

HY-TTC 500

I/O Driver Manual

Document Number: S-TTC5F-G-20-001

Document Version: 3.4.1

Date: 15-Jun-2022

Status: Released

TTControl GmbH

Schoenbrunner Str. 7, A-1040 Vienna, Austria, Tel. +43 1 585 34 34-0, Fax +43 1 585 34 34-90, office@ttcontrol.com

No part of the document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the written permission of TTControl GmbH. Company or product names mentioned in this document may be trademarks or registered trademarks of their respective holders. TTControl GmbH undertakes no further obligation or relation to this document.

Legal Notice

Document Number: S-TTC5F-G-20-001

The data in this document may not be altered or amended without special notification from TTControl GmbH. TTControl GmbH undertakes no further obligation in relation to this document. The software described in it can only be used if the customer is in possession of a general license agreement or single license.

The information contained in this document does not affect or change any General Terms and Conditions of TTControl GmbH and/or any agreements existing between TTControl GmbH and the recipient regarding the product or Sample concerned. The reader acknowledges that this document may not be reproduced, stored in a retrieval system, transmitted, changed, or translated, in whole or in part, without the express prior written consent of TTControl GmbH. The reader acknowledges that any and all of the copyrights, trademarks, trade names, patents (whether registrable or not) and other intellectual property rights embodied in or in connection with this document are and will remain the sole property of TTControl GmbH or the respective right holder. Nothing contained in this legal notice, the document or in any web site of TTControl GmbH shall be construed as conferring to the recipient any license under any intellectual property rights, whether explicit, by estoppel, implication, or otherwise.

The product or Sample is only allowed to be used in the scope as described in this document. TTControl GmbH provides this document "as is" and disclaims all warranties of any kind. The entire risk, as to quality, use or performance of the document remains with the recipient.

All trademarks mentioned in this document belong to their respective owners.

Copyright © 2022 TTControl GmbH. All rights reserved.

Contents

Table of Contents	1
I API Description	2
1 HY-TTC 500 I/O Driver Manual	3
2 Endianness	4
3 I/O Driver Memory Protection	5
3.1 Concept	5
3.1.1 MPU Protection Policies	5
3.1.2 MPU Violations	6
3.2 I/O Driver Requirements	6
3.2.1 I/O Driver Memory Sections	6
3.2.2 Peripherals	7
3.2.3 Usage	7
3.3 How to Use	8
3.3.1 Legacy Implementation	8
3.3.2 Full Solution	9
3.4 Limitations	10
4 Data Structure Index	11
4.1 Data Structures	11
5 File Index	12
5.1 File List	12
6 Data Structure Documentation	14
6.1 bl_apdb_Struct Reference	14
6.1.1 Detailed Description	15
6.1.2 Field Documentation	15
6.2 bl_t_can_id_Struct Reference	19
6.2.1 Detailed Description	20
6.2.2 Field Documentation	20
6.3 bl_t_date_Struct Reference	20
6.3.1 Detailed Description	20
6.3.2 Field Documentation	20
6.4 diag_errorcode_Struct Reference	21
6.4.1 Detailed Description	21
6.4.2 Field Documentation	21
6.5 io_adc_safety_conf_Struct Reference	22
6.5.1 Detailed Description	22
6.5.2 Field Documentation	22
6.6 io_can_data_frame_Struct Reference	23
6.6.1 Detailed Description	23
6.6.2 Field Documentation	23
6.7 io_dio_limits_Struct Reference	24
6.7.1 Detailed Description	24

6.7.2	Field Documentation	24
6.8	io_do_safety_conf_Struct Reference	25
6.8.1	Detailed Description	25
6.8.2	Field Documentation	25
6.9	io_driver_safety_conf_Struct Reference	26
6.9.1	Detailed Description	26
6.9.2	Field Documentation	27
6.10	io_lin_data_frame_Struct Reference	28
6.10.1	Detailed Description	28
6.10.2	Field Documentation	28
6.11	io_pwd_cnt_conf_Struct Reference	29
6.11.1	Detailed Description	29
6.11.2	Field Documentation	29
6.12	io_pwd_cplx_conf_Struct Reference	29
6.12.1	Detailed Description	30
6.12.2	Field Documentation	30
6.13	io_pwd_cplx_safety_conf_Struct Reference	30
6.13.1	Detailed Description	31
6.13.2	Field Documentation	31
6.14	io_pwd_inc_conf_Struct Reference	31
6.14.1	Detailed Description	32
6.14.2	Field Documentation	32
6.15	io_pwd_inc_safety_conf_Struct Reference	32
6.15.1	Detailed Description	32
6.15.2	Field Documentation	33
6.16	io_pwd_pulse_samples_Struct Reference	33
6.16.1	Detailed Description	33
6.16.2	Field Documentation	33
6.17	io_pwd_universal_safety_conf_Struct Reference	34
6.17.1	Detailed Description	34
6.17.2	Field Documentation	34
6.18	io_pwm_current_queue_Struct Reference	35
6.18.1	Detailed Description	35
6.18.2	Field Documentation	35
6.19	io_pwm_safety_conf_Struct Reference	36
6.19.1	Detailed Description	36
6.19.2	Field Documentation	36
7	File Documentation	38
7.1	APDB.h File Reference	38
7.1.1	Detailed Description	39
7.1.2	APDB Code Example	39
7.1.3	Macro Definition Documentation	40
7.1.4	Typedef Documentation	41
7.1.5	Variable Documentation	41

7.2	DIAG_Constants.h File Reference	41
7.2.1	Detailed Description	46
7.2.2	Diagnostic state machine error reporting	46
7.2.3	Diagnostic state machine error codes	46
7.2.4	Macro Definition Documentation	49
7.2.5	Typedef Documentation	74
7.3	DIAG_Functions.h File Reference	75
7.3.1	Detailed Description	76
7.3.2	Function Documentation	77
7.4	IO_ADC.h File Reference	80
7.4.1	Detailed Description	81
7.4.2	ADC input protection	83
7.4.3	ADC Code Examples	83
7.4.4	Macro Definition Documentation	84
7.4.5	Typedef Documentation	85
7.4.6	Function Documentation	86
7.5	IO_CAN.h File Reference	95
7.5.1	Detailed Description	97
7.5.2	Usage of the acceptance masks	97
7.5.3	CAN handles and message objects	97
7.5.4	CAN Code Examples	98
7.5.5	Macro Definition Documentation	100
7.5.6	Typedef Documentation	101
7.5.7	Function Documentation	102
7.6	IO_DEBUG.h File Reference	111
7.6.1	Detailed Description	112
7.6.2	DEBUG Code Examples	112
7.6.3	Macro Definition Documentation	113
7.6.4	Function Documentation	114
7.7	IO_DIO.h File Reference	116
7.7.1	Detailed Description	118
7.7.2	Digital output protection	118
7.7.3	DIO Code Examples	119
7.7.4	Macro Definition Documentation	119
7.7.5	Typedef Documentation	120
7.7.6	Function Documentation	121
7.8	IO_DOWNLOAD.h File Reference	130
7.8.1	Detailed Description	131
7.8.2	DOWNLOAD Code Examples	131
7.8.3	Function Documentation	132
7.9	IO_Driver.h File Reference	133
7.9.1	Detailed Description	135
7.9.2	Basic structure of an application	136
7.9.3	Limitations during startup	136

7.9.4	IO Driver Code Examples	136
7.9.5	Macro Definition Documentation	141
7.9.6	Typedef Documentation	145
7.9.7	Function Documentation	146
7.10	IO_EEPROM.h File Reference	153
7.10.1	Detailed Description	154
7.10.2	Speed of the EEPROM/FRAM Operations	154
7.10.3	EEPROM Code Examples	155
7.10.4	Macro Definition Documentation	156
7.10.5	Function Documentation	157
7.11	IO_Error.h File Reference	159
7.11.1	Detailed Description	162
7.11.2	Macro Definition Documentation	163
7.11.3	Typedef Documentation	183
7.12	IO_FLASH.h File Reference	183
7.12.1	Detailed Description	185
7.12.2	Blocks in the Flash Chip	185
7.12.3	Flash Banks	186
7.12.4	Speed of Flash Operations	186
7.12.5	Flash Code Examples	187
7.12.6	Macro Definition Documentation	189
7.12.7	Function Documentation	189
7.13	IO_LIN.h File Reference	195
7.13.1	Detailed Description	197
7.13.2	LIN Code Examples	197
7.13.3	Macro Definition Documentation	198
7.13.4	Typedef Documentation	199
7.13.5	Function Documentation	199
7.14	IO_MPU.h File Reference	201
7.14.1	Detailed Description	204
7.14.2	MPU initialization	204
7.14.3	MPU access settings	204
7.14.4	MPU fault handling	204
7.14.5	MPU subregions	204
7.14.6	MPU code example	204
7.14.7	Macro Definition Documentation	206
7.14.8	Function Documentation	211
7.15	IO_Pin.h File Reference	214
7.15.1	Detailed Description	223
7.15.2	Macro Definition Documentation	225
7.16	IO_POWER.h File Reference	279
7.16.1	Detailed Description	281
7.16.2	Power Code Examples	282
7.16.3	Macro Definition Documentation	282

7.16.4	Function Documentation	284
7.17	IO_PVG.h File Reference	288
7.17.1	Detailed Description	289
7.17.2	PVG output protection	290
7.17.3	PVG Code Examples	290
7.17.4	Function Documentation	290
7.18	IO_PWD.h File Reference	294
7.18.1	Detailed Description	297
7.18.2	PWD input protection	298
7.18.3	PWD code examples	299
7.18.4	Macro Definition Documentation	299
7.18.5	Typedef Documentation	301
7.18.6	Function Documentation	302
7.19	IO_PWM.h File Reference	320
7.19.1	Detailed Description	321
7.19.2	PWM output protection	321
7.19.3	PWM code examples	322
7.19.4	Macro Definition Documentation	323
7.19.5	Typedef Documentation	323
7.19.6	Function Documentation	323
7.20	IO_RTC.h File Reference	332
7.20.1	Detailed Description	334
7.20.2	RTC Code Examples	335
7.20.3	Macro Definition Documentation	336
7.20.4	Typedef Documentation	336
7.20.5	Function Documentation	336
7.21	IO_UART.h File Reference	341
7.21.1	Detailed Description	343
7.21.2	UART Code Examples	343
7.21.3	Macro Definition Documentation	344
7.21.4	Function Documentation	345
7.22	IO_UDP.h File Reference	348
7.22.1	Detailed Description	349
7.22.2	Limitations	349
7.22.3	Macro Definition Documentation	349
7.22.4	Function Documentation	350
7.23	IO_VOUT.h File Reference	354
7.23.1	Detailed Description	355
7.23.2	VOUT output protection	355
7.23.3	VOUT Code Examples	355
7.23.4	Function Documentation	356
7.24	ptypes_apdb.h File Reference	359
7.24.1	Detailed Description	360
7.24.2	Macro Definition Documentation	361

7.24.3 Function Documentation	361
7.25 ptypes_tms570.h File Reference	361
7.25.1 Detailed Description	363
7.25.2 Macro Definition Documentation	363
7.25.3 Typedef Documentation	364
Index	367
Disposal	367
Legal Disclaimer	368

Part I

API Description

1 HY-TTC 500 I/O Driver Manual

Version

3.4.1

Date

15.06.2022

This document describes the API of the HY-TTC 500 I/O Driver.

The following driver functions are described:

- [Driver main components](#)
- [Error definitions](#)
- [Pin definitions](#)
- [Application database for bootloader](#)

- [DIAG - Diagnostic error reporting](#)
- [DIAG - Diagnostic constants](#)
- [DIAG - Diagnostic functions](#)

- [ADC - Analog to Digital Converter driver](#)
- [CAN - Controller Area Network driver](#)
- [DEBUG - Debugging utility functions](#)
- [DIO - Driver for digital inputs and outputs](#)
- [DOWNLOAD - Driver for Ethernet download](#)
- [EEPROM - External EEPROM/FRAM non-volatile memory driver](#)
- [FLASH - External flash non-volatile memory driver](#)
- [LIN - Local Interconnect Network driver](#)
- [MPU - Memory Protection Unit](#)
- [POWER - Driver for ECU power functions](#)
- [PVG - Proportional Valve output driver](#)
- [PWD - Pulse Width Decode and digital timer input driver](#)
- [PWM - Pulse Width Modulation driver](#)
- [RTC - Real Time Clock driver](#)
- [UDP - User Datagram Protocol communication driver](#)
- [UART - Universal Asynchronous Receiver Transmitter driver](#)
- [VOUT - Voltage Output driver](#)

2 Endianness

The endianness of the ARM Cortex-R4F core of the TI TMS570 CPU is configured to **BE32**. Big-endian systems store the most significant byte of a multi-byte data field in the lowest memory address. Also, the address of the multi-byte data field is the lowest address.

Note

The endianness of the HY-TTC 500 controllers can not be changed.

3 I/O Driver Memory Protection

The concept of memory protection becomes relevant when the user application is partitioned into components with different safety levels. The aim is that the memory reserved for safety critical software components (safe code) cannot be corrupted by faults in non-critical components (unsafe code).

The CPU supports the memory protection on the lowest (hardware) level by means of the Memory Protection Unit (MPU). The I/O Driver includes an interface for a limited control of the MPU.

It should be noted that the support for the memory protection in the I/O Driver is very basic and allows only for rudimentary safety mechanisms. TTControl GmbH offers state of the art memory protection architecture in the SafeRTOS Integration product. It contains a safety-certified operating system that ensures full support for the Freedom from interference, as required by the current safety standards.

3.1 Concept

The MPU defines access permissions to the CPU address space in twelve MPU regions. The IO_MPU module in the I/O Driver allows limited control of four highest-priority regions; the other eight are used to configure the default access privileges to all available areas of the CPU address space, and are invisible to the user.

The IO_MPU module allows these four User MPU regions to be initialized and enabled/disabled from the user application to control the access permissions to the memories or memory-mapped peripherals. The I/O Driver start-up code ensures that all available memory is fully accessible by default. The intended use of the User MPU regions is therefore to restrict (deny) access to sensitive memory areas for unsafe code.

3.1.1 MPU Protection Policies

The I/O Driver uses internal data that is necessary for its correct function. The RAM memory where this data is located should be among the protected memory areas when unsafe code runs, however this interferes with the intended functionality of the I/O Driver. The I/O Driver uses several interrupts that can occur during the execution of the unsafe code and that require access to the I/O Driver data, which cannot work with the memory protection in place. For this reason, the I/O Driver can be configured to deactivate the User MPU regions for the execution of its internal interrupt code. The regions are restored when the interrupt returns so that it has no impact on the user application. The I/O Driver offers three different configurations, called MPU protection policies:

- **IO_MPU_POLICY_REGION0:** The I/O Driver deactivates User MPU region 0 (`IO_MPU_REGION_0`) for its internal tasks. This is the legacy setting that allows using `IO_MPU_REGION_0` for memory protection that may interfere with the I/O Driver. Other User MPU regions can be used for user-specific data but cannot include addresses that are accessed by the I/O Driver.
- **IO_MPU_POLICY_ALLREGIONS:** The I/O Driver deactivates all User MPU regions for its internal tasks. This policy allows all regions to be used freely (provided that the requirements

below are fulfilled), including e.g. the memory-mapped peripherals, which is generally required in safety-critical applications.

- **IO_MPU_POLICY_OFF**: The I/O Driver does not make any changes to the MPU configuration in its internal tasks. This is used when the MPU is fully managed by the user application or controlled by an external software component, e.g. SafeRTOS or CODESYS. This setting can be used if the MPU configuration does not obstruct the execution of the I/O Driver internal tasks in any way.

Attention

The user callbacks and event handlers that are called from the I/O Driver's interrupt code will execute with the same MPU protection state as the I/O Driver code itself, i.e. with the User MPU regions deactivated depending on the selected MPU protection policy. The callbacks include the following:

- The error callback ([DIAG_ERROR_CB](#)) and notification callback ([DIAG_NOTIFY_CB](#)).
- The FPU exception callback ([IO_DRIVER_FPU_HANDLER](#)).
- The RTC periodic event handler ([IO_RTC_EVENT_HANDLER](#)).

3.1.2 MPU Violations

When the CPU attempts to access memory for which it does not have sufficient permissions, one of two CPU Exceptions will occur: Data Abort for data accesses, and Prefetch Abort for instruction fetches. In both cases the application is notified in the Notification callback (with the error code [DIAG_E_DATA_ABORT](#) or [DIAG_E_PREFETCH_ABORT](#)), the unit goes to the Safe state, and as the I/O Driver cannot reliably determine how to recover from the faulty memory access, the application execution does *not* continue.

3.2 I/O Driver Requirements

The MPU defines the CPU's access permissions to the full address space. There are several protection checks built in the IO_MPU API functions that prevent incorrect configuration, though ultimately the user application is in control. The application's MPU configuration can deny access to the memory for the unsafe code as well as the I/O Driver functions, thus preventing the correct functionality of the I/O Driver. This chapter lists the requirements that the user application must follow in order to avoid interfering with the I/O Driver.

3.2.1 I/O Driver Memory Sections

The following requirements ensure that the I/O Driver has sufficient access permissions to the I/O Driver code, constants and data memory. The memory sections are defined in the linker file (.lif) which is part of the I/O Driver release package.

Requirement "Code"

The user application shall ensure that the I/O Driver code has read and execute permissions for the following internal flash memory sections:

- CSM_CODE,
- IO_DRIVER_CODE,
- Exception vectors and the Bootloader memory area (address range 0x0 to 0x0001FFFF).

Requirement "Const"

The user application shall ensure that the I/O Driver code has read permissions for the following memory sections:

- CSM_CONST,
- IO_DRIVER_CONST.

Requirement "Data"

The user application shall ensure that the I/O Driver code has read and write permissions for the following RAM memory sections:

- CSM_VAR_ZERO_INIT_UNSPECIFIED, CSM_VAR_NO_INIT_UNSPECIFIED,
- IO_DRIVER_DATA_NORMAL,
- IO_DRIVER_DATA_COMMON (address range 0x0803FAE0 to 0x0803FEDF),
- Shared Memory area (address range 0x0803FEE0 to 0x0803FFFF).

Attention

Note that any modification to the size and location of the memory section IO_DRIVER_DATA_COMMON or the Shared Memory area is not permitted as they are a part of the interface to the Bootloader.

3.2.2 Peripherals

The MPU can also protect the CPU peripherals which are controlled from memory-mapped registers. In general, unsafe code should have no access to peripherals, however there is one exception: The VIM peripheral is responsible for dispatching CPU interrupts. The VIM registers must be accessible even from unsafe code, otherwise the I/O Driver interrupts will not execute.

Requirement "Peripherals"

The user application shall always grant at least read access permissions to the VIM (Vectored Interrupt Manager) peripheral memory area (address range 0xFFFFFE00 to 0xFFFFFEFF).

3.2.3 Usage

The I/O Driver code executes in the main application thread and in interrupts. The above requirements for the MPU access permissions must be fulfilled in both cases:

In the main thread, the application enables and disables the User MPU regions using the IO_MPU API functions (e.g. [IO_MPU_EnableAll\(\)](#) and [IO_MPU_DisableAll\(\)](#)) as needed to comply with

the I/O Driver requirements and with the safety requirements specific to the application. When a User MPU region is configured to deny access to one of the I/O Driver memory sections, this region has to be disabled from the API before any I/O Driver function can be called.

The behavior in interrupts depends on the selected MPU protection policy. As explained in [MPU Policies](#) above, the I/O Driver automatically disables selected User MPU regions in order to ensure the necessary access permissions. This happens without direct involvement of the user application, i.e. the user's only responsibility is to correctly configure the User MPU regions.

3.3 How to Use

The solution selected to implement the memory protection depends on the safety level of the user application. The legacy approach only protects sensitive data in the RAM. The [IO_MPUPOLICY_ALLREGIONS](#) policy allows for more thorough protection by including e.g. the CPU peripherals.

3.3.1 Legacy Implementation

The original approach allowed protecting the I/O Driver data together with safety-critical data of the user application in one User MPU region [IO_MPUREGION_0](#). This legacy implementation uses the MPU protection policy [IO_MPUPOLICY_REGION0](#).

The I/O Driver data is placed in dedicated data sections (listed in the "[Data requirement](#)") by the linker. Some I/O Driver data sections have a fixed location at the end of the internal RAM. The MPU region that covers these sections can be configured as follows:

```
io_error = IO_MPUMainInit (IO_MPUREGION_0,
                           0x08038000U,
                           IO_MPUSIZE_32_KB,
                           IO_MPUENABLE_ALL_SUBREGIONS,
                           IO_MPUAUTHORITY_READ);
```

This 32 KB MPU region, when used with the recommended linker file, will include all the I/O Driver data and still leave space for the safety-critical data of the user application. The region will be write-protected (i.e. read-only).

The protection is activated when [IO_MPUREGION_0](#) is enabled, and deactivated when disabled:

```
// Enable protection of the I/O Driver RAM.
io_error = IO_MPUMainEnable (IO_MPUREGION_0);

// Unsafe code:
// The I/O Driver RAM (together with other data in the same memory area) is now write protected.
// Do not call any IO_...() functions here!

// Disable protection of the I/O Driver RAM.
io_error = IO_MPUMainDisable (IO_MPUREGION_0);
```

If [IO_MPUREGION_0](#) is used for the I/O Driver data together with the data of the user application and its size is not sufficient, it can be enlarged. In such cases, the start address has to be adjusted accordingly.

The other three User MPU regions can be used for user-specific data but cannot include any I/O Driver data or memory addresses that are accessed by the I/O Driver.

3.3.2 Full Solution

Selecting the `IO_MPUPOLICY_ALLREGIONS` policy allows use of all four User MPU regions for memory protection, regardless of if they cover I/O Driver data, CPU peripherals or other types of memory.

The memory location of user-specific data can be controlled by the linker file. This approach has many benefits over manual placement: Among other things, the linker will reserve a memory area of defined size for the memory region and ensure that its location is aligned according to the size:

```
// Memory section reserved for safe user data.
USER_SAFE_DATA : load=ram, palign=0x400, START(UserSafeData_Start), SIZE(UserSafeData_Size)
```

This memory section can be used in a C source file to store safe data as follows:

```
// Safe user data - placed in the protected memory area.
#pragma DATA_SECTION (safe_data, "USER_SAFE_DATA")
ubyte4 safe_data = 0u;
```

The linker will also export two symbols, `UserSafeData_Start` and `UserSafeData_Size`, which can be used to initialize the User MPU regions. The following code snippet shows an example MPU configuration that protects I/O Driver data, safe user data and all peripherals except the RTI and VIM:

```
extern ubyte4 UserSafeData_Start;
extern ubyte4 UserSafeData_Size;

...
// Configure the MPU protection policy.
io_error = IO_MPUPolicy (IO_MPUPOLICY_ALLREGIONS);

// User MPU region 0: I/O Driver protected data.
io_error = IO_MPUPInit (IO_MPUREGION_0,
                       0x08038000UL,
                       IO_MPUSIZE_32_KB,
                       IO_MPUENABLE_ALL_SUBREGIONS,
                       IO_MPUPACCESS_READ);

// User MPU region 1: Protect all peripherals (no access).
io_error = IO_MPUPInit (IO_MPUREGION_1,
                       0xFC000000UL,
                       IO_MPUSIZE_64_MB,
                       IO_MPUENABLE_ALL_SUBREGIONS,
                       IO_MPUPACCESS_NONE);

// User MPU region 2: Allow access to peripherals RTI + VIM (override IO_MPUREGION_1).
io_error = IO_MPUPInit (IO_MPUREGION_2,
                       0xFFFFF800UL,
                       IO_MPUSIZE_2_KB,
```

```

        IO_MPU_ENABLE_SUBREGION_4 |
        IO_MPU_ENABLE_SUBREGION_6,
        IO_MPU_ACCESS_READ);

// User MPU region 3: Safe user data.
io_error = IO_MPU_Init (IO_MPU_REGION_3,
                        (ubyte4) &UserSafeData_Start,
                        (ubyte4) &UserSafeData_Size,
                        IO_MPU_ENABLE_ALL_SUBREGIONS,
                        IO_MPU_ACCESS_READ);
    
```

Using this MPU configuration is as simple as before:

```

// Enable MPU protection.
io_error = IO_MPU_EnableAll();

// Unsafe code:
// All four User MPU regions are now enabled.
// Do not call any IO_...() functions here!

// Disable MPU protection.
io_error = IO_MPU_DisableAll();
    
```

3.4 Limitations

As briefly mentioned at the beginning, the support for the memory protection is very basic and has several weaknesses. Probably the most severe limitation comes from the fact that without the help of an operating system (OS), pure I/O Driver applications run in the privileged (SVC) mode of the CPU, regardless if the code is safe or unsafe. This means that the unsafe code can reconfigure or even deactivate the MPU – thus disabling whatever protection was in place. The compliance with the safety standards in this case has to be ensured by other means (e.g. a code review).

When unsafe functions are called directly from safe functions, they will share the same stack. The MPU module does not offer any special functions to protect the safe part of the stack from the unsafe code. This protection, if necessary, has to be again achieved using other mechanisms, e.g. calculating the CRC of the safe stack just before and checking it just after calling the unsafe function.

4 Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

bl_apdb_	APDB structure	14
bl_t_can_id_	CAN ID structure	19
bl_t_date_	Date structure	20
diag_errorcode_	Diagnostic error code structure	21
io_adc_safety_conf_	Safety configuration for the ADC inputs	22
io_can_data_frame_	CAN data frame	23
io_dio_limits_	Voltage limits for digital inputs	24
io_do_safety_conf_	Safety configuration for the digital outputs	25
io_driver_safety_conf_	Driver Safety Configuration	26
io_lin_data_frame_	LIN data frame	28
io_pwd_cnt_conf_	Edge counter configuration for the Universal PWD inputs	29
io_pwd_cplx_conf_	Complex configuration for the Universal PWD inputs	29
io_pwd_cplx_safety_conf_	Safety configuration for the Complex PWD inputs	30
io_pwd_inc_conf_	Incremental configuration for the Universal PWD inputs	31
io_pwd_inc_safety_conf_	Safety configuration for the Incremental or Counter PWD inputs	32
io_pwd_pulse_samples_	PWD pulse-width data structure	33
io_pwd_universal_safety_conf_	Safety configuration for the Universal PWD inputs	34
io_pwm_current_queue_	PWM current measurement queue	35
io_pwm_safety_conf_	Safety configuration for the PWM outputs	36

5 File Index

5.1 File List

Here is a list of all files with brief descriptions:

APDB.h	The Application Descriptor Block (APDB)	38
DIAG_Constants.h	Global defines for IO Driver diagnostic module	41
DIAG_Functions.h	Auxiliary functions for watchdog handling and Flash/RAM/CfgFlash correctable errors monitoring	75
IO_ADC.h	IO Driver functions for ADC	80
IO_CAN.h	IO Driver functions for CAN communication	95
IO_DEBUG.h	IO Driver functions for DEBUG utilities	111
IO_DIO.h	IO Driver functions for Digital Input/Output	116
IO_DOWNLOAD.h	IO Driver functions for handling Ethernet download requests	130
IO_Driver.h	High level interface to IO Driver	133
IO_EEPROM.h	IO Driver functions for external EEPROM/FRAM	153
IO_Error.h	Global error defines for IO driver	159
IO_FLASH.h	IO Driver functions for handling the external flash	183
IO_LIN.h	IO Driver functions for LIN communication	195
IO_MPU.h	IO Driver functions for the Memory Protection Unit (MPU)	201
IO_Pin.h	This header file contains pin definitions for the I/O driver, and aliases for the pins	214
IO_POWER.h	IO Driver functions for Power IC control	279
IO_PVG.h	IO Driver functions for PVG channels	288
IO_PWD.h	IO Driver functions for timer input channels	294

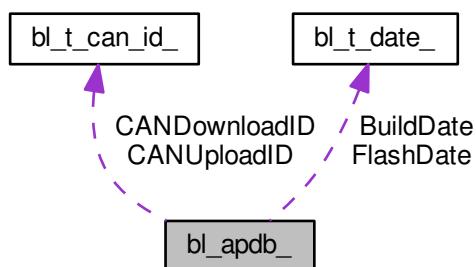
IO_PWM.h	IO Driver functions for PWM channels	320
IO_RTC.h	RTC functions, provides exact timing functions and services	332
IO_UART.h	IO Driver functions for UART communication	341
IO_UDP.h	IO Driver functions for UDP communication	348
IO_VOUT.h	IO Driver functions for voltage outputs	354
ptypes_apdb.h	APDB target abstraction	359
ptypes_tms570.h	Primitive data types	361

6 Data Structure Documentation

6.1 bl_apdb_ Struct Reference

APDB structure.

Collaboration diagram for bl_apdb_:



Data Fields

- ubyte4 ABRDTimeout
- ubyte4 APDBVersion
- ubyte4 ApplicationCRC
- ubyte1 ApplicationID
- ubyte4 ApplicationVersion
- BL_T_DATE BuildDate
- ubyte4 CANBaudrate
- ubyte4 CANChannel
- BL_T_CAN_ID CANDownloadID
- BL_T_CAN_ID CANUploadID
- ubyte4 CodeSize
- ubyte4 CRCSeed
- ubyte4 CRCStartAddress
- ubyte4 DebugKey
- ubyte1 DLMulticastIPAddress [4]
- ubyte4 Flags
- BL_T_DATE FlashDate
- ubyte4 HeaderCRC
- ubyte4 Hook1
- ubyte4 Hook2
- ubyte4 Hook3
- ubyte4 LegacyApplicationCRC
- ubyte4 LegacyHeaderCRC
- ubyte4 MagicSeed

- `ubyte4 MainAddress`
- `ubyte1 ManufacturerID`
- `ubyte4 NodeNumber`
- `ubyte4 NodeType`
- `ubyte4 Password`
- `ubyte2 Reserved`
- `ubyte1 SubnetMask [4]`
- `ubyte1 TargetIPAddress [4]`

6.1.1 Detailed Description

APDB structure.

Data structure for accessing the Application Descriptor Block.

Definition at line 194 of file APDB.h.

6.1.2 Field Documentation

6.1.2.1 `ubyte4 bl_apdb_::ABRDTTimeout`

The timeout for automatic CAN baud rate detection in seconds (HY-TTC 30X family only).

Definition at line 290 of file APDB.h.

6.1.2.2 `ubyte4 bl_apdb_::APDBVersion`

The APDB version (see [APDB_VERSION](#)):

- bit 0-7 ... minor number
- bit 8-15 ... major number

Definition at line 196 of file APDB.h.

6.1.2.3 `ubyte4 bl_apdb_::ApplicationCRC`

CRC-32 value calculated over the application or if a CRC table is used, CRC-32 value calculated over the CRC table (automatically provided by the TTC-Downloader).

Definition at line 220 of file APDB.h.

6.1.2.4 `ubyte1 bl_apdb_::ApplicationID`

The application identifier (must be provided by the application).

Definition at line 297 of file APDB.h.

6.1.2.5 ubyte4 bl_apdb_::ApplicationVersion

The application version (must be provided by the application):

- bit 0-15 ... revision number
- bit 16-23 ... minor number
- bit 24-31 ... major number

Definition at line 258 of file APDB.h.

6.1.2.6 BL_T_DATE bl_apdb_::BuildDate

The application's build date (must be provided by the application).

Definition at line 203 of file APDB.h.

6.1.2.7 ubyte4 bl_apdb_::CANBaudrate

Baud rate in kbit/s used for CAN communication (must be provided by the application).

Definition at line 264 of file APDB.h.

6.1.2.8 ubyte4 bl_apdb_::CANChannel

The channel used for CAN communication (must be provided by the application).

Definition at line 267 of file APDB.h.

6.1.2.9 BL_T_CAN_ID bl_apdb_::CANDownloadID

The CAN identifier used for download direction (TTC-Downloader -> target, must be provided by the application).

Definition at line 247 of file APDB.h.

6.1.2.10 BL_T_CAN_ID bl_apdb_::CANUploadID

The CAN identifier used for upload direction (target -> TTC-Downloader, must be provided by the application).

Definition at line 251 of file APDB.h.

6.1.2.11 ubyte4 bl_apdb_::CodeSize

Code size in bytes (used for CRC calculation) or if a CRC table is used, number of CRC table entries (automatically provided by the TTC-Downloader).

Definition at line 213 of file APDB.h.

6.1.2.12 ubyte4 bl_apdb_::CRCSeed

Seed for application CRC calculation (automatically provided by the TTC-Downloader).

Definition at line 228 of file APDB.h.

6.1.2.13 ubyte4 bl_apdb_::CRCStartAddress

Start address for CRC calculation or if a CRC table is used, start address of the CRC table (automatically provided by the TTC-Downloader).

Definition at line 209 of file APDB.h.

6.1.2.14 ubyte4 bl_apdb_::DebugKey

Debug key for booting the device in debug mode (HY-TTC 500 family only).

Definition at line 287 of file APDB.h.

6.1.2.15 ubyte1 bl_apdb_::DLMulticastIPAddress[4]

Multicast IP address of the TTC-Downloader (most significant byte first, HY-TTC 500 family only).

Definition at line 284 of file APDB.h.

6.1.2.16 ubyte4 bl_apdb_::Flags

Predefined application flags can be specified here. Following flags can be specified:

- bit 0 ... AutoBaudrateDetectionEnable (HY-TTC 30X family only):
 - 0 = disable automatic baud rate detection
 - 1 = enable automatic baud rate detection

Definition at line 231 of file APDB.h.

6.1.2.17 BL_T_DATE bl_apdb_::FlashDate

The date when the application has been flashed (automatically provided by the TTC-Downloader).

Definition at line 200 of file APDB.h.

6.1.2.18 ubyte4 bl_apdb_::HeaderCRC

The CRC value calculated over the whole APDB (automatically provided by the TTC-Downloader).

Definition at line 301 of file APDB.h.

6.1.2.19 ubyte4 bl_apdb_::Hook1

Custom hook 1.

Definition at line 238 of file APDB.h.

6.1.2.20 ubyte4 bl_apdb_::Hook2

Custom hook 2.

Definition at line 239 of file APDB.h.

6.1.2.21 ubyte4 bl_apdb_::Hook3

Custom hook 3.

Definition at line 240 of file APDB.h.

6.1.2.22 ubyte4 bl_apdb_::LegacyApplicationCRC

Legacy application CRC for flash checker (automatically provided by the TTC-Downloader).

Definition at line 217 of file APDB.h.

6.1.2.23 ubyte4 bl_apdb_::LegacyHeaderCRC

Legacy header CRC for flash checker (automatically provided by the TTC-Downloader).

Definition at line 255 of file APDB.h.

6.1.2.24 ubyte4 bl_apdb_::MagicSeed

Seed for CRC calculation with the MCHK HW module (automatically provided by the TTC-Downloader).

Definition at line 275 of file APDB.h.

6.1.2.25 ubyte4 bl_apdb_::MainAddress

The application's start address. Note that the bootloader uses this address to start the application after reset/power-up. See [APPL_START](#) defined in file [ptypes_apdb.h](#) (must be provided by the application).

Definition at line 241 of file APDB.h.

6.1.2.26 ubyte1 bl_apdb_::ManufacturerID

The manufacturer identifier (only required for applications that are flashed during production of the ECU, provided by TTControl GmbH, the generic ID 0xFF can be used for other applications)

Definition at line 293 of file APDB.h.

6.1.2.27 ubyte4 bl_apdb_::NodeNumber

The unique node number used to identify nodes of a CODESYS application. Note that only values from 0 to 127 are allowed (must be provided by the application).

Definition at line 224 of file APDB.h.

6.1.2.28 ubyte4 bl_apdb_::NodeType

The hardware type the application is built for (automatically provided by the TTC-Downloader).

Definition at line 206 of file APDB.h.

6.1.2.29 ubyte4 bl_apdb_::Password

The password hash for memory access. Set this field to 0 or 0xFFFFFFFF to disable password protection. Note that it is highly recommended to set a password upon application download with the TTC-Downloader.

Definition at line 270 of file APDB.h.

6.1.2.30 ubyte2 bl_apdb_::Reserved

Reserved for future use. Shall be set to 0.

Definition at line 300 of file APDB.h.

6.1.2.31 ubyte1 bl_apdb_::SubnetMask[4]

Subnet mask for Ethernet download (most significant byte first, HY-TTC 500 family only).

Definition at line 281 of file APDB.h.

6.1.2.32 ubyte1 bl_apdb_::TargetIPAddress[4]

Target IP address for Ethernet download (most significant byte first, HY-TTC 500 family only).

Definition at line 278 of file APDB.h.

6.2 bl_t_can_id_Struct Reference

CAN ID structure.

Data Fields

- [ubyte4 extended](#)
- [ubyte4 ID](#)

6.2.1 Detailed Description

CAN ID structure.

Definition at line 175 of file APDB.h.

6.2.2 Field Documentation

6.2.2.1 ubyte4 bl_t_can_id_::extended

Type of CAN identifier to be used. Valid values are:

- 0 ... standard CAN identifier is used
- 1 ... extended CAN identifier is used

Definition at line 177 of file APDB.h.

6.2.2.2 ubyte4 bl_t_can_id_::ID

The CAN identifier (LSB must start at bit 0):

- bit 0-10 ... if standard CAN identifier is used
- bit 0-28 ... if extended CAN identifier is used

Definition at line 181 of file APDB.h.

6.3 bl_t_date_ Struct Reference

Date structure.

Data Fields

- [ubyte4 date](#)

6.3.1 Detailed Description

Date structure.

Data structure for saving dates like flash or build date.

Definition at line 158 of file APDB.h.

6.3.2 Field Documentation

6.3.2.1 ubyte4 bl_t_date_::date

Date in format YYYY/MM/DD hh:mm. The fields are defined as followed:

- bit 0-11 ... year

- bit 12-15 ... month
- bit 16-20 ... day
- bit 21-25 ... hour
- bit 26-31 ... minute

Definition at line 160 of file APDB.h.

6.4 diag_errorcode_Struct Reference

Diagnostic error code structure.

Data Fields

- `ubyte1 device_num`
- `ubyte1 error_code`
- `ubyte4 faulty_value`

6.4.1 Detailed Description

Diagnostic error code structure.

Stores all relevant error parameters returned from the diagnostic state machine or returned from the WD. See [Diagnostic state machine error reporting](#) for details about the diagnostic error reporting mechanism.

Definition at line 612 of file DIAG_Constants.h.

6.4.2 Field Documentation

6.4.2.1 `ubyte1 diag_errorcode_::device_num`

The device number which caused the error. This can either be an internal device (see [Diagnostic devices](#)) or an I/O pin (see [Connector pins](#)).

Definition at line 615 of file DIAG_Constants.h.

6.4.2.2 `ubyte1 diag_errorcode_::error_code`

The error code (see [Diagnostic state machine error codes](#)).

Definition at line 614 of file DIAG_Constants.h.

6.4.2.3 `ubyte4 diag_errorcode_::faulty_value`

The value which caused the error

Definition at line 620 of file DIAG_Constants.h.

6.5 io_adc_safety_conf_Struct Reference

Safety configuration for the ADC inputs.

Data Fields

- `ubyte1 adc_val_lower`
- `ubyte1 adc_val_upper`
- `ubyte1 redundant_channel`

6.5.1 Detailed Description

Safety configuration for the ADC inputs.

Stores all relevant safety configuration parameters for the ADC inputs. The internal checker modules verify that this inputs contain valid values

Attention

For the safety configuration of a 2 mode ADC channel, the following rules need to be fulfilled:

- The primary channel needs to specify a redundant channel (`IO_ADC_08 .. IO_ADC_23`) at the `redundant_channel` field.
- The redundant channel must not reference a redundant channel by itself. Thus `IO_PIN_NONE` shall be used for the redundant channel.
- Every redundant channel can only be used once as redundant channel.
- It's not allowed to have a redundant channel which is not used by a primary channel.
- If the ratiometric measurement mode is used, the primary and the redundant channel must use different sensor supplies (but not `IO_SENSOR_SUPPLY_2`).

For the safety configuration of a 3 mode ADC channel, the following rules need to be fulfilled:

- The channel must have `IO_PIN_NONE` in the `redundant_channel` field.
- If the ratiometric measurement mode is used, the channel must not use `IO_SENSOR_SUPPLY_2`.

Definition at line 238 of file `IO_ADC.h`.

6.5.2 Field Documentation

6.5.2.1 `ubyte1 io_adc_safety_conf::adc_val_lower`

Lower ADC limit in % [4..96]

Definition at line 240 of file `IO_ADC.h`.

6.5.2.2 `ubyte1 io_adc_safety_conf::adc_val_upper`

Upper ADC limit in % [4..96]

Definition at line 241 of file `IO_ADC.h`.

6.5.2.3 ubyte1 io_adc_safety_conf_::redundant_channel

Redundant channel for 2 mode inputs.

Definition at line 242 of file IO_ADC.h.

6.6 io_can_data_frame_ Struct Reference

CAN data frame.

Data Fields

- `ubyte1 data [8]`
- `ubyte4 id`
- `ubyte1 id_format`
- `ubyte1 length`

6.6.1 Detailed Description

CAN data frame.

Stores a data frame for the CAN communication.

Definition at line 294 of file IO_CAN.h.

6.6.2 Field Documentation

6.6.2.1 ubyte1 io_can_data_frame_::data[8]

data buffer

Definition at line 296 of file IO_CAN.h.

6.6.2.2 ubyte4 io_can_data_frame_::id

ID for CAN communication

Definition at line 299 of file IO_CAN.h.

6.6.2.3 ubyte1 io_can_data_frame_::id_format

standard or extended format

Definition at line 298 of file IO_CAN.h.

6.6.2.4 ubyte1 io_can_data_frame_::length

number of words in transmit buffer

Definition at line 297 of file IO_CAN.h.

6.7 io_dio_limits_ Struct Reference

Voltage limits for digital inputs.

Data Fields

- `ubyte2 high_thresh1`
- `ubyte2 high_thresh2`
- `ubyte2 low_thresh1`
- `ubyte2 low_thresh2`

6.7.1 Detailed Description

Voltage limits for digital inputs.

Contains the thresholds for valid low- and high-levels for digital inputs.

The range for the low-level is defined by the voltages `low_thresh1` and `low_thresh2`, where `low_thresh1` is the lower limit for a low-level and `low_thresh2` the upper limit.

The range for the high-level is defined by the voltages `high_thresh1` and `high_thresh2`, where `high_thresh1` is the lower limit for a high-level and `high_thresh2` the upper limit.

The value of `low_thresh1` must always be smaller than `low_thresh2` and `high_thresh1` must always be smaller than `high_thresh2`.

The value of `low_thresh2` must always be smaller than `high_thresh1`.

Examples:

```
// voltage limits
IO_DIO_LIMITS limits1 = { 0, 2000, 3000, 5000 };
```

In the above example `limits1` defines the range 0-2000mV as valid low-level and 3000-5000mV as valid high-level.

Note

If no limits will be specified by the application, the following default limits will be applied: { 0, 2500, 2500, 32000 }

Definition at line 148 of file IO_DIO.h.

6.7.2 Field Documentation

6.7.2.1 `ubyte2 io_dio_limits_::high_thresh1`

Defines the lower voltage limit of valid high signal (1mV ... 32000mV)

Definition at line 154 of file IO_DIO.h.

6.7.2.2 `ubyte2 io_dio_limits_::high_thresh2`

Defines the upper voltage limit of valid high signal (1mV ... 32000mV)

Definition at line 156 of file IO_DIO.h.

6.7.2.3 ubyte2 io_dio_limits_::low_thresh1

Defines the lower voltage limit of valid low signal (0mV ... 32000mV)

Definition at line 150 of file IO_DIO.h.

6.7.2.4 ubyte2 io_dio_limits_::low_thresh2

Defines the upper voltage limit of valid low signal (0mV ... 32000mV)

Definition at line 152 of file IO_DIO.h.

6.8 io_do_safety_conf_Struct Reference

Safety configuration for the digital outputs.

Data Fields

- [ubyte1 low_side_channel](#)

6.8.1 Detailed Description

Safety configuration for the digital outputs.

Stores all relevant safety configuration parameters for the digital outputs. The internal checker modules verify that these outputs still work correctly.

Attention

In order to make diagnostics on channels [IO_DO_00 .. IO_DO_15](#) (against open load and short to VBAT) possible, a delay time of at least 20 ms is needed between transitions of the output state. If this timing is not fulfilled, a diagnostic error may get raised.

Definition at line 172 of file IO_DIO.h.

6.8.2 Field Documentation

6.8.2.1 ubyte1 io_do_safety_conf_::low_side_channel

Low side channel. This channel has to be connected together with the specified high side output channel. The low side outputs are grouped as follows:

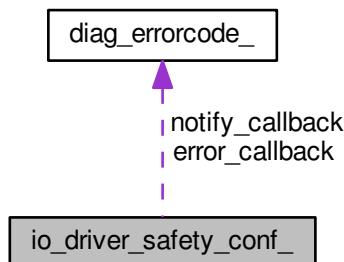
- [IO_DO_08 .. IO_DO_15](#)

Definition at line 174 of file IO_DIO.h.

6.9 io_driver_safety_conf_Struct Reference

Driver Safety Configuration.

Collaboration diagram for io_driver_safety_conf_:



Data Fields

- `ubyte4 command_period`
- `DIAG_ERROR_CB error_callback`
- `ubyte1 glitch_filter_time`
- `DIAG_NOTIFY_CB notify_callback`
- `ubyte1 reset_behavior`
- `ubyte1 window_size`

6.9.1 Detailed Description

Driver Safety Configuration.

This structure is used to pass the configuration for safety critical application to the IO Driver.

Note

Note that the hereby defined values are the configuration values of the TMS570 watchdog configuration values! The processor offers a unique watchdog triggering technic, which does not based on a +/- window manner. In order to support a command period centric +/- watchdog window approach the underlying implementation - due to processor limitations - will distort the window size that was requested through the safety configuration.

To calculate your exact window size (in %) you can use the following formula:

$$\text{actual_window_size} = \{400/(200-\text{choosen_wsize})-2\} * 100$$

where `choosen_wsize` is one of the `SAFETY_CONF_WINDOW_SIZE` below (6.25%,12.5%,25%,etc).

Example, with a windows size choose 25%, the actual size will be 28,57%.

Definition at line 684 of file IO_Driver.h.

6.9.2 Field Documentation

6.9.2.1 ubyte4 io_driver_safety_conf_::command_period

Time in [us], interval between two consecutive software cycles (1000..50000).

Definition at line 689 of file IO_Driver.h.

6.9.2.2 DIAG_ERROR_CB io_driver_safety_conf_::error_callback

Callback function for non-fatal errors. Set this parameter to [NULL](#) to disable this feature.

Definition at line 712 of file IO_Driver.h.

6.9.2.3 ubyte1 io_driver_safety_conf_::glitch_filter_time

Only if an error condition persists after expiration of this time range, an error reaction is taken(1..180 [ms]).

Definition at line 686 of file IO_Driver.h.

6.9.2.4 DIAG_NOTIFY_CB io_driver_safety_conf_::notify_callback

Callback function for fatal errors. Set this parameter to [NULL](#) to disable this feature.

Definition at line 715 of file IO_Driver.h.

6.9.2.5 ubyte1 io_driver_safety_conf_::reset_behavior

Watchdog reset behavior. One of:

- [SAFETY_CONF_RESETS_DISABLED](#)
- [SAFETY_CONF_RESETS_1](#)
- [SAFETY_CONF_RESETS_2](#)
- [SAFETY_CONF_RESETS_3](#)
- [SAFETY_CONF_RESETS_4](#)
- [SAFETY_CONF_RESETS_5](#)
- [SAFETY_CONF_RESETS_6](#)
- [SAFETY_CONF_RESETS_7](#)
- [SAFETY_CONF_RESETS_8](#)
- [SAFETY_CONF_RESETS_9](#)

Definition at line 700 of file IO_Driver.h.

6.9.2.6 ubyte1 io_driver_safety_conf_::window_size

Watchdog window size. One of:

- SAFETY_CONF_WINDOW_SIZE_100_PERCENT
- SAFETY_CONF_WINDOW_SIZE_50_PERCENT
- SAFETY_CONF_WINDOW_SIZE_25_PERCENT
- SAFETY_CONF_WINDOW_SIZE_12_5_PERCENT
- SAFETY_CONF_WINDOW_SIZE_6_25_PERCENT
- SAFETY_CONF_WINDOW_SIZE_3_125_PERCENT

Definition at line 692 of file IO_Driver.h.

6.10 io_lin_data_frame_ Struct Reference

LIN data frame.

Data Fields

- ubyte1 data [8]
- ubyte1 id
- ubyte1 length

6.10.1 Detailed Description

LIN data frame.

Stores a data frame for the LIN communication.

Definition at line 136 of file IO_LIN.h.

6.10.2 Field Documentation

6.10.2.1 ubyte1 io_lin_data_frame_::data[8]

data buffer

Definition at line 140 of file IO_LIN.h.

6.10.2.2 ubyte1 io_lin_data_frame_::id

Frame identifier (0..63)

Definition at line 138 of file IO_LIN.h.

6.10.2.3 ubyte1 io_lin_data_frame_::length

Number of bytes to be read/written (1..8)

Definition at line 139 of file IO_LIN.h.

6.11 io_pwd_cnt_conf_Struct Reference

Edge counter configuration for the Universal PWD inputs.

Data Fields

- `ubyte1 direction`
- `ubyte2 init`
- `ubyte1 mode`

6.11.1 Detailed Description

Edge counter configuration for the Universal PWD inputs.

Stores all relevant configuration parameters for the edge counting mode of Universal PWD inputs.

Definition at line 315 of file IO_PWD.h.

6.11.2 Field Documentation

6.11.2.1 ubyte1 io_pwd_cnt_conf_::direction

Specifies the counting direction

- `IO_PWD_UP_COUNT`: count up
- `IO_PWD_DOWN_COUNT`: count down

Definition at line 322 of file IO_PWD.h.

6.11.2.2 ubyte2 io_pwd_cnt_conf_::init

Initial value of the edge counter [0..65535]

Definition at line 326 of file IO_PWD.h.

6.11.2.3 ubyte1 io_pwd_cnt_conf_::mode

Defines the behavior of the edge counter

- `IO_PWD_RISING_COUNT`: count on a rising edge
- `IO_PWD_FALLING_COUNT`: count on a falling edge
- `IO_PWD_BOTH_COUNT`: count on both edges

Definition at line 317 of file IO_PWD.h.

6.12 io_pwd_cplx_conf_Struct Reference

Complex configuration for the Universal PWD inputs.

Data Fields

- `ubyte1 capture_count`
- `ubyte1 freq_mode`
- `ubyte1 pulse_mode`

6.12.1 Detailed Description

Complex configuration for the Universal PWD inputs.

Stores all relevant configuration parameters for the complex mode of Universal PWD inputs.

Definition at line 291 of file IO_PWD.h.

6.12.2 Field Documentation

6.12.2.1 `ubyte1 io_pwd_cplx_conf_::capture_count`

Number of frequency/pulse-width measurements that will be accumulated [1..8]

Definition at line 304 of file IO_PWD.h.

6.12.2.2 `ubyte1 io_pwd_cplx_conf_::freq_mode`

Specifies the variable edge

- `IO_PWD_RISING_VAR`: rising edge is variable this means, frequency is measured on falling edges
- `IO_PWD_FALLING_VAR`: falling edge is variable this means, frequency is measured on rising edges

Definition at line 298 of file IO_PWD.h.

6.12.2.3 `ubyte1 io_pwd_cplx_conf_::pulse_mode`

Specifies the pulse measurement mode

- `IO_PWD_HIGH_TIME`: configuration to measure pulse-high-time
- `IO_PWD_LOW_TIME`: configuration to measure pulse-low-time
- `IO_PWD_PERIOD_TIME`: configuration to measure period

Definition at line 293 of file IO_PWD.h.

6.13 `io_pwd_cplx_safety_conf_ Struct Reference`

Safety configuration for the Complex PWD inputs.

Data Fields

- `ubyte4 pwd_freq_val_lower`
- `ubyte4 pwd_freq_val_upper`
- `ubyte4 pwd_pulse_val_lower`
- `ubyte4 pwd_pulse_val_upper`

6.13.1 Detailed Description

Safety configuration for the Complex PWD inputs.

Stores all relevant safety configuration parameters for the Complex PWD inputs. The internal checker modules verify that this input is in the valid range.

Definition at line 277 of file IO_PWD.h.

6.13.2 Field Documentation

6.13.2.1 `ubyte4 io_pwd_cplx_safety_conf_::pwd_freq_val_lower`

Lower PWD frequency limit in mHz [100..20000000]

Definition at line 279 of file IO_PWD.h.

6.13.2.2 `ubyte4 io_pwd_cplx_safety_conf_::pwd_freq_val_upper`

Upper PWD frequency limit in mHz [100..20000000]

Definition at line 280 of file IO_PWD.h.

6.13.2.3 `ubyte4 io_pwd_cplx_safety_conf_::pwd_pulse_val_lower`

Lower PWD pulse limit in us [20..10000000]

Definition at line 281 of file IO_PWD.h.

6.13.2.4 `ubyte4 io_pwd_cplx_safety_conf_::pwd_pulse_val_upper`

Upper PWD pulse limit in us [20..10000000]

Definition at line 282 of file IO_PWD.h.

6.14 `io_pwd_inc_conf_Struct Reference`

Incremental configuration for the Universal PWD inputs.

Data Fields

- `ubyte2 init`
- `ubyte1 mode`

6.14.1 Detailed Description

Incremental configuration for the Universal PWD inputs.

Stores all relevant configuration parameters for the incremental mode of Universal PWD inputs.

Definition at line 335 of file IO_PWD.h.

6.14.2 Field Documentation

6.14.2.1 `ubyte2 io_pwd_inc_conf_::init`

Initial value of the incremental counter [0..65535]

Definition at line 346 of file IO_PWD.h.

6.14.2.2 `ubyte1 io_pwd_inc_conf_::mode`

Defines the behavior of the incremental counter

- `IO_PWD_INC_2_COUNT`: Counts up/down on any edge of the two input channels
- `IO_PWD_INC_1_COUNT`: Counts up/down on any edge of the 1st input channel only:
 - count on `IO_PWD_00` for 1st incremental interface
 - count on `IO_PWD_02` for 2nd incremental interface
 - count on `IO_PWD_04` for 3rd incremental interface

Definition at line 337 of file IO_PWD.h.

6.15 `io_pwd_inc_safety_conf_struct` Reference

Safety configuration for the Incremental or Counter PWD inputs.

Data Fields

- `ubyte2 pwd_cnt_val_lower`
- `ubyte2 pwd_cnt_val_upper`

6.15.1 Detailed Description

Safety configuration for the Incremental or Counter PWD inputs.

Stores all relevant safety configuration parameters for the Incremental or counter PWD inputs. The internal checker modules verify that this input is in the valid range.

Definition at line 250 of file IO_PWD.h.

