

# **vVIRTUALtarget**

## Technical Reference

Version 1.0.4

Authors	Sebastian Bauer, 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

### Reference Documents

No.	Source	Title	Version
[1]	Vector	Technical Reference of vVIRTUALtarget OS File: MicrosarOS_CANoeEmuVTT.pdf	as delivered
[2]	Vector	Technical References of vVIRTUALtarget MCAL modules Files: TechnicalReference_<Module>_VTT.pdf	as delivered
[3]	Vector	Startup Manual with vVIRTUALtarget File: Startup_<Program>_vVIRTUALtarget.pdf	as delivered
[4]	Vector	Technical Reference of VTTCntrl TechnicalReference_VttCntrl.pdf	as delivered
[5]	Vector	Tool Help of Vector vVIRTUALtarget basic	

**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.

## Contents

<b>1</b>	<b>Introduction .....</b>	<b>6</b>
1.1	Use Cases of vVIRTUALtarget .....	6
1.1.1	VTT Only .....	6
1.1.2	VTT Dual Target .....	6
1.2	Scope of this Document .....	6
<b>2</b>	<b>Functional Description .....</b>	<b>8</b>
2.1	Supported Features.....	8
2.2	Representable Features.....	8
2.3	Unsupported Features.....	9
<b>3</b>	<b>Workflow Details .....</b>	<b>10</b>
3.1	Project Setup.....	10
3.2	Synchronization of Configuration Parameters.....	12
3.3	Using User-Defined Parameters in Configuration .....	13
3.4	Custom Integration in VTT.....	13
3.4.1	Realizing VTT-Specific Code in Dual Target Projects .....	13
3.4.2	Virtual MCU Mode Handling .....	13
3.4.3	User-Defined System Variables.....	15
3.5	Configuration of Representable Features .....	16
3.5.1	Hardware OS Counters .....	16
3.5.2	High Resolution Timers .....	18
3.5.3	Background Tasks.....	20
3.5.4	Handling Different MCU Clock Offsets.....	20
3.6	Project Migration .....	21
<b>4</b>	<b>Glossary and Abbreviations .....</b>	<b>24</b>
4.1	Glossary.....	24
4.2	Abbreviations .....	24
<b>5</b>	<b>Contact.....</b>	<b>25</b>

## Illustrations

Figure 3-1	Selection of target type, case VTT Dual Target .....	10
Figure 3-2	Generation dialog with Virtual Target enabled .....	10
Figure 3-3	Exemplary set of modules in a VTT project.....	12
Figure 3-4	Overwriting the automatic synchronization using "Set user defined" .....	13
Figure 3-5	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.....	14
Figure 3-6	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. ....	14
Figure 3-7	Hardware timer configured for a hardware OS.....	16
Figure 3-8	VTTs realizes additional timers as software timers.....	16
Figure 3-9	GTP channel configuration .....	17
Figure 3-10	OsCounters – Adjust the Tick Time .....	19
Figure 3-11	The OS Tick Time should be the greatest common divisor.....	19
Figure 3-12	DaVinci Developer - Add background runnable with background trigger ...	20
Figure 3-13	Alternative module definition dialog for switching module definitions .....	22
Figure 3-14	Button for editing the project settings .....	22

## Tables

Table 2-1	Representable features with limited support .....	8
Table 4-1	Glossary .....	24
Table 4-2	Abbreviations.....	24

# 1 Introduction

vVIRTUALtarget (VTT) is a platform for simulating and testing MICROSAR-based ECU software in a virtual environment. The hardware is simulated on a PC by replacing OS and MCAL modules for the target hardware by “virtual” modules that are capable to communicate with a virtual test environment.

vVIRTUALtarget projects can be used to test ECU software in an early development stage when no hardware is available or the hardware-specific MICROSAR stack is not prepared yet. 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 both MCAL modules for virtual and real target. Configuration parameters have to be maintained for the hardware MCAL modules only, 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. This enables code generation for vVIRTUALtarget and hardware target from a single configuration project.

