Share FEE Blocks Between Application and Bootloader

Version 1.01.01
2017-01-11
Application Note AN-ISC-8-1173

Author(s) Vogel, Nils, Andreas Wenckebach
Restrictions Customer confidential - Vector decides
Abstract This document shows how to share nonvolatile data between application and bootloader using the Vector FEE and DaVinci Configurator Pro for AUTOSAR 4 projects.

Table of Contents

1.0 Overview

This application note addresses the case that nonvolatile data should be shared between application and bootloader and how this can be achieved by using the AR memory stack within the bootloader.

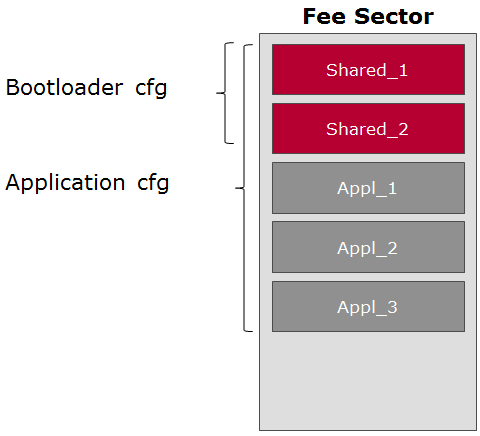

Figure 4-1 Shared FEE-Blocks between application and bootloader

Chapter 2 explains how the FEE works in the given use case and which are the necessary preconditions on the FEE layout.

Chapter 3 describes the necessary actions in the DaVinci Configurator Pro 5 step by step.

2.0 Preconditions on the FEE Layout

In general only a subset of the nonvolatile blocks needs to be shared between the application and the bootloader. This leads to some preconditions on the FEE layout which will be described in the following sub chapter.

Please note it is assumed that all the bootloader blocks are known by the application.

2.1 MSR Module Precondition

At minimum the following AR Modules have to be used in bootloader:

- AR MemIF
- AR FEE
- AR FLS
- Stub for AR NVM or AR NVM (compare 2.2.1, 3.4)

In case the flash bootloader does not include ASR components the following is needed additionally

- Stubs for AR DET and SCHM

This is necessary due to the fact that the FEE uses a dynamic data layout. Therefore it is not possible to address a memory block directly as it is in case of using EEPROM.

2.2 Config Limitations

This chapter describes the bootloader configuration limitations.

2.2.1 NvM Configuration in the Bootloader

Using the AR NvM module is optional and depends on the project specific requirements. If NvM features are necessary for the block to be shared (e.g. RAM Block CRC or NvBlockType redundant block) the NvM has to be integrated as well. Please note that this approach leads to higher resource consumptions of about 30-40 kByte ROM in the bootloader.

Real ASR Nvm module usage is not detailed within this document.

In any case, you have to configure `NvMDataSelectionBits` parameter in order to allow the FEE calculates valid block numbers. Currently it is therefore mandatory create an NvM stub in case no real NVM is present.

2.2.2 Block Limitation

The integrator has to assure that the block ID of the given FEE block is the same in both configurations APPL and bootloader. It is therefore recommended that NvM/FEE-Blocks to be shared are located at the beginning of the FEE Layout to be robust against later layout changes.

You need to configure `Fee Block ID fixed` so that it does not get recalculated after layout changes as depicted in the following figures.

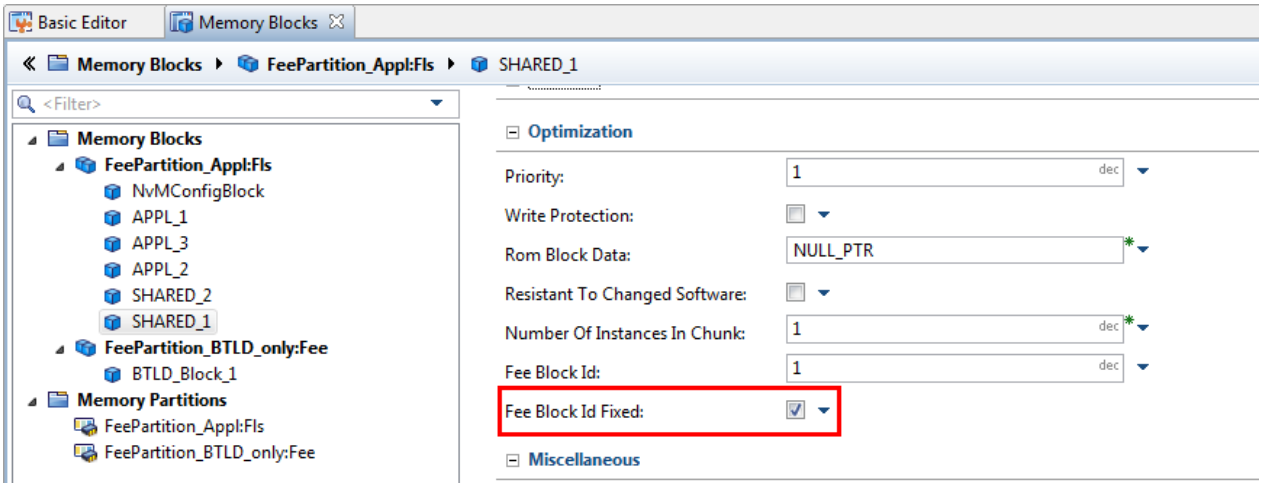

Figure 4-2 Parameter FeeBlockIdFixed in DaVinci Configurator Pro Comfort Editor Memory Blocks

