure below shows the workflow with **CANoe** as runtime environment, see 1.3. For usage with **vVIRTUALtarget executor** additionally a Test API and a configuration file is generated. More information can be found in the online help of vVIRTUALtarget executor.

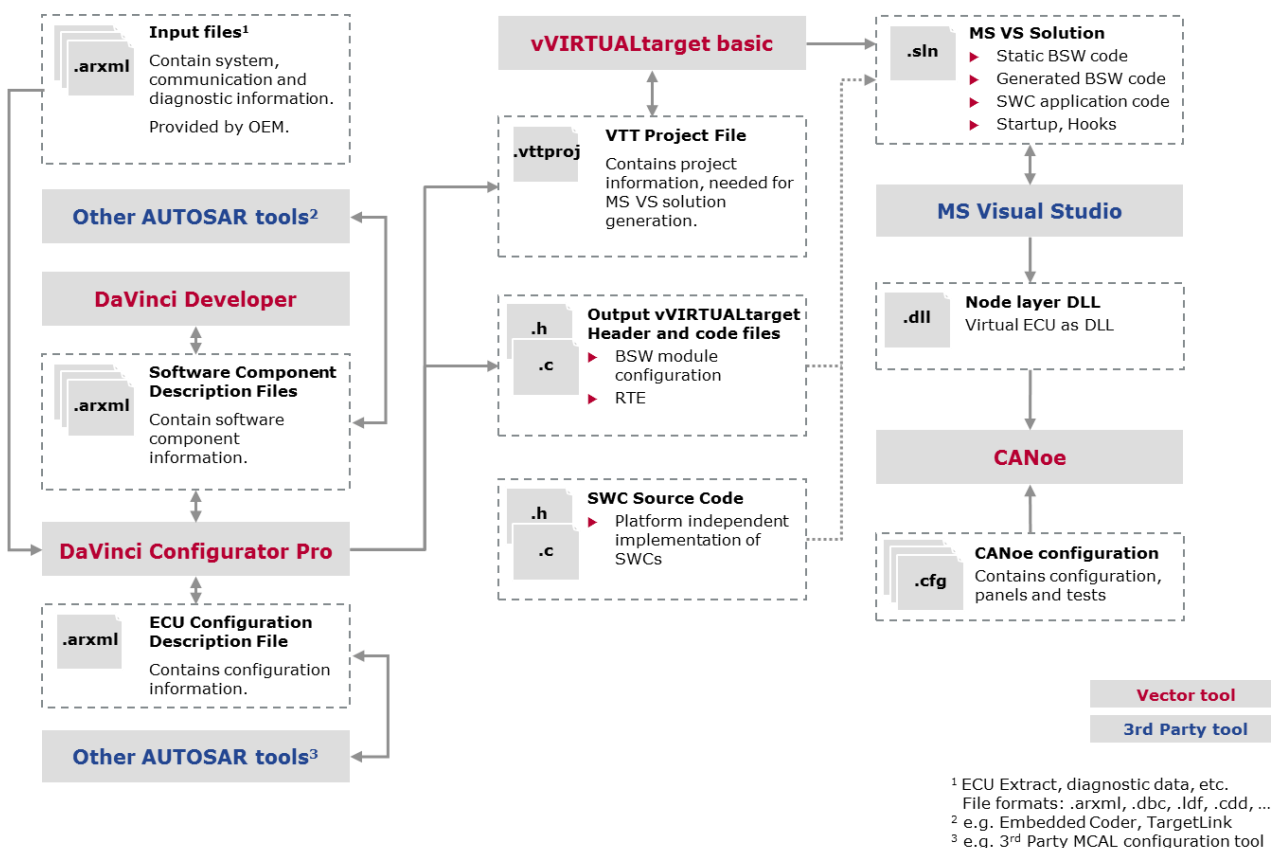


Figure 3-1 Workflow of vVIRTUALtarget basic

**Reference**

For the general introduction to a MICROSAR SIP, how to setup a new project in DaVinci Configurator Pro and how to configure the most important BSW modules please read the startup manual first, see [2].

### 3.1 Setting Up a VTT Project

In the project creation wizard of DaVinci Configurator Pro, the available target types depend on your MICROSAR delivery. Figure 3-2 shows an example of a project for a VTT Dual-Target SIP.

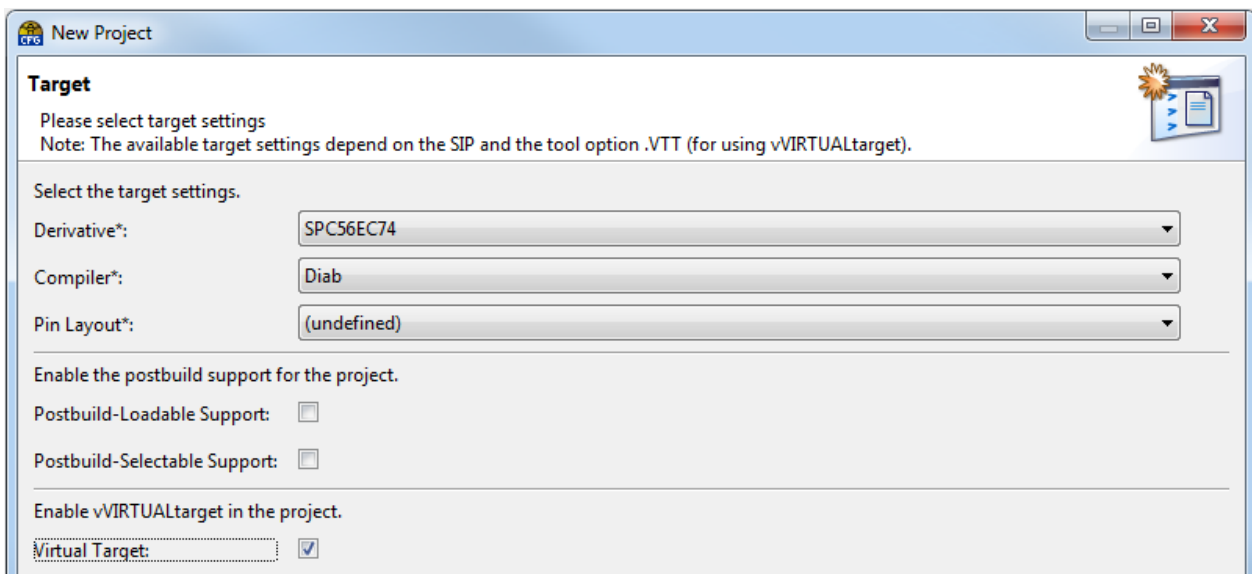


Figure 3-2 Selection of target type, case VTT Dual Target

In case of VTT Only the option Virtual Target is preselected and cannot be changed.

In the vVIRTUALtarget page of the project wizard the vVIRTUALtarget application must be configured, see Figure 3-3. The vVIRTUALtarget project file (extension .vttproj) is needed for code generation in DaVinci Configurator Pro and later for the generation of the Visual Studio solution in the vVIRTUALtarget application. The project wizard also needs a reference to the command line application of vVIRTUALtarget (VttCmd.exe). You can use the latest application (recommended), or specify a path to a specific version of VttCmd.exe if more than one version of vVIRTUALtarget is installed.

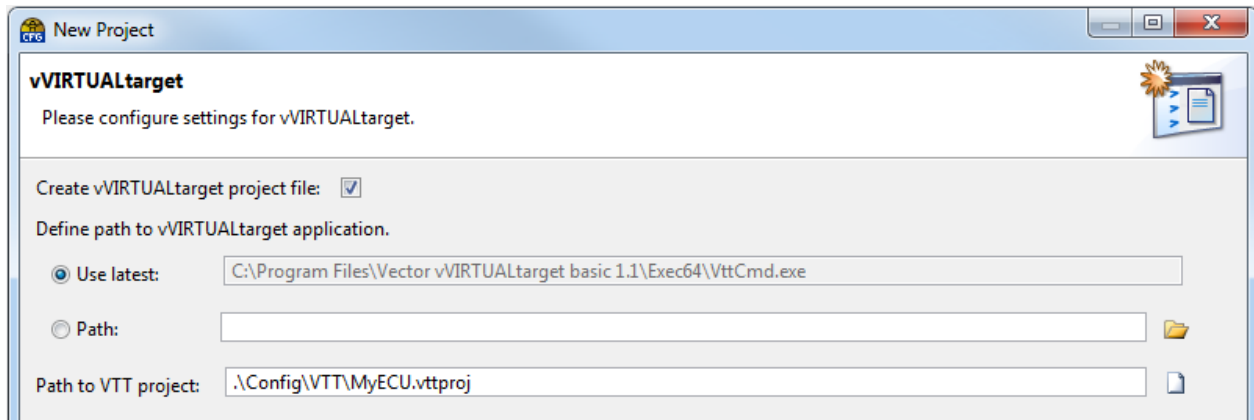


Figure 3-3 Path to VTT project file in the project wizard

### 3.2 Configuring the ECU

Once the input files have been defined and an ECU update has been executed (see startup manual [2] for a step-by-step description), missing VTT modules can be added in the **Project Settings Editor**.

