DIO 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 digital input/output (DIO) driver. This guide also explains the functionality of the driver and provides a reference for 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* **[8]** for a detailed description of these topics.

Intended audience

This document is intended for anyone using the MCAL software.

Document structure

The **1 General overview** chapter gives a brief introduction to the DIO driver, explains the embedding of the driver in the AUTOSAR environment and describes the supported hardware and development environment.

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

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

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

Chapter 5 Functional description gives a functional description of all services offered by the DIO driver.

Chapter 6 Hardware resources describes the hardware resources used.

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

Abbreviations and definitions

Table 1 Abbreviations

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.
Channel	Represents a single general purpose digital input/output pin.
Channel Group	A set of adjoining channels, all within a given port, which can be accessed as a single entity by its group name.
DEM	Diagnostic Event Manager
DET	Default Error Tracer

About this document

Abbreviations	Description
DIO	Digital input/output
EB tresos Studio	Elektrobit Automotive configuration framework
HW	Hardware
LSB	Least Significant Bit
MCAL	Microcontroller Abstraction Layer
MCU	Microcontroller Unit
MSB	Most Significant Bit
Port	Represents several DIO Channels that are grouped by hardware, typically controlled by one hardware register.
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 DIO driver, AUTOSAR release 4.2.2.

[3] Specification of PORT driver, AUTOSAR release 4.2.2.

[4] Specification of standard types, AUTOSAR release 4.2.2.

[5] Specification of ECU configuration parameters, AUTOSAR release 4.2.2.

[6] Specification of default error tracer, AUTOSAR release 4.2.2.

[7] Specification of memory mapping, AUTOSAR release 4.2.2.

Elektrobit automotive documentation

Bibliography

[8] EB tresos Studio for ACG8 user’s guide.

Hardware documentation

The hardware documents are listed in the delivery notes.

Related standards and norms

Bibliography

[9] Layered software architecture, AUTOSAR release 4.2.2.

Table of contents

Table of contents

1 General overview

1.1 Introduction to the DIO driver

The DIO driver is a set of software routines that enables the reading and writing of the microcontroller's general-purpose input and output pins. For this, it provides services for modifying the levels of the following:

- DIO channels (Pins)
- DIO ports
- DIO channel groups

The DIO driver is not responsible for initializing or configuring the mode (DIO, PWM, ADC, ...) of the hardware ports. This is the task of the PORT driver. The DIO driver does not provide the `DIO_Init()` function.

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

Furthermore, the DIO driver is delivered with a plugin for the EB tresos Studio, which can be used for static configuration of the driver. The driver also provides an interface to enable ports and channels to define symbolic names, and to create channel groups.

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

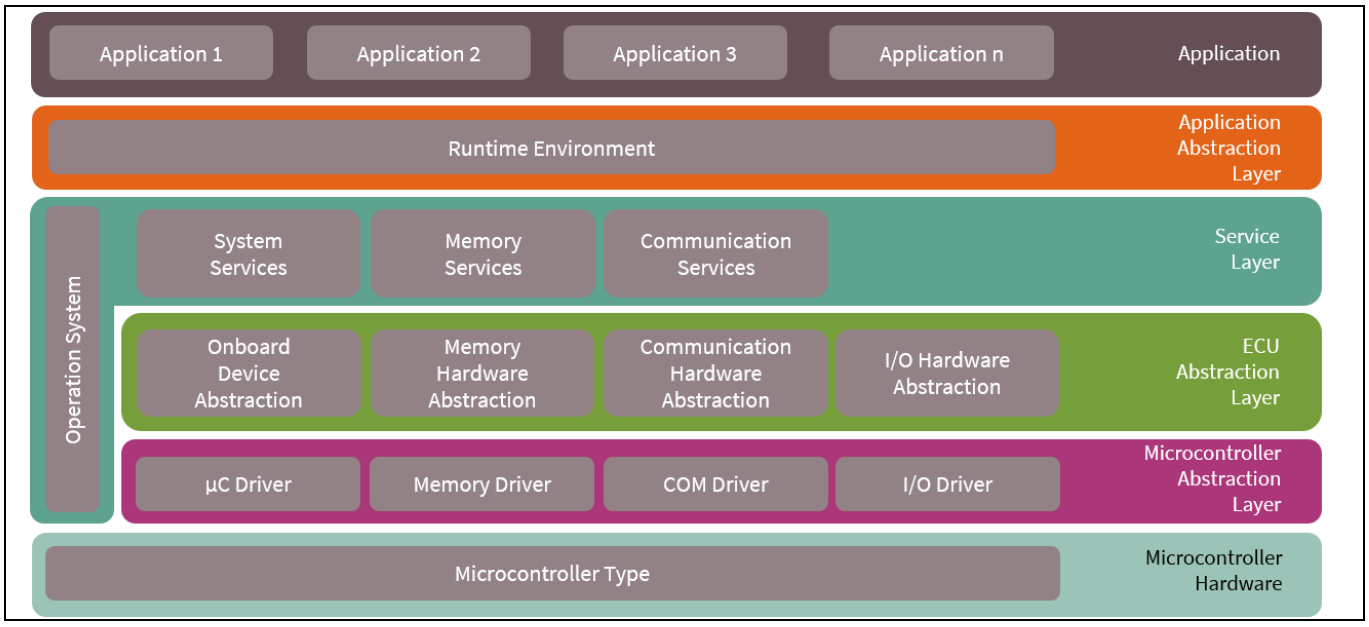

Figure 1 Overview of AUTOSAR software layers

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

As an internal I/O driver, it provides a standardized and µC independent interface to higher software layers for accessing digital input/output pins of the ECU hardware.

For a thorough overview of the AUTOSAR layered software architecture, refer to *Layered software architecture* **[9]**.