6.15.2 Field Documentation

6.15.2.1 ubyte2 io_pwd_inc_safety_conf_::pwd_cnt_val_lower

Lower PWD counter limit [1..65534]

Definition at line 252 of file IO_PWD.h.

6.15.2.2 ubyte2 io_pwd_inc_safety_conf_::pwd_cnt_val_upper

Upper PWD counter limit [1..65534]

Definition at line 253 of file IO_PWD.h.

6.16 io_pwd_pulse_samples_ Struct Reference

PWD pulse-width data structure.

Data Fields

- `ubyte4 pulse_sample[IO_PWD_MAX_PULSE_SAMPLES]`
- `ubyte1 pulse_samples_count`

6.16.1 Detailed Description

PWD pulse-width data structure.

stores each captured pulse-width for one measurement.

Definition at line 263 of file IO_PWD.h.

6.16.2 Field Documentation

6.16.2.1 ubyte4 io_pwd_pulse_samples_::pulse_sample[IO_PWD_MAX_PULSE_SAMPLES]

stores each captured pulse-width for one measurement

Definition at line 266 of file IO_PWD.h.

6.16.2.2 ubyte1 io_pwd_pulse_samples_::pulse_samples_count

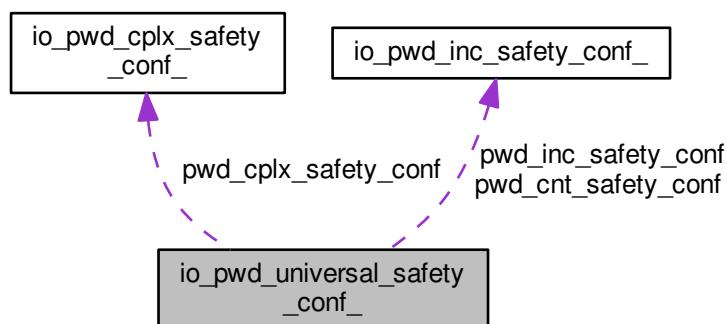
number of pulse_samples

Definition at line 265 of file IO_PWD.h.

6.17 io_pwd_universal_safety_conf_Struct Reference

Safety configuration for the Universal PWD inputs.

Collaboration diagram for `io_pwd_universal_safety_conf_`:



Data Fields

- const `IO_PWD_INC_SAFETY_CONF` * `pwd_cnt_safety_conf`
- const `IO_PWD_CPLX_SAFETY_CONF` * `pwd_cplx_safety_conf`
- const `IO_PWD_INC_SAFETY_CONF` * `pwd_inc_safety_conf`

6.17.1 Detailed Description

Safety configuration for the Universal PWD inputs.

Stores all relevant safety configuration parameters for the Universal PWD inputs. The internal checker modules verify that this input is in the valid range.

Definition at line 356 of file `IO_PWD.h`.

6.17.2 Field Documentation

6.17.2.1 const `IO_PWD_INC_SAFETY_CONF`* `io_pwd_universal_safety_conf_::pwd_cnt_safety_conf`

Safety configuration for the edge counting timer mode.

Definition at line 359 of file `IO_PWD.h`.

6.17.2.2 const IO_PWD_CPLX_SAFETY_CONF* io_pwd_universal_safety_conf_::pwd_cplx_safety_conf

Safety configuration for the complex timer mode.

Definition at line 360 of file IO_PWD.h.

6.17.2.3 const IO_PWD_INC_SAFETY_CONF* io_pwd_universal_safety_conf_::pwd_inc_safety_conf

Safety configuration for the incremental timer mode.

Definition at line 358 of file IO_PWD.h.

6.18 io_pwm_current_queue_Struct Reference

PWM current measurement queue.

Data Fields

- `ubyte1 count`
- `bool overrun`
- `ubyte2 values [IO_PWM_CURRENT_QUEUE_MAX]`

6.18.1 Detailed Description

PWM current measurement queue.

Stores results of the equidistant current measurement.

The queue holds all current measurement since the last retrieval via the step function `IO_PWM_GetCur()` or `IO_PWM_GetCurQueue()`.

Definition at line 162 of file IO_PWM.h.

6.18.2 Field Documentation

6.18.2.1 ubyte1 io_pwm_current_queue_::count

Number of results stored in the queue

Definition at line 164 of file IO_PWM.h.

6.18.2.2 bool io_pwm_current_queue_::overrun

Signal queue overrun.

`TRUE` means queue is full and older measurement results may have been dropped.

`FALSE` means queue is not overrun

Definition at line 165 of file IO_PWM.h.

6.18.2.3 `ubyte2 io_pwm_current_queue_::values[IO_PWM_CURRENT_QUEUE_MAX]`

Buffer holding the measurement values in mA [0 .. 7500].

Oldest value is `values[0]`.

Definition at line 169 of file IO_PWM.h.

6.19 `io_pwm_safety_conf` Struct Reference

Safety configuration for the PWM outputs.

Data Fields

- `ubyte2 current_limit`
- `bool enable_current_check`
- `ubyte1 low_side_channel`

6.19.1 Detailed Description

Safety configuration for the PWM outputs.

Stores all relevant safety configuration parameters for the PWM outputs. The internal checker modules verifies that this inputs contain valid values.

Definition at line 130 of file IO_PWM.h.

6.19.2 Field Documentation

6.19.2.1 `ubyte2 io_pwm_safety_conf_::current_limit`

Current limit in mA [0 .. 7500]. The diagnostic component of the I/O driver will check if the electric current through the load does not exceed the specified limit

Definition at line 135 of file IO_PWM.h.

6.19.2.2 `bool io_pwm_safety_conf_::enable_current_check`

If set to `TRUE` the diagnostic component of the I/O driver will check if the electric current through the load is within the specified limits

Definition at line 132 of file IO_PWM.h.

6.19.2.3 `ubyte1 io_pwm_safety_conf_::low_side_channel`

Low side channel which is connected to the load on the configured PWM channel as a tertiary shut-off path. This channel is switched on and off together with the PWM channel's secondary shut-off path, the safety switch. Only one safety-critical PWM channel or HS digital output can be connected to a low-side channel.

- `IO_PIN_NONE` if the load is connected to the ground
- `IO_DO_08 .. IO_DO_15` if the load is connected to a low side switch

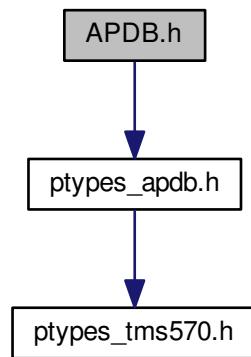
Definition at line 139 of file `IO_PWM.h`.

7 File Documentation

7.1 APDB.h File Reference

The Application Descriptor Block (APDB)

Include dependency graph for APDB.h:



Data Structures

- struct `bl_apdb_`
APDB structure.
- struct `bl_t_can_id_`
CAN ID structure.
- struct `bl_t_date_`
Date structure.

Macros

- #define `APDB_VERSION` 0x00000206UL

Typedefs

- typedef struct `bl_apdb_BL_APDB`
APDB structure.
- typedef struct `bl_t_can_id_BL_T_CAN_ID`
CAN ID structure.
- typedef struct `bl_t_date_BL_T_DATE`

Date structure.

APDB flags

Defined APDB flags.

- #define APDB_FLAGS_ABRD_ENABLE 0x00000001UL
- #define APDB_FLAGS_CRC64_ENABLE 0x40000000UL
- #define APDB_FLAGS_MULTI_APP 0x80000000UL

Application Descriptor Block (APDB)

The APDB, which is defined by the user.

- volatile const BL_APDB Apdb_t

7.1.1 Detailed Description

The Application Descriptor Block (APDB)

Contains the definition for the application descriptor block. This block contains information about a certain application, such as connection settings for CAN and Ethernet, application CRC and the application entry point.

The bootloader needs this information to determine whether or not an application is valid and where the application actually starts. For this reason the field `BL_APDB.MainAddress` must be provided by the application. Note that some fields are automatically provided by the TTC-Downloader.

APDB Usage:

- [Example for APDB definition](#)

7.1.2 APDB Code Example

Example for using the APDB

7.1.2.1 Example for APDB definition in an application

```
volatile const BL_APDB Apdb_t =
{
    APDB_VERSION,           // APDB version
    {0},                   // Flash date (provided by the TTC-Downloader)
    // Build date
    {(((RTS_TTC_FLASH_DATE_YEAR) & 0xFF) << 0) |
     (((RTS_TTC_FLASH_DATE_MONTH) & 0x0F) << 12) |
     (((RTS_TTC_FLASH_DATE_DAY) & 0x1F) << 16) |
     (((RTS_TTC_FLASH_DATE_HOUR) & 0x1F) << 21) |
     (((RTS_TTC_FLASH_DATE_MINUTE) & 0x3F) << 26)}},
    0,                     // Node type (provided by the TTC-Downloader)
    0,                     // CRC start address (provided by the TTC-Downloader)
    0,                     // Code size (provided by the TTC-Downloader)
    0,                     // Legacy application CRC (provided by the TTC-Downloader)
```

```

0,                                // Application CRC (provided by the TTC-Downloader)
1,                                // Node number
0,                                // CRC seed (provided by the TTC-Downloader)
0,                                // Flags
0,                                // Hook 1
0,                                // Hook 2
0,                                // Hook 3
APPL_START,                        // Start address, i.e., application entry point
{0, 1},                            // CAN download ID (standard format, ID 0x1)
{0, 2},                            // CAN upload ID (standard format, ID 0x2)
0,                                // Legacy header CRC (provided by the TTC-Downloader)
0,                                // Application version (major.minor.revision)
(((ubyte4)REVISION_NUMBER) << 0) |
(((ubyte4) MINOR_NUMBER) << 16) |
(((ubyte4) MAJOR_NUMBER) << 24)),
500,                               // CAN baud rate in kbps
0,                                // CAN channel
0,                                // Password (disable password protection)
0,                                // Magic seed (provided by the TTC-Downloader)
{ 10, 100, 30, 200 },             // Target IP address (HY-TTC 500 family only)
{255, 255, 0, 0},                // Subnet mask (HY-TTC 500 family only)
{239, 0, 0, 1},                  // Multicast IP address (HY-TTC 500 family only)
0,                                // Debug key
0,                                // Automatic baud rate detection timeout in seconds
// (HY-TTC 30X family only)
0x00,                             // Manufacturer ID (only required for applications that
// are flashed during production of the ECU, provided by
// TTControl GmbH, the generic ID 0xFF can be used for
// other applications)
0x00,                             // Application ID (only required for applications that
// are flashed during production of the ECU, provided by
// TTControl GmbH)
{0},                               // Reserved, must be set to zero
0                                 // Header CRC (provided by the TTC-Downloader)
};


```

7.1.3 Macro Definition Documentation

7.1.3.1 #define APDB_FLAGS_ABRD_ENABLE 0x00000001UL

Enables automatic baudrate detection at start-up (HY-TTC 30X family only). Access mode: read/write.

Definition at line 120 of file APDB.h.

7.1.3.2 #define APDB_FLAGS_CRC64_ENABLE 0x40000000UL

Indicates whether or not CRC-64 is used for application CRC. Access mode: read only.

Definition at line 127 of file APDB.h.

7.1.3.3 #define APDB_FLAGS_MULTI_APP 0x80000000UL

Indicates whether or not the application is distributed over multiple (incoherent) application regions. Access mode: read only.

Definition at line 134 of file APDB.h.

7.1.3.4 #define APDB_VERSION 0x00000206UL

Current APDB version is version 2.6. This value should be set for field [BL_APDB.APDBVersion](#).

Definition at line 102 of file APDB.h.

7.1.4 Typedef Documentation

7.1.4.1 `typedef struct bl_apdb_ BL_APDB`

APDB structure.

Data structure for accessing the Application Descriptor Block.

7.1.4.2 `typedef struct bl_t_can_id_ BL_T_CAN_ID`

CAN ID structure.

7.1.4.3 `typedef struct bl_t_date_ BL_T_DATE`

Date structure.

Data structure for saving dates like flash or build date.

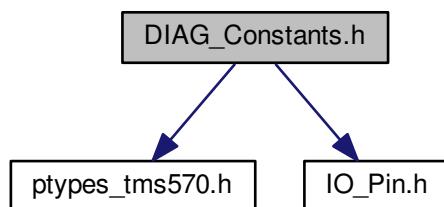
7.1.5 Variable Documentation

7.1.5.1 `volatile const BL_APDB Apdb_t`

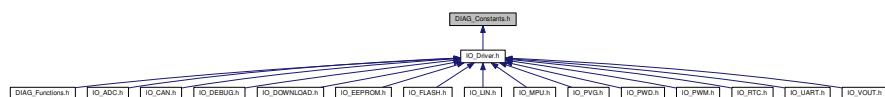
7.2 DIAG_Constants.h File Reference

Global defines for IO Driver diagnostic module.

Include dependency graph for DIAG_Constants.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct `diag_errorcode`
Diagnostic error code structure.

Typedefs

- typedef `ubyte2(* DIAG_ERROR_CB)(ubyte1 diag_state, ubyte1 watchdog_state, DIAG_ER-
RORCODE *const error)`
Callback function for non-fatal errors.
- typedef struct `diag_errorcode_` `DIAG_ERRORCODE`
Diagnostic error code structure.
- typedef `void(* DIAG_NOTIFY_CB)(ubyte1 diag_state, ubyte1 watchdog_state, DIAG_ER-
RORCODE *const error)`
Callback notification function for fatal errors.

States of the diagnostic state machine

Diagnostic state information returned by the function `DIAG_Status()`

- #define `DIAG_STATE_DISABLED` 0x00UL
- #define `DIAG_STATE_INIT` 0x03UL
- #define `DIAG_STATE_CONFIG` 0x05UL
- #define `DIAG_STATE_MAIN` 0x06UL
- #define `DIAG_STATE_SAFE` 0x09UL

States of the watchdog CPU

Watchdog state information returned by the function `DIAG_Status()`

- #define `DIAG_WD_STATE_STANDBY` 0x01U
- #define `DIAG_WD_STATE_RESET` 0x03U
- #define `DIAG_WD_STATE_DIAGNOSTIC` 0x05U
- #define `DIAG_WD_STATE_ACTIVE` 0x06U
- #define `DIAG_WD_STATE_SAFE` 0x09U
- #define `DIAG_WD_STATE_UNKNOWN` 0xFFU

Diagnostic state machine error values

These errors codes are used by the function `DIAG_Status()` and by the notify and error callbacks in the parameter `error_code` of the structure `DIAG_ERRORCODE`.

- #define `DIAG_E_NOERROR` 0U
- #define `DIAG_E_ADC_3MODE_SWITCH_TEST` 1U
- #define `DIAG_E_ADC_3MODE_SWITCH_PERIODIC` 2U
- #define `DIAG_E_ADC_2MODE_RED_CHANNEL_TEST` 3U
- #define `DIAG_E_ADC_SR_CONF_CHECK` 4U
- #define `DIAG_E_ADC_RANGE` 5U
- #define `DIAG_E_ADC_UBAT` 6U
- #define `DIAG_E_ADC_BOARD_TEMP` 7U
- #define `DIAG_E_ADC_SENSOR_SUPPLY` 8U
- #define `DIAG_E_ADC_2V5_REF` 9U
- #define `DIAG_E_ADC_1V2` 10U
- #define `DIAG_E_ADC_VPGATE` 11U
- #define `DIAG_E_PWM_SHORT_CIRCUIT` 12U
- #define `DIAG_E_PWM_OPEN_LOAD` 13U
- #define `DIAG_E_PWM_FEEDBACK` 14U
- #define `DIAG_E_PWM_CURRENT` 15U
- #define `DIAG_E_DO_SHORT_CIRCUIT` 16U
- #define `DIAG_E_DO_OPEN_LOAD` 17U
- #define `DIAG_E_DO_FEEDBACK` 18U
- #define `DIAG_E_PWD_RANGE` 19U
- #define `DIAG_E_PWD_CURRENT` 20U
- #define `DIAG_E_PWD_THRESH` 21U
- #define `DIAG_E_SSW_TEST` 22U
- #define `DIAG_E_SSW_PERIODIC` 23U
- #define `DIAG_E_SSW_EXT_SHUTOFF` 24U
- #define `DIAG_E_VMON_TEST` 25U
- #define `DIAG_E_VMON_PERIODIC` 26U
- #define `DIAG_E_ENABLE_TREE_TEST` 27U
- #define `DIAG_E_WD_INIT` 28U
- #define `DIAG_E_WD_ACTIVATION` 29U
- #define `DIAG_E_WD_TRIGGER` 30U
- #define `DIAG_E_INIT_CORE_L2L3` 31U
- #define `DIAG_E_INIT_CORE_EFUSE_ECC` 32U
- #define `DIAG_E_INIT_CORE_FLASH_WR_ECC` 33U
- #define `DIAG_E_INIT_CORE_STC_TEST` 34U
- #define `DIAG_E_INIT_CORE_SELFTEST` 35U
- #define `DIAG_E_INIT_CORE_PSCON_SELFTEST` 36U
- #define `DIAG_E_INIT_CORE_PBIST_TEST` 37U
- #define `DIAG_E_INIT_CORE_RAM_ECC_B0` 38U
- #define `DIAG_E_INIT_CORE_RAM_ECC_B1` 39U
- #define `DIAG_E_INIT_CORE_FLASH_DATA_ECC` 40U
- #define `DIAG_E_INIT_CORE_CFG_FLASH_ECC` 41U
- #define `DIAG_E_INIT_CORE_IOMM_LOCK` 42U
- #define `DIAG_E_INIT_CORE_CCM_SELFTEST` 43U
- #define `DIAG_E_INIT_CORE_ADD_DECODE_B0` 44U
- #define `DIAG_E_INIT_CORE_ADD_DECODE_B1` 45U
- #define `DIAG_E_INIT_CORE_DCC1_SELFTEST` 46U
- #define `DIAG_E_INIT_CORE_DCC2_SELFTEST` 47U
- #define `DIAG_E_INIT_CORE_ERROR_PIN_TEST` 48U
- #define `DIAG_E_INIT_CORE_MPU_TEST` 49U
- #define `DIAG_E_INIT_CORE_RAM_PARITY_TEST` 50U

- #define **DIAG_E_INIT_CORE_RAM_PBIST** 51U
- #define **DIAG_E_CORE_READBACK** 52U
- #define **DIAG_E_APPL_SAFE_STATE** 53U
- #define **DIAG_E_DRIVER_INIT** 54U
- #define **DIAG_E_DATA_ABORT** 55U
- #define **DIAG_E_PREFETCH_ABORT** 56U
- #define **DIAG_E_UNDEF_INSTRUCTION** 57U
- #define **DIAG_E_ESM_HLI** 58U
- #define **DIAG_E_ESM_LLI** 59U
- #define **DIAG_E_ESM_LLI_CALLBACK** 60U
- #define **DIAG_E_PARITY_FALLBACK** 61U
- #define **DIAG_E_INVALID_DIAG_STATE** 62U
- #define **DIAG_E_INVALID_IRQ** 63U
- #define **DIAG_E_PRG_OVERFLOW** 64U
- #define **DIAG_E_MAIN_LOOP** 65U
- #define **DIAG_E_INIT_CORE_IOMM_PROT_TEST** 66U
- #define **DIAG_E_INIT_CORE_PLL1_SLIP_TEST** 67U
- #define **DIAG_E_INIT_CORE_PLL2_SLIP_TEST** 68U
- #define **DIAG_E_INIT_CORE_OSC_FAIL_TEST** 69U
- #define **DIAG_E_INIT_CORE_ADD_BUS_PAR_B0** 70U
- #define **DIAG_E_INIT_CORE_ADD_BUS_PAR_B1** 71U
- #define **DIAG_E_INIT_CORE_FLASH_BUS1_PAR** 72U
- #define **DIAG_E_INIT_CORE_DMA_BASIC_TEST** 73U
- #define **DIAG_E_ERROR_CALLBACK_RECUSION** 74U

Diagnostic devices

Additional device defines.

- #define **DIAG_DEV_SAFETY_SW_VP** IO_INT_PIN_SAFETY_SW_VP
- #define **DIAG_DEV_PWD** IO_INT_PIN_PWD
- #define **DIAG_DEV_REF_2V5** IO_INT_PIN_REF_2V5
- #define **DIAG_DEV_1V2** IO_INT_PIN_1V2
- #define **DIAG_DEV_VMON** IO_INT_PIN_VMON
- #define **DIAG_DEV_ESM** 133U
- #define **DIAG_DEV_VIM** 134U
- #define **DIAG_DEV_ADC** 135U
- #define **DIAG_DEV_SPI** 136U
- #define **DIAG_DEV_DIO** 137U
- #define **DIAG_DEV_RTC** 138U
- #define **DIAG_DEV_DMA** 139U
- #define **DIAG_DEV_NHET** 140U
- #define **DIAG_DEV_ESM_MIBADC2_PARITY** 141U
- #define **DIAG_DEV_ESM_DMA_MPV_VIOLATION** 142U
- #define **DIAG_DEV_ESM_DMA_PARITY** 143U
- #define **DIAG_DEV_ESM_DMA_DMM_IMPR_READ** 144U
- #define **DIAG_DEV_ESM_N2HET1_2_PARITY** 145U
- #define **DIAG_DEV_ESM_HET_TU1_2_PARITY** 146U
- #define **DIAG_DEV_ESM_PLL1_SLIP** 147U
- #define **DIAG_DEV_ESM_CLOCK_MONITOR** 148U
- #define **DIAG_DEV_ESM_DMA_DMM_IMPR_WRITE** 149U
- #define **DIAG_DEV_ESM_VIM_RAM_PARITY** 150U
- #define **DIAG_DEV_ESM_MIBSPI1_PARITY** 151U
- #define **DIAG_DEV_ESM_MIBSPI3_PARITY** 152U
- #define **DIAG_DEV_ESM_MIBADC1_PARITY** 153U

- #define **DIAG_DEV_ESM_CPU_SLFTST** 154U
- #define **DIAG_DEV_ESM_DCC1_ERROR** 155U
- #define **DIAG_DEV_ESM_CCM_R4_SLFTST** 156U
- #define **DIAG_DEV_ESM_FMC_CFG_FLASH_UNC_ERR** 157U
- #define **DIAG_DEV_ESM_IOMM_MUX_CONFIG** 158U
- #define **DIAG_DEV_ESM_PWR_DOM_CNTL_COMP** 159U
- #define **DIAG_DEV_ESM_PWR_DOM_CNTL_SLFTST** 160U
- #define **DIAG_DEV_ESM_E_FUSE_CNTL** 161U
- #define **DIAG_DEV_ESM_E_FUSE_CNTL_SLFTST** 162U
- #define **DIAG_DEV_ESM_PLL2_SLIP** 163U
- #define **DIAG_DEV_ESM_DCC2** 164U
- #define **DIAG_DEV_ESM_CCMR4_COMPARE** 165U
- #define **DIAG_DEV_ESM_FMC_B1_UNC_ERR** 166U
- #define **DIAG_DEV_ESM_RAM_B0_UNC_ERR** 167U
- #define **DIAG_DEV_ESM_RAM_B1_UNC_ERR** 168U
- #define **DIAG_DEV_ESM_RAM_B0_ADDR_PARITY** 169U
- #define **DIAG_DEV_ESM_RAM_B1_ADDR_PARITY** 170U
- #define **DIAG_DEV_ESM_FLASH_ECC_LIVE_LOCK** 171U
- #define **DIAG_DEV_ESM_RTI_WDD_NMI** 172U
- #define **DIAG_DEV_ESM_E_FUSE_AUTOLOAD** 173U
- #define **DIAG_DEV_ESM_RAM_B0_ECC_UNC_ERR** 174U
- #define **DIAG_DEV_ESM_RAM_B1_ECC_UNC_ERR** 175U
- #define **DIAG_DEV_ESM_FMC_B1_B2_UNC_ERR** 176U
- #define **DIAG_DEV_ESM_UNKNOWN** 177U
- #define **DIAG_DEV_2MODE_CONF_0** 178U
- #define **DIAG_DEV_2MODE_CONF_1** 179U
- #define **DIAG_DEV_2MODE_CONF_2** 180U
- #define **DIAG_DEV_2MODE_CONF_3** 181U
- #define **DIAG_DEV_2MODE_CONF_4** 182U
- #define **DIAG_DEV_2MODE_CONF_5** 183U
- #define **DIAG_DEV_2MODE_CONF_6** 184U
- #define **DIAG_DEV_2MODE_CONF_7** 185U
- #define **DIAG_DEV_PWD_CONF_0** 186U
- #define **DIAG_DEV_PWD_CONF_1** 187U
- #define **DIAG_DEV_PWD_CONF_2** 188U
- #define **DIAG_DEV_PWD_CONF_3** 189U
- #define **DIAG_DEV_PWD_CONF_4** 190U
- #define **DIAG_DEV_PWD_CONF_5** 191U
- #define **DIAG_DEV_DO_CONF_0** 192U
- #define **DIAG_DEV_DO_CONF_1** 193U
- #define **DIAG_DEV_DO_CONF_2** 194U
- #define **DIAG_DEV_EXT_SHUTOFF_0** 195U
- #define **DIAG_DEV_EXT_SHUTOFF_1** 196U
- #define **DIAG_DEV_EXT_SHUTOFF_2** 197U
- #define **DIAG_DEV_WATCHDOG_CPU** 198U
- #define **DIAG_DEV_MAIN_CPU** 199U
- #define **DIAG_DEV MCU** 200U
- #define **DIAG_DEV_MAX** 201U
- #define **DIAG_DEV_NONE IO_PIN_NONE**

Error callback reaction

Allowed return values of error callback function. Except for **DIAG_ERR_NOACTION** and **DIAG_ERR_SAFESTATE** all values may be combined (OR- bitmask)

- #define **DIAG_ERR_NOACTION** 0x0001U
- #define **DIAG_ERR_SAFESTATE** 0x0002U
- #define **DIAG_ERR_DISABLE_SSW0** 0x0004U
- #define **DIAG_ERR_DISABLE_SSW1** 0x0008U
- #define **DIAG_ERR_DISABLE_SSW2** 0x0010U
- #define **DIAG_ERR_DISABLE_HS00** 0x0020U
- #define **DIAG_ERR_DISABLE_HS01** 0x0040U
- #define **DIAG_ERR_DISABLE_HS02** 0x0080U
- #define **DIAG_ERR_DISABLE_HS03** 0x0100U
- #define **DIAG_ERR_DISABLE_HS04** 0x0200U
- #define **DIAG_ERR_DISABLE_HS05** 0x0400U
- #define **DIAG_ERR_DISABLE_HS06** 0x0800U
- #define **DIAG_ERR_DISABLE_HS07** 0x1000U

7.2.1 Detailed Description

Global defines for IO Driver diagnostic module.

This header file defines the Error Codes for the diagnostic and watchdog modules.

7.2.2 Diagnostic state machine error reporting

Details about the error reporting of the diagnostic state machine.

Errors of the diagnostic state machine are either returned by the **DIAG_Status()** function or by the callback functions (**DIAG_ERROR_CB** and **DIAG_NOTIFY_CB**).

Each reported error consists of an error code, a device number and a faulty value (see **DIAG_ERRORCODE**). The device number specifies the source of the error. This can either be an internal device (see [Diagnostic devices](#)) or an I/O pin (see [Connector pins](#)). The faulty value depends on the reported error code (see [Diagnostic state machine error codes](#) for a list of error codes).

7.2.3 Diagnostic state machine error codes

Details about the errors of the diagnostic state machine.

The following table gives an overview of the error codes and their types.

There are **four types of errors**:

- **Persistent fatal errors**

If a fatal error is detected, the diagnostic state machine always activates the safe state. The application will be informed about the error before a reset occurs (via the notification callback, see [DIAG_NOTIFY_CB](#)).

- **Persistent non-fatal errors**

For non-fatal errors the application can decide which action to take (via the error callback, see [DIAG_ERROR_CB](#)).

- **Temporary fatal errors**

For these errors a glitch filter (de-bounce) is implemented, which means that they are reported after the configured anti-glitch time. After the glitch filter time has expired, the error will be treated like a persistent fatal error.

- **Temporary non-fatal errors**

For these errors a glitch filter (de-bounce) is implemented, which means that they are reported after the configured anti-glitch time. After the glitch filter time has expired, the error will be treated like a persistent non-fatal error.

No.	Name	Fatal?	Temporary?	Short description
0	DIAG_E_NOERROR			No error
1	DIAG_E_ADC_3MODE_SWITCH_TEST	no	no	Start-up test error with ADC mode selection switches
2	DIAG_E_ADC_3MODE_SWITCH_PERIODIC	no	yes	Run-time test error with ADC mode selection switches
3	DIAG_E_ADC_2MODE_RED_CHANNEL_TEST	no	no	Redundant channel start-up test error of 2 mode ADCs
4	DIAG_E_ADC_SR_CONF_CHECK	yes	no	Internal shift register readback error
5	DIAG_E_ADC_RANGE	no	yes	ADC measurement range check error
6	DIAG_E_ADC_UBAT	yes	yes	Battery voltage out of range
7	DIAG_E_ADC_BOARD_TEMP	yes	yes	Board temperature out of range
8	DIAG_E_ADC_SENSOR_SUPPLY	no	yes	Sensor supply voltage out of range
9	DIAG_E_ADC_2V5_REF	yes	yes	Internal 2V5 reference voltage out of range
10	DIAG_E_ADC_1V2	yes	yes	Internal 1V2 supply voltage out of range
11	DIAG_E_ADC_VPGATE	yes	yes	Internal VP Gate voltage out of range
12	DIAG_E_PWM_SHORT_CIRCUIT	no	no	PWM start-up test detected a short circuit
13	DIAG_E_PWM_OPEN_LOAD	no	no	PWM start-up test detected an open load
14	DIAG_E_PWM_FEEDBACK	no	yes	PWM pulse or period feedback out of range
15	DIAG_E_PWM_CURRENT	no	yes	PWM current check detected an overcurrent
16	DIAG_E_DO_SHORT_CIRCUIT	no	no	DO start-up test detected a short circuit
17	DIAG_E_DO_OPEN_LOAD	no	no	DO check detected an open load
18	DIAG_E_DO_FEEDBACK	no	yes	DO feedback value out of range
19	DIAG_E_PWD_RANGE	no	yes	PWD counter value, frequency or pulse width out of range
20	DIAG_E_PWD_CURRENT	no	yes	PWD current check error
21	DIAG_E_PWD_THRESH	no	yes	PWD threshold analog feedback out of range
22	DIAG_E_SSW_TEST	no	no	Safety switch start-up test error
23	DIAG_E_SSW_PERIODIC	no	yes	Safety switch periodic check error
24	DIAG_E_SSW_EXT_SHUTOFF	no	yes	Safety switch external shut-off redundancy check error
25	DIAG_E_VMON_TEST	yes	no	Internal voltage monitor start-up test error
26	DIAG_E_VMON_PERIODIC	yes	yes	Internal voltage monitor periodic check error
27	DIAG_E_ENABLE_TREE_TEST	yes	no	Enable tree start-up test error
28	DIAG_E_WD_INIT	yes	no	Watchdog initialization failed
29	DIAG_E_WD_ACTIVATION	yes	no	Watchdog activation failed
30	DIAG_E_WD_TRIGGER	yes	no	Watchdog triggering failed
31	DIAG_E_INIT_CORE_L2L3	yes	no	L2L3 interconnect test error
32	DIAG_E_INIT_CORE_EFUSE_ECC	yes	no	eFuse controller test error
33	DIAG_E_INIT_CORE_FLASH_WR_ECC	yes	no	Flash Module Controller - ECC logic test error

No.	Name	Fatal?	Temporary?	Short description
34	DIAG_E_INIT_CORE_STC_TEST	yes	no	CPU self-test controller test error
35	DIAG_E_INIT_CORE_SELF-TEST	yes	no	CPU self-test error
36	DIAG_E_INIT_CORE_PSCON_SELFTEST	yes	no	Diagnostic power state controller test error
37	DIAG_E_INIT_CORE_PBIST_TEST	yes	no	Programmable built-in self-test controller test error
38	DIAG_E_INIT_CORE_RAM_ECC_B0	yes	no	CPU ECC logic for accesses to TCRAM B0 test error
39	DIAG_E_INIT_CORE_RAM_ECC_B1	yes	no	CPU ECC logic for accesses to TCRAM B1 test error
40	DIAG_E_INIT_CORE_FLASH_DATA_ECC	yes	no	CPU ECC logic accesses to program flash test error
41	DIAG_E_INIT_CORE_CFG_FLASH_ECC	yes	no	CPU ECC logic accesses to configuration flash test error
42	DIAG_E_INIT_CORE_IOMM_LOCK	yes	no	IOMM Lock test error
43	DIAG_E_INIT_CORE_CCM_SELFTEST	yes	no	CCM-R4F module test error
44	DIAG_E_INIT_CORE_ADD_DECODE_B0	yes	no	TCRAM B0 address redundant comparator test error
45	DIAG_E_INIT_CORE_ADD_DECODE_B1	yes	no	TCRAM B1 address redundant comparator test error
46	DIAG_E_INIT_CORE_DCC1_SELFTEST	yes	no	DCC1 self-test error
47	DIAG_E_INIT_CORE_DCC2_SELFTEST	yes	no	DCC2 self-test error
48	DIAG_E_INIT_CORE_ERROR_PIN_TEST	yes	no	Error pin test error during start-up
49	DIAG_E_INIT_CORE_MPU_TEST	yes	no	MPU test error during start-up
50	DIAG_E_INIT_CORE_RAM_PARITY_TEST	yes	no	RAM parity test error during start-up
51	DIAG_E_INIT_CORE_RAM_PBIST	yes	no	RAM self-test error during start-up
52	DIAG_E_CORE_READBACK	yes	no	Configuration read back error
53	DIAG_E_APPL_SAFE_STATE	yes	no	Application requested to activate the safe state
54	DIAG_E_DRIVER_INIT	yes	no	Error detected during the execution of <code>IO_Driver_Init()</code>
55	DIAG_E_DATA_ABORT	yes	no	CPU generated a data abort
56	DIAG_E_PREFETCH_ABORT	yes	no	CPU generated a prefetch abort
57	DIAG_E_UNDEF_INSTRUCTION	yes	no	CPU generated an undefined instruction exception
58	DIAG_E_ESM_HLI	yes	no	ESM exception generated - fatal High Level Interrupt
59	DIAG_E_ESM_LLI	yes	no	ESM exception generated - fatal Low Level Interrupt
60	DIAG_E_ESM_LLI_CALLBACK	no	no	ESM exception generated - non-fatal Low Level Interrupt

No.	Name	Fatal?	Temporary?	Short description
61	DIAG_E_PARITYIY_FALLBACK	yes	no	CPU generated an interrupt vector parity fallback excepti
62	DIAG_E_INVALID_DIAG_STATE	yes	no	Invalid state for diagnostic state machine
63	DIAG_E_INVALID_IRQ	yes	no	Invalid interrupt detected
64	DIAG_E_PRG_OVERFLOW	yes	no	NHET program overflow error
65	DIAG_E_MAIN_LOOP	yes	no	The application returned from the <code>main()</code> function
66	DIAG_E_INIT_CORE_IOMM_PROT_TEST	yes	no	IOMM lock test error
67	DIAG_E_INIT_CORE_PLL1_SLIP_TEST	yes	no	PLL1 slip test error
68	DIAG_E_INIT_CORE_PLL2_SLIP_TEST	yes	no	PLL2 slip test error
69	DIAG_E_INIT_CORE_OSC_FAIL_TEST	yes	no	Main oscillator test error
70	DIAG_E_INIT_CORE_ADD_BUS_PAR_B0	yes	no	TCRAM B0 address bus parity test error
71	DIAG_E_INIT_CORE_ADD_BUS_PAR_B1	yes	no	TCRAM B1 address bus parity test error
72	DIAG_E_INIT_CORE_FLASH_BUS1_PAR	yes	no	Flash address bus1 parity test error
73	DIAG_E_INIT_CORE_DMA_BASIC_TEST	yes	no	DMA Basic functionality test error
74	DIAG_E_ERROR_CALLBACK_RECURSION	yes	no	An error callback recursion has been detected

7.2.4 Macro Definition Documentation

7.2.4.1 #define DIAG_DEV_1V2 IO_INT_PIN_1V2

Internal 1.2V supply voltage

Definition at line 1274 of file DIAG_Constants.h.

7.2.4.2 #define DIAG_DEV_2MODE_CONF_0 178U

2Mode ADC Configuration 0

Definition at line 1325 of file DIAG_Constants.h.

7.2.4.3 #define DIAG_DEV_2MODE_CONF_1 179U

2Mode ADC Configuration 1

Definition at line 1326 of file DIAG_Constants.h.

7.2.4.4 #define DIAG_DEV_2MODE_CONF_2 180U

2Mode ADC Configuration 2

Definition at line 1327 of file DIAG_Constants.h.

7.2.4.5 #define DIAG_DEV_2MODE_CONF_3 181U

2Mode ADC Configuration 3

Definition at line 1328 of file DIAG_Constants.h.

7.2.4.6 #define DIAG_DEV_2MODE_CONF_4 182U

2Mode ADC Configuration 4

Definition at line 1329 of file DIAG_Constants.h.

7.2.4.7 #define DIAG_DEV_2MODE_CONF_5 183U

2Mode ADC Configuration 5

Definition at line 1330 of file DIAG_Constants.h.

7.2.4.8 #define DIAG_DEV_2MODE_CONF_6 184U

2Mode ADC Configuration 6

Definition at line 1331 of file DIAG_Constants.h.

7.2.4.9 #define DIAG_DEV_2MODE_CONF_7 185U

2Mode ADC Configuration 7

Definition at line 1332 of file DIAG_Constants.h.

7.2.4.10 #define DIAG_DEV_ADC 135U

TMS570 ADC peripheral

Definition at line 1280 of file DIAG_Constants.h.

7.2.4.11 #define DIAG_DEV_DIO 137U

TMS570 DIO peripheral

Definition at line 1282 of file DIAG_Constants.h.

7.2.4.12 #define DIAG_DEV_DMA 139U

TMS570 DMA peripheral

Definition at line 1284 of file DIAG_Constants.h.

7.2.4.13 #define DIAG_DEV_DO_CONF_0 192U

DO Configuration 0

Definition at line 1341 of file DIAG_Constants.h.

7.2.4.14 #define DIAG_DEV_DO_CONF_1 193U

DO Configuration 1

Definition at line 1342 of file DIAG_Constants.h.

7.2.4.15 #define DIAG_DEV_DO_CONF_2 194U

DO Configuration 2

Definition at line 1343 of file DIAG_Constants.h.

7.2.4.16 #define DIAG_DEV_ESM 133U

TMS570 ESM peripheral

Definition at line 1278 of file DIAG_Constants.h.

7.2.4.17 #define DIAG_DEV_ESM_CCM_R4_SLFTST 156U

CCM-R4 - self-test error

Definition at line 1302 of file DIAG_Constants.h.

7.2.4.18 #define DIAG_DEV_ESM_CCMR4_COMPARE 165U

CCMR4 - compare error

Definition at line 1311 of file DIAG_Constants.h.

7.2.4.19 #define DIAG_DEV_ESM_CLOCK_MONITOR 148U

Clock Monitor - interrupt

Definition at line 1294 of file DIAG_Constants.h.

7.2.4.20 #define DIAG_DEV_ESM_CPU_SLFTST 154U

CPU - self-test error

Definition at line 1300 of file DIAG_Constants.h.

7.2.4.21 #define DIAG_DEV_ESM_DCC1_ERROR 155U

DCC1 - error

Definition at line 1301 of file DIAG_Constants.h.

7.2.4.22 #define DIAG_DEV_ESM_DCC2 164U

DCC2 - error

Definition at line 1310 of file DIAG_Constants.h.

7.2.4.23 #define DIAG_DEV_ESM_DMA_DMM_IMPR_READ 144U

DMA - imprecise read error

Definition at line 1290 of file DIAG_Constants.h.

7.2.4.24 #define DIAG_DEV_ESM_DMA_DMM_IMPR_WRITE 149U

DMA - imprecise write error

Definition at line 1295 of file DIAG_Constants.h.

7.2.4.25 #define DIAG_DEV_ESM_DMA_MPUM_VIOLATION 142U

DMA - MPU

Definition at line 1288 of file DIAG_Constants.h.

7.2.4.26 #define DIAG_DEV_ESM_DMA_PARITY 143U

DMA - parity error

Definition at line 1289 of file DIAG_Constants.h.

7.2.4.27 #define DIAG_DEV_ESM_E_FUSE_AUTOLOAD 173U

eFuse Controller - autoload error

Definition at line 1319 of file DIAG_Constants.h.

7.2.4.28 #define DIAG_DEV_ESM_E_FUSE_CNTL 161U

eFuse Controller error

Definition at line 1307 of file DIAG_Constants.h.

7.2.4.29 #define DIAG_DEV_ESM_E_FUSE_CNTL_SLFTST 162U

eFuse Controller - self-test error

Definition at line 1308 of file DIAG_Constants.h.

7.2.4.30 #define DIAG_DEV_ESM_FLASH_ECC_LIVE_LOCK 171U

Flash (ATCM) - ECC live lock detect

Definition at line 1317 of file DIAG_Constants.h.

7.2.4.31 #define DIAG_DEV_ESM_FMC_B1_B2_UNC_ERR 176U

FMC - uncorrectable error: bus1 and bus2 interfaces

Definition at line 1322 of file DIAG_Constants.h.

7.2.4.32 #define DIAG_DEV_ESM_FMC_B1_UNC_ERR 166U

FMC - uncorrectable error (address parity on bus1 accesses)

Definition at line 1312 of file DIAG_Constants.h.

7.2.4.33 #define DIAG_DEV_ESM_FMC_CFG_FLASH_UNC_ERR 157U

FMC - uncorrectable error (configuration flash bank access)

Definition at line 1303 of file DIAG_Constants.h.

7.2.4.34 #define DIAG_DEV_ESM_HET_TU1_2_PARITY 146U

HET TU1/HET TU2 - parity error

Definition at line 1292 of file DIAG_Constants.h.

7.2.4.35 #define DIAG_DEV_ESM_IOMM_MUX_CONFIG 158U

IOMM - Mux configuration error

Definition at line 1304 of file DIAG_Constants.h.

7.2.4.36 #define DIAG_DEV_ESM_MIBADC1_PARITY 153U

MibADC1 - parity error

Definition at line 1299 of file DIAG_Constants.h.

7.2.4.37 #define DIAG_DEV_ESM_MIBADC2_PARITY 141U

MibADC2 - parity error

Definition at line 1287 of file DIAG_Constants.h.

7.2.4.38 #define DIAG_DEV_ESM_MIBSPI1_PARITY 151U

MibSPI1 - parity error

Definition at line 1297 of file DIAG_Constants.h.

7.2.4.39 #define DIAG_DEV_ESM_MIBSPI3_PARITY 152U

MibSPI3 - parity error

Definition at line 1298 of file DIAG_Constants.h.

7.2.4.40 #define DIAG_DEV_ESM_N2HET1_2_PARITY 145U

N2HET1/N2HET2 - parity error

Definition at line 1291 of file DIAG_Constants.h.

7.2.4.41 #define DIAG_DEV_ESM_PLL1_SLIP 147U

PLL - slip

Definition at line 1293 of file DIAG_Constants.h.

7.2.4.42 #define DIAG_DEV_ESM_PLL2_SLIP 163U

PLL2 - slip

Definition at line 1309 of file DIAG_Constants.h.

7.2.4.43 #define DIAG_DEV_ESM_PWR_DOM_CNTL_COMP 159U

Power domain controller compare error

Definition at line 1305 of file DIAG_Constants.h.

7.2.4.44 #define DIAG_DEV_ESM_PWR_DOM_CNTL_SLFTST 160U

Power domain controller self-test error

Definition at line 1306 of file DIAG_Constants.h.

7.2.4.45 #define DIAG_DEV_ESM_RAM_B0_ADDR_PARITY 169U

RAM even bank (B0TCM) - address bus parity error

Definition at line 1315 of file DIAG_Constants.h.

7.2.4.46 #define DIAG_DEV_ESM_RAM_B0_ECC_UNC_ERR 174U

RAM even bank (B0TCM) - ECC uncorrectable error

Definition at line 1320 of file DIAG_Constants.h.

7.2.4.47 #define DIAG_DEV_ESM_RAM_B0_UNC_ERR 167U

RAM even bank (B0TCM) - uncorrectable error

Definition at line 1313 of file DIAG_Constants.h.

7.2.4.48 #define DIAG_DEV_ESM_RAM_B1_ADDR_PARITY 170U

RAM odd bank (B1TCM) - address bus parity error

Definition at line 1316 of file DIAG_Constants.h.

7.2.4.49 #define DIAG_DEV_ESM_RAM_B1_ECC_UNC_ERR 175U

RAM odd bank (B1TCM) - ECC uncorrectable error

Definition at line 1321 of file DIAG_Constants.h.

7.2.4.50 #define DIAG_DEV_ESM_RAM_B1_UNC_ERR 168U

RAM odd bank (B1TCM) - uncorrectable error

Definition at line 1314 of file DIAG_Constants.h.

7.2.4.51 #define DIAG_DEV_ESM_RTI_WDD_NMI 172U

RTI_WWD_NMI

Definition at line 1318 of file DIAG_Constants.h.

7.2.4.52 #define DIAG_DEV_ESM_UNKNOWN 177U

Unknown ESM device

Definition at line 1323 of file DIAG_Constants.h.

7.2.4.53 #define DIAG_DEV_ESM_VIM_RAM_PARITY 150U

VIM RAM - parity error

Definition at line 1296 of file DIAG_Constants.h.

7.2.4.54 #define DIAG_DEV_EXT_SHUTOFF_0 195U

External shut-off group 0

Definition at line 1345 of file DIAG_Constants.h.

7.2.4.55 #define DIAG_DEV_EXT_SHUTOFF_1 196U

External shut-off group 1

Definition at line 1346 of file DIAG_Constants.h.

7.2.4.56 #define DIAG_DEV_EXT_SHUTOFF_2 197U

External shut-off group 2

Definition at line 1347 of file DIAG_Constants.h.

7.2.4.57 #define DIAG_DEV_MAIN_CPU 199U

Main CPU

Definition at line 1350 of file DIAG_Constants.h.

7.2.4.58 #define DIAG_DEV_MAX 201U

Definition at line 1354 of file DIAG_Constants.h.

7.2.4.59 #define DIAG_DEV MCU 200U

TMS570 MCU peripheral This item used to be device number 132

Definition at line 1352 of file DIAG_Constants.h.

7.2.4.60 #define DIAG_DEV_NHET 140U

TMS570 NHET peripheral

Definition at line 1285 of file DIAG_Constants.h.

7.2.4.61 #define DIAG_DEV_NONE IO_PIN_NONE

Definition at line 1355 of file DIAG_Constants.h.

7.2.4.62 #define DIAG_DEV_PWD IO_INT_PIN_PWD

PWD threshold comparator

Definition at line 1272 of file DIAG_Constants.h.

7.2.4.63 #define DIAG_DEV_PWD_CONF_0 186U

PWD Configuration 0

Definition at line 1334 of file DIAG_Constants.h.

7.2.4.64 #define DIAG_DEV_PWD_CONF_1 187U

PWD Configuration 1

Definition at line 1335 of file DIAG_Constants.h.

7.2.4.65 #define DIAG_DEV_PWD_CONF_2 188U

PWD Configuration 2

Definition at line 1336 of file DIAG_Constants.h.

7.2.4.66 #define DIAG_DEV_PWD_CONF_3 189U

PWD Configuration 3

Definition at line 1337 of file DIAG_Constants.h.

7.2.4.67 #define DIAG_DEV_PWD_CONF_4 190U

PWD Configuration 4

Definition at line 1338 of file DIAG_Constants.h.

7.2.4.68 #define DIAG_DEV_PWD_CONF_5 191U

PWD Configuration 5

Definition at line 1339 of file DIAG_Constants.h.

7.2.4.69 #define DIAG_DEV_REF_2V5 IO_INT_PIN_REF_2V5

Internal 2.5V reference voltage

Definition at line 1273 of file DIAG_Constants.h.

7.2.4.70 #define DIAG_DEV_RTC 138U

TMS570 RTC peripheral

Definition at line 1283 of file DIAG_Constants.h.

7.2.4.71 #define DIAG_DEV_SAFETY_SW_VP IO_INT_PIN_SAFETY_SW_VP

Internal VP Gate

Definition at line 1271 of file DIAG_Constants.h.

7.2.4.72 #define DIAG_DEV_SPI 136U

TMS570 SPI peripheral

Definition at line 1281 of file DIAG_Constants.h.

7.2.4.73 #define DIAG_DEV_VIM 134U

TMS570 VIM peripheral

Definition at line 1279 of file DIAG_Constants.h.

7.2.4.74 #define DIAG_DEV_VMON IO_INT_PIN_VMON

Internal voltage monitor

Definition at line 1275 of file DIAG_Constants.h.

7.2.4.75 #define DIAG_DEV_WATCHDOG_CPU 198U

Safety companion

Definition at line 1349 of file DIAG_Constants.h.

7.2.4.76 #define DIAG_E_ADC_1V2 10U

Internal 1V2 supply voltage out of range

Definition at line 833 of file DIAG_Constants.h.

7.2.4.77 #define DIAG_E_ADC_2MODE_RED_CHANNEL_TEST 3U

Redundant channel start-up test error of 2 mode ADCs

This error is raised if the redundant channel configuration doesn't comply with the rules in the description of [IO_ADC_SAFETY_CONF](#).

- Error device: [DIAG_DEV_ADC](#)
- Faulty value: number of safety critical 2 mode ADC inputs

- Error device: pin number of 2 mode ADC input ([IO_ADC_08..IO_ADC_23](#))
- Faulty value: pin number of related ADC input with incorrect configuration

Definition at line 782 of file DIAG_Constants.h.

7.2.4.78 #define DIAG_E_ADC_2V5_REF 9U

Internal 2V5 reference voltage out of range

Definition at line 828 of file DIAG_Constants.h.

7.2.4.79 #define DIAG_E_ADC_3MODE_SWITCH_PERIODIC 2U

Run-time test error with ADC mode selection switches

If this error is raised, the ADC channel in the `error_device` parameter is not functional anymore.

- Error device: pin number of 3 mode ADC input whose periodic test failed ([IO_ADC_00..IO_ADC_07](#))

Definition at line 768 of file DIAG_Constants.h.

7.2.4.80 #define DIAG_E_ADC_3MODE_SWITCH_TEST 1U

Start-up test error with ADC mode selection switches

If this error is raised, the ADC channel in the `error_device` parameter is not functional.

- Error device: pin number of 3 mode ADC input whose start-up test failed ([IO_ADC_00..IO_ADC_07](#))

Definition at line 759 of file DIAG_Constants.h.

7.2.4.81 #define DIAG_E_ADC_BOARD_TEMP 7U

Board temperature out of range

- Error device: [IO_ADC_BOARD_TEMP](#)
- Faulty value: raw ADC value of board temperature (value can be converted with [IO_ADC_BoardTempFloat\(\)](#) or [IO_ADC_BoardTempSbyte\(\)](#))

Definition at line 815 of file DIAG_Constants.h.

7.2.4.82 #define DIAG_E_ADC_RANGE 5U

ADC measurement range check error

This error is raised when the ADC value falls outside the range defined in the safety configuration of the pin.

- Error device: pin number of safe ADC input ([IO_ADC_00..IO_ADC_07](#), [IO_ADC_08..IO_ADC_23](#))
- Faulty value: last reading of ADC input in physical units (mV/uA/ohm)

Definition at line 798 of file DIAG_Constants.h.

7.2.4.83 #define DIAG_E_ADC_SENSOR_SUPPLY 8U

Sensor supply voltage out of range

- Error device: pin number of safety-critical sensor supply ([IO_SENSOR_SUPPLY_0](#) or [IO_SENSOR_SUPPLY_1](#))

Definition at line 823 of file DIAG_Constants.h.

7.2.4.84 #define DIAG_E_ADC_SR_CONF_CHECK 4U

Internal shift register readback error

Definition at line 787 of file DIAG_Constants.h.

7.2.4.85 #define DIAG_E_ADC_UBAT 6U

Battery voltage out of range

- Error device: [IO_ADC_UBAT](#)
- Faulty value: CPU supply voltage in mV

Definition at line 806 of file DIAG_Constants.h.

7.2.4.86 #define DIAG_E_ADC_VPGATE 11U

Internal VP Gate voltage out of range

Definition at line 838 of file DIAG_Constants.h.

7.2.4.87 #define DIAG_E_APPL_SAFE_STATE 53U

Application requested to activate the safe state

- Error device: [DIAG_DEV_MAIN_CPU](#)
- Faulty value: 0

Definition at line 1123 of file [DIAG_Constants.h](#).

7.2.4.88 #define DIAG_E_CORE_READBACK 52U

Configuration read back error

Definition at line 1115 of file [DIAG_Constants.h](#).

7.2.4.89 #define DIAG_E_DATA_ABORT 55U

CPU generated a data abort

When a data abort is generated due to MPU violation, the error device is [DIAG_DEV_MAIN_CPU](#).

- Error device:
 - [DIAG_DEV_MAIN_CPU](#) if the source of the abort is the CPU core,
 - otherwise one of the diagnostic error devices starting with "DIAG_DEV_ESM_" (e.g. [DIAG_DEV_ESM_UNKNOWN](#))
- Faulty value: holds the address of the data which caused a data abort

Definition at line 1144 of file [DIAG_Constants.h](#).

7.2.4.90 #define DIAG_E_DO_FEEDBACK 18U

DO feedback value out of range

- Error device: pin number of safe high-side digital output ([IO_DO_00..IO_DO_07](#))
- Faulty value: error code returned by [IO_DO_Set\(\)](#)

Definition at line 900 of file [DIAG_Constants.h](#).

7.2.4.91 #define DIAG_E_DO_OPEN_LOAD 17U

DO check detected an open load

- Error device: pin number of safe high-side digital output which has an open load ([IO_DO_00..IO_DO_07](#))
- Faulty value: 0

Definition at line 892 of file [DIAG_Constants.h](#).

7.2.4.92 #define DIAG_E_DO_SHORT_CIRCUIT 16U

DO start-up test detected a short circuit

- Error device: pin number of safe high-side or low-side digital output which has a short circuit ([IO_DO_00..IO_DO_07](#), [IO_DO_08..IO_DO_15](#))
 - Faulty value: pin number of one of the pins which the error device is incorrectly connected to
- Definition at line 884 of file DIAG_Constants.h.

7.2.4.93 #define DIAG_E_DRIVER_INIT 54U

Error detected during the execution of [IO_Driver_Init\(\)](#)

- Error device: [DIAG_DEV_MAIN_CPU](#)
- Faulty value: one of the error definitions in [IO_Error.h](#)

Definition at line 1131 of file DIAG_Constants.h.