At least the modules VTTCntrl and VTTEcuC must be present. Moreover, for each hardware-dependent BSW module <ModuleName> the corresponding VTT module VTT<ModuleName> must be present. This correspondence between hardware module and VTT module is of importance for the synchronization of configurations explained in the next section. Figure 3-4 below shows an example of a correct and complete selection of modules.



#### FAQ

In a VTT Only SIP there is also the need to enable for each MCAL module both, the “placeholder” MCAL that replaces the hardware MCAL as well as the corresponding VTT module. Technically a VTT Only SIP uses the same synchronization mechanisms as a Dual-Target SIP, however, instead of hardware MCAL modules special placeholder MCAL modules are provided. These placeholder modules do not come with a code generator.

Most parts of the configuration of VTT modules, the modules with prefix “VTT” in Figure 3-4 below, are derived from the configuration of the corresponding placeholder (VTT Only) or hardware (VTT Dual-Target) modules. The synchronization happens in the background as soon as configuration parameters are changed in the non-VTT modules. In the configuration of VTT modules synchronized configuration parameters are greyed out and cannot be changed. It is still possible to override them for special cases by setting them to user-defined, explained in the following.

**Note**

The VTT modules VTTcntrl and VTTEcuC have to be configured manually as these modules are VTT specific. The configuration of ECUC is not synchronized with the configuration of VTTEcuC.

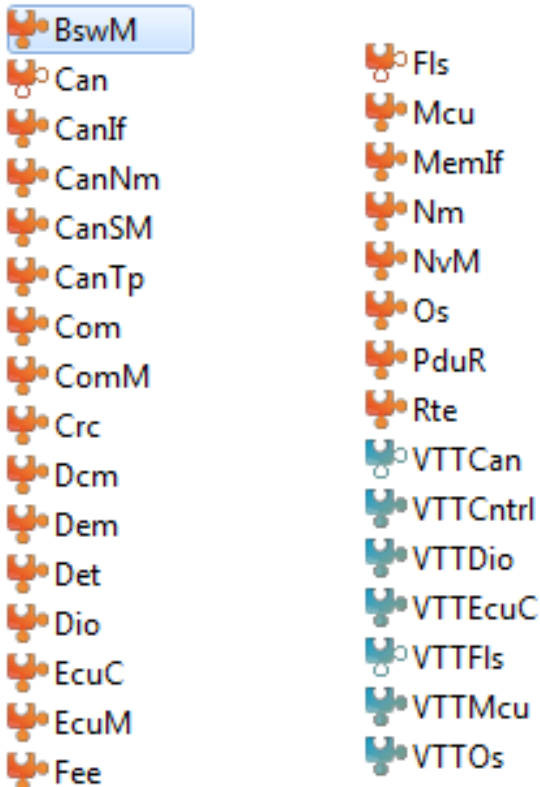


Figure 3-4 Exemplary set of modules in a VTT project

**Caution**

Interrupts configured in the hardware OS are not synchronized to the VTT OS configuration. The reason is that supported interrupts may differ between hardware and VTT, hence there is no synchronization possible at this point.

To assist the user with configuring the interrupt table in VTT OS, VTT drivers automatically register their needed interrupts. Any custom interrupts in the hardware OS must be manually copied to the VTT OS.

In practice it can happen that due to a required workaround the configuration settings of VTT modules deviate from those of the hardware MCAL modules. In this case it is possible to disable automatic synchronization for a certain parameter and to override its value.

Whenever a configuration parameter for a VTT module must be set to a user-defined value and this parameter is locked (greyed out, because it is synchronized with the configuration parameter of the hardware MCAL), right click on the parameter name and select “Set user



defined”, see Figure 3-5. The parameter can now be changed. The parameter will not be synchronized as long as it is set to user-defined. Validation errors linked to user-defined parameter are downgraded to warnings.

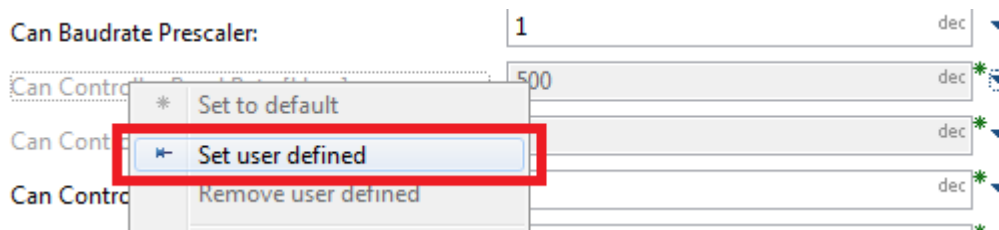


Figure 3-5 Overwriting the automatic synchronization using “Set user defined”

### 3.3 Generating Code

The option Virtual Target activates VTT in your project and adds the generation target “Virtual Target (vVIRTUALtarget)” to the code generation dialog, see Figure 3-6.

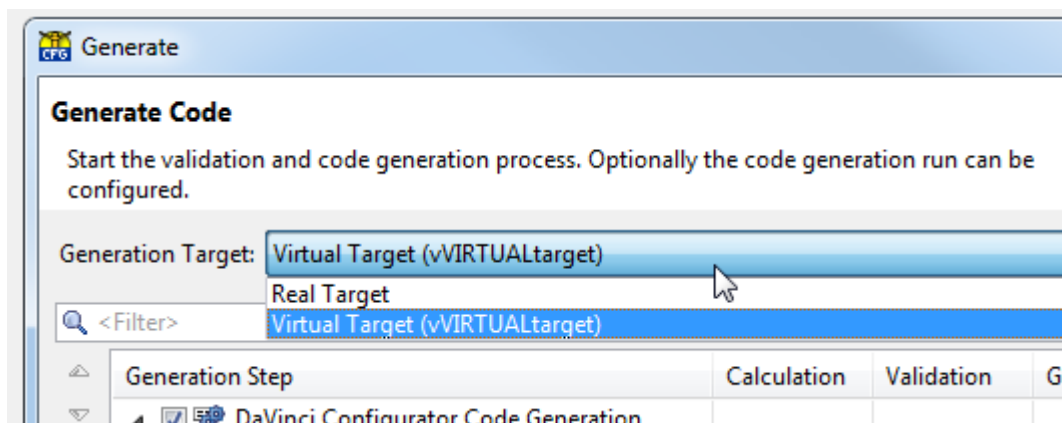


Figure 3-6 Generation dialog with Virtual Target enabled

In case of VTT Only there is only one option possible for the generation target, namely “Virtual Target (vVIRTUALtarget)”.

If the setting “Build VTT Project” is enabled in the project settings, see Figure 3-7, the tool vVIRTUALtarget is called after successful code generation and all the build steps configured in the .vttproj file are executed.

