VECTOR

Freshness Value Manager

Technical Reference

Complex Device Driver

Version 2.0.0

Authors	Florian Röhm Heiko Hübler
Status	Released

Document Information

History

Author	Date	Version	Remarks
Florian Röhm	2017-02-20	1.0.0	Initial creation
Heiko Hübler	2017-02-23	1.0.0	FEATC-1186: Release TMC Security Modules: Freshness Handling (FvM)
Heiko Hübler	2017-08-09	2.0.0	STORYC-2127: Cleanup FvM Backlog

Reference Documents

No.	Source	Title	Version
[1]	Toyota	E/E PF BSW Reference Implementation Requirement Specification	19pfbswrefrs-a00-00-a
[2]	AUTOSAR	AUTOSAR_SWS_DET.pdf	4.0.3
[3]	AUTOSAR	AUTOSAR_BasicSoftwareModules.pdf	V1.0.0

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	> Initial version
1.01.01	> Support of Freshness verify value padding > Rx Freshness value construction pattern according to [1]
1.02.00	> QM Release of the Freshness Value Manager module

Table 1-1 Component history

2 Introduction

This document describes the functionality, API and configuration of the CDD FvM as specified in [1].

Supported AUTOSAR Release*:	4.3	
Supported Configuration Variants:	Pre-compile	
Vendor ID:	FvM_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	FvM_MODULE_ID	255 decimal (according to ref. [3])

* For the detailed functional specification please also refer to the corresponding AUTOSAR SWS.

The Freshness Value Manager stores the freshness values used by the SecOC module.

It is responsible to keep track of the correct incrementation of the counters (Tx path) and the correct reconstruction of the counter (Rx path). The global sync messages are received by the FvM as well and will be stored and used for the generation of the freshness values.