7.2.4.94 #define DIAG_E_ENABLE_TREE_TEST 27U

Enable tree start-up test error

Definition at line 981 of file DIAG_Constants.h.

7.2.4.95 #define DIAG_E_ERROR_CALLBACK_RECURSION 74U

An error callback recursion has been detected

Definition at line 1257 of file DIAG_Constants.h.

7.2.4.96 #define DIAG_E_ESM_HLI 58U

ESM exception generated - fatal High Level Interrupt

Definition at line 1168 of file DIAG_Constants.h.

7.2.4.97 #define DIAG_E_ESM_LLI 59U

ESM exception generated - fatal Low Level Interrupt

Definition at line 1173 of file DIAG_Constants.h.

7.2.4.98 #define DIAG_E_ESM_LLI_CALLBACK 60U

ESM exception generated - non-fatal Low Level Interrupt

- Error device: [DIAG_DEV_ESM_FMC_CFG_FLASH_UNC_ERR](#)
- Faulty value:

- 0xFFFFFFFF if the SECDED loses the capability to properly correct a 1-bit ECC error for access to EEPROM region
- else the address of the last uncorrectable error in the configuration flash

Definition at line 1184 of file DIAG_Constants.h.

7.2.4.99 #define DIAG_E_INIT_CORE_ADD_BUS_PAR_B0 70U

TCRAM B0 address bus parity test error

Definition at line 1237 of file DIAG_Constants.h.

7.2.4.100 #define DIAG_E_INIT_CORE_ADD_BUS_PAR_B1 71U

TCRAM B1 address bus parity test error

Definition at line 1242 of file DIAG_Constants.h.

7.2.4.101 #define DIAG_E_INIT_CORE_ADD_DECODE_B0 44U

TCRAM B0 address redundant comparator test error

Definition at line 1075 of file DIAG_Constants.h.

7.2.4.102 #define DIAG_E_INIT_CORE_ADD_DECODE_B1 45U

TCRAM B1 address redundant comparator test error

Definition at line 1080 of file DIAG_Constants.h.

7.2.4.103 #define DIAG_E_INIT_CORE_CCM_SELFTEST 43U

CCM-R4F module test error

Definition at line 1070 of file DIAG_Constants.h.

7.2.4.104 #define DIAG_E_INIT_CORE_CFG_FLASH_ECC 41U

CPU ECC logic accesses to configuration flash test error

Definition at line 1060 of file DIAG_Constants.h.

7.2.4.105 #define DIAG_E_INIT_CORE_DCC1_SELFTEST 46U

DCC1 self-test error

Definition at line 1085 of file DIAG_Constants.h.

7.2.4.106 #define DIAG_E_INIT_CORE_DCC2_SELFTEST 47U

DCC2 self-test error

Definition at line 1090 of file DIAG_Constants.h.

7.2.4.107 #define DIAG_E_INIT_CORE_DMA_BASIC_TEST 73U

DMA Basic functionality test error

Definition at line 1252 of file DIAG_Constants.h.

7.2.4.108 #define DIAG_E_INIT_CORE_EFUSE_ECC 32U

eFuse controller test error

Definition at line 1015 of file DIAG_Constants.h.

7.2.4.109 #define DIAG_E_INIT_CORE_ERROR_PIN_TEST 48U

Error pin test error during start-up

Definition at line 1095 of file DIAG_Constants.h.

7.2.4.110 #define DIAG_E_INIT_CORE_FLASH_BUS1_PAR 72U

Flash address bus1 parity test error

Definition at line 1247 of file DIAG_Constants.h.

7.2.4.111 #define DIAG_E_INIT_CORE_FLASH_DATA_ECC 40U

CPU ECC logic accesses to program flash test error

Definition at line 1055 of file DIAG_Constants.h.

7.2.4.112 #define DIAG_E_INIT_CORE_FLASH_WR_ECC 33U

Flash Module Controller - ECC logic test error

Definition at line 1020 of file DIAG_Constants.h.

7.2.4.113 #define DIAG_E_INIT_CORE_IOMM_LOCK 42U

IOMM Lock test error

Definition at line 1065 of file DIAG_Constants.h.

7.2.4.114 #define DIAG_E_INIT_CORE_IOMM_PROT_TEST 66U

IOMM lock test error

Definition at line 1217 of file DIAG_Constants.h.

7.2.4.115 #define DIAG_E_INIT_CORE_L2L3 31U

L2L3 interconnect test error

Definition at line 1010 of file DIAG_Constants.h.

7.2.4.116 #define DIAG_E_INIT_CORE_MPUM_TEST 49U

MPU test error during start-up

Definition at line 1100 of file DIAG_Constants.h.

7.2.4.117 #define DIAG_E_INIT_CORE_OSC_FAIL_TEST 69U

Main oscillator test error

Definition at line 1232 of file DIAG_Constants.h.

7.2.4.118 #define DIAG_E_INIT_CORE_PBIST_TEST 37U

Programmable built-in self-test controller test error

Definition at line 1040 of file DIAG_Constants.h.

7.2.4.119 #define DIAG_E_INIT_CORE_PLL1_SLIP_TEST 67U

PLL1 slip test error

Definition at line 1222 of file DIAG_Constants.h.

7.2.4.120 #define DIAG_E_INIT_CORE_PLL2_SLIP_TEST 68U

PLL2 slip test error

Definition at line 1227 of file DIAG_Constants.h.

7.2.4.121 #define DIAG_E_INIT_CORE_PSCON_SELFTEST 36U

Diagnostic power state controller test error

Definition at line 1035 of file DIAG_Constants.h.

7.2.4.122 #define DIAG_E_INIT_CORE_RAM_ECC_B0 38U

CPU ECC logic for accesses to TCRAM B0 test error

Definition at line 1045 of file DIAG_Constants.h.

7.2.4.123 #define DIAG_E_INIT_CORE_RAM_ECC_B1 39U

CPU ECC logic for accesses to TCRAM B1 test error

Definition at line 1050 of file DIAG_Constants.h.

7.2.4.124 #define DIAG_E_INIT_CORE_RAM_PARITY_TEST 50U

RAM parity test error during start-up

Definition at line 1105 of file DIAG_Constants.h.

7.2.4.125 #define DIAG_E_INIT_CORE_RAM_PBIST 51U

RAM self-test error during start-up

Definition at line 1110 of file DIAG_Constants.h.

7.2.4.126 #define DIAG_E_INIT_CORE_SELFTEST 35U

CPU self-test error

Definition at line 1030 of file DIAG_Constants.h.

7.2.4.127 #define DIAG_E_INIT_CORE_STC_TEST 34U

CPU self-test controller test error

Definition at line 1025 of file DIAG_Constants.h.

7.2.4.128 #define DIAG_E_INVALID_DIAG_STATE 62U

Invalid state for diagnostic state machine

Definition at line 1194 of file DIAG_Constants.h.

7.2.4.129 #define DIAG_E_INVALID_IRQ 63U

Invalid interrupt detected

Definition at line 1199 of file DIAG_Constants.h.

7.2.4.130 #define DIAG_E_MAIN_LOOP 65U

The application returned from the `main()` function

- Error device: [DIAG_DEV_MAIN_CPU](#)
- Faulty value: 0

Definition at line 1212 of file `DIAG_Constants.h`.

7.2.4.131 #define DIAG_E_NOERROR 0U

No error

Definition at line 750 of file `DIAG_Constants.h`.

7.2.4.132 #define DIAG_E_PARITY_FALLBACK 61U

CPU generated an interrupt vector parity fallback exception

Definition at line 1189 of file `DIAG_Constants.h`.

7.2.4.133 #define DIAG_E_PREFETCH_ABORT 56U

CPU generated a prefetch abort

- Error device:
 - [DIAG_DEV_MAIN_CPU](#) if the source of the abort is the CPU core,
 - otherwise one of the diagnostic error devices starting with "DIAG_DEV_ESM_" (e.g. [DIAG_DEV_ESM_UNKNOWN](#))
- Faulty value: holds the address of the instruction that caused a prefetch abort

Definition at line 1155 of file `DIAG_Constants.h`.

7.2.4.134 #define DIAG_E_PRG_OVERFLOW 64U

NHET program overflow error

Definition at line 1204 of file `DIAG_Constants.h`.

7.2.4.135 #define DIAG_E_PWD_CURRENT 20U

PWD current check error

This error is thrown if a PWD channel was initialized as a current input (with [IO_PWD_PD_90](#) pull resistor) and its current is outside the range of 4..20 mA, or if the [IO_PWD_GetCurrent\(\)](#) function was not called recently.

- Error device: pin number of safe PWD input ([IO_PWD_00..IO_PWD_05](#))

Definition at line 923 of file `DIAG_Constants.h`.

7.2.4.136 #define DIAG_E_PWD_RANGE 19U

PWD counter value, frequency or pulse width out of range

- Error device: pin number of safe PWD input ([IO_PWD_00..IO_PWD_05](#))
- Faulty value:
 - 0xFFFFFFFF (4,294,967,295) if [IO_PWD_ComplexGet\(\)](#) or [IO_PWD_UniversalGet\(\)](#) was not called recently
 - if bit 31 is set, bits 0..27 denote the PWD counter value which falls outside the defined safety range
 - if bit 30 is set, bits 0..27 denote the PWD frequency value which falls outside the defined safety range
 - if bit 29 is set, bits 0..27 denote the PWD pulse width value which falls outside the defined safety range

Definition at line 912 of file [DIAG_Constants.h](#).

7.2.4.137 #define DIAG_E_PWD_THRESH 21U

PWD threshold analog feedback out of range

This error means that digital reading on any PWD input (neither on [IO_PWD_00..IO_PWD_05](#) nor on [IO_PWD_06..IO_PWD_11](#)) cannot be trusted because the comparator threshold used for digitizing the analog signals falls outside the safe range.

This error doesn't affect the analog reading of the PWD inputs.

- Error device: [DIAG_DEV_PWD](#)

Definition at line 936 of file [DIAG_Constants.h](#).

7.2.4.138 #define DIAG_E_PWM_CURRENT 15U

PWM current check detected an overcurrent

- Error device: pin number of safe PWM channel ([IO_PWM_00..IO_PWM_35](#))
- Faulty value:
 - 0xFFFF (65535) if neither [IO_PWM_GetCur\(\)](#) nor [IO_PWM_GetCurQueue\(\)](#) have been called recently
 - otherwise the PWM current in milliamperes, as returned by [IO_PWM_GetCur\(\)](#) or [IO_PWM_GetCurQueue\(\)](#)

Definition at line 875 of file [DIAG_Constants.h](#).

7.2.4.139 #define DIAG_E_PWM_FEEDBACK 14U

PWM pulse or period feedback out of range

- Error device: pin number of safe PWM channel ([IO_PWM_00..IO_PWM_35](#))
- Faulty value:

- 0xFFFFFFFF (4,294,967,295) if [IO_PWM_SetDuty\(\)](#) hasn't been called recently, otherwise
- bits 0..15: duty cycle feedback in microseconds
- bits 16..31: period feedback in microseconds

Definition at line 865 of file DIAG_Constants.h.

7.2.4.140 #define DIAG_E_PWM_OPEN_LOAD 13U

PWM start-up test detected an open load

- Error device: pin number of safe PWM channel which has an open load ([IO_PWM_00..IO_PWM_35](#))
- Faulty value: 0

Definition at line 854 of file DIAG_Constants.h.

7.2.4.141 #define DIAG_E_PWM_SHORT_CIRCUIT 12U

PWM start-up test detected a short circuit

- Error device: pin number of safe PWM channel which has a short circuit ([IO_PWM_00..IO_PWM_35](#))
- Faulty value: pin number of one of the pins which the error device is incorrectly connected to

Definition at line 846 of file DIAG_Constants.h.

7.2.4.142 #define DIAG_E_SSW_EXT_SHUTOFF 24U

Safety switch external shut-off redundancy check error

This error is raised if the two inputs of an external shut-off group have an invalid reading (both low or both high).

- Error device: [DIAG_DEV_EXT_SHUTOFF_0..DIAG_DEV_EXT_SHUTOFF_2](#)
- Faulty value:
 - bits 0..15: analog reading of input 1 of the given external shut-off group, in millivolts
 - bits 16..31: analog reading of input 0 of the given external shut-off group, in millivolts

Definition at line 966 of file DIAG_Constants.h.

7.2.4.143 #define DIAG_E_SSW_PERIODIC 23U

Safety switch periodic check error

- Error device: pin number of safety critical safety switch ([IO_INT_SAFETY_SW_0..IO_INT_SAFETY_SW_2](#))

Definition at line 953 of file DIAG_Constants.h.

7.2.4.144 #define DIAG_E_SSW_TEST 22U

Safety switch start-up test error

- Error device: pin number of safety critical safety switch ([IO_INT_SAFETY_SW_0..IO_INT_SAFETY_SW_2](#))
- Faulty value:
 - 0 if the safety switch cannot be switched on
 - 1 if the safety switch cannot be switched off

Definition at line 946 of file DIAG_Constants.h.

7.2.4.145 #define DIAG_E_UNDEF_INSTRUCTION 57U

CPU generated an undefined instruction exception

- Error device: [DIAG_DEV_MAIN_CPU](#)
- Faulty value: 0

Definition at line 1163 of file DIAG_Constants.h.

7.2.4.146 #define DIAG_E_VMON_PERIODIC 26U

Internal voltage monitor periodic check error

Definition at line 976 of file DIAG_Constants.h.

7.2.4.147 #define DIAG_E_VMON_TEST 25U

Internal voltage monitor start-up test error

Definition at line 971 of file DIAG_Constants.h.

7.2.4.148 #define DIAG_E_WD_ACTIVATION 29U

Watchdog activation failed

- Error device: [DIAG_DEV_WATCHDOG_CPU](#)
- Faulty value: one of the error definitions in [IO_Error.h](#)

Definition at line 997 of file DIAG_Constants.h.

7.2.4.149 #define DIAG_E_WD_INIT 28U

Watchdog initialization failed

- Error device: [DIAG_DEV_MAIN_CPU](#) or [DIAG_DEV_WATCHDOG_CPU](#)
- Faulty value: one of the error definitions in [IO_Error.h](#)

Definition at line 989 of file DIAG_Constants.h.

7.2.4.150 #define DIAG_E_WD_TRIGGER 30U

Watchdog triggering failed

- Error device: [DIAG_DEV_MAIN_CPU](#) or [DIAG_DEV_WATCHDOG_CPU](#),
- Faulty value: one of the error definitions in [IO_Error.h](#)

Definition at line 1005 of file [DIAG_Constants.h](#).

7.2.4.151 #define DIAG_ERR_DISABLE_HS00 0x0020U

disable the high side digital output 0 ([IO_DO_00](#))

Definition at line 1374 of file [DIAG_Constants.h](#).

7.2.4.152 #define DIAG_ERR_DISABLE_HS01 0x0040U

disable the high side digital output 1 ([IO_DO_01](#))

Definition at line 1375 of file [DIAG_Constants.h](#).

7.2.4.153 #define DIAG_ERR_DISABLE_HS02 0x0080U

disable the high side digital output 2 ([IO_DO_02](#))

Definition at line 1376 of file [DIAG_Constants.h](#).

7.2.4.154 #define DIAG_ERR_DISABLE_HS03 0x0100U

disable the high side digital output 3 ([IO_DO_03](#))

Definition at line 1377 of file [DIAG_Constants.h](#).

7.2.4.155 #define DIAG_ERR_DISABLE_HS04 0x0200U

disable the high side digital output 4 ([IO_DO_04](#))

Definition at line 1378 of file [DIAG_Constants.h](#).

7.2.4.156 #define DIAG_ERR_DISABLE_HS05 0x0400U

disable the high side digital output 5 ([IO_DO_05](#))

Definition at line 1379 of file [DIAG_Constants.h](#).

7.2.4.157 #define DIAG_ERR_DISABLE_HS06 0x0800U

disable the high side digital output 6 ([IO_DO_06](#))

Definition at line 1380 of file [DIAG_Constants.h](#).

7.2.4.158 #define DIAG_ERR_DISABLE_HS07 0x1000U

disable the high side digital output 7 ([IO_DO_07](#))

Definition at line 1381 of file DIAG_Constants.h.

7.2.4.159 #define DIAG_ERR_DISABLE_SSW0 0x0004U

disable the shut-off group 0 ([IO_INT_SAFETY_SW_0](#))

Definition at line 1371 of file DIAG_Constants.h.

7.2.4.160 #define DIAG_ERR_DISABLE_SSW1 0x0008U

disable the shut-off group 1 ([IO_INT_SAFETY_SW_1](#))

Definition at line 1372 of file DIAG_Constants.h.

7.2.4.161 #define DIAG_ERR_DISABLE_SSW2 0x0010U

disable the shut-off group 2 ([IO_INT_SAFETY_SW_2](#))

Definition at line 1373 of file DIAG_Constants.h.

7.2.4.162 #define DIAG_ERR_NOACTION 0x0001U

take no action (ignore the error)

Definition at line 1369 of file DIAG_Constants.h.

7.2.4.163 #define DIAG_ERR_SAFESTATE 0x0002U

enter the safe state (switch off all outputs)

Definition at line 1370 of file DIAG_Constants.h.

7.2.4.164 #define DIAG_STATE_CONFIG 0x05UL

Diagnostic state machine is in config state

Definition at line 708 of file DIAG_Constants.h.

7.2.4.165 #define DIAG_STATE_DISABLED 0x00UL

Diagnostic state machine is disabled

Definition at line 706 of file DIAG_Constants.h.

7.2.4.166 #define DIAG_STATE_INIT 0x03UL

Diagnostic state machine is in init state

Definition at line 707 of file DIAG_Constants.h.

7.2.4.167 #define DIAG_STATE_MAIN 0x06UL

Diagnostic state machine is in main state

Definition at line 709 of file DIAG_Constants.h.

7.2.4.168 #define DIAG_STATE_SAFE 0x09UL

Diagnostic state machine is in safe state

Definition at line 710 of file DIAG_Constants.h.

7.2.4.169 #define DIAG_WD_STATE_ACTIVE 0x06U

Watchdog CPU is in active state

Definition at line 729 of file DIAG_Constants.h.

7.2.4.170 #define DIAG_WD_STATE_DIAGNOSTIC 0x05U

Watchdog CPU is in diagnostic state

Definition at line 728 of file DIAG_Constants.h.

7.2.4.171 #define DIAG_WD_STATE_RESET 0x03U

Watchdog CPU is in reset state

Definition at line 727 of file DIAG_Constants.h.

7.2.4.172 #define DIAG_WD_STATE_SAFE 0x09U

Watchdog CPU is in safe state

Definition at line 730 of file DIAG_Constants.h.

7.2.4.173 #define DIAG_WD_STATE_STANDBY 0x01U

Watchdog CPU is in standby state

Definition at line 726 of file DIAG_Constants.h.

7.2.4.174 #define DIAG_WD_STATE_UNKNOWN 0xFFU

Watchdog CPU is in an unknown state

Definition at line 731 of file DIAG_Constants.h.

7.2.5 Typedef Documentation

7.2.5.1 `typedef ubyte2(* DIAG_ERROR_CB)(ubyte1 diag_state, ubyte1 watchdog_state, DIAG_ERRORCODE *const error)`

Callback function for non-fatal errors.

If a non-fatal error occurs, the diagnostic state machine asks the application which action to take by calling this callback function. The callback function is passed to the function `IO_Driver_Init()` in a field of the safety configuration (`IO_DRIVER_SAFETY_CONF`).

Please refer to [Diagnostic state machine error codes](#) for details on the error types and how they are treated by the diagnostic state machine.

Parameters

<code>diag_state</code>	current state of the diagnostic state machine
<code>watchdog_state</code>	current state of the watchdog CPU
<code>out</code>	error code structure, describing the detected error

Returns

`ubyte2`

Except for `DIAG_ERR_NOACTION` and `DIAG_ERR_SAFESTATE` all values may be combined (OR-bitmask).

Return values

<code>DIAG_ERR_NOACTION</code>	take no action (ignore the error)
<code>DIAG_ERR_SAFESTATE</code>	enter the safe state (switch off all outputs)
<code>DIAG_ERR_DISABLE_SSW0</code>	disable the shut-off group 0 (<code>IO_INT_SAFETY_SW_0</code>)
<code>DIAG_ERR_DISABLE_SSW1</code>	disable the shut-off group 1 (<code>IO_INT_SAFETY_SW_1</code>)
<code>DIAG_ERR_DISABLE_SSW2</code>	disable the shut-off group 2 (<code>IO_INT_SAFETY_SW_2</code>)
<code>DIAG_ERR_DISABLE_HS00</code>	disable the high side digital output 0 (<code>IO_DO_00</code>)
<code>DIAG_ERR_DISABLE_HS01</code>	disable the high side digital output 1 (<code>IO_DO_01</code>)
<code>DIAG_ERR_DISABLE_HS02</code>	disable the high side digital output 2 (<code>IO_DO_02</code>)
<code>DIAG_ERR_DISABLE_HS03</code>	disable the high side digital output 3 (<code>IO_DO_03</code>)
<code>DIAG_ERR_DISABLE_HS04</code>	disable the high side digital output 4 (<code>IO_DO_04</code>)
<code>DIAG_ERR_DISABLE_HS05</code>	disable the high side digital output 5 (<code>IO_DO_05</code>)
<code>DIAG_ERR_DISABLE_HS06</code>	disable the high side digital output 6 (<code>IO_DO_06</code>)
<code>DIAG_ERR_DISABLE_HS07</code>	disable the high side digital output 7 (<code>IO_DO_07</code>)

Remarks

See [Example implementation for Safety-Callback](#).

Definition at line 660 of file DIAG_Constants.h.

7.2.5.2 **typedef struct diag_errorcode_ DIAG_ERRORCODE**

Diagnostic error code structure.

Stores all relevant error parameters returned from the diagnostic state machine or returned from the WD. See [Diagnostic state machine error reporting](#) for details about the diagnostic error reporting mechanism.

7.2.5.3 **typedef void(* DIAG_NOTIFY_CB)(ubyte1 diag_state, ubyte1 watchdog_state, DIAG_ERRORCODE *const error)**

Callback notification function for fatal errors.

If a fatal error occurs, the diagnostic state machine notifies the application about the error occurrence by calling this callback function. The implementation of this callback is passed to [IO_Driver_Init\(\)](#) in the `notify_callback` field of the safety configuration structure [IO_DRIVER_SAFETY_CONF](#).

The safe state is activated **before** this callback is called. If the I/O Driver is configured with resets (field `reset_behavior` in [IO_DRIVER_SAFETY_CONF](#)), a reset is expected after 22 ms.

Please refer to [Diagnostic state machine error codes](#) for details on the error types and how they are treated by the diagnostic state machine.

Parameters

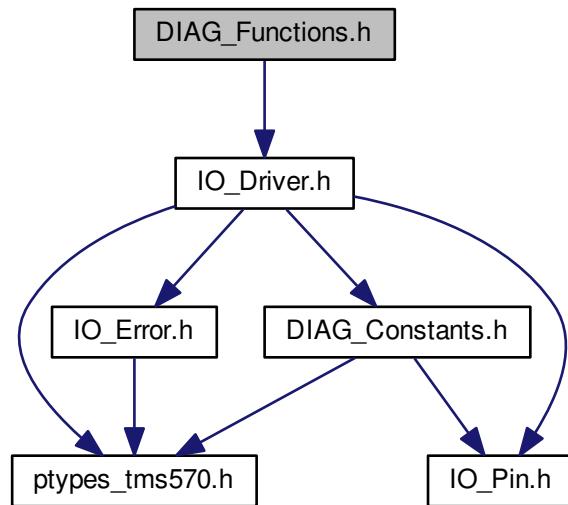
<i>diag_state</i>	current state of the diagnostic state machine
<i>watchdog_state</i>	current state of the watchdog CPU
<i>out</i>	<i>error</i> error code structure, describing the detected error

Definition at line 684 of file DIAG_Constants.h.

7.3 **DIAG_Functions.h File Reference**

Auxiliary functions for watchdog handling and Flash/RAM/CfgFlash correctable errors monitoring.

Include dependency graph for DIAG_Functions.h:



Functions

- `IO_ErrorType DIAG_EnterSafestate (void)`
Allows an application driven safe state.
- `IO_ErrorType DIAG_GetCfgFlashErrors (ubyte2 *err_cnt, bool *overflow)`
Reads the number of correctable ECC errors in the configuration flash.
- `IO_ErrorType DIAG_GetFlashErrors (ubyte2 *err_cnt, bool *overflow)`
Reads the number of correctable ECC errors in the internal flash.
- `IO_ErrorType DIAG_GetRamB0Errors (ubyte2 *err_cnt, bool *overflow)`
Reads the number of correctable ECC errors in bank 0 of the internal RAM.
- `IO_ErrorType DIAG_GetRamB1Errors (ubyte2 *err_cnt, bool *overflow)`
Reads the number of correctable ECC errors in bank 1 of the internal RAM.
- `IO_ErrorType DIAG_Status (ubyte1 *diag_state, ubyte1 *watchdog_state, DIAG_ERRORCODE *diag_error, DIAG_ERRORCODE *watchdog_error, ubyte1 *error_count)`
Status function for diagnostic state machine and watchdog states.

7.3.1 Detailed Description

Auxiliary functions for watchdog handling and Flash/RAM/CfgFlash correctable errors monitoring.
Provides the interface to the watchdog infrastructure and Flash/RAM/CfgFlash correctable errors monitoring.

7.3.2 Function Documentation

7.3.2.1 IO_ErrorType DIAG_EnterSafestate (void)

Allows an application driven safe state.

The safe state is activated immediately. When resets are configured in the I/O Driver safety configuration (field `reset_behavior` in `IO_DRIVER_SAFETY_CONF`), the ECU will reset after 22 ms.

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	Everything fine.
----------------------	------------------

7.3.2.2 IO_ErrorType DIAG_GetCfgFlashErrors (ubyte2 * err_cnt, bool * overflow)

Reads the number of correctable ECC errors in the configuration flash.

This function returns the number of correctable ECC errors in the configuration flash since start-up.

If there is an overflow in the 16 bit internal register in the CPU, the `overflow` parameter is set to `TRUE` and `err_cnt` is set to 65,535.

Parameters

<code>out</code>	<code>err_cnt</code>	Returns the correctable ECC error counter value of the configuration flash.
<code>out</code>	<code>overflow</code>	<p>Returns the counter overflow status. This parameter is optional (it's not set if <code>NULL</code>). Possible values are:</p> <ul style="list-style-type: none"> • <code>FALSE</code>: No overflow occurred. • <code>TRUE</code>: An overflow occurred in the counter. The actual number of errors is anything larger than 65,535.

Return values

<code>IO_E_OK</code>	Operation executed successfully.
<code>IO_E_NULL_POINTER</code>	The <code>err_cnt</code> parameter is a NULL pointer.

7.3.2.3 IO_ErrorType DIAG_GetFlashErrors (ubyte2 * err_cnt, bool * overflow)

Reads the number of correctable ECC errors in the internal flash.

This function returns the number of correctable ECC errors in the flash since start-up.

If there is an overflow in the 16 bit internal register in the CPU, the `overflow` parameter is set to `TRUE` and `err_cnt` is set to 65,535.

Parameters

out	<i>err_cnt</i>	Returns the correctable ECC error counter value of the flash.
out	<i>overflow</i>	<p>Returns the counter overflow status. This parameter is optional (it's not set if <code>NULL</code>).</p> <p>Possible values are:</p> <ul style="list-style-type: none"> • <code>FALSE</code>: No overflow occurred. • <code>TRUE</code>: An overflow occurred in the counter. The actual number of errors is anything larger than 65,535.

Return values

<code>IO_E_OK</code>	Operation executed successfully.
<code>IO_E_NULL_POINTER</code>	The <i>err_cnt</i> parameter is a NULL pointer.

7.3.2.4 IO_ErrorType DIAG_GetRamB0Errors (`ubyte2 * err_cnt, bool * overflow`)

Reads the number of correctable ECC errors in bank 0 of the internal RAM.

This function returns the number of correctable ECC errors in bank 0 of the internal RAM since start-up.

If there is an overflow in the 16 bit internal register in the CPU, the *overflow* parameter is set to `TRUE` and *err_cnt* is set to 65,535.

Parameters

out	<i>err_cnt</i>	Returns the correctable ECC error counter value of bank 0 of the internal RAM.
out	<i>overflow</i>	<p>Returns the counter overflow status. This parameter is optional (it's not set if <code>NULL</code>).</p> <p>Possible values are:</p> <ul style="list-style-type: none"> • <code>FALSE</code>: No overflow occurred. • <code>TRUE</code>: An overflow occurred in the counter. The actual number of errors is anything larger than 65,535.

Return values

<code>IO_E_OK</code>	Operation executed successfully.
<code>IO_E_NULL_POINTER</code>	The <i>err_cnt</i> parameter is a NULL pointer.

7.3.2.5 IO_ErrorType DIAG_GetRamB1Errors (`ubyte2 * err_cnt, bool * overflow`)

Reads the number of correctable ECC errors in bank 1 of the internal RAM.

This function returns the number of correctable ECC errors in bank 1 of the internal RAM since start-up.

If there is an overflow in the 16 bit internal register in the CPU, the *overflow* parameter is set to `TRUE` and *err_cnt* is set to 65,535.

Parameters

out	<i>err_cnt</i>	Returns the correctable ECC error counter value of bank 1 of the internal RAM.
out	<i>overflow</i>	<p>Returns the counter overflow status. This parameter is optional (it's not set if NULL).</p> <p>Possible values are:</p> <ul style="list-style-type: none"> • FALSE: No overflow occurred. • TRUE: An overflow occurred in the counter. The actual number of errors is anything larger than 65,535.

Return values

<i>IO_E_OK</i>	Operation executed successfully.
<i>IO_E_NULL_POINTER</i>	The <i>err_cnt</i> parameter is a NULL pointer.

7.3.2.6 **IO_ErrorType** **DIAG_Status** (**ubyte1** * *diag_state*, **ubyte1** * *watchdog_state*, **DIAG_ERRORCODE** * *diag_error*, **DIAG_ERRORCODE** * *watchdog_error*, **ubyte1** * *error_count*)

Status function for diagnostic state machine and watchdog states.

Returns the current status as well as the error codes of the diagnostic state machine and the watchdog CPU.

Parameters

out	<i>diag_state</i>	current state of the diagnostic state machine
out	<i>watchdog_state</i>	current state of the watchdog CPU
out	<i>diag_error</i>	error codes of the diagnostic state machine
out	<i>watchdog_error</i>	error codes of the watchdog CPU
out	<i>error_count</i>	watchdog CPU device error counter

Returns

IO_ErrorType

Return values

<i>IO_E_OK</i>	everything fine
<i>IO_E_UNKNOWN</i>	an unknown error occurred
<i>IO_E_NULL_POINTER</i>	null pointer has been passed

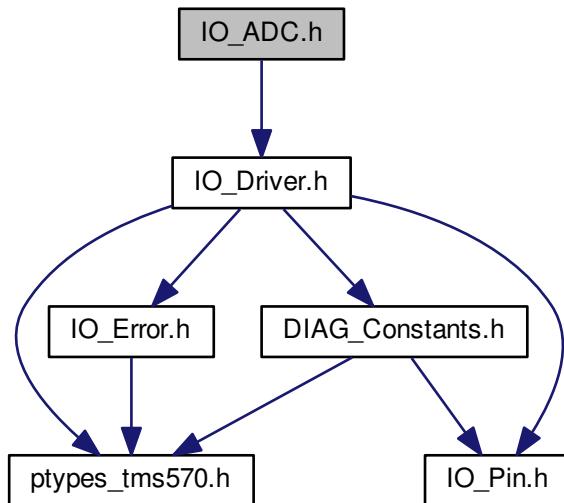
Note

When debug key in the APDB is set, the error count will be by 1 higher than not in debug mode, because of the initial reset of the debugger.

7.4 IO_ADC.h File Reference

IO Driver functions for ADC.

Include dependency graph for IO_ADC.h:



Data Structures

- struct `io_adc_safety_conf_`
Safety configuration for the ADC inputs.

Typedefs

- typedef struct `io_adc_safety_conf_` `IO_ADC_SAFETY_CONF`
Safety configuration for the ADC inputs.

Functions

- `float4 IO_ADC_BoardTempFloat (ubyte4 raw_value)`
Calculates the board temperature in degree Celsius.
- `sbyte2 IO_ADC_BoardTempSbyte (ubyte4 raw_value)`
Calculates the board temperature in degree Celsius.
- `IO_ErrorType IO_ADC_ChannelDelInit (ubyte1 adc_channel)`
Deinitializes one ADC input.
- `IO_ErrorType IO_ADC_ChannellInit (ubyte1 adc_channel, ubyte1 type, ubyte1 range, ubyte1 pupd, ubyte1 sensor_supply, const IO_ADC_SAFETY_CONF *const safety_conf)`
Setup one ADC channel.
- `IO_ErrorType IO_ADC_Get (ubyte1 adc_channel, ubyte4 *const adc_value, bool *const fresh)`
Returns the value of the given ADC channel.
- `IO_ErrorType IO_ADC_ResetProtection (ubyte1 adc_channel, ubyte1 *const reset_cnt)`
Reset the FET protection for an ADC channel.

Types of configurable Analog inputs

Input configuration for configurable ADC inputs. These defines can be used for the `type` parameter of the function `IO_ADC_ChannelInit()`.

- `#define IO_ADC_RATIOMETRIC 0x00U`
- `#define IO_ADC_CURRENT 0x01U`
- `#define IO_ADC_RESISTIVE 0x02U`
- `#define IO_ADC_ABSOLUTE 0x03U`

Pull up / Pull down configuration for ADC inputs

Configuration of the pull up or pull down resistors on the ADC inputs. These defines can be used for the `pupd` parameter of the function `IO_ADC_ChannelInit()`.

- `#define IO_ADC_NO_PULL 0x00U`
- `#define IO_ADC_PU_10K 0x02U`
- `#define IO_ADC_PD_10K 0x01U`

Range configuration for ADC inputs

Configuration of the ADC input range. These defines can be used for the `range` parameter of the function `IO_ADC_ChannelInit()`.

- `#define IO_ADC_NO_RANGE 0x00U`
- `#define IO_ADC_RANGE_5V 0x01U`
- `#define IO_ADC_RANGE_10V 0x02U`
- `#define IO_ADC_RANGE_32V 0x03U`

7.4.1 Detailed Description

IO Driver functions for ADC.

Contains all service functions for the ADC. There are three groups of ADC inputs:

- 3 mode ADC:

- 5V ADC inputs.
- Up to 8 ADC inputs for 0-5V measurement ([IO_ADC_00..IO_ADC_07](#)).
- Can be configured as resistive, current, absolute or ratiometric input. An additional reference channel or a sensor supply measurement (for the ratiometric case) will be configured and serves to correct the ADC signal.
- Range: 0-5V
- 2 mode 10V ADC:
 - Software configurable ADC inputs.
 - Up to 8 ADC inputs for 0-10V measurement ([IO_ADC_08..IO_ADC_15](#)).
 - Can be configured as current, absolute or ratiometric input. An additional reference channel or a sensor supply measurement (for the ratiometric case) will be configured and serves to correct the ADC signal.
 - Range: can be configured to 0-5V or 0-10V for absolute and ratiometric measurement.
- 2 mode 32V ADC:
 - Software configurable ADC inputs.
 - Up to 8 ADC inputs for 0-32V measurement ([IO_ADC_16..IO_ADC_23](#)).
 - Can be configured as current, absolute or ratiometric input. An additional reference channel or a sensor supply measurement (for the ratiometric case) will be configured and serves to correct the ADC signal.
 - Range: can be configured to 0-5V or 0-32V for absolute and ratiometric measurement.
- Normal ADC:
 - Various ADC inputs for retrieving onboard voltages (UBat, Sensor-Supply, Board-Temperature, ...)

Note

The ADC channels will be sampled every 2ms. After this time a new measurement is available and will be flagged as fresh via the parameter `fresh` of [IO_ADC_Get\(\)](#).

Attention

For [IO_ADC_RATIOMETRIC](#) mode the following configuration options are recommended:

- 3 mode ADC: ([IO_ADC_00..IO_ADC_07](#)) with [IO_ADC_SENSOR_SUPPLY_0](#) or [IO_ADC_SENSOR_SUPPLY_1](#).
- 3 mode ADC: ([IO_ADC_00..IO_ADC_07](#)) with [IO_ADC_SENSOR_SUPPLY_2](#) (sensor supply voltage setting 5V ... 10V).
- 2 mode 10V ADC: ([IO_ADC_08..IO_ADC_15](#)) configured in [IO_ADC_RANGE_5V](#) with [IO_ADC_SENSOR_SUPPLY_0](#) or [IO_ADC_SENSOR_SUPPLY_1](#).
- 2 mode 10V ADC: ([IO_ADC_08..IO_ADC_15](#)) configured in [IO_ADC_RANGE_5V](#) range with [IO_ADC_SENSOR_SUPPLY_2](#) (sensor supply voltage setting 5V ... 10V)
- 2 mode 10V ADC: ([IO_ADC_08..IO_ADC_15](#)) configured in [IO_ADC_RANGE_10V](#) range with [IO_ADC_SENSOR_SUPPLY_2](#) (sensor supply voltage setting 6V ... 10V).

The precision of the remaining configuration options in [IO_ADC_RATIOMETRIC](#) mode for reading a ratiometric sensor connected to one of the sensor supplies is limited by the hardware design and cannot be significantly improved by selecting the ratiometric mode. **The configuration options not listed above are NOT recommended.**

ADC-API Usage:

- Examples for ADC API functions

7.4.2 ADC input protection

Each ADC input that is configured for current measurement mode is individually protected against overcurrent. Whenever two consecutive samples above 27234uA are measured, the input protection is enabled latest within 22ms.

When entering the protection state, the ADC input has to remain in this state for at least 1s. After this wait time the ADC input can be reenabled via function `IO_ADC_ResetProtection()`. Note that the number of reenabling operations for a single ADC input is limited to 10. Refer to function `IO_ADC_ResetProtection()` for more information on how to reenable an ADC input.

Attention

ADC input protection is only applicable to ADC channels `IO_ADC_00 .. IO_ADC_23` when configured for current measurement mode.

7.4.3 ADC Code Examples

Examples for using the ADC API

7.4.3.1 Example for ADC initialization

```
// 3 mode ADC:
IO_ADC_ChannelInit(IO_ADC_00,
                    IO_ADC_RATIOMETRIC,           // ratiometric configuration
                    IO_ADC_NO_RANGE,
                    IO_ADC_NO_PULL,
                    IO_SENSOR_SUPPLY_0,          // sensor supply 0 is used
                    NULL);                      // not safety critical

// 2 mode 10V ADC:
IO_ADC_ChannelInit(IO_ADC_08,
                    IO_ADC_ABSOLUTE,             // absolute configuration
                    IO_ADC_RANGE_10V,            // 10V configuration
                    IO_ADC_NO_PULL,
                    IO_PIN_NONE,
                    NULL);                      // not safety critical

// Normal ADC:
IO_ADC_ChannelInit(IO_ADC_UBAT,
                    IO_ADC_ABSOLUTE,
                    IO_ADC_RANGE_32V,
                    IO_ADC_NO_PULL,
                    IO_PIN_NONE,
                    NULL);                      // not safety critical
```

7.4.3.2 Example for ADC task function call

This function call is identical for every type of ADC inputs.

```
ubyte4 adc_val_20;
bool adc_fresh_20;
IO_ErrorType rc;

rc = IO_ADC_Get(IO_ADC_20,
                 &adc_val_20,
                 &adc_fresh_20);
```

7.4.4 Macro Definition Documentation

7.4.4.1 #define IO_ADC_ABSOLUTE 0x03U

Absolute voltage measurement

use this configuration to measure an absolute voltage signal

Task function returns voltage in [mV]

Definition at line 177 of file IO_ADC.h.

7.4.4.2 #define IO_ADC_CURRENT 0x01U

Current loop configuration

use this configuration if the connected sensor delivers a current signal (4..25mA sensors)

Task function returns current in [uA]

Definition at line 167 of file IO_ADC.h.

7.4.4.3 #define IO_ADC_NO_PULL 0x00U

fixed pull resistor

Definition at line 189 of file IO_ADC.h.

7.4.4.4 #define IO_ADC_NO_RANGE 0x00U

ADC range not configurable

Definition at line 203 of file IO_ADC.h.

7.4.4.5 #define IO_ADC_PD_10K 0x01U

10 kOhm pull down

Definition at line 191 of file IO_ADC.h.

7.4.4.6 #define IO_ADC_PU_10K 0x02U

10 kOhm pull up

Definition at line 190 of file IO_ADC.h.

7.4.4.7 #define IO_ADC_RANGE_10V 0x02U

ADC range 0V .. 10V

Definition at line 205 of file IO_ADC.h.

7.4.4.8 #define IO_ADC_RANGE_32V 0x03U

ADC range 0V .. 32V

Definition at line 206 of file IO_ADC.h.

7.4.4.9 #define IO_ADC_RANGE_5V 0x01U

ADC range 0V .. 5V

Definition at line 204 of file IO_ADC.h.

7.4.4.10 #define IO_ADC_RATIOMETRIC 0x00U

Ratiometric configuration

use this configuration if the connected sensor is supplied by one of the sensor supplies ([IO_SENSOR_SUPPLY_0](#) or [IO_SENSOR_SUPPLY_1](#) or [IO_SENSOR_SUPPLY_2](#)) and delivers a voltage signal.

Task function returns voltage in [mV]

Definition at line 162 of file IO_ADC.h.

7.4.4.11 #define IO_ADC_RESISTIVE 0x02U

Resistive configuration

use this configuration if the sensor value shall be determined by measuring its resistance

Task function returns resistance in [Ohm]

Definition at line 172 of file IO_ADC.h.

7.4.5 Typedef Documentation**7.4.5.1 typedef struct io_adc_safety_conf_ IO_ADC_SAFETY_CONF**

Safety configuration for the ADC inputs.

Stores all relevant safety configuration parameters for the ADC inputs. The internal checker modules verify that these inputs contain valid values

Attention

For the safety configuration of a 2 mode ADC channel, the following rules need to be fulfilled:

- The primary channel needs to specify a redundant channel (`IO_ADC_08 .. IO_ADC_23`) at the `redundant_channel` field.
- The redundant channel must not reference a redundant channel by itself. Thus `IO_PIN_NONE` shall be used for the redundant channel.
- Every redundant channel can only be used once as redundant channel.
- It's not allowed to have a redundant channel which is not used by a primary channel.
- If the ratiometric measurement mode is used, the primary and the redundant channel must use different sensor supplies (but not `IO_SENSOR_SUPPLY_2`).

For the safety configuration of a 3 mode ADC channel, the following rules need to be fulfilled:

- The channel must have `IO_PIN_NONE` in the `redundant_channel` field.
- If the ratiometric measurement mode is used, the channel must not use `IO_SENSOR_SUPPLY_2`.

7.4.6 Function Documentation

7.4.6.1 float4 IO_ADC_BoardTempFloat (ubyte4 raw_value)

Calculates the board temperature in degree Celsius.

The function converts the raw ADC value (retrieved from `IO_ADC_Get()`) to a temperature in degree Celsius and returns it as a float value.

Parameters

<code>raw_value</code>	raw adc board temperature returned from the <code>IO_ADC_Get()</code> function
------------------------	--

Returns

the board temperature in degree Celsius (-63.00 .. 152.50 degree C)

Remarks

Usage:

```

1 IO_ADC_Get(IO_ADC_BOARD_TEMP,
2         &raw_value,
3         &fresh);
4
5 temp = IO_ADC_BoardTempFloat(raw_value);

```

7.4.6.2 sbyte2 IO_ADC_BoardTempSbyte (ubyte4 raw_value)

Calculates the board temperature in degree Celsius.

The function converts the raw ADC value (retrieved from `IO_ADC_Get()`) to a temperature in degree Celsius and returns it as a signed value.

Parameters

<code>raw_value</code>	raw adc board temperature returned from the <code>IO_ADC_Get()</code> function
------------------------	--

Returns

the board temperature in degree Celsius (-63 .. 152 degree C)

Remarks

Usage:

```
1 IO_ADC_Get(IO_ADC_BOARD_TEMP,
2         &raw_value,
3         &fresh);
4
5 temp = IO_ADC_BoardTempSbyte(raw_value);
```

7.4.6.3 IO_ErrorType IO_ADC_ChannelDeInit (ubyte1 *adc_channel*)

Deinitializes one ADC input.

deinitializes the given ADC channel, allows reconfiguration by `IO_ADC_ChannelInit()`

Parameters

<code>adc_channel</code>	ADC channel, one of: <ul style="list-style-type: none"> • <code>IO_ADC_00 .. IO_ADC_07</code> • <code>IO_ADC_08 .. IO_ADC_15</code> • <code>IO_ADC_16 .. IO_ADC_23</code> • <code>IO_ADC_24 .. IO_ADC_35</code> • <code>IO_ADC_36 .. IO_ADC_43</code> • <code>IO_ADC_44 .. IO_ADC_51</code> • <code>IO_ADC_52 .. IO_ADC_59</code> • <code>IO_ADC_SENSOR_SUPPLY_0</code> • <code>IO_ADC_SENSOR_SUPPLY_1</code> • <code>IO_ADC_SENSOR_SUPPLY_2</code> • <code>IO_ADC_K15</code> • <code>IO_ADC_WAKE_UP</code> • <code>IO_ADC_UBAT</code> • <code>IO_ADC_SAFETY_SW_0</code> • <code>IO_ADC_SAFETY_SW_1</code> • <code>IO_ADC_SAFETY_SW_2</code> • <code>IO_ADC_BOARD_TEMP</code>
--------------------------	---

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist
<code>IO_E_CH_CAPABILITY</code>	the given channel is not an ADC channel
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the given channel is not configured
<code>IO_E_DRIVER_NOT_INITIALIZED</code>	the common driver init function has not been called before
<code>IO_E_UNKNOWN</code>	an unknown error occurred

7.4.6.4 IO_ErrorType IO_ADC_ChannelInit (ubyte1 *adc_channel*, ubyte1 *type*, ubyte1 *range*, ubyte1 *pupd*, ubyte1 *sensor_supply*, const IO_ADC_SAFETY_CONF *const *safety_conf*)

Setup one ADC channel.

Parameters

<i>adc_channel</i>	ADC channel, one of: <ul style="list-style-type: none"> • <code>IO_ADC_00 .. IO_ADC_07</code> • <code>IO_ADC_08 .. IO_ADC_15</code> • <code>IO_ADC_16 .. IO_ADC_23</code> • <code>IO_ADC_24 .. IO_ADC_35</code> • <code>IO_ADC_36 .. IO_ADC_43</code> • <code>IO_ADC_44 .. IO_ADC_51</code> • <code>IO_ADC_52 .. IO_ADC_59</code> • <code>IO_ADC_SENSOR_SUPPLY_0</code> • <code>IO_ADC_SENSOR_SUPPLY_1</code> • <code>IO_ADC_SENSOR_SUPPLY_2</code> • <code>IO_ADC_K15</code> • <code>IO_ADC_WAKE_UP</code> • <code>IO_ADC_UBAT</code> • <code>IO_ADC_SAFETY_SW_0</code> • <code>IO_ADC_SAFETY_SW_1</code> • <code>IO_ADC_SAFETY_SW_2</code> • <code>IO_ADC_BOARD_TEMP</code>
<i>type</i>	Type of input: <ul style="list-style-type: none"> • <code>IO_ADC_RATIOMETRIC</code>: voltage input on sensor supply • <code>IO_ADC_CURRENT</code>: 0-25mA input • <code>IO_ADC_RESISTIVE</code>: 0-100000Ohm input • <code>IO_ADC_ABSOLUTE</code>: normal voltage input
<i>range</i>	Range of input: <ul style="list-style-type: none"> • <code>IO_ADC_NO_RANGE</code>: voltage input with fixed range • <code>IO_ADC_RANGE_5V</code>: voltage input 0-5000mV • <code>IO_ADC_RANGE_10V</code>: voltage input 0-10200mV • <code>IO_ADC_RANGE_32V</code>: voltage input 0-32000mV
<i>pupd</i>	Pull up/down configuration: <ul style="list-style-type: none"> • <code>IO_ADC_NO_PULL</code>: fixed pull resistor • <code>IO_ADC_PU_10K</code>: pull up 10 kOhm • <code>IO_ADC_PD_10K</code>: pull down 10 kOhm

	<i>sensor_supply</i>	Sensor supply: <ul style="list-style-type: none"> • <code>IO_SENSOR_SUPPLY_0</code>: 5V • <code>IO_SENSOR_SUPPLY_1</code>: 5V • <code>IO_SENSOR_SUPPLY_2</code>: variable • <code>IO_PIN_NONE</code>: no sensor supply is used
in	<i>safety_conf</i>	Relevant safety configurations for the checker modules. The following ADC channels can be configured safety relevant: <ul style="list-style-type: none"> • <code>IO_ADC_00 .. IO_ADC_07</code> • <code>IO_ADC_08 .. IO_ADC_15</code> • <code>IO_ADC_16 .. IO_ADC_23</code>

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist or is not available on the used ECU variant
<code>IO_E_CH_CAPABILITY</code>	the given channel is not an ADC channel or the used ECU variant does not support this function
<code>IO_E_INVALID_PARAMETER</code>	parameter is out of range
<code>IO_E_CHANNEL_BUSY</code>	the ADC input channel is currently used by another function
<code>IO_E_DRIVER_NOT_INITIALIZED</code>	the common driver init function has not been called before
<code>IO_E_INVALID_SAFETY_CONFIG</code>	an invalid safety configuration has been passed
<code>IO_E_DRV_SAFETY_CONF_NOT_CONFIG</code>	global safety configuration is missing
<code>IO_E_DRV_SAFETY_CYCLE_RUNNING</code>	the init function was called after the task begin function
<code>IO_E_UNKNOWN</code>	an unknown error occurred

Remarks

- The supported features depend on the selected channel:
 - `IO_ADC_00 .. IO_ADC_07`:
 - * type: `IO_ADC_RATIOMETRIC`, `IO_ADC_CURRENT`, `IO_ADC_RESISTIVE` or `IO_ADC_ABSOLUTE`
 - * `sensor_supply`: `IO_SENSOR_SUPPLY_0`, `IO_SENSOR_SUPPLY_1` or `IO_SENSOR_SUPPLY_2`
 - * `safety_conf`: Safety configuration
 - `IO_ADC_08 .. IO_ADC_15`:
 - * type: `IO_ADC_RATIOMETRIC`, `IO_ADC_CURRENT` or `IO_ADC_ABSOLUTE`
 - * range: `IO_ADC_RANGE_5V` or `IO_ADC_RANGE_10V` (supported for type `IO_ADC_ABSOLUTE` and `IO_ADC_RATIOMETRIC`)
 - * `sensor_supply`: `IO_SENSOR_SUPPLY_0`, `IO_SENSOR_SUPPLY_1` or `IO_SENSOR_SUPPLY_2`

- * safety_conf: Safety configuration
- IO_ADC_16 .. IO_ADC_23:
 - * type: IO_ADC_RATIOMETRIC, IO_ADC_CURRENT or IO_ADC_ABSOLUTE
 - * range: IO_ADC_RANGE_5V or IO_ADC_RANGE_32V (supported for type IO_ADC_ABSOLUTE and IO_ADC_RATIOMETRIC)
 - * sensor_supply: IO_SENSOR_SUPPLY_0, IO_SENSOR_SUPPLY_1 or IO_SENSOR_SUPPLY_2
 - * safety_conf: Safety configuration
- IO_ADC_24 .. IO_ADC_35
 - * type: IO_ADC_RATIOMETRIC or IO_ADC_ABSOLUTE
 - * sensor_supply: IO_SENSOR_SUPPLY_0, IO_SENSOR_SUPPLY_1 or IO_SENSOR_SUPPLY_2 (supported for type IO_ADC_RATIOMETRIC)
 - * pupd: IO_ADC_PU_10K or IO_ADC_PD_10K
- IO_ADC_36 .. IO_ADC_43
 - * type: IO_ADC_RATIOMETRIC or IO_ADC_ABSOLUTE
 - * sensor_supply: IO_SENSOR_SUPPLY_0, IO_SENSOR_SUPPLY_1 or IO_SENSOR_SUPPLY_2 (supported for type IO_ADC_RATIOMETRIC)
 - * pupd: IO_ADC_PU_10K or IO_ADC_PD_10K
- IO_ADC_44 .. IO_ADC_51
 - * type: IO_ADC_RATIOMETRIC or IO_ADC_ABSOLUTE
 - * sensor_supply: IO_SENSOR_SUPPLY_0, IO_SENSOR_SUPPLY_1 or IO_SENSOR_SUPPLY_2 (supported for type IO_ADC_RATIOMETRIC)
- IO_ADC_52 .. IO_ADC_59
 - * type: IO_ADC_RATIOMETRIC or IO_ADC_ABSOLUTE
 - * sensor_supply: IO_SENSOR_SUPPLY_0, IO_SENSOR_SUPPLY_1 or IO_SENSOR_SUPPLY_2 (supported for type IO_ADC_RATIOMETRIC)
 - * pupd: IO_ADC_PU_10K or IO_ADC_PD_10K
- IO_ADC_SENSOR_SUPPLY_0:
 - * type: IO_ADC_ABSOLUTE
- IO_ADC_SENSOR_SUPPLY_1:
 - * type: IO_ADC_ABSOLUTE
- IO_ADC_SENSOR_SUPPLY_2:
 - * type: IO_ADC_ABSOLUTE
- IO_ADC_K15
 - * type: IO_ADC_ABSOLUTE
- IO_ADC_WAKE_UP
 - * type: IO_ADC_ABSOLUTE
- IO_ADC_UBAT
 - * type: IO_ADC_ABSOLUTE
- IO_ADC_SAFETY_SW_0
 - * type: IO_ADC_ABSOLUTE
- IO_ADC_SAFETY_SW_1
 - * type: IO_ADC_ABSOLUTE
- IO_ADC_SAFETY_SW_2
 - * type: IO_ADC_ABSOLUTE

- * type: `IO_ADC_ABSOLUTE`
- `IO_ADC_BOARD_TEMP`
 - * type: `IO_ADC_ABSOLUTE`
- If a channel does not support a function, the value of the associated parameter will be ignored.

Attention

For safety critical configuration (`safety_conf != NULL`) following restrictions additionally apply:

- Resistive measurement type `IO_ADC_RESISTIVE` is not valid
- Variable sensor supply `IO_SENSOR_SUPPLY_2` is not valid

Note

If a channel is configured as safety critical and a sensor supply was configured, also the sensor supply voltage will be checked by the diagnostic modules.

7.4.6.5 IO_ErrorType IO_ADC_Get (ubyte1 adc_channel, ubyte4 *const adc_value, bool *const fresh)

Returns the value of the given ADC channel.