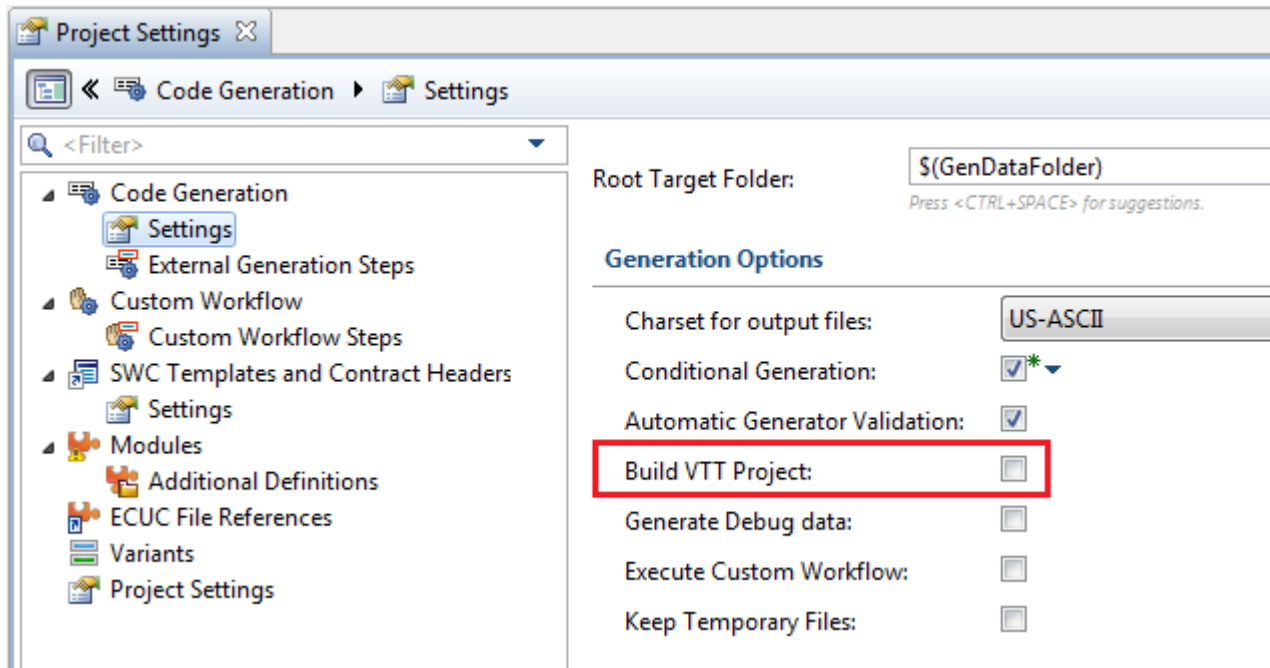


Figure 3-7 Option to build VTT project after code generation

### 3.4 Generating the Solution

As soon as validation and code generation of the configuration project are successful, the .vttproj file of this project can be opened in the tool vVIRTUALtarget. The main purpose of the tool is to generate a Microsoft Visual Studio solution to create a DLL for the virtual ECU. The most important configuration steps in the tool include:

- > Define application source files that shall be included in the solution, in addition to the BSW code that is added by default.
- > Define include directories that shall be set in the solution.
- > Define static libraries and further configuration settings like compiler and linker options.
- > Choose between the available build options:
  - > Generate solution
  - > Build software
  - > Generate test API.
- > Choose a runtime environment: either CANoe or VTT executor.

For a detailed description of the different options and configuration parameters please read the online help of vVIRTUALtarget.

Figure 3-8 shows the build tab of the tool. If CANoe is used as runtime environment, the build step "Generate test API" is not necessary. The build step "Build software" is additionally initiates the compile and link process for the generated solution without opening the IDE. You can also deactivate this option, open the solution in Visual Studio by clicking the "Open IDE" button and manually start the build process. This is in particular recommended for analyzing possible compiler problems.

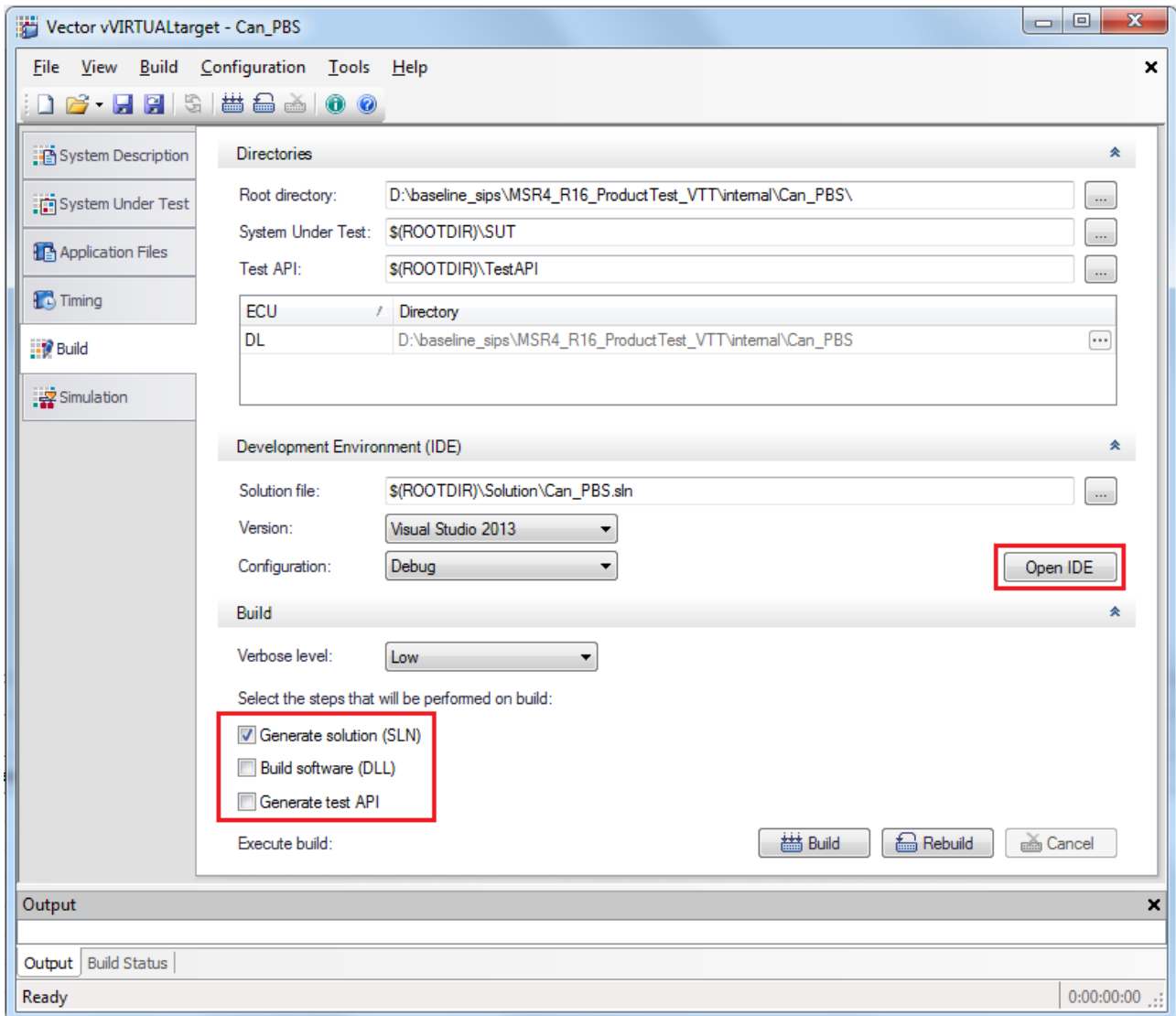


Figure 3-8 Available build steps of vVIRTUALtarget

If the parameter “Build VTT Project” is activated in the DaVinci Configurator Pro, see Figure 3-7, then the build steps configured in vVIRTUALtarget are executed. Activating the option in DaVinci Configurator Pro and activating “Generate solution” and “Build software” in vVIRTUALtarget allows you to generate code and build a new DLL with a single click.

The following files are generated by vVIRTUALtarget and automatically added to the solution:

- > CANoeEmu.rc  
This resource file provides version information for the DLL.
- > CANoeEmu\_cfg.c  
This source file provides handler functions for the runtime library.
- > Vtt\_Hook.c and Vtt\_Hook.h  
These files provide hook and handler functions for VTT.

After each successful build in Microsoft Visual Studio the DLL of the virtual ECU is automatically copied to a “System under test (SUT)” subfolder of your project. The used directory can be configured in the build tab.

The main entry point is configurable in VTTCntrl in DaVinci Configurator Pro. Either EcuM\_Init can be configured, or alternatively a main function e.g. “main()” is configured that must then be implemented by the integrator. In the simplest case the main function looks as follows:

```
void main(void)
{
    EcuM_Init();
}
```

### 3.5 Execution and Test in CANoe

The DLL can be used in CANoe as node layer DLL for a network node. In the Node Configuration window the DLL can be added as a component of the node, see Figure 3-9. In general no CANoe interaction layer is needed in this node as the virtual ECU is responsible for all bus communication.

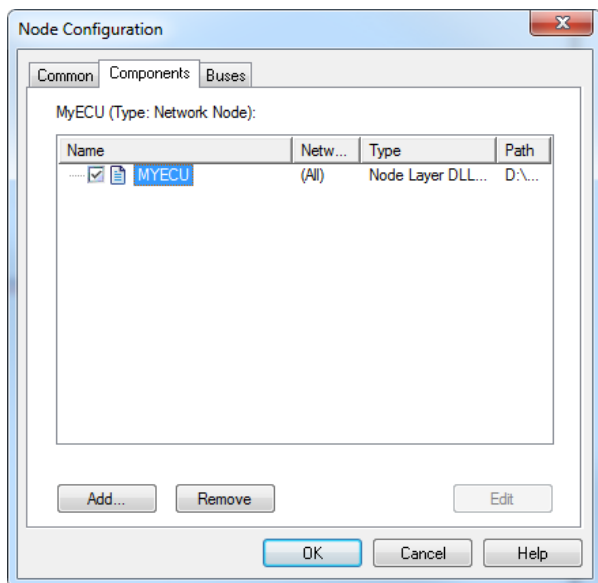


Figure 3-9 Referencing a Node Layer DLL in a Node in CANoe

It is important that the network channels have exactly the same names as specified in the VTT module configuration. The bus names configured in the VTT module configurations in the DaVinci Configurator Pro must match the names of the networks in CANoe. See Figure 3-10 for an example.