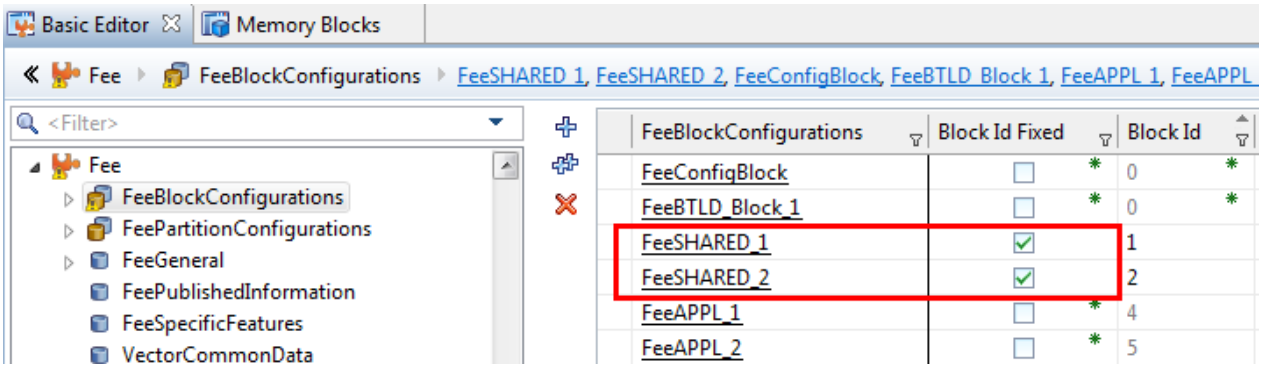

Figure 4-3 Parameter FeeBlockIdFixed in the DaVinci Configurator Pro Basic Editor

2.3 FEE Feature FeeFblConfig

The Vector FEE provides a feature called `FeeFblConfig` which sets the FEE into a special bootloader mode.

In this mode the FEE is able to detect and handle blocks which are not part of the bootloader configuration but which need to be handled in case of a sector switch.

This chapter describes quickly the problem non Vector solutions may have and how the Vector solution addresses this problem.

2.3.1 Potential Problem with Regular AR Fee

The bootloader knows only its own blocks. But what happens in case of a sector switch in the bootloader? In case a regular FEE is used only the blocks known in the current configuration will be handled. This would lead to a loss of all data stored within the application blocks. To overcome this problem you would have to use an exact copy of your application configuration within your bootloader. But this approach would lead to the fact that you have to update the bootloader every time you change the Nv-layout of some of the application blocks which is not feasible.

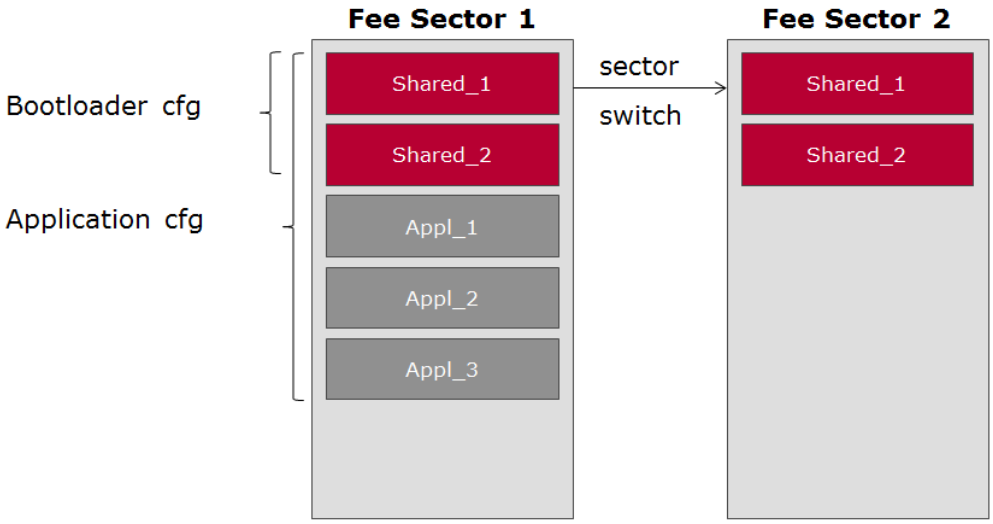

Figure 4-4 FEE Sector Switch with Regular AR FEE

2.3.2 Vector FEE Bootloader Configuration

Therefore the Vector FEE can be switched in a special bootloader mode. This enables the FEE to scan the flash image and detect valid application blocks even if they are not part of the bootloader configuration. This way Nv-blocks can easily be shared between bootloader and application without the need to keep all the application blocks within your bootloader configuration.