## 1.1 Use Cases of vVIRTUALtarget

### 1.1.1 VTT Only

The VTT Only use case provides two sets of OS and MCAL modules: OS and MCAL modules as placeholders for a hardware-specific modules, and virtual OS and MCAL modules that support the runtime environment of vVIRTUALtarget. A VTT Only delivery may be used e.g. in the following scenarios:

- > Provide an early delivery even before the hardware-specific MCAL modules are available. Developers can start configuration of the MICROSAR BSW and test their ECU software on a PC. Later they can migrate to a hardware-specific MICROSAR delivery (without vVIRTUALtarget) or a Dual Target delivery (see 1.1.2) with both vVIRTUALtarget and hardware-specific modules.
- > Prototyping and evaluation of MICROSAR BSW modules on a developer PC.
- > Development of SWCs in a virtual environment.

### 1.1.2 VTT Dual Target

The VTT Dual Target use case uses also provides two sets of OS and 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. When starting the code generation the user can select either the virtual target or the hardware target.

## 1.2 Scope of this Document

This document aims at describing features of vVIRTUALtarget (VTT) and provides guidelines for configuration of vVIRTUALtarget projects.

**Caution**

vVIRTUALtarget does not provide a complete and accurate simulation of your hardware ECU. For instance the runtime behavior of the simulated 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.

## 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
- > **Representable Features**  
Features that are supported API wise but the emulated 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

### 2.1 Supported Features

The supported features of the VTT modules are listed in the Technical References of the modules, see [1] and [2].

### 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	Hardware Counters	Only one single hardware counter is supported. Further (software) counters must be incremented manually.	See 3.5.1
OS	High Resolution Timer	High Resolution Timers can be emulated by manual configuration.	See 3.5.2
OS	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 3.5.3
MCU	Clock Offset	Offset mismatches between hardware MCU and virtual MCU must be handled by adapting code.	See 3.5.4

Table 2-1 Representable features with limited support



## 2.3 Unsupported Features

### > OS Scalability Classes

VTTs only support Scalability Class 1.

### > MultiCore OS

VTT does not support Multi-Core OS configurations.

### > Variant Handling – Identity Manager

VTT modules do not support post-build selectable as used by the MICROSAR Identity Manager.

### > 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.

### > 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.

### > Partial Networking

VTT does not support simulation of partial networking for selective wake-up of an ECU.

## 3 Workflow Details

### 3.1 Project Setup

In the project creation wizard of DaVinci Configurator Pro, the available target types depend on your MICROSAR delivery. Figure 3-1 shows the dialog for the case VTT Dual Target.