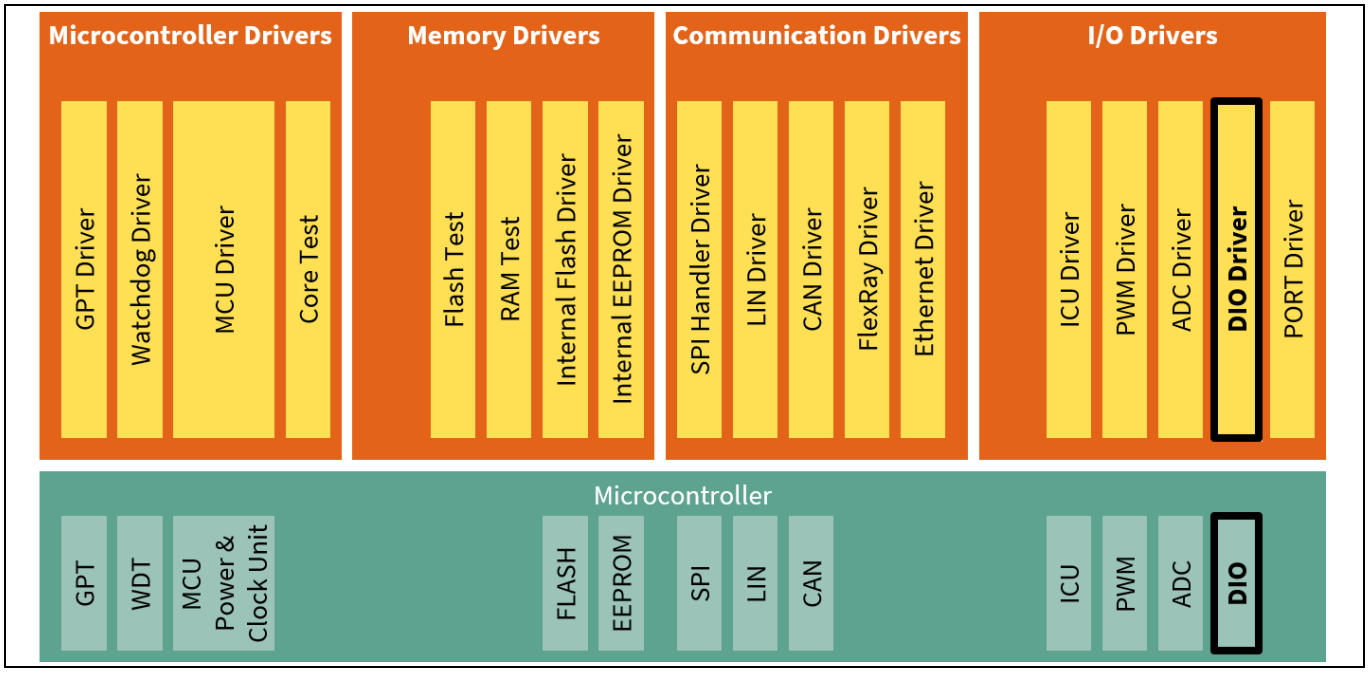

Figure 2 DIO driver in MCAL layer

1.4 Supported hardware

This version of the DIO driver supports the TRAVEO™ T2G microcontroller. Smaller derivatives have only a subset of the ports and pins defined for the microcontroller. The supported derivatives are listed in the release notes.

New derivatives will be supported on request or via resource file updates. For an overview of all supported derivatives, see the Resource plugin. All additional derivatives not mentioned in this user guide are similar to or compatible with the base implementation of this driver.

1.5 Development environment

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

1.6 Character set and encoding

All source code files of the DIO 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 DIO driver supports the multicore type III. For each multicore type, see following sections.

Note: *If multicore type III is desired, the section including the data related to read-only API or atomic write API must be allocated to the memory that can be read from any cores.*

Note: *Functions `Dio_ReadChannel`, `Dio_FlipChannel`, `Dio_ReadChannelGroup`, `Dio_ReadPort` may read incorrect values if the pin direction is changed at the same time on another core (`Port_SetPinDirection`). This scenario is limited to the same pin. Reading and changing of different pins at the same time is not affected.*
It is recommended to execute `Port_SetPinDirection` on the same core that also reads the pin. Otherwise, the application must assure that the functions are never executed in parallel.

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.

Multicore type I has the following characteristics:

- The peripheral channels are accessed by only one core.

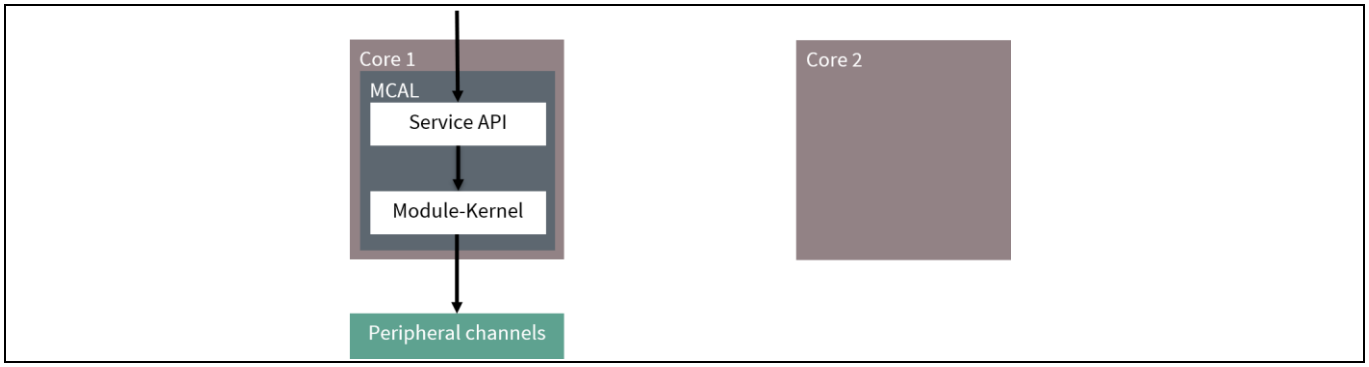

Figure 3 Overview of the multicore type I

1.7.1.2 Core-dependent instances (multicore type II)

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

Multicore type II has the following characteristics:

- The code of the driver is shared among all cores:
  - The common binary is used for all cores.
  - The 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 is called satellite cores.

