MICROSAR vMem

Technical Reference

External SPI flash devices (vMem_30_XXspi01)

Version 1.0.0

Authors	Lukas Zirngibl, Andreas Lackner, Matthias Scheid
Status	Released

Document Information

History

Author	Date	Version	Remarks
Lukas Zirngibl, Andreas Lackner	[2019-01-24]	1.00.00	Initial creation

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SWS_DevelopmentErrorTracer.pdf	3.2.0
[2]	Vector	TechnicalReference_vMem.pdf	01.03.00
[3]	Micron Technology	Micron Serial NOR Flash Memory, mt25q-qlkt-U01-BBB-xxT	- Rev. D 05/18 EN

Scope of the Document

This technical reference describes the specific use of the vMem_30_XXspi01 driver software. It supplements the general vMem driver technical reference [2].

Please note that this document only describes low-level specific features. A general description can be found at the vMem__core [2].

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

Illustrations

Tables

1 Component History

The component history gives an overview over the important milestones that are supported in the different versions of the component.

Component Version	New Features
1.00.00	Creation of component.

Table 1-1 Component history

2 Introduction

This document describes the functionality, API and configuration of the MICROSAR BSW module vMem_30_XXspi01.

Supported AUTOSAR Release Version:	4.0.3, 4.x.x	
Supported Configuration Variants:	pre-compile	
Vendor ID:	VMEM_30_XXSPI01_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	VMEM_30_XXSPI01_MODULE_ID	255 decimal

The vMem_30_XXspi01 is the driver to access non-volatile memory on a range of different external flash devices connected via the controller's SPI interface. As a reference for the software development for external SPI flash memory devices the "MT25QL01" (see [3]) from "Micron Technology" has been used.

The vMem_30_XXspi01 is designed to support different SPI flash devices which share certain HW-specific features.

Please note that the supported flash chips have the same architecture and are behaviorally compatible with "Micron Technology MT25QL01".

2.1 Architecture Overview

Refer to [2].

2.2 Component Interfaces

The next figure shows the interfaces to adjacent modules of the vMem_30_XXspi01. Note that the infix of all interfaces of this specific driver is `vMem_30_XXspi01`.

All standard interfaces (e.g. `Read`, `Write`, `Erase` and `GetJobResult`) are described in [2].

