MICROSAR Classic ARTI

Technical Reference

Autosar Run-Time Interface for debugging and tracing

Version 1.02.00

Authors	visrk, virleh, visdqk
Status	Released

Document Information

History

Author	Date	Version	Remarks
visrk	2022-06-21	0.01.00	Creation
visrk	2022-06-23	1.00.00	First complete version
virleh	2022-08-22	1.01.00	Adapted BSWMD parameters after BSWMD change
visdqk	2023-07-14	1.02.00	Update to ARTI SWS R22.11. Added support for the ARTI RTE VFB Trace Client.

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	Specification of AUTOSAR Run-Time Interface	R22.11
[2]	AUTOSAR	Specification of Default Error Tracer	see delivery
[3]	AUTOSAR	Specification of Diagnostic Event Manager	see delivery
[4]	AUTOSAR	List of Basic Software Modules	R21.11

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

Illustrations

Tables

1 Introduction

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

Supported Configuration Variants:	pre-compile	
Vendor ID:	ARTI_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	ARTI_MODULE_ID	005 decimal (according to ref. [4])

The AUTOSAR Run-Time Interface (ARTI) is used as an interface between basic software components, like OS or RTE, and tools for debugging and tracing.

1.1 Architecture Overview

The next figure shows the runtime interfaces to adjacent components and tools of the ARTI. These interfaces are described in chapter 4. Mind that the ARTI configuration in the ARTI ECUC file is also used as an interface between other components, like OS or RTE, and the tools for debugging and tracing.

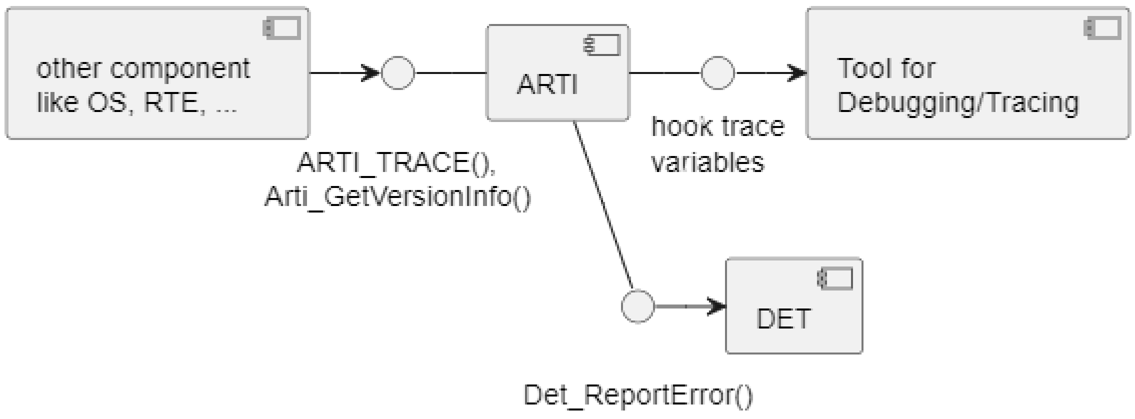
other component like OS, RTE, ...
ARTI
Tool for Debugging/Tracing
ARTI_TRACE(), Arti_GetVersionInfo()
hook trace variables
DET
Det_ReportError()

Figure 1-1 Runtime interfaces to adjacent components and tools of the ARTI

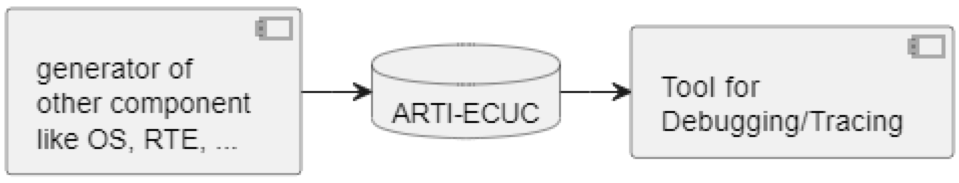
generator of other component like OS, RTE, ...
ARTI-ECUC
Tool for Debugging/Tracing

Figure 1-2 ARTI configuration as an interface between other components and debug/trace-tools

The AUTOSAR standard does not define the interface between ARTI and the debug/trace tools. Additionally, the AUTOSAR standard does also not define which features shall be implemented in ARTI and which features may be implemented in the debug/trace tools.

Therefore, MICROSAR ARTI allows the user to configure the interface to the debug and tracing tools very generically. This allows to run MICROSAR ARTI with many different debug/trace tools on very different hardware.

MICROSAR ARTI expects that all features beyond the pure interface are implemented in the debug/trace tools.

