Port 3.0 driver user guide

TRAVEO™ T2G family

About this document

Scope and purpose

This user guide describes the architecture, configuration, and use of the PORT driver. This guide also explains the functionality of the driver and provides a reference to the driver's API.

The installation, build process, and general information about the use of the EB tresos Studio are not within the scope of this document. See the *EB tresos Studio for ACG8 user's guide* [7] for a detailed description of these topics.

Intended audience

This document is intended for anyone who uses the PORT driver of the TRAVEO™ T2G family.

Document structure

Chapter 1 General overview provides a brief introduction to the PORT driver, explains the embedding of the driver in the AUTOSAR environment, and describes the supported hardware and development environment.

Chapter 2 Using the PORT driver details the steps required to use the PORT driver in your application.

Chapter 3 Structure and dependencies describes the file structure and the dependencies for the PORT driver.

Chapter 4 EB tresos Studio configuration interface describes the driver's configuration with the EB tresos Studio.

Chapter 5 Functional description provides a functional description of all services offered by the PORT driver.

Chapter 6 Hardware resources describes the hardware resources used.

The Appendix provides a complete API reference and access register table.

Abbreviations and definitions

Abbreviations	Description
API	Application Programming Interface
ASIL	Automotive Safety Integrity Level
AUTOSAR	Automotive Open System Architecture
BSW	Basic Software. Standardized part of software which does not fulfill a vehicle functional job.
DEM	Diagnostic Event Manager
DET	Default Error Tracer
EB tresos Studio	Elektrobit Automotive configuration framework
HSIOM	High-Speed I/O Matrix
HW	Hardware

About this document

Abbreviations	Description
MCAL	Microcontroller Abstraction Layer
MCU	Microcontroller Unit
OS	Operating System
Smart I/O	Programmable I/O
SW	Software
UTF-8	8-Bit Universal Character Set Transformation Format
µC	Microcontroller

Related documents

AUTOSAR requirements and specifications

Bibliography

[1] General specification of basic software modules, AUTOSAR release 4.2.2

[2] Specification of PORT driver, AUTOSAR release 4.2.2

[3] Specification of standard types, AUTOSAR release 4.2.2

[4] Specification of ECU configuration parameters, AUTOSAR release 4.2.2

[5] Specification of default error tracer, AUTOSAR release 4.2.2

[6] Specification of memory mapping, AUTOSAR release 4.2.2

Elektrobit automotive documentation

Bibliography

[7] EB tresos Studio for ACG8 user's guide

Hardware documentation

The hardware documents are listed in the delivery notes.

Related standards and norms

Bibliography

[8] Layered software architecture, AUTOSAR release 4.2.2

Table of contents

Table of contents

Table of contents

1 General overview

1.1 Introduction to the PORT driver

The PORT driver is a set of software routines for initializing the whole port structure of the microcontroller.

Many port pins can be assigned to various functionalities such as the following:

- GPIO
- ADC
- CAN
- ICU
- PWM
- SPI

For this purpose, the PORT driver provides configuration options for each port pin for the following:

- Mode
- Direction
- Level value
- Direction changeable flag
- Mode changeable flag

The driver conforms to the AUTOSAR standard and is implemented according to the *specification of PORT driver* [2].

In addition, the PORT driver is delivered with a plugin for the EB tresos Studio, which allows the user to statically configure the driver. It provides an interface to define symbolic names and functionality of all port pins used in GPIO or alternative mode.

The PORT driver provides software routines for initializing the high-speed I/O matrix (HSIOM), the programmable I/O (SMART I/O), and the trigger multiplexers. The trigger multiplexers control the communication between peripherals.

For this purpose, the PORT driver provides configuration options for the following:

- HSIOM setting
- SMART I/O setting
- Trigger group
- Trigger 1-to-1 group

1.2 User profile

This guide is intended for users with a basic knowledge of the following:

- Embedded systems
- The C programming language
- The AUTOSAR standard
- The target hardware architecture

1.3 Embedding in the AUTOSAR environment

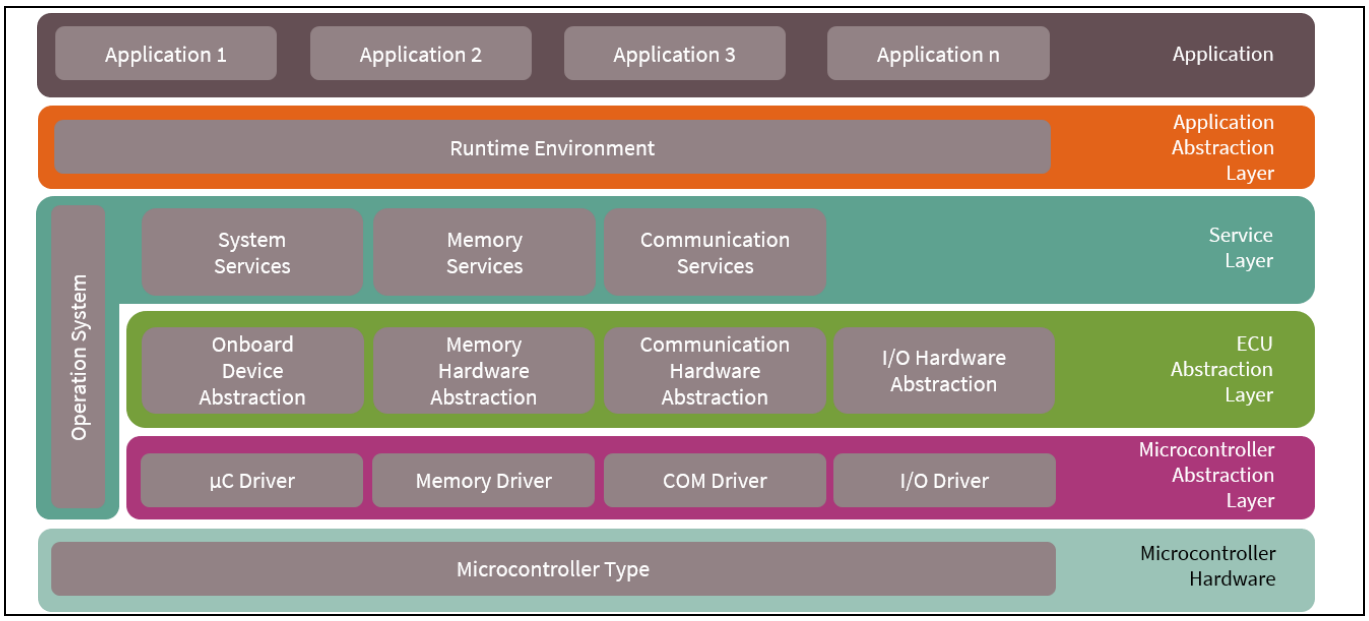

Figure 1 Overview of AUTOSAR software layers

Figure 1 depicts the layered AUTOSAR software architecture. The PORT driver (Figure 2) is part of the microcontroller abstraction layer (MCAL), the lowest layer of the basic software in the AUTOSAR environment.

As an internal I/O driver, it provides a standardized and microcontroller-independent interface to higher software layers for accessing ports and port pins of the ECU hardware.

For a thorough overview of the AUTOSAR-layered software architecture, refer to *Layered software architecture* [8].

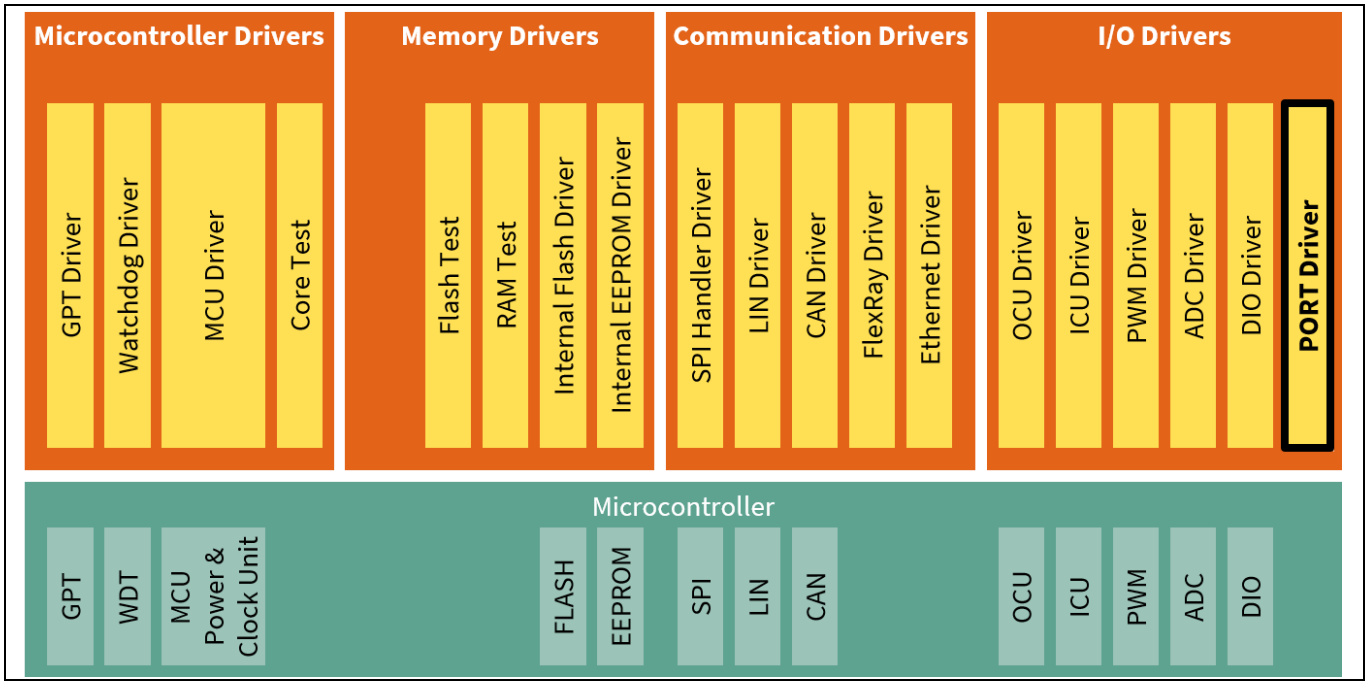

Figure 2 PORT driver in MCAL layer

1.4 Supported hardware

This version of the PORT driver supports the TRAVEO™ T2G microcontroller. The supported derivatives are listed in the release notes.

Smaller derivatives have only a subset of the available port pins. Additional derivatives can be implemented or supported via a resource file without change of the PORT driver and/or this document. Refer to the Resource plugin for a list of all supported derivatives.

1.5 Development environment

The development environment corresponds to AUTOSAR release 4.2.2. The modules BASE, Make, and Resource are needed for proper functionality of the PORT driver.

1.6 Character set and encoding

All source code files of the PORT driver are restricted to the ASCII character set. The files are encoded in UTF-8 format, with only the 7-bit subset (values 0x00 … 0x7F) being used.

1.7 Multicore support

The PORT driver supports the multicore type I. The multicore type III can also be supported for some APIs (for example, read-only API or atomic-write API). For each multicore type, see following sections.

1.7.1 Multicore type

In this section, type I, type II, and type III are defined as multicore characteristics.

1.7.1.1 Single core only (multicore type I)

For this multicore type, the driver is available only on a single core. This type is referred as multicore type I.

Following is the characteristic of multicore type I:

- The peripheral channels are accessed by only one core.

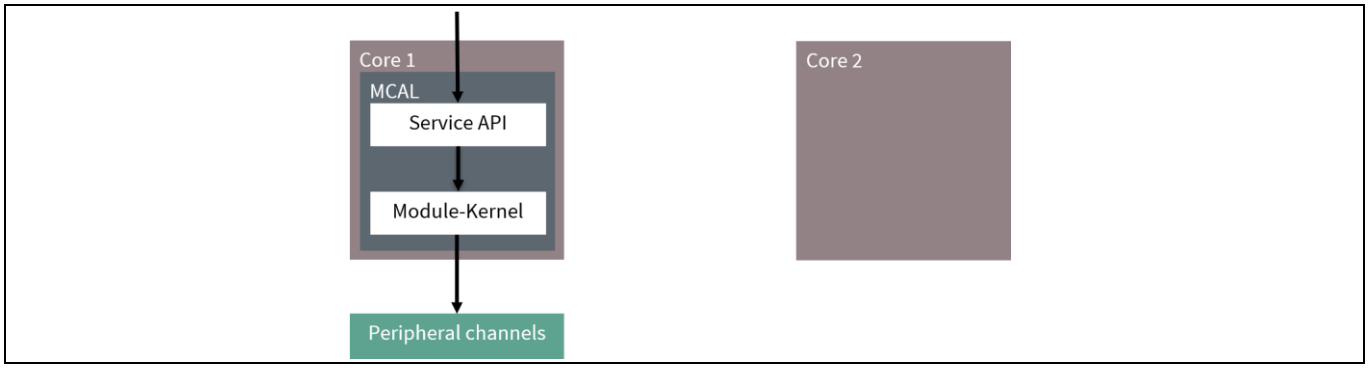

Figure 3 Overview of the multicore type I

1.7.1.2 Core-dependent instances (multicore type II)

For this multicore type, the driver has core-dependent instances with individually allocable hardware. This type is referred as multicore type II.

Multicore type II has the following characteristics:

- The driver code is shared among all cores:
  - A common binary is used for all cores.
  - A configuration is common for all cores.
- Each core runs an instance of the driver.
- Peripheral channels and their data can be individually allocated to cores, but cannot be shared among cores.
- One core will be the master; the master core must be initialized first:
  - Cores other than the master core are called satellite cores.

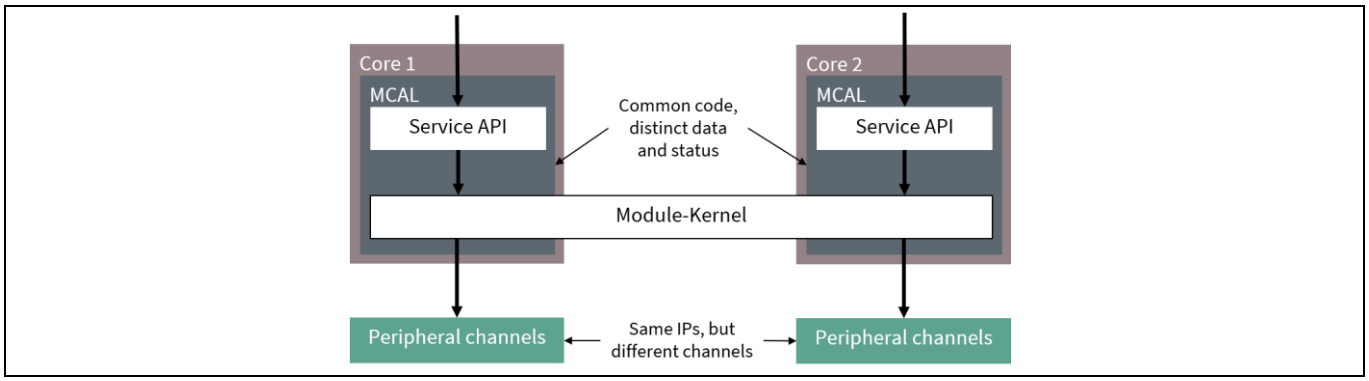

Figure 4 Overview of the multicore type II

1.7.1.3 Core-independent instances (multicore type III)

For this multicore type, the driver has core-independent instances with globally available hardware. This type is referred as multicore type III.

Multicore type III has the following characteristics:

- The driver code is shared among all cores:
  - A common binary is used for all cores.
  - A configuration is common for all cores.
- Each core runs an instance of the driver.
- Peripheral channels are globally available for all cores.

