VECTOR

MICROSAR Classic MemMap

Technical Reference

AUTOSAR Memory Mapping Generator

Version 1.4.0

Authors	virbse, virleh, sreif
Status	Released

Document Information

History

Author	Date	Version	Remarks
virbse	2021-03-01	1.0.0	Initial version
virbse, virleh	2021-04-27	1.1.0	Added supported compiler (HighTec LLVM) Added new error check type Added ECUC MemorySections
virleh, virbse	2021-08-11	1.2.0	Added supported compiler (Diab LLVM) Fixed error in include structure Added hint for static code analysis Updated chapter *Embedded Implementation* Updated chapter *Additional Memory Sections* Added chapter *Definition of Terms* Added chapter *Configuration Hints*
virleh	2021-10-07	1.3.0	Added supported compiler (TexasInstruments LLVM) Added chapter *Near Addressing Modes*
sreif	2024-02-06	1.4.0	Changed name to MICROSAR Classic MemMap

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	Specification of Memory Mapping	R19-11
[2]	AUTOSAR	List of Basic Software Modules	R19-11

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

1 Introduction 6
1.1 Architecture Overview 7
1.2 Memory Mapping Overview 7
1.3 Definition of Terms 8
1.4 Supported Compilers 9

2 Functional Description 11
2.1 Features 11
2.1.1 Deviations 11
2.1.2 Additions / Extensions 12
2.2 Initialization 12
2.3 Main Functions 12
2.4 Error Handling 12
2.4.1 Activation 13
2.5 Automatic Pragma Generation 13
2.5.1 Preconditions 13
2.5.2 Activation 13
2.5.3 AUTOSAR Compliance 13
2.6 Compatibility Memory Mapping Header 14
2.6.1 Activation 14
2.7 Empty File Generation 14
2.7.1 Activation 14
2.8 Additional Memory Sections 14
2.8.1 Activation 15
2.9 Near Addressing Modes 15
2.9.1 Activation 15

3 Integration 16
3.1 Embedded Implementation 16
3.2 Compiler Restrictions 16
3.2.1 Tasking for ARM 17
3.3 Use-Cases 17
3.3.1 MemMap_ARM_Derivative 17
3.4 Default Memory Mapping 17
3.4.1 Generic Mapping 17
3.4.2 Specific Mapping 18
3.5 Static Code Analysis 19

4 API Description 20

5 Configuration 21
5.1 Configuration Variants 21
5.2 Configuration Hints 21
5.2.1 Split Generic Mappings 21
5.2.2 Overwrite Generic Mappings 23
5.2.3 Additional Memory Sections 24

6 Glossary and Abbreviations 26
6.1 Glossary 26
6.2 Abbreviations 26

7 Contact 27

Illustrations

Figure 1-1 File structure of the MICROSAR Classic MemMap 7
Figure 1-2 Memory Mapping Overview 8
Figure 5-1 Generic mapping in DaVinci Configurator 22

Tables

Table 1-1 Supported Compilers 10
Table 2-1 Supported AUTOSAR standard conform features 11
Table 2-2 Not supported AUTOSAR standard conform features 12
Table 2-3 Features provided beyond the AUTOSAR standard 12
Table 3-1 Implementation files 16
Table 3-2 Additional files 16
Table 6-1 Glossary 26
Table 6-2 Abbreviations 26

1 Introduction

This document describes the functionality, API and configuration of the AUTOSAR BSW module MemMap as specified in [1].

Supported Configuration Variants:	pre-compile	
Vendor ID:	MemMap_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	MemMap_MODULE_ID	195 decimal (according to ref. [2])

The MICROSAR Classic MemMap is the MICROSAR solution of the AUTOSAR memory mapping module. It is used to generate dedicated memory mapping header files for BSW modules as well as for software component types. These memory mapping header files contain all memory allocation keywords (derived from the `MEMORY-SECTIONS`) of a BSW module or software component type.

Memory allocation keywords are defined as preprocessor macros and always occur in pairs to open (START) and close (STOP) a memory section. Within these defines, compiler specific pragma statements are generated to map code, constants or variables into dedicated linker sections.

The compiler specific pragmas can either be added manually in the configuration or generated automatically by the MICROSAR Classic MemMap generator.

1.1 Architecture Overview

Since the MICROSAR Classic MemMap does not provide API functions, the following figure does not show any interfaces to adjacent modules. Instead, it shows the interaction of the various memory mapping header files.