2.1 Architecture Overview

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

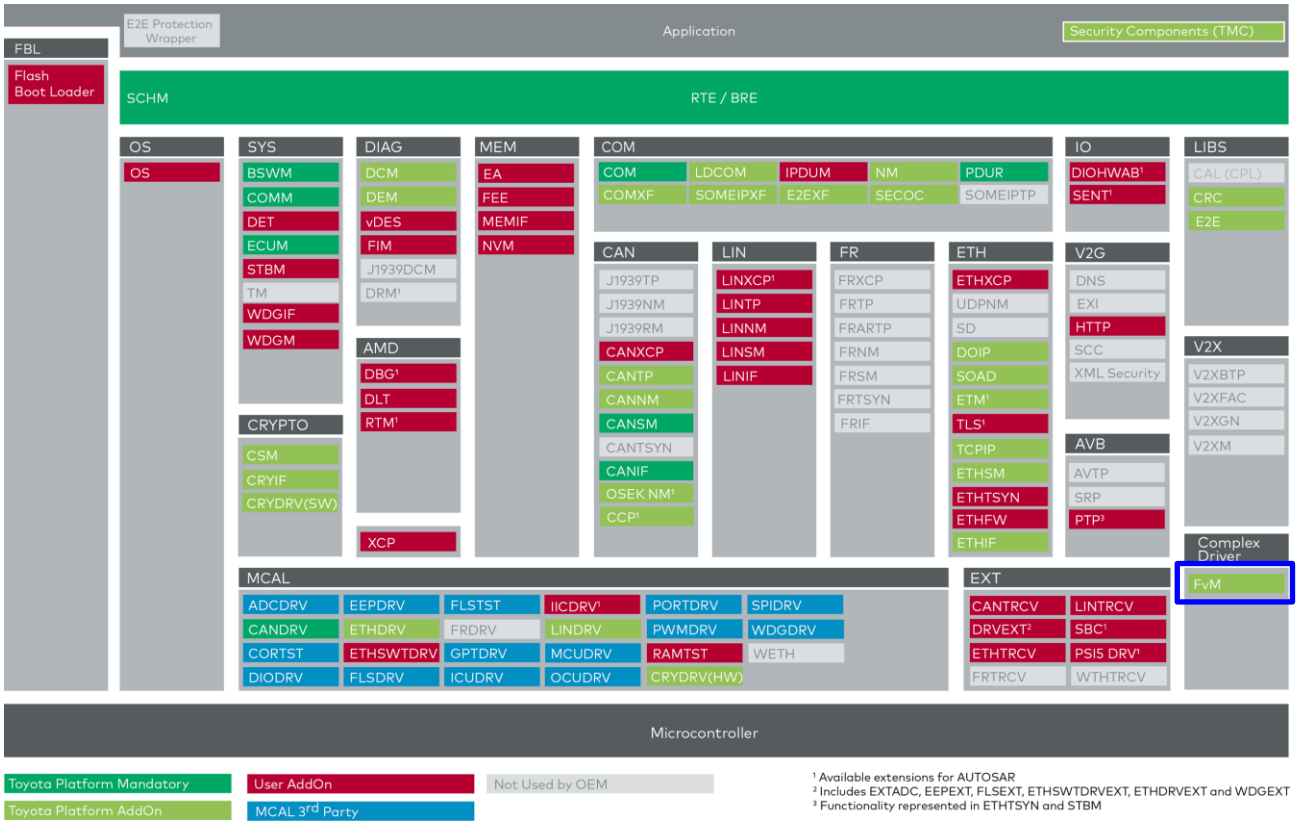

Figure 2-1 AUTOSAR Architecture Overview

3 Functional Description

3.1 Features

The features listed in the following tables cover the complete functionality specified for the FvM.

The functionality is specified in [1], the corresponding features are listed in Table 3-1.

The following features specified in [1] are supported:

Supported Features
> Providing Tx and Rx freshness values for SecOC
> Trip and Reset Counter synchronization

Table 3-1 Supported features

3.2 Initialization

The FvM has to be initialized by calling FvM_Init() before it can be used.

If variables exist which cannot be initialized by the startup code the function FvM_InitMemory() has to be called.

3.3 States

Initially the FvM is in state uninitialized. In this state no API can be used if the parameter checks are activated. The FvM can be brought to state initialized by calling the FvM_Init API. It can be uninitialized again by calling FvM_DeInit.

Furthermore the FvM has a sync state. Figure 3-1 shows the corresponding state machine.

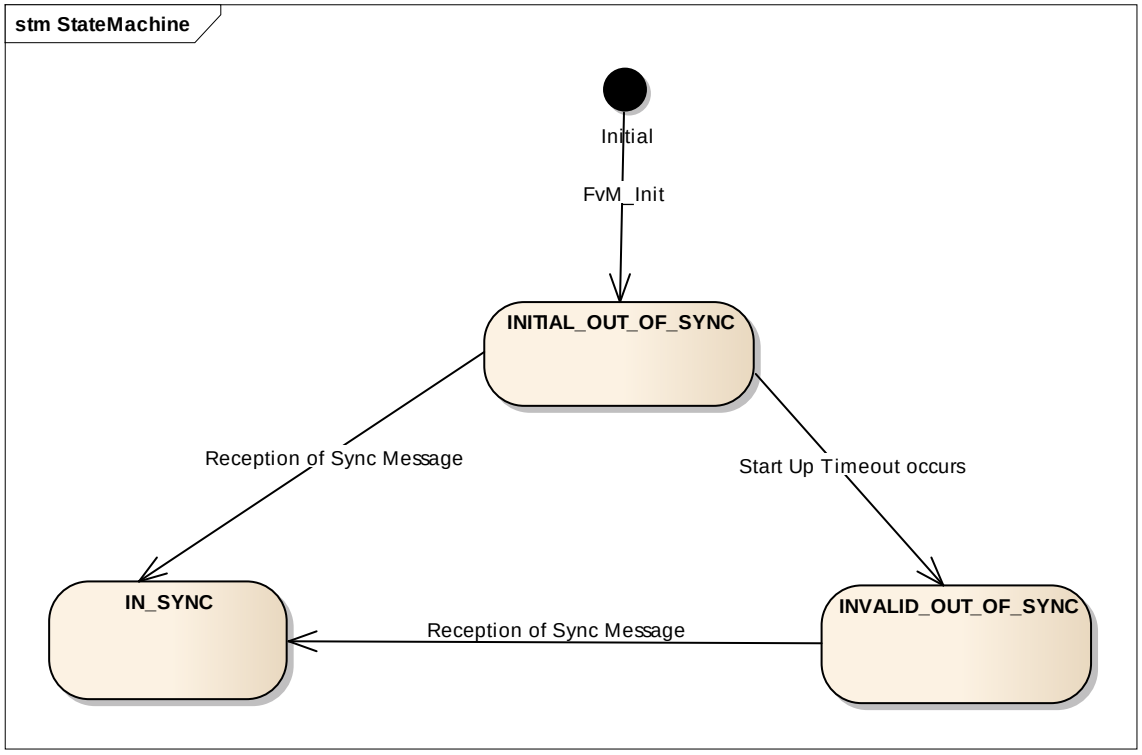

Figure 3-1 Sync Status State machine

3.4 Freshness Value

The freshness verify value is constructed with Trip-Counter, Reset-Counter, Message-Counter and Rest Flag.