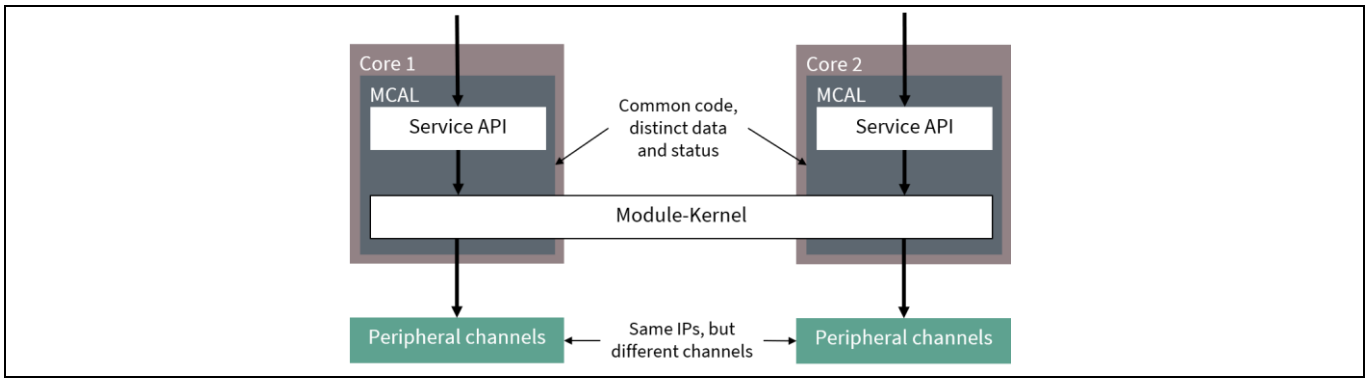

Figure 4 Overview of the multicore type II

1.7.1.3 Core-independent instances (multicore type III)

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

Multicore type III has the following characteristics:

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

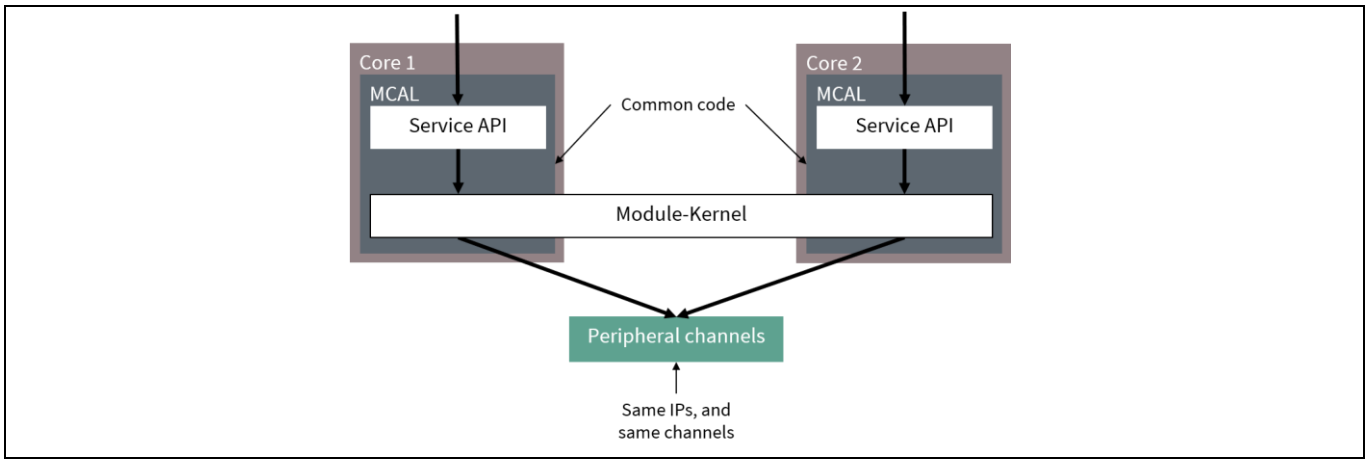

Figure 5 Overview of the multicore type III

2 Using the DIO driver

This chapter describes all the steps necessary to incorporate the DIO driver into your application.

2.1 Installation and prerequisites

*Note: Before you start, see the EB tresos Studio for ACG8 user's guide **[8]** for the following information.*

1. *The installation procedure of EB tresos ECU AUTOSAR components.*
2. *The usage of the EB tresos Studio.*
3. *The usage of the EB tresos ECU AUTOSAR build environment (It includes an explanation of how to set up and integrate your application within the EB tresos ECU AUTOSAR build environment).*

The installation of the DIO 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* **[8]**).

In the following sections, it is assumed that the project is properly set up and uses the application template as described in the *EB tresos Studio for ACG8 user's guide* **[8]**. 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.

All needed port pins must be configured in the PORT driver to use digital input/output functionality of the DIO driver. Check the *Specification of PORT driver* **[3]** for PORT driver configuration.

2.2 Configuring the DIO driver

This section gives a brief overview of the configuration structure defined by AUTOSAR to use the DIO driver.

The following basic containers are used to configure common behavior:

- *DioGeneral*: This container is mainly used to restrict/extend the DIO driver API and enable/disable DET.
- *DioConfig*: This container has the configuration parameters and sub containers of the AUTOSAR DIO driver. It includes the *DioPort* containers.

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

The *DioConfig* container contains the *DioPort* containers, which define the ports including channels and channel groups for each port. Each *DioPort* has the `DioPortId` parameter, which assigns the configured port to the underlying hardware. Refer to each hardware manual for a list of hardware supported ports (refer to hardware documentation for the hardware manual version and subderivative).

For a *DioPort*, many *DioChannels* (single pins) can be configured by setting `DioChannelId`. This value specifies the absolute position of the channel (starting with 0). A DIO pin can be accessed later via its symbolic name and the read/write channel functions of the DIO driver. The number of available pins in the port is described in each hardware manual (refer to hardware documentation for the hardware manual version and subderivative).