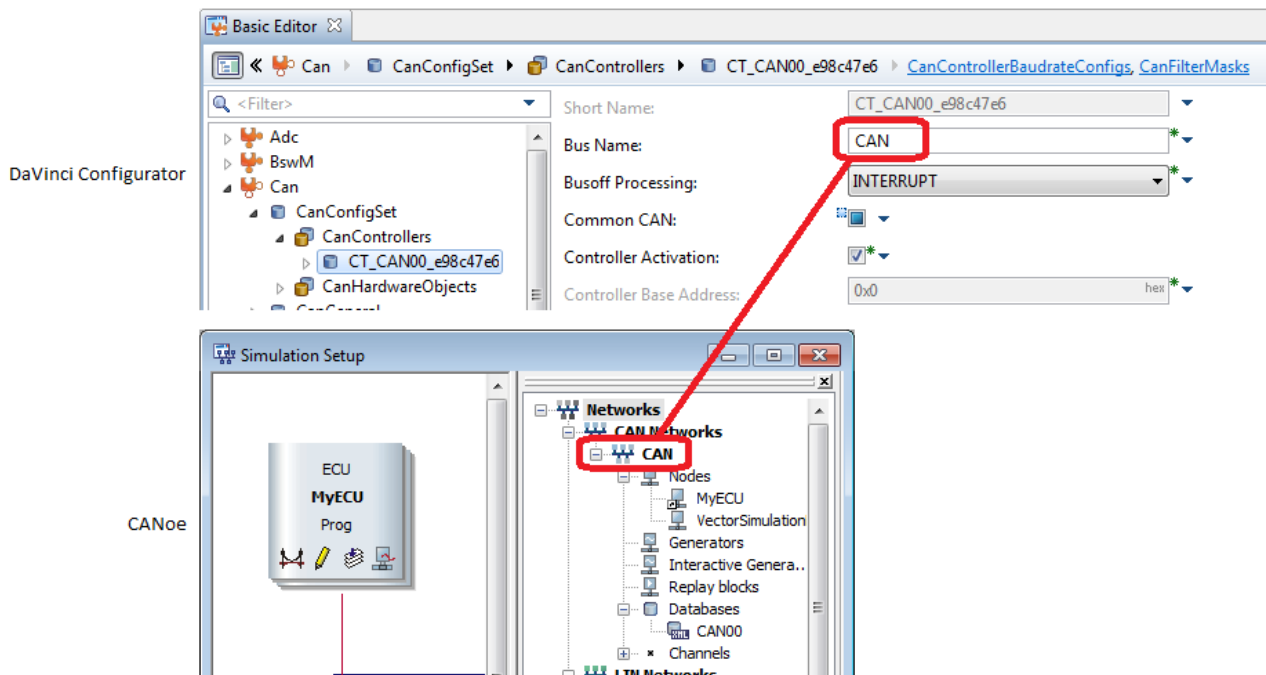


Figure 3-10 Channel Mapping: Bus name configured in DaVinci Configurator Pro must match the network name in CANoe

When the DLL is loaded any system variables defined by the virtual ECU are registered and become visible in CANoe, see Figure 3-11. The system variables are accessible in e.g. test modules or panels, for automated or interactive tests.

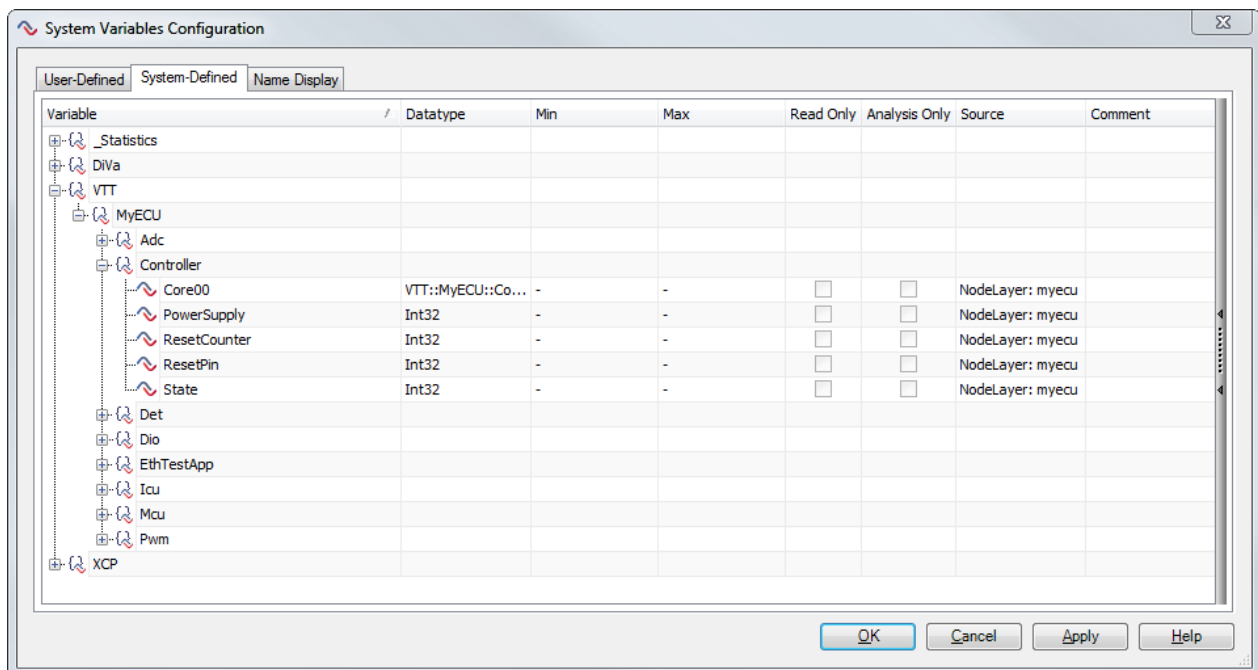


Figure 3-11 System Variables of a virtual ECU that are automatically registered in CANoe

The system variables of the MCAL modules are explained in the technical references [1]. In the namespace Controller there are several system variables for the status of the emulated controller.

- > Core<x>.ErrorReason: This variable shows the error reason of the last error in the core <x>.
- > Core<x>.State: This variable shows the current state of the core.
- > PowerSupply: With this variable the power supply can enabled or disabled. By disabling and enabling the power supply the ECU can be restarted.
- > ResetCounter: This variable counts the number of resets since the last measurement start.
- > ResetPin: A reset of the ECU is performed whenever the value switches to 1.
- > State: This variable shows the simulation state of the ECU in CANoe.

There are three possible working modes in CANoe, see Figure 3-12. In Real Bus mode CANoe can communicate with real buses via a hardware interface. The virtual ECU is one additional node that is simulated in parallel. In Simulated Bus mode CANoe can operate without hardware and all buses are completely simulated.

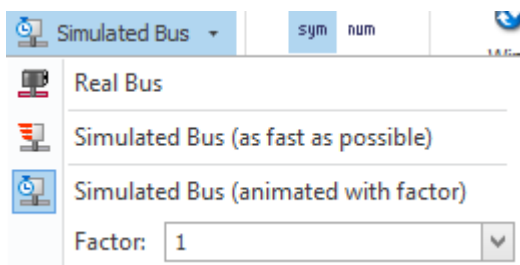


Figure 3-12 Working Modes in CANoe

In Simulated Bus mode there are two options:

- > **As fast as possible**  
In this case the simulation does not run with constant speed but as fast as possible. The performance depends mainly on the performance of the computer and on the complexity of the simulation.
- > **Animated with factor**  
In this mode the simulation runs with constant speed according to the given factor. The measurement is either slowed-down (factor > 1) or accelerated (factor < 1) according to the given factor value.

### 3.6 Debugging

Prerequisite to debugging is that the virtual ECU has been built with Debug symbols. The build configuration (Debug or Release) can be set already in vVIRTUALtarget in the build tab.



### Performance of Debug Builds

A Debug build of a virtual ECU can be substantially slower than a Release build. For executing a virtual ECU with real bus and real hardware it is highly recommended to use the Release build and to use a VN89xx to run the CANoe simulation. Further information can be found in the product information on the Vector Website [7].

To debug the virtual ECU, the solution has to be opened in Microsoft Visual Studio. Go to **Debug|Attach to process**, select native code as code type and select the RuntimeKernel.exe of the CANoe instance where the virtual ECU is running. If only a single instance of CANoe is running, it is sufficient to press <F5> or select **Debug|Start debugging**.