Freshness Value Parts:

Trip-Counter	Counter is incremented per driving cycle. This freshness value part is equal for each freshness value.
Reset-Counter	Counter is incremented cyclically by the freshness value master and reset if the Trip-Counter is incremented. This freshness value part is equal for each freshness value.
Message-Counter	Counter is incremented on message transmit of a specific message. This freshness value part exists per message.
Reset Flag	Least significant bits of the Reset-Counter

3.4.1 Freshness Value Synchronization

The Trip-Counter and Reset-Counter are transmitted by the freshness value master in a Sync-Message. The Sync-Message is protected by a MAC against manipulation. The MAC verification is handled by the SecOC.

3.4.2 Start Up Acceptance Time

If the parameter FvMStartUpAcceptanceTime is configured, the FvM will set the SecOC in an “accept all messages” state after start up. This “accept all messages” state of the SecOC is left if the “Start Up Acceptance Time” is over.

3.5 Main Functions

FvM provides following functions listed in Table 3-2 that have to be called cyclically by the Basic Software Scheduler or a similar component.

Main Function	Description
FvM_MainFunction()	This function decrements the Startup Cycle Counter. This function must be called cyclically with a cycle time identical to the configured Main Function Period.

Table 3-2 Main Functions

3.6 Error Handling

3.6.1 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 `FvM_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 FvM ID is 255.

The reported service IDs identify the services which are described in 5.1. The following table presents the service IDs and the related services:

Service ID	Service
FVM_SID_INIT	FvM_Init
FVM_SID_DEINIT	FvM_DeInit
FVM_SID_GET_VERSION_INFO	FvM_GetVersionInfo
FVM_SID_GET_RX_FRESHNESS	FvM_GetRxFreshness
FVM_SID_GET_TX_FRESHNESS	SecOC_GetTxFreshnessTruncData
FVM_SID_TX_FRESHNESS_CONFIRMATION	SecOC_SPduTxConfirmation
FVM_SID_VERIFICATION_STATUS_CALLOUT	FvM_VerificationStatusCallout

Table 3-3 Service IDs

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

Error Code	Description
FVM_E_PARAM_CONFIG	API service FvM_Init() called with wrong parameter.

Error Code	Description
FVM_E_PARAM	API service called with wrong parameter.
FVM_E_PARAM_POINTER	API service used with invalid pointer parameter (NULL).
FVM_E_UNINIT	API service used without module initialization.
FVM_E_ALREADY_INITIALIZED	The service FvM_Init() is called while the module is already initialized.
FVM_E_INVALID_REQUEST	The service FvM_DeInit() is called although the module is already de-initialized.

Table 3-4 Errors reported to DET

4 Integration

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

4.1 Scope of Delivery

The delivery of the FvM contains the files which are described in the chapters 4.1.1 and 4.1.2:

4.1.1 Static Files

File Name	Description
FvM.c	This is the source file of the FvM.
FvM.h	This is the header file of the FvM.

Table 4-1 Static files

4.1.2 Dynamic Files

The dynamic files are generated by the configuration tool Davinci Configurator 5.

File Name	Description
FvM_Cfg.h	This file contains: ▶ global constant macros ▶ global function macros ▶ global data types and structures ▶ global data prototypes ▶ global function prototypes of CONFIG-CLASS PRE-COMPILE data.
FvM_Lcfg.h	This file contains: ▶ global constant macros ▶ global function macros ▶ global data types and structures ▶ global data prototypes ▶ global function prototypes of CONFIG-CLASS LINK data.
FvM_Lcfg.c	This file contains: ▶ local constant macros ▶ local function macros ▶ local data types and structures ▶ local data prototypes ▶ local data ▶ global data of CONFIG-CLASS LINK and PRE-COMPILE data.
FvM_PBcfg.h	This file contains: ▶ global constant macros ▶ global function macros

File Name	Description
	▶ global data types and structures ▶ global data prototypes ▶ global function prototypes of CONFIG-CLASS POST-BUILD data.
FvM_PBcfg.c	This file contains: ▶ local constant macros ▶ local function macros ▶ local data types and structures ▶ local data prototypes ▶ local data ▶ global data of CONFIG-CLASS POST-BUILD data.
FvM_Types.h	This file contains types and defines for the FvM.