Note: At first, configure the `DioChannelPin` to get the correct port pin number easily. Then press the calculate button of the `DioChannelId` parameter in EB tresos Studio to get the corresponding `DioChannelId`.

A `DioChannelGroup` is defined by setting a bit mask in `DioPortMask`. All '1' bits in this consecutive mask define port pins considered in this channel group. The corresponding offset will be calculated automatically during the generation process from the mask value. The number of channel groups within a *DioPort* is not restricted; overlapping groups are allowed. The group belongs to the parent *DioPort* container.

2.2.1 Architecture specifics

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

2.3 Adapting your application

To use the DIO driver in your application, do the following:

Step 1: Include the DIO and the PORT driver header files by adding the following lines to your source file:

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

This publishes all needed types, function prototypes, and symbolic names of the configuration into the application.

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

To do this, declare the error callout function in a file specified by the `DioIncludeFile` parameter and implement it in your application (see **7.1.5 Required callback functions**, error callout API).

The `DioErrorCalloutFunction` parameter can configure the error callout function name.

Step 3: Initialize and configure the ports.

For this example, configure the DIO port as output; otherwise, the DIO driver will exhibit undefined behavior. The PORT driver user guide explains how to configure the port with the PORT driver in the EB tresos Studio.

The port initialization can be done with:

For the master core:

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

The function `Port_Init()` must be called on the master core only.

After that, you can call API functions with the symbolic names defined in your configuration (such as LED_RED_PORT for the DioPort and LED_RED_CHANNEL_2 for bit 2 in this port).

Code Listing 1 Example LED access

```
Dio_WritePort(DioConf_DioPort_LED_RED_PORT, 0xAA);
Dio_LevelType result = Dio_ReadChannel(DioConf_DioChannel_LED_RED_CHANNEL_2);
```

Available API functions and data types are provided in **7.1 API reference** and **5.4 Write services**.

2.4 Starting the build process

To build your application, do the following:

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

The following steps are necessary to measure 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 measuring the stack consumption.*

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. Use the instructions in the release notes to measure the stack consumption.

2.6 Memory mapping

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

2.6.1 Memory allocation keyword

- `DIO_START_SEC_CODE_ASIL_B / DIO_STOP_SEC_CODE_ASIL_B`
  Memory section type is CODE. All executable code is allocated in this section.
- `DIO_START_SEC_CONST_ASIL_B_UNSPECIFIED / DIO_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

3 Structure and dependencies

The DIO driver consists of static, configuration, and generated files.

3.1 Static files

Table 2 Static files

Folder	Description
*$(PLUGIN_PATH)= $(TRESOS_BASE)/plugins/Dio_TS_**	Path to the DIO driver plugin.
$(PLUGIN_PATH)/lib_src	Contains all static source files of the DIO 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)/lib_include	Contains all the internal header files for the DIO driver.
$(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 DIO driver is enabled.
$(PLUGIN_PATH)/include	Basic public include directory needed to include *Dio.h*.
$(PLUGIN_PATH)/autosar	Contains the AUTOSAR ECU parameter definition with adaptations specific to the vendor, architecture, and derivative to create a correct matching parameter configuration for the DIO driver.

3.2 Configuration files

The DIO driver can be configured with EB tresos Studio. When saving a project, the configuration of the DIO driver is saved in the *Dio.xdm* file in the *$(PROJECT_ROOT)/config* directory of your project folder. This file serves as an input for the generation of the 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.

Table 3 Generated files

File	Description
include/Dio_Cfg.h	Provides all symbolic names of the configuration and are included by *Dio.h*.
include/Dio_Cfg_Internal.h	A driver-internal generated file including derivative-specific overall settings. This file is automatically included by the driver but not published via *Dio.h* because the settings are not public.
swcd/Dio_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, follow the menu path **Project** > **Build Project** and select **generate_swcd**.*

3.4 Dependencies

3.4.1 DET

If the default error detection is enabled in the DIO driver configuration, DET needs to be installed, configured, and built into the application.

This driver reports DET error codes as instance 0.

3.4.2 PORT driver

Although the DIO driver can be successfully compiled and linked without an AUTOSAR-compliant PORT driver, the latter is required to configure and initialize all ports. Otherwise, the DIO driver will show undefined behavior. Section **2.3 Adapting your application** explains how you can add the PORT driver to your application.

3.4.3 BSW scheduler

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

- SchM_Enter_Dio_DIO_EXCLUSIVE_AREA_0(void)
- SchM_Exit_Dio_DIO_EXCLUSIVE_AREA_0(void)

Make sure that the BSW scheduler is properly configured and initialized before using the DIO 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 configuration parameter `DioErrorCalloutFunction`.

4 EB tresos Studio configuration interface

The GUI is not part of the current delivery. For further information, see the *EB tresos Studio for ACG8 user's guide* **[8]**.

4.1 General configuration

The module comes preconfigured with the default settings. These settings must be adapted when necessary.

Table 4 Settings

Parameter	Description
`DioDevErrorDetect`	Enables or disables default error notification for the DIO driver. 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 safety mechanisms (fault detection).
`DioVersionInfoApi`	Adds or removes the `Dio_GetVersionInfo()` service to/from the code.
`DioFlipChannelApi`	Adds or removes the `Dio_FlipChannel()` service to/from the code.
`DioMaskedWritePortApi`	Adds or removes the `Dio_MaskedWritePort()` service to/from the code.
`DioErrorCalloutFunction`	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 DIO driver. *Note:* `DioErrorCalloutFunction` *must be a valid C function name; otherwise, an error occurs in the configuration phase.*
`DioIncludeFile`	Lists the files to be included within the driver. Any application-specific symbol (such as the error callout function) that is used by the Dio configuration should be included by configuring this parameter. *Note:* `DioIncludeFile` *must be a file name with a .h extension and a unique name; otherwise, some errors occur during configuration.*

4.2 Port configuration

Table 5 Port configuration

Parameter	Description
`DioPortId`	The port number. For example, the appropriate `DioPortId` for P000_1 or P009_3 are 0 and 9. *Note: Because not all microcontroller ports may be used for DIO, there may be "gaps" in the list of all IDs. This value will be assigned to the following symbolic names:* The symbolic name derived from the *DioPort* container short name prefixed with "*DioConf_DioPort_*".

