MICROSAR IOHWAB

Technical Reference

Version 3.00.02

Authors	Christian Leder
Status	Released

Document Information

History

Author	Date	Version	Remarks
Christian Marchl	2007-02-09	1.00.00	Initial version
Christian Marchl	2007-08-09	1.01.00	Typos corrected; Added description for component name field
Christian Marchl	2007-12-13	1.01.01	Version adapted according to new version scheme
Christoph Ederer	2008-05-21	2.00.00	Transfer of the document to new Technical Reference template; Adapted descriptions and screenshots to new software version
Christoph Ederer	2008-07-11	2.00.01	Update of document due to changes in DCM interface and RTE usage
Christoph Ederer	2009-01-14	2.01.00	Update of the naming of graphical elements in the configuration, Screenshots reworked, DCM subfunctions reworked, Added description of default value in configuration
Christoph Ederer	2009-03-23	2.01.01	Updated development error detection in GUI description, toolchain naming updated, hints added to chapter 4.1.2
Christoph Ederer	2009-07-21	2.02.00	Updated description of the generation process (user blocks, autom. SWC generation), updated AUTOSAR figure, added information on user defined signals
Christoph Ederer	2009-09-25	2.02.01	Reworked description of DCM interface
Christoph Ederer	2010-11-26	2.02.02	> Added chapter 4.3 Critical Sections > GUI description updated > Added information about necessary make process modifications to 4.1.2
Christoph Ederer	2011-03-02	2.02.03	Reworked the service descriptions in chapters 5.4.3, 5.4.7 and 5.4.10: parameter 'signal' is [inout], now
Christoph Ederer	2013-04-10	3.00.00	> Component rework and update to AUTOSAR 4 > Update to new Configuration Tooling 'DaVinci Configurator 5'
Christian Leder	2014-03-14	3.00.01	Generated file adapted from IoHwAb.c to IoHwAb_30.c
Christian Leder	2014-06-18	3.00.02	ReturnType changed in CS port operation

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SWS_IOHardwareAbstraction.pdf (available at: http://www.autosar.org/download/R4.0/AUTOSAR_SWS_IOHardwareAbstraction.pdf)	V3.2.0
[2]	AUTOSAR	AUTOSAR_SWS_DevelopmentErrorTracer.pdf (available at: http://www.autosar.org/download/R4.0/AUTOSAR_SWS_DevelopmentErrorTracer.pdf)	V3.2.0
[3]	AUTOSAR	AUTOSAR_TR_BSWModuleList.pdf (available at: http://www.autosar.org/download/R4.0/AUTOSAR_TR_BSWModuleList.pdf)	V1.6.0
[4]	AUTOSAR	AUTOSAR_EXP_VFB.pdf (available at: http://www.autosar.org/download/R4.0/AUTOSAR_EXP_VFB.pdf)	V2.2.0

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
3.00.00	> Update to AUTOSAR 4 > Full support of Client/Server port functionality > Full support of Sender/Receiver port functionality > Full BSW Schedulable Entity support > Full Runnable Entity Support > Update of configuration tooling to DaVinci Configurator 5.x

Table 1-1 Component history

2 Introduction

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

Supported AUTOSAR Release*:	4	
Supported Configuration Variants:	pre-compile	
Vendor ID:	IOHWAB_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	IOHWAB_MODULE_ID	254 decimal (according to ref. [3])

* For the precise AUTOSAR Release 3.x please see the release specific documentation.

The aim of the IOHWAB is to provide ECU hardware-independent data transition from driver modules up to the Software Components. To fulfill this task the IOHWAB generates a Software Component description from the user configuration. This Software Component description can be imported by the 'Vector DaVinci Developer' and/or the RTE module.

After import, the IOHWAB is displayed like a normal Software Component (SW-C) with a set of

> Client/Server Ports,

> Sender/Receiver Ports,

> Runnable Entities

within the 'DaVinci Developer' modeling tool. Other SW-Cs can access I/O modules via these ports through the RTE and the I/O Hardware Abstraction.

This release of the IOHWAB is a completely generic approach and does not generate ready-to-use code files. Instead, it generates function stubs (containing user code sections, called 'User Blocks' – see 7.1, 0) that are accessible via the supported AUTOSAR interfaces: Sender/Receiver Ports, Client/Server Ports, Runnable entities.