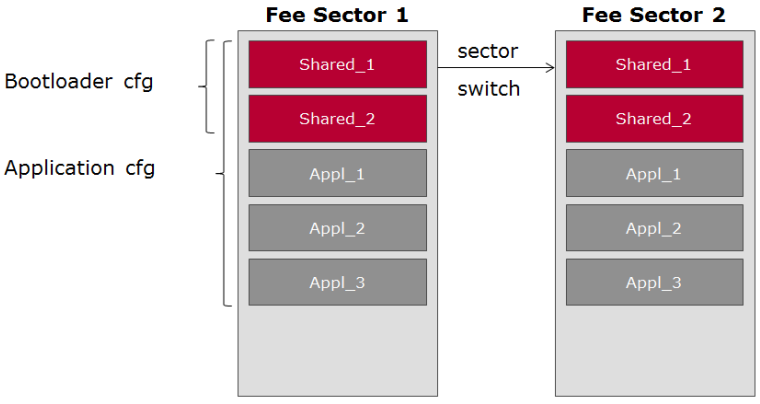

Figure 4-5 FEE Sector Switch with Vector FEE

Please note that the background sector switch (BSS) is disabled in this mode. The FEE works according to a single sector principle.

3.0 FEE Configuration with the DaVinci Configurator Pro

This chapter shows the necessary actions to be done in the DaVinci Configurator Pro in order to share Nv-blocks between application and bootloader. The described use case just uses FLS and FEE. If you plan to use the NvM in the bootloader as well you can skip step 4

Note:	Details on NvM and FEE configuration are not covered by this document

For further information on how to configure NvM and FEE in general can be found in the technical reference of these modules.

3.1 Step 1 Configure the NvM/FEE on Application Side.

Configure your NV layout as needed for your project and start with the blocks to be shared. Make sure that the block IDs are fixed as described in 2.2.2. Verify that the shared blocks do have the smallest block IDs (beside the FeeConfigBlock) in your FEE partition.

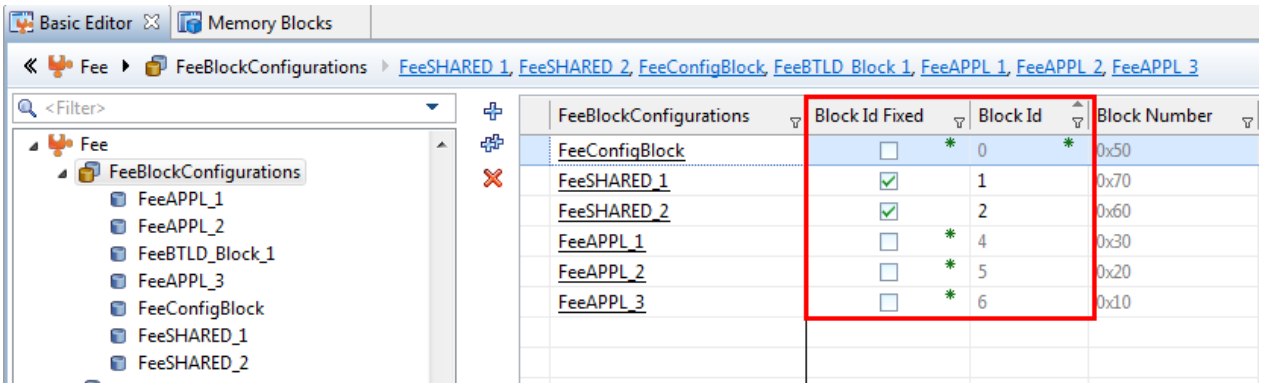

Figure 4-6 Step1 Configure NvM/FEE

3.2 Step 2 Transfer the MEM Stack Configuration to the Bootloader

Now you can export the MEM stack configuration from the application and import it into the bootloader.

This gives you an easy start for the mem stack configuration of the bootloader.

To export the configurations use the export wizard of the DaVinci Configurator Pro.

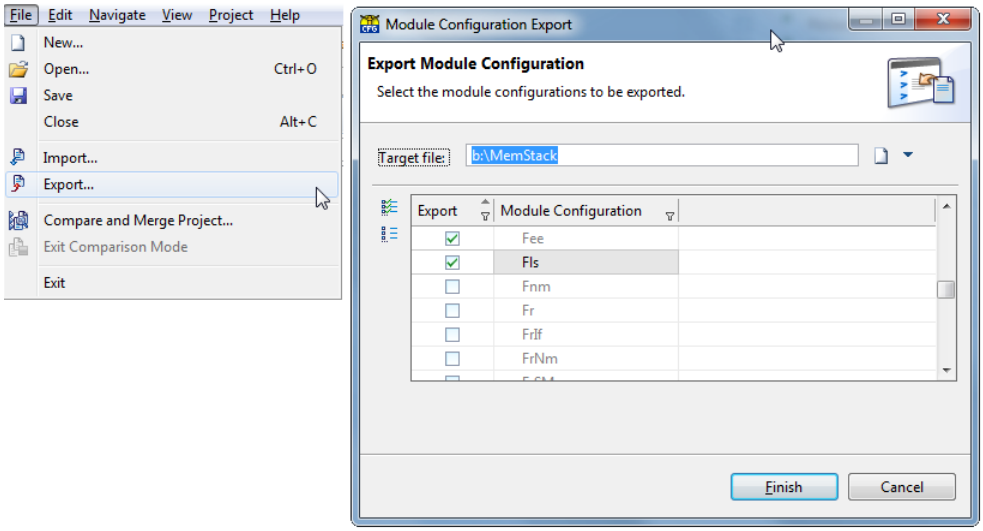

Figure 4-7 Export Configurations with DaVinci Configurator Pro

Once this is done, you can import the configuration into your bootloader project.