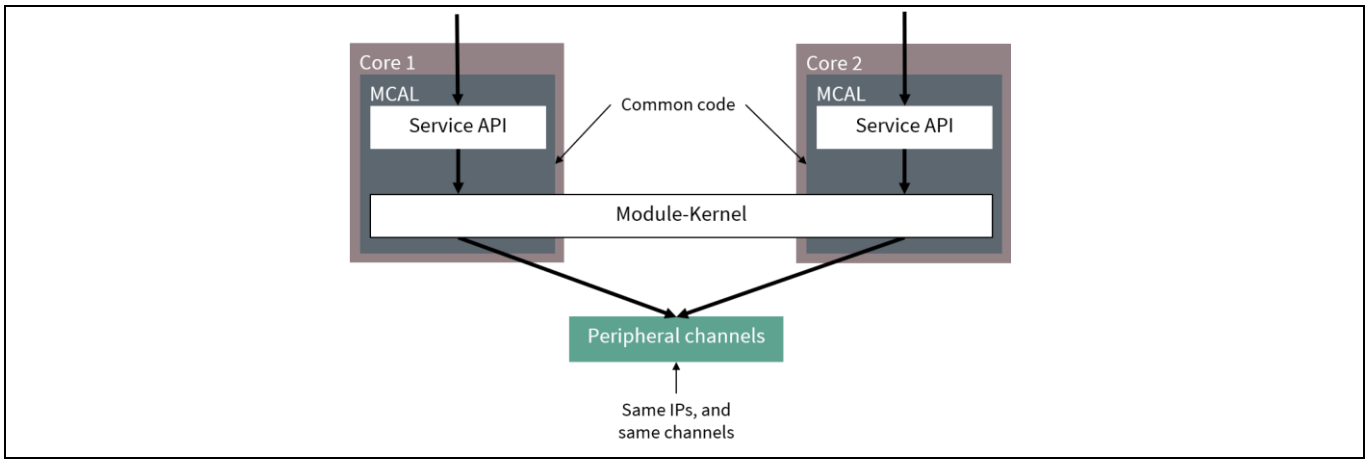

Figure 5 Overview of the multicore type III

2 Using the PORT driver

2.1 Installation and prerequisites

Note: *Before you start, see the EB tresos Studio for ACG8 user's guide for the following information:*

1. *How to install EB tresos ECU AUTOSAR components.*
2. *How to use EB tresos Studio.*
3. *How to use the EB tresos ECU AUTOSAR build environment (includes an explanation of how to set up and integrate your application within the EB tresos ECU AUTOSAR build environment).*

The installation of the PORT driver complies with the general installation procedure for EB tresos ECU AUTOSAR components given in the documents mentioned above. If the driver is successfully installed, it will appear in the module list of the EB tresos Studio (see *EB tresos Studio for ACG8 user's guide* [7]).

In the following section, it is assumed that the project is properly set up and is using the application template as described in the *EB tresos Studio for ACG8 user's guide* [7]. This template provides the necessary folder structure, project and Makefiles needed to configure and compile your application within the build environment. You need to be familiar with the usage of the command shell.

2.2 Configuring the PORT driver

The following container is used to configure common behavior.

- *PortGeneral*: This container is mainly used to restrict/extend the API of the PORT module and enable/disable default error trace.

For detailed information and description, see chapter 4 EB tresos Studio configuration interface.

Note: *Ensure that your application also includes an AUTOSAR-compliant DET when default error detection is enabled. Otherwise, your application will not compile if the* ***default error detection*** *option is switched on.*

You should provide values for the following characteristics for each configured or required port pin:

- PortPinDirection
- PortPinLevelValue
- PortPinDirectionChangeable at runtime
- PortPinModeChangeable at runtime

The *PortPin* container describes the information of the individual port pin. It has a user-changeable symbolic name which can be used in the application.

The PORT driver initializes the whole port structure of the microcontroller. All port pins that are not configured are initialized with the value in port default pin configuration. For detailed information and description, see 0

Port default pin configuration.

2.2.1 Architecture specifics

Refer to Chapter 4 EB tresos Studio configuration interface where all configuration parameters are described.

2.3 Adapting your application

1. To use the PORT driver in your application, include the PORT driver header file by adding the following line to your source file:

   ```
   #include "Port.h"   /* PORT Driver */
   ```

   This publishes all needed function/data prototypes and symbolic names of the configuration into the application.

2. Implement the error callout function for ASIL safety extension.

   Declare the error callout function in the specified file by the `PortIncludeFile` parameter and implement in your application (see 7.4 Required callback functions, Error callout API).

3. Implement callout functions for multicore exclusive access control.

   Declare the callout functions in the specified file by the `PortIncludeFile` parameter and implement in your application (see 7.4 Required callback functions, Callout port enter and callout port exit).

4. Initialize and configure the ports and port pins as described in chapter 4 EB tresos Studio configuration interface. The PORT module will be automatically enabled if an appropriate parameter configuration of the PORT module is available in your application.

   The port initialization can be done (in an initialization task) with the following function call and parameter.

   ```
   Port_Init(&Port_Config[0]);
   ```

Note: *`Port_Init()` must be called on the master core.*

   API functions can be called with the symbolic names (for example, `PortConf_PortPin_MY_IO_PIN`, `PortConf_PortTrGroupContainer_MY_TRIGGER_GROUP`, `PortConf_PortOutputTrigger_MY_OUTPUT_TRIGGER`, `PortConf_PortTr1To1GroupContainer_MY_TRIGGER_1TO1_GROUP` and `PortConf_Port1To1OutputTrigger_MY_1TO1_OUTPUT_TRIGGER`) from your configuration after `Port_Init()`.
   For more information about the `Port_SetPinDirection` function and other services, see chapter 5 Functional description.

   ```
   Port_SetPinDirection(PortConf_PortPin_MY_IO_PIN, PORT_PIN_OUT);
   ```

The pin mode can be changed at runtime using the `Port_SetPinMode()` function. The user is responsible to pass the architecture-specific mode value listed the datasheet for the selected subderivative. A list of all available modes can be found in the */generated/include/Port_Cfg_Der.h* file.

```
Port_SetPinMode(PortConf_PortPin_MY_IO_PIN, PORT_PIN_MODE_P000_0_GPIO);
```

The pin mode can be switched between DIO mode (`PORT_PIN_MODE_GPIO`) and configured mode at runtime using the `Port_SetToDioMode()` and `Port_SetToAlternateMode()` functions. If the pin's configured mode is DIO mode, these APIs have no effect.

```
/* Switch from the configured mode to DIO mode */
```

```
Port_SetToDioMode(PortConf_PortPin_MY_IO_PIN);


/* Switch from DIO mode to the configured mode */
Port_SetToAlternateMode(PortConf_PortPin_MY_IO_PIN);
```

The trigger setting can be changed at runtime using the `Port_SetTrigger()` function. The user is responsible to pass the architecture-specific input trigger listed the datasheet for the selected subderivative. A list of all available input triggers can be found in the */generated/include/Port_Cfg_Der.h* file.

```
Port_SetTrigger(PortConf_PortTrGroupContainer_MY_TRIGGER_GROUP,
                PortConf_PortOutputTrigger_MY_OUTPUT_TRIGGER,
                PORT_TR_GROUP_DW0_TR_IN_CPUSS_DW0_TR_OUT_0,
                FALSE, PORT_TR_SENSITIVE_LEVEL, FALSE);
```

Users can control trigger activation with the trigger command at runtime. The specified output trigger can be activated or deactivated using the `Port_ActTrigger()` and `Port_DeactTrigger()` functions. The trigger command functionality can control only one trigger activation. This feature cannot be used by multiple tasks or modules at the same time. The status of the trigger command can be acquired using the `Port_GetTriggerCmdStatus()` function.

```
/* If trcmd_status.activate = 1U, Trigger Command is not available because used by
other tasks or modules. */
Port_GetTriggerCmdStatus(&trcmd_status);
/* If the sensitive type is set to PORT_TR_SENSITIVE_LEVEL, the specified output
trigger / 1-to-1 output trigger continues the activated status until calling
Port_DeactTrigger(). */
Port_ActTrigger(PortConf_PortTrGroupContainer_MY_TRIGGER_GROUP,
                PortConf_PortOutputTrigger_MY_OUTPUT_TRIGGER,
                PORT_TR_ACTIVATION_OUTPUT,
                PORT_TR_SENSITIVE_LEVEL);
```

(`PortConf_PortOutputTrigger_MY_OUTPUT_TRIGGER` is being activated)

```
/* The activating output trigger is deactivated */
Port_DeactTrigger();
/* If the sensitive type is set to PORT_TR_SENSITIVE_EDGE, the specified output
trigger is activated for two "clk_peri" cycles. Port_DeactTrigger does not have to be
called. After generating two cycle pulse, the output trigger is deactivated by
hardware. */
Port_ActTrigger(PortConf_PortTrGroupContainer_MY_TRIGGER_GROUP,
                PortConf_PortOutputTrigger_MY_OUTPUT_TRIGGER,
                PORT_TR_ACTIVATION_OUTPUT,
                PORT_TR_SENSITIVE_EDGE);
/* If setting the input trigger for Trigger Command, the multiple output triggers
connected to the specified input trigger can be activated at the same time. */
Port_ActTrigger(PortConf_PortTrGroupContainer_MY_TRIGGER_GROUP,
                PORT_TR_GROUP_DW0_TR_IN_CPUSS_DW0_TR_OUT_0,
                PORT_TR_ACTIVATION_INPUT,
                PORT_TR_SENSITIVE_EDGE);
```

Additional functions:

- `Port_RefreshPortDirection();`

- `Port_GetVersionInfo(Std_VersionInfoType*);`
- `Port_GetStatus(PortConf_PortPin_MY_IO_PIN, Port_StatusType*);`
- `Port_GetAmuxSplitCtlStatus(PortConf_PortAmuxSplitCell_MY_CELL, Port_AmuxSplitCtlStatusType*);`
- `Port_GetTriggerIdStatus(PortConf_PortTrGroupContainer_MY_TRIGGER_GROUP, PortConf_PortOutputTrigger_MY_OUTPUT_TRIGGER, Port_TriggerIdStatusType*);`

These functions can be used wherever needed.

2.4 Starting the build process

Do the following to build your application:

Note: *For a clean build, use the build command with target `clean_all` before! (`make clean_all`).*

1. On the command shell, type the following command to generate the necessary configuration-dependent files. See 3.3 Generated files:

   ```
   > make generate
   ```

2. Type the following command to resolve the required file dependencies:

   ```
   > make depend
   ```

3. Type the following command to compile and link the application:

   ```
   > make (optional target: all)
   ```

The application is now built. All files are compiled and linked to a binary file which can be downloaded to the target CPU cores.

2.5 Measuring the stack consumption

Do the following to measure the stack consumption. The BASE module is needed for proper measurement.

Note: *All files (including library files) should be rebuilt with the dedicated compiler option. The executable file built by this step must be used only for stack consumption measurement.*

1. Add the following compiler option to the Makefile to enable stack consumption measurement.

   ```
   -DSTACK_ANALYSIS_ENABLE
   ```

2. Type the following command to clean library files.

   ```
   > make clean_lib
   ```

3. Follow the build process described in 2.4 Starting the build process.
4. Follow the instructions in the release notes and measure the stack consumption.

2.6 Memory mapping

The Port_MemMap.h file in the `$(TRESOS_BASE)/plugins/MemMap_TS_T40D13M0I0R0/include` directory is a sample. This sample file is replaced by the file generated by the MEMMAP module. Input to the MEMMAP module is generated as *Port_Bswmd.arxml* in the `$(PROJECT_ROOT)/output/generated/swcd` directory of your project folder.

2.6.1 Memory allocation keyword

- PORT_START_SEC_CODE_ASIL_B / PORT_STOP_SEC_CODE_ASIL_B

  Memory section type is CODE. All executable code is allocated in this section.

- PORT_START_SEC_CONST_ASIL_B_UNSPECIFIED / PORT_STOP_SEC_CONST_ASIL_B_UNSPECIFIED

  Memory section type is CONST. The following constants are allocated in this section:

  - Port configuration data
  - Hardware register base address data

- PORT_START_SEC_VAR_INIT_ASIL_B_UNSPECIFIED / PORT_STOP_SEC_VAR_INIT_ASIL_B_UNSPECIFIED

  Memory section type is VAR. The following variable is allocated in this section:

  - Pointer to the configuration data

2.6.2 Memory allocation and constraints

All the memory sections must be zero-initialized before any driver function is executed on any core.

- `VAR_INIT_ASIL_B_UNSPECIFIED`

  This section is read/write-accessed from the master core and read-accessed from other cores. So, this section must not be allocated to TCRAM. For the master core, this section must be allocated to either non-cache or write-through cache SRAM area. For other cores, this section must be allocated to the non-cache SRAM area.

3 Structure and dependencies

The PORT driver consists of a static configuration and generated files.

3.1 Static files

Folder	Description
$(PLUGIN_PATH)= $(TRESOS_BASE)/plugins/Port_TS_*	Path to the PORT module plugin.
$(PLUGIN_PATH)/lib_src	Contains all static source files of the PORT driver. These files contain the functionality of the driver, which does not depend on the current configuration. The files are grouped into a static library.
$(PLUGIN_PATH)/src	Contains configuration-dependent source files or special derivative files. Each file will be built again when the configuration is changed. All necessary source files will be automatically compiled and linked during the build process and all include paths will be set if the PORT driver is enabled.
$(PLUGIN_PATH)/include	Basic public include directory needed to include *Port.h*.
$(PLUGIN_PATH)/autosar	Contains the AUTOSAR ECU parameter definition with adaptations specific to vendor, architecture, and derivative to create a correct matching parameter configuration for the PORT module.

3.2 Configuration files

The configuration of the PORT driver is done with the EB tresos Studio. When saving a project, the configuration description is written to the *Port.xdm* file in `$(PROJECT_ROOT)/config` in your project folder. This file is the input to generate configuration-dependent source and header files during the build process.

3.3 Generated files

During the build process, the following files are generated based on the current configuration description. These files are in the sub folder `output/generated` of your project folder.

File	Description
include/Port_Cfg.h include/Port_Cfg_Der.h	Define all symbolic names for configured port pins. These files will be included in *Port.h*. Settings are separated into architecture- and derivative-specific.
include/Port_Cfg_Der_Internal_User.h	Defines the configuration for the user configured port pins.
include/Port_Cfg_Der_Internal_Defaults.h	Defines subderivative-specific default values for all pins. If a port pin is not configured by the user, the default configuration of the pin is added and used to create a complete port configuration structure to initialize the whole port structure of the microcontroller.
include/Port_Cfg_Include.h	Defines the custom include headers generated from the configuration.
src/Port_PBcfg.c	Contains the constant structure for the PORT configuration.
src/Port_PBcfg_Der.c	Contains the error check function dependent on the hardware.

File	Description
swcd/Port_Bswmd.arxml	Contains BswModuleDescription.

Note: *You do not need to add the generated source files to your application make file. They will be compiled and linked automatically during the build process.*

Note: *Additional steps are required for the generation of BSW module description. In EB tresos Studio, select **Project** > **Build Project** > **generate_swcd**.*

3.4 Dependencies

3.4.1 DET

If default error detection is enabled in the PORT module configuration, DET must be installed, configured, and built into the application.

3.4.2 Resource file

The PORT driver needs registered and configured Resource module. The PORT driver will report errors such as “port pin not available in this derivative” or “*PortPinMode/PortPinDirection* not possible” if this dependency is not resolved properly.

3.4.3 BSW scheduler

The Port handler/driver uses the following services of the BSW scheduler to enter and leave critical sections:

- `SchM_Enter_Port_PORT_EXCLUSIVE_AREA_0(void)`
- `SchM_Exit_Port_PORT_EXCLUSIVE_AREA_0(void)`

Make sure that the BSW scheduler is properly configured and initialized before using the Port handler/driver.

3.4.4 Error callout handler

The error callout handler is called on every error that is detected regardless of whether the default error detection is enabled or disabled. The error callout handler is an ASIL safety extension that is not specified by AUTOSAR. It is configured via the `PortErrorCalloutFunction` configuration parameter.

3.4.5 Callout functions for multicore exclusive access

Callout functions are called where multicore exclusive access is needed. Callout functions are configured via the `PortCalloutPortEnter` and `PortCalloutPortExit` configuration parameters.

4 EB tresos Studio configuration interface

The GUI is not part of the current delivery. For further information, see *EB tresos Studio for ACG8 user's guide* [7].