Parameters

<i>adc_channel</i>	ADC channel, one of: <ul style="list-style-type: none"> • <code>IO_ADC_00 .. IO_ADC_07</code> • <code>IO_ADC_08 .. IO_ADC_15</code> • <code>IO_ADC_16 .. IO_ADC_23</code> • <code>IO_ADC_24 .. IO_ADC_35</code> • <code>IO_ADC_36 .. IO_ADC_43</code> • <code>IO_ADC_44 .. IO_ADC_51</code> • <code>IO_ADC_52 .. IO_ADC_59</code> • <code>IO_ADC_SENSOR_SUPPLY_0</code> • <code>IO_ADC_SENSOR_SUPPLY_1</code> • <code>IO_ADC_SENSOR_SUPPLY_2</code> • <code>IO_ADC_K15</code> • <code>IO_ADC_WAKE_UP</code> • <code>IO_ADC_UBAT</code> • <code>IO_ADC_SAFETY_SW_0</code> • <code>IO_ADC_SAFETY_SW_1</code> • <code>IO_ADC_SAFETY_SW_2</code> • <code>IO_ADC_BOARD_TEMP</code>
--------------------	---

out	<i>adc_value</i>	ADC value, the range depends on the input group and its configuration (type parameter of <code>IO_ADC_ChannelInit()</code>): <ul style="list-style-type: none"> • <code>IO_ADC_00 .. IO_ADC_07</code> <ul style="list-style-type: none"> – <code>IO_ADC_RATIOMETRIC</code>: 0..5000 (0V..5.000V) – <code>IO_ADC_CURRENT</code>: 0.25000 (0mA..25.000mA) – <code>IO_ADC_RESISTIVE</code>: 0..100000 (0Ohm..100000Ohm) – <code>IO_ADC_ABSOLUTE</code>: 0..5000 (0V..5.000V) • <code>IO_ADC_08 .. IO_ADC_15</code> <ul style="list-style-type: none"> – <code>IO_ADC_RATIOMETRIC</code>: 0..5000 (0V..5.000V) – <code>IO_ADC_CURRENT</code>: 0.25000 (0mA..25.000mA) – <code>IO_ADC_ABSOLUTE</code>: 0..10200 (0V..10.200V) • <code>IO_ADC_16 .. IO_ADC_23</code> <ul style="list-style-type: none"> – <code>IO_ADC_RATIOMETRIC</code>: 0..5000 (0V..5.000V) – <code>IO_ADC_CURRENT</code>: 0..25000 (0mA..25.000mA) – <code>IO_ADC_ABSOLUTE</code>: 0..32000 (0V..32.000V) • <code>IO_ADC_24 .. IO_ADC_35</code> <ul style="list-style-type: none"> – <code>IO_ADC_RATIOMETRIC</code>: 0..32000 (0V..32.000V) – <code>IO_ADC_ABSOLUTE</code>: 0..32000 (0V..32.000V) • <code>IO_ADC_36 .. IO_ADC_43</code> <ul style="list-style-type: none"> – <code>IO_ADC_RATIOMETRIC</code>: 0..32000 (0V..32.000V) – <code>IO_ADC_ABSOLUTE</code>: 0..32000 (0V..32.000V) • <code>IO_ADC_44 .. IO_ADC_51</code> <ul style="list-style-type: none"> – <code>IO_ADC_RATIOMETRIC</code>: 0..32000 (0V..32.000V) – <code>IO_ADC_ABSOLUTE</code>: 0..32000 (0V..32.000V) • <code>IO_ADC_52 .. IO_ADC_59</code> <ul style="list-style-type: none"> – <code>IO_ADC_RATIOMETRIC</code>: 0..32000 (0V..32.000V) – <code>IO_ADC_ABSOLUTE</code>: 0..32000 (0V..32.000V) • <code>IO_ADC_SENSOR_SUPPLY_0</code> <ul style="list-style-type: none"> – <code>IO_ADC_ABSOLUTE</code>: 0..5254 (0V..5.254V) • <code>IO_ADC_SENSOR_SUPPLY_1</code> <ul style="list-style-type: none"> – <code>IO_ADC_ABSOLUTE</code>: 0..5254 (0V..5.254V) • <code>IO_ADC_SENSOR_SUPPLY_2</code> <ul style="list-style-type: none"> – <code>IO_ADC_ABSOLUTE</code>: 0..10560 (0V..10.560V) • <code>IO_ADC_K15</code> <ul style="list-style-type: none"> – <code>IO_ADC_ABSOLUTE</code>: 0..55000 (0V..55.000V) • <code>IO_ADC_WAKE_UP</code> <ul style="list-style-type: none"> – <code>IO_ADC_ABSOLUTE</code>: 0..55000 (0V..55.000V) • <code>IO_ADC_UBAT</code> <ul style="list-style-type: none"> – <code>IO_ADC_ABSOLUTE</code>: 0..55000 (0V..55.000V) • <code>IO_ADC_SAFETY_SW_0</code> <ul style="list-style-type: none"> – <code>IO_ADC_ABSOLUTE</code>: 0..55000 (0V..55.000V) • <code>IO_ADC_SAFETY_SW_1</code> <ul style="list-style-type: none"> – <code>IO_ADC_ABSOLUTE</code>: 0..55000 (0V..55.000V) • <code>IO_ADC_SAFETY_SW_2</code> <ul style="list-style-type: none"> – <code>IO_ADC_ABSOLUTE</code>: 0..55000 (0V..55.000V) • <code>IO_ADC_BOARD_TEMP</code> <ul style="list-style-type: none"> – <code>IO_ADC_ABSOLUTE</code>: 3700..25250 (-63.00..152.50 degree C)
out	<i>fresh</i>	Status of the ADC value <ul style="list-style-type: none"> • <code>TRUE</code>: ADC value is fresh (channel has been converted) • <code>FALSE</code>: ADC value is old (channel has not been converted)

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist
<code>IO_E_CH_CAPABILITY</code>	the given channel is not an ADC channel
<code>IO_E_STARTUP</code>	the input is in the startup phase
<code>IO_E_FET_PROT_ACTIVE</code>	the input FET protection is active and has deactivated the internal switch. It can be reset with <code>IO_ADC_ResetProtection()</code> after the wait time is passed
<code>IO_E_FET_PROT_REENABLE</code>	the input FET protection is ready to be reset with <code>IO_ADC_ResetProtection()</code>
<code>IO_E_FET_PROT_PERMANENT</code>	the input FET is protected permanently because it was already reset more than 10 times
<code>IO_E_REFERENCE</code>	A reference voltage (sensor supply or internal reference) is out of range.
<code>IO_E_NULL_POINTER</code>	a NULL pointer has been passed
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the given channel is not configured
<code>IO_E_DRIVER_NOT_INITIALIZED</code>	the common driver init function has not been called before
<code>IO_E_UNKNOWN</code>	an unknown error occurred

Remarks

- If there is no new measurement value available (for example the function `IO_ADC_Get()` gets called more frequently than the AD sampling) the flag `fresh` will be set to `FALSE`.
- The temperature value in degree C must be recalculated in the following manner, since the value has an offset of 10000 and is multiplied with 100:

$$\text{degree_value} = ((\text{float4}) \text{adc_value} - 10000) / 100$$

7.4.6.6 `IO_ErrorType IO_ADC_ResetProtection (ubyte1 adc_channel, ubyte1 *const reset_cnt)`

Reset the FET protection for an ADC channel.

Parameters

	<code>adc_channel</code>	ADC channel: <ul style="list-style-type: none"> • <code>IO_ADC_00 .. IO_ADC_07</code>: – configured type: <code>IO_ADC_CURRENT</code> • <code>IO_ADC_08 .. IO_ADC_15</code>: – configured type: <code>IO_ADC_CURRENT</code> • <code>IO_ADC_16 .. IO_ADC_23</code>: – configured type: <code>IO_ADC_CURRENT</code>
out	<code>reset_cnt</code>	Protection reset counter. Indicates how often the application already reset the protection.

Returns

[IO_ErrorType](#)

Return values

IO_E_OK	everything fine
IO_E_INVALID_CHANNEL_ID	the given channel id does not exist
IO_E_FET_PROT_NOT_ACTIVE	no input FET protection is active
IO_E_FET_PROT_PERMANENT	the input FET is protected permanently because it was already reset more than 10 times
IO_E_FET_PROT_WAIT	the input FET protection can not be reset, as the wait time of 1s is not already passed
IO_E_CHANNEL_NOT_CONFIGURED	the given channel is not configured
IO_E_CH_CAPABILITY	the given channel has no FET protection
IO_E_UNKNOWN	an unknown error occurred

Attention

The protection can be reset 10 times, afterwards the input will remain permanently protected

Note

- The input can only enter a protection state if configured in current mode
- After entering the input protection, a certain time has to pass before the input protection can be reset:
 - 1s for [IO_ADC_00 .. IO_ADC_07](#)
 - 1s for [IO_ADC_08 .. IO_ADC_15](#)
 - 1s for [IO_ADC_16 .. IO_ADC_23](#)
- This function will close the internal switch to enable the input back again. After calling this function new measurement values can be read with [IO_ADC_Get\(\)](#)

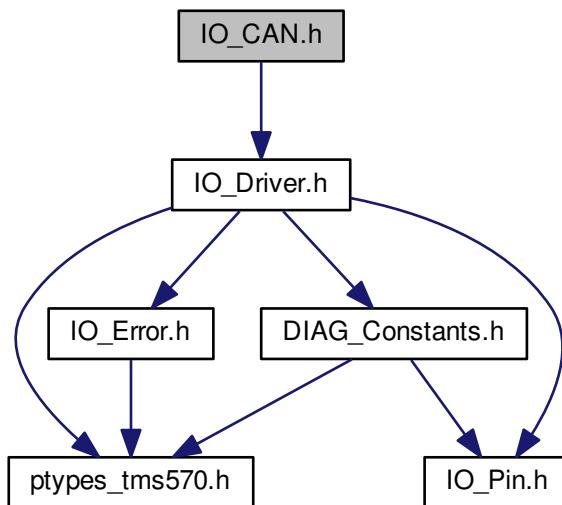
Remarks

If the parameter `reset_cnt` is `NULL`, the parameter is ignored. The parameter `reset_cnt` returns the number of resets already performed.

7.5 IO_CAN.h File Reference

IO Driver functions for CAN communication.

Include dependency graph for IO_CAN.h:



Data Structures

- struct `io_can_data_frame_`
CAN data frame.

Typedefs

- typedef struct `io_can_data_frame_` `IO_CAN_DATA_FRAME`
CAN data frame.

Functions

- `IO_ErrorType IO_CAN_ConfigFIFO (ubyte2 *const handle, ubyte1 channel, ubyte1 size, ubyte1 id_format, ubyte4 id, ubyte4 ac_mask)`
Configures a receive FIFO buffer for the given CAN channel.
- `IO_ErrorType IO_CAN_ConfigMsg (ubyte2 *const handle, ubyte1 channel, ubyte1 mode, ubyte1 id_format, ubyte4 id, ubyte4 ac_mask)`
Configures a message object for the given CAN channel.
- `IO_ErrorType IO_CAN_Delnit (ubyte1 channel)`
Deinitializes the given CAN channel.
- `IO_ErrorType IO_CAN_DelnitHandle (ubyte2 handle)`
Deinitializes a single message handle.
- `IO_ErrorType IO_CAN_FIFOStatus (ubyte2 handle)`
Returns the status of a FIFO buffer.
- `IO_ErrorType IO_CAN_Init (ubyte1 channel, ubyte2 baudrate, ubyte1 tseg1, ubyte1 tseg2, ubyte1 sjw, ubyte1 brp)`
Initialization of the CAN communication driver.
- `IO_ErrorType IO_CAN_MsgStatus (ubyte2 handle)`
Returns the status of a message buffer object.
- `IO_ErrorType IO_CAN_ReadFIFO (ubyte2 handle, IO_CAN_DATA_FRAME *const buffer, ubyte1 buffer_size, ubyte1 *const rx_frames)`
Reads the data from a FIFO buffer.
- `IO_ErrorType IO_CAN_ReadMsg (ubyte2 handle, IO_CAN_DATA_FRAME *const buffer)`
Reads a message from a given message object.
- `IO_ErrorType IO_CAN_Status (ubyte1 channel, ubyte1 *const rx_error_counter, ubyte1 *const tx_error_counter)`
Returns the error counters of the CAN channel.
- `IO_ErrorType IO_CAN_WriteMsg (ubyte2 handle, const IO_CAN_DATA_FRAME *const data)`
Transmits a CAN message, using the given channel and message object.

Message buffer direction

Selects the transmission direction of a CAN message buffer

- `#define IO_CAN_MSG_READ 0U`
- `#define IO_CAN_MSG_WRITE 1U`

CAN frame format

Selects the format for a CAN frame

- `#define IO_CAN_STD_FRAME 0U`
- `#define IO_CAN_EXT_FRAME 1U`

CAN baudrate

Selects the baudrate for a CAN channel

- `#define IO_CAN_BIT_USER 0U`
- `#define IO_CAN_BIT_50_KB 50U`
- `#define IO_CAN_BIT_100_KB 100U`
- `#define IO_CAN_BIT_125_KB 125U`

- #define `IO_CAN_BIT_250_KB` 250U
- #define `IO_CAN_BIT_500_KB` 500U
- #define `IO_CAN_BIT_1000_KB` 1000U

7.5.1 Detailed Description

IO Driver functions for CAN communication.

The CAN driver uses the DCAN module of the TMS570 CPU.

The CAN driver supports up to 7 CAN channels. The number of available channels depends on the used HW variant.

All CAN nodes have an own set of message objects and can not be allocated to other CAN nodes. For further details on message objects see [CAN handles and message objects](#).

CAN-API Usage:

- [Examples for CAN API functions](#)

7.5.2 Usage of the acceptance masks

The acceptance mask defines the relevant bits of the CAN ID. A binary 1 marks a relevant bit in the CAN ID on the same position.

Setting all bits of the acceptance mask (0xFFFFFFFF) only accepts the ID set with the ID parameter and rejects all other IDs. Setting the acceptance mask to 0 causes the message buffer to accept any IDs.

Using this mechanism a message buffer can be used to accept a range of CAN IDs.

Example for an 11-bit ID:

```
ac_mask = 0x00000700 = 0 b 0 0000 0000 0000 0000 0111 0000 0000
id      = 0x00000200 = 0 b 0 0000 0000 0000 0000 0010 0000 0000
```

in this example all messages with an ID between 0x200 and 0x2FF are accepted.

7.5.3 CAN handles and message objects

The DCAN module of the TMS570 CPU provides a certain number of so called *CAN message objects*. A message object is a dedicated memory area which is used to store a CAN message. Every message object has its own identifier and acceptance mask. The number of available message objects depends on the CAN channel:

CAN channel	message objects
<code>IO_CAN_CHANNEL_0</code>	64 for receive and transmit
<code>IO_CAN_CHANNEL_1</code>	64 for receive and transmit
<code>IO_CAN_CHANNEL_2</code>	64 for receive and transmit
<code>IO_CAN_CHANNEL_3</code>	24 for receive, 8 for transmit
<code>IO_CAN_CHANNEL_4</code>	24 for receive, 8 for transmit
<code>IO_CAN_CHANNEL_5</code>	24 for receive, 8 for transmit
<code>IO_CAN_CHANNEL_6</code>	24 for receive, 8 for transmit

For receiving and transmitting CAN messages a message object needs to be configured. During the configuration the driver will return a so called *CAN handle*.

The *CAN handle* is a reference to the configured message object and is used used to exchange data with the driver. There are two types of CAN handles:

- Standard message handles for receive and transmit
- FIFO buffer handles for receive and transmit

The number of CAN handles is identical to the number of message objects.

7.5.3.1 Standard message handles

A standard message handle is configured by calling the driver function `IO_CAN_ConfigMsg()`. It references to a single message object which can store *one* CAN message.

For receiving/transmitting data via a single message object the driver functions `IO_CAN_ReadMsg()` and `IO_CAN_WriteMsg()` are provided.

The status of a standard message handle can be checked with the function `IO_CAN_MsgStatus()`.

7.5.3.2 FIFO buffer handles

Attention

The driver supports FIFO buffers only for receive direction.

A FIFO buffer can be created by linking several message objects to one FIFO buffer which is able to store multiple CAN message.

A FIFO buffer handle is configured by calling the function `IO_CAN_ConfigFIFO()`. The desired size of the buffer is passed to the function via a parameter.

For receiving data via FIFO buffers the driver provides the function `IO_CAN_ReadFIFO()`. With this function multiple CAN messages can be retrieved with one function call.

The status of a FIFO buffer handle can be checked with the function `IO_CAN_FIFOSatus()`.

7.5.4 CAN Code Examples

Example for using the CAN API

7.5.4.1 Examples for CAN initialization

```
ubyte2 handle_w, handle_r;  
  
IO_CAN_Init(IO_CAN_CHANNEL_0,  
            IO_CAN_BIT_500_KB,  
            0,      // default  
            0,      // default  
            0,      // default  
            0);    // default
```

```
// standard message buffers

IO_CAN_ConfigMsg(&handle_w,
    IO_CAN_CHANNEL_0, // channel 0
    IO_CAN_MSG_WRITE, // transmit message buffer
    IO_CAN_STD_FRAME, // standard ID
    0,
    0);

IO_CAN_ConfigMsg(&handle_r,
    IO_CAN_CHANNEL_0, // channel 0
    IO_CAN_MSG_READ, // receive message buffer
    IO_CAN_STD_FRAME, // standard ID
    1,
    0x1FFFFFFF); // accept only ID 1
```

7.5.4.2 Example for CAN task function call

```
IO_CAN_DATA_FRAME can_frame;

// check if new message has been received
if (IO_CAN_MsgStatus(handle_r) == IO_E_OK)
{
    // if message has been received, read the message from the buffer
    IO_CAN_ReadMsg(handle_r,
        &can_frame);

    // received message is now stored in can_frame
    // and can be used by the application
}

// assemble CAN frame:
can_frame.id = 1;
can_frame.id_format = IO_CAN_STD_FRAME;
can_frame.length = 4;
can_frame.data[0] = 1;
can_frame.data[1] = 2;
can_frame.data[2] = 3;
can_frame.data[3] = 4;

// transmit message
IO_CAN_WriteMsg(handle_w,
    &can_frame);

// wait until the transmission has been finished:
while (IO_CAN_MsgStatus(handle_w) != IO_E_OK);
```

7.5.4.3 Example for CAN FIFO usage

```
ubyte2 handle_fifo;
ubyte1 rx_count;
IO_CAN_DATA_FRAME fifo_frame[20];

// FIFO message buffer initialization

IO_CAN_ConfigFIFO(&handle_fifo,
    IO_CAN_CHANNEL_0, // channel 0
    20, // 20 items
```

```
    IO_CAN_STD_FRAME, // standard ID
    0x700,           // accept all IDs from
    0x700);          // 0x700 - 0x7FF

while (1)
{
    // Read FIFO buffer
    IO_CAN_ReadFIFO(handle_fifo,
                      &fifo_frame[0],
                      sizeof(fifo_frame) / sizeof(IO_CAN_DATA_FRAME),
                      &rx_count);

    if (rx_count > 0)
    {
        // process received frames
    }
}
```

7.5.5 Macro Definition Documentation

7.5.5.1 #define IO_CAN_BIT_1000_KB 1000U

Configure CAN Channel with 1000 kBit/s

Definition at line 278 of file IO_CAN.h.

7.5.5.2 #define IO_CAN_BIT_100_KB 100U

Configure CAN Channel with 100 kBit/s

Definition at line 274 of file IO_CAN.h.

7.5.5.3 #define IO_CAN_BIT_125_KB 125U

Configure CAN Channel with 125 kBit/s

Definition at line 275 of file IO_CAN.h.

7.5.5.4 #define IO_CAN_BIT_250_KB 250U

Configure CAN Channel with 250 kBit/s

Definition at line 276 of file IO_CAN.h.

7.5.5.5 #define IO_CAN_BIT_500_KB 500U

Configure CAN Channel with 500 kBit/s

Definition at line 277 of file IO_CAN.h.

7.5.5.6 #define IO_CAN_BIT_50KB 50U

Configure CAN Channel with 50 kBit/s

Definition at line 273 of file IO_CAN.h.

7.5.5.7 #define IO_CAN_BIT_USER 0U

Configure CAN Channel with user defined baudrate (tseg1, tseg2, sjw and brp)

Definition at line 270 of file IO_CAN.h.

7.5.5.8 #define IO_CAN_EXT_FRAME 1U

the ID parameter holds an extended (29-bit) ID

Definition at line 259 of file IO_CAN.h.

7.5.5.9 #define IO_CAN_MSG_READ 0U

used to setup a message buffer for receiving

Definition at line 246 of file IO_CAN.h.

7.5.5.10 #define IO_CAN_MSG_WRITE 1U

used to setup a message buffer for transmitting

Definition at line 247 of file IO_CAN.h.

7.5.5.11 #define IO_CAN_STD_FRAME 0U

the ID parameter holds a standard (11-bit) ID

Definition at line 258 of file IO_CAN.h.

7.5.6 Typedef Documentation**7.5.6.1 typedef struct io_can_data_frame_ IO_CAN_DATA_FRAME**

CAN data frame.

Stores a data frame for the CAN communication.

7.5.7 Function Documentation

7.5.7.1 IO_ErrorType IO_CAN_ConfigFIFO (**ubyte2 *const handle**, **ubyte1 channel**, **ubyte1 size**, **ubyte1 id_format**, **ubyte4 id**, **ubyte4 ac_mask**)

Configures a receive FIFO buffer for the given CAN channel.

Returns a FIFO buffer handle.

Parameters

out	<i>handle</i>	Returns the FIFO buffer handle
	<i>channel</i>	CAN channel, one of: <ul style="list-style-type: none"> • IO_CAN_CHANNEL_0 • IO_CAN_CHANNEL_1 • IO_CAN_CHANNEL_2 • IO_CAN_CHANNEL_3 • IO_CAN_CHANNEL_4 • IO_CAN_CHANNEL_5 • IO_CAN_CHANNEL_6
	<i>size</i>	Size of the FIFO buffer: <ul style="list-style-type: none"> • IO_CAN_CHANNEL_0 .. IO_CAN_CHANNEL_2: [2..63] • IO_CAN_CHANNEL_3 .. IO_CAN_CHANNEL_6: [2..24]
	<i>id_format</i>	Format of message identifier, one of: <ul style="list-style-type: none"> • IO_CAN_STD_FRAME, • IO_CAN_EXT_FRAME
	<i>id</i>	CAN message identifier <ul style="list-style-type: none"> • 0..0x7FF for IO_CAN_STD_FRAME • 0..0xFFFFFFFF for IO_CAN_EXT_FRAME
	<i>ac_mask</i>	CAN acceptance mask. Refer to Usage of the acceptance mask " for further details. <ul style="list-style-type: none"> • 0..0x7FF for IO_CAN_STD_FRAME • 0..0xFFFFFFFF for IO_CAN_EXT_FRAME

Returns

[IO_ErrorType](#)

Return values

IO_E_OK	everything fine
IO_E_CAN_MAX_MO_REACHED	no more HW message objects are available
IO_E_CAN_MAX_HANDLES_REACHED	no more handles are available
IO_E_NULL_POINTER	null Pointer has been passed
IO_E_INVALID_CHANNEL_ID	the given channel id does not exist
IO_E_CH_CAPABILITY	the given channel is not a CAN channel
IO_E_INVALID_PARAMETER	invalid parameter has been passed
IO_E_CHANNEL_NOT_CONFIGURED	the given channel was not initialized

<code>IO_E_INTERNAL_CSM</code>	an internal error occurred
<code>IO_E_UNKNOWN</code>	an unknown error occurred

Note

FIFO buffers are only available for receive direction.

Remarks

For the CAN channels `IO_CAN_CHANNEL_0`, `IO_CAN_CHANNEL_1` and `IO_CAN_CHANNEL_2` the function has to reserve internally **size + 1** mailboxes for each configured FIFO. Therefore, on those channels not all **64** mailboxes are available for FIFO usage.

7.5.7.2 `IO_ErrorType IO_CAN_ConfigMsg (ubyte2 *const handle, ubyte1 channel, ubyte1 mode, ubyte1 id_format, ubyte4 id, ubyte4 ac_mask)`

Configures a message object for the given CAN channel.

Returns a message object handle.

Parameters

<code>out handle</code>	Returns the message object handle
<code>channel</code>	CAN channel, one of: <ul style="list-style-type: none">• <code>IO_CAN_CHANNEL_0</code>• <code>IO_CAN_CHANNEL_1</code>• <code>IO_CAN_CHANNEL_2</code>• <code>IO_CAN_CHANNEL_3</code>• <code>IO_CAN_CHANNEL_4</code>• <code>IO_CAN_CHANNEL_5</code>• <code>IO_CAN_CHANNEL_6</code>
<code>mode</code>	Mode for CAN Message, one of: <ul style="list-style-type: none">• <code>IO_CAN_MSG_READ</code>,• <code>IO_CAN_MSG_WRITE</code>
<code>id_format</code>	Format of message identifier, one of: <ul style="list-style-type: none">• <code>IO_CAN_STD_FRAME</code>,• <code>IO_CAN_EXT_FRAME</code>
<code>id</code>	CAN message identifier <ul style="list-style-type: none">• 0..0x7FF for <code>IO_CAN_STD_FRAME</code>• 0..0x1FFFFFFF for <code>IO_CAN_EXT_FRAME</code>
<code>ac_mask</code>	CAN acceptance mask. Refer to Usage of the acceptance mask for further details. <ul style="list-style-type: none">• 0..0x7FF for <code>IO_CAN_STD_FRAME</code>• 0..0x1FFFFFFF for <code>IO_CAN_EXT_FRAME</code>

Returns

[IO_ErrorType](#)

Return values

IO_E_OK	everything fine
IO_E_CAN_MAX_MO_REACHED	no more HW message objects are available
IO_E_CAN_MAX_HANDLES_REACHED	no more handles are available
IO_E_NULL_POINTER	null Pointer has been passed
IO_E_INVALID_CHANNEL_ID	the given channel id does not exist
IO_E_CH_CAPABILITY	the given channel is not a CAN channel
IO_E_INVALID_PARAMETER	invalid parameter has been passed
IO_E_CHANNEL_NOT_CONFIGURED	the given channel was not initialized
IO_E_INTERNAL_CSM	an internal error occurred
IO_E_UNKNOWN	an unknown error occurred

7.5.7.3 [IO_ErrorType IO_CAN_DelInit \(ubyte1 channel \)](#)

Deinitializes the given CAN channel.

Allows re-initialization by [IO_CAN_Init \(\)](#)

Parameters

<i>channel</i>	CAN channel, one of: • IO_CAN_CHANNEL_0 • IO_CAN_CHANNEL_1 • IO_CAN_CHANNEL_2 • IO_CAN_CHANNEL_3 • IO_CAN_CHANNEL_4 • IO_CAN_CHANNEL_5 • IO_CAN_CHANNEL_6
----------------	--

Returns

[IO_ErrorType](#)

Return values

IO_E_OK	everything fine
IO_E_INVALID_CHANNEL_ID	the given channel id does not exist
IO_E_CH_CAPABILITY	the given channel is not a CAN channel
IO_E_CHANNEL_NOT_CONFIGURED	channel has not been initialized
IO_E_CAN_TIMEOUT	the CAN node reported a timeout
IO_E_INTERNAL_CSM	an internal error occurred
IO_E_UNKNOWN	an unknown error occurred

Remarks

This function will also reset all the handles related to this channel.

7.5.7.4 IO_ErrorType IO_CAN_DelInitHandle (ubyte2 handle)

Deinitializes a single message handle.

Allows re-initialization by [IO_CAN_ConfigMsg\(\)](#).

Parameters

<i>handle</i>	CAN message handle (retrieved from IO_CAN_ConfigMsg())
---------------	---

Returns

[IO_ErrorType](#)

Return values

IO_E_OK	everything fine
IO_E_CAN_WRONG_HANDLE	a wrong handle has been passed
IO_E_CHANNEL_NOT_CONFIGURED	handle has not been initialized
IO_E_INTERNAL_CSM	an internal error occurred
IO_E_UNKNOWN	an unknown error occurred

7.5.7.5 IO_ErrorType IO_CAN_FIFOStatus (ubyte2 handle)

Returns the status of a FIFO buffer.

Parameters

<i>handle</i>	CAN FIFO buffer handle (retrieved from IO_CAN_ConfigFIFO())
---------------	--

Returns

[IO_ErrorType](#)

Return values

IO_E_OK	everything fine
IO_E_CAN_FIFO_FULL	the FIFO is full, new messages were lost
IO_E_CAN_OLD_DATA	no new data received
IO_E_CAN_WRONG_HANDLE	invalid handle has been passed
IO_E_CHANNEL_NOT_CONFIGURED	the given handle has not been configured
IO_E_CAN_TIMEOUT	the CAN node reported a timeout
IO_E_INTERNAL_CSM	an internal error occurred
IO_E_UNKNOWN	an unknown error occurred

7.5.7.6 IO_ErrorType IO_CAN_Init (ubyte1 channel, ubyte2 baudrate, ubyte1 tseg1, ubyte1 tseg2, ubyte1 sjw, ubyte1 brp)

Initialization of the CAN communication driver.

The function

- Enables the module
- Sets the module clock to 16MHz
- Automatically sets up the bit timing for the given baudrate

Parameters

<i>channel</i>	CAN channel, one of: • IO_CAN_CHANNEL_0 • IO_CAN_CHANNEL_1 • IO_CAN_CHANNEL_2 • IO_CAN_CHANNEL_3 • IO_CAN_CHANNEL_4 • IO_CAN_CHANNEL_5 • IO_CAN_CHANNEL_6
<i>baudrate</i>	Baud rate in kbit/s, one of: • IO_CAN_BIT_50_KB • IO_CAN_BIT_100_KB • IO_CAN_BIT_125_KB • IO_CAN_BIT_250_KB • IO_CAN_BIT_500_KB • IO_CAN_BIT_1000_KB • IO_CAN_BIT_USER
<i>tseg1</i>	Time segment before sample point (3 ... 16)
<i>tseg2</i>	Time segment after sample point (2 ... 8)
<i>sjw</i>	Synchronization jump width (1 ... 4)
<i>brp</i>	Baud rate prescaler (1 ... 64)

Returns

[IO_ErrorType](#)

Return values

IO_E_OK	everything fine
IO_E_INVALID_PARAMETER	invalid parameter has been passed
IO_E_INVALID_CHANNEL_ID	the given channel id does not exist or is not available on the used ECU variant
IO_E_CH_CAPABILITY	the given channel is not a CAN channel
IO_E_CHANNEL_BUSY	channel has been initialized before
IO_E_DRIVER_NOT_INITIALIZED	the common driver init function has not been called before
IO_E_FPGA_NOT_INITIALIZED	the FPGA has not been initialized
IO_E_CAN_TIMEOUT	the CAN node reported a timeout
IO_E_INTERNAL_CSM	an internal error occurred
IO_E_UNKNOWN	an unknown error occurred

Remarks

- Module is initialized only once. To re-initialize the module, the function `IO_CAN_DelInit()` needs to be called.
- For baudrates unequal to `IO_CAN_BIT_USER` the following bit timings are set, independent from the user configuration:
 - tseg1 = 6
 - tseg2 = 1
 - sjw = 1
 - Sampling Point: 87.5%
- Set the parameter baudrate to `IO_CAN_BIT_USER`, to set custom bit timings. The following has to be fulfilled:
 - tseg1 >= tseg2
 - tseg2 > sjw
- The timing parameters and baudrate are calculated as follows:
 The time quanta "tq" results from the baudrate prescaler:
 $tq = (1 / 16,000,000Hz) * (brp * 2)$

The synchronization time (**Tsync [s]**), Phase Buffer Segment Time 1 (**Tseg1 [s]**) and Phase Buffer Segment Time (**Tseg2 [s]**) are calculated as follows based upon "tq":

$$\begin{aligned} \text{Tsync} &= 1 * tq \\ \text{TSeg1} &= \text{tseg1} * tq \\ \text{TSeg2} &= \text{tseg2} * tq \end{aligned}$$

The overall **bit time [s]** and **baudrate [bit/s]** are calculated with:

$$\begin{aligned} \text{bit_time} &= \text{Tsync} + \text{TSeg1} + \text{TSeg2} \\ \text{baudrate} &= 1 / \text{bit_time} \end{aligned}$$

Example:

$$\begin{aligned} \text{brp} &= 4, \text{tseg1} = 3, \text{tseg2} = 4 \text{ and sjw} = 1 \\ \text{tq} &= (1 / 16,000,000Hz) * (4 * 2) = 500\text{ns} \\ \text{Tsync} &= 1 * 500\text{ns} = 500\text{ns} \\ \text{TSeg1} &= 3 * 500\text{ns} = 1500\text{ns} \\ \text{TSeg2} &= 4 * 500\text{ns} = 2000\text{ns} \end{aligned}$$

$$\begin{aligned} \text{bit_time} &= 500\text{ns} + 1500\text{ns} + 2000\text{ns} = 4000\text{ns} \\ \text{baudrate} &= 1/4000\text{ns} = 250,000 \text{ bit/s} \end{aligned}$$

The **sampling point** can be calculated with:

$$\text{sampling_point} = (1 + \text{tseg1}) / (1 + \text{tseg1} + \text{tseg2})$$

7.5.7.7 IO_ErrorType IO_CAN_MsgStatus (*ubyte2 handle*)

Returns the status of a message buffer object.

Parameters

<i>handle</i>	CAN message object handle (retrieved from IO_CAN_ConfigMsg())
---------------	--

Returns

[IO_ErrorType](#)

Return values

IO_E_OK	everything fine
IO_E_BUSY	transmission is ongoing
IO_E_CAN_OVERFLOW	message object overflow
IO_E_CAN_OLD_DATA	no new data received
IO_E_CAN_WRONG_HANDLE	invalid handle has been passed
IO_E_CHANNEL_NOT_CONFIGURED	the given handle has not been configured
IO_E_INTERNAL_CSM	an internal error occurred
IO_E_UNKNOWN	an unknown error occurred

7.5.7.8 IO_ErrorType IO_CAN_ReadFIFO (*ubyte2 handle*, *IO_CAN_DATA_FRAME *const buffer*, *ubyte1 buffer_size*, *ubyte1 *const rx_frames*)

Reads the data from a FIFO buffer.

Copies all received CAN frames from a given FIFO buffer since the last call.

Parameters

	<i>handle</i>	CAN FIFO buffer handle (retrieved from IO_CAN_ConfigFIFO())
out	<i>buffer</i>	Pointer to data buffer structure. The received frames will be stored there.
	<i>buffer_size</i>	Size of <i>buffer</i> . The buffer has to be of the same length as the FIFO was configured.
out	<i>rx_frames</i>	Number of frames which have been received since the last call and copied to <i>buffer</i>

Returns

[IO_ErrorType](#)

Return values

IO_E_OK	everything fine
IO_E_NULL_POINTER	null pointer has been passed to function
IO_E_INVALID_PARAMETER	the buffer size does not match with the configuration
IO_E_CAN_FIFO_FULL	the FIFO is full, new messages were lost
IO_E_CAN_OLD_DATA	no new data received
IO_E_CAN_WRONG_HANDLE	invalid handle has been passed

<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the given handle has not been configured
<code>IO_E_CAN_TIMEOUT</code>	the CAN node reported a timeout
<code>IO_E_INTERNAL_CSM</code>	an internal error occurred
<code>IO_E_UNKNOWN</code>	an unknown error occurred

Remarks

The complete FIFO buffers has to be read with this function call. The provided buffer `buffer` has to be of the same size as the configured FIFO size.

7.5.7.9 `IO_ErrorType IO_CAN_ReadMsg (ubyte2 handle, IO_CAN_DATA_FRAME *const buffer)`

Reads a message from a given message object.

Returns the data of a message object and whether the message is new or not.

Parameters

	<code>handle</code>	CAN message object handle (retrieved from <code>IO_CAN_ConfigMsg()</code>)
out	<code>buffer</code>	Pointer to data buffer structure. The received frame will be stored there.

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_CAN_OVERFLOW</code>	overflow of message object
<code>IO_E_CAN_OLD_DATA</code>	no new data has been received since the last read
<code>IO_E_CAN_WRONG_HANDLE</code>	invalid handle has been passed
<code>IO_E_NULL_POINTER</code>	null pointer has been passed
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the given handle was not configured
<code>IO_E_CAN_TIMEOUT</code>	the CAN node reported a timeout
<code>IO_E_INTERNAL_CSM</code>	an internal error occurred
<code>IO_E_UNKNOWN</code>	an unknown error occurred

7.5.7.10 `IO_ErrorType IO_CAN_Status (ubyte1 channel, ubyte1 *const rx_error_counter, ubyte1 *const tx_error_counter)`

Returns the error counters of the CAN channel.

Parameters

	<i>channel</i>	CAN channel, one of: <ul style="list-style-type: none"> • <code>IO_CAN_CHANNEL_0</code> • <code>IO_CAN_CHANNEL_1</code> • <code>IO_CAN_CHANNEL_2</code> • <code>IO_CAN_CHANNEL_3</code> • <code>IO_CAN_CHANNEL_4</code> • <code>IO_CAN_CHANNEL_5</code> • <code>IO_CAN_CHANNEL_6</code>
out	<i>rx_error_counter</i>	Value of the receive error counter
out	<i>tx_error_counter</i>	Value of the transmit error counter

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_CAN_ERROR_WARNING</code>	error counter has reached the warning limit, controller is still in active state
<code>IO_E_CAN_ERROR_PASSIVE</code>	controller is in error passive state
<code>IO_E_CAN_BUS_OFF</code>	controller is in bus off state
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist
<code>IO_E_CH_CAPABILITY</code>	the given channel is not a CAN channel
<code>IO_E_NULL_POINTER</code>	null pointer has been passed
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the given channel has not been initialized
<code>IO_E_UNKNOWN</code>	an unknown error occurred

Attention

- If the CAN is in bus off, it will automatically recover after 128 occurrences of bus idle (128 • 11 consecutive recessive bits). During this state messages can neither be received or transmitted.

Note

If the CAN is in error passive, the transmitter may still transmit CAN frames and passive error frames.

7.5.7.11 `IO_ErrorType IO_CAN_WriteMsg (ubyte2 handle, const IO_CAN_DATA_FRAME *const data)`

Transmits a CAN message, using the given channel and message object.

Returns whether the transmission has been started successfully or not.

Parameters

	<i>handle</i>	CAN message object handle (retrieved from <code>IO_CAN_ConfigMsg()</code>)
in	<i>data</i>	Pointer to data structure. The data in this structure will be transmitted.

Returns

`IO_ErrorType`

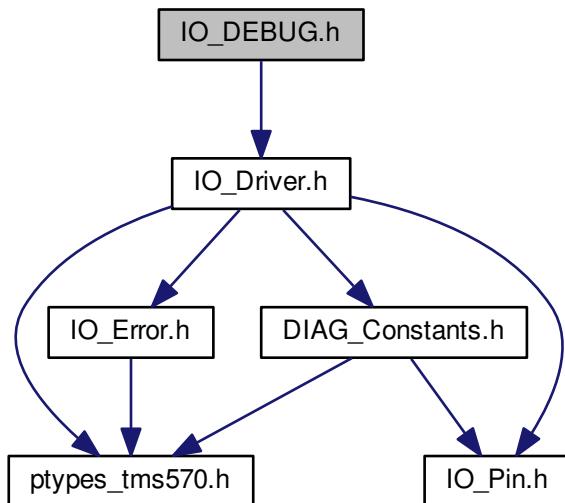
Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_BUSY</code>	message object busy - no data has been transmitted
<code>IO_E_CAN_WRONG_HANDLE</code>	invalid handle has been passed
<code>IO_E_NULL_POINTER</code>	null pointer has been passed
<code>IO_E_INVALID_PARAMETER</code>	an invalid message has been passed
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the given handle has not been configured
<code>IO_E_CAN_TIMEOUT</code>	the CAN node reported a timeout
<code>IO_E_INTERNAL_CSM</code>	an internal error occurred
<code>IO_E_UNKNOWN</code>	an unknown error occurred

7.6 IO_DEBUG.h File Reference

IO Driver functions for DEBUG utilities.

Include dependency graph for IO_DEBUG.h:



Macros

- #define `IO_DEBUG_OUTPUT_PIN_0` 0U

- #define IO_DEBUG_OUTPUT_PIN_1 1U
- #define IO_DEBUG_OUTPUT_PIN_2 2U
- #define IO_DEBUG_WD_NORMAL 0U
- #define IO_DEBUG_WD_STATE_ACTIVE 2U
- #define IO_DEBUG_WD_STATE_PREPARED 1U
- #define IO_DEBUG_WD_STATE_SAFE 3U
- #define IO_DEBUG_WD_STATE_UNKNOWN 4U

Functions

- **IO_ErrorType IO_DEBUG_GetTxStatus (void)**
Checks if the stdout transmission buffers are empty.
- **IO_ErrorType IO_DEBUG_GetWatchdogState (ubyte1 *const wd_state)**
Returns the current debug state of the watchdog module.
- **IO_ErrorType IO_DEBUG_SetOutputPin (ubyte1 pin, bool value)**
Sets a debug output pin (LED).
- **IO_ErrorType IO_DEBUG_StdioDelInit (void)**
Deinitializes the UART interface for standard IO usage.
- **IO_ErrorType IO_DEBUG_StdioInit (void)**
Initializes the UART interface for standard IO usage.

7.6.1 Detailed Description

IO Driver functions for DEBUG utilities.

Provides functions to set debug output pins (LEDs), to interface the UART in an Stdio like manner (e.g. printf) and to obtain the watchdog's state when running in debug mode.

The pins (LEDs) are only accessible via the JTAG adapter board and thus only with open devices and connected JTAG adapter board. These pins are only suitable for debugging purposes, like function runtime measurements, or debug status outputs.

DEBUG-API Usage:

- [Examples for DEBUG API functions](#)

Note

The standard output line buffered. This means that strings which do not end with a line break character (' \n ') and are less than 80 characters long will stay in the stdout buffer. This buffer can be flushed immediately with fflush(stdout) ;

7.6.2 DEBUG Code Examples

Examples for using the DEBUG API

7.6.2.1 Example for DEBUG pin setting

```
// set debug pin 0 (LED 0) to high
IO_DEBUG_SetOutputPin(IO_DEBUG_OUTPUT_PIN_0, //pin 0
                      TRUE); //set high
```

```
task()

// set debug pin 0 (LED 0) to low
IO_DEBUG_SetOutputPin(IO_DEBUG_OUTPUT_PIN_0, //pin 0
                      FALSE);                //set low
```

7.6.2.2 Example for Stdio usage

```
IO_ErrorType io_error;

// initialize UART for standard I/O interaction
IO_DEBUG_StdioInit();

// output a string using printf
printf("Hello World!\r\n");

printf("This is a line without a line break.");

// flush stdout after printing the last line without a line break
fflush(stdout);

// wait for the UART buffers to be completely emptied
do
{
    io_error = IO_DEBUG_GetTxStatus();
} while (io_error != IO_E_OK);

// deinitialize UART for standard I/O interaction
IO_DEBUG_StdioDeInit();
```

7.6.3 Macro Definition Documentation

7.6.3.1 #define IO_DEBUG_OUTPUT_PIN_0 0U

Debug Pin 0

Definition at line 94 of file IO_DEBUG.h.

7.6.3.2 #define IO_DEBUG_OUTPUT_PIN_1 1U

Debug Pin 1

Definition at line 95 of file IO_DEBUG.h.

7.6.3.3 #define IO_DEBUG_OUTPUT_PIN_2 2U

Debug Pin 2

Definition at line 96 of file IO_DEBUG.h.

7.6.3.4 #define IO_DEBUG_WD_NORMAL 0U

Watchdog mode is normal

Definition at line 98 of file IO_DEBUG.h.

7.6.3.5 #define IO_DEBUG_WD_STATE_ACTIVE 2U

Watchdog mode is debug with state "active"

Definition at line 100 of file IO_DEBUG.h.

7.6.3.6 #define IO_DEBUG_WD_STATE_PREPARED 1U

Watchdog mode is debug with state "prepared"

Definition at line 99 of file IO_DEBUG.h.

7.6.3.7 #define IO_DEBUG_WD_STATE_SAFE 3U

Watchdog mode is debug with state "safe"

Definition at line 101 of file IO_DEBUG.h.

7.6.3.8 #define IO_DEBUG_WD_STATE_UNKNOWN 4U

Watchdog mode is debug with an unknown state

Definition at line 102 of file IO_DEBUG.h.

7.6.4 Function Documentation

7.6.4.1 IO_ErrorType IO_DEBUG_GetTxStatus (void)

Checks if the stdio transmission buffers are empty.

With the help of this function it can be checked if all the parameters of a printf() have been transmitted. This doesn't include the line buffering of stdio – for details see the [note about stdio buffering](#).

Returns

IO_ErrorType

Return values

IO_E_OK	All the characters printed to stdio have been transmitted.
IO_E_BUSY	There is at least one character in the UART module which has not been printed out yet.
IO_E_CHANNEL_NOT_CONFIGURED	The UART interface has not been initialized for Stdio usage.

7.6.4.2 IO_ErrorType IO_DEBUG_GetWatchdogState (**ubyte1 *const wd_state**)

Returns the current debug state of the watchdog module.

Parameters

out	<i>wd_state</i>	Indicates the watchdog debug state <ul style="list-style-type: none"> • IO_DEBUG_WD_NORMAL: Watchdog module runs in normal mode • IO_DEBUG_WD_STATE_PREPARED: Watchdog module is in prepared debug state • IO_DEBUG_WD_STATE_ACTIVE: Watchdog module is in active debug state • IO_DEBUG_WD_STATE_SAFE: Watchdog module is in safe debug state • IO_DEBUG_WD_STATE_UNKNOWN: Watchdog module runs in debug mode but the state could not be obtained
-----	-----------------	---

Returns

[IO_ErrorType](#)

Return values

IO_E_OK	everything fine
IO_E_NULL_POINTER	a NULL pointer has been passed to the function
IO_E_WD_STATUS_INVALID	the watchdog's status information could not be obtained

7.6.4.3 IO_ErrorType IO_DEBUG_SetOutputPin (**ubyte1 pin, bool value**)

Sets a debug output pin (LED).

Parameters

<i>pin</i>	Debug output pin, one of: <ul style="list-style-type: none"> • IO_DEBUG_OUTPUT_PIN_0 • IO_DEBUG_OUTPUT_PIN_1 • IO_DEBUG_OUTPUT_PIN_2
<i>value</i>	Output value, one of: <ul style="list-style-type: none"> • TRUE: High level • FALSE: Low level

Returns

[IO_ErrorType](#)

Return values

<i>IO_E_OK</i>	everything fine
<i>IO_E_CH_CAPABILITY</i>	pin parameter is invalid
<i>IO_E_DRIVER_NOT_INITIALIZED</i>	the common driver init function has not been called before
<i>IO_E_FPGA_NOT_INITIALIZED</i>	the FPGA has not been initialized
<i>IO_E_UNKNOWN</i>	an unknown error occurred

7.6.4.4 IO_ErrorType IO_DEBUG_StdioDeInit (void)

Deinitializes the UART interface for standard IO usage.

Returns

IO_ErrorType

Return values

<i>IO_E_OK</i>	everything fine
<i>IO_E_CHANNEL_NOT_CONFIGURED</i>	channel has not been initialized
<i>IO_E_UNKNOWN</i>	an unknown error occurred

7.6.4.5 IO_ErrorType IO_DEBUG_StdioInit (void)

Initializes the UART interface for standard IO usage.

Used settings are: 115.200 baud, 8 databits, 1 stopbit, no parity

Returns

IO_ErrorType

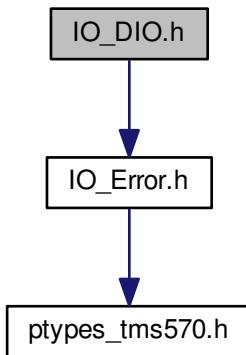
Return values

<i>IO_E_OK</i>	everything fine
<i>IO_E_DRIVER_NOT_INITIALIZED</i>	the common driver init function has not been called before
<i>IO_E_CHANNEL_BUSY</i>	channel has been initialized before
<i>IO_E_UNKNOWN</i>	an unknown error occurred

7.7 IO_DIO.h File Reference

IO Driver functions for Digital Input/Output.

Include dependency graph for IO_DIO.h:



Data Structures

- struct `io_dio_limits_`
Voltage limits for digital inputs.
- struct `io_do_safety_conf_`
Safety configuration for the digital outputs.

Typedefs

- typedef struct `io_dio_limits_` `IO_DIO_LIMITS`
Voltage limits for digital inputs.
- typedef struct `io_do_safety_conf_` `IO_DO_SAFETY_CONF`
Safety configuration for the digital outputs.

Functions

- `IO_ErrorType IO_DI_DelInit (ubyte1 di_channel)`
Deinitializes a digital input.
- `IO_ErrorType IO_DI_Get (ubyte1 di_channel, bool *const di_value)`
Gets the value of a digital input.
- `IO_ErrorType IO_DI_Init (ubyte1 di_channel, ubyte1 pupd, const IO_DIO_LIMITS *const limits)`
Setup a digital input.
- `IO_ErrorType IO_DO_DelInit (ubyte1 do_channel)`
Deinitializes a digital output.
- `IO_ErrorType IO_DO_GetCur (ubyte1 do_channel, ubyte2 *const current, bool *const fresh)`

Returns the measured current of a digital output.

- `IO_ErrorType IO_DO_GetVoltage (ubyte1 do_channel, ubyte2 *const voltage, bool *const fresh)`
Returns the measured voltage of a digital output.
- `IO_ErrorType IO_DO_Init (ubyte1 do_channel, bool diagnostic, const IO_DO_SAFETY_CONF *const safety_conf)`
Setup a digital output.
- `IO_ErrorType IO_DO_ResetProtection (ubyte1 do_channel, ubyte1 *const reset_cnt)`
Reset the output protection for a digital output.
- `IO_ErrorType IO_DO_Set (ubyte1 do_channel, bool do_value)`
Sets the value of a digital output.

Pull up / Pull down configuration for digital inputs

Configuration of the pull up or pull down resistors on the digital inputs. These defines can be used for the `pupd` parameter of the function `IO_DI_Init()`.

- `#define IO_DI_NO_PULL 0x00U`
- `#define IO_DI_PU_10K 0x01U`
- `#define IO_DI_PD_10K 0x02U`

7.7.1 Detailed Description

IO Driver functions for Digital Input/Output.

Contains all service functions for the digital in/outputs.

Note

The digital inputs reflect the current status of the input at the point in time where the function is called.

The digital outputs `IO_DO_00 .. IO_DO_15` are controlled over SPI shift registers, therefore the outputs will be periodically updated with a cycle of 1ms.

DIO-API Usage:

- [Examples for DIO API functions](#)

7.7.2 Digital output protection

Each digital output is individually protected against overcurrent. Whenever a fatal overcurrent situation is detected on any safe or unsafe digital output, the output is disabled by software. Note that disabled by software means that the output is set to low but the state of the safety switch (if configured) remains unchanged.

When entering the protection state, the digital output has to remain in this state for the given wait time. After this wait time the digital output can be reenabled via function `IO_DO_ResetProtection()`. Note that the number of reenabling operations for a single digital output is limited to 10. Refer to function `IO_DO_ResetProtection()` for more information on how to reenable a digital output.

Detailed information about fatal overcurrent situations, the reaction time and wait time are listed in the following table.

Digital output	Fatal overcurrent situation	Latest disabled within	Wait time
IO_DO_00 .. IO_DO_15, IO_DO_52 .. IO_DO_59	3.5A if board temperature > 85°C and persistent for > 100ms	12ms	1s
	4.7A if board temperature ≤ 85°C and persistent for > 100ms		
	5 samples above 7A		
	the switch has already switched off by itself		
	3.1A if board temperature > 85°C and persistent for > 100ms		
	4.1A if board temperature ≤ 85°C and persistent for > 100ms		
	1 sample above 5.5A		
IO_DO_16 .. IO_DO_51		6ms	10s

7.7.3 DIO Code Examples

Examples for using the DIO API

7.7.3.1 DIO initialization example

```
IO_DI_Init(IO_DI_00,
           IO_DI_NO_PULL,
           NULL);           // digital input
                  // fixed pull resistor
                  // no limit configuration (only for analog channels)

IO_DO_Init(IO_DO_18,
           FALSE,
           NULL);          // digital output
                  // diagnostic disabled
                  // no safety configuration
```

7.7.3.2 DIO task function example

```
bool di_val_0;

IO_DI_Get(IO_DI_00,           // read value of digital input
          &di_val_0);

IO_DO_Set(IO_DO_18,          // set digital output value
          TRUE);
```

7.7.4 Macro Definition Documentation

7.7.4.1 #define IO_DI_NO_PULL 0x00U

fixed pull resistor

Definition at line 109 of file IO_DIO.h.

7.7.4.2 #define IO_DI_PD_10K 0x02U

10 kOhm pull down

Definition at line 111 of file IO_DIO.h.

7.7.4.3 #define IO_DI_PU_10K 0x01U

10 kOhm pull up

Definition at line 110 of file IO_DIO.h.

7.7.5 Typedef Documentation

7.7.5.1 typedef struct io_dio_limits_ IO_DIO_LIMITS

Voltage limits for digital inputs.

Contains the thresholds for valid low- and high-levels for digital inputs.

The range for the low-level is defined by the voltages low_thresh1 and low_thresh2, where low_thresh1 is the lower limit for a low-level and low_thresh2 the upper limit.

The range for the high-level is defined by the voltages high_thresh1 and high_thresh2, where high_thresh1 is the lower limit for a high-level and high_thresh2 the upper limit.

The value of low_thresh1 must always be smaller than low_thresh2 and high_thresh1 must always be smaller than high_thresh2.

The value of low_thresh2 must always be smaller than high_thresh1.

Examples:

```
1 // voltage limits
2 IO_DIO_LIMITS limits1 = { 0, 2000, 3000, 5000 };
```

In the above example limits1 defines the range 0-2000mV as valid low-level and 3000-5000mV as valid high-level.

Note

If no limits will be specified by the application, the following default limits will be applied: { 0, 2500, 2500, 32000 }

7.7.5.2 typedef struct io_do_safety_conf_ IO_DO_SAFETY_CONF

Safety configuration for the digital outputs.

Stores all relevant safety configuration parameters for the digital outputs. The internal checker modules verify that these outputs still work correctly.

Attention

In order to make diagnostics on channels `IO_DO_00 .. IO_DO_15` (against open load and short to VBAT) possible, a delay time of at least 20 ms is needed between transitions of the output state. If this timing is not fulfilled, a diagnostic error may get raised.

7.7.6 Function Documentation

7.7.6.1 IO_ErrorType IO_DI_DelInit (ubyte1 *di_channel*)

Deinitializes a digital input.

