VECTOR

MICROSAR vIpcMemIfCp

Technical Reference

Inter Processor Communication via Shared Memory

Version 1.0.0

Authors	Benjamin Seifert
Status	Released

Document Information

History

Author	Date	Version	Remarks
Benjamin Seifert	2017-08-17	1.0.0	Initial version

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_BasicSoftwareModules.pdf	V1.0.0
[2]	AUTOSAR	AUTOSAR_SWS_DevelopmentErrorTracer.pdf	V3.2.0
[3]	Vector	TechnicalReference_PostBuildLoadable.pdf	see delivery

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 Introduction

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

Supported AUTOSAR Release:	4	
Supported Configuration Variants:	pre-compile, post-build loadable	
Vendor ID:	VIPCMEMIFCP_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	VIPCMEMIFCP _MODULE_ID	255 decimal (according to ref. [1])

The vIpcMemIfCp (Vector interprocessor communication interface for shared memory in classic AUTOSAR) module is the interface between the generic upper layer module vIpc and the hardware dependent lower layer module vIpcMemCp to transfer data between two or more instances (ECUs) via shared memory.

It forwards the transmission requests from the upper layer to the lower module and supports different channel states to be able to repeat unfinished transmissions or receptions.

Multiple channels are supported. Each channel is unidirectional. The channels are fully independent from each other.

1.1 Architecture Overview

The following figure shows where the vIpcMemIfCp is located in the AUTOSAR architecture.