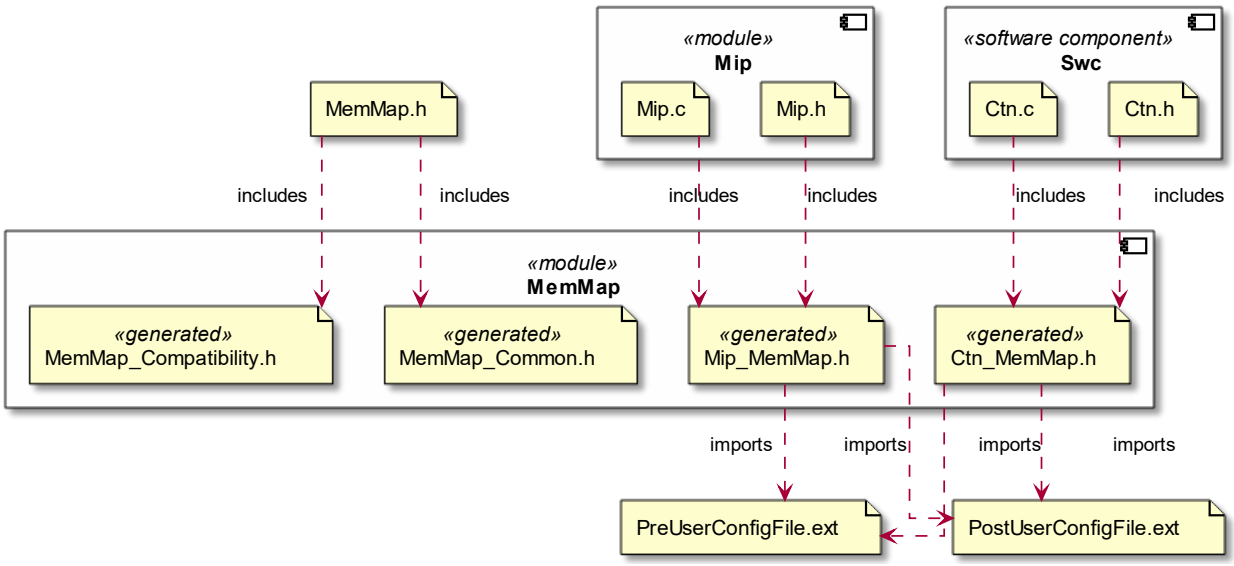

Figure 1-1 File structure of the MICROSAR Classic MemMap

1.2 Memory Mapping Overview

The following figure shows a simplified view of the relationships between the individual artifacts of the memory mapping mechanism.

It shows exemplarily how the function `Mip_MainFunction()` is mapped to the memory section `MIP_CODE` and thus gets the correct pragma statements in the generated *Mip_MemMap.h* via the `MSR_CODE` default mapping (see 3.2) of the MICROSAR Classic MemMap.

Note
The connection between the MICROSAR Classic MemMap and vLinkGen is a MICROSAR extension to the AUTOSAR specification and is only available if vLinkGen is also part of the project, i.e. active in the DaVinci Configurator project (see 2.5).

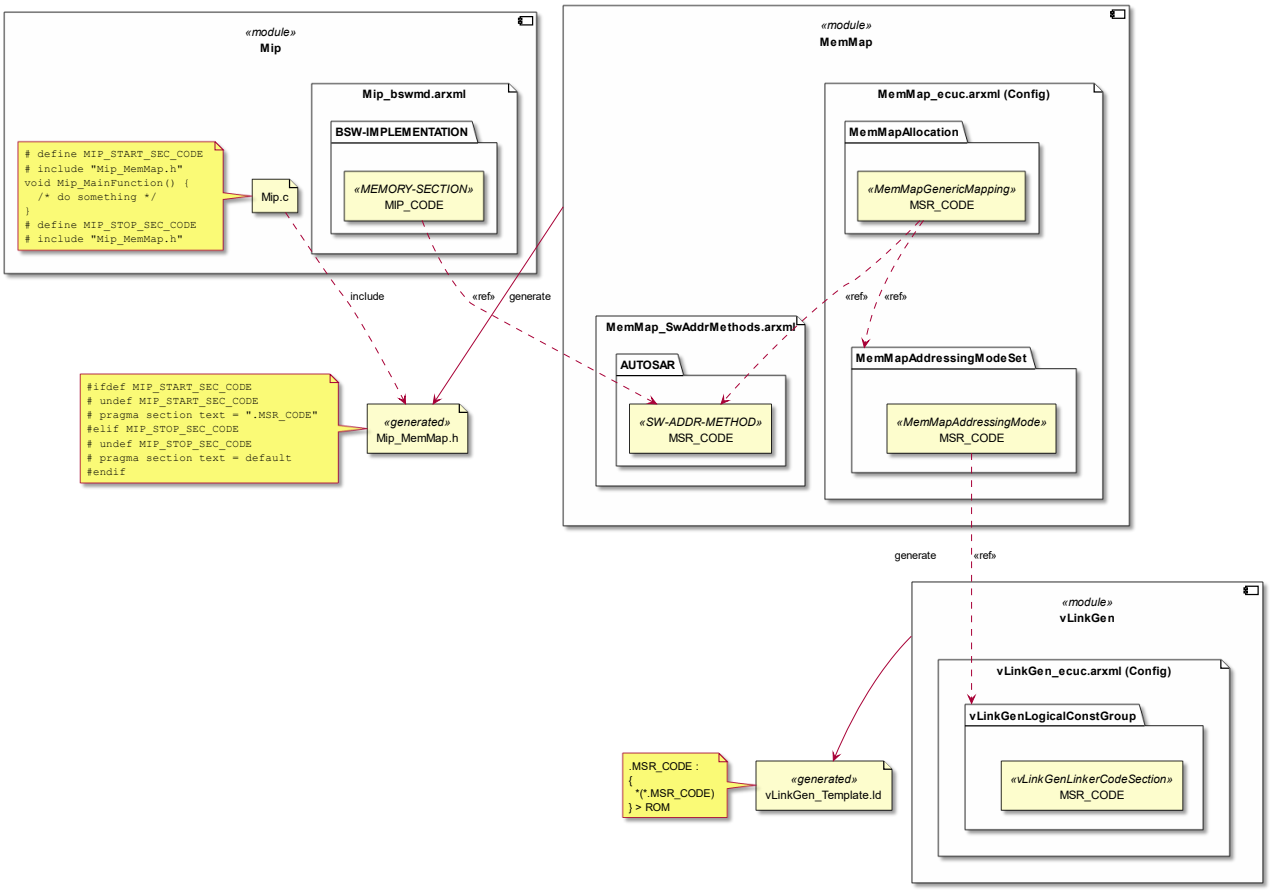

Figure 1-2 Memory Mapping Overview

1.3 Definition of Terms

This chapter describes the different terms used for the AUTOSAR memory mapping mechanism.