Parameters

<i>di_channel</i>	Digital input: <ul style="list-style-type: none"> • <code>IO_DI_00 .. IO_DI_35</code> • <code>IO_DI_36 .. IO_DI_47</code> • <code>IO_DI_48 .. IO_DI_55</code> • <code>IO_DI_56 .. IO_DI_63</code> • <code>IO_DI_64 .. IO_DI_71</code> • <code>IO_DI_72 .. IO_DI_79</code> • <code>IO_DI_80 .. IO_DI_87</code> • <code>IO_DI_88 .. IO_DI_95</code>
-------------------	--

Returns

`IO_ErrorType`:

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the given channel is not configured
<code>IO_E_CH_CAPABILITY</code>	the given channel is not a digital input channel
<code>IO_E_INVALID_PARAMETER</code>	internally an invalid parameter has been passed
<code>IO_E_UNKNOWN</code>	an unknown error occurred

7.7.6.2 IO_ErrorType IO_DI_Get (ubyte1 *di_channel*, bool *const *di_value*)

Gets the value of a digital input.

Parameters

<i>di_channel</i>	Digital input: <ul style="list-style-type: none"> • <code>IO_DI_00 .. IO_DI_35</code> • <code>IO_DI_36 .. IO_DI_47</code> • <code>IO_DI_48 .. IO_DI_55</code> • <code>IO_DI_56 .. IO_DI_63</code> • <code>IO_DI_64 .. IO_DI_71</code> • <code>IO_DI_72 .. IO_DI_79</code> • <code>IO_DI_80 .. IO_DI_87</code> • <code>IO_DI_88 .. IO_DI_95</code>
out	<i>di_value</i> Input value: <ul style="list-style-type: none"> • <code>TRUE</code>: High level • <code>FALSE</code>: Low level

Returns

`IO_ErrorType`:

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_SHORT_BAT</code>	the measured analog value is above the valid high band
<code>IO_E_SHORT_GND</code>	the measured analog value is below the valid low band
<code>IO_E_INVALID_VOLTAGE</code>	the measured analog value is in between the valid bands for high and low
<code>IO_E_INVALID_CHANNEL_ID</code>	the channel id does not exist
<code>IO_E_NULL_POINTER</code>	NULL pointer has been passed
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the given channel is not configured
<code>IO_E_CH_CAPABILITY</code>	the given channel is not a digital input channel
<code>IO_E_STARTUP</code>	the output is in the startup phase
<code>IO_E_INTERNAL_CSM</code>	an internal error occurred
<code>IO_E_REFERENCE</code>	the internal reference voltage is out of range
<code>IO_E_UNKNOWN</code>	an unknown error occurred

Remarks

This function returns the value of the digital input pins at the time when the function is called for:

- `IO_DI_00 .. IO_DI_35`
- `IO_DI_36 .. IO_DI_47`

This function returns the value of the digital input pins at the time when the last AD value was sampled for.

- `IO_DI_48 .. IO_DI_55`
- `IO_DI_56 .. IO_DI_63`
- `IO_DI_64 .. IO_DI_71`
- `IO_DI_72 .. IO_DI_79`
- `IO_DI_80 .. IO_DI_87`
- `IO_DI_88 .. IO_DI_95`

The error codes `IO_E_SHORT_BAT`, `IO_E_SHORT_GND` and `IO_E_INVALID_VOLTAGE` are only returned for `IO_DI_56 .. IO_DI_63`, `IO_DI_64 .. IO_DI_71`, `IO_DI_72 .. IO_DI_79`, `IO_DI_80 .. IO_DI_87` and `IO_DI_88 .. IO_DI_95` and only if corresponding limits are defined.

7.7.6.3 `IO_ErrorType IO_DI_Init (ubyte1 di_channel, ubyte1 pupd, const IO_DIO_LIMITS *const limits)`

Setup a digital input.

Parameters

<i>di_channel</i>	Digital input: <ul style="list-style-type: none"> • <code>IO_DI_00 .. IO_DI_35</code> • <code>IO_DI_36 .. IO_DI_47</code> • <code>IO_DI_48 .. IO_DI_55</code> • <code>IO_DI_56 .. IO_DI_63</code> • <code>IO_DI_64 .. IO_DI_71</code> • <code>IO_DI_72 .. IO_DI_79</code> • <code>IO_DI_80 .. IO_DI_87</code> • <code>IO_DI_88 .. IO_DI_95</code>
<i>pupd</i>	Pull up/down configuration: <ul style="list-style-type: none"> • <code>IO_DI_NO_PULL</code>: fixed pull resistor • <code>IO_DI_PU_10K</code>: Pull up 10 kOhm • <code>IO_DI_PD_10K</code>: Pull down 10 kOhm
<i>in</i>	<i>limits</i> Voltage limits for low/high-levels. If <code>NULL</code> , default limits will be used. See <code>IO_DIO_LIMITS</code> for details.

Returns

`IO_ErrorType`:

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist or is not available on the used ECU variant
<code>IO_E_CH_CAPABILITY</code>	the given channel is not a digital input channel or the used ECU variant does not support this function
<code>IO_E_CHANNEL_BUSY</code>	the digital output channel is currently used by another function
<code>IO_E_INVALID_PARAMETER</code>	parameter is out of range
<code>IO_E_INVALID_LIMITS</code>	the given voltage limits are not valid
<code>IO_E_FPGA_NOT_INITIALIZED</code>	the FPGA has not been initialized
<code>IO_E_DRIVER_NOT_INITIALIZED</code>	the common driver init function has not been called before
<code>IO_E_UNKNOWN</code>	an unknown error occurred

Remarks

- The supported features depend on the selected channel:
 - `IO_DI_00 .. IO_DI_35`:
 - * `pupd` and `limits` ignored
 - `IO_DI_36 .. IO_DI_47`:
 - * `pupd`: `IO_DI_PU_10K` or `IO_DI_PD_10K`
 - `IO_DI_48 .. IO_DI_55`:
 - * `pupd` and `limits` ignored
 - `IO_DI_56 .. IO_DI_63`:
 - * `limits`: Voltage limits for low/high-levels
 - `IO_DI_64 .. IO_DI_71`:
 - * `limits`: Voltage limits for low/high-levels

- `IO_DI_72 .. IO_DI_79`:
 - * `pupd`: `IO_DI_PU_10K` or `IO_DI_PD_10K`
 - * `limits`: Voltage limits for low/high-levels
- `IO_DI_80 .. IO_DI_87`:
 - * `limits`: Voltage limits for low/high-levels
- `IO_DI_88 .. IO_DI_95`:
 - * `pupd`: `IO_DI_PU_10K` or `IO_DI_PD_10K`
 - * `limits`: Voltage limits for low/high-levels
- The inputs `IO_DI_80 .. IO_DI_87` have a fixed pull up. Depending on the parameter `limits` switches to ground or BAT can be read.
- The inputs `IO_DI_00 .. IO_DI_35`, `IO_DI_36 .. IO_DI_47` and `IO_DI_48 .. IO_DI_55` have a fixed switching threshold of 2.5V.

Attention

The inputs `IO_DI_00 .. IO_DI_35` and `IO_DI_48 .. IO_DI_55` have a fixed pull up and are only suitable for switches to ground.

The inputs `IO_DI_56 .. IO_DI_63` and `IO_DI_64 .. IO_DI_71` are only suitable for switches to BAT.

7.7.6.4 IO_ErrorType IO_DO_DelInit (`ubyte1 do_channel`)

Deinitializes a digital output.

Parameters

<code>do_channel</code>	Digital output: • <code>IO_DO_00 .. IO_DO_07</code> • <code>IO_DO_08 .. IO_DO_15</code> • <code>IO_DO_16 .. IO_DO_51</code> • <code>IO_DO_52 .. IO_DO_59</code>
-------------------------	---

Returns

`IO_ErrorType`:

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the given channel is not configured
<code>IO_E_CH_CAPABILITY</code>	the given channel is not a digital output channel
<code>IO_E_INTERNAL_CSM</code>	an internal error occurred
<code>IO_E_UNKNOWN</code>	an unknown error occurred

7.7.6.5 IO_ErrorType IO_DO_GetCur (**ubyte1 do_channel, ubyte2 *const current, bool *const fresh**)

Returns the measured current of a digital output.

Parameters

	<i>do_channel</i>	Digital output: <ul style="list-style-type: none"> • IO_DO_00 .. IO_DO_07 • IO_DO_08 .. IO_DO_15 • IO_DO_16 .. IO_DO_51 • IO_DO_52 .. IO_DO_59
out	<i>current</i>	Measured current in mA Range: 0..7500 (0A .. 7.500A)
out	<i>fresh</i>	Indicates if new values are available since the last call. <ul style="list-style-type: none"> • TRUE: Value in "current" is valid • FALSE: No new value available.

Returns

[IO_ErrorType](#):

Return values

IO_E_OK	everything fine
IO_E_INVALID_CHANNEL_ID	the channel id does not exist
IO_E_CHANNEL_NOT_CONFIGURED	the given channel is not configured
IO_E_NULL_POINTER	a NULL pointer has been passed to the function
IO_E_CH_CAPABILITY	the given channel is not a digital output channel
IO_E_CM_CALIBRATION	the zero current calibration failed
IO_E_REFERENCE	the internal reference voltage is out of range
IO_E_UNKNOWN	an unknown error occurred

Note

The error code [IO_E_CM_CALIBRATION](#) is only returned for [IO_DO_16 .. IO_DO_51](#)

Remarks

If there is no new current value available (for example the function [IO_DO_GetCur\(\)](#) gets called more frequently than the AD sampling) the flag *fresh* will be set to [FALSE](#).

7.7.6.6 IO_ErrorType IO_DO_GetVoltage (**ubyte1 do_channel, ubyte2 *const voltage, bool *const fresh**)

Returns the measured voltage of a digital output.

Parameters

	<i>do_channel</i>	Digital output: <ul style="list-style-type: none"> • <code>IO_DO_00 .. IO_DO_07</code> • <code>IO_DO_52 .. IO_DO_59</code>
out	<i>voltage</i>	Measured voltage in mV. Range: 0..32000 (0V..32.000V)
out	<i>fresh</i>	Indicates if new values are available since the last call. <ul style="list-style-type: none"> • <code>TRUE</code>: Value in "voltage" is valid • <code>FALSE</code>: No new value available.

Returns

`IO_ErrorType`:

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	the channel id does not exist
<code>IO_E_NULL_POINTER</code>	a NULL pointer has been passed to the function
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the given channel is not configured
<code>IO_E_CH_CAPABILITY</code>	the given channel is not a digital output channel
<code>IO_E_REFERENCE</code>	the internal reference voltage is out of range

Remarks

If there is no new voltage value available (for example the function `IO_DO_GetVoltage()` gets called more frequently than the AD sampling) the flag `fresh` will be set to `FALSE`.

7.7.6.7 `IO_ErrorType IO_DO_Init (ubyte1 do_channel, bool diagnostic, const IO_DO_SAFETY_CONF *const safety_conf)`

Setup a digital output.

Parameters

	<i>do_channel</i>	Digital output: <ul style="list-style-type: none"> • <code>IO_DO_00 .. IO_DO_07</code> • <code>IO_DO_08 .. IO_DO_15</code> • <code>IO_DO_16 .. IO_DO_51</code> • <code>IO_DO_52 .. IO_DO_59</code>
	<i>diagnostic</i>	Output configuration: <ul style="list-style-type: none"> • <code>TRUE</code>: diagnostic pull-up enabled • <code>FALSE</code>: diagnostic pull-up disabled
in	<i>safety_conf</i>	Relevant safety configurations for the checker modules. The following DO channels can be configured safety relevant: <ul style="list-style-type: none"> • <code>IO_DO_00 .. IO_DO_07</code>

Returns

`IO_ErrorType`:

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist or is not available on the used ECU variant
<code>IO_E_CH_CAPABILITY</code>	the given channel is not a digital output channel or the used ECU variant does not support this function
<code>IO_E_CHANNEL_BUSY</code>	the digital output channel is currently used by another function
<code>IO_E_CM_CALIBRATION</code>	the zero current calibration failed
<code>IO_E_FPGA_NOT_INITIALIZED</code>	the FPGA has not been initialized
<code>IO_E_DRIVER_NOT_INITIALIZED</code>	the common driver init function has not been called before
<code>IO_E_INVALID_SAFETY_CONFIG</code>	an invalid safety configuration has been passed
<code>IO_E_DRV_SAFETY_CONF_NOT_CONFIG</code>	global safety configuration is missing
<code>IO_E_DRV_SAFETY_CYCLE_RUN_NING</code>	the init function was called after the task begin function
<code>IO_E_INTERNAL_CSM</code>	an internal error occurred
<code>IO_E_UNKNOWN</code>	an unknown error occurred

Note

The error code `IO_E_CM_CALIBRATION` is only returned for `IO_DO_16 .. IO_DO_51`

Remarks

- The digital output channels `IO_DO_00 .. IO_DO_15` are controlled over SPI shift registers. Therefore the outputs will be periodically updated with a cycle of 1ms.
- The digital output channels `IO_DO_16 .. IO_DO_51` are an alternative function to `IO_PWM_00 .. IO_PWM_35`.
- The digital output channels `IO_DO_52 .. IO_DO_59` are an alternative function to `IO_PVG_00 .. IO_PVG_07`.
- The parameter `diagnostic` is only applied to the channels `IO_DO_00 .. IO_DO_07` and `IO_DO_52 .. IO_DO_59`. If `diagnostic` is `TRUE`, the output can detect open load and short circuit. If it is `FALSE`, the output can not detect open load or short circuit. Select `FALSE` for loads with low current consumption like LEDs. With `diagnostic == FALSE` the pull up will be switched off.
- If `safety_conf != NULL`, a low side and high side channel have to be connected together. The internal checker modules check the given channels against the parameter in `safety_conf`. For more detail about each parameter look on the definition of `IO_DO_SAFETY_CONF`
- If `safety_conf != NULL`, the parameter `diagnostic` is forced to `TRUE` to allow diagnostics

7.7.6.8 IO_ErrorType IO_DO_ResetProtection (`ubyte1 do_channel, ubyte1 *const reset_cnt`)

Reset the output protection for a digital output.

Parameters

	<i>do_channel</i>	Digital output: <ul style="list-style-type: none"> • <code>IO_DO_00 .. IO_DO_07</code> • <code>IO_DO_08 .. IO_DO_15</code> • <code>IO_DO_16 .. IO_DO_51</code> • <code>IO_DO_52 .. IO_DO_59</code>
out	<i>reset_cnt</i>	Protection reset counter. Indicates how often the application already reset the protection.

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist
<code>IO_E_FET_PROT_NOT_ACTIVE</code>	no output FET protection is active
<code>IO_E_FET_PROT_PERMANENT</code>	the output FET is protected permanently because it was already reset more than 10 times
<code>IO_E_FET_PROT_WAIT</code>	the output FET protection can not be reset, as the wait time of 1s is not already passed
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the given channel is not configured
<code>IO_E_CH_CAPABILITY</code>	the given channel is not a digital output channel
<code>IO_E_UNKNOWN</code>	an unknown error occurred

Attention

The protection can be reset 10 times, afterwards the output will remain permanently protected

Note

- After entering the output protection, a certain time has to pass before the output protection can be reset:
 - 1s for `IO_DO_00 .. IO_DO_07`
 - 1s for `IO_DO_08 .. IO_DO_15`
 - 1s for `IO_DO_52 .. IO_DO_59`
 - 10s for `IO_DO_16 .. IO_DO_51`
- This function will not set the output back again to high. After calling this function the output has to be set to the intended level with `IO_DO_Set()`

Remarks

If the parameter `reset_cnt` is `NULL`, the parameter is ignored. The parameter `reset_cnt` returns the number of resets already performed.

7.7.6.9 IO_ErrorType IO_DO_Set (*ubyte1 do_channel, bool do_value*)

Sets the value of a digital output.

Parameters

<i>do_channel</i>	Digital output: <ul style="list-style-type: none"> • <code>IO_DO_00 .. IO_DO_07</code> • <code>IO_DO_08 .. IO_DO_15</code> • <code>IO_DO_16 .. IO_DO_51</code> • <code>IO_DO_52 .. IO_DO_59</code>
<i>do_value</i>	Output value: <ul style="list-style-type: none"> • <code>TRUE</code>: High level • <code>FALSE</code>: Low level

Returns

`IO_ErrorType`:

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	the channel id does not exist
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the given channel is not configured
<code>IO_E_CH_CAPABILITY</code>	the given channel is not a digital output channel
<code>IO_E_STARTUP</code>	the output is in the startup phase
<code>IO_E_NO_DIAG</code>	no diagnostic feedback available
<code>IO_E_OPEN_LOAD</code>	open load has been detected
<code>IO_E_SHORT_GND</code>	short circuit has been detected
<code>IO_E_SHORT_BAT</code>	short to battery voltage has been detected
<code>IO_E_OPEN_LOAD_OR_SHORT_BAT</code>	open load or short to battery voltage has been detected
<code>IO_E_FET_PROT_ACTIVE</code>	the output FET protection is active and has set the output to low. It can be reset with <code>IO_DO_ResetProtection()</code> after the wait time is passed
<code>IO_E_FET_PROT_RESETABLE</code>	the output FET protection is ready to be reset with <code>IO_DO_ResetProtection()</code>
<code>IO_E_FET_PROT_PERMANENT</code>	the output FET is protected permanently because it was already reset more than 10 times. The output will remain low
<code>IO_E_SAFETY_SWITCH_DISABLED</code>	The safety switch of the corresponding output is disabled. The output is currently forced to low.
<code>IO_E_INTERNAL_CSM</code>	an internal error occurred
<code>IO_E_SAFE_STATE</code>	the digital output channel is in a safe state
<code>IO_E_UNKNOWN</code>	an unknown error occurred

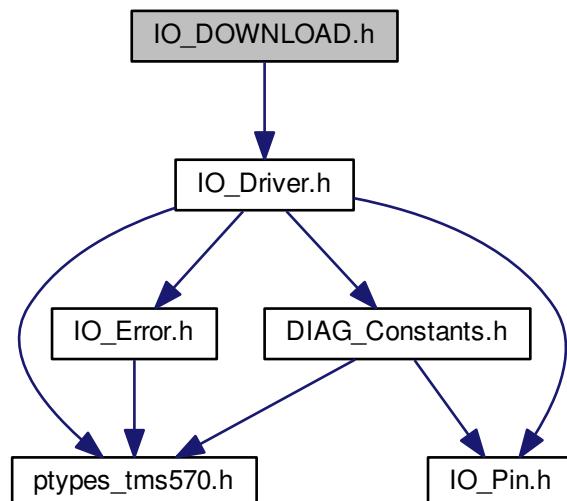
Remarks

- The digital output channels `IO_DO_00 .. IO_DO_15` are controlled over SPI shift registers. Therefore the outputs will be periodically updated with a cycle of 1ms.
- The digital output channels `IO_DO_16 .. IO_DO_51` are an alternative function to `IO_PWM_00 .. IO_PWM_35`.
- The digital output channels `IO_DO_52 .. IO_DO_59` are an alternative function to `IO_PWD_00 .. IO_PWD_11`.

7.8 IO_DOWNLOAD.h File Reference

IO Driver functions for handling Ethernet download requests.

Include dependency graph for `IO_DOWNLOAD.h`:



Functions

- `IO_ErrorType IO_DOWNLOAD_CheckRequest (void)`
Checks, if an Ethernet download request is currently pending.
- `IO_ErrorType IO_DOWNLOAD_Init (void)`
Initialization of the Ethernet download communication driver.
- `IO_ErrorType IO_DOWNLOAD_Launch (void)`
Restarts the ECU in Ethernet download mode.

7.8.1 Detailed Description

IO Driver functions for handling Ethernet download requests.

Because an Ethernet link establishment can take up to 3 seconds, any Ethernet download request from the TTC-Downloader cannot be checked during power up (as it is the case for CAN). To make it possible anyway, the DOWNLOAD module can be used for. It's being initialized with `IO_DOWNLOAD_Init()`.

The DOWLNLOAD module automatically checks for TTC-Downloader requests on the Ethernet interface. This request consists of a handshake between the TTC-Downloader and the ECU. To check, if any request is pending, `IO_DOWNLOAD_CheckRequest()` can be used.

After a detected request, the application has a time of 3 seconds to launch the download mode. Within this time, the application needs to shut down and all non-volatile memories need to be stored. With a call to `IO_DOWNLOAD_Launch()`, the ECU restarts in Ethernet download mode. If the download mode is not launched during this time, the request becomes invalid and a new request is necessary.

The configuration for setting up the download capability is taken from the APDB. There, the fields

- TargetIPAddress
- SubnetMask
- DLMulticastIPAddress

are used. If an enforcement to default settings has been detected during startup, the bootloader's default settings will be applied.

DOWNLOAD-API Usage:

- [Examples for DOWNLOAD API functions](#)

7.8.2 DOWNLOAD Code Examples

Examples for using the DOWNLOAD API

```
// initialize Ethernet interface for handling download requests
IO_DOWNLOAD_Init();

// application cycle
while (1)
{
    ...

    // check, if a download request is pending
    if (IO_DOWNLOAD_CheckRequest() == IO_E_OK)
    {
        // shut down application
        ...

        // save memories
        ...

        // launch ECU in download mode
        IO_DOWNLOAD_Launch();
    }
}
```

{ ... }

7.8.3 Function Documentation

7.8.3.1 IO_ErrorType IO_DOWNLOAD_CheckRequest (void)

Checks, if an Ethernet download request is currently pending.

The function returns the information, if a download request is currently pending with respect of the timeout limits. Once the timing exceeds the limits, the function will return [IO_E_DOWNLOAD_NO_REQ](#).

Returns

[IO_ErrorType](#)

Return values

IO_E_OK	a download request is pending
IO_E_CHANNEL_NOT_CONFIGURED	the driver hasn't been initialized before
IO_E_DOWNLOAD_NO_REQ	no download request is pending

7.8.3.2 IO_ErrorType IO_DOWNLOAD_Init (void)

Initialization of the Ethernet download communication driver.

The function

- Initializes the Ethernet communications interface
- Sets up all internal modules for automatic download request handling

Returns

[IO_ErrorType](#)

Return values

IO_E_OK	everything fine
IO_E_INVALID_CHANNEL_ID	the used ECU variant does not support this function
IO_E_DRIVER_NOT_INITIALIZED	the common driver init function has not been called before
IO_E_CHANNEL_BUSY	channel has been initialized before

7.8.3.3 IO_ErrorType IO_DOWNLOAD_Launch (void)

Restarts the ECU in Ethernet download mode.

As a precondition, a download request must be pending. The function restarts the ECU. On success, the function will NOT return. Instead, the ECU enters the bootloader and activates Ethernet download mode there.

Returns

`IO_ErrorType`

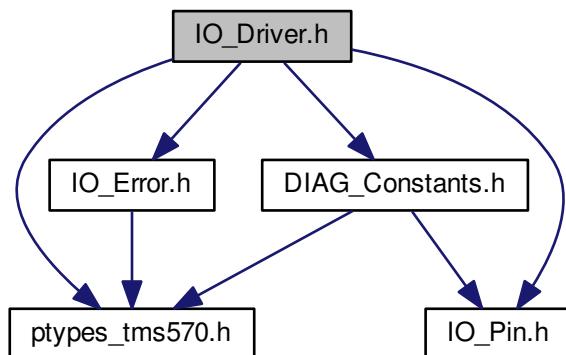
Return values

<code>IO_E_OK</code>	everything fine (theoretically, see above)
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the driver hasn't been initialized before
<code>IO_E_DOWNLOAD_NO_REQ</code>	no download request is pending

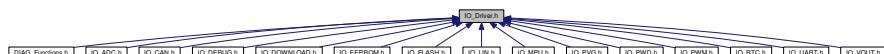
7.9 IO_Driver.h File Reference

High level interface to IO Driver.

Include dependency graph for IO_Driver.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct `io_driver_safety_conf_`
Driver Safety Configuration.

Typedefs

- `typedef void(* IO_DRIVER_FPU_HANDLER) (bool division_by_zero, bool input_denormal, bool invalid_operation, bool overflow, bool underflow)`
Function pointer for FPU exception handler.
- `typedef struct io_driver_safety_conf_ IO_DRIVER_SAFETY_CONF`
Driver Safety Configuration.

Functions

- `IO_ErrorType IO_Driver_GetMacAddress (ubyte1 *const macaddress)`
Returns the ECU's MAC address.
- `IO_ErrorType IO_Driver_GetProdCode (ubyte1 *const prodcode)`
Returns the ECU's production code.
- `IO_ErrorType IO_Driver_GetSerialNumber (ubyte1 *const serialnumber)`
Returns the ECU's serial number.
- `IO_ErrorType IO_Driver_GetVersionOfBootloader (ubyte1 *const major, ubyte1 *const minor)`
Returns the version number of the Bootloader.
- `IO_ErrorType IO_Driver_GetVersionOfDriver (ubyte1 *const major, ubyte1 *const minor, ubyte2 *const patchlevel)`
Returns the version number of the IO driver.
- `IO_ErrorType IO_Driver_GetVersionOfFPGA (ubyte2 *const rev0, ubyte2 *const rev1, ubyte2 *const rev2, ubyte1 *const device, ubyte1 *const release, ubyte1 *const patchlevel)`
Returns the version number of the FPGA IP.
- `IO_ErrorType IO_Driver_Init (const IO_DRIVER_SAFETY_CONF *const safety_conf)`
Global initialization of IO driver.
- `IO_ErrorType IO_Driver_Reset (void)`
Performs a software reset of the device. The intended use of this function is to enter the bootloader from an application.
- `IO_ErrorType IO_Driver_SetFPUHandler (IO_DRIVER_FPU_HANDLER fpu_handler)`
Registers an application callback for FPU exceptions.
- `IO_ErrorType IO_Driver_SetIntegerDivisionByZeroException (bool enable)`
Enables/disables integer division by zero exceptions.
- `IO_ErrorType IO_Driver_TaskBegin (void)`
Task function for IO Driver. This function shall be called at the beginning of the task.
- `IO_ErrorType IO_Driver_TaskEnd (void)`
Task function for IO Driver. This function shall be called at the end of the task.

Global safety configuration watchdog window size definitions

These definitions have to be used when configuring the window size inside the global safety configuration.

The window margin size is a percentage of the maximum trigger time, not of the command period. For example if the command period is 10 ms and the window size is 50 percent, the maximum distance between two watchdog triggers will be 13.33 ms, and the watchdog can be triggered in the last 50% of this window, meaning from 6.67 ms to 13.33 ms (a window margin of $\pm 33\%$ of the

command period). This way a correct triggering would happen at the middle of the watchdog trigger window.

- #define SAFETY_CONF_WINDOW_SIZE_100_PERCENT (0U)
- #define SAFETY_CONF_WINDOW_SIZE_50_PERCENT (1U)
- #define SAFETY_CONF_WINDOW_SIZE_25_PERCENT (2U)
- #define SAFETY_CONF_WINDOW_SIZE_12_5_PERCENT (3U)
- #define SAFETY_CONF_WINDOW_SIZE_6_25_PERCENT (4U)
- #define SAFETY_CONF_WINDOW_SIZE_3_125_PERCENT (5U)

Global safety configuration reset behavior definitions

This definitions have to be used when configuring the reset behavior inside the global safety configuration.

- #define SAFETY_CONF_RESETS_DISABLED (0U)
- #define SAFETY_CONF_RESETS_1 (1U)
- #define SAFETY_CONF_RESETS_2 (2U)
- #define SAFETY_CONF_RESETS_3 (3U)
- #define SAFETY_CONF_RESETS_4 (4U)
- #define SAFETY_CONF_RESETS_5 (5U)
- #define SAFETY_CONF_RESETS_6 (6U)
- #define SAFETY_CONF_RESETS_7 (7U)
- #define SAFETY_CONF_RESETS_8 (8U)
- #define SAFETY_CONF_RESETS_9 (9U)

Number of bytes of the ECU's serial number

Defines the number of bytes of the ECU's serial number

- #define IO_DRIVER_ECU_SERIAL_LENGTH (14U)

Number of bytes of the ECU's MAC address

Defines the number of bytes of the ECU MAC address

- #define IO_DRIVER_ECU_MAC_ADD_LENGTH (12U)

Number of bytes of the ECU's production code

Defines the number of bytes of the ECU's production code

- #define IO_DRIVER_ECU_PROD_CODE_LENGTH (30U)

7.9.1 Detailed Description

High level interface to IO Driver.

The IO Driver high level interface provides a general initialization function, a version API and general task functions which shall wrap the whole user application.

7.9.2 Basic structure of an application

The IO Driver API provides two different types of functions:

- Initialization functions: These functions are designed to be called once at the beginning of an application.
- Task functions: These functions are designed to be called periodically at runtime.

The function [IO_Driver_Init\(\)](#) needs to be the first function which is called during the initialization.

All task functions need to be enclosed by the functions [IO_Driver_TaskBegin\(\)](#) and [IO_Driver_TaskEnd\(\)](#)

7.9.3 Limitations during startup

During the first cycles some measurement values (eg. ADC, PWD, etc.) are invalid, which is indicated by the corresponding fresh flag when reading the value. After some cycles the actual measurement values are available.

Further limitations for safety controllers: Various tests are performed at startup. For this reason the measurement values are not available during this process. The application software should wait until the CPU has entered the Main state before evaluating the input variables. For details on CPU states and how to retrieve the current state, refer to [Functions for ECU diagnostics](#)

7.9.4 IO Driver Code Examples

Examples for using the IO Driver API

7.9.4.1 Example of an application

```
static ubyte4 timestamp = 0;

void task (void)
{
    IO_Driver_TaskBegin();

    // User Application
    // and calls to driver task functions.

    IO_Driver_TaskEnd();

    while (IO_RTC_GetTimeUS(timestamp) < 5000);      // wait until 5ms have passed
    timestamp += 5000;                                // increase time stamp by cycle time

    // Note: If one cycle takes longer than the configured cycle time, for the next
    // cycle less time is available. This method helps to prevent a phase shift between
    // application runtime and hardware runtime
}

void main (void)
{
    ubyte1 ecu_serial[IO_DRIVER_ECU_SERIAL_LENGTH];
```

```

//-----//
// start of driver initialization //
//-----//

// IO_Driver_Init() is the first function:
IO_Driver_Init(NULL); // no safety critical application

//-----//
// end of driver initialization //
//-----//

// Read ECU serial number
IO_Driver_GetSerialNumber(ecu_serial);

// Get timestamp
IO_RTC_StartTime(&timestamp);

//-----//
// from now on only the task function is called //
//-----//
while (1)
{
    task();
}

```

The task function is called every 5000us = 5ms. Please refer to the [Real Time Clock](#) documentation for details on how to use the RTC functions.

7.9.4.2 Example implementation for application Safety-Callback

See [DIAG_ERROR_CB](#) for details on the application safety callback function.

```

static ubyte2 APPL_SafetyCb(ubyte1 diag_state,
                            ubyte1 watchdog_state,
                            DIAG_ERRORCODE * const error)
{
    ubyte2 action;

    // Error codes can for example be saved to error memory of application software
    // (APPL_WriteErrMem is a hypothetical function of the application software).
    APPL_WriteErrMem( diag_state, watchdog_state, error );

    // Just a simple example which does
    // not take the type of error into account
    switch (error->device_num)
    {
        case IO_ADC_00:
        case IO_ADC_01:
            // ...
            // Do something sophisticated
            // ...
            action = DIAG_ERR_NOACTION;
            break;
        case IO_PWM_00:
        case IO_PWM_01:
        case IO_PWM_02:
        case IO_PWM_03:

```

```

// ...
// Do something sophisticated
// ...
action = DIAG_ERR_DISABLE_SSW0;
break;
case IO_PWM_04:
case IO_PWM_05:
case IO_PWM_06:
case IO_PWM_07:
// ...
// Do something sophisticated
// ...
action = (DIAG_ERR_DISABLE_SSW0 |
DIAG_ERR_DISABLE_SSW1);
break;
case IO_PWM_08:
case IO_PWM_09:
case IO_PWM_10:
case IO_PWM_11:
// ...
// Do something sophisticated
// ...
action = (DIAG_ERR_DISABLE_SSW2 |
DIAG_ERR_DISABLE_HS05 | DIAG_ERR_DISABLE_HS06);
break;
case IO_PWM_12:
case IO_PWM_13:
case IO_PWM_14:
case IO_PWM_15:
// ...
// Do something sophisticated
// ...
action = DIAG_ERR_DISABLE_SSW2;
break;
default:
action = DIAG_ERR_SAFESTATE;
break;
}
return action;
}

```

7.9.4.3 Example implementation for application Notification-Callback

See [DIAG_NOTIFY_CB](#) for details on the application notification callback function.

```

static void APPL_NotifyCb(ubyte1 diag_state,
                          ubyte1 watchdog_state,
                          DIAG_ERRORCODE * const error)
{
    IO_CAN_DATA_FRAME can_frame = { 0 };

    // Error codes can for example be send over CAN
    // A CAN channel needs to be initialized before with \c #IO_CAN_Init()
    // and a mailbox for transmission needs to be set up

    // assemble frame
    can_frame.id = 0x22;
    can_frame.id_format = IO_CAN_STD_FRAME;
    can_frame.length = 8;

```

```

can_frame.data[0] = diag_state;
can_frame.data[1] = watchdog_state;
can_frame.data[2] = error->device_num;
can_frame.data[3] = error->error_code;
can_frame.data[4] = (ubyte1)(error->faulty_value >> 24);
can_frame.data[5] = (ubyte1)(error->faulty_value >> 16);
can_frame.data[6] = (ubyte1)(error->faulty_value >> 8);
can_frame.data[7] = (ubyte1)error->faulty_value;

// write CAN message
IO_CAN_WriteMsg(can_handle_cb, // handle for writing
                &can_frame); // local CAN buffer
}

```

7.9.4.4 Example for safety critical Driver configuration

```

// Safety configuration for IO Driver
static const IO_DRIVER_SAFETY_CONF c_driver_safety_conf =
{
    40,                                // glitch_filter_time [ms]
    10000,                             // command_period [us]
    SAFETY_CONF_WINDOW_SIZE_50_PERCENT, // window_size: 50 percent
    watchdog,                          // watchdog window
    SAFETY_CONF_RESETS_DISABLED,        // reset_behavior: no resets allowed

    // error_callback:
    // (Set to \c #NULL if IO Driver should decide what to do on errors)
    &APPL_SafetyCb,

    // notify_callback:
    // (Set to \c #NULL if the application is not interested in the error code when the
    // safe state is entered and before a configured reset takes place)
    &APPL_NotifyCb
};

static ubyte4 timestamp = 0;

void task (void)
{
    IO_ErrorType rc_driv_begin;
    IO_ErrorType rc_driv_end;

    rc_driv_begin = IO_Driver_TaskBegin();

    if (rc_driv_begin != IO_E_OK)
    {
        // User code
    }

    // User Application
    // and calls to driver task functions.

    rc_driv_end = IO_Driver_TaskEnd();

    if (rc_driv_end != IO_E_OK)
    {
        // User code
    }

    while (IO_RTC_GetTimeUS(timestamp) < 10000); // wait until 10ms have passed
    timestamp += 10000;                         // increase time stamp by cycle time
}

```

```

}

void main (void)
{
    IO_ErrorType rc_driv_init;

    //-----
    // start of driver initialization //
    //-----

    // IO_Driver_Init() is the first function:
    rc_driv_init = IO_Driver_Init(&c_driver_safety_conf); // safety critical application

    if (rc_driv_init != IO_E_OK)
    {
        // User code
    }

    //-----
    // end of driver initialization //
    //-----


    // Get timestamp
    IO_RTC_StartTime(&timestamp);

    //-----
    // from now on only task functions are called //
    //-----
    while (1)
    {
        task();
    }
}

```

7.9.4.5 Example implementation for FPU-Callback

See [IO_DRIVER_FPU_HANDLER](#) for details on the FPU callback function.

```

static ubyte4 timestamp = 0;

static void FPU_Handler(bool division_by_zero,
                       bool input_denormal,
                       bool invalid_operation,
                       bool overflow,
                       bool underflow)
{
    if (division_by_zero != FALSE)
    {
        // Application specific handling
    }

    if (input_denormal != FALSE)
    {
        // Application specific handling
    }

    if (invalid_operation != FALSE)
    {
        // Application specific handling
    }
}

```

```

    }

    if (overflow != FALSE)
    {
        // Application specific handling
    }

    if (underflow != FALSE)
    {
        // Application specific handling
    }
}

void task (void)
{
    IO_Driver_TaskBegin();

    // User Application
    // and calls to driver task functions.

    IO_Driver_TaskEnd();

    while (IO_RTC_GetTimeUS(timestamp) < 5000);      // wait until 5ms have passed
    timestamp += 5000;                                // increase time stamp by cycle time
}

void main (void)
{
    //-----
    // start of driver initialization //
    //-----

    // IO_Driver_Init() is the first function:
    IO_Driver_Init(NULL); // no safety critical application

    // Enable FPU callback
    IO_Driver_SetFPUHandler(&FPU_Handler);

    //-----
    // end of driver initialization //
    //-----

    // Get timestamp
    IO_RTC_StartTime(&timestamp);

    //-----
    // from now on only task functions are called //
    //-----
    while(1)
    {
        task();
    }
}

```

7.9.5 Macro Definition Documentation

7.9.5.1 #define IO_DRIVER_ECU_MAC_ADD_LENGTH (12U)

ECU MAC address length.

Definition at line 607 of file IO_Driver.h.

7.9.5.2 #define IO_DRIVER_ECU_PROD_CODE_LENGTH (30U)

ECU production code length.

Definition at line 621 of file IO_Driver.h.

7.9.5.3 #define IO_DRIVER_ECU_SERIAL_LENGTH (14U)

ECU serial number length.

Definition at line 593 of file IO_Driver.h.

7.9.5.4 #define SAFETY_CONF_RESETS_1 (1U)

1 Reset allowed.

This means that the watchdog CPU can reset the device and try to restart the device for 1 time. For any further error the diagnostic state machine will enter the safe state.

Definition at line 531 of file IO_Driver.h.

7.9.5.5 #define SAFETY_CONF_RESETS_2 (2U)

2 Resets allowed.

This means that the watchdog CPU can reset the device and try to restart the device for 2 times. For any further error the diagnostic state machine will enter the safe state.

Definition at line 537 of file IO_Driver.h.

7.9.5.6 #define SAFETY_CONF_RESETS_3 (3U)

3 Resets allowed.

This means that the watchdog CPU can reset the device and try to restart the device for 3 times. For any further error the diagnostic state machine will enter the safe state.

Definition at line 543 of file IO_Driver.h.

7.9.5.7 #define SAFETY_CONF_RESETS_4 (4U)

4 Resets allowed.

This means that the watchdog CPU can reset the device and try to restart the device for 4 times. For any further error the diagnostic state machine will enter the safe state.

Definition at line 549 of file IO_Driver.h.

7.9.5.8 #define SAFETY_CONF_RESETS_5 (5U)

5 Resets allowed.

This means that the watchdog CPU can reset the device and try to restart the device for 5 times. For any further error the diagnostic state machine will enter the safe state.

Definition at line 555 of file IO_Driver.h.

7.9.5.9 #define SAFETY_CONF_RESETS_6 (6U)

6 Resets allowed.

This means that the watchdog CPU can reset the device and try to restart the device for 6 times. For any further error the diagnostic state machine will enter the safe state.

Definition at line 561 of file IO_Driver.h.

7.9.5.10 #define SAFETY_CONF_RESETS_7 (7U)

7 Resets allowed.

This means that the watchdog CPU can reset the device and try to restart the device for 7 times. For any further error the diagnostic state machine will enter the safe state.

Definition at line 567 of file IO_Driver.h.

7.9.5.11 #define SAFETY_CONF_RESETS_8 (8U)

8 Resets allowed.

This means that the watchdog CPU can reset the device and try to restart the device for 8 times. For any further error the diagnostic state machine will enter the safe state.

Definition at line 573 of file IO_Driver.h.

7.9.5.12 #define SAFETY_CONF_RESETS_9 (9U)

9 Resets allowed.

This means that the watchdog CPU can reset the device and try to restart the device for 9 times. For any further error the diagnostic state machine will enter the safe state.

Definition at line 579 of file IO_Driver.h.

7.9.5.13 #define SAFETY_CONF_RESETS_DISABLED (0U)

Resets disabled.

This means that the watchdog CPU can not reset the device and restart the device again. On any error the diagnostic state machine will directly enter a safe state.

Definition at line 525 of file IO_Driver.h.

7.9.5.14 #define SAFETY_CONF_WINDOW_SIZE_100_PERCENT (0U)

Watchdog window size is 100 percent of the maximum trigger time.

The resulting window starts from 0% and goes up to 200% the configured command period.

Remarks

This setting results in a timeout-only watchdog, i.e., the watchdog can be triggered any time before the timeout of twice the configured command period is reached.

Definition at line 473 of file IO_Driver.h.

7.9.5.15 #define SAFETY_CONF_WINDOW_SIZE_12_5_PERCENT (3U)

Watchdog window size is 12.5 percent of the maximum trigger time.

The resulting window starts from 93.4% and goes up to 106.6% of the configured command period.

The **actual window size** thus will be 13.3% (+/-6.6%) of the command period

Definition at line 494 of file IO_Driver.h.

7.9.5.16 #define SAFETY_CONF_WINDOW_SIZE_25_PERCENT (2U)

Watchdog window size is 25 percent of the maximum trigger time.

The resulting window starts from 86% and goes up to 114% of the configured command period.

The **actual window size** thus will be 28% (+/-14%) of the command period

Definition at line 487 of file IO_Driver.h.

7.9.5.17 #define SAFETY_CONF_WINDOW_SIZE_3_125_PERCENT (5U)

Watchdog window size is 3.125 percent of the maximum trigger time.

The resulting window starts from 98.5% and goes up to 101.5% of the configured command period.

The **actual window size** thus will be 3.174% (+/-1.587%) of the command period

Definition at line 508 of file IO_Driver.h.

7.9.5.18 #define SAFETY_CONF_WINDOW_SIZE_50_PERCENT (1U)

Watchdog window size is 50 percent of the maximum trigger time.

The resulting window starts from 66% and goes up to 133% of the configured command period.

The **actual window size** thus will be 66% (+/-33%) of the command period

Definition at line 480 of file IO_Driver.h.

7.9.5.19 #define SAFETY_CONF_WINDOW_SIZE_6_25_PERCENT (4U)

Watchdog window size is 6.25 percent of the maximum trigger time.

The resulting window starts from 96.7% and goes up to 103.2% of the configured command period.

The **actual window size** thus will be 6.45% (+/-3.22%) of the command period

Definition at line 501 of file IO_Driver.h.

7.9.6 Typedef Documentation

7.9.6.1 **typedef void(* IO_DRIVER_FPU_HANDLER)(bool division_by_zero, bool input_denormal, bool invalid_operation, bool overflow, bool underflow)**

Function pointer for FPU exception handler.

If a FPU exception occurs, the FPU unit notifies the application of the exception reason by calling this callback function.

The callback function is passed to the function [IO_Driver_SetFPUHandler\(..\)](#).

The exception reasons are passed to the callback function as parameter:

Parameters

<i>division_by_zero</i>	The exception is caused if a divide operation has a zero divisor and a dividend that is not zero, an infinity or a NaN
<i>input_denormal</i>	The exception is caused if a denormalized input operand is replaced in the computation by a zero
<i>invalid_operation</i>	The exception is caused if the result of an operation has no mathematical value or cannot be represented
<i>overflow</i>	The exception is caused if the absolute value of the result of an operation, produced after rounding, is greater than the maximum positive normalized number for the destination precision.
<i>underflow</i>	The exception is caused if the absolute value of the result of an operation, produced before rounding, is less than the minimum positive normalized number for the destination precision and the rounded result is inexact.

Remarks

See [Example implementation for FPU-Callback](#).

Definition at line 654 of file IO_Driver.h.

7.9.6.2 **typedef struct io_driver_safety_conf_ IO_DRIVER_SAFETY_CONF**

Driver Safety Configuration.

This structure is used to pass the configuration for safety critical application to the IO Driver.

Note

Note that the hereby defined values are the configuration values of the TMS570 watchdog configuration values! The processor offers a unique watchdog triggering technic, which does not based on a +/- window manner. In order to support a command period centric +/- watchdog window approach the underlying implementation - due to processor limitations - will distort the window size that was requested through the safety configuration.

To calculate your exact window size (in %) you can use the following formula:

actual_window_size = {400/(200-chooseen_wsize)-2}*100

where *chooseen_wsize* is one of the *SAFETY_CONF_WINDOW_SIZE* below (6.25%,12.5%,25%,etc).

Example, with a windows size choose 25%, the actual size will be 28,57%.

7.9.7 Function Documentation

7.9.7.1 IO_ErrorType IO_Driver_GetMacAddress (ubyte1 *const *macaddress*)

Returns the ECU's MAC address.

Parameters

out	<i>macaddress</i>	Pointer to the ECU's MAC address data structure of size <i>IO_DRIVER_ECU_MAC_ADD_LENGTH</i> . The read MAC address will be stored here, but not as a null-terminated string, only a list of characters.
-----	-------------------	---

Returns

IO_ErrorType

Return values

<i>IO_E_OK</i>	everything fine
<i>IO_E_NULL_POINTER</i>	null pointer has been passed
<i>IO_E_INVALID_PROD_DATA</i>	the Production Data is invalid

Remarks

The 12 digits of the MAC address are returned as ASCII code. The application has to convert it to a number representation if desired.

7.9.7.2 IO_ErrorType IO_Driver_GetProdCode (ubyte1 *const *prodcodes*)

Returns the ECU's production code.

Parameters

out	<i>prodcodes</i>	Pointer to the ECU's production code data structure of size <i>IO_DRIVER_ECU_PROD_CODE_LENGTH</i> . The read production code will be stored here, but not as a null-terminated string, only a list of characters.
-----	------------------	---

Byte	Description
00-06	ECU type
07-11	Further ECU details
12	Housing (S = sealed O = open)
13-20	Production BOM Version
21-25	Software Bundle ID
26-29	Product Label Version

Example:
 "TTC-58000000SV010000BU0368L51A"

Values	Description
TTC-580	ECU type HY TTC-580
00000	Further ECU details
S	S = sealed
V010000B	Production BOM Version V01.00.00-B
U0368	Software Bundle U0368
L51A	Product Label Version L51A

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_NULL_POINTER</code>	null pointer has been passed
<code>IO_E_INVALID_PROD_DATA</code>	the Production Data is invalid

Remarks

The 30 digits of the production code are returned as ASCII code. The application has to convert it to a number representation if desired.

7.9.7.3 `IO_ErrorType IO_Driver_GetSerialNumber (ubyte1 *const serialnumber)`

Returns the ECU's serial number.

Parameters

out	<code>serialnumber</code>	Pointer to the ECU's serial number data structure of size <code>IO_DRIVER_ECU_SERIAL_LENGTH</code> . The read serial number will be stored here, but not as a null-terminated string, only a list of characters.
-----	---------------------------	--

Byte	Description
00	last digit of year
01-02	calendar week
03	day of the week (1=Monday)
04-07	daily counter

Byte	Description
08-12	product number
13	product index

Example:
"30820015008180"

Values	Description
3	year 2013
08	Calendar week 8
2	day of week (Tuesday)
0015	daily counter
00818	product number 00818
0	product index 0

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_NULL_POINTER</code>	null pointer has been passed
<code>IO_E_INVALID_PROD_DATA</code>	the Production Data is invalid
<code>IO_E_INVALID_SERIAL_NUMBER</code>	the ECU's serial number is invalid

Remarks

The 14 digits of the serial number are returned as ASCII code. The application has to convert it to a number representation if desired.

7.9.7.4 `IO_ErrorType IO_Driver_GetVersionOfBootloader (ubyte1 *const major, ubyte1 *const minor)`

Returns the version number of the Bootloader.

Parameters

out	<code>major</code>	Major version
out	<code>minor</code>	Minor version

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_NULL_POINTER</code>	null pointer has been passed
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	shared memory is not initialized
<code>IO_E_SHM_INTEGRITY</code>	shared memory is corrupted

7.9.7.5 IO_ErrorType IO_Driver_GetVersionOfDriver (ubyte1 *const *major*, ubyte1 *const *minor*, ubyte2 *const *patchlevel*)

Returns the version number of the IO driver.

Parameters

out	<i>major</i>	Major version
out	<i>minor</i>	Minor version
out	<i>patchlevel</i>	Patchlevel

Returns

IO_ErrorType

Return values

<i>IO_E_OK</i>	everything fine
<i>IO_E_NULL_POINTER</i>	null pointer has been passed

7.9.7.6 IO_ErrorType IO_Driver_GetVersionOfFPGA (ubyte2 *const *rev0*, ubyte2 *const *rev1*, ubyte2 *const *rev2*, ubyte1 *const *device*, ubyte1 *const *release*, ubyte1 *const *patchlevel*)

Returns the version number of the FPGA IP.

Parameters

out	<i>rev0</i>	Revision 0, for internal use only
out	<i>rev1</i>	Revision 1, for internal use only
out	<i>rev2</i>	Revision 2, for internal use only
out	<i>device</i>	Device type
out	<i>release</i>	Release version
out	<i>patchlevel</i>	Patchlevel

Returns

IO_ErrorType

Return values

<i>IO_E_OK</i>	everything fine
<i>IO_E_NULL_POINTER</i>	null pointer has been passed
<i>IO_E_FPGA_NOT_INITIALIZED</i>	the FPGA has not been initialized

7.9.7.7 IO_ErrorType IO_Driver_Init (const IO_DRIVER_SAFETY_CONF *const *safety_conf*)

Global initialization of IO driver.

This function shall be called before any other driver function (except functions `IO_Driver_GetVersionOfDriver()`, `IO_Driver_GetVersionOfBootloader()`, `IO_Driver_GetSerialNumber()` and `IO_Driver_GetMacAddress()`).

- Switches off all power outputs
- Initializes the RTC
- Switches on the interrupts of the CPU
- Initializes the internal measurements
- Initializes the checker modules

Parameters

<code>in</code>	<code>safety_conf</code>	Configuration for safety critical applications. Set pointer to <code>NULL</code> to disable the safety features.
-----------------	--------------------------	--

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	Initialization successful
<code>IO_E_INVALID_SAFETY_CONFIG</code>	An invalid safety configuration has been passed
<code>IO_E_CHANNEL_BUSY</code>	The I/O Driver was already initialized
<code>IO_E_INVALID_PROD_DATA</code>	The ECU's production data is corrupted
<code>IO_E_INVALID_SERIAL_NUMBER</code>	The ECU's serial number is invalid
<code>IO_E_INVALID_VARIANT</code>	The used ECU variant is not valid
<code>IO_E_FPGA_TIMEOUT</code>	The FPGA did not respond in time during its configuration
<code>IO_E_FPGA_CRC_ERROR</code>	The FPGA reported a CRC error
<code>IO_E_FPGA_VERSION</code>	The FPGA version does not match with this I/O Driver version
<code>IO_E_FPGA_IMAGE</code>	Error in the FPGA image
<code>IO_E_WD_INITIALIZATION</code>	The watchdog system could not be initialized
<code>IO_E_WD_PRECISION</code>	The configured limits require too high precision
<code>IO_E_WD_SAFE_LOCK</code>	The ECU is already locked in safe state
<code>IO_E_WD_DEBUGGING_PREPARED</code>	The watchdog system has been prepared for debugging
<code>IO_E_INTERNAL_CSM</code>	An error in the internal low level driver occurred
<code>IO_E_UNKNOWN</code>	A further unspecified error occurred

Remarks

If `safety_conf != NULL`, the application is configured safety critical. The parameter `safety_conf` defines the global safety properties. Then in other Init functions, like `IO_ADC_ChannelInit()`, each channel itself can be configured safety critical.

Note

The watchdog window size has to be configured to provide a minimum positive window of at least 700 microseconds (including losses caused by rounding) in order to guarantee that all internal background mechanisms cannot lead to runtime violations in worst case scenarios. In case this limit is not fulfilled, `IO_E_WD_PRECISION` is returned.

7.9.7.8 IO_ErrorType IO_Driver_Reset (void)

Performs a software reset of the device. The intended use of this function is to enter the bootloader from an application.

Remarks

In order to perform the reset the watchdog must be in either state `DIAG_WD_STATE_SAFE` or `DIAG_WD_STATE_ACTIVE`.

Upon reset if the application is entered the device will boot into safe state and the `IO_Driver_Init` will return an error.

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_BUSY</code>	reset could not be executed due to a wrong state
<code>IO_E_WD_STATUS_INVALID</code>	watchdog status couldn't be obtained
<code>IO_E_SHM_INTEGRITY</code>	the shared memory's integrity is invalid
<code>IO_E_INTERNAL_CSM</code>	an internal error occurred

7.9.7.9 IO_ErrorType IO_Driver_SetFPUHandler (IO_DRIVER_FPU_HANDLER *fpu_handler*)

Registers an application callback for FPU exceptions.

Parameters

in	<i>fpu_handler</i>	Application FPU handler
----	--------------------	-------------------------

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_DRIVER_NOT_INITIALIZED</code>	the driver has not been initialized
<code>IO_E_UNKNOWN</code>	an unknown error occurred

Remarks

If the parameter `fpu_handler` is set to `NULL` the FPU exception application callback is disabled again.

7.9.7.10 IO_ErrorType IO_Driver_SetIntegerDivisionByZeroException (bool *enable*)

Enables/disables integer division by zero exceptions.

Parameters

in	<i>enable</i>	Integer division by zero exception mode: • TRUE : integer division by zero exception enabled • FALSE : integer division by zero exception disabled
----	---------------	--

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_DRIVER_NOT_INITIALIZED</code>	the driver has not been initialized

Remarks

If the integer division by zero exception is disabled, an integer division by zero always results in the value of zero.

The integer division by zero exception is disabled by default.

7.9.7.11 `IO_ErrorType IO_Driver_TaskBegin (void)`

Task function for IO Driver. This function shall be called at the beginning of the task.

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_WD_SELF_MONITORING</code>	the function has not been called at the right point in time
<code>IO_E_WD_VICE_VERSA_MONITORING</code>	the triggering of the external watchdog exceeded it's limit
<code>IO_E_DRIVER_NOT_INITIALIZED</code>	the driver has not been initialized
<code>IO_E_UNKNOWN</code>	an unknown error occurred

7.9.7.12 `IO_ErrorType IO_Driver_TaskEnd (void)`

Task function for IO Driver. This function shall be called at the end of the task.