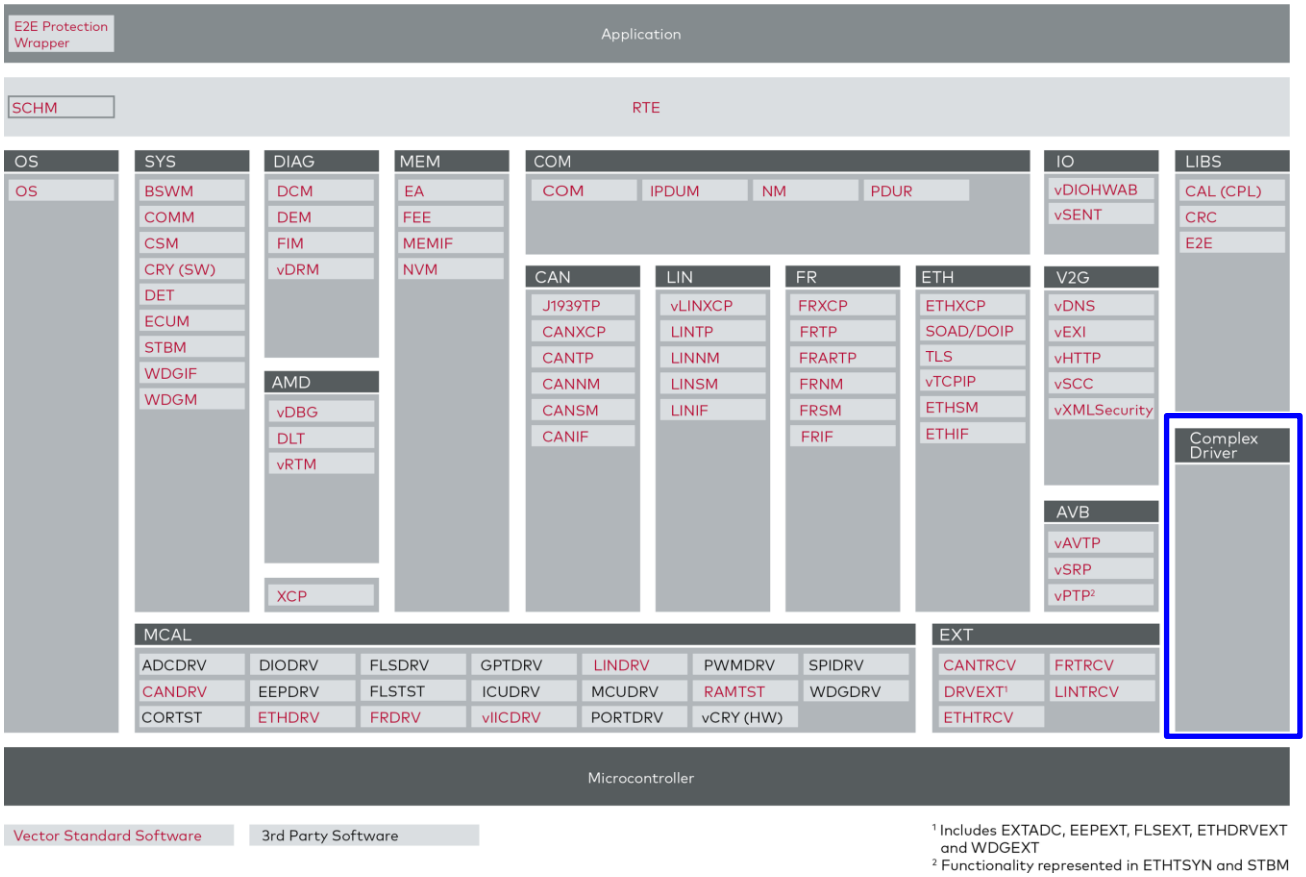

Figure 1-1 AUTOSAR 4.x Architecture Overview

The next figure shows the interfaces to adjacent modules of the vIpcMemIfCp. These interfaces are described in chapter 4.