2 Functional Description

The ARTI component is typically expected to be delivered with debug/trace tools by the respective tool vendor. However, MICROSAR ARTI is delivered independently from the tool vendor with the expectation that it works with most debugging and tracing tools, if configured correctly.

The interface to the tools for debugging or tracing was chosen to be a configurable set of variables. As these variables are used to trace the parameter values of hook function calls, we call them hook trace variables in the following. Some debug interfaces need additional trigger actions or require the values to be written to a register. Therefore, the MICROSAR ARTI allows the user to provide a macro that performs these actions.

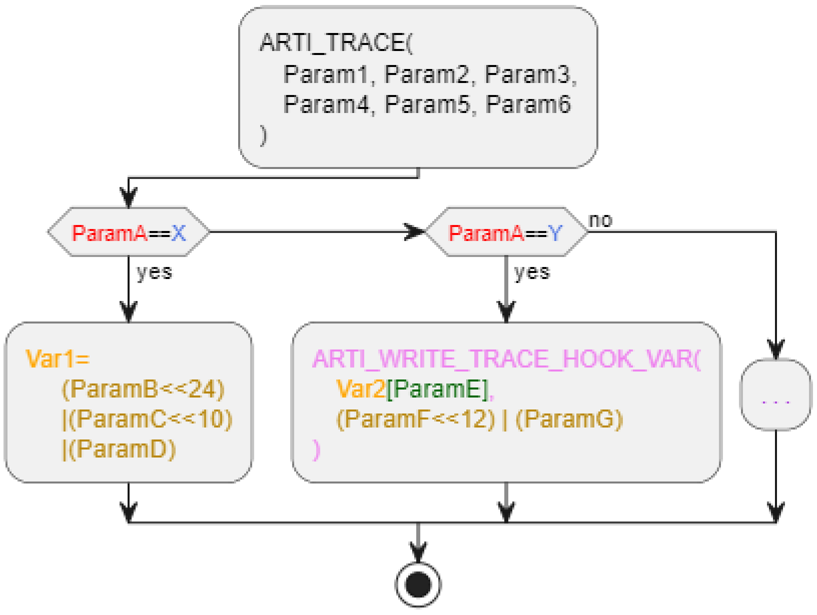

Figure 2-1 Example for functionality of ARTI. Parts in different color can be configured by different types of configuration parameters/containers. The implementation is generated based on the configuration. Parameter names are chosen as Param1 to Param6 and ParamA to ParamG to emphasize that the customer shall define a mapping of Param1-6 to ParamA-G.

So, the user can configure:

- Different hook trace variables as the interface to a debugger or tracing tool.
- Access to these hook trace variables via macro call or direct (the macro `ARTI_WRITE_TRACE_HOOK_VAR()` is to be implemented by the user and may contain additional actions to trigger transmission of the new variable value).
- Conditions for writing to different hook trace variables, based on parameters of `ARTI_TRACE()`.
- Whether or not a hook trace variable is an array variable and which parameter is used as array index.
- Which parameters of `ARTI_TRACE()` shall be stored in a specific hook trace variable and in which bits of the variable.

2.1 Features

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

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 ARTI 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
ARTI-TRACE() define macro interface.
Arti_GetVersionInfo()

Table 2-1 Supported AUTOSAR Standard Conform Features

2.1.1 Deviations

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

Category	Description
Functional	ARTI generic Stopwatch – only the interface to the debug/trace tool is implemented, implementation in the debug/trace tool is expected.
Functional	ARTI generic Datapoint – only the interface to the debug/trace tool is implemented, implementation in the debug/trace tool is expected.
Functional	ARTI Category 1 Interrupts – only the interface to the debug/trace tool is implemented, implementation in the debug/trace tool is expected.
API	Arti_Init()
Functional	ARTI Tracing Macro with Multiple Parameters is not supported (ARTI_TRACE_N).
Functional	ARTI RTE VFB Trace Client - only partial support provided. Deviations: - Trace class AR_CP_RTE_API is not supported. RTE API VFB Trace hooks are generated as empty function-like macros. - Trace class AR_CP_VOID is not supported. Unsupported VFB Trace hooks are generated as empty function-like macros.
Functional	ARTI BSW Module Interface is not supported.
Config	ArtiRteVfbTraceHooks – Deviations: - Subcontainer RteVfbTraceHook/RteVfbTraceHookVendorId (ECUC_Rte_09188) has an optional parameter to configure the vendor ID as integer value.

Category	Description
	- Changed lower multiplicity of reference parameters in the subcontainers of an RteVfbTraceHook to 0 to simplify the configuration of VFB Trace hooks.

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
Interface to the debug/trace tools via hook trace variables.