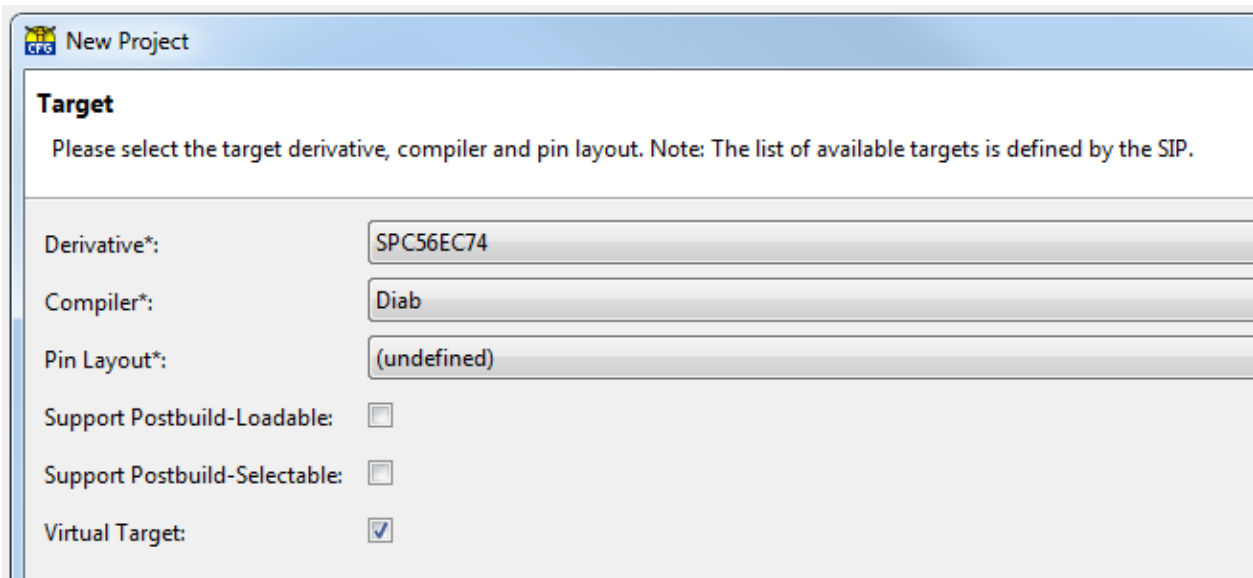


Figure 3-1 Selection of target type, case VTT Dual Target

In case of VTT Only all options are preselected in this dialog (Virtual Target option is enabled) and cannot be edited.

The option Virtual Target activates VTT in your project and adds the generation target “Virtual Target (vVIRTUALtarget)” to the code generation dialog, see Figure 3-2.

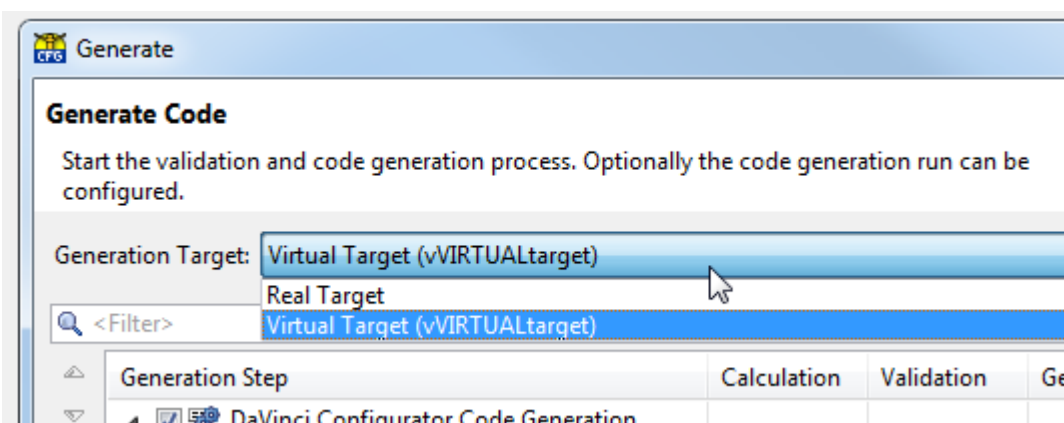


Figure 3-2 Generation dialog with Virtual Target enabled

In case of VTT Only the single option for generation target is “Virtual Target (vVIRTUALtarget)”.

When the project with activated option Virtual Target is created, BSW modules must be added in the Project Settings Editor. At least the modules VTTCntrl and VTTEcuC must be added. Moreover, for each hardware-dependent BSW module <ModuleName> the corresponding VTT module VTT<ModuleName> must be added. This correspondence between hardware module and VTT module is of importance for the synchronization of configurations explained in the next section. Figure 3-3 shows an example of a correct selection of modules.



---

**FAQ**

If your project uses the VTT Only use case there is also the need to enable for each MCAL module both, the dummy MCAL that replaces the hardware MCAL as well as the corresponding VTT module.

---



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

### 3.2 Synchronization of Configuration Parameters

Most parts of the configuration of VTT modules, the blue modules in Figure 3-3, are derived from the configuration of the corresponding dummy (VTT Only) or hardware (VTT Dual Target) modules, the red modules in Figure 3-3. The synchronization happens in the background as soon as configuration parameters are changed in the red 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 (see 3.3).

**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 to VTTEcuC.

### 3.3 Using User-Defined Parameters in Configuration

In practice it can happen that due to a required workaround the configuration of VTT modules deviate from 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 3-4. The parameter can now be changed. The parameter will not be synchronized as long as it is set to user defined.