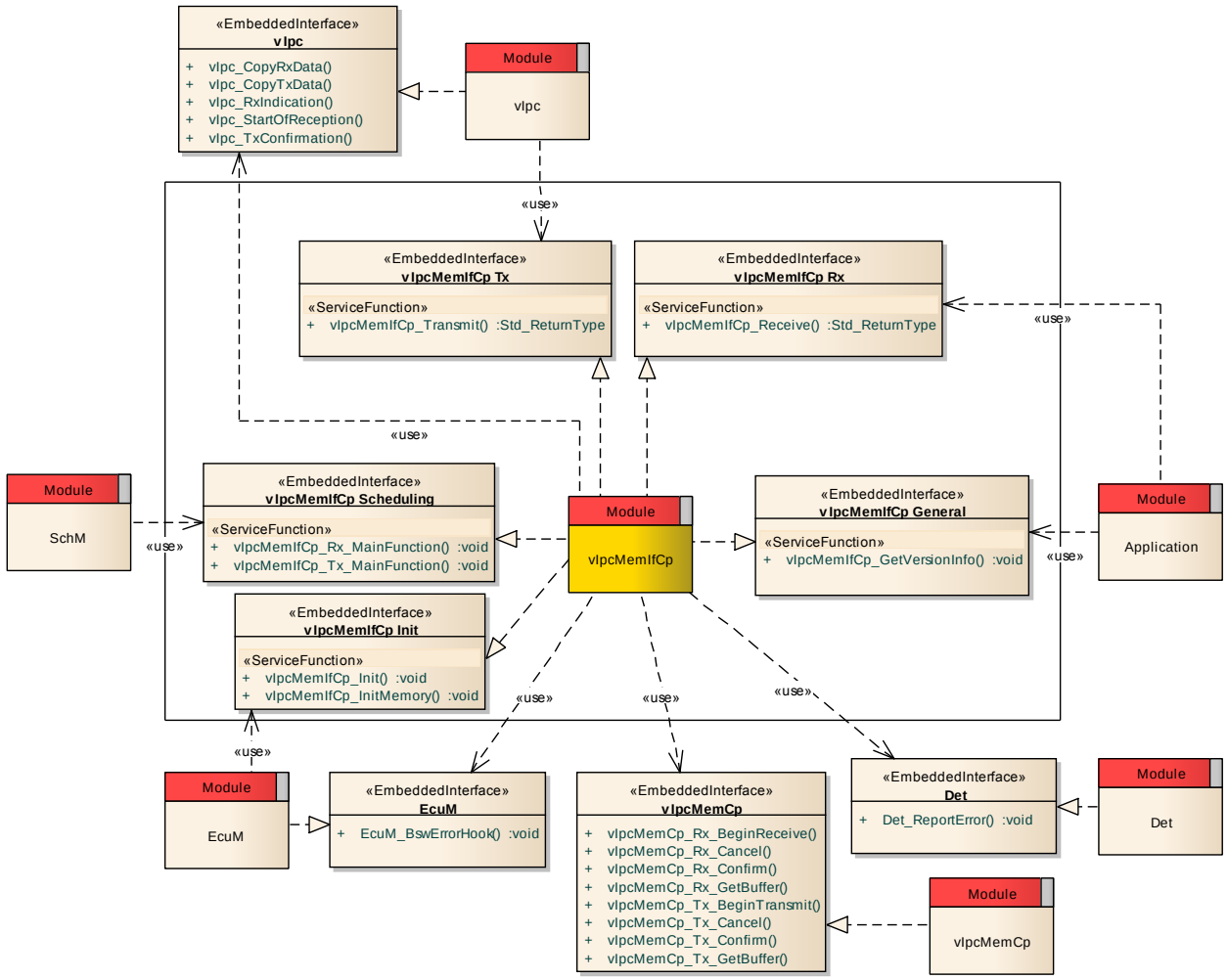

Figure 1-2 Interfaces to adjacent modules of the vIpcMemIfCp

2 Functional Description

2.1 Initialization

The vIpcMemIfCp module is pre-initialized by a call to function `vIpcMemIfCp_InitMemory()`.

The vIpcMemIfCp module is initialized by a call to function `vIpcMemIfCp_Init()`.

Caution
Initialization of the vIpcMemIfCp must not be done before the used lower layer (e.g. vIpcMemCp) is initialized.

2.2 States

The vIpcMemIfCp does not provide any state machines.

2.3 Main Functions

The vIpcMemIfCp provides two main functions which have to be called cyclically. For detailed descriptions of these functions see chapter 4.1.

vIpcMemIfCp_Tx_MainFunction	Main function to handle unfinished transmissions for all channels.
vIpcMemIfCp_Rx_MainFunction	Main function to handle receptions for all channels.

Table 2-1 vIpcMemIfCp main functions

Note
Cyclically calling of the main functions has to be done by the application code.

2.4 Error Handling

2.4.1 Development Error Reporting

By default, development errors are reported to the DET using the service `Det_ReportError()` as specified in [2], if development error reporting is enabled (i.e. pre-compile parameter `vIpcMemIfCp_DEV_ERROR_DETECT==STD_ON`).

If another module is used for development error reporting, the function prototype for reporting the error can be configured by the integrator, but must have the same signature as the service `Det_ReportError()`.

The reported vIpcMemIfCp ID is 255.

The reported service IDs identify the services which are described in chapter 4.1. The following table presents the service IDs and the related services:

Service ID		Service
0x00	VIPCMEMIFCP_SID_INIT	vIpcMemIfCp_Init
0x01	VIPCMEMIFCP_SID_INIT_MEMORY	vIpcMemIfCp_InitMemory
0x02	VIPCMEMIFCP_SID_GET_VERSION_INFO	vIpcMemIfCp_GetVersionInfo
0x03	VIPCMEMIFCP_SID_TRANSMIT	vIpcMemIfCp_Transmit
0x04	VIPCMEMIFCP_SID_RECEIVE	vIpcMemIfCp_Receive

Table 2-2 Service IDs

The errors reported to DET are described in the following table:

Error Code		Description
0x01	VIPCMEMIFCP_E_INV_PTR	Invalid pointer passed to function.
0x02	VIPCMEMIFCP_E_PARAM	Invalid parameter passed to function.
0x03	VIPCMEMIFCP_E_UNINIT	API service used without module initialization.

Table 2-3 Errors reported to DET

2.4.2 Production Code Error Reporting

Production code error reporting is not supported in the current version of the vIpcMemIfCp.

2.5 Transmission (TX)

A transmission to a specific channel can only be started with the API function `vIpcMemIfCp_Transmit()` which is described in chapter 4.1. It has to be called by the application.

In case the transmission cannot be finished within the `vIpcMemIfCp_Transmit()` API, it is repeated in the `vIpcMemIfCp_Tx_MainFunction()` until the data copy process was successful. The TX main function is described in chapter 2.3 and 4.1.

The TX end confirmation is always done in the `vIpcMemIfCp_Tx_MainFunction()`. It also calls the user configurable TX confirmation callback with the current channel ID which is described in chapter 4.3.1.

2.6 Reception (RX)

Receiving data is done within the `vIpcMemIfCp_Rx_MainFunction()`. It checks all configured RX channels for available data and starts a reception if necessary. The RX main function is described in chapter 2.3 and 4.1.

In addition a reception from a specific channel can be triggered with the API function `vIpcMemIfCp_Receive()` which is described in chapter 4.1. It has to be called by the application.

In case the reception cannot be finished it is repeated in the `vIpcMemIfCp_Rx_MainFunction()` until the data copy process was successful.