Note:	If your Fbl does not use DaVinci Configurator, but another tool (e.g. GENy), you have to create an empty configuration withinthe DaVinci Configurator first and import the configuration here in order to generate your Fee configuration (chapter 3.5 describes some more aspects for this situation).

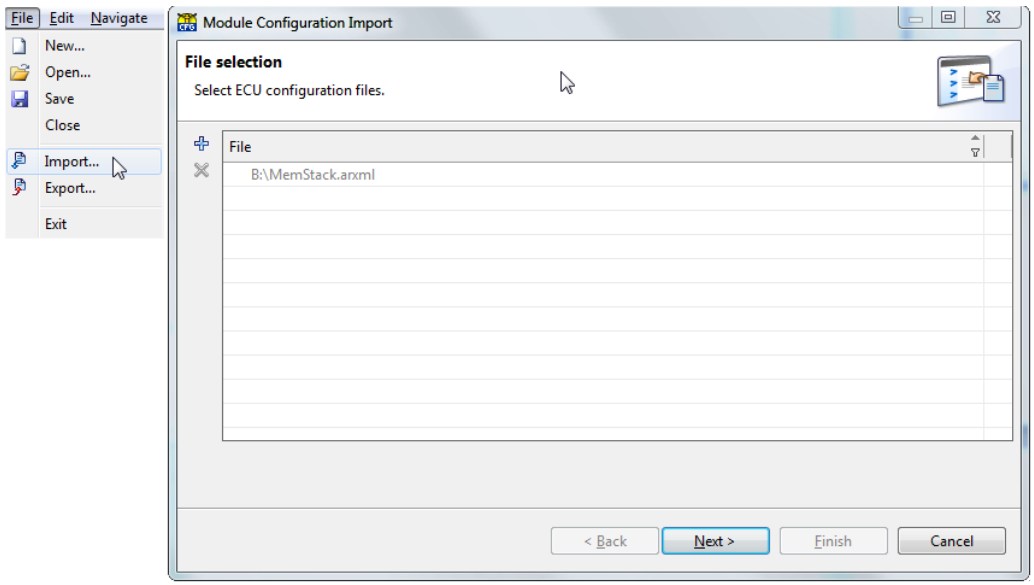

Figure 4-8 Import Configuration with DaVinci Configurator

The DaVinci Configurator automatically adds the new modules to your configuration.

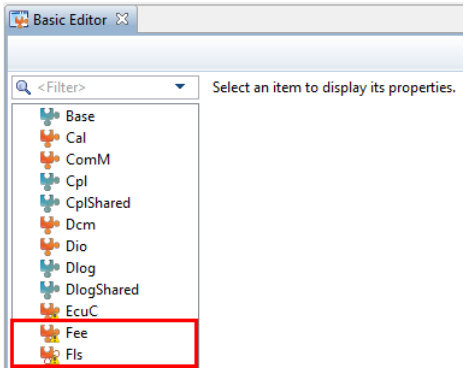

Figure 4-9 Imported MEM Stack Modules

3.3 Step 3 Adapt the Imported Configuration to Your Needs

Once the import is done you can start to adapt the configuration to the needs of the bootloader. As we learned in chapter 2.3 we have to switch the FEE into the bootloader mode.

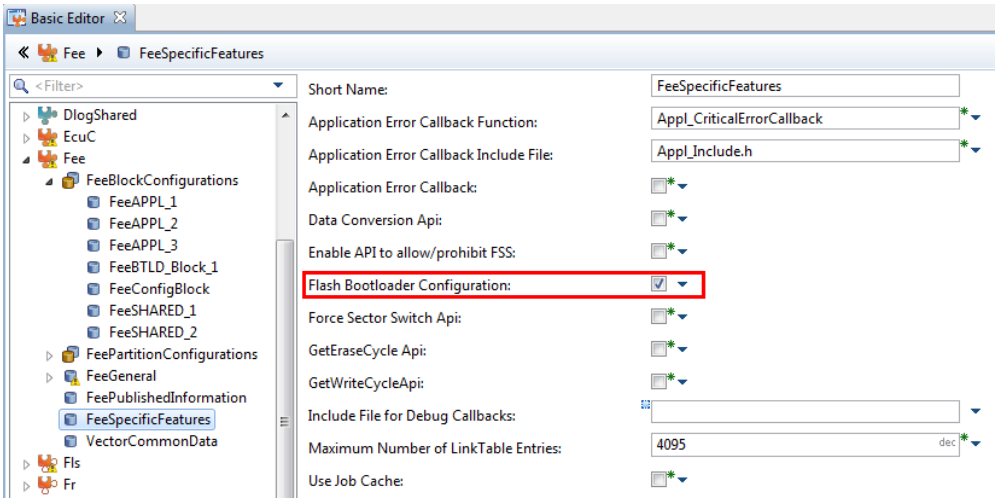

Figure 4-10 FEE Bootloader Mode

Now we can delete the blocks which are not needed in the bootloader. This can be easily done by the multi select feature of the DaVinci Configurator Pro. Mark the blocks you want to delete and hit the delete button.