Returns

`IO_ErrorType`

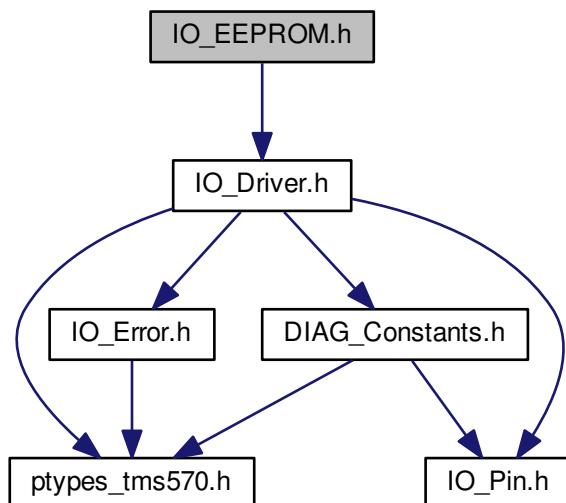
Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_DRIVER_NOT_INITIALIZED</code>	the driver has not been initialized

7.10 IO_EEPROM.h File Reference

IO Driver functions for external EEPROM/FRAM.

Include dependency graph for IO_EEPROM.h:



Macros

- `#define IO_EEPROM_Process IO_EEPROM_GetStatus`
The Macro maintains backwards compatibility for applications using the `IO_EEPROM_Process()` from I/O Driver 3.1.

Functions

- `IO_ErrorType IO_EEPROM_DelInit(void)`
Deinitializes the EEPROM driver.
- `IO_ErrorType IO_EEPROM_GetStatus(void)`
Returns the status of the EEPROM driver and triggers the processing of the read and write operations.

- `IO_ErrorType IO_EEPROM_Init (void)`
Initialization of the EEPROM driver.
- `IO_ErrorType IO_EEPROM_Read (ubyte2 offset, ubyte4 length, ubyte1 *const data)`
Read data from the EEPROM/FRAM.
- `IO_ErrorType IO_EEPROM_Write (ubyte2 offset, ubyte4 length, const ubyte1 *const data)`
Write data to the EEPROM/FRAM.

7.10.1 Detailed Description

IO Driver functions for external EEPROM/FRAM.

The EEPROM module allows writing to and reading from the external SPI EEPROM or FRAM (depending on the product variant).

All functions are strictly non-blocking (i.e. they return immediately). The EEPROM/FRAM operations, including the communication on the SPI bus, can be lengthy and for this reason they are not performed in the API functions but rather "in the background": The API calls (`IO_EEPROM_Read()` or `IO_EEPROM_Write()`) initiate the reading/writing, after which the `IO_EEPROM_GetStatus()` function can be called to see if the operation finished or is still ongoing.

The background processing of the read or write operations takes place in a periodic (1 ms) interrupt. This happens without the user application's intervention and guarantees the minimum read and write data transfer rates (see below). If these minimum rates are not sufficient, they can be substantially increased by the user application calling `IO_EEPROM_GetStatus()` in a polling loop. Other than returning the status of the EEPROM/FRAM driver, this function performs the processing of the read/write operations. These operations consist of a sequence of SPI commands that take typ. several us. When the `IO_EEPROM_GetStatus()` is polled, it will advance to the next stage as soon as the previous one is finished without waiting for the next 1 ms period.

Attention

In situations where interrupts are disabled, such as in the error or notification callbacks, the polling method is necessary.

7.10.2 Speed of the EEPROM/FRAM Operations

Due to internal limitations, the size of the SPI communication buffer is limited to 64 B. This is also the effective page size for the EEPROM/FRAM.

Note

The communication buffer size has impact on the performance of the read/write operations but it does not limit the size of the data that can be read or written by the API functions. The API functions will process the read/write requests over multiple pages if necessary.

7.10.2.1 Read and write speeds of the background processing

The maximum duration of **reading** an EEPROM/FRAM page, when relying on the background interrupt, is 3 ms. The full page buffer is used when the data start address and size is aligned to 64 B. When this is fulfilled, the EEPROM driver guarantees the minimum read data rate of 21333 B/s.

Writing requires min. 4 ms per page. The EEPROM needs additional 5 ms for the internal programming cycle (max), i.e. the maximum duration for writing one page is 9 ms. With the data aligned to 64 B, the minimum write data rate for the EEPROM is 7111 B/s. The FRAM adds no additional waiting time for its internal programming. The maximum write time of one page is 5 ms and the minimum write data rate is 12800 B/s.

7.10.2.2 Read and write speeds with the IO_EEPROM_GetStatus() polling

The values given below assume that the minimum read/written data size is 64 B. The alignment is not required.

With the polling method, the speed of reading the EEPROM or FRAM is limited by the bandwidth of the SPI bus. The **read** data rate that can be achieved is 267000 B/s and higher.

The **write** speed achievable for the FRAM is 247000 B/s. The EEPROM write speed cannot be significantly improved by the polling because it is limited by the internal 5 ms programming delay (max), i.e. to 12800 B/s.

The data rates for the polling method are not guaranteed because they depend on the frequency of the polling. The desired polling period is 1.6 us and shorter: If this is fulfilled, the specified data rates will surely be exceeded. In practice, however, the polling frequency depends on various CPU load conditions (such as the background interrupts) that are difficult to control. The given data rates are typical values verified by measurements in a test application. They should be met with a high degree of confidence in the majority of customer applications.

7.10.3 EEPROM Code Examples

Examples for using the EEPROM API

7.10.3.1 Example for EEPROM/FRAM initialization

EEPROM needs to be initialized after `IO_Driver_Init()`.

```
IO_ErrorType io_error;  
// initialize driver
```

```

io_error = IO_Driver_Init(NULL);

// initialize EEPROM module
io_error = IO_EEPROM_Init();

```

7.10.3.2 Example for EEPROM/FRAM write

```

ubyte1 data[6] = {0, 1, 2, 3, 4, 5};

// check if EEPROM is busy
if (IO_EEPROM_GetStatus() == IO_E_OK)
{
    // if not busy write data
    io_error = IO_EEPROM_Write(0,      // offset
                               6,       // length
                               data);   // data buffer
}

// Write is complete when IO_EEPROM_GetStatus() returns IO_E_OK again.

```

7.10.3.3 Example for EEPROM/FRAM read

```

ubyte1 data[2000] = {0};

// check if EEPROM is busy
if (IO_EEPROM_GetStatus() == IO_E_OK)
{
    // if not busy start reading
    io_error = IO_EEPROM_Read(0,      // offset
                             2000,    // length
                             data);   // data buffer
}

// Data is not yet available!
// Data is available when IO_EEPROM_GetStatus() returns IO_E_OK again.

```

7.10.3.4 Polling the EEPROM/FRAM status

```

while (IO_EEPROM_GetStatus() == IO_E_BUSY)
{
    // The user application can do its internal processing here
    // but decreasing the frequency of the polling will decrease the attained read/write speed.
}

if (IO_EEPROM_GetStatus() == IO_E_OK)
{
    // The previous read or write operation completed successfully.
}

```

7.10.4 Macro Definition Documentation

7.10.4.1 #define IO_EEPROM_Process IO_EEPROM_GetStatus

The Macro maintains backwards compatibility for applications using the `IO_EEPROM_Process()` from I/O Driver 3.1.

Definition at line 162 of file IO_EEPROM.h.

7.10.5 Function Documentation

7.10.5.1 IO_ErrorType IO_EEPROM_DeInit (void)

Deinitializes the EEPROM driver.

Returns

[IO_ErrorType](#)

Return values

IO_E_OK	Everything fine.
IO_E_CHANNEL_NOT_CONFIGURED	The module is not initialized.

7.10.5.2 IO_ErrorType IO_EEPROM_GetStatus (void)

Returns the status of the EEPROM driver and triggers the processing of the read and write operations.

It can be used to determine whether the EEPROM/FRAM is idle or if a read or write operation is ongoing. After an unsuccessful read or write operation, [IO_EEPROM_GetStatus\(\)](#) returns the error code and clears it.

This function also triggers the internal processing of the read/write operations. This feature can be used to speed up the read/write operations (see [Speed of the EEPROM/FRAM Operations](#)).

Remarks

Periodic polling using this function is necessary to trigger the processing of the read/write operations in situations where interrupts are not available, such as within the error or notification callbacks.

If used from the error/notification callback, the nature of the failure will effect whether the EEPROM is usable. In these cases it is important to include a timeout on any operations.

Returns

[IO_ErrorType](#)

Return values

IO_E_OK	Everything fine, driver is idle.
IO_E_BUSY	A read or a write operation is ongoing, driver is busy.
IO_E_UNKNOWN	An internal error has occurred.
IO_E EEPROM_SPI	An internal SPI error has occurred when communicating with the EEPROM/FRAM.
IO_E_CHANNEL_NOT_CONFIGURED	The module is not initialized.

7.10.5.3 IO_ErrorType IO_EEPROM_Init (void)

Initialization of the EEPROM driver.

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	Everything fine.
<code>IO_E_INVALID_CHANNEL_ID</code>	EEPROM/FRAM is not supported by the used ECU variant.
<code>IO_E_INVALID_PIN_CONFIG</code>	Error in the ECU variant configuration data.
<code>IO_E_INVALID_VARIANT</code>	The detected ECU variant is not supported.
<code>IO_E_CHANNEL_BUSY</code>	Module has been initialized before.
<code>IO_E EEPROM_SPI</code>	SPI bus was not correctly initialized.
<code>IO_E_DRIVER_NOT_INITIALIZED</code>	The <code>IO_Driver_Init</code> function has not been called.

Remarks

The EEPROM driver can only be initialized once. Prior to re-initialization, the `IO_EEPROM_DeInit()` function must be called.

7.10.5.4 IO_ErrorType IO_EEPROM_Read (ubyte2 offset, ubyte4 length, ubyte1 *const data)

Read data from the EEPROM/FRAM.

This function triggers the read operation.

The read operation is processed in the background; its state can be polled using the `IO_EEPROM_GetStatus()` function.

The read data is available in the `data` buffer after the read operation finishes.

Parameters

<code>offset</code>	EEPROM/FRAM memory offset (0..65535 for EEPROM, 0..32767 for FRAM)
<code>length</code>	Length of data to be read (1..65536 for EEPROM, 1..32768 for FRAM)
<code>out</code>	Pointer to the address where the data shall be stored to

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	Everything fine.
<code>IO_E_BUSY</code>	EEPROM module is still busy.
<code>IO_E_INVALID_PARAMETER</code>	Length is zero.
<code>IO_E EEPROM_RANGE</code>	Invalid address offset or range.

<code>IO_E_NULL_POINTER</code>	A NULL pointer has been passed.
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	The module is not initialized.

7.10.5.5 IO_ErrorType IO_EEPROM_Write (**ubyte2 offset**, **ubyte4 length**, **const ubyte1 *const data**)

Write data to the EEPROM/FRAM.

The function triggers a write operation.

The write operation is processed in the background; its state can be polled using the `IO_EEPROM_GetStatus()` function.

The data to be written must be available in the `data` buffer for the whole duration of the the write operation!

Parameters

<code>offset</code>	EEPROM/FRAM memory offset (0..65535 for EEPROM, 0..32767 for FRAM)
<code>length</code>	Length of the data to be written (1..65536 for EEPROM, 1..32768 for FRAM)
<code>in</code>	<code>data</code> Pointer to the data to be written

Returns

`IO_ErrorType`

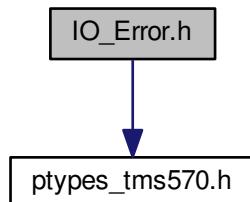
Return values

<code>IO_E_OK</code>	Everything fine.
<code>IO_E_BUSY</code>	EEPROM module is still busy.
<code>IO_E_INVALID_PARAMETER</code>	Length is zero.
<code>IO_E EEPROM_RANGE</code>	Invalid address offset or range.
<code>IO_E NULL_POINTER</code>	A NULL pointer has been passed.
<code>IO_E CHANNEL_NOT_CONFIGURED</code>	The module is not initialized.

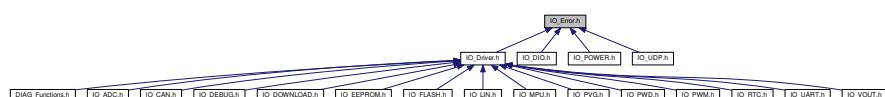
7.11 IO_Error.h File Reference

Global error defines for IO driver.

Include dependency graph for IO_Error.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define `IO_E_BUSY` 2U
- #define `IO_E_CAN_BUS_OFF` 47U
- #define `IO_E_CAN_ERROR_PASSIVE` 46U
- #define `IO_E_CAN_ERROR_WARNING` 48U
- #define `IO_E_CAN_FIFO_FULL` 44U
- #define `IO_E_CAN_MAX_HANDLES_REACHED` 43U
- #define `IO_E_CAN_MAX_MO_REACHED` 42U
- #define `IO_E_CAN_OLD_DATA` 45U
- #define `IO_E_CAN_OVERFLOW` 40U
- #define `IO_E_CAN_TIMEOUT` 49U
- #define `IO_E_CAN_WRONG_HANDLE` 41U
- #define `IO_E_CH_CAPABILITY` 37U
- #define `IO_E_CHANNEL_BUSY` 32U
- #define `IO_E_CHANNEL_NOT_CONFIGURED` 33U
- #define `IO_E_CM_CALIBRATION` 146U
- #define `IO_E_CORE_TEST_FAILED` 286U
- #define `IO_E_DOWNLOAD_HANDSHAKE` 312U
- #define `IO_E_DOWNLOAD_NO_REQ` 310U
- #define `IO_E_DOWNLOAD_TIMEOUT` 311U
- #define `IO_E_DRIVER_NOT_INITIALIZED` 38U
- #define `IO_E_DRV_SAFETY_CONF_NOT_CONFIG` 5U
- #define `IO_E_DRV_SAFETY_CYCLE_RUNNING` 7U
- #define `IO_E_EEPROM_RANGE` 60U

- #define IO_E EEPROM_SPI 61U
- #define IO_E_ERROR_PIN_TEST_FAILED 288U
- #define IO_E_ERROR_PIN_TEST_TIMEOUT 287U
- #define IO_E_ETH_DEINIT_TIMEOUT 302U
- #define IO_E_ETH_INIT_FAIL 300U
- #define IO_E_ETH_INIT_TIMEOUT 301U
- #define IO_E_ETH_MAC_INVALID 303U
- #define IO_E_ETH_MDIO_READ 308U
- #define IO_E_ETH_MDIO_TIMEOUT 307U
- #define IO_E_ETH_NO_LINK 306U
- #define IO_E_ETH_READ_FAIL 304U
- #define IO_E_ETH_WRITE_FAIL 305U
- #define IO_E_FET_PROT_ACTIVE 20U
- #define IO_E_FET_PROT_NOT_ACTIVE 24U
- #define IO_E_FET_PROT_PERMANENT 21U
- #define IO_E_FET_PROT_REENABLE 22U
- #define IO_E_FET_PROT_WAIT 23U
- #define IO_E_FLASH_BLANK_CHECK_FAILED 270U
- #define IO_E_FLASH_OP_FAILED 271U
- #define IO_E_FLASH_OP_TIMEOUT 272U
- #define IO_E_FLASH_SUSPENDED 273U
- #define IO_E_FLASH_WRONG_DEVICE_ID 269U
- #define IO_E_FPGA_CRC_ERROR 202U
- #define IO_E_FPGA_IMAGE 204U
- #define IO_E_FPGA_NOT_INITIALIZED 200U
- #define IO_E_FPGA_TIMEOUT 201U
- #define IO_E_FPGA_VERSION 203U
- #define IO_E_INTERNAL_CSM 292U
- #define IO_E_INTERNAL_MEM_FAILED 289U
- #define IO_E_INVALID_CHANNEL_ID 34U
- #define IO_E_INVALID_ESM_INIT_STATUS 290U
- #define IO_E_INVALID_LIMITS 35U
- #define IO_E_INVALID_OPERATION 39U
- #define IO_E_INVALID_PARAMETER 31U
- #define IO_E_INVALID_PIN_CONFIG 284U
- #define IO_E_INVALID_PRD_DATA 283U
- #define IO_E_INVALID_SAFETY_CONFIG 6U
- #define IO_E_INVALID_SERIAL_NUMBER 285U
- #define IO_E_INVALID_VARIANT 282U
- #define IO_E_INVALID_VOLTAGE 14U
- #define IO_E_LIN_BIT 260U
- #define IO_E_LIN_CHECKSUM 262U
- #define IO_E_LIN_FRAMING 265U
- #define IO_E_LIN_INCONSISTENT_SYNCH_FIELD 263U
- #define IO_E_LIN_NO_RESPONSE 264U
- #define IO_E_LIN_OVERRUN 266U
- #define IO_E_LIN_PARITY 267U
- #define IO_E_LIN_PHYSICAL_BUS 261U
- #define IO_E_LIN_TIMEOUT 268U
- #define IO_E_MPU_REGION_DISABLED 331U
- #define IO_E_MPU_REGION_ENABLED 330U
- #define IO_E_NO_DIAG 15U
- #define IO_E_NULL_POINTER 30U
- #define IO_E_OK 0U
- #define IO_E_OPEN_LOAD 10U
- #define IO_E_OPEN_LOAD_OR_SHORT_BAT 13U

- #define IO_E_OPTION_NOT_SUPPORTED 340U
- #define IO_E_PERIODIC_NOT_CONFIGURED 36U
- #define IO_E_PWD_CURRENT_THRESH_HIGH 104U
- #define IO_E_PWD_CURRENT_THRESH_LOW 105U
- #define IO_E_PWD_NOT_FINISHED 101U
- #define IO_E_PWD_OVERFLOW 102U
- #define IO_E_PWM_NO_DIAG IO_E_NO_DIAG
- #define IO_E_REFERENCE 18U
- #define IO_E_RESOLVING 25U
- #define IO_E_RESOLVING_FAILED 26U
- #define IO_E_RTC_CLOCK_INTEGRITY_FAILED 250U
- #define IO_E_SAFE_STATE 17U
- #define IO_E_SAFETY_SWITCH_DISABLED 19U
- #define IO_E_SHM_INTEGRITY 320U
- #define IO_E_SHORT_BAT 12U
- #define IO_E_SHORT_GND 11U
- #define IO_E_SOCKET_NOT_INITIALIZED 444U
- #define IO_E_STARTUP 16U
- #define IO_E_UART_BUFFER_EMPTY 71U
- #define IO_E_UART_BUFFER_FULL 70U
- #define IO_E_UART_DMA 75U
- #define IO_E_UART_FRAMING 74U
- #define IO_E_UART_OVERFLOW 72U
- #define IO_E_UART_PARITY 73U
- #define IO_E_UDP_ARP RECEIVED 449U
- #define IO_E_UDP_INVALID_BUFFER 446U
- #define IO_E_UDP_NOMORESOCKETS 440U
- #define IO_E_UDP_OVERFLOW 442U
- #define IO_E_UDP_WRONG_HANDLE 443U
- #define IO_E_UDP_WRONG_PORT 447U
- #define IO_E_UNKNOWN 3U
- #define IO_E_WD_ACTIVATION 243U
- #define IO_E_WD_DEBUGGING_PREPARED 246U
- #define IO_E_WD_INITIALIZATION 241U
- #define IO_E_WD_PRECISION 242U
- #define IO_E_WD_RANGE 240U
- #define IO_E_WD_SAFE_LOCK 245U
- #define IO_E_WD_SELF_MONITORING 247U
- #define IO_E_WD_STATUS_INVALID 249U
- #define IO_E_WD_TRIGGER 244U
- #define IO_E_WD_VICE_VERSA_MONITORING 248U
- #define IO_E_WRONG_ADDRESS 445U

Typedefs

- typedef ubyte2 IO_ErrorType
IO Driver error type.

7.11.1 Detailed Description

Global error defines for IO driver.

This header file defines all the error codes for the IO driver.

7.11.2 Macro Definition Documentation

7.11.2.1 #define IO_E_BUSY 2U

Module or function is busy.

This error is reported if a function or module has not yet finished its task. For example the EEPROM write function will return this error code if a previous write command has not been finished yet.

Definition at line 63 of file IO_Error.h.

7.11.2.2 #define IO_E_CAN_BUS_OFF 47U

The CAN node is in bus off state.

The transmit error counter has reached 255. The device remains in this state until 128 occurrences of bus idle (128 * 11 consecutive recessive bits). The bus-off recovery sequence is triggered automatically.

Definition at line 430 of file IO_Error.h.

7.11.2.3 #define IO_E_CAN_ERROR_PASSIVE 46U

The CAN node is in error passive state.

At least one of the error counters has reached 128.

Definition at line 423 of file IO_Error.h.

7.11.2.4 #define IO_E_CAN_ERROR_WARNING 48U

The error counters of the CAN node have reached the warning state. The CAN node is still in active state.

At least one of the error counters has reached the warning limit of 96.

Definition at line 436 of file IO_Error.h.

7.11.2.5 #define IO_E_CAN_FIFO_FULL 44U

The FIFO is full, one or more new messages have been lost.

Definition at line 413 of file IO_Error.h.

7.11.2.6 #define IO_E_CAN_MAX_HANDLES_REACHED 43U

No more message handles are available.

The maximum number of message handles has been reached. A message handle is generated every time the function `IO_CAN_ConfigMsg()` or `IO_CAN_ConfigFIFO()` is called without returning an error.

Definition at line 410 of file IO_Error.h.

7.11.2.7 #define IO_E_CAN_MAX_MO_REACHED 42U

No more message objects are available.

The maximum number of available message objects has been reached. A single message object is needed to setting up a single message object with the function [IO_CAN_ConfigMsg\(\)](#). When configuring a FIFO buffer with the function [IO_CAN_ConfigFIFO\(\)](#) the number of needed message objects equals the size of the FIFO buffer (single message objects are joined together to a FIFO buffer).

Definition at line 403 of file IO_Error.h.

7.11.2.8 #define IO_E_CAN_OLD_DATA 45U

No new data is available.

This error is returned if no CAN frame has been received since the last successful read.

Definition at line 418 of file IO_Error.h.

7.11.2.9 #define IO_E_CAN_OVERFLOW 40U

Message object or FIFO buffer overflow.

This error is reported if CAN messages have been lost due to a full buffer. To avoid this error FIFO buffers can be used. If FIFO buffers are already used, try to increase the buffer size.

Definition at line 386 of file IO_Error.h.

7.11.2.10 #define IO_E_CAN_TIMEOUT 49U

The CAN node reported a timeout during initialization or message transfer.

Try again to initialize or to transfer.

Definition at line 441 of file IO_Error.h.

7.11.2.11 #define IO_E_CAN_WRONG_HANDLE 41U

A wrong or invalid handle has been used.

This error is reported if:

- a non-existent handle has been used.
- if a write handle has been passed to a read function or vice versa
- if a message object handle has been passed to a FIFO function or vice versa

Definition at line 394 of file IO_Error.h.

7.11.2.12 #define IO_E_CH_CAPABILITY 37U

The IO channel (IO pin) does not support the requested feature.

Two conditions can lead to this error code:

- When, for example, trying to initialize or use a ADC pin as PWM output.
- When trying to initialize an IO for a pin function which is not available on the ECU variant. (For example when trying to initialize a PWD input but it is not physically mounted on the used ECU variant)

Definition at line 363 of file IO_Error.h.

7.11.2.13 #define IO_E_CHANNEL_BUSY 32U

The IO channel (IO pin) is busy

This error is reported if a IO Pin has been initialized before.

To change the configuration of the channel during runtime the according DelInit function needs to be called before the channel can be initialized with a new configuration.

Definition at line 330 of file IO_Error.h.

7.11.2.14 #define IO_E_CHANNEL_NOT_CONFIGURED 33U

The IO channel (IO pin) has not been initialized.

This error is reported by an IO driver task function if the channel has not been initialized. To initialize the channel, the according Init function needs to be called.

Definition at line 336 of file IO_Error.h.

7.11.2.15 #define IO_E_CM_CALIBRATION 146U

The current measurement calibration failed

Definition at line 515 of file IO_Error.h.

7.11.2.16 #define IO_E_CORE_TEST_FAILED 286U

An error has been detected during the test of the core.

Definition at line 706 of file IO_Error.h.

7.11.2.17 #define IO_E_DOWNLOAD_HANDSHAKE 312U

An error as been detected when establishing the handshake with the TTC-Downloader.

Definition at line 781 of file IO_Error.h.

7.11.2.18 #define IO_E_DOWNLOAD_NO_REQ 310U

No valid request from the TTC-Downloader has been detected.

Definition at line 773 of file IO_Error.h.

7.11.2.19 #define IO_E_DOWNLOAD_TIMEOUT 311U

A timeout has been detected when establishing the handshake with the TTC-Downloader.

Definition at line 777 of file IO_Error.h.

7.11.2.20 #define IO_E_DRIVER_NOT_INITIALIZED 38U

The common driver init function `IO_Driver_Init()` has not been called.

This error code is reported by the I/O driver init functions if the common init function `IO_Driver_Init()` has not been called or reported an error.

Definition at line 369 of file IO_Error.h.

7.11.2.21 #define IO_E_DRV_SAFETY_CONF_NOT_CONFIG 5U

Global safety configuration is missing

This error is reported if an IO is defined as safety critical although no safety configuration has been passed to the `IO_Driver_Init()` (parameter `safety_conf`) function. An IO pin is considered as safety critical if a valid safety configuration has been passed to the init function (see `safety_conf` parameter of the functions `IO_ADC_ChannelInit()`, `IO_PWM_Init()`, `IO_DO_Init()`, `IO_PWD_IncInit()`, `IO_PWD_CountInit()` and `IO_PWD_ComplexInit()`)

Definition at line 78 of file IO_Error.h.

7.11.2.22 #define IO_E_DRV_SAFETY_CYCLE_RUNNING 7U

The pin can not be configured anymore as safety critical, as the application cycle was already started.

This error is reported if a pin is initialized as safety critical after the cyclic task begin function `IO_Driver_TaskBegin()` is called. An IO pin is considered as safety critical if a valid safety configuration has been passed to the init function (see `safety_conf` parameter of the functions `IO_ADC_ChannelInit()`, `IO_PWM_Init()`, `IO_DO_Init()`, `IO_PWD_IncInit()`, `IO_PWD_CountInit()` and `IO_PWD_ComplexInit()`)

Definition at line 96 of file IO_Error.h.

7.11.2.23 #define IO_E EEPROM RANGE 60U

Invalid address range.

This error is reported if read or write operations are requested for non-existent EEPROM addresses.

Definition at line 450 of file IO_Error.h.

7.11.2.24 #define IO_E EEPROM_SPI 61U

Internal SPI error.

This error is reported if an error is detected in the SPI communication with the external EEPROM/FRAM.

Definition at line 455 of file IO_Error.h.

7.11.2.25 #define IO_E_ERROR_PIN_TEST_FAILED 288U

An error has been detected during the test of the TMS error pin.

Definition at line 714 of file IO_Error.h.

7.11.2.26 #define IO_E_ERROR_PIN_TEST_TIMEOUT 287U

A timeout error has been detected during the test of the TMS error pin.

Definition at line 710 of file IO_Error.h.

7.11.2.27 #define IO_E_ETH_DEINIT_TIMEOUT 302U

A timeout has been detected during the de-initialization of the Ethernet interface.

Definition at line 741 of file IO_Error.h.

7.11.2.28 #define IO_E_ETH_INIT_FAIL 300U

An error has been detected during the initialization of the Ethernet interface.

Definition at line 733 of file IO_Error.h.

7.11.2.29 #define IO_E_ETH_INIT_TIMEOUT 301U

A timeout has been detected during the initialization of the Ethernet interface.

Definition at line 737 of file IO_Error.h.

7.11.2.30 #define IO_E_ETH_MAC_INVALID 303U

An invalid MAC address has been read out from the production database

Definition at line 745 of file IO_Error.h.

7.11.2.31 #define IO_E_ETH_MDIO_READ 308U

An error has been detected during a read operation on the MDIO interface.

Definition at line 765 of file IO_Error.h.

7.11.2.32 #define IO_E_ETH_MDIO_TIMEOUT 307U

A timeout has been detected during a read operation on the MDIO interface.

Definition at line 761 of file IO_Error.h.

7.11.2.33 #define IO_E_ETH_NO_LINK 306U

No valid link has been established on the Ethernet interface.

Definition at line 757 of file IO_Error.h.

7.11.2.34 #define IO_E_ETH_READ_FAIL 304U

An error has been detected during a read operation on the Ethernet interface.

Definition at line 749 of file IO_Error.h.

7.11.2.35 #define IO_E_ETH_WRITE_FAIL 305U

An error has been detected during a write operation on the Ethernet interface.

Definition at line 753 of file IO_Error.h.

7.11.2.36 #define IO_E_FET_PROT_ACTIVE 20U

An internal switch (FET) or output has been disabled to protect it from damage.

The output or switch can be reenabled after a wait time. When this time has passed, [IO_E_FET_PROT_REENABLE](#) will be returned.

If the current on an internal FET or an output is too high, the FET will be switched off by software to protect it from damage. When a FET has been switched off by the protection mechanism, this error code will be returned by the respective task function. The measured values are therefore invalid and should not be used for further calculations.

For more details on the output/input protection mechanism refer to the following sections:

- [Digital Output Protection](#)
- [PVG Output Protection](#)
- [PWM Output Protection](#)
- [VOUT Output Protection](#)
- [ADC Input Protection](#)
- [PWD Input Protection](#)

Definition at line 279 of file IO_Error.h.

7.11.2.37 #define IO_E_FET_PROT_NOT_ACTIVE 24U

The internal switch (FET) or output is not protected, nothing can be reenabled.

Definition at line 298 of file IO_Error.h.

7.11.2.38 #define IO_E_FET_PROT_PERMANENT 21U

An internal switch (FET) or output has been disabled permanently to protect it from damage.
The output can not be reenabled anymore, as the maximum reset count was reached.

Definition at line 284 of file IO_Error.h.

7.11.2.39 #define IO_E_FET_PROT_REENABLE 22U

An internal switch (FET) or output has been disabled to protect it from damage.
The switch or output is ready to be reenabled via the protection reset function.

Definition at line 289 of file IO_Error.h.

7.11.2.40 #define IO_E_FET_PROT_WAIT 23U

The internal switch (FET) or output can not be reenabled, as the wait time is not passed already.

Definition at line 294 of file IO_Error.h.

7.11.2.41 #define IO_E_FLASH_BLANK_CHECK_FAILED 270U

Blank check during a flash write command failed.

This error is reported when during a write operation the flash range includes a byte which is not blank (i.e. its value is not 0xFF).

Definition at line 663 of file IO_Error.h.

7.11.2.42 #define IO_E_FLASH_OP_FAILED 271U

Flash operation failed.

This error is reported by the [IO_FLASH_GetStatus\(\)](#) function when the last flash operation failed (failure reported by the external flash chip).

Definition at line 669 of file IO_Error.h.

7.11.2.43 #define IO_E_FLASH_OP_TIMEOUT 272U

Flash operation resulted in timeout.

This error is reported by the [IO_FLASH_GetStatus\(\)](#) function when the last flash operation took more time than expected.

Definition at line 675 of file IO_Error.h.

7.11.2.44 #define IO_E_FLASH_SUSPENDED 273U

Flash module in suspended state.

This error is reported by `IO_FLASH_GetStatus()` when the `IO_FLASH_Suspend()` suspends the flash operation.

Definition at line 681 of file `IO_Error.h`.

7.11.2.45 #define IO_E_FLASH_WRONG_DEVICE_ID 269U

The flash chip did not return the expected device ID.

This error is reported if during the initialization of the flash driver the checking of the flash chip device ID did not succeed.

Note

This error is not expected after end-of-line testing.

Definition at line 657 of file `IO_Error.h`.

7.11.2.46 #define IO_E_FPGA_CRC_ERROR 202U

The FPGA reported a CRC error. Some ECU functions will not be available.

Has the correct FPGA image been flashed in the CPU flash memory?

Definition at line 531 of file `IO_Error.h`.

7.11.2.47 #define IO_E_FPGA_IMAGE 204U

Error in the FPGA image. Some ECU functions will not be available.

The FPGA image in the CPU flash memory is corrupted or its format is not supported.

Definition at line 541 of file `IO_Error.h`.

7.11.2.48 #define IO_E_FPGA_NOT_INITIALIZED 200U

The `IO_Driver_Init()` function has not been called, or reported one of the `IO_E_FPGA_...` errors.

Definition at line 523 of file `IO_Error.h`.

7.11.2.49 #define IO_E_FPGA_TIMEOUT 201U

The FPGA did not respond in time during its configuration. Some ECU functions will not be available.

Definition at line 526 of file `IO_Error.h`.

7.11.2.50 #define IO_E_FPGA_VERSION 203U

The FPGA version does not match with this I/O Driver version. Some ECU functions will not be available.

Has the correct FPGA image been flashed in the CPU flash memory?

Definition at line 536 of file IO_Error.h.

7.11.2.51 #define IO_E_INTERNAL_CSM 292U

An error has been detected in the internal COSMO low level drivers.

Definition at line 726 of file IO_Error.h.

7.11.2.52 #define IO_E_INTERNAL_MEM_FAILED 289U

An error has been detected in the internal memory.

Definition at line 718 of file IO_Error.h.

7.11.2.53 #define IO_E_INVALID_CHANNEL_ID 34U

The IO channel (IO pin) does not exist.

Two conditions can lead to this error code:

- When a non-existent channel ID has been passed to the function.
- When trying to initialize an IO which is not available on the ECU variant.

Definition at line 343 of file IO_Error.h.

7.11.2.54 #define IO_E_INVALID_ESM_INIT_STATUS 290U

An error has been detected in the start status of the ESM module.

Definition at line 722 of file IO_Error.h.

7.11.2.55 #define IO_E_INVALID_LIMITS 35U

An invalid limit configuration has been passed to the function.

This error is reported if the limits configuration structure of a digital input is wrong

Definition at line 348 of file IO_Error.h.

7.11.2.56 #define IO_E_INVALID_OPERATION 39U

The requested operation cannot be performed.

This error code is returned if a valid set of parameters is provided but the requested operation is not allowed due to some internal state or condition.

Example: Function `IO_PWM_ResolveOpenLoadShortCircuit()` has been called for a PWM channel that does not return `IO_E_OPEN_LOAD_OR_SHORT_BAT`.

Definition at line 377 of file IO_Error.h.

7.11.2.57 #define IO_E_INVALID_PARAMETER 31U

An invalid parameter has been passed to the function.

This error is reported if at least one of the parameters which have been passed to the function is outside the allowed range.

Definition at line 323 of file IO_Error.h.

7.11.2.58 #define IO_E_INVALID_PIN_CONFIG 284U

An error has been detected during the integrity check of the variant configuration data.

Definition at line 695 of file IO_Error.h.

7.11.2.59 #define IO_E_INVALID_PROD_DATA 283U

An error has been detected during the integrity check of the Production Data area.

Definition at line 691 of file IO_Error.h.

7.11.2.60 #define IO_E_INVALID_SAFETY_CONFIG 6U

The safety configuration for the channel to be configured is invalid

This error is reported if a parameter in the safety configuration structure used for configuring a IO channel is wrong. (see `safety_conf` parameter of the functions `IO_ADC_ChannelInit()`, `IO_PWM_Init()`, `IO_DO_Init()`, `IO_PWD_IncInit()`, `IO_PWD_CountInit()` and `IO_PWD_ComplexInit()`)

Definition at line 86 of file IO_Error.h.

7.11.2.61 #define IO_E_INVALID_SERIAL_NUMBER 285U

An error has been detected during the check of the Serial Number.

Definition at line 699 of file IO_Error.h.

7.11.2.62 #define IO_E_INVALID_VARIANT 282U

The detected ECU variant is not supported.

Definition at line 687 of file IO_Error.h.

7.11.2.63 #define IO_E_INVALID_VOLTAGE 14U

The measured voltage does not fit to a voltage band.

Digital inputs:

This error code is reported when the measured voltage is in between the defined voltage bands for high and low.

Definition at line 194 of file IO_Error.h.

7.11.2.64 #define IO_E_LIN_BIT 260U

A LIN bit error has been detected during the last transfer.

Definition at line 613 of file IO_Error.h.

7.11.2.65 #define IO_E_LIN_CHECKSUM 262U

A LIN checksum error has been detected during the last data reception.

Definition at line 621 of file IO_Error.h.

7.11.2.66 #define IO_E_LIN_FRAMING 265U

A LIN framing error has been detected during the last transfer

Definition at line 633 of file IO_Error.h.

7.11.2.67 #define IO_E_LIN_INCONSISTENT_SYNCH_FIELD 263U

A LIN inconsistent synch field error has been detected during the last header reception.

Definition at line 625 of file IO_Error.h.

7.11.2.68 #define IO_E_LIN_NO_RESPONSE 264U

A LIN no-response error has been detected during the last transfer

Definition at line 629 of file IO_Error.h.

7.11.2.69 #define IO_E_LIN_OVERRUN 266U

A LIN overrun error has been detected during the last transfer

Definition at line 637 of file IO_Error.h.

7.11.2.70 #define IO_E_LIN_PARITY 267U

A LIN parity error has been detected during the last header reception.

Definition at line 641 of file IO_Error.h.

7.11.2.71 #define IO_E_LIN_PHYSICAL_BUS 261U

A LIN physical bus error has been detected during the last transfer.

Definition at line 617 of file IO_Error.h.

7.11.2.72 #define IO_E_LIN_TIMEOUT 268U

A LIN driver timeout error has been detected during the last transfer.

Definition at line 645 of file IO_Error.h.

7.11.2.73 #define IO_E_MPU_REGION_DISABLED 331U

`IO_MPU_Disable()` was called on a region which is already disabled.

Definition at line 799 of file IO_Error.h.

7.11.2.74 #define IO_E_MPU_REGION_ENABLED 330U

`IO_MPU_Enable()` was called on a region which is already enabled.

Definition at line 795 of file IO_Error.h.

7.11.2.75 #define IO_E_NO_DIAG 15U

No output diagnostic is currently possible.

This error code is reported in several different cases:

PWM - Not enough samples:

The timer feedback has not captured enough samples yet in order to provide valid diagnostic information. Either the output was just enabled or just entered the valid diagnostic margin. The error condition is reset as soon as enough feedback samples have been captured.

PWM - Out of diagnostic range:

The PWM output was configured to not apply any diagnostic margin and the output was set to a duty cycle below the minimum on-time or above the minimum off-time. In this case the timer feedback of the output is not able to read a valid signal. The error condition is reset as soon as a duty cycle above the minimum on-time and below the minimum off-time is set and enough feedback samples have been captured.

DO - No feedback after level change:

After the level of a digital output was changed, no diagnostic feedback is available directly afterwards. The error condition is reset as soon as the output stays constantly at one level for at least 10 ms.

Definition at line 218 of file IO_Error.h.

7.11.2.76 #define IO_E_NULL_POINTER 30U

A NULL pointer has been passed to the function.

This error is reported if a non-optional pointer parameter of the function has been set to [NULL](#).

Definition at line 317 of file IO_Error.h.

7.11.2.77 #define IO_E_OK 0U

everything is fine, no error has occurred.

Definition at line 56 of file IO_Error.h.

7.11.2.78 #define IO_E_OPEN_LOAD 10U

A open load condition has been detected on an output.

Digital outputs:

This error code is reported when the output is set to low (power stage is switched off) and the analog feedback signal is between 1.5V and 5.5V.

The error condition is reset as soon as the feedback voltage is below 1.5V or the output is set to high.

Definition at line 110 of file IO_Error.h.

7.11.2.79 #define IO_E_OPEN_LOAD_OR_SHORT_BAT 13U

An open load or short circuit to battery voltage condition has been detected.

PWM output:

This error code is reported if the PWM output was configured to apply a diagnostic margin and the digital timer feedback of the output was not able to read a valid signal. The feedback signal was read as high.

The error condition is reset as soon as the timer feedback signal returns a valid signal again. In order to distinguish between open load and short to UBAT, refer to function [IO_PWM_ResolveOpenLoadShortCircuit](#)

PWM output in digital output mode:

This error code is reported when no load is connected and the output is off. In this case the level of the feedback channel is already high (pull up voltage) and can not be distinguished from a short to battery voltage.

The error condition is reset as soon as a load is connected. This I/O type cannot distinguish between open load and short to UBAT.

Definition at line 185 of file IO_Error.h.

7.11.2.80 #define IO_E_OPTION_NOT_SUPPORTED 340U

The option encoded in the variant configuration table is not supported.

Definition at line 806 of file IO_Error.h.

7.11.2.81 #define IO_E_PERIODIC_NOT_CONFIGURED 36U

The periodic interrupt timer has not been initialized.

This error code is reported if trying to disable the periodic interrupt timer although it has not been setup.

Definition at line 354 of file IO_Error.h.

7.11.2.82 #define IO_E_PWD_CURRENT_THRESH_HIGH 104U

A current out of range was detected (too high)

Definition at line 506 of file IO_Error.h.

7.11.2.83 #define IO_E_PWD_CURRENT_THRESH_LOW 105U

A current out of range was detected (too low)

Definition at line 509 of file IO_Error.h.

7.11.2.84 #define IO_E_PWD_NOT_FINISHED 101U

The required amount of signal edges couldn't be captured

Definition at line 500 of file IO_Error.h.

7.11.2.85 #define IO_E_PWD_OVERFLOW 102U

An overflow occurred during capturing

Definition at line 503 of file IO_Error.h.

7.11.2.86 #define IO_E_PWM_NO_DIAG IO_E_NO_DIAG

See [IO_E_NO_DIAG](#).

Definition at line 223 of file IO_Error.h.

7.11.2.87 #define IO_E_REFERENCE 18U

A reference voltage (sensor supply or internal reference) is out of range.

This error is reported for pins with analog feedback if:

- the sensor supply is out of range in ratiometric mode
- the 2.5 reference is invalid in absolute mode
- the reference voltages are unplausible in current mode
- the reference is out of range in resistive mode

Definition at line 254 of file IO_Error.h.

7.11.2.88 #define IO_E_RESOLVING 25U

The resolving operation of a formerly detected open load or short circuit to battery voltage condition is still ongoing. The output is disabled by software and cannot be reenabled.

Definition at line 303 of file IO_Error.h.

7.11.2.89 #define IO_E_RESOLVING_FAILED 26U

The resolving operation of a formerly detected open load or short circuit to battery voltage condition failed. No detailed information about the actual reason is available. The output is disabled by software and cannot be reenabled.

Definition at line 309 of file IO_Error.h.

7.11.2.90 #define IO_E_RTC_CLOCK_INTEGRITY_FAILED 250U

The integrity check of the Real Time Clock (responsible for date and time) failed.

Date and Time is lost. This can happen if a power loss on the RTC exceeds the buffer time (approximately 10 minutes). To make the Real Time Clock counting again, it must be re-set by calling [IO_RTC_SetDateAndTime\(\)](#) with the appropriate date/time information.

Definition at line 606 of file IO_Error.h.

7.11.2.91 #define IO_E_SAFE_STATE 17U

The pin is in a safe state.

This error is reported if a pin or an entire shut off group was disabled by the application callback or the global safe state was entered.

Definition at line 245 of file IO_Error.h.

7.11.2.92 #define IO_E_SAFETY_SWITCH_DISABLED 19U

A safety switch is disabled.

This error is reported for pins configured as outputs in the shut off groups [IO_INT_SAFETY_SW_0](#), [IO_INT_SAFETY_SW_1](#) and [IO_INT_SAFETY_SW_2](#) when its corresponding safety switch is disabled.

Definition at line 260 of file IO_Error.h.

7.11.2.93 #define IO_E_SHM_INTEGRITY 320U

An error as been detected when verifying the shared memory's integrity.

Definition at line 788 of file IO_Error.h.

7.11.2.94 #define IO_E_SHORT_BAT 12U

A short circuit to battery voltage condition has been detected on an output.

Digital outputs high side:

This error code is reported when the output is set to low (power stage is switched off) and the analog feedback signal is above 5.5V.

The error condition is reset as soon as the feedback voltage is below 1.5V, or if the output is set to high.

Digital outputs low side:

This error code is reported when the output is set to high (power stage is switched on) and the analog feedback signal is out of range.

The error condition is reset as soon as the output is set to low.

Digital inputs:

This error code is reported when the measured analog input voltage if above the defined high band.

The error condition is reset as soon as the voltage is inside a defined voltage band.

Definition at line 165 of file IO_Error.h.

7.11.2.95 #define IO_E_SHORT_GND 11U

A short circuit to ground condition has been detected on an output.

PWM output:

This error code is reported if the PWM output was configured to apply a diagnostic margin and the digital timer feedback of the output was not able to read a valid signal. The feedback signal was read as low.

The error condition is reset as soon as the timer feedback signal returns a valid signal again.

PWM output in digital output mode:

This error code is reported when the output is set to high (power stage is switched on) and the digital feedback signal is low.

The error condition is reset as soon as the feedback signal is high, or if the output is set to low.

Digital outputs high side:

This error code is reported when the output is set to high (power stage is switched on) and the analog feedback signal is below 1.5V.

The error condition is reset as soon as the feedback voltage is above 1.5V, or if the output is set to low.

Digital outputs low side:

This error code is reported when the output is set to low (power stage is switched off) and the analog feedback signal is below 3V.

The error condition is reset as soon as the feedback voltage is above 3V, or if the output is set to high.

Digital inputs:

This error code is reported when the measured analog input voltage is below the defined low band.

The error condition is reset as soon as the voltage is inside a defined voltage band.

Definition at line 144 of file IO_Error.h.

7.11.2.96 #define IO_E_SOCKET_NOT_INITIALIZED 444U

socket not initialized

Definition at line 827 of file IO_Error.h.

7.11.2.97 #define IO_E_STARTUP 16U

The pin is in the startup phase.

This error is reported during the startup phase while several startup checks on the pins are made. During this phase, the pin can not be controlled by the application. The maximum duration of this startup phase depends on the I/O module:

- ADC module: max 50ms from the point in time the diagnostic state machine has reached the main state
- DIO module: max 30ms from the point in time the diagnostic state machine has reached the main state, max 20ms after changing the state of the digital output
- PVG module: max 110ms from the point in time the output has been initialized
- PWD module: max 120ms from the point in time the input has been initialized
- PWM module: max 5ms from the point in time the diagnostic state machine has reached the main state, max 5ms after re-enabling the corresponding safety switch

- VOUT module: max 110ms from the point in time the output has been initialized
- Definition at line 239 of file IO_Error.h.

7.11.2.98 #define IO_E_UART_BUFFER_EMPTY 71U

The UART software buffer is empty.
This error code is not used.

Definition at line 474 of file IO_Error.h.

7.11.2.99 #define IO_E_UART_BUFFER_FULL 70U

The UART software buffer is full.

- When receiving: Too much data has been received since the last successful read operation.
Data has been lost.
- When transmitting: The given data does not fit into the buffer, data has been rejected. Try again when there is more space in the buffer.

The UART module has two buffers, one for transmitting and one for receiving, both with a size of [IO_UART_BUFFER_LEN](#).

Definition at line 469 of file IO_Error.h.

7.11.2.100 #define IO_E_UART_DMA 75U

Error in the UART DMA data processing.
An error occurred in the UART DMA. Data has been lost.
Definition at line 494 of file IO_Error.h.

7.11.2.101 #define IO_E_UART_FRAMING 74U

UART framing error.
An expected stop bit hasn't been found.
Definition at line 489 of file IO_Error.h.

7.11.2.102 #define IO_E_UART_OVERFLOW 72U

UART hardware reception buffer overrun.
The UART receiver has not been serviced by the DMA in time. Data has been lost.
Definition at line 479 of file IO_Error.h.

7.11.2.103 #define IO_E_UART_PARITY 73U

UART parity error.

The received parity bit doesn't match the calculated one.

Definition at line 484 of file IO_Error.h.

7.11.2.104 #define IO_E_UDP_ARP RECEIVED 449U

ARP package received

Definition at line 848 of file IO_Error.h.

7.11.2.105 #define IO_E_UDP_INVALID_BUFFER 446U

invalid buffer has been passed

Definition at line 837 of file IO_Error.h.

7.11.2.106 #define IO_E_UDP_NOMORESOCKETS 440U

no more UDP sockets are available

Definition at line 812 of file IO_Error.h.

7.11.2.107 #define IO_E_UDP_OVERFLOW 442U

overflow of message object

Definition at line 817 of file IO_Error.h.

7.11.2.108 #define IO_E_UDP_WRONG_HANDLE 443U

data overflow on receive, tx data overflow on send

Definition at line 822 of file IO_Error.h.

7.11.2.109 #define IO_E_UDP_WRONG_PORT 447U

invalid port; only relevant for SafeRTOS

Definition at line 842 of file IO_Error.h.

7.11.2.110 #define IO_E_UNKNOWN 3U

General error. No further information can be provided.

Definition at line 66 of file IO_Error.h.

7.11.2.111 #define IO_E_WD_ACTIVATION 243U

One of the two available watchdog activation functions reported an error.
This error code is used internally. It is not reported by any API function.

Definition at line 565 of file IO_Error.h.

7.11.2.112 #define IO_E_WD_DEBUGGING_PREPARED 246U

The debug key has been set to 0xC0FFEE in the corresponding field of the APDB and the very first life cycle of the ECU has been detected. In this state, the external watchdog device is kept in a frozen state to allow a reset and connection with the debugger. In this state no watchdog activation takes place.

Definition at line 583 of file IO_Error.h.

7.11.2.113 #define IO_E_WD_INITIALIZATION 241U

One of the two available watchdog initialization functions of the watchdog system reported an error.
Definition at line 553 of file IO_Error.h.

7.11.2.114 #define IO_E_WD_PRECISION 242U

The given configuration parameters cannot be configured due to precision limitations of the internal watchdog module's prescaler. Either the task cycle time is too short or the window size is too narrow for the configured task cycle time.

Either the window size or the task cycle time must be increased.

Definition at line 560 of file IO_Error.h.

7.11.2.115 #define IO_E_WD_RANGE 240U

The given configuration parameters for configuring the watchdogs are out of range.

Definition at line 548 of file IO_Error.h.

7.11.2.116 #define IO_E_WD_SAFE_LOCK 245U

The watchdog's device error counter reached its maximum value. Therefore the external watchdog stays locked in safe state. No reset takes place. No more activation is possible until an externally asserted power cycle of the ECU.

Definition at line 576 of file IO_Error.h.

7.11.2.117 #define IO_E_WD_SELF_MONITORING 247U

A watchdog task detected a timing error in it's own calling interval.
This can be either a too early or a too late call.

Definition at line 588 of file IO_Error.h.

7.11.2.118 #define IO_E_WD_STATUS_INVALID 249U

The watchdog's status information could not be retrieved.

Definition at line 597 of file IO_Error.h.

7.11.2.119 #define IO_E_WD_TRIGGER 244U

One of the two available watchdog trigger functions reported an error.
This error code is used internally. It is not reported by any API function.

Definition at line 570 of file IO_Error.h.

7.11.2.120 #define IO_E_WD_VICE_VERSA_MONITORING 248U

A watchdog task detected a timing error of the other watchdog.
The other watchdog has not been triggered until the latest possible point in time.
Definition at line 593 of file IO_Error.h.

7.11.2.121 #define IO_E_WRONG_ADDRESS 445U

IP address or port do not fit

Definition at line 832 of file IO_Error.h.

7.11.3 Typedef Documentation**7.11.3.1 typedef ubyte2 IO_ErrorType**

IO Driver error type.

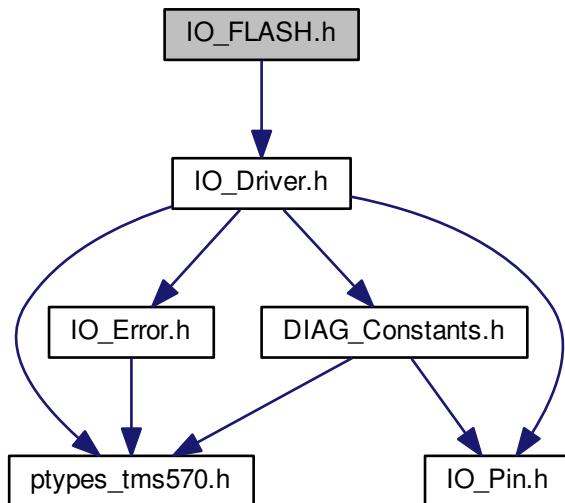
<Type definition for all the IO Driver errors.

Definition at line 41 of file IO_Error.h.

7.12 IO_FLASH.h File Reference

IO Driver functions for handling the external flash.

Include dependency graph for IO_FLASH.h:



Macros

- #define `IO_FLASH_BYTE_SIZE` 0x800000U
- #define `IO_FLASH_SetBusy` `IO_FLASH_Suspend`

The Macro maintains backwards compatibility for applications using `IO_FLASH_SetBusy()` from I/O Driver 3.1.

Functions

- `IO_ErrorType IO_FLASH_BankSelect (ubyte1 Bank)`
Sets the active flash bank.
- `IO_ErrorType IO_FLASH_BlockErase (ubyte4 offset)`
Erases one block in the external flash.
- `IO_ErrorType IO_FLASH_ChipErase (void)`
Erases the whole external flash chip.
- `IO_ErrorType IO_FLASH_DeInit (void)`
Deinitializes the flash driver.
- `IO_ErrorType IO_FLASH_GetBank (ubyte1 *Bank)`
Gets the active flash bank.
- `IO_ErrorType IO_FLASH_GetStatus (void)`
Returns the status of the last flash operation.
- `IO_ErrorType IO_FLASH_Init (void)`
Initializes the flash driver.

- `IO_ErrorType IO_FLASH_Read (ubyte4 offset, ubyte4 length, ubyte1 *const data)`
Reads data from the external flash.
- `IO_ErrorType IO_FLASH_Suspend (bool suspend)`
Suspends (or Resumes) the API access to the external flash.
- `IO_ErrorType IO_FLASH_Write (ubyte4 offset, ubyte4 length, const ubyte1 *const data)`
Writes data to the external flash.

7.12.1 Detailed Description

IO Driver functions for handling the external flash.

The flash driver allows reading from, writing to and erasing the external flash chip.

7.12.2 Blocks in the Flash Chip

The flash chip can be written to and read from any flash offset, but it can only be erased in blocks.

The block dimensions are fixed and given by the flash chip. They can vary between Flash chips or even within the same flash chip.

7.12.2.1 Product Version 1.03

The blocks have uneven sizes depending on the offset (internal address in the flash memory):

- Offsets from 0x000000 to 0x00FFFF (the first 8 blocks): Block size is 0x2000 (8 KiB).
- Offsets from 0x010000 to 0x7F0000 (remaining blocks): Block size is 0x10000 (64 KiB).

The whole memory is partitioned as follows:

Beginning Offset	End Offset	Block Size
0x000000	0x001FFF	0x2000
0x002000	0x003FFF	0x2000
...
0x00E000	0x00FFFF	0x2000
0x010000	0x01FFFF	0x10000
0x020000	0x02FFFF	0x10000
...
0x7F0000	0x7FFFFFFF	0x10000

This product version does not support multiple flash banks.