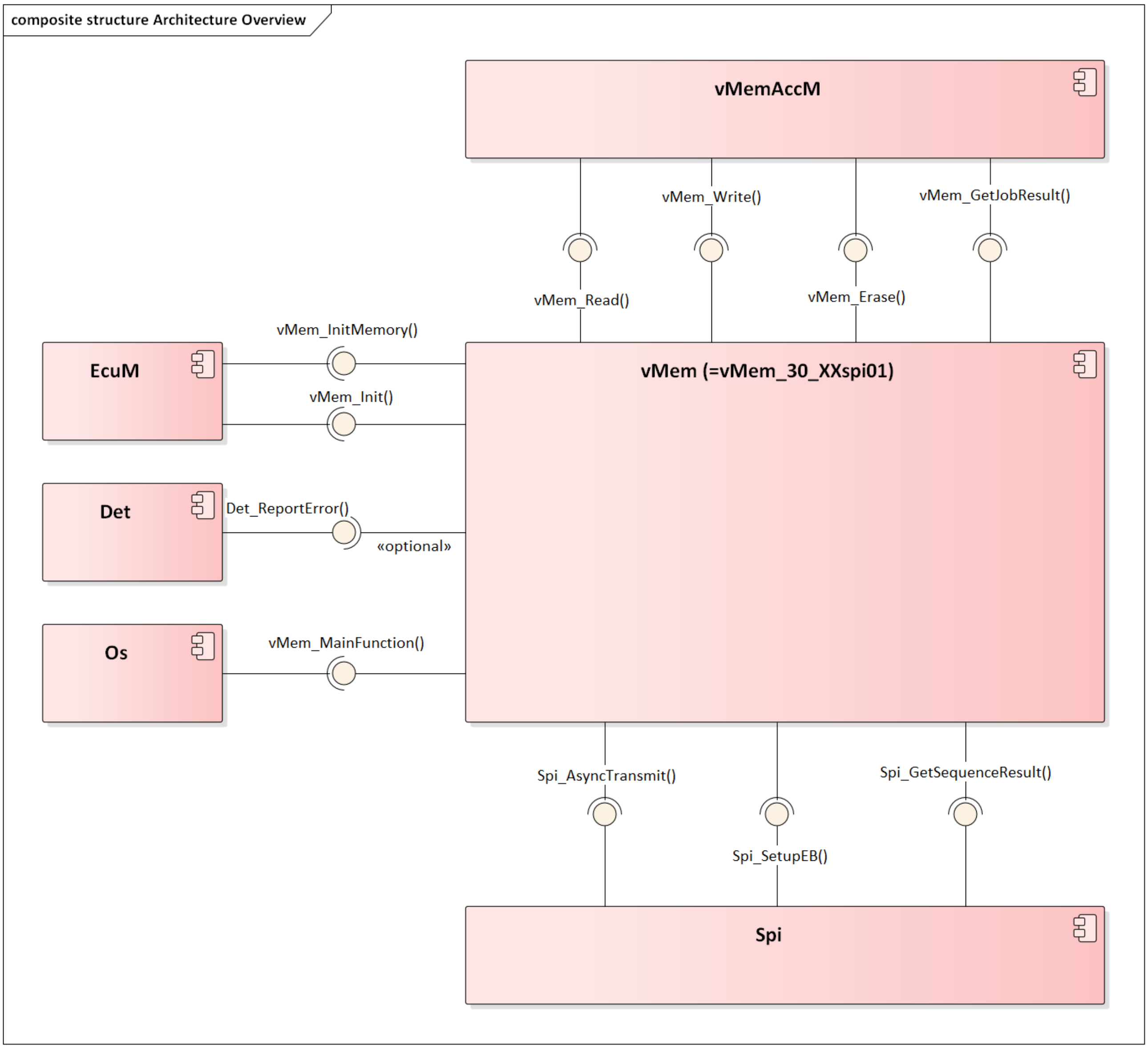


Figure 2-1 Interfaces to adjacent modules of the vMem_30_XXspi01

3 Functional Description

3.1 Features

The standard functionality, which is provided by all vMem drivers, see [2].

3.2 Limitations

3.2.1 Spi_DataType

The Spi_DataType is specified by AUTOSAR as either uint8, uint16 or uint32. But the vMem_30_XXspi01 is only capable of an uint8 as Spi_DataType. Therefore, it must be ensured that the Spi_DataType is defined as an 8 bit integer.

Otherwise it cannot be guaranteed that memory corruption will be prevented.

3.3 Initialization

Refer to [2].

3.4 Main Functions

The vMem_30_XXspi01 has one central processing function `vMem_30_XXspi01_MainFunction()`, which can be called with a fixed cycle time (see [2]).

3.5 Development Error Reporting

Refer to [2].

3.5.1 Hardware Software Interface

The vMem_30_XXspi01 accesses the register of the “MT25QL01” to check the flash status, flash operation and the error status. The SPI command sequence to read and to modify the registers for the standard interface are described in [3].

4 Integration

This chapter gives necessary information for the integration of the MICROSAR vMem_30_XXspi01 into an application environment of an ECU.

4.1 Scope of Delivery

Additionally to the files described in [2] the delivery of vMem_30_XXspi01 contains the files which are described in Table 4-1 and Table 4-2.