4.1 General configuration

Parameter	Description
`PortDevErrorDetect`	Enables or disables the default error notification for the PORT module. Setting this parameter to false will disable the notification of development errors via DET. However, in contrast to the AUTOSAR specification, detection of development errors is still enabled as a safety mechanism (fault detection).
`PortSetPinDirectionApi`	Enables or disables the `Port_SetPinDirection` function.
`PortSetPinModeApi`	Enables or disables the `Port_SetPinMode` function.
`PortVersionInfoApi`	Enables or disables the `Port_GetVersionInfo` function.
`PortSafetyFunctionApi`	Enables or disables the `Port_GetStatus`, `Port_GetAmuxSplitCtlStatus`, and `Port_GetTriggerIdStatus` functions.
`PortSetToDioAlternateModeApi`	Enables or disables the `Port_SetToDioMode` and `Port_SetToAlternateMode` functions.
`PortSetTriggerApi`	Enables or disables the `Port_SetTrigger` function.
`PortTriggerCommandApi`	Enables or disables the `Port_ActTrigger`, `Port_DeactTrigger`, and `Port_GetTriggerCmdStatus` functions.
`PortErrorCalloutFunction`	Used to specify the error callout function name. The function is called on every error. The ASIL level of this function limits the ASIL level of the PORT driver. *Note:* `PortErrorCalloutFunction` *must be valid C function name, otherwise an error would occur in the configuration phase.*
`PortCalloutPortEnter`	Used to specify the callout function name. The function is called on at the start of area where multicore exclusive access is needed. *Note:* `PortCalloutPortEnter` *must be valid C function name, otherwise an error would occur in the configuration phase.*
`PortCalloutPortExit`	Used to specify the callout function name. The function is called on at the end of area where multicore exclusive access is needed. *Note:* `PortCalloutPortExit` *must be valid C function name, otherwise an error would occur in the configuration phase.*

Parameter	Description
PortIncludeFile	Lists the file names that shall be included within the driver. Any application-specific symbol that is used by the Port configuration (e.g., error callout function) should be included by configuring this parameter. *Note:* *PortIncludeFile must be a filename with extension .h and a unique name, otherwise some errors would occur in the configuration phase.*

4.2 Port configuration set

Port configuration is used as a configuration pointer parameter of Port_Init(). The symbolic name is derived from the PortConfigSet container short name prefixed with "PortConf_PortConfigSet_".

The value is configuration data (that is, Port_Config[0], Port_Config[1]) and defined as the user-changeable symbolic name which can be used in the application.

- PortTriggerReference refers to the PortTriggerConfigSet for this PortConfigSet.

4.3 Port configuration

The following parameters are available to configure the ports:

Parameter	Description
PortNumberOfPortPins	Number of specified *PortPins* in this *PortContainer*.
PortId	Port ID in this *PortContainer*.
PortCalloutEnable	Specifies whether PortCalloutPortEnter and PortCalloutPortExit callouts will be called when processing the port. If a port will be configured on several cores, it needs to be set true. If enabled, PortCalloutPortEnter and PortCalloutPortExit are called inside of a critical section. The integrator shall be aware that the execution time of those functions adds up to the blocking time of the critical section.
PortSmartioClockSource	Source of clock ("clk_fabric") and reset ("rst_fabric_n") for the SMART I/O block.
PortSmartioHoldOverrideEnable	If checked, the SMART I/O controls the IO cell hold override functionality for the SMART I/O block. If not checked, it is controlled by the HSIOM.
PortSmartioPipelineEnable	If checked, the pipeline setting is enabled for the SMART I/O block.
PortSmartioEnable	If checked, the programmable I/O setting is enabled for the SMART I/O block.
PortSmartioDataUnitTr0Source	Data unit input signal "tr0_in" source for the SMART I/O block. Refer to Table 17 for selectable values.
PortSmartioDataUnitTr1Source	Data unit input signal "tr1_in" source for the SMART I/O block. Refer to Table 17 for selectable values.

Parameter	Description
PortSmartioDataUnitTr2Source	Data unit input signal "tr2_in" source for the SMART I/O block. Refer to Table 17 for selectable values.
PortSmartioDataUnitData0Source	Data unit input data "data0_in" source for the SMART I/O block. Refer to Table 18 for selectable values.
PortSmartioDataUnitData1Source	Data unit input data "data1_in" source for the SMART I/O block. Refer to Table 18 for selectable values.
PortSmartioDataUnitBitSize	Size/width of the data unit data operands (in bits) for the SMART I/O block.
PortSmartioDataUnitOperation	Data unit opcode specifies the data unit operation for the SMART I/O block. Refer to Table 19 for selectable values.
PortSmartioDataUnitSource	Data unit input data source for the SMART I/O block.

Note: You can configure some PortContainers into a PortConfigSet. Only one port can be configured in one PortContainer. Therefore, configure the required PortContainers for each port and create only those PortPins which you want to use in each PortContainer.

Note: The parameters for the SMART I/O block can be changed only if the SMART I/O functionality of the specified port is valid.
See the hardware manual for details about the SMART I/O configuration.

4.4 Port pin configuration

The following parameters are available to configure port pins:

Parameter	Description
PortPinDirection	The initial direction of the pin (IN, OUT or IN/OUT disabled). PORT_PIN_IN_OUT_DISABLED is for Analog I/O. It can be set if PortPinInitialMode is AMUXA, AMUXB, AMUXA_DSI, or AMUXB_DSI. Refer to Port_PinDirectionType for selectable values. *Note: If the pin is used in the AMUXA, AMUXA_DSI, AMUXB, or AMUXB_DSI mode, the pin direction must be set to PORT_PIN_IN_OUT_DISABLED. For other modes, the pin direction must be set to PORT_PIN_IN or PORT_PIN_OUT.* *Note: If the pin is used for the analog port in the GPIO mode, set the pin direction to PORT_PIN_OUT. In addition, set the PortPinOutputInBufEnable to false and PortPinOutputDrive to PORT_PIN_OUT_MODE_HIGHZ.* *Note: Hardware registers do not have the feature to fix the direction to input only. When the pin direction is set to PORT_PIN_IN, the pin drive mode is forced*

Parameter	Description
	to be `PORT_PIN_OUT_MODE_HIGHZ` *to disable the output direction. If any drive mode, other than* `PORT_PIN_OUT_MODE_HIGHZ`*, is required for the applications, set the pin direction to* `PORT_PIN_OUT` *with* `PortPinOutputInBufEnable` *as true (enable input and output direction).*
`PortPinDirectionChangeable`	If checked, the direction is changeable on a port pin during runtime.
`PortPinId`	The pin ID of the port pin. This value will be assigned to the symbolic name derived from the port pin container short name.
`PortPinName`	The port pin name. The symbolic name derived from the *PortPin* container, a short name prefixed with "`PortConf_PortPin_`".
`PortPinInitialMode`	The initial mode of the port pin. The allowed modes for a certain pin are hardware-dependent. Note that `PortPinInitialMode` influences the I/O behavior.
`PortPinModeChangeable`	If checked, the mode can be changed on a port pin during runtime.
`PortPinLevelValue`	The level value from the port pin and can be configured with each *PortPin*. Refer to Port_PinLevelValueType for selectable values. *Note:* *When the pin mode is not GPIO, the pin output shall be controlled by the peripheral functionality. Therefore, if the initialized pin mode is not GPIO, the configuration parameter* `PortPinLevelValue` *is ignored.*
`PortPinOutputDrive`	The output drive mode of the port pin and can be configured the available output drive with each *PortPin*. Refer to Table 7 for selectable values.
`PortPinOutputInBufEnable`	If checked, the input buffer is enabled when the pin direction is `PORT_PIN_OUT`. The actual output level can be read by this setting. If not checked, the input buffer is disabled and the level setting on the hardware can be read.
`PortPinMode`	`PortPinMode` will be ignored; the parameter `PortPinInitialMode` will be used. Selectable port pin mode is fixed in each subderivative.
`PortPinSmartioBypassEnable`	If checked, the SMART I/O path is bypassed for this pin's signal. It means that the SMART I/O block is not present for this port pin.

Parameter	Description
PortPinSmartioSyncIoEnable	If checked, I/O pin input signals to the SMART I/O block are synchronized to "clk_fabric", which is configured with PortSmartioClockSource.
PortPinSmartioSyncChipEnable	If checked, chip pin input signals to the SMART I/O block are synchronized to "clk_fabric", which is configured with PortSmartioClockSource.
PortPinSmartioLutTr0Source	The source of the LUT input signal "tr0_in". Refer to Table 15 for selectable values.
PortPinSmartioLutTr1Source	The source of the LUT input signal "tr1_in". Refer to Table 15 for selectable values.
PortPinSmartioLutTr2Source	The source of the LUT input signal "tr2_in". Refer to Table 15 for selectable values.
PortPinSmartioLut	The LUT configuration setting. It is used to determine the LUT output signal and the next sequential state.
PortPinSmartioLutOperation	The LUT opcode that specifies the LUT operation configuration. Refer to Table 16 for selectable values.
PortPinInputBufferMode	The input buffer mode (trip points and hysteresis). This parameter is available for IO cell types other than HSIO_ENH, HSIO_ENH_PDIFF, HSIO_ENH_STG and HSIO_ENH_PDIFF_STG. See Table 8 for selectable values.
PortPinOutputSlowSlewRateEnable	If checked, this port pin works in slow slew rate. If not checked, this port pin works in fast slew rate. *Note:* *PortPinOutputSlowSlewRateEnable can be changed only if the specified pin has slow slew rate functionality.*
PortPinOutputDriveStrength	This parameter is available for IO cell types other than HSIO_STDLN, HSIO_ENH, HSIO_ENH_PDIFF, HSIO_ENH_STG and HSIO_ENH_PDIFF_STG. The GPIO drive strength setting. See Table 9 for selectable values.
Port5VPinInputBufferMode	The input buffer mode (trip points and hysteresis) for S40E GPIO upper bit. This parameter is available for IO cell types other than HSIO_STD, HSIO_STDLN, HSIO_STD_STG, HSIO_ENH, HSIO_ENH_PDIFF, HSIO_ENH_STG and HSIO_ENH_PDIFF_STG. See Table 10 for selectable values.
PortPinOutputDriveSelectTrim	The GPIO drive select trim setting. Refer to Table 11 for selectable values. This parameter is available for TRAVEO™ T2G 2D cluster series only.
PortPinOutputSlewExt	The GPIO output extra slew rate control. See Table 12 for selectable values.

Parameter	Description
	Note: `PortPinOutputSlewExt` *is editable only if the extra slew rate control functionality of the specified pin is valid.*
`PortPinOutputDriveExt`	The GPIO output extra drive strength control. See Table 13 for selectable values. *Note:* `PortPinOutputDriveExt` *is editable only if the extra drive strength functionality of the specified pin is valid.*

4.5 AMUX splitter cell configuration

The following parameters are available to configure AMUX splitter cells. `PortAmuxSplitCell` is the name of this container.

Parameter	Description
`PortAmuxSplitCellId`	The cell ID of the AMUX splitter cell in the `PortAmuxSplitCell` container. The range is 0 to (the number of AMUX splitter cells - 1).
`PortAmuxBusaSwitchSL`	The T-switch control for left AMUXBUSA switch in this `PortAmuxSplitCell`. See Table 20 for selectable values.
`PortAmuxBusaSwitchSR`	The T-switch control for right AMUXBUSA switch in this `PortAmuxSplitCell`. See Table 20 for selectable values.
`PortAmuxBusaSwitchS0`	The T-switch control for AMUXBUSA `vssa/ground` switch in this `PortAmuxSplitCell`. See Table 20 for selectable values.
`PortAmuxBusbSwitchSL`	The T-switch control for left AMUXBUSB switch in this `PortAmuxSplitCell`. See Table 20 for selectable values.
`PortAmuxBusbSwitchSR`	The T-switch control for right AMUXBUSB switch in this `PortAmuxSplitCell`. See Table 20 for selectable values.
`PortAmuxBusbSwitchS0`	The T-switch control for AMUXBUSB `vssa/ground` switch in this `PortAmuxSplitCell`. See Table 20 for selectable values.

4.6 Port default configuration set

The port default configuration is used as default parameters for unconfigured port pins and AMUX splitter cells. Pins and cells that are not configured in `PortConfigSet` are initialized with `PortDefaultConfigSet`.

4.7 Port default configuration

- The PortDefaultContainer configuration provides the default parameters for unconfigured ports.

Parameter	Description
`PortDefCalloutEnable`	Specifies whether `PortCalloutPortEnter` and `PortCalloutPortExit` callouts will be called when processing the port. If default ports are configured on several cores, it needs to be set true. If enabled, `PortCalloutPortEnter` and `PortCalloutPortExit` are called inside of a critical section. The integrator must note that the execution time of those functions adds up to the blocking time of the critical section.
`PortDefSmartioClockSource`	The source of a clock ("clk_fabric") and a reset ("rst_fabric_n") for the SMART I/O block.
`PortDefSmartioHoldOverrideEnable`	If checked, the SMART I/O controls the I/O cell hold override functionality for the SMART I/O block. If not checked, it is controlled by the HSIOM.
`PortDefSmartioPipelineEnable`	If checked, the pipeline setting is enabled for the SMART I/O block.
`PortDefSmartioEnable`	If checked, the programmable I/O setting is enabled for the SMART I/O block.
`PortDefSmartioDataUnitTr0Source`	The source of a data unit input signal ("tr0_in") for the SMART I/O block. See Table 17 for selectable values.
`PortDefSmartioDataUnitTr1Source`	The source of a data unit input signal ("tr1_in") for the SMART I/O block. See Table 17 for selectable values.
`PortDefSmartioDataUnitTr2Source`	The source of a data unit input signal ("tr2_in") for the SMART I/O block. See Table 17 for selectable values.
`PortDefSmartioDataUnitData0Source`	The source of a data unit input data ("data0_in") for the SMART I/O block. See Table 18 for selectable values.
`PortDefSmartioDataUnitData1Source`	The source of a data unit input data ("data1_in") for the SMART I/O block. See Table 18 for selectable values.
`PortDefSmartioDataUnitBitSize`	The size/width of the data unit data operands (in bits) for the SMART I/O block.
`PortDefSmartioDataUnitOperation`	The data unit opcode that specifies the data unit operation for the SMART I/O block. See Table 19 for selectable values.
`PortDefSmartioDataUnitSource`	The data unit input data source for the SMART I/O block.

4.8 Port default pin configuration

The port default pin configuration provides the default parameters for unconfigured port pins.

Parameter	Description
PortDefPinDirection	The initial direction of unconfigured pins. (IN, OUT, or IN/OUT disabled). PORT_PIN_IN_OUT_DISABLED is for analog I/O. It can be set if PortDefPinInitialMode is AMUXA, AMUXB, AMUXA_DSI, or AMUXB_DSI. See Port_PinDirectionType for selectable values.
PortDefPinDirectionChangeable	If checked, the direction can be changed on unconfigured port pins during runtime.
PortDefPinInitialMode	The initial mode of unconfigured pins.
PortDefPinModeChangeable	If checked, the mode can be changed on unconfigured port pins during runtime.
PortDefPinLevelValue	The level value from unconfigured port pins. Refer to Port_PinLevelValueType for selectable values. *Note: When the pin mode is not GPIO, the pin output is controlled by the peripheral functionality. Therefore, if the initialized pin mode is not GPIO, the configuration parameter PortDefPinLevelValue is ignored.*
PortDefPinOutputDrive	The output drive mode of unconfigured port pins. See Table 7 for selectable values. *Note: When the pin mode is set to PORT_PIN_OUT_MODE_PULLUP_DOWN_ATST, the unconfigured pins will generally perform the same as the STRONG drive mode although this is not guaranteed for all use cases.*
PortDefPinOutputInBufEnable	If checked, the input buffer is enabled when the pin direction is PORT_PIN_OUT. The actual output level can be read by this setting. If not checked, the input buffer is disabled and the level setting on the hardware can be read.
PortDefPinSmartioBypassEnable	If checked, the SMART I/O path is bypassed for unconfigured pins' signal. It means that the SMART I/O blocks are not present for unconfigured port pins.
PortDefPinSmartioSyncIoEnable	If checked, the I/O pin input signals to the SMART I/O block are synchronized to "clk_fabric", which is configured with PortDefSmartioClockSource.
PortDefPinSmartioSyncChipEnable	If checked, the chip pin input signals to the SMART I/O block are synchronized to "clk_fabric", which is configured with PortDefSmartioClockSource.