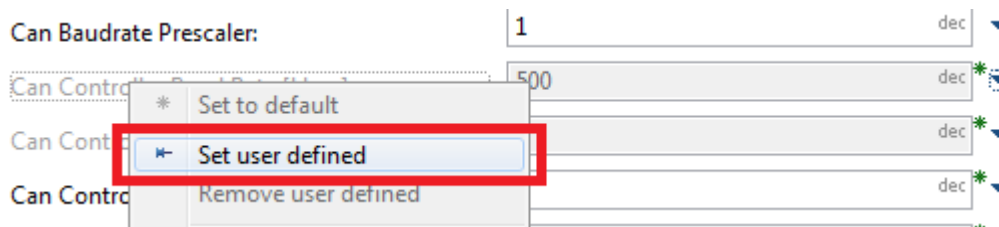


Figure 3-4 Overwriting the automatic synchronization using “Set user defined”

### 3.4 Custom Integration in VTT

#### 3.4.1 Realizing VTT-Specific Code 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 vVIRTUALtarget provides the following `#define`.

```
#define _MICROSOFT_C_VTT_
```

If the define is set the code is compiled for vVIRTUALtarget if it is not available the current compile process is intended for the hardware. Use a simple preprocessor `#ifdef` statement to assess the target.

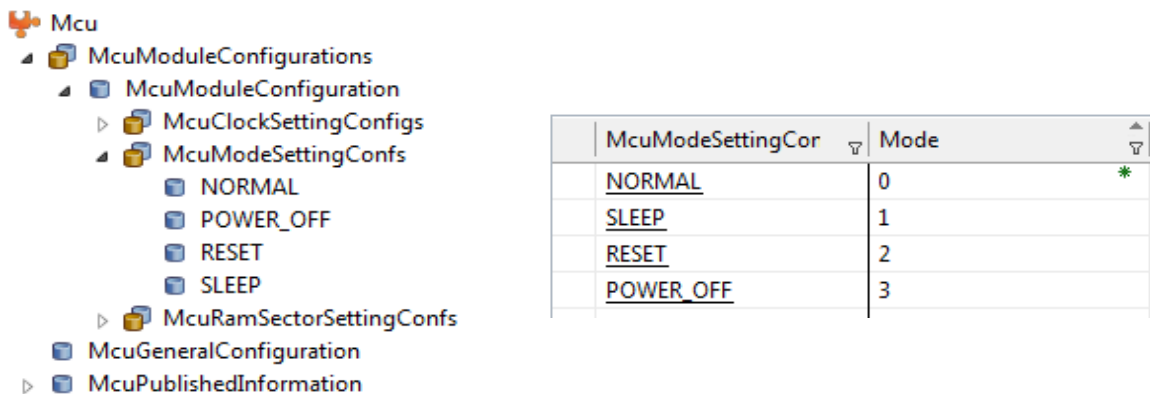
#### 3.4.2 Virtual MCU Mode Handling

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.

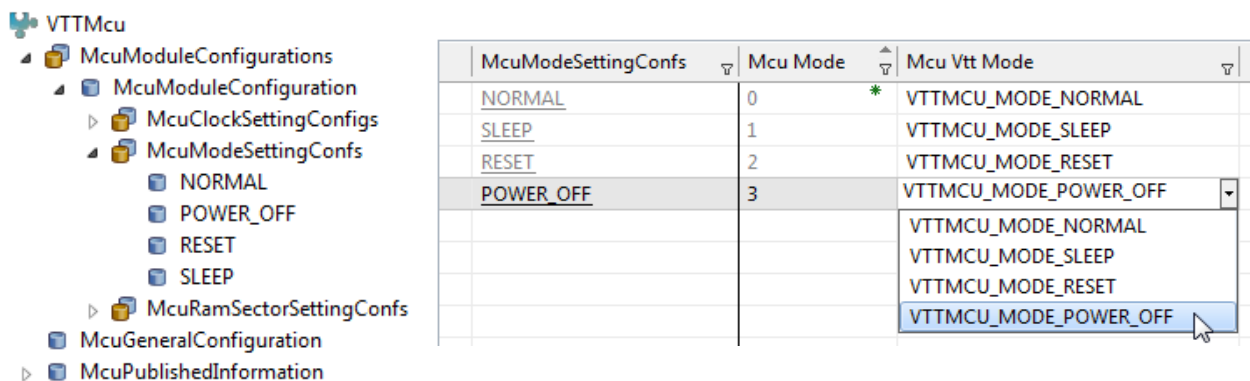
For this purpose, you need first to configure the MCU modes in the hardware MCU module. Assume the modes NORMAL (0), SLEEP (1), RESET (2) and POWER\_OFF (3) are configured as shown in Figure 3-5.