- **Memory Allocation Keywords** are defined as preprocessor macros and always occur in pairs to open (`START`) and close (`STOP`) a memory section. Within these defines, compiler specific pragma statements are generated to map code, constants or variables into dedicated linker sections. The syntax of the memory allocation keywords is defined as follows:

  `{PREFIX}_(START|STOP)_SEC_{NAME}`

  For details, please refer to [1].

- **Memory Sections** (`MEMORY-SECTION`) are the definition of an area to group certain functions, constants or variables of a BSW module or software component. For each of these memory sections, the memory allocation keywords are generated in the corresponding memory mapping header file.

  In AUTOSAR XML, these sections are defined as follows:

  BSW: `/BSW-IMPLEMENTATION/RESOURCE-CONSUMPTION/MEMORY-SECTION`

  SWC: `/SWC-IMPLEMENTATION/RESOURCE-CONSUMPTION/MEMORY-SECTION`

In addition, the MICROSAR Classic MemMap allows also to add additional memory sections in the ECUC (see 2.8).

- **SwAddrMethods** (`SW-ADDR-METHOD`) are the grouping of related memory sections. Each memory section must be mapped to a software addressing method.

  With the help of SwAddrMethods the default mapping of a memory section is ensured. However, this can be overwritten later in the configuration (see 3.4.2).

  The MICROSAR Classic MemMap provides a set of default SwAddrMethods including their complete mapping. This means that a memory section that is assigned to one of these SwAddrMethods is automatically assigned to the correct linker section (see 2.5). The default SwAddrMethods are defined in *MemMap_SwAddrMethods.arxml*.

- **Addressing Mode Sets** (`MemMapAddressingModeSet`) contain a list of Addressing Modes (`MemMapAddressingMode`).
  - In the **Addressing Modes**, either the compiler-specific pragmas are specified or a vLinkGen group is referenced. The latter is a MICROSAR extension that allows the MICROSAR Classic MemMap to automatically generate the correct pragmas (see 2.5).
- **Allocations** (`MemMapAllocation`) link an addressing mode set with several memory sections. Thus the MICROSAR Classic MemMap can assign the correct pragmas to the memory sections and generate them accordingly.

  The linking is done either via a SwAddrMethod (`MemMapGenericMapping`) or individual memory sections directly (`MemMapSectionSpecificMapping`). In addition, it is also possible to link memory sections created in the ECUC with addressing mode sets (`MemMapAdditionalSpecificMapping`). For details see 3.2.

1.4 Supported Compilers

The MICROSAR Classic MemMap offers the possibility to automatically generate compiler specific pragma statements (see 2.5). The following table shows for which compilers the pragma generation is supported. The entry “-“ in the **Version Restrictions** column indicates that there are no known restrictions regarding the compiler version.

Compiler	Version Restrictions
ARM	only 5.x.x, 6.x.x or later
Wind River Diab	-
Wind River Diab (LLVM)	-
Green Hills	-
HighTec (GCC)	-
HighTec (LLVM)	-
IAR	only 7.x.x or later
NXP (GCC)	-
Renesas	-
Tasking	-

Texas Instruments	-
Texas Instruments (LLVM)	-

Table 1-1 Supported Compilers

For unsupported compilers, the pragmas can be specified manually in the configuration, as specified by AUTOSAR (see 2.5.3).

Note
Not all versions were tested for each compiler. It is assumed that all common versions will work with the generated pragma statements.

2 Functional Description

2.1 Features

The features listed in the following tables cover the functionality specified for the MICROSAR Classic MemMap.

The AUTOSAR standard functionality is specified in [1], the corresponding features are listed in the tables

> Table 2-1 Supported AUTOSAR standard conform features

> Table 2-2 Not supported AUTOSAR standard conform features

Vector Informatik provides further MemMap functionality beyond the AUTOSAR standard. The corresponding features are listed in the table

> Table 2-3 Features provided beyond the AUTOSAR standard

The following features specified in [1] are supported:

Supported AUTOSAR Standard Conform Features
Generation of <*Mip*>*_MemMap.h* for each BSW module description which is part of the input configuration.
Generation of <*Ctn*>*_MemMap.h* for each software component type which is part of the input configuration.
Error checks to prevent misusage of memory allocation keywords (see 2.4).