4.3 Channel configuration

4.3.1 Individual DIO channel configuration

Besides a HW-specific channel name, which is typically fixed for a specific microcontroller, additional symbolic names can be defined per channel.

Note: This container definition does not explicitly define a symbolic name parameter. Instead, the container's short name will be used in the ECU configuration description to specify the symbolic name of the channel.

Table 6 Individual DIO channel configuration

Parameter	Description
`DioChannelId`	The port pin number. This value specifies the absolute position of the channel. For example, to specify pin number 0 of port number 1, set `DioChannelId` as 8. This means that `DioChannelId` is calculated with the following formula: `DioChannelId` of Pxxx_y = (xxx * 8) + y This value will be assigned to the following symbolic names: The symbolic name derived from the *DioChannel* container short name prefixed with "*DioConf_DioChannel_*" .
`DioChannelPin`	The port pin name. For example, the appropriate `DioChannelPin` for P000_1 or P022_7 are P000_1 and P022_7.

4.4 Channel group configuration

A channel group represents several adjoining DIO channels represented by a logical group.

Note: *This container definition does not explicitly define a symbolic name parameter. Instead, the container's short name will be used in the ECU configuration description to specify the symbolic name of the channel group.*

Table 7 Channel group configuration

Parameter	Description
`DioChannelGroupIdentification`	Identified in the DIO API by a pointer to a data structure (of type `Dio_ChannelGroupType`). That data structure contains the channel group information. This parameter contains the code fragment that must be inserted in the API call of the calling module to get the address of the variable in memory, which holds the channel group information. An example value is `&Dio_ChannelGroupCfg[0]`. Only the array index can be changed. This value will be assigned to the following symbolic names: The symbolic name derived from the *DioChannelGroup* container short name prefixed with "*DioConf_DioChannelGroup_*".
`DioPortMask`	A mask that defines the positions of the channel group. The data type depends on the port width. Because it is not possible to identify the position of ChannelGroup, `DioPortMask` cannot be set to 0.
`DioPortOffset`	A position of the channel group on the port, counted from the LSB. `DioPortOffset` will be calculated automatically during the generation process from the mask value. Therefore, `DioPortOffset` is not changeable.

5 Functional description

5.1 Initialization

The DIO driver does not provide a function for port configuration and initialization. This must be done with the PORT driver **[3]** prior to using the DIO driver.

5.2 Runtime reconfiguration

The DIO driver is not runtime configurable. Changing the port pin direction during runtime is done by the PORT driver.

5.3 Channels, ports and channel groups

A channel represents a single general-purpose digital I/O pin. The value of a single channel is of the type `Dio_LevelType` and can either be `STD_HIGH`, which corresponds to physical high, or `STD_LOW`, which corresponds to physical low.

A port represents several channels, which are grouped by hardware and are controlled by one hardware register. The corresponding data type for a value of a port is `Dio_PortLevelType` whose size depends on the width of the largest port. When reading a port with a smaller width, the upper bits are set to '0'. Writing to a port with a smaller width ignores the upper bits.

A channel group consists of several adjoining channels within a port. Its value is also of the type `Dio_PortLevelType`.

A channel group is specified by three parameters: the port it belongs to (the *DioChannelGroup* container is inside the *DioPort* container), a mask, and an offset. The offset is derived from the mask during the generation process and does not need to be configured by the user.

The value of a channel group is aligned to the LSB, that is, the group's read and write services do the shifting and masking (See **Figure 6**).

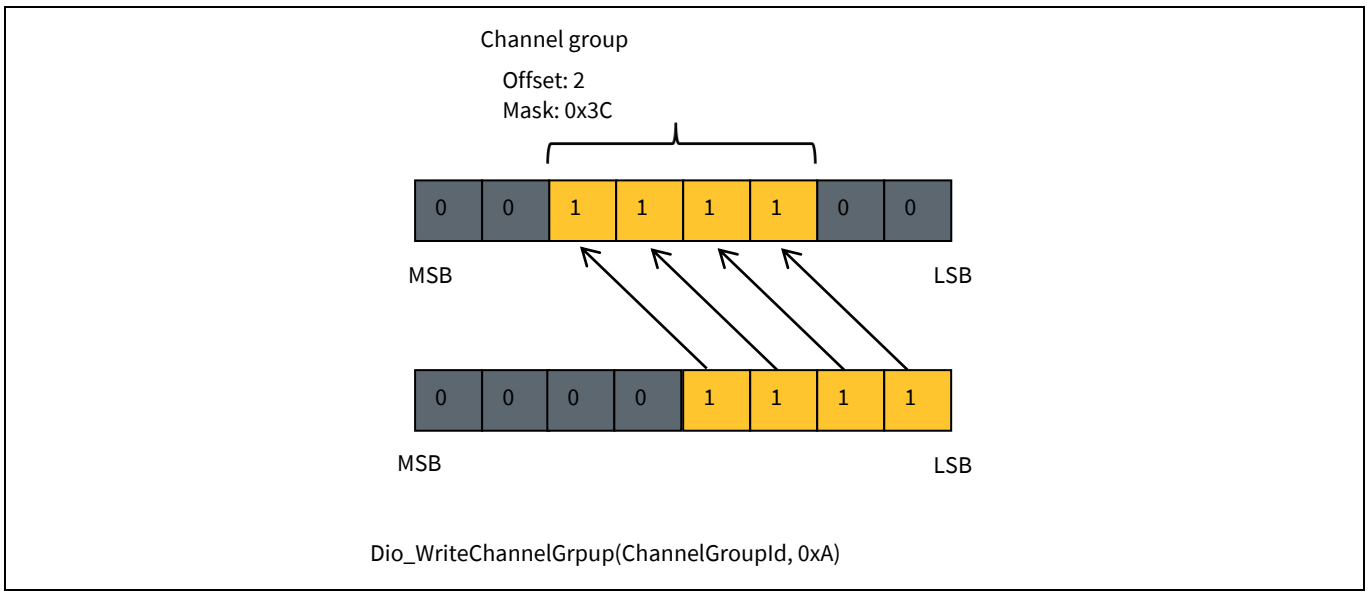

Figure 6 DIO channel groups masking and alignment

5.4 Write services

The DIO driver provides the following synchronous API functions to modify the level of output channels, ports, and groups:

- `void Dio_WriteChannel(Dio_ChannelType ChannelId, Dio_LevelType Level)`
- `void Dio_WritePort(Dio_PortType PortId, Dio_PortLevelType Level)`
- `void Dio_WriteChannelGroup(Dio_ChannelGroupType *ChannelGroupIdPtr, Dio_PortLevelType Level)`
- `Dio_LevelType Dio_FlipChannel(Dio_ChannelType ChannelId)`
- `void Dio_MaskedWritePort(Dio_PortType PortId, Dio_PortLevelType Level, Dio_PortLevelType Mask)`

All channels, ports, and channel groups are referred to by symbolic names, which must be configured. The following example demonstrates how to use the write services:

Code Listing 2 Example Write services

```
#include "Dio.h"
/* ... */
/* initialize ports */
/* .... */
Dio_LevelType c;
Dio_WriteChannel(DioConf_DioChannel_MY_LED_0, STD_HIGH);
Dio_WritePort(DioConf_DioPort_MY_LED_PORT, 0xFF);
Dio_WriteChannelGroup(DioConf_DioChannelGroup_MY_LED_GROUP_0, 0x0);
c = Dio_FlipChannel(DioConf_DioChannel_MY_LED_0);
Dio_MaskedWritePort(DioConf_DioPort_MY_LED_PORT, 0xFF, 0xAA);
```

- Moreover, writing to a channel group does not affect the remaining channels of the port, which are not members of the channel group.

Note: In multicore case, Dio_WriteChannelGroup and Dio_MaskedWritePort have a small time lag between the falling edge and the rising edge.

- When writing to a port with a smaller width than the size of `Dio_PortLevelType`, the upper bits are ignored.

Note: The Port pin output is not affected from writing the value when port pin mode is other than GPIO. The Port pin output is not affected from writing the value during port pin direction is other than output. The written value will be reflected to the output level when then port pin direction is changed to output.

5.5 Read services

For reading channels, ports and channel groups, the driver offers the following synchronous functions:

- `Dio_LevelType Dio_ReadChannel(Dio_ChannelType ChannelId)`
- `Dio_PortLevelType Dio_ReadPort(Dio_PortType PortId)`
- `Dio_PortLevelType Dio_ReadChannelGroup(Dio_ChannelGroupType *ChannelGroupIdPtr)`

Channels, ports, and channel groups are also referred to by the symbolic names configured in the EB tresos Studio. The following example shows how to use the read services:

Code Listing 3 Example Read services

```
#include "Dio.h"
/* ... */
/* initialize ports */
/* .... */
Dio_LevelType c;
Dio_PortLevelType p;
Dio_PortLevelType g;
c = Dio_ReadChannel(DioConf_DioChannel_MY_DIP_SWITCH_0);
p = Dio_ReadPort(DioConf_DioPort_MY_DIP_PORT);
g = Dio_ReadChannelGroup(DioConf_DioChannelGroup_MY_DIP_SWITCH_GROUP_0);
```

- When reading from a port with a smaller width than the size of `Dio_PortLevelType`, the upper bits are set to 0.
- When reading ports with pins, which are not available in hardware, the missing pin levels are always returned as 0.
- When reading pins whose directions are input, actual input levels can be acquired.
- When reading pins whose directions are output and whose input buffers are enabled, actual output levels can be acquired.
- When reading pins whose directions are output and whose input buffers are disabled, output level settings can be acquired.
- When reading pins whose directions are input/output disabled, output level settings can be acquired.

5.6 API parameter checking

The driver's services perform regular error checks.

When an error occurs, the error hook routine (configured via `DioErrorCalloutFunction`) 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 DIO driver:

- The `Dio_GetVersionInfo` function checks whether the `VersionInfo` pointer is NULL. In this case, the error code `DIO_E_PARAM_POINTER` is reported.
- The `Dio_ReadChannel`, `Dio_WriteChannel`, and `Dio_FlipChannel` functions check whether the `ChannelId` parameter is valid. In case of an invalid `ChannelId`, the error code `DIO_E_PARAM_INVALID_CHANNEL_ID` is reported.

Note: *`DIO_E_PARAM_INVALID_CHANNEL_ID` will also be reported when the accessed port pin is not available on the derivative/target.*

- `Dio_WriteChannel` additionally reports `DIO_E_PARAM_INVALID_LEVEL_VALUE` if the given level does not match `STD_HIGH` or `STD_LOW`.
- Dio_ReadPort, Dio_WritePort, Dio_MaskedWritePort, Dio_ReadChannelGroup, and Dio_WriteChannelGroup report the error code DIO_E_PARAM_INVALID_PORT_ID in case of an invalid PortId.
- Finally, for Dio_ReadChannelGroup and Dio_WriteChannelGroup, passing an invalid ChannelGroupIdPtr leads to the error code DIO_E_PARAM_INVALID_GROUP.

Furthermore, in case of an error, all read services return 0 and all write services do not process the write request. This process is the same even if default error detection is disabled.

A complete list of all error codes is given in **7.1.3 Constants**.

5.7 Configuration checking

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

- Channel group mask must fit into the port.
  The channel group mask describes a consecutive number of pins in the port, which make up the channel group. The lower limit for the mask is 1 and the upper limit is 2^(width of the port) - 1.
- Unique user names for DioPort, DioChannels, and DioChannelGroups.
  The DIO driver generates macros of user-configured names for architecture-independent implementation. Ensure that no name is used more than once.
- Architecture-specific port and port pin check.
  Check that the selected `DioPortId` and the configured channels in this port (bit number in this port) are defined for the hardware derivative selected.