McuModeSettingCor	Mode
NORMAL	0
SLEEP	1
RESET	2
POWER_OFF	3

Figure 3-5 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 3-6. By default, each hardware MCU mode is mapped to virtual MCU mode VTTMCU\_MODE\_NORMAL.



McuModeSettingConfs	Mcu Mode	Mcu Vtt Mode
NORMAL	0	VTTMCU_MODE_NORMAL
SLEEP	1	VTTMCU_MODE_SLEEP
RESET	2	VTTMCU_MODE_RESET
POWER_OFF	3	VTTMCU_MODE_POWER_OFF

Figure 3-6 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.

### 3.4.3 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 `VttCtrl_SysVar.h` must be included.

For more details on configuring and using system variables via VTTCtrl please refer to [4].

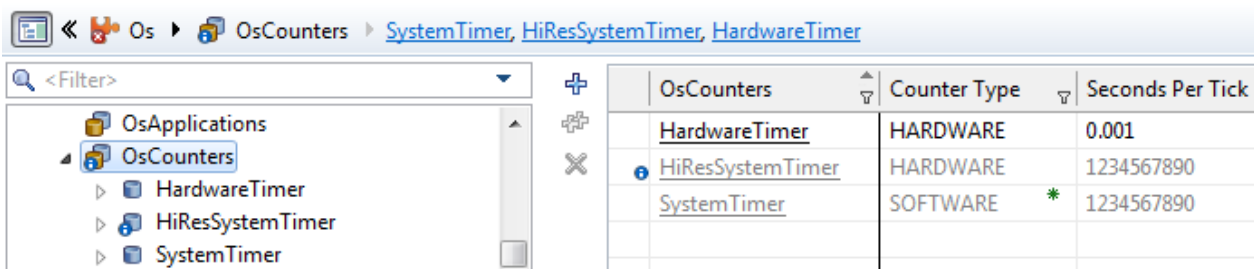
### 3.5 Configuration of Representable Features

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.

#### 3.5.1 Hardware OS Counters

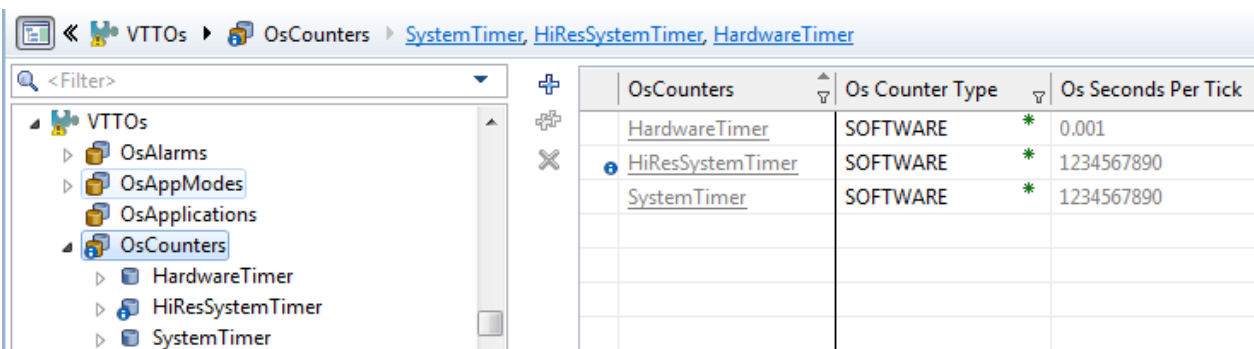
VTTos supports a single hardware counter only (Container **/VTTos/OsCounter**). This counter must be named "SystemTimer". All settings made for this counter are ignored. This counter is configured in container **/VTTos/OsOS/OsOSSystemTimer/Standard**.