### Note

In the Express edition of Microsoft Visual Studio 2010 the debug commands have to be initially activated by enabling the expert settings by option **Tools|Settings|Expert Settings**.

The Debug build has also the advantage that CANoe automatically reloads the DLL when the measurement is stopped. This speeds up the development and debugging because CANoe does not need to be closed. Just make sure that measurement is stopped before you start the compilation in Visual Studio.

## 4 Integration Notes

### 4.1 Code Integration in Dual Target Projects

Within your application or integration code it might be required to implement hardware and/or VTT-specific code. In order to differentiate the target the following `#define` can be used to differentiate the target.

```
#define _MICROSOFT_C_VTT_
```

If the define is set the code is compiled for vVIRTUALtarget. If it is undefined the current compile process is intended for the hardware. Use a simple preprocessor `#ifdef` statement to assess the target.

In this way complex drivers can be adapted to also work in the virtual environment. Any direct hardware accesses or accesses to the MCAL module SPI (which has no transmission functionality in VTT, see [1]) can be replaced by accesses to system variables. System variables allow communication with the test environment. How user-defined system variables can be configured and used is described in 4.3.

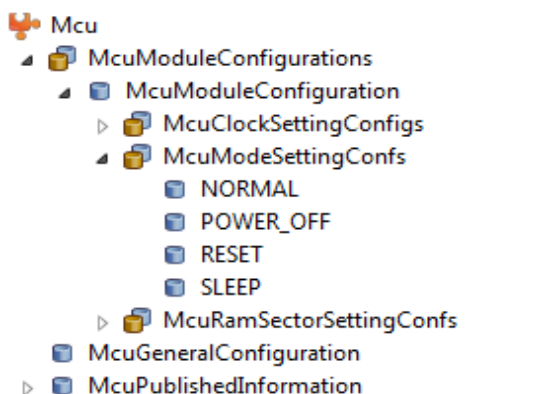
### 4.2 Virtual MCU Mode Handling

The virtual MCU offers four modes:

- > `VTTMCU_MODE_NORMAL`, the MCU is up and running.
- > `VTTMCU_MODE_SLEEP`, the MCU is sleeping.
- > `VTTMCU_MODE_RESET`, the MCU is executing a (software) reset.
- > `VTTMCU_MODE_POWER_OFF`, the MCU is powered off.

When working in a Dual Target project a suitable MCU mode mapping has to be configured in order to simulate the correct mode handling of the hardware MCU.

For this purpose, you need first to configure the MCU modes in the hardware MCU module. Assume the modes NORMAL (0), SLEEP (1), RESET (2) and POWER\_OFF (3) are configured as shown in Figure 4-1.



McuModeSettingCor	Mode
NORMAL	0
SLEEP	1
RESET	2
POWER_OFF	3

Figure 4-1 Hardware MCU module configured with modes NORMAL, SLEEP, RESET and POWER\_OFF. Here, mode NORMAL is identified with mode number 0, mode SLEEP is configured as mode number 1, mode RESET is configured as mode number 2 and mode POWER\_OFF is configured as mode number 3.



The configured modes of the hardware MCU will be synchronized to the virtual MCU module. However, one has to specify for each hardware MCU mode to which of the supported virtual MCU modes it maps, refer to Figure 4-2. By default, each hardware MCU mode is mapped to virtual MCU mode `VTTMCU_MODE_NORMAL`.

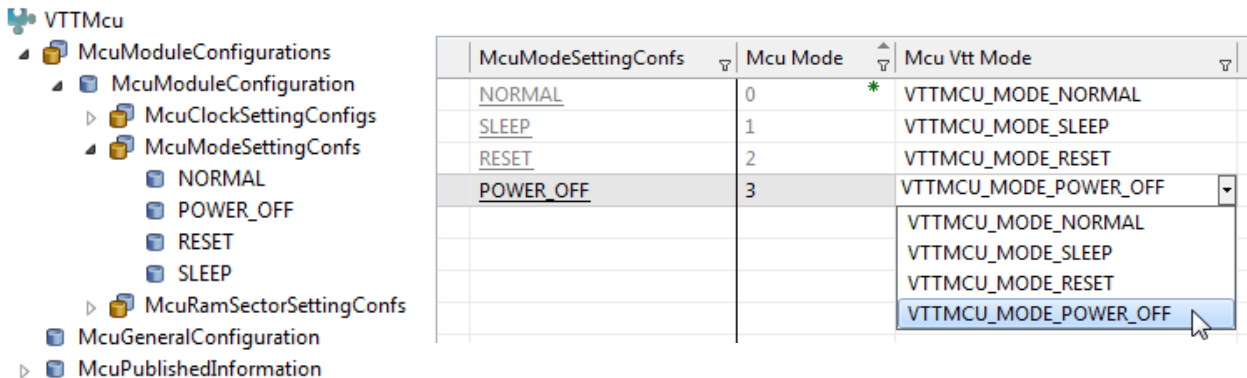


Figure 4-2 All hardware MCU modes are synchronized to the virtual MCU module. In order to simulate the correct mode handling of the hardware MCU, for each hardware MCU mode a suitable virtual MCU mode has to be selected.



#### Note

Multiple hardware MCU modes can be mapped to the *same* virtual MCU mode. For example, if the hardware MCU supports two distinct sleep modes, both can be mapped to the single virtual MCU mode `VTTMCU_MODE_SLEEP`.

After code generation with DaVinci Configurator Pro, the symbolic name values for the hardware MCU modes are accessible via header file `Mcu_Cfg.h`, e.g.:

```
#define McuConf_McuModeSettingConf_RESET      (2u)
#define McuConf_McuModeSettingConf_NORMAL     (0u)
#define McuConf_McuModeSettingConf_POWER_OFF  (3u)
#define McuConf_McuModeSettingConf_SLEEP      (1u)
```

These symbolic name values can be used in the EcuM Callouts `EcuM_AL_Reset`, `EcuM_AL_SwitchOff` and `EcuM_McuSetMode` in file `EcuM_Callout_Stubs.c`.

The callout `EcuM_AL_Reset` may be implemented as:

```
FUNC(void, ECUM_CODE) EcuM_AL_Reset(EcuM_ResetType Reset)
{
    #if (STD_ON == MCU_PERFORM_RESET_API)
        Mcu_PerformReset();
    #else
        Mcu_SetMode(McuConf_McuModeSettingConf_RESET);
    #endif
}
```

Here, in case the Perform Reset API of module MCU is not available, the reset can be achieved via calling `Mcu_SetMode` with the symbolic name value representing the reset mode of the hardware MCU. For powering off the virtual MCU, callout `EcuM_AL_SwitchOff` may be implemented as following:

```
FUNC(void, ECUM_CODE) EcuM_AL_SwitchOff(void)
{
    Mcu_SetMode(McuConf_McuModeSettingConf_POWER_OFF);
}
```

Finally, the callout `EcuM_McuSetMode` has to delegate the MCU mode to the MCU via API `Mcu_SetMode`:

```
FUNC(void, ECUM_CODE) EcuM_McuSetMode(Mcu_ModeType McuMode)
{
    Mcu_SetMode(McuMode);
}
```

**Note**

In case of a VTT Only project, the hardware MCU module is pre-configured with four modes: NORMAL, RESET, SLEEP and POWER\_OFF. When the project is migrated later on to a Dual Target setup, you **must** revise the configuration to enable the correct simulation of the mode handling of the chosen hardware MCU.

### 4.3 User-Defined System Variables

System variables are the interface to CANoe (or 3<sup>rd</sup> party test tool via Test API) to exchange data and may be used by the customer to implement own complex drivers. As an example, system variables are used in the VTT IO modules to communicate with CANoe.

The vVIRTUALtarget module VTTCntrl offers the possibility to specify system variables by means of configuration elements. VTTCntrl generates API functions for reading and writing these variables. System variables are accessible by symbolic name values. To use the API the header file `VttCntrl_SysVar.h` must be included.

For more details on configuring and using system variables via VTTCntrl please see [3].

## 5 Configuration of Representable Features

The configuration of BSW modules follows the general approach of a MICROSAR stack. Technical References and online help of BSW modules will provide necessary details for general settings. This chapter provides special hints and details on representable features (see 2.2) that require special attention if being used with vVIRTUALtarget.