Table 2-3 Features Provided Beyond the AUTOSAR Standard

2.1.3 Limitations

2.1.3.1 Data Consistency

The implementation of the ARTI_TRACE macro in MICROSAR ARTI does not prevent data consistency problems. The typical usage of MICROSAR ARTI will not need such a data consistency prevention as it results in a single write to a 32 bit variable. This will not result in data consistency problems on 32 bit platforms when the variable is 4 byte aligned.

Only by configuring the access to a hook trace variable via a macro, there may be more than one data write instruction per call of ARTI_TRACE macro. If an additional write instruction is needed in such a variable access macro, data consistency problems may occur due to an interrupted macro (by another ARTI_TRACE macro call on the same core) or by a parallel macro call (by another ARTI_TRACE macro call on another core).

The implementer of the access macro shall consider these data consistency problems.

Data consistency problems due to parallel calls of ARTI_TRACE shall be prevented by writing to core specific memory locations.

Data consistency problems due to interrupted macro calls may only occur if calls of ARTI_TRACE exist, with parameter `_contextName` unequal to `NOSUSP`. In this case, the access macro shall contain code to disable interrupts as described in [1].

Caution
In case, a platform supports unaligned variable access, make sure that at least the hook trace variables of ARTI are linked in an aligned way, in order to prevent data consistency problems.

Caution
In case you implement an access macro with more than one memory accesses, be aware about the potential data consistency problems and perform the counter measures as described in this chapter.

2.2 Initialization

MICROSAR ARTI does not need initialization.

2.3 Main Functions

MICROSAR ARTI has no main function.

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 `ARTI_DEV_ERROR_REPORT==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 ARTI ID is 005.

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

Service ID	Service
0x01	Arti_GetVersionInfo

Table 2-4 Service IDs

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

Error Code	Description
0x01	NULL_PTR detected in Arti_GetVersionInfo

Table 2-5 Errors Reported to DET

2.4.2 Production Code Error Reporting

The functionality provided by MICROSAR ARTI is only relevant at development time of an ECU. Therefore, it does not report production code related errors.

3 Integration

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

3.1 Embedded Implementation

The delivery of the ARTI contains these source code files:

File Name	Description	Integration Tasks
Arti.h	This is the main header file of ARTI. This file may be included by other components and contains (directly or by means of included files) all type declarations, macro definitions and function/variable declarations that might be needed externally in order to use ARTI.	-
Arti.c	This file contains the implementation of all static parts of ARTI.	-
Arti_Cfg.h	This is a generated file which contains preprocessor switches to enable or disable features.	-
Arti_Lcfg.h	This generated file contains variable declarations and the implementation of function like define macros.	-
Arti_Lcfg.c	This generated file contains the definitions of configuration dependent variables.	-
Rte_Hook_Arti.h	This generated file contains the implementation of the Rte Vfb Trace hooks. Implemented as function-like macros.	

Table 3-1 Implementation Files

3.2 Critical Sections

ARTI does not contain critical sections as it has no means to prevent data consistency problems. See 2.1.3.1 for more details.

4 API Description

For an interfaces overview please see Figure 1-1.

4.1 Services Provided by ARTI

4.1.1 Arti_GetVersionInfo

Prototype	
`void Arti_GetVersionInfo (Std_VersionInfoType *VersionInfo )`	
Parameter	
`VersionInfo`	Pointer to a data structure, where this function shall store the version information.
Return code	
--	--
Functional Description	
This function provides the version information of ARTI.	
Particularities and Limitations	
> Service ID: > This function is synchronous and reentrant.	
Call Context	
> Any	

Table 4-1 Arti_GetVersionInfo

4.1.2 ARTI_TRACE

Prototype	
`#define ARTI_TRACE (_contextName, _className, _instanceName, instanceParameter, _eventName, eventParameter)`	
Parameter	
`_contextName`	Token literal text, name of the context. One of the following: • **NOSUSP**: Indicating that the hook gets called in a context where interrupts are disabled. • **SPRVSR**: Indicating that the hook gets called in a context with privileged access rights where interrupts are not disabled. • **USER**: Indicating that the hook gets called in a context with non-privileged access rights where interrupts are not disabled.
`_className`	Token literal text, name of the class of macros, as configured.
`_instanceName`	Token literal text, name of the calling instance, as configured.
`instanceParameter`	A Value in uint32 value range. Meaning of the values shall be configured by means of parameter: `/MICROSAR/Arti/ArtiValues/ArtiHook/ArtiHookInstanceParameterTypeRef`. The possible values for this parameter shall start with 0 and be consecutive per `_instanceName`.