Parameter	Description
PortDefPinSmartioLutTr0Source	The source of the LUT input signal "tr0_in" on unconfigured port pins. See Table 15 for selectable values.
PortDefPinSmartioLutTr1Source	The source of the LUT input signal "tr1_in" on unconfigured port pins. See Table 15 for selectable values.
PortDefPinSmartioLutTr2Source	The source of the LUT input signal "tr2_in" on unconfigured port pins. See Table 15 for selectable values.
PortDefPinSmartioLut	The LUT configuration setting. It is used to determine the LUT output signal and the next sequential state.
PortDefPinSmartioLutOperation	The LUT opcode that specifies the LUT operation setting on unconfigured port pins. See Table 16 for selectable values.
PortDefPinInputBufferMode	The input buffer mode (trip points and hysteresis) on unconfigured port pins. See Table 8 for selectable values.
PortDefPinOutputSlowSlewRateEnable	If checked, unconfigured port pins work in slow slew rate. If not checked, these port pins work in fast slew rate.
PortDefPinOutputDriveStrength	The GPIO drive strength setting on unconfigured port pins. See Table 9 for selectable values.
PortDef5VPinInputBufferMode	The input buffer mode (trip points and hysteresis) for S40E GPIO upper bit on unconfigured port pins. See Table 10 for selectable values.
PortDefPinOutputDriveSelectTrim	The GPIO drive select trim setting on unconfigured port pins. See Table 11 for selectable values. This parameter is available for TRAVEO™ T2G 2D cluster series only.
PortDefPinOutputSlewExt	The GPIO output extra slew rate control. See Table 12 for selectable values. This parameter is available for TRAVEO™ T2G 2D cluster series only.
PortDefPinOutputDriveExt	The GPIO output extra drive strength control. See Table 13 for selectable values. This parameter is available for TRAVEO™ T2G 2D cluster series only.

4.9 Port default AMUX splitter cell configuration

The PortDefaultAmuxSplitCell configuration provides the default parameters for unconfigured AMUX splitter cells.

Parameter	Description
PortDefAmuxBusaSwitchSL	The T-switch control for the left AMUXBUSA switch. See Table 20 for selectable values.
PortDefAmuxBusaSwitchSR	The T-switch control for the right AMUXBUSA switch. See Table 20 for selectable values.
PortDefAmuxBusaSwitchS0	The T-switch control for the AMUXBUSA vssa/ground switch. See Table 20 for selectable values.
PortDefAmuxBusbSwitchSL	The T-switch control for the left AMUXBUSB switch. See Table 20 for selectable values.
PortDefAmuxBusbSwitchSR	The T-switch control for the right AMUXBUSB switch. See Table 20 for selectable values.

Parameter	Description
PortDefAmuxBusbSwitchS0	The T-switch control for the AMUXBUSB vssa/ground switch. See Table 20 for selectable values.

4.10 Port trigger configuration set

The Port Trigger Configuration provides the configuration parameters for trigger groups and trigger 1-to-1 groups. See the hardware manual for details about triggers.

Parameter	Description
PortTriggerConfigSetId	The ID number of *PortTriggerConfigSet*. The range is 0 to (the number of *PortTriggerConfigSet* - 1).

4.11 Port trigger group configuration

The port trigger group configuration provides the configuration parameters for trigger groups.

Parameter	Description
PortTrGroupId	The ID of the trigger group specified by PortTrGroupName.
PortTrGroupName	The name of selected trigger group. Selectable trigger groups are hardware dependent. The symbolic name is derived from the *PortTrGroupContainer* container short name prefixed with "PortConf_PortTrGroupContainer_". *Note:* *Set PortTrGroupName, and then press the **calculate** button of the PortTrGroupId parameter to get the corresponding PortTrGroupId.*

4.12 Port output trigger configuration

The port output trigger configuration provides the configuration parameters for output triggers.

Parameter	Description
PortTrOutputName	The name of selected output trigger. Selectable output triggers depend on the trigger group in PortTrGroupName. The symbolic name is derived from the *PortOutputTrigger* container short name prefixed with "PortConf_PortOutputTrigger_".
PortTrInputName	The name of input trigger connected to output trigger for this *PortOutputTrigger*. The selectable input triggers depend on the trigger group in PortTrGroupName.
PortTrInvertEnable	If checked, the output trigger invert functionality is enabled in this *PortOutputTrigger*.
PortTrSensitiveType	The output trigger edge sensitive functionality type in this *PortOutputTrigger*. See Port_TriggerSensitiveType for selectable values.

Parameter	Description
	Note: `PortTrInvertEnable` *and* `PortTrSensitiveType` *can be changed only if trigger groups have trigger manipulation features.*
`PortTrDbgFreezeEnable`	If checked, the debug freeze functionality is enabled in this *PortOutputTrigger*. *Note:* *When calling* `Port_Init()`*, unconfigured output triggers are initialized as follows:* `PortTrInputName`*=0,* `PortTrInvertEnable`*=disable,* `PortTrSensitiveType`*="Level Sensitive" and* `PortTrDbgFreezeEnable`*=disable.*

4.13 Port trigger 1-to-1 group configuration

The port trigger 1-to-1 group configuration provides the configuration parameters for trigger 1-to-1 groups.

Parameter	Description
`PortTr1To1GroupId`	The ID of the trigger 1-to-1 group specified by `PortTr1To1GroupName`.
`PortTr1To1GroupName`	The name of selected trigger 1-to-1 group. Selectable trigger 1-to-1 groups are hardware-dependent. The symbolic name is derived from the *PortTr1To1GroupContainer* container short name prefixed with "`PortConf_PortTr1To1GroupContainer_`". *Note:* *Set* `PortTr1To1GroupName` *and then press the calculate button of the* `PortTr1To1GroupId` *parameter to get the corresponding* `PortTr1To1GroupId`*.*

4.14 Port 1-to-1 output trigger configuration

The port 1-to-1 output trigger configuration provides the configuration parameters for 1-to-1 output triggers.

Parameter	Description
`PortTr1To1OutputName`	The name of selected 1-to-1 output trigger. Selectable 1-to-1 output triggers depend on the trigger 1-to-1 group in `PortTr1To1GroupName`. The symbolic name is derived from the *Port1To1OutputTrigger* container short name prefixed with "`PortConf_Port1To1OutputTrigger_`".
`PortTr1To1InputType`	The input trigger type in this *Port1To1OutputTrigger*. See Table 21 for selectable values.
`PortTr1To1InvertEnable`	If checked, the output trigger invert functionality is enabled in this *Port1To1OutputTrigger*.
`PortTr1To1SensitiveType`	`PortTr1To1SensitiveType` is the output trigger edge sensitive functionality type in this *Port1To1OutputTrigger*. See Port_TriggerSensitiveType for selectable values.

Parameter	Description
	Note: `PortTr1To1InvertEnable` *and* `PortTr1To1SensitiveType` *can be changed only if trigger 1-to-1 groups have trigger manipulation features.*
`PortTr1To1DbgFreezeEnable`	If checked, the debug freeze functionality is enabled in this *Port1To1OutputTrigger*. *Note:* *When calling* `Port_Init()`*, unconfigured 1-to-1 output triggers are initialized as follows:* `PortTr1To1InputType`*=Constant signal level '0',* `PortTr1To1InvertEnable`*=disable,* `PortTr1To1SensitiveType`*="Level Sensitive" and* `PortTr1To1DbgFreezeEnable`*=disable.*

5 Functional description

5.1 Required header file

The *Port.h* file includes all necessary external identifiers. Therefore, the application only needs to include *Port.h* to make all API functions and data types available.

5.2 Initialization

The PORT driver provides an initialization function for initializing the microcontroller's whole port structure. As it is possible to configure more than one configuration, the `Port_Init` function can be called with different configuration sets. The `Port_Init` function must be called at least once after reset; it can be called several times in order to reconfigure the ports and port pins of the microcontroller. It must be called on the master core.

Code Listing 1 Example using the Port_Init() function using the 1st configuration set

```
Port_Init(&Port_Config[0]);
```

Note: The PORT driver module's environment must call the `Port_Init` function first to initialize the port for use.
If this function is not called first, no operation can occur on microcontroller ports and port pins.
When the initialized pin direction is output and is the same as the previous setting, `Port_Init` will keep the pin output. However, if the pin's output configuration is updated with this function call, output signals may be sent with the previous setting for a short period.
To configure SMART I/O, the functionality must be disabled while SMARTIO_PRT registers are updated. Note that the disabled period must occur during initialization.
This function enters/exits critical sections of every initialization of each port or trigger group. The duration for initializing a trigger group depends on the number of output triggers. Therefore, if a trigger group has many output triggers, the exclusive control period becomes long.
Flash boot configures the JTAG reset input pin to SWJ_TRSTN mode upon reset (see the device datasheet for the exact pin number). This pin is optional for JTAG debug protocol and can be configured to some other mode. If the debug protocol is JTAG and this pin is being configured to some other mode (GPIO, for example) via `Port_Init()`, then debug session becomes unstable. This applies only when the debug protocol is JTAG. This issue can be avoided by modifying the JTAG reset input to GPIO in the scope of the debug script with the following register modification sequence before executing `Port_Init()` inside the application code.
1. Modify the HSIOM_PRT_SEL register.
2. Modify the GPIO_PRT_CFG register.

5.3 Runtime reconfiguration

Changing the port pin direction is available only for port pins configured with the ability to change the direction.

Changing the port pin mode is available only for port pins configured with the ability to change mode.

Note: *`Port_Init()` can be used to reconfigure the whole port structure according to the configuration set provided by the `ConfigPtr` parameter.*

Code Listing 2 Example using the Port_SetPinDirection() function and Port_SetPinMode() function

```
#include "Port.h"
/* ... */
/* initialize ports */
/* .... */
/* set pin direction */
Port_SetPinDirection(PortConf_PortPin_MY_IO_PIN, PORT_PIN_IN);
/* set pin mode */
Port_SetPinMode(PortConf_PortPin_MY_IO_PIN, my_pin_mode);
```

The direction of port pins configured as direction unchangeable can be refreshed. All changes at runtime for port directions of port pins that are configured as "direction changeable at runtime" keep their last state.

Code Listing 3 Example using the Port_RefreshPortDirection() function

```
#include "Port.h"
/* ... */
/* initialize ports */
/* .... */
/* refresh pin direction */
Port_RefreshPortDirection(void);
```

The PORT driver provides a reconfiguration function for an output trigger setting and a 1-to-1 output trigger setting.

Code Listing 4 Example using the Port_SetTrigger() function for output trigger and 1-to-1 output trigger

```
#include "Port.h"
/* ... */
/* initialize ports */
/* .... */
/* set output trigger */
Port_SetTrigger(PortConf_PortTrGroupContainer_PortTrGroupContainer_0,
                PortConf_PortOutputTrigger_PortOutputTrigger_0,
```

Code Listing 4 Example using the Port_SetTrigger() function for output trigger and 1-to-1 output trigger

```
                PORT_TR_GROUP_DW0_TR_IN_CPUSS_DW0_TR_OUT_0, FALSE,
                PORT_TR_SENSITIVE_LEVEL, FALSE);
/* set 1-to-1 output trigger */
Port_SetTrigger(PortConf_PortTr1To1GroupContainer_PortTr1To1GroupContainer_
0,
                PortConf_Port1To1OutputTrigger_Port1To1OutputTrigger_0,
                PORT_TR_1TO1_IN_INPUT, FALSE, PORT_TR_SENSITIVE_LEVEL,
FALSE);
```

5.4 Ports and port pins

A port pin represents a single pin of the microcontroller. The value of a single pin can either be 0 or 1.

A port represents several port pins, which are grouped according to hardware and are controlled by one hardware register. The corresponding data type for a port's value depends on the bit width of the largest port. Writing to a port with a smaller width ignores the upper bits.

Note: The PORT module only handles port pins. An overall grouping or configuring into ports is not possible.

5.5 Trigger command

The trigger command provides software control over trigger activation. This is useful for software-initiated triggers or for debugging purposes. The command enables software activation of one specific input trigger or output trigger of the trigger multiplexer setting.

Code Listing 5 Example using the Port_ActTrigger() and Port_DeactTrigger() function.

```
#include "Port.h"
/* ... */
/* initialize ports */
/* .... */
/* activate trigger command */
Port_ActTrigger(PortConf_PortTrGroupContainer_PortTrGroupContainer_0,
                PortConf_PortOutputTrigger_PortOutputTrigger_0,
                PORT_TR_ACTIVATION_OUTPUT, PORT_TR_SENSITIVE_LEVEL);
/* .... */
/* deactivate trigger command */
Port_DeactTrigger(void);
```

Note: Modules that use the trigger command must call the `Port_GetTriggerCmdStatus` function to confirm that the edge-sensitive trigger has been completed.
Modules that use the trigger command must ensure that the trigger command activation status is deactivated using `Port_GetTriggerCmdStatus` before calling `Port_ActTrigger`.

Modules that use the trigger command must call the `Port_DeactTrigger` function only if the trigger command is successfully activated as level-sensitive.

5.6 API parameter checking

The driver's services perform regular error checks.

When an error occurs, the error hook routine (configured via `PortErrorCalloutFunction`) is called, and the error code, as well as service ID, module ID and instance ID are passed as parameters.

If default error detection is enabled, all errors are also reported to the DET, a central error hook function within the AUTOSAR environment. The checking itself cannot be deactivated for safety reasons.

The following development error checks are performed by the services of the PORT driver:

- The `Port_Init` function checks whether the parameter configuration pointer `ConfigPtr` is valid.
  - If invalid, the error code *PORT_E_INIT_FAILED* is reported.
- The following functions check whether the ports and port pins have been initialized. If the module initialization function is not called before, or configuration is invalid, *PORT_E_UNINIT* is reported.
  - `Port_SetPinDirection`
  - `Port_SetPinMode`
  - `Port_RefreshPortDirection`
  - `Port_GetStatus`
  - `Port_SetToDioMode`
  - `Port_SetToAlternateMode`
  - `Port_GetAmuxSplitCtlStatus`
  - `Port_SetTrigger`
  - `Port_ActTrigger`
  - `Port_DeactTrigger`
  - `PortGetTriggerIdStatus`
  - `Port_GetTriggerCmdStatus`
- The `Port_SetPinDirection` function checks whether the parameter pin identifier `Pin` is valid.
  - If invalid, the error code *PORT_E_PARAM_PIN* is reported.
  - If `PortPinDirectionChangeable` is set to unchangeable for this `Pin`, the error code *PORT_E_DIRECTION_UNCHANGEABLE* is reported.
  - When a direction change to an unknown `Direction` (different from PORT_PIN_IN, PORT_PIN_OUT or PORT_PIN_IN_OUT_DISABLED) is requested, the error code *PORT_E_PARAM_INVALID_DIRECTION* will be reported.
- The `Port_SetPinMode` function checks whether the parameter pin identifier `Pin` is valid.
  - If invalid, the error code *PORT_E_PARAM_PIN* is reported.
  - If `PortPinModeChangeable` is set to unchangeable for this `Pin`, the error code *PORT_E_MODE_UNCHANGEABLE* is reported.
  - If the new requested `Mode` is not valid, the error code *PORT_E_PARAM_INVALID_MODE* is reported.
- The `Port_GetVersionInfo` function checks whether the parameter version information pointer `versioninfo` is a NULL pointer.
  - If NULL pointer, the error code *PORT_E_PARAM_POINTER* is reported.