4.1.1 Static Files

File Name	Description
vMem_30_XXspi01_LL_FLsCmdSeqBuilder.c	The vMem_LL_FlsCmdSeqBuilder provides services to receive the configured high-level sequences. For each sequence the SeqBuilder provides a separate service function.
vMem_30_XXspi01_LL_FLsCmdSeqBuilder.h	Header file of the vMem_LL_FlsCmdSeqBuilder.
vMem_30_XXspi01_LL_FlsCmdSeqExecuter.c	The vMem_LL_FlsCmdSeqExecuter provides an interface to the underlying bus driver. It sets up and triggers high-level sequences received from the LL.
vMem_30_XXspi01_LL_FlsCmdSeqExecuter.h	Header file of the vMem_LL_FlsCmdSeqExecuter.
vMem_30_XXspi01_LL_Types.h	Defines Low-level specific data types.

Table 4-1 Static files

4.1.2 Dynamic Files

File Name	Description
vMem_30_XXspi01_Cbk.c	Contains SeqEndNotification for each configured sequence.
vMem_30_XXspi01_Cbk.h	Contains declaration of SeqEndNotification for each configured sequence.

Table 4-2 Dynamic files

5 API Description

For an interface overview please see Figure 2-1.

5.1 Type Definitions

For the definition of the standard vMem types see [2]. There are no further Low-level specific type definitions.

5.2 Services used by the vMem

Additionally, to the services described within [2], the vMem driver uses the following services provided by other components. For details about prototype and functionality refer to the documentation of the providing component.

Component	API
SPI	Spi_SetupEB
SPI	Spi_AsyncTransmit
SPI	Spi_GetSequenceResult

Table 5-1 Services used by vMem_30_XXspi01

5.3 Callback Functions

This chapter describes the callback functions that are implemented by vMem and can be invoked by other modules. The prototypes of the callback functions are provided in the header file *vMem_30_XXspi01_Cbk.h*.

5.3.1 Communication End Notification

The communication between the external flash device and the vMem is performed asynchronously via SPI. For synchronization of vMem and SPI driver a callback service is generated for each configured *vMemInstance*.

In the configuration of the SPI the header file *vMem_30_XXspi01_Cbk.h* must be included and the callback function needs to be configured as sequence end notification for all SPI sequences referenced by vMem.

Prototype	
`void vMem_30_XXspi01_Cbk_SpiEndNotification_Inst`<*vMemInstanceId*>	
Parameter	
`void`	--
Return code	
`void`	--
Functional Description	
Callback service to notify the vMem about a finished sequence for vMemInstance <*vMemInstanceId*>. The call occurs on completion of each SPI sequence that is used by vMem.	

Particularities and Limitations
> This function is generated for each configured vMemInstance
Call context
> Expected to be called in any context. (Depends on how SPI driver implements sequence-end notification).

Table 5-2 vMem_30_XXspi01_Cbk_SpiEndNotification_Inst<vMemInstanceId>

6 Configuration

The vMem_30_XXspi01 module can be configured through the Vector configuration and generation tool DaVinci Configurator Pro.

6.1 Configuration Variants

The vMem_30_XXspi01 supports the configuration variants

> `VARIANT-PRE-COMPILE`

The configuration classes of the vMem_30_XXspi01 parameters depend on the supported configuration variants. For their definitions please see the vMem_30_XXspi01 _bswmd.arxml file.

6.2 Description files

In general, the configuration effort is reduced to minimum. There are different types of description files – all of them having their own definition

> vMem_30_XXspi01 _bswmd.arxml

General description file for the vMem_30_XXspi01 module.

6.3 SPI Configuration

The vMem driver performs hardware access by calling service functions of the lower layer component SPI driver.

Therefore, the SPI driver must be configured to allow access to the external device. This means that *SpiExternalDevices*, *SpiSequences*, *SpiJobs* and *SpiChannels* must be defined prior using the vMem driver. Figure 6-1 illustrates the SPI configuration structure.