5.8 Reentrancy

All services are reentrant, that is, they can be accessed by different upper layers or drivers concurrently.

5.9 Debugging support

The DIO driver does not support debugging.

6 Hardware resources

6.1 Ports and pins

The DIO driver can be used on all general-purpose pins of the TRAVEO™ T2G microcontroller as digital input/output.

Any port can operate in 8 pins or 1 pin units and all registers can be read or written in 1 bit or 32 bits units. For more details, refer to each hardware manual (See hardware documentation for the hardware manual version and subderivative).

6.2 Timer

The DIO driver does not use any hardware timers.

6.3 Interrupts

The DIO driver does not use any interrupts.

7 Appendix A

7.1 API reference

7.1.1 Include files

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

7.1.2 Data types

7.1.2.1 Dio_ChannelType

Type

`uint16`

Description

Type for storing channel IDs. For parameter values of type `Dio_ChannelType`, the symbolic names provided by the configuration description must be used.

7.1.2.2 Dio_PortType

Type

`uint8`

Description

Type for storing port IDs. For parameter values of type `Dio_PortType`, the symbolic names provided by the configuration description must be used.

7.1.2.3 Dio_ChannelGroupType

Type

`typedef struct`

Description

This type defines a struct describing a channel group. A channel group consists of several adjoining channels of the port. The mask contains the bit mask defining all channels of the port which are valid for the channel group. The mask is aligned with the LSB of the port. The offset gives the position of the first set bit in the bit mask, counted from the LSB of the port. See **Figure 6** for an example. For parameter values of type `Dio_ChannelGroupType`, the symbolic names provided by the configuration description must be used.

7.1.2.4 Dio_LevelType

Type

`uint8`

Description

Type for representing the level of a channel.

7.1.2.5 Dio_PortLevelType

Type

`uint8`

Description

Type for storing the values of a port.

7.1.2.6 Dio_ConfigType

Type

`typedef struct`

Description

`Dio_ConfigType` defines a structure that holds the DIO driver configuration set.

7.1.3 Constants

7.1.3.1 Error codes

The service might return the error codes, shown in **Table 8**, if default error detection is enabled:

Table 8 Error codes

Name	Value	Description
`DIO_E_PARAM_INVALID_CHANNEL_ID`	0x0A	Invalid `ChannelId` passed
`DIO_E_PARAM_INVALID_LEVEL_VALUE`	0x0B	Level value is not `STD_HIGH` or `STD_LOW`
`DIO_E_PARAM_CONFIG`	0x10	Invalid `ConfigPtr` passed
`DIO_E_PARAM_INVALID_PORT_ID`	0x14	Invalid `PortId` passed
`DIO_E_PARAM_INVALID_GROUP`	0x1F	Invalid `ChannelGroupIdPtr` passed
`DIO_E_PARAM_POINTER`	0x20	Invalid `VersionInfo` passed

7.1.3.2 Version information

Table 9 lists the version information published in the driver's header file.

Table 9 Version information

Name	Value	Description
DIO_SW_MAJOR_VERSION	Refer to release notes	Software major version number
DIO_SW_MINOR_VERSION	Refer to release notes	Software minor version number
DIO_SW_PATCH_VERSION	Refer to release notes	Software patch version number

7.1.3.3 Module information

Table 10 Module information

Name	Value	Description
DIO_MODULE_ID	120	Module ID
DIO_VENDOR_ID	66	Vendor ID

7.1.3.4 API service IDs

The API Service IDs, listed in Table 11, are published in the Driver's header file.

Table 11 API Service IDs

Name	Value	Description
DIO_API_READ_CHANNEL	0x00	Channel read service
DIO_API_WRITE_CHANNEL	0x01	Channel write service
DIO_API_READ_PORT	0x02	Port read service
DIO_API_WRITE_PORT	0x03	Port write service
DIO_API_READ_CHANNEL_GROUP	0x04	Channel group read service
DIO_API_WRITE_CHANNEL_GROUP	0x05	Channel group write service
DIO_API_FLIP_CHANNEL	0x11	Channel flip service
DIO_API_GET_VERSION_INFO	0x12	Version read service
DIO_API_MASKED_WRITE_PORT	0x20	Port masked write service

7.1.4 Functions

7.1.4.1 Dio_ReadChannel

Syntax

```
Dio_LevelType Dio_ReadChannel
(
   Dio_ChannelType ChannelId
)
```

Service ID

0x00

Parameters (in)

- `ChannelId` - ID of DIO channel.

Parameters (out)

None

Return value

- `STD_HIGH` - The physical level of the pin is high.
- `STD_LOW` - The physical level of the pin is low.

DET errors

- `DIO_E_PARAM_INVALID_CHANNEL_ID` - ChannelId is invalid.

DEM errors

None

Description

Reads and returns the value of the channel specified by the `ChannelId` parameter.

7.1.4.2 Dio_WriteChannel

Syntax

```
void Dio_WriteChannel
(
   Dio_ChannelType ChannelId,
   Dio_LevelType Level
)
```

Service ID

0x01

Parameters (in)

- `ChannelId` - ID of DIO channel.
- `Level` - Value to be written.

Parameters (out)

None

Return value

None

DET errors

- `DIO_E_PARAM_INVALID_CHANNEL_ID` - ChannelId is invalid.
- `DIO_E_PARAM_INVALID_LEVEL_VALUE` - The level value does not match STD_LOW or STD_HIGH.

DEM errors

None

Description

Writes the level value to the channel specified by `ChannelId`.

7.1.4.3 Dio_ReadPort

Syntax

```
Dio_PortLevelType Dio_ReadPort
(
   Dio_PortType PortId
)
```

Service ID

0x02

Parameters (in)

- `PortId` - ID of DIO Port.

Parameters (out)

None

Return value

Value of the port.