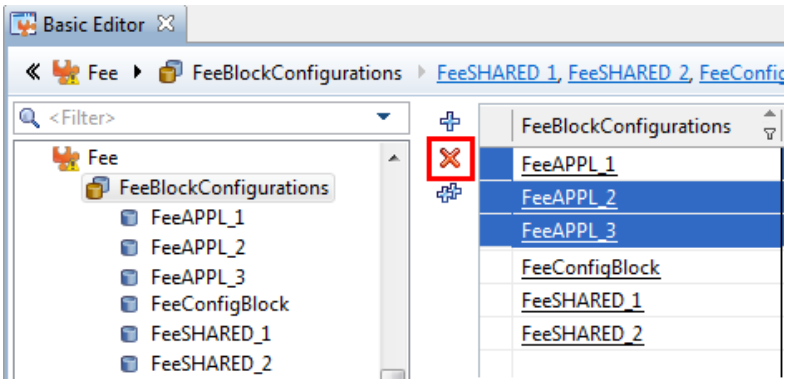

Figure 4-11 Delete Not Used FEE Blocks

Last step: Disable features which are enabled for the application configuration but not used in the bootloader. The development and production error detection for example. You can use the validation messages of DaVinci Configurator Pro. It will execute the necessary actions automatically.

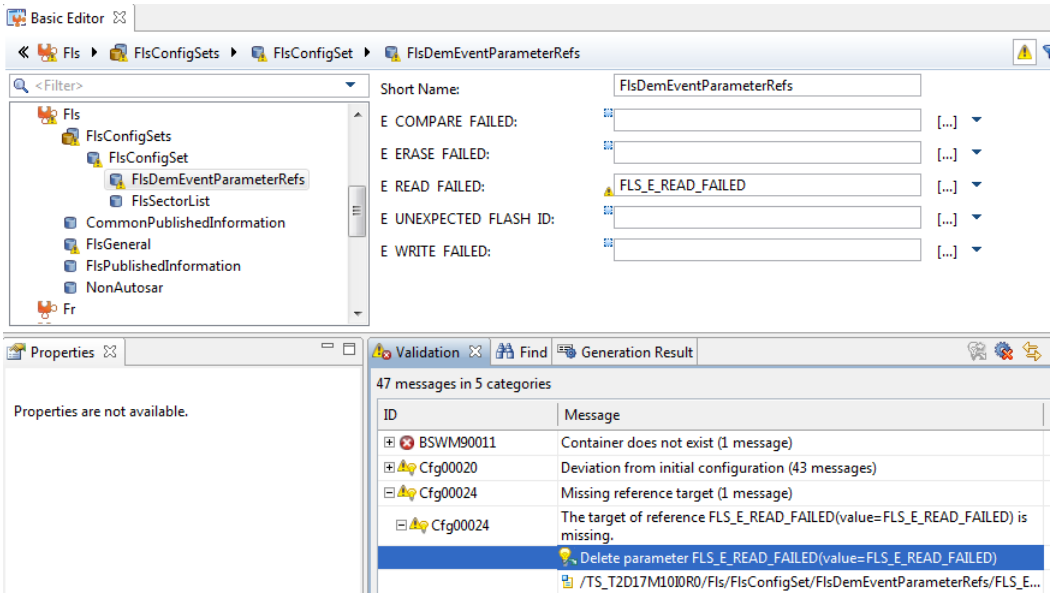

Figure 4-12 DaVinci Configurator Validation Rules

3.4 Step 4 Add the Standard Definition of the NvM

To be able to configure the parameter `NvMDataSelectionBits` described in chapter 2.2.1 you have to add the AUTOSAR standard NvM definition to your configuration. This is only the case if you do not want to use the NvM in the bootlader.

To do so, add the NvM standard definition using the project settings editor of the DaVinci Configurator Pro.