7.12.2.2 Product Version 1.08

The block size is uniform for the whole memory: 0x20000 (128 KiB).

7.12.3 Flash Banks

Product variants with flash memory larger than `IO_FLASH_BYTE_SIZE` have the flash memory divided into banks. Flash operations than depend on `offset` are limited to the selected flash bank. The `offset` parameter is therefore the offset within the selected flash bank. The internal address in the flash memory is calculated as `address = bank * IO_FLASH_BYTE_SIZE + offset`.

The selected flash bank also determines the part of the flash memory that is visible directly in the CPU address space (starting at address 0x64000000).

This is also the reason that the MPU protection, if enabled, acts on the currently selected flash bank and is not synchronized with the bank switching. If such synchronization is required (i.e. if different flash banks use different MPU protection), it has to be done manually by enabling and disabling the User MPU region together with switching the flash bank.

Product variants with flash memory size equal to `IO_FLASH_BYTE_SIZE` do not require bank switching. All the bank-dependent functions will act as if bank 0 was selected.

7.12.4 Speed of Flash Operations

The communication with the flash chip is handled in a cyclic manner in the background, because flash operations take multiple cycles. Exactly one cycle is executed in every millisecond. Every user operation is triggered by calling the respective function. A status function provides information about the result of the last operation.

7.12.4.1 Speed of a read operation

In every cycle at most 256 bytes will be copied from the external flash to the internal memory. The number of cycles required for the whole read operation to complete is `ceil (length / 256)`.

It means the read operation will take more than one cycle to be completed if `length` is greater than 256, and is most effective when `length` is a multiple of 256 bytes.

7.12.4.2 Speed of a write operation

A write operation is handled in two parts: first it ensures the destination area is blank, and then copies the data to the flash area. Blank check is done 256 bytes per cycle, so it takes `ceil (length / 256)` cycles to check `length` bytes of data.

In every cycle at most 32 bytes will be copied from the internal memory to the external flash. If the area in the flash crosses a 32-byte boundary, the operation takes one cycle more. Thus the number of cycles required for the whole write operation to complete is maximum `ceil (length / 256) + ceil (length / 32) + 2`.

It means a write operation will always take more than one cycle to be completed. Writing to flash is most efficient when `length` is a multiple of 256 bytes, and when the range does not cross a 32-byte boundary, for example:

- for `length = 1` it will take 3 cycles,

- for length = 2, offset = 31 it will take 4 cycles (range crosses a 32-byte boundary),
- for length = 32, offset = 0 it will take 3 cycles,
- for length = 256, offset = 1024 it will take 10 cycles.
- for length = 256, offset = 1025 it will take 11 cycles (range crosses a 32-byte boundary).
- for length = 257, offset = 1024 it will take 12 cycles (range crosses a 32-byte boundary and length > 256).

7.12.5 Flash Code Examples

Examples for using the flash API

7.12.5.1 Example for flash initialization

The external flash can be initialized only after `IO_Driver_Init()`. The function `IO_FLASH_Init()` needs to be called before any other flash API function.

```
IO_ErrorType local_ret;

// Initialize HY-TTC 500 driver
local_ret = IO_Driver_Init(NULL);

if (local_ret == IO_E_OK)
{
    // Initialize Flash driver
    local_ret = IO_FLASH_Init();
}

if (local_ret == IO_E_OK)
{
    // Flash driver is initialized successfully, act accordingly
}
else
{
    // HY-TTC 500 driver or flash driver was not initialized successfully,
    // act accordingly
}
```

7.12.5.2 Example for checking the status of the driver

After starting a read, write or erase operation, the flash driver is busy until that operation completes. The status of the driver can be checked by calling periodically the `IO_FLASH_GetStatus()` function. If the driver is already initialized, it returns if the driver is busy (i.e. the last operation is being performed) or idle. If the driver is idle (i.e. the last operation has ended), the return value indicates whether the last operation succeeded, or if not, what the error was.

Non-blocking flash application example:

```
IO_RTC_StartTime(&timestamp);

while (1)
{
```

```

// task begins
IO_Driver_TaskBegin();

// the IO_FLASH_Read() function is called only on the first cycle
if (CycleCounter == 0)
{
    io_error = IO_FLASH_Read(offset, length, dataRead);
    printf("IO_FLASH_Read() error code: %d\r\n", io_error);
    if (IO_E_OK != io_error)
    {
        ProcessEnd = TRUE;
    }
}

io_error = IO_FLASH_GetStatus();

if (io_error == IO_E_BUSY)
{
    // do nothing, chip is busy
}
else if (io_error == IO_E_OK)
{
    ProcessEnd = TRUE;
    printf("read finished successfully\r\n");
}
else
{
    ProcessEnd = TRUE;
    printf("read finished with error code: %d\r\n", io_error);
}

// task ends
IO_Driver_TaskEnd();

CycleCounter += 1;

while (IO_RTC_GetTimeUS(timestamp) < 10000 );    // wait until 10ms have passed
timestamp += 10000;                                // increase time stamp by cycle time
}

```

Blocking flash application example:

```

// starting chip erase
io_error = IO_FLASH_BlockErase(blockOffset_u4);

if (io_error != IO_E_OK)
{
    / an error occurred when trying to erase the block
    printf("IO_FLASH_BlockErase returned %d\r\n", io_error);
}

// wait for flash driver to become ready again
do
{
    io_error = IO_FLASH_GetStatus();
} while (io_error == IO_E_BUSY);

// check last status value
if (io_error == IO_E_OK)
{
    // block was erased successfully
}

```

```

    printf("Block was erased successfully\r\n");
}
else
{
    // an error occurred when trying to erase the block
    printf("Block erase finished with error code: %d\r\n", io_error);
}

```

7.12.6 Macro Definition Documentation

7.12.6.1 #define IO_FLASH_BYTE_SIZE 0x800000U

Defines the size of the flash bank.

(8 MiB = 8,388,608 bytes)

Definition at line 287 of file IO_FLASH.h.

7.12.6.2 #define IO_FLASH_SetBusy IO_FLASH_Suspend

The Macro maintains backwards compatibility for applications using `IO_FLASH_SetBusy()` from I/O Driver 3.1.

Definition at line 294 of file IO_FLASH.h.

7.12.7 Function Documentation

7.12.7.1 IO_ErrorType IO_FLASH_BankSelect (ubyte1 Bank)

Sets the active flash bank.

The Flash memory is broken into banks of size `IO_FLASH_BYTE_SIZE`. This function sets the active flash bank.

See [Flash Banks](#) for details.

Parameters

<code>Bank</code>	Requested flash bank
-------------------	----------------------

Returns

`IO_ErrorType`

Return values

<code>IO_E_BUSY</code>	Driver is busy, last operation is still ongoing
<code>IO_E_OK</code>	Driver is idle, last operation finished successfully
<code>IO_E_INVALID_PARAMETER</code>	Invalid flash bank requested
<code>IO_E_UNKNOWN</code>	Driver is idle, during the last operation an internal or unknown error occurred

7.12.7.2 IO_ErrorType IO_FLASH_BlockErase (ubyte4 offset)

Erases one block in the external flash.

The flash memory block which begins at the offset specified by the `offset` parameter in currently selected flash bank will be erased. The function only triggers a block erase operation, it doesn't happen instantly.

The `offset` parameter must point exactly to the beginning of a block. For a description of the size of blocks please see the [Blocks in the Flash Chip](#) section.

After calling `IO_FLASH_BlockErase()`, the `IO_FLASH_GetStatus()` function should be called periodically to check if the block erase operation has finished.

If a block erase operation is interrupted by a power-down or a call to `IO_FLASH_DeInit()`, the result is undefined.

Parameters

<code>offset</code>	Flash memory offset (beginning of block)
---------------------	--

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	Block erase operation started fine
<code>IO_E_BUSY</code>	Last operation of flash module is still in progress
<code>IO_E_INVALID_PARAMETER</code>	Invalid block offset
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	The flash module has not been initialized
<code>IO_E_FLASH_SUSPENDED</code>	The Flash is in a suspended state

Note

- Constant data that has been linked to the external flash must not be erased. Otherwise the application CRC will be invalidated and thus the bootloader will not start the application anymore.

7.12.7.3 IO_ErrorType IO_FLASH_ChipErase (void)

Erases the whole external flash chip.

The whole external flash chip will be erased, regardless of the selected flash bank. This function only triggers a chip erase operation, it doesn't happen instantly.

After calling `IO_FLASH_ChipErase()`, the `IO_FLASH_GetStatus()` function should be called periodically to check if the chip erase operation has finished.

If a chip erase operation is interrupted by a power-down or a call to `IO_FLASH_DeInit()`, the result is undefined.

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	Chip erase operation started fine
<code>IO_E_BUSY</code>	Last operation of flash module is still in progress
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	The flash module has not been initialized
<code>IO_E_FLASH_SUSPENDED</code>	The Flash is in a suspended state

Note

- Constant data that has been linked to the external flash must not be erased. Otherwise the application CRC will be invalidated and thus the bootloader will not start the application anymore.

7.12.7.4 `IO_ErrorType IO_FLASH_DelInit (void)`

Deinitializes the flash driver.

This puts the external flash chip into the hardware reset state.

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the module is not initialized

7.12.7.5 `IO_ErrorType IO_FLASH_GetBank (ubyte1 * Bank)`

Gets the active flash bank.

The flash memory is broken into banks of size `IO_FLASH_BYTE_SIZE`. This function returns the active flash bank.

See [Flash Banks](#) for details.

Parameters

<code>Bank</code>	Currently active flash bank
-------------------	-----------------------------

Returns

`IO_ErrorType`

Return values

<i>IO_E_BUSY</i>	Driver is busy, last operation is still ongoing
<i>IO_E_OK</i>	Driver is idle, last operation finished successfully
<i>IO_E_INVALID_PARAMETER</i>	Invalid flash bank requested
<i>IO_E_UNKNOWN</i>	Driver is idle, during the last operation an internal or unknown error occurred

7.12.7.6 IO_ErrorType IO_FLASH_GetStatus (void)

Returns the status of the last flash operation.

This function returns the result of the last operation accepted by the flash driver.

The driver is idle if the return value is not equal to *IO_E_BUSY*. (If the last operation did not finish successfully, the flash chip and the internal states are reset automatically.)

For code examples see the section [Example for checking the status of the driver](#).

Returns

IO_ErrorType

Return values

<i>IO_E_BUSY</i>	Driver is busy, last operation is still ongoing
<i>IO_E_OK</i>	Driver is idle, last operation finished successfully
<i>IO_E_FLASH_OP_TIMEOUT</i>	Driver is idle, the last operation exceeded the allowed time
<i>IO_E_FLASH_OP_FAILED</i>	Driver is idle, the last operation has failed
<i>IO_E_FLASH_BLANK_CHECK_FAILED</i>	Driver is idle, during the last write operation a non-blank byte was found
<i>IO_E_UNKNOWN</i>	Driver is idle, during the last operation an internal or unknown error occurred
<i>IO_E_CHANNEL_NOT_CONFIGURED</i>	The flash module has not been initialized
<i>IO_E_FLASH_SUSPENDED</i>	The Flash is in a suspended state

7.12.7.7 IO_ErrorType IO_FLASH_Init (void)

Initializes the flash driver.

For code examples see the section [Example for flash initialization](#).

Returns

IO_ErrorType

Return values

<i>IO_E_OK</i>	Everything fine
<i>IO_E_FLASH_WRONG_DEVICE_ID</i>	The flash chip did not return the expected device ID
<i>IO_E_INVALID_CHANNEL_ID</i>	The flash module is not available on the used ECU variant
<i>IO_E_CHANNEL_BUSY</i>	Module has been initialized before
<i>IO_E_DRIVER_NOT_INITIALIZED</i>	The common driver init function has not been called before
<i>IO_E_OPTION_NOT_SUPPORTED</i>	The type of the flash chip is not supported

<code>IO_E_INTERNAL_CSM</code>	Internal error
<code>IO_E_BUSY</code>	Driver is busy, last operation is still ongoing
<code>IO_E_FLASH_OP_FAILED</code>	Driver is idle, the last operation has failed

Remarks

- If the flash driver needs to be re-initialized, the function `IO_FLASH_DeInit()` has to be called first.

7.12.7.8 IO_ErrorType IO_FLASH_Read (`ubyte4 offset`, `ubyte4 length`, `ubyte1 *const data`)

Reads data from the external flash.

Data is read from the offset specified by the `offset` parameter in currently selected flash bank and written to the memory location pointed to by the `data` parameter. The number of bytes to be read is denoted by the `length` parameter. The function only triggers a read operation, data is never returned instantly.

After calling `IO_FLASH_Read()`, the `IO_FLASH_GetStatus()` function should be called periodically to check if the read operation has finished.

Parameters

<code>offset</code>	Flash memory offset (0 .. (<code>IO_FLASH_BYTE_SIZE</code> - 1))
<code>length</code>	Length of data to be read, in bytes (1 .. <code>IO_FLASH_BYTE_SIZE</code>)
<code>out</code>	Pointer to data in the internal memory.

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	Read operation started fine
<code>IO_E_BUSY</code>	Last operation of flash module is still in progress
<code>IO_E_INVALID_PARAMETER</code>	Invalid address offset or length
<code>IO_E_NULL_POINTER</code>	Parameter <code>data</code> is a NULL pointer
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	The flash module has not been initialized
<code>IO_E_FLASH_SUSPENDED</code>	The Flash is in a suspended state

Note

- (offset + length) must not be greater than `IO_FLASH_BYTE_SIZE`.
- The buffer passed via parameter `data` must be valid throughout the whole read procedure (e.g. not located on the stack).
- A read operation to external flash must not be requested while the external flash chip is busy performing a write, block erase or chip erase operation.

7.12.7.9 IO_ErrorType `IO_FLASH_Suspend (bool suspend)`

Suspends (or Resumes) the API access to the external flash.

The external flash memory can be accessed using the API functions in this module (read, write and erase access) but it is also mapped directly to the CPU address space (starting at address 0x64000000, only the active bank), which allows direct read access. To get correct and consistent data, the direct read access should not be used together with the API functions (specifically not with erasing and writing). The function `IO_FLASH_Suspend()` allows internal suspending of the API functions to ensure that they cannot interfere with the direct read access.

Another feature of this function is to suspend an ongoing Block Erase. If a Block Erase is in progress when the function `IO_FLASH_Suspend()` is called the erase operation will be suspended until the flash module is resumed. While in a suspended block erase state any data can be read from the external flash except the block which is being erased.

Note

The following operations can not be executed while the flash is in a suspended state: `IO_FLASH_BlockErase()`, `IO_FLASH_ChipErase()`, `IO_FLASH_Read()`, and `IO_FLASH_Write()`. Switching flash banks is not suspended.

If the application changes the flash bank while in a suspended state, upon resuming future operations will take place on the new flash bank.

If a Block Erase was suspended with `IO_FLASH_Suspend()` and during the suspend the flash bank was changed, upon resuming the Block Erase will complete successfully on the originally intended Block (regardless of flash bank). After this future operations will take place on the new flash bank.

Parameters

<code>suspend</code>	<code>TRUE</code> = flash API use suspended, <code>FALSE</code> = flash API use resumed
----------------------	---

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	Operation successful
<code>IO_E_BUSY</code>	Last operation of flash module is still in progress
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	The flash module has not been initialized
<code>IO_E_FLASH_SUSPENDED</code>	The Flash is already in a suspended state

7.12.7.10 IO_ErrorType IO_FLASH_Write (ubyte4 offset, ubyte4 length, const ubyte1 *const data)

Writes data to the external flash.

Data is written from the memory location pointed to by the `data` parameter to the flash offset specified by the `offset` parameter in currently selected flash bank. The number of bytes to be written is denoted by the `length` parameter. The function only triggers a write operation, data is not written instantly.

Before the write operation begins, the destination flash memory range is checked to be in erased state. The write operation continues only if all bytes are found to be blank, i.e. their value is 0xFF.

After calling `IO_FLASH_Write()`, the `IO_FLASH_GetStatus()` function should be called periodically to check if the write operation has finished.

Parameters

<code>offset</code>	Flash memory offset (0 .. (<code>IO_FLASH_BYTE_SIZE</code> - 1))
<code>length</code>	Length of data to be written, in bytes (1 .. <code>IO_FLASH_BYTE_SIZE</code>)
<code>in</code>	Pointer to data in the internal memory.

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	Write operation started fine
<code>IO_E_BUSY</code>	Last operation of flash module is still in progress
<code>IO_E_INVALID_PARAMETER</code>	Invalid address offset or length
<code>IO_E_NULL_POINTER</code>	Parameter <code>data</code> is a NULL pointer
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	The flash module has not been initialized
<code>IO_E_FLASH_SUSPENDED</code>	The Flash is in a suspended state

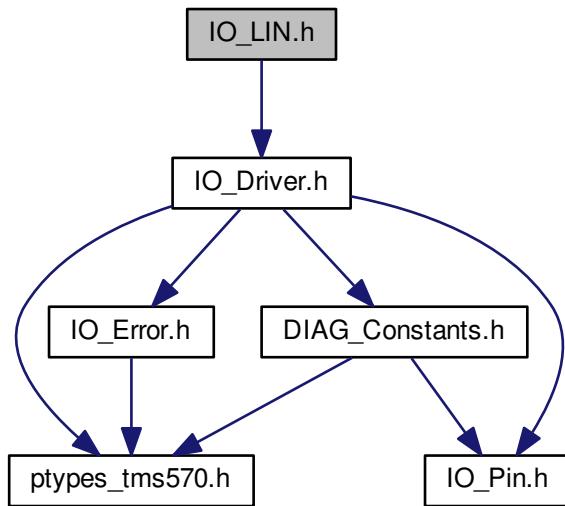
Note

- `(offset + length)` must not be greater than `IO_FLASH_BYTE_SIZE`.
- The buffer passed via parameter `data` must be valid throughout the whole write procedure (e.g. not located on the stack).
- Constant data that has been linked to the external flash must not be erased. Otherwise the application CRC will be invalidated and thus the bootloader will not start the application anymore.

7.13 IO_LIN.h File Reference

IO Driver functions for LIN communication.

Include dependency graph for IO_LIN.h:



Data Structures

- struct `io_lin_data_frame_`
LIN data frame.

Typedefs

- typedef struct `io_lin_data_frame_` `IO_LIN_DATA_FRAME`
LIN data frame.

Functions

- `IO_ErrorType IO_LIN_DelInit (void)`
Deinitializes the LIN communication driver.
- `IO_ErrorType IO_LIN_GetStatus (void)`
Returns the status of the LIN channel.
- `IO_ErrorType IO_LIN_Init (ubyte2 baudrate, ubyte1 checksum_type)`
Initialization of the LIN communication driver.
- `IO_ErrorType IO_LIN_Read (IO_LIN_DATA_FRAME *const frame)`
Reads a LIN frame with the given id and of the given length.
- `IO_ErrorType IO_LIN_Write (const IO_LIN_DATA_FRAME *const frame)`
Transmit a LIN frame with the given id and of the given length.

Checksum type

Defines the LIN checksum type

- #define IO_LIN_CLASSIC 0U
- #define IO_LIN_ENHANCED 1U

Baudrate configuration

Defines the minimum and maximum possible baudrate

- #define IO_LIN_BAUDRATE_MIN 1000U
- #define IO_LIN_BAUDRATE_MAX 20000U

7.13.1 Detailed Description

IO Driver functions for LIN communication.

The LIN driver uses the LIN module of the TMS570 CPU.

The interface is compliant to LIN 2.1 and supports baudrates up to 20 Kbit/s. As the LIN is only specified for 12V systems, it operates always at 12V, also in a 24V system.

The LIN module automatically generates the header when initiating a read or write transfer. The LIN supports LIN 2.0 checksum. The LIN also detects short circuits on the bus.

LIN-API Usage:

- [Examples for LIN API functions](#)

7.13.2 LIN Code Examples

Examples for using the LIN API

7.13.2.1 Example for LIN initialization

```
// init LIN master
IO_LIN_Init(20000,           //20 kB/s
            IO_LIN_ENHANCED); //enhanced checksum
```

7.13.2.2 Example for LIN task function call

```
IO_LIN_DATA_FRAME lin_frame_tx;
IO_LIN_DATA_FRAME lin_frame_rx;

// wait for LIN Bus to be ready
while (IO_LIN_GetStatus() == IO_E_BUSY);

// set id
lin_frame_tx.id = 0x23; // id 0x23

// assemble data
lin_frame_tx.data[0] = 0x12;
```

```

lin_frame_tx.data[1] = 0x25;
lin_frame_tx.data[2] = 0xAC;
lin_frame_tx.data[3] = 0xFE;

// set length of data
lin_frame_tx.length = 4; // 4 bytes

// write LIN frame
IO_LIN_Write(&lin_frame_tx);

// wait for LIN Bus to be ready
while (IO_LIN_GetStatus() == IO_E_BUSY);

// set id
lin_frame_rx.id = 0x23; // id 0x23

// set length of data
lin_frame_rx.length = 4; // 4 bytes

// read a LIN frame
IO_LIN_Read(&lin_frame_rx);

// wait for LIN Read to be complete
while (IO_LIN_GetStatus() == IO_E_BUSY);

// check if read was successful
if (IO_LIN_GetStatus() == IO_E_OK)
{
    // process received data
}
else
{
    // error
}

```

7.13.3 Macro Definition Documentation

7.13.3.1 #define IO_LIN_BAUDRATE_MAX 20000U

Maximum LIN baudrate

Definition at line 126 of file IO_LIN.h.

7.13.3.2 #define IO_LIN_BAUDRATE_MIN 1000U

Minimum LIN baudrate

Definition at line 125 of file IO_LIN.h.

7.13.3.3 #define IO_LIN_CLASSIC 0U

Classic checksum

Definition at line 114 of file IO_LIN.h.

7.13.3.4 #define IO_LIN_ENHANCED 1U

Enhanced LIN 2.0 checksum

Definition at line 115 of file IO_LIN.h.

7.13.4 Typedef Documentation

7.13.4.1 typedef struct io_lin_data_frame_ IO_LIN_DATA_FRAME

LIN data frame.

Stores a data frame for the LIN communication.

7.13.5 Function Documentation

7.13.5.1 IO_ErrorType IO_LIN_Delnit (void)

Deinitializes the LIN communication driver.

Returns

IO_ErrorType

Return values

<i>IO_E_OK</i>	everything fine
<i>IO_E_CHANNEL_NOT_CONFIGURED</i>	the channel was not configured
<i>IO_E_UNKNOWN</i>	an unknown error occurred

7.13.5.2 IO_ErrorType IO_LIN_GetStatus (void)

Returns the status of the LIN channel.

Returns

IO_ErrorType

Return values

<i>IO_E_OK</i>	everything fine
<i>IO_E_BUSY</i>	the channel is busy
<i>IO_E_LIN_TIMEOUT</i>	LIN timeout detected, if frame duration exceeded the maximum frame time
<i>IO_E_LIN_PARITY</i>	parity error detected during receiving
<i>IO_E_LIN_OVERRUN</i>	unread data was overwritten
<i>IO_E_LIN_FRAMING</i>	an expected stop bit is not found
<i>IO_E_LIN_NO_RESPONSE</i>	no response from the slave
<i>IO_E_LIN_INCONSISTENT_SYNCH_FIELD</i>	inconsistent synch field error was detected
<i>IO_E_LIN_CHECKSUM</i>	a checksum error occurred
<i>IO_E_LIN_PHYSICAL_BUS</i>	a physical bus error was detected

IO_E_LIN_BIT	a bit error was detected
IO_E_CHANNEL_NOT_CONFIGURED	the channel was not configured
IO_E_UNKNOWN	an unknown error occurred

7.13.5.3 IO_ErrorType IO_LIN_Init (**ubyte2 baudrate, ubyte1 checksum_type**)

Initialization of the LIN communication driver.

The function

- Enables the module and sets the LIN to master mode
- Automatically sets up the bit timing for the given baudrate

Parameters

<i>baudrate</i>	Baud rate in bit/s (1000..20000)
<i>checksum_type</i>	Checksum type, one of: <ul style="list-style-type: none"> • IO_LIN_CLASSIC, • IO_LIN_ENHANCED

Returns

[IO_ErrorType](#)

Return values

IO_E_OK	everything fine
IO_E_INVALID_CHANNEL_ID	LIN is not supported on the used ECU variant
IO_E_INVALID_PARAMETER	invalid parameter has been passed
IO_E_DRIVER_NOT_INITIALIZED	the common driver init function has not been called before
IO_E_CHANNEL_BUSY	channel has been initialized before
IO_E_UNKNOWN	an unknown error occurred

7.13.5.4 IO_ErrorType IO_LIN_Read (**IO_LIN_DATA_FRAME *const frame**)

Reads a LIN frame with the given id and of the given length.

Transmits the LIN Header. After this function returns, the user has to check with [IO_LIN_GetStatus\(\)](#) for receive complete.

Parameters

in, out	<i>frame</i>	Pointer to LIN frame structure. ID and length must be set before. The received frame will be stored in the data part.
---------	--------------	---

Returns**IO_ErrorType****Return values**

<i>IO_E_OK</i>	everything fine
<i>IO_E_BUSY</i>	the channel is busy, no new data received
<i>IO_E_INVALID_PARAMETER</i>	invalid parameter has been passed
<i>IO_E_NULL_POINTER</i>	null pointer has been passed
<i>IO_E_CHANNEL_NOT_CONFIGURED</i>	the channel was not configured
<i>IO_E_UNKNOWN</i>	an unknown error occurred

7.13.5.5 IO_ErrorType IO_LIN_Write (const IO_LIN_DATA_FRAME *const frame)

Transmit a LIN frame with the given id and of the given length.

Parameters

in	<i>frame</i>	Pointer to LIN frame structure. ID and length and data parts must be provided.
----	--------------	--

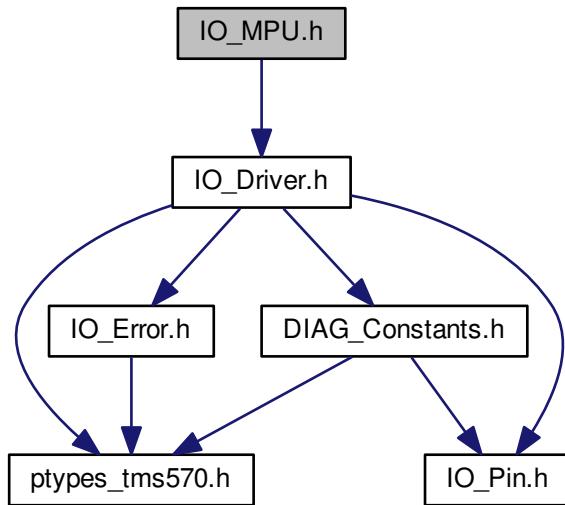
Returns**IO_ErrorType****Return values**

<i>IO_E_OK</i>	everything fine
<i>IO_E_BUSY</i>	the channel is busy, no new data received
<i>IO_E_INVALID_PARAMETER</i>	invalid parameter has been passed
<i>IO_E_NULL_POINTER</i>	null pointer has been passed
<i>IO_E_CHANNEL_NOT_CONFIGURED</i>	the channel was not configured

7.14 IO_MPU.h File Reference

IO Driver functions for the Memory Protection Unit (MPU)

Include dependency graph for IO_MPU.h:



Functions

- **IO_ErrorType IO_MPU_Disable (ubyte1 region_id)**
Disables (deactivates) the given User MPU region configuration.
- **IO_ErrorType IO_MPU_DisableAll (void)**
Disables (deactivates) all initialized User MPU regions.
- **IO_ErrorType IO_MPU_Enable (ubyte1 region_id)**
Enables (activates) the given User MPU region configuration.
- **IO_ErrorType IO_MPU_EnableAll (void)**
Enables (activates) all initialized User MPU regions.
- **IO_ErrorType IO_MPU_Init (ubyte1 region_id, ubyte4 start_address, ubyte4 size, ubyte1 sub-regions, ubyte1 attributes)**
Initializes a User MPU region.
- **IO_ErrorType IO_MPU_Policy (ubyte4 policy)**
Configures the MPU protection policy.

User MPU regions

- #define **IO_MPU_REGION_0** 0U
- #define **IO_MPU_REGION_1** 1U
- #define **IO_MPU_REGION_2** 2U
- #define **IO_MPU_REGION_3** 3U

MPU subregions

- #define `IO_MPU_ENABLE_SUBREGION_0` 0x01U
- #define `IO_MPU_ENABLE_SUBREGION_1` 0x02U
- #define `IO_MPU_ENABLE_SUBREGION_2` 0x04U
- #define `IO_MPU_ENABLE_SUBREGION_3` 0x08U
- #define `IO_MPU_ENABLE_SUBREGION_4` 0x10U
- #define `IO_MPU_ENABLE_SUBREGION_5` 0x20U
- #define `IO_MPU_ENABLE_SUBREGION_6` 0x40U
- #define `IO_MPU_ENABLE_SUBREGION_7` 0x80U
- #define `IO_MPU_ENABLE_ALL_SUBREGIONS` 0xFFU

MPU region sizes

- #define `IO_MPU_SIZE_32_B` 0x00000020UL
- #define `IO_MPU_SIZE_64_B` 0x00000040UL
- #define `IO_MPU_SIZE_128_B` 0x00000080UL
- #define `IO_MPU_SIZE_256_B` 0x00000100UL
- #define `IO_MPU_SIZE_512_B` 0x00000200UL
- #define `IO_MPU_SIZE_1_KB` 0x00000400UL
- #define `IO_MPU_SIZE_2_KB` 0x00000800UL
- #define `IO_MPU_SIZE_4_KB` 0x00001000UL
- #define `IO_MPU_SIZE_8_KB` 0x00002000UL
- #define `IO_MPU_SIZE_16_KB` 0x00004000UL
- #define `IO_MPU_SIZE_32_KB` 0x00008000UL
- #define `IO_MPU_SIZE_64_KB` 0x00010000UL
- #define `IO_MPU_SIZE_128_KB` 0x00020000UL
- #define `IO_MPU_SIZE_256_KB` 0x00040000UL
- #define `IO_MPU_SIZE_512_KB` 0x00080000UL
- #define `IO_MPU_SIZE_1_MB` 0x00100000UL
- #define `IO_MPU_SIZE_2_MB` 0x00200000UL
- #define `IO_MPU_SIZE_4_MB` 0x00400000UL
- #define `IO_MPU_SIZE_8_MB` 0x00800000UL
- #define `IO_MPU_SIZE_16_MB` 0x01000000UL
- #define `IO_MPU_SIZE_32_MB` 0x02000000UL
- #define `IO_MPU_SIZE_64_MB` 0x04000000UL

MPU region access attributes

- #define `IO_MPU_ACCESS_NONE` 0x0U
- #define `IO_MPU_ACCESS_READ` 0x1U
- #define `IO_MPU_ACCESS_READ_WRITE` 0x3U
- #define `IO_MPU_ACCESS_READ_EXECUTE` 0x5U
- #define `IO_MPU_ACCESS_ANY` 0x7U

MPU protection policies

- #define `IO_MPU_POLICY_REGION0` 0x0U
- #define `IO_MPU_POLICY_OFF` 0x1U
- #define `IO_MPU_POLICY_ALLREGIONS` 0x2U

7.14.1 Detailed Description

IO Driver functions for the Memory Protection Unit (MPU)

The MPU allows the user to set the access attributes of certain parts of the memory.

If an address falls in more than one enabled regions, the one with the greater ID has the higher priority and will determine the access rights. If a region is disabled, it does not affect the access rights in any way.

An overview of the memory protection concept and a discussion about the solutions available for the user application can be found in [I/O Driver Memory Protection](#).

7.14.2 MPU initialization

User MPU regions can be initialized only after `IO_Driver_Init()` has been called.

Once a region is initialized, it cannot be deinitialized or reconfigured to different parameters.

7.14.3 MPU access settings

A User MPU region's access settings can be combined by setting the read, write or executable bits in the `attributes` parameter of `IO_MPU_Init()`, with the exception that an area cannot be write-only (i.e. write but not read access).

7.14.4 MPU fault handling

See [MPU Violations](#) in the I/O Driver Memory Protection.

7.14.5 MPU subregions

Every region is divided into 8 equally-sized subregions. The integrator can freely choose which subregions of a region should be enabled, but it is not possible to disable all subregions. After a region has been initialized, the list of its enabled subregions cannot be changed.

For example, to enable subregion 5 through 7, the following needs to be used as the `subregions` parameter of `IO_MPU_Init()`:

```
IO_MPU_ENABLE_SUBREGION_5 | IO_MPU_ENABLE_SUBREGION_6 |
    IO_MPU_ENABLE_SUBREGION_7
```

All regions are enabled using the macro `IO_MPU_ENABLE_ALL_SUBREGIONS`.

7.14.6 MPU code example

```
// Initialize HY-TTC 500 driver
io_error = IO_Driver_Init(NULL);

// Configure User MPU region 0 to protect the I/O Driver RAM (read only).
io_error = IO_MPU_Init(IO_MPU_REGION_0,
```

```

        0x08038000UL,
        IO_MPU_SIZE_32_KB,
        IO_MPU_ENABLE_ALL_SUBREGIONS,
        IO_MPU_ACCESS_READ);

// Initialize User MPU region 1 with read-only access to a custom protected data location.
// This region will be enabled before calling the function with limited access
// and disabled after it returns.
io_error = IO_MPU_Init(IO_MPU_REGION_1,
                       PROTECTED_DATA_START,
                       PROTECTED_DATA_SIZE,
                       IO_MPU_ENABLE_ALL_SUBREGIONS,
                       IO_MPU_ACCESS_READ);

// start the RTC
IO_RTC_StartTime(&timestamp);

// loop forever
while (1)
{
    io_error = IO_Driver_TaskBegin();

    // application begin

    // call a function without any active access limitations
    any_safe_function();

    IO_MPU_Enable(IO_MPU_REGION_1);
    {
        // protection begin

        // call a function with limited access rights on the custom memory area
        limited_function();

        // protection end
    }
    IO_MPU_Disable(IO_MPU_REGION_1);

    // call a function without any active access limitations
    any_safe_function();

    IO_MPU_Enable(IO_MPU_REGION_1);
    IO_MPU_Enable(IO_MPU_REGION_0);
    {
        // unsafe code begin

        // call a unsafe function with (highly) limited access rights
        unsafe_function();

        // unsafe code end
    }
    IO_MPU_Disable(IO_MPU_REGION_0);
    IO_MPU_Disable(IO_MPU_REGION_1);

    // application end
    io_error = IO_Driver_TaskEnd();

    while (IO_RTC_GetTimeUS(timestamp) < CYCLE_TIME); // wait until cycle time has passed
    timestamp += CYCLE_TIME; // increase time stamp by cycle time
}

```

7.14.7 Macro Definition Documentation

7.14.7.1 #define IO_MPU_ACCESS_ANY 0x7U

This access setting allows any access to the memory region (read, write and execute).

Definition at line 573 of file IO_MPU.h.

7.14.7.2 #define IO_MPU_ACCESS_NONE 0x0U

This access setting doesn't allow any access to the memory region.

Definition at line 561 of file IO_MPU.h.

7.14.7.3 #define IO_MPU_ACCESS_READ 0x1U

This access setting allows only read access to the memory region.

Definition at line 564 of file IO_MPU.h.

7.14.7.4 #define IO_MPU_ACCESS_READ_EXECUTE 0x5U

This access setting allows read and execute accesses to the memory region.

Definition at line 570 of file IO_MPU.h.

7.14.7.5 #define IO_MPU_ACCESS_READ_WRITE 0x3U

This access setting allows read and write accesses to the memory region.

Definition at line 567 of file IO_MPU.h.

7.14.7.6 #define IO_MPU_ENABLE_ALL_SUBREGIONS 0xFFU

This MPU setting enables all subregions

Definition at line 476 of file IO_MPU.h.

7.14.7.7 #define IO_MPU_ENABLE_SUBREGION_0 0x01U

This MPU setting enables subregion 0

Definition at line 452 of file IO_MPU.h.

7.14.7.8 #define IO_MPU_ENABLE_SUBREGION_1 0x02U

This MPU setting enables subregion 1

Definition at line 455 of file IO_MPU.h.

7.14.7.9 #define IO_MPU_ENABLE_SUBREGION_2 0x04U

This MPU setting enables subregion 2

Definition at line 458 of file IO_MPU.h.

7.14.7.10 #define IO_MPU_ENABLE_SUBREGION_3 0x08U

This MPU setting enables subregion 3

Definition at line 461 of file IO_MPU.h.

7.14.7.11 #define IO_MPU_ENABLE_SUBREGION_4 0x10U

This MPU setting enables subregion 4

Definition at line 464 of file IO_MPU.h.

7.14.7.12 #define IO_MPU_ENABLE_SUBREGION_5 0x20U

This MPU setting enables subregion 5

Definition at line 467 of file IO_MPU.h.

7.14.7.13 #define IO_MPU_ENABLE_SUBREGION_6 0x40U

This MPU setting enables subregion 6

Definition at line 470 of file IO_MPU.h.

7.14.7.14 #define IO_MPU_ENABLE_SUBREGION_7 0x80U

This MPU setting enables subregion 7

Definition at line 473 of file IO_MPU.h.

7.14.7.15 #define IO_MPU_POLICY_ALLREGIONS 0x2U

I/O Driver deactivates all [User MPU regions](#) for its internal tasks. With this setting, users can use all [User MPU regions](#) freely to implement the memory protection needed in their application.

See also [I/O Driver Memory Protection](#).

Definition at line 605 of file IO_MPU.h.

7.14.7.16 #define IO_MPUM_POLICY_OFF 0x1U

I/O Driver does not make any changes to the MPU configuration in its internal tasks. This setting is used when the MPU is fully controlled by an external software component, e.g. SafeRTOS or CODESYS. User applications can use this setting if their configuration of the [User MPU regions](#) does not obstruct the execution of the I/O Driver internal tasks.

Definition at line 597 of file IO_MPUM.h.

7.14.7.17 #define IO_MPUM_POLICY_REGION0 0x0U

I/O Driver deactivates [IO_MPUM_REGION_0](#) for its internal tasks. This is the behavior of I/O Driver version 3.0.0 and earlier. This setting is the default (i.e. it will be used if [IO_MPUM_Policy\(\)](#) is never called) to maintain backwards compatibility.

See also [I/O Driver Memory Protection](#).

Definition at line 590 of file IO_MPUM.h.

7.14.7.18 #define IO_MPUM_REGION_0 0U

User MPU region 0 (lowest priority region)

Definition at line 432 of file IO_MPUM.h.

7.14.7.19 #define IO_MPUM_REGION_1 1U

User MPU region 1

Definition at line 435 of file IO_MPUM.h.

7.14.7.20 #define IO_MPUM_REGION_2 2U

User MPU region 2

Definition at line 438 of file IO_MPUM.h.

7.14.7.21 #define IO_MPUM_REGION_3 3U

User MPU region 3 (highest priority region)

Definition at line 441 of file IO_MPUM.h.

7.14.7.22 #define IO_MPUM_SIZE_128_B 0x00000080UL

This size setting defines the memory region size to be 128 bytes

Definition at line 493 of file IO_MPUM.h.

7.14.7.23 #define IO_MPU_SIZE_128_KB 0x00020000UL

This size setting defines the memory region size to be 128 KB (131,072 bytes)
Definition at line 523 of file IO_MPU.h.

7.14.7.24 #define IO_MPU_SIZE_16_KB 0x00004000UL

This size setting defines the memory region size to be 16 KB (16,384 bytes)
Definition at line 514 of file IO_MPU.h.

7.14.7.25 #define IO_MPU_SIZE_16_MB 0x01000000UL

This size setting defines the memory region size to be 16 MB (16,777,216 bytes)
Definition at line 544 of file IO_MPU.h.

7.14.7.26 #define IO_MPU_SIZE_1_KB 0x00000400UL

This size setting defines the memory region size to be 1 KB (1024 bytes)
Definition at line 502 of file IO_MPU.h.

7.14.7.27 #define IO_MPU_SIZE_1_MB 0x00100000UL

This size setting defines the memory region size to be 1 MB (1,048,576 bytes)
Definition at line 532 of file IO_MPU.h.

7.14.7.28 #define IO_MPU_SIZE_256_B 0x00000100UL

This size setting defines the memory region size to be 256 bytes
Definition at line 496 of file IO_MPU.h.

7.14.7.29 #define IO_MPU_SIZE_256_KB 0x00040000UL

This size setting defines the memory region size to be 256 KB (262,144 bytes)
Definition at line 526 of file IO_MPU.h.

7.14.7.30 #define IO_MPU_SIZE_2_KB 0x00000800UL

This size setting defines the memory region size to be 2 KB (2048 bytes)
Definition at line 505 of file IO_MPU.h.

7.14.7.31 #define IO_MPUSIZE_2_MB 0x00200000UL

This size setting defines the memory region size to be 2 MB (2,097,152 bytes)
Definition at line 535 of file IO_MPUS.h.

7.14.7.32 #define IO_MPUSIZE_32_B 0x00000020UL

This size setting defines the memory region size to be 32 bytes
Definition at line 487 of file IO_MPUS.h.

7.14.7.33 #define IO_MPUSIZE_32_KB 0x00008000UL

This size setting defines the memory region size to be 32 KB (32,768 bytes)
Definition at line 517 of file IO_MPUS.h.

7.14.7.34 #define IO_MPUSIZE_32_MB 0x02000000UL

This size setting defines the memory region size to be 32 MB (33,554,432 bytes)
Definition at line 547 of file IO_MPUS.h.

7.14.7.35 #define IO_MPUSIZE_4_KB 0x00001000UL

This size setting defines the memory region size to be 4 KB (4096 bytes)
Definition at line 508 of file IO_MPUS.h.

7.14.7.36 #define IO_MPUSIZE_4_MB 0x00400000UL

This size setting defines the memory region size to be 4 MB (4,194,304 bytes)
Definition at line 538 of file IO_MPUS.h.

7.14.7.37 #define IO_MPUSIZE_512_B 0x00000200UL

This size setting defines the memory region size to be 512 bytes
Definition at line 499 of file IO_MPUS.h.

7.14.7.38 #define IO_MPUSIZE_512_KB 0x00080000UL

This size setting defines the memory region size to be 512 KB (524,288 bytes)
Definition at line 529 of file IO_MPUS.h.

7.14.7.39 #define IO_MPU_SIZE_64_B 0x00000040UL

This size setting defines the memory region size to be 64 bytes

Definition at line 490 of file IO_MPU.h.

7.14.7.40 #define IO_MPU_SIZE_64_KB 0x00010000UL

This size setting defines the memory region size to be 64 KB (65,536 bytes)

Definition at line 520 of file IO_MPU.h.

7.14.7.41 #define IO_MPU_SIZE_64_MB 0x04000000UL

This size setting defines the memory region size to be 64 MB (67,108,864 bytes)

Definition at line 550 of file IO_MPU.h.

7.14.7.42 #define IO_MPU_SIZE_8_KB 0x00002000UL

This size setting defines the memory region size to be 8 KB (8192 bytes)

Definition at line 511 of file IO_MPU.h.

7.14.7.43 #define IO_MPU_SIZE_8_MB 0x00800000UL

This size setting defines the memory region size to be 8 MB (8,388,608 bytes)

Definition at line 541 of file IO_MPU.h.

7.14.8 Function Documentation**7.14.8.1 IO_ErrorType IO_MPU_Disable (ubyte1 region_id)**

Disables (deactivates) the given User MPU region configuration.

Parametersregion_idRegion ID. One of the macros in the [User MPU regions](#).**Returns**[IO_ErrorType](#)**Return values**

IO_E_OK	Everything went fine.
IO_E_CHANNEL_NOT_CONFIGURED	The given region has not been initialized.
IO_E_MPU_REGION_DISABLED	The given region is already disabled.
IO_E_INVALID_PARAMETER	An invalid region ID was given.

7.14.8.2 IO_ErrorType IO_MPUs_DisableAll (void)

Disables (deactivates) all initialized [User MPU regions](#).

Returns

[IO_ErrorType](#)

Return values

IO_E_OK	Operation successful.
IO_E_MPUS_REGION_DISABLED	One of the regions is already disabled.
IO_E_DRIVER_NOT_INITIALIZED	The common driver initialization function has not been called before.

7.14.8.3 IO_ErrorType IO_MPUs_Enable (ubyte1 region_id)

Enables (activates) the given User MPU region configuration.

Parameters

<i>region_id</i>	Region ID. One of the macros in the User MPU regions .
------------------	--

Returns

[IO_ErrorType](#)

Return values

IO_E_OK	Everything went fine.
IO_E_CHANNEL_NOT_CONFIGURED	The given region has not been initialized.
IO_E_MPUS_REGION_ENABLED	The given region is already enabled.
IO_E_INVALID_PARAMETER	An invalid region ID was given.

7.14.8.4 IO_ErrorType IO_MPUs_EnableAll (void)

Enables (activates) all initialized [User MPU regions](#).

Returns

[IO_ErrorType](#)

Return values

IO_E_OK	Operation successful.
IO_E_MPUS_REGION_ENABLED	One of the regions is already enabled.
IO_E_DRIVER_NOT_INITIALIZED	The common driver initialization function has not been called before.

7.14.8.5 IO_ErrorType IO_MPU_Init (ubyte1 *region_id*, ubyte4 *start_address*, ubyte4 *size*, ubyte1 *subregions*, ubyte1 *attributes*)

Initializes a User MPU region.

Parameters

<i>region_id</i>	Region ID. One of the macros in the User MPU regions .
<i>start_address</i>	Start address of the region. Must be a multiple of the region size.
<i>size</i>	Size of the region. One of the macros in the MPU region sizes .
<i>subregions</i>	Defines which subregions of this region are enabled, where the N-th least significant bit denotes the N-th subregion. See section MPU subregions for details.
<i>attributes</i>	Access attributes of the region. One of the macros in the MPU region access attributes .

Returns

[IO_ErrorType](#)

Return values

IO_E_OK	Everything went fine.
IO_E_BUSY	The MPU region has already been initialized once.
IO_E_INVALID_PARAMETER	An invalid parameter was given.
IO_E_DRIVER_NOT_INITIALIZED	The common driver init function has not been called before

Remarks

The region must lie entirely (including its beginning and end):

- between 0x0000.0000 and 0x002F.FFFF for code regions (internal flash memory),
- between 0x0800.0000 and 0x0803.FFFF for regions in the internal RAM,
- between 0x6000.0000 and 0x601F.FFFF for regions in the external RAM,
- between 0x6400.0000 and 0x647F.FFFF for regions in the external flash memory,
- between 0xF000.0000 and 0xF07F.FFFF for regions in the OTP, ECC, and application configuration data region (internal flash memory).
- between 0xFC00.0000 and 0xFFFF.FFFF for regions in the peripherals address space.

7.14.8.6 IO_ErrorType IO_MPU_Policy (ubyte4 *policy*)

Configures the MPU protection policy.

Parameters

<i>policy</i>	The policy that will be used for handling the User MPU regions in the I/O Driver internal tasks: <ul style="list-style-type: none"> • IO_MPU_POLICY_REGION0 • IO_MPU_POLICY_OFF • IO_MPU_POLICY_ALLREGIONS
---------------	---

Note

The I/O Driver defaults to use the `IO_MPUPOLICY_REGION0` setting if this function is not called.

Attention

Using this function requires Bootloader 3.2 or newer.

If an upgrade of the Bootloader is not possible, the function can be used with older Bootloader versions provided that the following requirement is met:

The memory area `IO_DRIVER_DATA_COMMON` (address range 0x0803FAE0 to 0x0803FEDF) shall be covered by `IO_MPUREGION_0` or no User MPU region at all.

Returns

`IO_ErrorType`

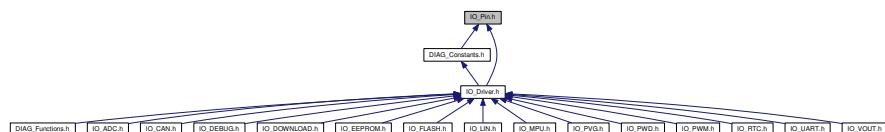
Return values

<code>IO_E_OK</code>	Operation successful.
<code>IO_E_INVALID_PARAMETER</code>	The specified policy is not valid.

7.15 IO_Pin.h File Reference

This header file contains pin definitions for the I/O driver, and aliases for the pins.

This graph shows which files directly or indirectly include this file:



Connector Pins

Defines for the Connector Pins.

- #define `IO_PIN_101` 80U
- #define `IO_PIN_102` 84U
- #define `IO_PIN_103` 0U
- #define `IO_PIN_104` 2U
- #define `IO_PIN_105` 4U
- #define `IO_PIN_106` 6U
- #define `IO_PIN_107` 8U
- #define `IO_PIN_108` 10U
- #define `IO_PIN_109` 12U
- #define `IO_PIN_110` 14U

- #define IO_PIN_111 16U
- #define IO_PIN_112 18U
- #define IO_PIN_113 20U
- #define IO_PIN_114 22U
- #define IO_PIN_115 24U
- #define IO_PIN_116 26U
- #define IO_PIN_117 28U
- #define IO_PIN_122 30U
- #define IO_PIN_123 32U
- #define IO_PIN_124 34U
- #define IO_PIN_125 81U
- #define IO_PIN_126 85U
- #define IO_PIN_127 1U
- #define IO_PIN_128 3U
- #define IO_PIN_129 5U
- #define IO_PIN_130 7U
- #define IO_PIN_131 9U
- #define IO_PIN_132 11U
- #define IO_PIN_133 13U
- #define IO_PIN_134 15U
- #define IO_PIN_135 17U
- #define IO_PIN_136 19U
- #define IO_PIN_137 21U
- #define IO_PIN_138 23U
- #define IO_PIN_139 25U
- #define IO_PIN_140 27U
- #define IO_PIN_141 29U
- #define IO_PIN_146 31U
- #define IO_PIN_147 33U
- #define IO_PIN_148 35U
- #define IO_PIN_149 36U
- #define IO_PIN_150 82U
- #define IO_PIN_151 86U
- #define IO_PIN_152 38U
- #define IO_PIN_153 52U
- #define IO_PIN_154 66U
- #define IO_PIN_155 40U
- #define IO_PIN_156 54U
- #define IO_PIN_157 68U
- #define IO_PIN_158 42U
- #define IO_PIN_159 56U
- #define IO_PIN_160 70U
- #define IO_PIN_161 88U
- #define IO_PIN_162 59U
- #define IO_PIN_163 73U
- #define IO_PIN_164 91U
- #define IO_PIN_165 61U
- #define IO_PIN_166 75U
- #define IO_PIN_167 93U
- #define IO_PIN_168 63U
- #define IO_PIN_169 77U
- #define IO_PIN_170 95U
- #define IO_PIN_171 65U
- #define IO_PIN_172 79U
- #define IO_PIN_173 37U
- #define IO_PIN_174 83U

- #define IO_PIN_175 87U
- #define IO_PIN_176 39U
- #define IO_PIN_177 53U
- #define IO_PIN_178 67U
- #define IO_PIN_179 41U
- #define IO_PIN_180 55U
- #define IO_PIN_181 69U
- #define IO_PIN_182 43U
- #define IO_PIN_183 57U
- #define IO_PIN_184 71U
- #define IO_PIN_185 89U
- #define IO_PIN_186 58U
- #define IO_PIN_187 72U
- #define IO_PIN_188 90U
- #define IO_PIN_189 60U
- #define IO_PIN_190 74U
- #define IO_PIN_191 92U
- #define IO_PIN_192 62U
- #define IO_PIN_193 76U
- #define IO_PIN_194 94U
- #define IO_PIN_195 64U
- #define IO_PIN_196 78U
- #define IO_PIN_207 97U
- #define IO_PIN_220 98U
- #define IO_PIN_221 101U
- #define IO_PIN_234 100U
- #define IO_PIN_238 45U
- #define IO_PIN_239 47U
- #define IO_PIN_240 49U
- #define IO_PIN_241 51U
- #define IO_PIN_246 96U
- #define IO_PIN_247 99U
- #define IO_PIN_251 44U
- #define IO_PIN_252 46U
- #define IO_PIN_253 48U
- #define IO_PIN_254 50U
- #define IO_PIN_118 253U
- #define IO_PIN_201 254U
- #define IO_PIN_NONE 255U

Internal Pins

These Pins are not available on the ECU Connector. They are needed for internal control or monitoring.

- #define IO_INT_PIN_CAN_CH0 102U
- #define IO_INT_PIN_CAN_CH1 103U
- #define IO_INT_PIN_CAN_CH2 104U
- #define IO_INT_PIN_CAN_CH3 105U
- #define IO_INT_PIN_CAN_CH4 106U
- #define IO_INT_PIN_CAN_CH5 107U
- #define IO_INT_PIN_CAN_CH6 108U
- #define IO_INT_PIN_SAFETY_SW_0 109U
- #define IO_INT_PIN_SAFETY_SW_1 110U
- #define IO_INT_PIN_SAFETY_SW_2 111U

- #define IO_INT_PIN_SAFETY_SW_VP 112U
- #define IO_INT_PIN_ENDRV_CPU 113U
- #define IO_INT_PIN_PWD 117U
- #define IO_INT_PIN_TEMP 120U
- #define IO_INT_PIN_REF_2V5 121U
- #define IO_INT_PIN_1V2 122U
- #define IO_INT_PIN_VMON 123U

Pin Aliases

Aliases for the pin definitions based on their function.