### 5.1 Hardware OS Counters (Gen6)

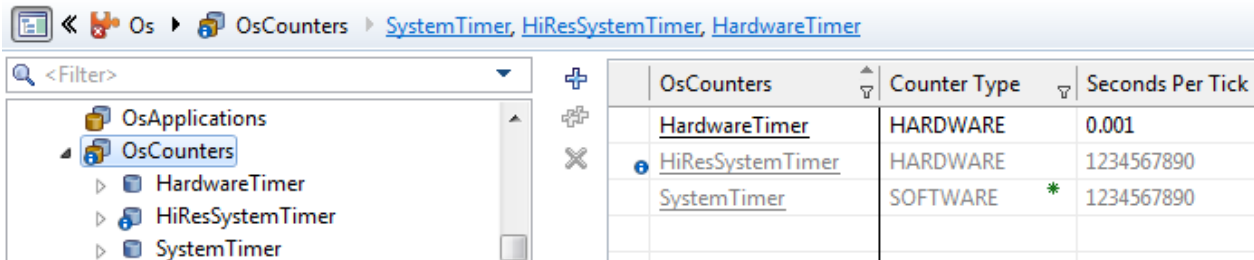


#### Caution

This workaround is only necessary for the version Gen6 of VTT OS. To check whether your VTT OS is version Gen6 please follow the steps in chapter 2. In version Gen7 this workaround is not necessary.

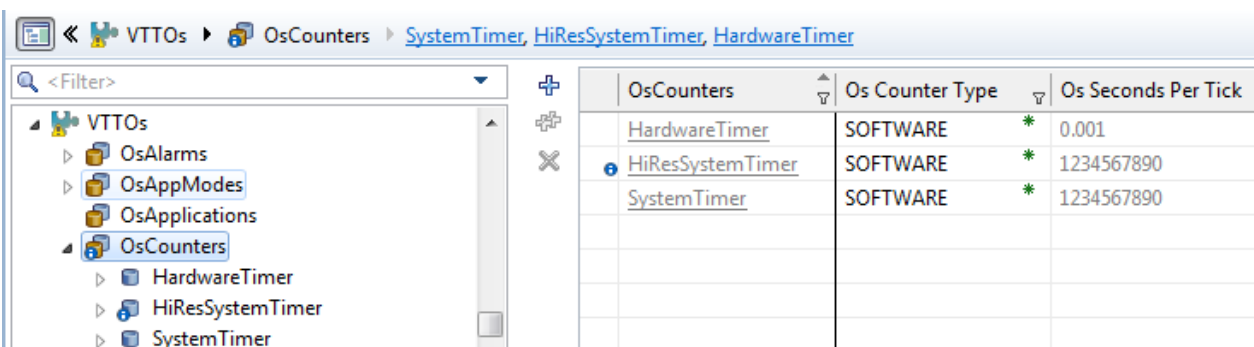
VTT Os supports a single hardware counter only (Container **/VTTos/OsCounter**). This counter must be named "SystemTimer". All settings made for this counter are ignored. This counter is configured in container **/VTTos/OsOS/OsOSSystemTimer/Standard**.

If the hardware OS configuration defined additional counters, these are synchronized to VTT Os. In contrast to the hardware OS, these additional counters are always configured as software counters, regardless of their configuration in the hardware OS (see Figure 5-1 and Figure 5-2).



OsCounters	Counter Type	Seconds Per Tick
HardwareTimer	HARDWARE	0.001
HiResSystemTimer	HARDWARE	1234567890
SystemTimer	SOFTWARE *	1234567890

Figure 5-1 Hardware timer configured for a hardware OS



OsCounters	Os Counter Type	Os Seconds Per Tick
HardwareTimer	SOFTWARE *	0.001
HiResSystemTimer	SOFTWARE *	1234567890
SystemTimer	SOFTWARE *	1234567890

Figure 5-2 VTT OS realizes additional timers as software timers

Different to hardware timers, software timers are not incremented automatically by the OS. When such counters are used in the ECU project, they must be incremented in VTT-specific code.

Example:

1. Create an additional GPT channel in VTT Gpt for each hardware timer:

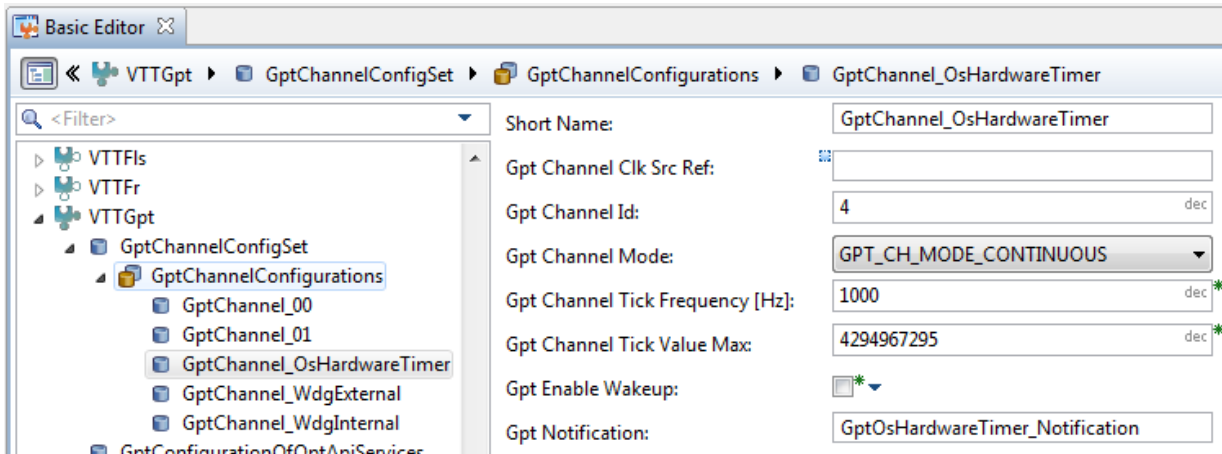


Figure 5-3 GTP channel configuration



**Note**

The Interrupt for this GPT channel should be set to Category 2.

2. Implement the GPT notification function:

```
void GptOsHardwareTimer_Notification(void)
{
    IncrementCounter(HardwareTimer);
}
```

2. Add the corresponding function declaration to a header file.
3. Add the name of the header file to the list in /MICROSAR/VTT/VTTGpt/GptDriverConfiguration/GptCfgIncludeList
4. Extend your Init\_Task to start the GPT channel:

```
TASK(Init_Task)
{
    EcuM_StartupTwo();

#ifdef _MICROSOFT_C_VTT_
```

```
Gpt_EnableNotification(GptConf_GptChannelConfiguration_GptChannel_OsHardwareTimer);  
  
Gpt_StartTimer(GptConf_GptChannelConfiguration_GptChannel_OsHardwareTimer, 1000);  
  
#endif  
  
    TerminateTask();  
};
```

The GPT channel now increments the HardwareTimer.

## 5.2 High Resolution Timers (Gen6)



### Caution

This workaround is only necessary for the version Gen6 of VTT OS. To check whether your VTT OS is version Gen6 please follow the steps in chapter 2. In version Gen7 this workaround is not necessary.

The VTT OS currently does not support high resolution timers. However, in most cases, the use of high resolution timers can be emulated in VTT.

The idea of the emulation is to have the regular OS timer tick at a rate that is greater or equal to the resolution of the high resolution timer.



### Note

Note that this will result in a large number of OS interrupts. Depending on the desired resolution, this may cause a significant performance hit. It is also possible to have the VTT OS timer tick at a rate that is lower than the resolution of the high resolution timer but greater or equal to the interval that the high resolution timer actually fires. In this case, the high resolution timer will fire with a reduced accuracy, but the performance of VTT will significantly improve. This will result in different time bases for the AUTOSAR stack in a Hardware-execution and an execution under VTT. In this case, the **VTT OS** must be configured to emulate the correct time base.

To perform the emulation, two tasks must be performed:

- > Adjust the Tick Time in VTTOs/OsOS/OsOSSystemTimer/Standard
- > Extend the **VTT OS interface** to scale the time base visible to the outside world



**Note**

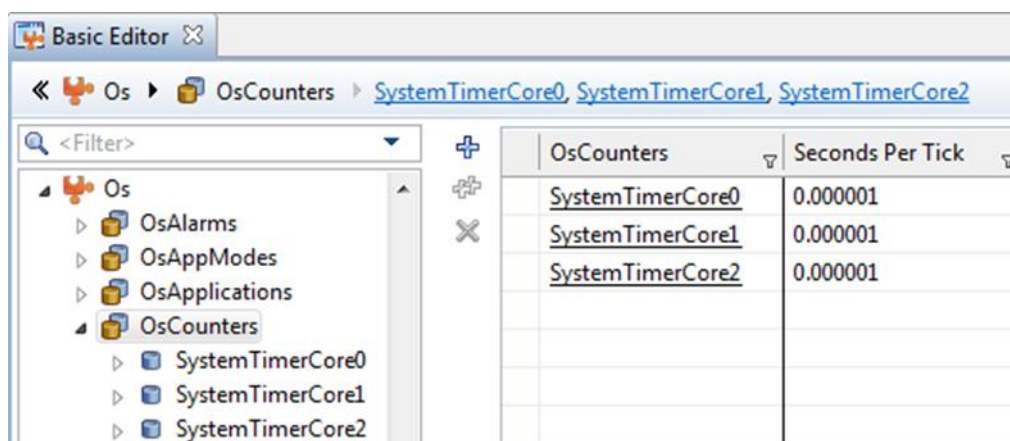
The Mapping of schedule tables is not performed automatically. All values for schedule tables in the VTT OS must be modified manually, analogous to the reconfiguration presented below.