2.1 Architecture Overview

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

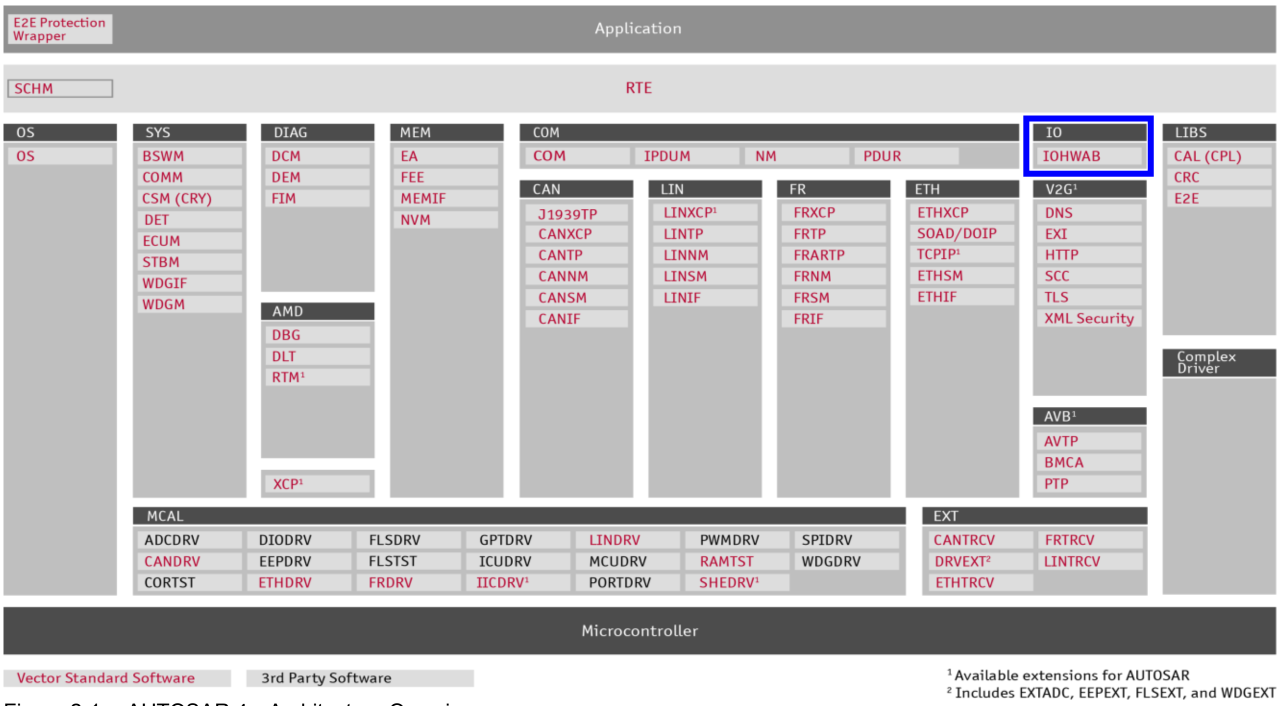


Figure 2-1 AUTOSAR 4.x Architecture Overview

The next figure shows the interfaces to adjacent modules of the IOHWAB. These interfaces are described in chapter 5.

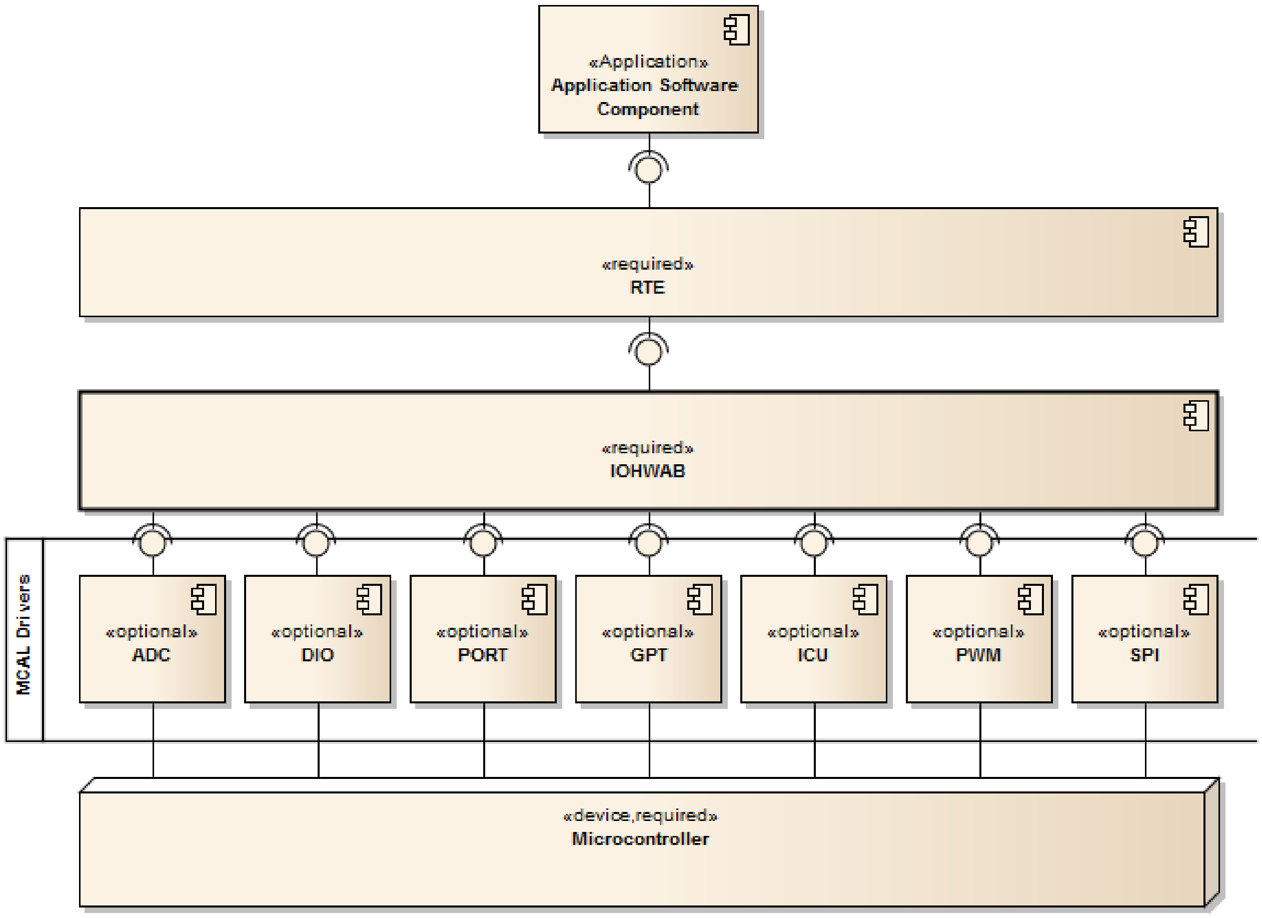


Figure 2-2 Interfaces to adjacent modules of the IOHWAB

Applications do not access the services of the BSW modules directly. They use the service ports provided by the BSW modules via the RTE. The service ports provided by the IOHWAB are listed in chapter 5.7 and are defined in [1].

3 Functional Description

3.1 Features

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

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

> Table 3-1 Supported AUTOSAR standard conform features

> Table 3-2 Not supported AUTOSAR standard conform features

The following features specified in [1] are supported:

Supported AUTOSAR Standard Conform Features
Abstraction of MCAL signals
Creation of a Software Component Description file that can be imported by DaVinci Developer

Table 3-1 Supported AUTOSAR standard conform features

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

Not Supported AUTOSAR Standard Conform Features
Automatic abstraction of I/O modules (Implementation of the ‘firmware’ has to be done by the user)
I/O control via DCM
Support of BswInterruptEntities
Automatic signal aging and debouncing
Production Error Detection
AUTOSAR Debugging

Table 3-2 Not supported AUTOSAR standard conform features

3.2 Initialization

Initialization is not necessarily needed for the IOHWAB. Therefore, the provided service for initialization can be removed by a pre-compile option during the configuration phase.

Basic Knowledge
It is not intended to initialize drivers in the IOHWAB initialization function. Driver initialization has to be done by the ECUM.

3.3 States

The IOHWAB itself neither has certain module states nor does any handling of states of lower-level drivers. Caring about time flows and states is the user’s responsibility.