Table 2-1 Supported AUTOSAR standard conform features

2.1.1 Deviations

The following features specified in [1] are not supported:

Category	Description
Config	`MemMapAddressingModeStart` Multiplicity has been changed from 1:1 to 0:1. Necessary to support automatic pragma generation feature (see 2.5).
Config	`MemMapAddressingModeStop` Multiplicity has been changed from 1:1 to 0:1. Necessary to support automatic pragma generation feature (see 2.5).
Config	`MemMapSwAddressMethodRef` Multiplicity has been changed from 1:1 to 1:N. Improves configurability by allowing a list of `SW-ADDR-METHODS` to be grouped into a `MemMapGenericMapping`.
Config	`MemMapMemorySectionRef` Multiplicity has been changed from 1:1 to 1:N. Improves configurability by allowing a list of `MEMORY-SECTIONS` to be grouped into a `MemMapSectionSpecificMapping`.
Config	`MemMapCompilerMemClassSymbolImpl` Not supported

Config	MemMapMappingSelectorRef Not supported
Config	MemMapMappingSelector Not supported
Config	MemMapGenericCompilerMemClass Not supported

Table 2-2 Not supported AUTOSAR standard conform features

2.1.2 Additions / Extensions

The following features are provided beyond the AUTOSAR standard:

Features Provided Beyond the AUTOSAR Standard
Automatic pragma generation in compiler specific syntax (see 2.5).
Compatibility memory mapping header file (see 2.6).
Empty memory mapping header files (see 2.7).
Additional memory sections (see 2.8).

Table 2-3 Features provided beyond the AUTOSAR standard

2.2 Initialization

The MICROSAR Classic MemMap only generates memory mapping header files, thus initialization is not applicable.

2.3 Main Functions

The MICROSAR Classic MemMap only generates memory mapping header files, thus main functions are not applicable.

2.4 Error Handling

The MICROSAR Classic MemMap only generates memory mapping header files, thus AUTOSAR standard error reporting mechanisms for development or production code (via modules like DET or DEM) are not applicable. Instead, it provides the possibility to report certain errors in the use of the memory mapping allocation keywords via error directives of the preprocessor. The following errors can be detected as required by the AUTOSAR standard:

- No START preceded by another START memory allocation keyword.
- No STOP without the corresponding START memory allocation keyword.
- No matching memory allocation keyword found (MEMMAP_ERROR).

In addition to the AUTOSAR standard, the MICROSAR Classic MemMap can detect the following errors:

- Simultaneous use of multiple memory allocation keywords.

2.4.1 Activation

To activate the generation of the error checks globally, the following parameter must be set to a value other than NON:

/MICROSAR/MemMap/MemMapGeneral/MemMapErrorChecks

If set, the error checks are generated for all memory allocation keywords in all memory mapping header files.

2.5 Automatic Pragma Generation

In addition to the AUTOSAR standard, the MICROSAR Classic MemMap provides the possibility to automatically generate pragma statements in the appropriate compiler syntax. These statements are generated within the memory allocation keywords in the memory mapping header files.

If used, the configuration also remains compiler independent and can be reused for other projects.

2.5.1 Preconditions

To use the automatic pragma generation, the following preconditions must be met:

- MICROSAR vLinkGen must be active in the DaVinci Configurator project.
- The selected compiler must be supported by MICROSAR Classic MemMap (see 1.3).
- All MEMORY-SECTIONS mapped to a MemMapAddressingMode that uses automatic pragma generation shall only reference SW-ADDR-METHODS with a defined sectionType. If sectionType is equal to VAR or VAR_FAST, the attribute sectionInitializationPolicy must also be set.

2.5.2 Activation

To activate the automatic pragma generation a vLinkGen logical group can be selected for each MemMapAddressingMode via the following reference:

/MICROSAR/MemMap/MemMapAddressingModeSet/MemMapAddressingMode/
MemMapLinkerLogicalGroupRef

If set, the MemMapAddressingMode is mapped into this logical group, respectively linked into one of the included linker sections. The selection of the correct linker section is done automatically, depending on the type and initialization policy of the MEMORY-SECTIONS mapped to the MemMapAddressingMode.

2.5.3 AUTOSAR Compliance

It is still possible to specify pragma statements directly in the configuration and generate them into the memory allocation keywords, as it is specified by AUTOSAR. To do so, the following parameters can be used to specify the pragma statements to start or to stop a memory section:

- /MICROSAR/MemMap/MemMapAddressingModeSet/MemMapAddressingMode/MemMapAddressingModeStart
- /MICROSAR/MemMap/MemMapAddressingModeSet/MemMapAddressingMode/MemMapAddressingModeStop

Note
Manually configured pragma statements will always overwrite the automatic pragma generation configured via the `MemMapLinkerLogicalGroupRef`.

2.6 Compatibility Memory Mapping Header

In addition to the specified memory mapping header files, the MICROSAR Classic MemMap provides the possibility to generate a compatibility memory mapping header file with the name *MemMap_Compatibility.h*.

It contains all memory allocation keywords for all BSW modules and can be included in other general memory mapping header files like *MemMap.h* or *MemMap_Common.h*. In MICROSAR SIPs it is included in the former one by default.

2.6.1 Activation

To activate the generation of the compatibility memory mapping header file, the following parameter must be set to a value other than `NON`:

`/MICROSAR/MemMap/MemMapGeneral/MemMapCompatibilityGeneration`

2.7 Empty File Generation

For testing purposes, the MICROSAR Classic MemMap supports the generation of either empty memory mapping header files or memory allocation keywords without pragma statements.

This can be useful to reduce build times during development of a project or to build the project for tests on a host, where memory mapping is not applicable (e.g. VTT).