If the hardware OS configuration defined additional counters, these are synchronized to VTTos. 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 3-7 and Figure 3-8).



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

Figure 3-7 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 3-8 VTTos 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 VTTGpt for each hardware timer:



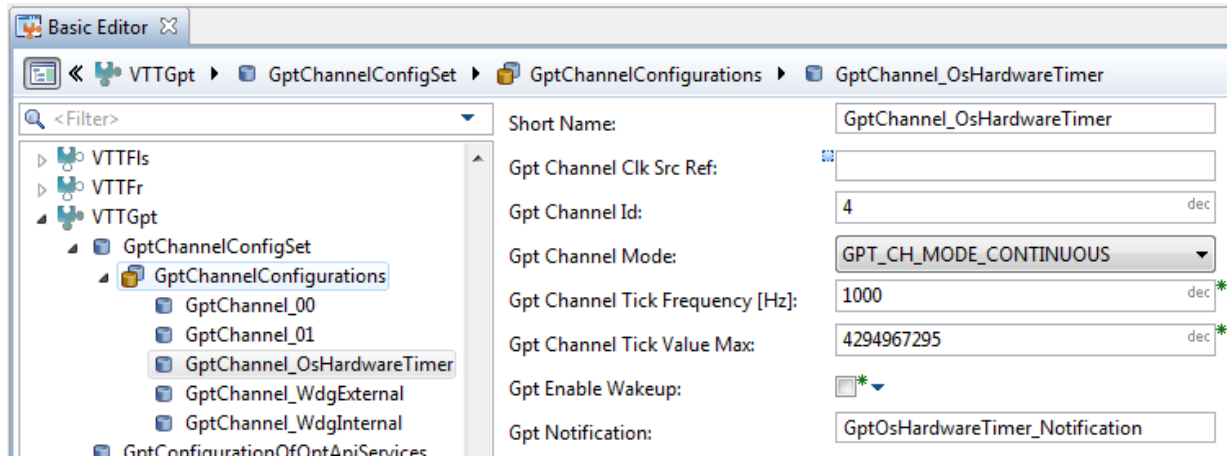


Figure 3-9 GTP channel configuration

**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);
}
```

## 3. Add the corresponding function declaration to a header file.

4. Add the name of the header file to the list in **/MICROSAR/VTT/VTTGpt/GptDriverConfiguration/GptCfgIncludeList**

## 5. 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.

### 3.5.2 High Resolution Timers

The VTTOs 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 VTTOs 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 **VTTOs** 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 **VTTOs 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 VTTOs 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.