3.4 Main Functions

The I/O Hardware Abstraction module does not provide (a) fix main function(s), but implements a generic approach, i.e. main functions can be created in the configuration and are generated into the configuration source files.

The I/O Hardware Abstraction refers to main functions as 'BSW Schedulable Entities', see 5.3.4. After creating a BSW Schedulable Entity in the configuration, the configured information is passed to the RTE/SCHM via 'Internal Behavior'. After the generation is finished, the Schedulable Entity functionality can be implemented in the function template in the file 'IoHwAb_30.c'. At runtime, it will be called cyclically by the SCHM as configured.

3.5 Error Handling

3.5.1 Development Error Reporting

By default, development errors are reported to the DET using the service `Det_ReportError()` as specified in [2], if 'Development Mode' and 'Development Error Reporting' are enabled in the configuration tool.

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 IOHWAB ID is 254.

The reported service IDs identify the services which are described in 5.2. The following table presents the service IDs and the related services:

Service ID	Service
0x01	IoHwAb_Init()
0x10	IoHwAb_GetVersionInfo()

Table 3-3 Service IDs

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

Error Code	Description
IOHWAB_E_INV_POINTER	API service called with "NULL pointer" as parameter

Table 3-4 Errors reported to DET

3.5.2 Production Code Error Reporting

As production error reporting is typically done on driver level, the IOHWAB does not provide any reporting mechanism. Nonetheless, production error reporting can be added manually to 'IoHwAb_30.c', if necessary.

4 Integration

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

4.1 Scope of Delivery

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

4.1.1 Static Files

File Name	Description
IoHwAb.h	Declares the interface of the MICROSAR component IOHWAB

Table 4-1 Static files

4.1.2 Dynamic Files

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

File Name	Description
IoHwAb_30.c	Code template that contains the interfaces of the IOHWAB as well as the generated API function templates.
IoHwAb_Cfg.h	Contains the static configuration of the IOHWAB
IoHwAb_Cbk.h	Code template for prototypes of callback functions
IoHwAb_Types.h	Contains all the data types that are used in the IOHWAB (File is only used, if RTE usage is deactivated)

Table 4-2 Generated files

Edit
The files ‘IoHwAb_30.c’ and ‘IoHwAb_Cbk.h’ are code templates, i.e. after generation they are not complete and require user modifications.

Edit
Generated code templates are not part of the AUTOSAR make files provided by the module, because the file may be copied to different locations in an integration package (e.g. a common folder for all editable code templates).

Please add all generated code templates to your make process, manually. If the Vector make environment is used, add the file to the variable ‘APP_SOURCE_LST’ in the file ‘Makefile.project.part.defines’.

For successfully integrating the IOHWAB in a MICROSAR environment, there are further files needed. These Files can be generated by the RTE generator.

File Name	Description
Rte_IoHwAb.h	Contains the prototypes of all the IOHWAB operation services
Rte_Type.h	Contains all the data types that are used in the IOHWAB

Table 4-3 Files generated by the RTE Generator

4.2 Critical Sections

The IOHWAB implements the following critical section:

> IOHWAB_EXCLUSIVE_AREA_0: This critical section can be used to protect code passages that may contain coherent operations. The critical section shall prevent task switches.

Edit
This critical section can be used in the user code. To create a protected code passage, call the macro IoHwAb_GlobalSuspend() before the protected part and IoHwAb_GlobalRestore() after it.

Caution
Be sure to use critical sections scarcely and keep them as short as possible, because they may affect the system performance. Furthermore, pay attention that you do not nest critical sections and always close opened sections.

4.3 Generated Template Files

A generated template file in this document is a file which:

> is generated by the generation tool at every generation process

> the user can modify for his needs

> preserves that the user implementation will not be overwritten during the next generation process

4.3.1 User Blocks

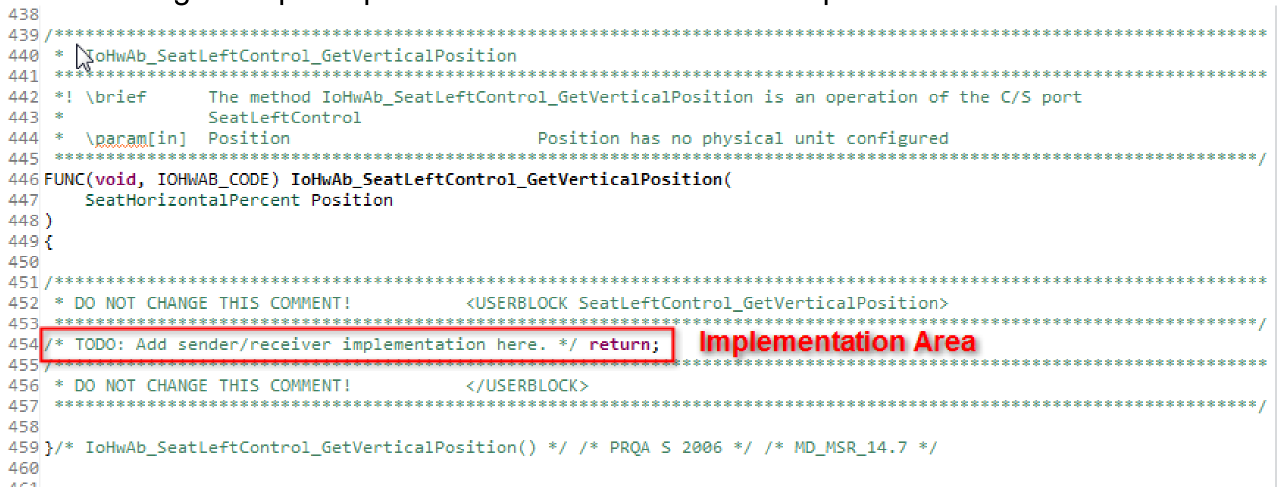
In order not to overwrite the user code, the template file contains special comments, where code can be inserted. The comments have the following format:

```
/**********************************************************************************
 * DO NOT CHANGE THIS COMMENT! <USERBLOCK NAME> DO NOT CHANGE THIS COMMENT
**********************************************************************************/

/**********************************************************************************
 * DO NOT CHANGE THIS COMMENT! </USERBLOCK> DO NOT CHANGE THIS COMMENT
**********************************************************************************/
```

Do not edit manually
Do not modify or delete the comment lines!