- The `Port_GetStatus` function checks whether the parameter port status information pointer `PortStatusInfoPtr` is a NULL pointer.
  - If NULL pointer, the error code *PORT_E_PARAM_POINTER* is reported.
- The `Port_GetStatus` function checks whether the parameter pin identifier `Pin` is valid.
  - If invalid, the error code *PORT_E_PARAM_PIN* is reported.
- The `Port_GetStatus` function checks whether the register values are valid.
  - If the register value is invalid, the error code *PORT_E_ REGISTER* is reported.
- The `Port_SetToDioMode` and `Port_SetToAlternateMode` functions check whether the parameter pin identifier `Pin` is valid.
  - If invalid, the error code *PORT_E_PARAM_PIN* is reported.
  - If `PortPinModeChangeable` is set to unchangeable for this `Pin`, the error code *PORT_E_MODE_UNCHANGEABLE* is reported.
- The `Port_GetAmuxSplitCtlStatus` function checks whether the parameter AMUX splitter cell status information pointer `AmuxSplitCtlStatusInfoPtr` is a NULL pointer.
  - If NULL pointer, the error code *PORT_E_PARAM_POINTER* is reported.
- The `Port_GetAmuxSplitCtlStatus` function checks whether the parameter cell identifier `Cell` is valid.
  - If invalid, the error code *PORT_E_PARAM_CELL* is reported.
- The `Port_SetTrigger`, `Port_ActTrigger`, and `Port_GetTriggerIdStatus` functions check whether the parameter trigger group identifier `group_id` is valid.
  - If invalid, the error code *PORT_E_PARAM_TR_GROUP* is reported.
- The `Port_SetTrigger` and `Port_GetTriggerIdStatus` functions check whether the parameter output trigger identifier `out_trg` is valid.
  - If invalid, the error code *PORT_E_PARAM_TR_OUTPUT* is reported.
- The `Port_SetTrigger` function checks whether the parameter trigger group identifier `group_id` is being used in trigger command.
  - If the trigger command is activated with the same trigger group as this `group_id`, the error code *PORT_E_TR_CMD_STATUS* is reported.
- `The Port_SetTrigger` function checks whether the parameter input trigger identifier `in_trg` is valid.
  - If invalid, the error code *PORT_E_PARAM_TR_INPUT* is reported.
- The `Port_SetTrigger` function checks whether the parameter trigger group identifier `group_id` indicates the trigger group which has trigger manipulation features.
  - If it does not have trigger manipulation features when `inv_flg` is true or `sensitive_type` is *PORT_TR_SENSITIVE_EDGE*, the error code *PORT_E_TR_MANIPULATION_NOT_PRESENT* is reported.
- The `Port_SetTrigger` and `Port_ActTrigger` functions check whether the parameter trigger sensitive type `sensitive_type` is valid.
  - If invalid, the error code *PORT_E_PARAM_TR_SENSITIVE* is reported.
- The `Port_ActTrigger` function checks whether the parameter trigger activation type `act_type` is valid.
  - If invalid, the error code *PORT_E_PARAM_TR_ACTIVATION* is reported.
- The `Port_ActTrigger` function checks whether the parameter trigger identifier `trg_id` is valid for input trigger when the parameter trigger activation type `act_type` is *PORT_TR_ACTIVATION_INPUT*.
  - If invalid, the error code *PORT_E_PARAM_TR_INPUT* is reported.
- The `Port_ActTrigger` function checks whether the parameter trigger identifier `trg_id` is valid for output trigger when the parameter trigger activation type `act_type` is *PORT_TR_ACTIVATION_OUTPUT*.

  - If invalid, the error code *PORT_E_PARAM_TR_OUTPUT* is reported.
- The `Port_ActTrigger` function checks whether the trigger command is deactivated.
  - If the trigger command is activated, the error code *PORT_E_TR_CMD_STATUS* is reported.
- The `Port_DeactTrigger` function checks whether the trigger command is activated as level sensitive.
  - If that trigger command is deactivated or activated as edge sensitive, the error code *PORT_E_TR_CMD_STATUS* is reported.
- The `Port_GetTriggerIdStatus` function checks whether the parameter trigger ID status information pointer `TrigIdStatusInfoPtr` is a NULL pointer.
  - If a NULL pointer, the error code *PORT_E_PARAM_POINTER* is reported.
- The `Port_GetTriggerCmdStatus` function checks whether the parameter trigger command status information pointer `TrigCmdStatusInfoPtr` is a NULL pointer.
  - If a NULL pointer, the error code *PORT_E_PARAM_POINTER* is reported.

5.7 Configuration checking

The following conditions are checked when configuring the PORT driver with EB tresos Studio:

- Multiple occurrences of the same short name for *PortConfigSet* containers

  The short name for the *PortConfigSet* container must be unique.
- Multiple occurrences of the same short name for *PortContainer* containers

  The short name for the *PortContainer* container must be unique in the same configuration set.
- The short name for the *PortContainer* container is different from *PortContainer* which has the same `PortId`.

  All `PortId` values of containers with the same short name should be the same in all configuration sets.
- Multiple occurrences of the same short name for *PortPin* containers

  The short name for the *PortPin* container must be unique in the same configuration set.
- The short name for the *PortPin* container is different from *PortPin* which has same `PortPinId`.

  All `PortPinId` values of the container with the same short name should be the same in all configuration sets.
- Multiple occurrences of the same `PortId`

  `PortId` must be unique in the same configuration set.
- Multiple occurrences of the same `PortPinId`

  Each `PortPinId` must be used once to create a symbolic name and configuration for this pin. It is not possible to configure additional pins (different symbolic name) referring the same `PortPinId`.
- Architecture- or derivative-specific port pin configuration check

  Some basic checks are done for each port pin to ensure that the initial mode, direction, level, and direction change could be covered by this pin. If the mode is not supported by this pin, an error is reported at project generation time.
- Multiple occurrences of the same `PortPinName`

  `PortPinName` must be unique in the same configuration set.

- Multiple occurrences of the same short name for *PortAmuxSplitCell* containers

  The short name for the *PortAmuxSplitCell* container must be unique in the same configuration set.

- The short name for the *PortAmuxSplitCell* container is different from *PortAmuxSplitCell,* which has same `PortAmuxSplitCellId`.

  All `PortAmuxSplitCellId` of the container with the same short name should be the same in all configuration sets.

- Multiple occurrences of the same `PortAmuxSplitCellId`

  `PortAmuxSplitCellId` must be unique in the same configuration set.

- Multiple occurrences of the same short name for *PortTriggerConfigSet* containers

  The short name for the *PortTriggerConfigSet* container must be unique.

- Multiple occurrences of the same `PortTriggerConfigSetId`

  `PortTriggerConfigSetId` must be unique across all port sequences.

- Multiple occurrences of the same short name for *PortTrGroupContainer* containers

  The short name for the *PortTrGroupContainer* container must be unique in the same configuration set.

- The short name for the *PortTrGroupContainer* container is different from *PortTrGroupContainer*, which has the same `PortTrGroupId`.

  All `PortTrGroupId` of the container with the same short name should be the same in all configuration sets.

- Multiple occurrences of same `PortTrGroupName`

  The trigger group name must be unique in the same configuration set.

- Multiple occurrences of the same short name for *PortOutputTrigger* containers

  The short name for the *PortOutputTrigger* container must be unique in the same configuration set.

- The short name for the *PortOutputTrigger* container is different from *PortOutputTrigger*, which has the same `PortTrOutputName`.

  All `PortTrOutputName` of the container with the same short name should be the same in all configuration sets.

- Multiple occurrences of the same `PortTrOutputName`

  The output trigger name must be unique in the same configuration set.

- Multiple occurrences of the same short name for *PortTr1To1GroupContainer* containers

  The short name for the *PortTr1To1GroupContainer* container must be unique in the same configuration set.

- The short name for the PortTr1To1GroupContainer container is different from PortTr1To1GroupContainer, which has the same `PortTr1To1GroupId`.

  All `PortTr1To1GroupId` of the container with the same short name should be the same in all configuration sets.

- Multiple occurrences of the same `PortTr1To1GroupName`

The trigger 1-to-1 group name must be unique in the same configuration set.

- Multiple occurrences of the same short name for *Port1To1OutputTrigger* containers

  The short name for the *Port1To1OutputTrigger* container must be unique in the same configuration set.

- The short name for *Port1To1OutputTrigger* container is different from *Port1To1OutputTrigger,* which has the same `PortTr1To1OutputName`.

  All `PortTr1To1OutputName` of the container with the same short name should be the same all configuration sets.

- Multiple occurrences of the same `PortTr1To1OutputName`

  The 1-to-1 output trigger name must be unique in the same configuration set.

5.8 Reentrancy

The following functions are not reentrant.

- `Port_Init()`
- `Port_RefreshPortDirection()`
- `Port_ActTrigger()`
- `Port_DeactTrigger()`

All other functions are re-entrant to each other and itself if accessing different pins (independent of a port) or different output triggers (independent of a trigger group).

5.9 Function availability in multicore

The following functions are available on any core without any restriction:

- `Port_GetVersionInfo()`
- `Port_GetStatus()`
- `Port_GetAmuxSplitCtlStatus()`
- `Port_GetTriggerIdStatus()`
- `Port_GetTriggerCmdStatus()`

The following function is available only on the master core:

- `Port_Init()`

The following functions are available only on one of the cores:

- `Port_SetTrigger()`
- `Port_ActTrigger()`
- `Port_DeactTrigger()`

The following functions are available on any cores with restrictions:

- `Port_SetPinDirection()`
- `Port_SetPinMode()`
- `Port_SetToDioMode()`
- `Port_SetToAlternateMode()`
- `Port_RefreshPortDirection()`

Note: *If these API functions are used on several cores to configure the same port, `PortCalloutEnable` in the `PortContainer` must be set to true. When the port is in `PortDefContainer`, `PortCalloutDefEnable` must be set to true. In addition, multicore access control callout functions (`PortCalloutPortEnter` and `PortCalloutPortExit`) must be implemented.*
If these API functions are used under following conditions, there is no restriction.
- When these API functions are used on a single core.
- When these API functions are used on several cores and these APIs aren't used to configure a same port.
In that case, `PortCalloutEnable` and `PortCalloutDefEnable` can be set to false, and function body of `PortCalloutPortEnter` and `PortCalloutPortExit` can be empty.

5.10 Debugging support

The PORT driver does not support debugging.

6 Hardware resources

6.1 Ports and pins

All GPIO pins of the TRAVEO™ T2G microcontroller can be used as digital I/O using the PORT driver.

Any port can operate in 8-pin or 1-pin units and all registers can be read or written in 1-bit or 32-bit units. The maximum number of ports and the available mode combination depend on the subderivative. For more details, refer to the respective Hardware Manual.

6.2 SMART I/O blocks

SMART I/O blocks allow Boolean operations on signals going to the GPIO pins from the device subsystems or on signals coming into the device. Operation can be synchronous or asynchronous and the blocks operate in low-power modes, such as DeepSleep.

Several ports have their own SMART I/O blocks depending on the subderivative. For more details, refer to the respective hardware manual.

6.3 AMUX splitter cells

An AMUX splitter cell is capable of grounding, disconnecting, or feeding through each AMUXBUS, and can act as a bridge between two AMUXBUS segments in different analog supply domains.

The number of AMUX splitter cells depends on the subderivative. For more details, refer to the respective hardware manual.

6.4 Trigger groups and trigger 1-to-1 groups

In general, a trigger input signal indicates the completion of a peripheral action or a peripheral event. A trigger output signal initiates a peripheral action. A trigger group is a multiplexer-based connectivity group. This type connects a peripheral input trigger to multiple peripheral output triggers. The selection is under software control. Trigger 1-to-1 group is a 1-to-1-based connectivity group. This type connects a peripheral input trigger to one specific peripheral output trigger.

The types of trigger groups/trigger 1-to-1 groups, the types of output triggers/1-to-1 output triggers for each trigger group/ trigger 1-to-1 group and the types of input triggers for each trigger group depend on the subderivative. For more details, refer to the respective Hardware Manual.

6.5 Timer

The PORT driver does not use any hardware timers.

6.6 Interrupts

The PORT driver does not use any interrupts.

7 Appendix A – API reference

7.1 Data types

7.1.1 Port_ConfigType

Type

```
typedef struct
```

Description

Defines a structure which holds the PORT driver configuration set and a pointer to the configuration data for each channel.

7.1.2 Port_PinType

Type

```
uint16
```

Description

Stores an identifier for each pin. It contains the port number of the pin in the upper 13 bits and the pin number within the port in the lower 3 bits.

7.1.3 Port_PinModeType

Type

```
uint8
```

Description

Stores the possible different modes that can be changed at runtime. Modes resulting in the same change of HW register are grouped together.

7.1.4 Port_PinDirectionType

Type

```
typedef enum
{
   PORT_PIN_IN = 0,
   PORT_PIN_OUT = 1,
   PORT_PIN_IN_OUT_DISABLED = 2
} Port_PinDirectionType
```

Description

This type defines an enum describing the possible directions a port pin can have. Use PORT_PIN_IN to set the pin input direction. Use PORT_PIN_OUT to set the pin output direction. Use PORT_PIN_IN_OUT_DISABLED to disable input and output directions while the pin mode is AMUXA, AMUXB, AMUXA_DSI, or AMUXB_DSI.

7.1.5 Port_PinLevelValueType

Type

```
typedef enum
{
   PORT_PIN_LEVEL_LOW  = 0,
   PORT_PIN_LEVEL_HIGH = 1
} Port_PinLevelValueType
```

Description

This type defines an enum describing the possible level value of a port pin. Use PORT_PIN_LEVEL_LOW to set the port pin as LOW. Use PORT_PIN_LEVEL_HIGH to set the port pin as HIGH.

7.1.6 Port_StatusType

Type

```
typedef struct
{
    uint8   direction;
    uint8   mode;
    uint8   outputDrive;
    uint8   inputBufferMode;
    uint8   outputSlowSlewRateEnable;
    uint8   outputDriveStrength;
    uint8   inputBufferMode5VPin;
    uint8   outputDriveSelectTrim;
    uint8   outputSlewRateExt;
    uint8   outputDriveStrengthExt;
    uint8   sioPinsOutputBufferMode;
    uint8   sioPinsInputBufferMode;
    uint8   sioPinsInputBufferVrefTripPoint;
    uint8   sioPinsInputBufferVohOutputLevel;
    uint8   sioPinsAnalogDftEnable;
    uint8   smartioBypassEnable;
    uint8   smartioClockSource;
    uint8   smartioHoldOverrideEnable;
    uint8   smartioPipelineEnable;
    uint8   smartioEnable;
    uint8   smartioSyncIoEnable;
    uint8   smartioSyncChipEnable;
    uint8   smartioLutTr0Source;
    uint8   smartioLutTr1Source;
    uint8   smartioLutTr2Source;
    uint8   smartioLut;
    uint8   smartioLutOperation;
    uint8   smartioDataUnitTr0Source;
    uint8   smartioDataUnitTr1Source;
    uint8   smartioDataUnitTr2Source;
    uint8   smartioDataUnitData0Source;
    uint8   smartioDataUnitData1Source;
    uint8   smartioDataUnitBitSize;
    uint8   smartioDataUnitOperation;
    uint8   smartioDataUnitSource;
} Port_StatusType
```

Description

Structure for informing the setting of the PORT channel read from the HW register.

- direction – Port pin direction (see Port_PinDirectionType).
  When port pin direction is PORT_PIN_OUT and GPIO output drive mode is configured to PORT_PIN_OUT_MODE_HIGHZ, PORT_PIN_IN or PORT_PIN_IN_OUT_DISABLED is returned. This is because the hardware status is the same as the returned direction. If `PortPinOutputInBufEnable` is true, PORT_PIN_IN is returned. If false, PORT_PIN_IN_OUT_DISABLED is returned.
- mode – Port pin mode (see Table 6).
- outputDrive – GPIO output drive mode (see Table 7).
- inputBufferMode – Input buffer mode (trip points and hysteresis) (see Table 8).
- outputSlowSlewRateEnable – Setting of the slow slew rate (Enabled=1U, Disabled=0U)
- outputDriveStrength – GPIO drive strength (see Table 9).
- inputBufferMode5VPin – Input buffer mode (trip points and hysteresis) for S40E GPIO upper bit (see Table 10).
- outputDriveSelectTrim – GPIO drive select trim. (see Table 11) If the pin does not have the drive select trim function, returns fixed 0x00.
- outputSlewRateExt – Output extra slew rate (see Table 12). If the pin does not have the output extra slew rate control function, returns fixed 0x00.
- outputDriveStrengthExt – Output extra drive strength (see Table 13). If the pin does not have the output extra drive strength control function, returns fixed 0x00.
- sioPinsOutputBufferMode – Output buffer mode. If the pin does not have the SIO function, returns fixed 0x00.
- sioPinsInputBufferMode – Input buffer trip point in single ended input buffer mode. If the pin does not have the SIO function, returns fixed 0x00.
- sioPinsInputBufferVrefTripPoint – Reference voltage (Vref) trip-point of the input buffer setting. If the pin does not have the SIO function, returns fixed 0x00.
- sioPinsInputBufferVohOutputLevel – Regulated V_{OH} output level and trip point of the input buffer for a specific SIO pin pair. If the pin does not have the SIO function, returns fixed 0x00.
- sioPinsAnalogDftEnable – Reserved.
- smartioBypassEnable – Setting of the programmable I/O bypass (Enabled=1U, Disabled=0U). If the pin does not have the SMART I/O function, returns fixed 0x00.
- smartioClockSource – Clock ("clk_fabric") and reset ("rst_fabric_n") source (see Table 14). If the pin does not have the SMART I/O function, returns fixed 0x00.
- smartioHoldOverrideEnable – Setting of the I/O cell hold override functionality (Enabled=1U, Disabled=0U). If the pin does not have the SMART I/O function, returns fixed 0x00.
- smartioPipelineEnable – Setting of the pipeline register (Enabled=1U, Disabled=0U). If the pin does not have the SMART I/O function, returns fixed 0x00.
- smartioEnable – Setting of the programmable I/O (Enabled=1U, Disabled=0U). If the pin does not have the SMART I/O function, returns fixed 0x00.
- smartioSyncIoEnable – Setting of the I/O pin input signal synchronization to "clk_fabric" (Enabled=1U, Disabled=0U). If the pin does not have the SMART I/O function, returns fixed 0x00.
- smartioSyncChipEnable – Setting of the chip input signal synchronization to "clk_fabric" (Enabled=1U, Disabled=0U). If the pin does not have the SMART I/O function, returns fixed 0x00.