Table 4-2 Generated files

4.2 Critical Sections

- FVM_EXCLUSIVE_AREA_RX_FRESHNESS

  This critical section protects RAM variables used for the Rx path.

- FVM_EXCLUSIVE_AREA_TX_FRESHNESS

  This critical section protects RAM variables used for the Tx Path.

5 API Description

5.1 Services provided by FvM

5.1.1 FvM_InitMemory

Prototype	
`void FvM_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	
> TASK > This function is Synchronous > This function is Non-Reentrant	

Table 5-1 FvM_InitMemory

5.1.2 FvM_DeInit

Prototype	
`void FvM_DeInit (void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
DeInitialization function.	
Particularities and Limitations	
Specification of module initialization > Interrupts are disabled. Module is initialized. This function sets the module state to uninitialized.	
Call context	
> TASK	

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

Table 5-2 FvM_DeInit

5.1.3 FvM_Init

Prototype	
`void FvM_Init (const FvM_ConfigType *ConfigPtr)`	
Parameter	
ConfigPtr [in]	Configuration structure for initializing the module
Return code	
void	none
Functional Description	
Initialization function.	
Particularities and Limitations	
Specification of module initialization > Interrupts are disabled. Module is uninitialized. FvM_InitMemory has been called unless FvM is initialized by start-up code. This function initializes the module FvM. It initializes all variables and sets the module state to initialized.	
Call context	
> TASK > This function is Synchronous > This function is Non-Reentrant	

Table 5-3 FvM_Init

5.1.4 FvM_GetVersionInfo

Prototype	
`void FvM_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	
none FvM_GetVersionInfo() returns version information, vendor ID and AUTOSAR module ID of the component.	
Call context	
> TASK|ISR2	

> This function is Synchronous > This function is Reentrant

Table 5-4 FvM_GetVersionInfo

5.1.5 SecOC_GetRxFreshness

Prototype	
`Std_ReturnType SecOC_GetRxFreshness ( uint16 SecOCFreshnessValueID, const uint8 *SecOCTruncatedFreshnessValue, uint32 SecOCTruncatedFreshnessValueLength, uint16 SecOCAuthVerifyAttempts, uint8 *SecOCFreshnessValue, uint32 *SecOCFreshnessValueLength)`	
Parameter	
SecOCFreshnessValueID	Holds the identifier of the freshness value.
SecOCTruncatedFreshnessValue	Holds the truncated freshness value that was contained in the Secured I-PDU.
SecOCTruncatedFreshnessValueLength	Holds the length in bits of the truncated freshness value.
SecOCAuthVerifyAttempts	Holds the number of authentication verify attempts of this PDU since the last reception. The value is 0 for the first attempt and incremented on every unsuccessful verification attempt.
SecOCFreshnessValue	Holds the length in bits of the freshness value.
SecOCFreshnessValueLength	Holds the freshness value to be used for the calculation of the authenticator.
Return code	
Std_ReturnType	E_OK: request successful E_NOT_OK: request failed, a freshness value cannot be provided due to general issues for freshness or this FreshnessValueId. E_BUSY: The freshness information can temporarily not be provided.
Functional Description	
This interface is used by the SecOC to obtain the current freshness value.	
Particularities and Limitations	
> None	
Call context	
> TASK	

Table 5-5 SecOC_GetRxFreshness

5.1.6 SecOC_GetTxFreshnessTruncData

Prototype	
Single Channel	
`Std_ReturnType SecOC_GetTxFreshnessTruncData ( uint16 SecOCFreshnessValueID, uint8* SecOCFreshnessValue, uint32* SecOCFreshnessValueLength, uint8* SecOCTruncatedFreshnessValue, uint32* SecOCTruncatedFreshnessValueLength)`	
Parameter	
SecOCFreshnessValueID	Holds the identifier of the freshness value.
SecOCFreshnessValueLength	Holds the length of the provided freshness in bits.
SecOCTruncatedFreshnessValueLength	Provides the truncated freshness length configured for this freshness. The function may adapt the value if needed or can leave it unchanged if the configured length and provided length is the same.
SecOCFreshnessValue	Holds the current freshness value
SecOCTruncatedFreshnessValue	Holds the truncated freshness to be included into the Secured I-PDU. The parameter is optional.
Return code	
Std_ReturnType	E_OK: request successful E_NOT_OK: request failed, a freshness value cannot be provided due to general issues for freshness or this FreshnessValueId. E_BUSY: The freshness information can temporarily not be provided
Functional Description	
This interface is used by the SecOC to obtain the current freshness value. The interface function provides the truncated freshness transmitted in the secured I-PDU as well.	
Particularities and Limitations	
> none	
Call context	
> TASK	