The following example explains where the code can be implemented:



```

/**********************************************************************************************
 * IoHwAb_SeatLeftControl_GetVerticalPosition
 **********************************************************************************************
 *! \brief      The method IoHwAb_SeatLeftControl_GetVerticalPosition is an operation of the C/S port
 *             SeatLeftControl
 *  \param[in]  Position                          Position has no physical unit configured
 *********************************************************************************************/
FUNC(void, IOHWAB_CODE) IoHwAb_SeatLeftControl_GetVerticalPosition(
    SeatHorizontalPercent Position
)
{

/**********************************************************************************************
 * DO NOT CHANGE THIS COMMENT!           <USERBLOCK SeatLeftControl_GetVerticalPosition>
 *********************************************************************************************/
/* TODO: Add sender/receiver implementation here. */ return;
/**********************************************************************************************
 * DO NOT CHANGE THIS COMMENT!           </USERBLOCK>
 *********************************************************************************************/

}/* IoHwAb_SeatLeftControl_GetVerticalPosition() */ /* PRQA S 2006 */ /* MD_MSR_14.7 */

```

Figure 4-1 User Block Implementation Area

Every code outside this area will be overwritten during the next generation process.

The I/O Hardware Abstraction provides the following generated template files:

> IoHwAb_30.c

> IoHwAb_Cbk.h

4.3.2 Unrecognized/Deleted Signals

If a Port Prototype, Runnable Entity or BSW Schedulable Entity is removed from or renamed in the configuration, possibly already implemented code would be lost, because the generated service would be deleted from the generated file. In this case, the 'IoHwAb_30.c' template file contains a section at the end, which receives the code from all services whose associated signals have been deleted:

```
/* Section for deleted/unrecognized user blocks: */
#if 0
#endif
```

Caution
This section only contains code from signals that have been deleted during the last generation process. During the next generation, the section will be cleared.

5 API Description

The API of the I/O Hardware Abstraction is mainly generic, only IoHwAb_Init and IoHwAb_GetVersionInfo are permanent. The fixed API is described in the chapter 5.2, the generic, configuration-specific API is described in 5.3.

5.1 Type Definitions

The I/O Hardware Abstraction module does not provide any default types, but implements a generic approach, i.e. types can be created in the configuration and are generated into the configuration source files (either by the I/O Hardware Abstraction itself or by the RTE).

The current implementation supports the creation of the following types:

> Typedefs to primitive types (provided by AUTOSAR Standard/Platform Types)

> Arrays (consisting of primitive types or types that are derived from these)

> Structs ((consisting of primitive types or types that are derived from these)

Types can be created/configured via the following configuration containers:

> …/IoHwAbDatatypes/IoHwAbImplementationTypes/IoHwAbTypeReference

> …/IoHwAbDatatypes/IoHwAbImplementationTypes/IoHwAbArray

> …/IoHwAbDatatypes/IoHwAbImplementationTypes/IoHwAbRecord

The destination file of the typedefs after the generation process depends on the following configuration switch: '/MICROSAR/IoHwAb/IoHwAbGeneral/IoHwAbUseRte'.

If the RTE is used, the file 'Rte_Type.h' will contain the typedefs. Otherwise, the typedefs are generated into the file 'IoHwAb_Types.h'.

5.2 Services provided by IOHWAB

5.2.1 IoHwAb_Init

Prototype	
`void` **`IoHwAb_Init`** `( void )`	
Parameter	
--	--
Return code	
`void`	--
Functional Description	
This function stub can be used to implement any initialization activities and shall be called by the ECUM. It is not intended to initialize drivers in the IOHWAB initialization function. Driver initialization has to be done by the ECUM.	

Particularities and Limitations
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant. > This service can be disabled using the configuration parameter '/MICROSAR/IoHwAb/IoHwAbGeneral/IoHwAbUseInitFunction'
Expected Caller Context
> Expected to be called in application context (EcuM)

Table 5-1 IoHwAb_Init

5.2.2 IoHwAb_GetVersionInfo

Prototype	
`void IoHwAb_GetVersionInfo ( Std_VersionInfoType *versioninfo )`	
Parameter	
`versioninfo`	Pointer to where the version information shall be stored
Return code	
`void`	--
Functional Description	
This service returns the version information of this module. The provided information is: > Module ID > Vendor ID > Version Information	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant. > This service can be disabled using the configuration parameter '/MICROSAR/IoHwAb/IoHwAbGeneral/IoHwAbVersionInfoApi'	
Expected Caller Context	
> Expected to be called in application context	

Table 5-2 IoHwAb_GetVersionInfo

5.3 Generated API Function Templates

The following service descriptions do not refer to certain API functions, but to function stubs that will be generated by the IOHWAB. The service name, as well as the parameter format is influenced by the configuration settings.

Caution
The following functions can be invoked concurrently by the RTE. As software on the lower layer may not support concurrency, the application should avoid parallel calls of signal services.

5.3.1 Client/Server Operation: IoHwAb_<CSPortPrototypeName>_<OperationName>

Prototype	
`Std_ReturnType` **`IoHwAb_<CSPortPrototypeName>_<OperationName>`** `( <ConfiguredType> <*><OperationArgumentName> )`	
Parameter	
`<OperationArgumentName>`	Configured operation parameter
Return code	
`Std_ReturnType`	ReturnType can be configured / defined in the configuration phase.
Functional Description	
This service is the implementation of an operation contained in a configured Client/Server Port Prototype.	
Particularities and Limitations	
> This function is synchronous. > This function is non-reentrant.	
Expected Caller Context	
> Expected to be called in application context (RTE)	

Table 5-3 IoHwAb_<CSPortPrototypeName>_<OperationName>

Practical Procedure

A Client/Server Operation can be created by the following steps:

> Create a Client/Server Interface by adding an instance of the configuration container '/MICROSAR/IoHwAb/IoHwAbCSPortInterface'

> Create the corresponding operations by adding instances of the configuration container '/MICROSAR/IoHwAb/IoHwAbCSPortInterface/IoHwAbOperation' and add operation arguments ('…/IoHwAbOperation/IoHwAbOperationArgument')

> Create a Client/Server PortPrototype by adding an instance of the configuration container '/MICROSAR/IoHwAb/IoHwAbCSPortPrototype' and set the contained parameter 'IoHwAbCSPortInterfaceRef' to the Port Interface created before

Edit

This function is a template that contains no implementation code. Add the implementation in the file 'IoHwAb_30.c'. More Information about where to implement your code can be found in 4.3.1.