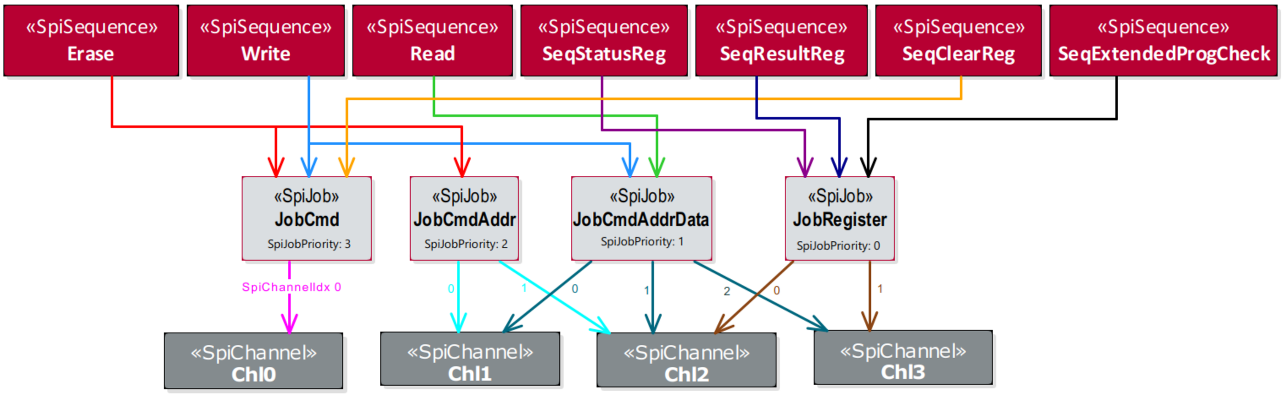

«SpiSequence» Erase
«SpiSequence» Write
«SpiSequence» Read
«SpiSequence» SeqStatusReg
«SpiSequence» SeqResultReg
«SpiSequence» SeqClearReg
«SpiSequence» SeqExtendedProgCheck
«SpiJob» JobCmd SpiJobPriority: 3
«SpiJob» JobCmdAddr SpiJobPriority: 2
«SpiJob» JobCmdAddrData SpiJobPriority: 1
«SpiJob» JobRegister SpiJobPriority: 0
SpiChannelIdx 0
0
1
0
1
2
0
1
«SpiChannel» Chl0
«SpiChannel» Chl1
«SpiChannel» Chl2
«SpiChannel» Chl3

Figure 6-1 Reference structure between SpiSequences, SpiJobs & SpiChannels.

Note
The following description of the SPI-specific configuration refers to a single instance of vMem (represented by vMemInstance container in DaVinci Cfg 5).

In case of a multi device use case a dedicated SPI configuration is required for each vMemInstance.

Only relevant parameters are described.

6.3.1 External Device Configuration

For configuration of the *SpiExternalDevice* refer to the manual of the external flash device.

The configuration described in Table 6-1 is exemplary for the MICRON MT25TL01G and can deviate for other devices/derivatives.

Attribute	Value
SpiCsPolarity	LOW
SpiDataShiftEdge	LEADING
SpiEnableCs	True
SpiShiftClockIdleLevel	LOW

Table 6-1 SPI External device configuration

Note
The configured *SpiExternalDevice* needs to be referenced by SpiJobs (refer to 6.3.3).

6.3.2 Sequence Configuration

Depending on the functionality of the device configured in a specific *vMemInstance* container the vMem driver needs to reference at least four *SpiSequences*.

Each of the referenced sequences need to be configured as described in Table 6-2.

Attribute	Value
SpiJobAssignment	Sequence-specific, refer to Figure 2-1.
SpiSeqEndNotification	vMem_30_XXspi01_Cbk_SpiEndNotification_Inst<vMemInstanceId>

Table 6-2 SPI Sequence configuration

6.3.3 Job Configuration

The configuration requires four *SpiJobs* per *vMemInstance* configuration.

Attribute	Value
SpiDeviceAssignment	*SpiExternalDevice* configured in 6.3.1.
SpiHwUnitSynchronous	ASYNCHRONOUS
SpiJobPriority	Refer to Figure 2-1.
SpiChannelList	Job-specific, refer to Figure 2-1.