`_eventName`	Token literal test, name of the event that shall be traced, as configured.
`eventParameter`	A Value in uint32 value range as an argument to a specific event. Meaning of the values shall be configured by means of parameter: `/MICROSAR/Arti/ArtiValues/ArtiHook/ArtiHookEventParameterTypeRef`.
Return code	
--	--
Functional Description	
The hook decides on the parameter values, in which hook trace variable the hook call shall be traced. Afterwards, the hook compacts the parameter values as configured for the variable and writes them into that variable.	
Particularities and Limitations	
> Mind that all calls of this hook need to be configured at: /MICROSAR/Arti/ArtiValues/ArtiHook with all used combinations of token literal text parameters (ArtiHookContext=_contextName, ArtiHookClass=_className, ArtiHookInstance=_instanceName, ArtiHookEventName=_eventName). For BSW components which support ARTI, the component generator will typically configure this automatically.	
Call context	
> Any context > Whenever the caller reaches a point in the code, where a state change shall be traced by an external tool.	

Table 4-2 ARTI_TRACE

4.2 Services used by ARTI

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

Component	API
DET	Det_ReportError

Table 4-3 Services Used by the ARTI

4.2.1 Hook Functions

Hook functions are called by ARTI if configured. They must be defined by the user.

4.2.1.1 ARTI_WRITE_TRACE_HOOK_VAR

Prototype	
`#define` **`ARTI_WRITE_TRACE_HOOK_VAR`** `(variable, value)`	
Parameter	
`variable`	Name of the variable, to be written to. In case of an array variable, the parameter includes the array index in "`[]`"

`value`	The value to be written to the variable.
Return code	
`--`	--
Functional Description	
The hook shall write the value to the variable and perform any additional action, which may be needed to successfully trace the variable value change in the connected tool for debugging or tracing.	
Particularities and Limitations	
> --	
Call context	
> Any context > Called from macro ARTI_TRACE(), whenever a hook trace variable shall be written, which is configured to be accessed by macro.	

Table 4-4 ARTI_WRITE_TRACE_HOOK_VAR

5 Configuration

5.1 Hook Trace Variables

ARTI can be configured by means of ECUC-Files with the DaVinci Configurator. However, most of the configuration will typically be generated by the generators of the components that call `ARTI_TRACE()`.

The only thing to be configured is, how the macro `ARTI_TRACE()` (see 4.1.2) shall put the parameter values into hook trace variables. Hook trace variables are the interface to the debug/trace tools.

Hook trace variables may be configured by adding an `ArtiHookTraceVariable` to the configuration object: `/MICROSAR/Arti/`.

Further sub-containers define which parameters of the `ARTI_TRACE()` hook shall be located in the variable and at which bits. Other sub-containers define conditions for the usage of the configured variable and an optional array index.

The parameter `/MICROSAR/Arti/ArtiHookTraceVariable/Arti-HookTraceVariableAccessType` specifies how the macro `ARTI-TRACE()` shall access the configured variable. Typically, the macro simply writes to the variable. However, ARTI can be configured to call the macro `ARTI_WRITE_TRACE_HOOK_VAR()` (see 4.2.1.1) if other actions are required.

Please see the comments in `Arti_bswmd.arxml` for more details about the configuration of ARTI.

5.2 ARTI RTE VFB Trace Client

The ARTI RTE VFB Trace Client can be enabled through the configuration parameter /`MICROSAR/Arti/ArtiGeneral/ArtiEnableVfbTraceClient`. Upon activation, the module will register itself as VFB Trace Client in the RTE configuration.

The module generator automatically creates an `/MICROSAR/ArtiValues/ArtiHook` for each supported VFB Trace Hook, which is configured in `/MICROSAR/Arti/ArtiRte/ArtiVfbTraceHooks`. The module SWS in [1] describes which VFB Trace Hooks are supported by the module. The created ARTI hooks can be mapped on hook trace variables by providing the correct variable conditions in the trace variable configuration. The condition values can be found in the configuration of the created ARTI hooks.

5.3 Configuration Variants

The ARTI supports the configuration variant:

> `VARIANT-PRE-COMPILE`

6 Glossary and Abbreviations

6.1 Abbreviations

Abbreviation	Description
API	Application Programming Interface
ARTI	Autosar Run-Time Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
DEM	Diagnostic Event Manager
DET	Development Error Tracer
ECU	Electronic Control Unit
ISR	Interrupt Service Routine
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
PPORT	Provide Port
RPORT	Require Port
RTE	Runtime Environment
SWC	Software Component
SWS	Software Specification

Table 6-1 Abbreviations

7 Contact

Visit our website for more information on

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

www.vector.com