5.3.2 Sender/Receiver Port Runnable: IoHwAb_<SRPortPrototypeName>_<Read/Write>

Prototype	
`void` **`IoHwAb_<SRPortPrototypeName>_<Read/Write>`** `( void )`	
Parameter	
--	--
Return code	
`void`	--
Functional Description	
This service is the implementation of a port element contained in a configured Sender/Receiver Port Prototype.	
Particularities and Limitations	
> This function is synchronous. > This function is non-reentrant.	
Expected Caller Context	
> Expected to be called in application context (RTE)	

Table 5-4 IoHwAb_<SRPortPrototypeName>_<Read/Write>

Practical Procedure
A Sender/Receiver port element can be created by the following steps:

> Create a Sender/Receiver Interface by adding an instance of the configuration container '/MICROSAR/IoHwAb/IoHwAbSRPortInterface'

> Create the corresponding port elements by adding instances of the configuration container '/MICROSAR/IoHwAb/IoHwAbSRPortInterface/IoHwAbPortElement'

> Create a Sender/Receiver PortPrototype by adding an instance of the configuration container '/MICROSAR/IoHwAb/IoHwAbSRPortPrototype' and set the contained parameter 'IoHwAbSRPortInterfaceRef' to the Port Interface created before

Edit
This function is a template that contains no implementation code. Add the implementation in the file 'IoHwAb_30.c'. More Information about where to implement your code can be found in 4.3.1.

5.3.3 Runnable Entity: IoHwAb_< RunnableName>

Prototype	
`void` **`IoHwAb_<RunnableName>`** `( )`	
Parameter	
--	--
Return code	
void	--
Functional Description	
This service is the implementation of a Runnable Entity.	
Particularities and Limitations	
> This function is synchronous. > This function is non-reentrant.	

Expected Caller Context
> Expected to be called in application context (RTE)

Table 5-5 IoHwAb_<PortPrototypeName>_<Read/Write>

Practical Procedure

A Runnable Entity can be created by the following steps:

> Create a Runnable Entity by adding an instance of the configuration container '/MICROSAR/IoHwAb/IoHwAbRunnable' and configure the contained parameters.

Edit

This function is a template that contains no implementation code. Add the implementation in the file 'IoHwAb_30.c'. More Information about where to implement your code can be found in 4.3.1.

5.3.4 Schedulable Entity: IoHwAb_<SchedulableName>

Prototype	
`void **IoHwAb_<SchedulableName>** ( void )`	
Parameter	
--	--
Return code	
void	--
Functional Description	
This function is the implementation of a BSW Schedulable Entity.	
Particularities and Limitations	
> This function is synchronous. > This function is non-reentrant.	

Expected Caller Context
> Expected to be called in application context (RTE)

Table 5-6 [Service name]

Practical Procedure
A BSW Schedulable Entity can be created by the following steps:
Create a Schedulable Entity by adding an instance of the configuration container '/MICROSAR/IoHwAb/IoHwAbSchedulable' and configure the contained parameters.

Edit
This function is a template that contains no implementation code. Add the implementation in the file 'IoHwAb_30.c'. More Information about where to implement your code can be found in 4.3.1.

5.4 Services used by IOHWAB

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

Component	API
DET	Det_ReportError (optional)
DIO	All (optional)
GPT	All (optional)
ICU	All (optional)
PORT	All (optional)
PWM	All (optional)
SPI	All (optional)

Table 5-7 Services used by the IOHWAB

5.5 Callback Functions

This release of the IOHWAB does not implement any callback functions. If necessary, the user may implement prototypes of callback functions in the file IoHwAb_Cbk.h.

Note
It is not possible to pass an approaching callback function through the RTE to the software component. Actions for handling callbacks have to be implemented inside the IOHWAB.

5.6 Configurable Interfaces

5.6.1 Notifications

The I/O Hardware Abstraction does not provide Notifications.

5.7 Service Ports

5.7.1 Client Server Interface

A client server interface is related to a Provide Port at the server side and a Require Port at client side.

5.7.1.1 Provide Ports on IOHWAB Side

All the Provide Ports of the IOHWAB the API functions described in 5.2 are available as Runnable Entities. The Runnable Entities are invoked via Operations. The mapping from a SWC client call to an Operation is performed by the RTE. In this mapping the RTE adds Port Defined Argument Values to the client call of the SWC, if configured.

The following sub-chapters present the Provide Ports defined for the IOHWAB and the Operations defined for the Provide Ports, the API functions related to the Operations and the Port Defined Argument Values to be added by the RTE.

5.7.1.1.1 Provide Ports

Operation	API Function	Port Defined Argument Values
<IoHwAbOperation>	IoHwAb_<IoHwAbCSPortPrototype>_<IoHwAbOperation>	<*><IoHwAbOperationArgument>

Table 5-8 Provide Ports

5.7.1.2 Require Ports on IOHWAB Side

The I/O Hardware Abstraction does not provide any Require Ports.

6 Configuration

In the IOHWAB the attributes can be configured with the following tools:

> Configuration in DaVinci Configurator

> Configuration in DaVinci Developer

6.1 Configuration Variants

The IOHWAB supports the configuration variants

> `VARIANT-PRE-COMPILE`

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

6.2 Configuration Workflow

This chapter describes the workflow that is necessary to integrate the I/O Hardware Abstraction into a MICROSAR system.

The I/O Hardware Abstraction is not a classic BSW module, as it has not only interfaces with other BSW modules, but an interface with one or more Application Components above the RTE. For this reason, the IOHWAB has to define a way to communicate with Application Components through the RTE.

The basic concepts for such interaction are described in [4]. This chapter gives only a description of the technical realization in the MICROSAR tool chain.

The following diagram gives an overview of the interaction between the involved tools and the user.