DET errors

- `DIO_E_PARAM_INVALID_PORT_ID` - PortId is invalid.

DEM errors

None

Description

Reads and returns the value of the port specified by `PortId`.

7.1.4.4 Dio_WritePort

Syntax

```
void Dio_WritePort
(
   Dio_PortType PortId,
   Dio_PortLevelType Level
)
```

Service ID

0x03

Parameters (in)

- `PortId` - ID of DIO port.
- `Level` - Value to be written

Parameters (out)

None

Return value

None

DET errors

- `DIO_E_PARAM_INVALID_PORT_ID` - The PortId is invalid.

DEM errors

None

Description

Writes the value Level to the port specified by `PortId`.

7.1.4.5 Dio_ReadChannelGroup

Syntax

```
Dio_PortLevelType Dio_ReadChannelGroup
(
   const Dio_ChannelGroupType * ChannelGroupIdPtr
)
```

Service ID

0x04

Parameters (in)

- `ChannelGroupIdPtr` - Pointer to DIO channel group.

Parameters (out)

None

Return value

The value of the channel group.

DET errors

- `DIO_E_PARAM_INVALID_GROUP` - The ChannelGroupIdPtr is invalid.
- `DIO_E_PARAM_INVALID_PORT_ID` - The port referenced in the channel group is not configured.

DEM errors

None

Description

Reads and returns the value of a channel group specified by `ChannelGroupIdPtr`.

7.1.4.6 Dio_WriteChannelGroup

Syntax

```
void Dio_WriteChannelGroup
(
   const Dio_ChannelGroupType *ChannelGroupIdPtr,
   Dio_PortLevelType Level
)
```

Service ID

0x05

Parameters (in)

- `ChannelGroupIdPtr` - Pointer to DIO channel group.
- `Level` - Value to be written.

Parameters (out)

None

Return value

None

DET errors

- `DIO_E_PARAM_INVALID_GROUP` - The ChannelGroupIdPtr is invalid.
- `DIO_E_PARAM_INVALID_PORT_ID` - The port referenced in the channel group is not configured.

DEM errors

None

Description

Writes the value level to the channel group specified by `ChannelGroupIdPtr`.

7.1.4.7 Dio_GetVersionInfo

Syntax

```
void Dio_GetVersionInfo
(
   Std_VersionInfoType* VersionInfo
)
```

Service ID

0x12

Parameters (in)

None

Parameters (out)

- `VersionInfo` - Pointer to where to store the version information of this module.

Return value

None

DET errors

- `DIO_E_PARAM_POINTER` - The VersionInfo is NULL pointer.

DEM errors

None

Description

Returns the module version, vendor, and module ID information of the DIO driver.

7.1.4.8 Dio_FlipChannel

Syntax

```
Dio_LevelType Dio_FlipChannel
(
   Dio_ChannelType ChannelId
)
```

Service ID

0x11

Parameters (in)

- `ChannelId` - ID of DIO channel.

Parameters (out)

None

Return value

- `STD_HIGH` - The physical level of the pin is high.
- `STD_LOW` - The physical level of the pin is low.

DET errors

- `DIO_E_PARAM_INVALID_CHANNEL_ID` - The ChannelId is invalid.

DEM errors

None

Description

Flips (change from 1 to 0 or from 0 to 1) the level of the channel specified by `ChannelId` and returns the level of the channel after flip.

7.1.4.9 Dio_MaskedWritePort

Syntax

```
void Dio_MaskedWritePort
(
   Dio_PortType PortId,
   Dio_PortLevelType Level,
   Dio_PortLevelType Mask
)
```

Service ID

0x20

Parameters (in)

- `PortId` - ID of DIO port.
- `Level` - Value to be written.
- `Mask` - Bit mask to select the pins that shall be modified.

Parameters (out)

None

Return value

None

DET errors

- `DIO_E_PARAM_INVALID_PORT_ID` - The PortId is invalid.

DEM errors

None

Description

Sets a subset of masked bits to a specified level.

7.1.5 Required callback functions

7.1.5.1 DET

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

7.1.5.1.1 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.

7.1.5.2 Callout functions

7.1.5.2.1 Error callout API

The AUTOSAR DIO 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 `DioErrorCalloutFunction` 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.

8 Appendix B

8.1 Access register table

8.1.1 GPIO

Table 12 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.	Output data for the I/O pins in the port.	Dio_ReadChannel Dio_ReadPort Dio_WritePort Dio_MaskedWritePort Dio_ReadChannelGroup Dio_WriteChannelGroup Dio_FlipChannel	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
GPIO_PRT_OUT_CLR (Port output data clear register)	31:0	Word (32 bits)	Depends on the configuration value or API	Clear output data of specific I/O pins in the corresponding port to '0'.	Dio_WriteChannel	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
GPIO_PRT_OUT_SET (Port output data set register)	31:0	Word (32 bits)	Depends on the configuration value or API	Set output data of specific I/O pins in the corresponding port to '1'.	Dio_WriteChannel	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
GPIO_PRT_OUT_INV (Port output data invert register)	31:0	Word (32 bits)	Depends on the configuration value or API	Invert output data of specific I/O pins in the corresponding port.	Dio_FlipChannel	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
`GPIO_PRT_IN` (Port input state register)	31:0	Word (32 bits)	Depends on the current pin status	Read current pin status for I/O pins.	`Dio_ReadChannel` `Dio_ReadPort` `Dio_ReadChannelGroup` `Dio_FlipChannel`	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	Read pin direction for the I/O pins.	`Dio_ReadChannel` `Dio_ReadPort` `Dio_ReadChannelGroup`	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)

Revision history

Revision	Issue date	Description of change
**	2020-09-10	New spec.
*A	2020-11-20	Updated using the DIO driver: Updated memory mapping: Changed a memmap file include folder. Updated to Infineon template.
*B	2021-12-22	Updated to Infineon style.
*C	2023-12-08	Web release. No content updates.

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

Edition 2023-12-08

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email:

erratum@infineon.com

Document reference

002-30201 Rev. *C

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.