2.7.1 Activation

- To activate the empty memory mapping header file generation, the following parameter must be set to `EMPTY`:

  `/MICROSAR/MemMap/MemMapGeneral/MemMapGeneration`

- To activate the generation of memory allocation keywords without pragma statements, the following parameter must be set to `FALSE`:

  `/MICROSAR/MemMap/MemMapGeneral/MemMapGeneratePragmas`

2.8 Additional Memory Sections

In addition to the `MEMORY-SECTIONS` defined by AUTOSAR, the MICROSAR Classic MemMap also supports the definition of additional memory sections in the ECUC. These are generated into their associated memory mapping header files in the same way as the AUTOSAR `MEMORY-SECTIONS`. The additional memory sections are mapped to `MemMapAddressingModeSets` via the generic and specific mapping mechanism.

This type of memory section is easier to create because it does not require direct modification of the AUTOSAR XML files. This might especially be useful for rapid prototyping or to work around known issues.

2.8.1 Activation

An additional memory section can be created and configured via the following container:

`/MICROSAR/MemMap/MemMapModule/MemMapAdditionalMemorySection`

For a better usability, additional memory sections are grouped underneath a `MemMapModule` container.

This container can also be assigned to a certain module in the configuration. With this optional reference set for a `MemMapModule`, the MICROSAR Classic MemMap is able to generate additional memory sections to the *<Mip>_MemMap.h* file. But it is also possible to define own file names and prefixes for each additional memory section.

Examples of how to create and map additional memory sections can be found in chapter 5.2.3.

2.9 Near Addressing Modes

To increase the performance of the software, Near Addressing Modes (also called Fast Addressing Modes) might be used. In that case, constants and variables must be mapped into linker sections with special pragmas.

The automatic pragma generation of the MICROSAR Classic MemMap (see 2.5) also supports the generation of special pragmas for Near Addressing Modes (e.g., sdata, sbss, sconst).

2.9.1 Activation

- To activate the automatic pragma generation for Near Addressing Modes, the following parameter must be set to `TRUE`:

  `/MICROSAR/MemMap/MemMapGeneral/MemMapGeneratePragmasForNearAddressing`

- If the MICROSAR OS is used, the following parameter must also be set to `TRUE`:

  `/MICROSAR/Os/OsOS/OsGenerateMemMapForNearAddressing`

Caution
Near Addressing Modes are not supported when compiling/linking for an ARM core. In that case both parameters described above must be set to `FALSE`. Otherwise, compiler/linker errors might occur.

3 Integration

This chapter gives necessary information for the integration of the MICROSAR Classic MemMap into an application environment of an ECU.

3.1 Embedded Implementation

The delivery of the MICROSAR Classic MemMap contains these source code files:

File Name	Description	Integration Task
<Mip>_MemMap.h	Generated memory mapping header file for a dedicated BSW module.	-
<Ctn>_MemMap.h	Generated memory mapping header file for a dedicated software component type.	-
MemMap_Compatibility.h	Generated memory mapping header file for compatibility with non AUTOSAR 4.x compliant BSW modules. Contains all memory allocation keywords of all BSW modules from the input configuration.	
MemMap_Common.h	Generated memory mapping header file for compatibility with non AUTOSAR MemMap 4.x compliant BSW modules. Contains standard memory allocation keywords for BSW modules not using the AUTOSAR MemMap 4.x mechanism. For example: `START_SEC_CODE` and `STOP_SEC_CODE`.	-

Table 3-1 Implementation files

In addition to the source code files described above, the MICROSAR Classic MemMap delivery also contains the following files:

File Name	Description	Integration Task
MemMap_SwAddrMethods.arxml	Contains the static set of default `SW-ADDR-METHODS`. These `SW-ADDR-METHODS` are automatically mapped to `MemMapAddressingModeSets` in the MICROSAR Classic MemMap configuration.	-

Table 3-2 Additional files

3.2 Compiler Restrictions

The following chapter indicates which features cannot be used at all or only with restrictions on certain compilers.

Parameters which are set to read-only in the DaVinci Configurator are not supported for the selected compiler.

3.2.1 Tasking for ARM

In case the Tasking compiler is used to compile/link for an ARM core, the following restrictions apply as compiler/linker errors might occur otherwise.

- The following parameter must be set to `FALSE`:

  `/MICROSAR/MemMap/MemMapGeneral/MemMapTypedPragmas`

- The generation of pragmas for Near Addressing Modes must be disabled (see 2.9)

3.3 Use-Cases

When creating a new ECU project containing the MICROSAR Classic MemMap, it is necessary to specify the concrete Use-Case. This is done by setting values for the following Use-Cases:

3.3.1 MemMap_ARM_Derivative

The value of this Use-Case must be set to *Enabled* when compiling/linking for an ARM core, otherwise it must be set to *None*. If this Use-Case is set to *Enabled*, the configuration is automatically adapted to comply with the restrictions for ARM cores described in 2.9 and 3.2.1.

3.4 Default Memory Mapping

For memory sections that do not require explicit mapping into specific memory areas, MICROSAR Classic MemMap provides a default mapping.

This means that the memory sections are automatically assigned to a `MemMapAddressingMode` and thus also to the correct linker section (only in combination with vLinkGen). In this case, the correct compiler specific pragma statements can be generated into the memory mapping header files (see 2.5).