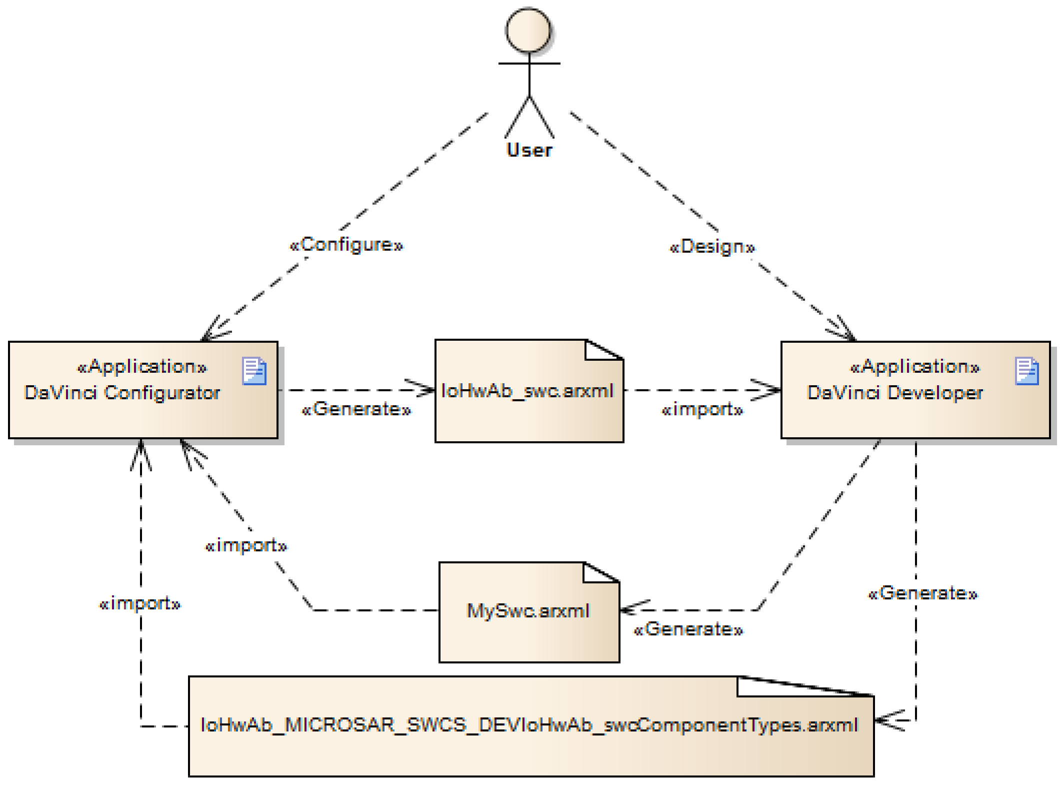


Figure 6-1 Configuration Workflow

For a successful integration of the I/O Hardware Abstraction into a MICROSAR system please execute the following steps:

- Configure the 'IoHwAb' module in the DaVinci Configurator and save the Configuration. The Software Component description of the I/O Hardware Abstraction is generated automatically upon saving and can be found in the SIP folder ('<SIP Folder>\internal\<Application Folder>\Config\ServiceComponents').

▶ Open the DaVinci Developer project or create a new one and import the I/O Hardware Abstraction's SWC file as follows:

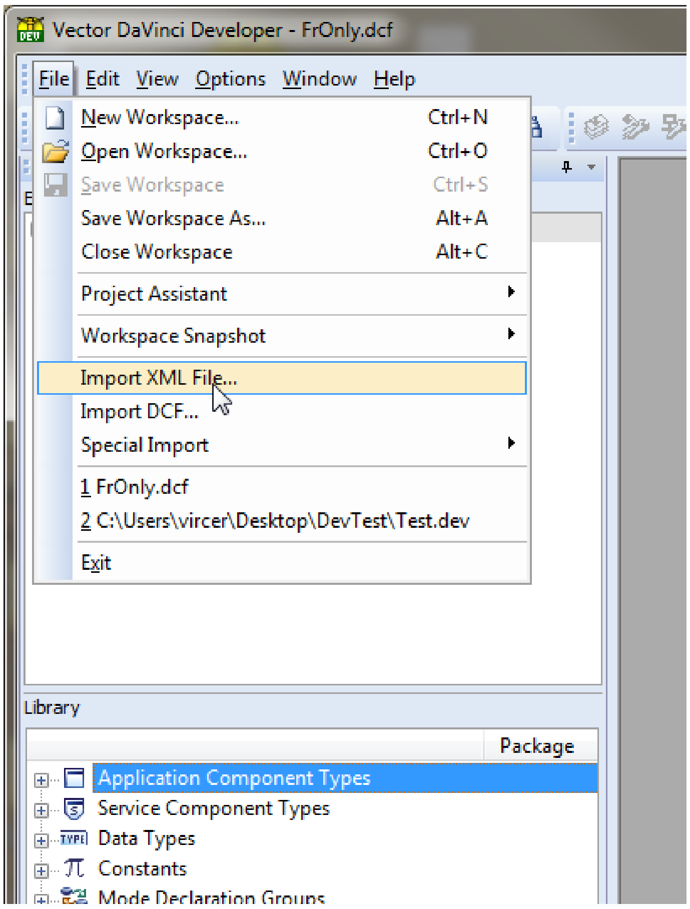


Figure 6-2 SWC import select file

▶ Select the SWC file and start the import. After the import is finished, the module with all configured ports, interfaces and data types should be present in your project:

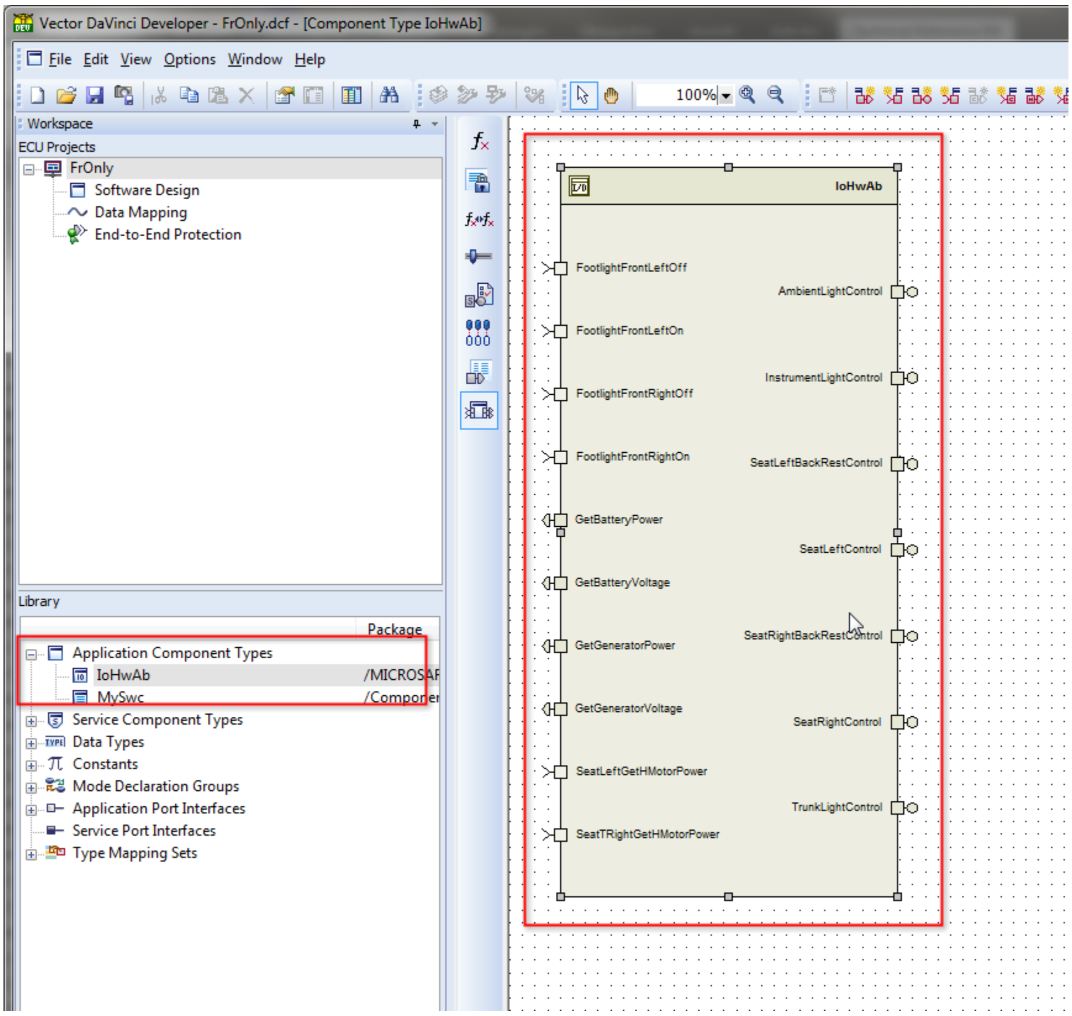


Figure 6-3 Imported Software Component

▶ Create a new application software component and create the counterparts of the ports that are provided/required by the I/O Hardware Abstraction component type.

Caution
Do not connect the SWC ports in the DaVinci Developer, because no more configuration of the RTE inside DaVinci Configurator would be possible, then.

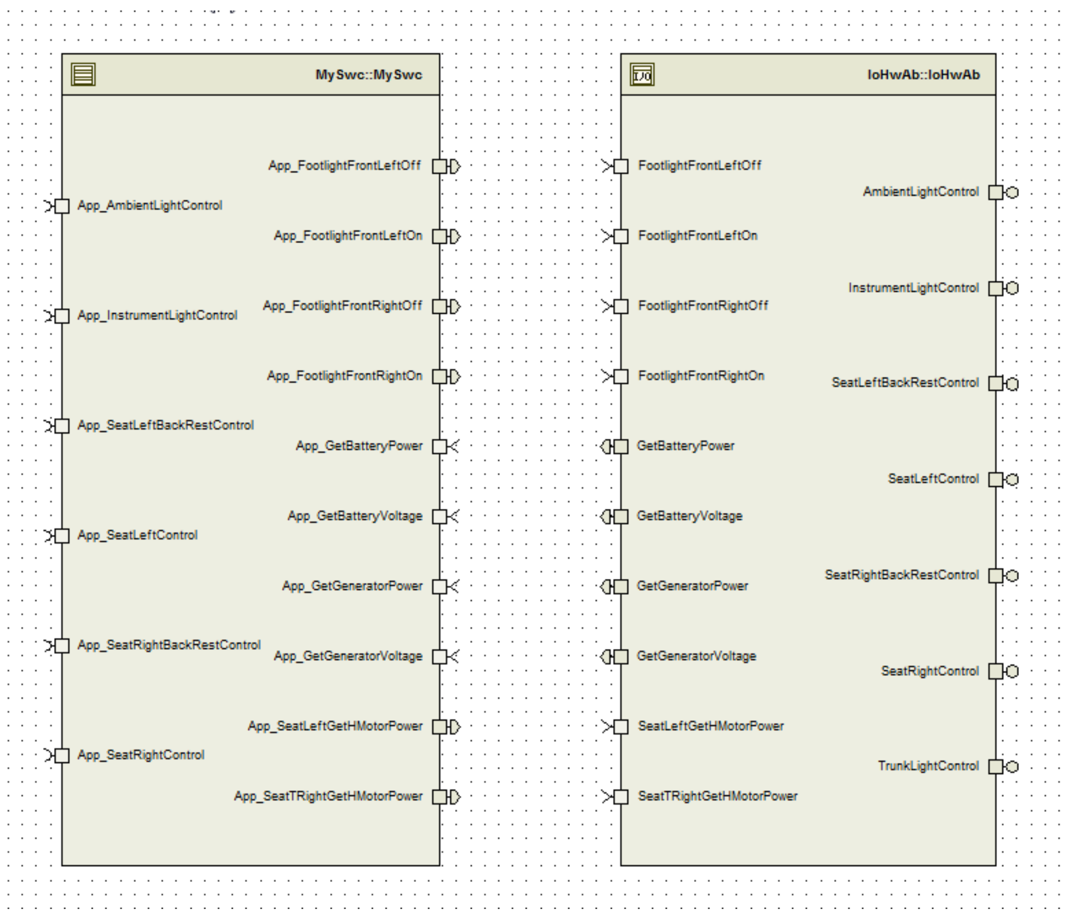


Figure 6-4 I/O Hardware and Application Software Component

- Save and close the DaVinci Developer project.
- Switch back to the DaVinci Configurator and configure the RTE module by using the 'Runtime System' domain editor.
- First, create the Software Components of I/O Hardware Abstraction and Application inside the RTE by using the 'ECU Software Components Editor'