Table 5-6 SecOC_GetTxFreshnessTruncData

5.1.7 SecOC_SPduTxConfirmation

Prototype	
Single Channel	
void SecOC_SPduTxConfirmation (uint16 SecOCFreshnessValueID)	
Parameter	
SecOCFreshnessValueID	Holds the identifier of the freshness value.
Return code	
void	None
Functional Description	
This interface is used by the SecOC to indicate that the Secured I-PDU has been initiated for transmission.	
Particularities and Limitations	
> none	
Call context	
> TASK | ISR2 > This function is synchronous. > This function is reentrant for different freshness value Ids.	

Table 5-7 SecOC_SPduTxConfirmation

5.2 Services used by FvM

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

Component	API
DET	Det_ReportError

Table 5-8 Services used by the FvM

5.3 Configurable Interfaces

5.3.1 Notification Functions

5.3.1.1 <FvMTripCounterReachMaxNotification>

Prototype	
`void` **`<FvMTripCounterReachMaxNotification>`** `(void)`	
Parameter	
none	none
Return code	
void	none
Functional Description	
Notification function that is called if the max value is reached.	
Particularities and Limitations	
none	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is not Reentrant	

Table 5-9 <FvMTripCounterReachMaxNotification>

5.3.1.2 <FvMResetCounterReachMaxNotification>

Prototype	
`void` **`<FvMResetCounterReachMaxNotification>`** `(void)`	
Parameter	
none	none
Return code	
void	none
Functional Description	
Notification function that is called if the max value is reached.	

Particularities and Limitations
none
Call context
> TASK|ISR2 > This function is Synchronous > This function is not Reentrant

Table 5-10 <FvMResetCounterReachMaxNotification>

5.3.1.3 <FvMRxMessageCounterReachMaxNotification>

Prototype	
`void` **`<FvMRxMessageCounterReachMaxNotification>`** `(uint16 freshnessValueID)`	
Parameter	
freshnessValueID[in]	freshness value id.
Return code	
void	none
Functional Description	
Notification function that is called if the max value is reached.	
Particularities and Limitations	
none	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Reentrant for different freshness value Ids	

Table 5-11 <FvMRxMessageCounterReachMaxNotification>

5.3.1.4 <FvMTxMessageCounterReachMaxNotification>

Prototype	
`void` **`<FvMTxMessageCounterReachMaxNotification>`** `(uint16 freshnessValueID)`	
Parameter	
freshnessValueID[in]	freshness value id.
Return code	
void	none
Functional Description	
Notification function that is called if the max value is reached.	
Particularities and Limitations	
none	
Call context	

> TASK|ISR2 > This function is Synchronous > This function is Reentrant for different freshness value Ids

Table 5-12 <FvMTxMessageCounterReachMaxNotification>

5.3.2 Callout Functions

5.3.2.1 FvM_VerificationStatusCallout

Prototype	
`void FvM_VerificationStatusCallout (SecOC_VerificationStatusType verificationStatus)`	
Parameter	
verificationStatus [in]	Status of the verification and freshness value id.
Return code	
void	none
Functional Description	
Called to indicate the verification status.	
Particularities and Limitations	
none FvM_VerificationStatusCallout() is called by the SecOC to indicate if the verification was successful or failed.	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Reentrant	

Table 5-13 FvM_VerificationStatusCallout

6 Configuration

6.1 Configuration Variants

The FvM supports the configuration variants

> VARIANT-PRE-COMPILE

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

7 Glossary and Abbreviations

7.1 Glossary

Term	Description
EAD	Embedded Architecture Designer; generation tool for MICROSAR components

Table 7-1 Glossary

7.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
CDD	Complex Device Driver
DET	Development Error Tracer
ECU	Electronic Control Unit
ISR	Interrupt Service Routine
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
RTE	Runtime Environment
SRS	Software Requirement Specification
SWC	Software Component
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