To ensure this, a recommended configuration is provided by the MICROSAR Classic MemMap module. This contains predefined `SW-ADDR-METHODS` and `MemMapAddressingModeSets` (incl. `MemMapAddressingModes`), which in turn can be referenced by the `MEMORY-SECTIONS`. The available `MEMORY-SECTIONS` are usually defined by the BSW modules and software component types.

If vLinkGen is not included in the project, the compiler specific pragmas must be entered manually in the configuration (see 2.5.3).

The provided default mapping of the `MEMORY-SECTIONS` can be overwritten at any time in the configuration in the following two ways.

3.4.1 Generic Mapping

Generic (static) mapping of a `MEMORY-SECTION` to a predefined `SW-ADDR-METHOD` is usually defined by the BSW modules or software component types and cannot be changed during the configuration.

However, the mapping of an `SW-ADDR-METHOD` to a `MemMapAddressingModeSet` is provided by the recommended configuration. This mapping can be changed in the configuration via the following container:

/MICROSAR/MemMap/MemMapAllocation/MemMapGenericMapping

Either the existing mapping can be changed or a new MemMapGenericMapping can be created where the SW-ADDR-METHOD is mapped differently. But each SW-ADDR-METHOD can only be referenced once in the generic mappings.

An example how to split the provided generic mappings can be found in chapter 5.2.1.

Note
An overview of all selectable MICROSAR SW-ADDR-METHODS can be found in the file *MemMap_SwAddrMethods.arxml*.

Note
As the additional memory sections also reference SW-ADDR-METHODS they use the same generic mapping mechanism as AUTOSAR MEMORY-SECTIONS.

3.4.2 Specific Mapping

The MICROSAR Classic MemMap also allows a direct (specific) mapping of a MEMORY-SECTION to a predefined or newly created MemMapAddressingModeSet. This can be set manually at configuration time via the following container:

/MICROSAR/MemMap/MemMapAllocation/MemMapSectionSpecificMapping

Each MEMORY-SECTION can only be referenced once in the specific mappings.

An example how to configure a specific mapping can be found in chapter 5.2.2.

Note
Specific mappings always overwrite predefined or manually configured generic mappings from 3.2.1.

Note
The additional memory sections also support the specific mapping mechanism. However, the following container must be used for them instead:
/MICROSAR/MemMap/MemMapAllocation/MemMapAdditionalSpecificMapping

3.5 Static Code Analysis

When running static code analysis tools, the preprocessor definition `MICROSAR_DISABLE_MEMMAP` should be set. It disables the memory mapping header files via the preprocessor and thus also the use of compiler specific pragmas. This helps to avoid errors, since static code analysis tools often cannot handle pragmas.

4 API Description

The MICROSAR Classic MemMap only generates memory mapping header files, thus API Description is not applicable.

5 Configuration

The MICROSAR Classic MemMap is configured with Vectors DaVinci Configurator.

Additional information about all configuration attributes can be found within the configuration tool.

5.1 Configuration Variants

The MICROSAR Classic MemMap supports the configuration variants

> `VARIANT-PRE-COMPILE`

The configuration classes of the MICROSAR Classic MemMap parameters depend on the supported configuration variants. For their definitions please see the *MemMap_bswmd.arxml* file.

5.2 Configuration Hints

The following chapter gives some instructions how to configure the typical use cases in the MICROSAR Classic MemMap.

5.2.1 Split Generic Mappings

As described, the MICROSAR Classic MemMap provides a set of default SwAddrMethods. In addition, a recommended mapping of these SwAddrMethods to addressing mode sets is provided. However, this mapping is kept as simple as possible, and it might be necessary to refine it further.

As an example, all memory sections for uninitialized and non-cached data are to be linked into a special area in the target memory. This means all SwAddrMethods for `VAR_NOCACHE_NOINIT` data must be moved from the recommended generic mapping `MSR_VAR_NO_INIT` into their own.

This new generic mapping (e.g., `MSR_VAR_NOCACHE_NO_INIT`) can then be assigned to a new addressing mode set, which in turn references its own logical group in vLinkGen. This logical group is assigned in the vLinkGen configuration to the corresponding Memory Region Block, which corresponds to an area in the target memory.

Caution
This example requires the vLinkGen active in the DaVinci Configurator project.

If vLinkGen is not part of the project, pragmas can be entered manually in the corresponding addressing mode set.