- smartioLutTr0Source – LUT input signal "tr0_in" source (see Table 15). If the pin does not have the SMART I/O function, returns fixed 0x00.
- smartioLutTr1Source – LUT input signal "tr1_in" source (see Table 15). If the pin does not have the SMART I/O function, returns fixed 0x00.
- smartioLutTr2Source – LUT input signal "tr2_in" source (see Table 15). If the pin does not have the SMART I/O function, returns fixed 0x00.
- smartioLut – LUT configuration. If the pin does not have the SMART I/O function, returns fixed 0x00.
- smartioLutOperation – LUT operation (see Table 16). If the pin does not have the SMART I/O function, returns fixed 0x00.
- smartioDataUnitTr0Source – Data unit input signal "tr0_in" source (see Table 17). If the pin does not have the SMART I/O function, returns fixed 0x00.
- smartioDataUnitTr1Source – Data unit input signal "tr1_in" source (see Table 17). If the pin does not have the SMART I/O function, returns fixed 0x00.
- smartioDataUnitTr2Source – Data unit input signal "tr2_in" source (see Table 17). If the pin does not have the SMART I/O function, returns fixed 0x00.
- smartioDataUnitData0Source – Data unit input data "data0_in" source (see Table 18). If the pin does not have the SMART I/O function, returns fixed 0x00.
- smartioDataUnitData1Source – Data unit input data "data1_in" source (see Table 18). If the pin does not have the SMART I/O function, returns fixed 0x00.
- smartioDataUnitBitSize – Size / width of the data unit data operands (in bits). If the pin does not have the SMART I/O function, returns fixed 0x00.
- smartioDataUnitOperation – Data unit operation (see Table 19). If the pin does not have the SMART I/O function, returns fixed 0x00.
- smartioDataUnitSource – Data unit input data source. If the pin does not have the SMART I/O function, returns fixed 0x00.

7.1.7 Port_AmuxCellType

Type

`uint8`

Description

Stores the AMUX splitter cell.
Available values are dependent on HW.

7.1.8 Port_AmuxSplitCtlStatusType

Type

```
typedef struct
{
    uint8   amuxBusaSwitchSL;
    uint8   amuxBusaSwitchSR;
    uint8   amuxBusaSwitchS0;
    uint8   amuxBusbSwitchSL;
    uint8   amuxBusbSwitchSR;
    uint8   amuxBusbSwitchS0;
} Port_AmuxSplitCtlStatusType
```

Description

Structure for informing the setting of AMUX splitter cell read from the HW register.

- amuxBusaSwitchSL – T-switch control for Left AMUXBUSA switch (see Table 20).
- amuxBusaSwitchSR – T-switch control for Right AMUXBUSA switch (see Table 20).
- amuxBusaSwitchS0 – T-switch control for AMUXBUSA vssa/ground switch (see Table 20).
- amuxBusbSwitchSL – T-switch control for Left AMUXBUSB switch (see Table 20).
- amuxBusbSwitchSR – T-switch control for Right AMUXBUSB switch (see Table 20).
- amuxBusbSwitchS0 – T-switch control for AMUXBUSB vssa/ground switch (see Table 20).

7.1.9 Port_TriggerGroupIdType

Type

`uint8`

Description

Stores the trigger group Id.
0 - 15: Trigger group
16 - 31: Trigger 1-to-1 group.
Available values are dependent on HW.

7.1.10 Port_TriggerIdType

Type

`uint8`

Description

Stores the trigger ID type for output triggers, input triggers, and 1-to-1 output triggers.
Available values are dependent on HW.

7.1.11 Port_TriggerSensitiveType

Type

```
typedef enum
{
    PORT_TR_SENSITIVE_LEVEL = 0,
    PORT_TR_SENSITIVE_EDGE  = 1
} Port_TriggerSensitiveType;
```

Description

Defines an enum describing the trigger sensitive type.
Use PORT_TR_SENSITIVE_LEVEL to set level-sensitive.
Use PORT_TR_SENSITIVE_EDGE to set edge-sensitive.

7.1.12 Port_TriggerActivationType

Type

```
typedef enum
{
    PORT_TR_ACTIVATION_INPUT  = 0,
    PORT_TR_ACTIVATION_OUTPUT = 1
} Port_TriggerActivationType;
```

Description

Defines an enum describing the trigger activation type.
Use PORT_TR_ACTIVATION_INPUT to set the input trigger.
Use PORT_TR_ACTIVATION_OUTPUT to set the output trigger.

7.1.13 Port_TriggerIdStatusType

Type

```
typedef struct
{
    Port_TriggerIdType          trInput;
    boolean                     trInvertEnable;
    Port_TriggerSensitiveType   trSensitiveType;
    boolean                     trDbgFreezeEnable;
} Port_TriggerIdStatusType;
```

Description

Structure for informing the setting of output trigger / 1-to-1 output trigger read from the HW register.

- trInput – Input trigger ID for output trigger. Input trigger type for 1-to-1 output trigger.
- trInvertEnable – Setting of the trigger invert (Enabled=1U, Disabled=0U).
- trSensitiveType – Trigger sensitive type.
- trDbgFreezeEnable – Setting of the trigger debug freeze (Enabled=1U, Disabled=0U).

7.1.14 Port_TriggerCmdStatusType

Type

```
typedef struct
{
    Port_TriggerGroupIdType     group_id;
    Port_TriggerIdType          trg_id;
    Port_TriggerActivationType  act_type;
    Port_TriggerSensitiveType   sensitive_type;
    uint8                       activate;
} Port_TriggerCmdStatusType;
```

Description

Structure for informing the setting of trigger command read from the HW register.

- group_id – Trigger group Id. 0 - 15: Trigger group 16 - 31: Trigger 1-to-1 group.
- trg_id – Output trigger ID or 1-to-1 output trigger ID in case of act_type=PORT_TR_ACTIVATION_OUTPUT. Input trigger ID in case of act_type=PORT_TR_ACTIVATION_INPUT.
- act_type – Trigger activation type.

- sensitive_type – Trigger sensitive type.
- activate – Trigger command activation status (Activated=1U, Deactivated=0U).

7.2 Constants

7.2.1 Error codes

The service might return the error codes, show in Table 1 if default error detection is enabled:

Table 1 Error codes

Name	Value	Description
PORT_E_PARAM_PIN	10	Incorrect port pin ID is passed.
PORT_E_DIRECTION_UNCHANGEABLE	11	Port pin is not configured as changeable.
PORT_E_INIT_FAILED	12	NULL pointer is passed, or parameter is out of range.
PORT_E_PARAM_INVALID_MODE	13	Requested mode is not valid.
PORT_E_MODE_UNCHANGEABLE	14	Mode is not changeable.
PORT_E_UNINIT	15	PORT driver is not initialized or configuration is invalid.
PORT_E_PARAM_POINTER	16	NULL pointer is passed.
PORT_E_PARAM_CELL	29	The cell value is invalid.
PORT_E_PARAM_INVALID_DIRECTION	32	Requested direction not valid.
PORT_E_REGISTER	33	The register value and the configuration do not match, or register value for setting the port pin is not valid.
PORT_E_PARAM_TR_GROUP	34	The trigger group ID value is invalid.
PORT_E_PARAM_TR_OUTPUT	35	The output trigger ID value is invalid.
PORT_E_PARAM_TR_INPUT	36	The input trigger ID value is invalid.
PORT_E_PARAM_TR_SENSITIVE	37	The trigger sensitive type value is invalid.
PORT_E_PARAM_TR_ACTIVATION	38	The trigger activation type value is invalid.
PORT_E_TR_CMD_STATUS	39	The trigger command status is invalid.
PORT_E_TR_MANIPULATION_NOT_PRESENT	40	The trigger group does not have trigger manipulation features.

7.2.2 Version information

Table 2 lists the version information published in the driver's header file.

Table 2 Version information

Name	Value	Description
PORT_SW_MAJOR_VERSION	Refer to release notes	Major version number
PORT_SW_MINOR_VERSION	Refer to release notes	Minor version number
PORT_SW_PATCH_VERSION	Refer to release notes	Patch version number

7.2.3 Module information

Table 3 Module information

Name	Value	Description
PORT_MODULE_ID	124	Module ID
PORT_VENDOR_ID	66	Vendor ID

7.2.4 API service IDs

The API service IDs, listed in Table 4, are published in the driver's header file.

Table 4 API service IDs

Name	Value	Description
PORT_API_INIT	0x0	Port initialization service
PORT_API_SET_PIN_DIRECTION	0x1	Set pin direction service
PORT_API_REFRESH_PORT_DIRECTION	0x2	Refresh port direction service
PORT_API_GET_VERSION_INFO	0x3	Get version information service
PORT_API_SET_PIN_MODE	0x4	Set pin mode service
PORT_API_SET_TRIGGER	0xF7	Set trigger service
PORT_API_ACT_TRIGGER	0xF8	Activate trigger command service
PORT_API_DEACT_TRIGGER	0xF9	Deactivate trigger command service
PORT_API_GET_STATUS_TR_ID	0xFA	Get trigger status service
PORT_API_GET_STATUS_TR_CMD	0xFB	Get trigger command status service
PORT_API_SET_TO_DIO_MODE	0xFC	Set GPIO mode service
PORT_API_SET_TO_ALTERNATE_MODE	0xFD	Set alternate mode service
PORT_API_GET_STATUS	0xFE	Get pin setting service
PORT_API_GET_AMUX_CTL_STATUS	0xFF	Get AMUX splitter cell status service

7.2.5 Port pin status

The following port pin status are published in the driver's header file:

Table 5 Common

Name	Value	Description
PORT_PIN_STATUS_FAILURE	0xFF	An abnormal value is set.

Table 6 Port pin mode

Name	Value	Description
PORT_PIN_MODE_GPIO	0	GPIO controls "out"
PORT_PIN_MODE_GPIO_DSI	1	GPIO controls "out", DSI controls "output enable"
PORT_PIN_MODE_DSI_DSI	2	DSI controls "out" and "output enable"
PORT_PIN_MODE_DSI_GPIO	3	DSI controls "out", GPIO controls "output enable"
PORT_PIN_MODE_AMUXA	4	Analog mux bus A

Name	Value	Description
PORT_PIN_MODE_AMUXB	5	Analog mux bus B
PORT_PIN_MODE_AMUXA_DSI	6	Analog mux bus A, DSI control
PORT_PIN_MODE_AMUXB_DSI	7	Analog mux bus B, DSI control

Other values (8-31) and corresponding modes are hardware dependent. Available pin modes are different for each port pin. For details of each type, see Hardware documentation.

Table 7 Output drive

Name	Value	Description
PORT_PIN_OUT_MODE_HIGHZ	0	High impedance (high-z)
PORT_PIN_OUT_MODE_PULLUP_DOWN_ATST	1	Resistive pull up and down at the same time for SMC
PORT_PIN_OUT_MODE_PULLUP	2	Resistive pull up
PORT_PIN_OUT_MODE_PULLDOWN	3	Resistive pull down
PORT_PIN_OUT_MODE_OD_LOW	4	Open drain, drives low
PORT_PIN_OUT_MODE_OD_HIGH	5	Open drain, drives high
PORT_PIN_OUT_MODE_STRONG	6	Strong
PORT_PIN_OUT_MODE_PULLUP_DOWN	7	Resistive pull up and down

For details of each type, see Hardware documentation.

Table 8 Input buffer mode

Name	Value	Description
PORT_PIN_IN_MODE_CMOS	0	Input buffer compatible with CMOS and I2C interfaces
PORT_PIN_IN_MODE_TTL	1	Input buffer compatible with TTL interfaces

For details of each type, see Hardware documentation.

Table 9 Output drive strength

Name	Value	Description
PORT_PIN_OUT_STRENGTH_DEFAULT	0	When the pin is GPIO_STD/GPIO_ENH/ GPIO_STD_STG/ GPIO_ENH_STG: GPIO drives current at its max rated spec. When the pin is SMC/HSIO_STD/SMC_STG/HSIO_STD_STG: SMC/HSIO_STD /SMC_STG/HSIO_STD_STG default mode.
PORT_PIN_OUT_STRENGTH_FULL	1	When the pin is GPIO_STD/GPIO_ENH/ GPIO_STD_STG/ GPIO_ENH_STG: GPIO drives current at its max rated spec (same meaning as PORT_PIN_OUT_STRENGTH_DEFAULT) When the pin is SMC/HSIO_STD/SMC_STG/HSIO_STD_STG: GPIO drives current at its max rated spec.
PORT_PIN_OUT_STRENGTH_1_2	2	GPIO drives current at 1/2 of its max rated spec.
PORT_PIN_OUT_STRENGTH_1_4	3	GPIO drives current at 1/4 of its max rated spec.

For details of each type, see Hardware documentation.

Table 10 Input buffer mode for S40E GPIO

Name	Value	Description
PORT_PIN_5V_IN_MODE_CMOS_OR_TTL	0	Input buffer is compatible with CMOS/TTL interfaces. If input buffer mode is PORT_PIN_IN_MODE_CMOS, the digital input buffer mode is set to CMOS. If input buffer mode is PORT_PIN_IN_MODE_TTL, the digital input buffer mode is set to TTL.
PORT_PIN_5V_IN_MODE_AUTO	1	Input buffer is compatible with AUTO (elevated Vil) interfaces.

For details of each type, see Hardware documentation.

Table 11 Output drive select trim

Name	Value	Description
PORT_PIN_OUT_TRIM_DEFAULT	0	Default (50 ohms)
PORT_PIN_OUT_TRIM_DS_120OHM	1	120 Ω
PORT_PIN_OUT_TRIM_DS_90OHM	2	90 Ω
PORT_PIN_OUT_TRIM_DS_60OHM	3	60 Ω
PORT_PIN_OUT_TRIM_DS_50OHM	4	50 Ω
PORT_PIN_OUT_TRIM_DS_30OHM	5	30 Ω
PORT_PIN_OUT_TRIM_DS_20OHM	6	20 Ω
PORT_PIN_OUT_TRIM_DS_15OHM	7	15 Ω

For details of each type, see Hardware documentation.

Table 12 Output extra slew rate control

Name	Value	Description
PORT_PIN_OUT_SLEW_EXT_FAST	0	Fastest slew rate
PORT_PIN_OUT_SLEW_EXT_SLOW	1	Slowest slew rate

For details of each type, see Hardware documentation.