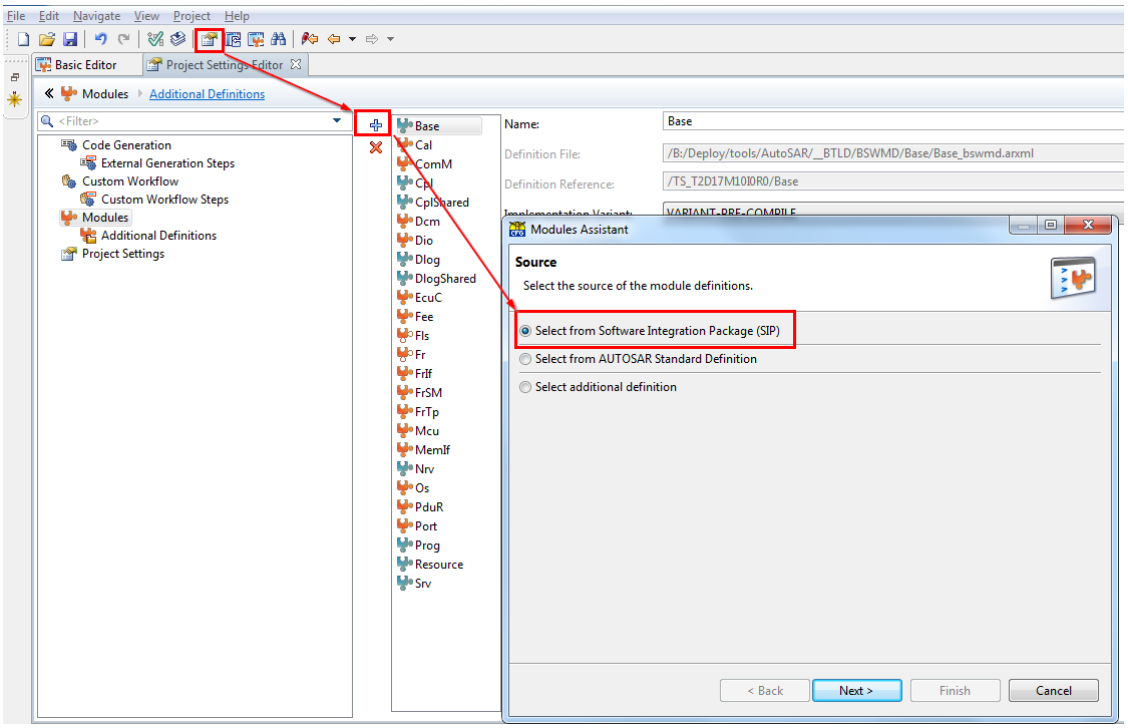

Figure 4-13 Add New Modules to Your Configuration

Now configure the parameter `NvMDataSelectionBits` to the same value as in the application configuration. All the other parameter errors and warnings can be ignored as we do not want to use nor generate the NvM module.