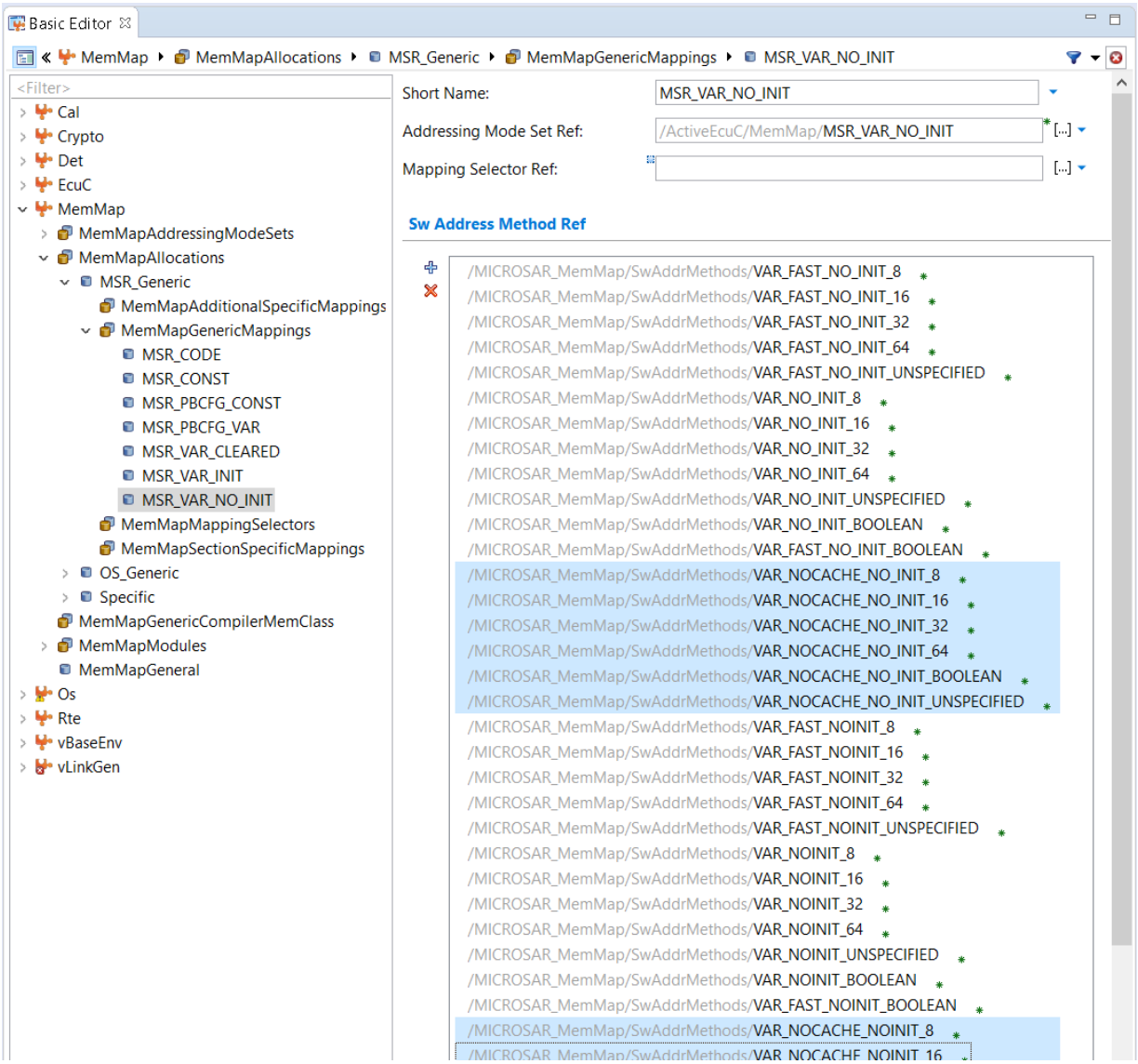

Figure 5-1 Generic mapping in DaVinci Configurator

To map the SwAddrMethods shown in Figure 1-1 as desired, the following steps must be performed:

1. Duplicate the addressing mode set **MSR_VAR_NO_INIT** (`MemMapAddressingModeSet`) including the underlying addressing mode (`MemMapAddressingMode`) and name the two containers accordingly, for example:

   MSR_VAR_NOCACHE_NO_INIT

2. Duplicate the target of the **vLinkGen Logical Group Ref MSR_VAR** (`MemMapLinkerLogicalGroupRef`) in the addressing mode of the newly created addressing mode set. Including all section groups and referenced linker sections. The new logical group must then be assigned to the corresponding Memory Region Block. However, this is not part of this description.

3. Duplicate the generic mapping **MSR_VAR_NO_INIT** (`MemMapGenericMapping`) and name it accordingly, for example:

MSR_VAR_NOCACHE_NO_INIT

4. Adjust the referenced SwAddrMethods in **Sw Address Method Ref** (`MemMapSwAddressMethodRef`) in both generic mappings (old and new) accordingly.
5. Set the **Addressing Mode Set Ref** (`MemMapAddressingModeSetRef`) in the new generic mapping to the previously created addressing mode set.
6. All memory sections that are assigned to a `VAR_NOCACHE_NO_INIT` SwAddrMethod are now linked to the desired location in the target memory via the new generic mapping `MSR_VAR_NOCACHE_NO_INIT` using the new addressing mode set.

5.2.2 Overwrite Generic Mappings

Sometimes it might be necessary to overwrite the generic mapping of a memory section. Which means the contents of a memory section shall be linked to a different location in the target memory than originally intended by the SwAddrMethod.

In the following example the memory section `ADC_CODE` of the component Adc is to be mapped into the linker section "`.AdcCodeSection`". In AUTOSAR XML the memory section is mapped to the MICROSAR default SwAddrMethod `CODE`.