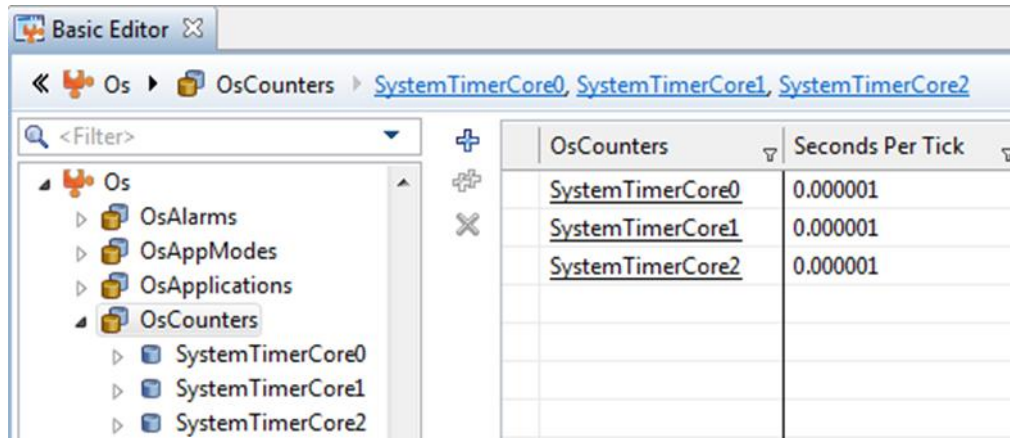


Figure 3-10 OsCounters – Adjust the Tick Time

Adjust the **VTTOS 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 **VTTOS** will provide the same time base as the hardware **OS**.

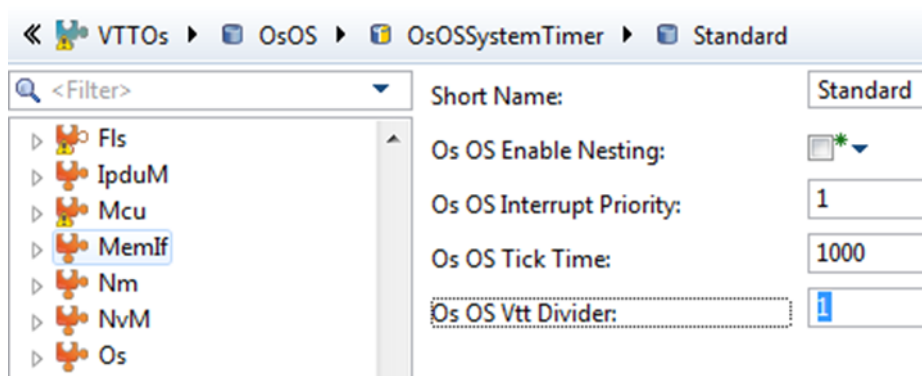


Figure 3-11 The OS Tick Time should be the greatest common divisor

If the **VTTOS** 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 (VTTOS Tick Time) / (Hardware OS Time Base \* 1000000).

### 3.5.3 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.

1. Go to DaVinci Developer.
2. Add a new runnable called **Background\_Runnable** to one of your software components.

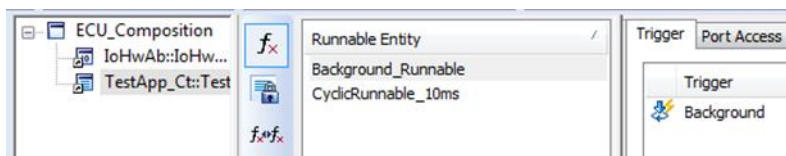


Figure 3-12 DaVinci Developer - Add background runnable with background trigger

3. Switch to DaVinci Configurator Pro and synchronize the configuration.
4. Map the **Background\_Runnable** and all configured background functions (e.g., Fee\_MainFunction) to your background task.
5. 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
}
```