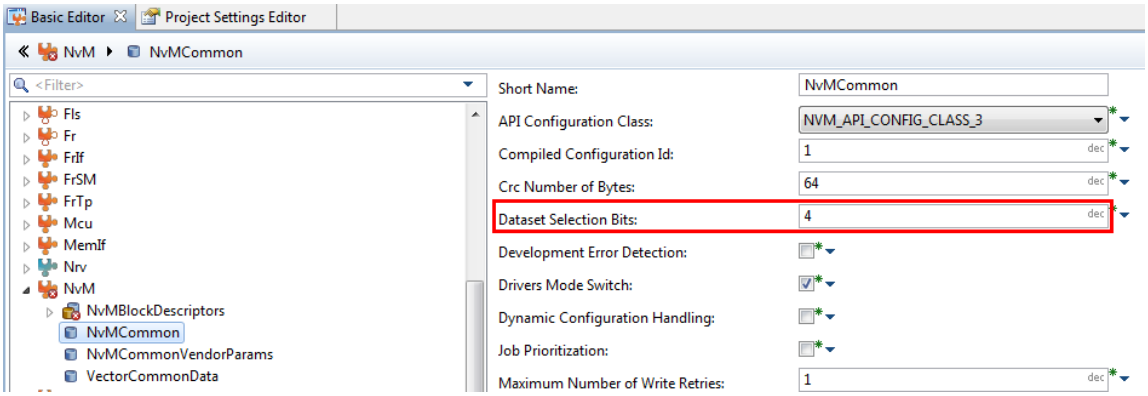

Figure 4-14 NvMDataSelectionBits

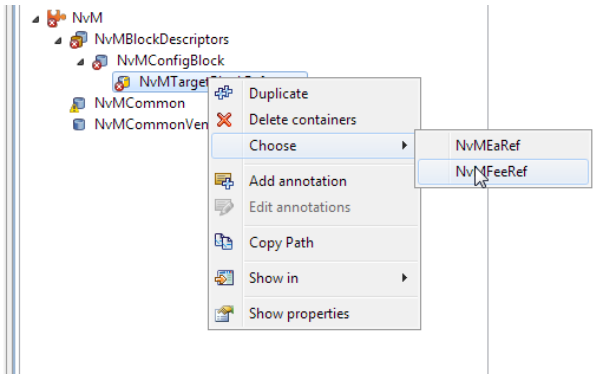

Figure 4-15 Choose Nvm Fee Ref

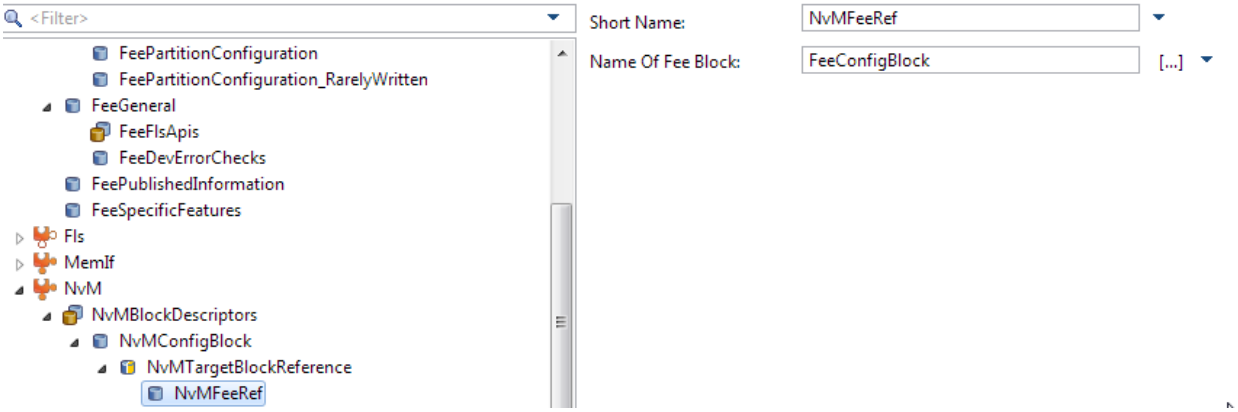

Figure 4-16 Nvm Stub Configuration to Allow Calculation of Valid Block Numbers

3.5 Special configuration Aspects in Non DaVinci Configurator Pro Environments

In case your flash bootloader is not configured with DaVinci Configurator Pro some further configuration aspects need to be considered.

3.5.1 Further ASR Component Stubs required

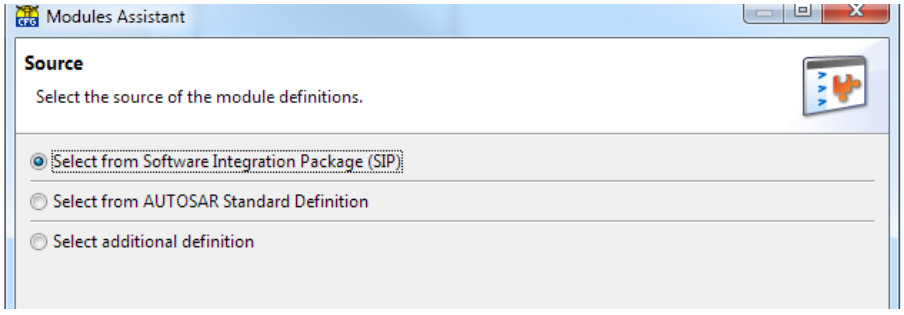

Figure 4-17 Add Stubs for Det, EcuC, MemIf from Application Software Integration Package

After adding stubs for Det, EcuC and MemIf, some error may occur. Follow DaVinci Configurator Pro propositions to resolve these problems (e.g. add Fee reference to MemIf, resolve EcuC related errors (HW-dependent)).

3.5.2 Generation of the Fee Config for the Fbl

In case the configuration was created for Fbl Fee configuration only, only the minimum configuration Fee and Fls need to be generated

Figure 4-18 Generation of Fbl Config

4.0 Usage of Fee Configuration in Flash Bootloader

Once you imported the Fee configuration to the flash bootloader and are able to generate Fls/Fee configuration, the created output has to be added and compiled with the flash bootloader project. Additionally static Asr modules and some static sub files have to be added to flash bootloader.

4.1 Modules Required in the Flash Bootloader

4.1.1 Asr BSW Modules to Be Added to Flash Bootloader

The following application core modules need to be added to your Fbl project:

Fee	.\Fee > all content
Det	.\Det > Det.h
Fls	.\Fls > all content
	.\Mcal_<Hw> (Register definitions for Fls)
MemIf	.\MemIf > all header files
Other	._Common > all
	.\include of your project > compiler_cfg.h
(remove further includes within compiler_cfg.h (like Rte_Compiler_Cfg.h)	

4.1.2 Asr Stub Modules to Be Used by the Fbl

The following stub modules are known to be required in the Fbl.

Det	Det.c/Det_Cfg.h: Provide Prototype for function Det_ReportError	
SchM	SchM_Fee.h	#define SchM_Enter_Fee_FEE_EXCLUSIVE_AREA_0()
		#define SchM_Exit_Fee_FEE_EXCLUSIVE_AREA_0()
	SchM_Fls.h	#define SchM_Enter_Fls(arg1, arg2)
		#define SchM_Exit_Fls(arg1, arg2)
		/* Function declaration is provided by SCHM */
		FUNC(void, FLS_CODE) Fls_MainFunction(void);

4.1.3 Generated Modules

The generated modules that should be added to the Fbl are

Fee	Fee_Cfg.h
	Fee_Lcfg.c
	Fee_PrivateCfg.h
Fls	Fls_Cfg.h

4.2 Initialization in Fbl

Depending on when the required Fee elements are to be read, the initialization has to happen early enough to allow access to the required elements.