The RX end confirmation is done right after the data reception from a channel was successful.

3 Integration

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

3.1 Scope of Delivery

The delivery of the vIpcMemIfCp contains the files which are described in the chapters 3.1.1 and 3.1.2:

3.1.1 Static Files

File Name	Description
vIpcMemIfCp.c	Contains the implementation of the external API functions.
vIpcMemIfCp.h	Contains the declaration of the external API functions.
vIpcMemIfCp_Int.h	Contains the declaration and definition of general internal functions.
vIpcMemIfCp_RxInt.h	Contains the declaration and definition of RX internal functions.
vIpcMemIfCp_TxInt.h	Contains the declaration and definition of TX internal functions.
vIpcMemIfCp_Types.h	Contains the type definitions of this module.

Table 3-1 Static files

3.1.2 Dynamic Files

The dynamic files are generated by the configuration tool DaVinci Configurator.

File Name	Description
vIpcMemIfCp_Cfg.c	Contains the pre-compile configuration.
vIpcMemIfCp_Cfg.h	Contains the pre-compile configuration data declarations.
vIpcMemIfCp_PBcfg.c	Contains the post-build configuration.
vIpcMemIfCp_PBcfg.h	Contains the post-build configuration data declarations.

Table 3-2 Generated files

3.2 Critical Sections

The vIpcMemIfCp uses the following exclusive area:

> VIPCMEMIFCP_EXCLUSIVE_AREA_0

It is entered while receiving data from a channel.

4 API Description

For an interfaces overview please see Figure 1-2.

4.1 Services provided by vIpcMemIfCp

4.1.1 vIpcMemIfCp_InitMemory

Prototype	
`void vIpcMemIfCp_InitMemory (void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
Function for *_INIT_*-variable initialization.	
Particularities and Limitations	
Module is uninitialized. Service to initialize module global variables at power up. This function initializes the variables in *_INIT_* sections. Used in case they are not initialized by the startup code.	
Call context	
> ANY > This function is Synchronous > This function is Non-Reentrant	

Table 4-1 vIpcMemIfCp_InitMemory

4.1.2 vIpcMemIfCp_Init

Prototype	
`void vIpcMemIfCp_Init (const vIpcMemIfCp_ConfigType *ConfigPtr)`	
Parameter	
ConfigPtr [in]	Pointer to the PostBuildLoadable configuration.
Return code	
void	none
Functional Description	
Initialization function.	
Particularities and Limitations	
Module is uninitialized. This function initializes the module vIpcMemIfCp. It initializes all intern variables.	
Call context	

> ANY > This function is Synchronous > This function is Non-Reentrant

Table 4-2 vIpcMemIfCp_Init

4.1.3 vIpcMemIfCp_GetVersionInfo

Prototype	
`void vIpcMemIfCp_GetVersionInfo (Std_VersionInfoType *VersionInfo)`	
Parameter	
VersionInfo [out]	Pointer to where to store the version information. Parameter must not be NULL.
Return code	
void	none
Functional Description	
Returns the version information.	
Particularities and Limitations	
This function returns version information, vendor ID and AUTOSAR module ID. Configuration Variant(s): VIPCMEMIFCP_VERSION_INFO_API	
Call context	
> ANY > This function is Synchronous > This function is Reentrant	

Table 4-3 vIpcMemIfCp_GetVersionInfo

4.1.4 vIpcMemIfCp_Transmit

Prototype	
`Std_ReturnType vIpcMemIfCp_Transmit (PduIdType ChannelId, const PduInfoType *PduInfoPtr)`	
Parameter	
ChannelId [in]	Id to identify the channel.
PduInfoPtr [in]	Pdu description.
Return code	
Std_ReturnType	E_OK Transmission request accepted. E_NOT_OK Transmission request failed.
Functional Description	
Requests a new transmission on the channel.	
Particularities and Limitations	
If the channel is idle, the transmission is accepted and processed.	
Call context	

> ANY > This function is Synchronous > This function is Reentrant for different channels

Table 4-4 vIpcMemIfCp_Transmit

4.1.5 vIpcMemIfCp_Receive

Prototype	
`Std_ReturnType vIpcMemIfCp_Receive (PduIdType ChannelId)`	
Parameter	
ChannelId [in]	Id to identify the channel.
Return code	
Std_ReturnType	E_OK Processing request accepted. E_NOT_OK Processing request failed.
Functional Description	
Triggers processing of a specific channel.	
Particularities and Limitations	
-	
Call context	
> ANY > This function is Synchronous > This function is Reentrant for different channels	