### 3.5.4 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 3.4.1.

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

### 3.6 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 3-13 for an example.

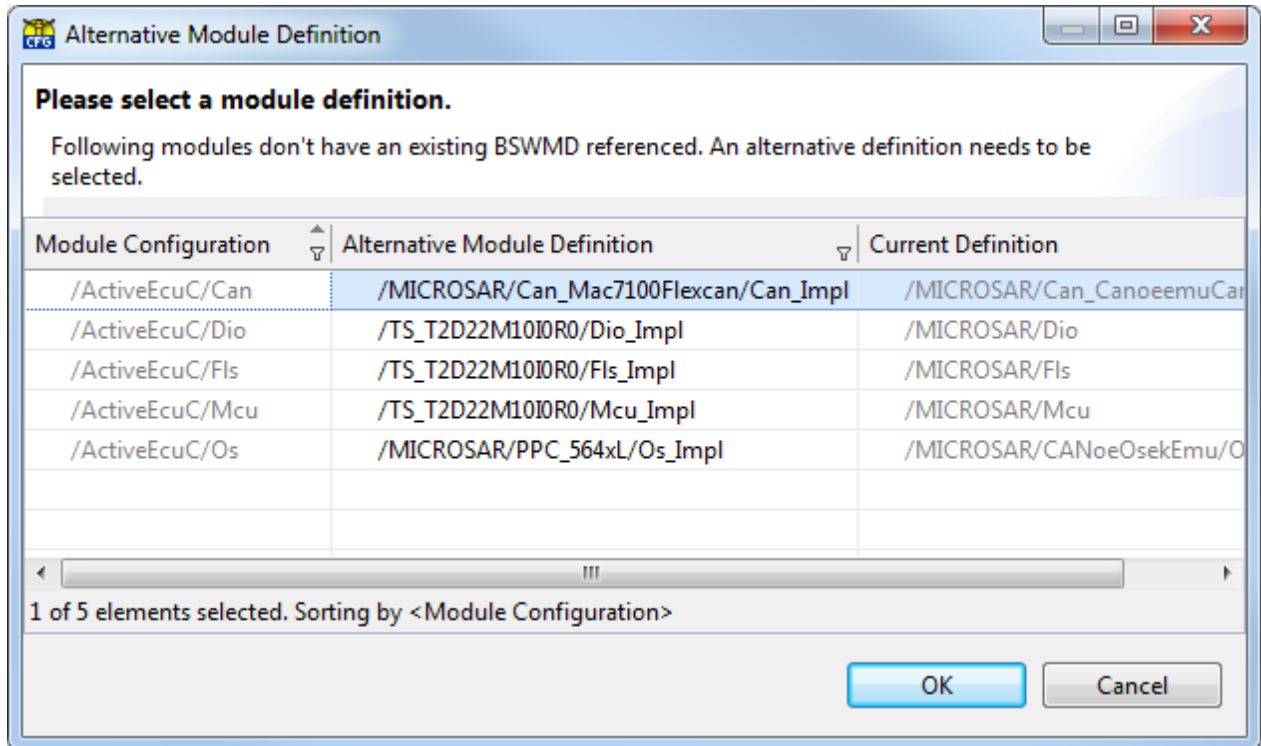


Figure 3-13 Alternative module definition dialog for switching module definitions

- Open the project settings and click the button shown in Figure 3-14 to edit the project settings.

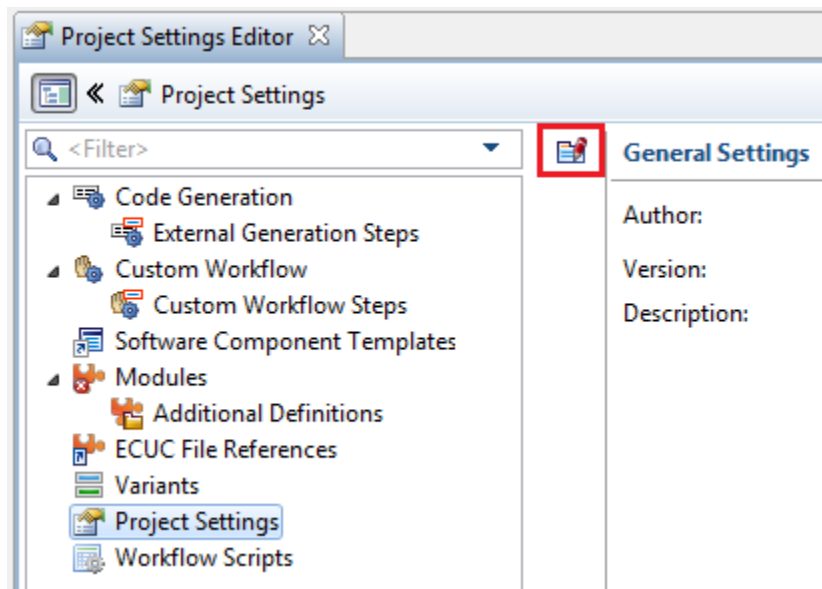


Figure 3-14 Button for editing the project settings

- 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.

## 4 Glossary and Abbreviations

### 4.1 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 4-1 Glossary

### 4.2 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 4-2 Abbreviations



## 5 Contact

Visit our website for more information on

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

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