Table 13 Output extra drive strength control

Name	Value	Description
PORT_PIN_OUT_STRENGTH_EXT_0	0	When PortPinOutputSlewExt is PORT_PIN_OUT_SLEW_EXT_FAST: 133 MHz at 15 pF, xSPI-266 mode (default) When PortPinOutputSlewExt is PORT_PIN_OUT_SLEW_EXT_SLOW: 125 MHz at 15 pF
PORT_PIN_OUT_STRENGTH_EXT_1	1	When PortPinOutputSlewExt is PORT_PIN_OUT_SLEW_EXT_FAST: 100 MHz at 15 pF, xSPI-200 mode When PortPinOutputSlewExt is PORT_PIN_OUT_SLEW_EXT_SLOW:

Name	Value	Description
		90 MHz at 15 pF
PORT_PIN_OUT_STRENGTH_EXT_2	2	When PortPinOutputSlewExt is PORT_PIN_OUT_SLEW_EXT_FAST: 80 MHz at 15 pF, Graphics When PortPinOutputSlewExt is PORT_PIN_OUT_SLEW_EXT_SLOW: 60 MHz at 15 pF
PORT_PIN_OUT_STRENGTH_EXT_3	3	When PortPinOutputSlewExt is PORT_PIN_OUT_SLEW_EXT_FAST: 64 MHz at 15 pF When PortPinOutputSlewExt is PORT_PIN_OUT_SLEW_EXT_SLOW: 50 MHz at 15 pF, Ethernet
PORT_PIN_OUT_STRENGTH_EXT_4	4	12 MHz at 20 pF, 25 MHz at 10 pF, SPI

For details of each type, see Hardware documentation.

Table 14 Clock and reset source

Name	Value	Description
PORT_SMART_CLK_SRC_IO0	0	Clock / reset sources are io_data_in[0] / '1'.
PORT_SMART_CLK_SRC_IO1	1	Clock / reset sources are io_data_in[1] / '1'.
PORT_SMART_CLK_SRC_IO2	2	Clock / reset sources are io_data_in[2] / '1'.
PORT_SMART_CLK_SRC_IO3	3	Clock / reset sources are io_data_in[3] / '1'.
PORT_SMART_CLK_SRC_IO4	4	Clock / reset sources are io_data_in[4] / '1'.
PORT_SMART_CLK_SRC_IO5	5	Clock / reset sources are io_data_in[5] / '1'.
PORT_SMART_CLK_SRC_IO6	6	Clock / reset sources are io_data_in[6] / '1'.
PORT_SMART_CLK_SRC_IO7	7	Clock / reset sources are io_data_in[7] / '1'.
PORT_SMART_CLK_SRC_CHIP0	8	Clock / reset sources are chip_data[0] / '1'.
PORT_SMART_CLK_SRC_CHIP1	9	Clock / reset sources are chip_data[1] / '1'.
PORT_SMART_CLK_SRC_CHIP2	10	Clock / reset sources are chip_data[2] / '1'.
PORT_SMART_CLK_SRC_CHIP3	11	Clock / reset sources are chip_data[3] / '1'.
PORT_SMART_CLK_SRC_CHIP4	12	Clock / reset sources are chip_data[4] / '1'.
PORT_SMART_CLK_SRC_CHIP5	13	Clock / reset sources are chip_data[5] / '1'.
PORT_SMART_CLK_SRC_CHIP6	14	Clock / reset sources are chip_data[6] / '1'.
PORT_SMART_CLK_SRC_CHIP7	15	Clock / reset sources are chip_data[7] / '1'.
PORT_SMART_CLK_SRC_CLK_SM_RST_ACT	16	Clock / reset sources are clk_smartio / rst_sys_act_n
PORT_SMART_CLK_SRC_CLK_SM_RST_DS	17	Clock / reset sources are clk_smartio / rst_sys_dpslp_n
PORT_SMART_CLK_SRC_CLK_SM_RST_DS_HIB	18	Same as *PORT_SMART_CLK_SRC_CLK_SM_RST_DS*. (See Hardware documentation)

Name	Value	Description
PORT_SMART_CLK_SRC_CLK_LF_RST_LF_DS	19	Clock / reset sources are clk_lf/rst_lf_dpslp_n (note that "clk_lf" is available in DeepSleep power mode)
PORT_SMART_CLK_SRC_CONST0	20	Clock / reset sources are constant '0'
PORT_SMART_CLK_SRC_ASYNC	31	Clock / reset sources are asynchronous mode / '1'

For details of each type, see Hardware documentation.

Table 15 LUT input signal source

Name	Value	Description
PORT_SMART_LUT_TR_DU	0	Data unit output (for "tr0_in")
PORT_SMART_LUT_TR_LUT0	0	LUT 0 output (for "tr1_in" or "tr2_in")
PORT_SMART_LUT_TR_LUT1	1	LUT 1 output
PORT_SMART_LUT_TR_LUT2	2	LUT 2 output
PORT_SMART_LUT_TR_LUT3	3	LUT 3 output
PORT_SMART_LUT_TR_LUT4	4	LUT 4 output
PORT_SMART_LUT_TR_LUT5	5	LUT 5 output
PORT_SMART_LUT_TR_LUT6	6	LUT 6 output
PORT_SMART_LUT_TR_LUT7	7	LUT 7 output
PORT_SMART_LUT_TR_CHIP04	8	chip_data[0] (for LUTs 0, 1, 2, 3) chip_data[4] (for LUTs 4, 5, 6, 7)
PORT_SMART_LUT_TR_CHIP15	9	chip_data[1] (for LUTs 0, 1, 2, 3) chip_data[5] (for LUTs 4, 5, 6, 7)
PORT_SMART_LUT_TR_CHIP26	10	chip_data[2] (for LUTs 0, 1, 2, 3) chip_data[6] (for LUTs 4, 5, 6, 7)
PORT_SMART_LUT_TR_CHIP37	11	chip_data[3] (for LUTs 0, 1, 2, 3) chip_data[7] (for LUTs 4, 5, 6, 7)
PORT_SMART_LUT_TR_IO04	12	io_data_in[0] (for LUTs 0, 1, 2, 3) io_data_in[4] (for LUTs 4, 5, 6, 7)
PORT_SMART_LUT_TR_IO15	13	io_data_in[1] (for LUTs 0, 1, 2, 3) io_data_in[5] (for LUTs 4, 5, 6, 7)
PORT_SMART_LUT_TR_IO26	14	io_data_in[2] (for LUTs 0, 1, 2, 3) io_data_in[6] (for LUTs 4, 5, 6, 7)
PORT_SMART_LUT_TR_IO37	15	io_data_in[3] (for LUTs 0, 1, 2, 3) io_data_in[7] (for LUTs 4, 5, 6, 7)

For details of each type, see Hardware documentation.

Table 16 LUT opcode

Name	Value	Description
PORT_SMART_LUT_OPC_COM_OUT_NO_FB	0	Combinatorial output, no feedback
PORT_SMART_LUT_OPC_COM_OUT_FB	1	Combinatorial output, feedback

Name	Value	Description
PORT_SMART_LUT_OPC_SEQ_OUT_NO_FB	2	Sequential output, no feedback
PORT_SMART_LUT_OPC_REG_ASYNC	3	Register with asynchronous set and reset

For details of each type, see Hardware documentation.

Table 17 Data unit input signal source

Name	Value	Description
PORT_SMART_DU_TR_CONST0	0	Constant '0'
PORT_SMART_DU_TR_CONST1	1	Constant '1'
PORT_SMART_DU_TR_DU	2	Data unit output
PORT_SMART_DU_TR_LUT0	3	LUT 0 output
PORT_SMART_DU_TR_LUT1	4	LUT 1 output
PORT_SMART_DU_TR_LUT2	5	LUT 2 output
PORT_SMART_DU_TR_LUT3	6	LUT 3 output
PORT_SMART_DU_TR_LUT4	7	LUT 4 output
PORT_SMART_DU_TR_LUT5	8	LUT 5 output
PORT_SMART_DU_TR_LUT6	9	LUT 6 output
PORT_SMART_DU_TR_LUT7	10	LUT 7 output

For details of each type, see Hardware documentation.

Table 18 Data unit input data source

Name	Value	Description
PORT_SMART_DU_DATA_CONST0	0	Constant "0"
PORT_SMART_DU_DATA_CHIP	1	chip_data[7:0]
PORT_SMART_DU_DATA_IO	2	io_data_in[7:0]
PORT_SMART_DU_DATA_DATA_REG	3	DATA.DATA MMIO register field

For details of each type, see Section Hardware documentation.

Table 19 Data unit opcode

Name	Value	Description
PORT_SMART_DU_OPC_INCR	1	INCR
PORT_SMART_DU_OPC_DECR	2	DECR
PORT_SMART_DU_OPC_INCR_WRAP	3	INCR_WRAP
PORT_SMART_DU_OPC_DECR_WRAP	4	DECR_WRAP
PORT_SMART_DU_OPC_INCR_DECR	5	INCR_DECR
PORT_SMART_DU_OPC_INCR_DECR_WRAP	6	INCR_DECR_WRAP
PORT_SMART_DU_OPC_ROR	7	ROR
PORT_SMART_DU_OPC_SHR	8	SHR
PORT_SMART_DU_OPC_AND_OR	9	AND_OR
PORT_SMART_DU_OPC_SHR_MAJ3	10	SHR_MAJ3

Name	Value	Description
PORT_SMART_DU_OPC_SHR_EQL	11	SHR_EQL

For details of each type, see Hardware documentation.

7.2.6 AMUX splitter cell status

The following AMUX splitter cell status is published in the driver's header file:

Table 20 AMUX T-switch

Name	Value	Description
PORT_AMUX_SWITCH_OPEN	0	AMUX T-switch is open
PORT_AMUX_SWITCH_CLOSED	1	AMUX T-switch is closed

For details of each type, see Hardware documentation.

7.2.7 Trigger status

The following trigger status is published in the driver's header file:

Table 21 Trigger 1-to-1 input type

Name	Value	Description
PORT_TR_1TO1_IN_CONST0	0	Constant signal level '0'.
PORT_TR_1TO1_IN_INPUT	1	Input trigger.

For details of each type, see Hardware documentation.

7.3 Functions

7.3.1 Port_Init

Syntax

```
void Port_Init
(
   const Port_ConfigType* ConfigPtr
)
```

Service ID

0x0

Parameters (in)

- `ConfigPtr` – Pointer to configuration set

Parameters (out)

None

Return value

None

DET errors

- *PORT_E_INIT_FAILED* – NULL pointer is given to this function, or `ConfigPtr` is not the pointer to the configuration information of the PORT module.

DEM errors

None

Description

This service initializes all ports and port pins, AMUX splitter cells, trigger groups, and trigger commands with the configuration set. It is the responsibility of the calling environment to execute `Port_Init` on the master core only, and not on any other core. The PORT module does not check whether `Port_Init` is called on the master core or not; therefore, any error report will not be generated if `Port_Init` is called from a core other than the master core.

7.3.2 Port_SetPinDirection

Syntax

```
void Port_SetPinDirection
(
   Port_PinType          Pin,
   Port_PinDirectionType Direction
)
```

Service ID

0x1

Parameters (in)

- `Pin` – Port pin ID.
- `Direction` – Port pin direction

Parameters (out)

None

Return value

None

DET errors

- *PORT_E_PARAM_PIN* – The port pin ID is invalid.
- *PORT_E_DIRECTION_UNCHANGEABLE* – Port pin is configured as direction unchangeable.
- *PORT_E_UNINIT* – The ports are not initialized, or configuration is invalid.
- *PORT_E_PARAM_INVALID_DIRECTION* – The direction requested is not valid.

DEM errors

None

Description

The service sets the port pin direction during runtime.

7.3.3 Port_RefreshPortDirection

Syntax

```
void Port_RefreshPortDirection
(
   void
)
```

Service ID

0x2

Parameters (in)

None

Parameters (out)

None

Return value

None

DET errors

- *PORT_E_UNINIT* – The ports are not initialized, or configuration is invalid.

DEM errors

None

Description

The service refreshes the directions of pins whose directions are configured as not changeable.

7.3.4 Port_GetVersionInfo

Syntax

```
void Port_GetVersionInfo
(
   Std_VersionInfoType* versioninfo
)
```

Service ID

0x3

Parameters (in)

None

Parameters (out)

- `versioninfo` – Pointer to where to store the version information of this module

Return value

None

DET errors

- *PORT_E_PARAM_POINTER* – Parameter `versioninfo` is a NULL pointer.

DEM errors

None

Description

This service returns the module version and vendor and module ID information of the PORT module.

7.3.5 Port_SetPinMode

Syntax

```
void Port_SetPinMode
(
   Port_PinType     Pin,
   Port_PinModeType Mode
)
```

Service ID

0x4

Parameters (in)

- `Pin` – Port in ID.
- `Mode` – Port pin mode.

Parameters (out)

None

Return value

None

DET Errors

- *PORT_E_PARAM_PIN* – The port pin ID is invalid.
- *PORT_E_UNINIT* – The ports are not initialized, or configuration is invalid.
- *PORT_E_PARAM_INVALID_MODE* – The mode is out of range.
- *PORT_E_MODE_UNCHANGEABLE* – Port pin is configured as mode unchangeable.

DEM Errors

None

Description

This service changes the current mode of the port pin.

7.3.6 Port_GetStatus

Syntax

```
void Port_GetStatus
(
   Port_PinType     Pin,
   Port_StatusType* PortStatusInfoPtr
)
```

Service ID

0xFE

Parameters (in)

- `Pin` – Port pin ID

Parameters (out)

- `PortStatusInfoPtr` – Where to place the port pin status information

Return value

None

DET errors

- *PORT_E_PARAM_POINTER* – Parameter `PortStatusInfoPtr` is a NULL pointer.
- *PORT_E_UNINIT* – The ports are not initialized, or configuration is invalid.
- *PORT_E_PARAM_PIN* – The port pin ID is invalid.
- *PORT_E_REGISTER* – The register value is invalid.

DEM errors

None

Description

This service returns the hardware register status of the specified pin.

7.3.7 Port_GetAmuxSplitCtlStatus

Syntax

```
void Port_GetAmuxSplitCtlStatus
(
   Port_AmuxCellType            Cell,
   Port_AmuxSplitCtlStatusType* AmuxSplitCtlStatusInfoPtr
)
```

Service ID

0xFF

Parameters (in)

- `Cell` –The ID of AMUX splitter cell.

Parameters (out)

- `AmuxSplitCtlStatusInfoPtr` – Where to place the AMUX splitter cell status information.

Return value

None

DET errors

- *PORT_E_UNINIT* – The ports are not initialized, or configuration is invalid.
- *PORT_E_PARAM_POINTER* – The `AmuxSplitCtlStatusInfoPtr` parameter is a NULL pointer.
- *PORT_E_PARAM_CELL* – The ID of AMUX splitter cell is invalid.

DEM errors

None

Description

This service returns the hardware register status of the specified AMUX splitter cell.

7.3.8 Port_SetToDioMode

Syntax

```
void Port_SetToDioMode
(
   Port_PinType     PortId
)
```

Service ID

0xFC

Parameters (in)

- `PortId` – Port pin ID

Parameters (out)

None

Return value

None

DET errors

- *PORT_E_UNINIT* – The ports are not initialized, or configuration is invalid.
- *PORT_E_PARAM_PIN* – The port pin ID is invalid.
- *PORT_E_MODE_UNCHANGEABLE* – Port pin is configured as mode unchangeable.

DEM errors

None

Description

This service switches a port pin from the configured mode to DIO mode (PORT_PIN_MODE_GPIO).

Note: *If the configured mode of a pin is “Dio”, this API has no effect.*

7.3.9 Port_SetToAlternateMode

Syntax

```
void Port_SetToAlternateMode
(
   Port_PinType     PortId
)
```

Service ID

0xFD

Parameters (in)

- `PortId` – Port pin ID

Parameters (out)

None

Return value

None

DET errors

- *PORT_E_UNINIT* – The ports are not initialized, or configuration is invalid.
- *PORT_E_PARAM_PIN* – The port pin ID is invalid.
- *PORT_E_MODE_UNCHANGEABLE* – Port pin is configured as mode unchangeable.

DEM errors

None

Description

This service switches a port pin from DIO mode (PORT_PIN_MODE_GPIO) to the configured mode.