Table 4-5 vIpcMemIfCp_Receive

4.1.6 vIpcMemIfCp_Tx_MainFunction

Prototype	
`void vIpcMemIfCp_Tx_MainFunction (void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
Controls the continuous processing of the transmissions.	
Particularities and Limitations	
-	
Call context	
> ANY > This function is Synchronous > This function is Non-Reentrant	

Table 4-6 vIpcMemIfCp_Tx_MainFunction

4.1.7 vIpcMemIfCp_Rx_MainFunction

Prototype	
`void vIpcMemIfCp_Rx_MainFunction (void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
Controls the continuous processing of the receptions.	
Particularities and Limitations	
-	
Call context	
> ANY > This function is Synchronous > This function is Non-Reentrant	

Table 4-7 vIpcMemIfCp_Rx_MainFunction

4.2 Services used by vIpcMemIfCp

In the following table services provided by other modules, which are used by the vIpcMemIfCp are listed. For details about prototype and functionality refer to the documentation of the providing module.

Module	API
Det	`Det_ReportError`
EcuM	`EcuM_BswErrorHook`
SchM	Enter and leave critical section

vIpc	`vIpc_CopyRxData`
	`vIpc_CopyTxData`
	`vIpc_RxIndication`
	`vIpc_StartOfReception`
	`vIpc_TxConfirmation`

vIpcMemCp	`vIpcMemCp_Rx_BeginReceive`
	`vIpcMemCp_Rx_Cancel`
	`vIpcMemCp_Rx_Confirm`
	`vIpcMemCp_Rx_GetBuffer`
	`vIpcMemCp_Tx_BeginTransmit`
	`vIpcMemCp_Tx_Cancel`
	`vIpcMemCp_Tx_Confirm`
	`vIpcMemCp_Tx_GetBuffer`

Table 4-8 Services used by the vIpcMemIfCp

4.3 Configurable Interfaces

4.3.1 User TX notification callback

The vIpcMemIfCp provides the possibility to configure a callback function for the transmission end notification. This function is called at the end of a transmission and can be used to notify the receiver side about a finished transmission. The implementation of this function is application specific and shall match the following signature:

Prototype	
`void [UserTxNotificationCbkName] (PduIdType ChannelId)`	
Parameter	
ChannelId	The upper layer Id to identify the channel.
Return code	
void	none
Functional Description	
User callback function for the transmission end notification.	
Particularities and Limitations	
-	
Call context	
> Called by vIpcMemIfCp at the end of a transmission	

Table 4-9 User TX notification callback

5 Configuration

5.1 Configuration Variants

The vIpcMemIfCp supports the configuration variants

- VARIANT-PRE-COMPILE
- VARIANT-POST-BUILD-LOADABLE

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

5.2 Configuration of Post-Build

The configuration of post-build loadable is described in see [3].

The vIpcMemIfCp uses a static RAM management which differs from the concept described in the mentioned post-build documentation.

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.
Callback function	This is a function provided by the application. E.g. the CAN Driver calls a callback function to allow the application to control some action, to make decisions at runtime and to influence the work of the driver.
Channel	A channel defines the assignment (1:1) between a physical communication interface and a physical layer, on which different modules are connected to. 1 channel consists of 1..X network(s).
Critical section	A critical section is a sequence of instructions where mutual exclusion must be ensured. Such a section is called 'critical' because shared data is modified within it.
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.
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
Post-build	This type of configuration is possible after building the software module or the ECU software. The software may either receive parameters of its configuration during the download of the complete ECU software resulting from the linkage of the code, or it may receive its configuration file that can be downloaded to the ECU separately, avoiding a re-compilation and re-build of the ECU software modules. In order to make the post-build time reconfiguration possible, the reconfigurable parameters shall be stored at a known memory location of ECU storage area.

Table 6-1 Glossary

6.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
DET	Development Error Tracer
ECU	Electronic Control Unit
HIS	Hersteller Initiative Software
IPC	Interprocessor Communication
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
RAM	Random Access Memory
RX	Reception
SWS	Software Specification
TX	Transmission

Table 6-2 Abbreviations

7 Contact

Visit our website for more information on

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

www.vector.com