First, find the time base used by the RTE (Value **Seconds Per Tick** of **OsCounters**). In this example, the desired time base is **0.000001**.



**Note**

This emulation is only possible if all high resolution timers use the same time base and use a **Ticks Per Base** value of **1**. The value in an **OsCounter** is given in seconds.



OsCounters	Seconds Per Tick
SystemTimerCore0	0.000001
SystemTimerCore1	0.000001
SystemTimerCore2	0.000001

Figure 5-4 OsCounters – Adjust the Tick Time

Adjust the **VTT OS System Timer** via **/VTTos/OsOS/OsOSSystemTimer/Standard/OsOSTickTime**. In this example, the required Tick Time is **1**.

**Note**

The value of the **Os OS Tick Time** should be the greatest common divisor (or a divisor of that) of all requested OS Alarms. The value of the **Os OS Tick Time** is given in **Microseconds**.

**Note**

In this example, the **VTT OS** will provide the same time base as the hardware **OS**.

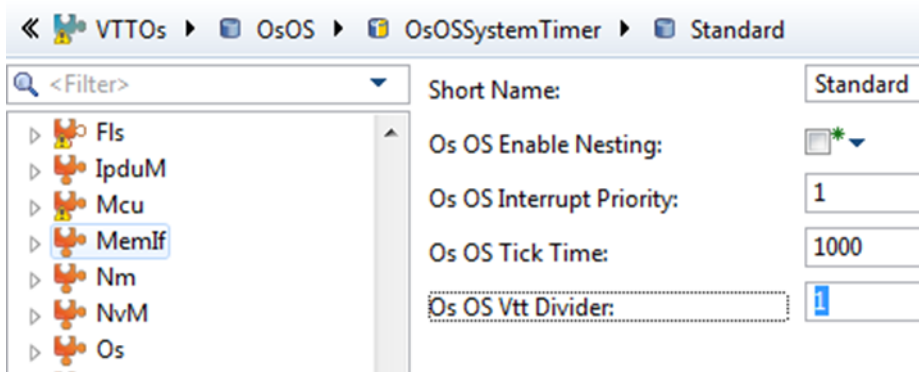


Figure 5-5 The OS Tick Time should be the greatest common divisor

If the **VTT OS** does not use the same tick time as the hardware OS, it provides timer ticks at a rate different to that expected by the RTE.

To compensate for this, the SetRelAlarm function must be patched to use a conversion factor. You must manually compute and set an appropriate conversion factor as parameter **VTTOS/OsOS/OsOSSystemTimer/Standard/OsOSVttDivider**, computed for your specific project settings.

Compute this factor as  $(\text{VTT OS Tick Time}) / (\text{Hardware OS Time Base} * 1000000)$ .

### 5.3 Background Tasks

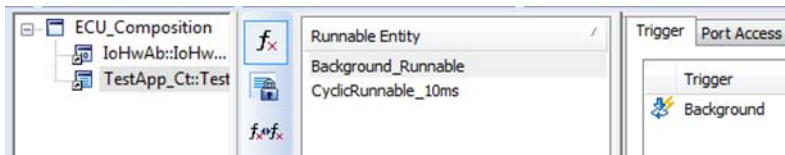
Some functions may be configured as background tasks, e.g., the FEE main function. Background tasks that do not advance simulation time are detected by the CANoe and the simulation will be stopped.

To resolve this problem the function **CANoeAPI\_ConsumeTicks()** must be called at least once within each cycle of the background task in order to consume and therefore advance simulation time. If the background task is implemented by the application, CANoeAPI\_ConsumeTicks() can be added directly.

If the background task is implemented by the RTE the CANoeAPI\_ConsumeTicks() call can be injected e.g. using the following pattern. The pattern is useful as it will not cause the RTE to overwrite the manual changes each time code is generated.

3. Go to DaVinci Developer.

5. Add a new runnable called **Background\_Runnable** to one of your software components.



- 6.
7. Figure 5-6 DaVinci Developer - Add background runnable with background trigger
8. Switch to DaVinci Configurator Pro and synchronize the configuration.
9. Map the **Background\_Runnable** and all configured background functions (e.g., Fee\_MainFunction) to your background task.
10. Implement the runnable. The runnable will just call **CANoeAPI\_ConsumeTicks()**.

```
FUNC(void, RTE_APPL_CODE) Background_Runnable(void)
{
#ifdef _MICROSOFT_C_VTT_
    CANoeAPI_ConsumeTicks(1);
#endif
}
```

## 5.4 Handling Different MCU Clock Offsets

AUTOSAR does not specify the handling of the `ClockSettingsConfig` parameter. Some MCAL modules use the value of this parameter with an offset of one while other MCAL modules start with zero.

VTTMcu uses zero as offset. If the hardware MCU uses one as offset, you have to make sure that `Mcu_InitClock` is called with a value with correct offset. To adapt the code accordingly a VTT macro can be used, see 4.1.

For example, in `EcuM_Callout_Stubs.c` the (different) value(s) of the argument of `Mcu_InitClock` must be switched using `#ifdef`.

## 5.5 NvM Block Length Strict Check

The NvM Block Length Strict Check verifies whether each NvMBlock (Container /MICROSAR/NvM/NvMBlockDescriptor) has the exact same size as the according variable generated by the compiler. As the size of the variable heavily depends on the compiler and the compiler options used, no single value will work for both the embedded compiler and Microsoft Visual Studio compiler. Therefore, Block Length Strict Check must not be enabled in Dual Target projects and the parameter

`/MICROSAR/NvM/NvMBlockDescriptor/NvMNvBlockLength`

must be set to the maximum of the values required by either compiler. If NvM Block Length Strict Check is enabled in DaVinci Configurator Pro, the resulting code might not compile.



## 6 Project Migration

Migration of a project from one VTT use case to another becomes imminent if the project is started with a VTT Only SIP and a SIP including the hardware MCAL is provided at a later point in time. DaVinci Configurator Pro allows the migration of a project that has been created with a VTT Only SIP to a hardware-based SIP.

The hardware SIP may or may not contain VTT as an option. If the hardware SIP includes the VTT option, the project can be migrated to a VTT Dual Target project.

In addition, the migration strategy differs if the hardware MCAL modules are configured using DaVinci Configurator or using 3<sup>rd</sup> party MCAL configuration tool different to DaVinci Configurator ("Two Tools Configuration Approach") are used. If another configuration tool is used to set up the MCAL configuration, the configuration must be exported as ARXML and imported into DaVinci Configurator Pro.

The project is migrated as follows:

1. Open the DaVinci Configurator Pro of the new SIP (with/without VTT option) and open the dpa file.
2. If you get a warning that the SIP is not compatible with the project, select the third option "Update the project by opening the project within the SIP of this DaVinci Configurator instance".
3. When the Alternative Module Definition window is shown, select the appropriate module definition for each module. See Figure 6-1 for an example.