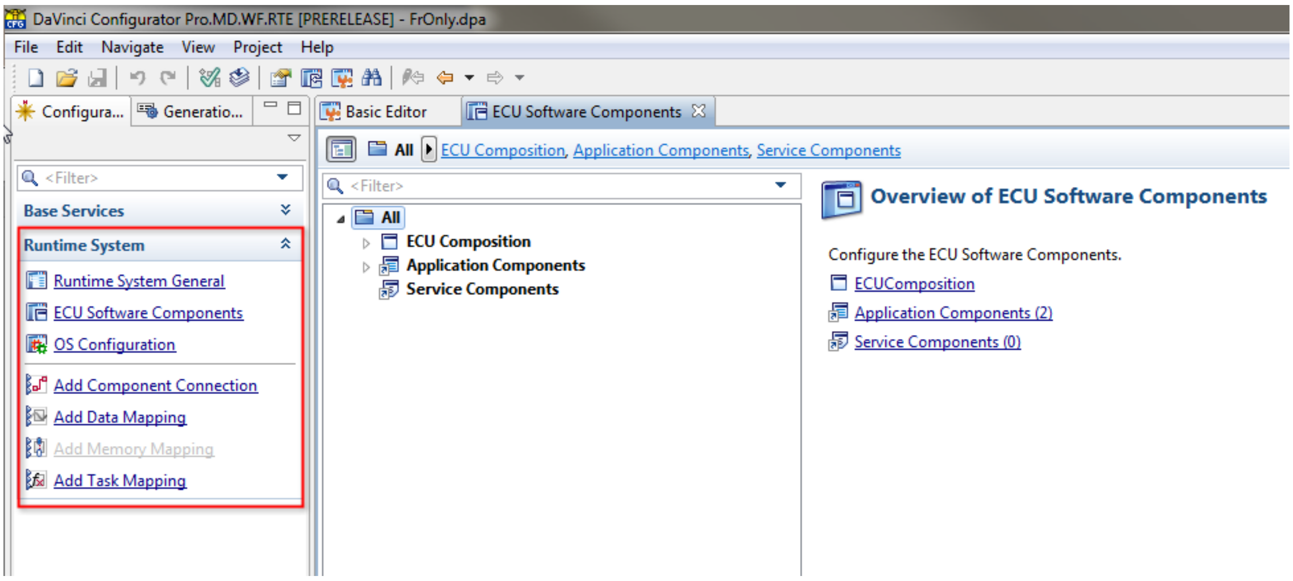
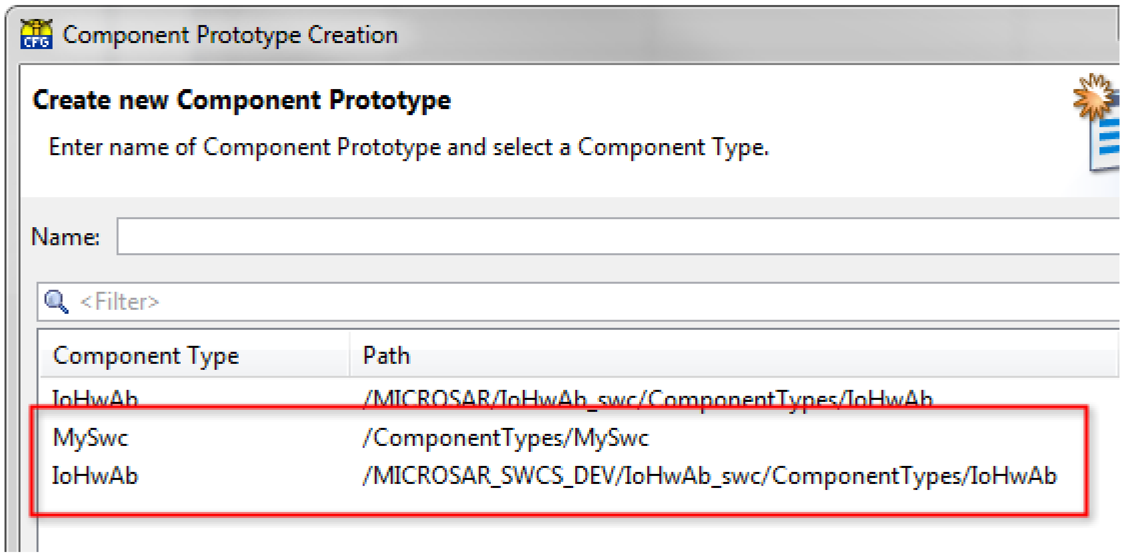


Figure 6-5

- Go to 'Application Components' container and add two Software Components, the 'IoHwAb' and the Software Component(s) created in the DaVinci Developer.


Component Prototype Creation
Create new Component Prototype
Enter name of Component Prototype and select a Component Type.
Name:
<Filter>

Component Type	Path
IoHwAb	/MICROSAR/IoHwAb_swc/ComponentTypes/IoHwAb
MySwc	/ComponentTypes/MySwc
IoHwAb	/MICROSAR_SWCS_DEV/IoHwAb_swc/ComponentTypes/IoHwAb

Caution
The I/O Hardware Abstraction SWC description exists twice in this selection. Please use the SWC description with the path '/MICROSAR_SWCS_DEV/IoHwAb_swc/ComponentTypes/IoHwAb'. This is the SWC description created by the DaVinci Developer. Otherwise, all modifications to the Software Component that have been done in the DaVinci Developer would not be adapted.

- The next step is connecting the ports of the components with the 'Add Component Connection' editor. Start the editor and select I/O Hardware Abstraction and

Application Software Component and map the ports as needed. If the ports of the both components can be connected 1:1, the ‘Automatic Mapping’ can be used.

- Last, the Runnable Entities of both components have to mapped to OS Tasks. This can be done by using the ‘Add Task Mapping’ editor.
- Validate and generate the configuration.

Caution
If you have to change the configuration of the I/O Hardware Abstraction after the SWC has been imported into the DaVinci Developer workspace, the SWC has to be imported again, every time it is changed, because the DaVinci Developer will not adapt the changes automatically.

Caution
If you have to change the DaVinci Developer project after the Software Components have been created in the RTE configuration, the DaVinci Configurator project has to be closed and re-opened and the Application Components in the ‘ECU Software Components’ editor have to be removed and re-added, because the DaVinci Configurator will not adapt the changes automatically.

7 Glossary and Abbreviations

7.1 Glossary

Term	Description
DaVinci Configurator	Configuration tool for MICROSAR components
User Block	Especially marked, user editable code section within a generated C-Source file – these code sections are not overwritten when the module configuration is being re-generated

Table 7-1 Glossary

7.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
DEM	Diagnostic Event Manager
DET	Development Error Tracer
ECU	Electronic Control Unit
HIS	Hersteller Initiative Software
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
PPORT	Provide Port
RPORT	Require Port
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