The expected output for the GCC compiler in *Adc_MemMap.h* is as follows:

Example
Generated (simplified) output in *Adc_MemMap.h* for the Green Hills compiler:

```
#if defined ADC_START_SEC_CODE
# pragma ghs section text = ".AdcCodeSection"
# undef ADC_START_SEC_CODE
# undef MEMMAP_ERROR
#elif defined ADC_STOP_SEC_CODE
# pragma ghs section text = default
# undef ADC_STOP_SEC_CODE
# undef MEMMAP_ERROR
```

To generate the above shown *Adc_MemMap.h* accordingly, the following steps must be performed in the configuration:

1. Create a new `MemMapAddressingModeSet` container including one `MemMapAddressingMode` sub-container.
2. Fill the **Addressing Mode Start** (`MemMapAddressingModeStart`) with the compiler specific pragma section start statement. In this example:

   ```
   # pragma ghs section text = ".AdcCodeSection"
   ```

3. Fill the **Addressing Mode Stop** (`MemMapAddressingModeStop`) with the compiler specific pragma section stop statement. In this example:

```
# pragma ghs section text = default
```

4. Alternatively, the **vLinkGen Logical Group Ref** (`MemMapLinkerLogicalGroupRef`) can be set (see 2.5). Then steps 2 and 3 can be skipped. How to create logical groups and its underlying linker sections in the vLinkGen configuration is not scope of this document.
5. Create a new `MemMapAllocation` container including one `MemMapSectionSpecificMapping` sub-container.
6. Add the previously created `MemMapAddressingModeSet` to the **Addressing Mode Set Ref** (`MemMapAddressingModeSetRef`).
7. Add the memory section `ADC_CODE` to the **Memory Section Ref** (`MemMapMemorySectionRef`).

5.2.3 Additional Memory Sections

As described in chapter 2.8, the MICROSAR Classic MemMap provides the possibility to define additional memory sections in the ECUC. This chapter shows how an additional memory section can be created and specifically mapped.

5.2.3.1 Create Additional Memory Section

To create an additional memory section, follow these steps:

1. Create a new `MemMapModule` container and name it as desired.
2. Ensure that **Generate Additional Memory Sections** (`MemMapGenerateAdditionalMemorySections`) is enabled.
3. Optionally set the **Module Ref** (`MemMapModuleRef`) to an existing module in the configuration.

   With the help of this reference the MICROSAR Classic MemMap can generate the additional memory sections into the *<Mip>_MemMap.h* file. Also, it can retrieve the correct prefix for the memory allocation keywords. However, both, the file name and the prefix, can be overwritten for each additional memory section individually.
4. Create a new `MemMapAdditionalMemorySection` sub-container under the previously created `MemMapModule` which represents one memory section.
5. Optionally set an **Alignment** (`MemMapMemorySectionAlignment`) for the additional memory section. If nothing is specified, `UNSPECIFIED` will be used.
6. Optionally set a **Filename** (`MemMapMemorySectionFileName`) to specify the name of the memory mapping header file into which this memory section shall be generated. If set, the file name retrieved from the Module Ref will be overwritten. At least one of the parameters (Module Ref or Filename) must be set.
7. Optionally set a **Prefix** (`MemMapMemorySectionPrefix`) which will be used for the generated memory allocation keywords of this memory section. If set, the prefix retrieved from the **Module Ref** will be overwritten. At least one of the parameters (Module Ref or Prefix) must be set.

8. Associate the memory section to a SwAddrMethod via the **Sw Address Method Ref** (`MemMapMemorySectionSwAddrMethodRef`). This is mandatory to ensure a default mapping. However, this can be overwritten specifically later via an `MemMapAdditionalSpecificMapping`.
9. Optionally set the **Symbol** (`MemMapMemorySectionSymbol`) for the additional memory section which is used to generate the `{NAME}` part of the memory allocation keywords. If not set, the short name of the `MemMapAdditionalMemorySection` container is used.

Basic Knowledge
Memory allocation keywords are defined as follows:
`{PREFIX}_(START|STOP)_SEC_{NAME}`

5.2.3.2 Map Additional Memory Section

To overwrite the generic mapping (via `MemMapMemorySectionSwAddrMethodRef`) of an additional memory section, follow these steps:

1. Create a new `MemMapAllocation` container including a `MemMapSectionAdditionalSpecificMapping` sub-container and name them as desired.
2. Set the **Addressing Mode Set Ref** (`MemMapAddressingModeSetRef`) to the expected addressing mode set. See 5.2.2 how to create an addressing mode set.
3. Set the **Additional Memory Section Ref** (`MemMapAdditionalMemorySectionRef`) to the additional memory section(s) which shall be specifically mapped.

6 Glossary and Abbreviations

6.1 Glossary

Term	Description
BSWMD	The BSWMD is a formal notation of all information belonging to a certain BSW artifact (BSW module or BSW cluster) in addition to the implementation of that artifact.
Electronic Control Unit	Also known as ECU. Small embedded computer system consisting of at least one CPU and corresponding periphery which is placed in one housing.
Memory Mapping Header File	Header file generated by the MICROSAR Classic MemMap containing the memory allocation keywords of all assigned memory sections.
Memory Allocation Keyword	Refer to chapter 1.3.
Memory Section	Refer to chapter 1.3.
SwAddrMethod	Refer to chapter 1.3.
Addressing Mode Set	Refer to chapter 1.3.
Allocation	Refer to chapter 1.3.

Table 6-1 Glossary

6.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basic Software
CTN	Component Type Name (see [2]).
DEM	Diagnostic Event Manager
DET	Development Error Tracer
ECU	Electronic Control Unit
HIS	Hersteller Initiative Software
MemMap	Memory Mapping
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
MIP	Module Integration Package which is composed according `<Msn>[_<vi>_<ai>]` for BSW modules (see [2]).
SWC	Software Component
SWS	Software Specification
vLinkGen	Vector’s Compiler Specific Linker Script Generator

Table 6-2 Abbreviations

7 Contact

Visit our website for more information on

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

www.vector.com