4.2.1 Initialization Required

Fls and Fee need to be initialized. In both initializations the following logic applies (replace Fxx with Fee/Fls)

```
Call Fxx_Init()
while MEMIF_IDLE != Fxx_GetStatus() call Fxx_MainFunction()
```

Fee specific:

```
Depending on configuration call Fee_EnableFss()
#if (FEE_FSS_CONTROL_API == STD_ON)
   Fee_EnableFss();
#endif
```

The location where the initialization needs to be done depends on the elements that need to be accessed via Fee. If elements need to be accessed before decision to jump to application (e.g. valid flag(s)), do the initialization within `ApplFblInit()`, otherwise within `ApplFblStartup()`, `initposition == kStartupPostInit`. If you need to do the initialization within `ApplFblInit()`, be sure to initialize watchdog and timer early in your configuration (set `FBL_ENABLE_PRE_WDINIT / FBL_ENABLE_PRE_TIMERINIT`)

4.3 Synchronized Calls to Fee

The bootloader usually requires synchronized calls to the Fee. The Fee API is usually asynchronous. A wrapper to embed the asynchronous API to synchronized calls should therefore be used.

4.3.1 Synchronized Read

Create a function to read from Fee in a synchronous way, e.g. with prototype

```
FblResult ApplFblNvRead ( vuint16 blockNumber, vuint16 blockOffset, V_MEMRAM1 vuint8
V_MEMRAM2 V_MEMRAM3 * pBuffer, vuint16 length )
```

Required implementation:

```
If  Fee_Read(blockNumber, blockOffset, pBuffer, length ) == E_OK
{
      While MEMIF_IDLE != Fee_GetStatus()
      {
            Call Fls_MainFunction()
            Call Fee_MainFunction()
      }
      //Check  Fee_GetJobResult() == MEMIF_JOB_OK / MEMIF_BLOCK_INCONSISTENT /
                                     MEMIF_BLOCK_INVALID
      If Fee_GetJobResult() == MEMIF_JOB_OK
            return OK
      Else If Fee_GetJobResult() == MEMIF_BLOCK_INCONSISTENT OR
                                    MEMIF_BLOCK_INVALID
            fill read buffer with FBL_FLASH_DELETED
            return OK
      Else
            return failed
}
Else return failed
```

4.3.2 Synchronized Write

Create a function to write to Fee in a synchronous way, e.g. with prototype

```
FblResult ApplFblNvWrite ( vuint16 blockNumber, V_MEMRAM1 vuint8   V_MEMRAM2
V_MEMRAM3 * pBuffer)
```

Required implementation:

```
If  Fee_Write(blockNumber, pBuffer) == E_OK
{
      While MEMIF_IDLE != Fee_GetStatus()
      {
            Call Fls_MainFunction()
            Call Fee_MainFunction()
      }
      If  Fee_GetJobResult() == MEMIF_JOB_OK
            return Ok
      Else
            return failed
}
Else return failed
```

4.4 Data elements to Be Read and Written by Fbl

In order to read and write from and to the Fee, the API requires the block numbers of elements. These block numbers can be found in `Fee_cfg.h`, macros starting with `FeeConf_FeeBlockConfiguration_Fee`, e.g.:

```
/*****************************************************************************
 * GENERAL CONFIGURATION PARAMETER
 *****************************************************************************/

#define FeeConf_FeeBlockConfiguration_FeeConfigBlock (64UL)
#define FeeConf_FeeBlockConfiguration_FeeFbl_ApplNBID (48UL)
#define FeeConf_FeeBlockConfiguration_FeeFbl_KeyNBID (32UL)
#define FeeConf_FeeBlockConfiguration_FeeFbl_SBATicket (16UL)
```

Figure 4-1 Example Fee Block Numbers Used in Flash Bootloader

The length information required for read information has to be defined somewhere in the flash bootloader, e.g. for the example given above:

```
#define FBL_NV_APPNBID_FEE_LENGTH                    2u
#define FBL_NV_KEYNBID_FEE_LENGTH                    2u
#define FBL_NV_SBAT_FEE_LENGTH                       822u
```

4.5 Configure Existing Flash Bootloader NVM Data for Fee

Existing NVM elements are read and written via the `fbl_apnv.c/.h` module. Usually functions and or macros are prepared to map to the generated WrapNv_cfg.h macros usable for either EEPROM or Eepm. For Fee configuration these elements now have to be mapped to synchronized Fee read/write functionality.

An example call that can be mapped to the corresponding fbl_apnv.c/.h function for reading the above mentioned element Fbl_SBATicket e.g. would be then:

```
ApplFblNvRead(FeeConf_FeeBlockConfiguration_FeeFbl_SBATicket, 0u,
(buffer),FBL_NV_SBAT_FEE_LENGTH)
```

4.6 Conclusion

After these steps you are ready to read and write the shared blocks within both configurations. Please test and verify your configuration.

5.0 Contacts

For a full list with all Vector locations and addresses worldwide, please visit http://vector.com/contact/.