Table 6-3 SPI Job configuration

For channel mapping configuration of the *SpiJobs* refer to Figure 2-1.

6.3.4 Channel Configuration

The configuration requires at least four *SpiChannels* per *vMemInstance* configuration.

Attribute	Value
SpiChannelType	EB
SpiDataWidth	8
SpiEbMaxLength	Channel specific[1]
SpiTransferStart	MSB

Table 6-4 SPI Channel configuration

Note
The structure described in Figure 6-1 shows an optimized configuration with minimum required number of *SpiChannels*. It is also possible to assign dedicated channels for each *SpiJob*.

[1] *SpiChannel* with *SpiChannelIndex* 2 of *JobCmdAddrData* (see Figure 6-1) requires value max(*vMemPageSize*, *vMemAccMReadRequestLength*), all other requires value 4 (address length).

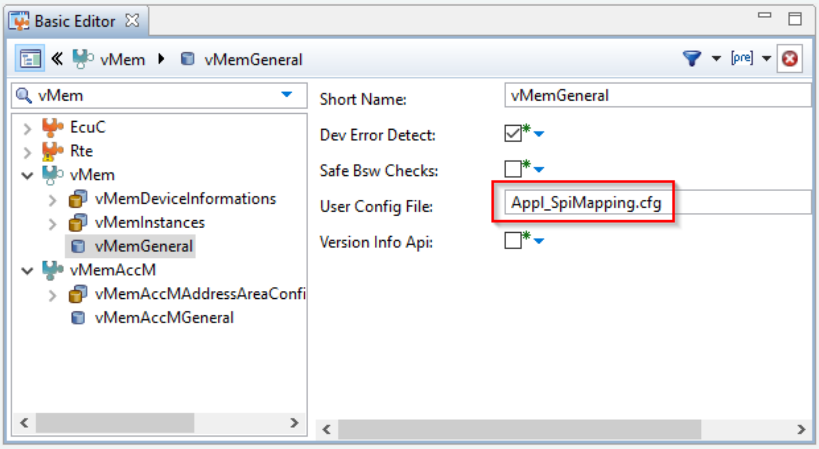
6.3.5 SPI driver compatibility

For compatibility reasons, the vMem driver provides the possibility to configure the interface parameters towards the AUTOSAR SPI driver by inserting corresponding defines into the user configuration file of the vMem driver module.

The following example shows how the vMem driver can be configured to use call a user-defined API instead of AUTOSAR Spi_AsyncTransmit. To get an overview of all remappable interfaces refer to SPI mapping section in vMem_30_XXspi01_Cfg.h.

Example

In case the vMem driver is configured like below, Appl_Spi_AsyncCallout is called instead of Spi_AsyncTransmit:



Short Name: vMemGeneral

Dev Error Detect: ☑

Safe Bsw Checks: □

User Config File: Appl_SpiMapping.cfg

Version Info Api: □

Figure 6-2 vMemUserConfigFile is used to remap SPI interfaces

Content of referenced file *Appl_SpiMapping.cfg*:

```
#define vMem_30_XXspi01_LL_SpiAsyncTransmit Appl_Spi_AsyncCallout
```

Note: The user-defined API needs the identical prototype as the AUTOSAR SPI equivalent.

7 Glossary and Abbreviations

7.1 Glossary

Term	Description
vMemAccM	Vector Memory Access Manager: the most likely user of the vMem driver. The vMemAccM partitions and distributes memory access jobs from multiple users to the various vMem drivers.

Table 7-1 Glossary

7.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
C55FMC	C55 Flash Main Control
DET	Development Error Tracer
EB	External Buffer
ECU	Electronic Control Unit
HIS	Hersteller Initiative Software
HSM	Hardware Security Module
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
MSB	Most significant bit
RTE	Runtime Environment
SPI	Serial Peripheral Interface
SWS	Software Specification

Table 7-2 Abbreviations

8 Contact

Visit our website for more information on

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

www.vector.com