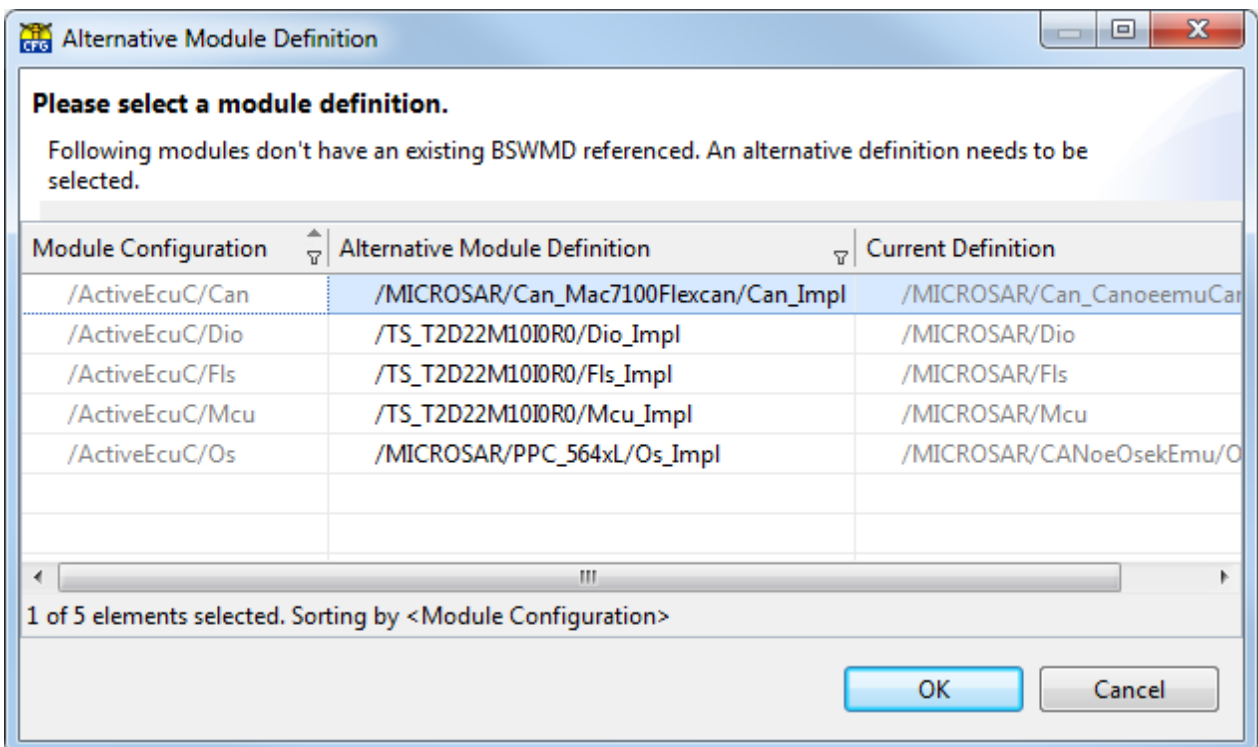


Figure 6-1 Alternative module definition dialog for switching module definitions

4. Open the project settings and click the button shown in Figure 6-2 to edit the project settings.

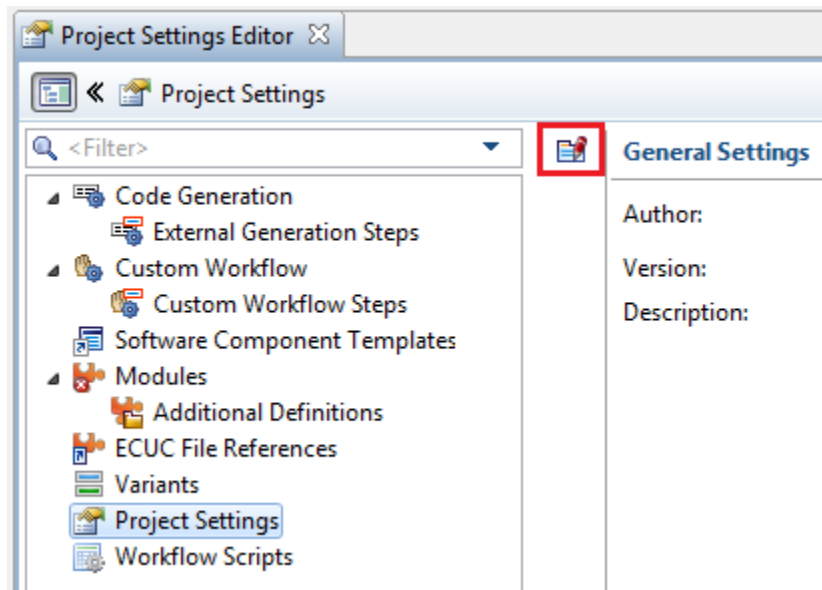


Figure 6-2 Button for editing the project settings

5. Edit derivative, compiler and pin layout in section Target according to your project requirements. If VTT is part of your delivery, activate option Virtual Target. Click again on the button to save project settings and to reload the project. The Alternative Module Definition dialog might be shown again to select module definitions for the remaining MCAL modules that were not shown in step 3 (this is due to the fact that some modules are only available if the correct target is configured).
6. **[Without VTT option]** Go to the project settings and remove all VTT modules.
7. Resolve all errors that are shown in the validation view. There are several common errors that are related to the migration:
  - a. If parameters were defined for the virtual MCAL modules but do not exist in the hardware MCAL modules, then delete them by executing the auto resolve action.
  - b. Set the correct compiler in the OS configuration.
  - c. Resolve further errors.



**Note for the Two Tools Configuration Approach:**

Instead of reusing the existing configuration of the dummy MCAL modules, delete them in the project settings and import the MCAL configuration that has been created before by the second configuration tool.

When migrating a VTT Only project to a Dual Target project, the directory with the include files should be duplicated, e.g. to a folder named includeVTT and includeHW. The file `Compiler_Cfg.h` and `MemMap.h` for the hardware should be used from the Common folder in BSW folder.

## 7 Post-Build Selectable

The MICROSAR Identity Manager (Post-Build Selectable) is also supported by vVIRTUALtarget. It allows the inclusion of several BSW configurations within a single ECU. At start-up the application determines the BSW variant to be activated and initializes the BSW accordingly.



### Reference

More information on the MICROSAR Identity Manager can be found in the Technical Reference [6].

For VTT there are some special aspects that must be considered.

### 7.1 Handling DLL names

When using variance in the project the Post-Build Selectable support of the module VTTCntrl must be activated in the projects settings. The configuration parameter /MICROSAR/VTT/VTTCntrl/VttGeneral/VttDllName must be variant and assigned a unique DLL name to each variant. A validator automatically sets this parameter to a name <EcuInstance>\_<Variant>.dll where <EcuInstance> is the name of the ECU instance in the system description of the project and <Variant> is the currently active variant.

Variants become also visible in vVIRTUALtarget. The generated solution includes a post-script that copies the (single) DLL to the DLLs with the different names as specified by VTTCntrl to the SUT directory. This enables the user to load several variants of the same ECU in a single CANoe configuration.

Note that the DLL names indicate the variant to be selected. This is important when loading the DLL in CANoe.

### 7.2 Variant Selection in EcuM

The code for the variant selection must be implemented in EcuM Callouts. The variant selection is based on the DLL file name of the virtual ECU. The following code shows how to query the DLL file name and return the corresponding configuration pointer of the active variant. In this example two variants exist: Left and Right. The header file VttCntrl.h must be included in EcuM\_Callout\_Stubs.c to use the generated macros of the configured file names.

```
FUNC(EcuM_GlobalConfigRefType, ECUM_CODE) EcuM_DeterminePbConfiguration(void)
{
/*****
*****
* DO NOT CHANGE THIS COMMENT!          <USERBLOCK EcuM_DeterminePbConfiguration>          DO NOT
CHANGE THIS COMMENT!

*****
*****/
/* Add implementation of EcuM_DeterminePbConfiguration() */
if (CANoeAPI_TestFileNameOfDLL(VTTCNTRL_DLL_NAME_Left))
{
    return &(EcuM_GlobalConfigRoot.Left);
}
```

```
if (CANoeAPI_TestFileNameOfDLL(VTTCNTRL_DLL_NAME_Right))
{
    return &(EcuM_GlobalConfigRoot.Right);
}
return NULL_PTR;
/*****
*****
* DO NOT CHANGE THIS COMMENT!          </USERBLOCK>          DO NOT
CHANGE THIS COMMENT!

*****
*****/
} /* End of EcuM_DeterminePbConfiguration() */
```

### 7.3 Debugging of Variants

Virtual ECUs can be debugged with Visual Studio. If two (or more) variants of the same ECU are executed in the same CANoe instance (i.e. in the same RuntimeKernel.exe) and a breakpoint is set in the ECU code in Visual Studio, both variants may hit that breakpoint. This hinders the debugging of a single variant of the ECU because all breakpoints are by all the variants. To debug just a single variant of the ECU there are several approaches possible:

- > To break on a function `func` in DLL `ECU_Left.dll` you can manually add a breakpoint (break at function) by using the function name `{, , ECU_Left.dll}func`.
- > When a breakpoint is set in the code the breakpoints view will show several sub-breakpoints for this breakpoint, one for each variant. You can manually deselect sub-breakpoints.
- > A breakpoint can be restricted to certain processes and threads (see the filter property of a breakpoint). To filter for certain processes does not help much since all virtual ECUs are executed in the same RuntimeKernel.exe. However, each task of each variant is an own thread with a unique name. This name could be used in the filter.

## 8 Glossary and Abbreviations

### 8.1 Glossary

Term	Description
CANoe	Tool for simulation and testing of networks and electronic control units
DaVinci Configurator Pro	Configuration and generation tool for MICROSAR components

Table 8-1 Glossary

### 8.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basic Software
MCAL	Microcontroller Abstraction Layer
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
OS	Operation System
SIP	Software Integration Package
VTT	vVIRTUALtarget

Table 8-2 Abbreviations

## 9 Contact

Visit our website for more information on

- > News
- > Products
- > Demo software
- > Support
- > Training data
- > Addresses

**[www.vector.com](http://www.vector.com)**