Note: *If the configured mode of a pin is “Dio”, this API has no effect.*

7.3.10 Port_SetTrigger

Syntax

```
void Port_SetTrigger
(
    Port_TriggerGroupIdType     group_id,
    Port_TriggerIdType          out_trg,
    Port_TriggerIdType          in_trg,
    boolean                     inv_flg,
    Port_TriggerSensitiveType   sensitive_type,
    boolean                     dbg_freeze_flg
)
```

Service ID

0xF7

Parameters (in)

- `group_id` – Trigger group ID.
- `out_trg` – Output trigger ID.
- `in_trg` – Input trigger ID if that `group_id` is trigger group. Input trigger type if that `group_id` is trigger 1-to-1 group.
- `inv_flg` – Trigger invert flag.
- `sensitive_type` – Trigger sensitive type.
- `dbg_freeze_flg` – Debug freeze flag.

Parameters (out)

None

Return value

None

DET errors

- *PORT_E_UNINIT* – The ports are not initialized, or configuration is invalid.
- *PORT_E_PARAM_TR_GROUP* – The trigger group ID is invalid.
- *PORT_E_PARAM_TR_OUTPUT* – The output trigger ID is invalid.
- *PORT_E_PARAM_TR_INPUT* – The input trigger ID or input trigger type is invalid.
- *PORT_E_TR_CMD_STATUS* – Trigger command is being activated with the trigger group ID.
- *PORT_E_PARAM_TR_SENSITIVE* – The trigger sensitive type is invalid.
- *PORT_E_TR_MANIPULATION_NOT_PRESENT* – The trigger group does not have trigger manipulation features.

DEM errors

None

Description

This service sets the configuration for specified trigger.

7.3.11 Port_ActTrigger

Syntax

```
boolean Port_ActTrigger
(
    Port_TriggerGroupIdType     group_id,
    Port_TriggerIdType          trg_id,
    Port_TriggerActivationType  act_type,
    Port_TriggerSensitiveType   sensitive_type
)
```

Service ID

0xF8

Parameters (in)

- `group_id` – Trigger group ID
- `trg_id` –Trigger ID
- `act_type` – Trigger activation type
- `sensitive_type` – Trigger sensitive type

Parameters (out)

None

Return value

- *TRUE* - Activation success
- *FALSE* - Activation failure

DET errors

- *PORT_E_UNINIT* – The ports are not initialized, or configuration is invalid.
- *PORT_E_PARAM_TR_GROUP* – The trigger group ID is invalid.
- *PORT_E_PARAM_TR_ACTIVATION* – The trigger activation type is invalid.
- *PORT_E_PARAM_TR_SENSITIVE* – The trigger sensitive type is invalid.
- *PORT_E_PARAM_TR_OUTPUT* –The trigger ID is invalid for output trigger when parameter `act_type` is *PORT_TR_ACTIVATION_OUTPUT*.
- *PORT_E_PARAM_TR_INPUT* – The trigger ID is invalid for input trigger when parameter `act_type` is *PORT_TR_ACTIVATION_INPUT*.
- *PORT_E_TR_CMD_STATUS* – Trigger command is being activated.

DEM errors

None

Description

This service activates the specified trigger.

7.3.12 Port_DeactTrigger

Syntax

```
void Port_DeactTrigger
(
    void
)
```

Service ID

0xF9

Parameters (in)

None

Parameters (out)

None

Return value

None

DET errors

- *PORT_E_UNINIT* – The ports are not initialized, or configuration is invalid.
- *PORT_E_TR_CMD_STATUS* – The trigger command is being deactivated or activated with edge sensitive.

DEM errors

None

Description

This service deactivates the trigger activated by `Port_ActTrigger`.
This function must be called only if the trigger command is successfully activated with the level sensitive.

7.3.13 Port_GetTriggerIdStatus

Syntax

```
void Port_GetTriggerIdStatus
(
    Port_TriggerGroupIdType  group_id,
    Port_TriggerIdType       out_trg,
    Port_TriggerIdStatusType* TrigIdStatusInfoPtr
)
```

Service ID

0xFA

Parameters (in)

- `group_id` – Trigger group ID
- `out_trg` – Output trigger ID

Parameters (out)

- `TrigIdStatusInfoPtr` – Where to place the output trigger status information.

Return value

None

DET errors

- *PORT_E_UNINIT* – The ports are not initialized, or configuration is invalid.
- *PORT_E_PARAM_POINTER* – Parameter `TrigIdStatusInfoPtr` is a NULL pointer.
- *PORT_E_PARAM_TR_GROUP* – The trigger group ID is invalid.
- *PORT_E_PARAM_TR_OUTPUT* – The output trigger ID is invalid.

DEM errors

None

Description

This service returns the hardware register status of the specified output trigger or 1-to-1 output trigger.

7.3.14 Port_GetTriggerCmdStatus

Syntax

```
void Port_GetTriggerCmdStatus
(
    Port_TriggerCmdStatusType* TrigCmdStatusInfoPtr
)
```

Service ID

0xFB

Parameters (in)

None

Parameters (out)

- `TrigCmdStatusInfoPtr` – Where to place the trigger command status information.

Return value

None

DET errors

- *PORT_E_UNINIT* – The ports are not initialized, or configuration is invalid.
- *PORT_E_PARAM_POINTER* – Parameter `TrigCmdStatusInfoPtr` is a NULL pointer.

DEM errors

None

Description

This service gets the status of trigger command.

7.4 Required callback functions

7.4.1 DET

If default error detection is enabled, the PORT driver uses the following callback function that is provided by DET. If you do not use DET, you have to implement this function within your application.

7.4.1.1 Det_ReportError

Syntax

```
Std_ReturnType Det_ReportError
(
    uint16 ModuleId,
    uint8 InstanceId,
    uint8 ApiId,
    uint8 ErrorId
)
```

Reentrancy

Reentrant

Parameters (in)

- `ModuleId` – Module ID of the calling module.
- `InstanceId` – Instance ID of the calling module.
- `ApiId` – ID of the API service that calls this function.
- `ErrorId` – ID of the detected development error.

Return value

Returns always E_OK (is required for services).

Description

Service for reporting development errors.

7.5 Callout functions

7.5.1 Error callout API

The AUTOSAR PORT module requires an error callout handler. Each error is reported to this handler; error checking cannot be switched off. The name of the function to be called can be configured with the `PortErrorCalloutFunction` parameter.

Syntax

```
void Error_Handler_Name
(
    uint16 ModuleId,
    uint8 InstanceId,
    uint8 ApiId,
    uint8 ErrorId
)
```

Reentrancy

Reentrant

Parameters (in)

- `ModuleId` – Module ID of the calling module.
- `InstanceId` – Instance ID of the calling module.
- `ApiId` – ID of the API service that calls this function.
- `ErrorId` – ID of the detected error.

Return value

None

Description

Service for reporting errors.

7.5.2 Callout Port Enter

The service will be called at the start of the area where exclusive control is needed. The name of the function to be called can be configured with the `PortCalloutPortEnter` parameter.

Syntax

```
void Callout_Port_Enter_Name
(
    Port_PortType PortId

)
```

Reentrancy

Reentrant

Parameters (in)

- `PortId` – The ID of a port that needs multicore exclusive access control.

Return value

None

Description

Port driver's environment must implement multicore exclusive access control for the `PortId` if the following APIs are called on several cores to configure the same `PortId`.

- `Port_SetPinDirection`
- `Port_SetPinMode`
- `Port_SetToDioMode`
- `Port_SetToAlternateMode`
- `Port_RefreshPortDirection`

If those APIs will be called only on a one of cores, the callout function body can be empty.

Note that a file that implements the callout function must include *Port.h* to refer the `Port_PortType`.

7.5.3 Callout Port Exit

The service will be called at the end of the area where exclusive control is needed. The name of the function to be called can be configured with the `PortCalloutPortExit` parameter.

Syntax

```
void Callout_Port_Exit_Name
(
    Port_PortType PortId

)
```

Reentrancy

Reentrant

Parameters (in)

- `PortId` – The ID of a port that needs multicore exclusive access control.

Return value

None

Description

Port driver's environment must implement multicore exclusive access control for the `PortId` if the following APIs are called on several cores to configure the same `PortId`.

- `Port_SetPinDirection`
- `Port_SetPinMode`
- `Port_SetToDioMode`
- `Port_SetToAlternateMode`
- `Port_RefreshPortDirection`

If those APIs will be called only on a one of cores, the callout function body can be empty.

Note that a file that implements the callout function must include *Port.h* to refer the `Port_PortType`.

8 Appendix B – Access register table

8.1 HSIOM

Table 22 HSIOM access register table

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
HSIOM_PRT_PORT_SEL0 (Port selection 0)	31:0	Word (32 bits)	Depends on the configuration value or API.	The hardware peripheral connection (pin mode) to I/O pins 0-3.	`Port_Init` `Port_SetPinMode` `Port_SetToDioMode` `Port_SetToAlternateMode` `Port_GetStatus`	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
HSIOM_PRT_PORT_SEL1 (Port selection 1)	31:0	Word (32 bits)	Depends on the configuration value or API.	The hardware peripheral connection (pin mode) to I/O pins 4-7.	`Port_Init` `Port_SetPinMode` `Port_SetToDioMode` `Port_SetToAlternateMode` `Port_GetStatus`	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
HSIOM_AMUX_SPLIT_CTL (AMUX splitter cell control)	31:0	Word (32 bits)	Depends on the configuration value or API.	This register controls the breaking of AMUX buses A and B into multiple segments.	`Port_Init` `Port_GetAmuxSplitCtlStatus`	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)

8.2 GPIO

Table 23 GPIO access register table

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
GPIO_PRT_OUT (Port output data register)	31:0	Word (32 bits)	Depends on the configuration value or API	The output data for the IO pins in the port.	`Port_Init`	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
GPIO_PRT_CFG (Port configuration register)	31:0	Word (32 bits)	Depends on the configuration value or API	Configuration of drive mode and input buffer enable for each pin.	`Port_Init` `Port_RefreshPortDirection` `Port_SetPinDirection` `Port_GetStatus`	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
GPIO_PRT_CFG_IN (Port input buffer configuration register)	31:0	Word (32 bits)	Depends on the configuration value or API	The input buffer for each pin. This register is common for S40S & S40E GPIO pins.	`Port_Init` `Port_GetStatus`	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
GPIO_PRT_CFG_OUT (Port output buffer configuration register)	31:0	Word (32 bits)	Depends on the configuration value or API	The output driver for each pin.	`Port_Init` `Port_GetStatus`	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
GPIO_PRT_CFG_SIO (Port SIO configuration register)	31:0	Word (32 bits)	Depends on the configuration value or API	The features that are specific to SIO ports.	`Port_Init` `Port_GetStatus`	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
GPIO_PRT_CFG_IN_AUTOLVL (Port S40E GPIO input buffer configuration register)	31:0	Word (32 bits)	Depends on the configuration value or API	The S40E GPIO input buffer upper bit for each pin.	`Port_Init` `Port_GetStatus`	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
GPIO_PRT_CFG_OUT2 (Port output buffer configuration register 2)	31:0	Word (32 bits)	Depends on the configuration value or API	The output driver for each pin. This register is for TRAVEO™ T2G 2D cluster series only.	`Port_Init` `Port_GetStatus`	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
CFG_SLEW_EXT (Port output buffer slew extension configuration register)	31:0	Word (32 bits)	Depends on the configuration value or API	Controls the slew rate for each pin. This register is for TRAVEO™ T2G 2D cluster series only.	`Port_Init` `Port_GetStatus`	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)

8 Appendix B – Access register table

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
CFG_DRIVE_EXT0 (Port output buffer drive extension configuration register)	31:0	Word (32 bits)	Depends on the configuration value or API	The output driver for each pin. This register is for TRAVEO™ T2G 2D cluster series only.	`Port_Init` `Port_GetStatus`	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
CFG_DRIVE_EXT1 (Port output buffer drive extension configuration register)	31:0	Word (32 bits)	Depends on the configuration value or API	The output driver for each pin. This register is for TRAVEO™ T2G 2D cluster series only.	`Port_Init` `Port_GetStatus`	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)

8.3 SMART I/O

Table 24 SMARTIO access register table

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
SMARTIO_PRT_CTL (Control register)	31:0	Word (32 bits)	Depends on the configuration value or API	This register controls programmable I/O port.	`Port_Init` `Port_GetStatus`	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
SMARTIO_PRT_SYNC_CTL (Synchronization control register)	31:0	Word (32 bits)	Depends on the configuration value or API	This register controls synchronization setting for each I/O pin.	`Port_Init` `Port_GetStatus`	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
SMARTIO_PRT_LUT_SELx (LUT component input selection for LUT x)	31:0	Word (32 bits)	Depends on the configuration value or API	LUT input signal source for LUT x (x is 0 to 7	`Port_Init` `Port_GetStatus`	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
SMARTIO_PRT_LUT_CTLx (LUT component control register for LUT x)	31:0	Word (32 bits)	Depends on the configuration value or API	Determines the LUT output signal for LUT x (x is 0 to 7)	`Port_Init` `Port_GetStatus`	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
SMARTIO_PRT_DU_SEL (Data unit component input selection)	31:0	Word (32 bits)	Depends on the configuration value or API	Data unit input signal/data source.	`Port_Init` `Port_GetStatus`	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
SMARTIO_PRT_DU_CTL (Data unit component control register)	31:0	Word (32 bits)	Depends on the configuration value or API	This register selects size/width of the data unit data operands.	`Port_Init` `Port_GetStatus`	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
SMARTIO_PRT_DATA (Data register)	31:0	Word (32 bits)	Depends on the configuration value or API	Data unit input data source.	`Port_Init` `Port_GetStatus`	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)

8.4 PERI

Table 25 PERI access register table

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
PERI_TR_CMD (Trigger command)	31:0	Word (32 bits)	Depends on the configuration value or API	This register provides SW control over trigger activation.	`Port_Init` `Port_ActTrigger` `Port_DeactTrigger` `Port_SetTrigger` `Port_GetTriggerCmdStatus`	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
PERI_TR_GR_TR_CTL (Trigger control register)	31:0	Word (32 bits)	Depends on the configuration value or API	This register specifies the input trigger for a specific output trigger.	`Port_Init` `Port_SetTrigger` `Port_GetTriggerIdStatus`	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
PERI_TR_1TO1_GR_TR_CTL (Trigger control register)	31:0	Word (32 bits)	Depends on the configuration value or API	This register specifies 1-to-1 triggers.	`Port_Init` `Port_SetTrigger` `Port_GetTriggerIdStatus`	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)

Revision history

Revision	Issue date	Description of change
**	2020-09-10	New spec.
*A	2020-11-19	Changed a memmap file include folder in section "Memory mapping". Updated definition values in Table 9. Output drive strength. Added description for new configurations in 4.4 Port pin configuration, 4.8 Port default pin configuration. Added configuration table in Table 12, Table 13. Added description for pin doesn't have specific function in 7.1.1 Data types Port_StatusType. Added registers in 8.1.2 GPIO. MOVED TO INFINEON TEMPLATE.
*B	2021-05-18	Added description for pins that do not have specific function in 4.4 Port pin configuration. Removed "reserve" from 7.1.8 Port_AmuxSplitCtlStatusType and 7.1.14 Port_TriggerCmdStatusType. Changed description of 4.7 Port default configuration, 4.8 Port default pin configuration, 4.9 Port default AMUX splitter cell configuration.
*C	2021-12-22	Updated to Infineon style.
*D	2022-06-27	Added the I/O cell type that do not have specific function in Section 4.4, "Port pin configuration". Changed the description in Table 9, "Output drive strength". Changed the description in Table 13, "Output extra drive strength control".
*E	2022-09-27	Added note for `PortPinDirection` parameter description in Port pin configuration section.
*F	2023-10-06	Add limitation for Port5VPinInputBufferMode in 4.4 Port Pin Configuration.
*G	2023-12-08	Web release. No content updates.
*H	2024-03-18	Added note for `PortPinDirection` parameter description in Port pin configuration section.
*I	2024-07-22	Added note for `PortDefPinOutputDrive` parameter description in Port default pin configuration section. Added definition for Resistive pull up and down at the same time for SMC in Port pin status section.

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition 2024-07-22
Published by

Infineon Technologies AG
81726 Munich, Germany

Do you have a question about this document?

Email:

erratum@infineon.com

Document reference

002-30202 Rev. *I

Important notice
The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics (“Beschaffenheitsgarantie”).

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

In addition, any information given in this document is subject to customer’s compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer’s products and any use of the product of Infineon Technologies in customer’s applications.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer’s technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

Warnings
Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies’ products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.