- #define IO_ADC_00 IO_PIN_103
- #define IO_ADC_01 IO_PIN_127
- #define IO_ADC_02 IO_PIN_104
- #define IO_ADC_03 IO_PIN_128
- #define IO_ADC_04 IO_PIN_105
- #define IO_ADC_05 IO_PIN_129
- #define IO_ADC_06 IO_PIN_106
- #define IO_ADC_07 IO_PIN_130
- #define IO_ADC_08 IO_PIN_107
- #define IO_ADC_09 IO_PIN_131
- #define IO_ADC_10 IO_PIN_108
- #define IO_ADC_11 IO_PIN_132
- #define IO_ADC_12 IO_PIN_109
- #define IO_ADC_13 IO_PIN_133
- #define IO_ADC_14 IO_PIN_110
- #define IO_ADC_15 IO_PIN_134
- #define IO_ADC_16 IO_PIN_111
- #define IO_ADC_17 IO_PIN_135
- #define IO_ADC_18 IO_PIN_112
- #define IO_ADC_19 IO_PIN_136
- #define IO_ADC_20 IO_PIN_113
- #define IO_ADC_21 IO_PIN_137
- #define IO_ADC_22 IO_PIN_114
- #define IO_ADC_23 IO_PIN_138
- #define IO_ADC_24 IO_PIN_115
- #define IO_ADC_25 IO_PIN_139
- #define IO_ADC_26 IO_PIN_116
- #define IO_ADC_27 IO_PIN_140
- #define IO_ADC_28 IO_PIN_117
- #define IO_ADC_29 IO_PIN_141
- #define IO_ADC_30 IO_PIN_122
- #define IO_ADC_31 IO_PIN_146
- #define IO_ADC_32 IO_PIN_123
- #define IO_ADC_33 IO_PIN_147
- #define IO_ADC_34 IO_PIN_124
- #define IO_ADC_35 IO_PIN_148
- #define IO_ADC_36 IO_PIN_149
- #define IO_ADC_37 IO_PIN_173
- #define IO_ADC_38 IO_PIN_152
- #define IO_ADC_39 IO_PIN_176
- #define IO_ADC_40 IO_PIN_155
- #define IO_ADC_41 IO_PIN_179
- #define IO_ADC_42 IO_PIN_158

- #define IO_ADC_43 IO_PIN_182
- #define IO_ADC_44 IO_PIN_251
- #define IO_ADC_45 IO_PIN_238
- #define IO_ADC_46 IO_PIN_252
- #define IO_ADC_47 IO_PIN_239
- #define IO_ADC_48 IO_PIN_253
- #define IO_ADC_49 IO_PIN_240
- #define IO_ADC_50 IO_PIN_254
- #define IO_ADC_51 IO_PIN_241
- #define IO_ADC_52 IO_PIN_161
- #define IO_ADC_53 IO_PIN_185
- #define IO_ADC_54 IO_PIN_188
- #define IO_ADC_55 IO_PIN_164
- #define IO_ADC_56 IO_PIN_191
- #define IO_ADC_57 IO_PIN_167
- #define IO_ADC_58 IO_PIN_194
- #define IO_ADC_59 IO_PIN_170
- #define IO_ADC_UBAT IO_PIN_246
- #define IO_ADC_K15 IO_K15
- #define IO_ADC_WAKE_UP IO_WAKEUP
- #define IO_ADC_SENSOR_SUPPLY_0 IO_SENSOR_SUPPLY_0
- #define IO_ADC_SENSOR_SUPPLY_1 IO_SENSOR_SUPPLY_1
- #define IO_ADC_SENSOR_SUPPLY_2 IO_SENSOR_SUPPLY_2
- #define IO_ADC_SAFETY_SW_0 IO_INT_SAFETY_SW_0
- #define IO_ADC_SAFETY_SW_1 IO_INT_SAFETY_SW_1
- #define IO_ADC_SAFETY_SW_2 IO_INT_SAFETY_SW_2
- #define IO_ADC_BOARD_TEMP IO_INT_PIN_TEMP
- #define IO_SENSOR_SUPPLY_0 IO_PIN_247
- #define IO_SENSOR_SUPPLY_1 IO_PIN_234
- #define IO_SENSOR_SUPPLY_2 IO_PIN_221
- #define IO_K15 IO_PIN_207
- #define IO_WAKEUP IO_PIN_220
- #define IO_GND IO_PIN_118
- #define IO_UBAT IO_PIN_201
- #define IO_PWD_00 IO_PIN_115
- #define IO_PWD_01 IO_PIN_139
- #define IO_PWD_02 IO_PIN_116
- #define IO_PWD_03 IO_PIN_140
- #define IO_PWD_04 IO_PIN_117
- #define IO_PWD_05 IO_PIN_141
- #define IO_PWD_06 IO_PIN_122
- #define IO_PWD_07 IO_PIN_146
- #define IO_PWD_08 IO_PIN_123
- #define IO_PWD_09 IO_PIN_147
- #define IO_PWD_10 IO_PIN_124
- #define IO_PWD_11 IO_PIN_148
- #define IO_PWD_12 IO_PIN_101
- #define IO_PWD_13 IO_PIN_125
- #define IO_PWD_14 IO_PIN_150
- #define IO_PWD_15 IO_PIN_174
- #define IO_PWD_16 IO_PIN_102
- #define IO_PWD_17 IO_PIN_126
- #define IO_PWD_18 IO_PIN_151
- #define IO_PWD_19 IO_PIN_175
- #define IO_PWM_00 IO_PIN_153
- #define IO_PWM_01 IO_PIN_177

- #define IO_PWM_02 IO_PIN_156
- #define IO_PWM_03 IO_PIN_180
- #define IO_PWM_04 IO_PIN_159
- #define IO_PWM_05 IO_PIN_183
- #define IO_PWM_06 IO_PIN_186
- #define IO_PWM_07 IO_PIN_162
- #define IO_PWM_08 IO_PIN_189
- #define IO_PWM_09 IO_PIN_165
- #define IO_PWM_10 IO_PIN_192
- #define IO_PWM_11 IO_PIN_168
- #define IO_PWM_12 IO_PIN_195
- #define IO_PWM_13 IO_PIN_171
- #define IO_PWM_14 IO_PIN_154
- #define IO_PWM_15 IO_PIN_178
- #define IO_PWM_16 IO_PIN_157
- #define IO_PWM_17 IO_PIN_181
- #define IO_PWM_18 IO_PIN_160
- #define IO_PWM_19 IO_PIN_184
- #define IO_PWM_20 IO_PIN_187
- #define IO_PWM_21 IO_PIN_163
- #define IO_PWM_22 IO_PIN_190
- #define IO_PWM_23 IO_PIN_166
- #define IO_PWM_24 IO_PIN_193
- #define IO_PWM_25 IO_PIN_169
- #define IO_PWM_26 IO_PIN_196
- #define IO_PWM_27 IO_PIN_172
- #define IO_PWM_28 IO_PIN_101
- #define IO_PWM_29 IO_PIN_125
- #define IO_PWM_30 IO_PIN_150
- #define IO_PWM_31 IO_PIN_174
- #define IO_PWM_32 IO_PIN_102
- #define IO_PWM_33 IO_PIN_126
- #define IO_PWM_34 IO_PIN_151
- #define IO_PWM_35 IO_PIN_175
- #define IO_PVG_00 IO_PIN_161
- #define IO_PVG_01 IO_PIN_185
- #define IO_PVG_02 IO_PIN_188
- #define IO_PVG_03 IO_PIN_164
- #define IO_PVG_04 IO_PIN_191
- #define IO_PVG_05 IO_PIN_167
- #define IO_PVG_06 IO_PIN_194
- #define IO_PVG_07 IO_PIN_170
- #define IO_VOUT_00 IO_PIN_161
- #define IO_VOUT_01 IO_PIN_185
- #define IO_VOUT_02 IO_PIN_188
- #define IO_VOUT_03 IO_PIN_164
- #define IO_VOUT_04 IO_PIN_191
- #define IO_VOUT_05 IO_PIN_167
- #define IO_VOUT_06 IO_PIN_194
- #define IO_VOUT_07 IO_PIN_170
- #define IO_DI_00 IO_PIN_153
- #define IO_DI_01 IO_PIN_177
- #define IO_DI_02 IO_PIN_156
- #define IO_DI_03 IO_PIN_180
- #define IO_DI_04 IO_PIN_159
- #define IO_DI_05 IO_PIN_183

- #define IO_DI_06 IO_PIN_186
- #define IO_DI_07 IO_PIN_162
- #define IO_DI_08 IO_PIN_189
- #define IO_DI_09 IO_PIN_165
- #define IO_DI_10 IO_PIN_192
- #define IO_DI_11 IO_PIN_168
- #define IO_DI_12 IO_PIN_195
- #define IO_DI_13 IO_PIN_171
- #define IO_DI_14 IO_PIN_154
- #define IO_DI_15 IO_PIN_178
- #define IO_DI_16 IO_PIN_157
- #define IO_DI_17 IO_PIN_181
- #define IO_DI_18 IO_PIN_160
- #define IO_DI_19 IO_PIN_184
- #define IO_DI_20 IO_PIN_187
- #define IO_DI_21 IO_PIN_163
- #define IO_DI_22 IO_PIN_190
- #define IO_DI_23 IO_PIN_166
- #define IO_DI_24 IO_PIN_193
- #define IO_DI_25 IO_PIN_169
- #define IO_DI_26 IO_PIN_196
- #define IO_DI_27 IO_PIN_172
- #define IO_DI_28 IO_PIN_101
- #define IO_DI_29 IO_PIN_125
- #define IO_DI_30 IO_PIN_150
- #define IO_DI_31 IO_PIN_174
- #define IO_DI_32 IO_PIN_102
- #define IO_DI_33 IO_PIN_126
- #define IO_DI_34 IO_PIN_151
- #define IO_DI_35 IO_PIN_175
- #define IO_DI_36 IO_PIN_115
- #define IO_DI_37 IO_PIN_139
- #define IO_DI_38 IO_PIN_116
- #define IO_DI_39 IO_PIN_140
- #define IO_DI_40 IO_PIN_117
- #define IO_DI_41 IO_PIN_141
- #define IO_DI_42 IO_PIN_122
- #define IO_DI_43 IO_PIN_146
- #define IO_DI_44 IO_PIN_123
- #define IO_DI_45 IO_PIN_147
- #define IO_DI_46 IO_PIN_124
- #define IO_DI_47 IO_PIN_148
- #define IO_DI_48 IO_PIN_103
- #define IO_DI_49 IO_PIN_127
- #define IO_DI_50 IO_PIN_104
- #define IO_DI_51 IO_PIN_128
- #define IO_DI_52 IO_PIN_105
- #define IO_DI_53 IO_PIN_129
- #define IO_DI_54 IO_PIN_106
- #define IO_DI_55 IO_PIN_130
- #define IO_DI_56 IO_PIN_107
- #define IO_DI_57 IO_PIN_131
- #define IO_DI_58 IO_PIN_108
- #define IO_DI_59 IO_PIN_132
- #define IO_DI_60 IO_PIN_109
- #define IO_DI_61 IO_PIN_133

- #define IO_DI_62 IO_PIN_110
- #define IO_DI_63 IO_PIN_134
- #define IO_DI_64 IO_PIN_111
- #define IO_DI_65 IO_PIN_135
- #define IO_DI_66 IO_PIN_112
- #define IO_DI_67 IO_PIN_136
- #define IO_DI_68 IO_PIN_113
- #define IO_DI_69 IO_PIN_137
- #define IO_DI_70 IO_PIN_114
- #define IO_DI_71 IO_PIN_138
- #define IO_DI_72 IO_PIN_149
- #define IO_DI_73 IO_PIN_173
- #define IO_DI_74 IO_PIN_152
- #define IO_DI_75 IO_PIN_176
- #define IO_DI_76 IO_PIN_155
- #define IO_DI_77 IO_PIN_179
- #define IO_DI_78 IO_PIN_158
- #define IO_DI_79 IO_PIN_182
- #define IO_DI_80 IO_PIN_251
- #define IO_DI_81 IO_PIN_238
- #define IO_DI_82 IO_PIN_252
- #define IO_DI_83 IO_PIN_239
- #define IO_DI_84 IO_PIN_253
- #define IO_DI_85 IO_PIN_240
- #define IO_DI_86 IO_PIN_254
- #define IO_DI_87 IO_PIN_241
- #define IO_DI_88 IO_PIN_161
- #define IO_DI_89 IO_PIN_185
- #define IO_DI_90 IO_PIN_188
- #define IO_DI_91 IO_PIN_164
- #define IO_DI_92 IO_PIN_191
- #define IO_DI_93 IO_PIN_167
- #define IO_DI_94 IO_PIN_194
- #define IO_DI_95 IO_PIN_170
- #define IO_DO_00 IO_PIN_149
- #define IO_DO_01 IO_PIN_173
- #define IO_DO_02 IO_PIN_152
- #define IO_DO_03 IO_PIN_176
- #define IO_DO_04 IO_PIN_155
- #define IO_DO_05 IO_PIN_179
- #define IO_DO_06 IO_PIN_158
- #define IO_DO_07 IO_PIN_182
- #define IO_DO_08 IO_PIN_251
- #define IO_DO_09 IO_PIN_238
- #define IO_DO_10 IO_PIN_252
- #define IO_DO_11 IO_PIN_239
- #define IO_DO_12 IO_PIN_253
- #define IO_DO_13 IO_PIN_240
- #define IO_DO_14 IO_PIN_254
- #define IO_DO_15 IO_PIN_241
- #define IO_DO_16 IO_PIN_153
- #define IO_DO_17 IO_PIN_177
- #define IO_DO_18 IO_PIN_156
- #define IO_DO_19 IO_PIN_180
- #define IO_DO_20 IO_PIN_159
- #define IO_DO_21 IO_PIN_183

- #define IO_DO_22 IO_PIN_186
- #define IO_DO_23 IO_PIN_162
- #define IO_DO_24 IO_PIN_189
- #define IO_DO_25 IO_PIN_165
- #define IO_DO_26 IO_PIN_192
- #define IO_DO_27 IO_PIN_168
- #define IO_DO_28 IO_PIN_195
- #define IO_DO_29 IO_PIN_171
- #define IO_DO_30 IO_PIN_154
- #define IO_DO_31 IO_PIN_178
- #define IO_DO_32 IO_PIN_157
- #define IO_DO_33 IO_PIN_181
- #define IO_DO_34 IO_PIN_160
- #define IO_DO_35 IO_PIN_184
- #define IO_DO_36 IO_PIN_187
- #define IO_DO_37 IO_PIN_163
- #define IO_DO_38 IO_PIN_190
- #define IO_DO_39 IO_PIN_166
- #define IO_DO_40 IO_PIN_193
- #define IO_DO_41 IO_PIN_169
- #define IO_DO_42 IO_PIN_196
- #define IO_DO_43 IO_PIN_172
- #define IO_DO_44 IO_PIN_101
- #define IO_DO_45 IO_PIN_125
- #define IO_DO_46 IO_PIN_150
- #define IO_DO_47 IO_PIN_174
- #define IO_DO_48 IO_PIN_102
- #define IO_DO_49 IO_PIN_126
- #define IO_DO_50 IO_PIN_151
- #define IO_DO_51 IO_PIN_175
- #define IO_DO_52 IO_PIN_161
- #define IO_DO_53 IO_PIN_185
- #define IO_DO_54 IO_PIN_188
- #define IO_DO_55 IO_PIN_164
- #define IO_DO_56 IO_PIN_191
- #define IO_DO_57 IO_PIN_167
- #define IO_DO_58 IO_PIN_194
- #define IO_DO_59 IO_PIN_170

Internal devices

These channels are used to control internal devices.

- #define IO_CAN_CHANNEL_0 IO_INT_PIN_CAN_CH0
- #define IO_CAN_CHANNEL_1 IO_INT_PIN_CAN_CH1
- #define IO_CAN_CHANNEL_2 IO_INT_PIN_CAN_CH2
- #define IO_CAN_CHANNEL_3 IO_INT_PIN_CAN_CH3
- #define IO_CAN_CHANNEL_4 IO_INT_PIN_CAN_CH4
- #define IO_CAN_CHANNEL_5 IO_INT_PIN_CAN_CH5
- #define IO_CAN_CHANNEL_6 IO_INT_PIN_CAN_CH6
- #define IO_INT_SAFETY_SW_0 IO_INT_PIN_SAFETY_SW_0
- #define IO_INT_SAFETY_SW_1 IO_INT_PIN_SAFETY_SW_1
- #define IO_INT_SAFETY_SW_2 IO_INT_PIN_SAFETY_SW_2
- #define IO_INT_POWERSTAGE_ENABLE IO_INT_PIN_ENDRV_CPU

7.15.1 Detailed Description

This header file contains pin definitions for the I/O driver, and aliases for the pins.

Connector pin	Internal pin number	.	.	Pin aliases	.	.
Pin 103	0	IO_ADC_00	IO_DI_48			
Pin 127	1	IO_ADC_01	IO_DI_49			
Pin 104	2	IO_ADC_02	IO_DI_50			
Pin 128	3	IO_ADC_03	IO_DI_51			
Pin 105	4	IO_ADC_04	IO_DI_52			
Pin 129	5	IO_ADC_05	IO_DI_53			
Pin 106	6	IO_ADC_06	IO_DI_54			
Pin 130	7	IO_ADC_07	IO_DI_55			
Pin 107	8	IO_ADC_08	IO_DI_56			
Pin 131	9	IO_ADC_09	IO_DI_57			
Pin 108	10	IO_ADC_10	IO_DI_58			
Pin 132	11	IO_ADC_11	IO_DI_59			
Pin 109	12	IO_ADC_12	IO_DI_60			
Pin 133	13	IO_ADC_13	IO_DI_61			
Pin 110	14	IO_ADC_14	IO_DI_62			
Pin 134	15	IO_ADC_15	IO_DI_63			
Pin 111	16	IO_ADC_16	IO_DI_64			
Pin 135	17	IO_ADC_17	IO_DI_65			
Pin 112	18	IO_ADC_18	IO_DI_66			
Pin 136	19	IO_ADC_19	IO_DI_67			
Pin 113	20	IO_ADC_20	IO_DI_68			
Pin 137	21	IO_ADC_21	IO_DI_69			
Pin 114	22	IO_ADC_22	IO_DI_70			
Pin 138	23	IO_ADC_23	IO_DI_71			
Pin 115	24	IO_PWD_00	IO_ADC_24	IO_DI_36		
Pin 139	25	IO_PWD_01	IO_ADC_25	IO_DI_37		
Pin 116	26	IO_PWD_02	IO_ADC_26	IO_DI_38		
Pin 140	27	IO_PWD_03	IO_ADC_27	IO_DI_39		
Pin 117	28	IO_PWD_04	IO_ADC_28	IO_DI_40		
Pin 141	29	IO_PWD_05	IO_ADC_29	IO_DI_41		
Pin 122	30	IO_PWD_06	IO_ADC_30	IO_DI_42		
Pin 146	31	IO_PWD_07	IO_ADC_31	IO_DI_43		
Pin 123	32	IO_PWD_08	IO_ADC_32	IO_DI_44		
Pin 147	33	IO_PWD_09	IO_ADC_33	IO_DI_45		
Pin 124	34	IO_PWD_10	IO_ADC_34	IO_DI_46		
Pin 148	35	IO_PWD_11	IO_ADC_35	IO_DI_47		
Pin 149	36	IO_DO_00	IO_ADC_36	IO_DI_72		
Pin 173	37	IO_DO_01	IO_ADC_37	IO_DI_73		
Pin 152	38	IO_DO_02	IO_ADC_38	IO_DI_74		
Pin 176	39	IO_DO_03	IO_ADC_39	IO_DI_75		

Connector pin	Internal pin number	.	.	Pin aliases	.	.
Pin 155	40	IO_DO_04	IO_ADC_40	IO_DI_76		
Pin 179	41	IO_DO_05	IO_ADC_41	IO_DI_77		
Pin 158	42	IO_DO_06	IO_ADC_42	IO_DI_78		
Pin 182	43	IO_DO_07	IO_ADC_43	IO_DI_79		
Pin 251	44	IO_DO_08	IO_ADC_44	IO_DI_80		
Pin 238	45	IO_DO_09	IO_ADC_45	IO_DI_81		
Pin 252	46	IO_DO_10	IO_ADC_46	IO_DI_82		
Pin 239	47	IO_DO_11	IO_ADC_47	IO_DI_83		
Pin 253	48	IO_DO_12	IO_ADC_48	IO_DI_84		
Pin 240	49	IO_DO_13	IO_ADC_49	IO_DI_85		
Pin 254	50	IO_DO_14	IO_ADC_50	IO_DI_86		
Pin 241	51	IO_DO_15	IO_ADC_51	IO_DI_87		
Pin 153	52	IO_PWM_00	IO_DO_16	IO_DI_00		
Pin 177	53	IO_PWM_01	IO_DO_17	IO_DI_01		
Pin 156	54	IO_PWM_02	IO_DO_18	IO_DI_02		
Pin 180	55	IO_PWM_03	IO_DO_19	IO_DI_03		
Pin 159	56	IO_PWM_04	IO_DO_20	IO_DI_04		
Pin 183	57	IO_PWM_05	IO_DO_21	IO_DI_05		
Pin 186	58	IO_PWM_06	IO_DO_22	IO_DI_06		
Pin 162	59	IO_PWM_07	IO_DO_23	IO_DI_07		
Pin 189	60	IO_PWM_08	IO_DO_24	IO_DI_08		
Pin 165	61	IO_PWM_09	IO_DO_25	IO_DI_09		
Pin 192	62	IO_PWM_10	IO_DO_26	IO_DI_10		
Pin 168	63	IO_PWM_11	IO_DO_27	IO_DI_11		
Pin 195	64	IO_PWM_12	IO_DO_28	IO_DI_12		
Pin 171	65	IO_PWM_13	IO_DO_29	IO_DI_13		
Pin 154	66	IO_PWM_14	IO_DO_30	IO_DI_14		
Pin 178	67	IO_PWM_15	IO_DO_31	IO_DI_15		
Pin 157	68	IO_PWM_16	IO_DO_32	IO_DI_16		
Pin 181	69	IO_PWM_17	IO_DO_33	IO_DI_17		
Pin 160	70	IO_PWM_18	IO_DO_34	IO_DI_18		
Pin 184	71	IO_PWM_19	IO_DO_35	IO_DI_19		
Pin 187	72	IO_PWM_20	IO_DO_36	IO_DI_20		
Pin 163	73	IO_PWM_21	IO_DO_37	IO_DI_21		
Pin 190	74	IO_PWM_22	IO_DO_38	IO_DI_22		
Pin 166	75	IO_PWM_23	IO_DO_39	IO_DI_23		
Pin 193	76	IO_PWM_24	IO_DO_40	IO_DI_24		
Pin 169	77	IO_PWM_25	IO_DO_41	IO_DI_25		
Pin 196	78	IO_PWM_26	IO_DO_42	IO_DI_26		
Pin 172	79	IO_PWM_27	IO_DO_43	IO_DI_27		
Pin 101	80	IO_PWM_28	IO_DO_44	IO_DI_28	IO_PWD_12	
Pin 125	81	IO_PWM_29	IO_DO_45	IO_DI_29	IO_PWD_13	
Pin 150	82	IO_PWM_30	IO_DO_46	IO_DI_30	IO_PWD_14	
Pin 174	83	IO_PWM_31	IO_DO_47	IO_DI_31	IO_PWD_15	
Pin 102	84	IO_PWM_32	IO_DO_48	IO_DI_32	IO_PWD_16	

Connector pin	Internal pin number	.	.	Pin aliases	.	.
Pin 126	85	IO_PWM_33	IO_DO_49	IO_DI_33	IO_PWD_17	
Pin 151	86	IO_PWM_34	IO_DO_50	IO_DI_34	IO_PWD_18	
Pin 175	87	IO_PWM_35	IO_DO_51	IO_DI_35	IO_PWD_19	
Pin 161	88	IO_PVG_00	IO_VOUT_00	IO_DO_52	IO_ADC_52	IO_DI_88
Pin 185	89	IO_PVG_01	IO_VOUT_01	IO_DO_53	IO_ADC_53	IO_DI_89
Pin 188	90	IO_PVG_02	IO_VOUT_02	IO_DO_54	IO_ADC_54	IO_DI_90
Pin 164	91	IO_PVG_03	IO_VOUT_03	IO_DO_55	IO_ADC_55	IO_DI_91
Pin 191	92	IO_PVG_04	IO_VOUT_04	IO_DO_56	IO_ADC_56	IO_DI_92
Pin 167	93	IO_PVG_05	IO_VOUT_05	IO_DO_57	IO_ADC_57	IO_DI_93
Pin 194	94	IO_PVG_06	IO_VOUT_06	IO_DO_58	IO_ADC_58	IO_DI_94
Pin 170	95	IO_PVG_07	IO_VOUT_07	IO_DO_59	IO_ADC_59	IO_DI_95

7.15.2 Macro Definition Documentation

7.15.2.1 #define IO_ADC_00 IO_PIN_103

main function 3 mode ADC input

Definition at line 1161 of file IO_Pin.h.

7.15.2.2 #define IO_ADC_01 IO_PIN_127

main function 3 mode ADC input

Definition at line 1162 of file IO_Pin.h.

7.15.2.3 #define IO_ADC_02 IO_PIN_104

main function 3 mode ADC input

Definition at line 1163 of file IO_Pin.h.

7.15.2.4 #define IO_ADC_03 IO_PIN_128

main function 3 mode ADC input

Definition at line 1164 of file IO_Pin.h.

7.15.2.5 #define IO_ADC_04 IO_PIN_105

main function 3 mode ADC input

Definition at line 1165 of file IO_Pin.h.

7.15.2.6 #define IO_ADC_05 IO_PIN_129

main function 3 mode ADC input

Definition at line 1166 of file IO_Pin.h.

7.15.2.7 #define IO_ADC_06 IO_PIN_106

main function 3 mode ADC input

Definition at line 1167 of file IO_Pin.h.

7.15.2.8 #define IO_ADC_07 IO_PIN_130

main function 3 mode ADC input

Definition at line 1168 of file IO_Pin.h.

7.15.2.9 #define IO_ADC_08 IO_PIN_107

main function 2 mode 10V ADC input

Definition at line 1171 of file IO_Pin.h.

7.15.2.10 #define IO_ADC_09 IO_PIN_131

main function 2 mode 10V ADC input

Definition at line 1172 of file IO_Pin.h.

7.15.2.11 #define IO_ADC_10 IO_PIN_108

main function 2 mode 10V ADC input

Definition at line 1173 of file IO_Pin.h.

7.15.2.12 #define IO_ADC_11 IO_PIN_132

main function 2 mode 10V ADC input

Definition at line 1174 of file IO_Pin.h.

7.15.2.13 #define IO_ADC_12 IO_PIN_109

main function 2 mode 10V ADC input

Definition at line 1175 of file IO_Pin.h.

7.15.2.14 #define IO_ADC_13 IO_PIN_133

main function 2 mode 10V ADC input

Definition at line 1176 of file IO_Pin.h.

7.15.2.15 #define IO_ADC_14 IO_PIN_110

main function 2 mode 10V ADC input

Definition at line 1177 of file IO_Pin.h.

7.15.2.16 #define IO_ADC_15 IO_PIN_134

main function 2 mode 10V ADC input

Definition at line 1178 of file IO_Pin.h.

7.15.2.17 #define IO_ADC_16 IO_PIN_111

main function 2 mode 32V ADC input

Definition at line 1181 of file IO_Pin.h.

7.15.2.18 #define IO_ADC_17 IO_PIN_135

main function 2 mode 32V ADC input

Definition at line 1182 of file IO_Pin.h.

7.15.2.19 #define IO_ADC_18 IO_PIN_112

main function 2 mode 32V ADC input

Definition at line 1183 of file IO_Pin.h.

7.15.2.20 #define IO_ADC_19 IO_PIN_136

main function 2 mode 32V ADC input

Definition at line 1184 of file IO_Pin.h.

7.15.2.21 #define IO_ADC_20 IO_PIN_113

main function 2 mode 32V ADC input

Definition at line 1185 of file IO_Pin.h.

7.15.2.22 #define IO_ADC_21 IO_PIN_137

main function 2 mode 32V ADC input

Definition at line 1186 of file IO_Pin.h.

7.15.2.23 #define IO_ADC_22 IO_PIN_114

main function 2 mode 32V ADC input

Definition at line 1187 of file IO_Pin.h.

7.15.2.24 #define IO_ADC_23 IO_PIN_138

main function 2 mode 32V ADC input

Definition at line 1188 of file IO_Pin.h.

7.15.2.25 #define IO_ADC_24 IO_PIN_115

alternative ADC function for [IO_PWD_00](#)

Definition at line 1190 of file IO_Pin.h.

7.15.2.26 #define IO_ADC_25 IO_PIN_139

alternative ADC function for [IO_PWD_01](#)

Definition at line 1191 of file IO_Pin.h.

7.15.2.27 #define IO_ADC_26 IO_PIN_116

alternative ADC function for [IO_PWD_02](#)

Definition at line 1192 of file IO_Pin.h.

7.15.2.28 #define IO_ADC_27 IO_PIN_140

alternative ADC function for [IO_PWD_03](#)

Definition at line 1193 of file IO_Pin.h.

7.15.2.29 #define IO_ADC_28 IO_PIN_117

alternative ADC function for [IO_PWD_04](#)

Definition at line 1194 of file IO_Pin.h.

7.15.2.30 #define IO_ADC_29 IO_PIN_141

alternative ADC function for [IO_PWD_05](#)

Definition at line 1195 of file IO_Pin.h.

7.15.2.31 #define IO_ADC_30 IO_PIN_122

alternative ADC function for [IO_PWD_06](#)

Definition at line 1196 of file IO_Pin.h.

7.15.2.32 #define IO_ADC_31 IO_PIN_146

alternative ADC function for [IO_PWD_07](#)

Definition at line 1197 of file IO_Pin.h.

7.15.2.33 #define IO_ADC_32 IO_PIN_123

alternative ADC function for [IO_PWD_08](#)

Definition at line 1198 of file IO_Pin.h.

7.15.2.34 #define IO_ADC_33 IO_PIN_147

alternative ADC function for [IO_PWD_09](#)

Definition at line 1199 of file IO_Pin.h.

7.15.2.35 #define IO_ADC_34 IO_PIN_124

alternative ADC function for [IO_PWD_10](#)

Definition at line 1200 of file IO_Pin.h.

7.15.2.36 #define IO_ADC_35 IO_PIN_148

alternative ADC function for [IO_PWD_11](#)

Definition at line 1201 of file IO_Pin.h.

7.15.2.37 #define IO_ADC_36 IO_PIN_149

alternative ADC function for [IO_DO_00](#)

Definition at line 1203 of file IO_Pin.h.

7.15.2.38 #define IO_ADC_37 IO_PIN_173

alternative ADC function for [IO_DO_01](#)

Definition at line 1204 of file IO_Pin.h.

7.15.2.39 #define IO_ADC_38 IO_PIN_152

alternative ADC function for [IO_DO_02](#)

Definition at line 1205 of file IO_Pin.h.

7.15.2.40 #define IO_ADC_39 IO_PIN_176

alternative ADC function for [IO_DO_03](#)

Definition at line 1206 of file IO_Pin.h.

7.15.2.41 #define IO_ADC_40 IO_PIN_155

alternative ADC function for [IO_DO_04](#)

Definition at line 1207 of file IO_Pin.h.

7.15.2.42 #define IO_ADC_41 IO_PIN_179

alternative ADC function for [IO_DO_05](#)

Definition at line 1208 of file IO_Pin.h.

7.15.2.43 #define IO_ADC_42 IO_PIN_158

alternative ADC function for [IO_DO_06](#)

Definition at line 1209 of file IO_Pin.h.

7.15.2.44 #define IO_ADC_43 IO_PIN_182

alternative ADC function for [IO_DO_07](#)

Definition at line 1210 of file IO_Pin.h.

7.15.2.45 #define IO_ADC_44 IO_PIN_251

alternative ADC function for [IO_DO_08](#)

Definition at line 1212 of file IO_Pin.h.

7.15.2.46 #define IO_ADC_45 IO_PIN_238

alternative ADC function for [IO_DO_09](#)

Definition at line 1213 of file IO_Pin.h.

7.15.2.47 #define IO_ADC_46 IO_PIN_252

alternative ADC function for [IO_DO_10](#)

Definition at line 1214 of file IO_Pin.h.

7.15.2.48 #define IO_ADC_47 IO_PIN_239

alternative ADC function for [IO_DO_11](#)

Definition at line 1215 of file IO_Pin.h.

7.15.2.49 #define IO_ADC_48 IO_PIN_253

alternative ADC function for [IO_DO_12](#)

Definition at line 1216 of file IO_Pin.h.

7.15.2.50 #define IO_ADC_49 IO_PIN_240

alternative ADC function for [IO_DO_13](#)

Definition at line 1217 of file IO_Pin.h.

7.15.2.51 #define IO_ADC_50 IO_PIN_254

alternative ADC function for [IO_DO_14](#)

Definition at line 1218 of file IO_Pin.h.

7.15.2.52 #define IO_ADC_51 IO_PIN_241

alternative ADC function for [IO_DO_15](#)

Definition at line 1219 of file IO_Pin.h.

7.15.2.53 #define IO_ADC_52 IO_PIN_161

alternative ADC function for [IO_PVG_00](#)

Definition at line 1221 of file IO_Pin.h.

7.15.2.54 #define IO_ADC_53 IO_PIN_185

alternative ADC function for [IO_PVG_01](#)

Definition at line 1222 of file IO_Pin.h.

7.15.2.55 #define IO_ADC_54 IO_PIN_188

alternative ADC function for [IO_PVG_02](#)

Definition at line 1223 of file IO_Pin.h.

7.15.2.56 #define IO_ADC_55 IO_PIN_164

alternative ADC function for [IO_PVG_03](#)

Definition at line 1224 of file IO_Pin.h.

7.15.2.57 #define IO_ADC_56 IO_PIN_191

alternative ADC function for [IO_PVG_04](#)

Definition at line 1225 of file IO_Pin.h.

7.15.2.58 #define IO_ADC_57 IO_PIN_167

alternative ADC function for [IO_PVG_05](#)

Definition at line 1226 of file IO_Pin.h.

7.15.2.59 #define IO_ADC_58 IO_PIN_194

alternative ADC function for [IO_PVG_06](#)

Definition at line 1227 of file IO_Pin.h.

7.15.2.60 #define IO_ADC_59 IO_PIN_170

alternative ADC function for [IO_PVG_07](#)

Definition at line 1228 of file IO_Pin.h.

7.15.2.61 #define IO_ADC_BOARD_TEMP IO_INT_PIN_TEMP

Definition at line 1248 of file IO_Pin.h.

7.15.2.62 #define IO_ADC_K15 IO_K15

Definition at line 1240 of file IO_Pin.h.

7.15.2.63 #define IO_ADC_SAFETY_SW_0 IO_INT_SAFETY_SW_0

Definition at line 1245 of file IO_Pin.h.

7.15.2.64 #define IO_ADC_SAFETY_SW_1 IO_INT_SAFETY_SW_1

Definition at line 1246 of file IO_Pin.h.

7.15.2.65 #define IO_ADC_SAFETY_SW_2 IO_INT_SAFETY_SW_2

Definition at line 1247 of file IO_Pin.h.

7.15.2.66 #define IO_ADC_SENSOR_SUPPLY_0 IO_SENSOR_SUPPLY_0

Definition at line 1242 of file IO_Pin.h.

7.15.2.67 #define IO_ADC_SENSOR_SUPPLY_1 IO_SENSOR_SUPPLY_1

Definition at line 1243 of file IO_Pin.h.

7.15.2.68 #define IO_ADC_SENSOR_SUPPLY_2 IO_SENSOR_SUPPLY_2

Definition at line 1244 of file IO_Pin.h.

7.15.2.69 #define IO_ADC_UBAT IO_PIN_246

This ADC channel can be used to monitor the voltage on connector pin K30-A, i.e., the BAT+ CPU supply. Note that connector pins K30-P, i.e., BAT+, cannot be monitored at application level!

Definition at line 1231 of file IO_Pin.h.

7.15.2.70 #define IO_ADC_WAKE_UP IO_WAKEUP

Definition at line 1241 of file IO_Pin.h.

7.15.2.71 #define IO_CAN_CHANNEL_0 IO_INT_PIN_CAN_CH0

Internal Pin for CAN channel 0

Definition at line 1821 of file IO_Pin.h.

7.15.2.72 #define IO_CAN_CHANNEL_1 IO_INT_PIN_CAN_CH1

Internal Pin for CAN channel 1

Definition at line 1822 of file IO_Pin.h.

7.15.2.73 #define IO_CAN_CHANNEL_2 IO_INT_PIN_CAN_CH2

Internal Pin for CAN channel 2

Definition at line 1823 of file IO_Pin.h.

7.15.2.74 #define IO_CAN_CHANNEL_3 IO_INT_PIN_CAN_CH3

Internal Pin for CAN channel 3

Definition at line 1824 of file IO_Pin.h.

7.15.2.75 #define IO_CAN_CHANNEL_4 IO_INT_PIN_CAN_CH4

Internal Pin for CAN channel 4

Definition at line 1825 of file IO_Pin.h.

7.15.2.76 #define IO_CAN_CHANNEL_5 IO_INT_PIN_CAN_CH5

Internal Pin for CAN channel 5

Definition at line 1826 of file IO_Pin.h.

7.15.2.77 #define IO_CAN_CHANNEL_6 IO_INT_PIN_CAN_CH6

Internal Pin for CAN channel 6

Definition at line 1827 of file IO_Pin.h.

7.15.2.78 #define IO_DI_00 IO_PIN_153

alternative digital input function for [IO_PWM_00](#)

Definition at line 1359 of file IO_Pin.h.

7.15.2.79 #define IO_DI_01 IO_PIN_177

alternative digital input function for [IO_PWM_01](#)

Definition at line 1362 of file IO_Pin.h.

7.15.2.80 #define IO_DI_02 IO_PIN_156

alternative digital input function for [IO_PWM_02](#)

Definition at line 1365 of file IO_Pin.h.

7.15.2.81 #define IO_DI_03 IO_PIN_180

alternative digital input function for [IO_PWM_03](#)

Definition at line 1368 of file IO_Pin.h.

7.15.2.82 #define IO_DI_04 IO_PIN_159

alternative digital input function for [IO_PWM_04](#)

Definition at line 1371 of file IO_Pin.h.

7.15.2.83 #define IO_DI_05 IO_PIN_183

alternative digital input function for [IO_PWM_05](#)

Definition at line 1374 of file IO_Pin.h.

7.15.2.84 #define IO_DI_06 IO_PIN_186

alternative digital input function for [IO_PWM_06](#)

Definition at line 1377 of file IO_Pin.h.

7.15.2.85 #define IO_DI_07 IO_PIN_162

alternative digital input function for [IO_PWM_07](#)

Definition at line 1380 of file IO_Pin.h.

7.15.2.86 #define IO_DI_08 IO_PIN_189

alternative digital input function for [IO_PWM_08](#)

Definition at line 1383 of file IO_Pin.h.

7.15.2.87 #define IO_DI_09 IO_PIN_165

alternative digital input function for [IO_PWM_09](#)

Definition at line 1386 of file IO_Pin.h.

7.15.2.88 #define IO_DI_10 IO_PIN_192

alternative digital input function for [IO_PWM_10](#)

Definition at line 1389 of file IO_Pin.h.

7.15.2.89 #define IO_DI_11 IO_PIN_168

alternative digital input function for [IO_PWM_11](#)

Definition at line 1392 of file IO_Pin.h.

7.15.2.90 #define IO_DI_12 IO_PIN_195

alternative digital input function for [IO_PWM_12](#)

Definition at line 1395 of file IO_Pin.h.

7.15.2.91 #define IO_DI_13 IO_PIN_171

alternative digital input function for [IO_PWM_13](#)

Definition at line 1398 of file IO_Pin.h.

7.15.2.92 #define IO_DI_14 IO_PIN_154

alternative digital input function for [IO_PWM_14](#)

Definition at line 1401 of file IO_Pin.h.

7.15.2.93 #define IO_DI_15 IO_PIN_178

alternative digital input function for [IO_PWM_15](#)

Definition at line 1404 of file IO_Pin.h.

7.15.2.94 #define IO_DI_16 IO_PIN_157

alternative digital input function for [IO_PWM_16](#)

Definition at line 1407 of file IO_Pin.h.

7.15.2.95 #define IO_DI_17 IO_PIN_181

alternative digital input function for [IO_PWM_17](#)

Definition at line 1410 of file IO_Pin.h.

7.15.2.96 #define IO_DI_18 IO_PIN_160

alternative digital input function for [IO_PWM_18](#)

Definition at line 1413 of file IO_Pin.h.

7.15.2.97 #define IO_DI_19 IO_PIN_184

alternative digital input function for [IO_PWM_19](#)

Definition at line 1416 of file IO_Pin.h.

7.15.2.98 #define IO_DI_20 IO_PIN_187

alternative digital input function for [IO_PWM_20](#)

Definition at line 1419 of file IO_Pin.h.

7.15.2.99 #define IO_DI_21 IO_PIN_163

alternative digital input function for [IO_PWM_21](#)

Definition at line 1422 of file IO_Pin.h.

7.15.2.100 #define IO_DI_22 IO_PIN_190

alternative digital input function for [IO_PWM_22](#)

Definition at line 1425 of file IO_Pin.h.

7.15.2.101 #define IO_DI_23 IO_PIN_166

alternative digital input function for [IO_PWM_23](#)

Definition at line 1428 of file IO_Pin.h.

7.15.2.102 #define IO_DI_24 IO_PIN_193

alternative digital input function for [IO_PWM_24](#)

Definition at line 1431 of file IO_Pin.h.

7.15.2.103 #define IO_DI_25 IO_PIN_169

alternative digital input function for [IO_PWM_25](#)

Definition at line 1434 of file IO_Pin.h.

7.15.2.104 #define IO_DI_26 IO_PIN_196

alternative digital input function for [IO_PWM_26](#)

Definition at line 1437 of file IO_Pin.h.

7.15.2.105 #define IO_DI_27 IO_PIN_172

alternative digital input function for [IO_PWM_27](#)

Definition at line 1440 of file IO_Pin.h.

7.15.2.106 #define IO_DI_28 IO_PIN_101

alternative digital input function for [IO_PWM_28](#)

Definition at line 1443 of file IO_Pin.h.

7.15.2.107 #define IO_DI_29 IO_PIN_125

alternative digital input function for [IO_PWM_29](#)

Definition at line 1446 of file IO_Pin.h.

7.15.2.108 #define IO_DI_30 IO_PIN_150

alternative digital input function for [IO_PWM_30](#)

Definition at line 1449 of file IO_Pin.h.

7.15.2.109 #define IO_DI_31 IO_PIN_174

alternative digital input function for [IO_PWM_31](#)

Definition at line 1452 of file IO_Pin.h.

7.15.2.110 #define IO_DI_32 IO_PIN_102

alternative digital input function for [IO_PWM_32](#)

Definition at line 1455 of file IO_Pin.h.

7.15.2.111 #define IO_DI_33 IO_PIN_126

alternative digital input function for [IO_PWM_33](#)

Definition at line 1458 of file IO_Pin.h.

7.15.2.112 #define IO_DI_34 IO_PIN_151

alternative digital input function for [IO_PWM_34](#)

Definition at line 1461 of file IO_Pin.h.

7.15.2.113 #define IO_DI_35 IO_PIN_175

alternative digital input function for [IO_PWM_35](#)

Definition at line 1464 of file IO_Pin.h.

7.15.2.114 #define IO_DI_36 IO_PIN_115

alternative digital input function for [IO_PWD_00](#)

Definition at line 1468 of file IO_Pin.h.

7.15.2.115 #define IO_DI_37 IO_PIN_139

alternative digital input function for [IO_PWD_01](#)

Definition at line 1471 of file IO_Pin.h.

7.15.2.116 #define IO_DI_38 IO_PIN_116

alternative digital input function for [IO_PWD_02](#)

Definition at line 1474 of file IO_Pin.h.

7.15.2.117 #define IO_DI_39 IO_PIN_140

alternative digital input function for [IO_PWD_03](#)

Definition at line 1477 of file IO_Pin.h.

7.15.2.118 #define IO_DI_40 IO_PIN_117

alternative digital input function for [IO_PWD_04](#)

Definition at line 1480 of file IO_Pin.h.

7.15.2.119 #define IO_DI_41 IO_PIN_141

alternative digital input function for [IO_PWD_05](#)

Definition at line 1483 of file IO_Pin.h.

7.15.2.120 #define IO_DI_42 IO_PIN_122

alternative digital input function for [IO_PWD_06](#)

Definition at line 1486 of file IO_Pin.h.

7.15.2.121 #define IO_DI_43 IO_PIN_146

alternative digital input function for [IO_PWD_07](#)

Definition at line 1489 of file IO_Pin.h.

7.15.2.122 #define IO_DI_44 IO_PIN_123

alternative digital input function for [IO_PWD_08](#)

Definition at line 1492 of file IO_Pin.h.

7.15.2.123 #define IO_DI_45 IO_PIN_147

alternative digital input function for [IO_PWD_09](#)

Definition at line 1495 of file IO_Pin.h.

7.15.2.124 #define IO_DI_46 IO_PIN_124

alternative digital input function for [IO_PWD_10](#)

Definition at line 1498 of file IO_Pin.h.

7.15.2.125 #define IO_DI_47 IO_PIN_148

alternative digital input function for [IO_PWD_11](#)

Definition at line 1501 of file IO_Pin.h.

7.15.2.126 #define IO_DI_48 IO_PIN_103

alternative digital input function for [IO_ADC_00](#)

Definition at line 1505 of file IO_Pin.h.

7.15.2.127 #define IO_DI_49 IO_PIN_127

alternative digital input function for [IO_ADC_01](#)

Definition at line 1508 of file IO_Pin.h.

7.15.2.128 #define IO_DI_50 IO_PIN_104

alternative digital input function for [IO_ADC_02](#)

Definition at line 1511 of file IO_Pin.h.

7.15.2.129 #define IO_DI_51 IO_PIN_128

alternative digital input function for [IO_ADC_03](#)

Definition at line 1514 of file IO_Pin.h.

7.15.2.130 #define IO_DI_52 IO_PIN_105

alternative digital input function for [IO_ADC_04](#)

Definition at line 1517 of file IO_Pin.h.

7.15.2.131 #define IO_DI_53 IO_PIN_129

alternative digital input function for [IO_ADC_05](#)

Definition at line 1520 of file IO_Pin.h.

7.15.2.132 #define IO_DI_54 IO_PIN_106

alternative digital input function for [IO_ADC_06](#)

Definition at line 1523 of file IO_Pin.h.

7.15.2.133 #define IO_DI_55 IO_PIN_130

alternative digital input function for [IO_ADC_07](#)

Definition at line 1526 of file IO_Pin.h.

7.15.2.134 #define IO_DI_56 IO_PIN_107

alternative digital input function for [IO_ADC_08](#)

Definition at line 1530 of file IO_Pin.h.

7.15.2.135 #define IO_DI_57 IO_PIN_131

alternative digital input function for [IO_ADC_09](#)

Definition at line 1533 of file IO_Pin.h.

7.15.2.136 #define IO_DI_58 IO_PIN_108

alternative digital input function for [IO_ADC_10](#)

Definition at line 1536 of file IO_Pin.h.

7.15.2.137 #define IO_DI_59 IO_PIN_132

alternative digital input function for [IO_ADC_11](#)

Definition at line 1539 of file IO_Pin.h.

7.15.2.138 #define IO_DI_60 IO_PIN_109

alternative digital input function for [IO_ADC_12](#)

Definition at line 1542 of file IO_Pin.h.

7.15.2.139 #define IO_DI_61 IO_PIN_133

alternative digital input function for [IO_ADC_13](#)

Definition at line 1545 of file IO_Pin.h.

7.15.2.140 #define IO_DI_62 IO_PIN_110

alternative digital input function for [IO_ADC_14](#)

Definition at line 1548 of file IO_Pin.h.

7.15.2.141 #define IO_DI_63 IO_PIN_134

alternative digital input function for [IO_ADC_15](#)

Definition at line 1551 of file IO_Pin.h.

7.15.2.142 #define IO_DI_64 IO_PIN_111

alternative digital input function for [IO_ADC_16](#)

Definition at line 1555 of file IO_Pin.h.

7.15.2.143 #define IO_DI_65 IO_PIN_135

alternative digital input function for [IO_ADC_17](#)

Definition at line 1558 of file IO_Pin.h.

7.15.2.144 #define IO_DI_66 IO_PIN_112

alternative digital input function for [IO_ADC_18](#)

Definition at line 1561 of file IO_Pin.h.

7.15.2.145 #define IO_DI_67 IO_PIN_136

alternative digital input function for [IO_ADC_19](#)

Definition at line 1564 of file IO_Pin.h.

7.15.2.146 #define IO_DI_68 IO_PIN_113

alternative digital input function for [IO_ADC_20](#)

Definition at line 1567 of file IO_Pin.h.

7.15.2.147 #define IO_DI_69 IO_PIN_137

alternative digital input function for [IO_ADC_21](#)

Definition at line 1570 of file IO_Pin.h.

7.15.2.148 #define IO_DI_70 IO_PIN_114

alternative digital input function for [IO_ADC_22](#)

Definition at line 1573 of file IO_Pin.h.

7.15.2.149 #define IO_DI_71 IO_PIN_138

alternative digital input function for [IO_ADC_23](#)

Definition at line 1576 of file IO_Pin.h.

7.15.2.150 #define IO_DI_72 IO_PIN_149

alternative digital input function for [IO_DO_00](#)

Definition at line 1580 of file IO_Pin.h.

7.15.2.151 #define IO_DI_73 IO_PIN_173

alternative digital input function for [IO_DO_01](#)

Definition at line 1583 of file IO_Pin.h.

7.15.2.152 #define IO_DI_74 IO_PIN_152

alternative digital input function for [IO_DO_02](#)

Definition at line 1586 of file IO_Pin.h.

7.15.2.153 #define IO_DI_75 IO_PIN_176

alternative digital input function for [IO_DO_03](#)

Definition at line 1589 of file IO_Pin.h.

7.15.2.154 #define IO_DI_76 IO_PIN_155

alternative digital input function for [IO_DO_04](#)

Definition at line 1592 of file IO_Pin.h.

7.15.2.155 #define IO_DI_77 IO_PIN_179

alternative digital input function for [IO_DO_05](#)

Definition at line 1595 of file IO_Pin.h.

7.15.2.156 #define IO_DI_78 IO_PIN_158

alternative digital input function for [IO_DO_06](#)

Definition at line 1598 of file IO_Pin.h.

7.15.2.157 #define IO_DI_79 IO_PIN_182

alternative digital input function for [IO_DO_07](#)

Definition at line 1601 of file IO_Pin.h.

7.15.2.158 #define IO_DI_80 IO_PIN_251

alternative digital input function for [IO_DO_08](#)

Definition at line 1605 of file IO_Pin.h.

7.15.2.159 #define IO_DI_81 IO_PIN_238

alternative digital input function for [IO_DO_09](#)

Definition at line 1608 of file IO_Pin.h.

7.15.2.160 #define IO_DI_82 IO_PIN_252

alternative digital input function for [IO_DO_10](#)

Definition at line 1611 of file IO_Pin.h.

7.15.2.161 #define IO_DI_83 IO_PIN_239

alternative digital input function for [IO_DO_11](#)

Definition at line 1614 of file IO_Pin.h.

7.15.2.162 #define IO_DI_84 IO_PIN_253

alternative digital input function for [IO_DO_12](#)

Definition at line 1617 of file IO_Pin.h.

7.15.2.163 #define IO_DI_85 IO_PIN_240

alternative digital input function for [IO_DO_13](#)

Definition at line 1620 of file IO_Pin.h.

7.15.2.164 #define IO_DI_86 IO_PIN_254

alternative digital input function for [IO_DO_14](#)

Definition at line 1623 of file IO_Pin.h.

7.15.2.165 #define IO_DI_87 IO_PIN_241

alternative digital input function for [IO_DO_15](#)

Definition at line 1626 of file IO_Pin.h.

7.15.2.166 #define IO_DI_88 IO_PIN_161

alternative digital input function for [IO_PVG_00](#)

Definition at line 1630 of file IO_Pin.h.

7.15.2.167 #define IO_DI_89 IO_PIN_185

alternative digital input function for [IO_PVG_01](#)

Definition at line 1633 of file IO_Pin.h.

7.15.2.168 #define IO_DI_90 IO_PIN_188

alternative digital input function for [IO_PVG_02](#)

Definition at line 1636 of file IO_Pin.h.

7.15.2.169 #define IO_DI_91 IO_PIN_164

alternative digital input function for [IO_PVG_03](#)

Definition at line 1639 of file IO_Pin.h.

7.15.2.170 #define IO_DI_92 IO_PIN_191

alternative digital input function for [IO_PVG_04](#)

Definition at line 1642 of file IO_Pin.h.

7.15.2.171 #define IO_DI_93 IO_PIN_167

alternative digital input function for [IO_PVG_05](#)

Definition at line 1645 of file IO_Pin.h.

7.15.2.172 #define IO_DI_94 IO_PIN_194

alternative digital input function for [IO_PVG_06](#)

Definition at line 1648 of file IO_Pin.h.

7.15.2.173 #define IO_DI_95 IO_PIN_170

alternative digital input function for [IO_PVG_07](#)

Definition at line 1651 of file IO_Pin.h.

7.15.2.174 #define IO_DO_00 IO_PIN_149

main function high side digital output

Definition at line 1656 of file IO_Pin.h.

7.15.2.175 #define IO_DO_01 IO_PIN_173

main function high side digital output

Definition at line 1657 of file IO_Pin.h.

7.15.2.176 #define IO_DO_02 IO_PIN_152

main function high side digital output

Definition at line 1658 of file IO_Pin.h.

7.15.2.177 #define IO_DO_03 IO_PIN_176

main function high side digital output

Definition at line 1659 of file IO_Pin.h.

7.15.2.178 #define IO_DO_04 IO_PIN_155

main function high side digital output

Definition at line 1660 of file IO_Pin.h.

7.15.2.179 #define IO_DO_05 IO_PIN_179

main function high side digital output

Definition at line 1661 of file IO_Pin.h.

7.15.2.180 #define IO_DO_06 IO_PIN_158

main function high side digital output

Definition at line 1662 of file IO_Pin.h.

7.15.2.181 #define IO_DO_07 IO_PIN_182

main function high side digital output

Definition at line 1663 of file IO_Pin.h.

7.15.2.182 #define IO_DO_08 IO_PIN_251

main function low side digital output

Definition at line 1665 of file IO_Pin.h.

7.15.2.183 #define IO_DO_09 IO_PIN_238

main function low side digital output

Definition at line 1666 of file IO_Pin.h.

7.15.2.184 #define IO_DO_10 IO_PIN_252

main function low side digital output

Definition at line 1667 of file IO_Pin.h.

7.15.2.185 #define IO_DO_11 IO_PIN_239

main function low side digital output

Definition at line 1668 of file IO_Pin.h.

7.15.2.186 #define IO_DO_12 IO_PIN_253

main function low side digital output

Definition at line 1669 of file IO_Pin.h.

7.15.2.187 #define IO_DO_13 IO_PIN_240

main function low side digital output

Definition at line 1670 of file IO_Pin.h.

7.15.2.188 #define IO_DO_14 IO_PIN_254

main function low side digital output

Definition at line 1671 of file IO_Pin.h.

7.15.2.189 #define IO_DO_15 IO_PIN_241

main function low side digital output

Definition at line 1672 of file IO_Pin.h.

7.15.2.190 #define IO_DO_16 IO_PIN_153

alternative digital output function for [IO_PWM_00](#) (shut off group 0)

Definition at line 1674 of file IO_Pin.h.

7.15.2.191 #define IO_DO_17 IO_PIN_177

alternative digital output function for [IO_PWM_01](#) (shut off group 0)

Definition at line 1677 of file IO_Pin.h.

7.15.2.192 #define IO_DO_18 IO_PIN_156

alternative digital output function for [IO_PWM_02](#) (shut off group 0)

Definition at line 1680 of file IO_Pin.h.

7.15.2.193 #define IO_DO_19 IO_PIN_180

alternative digital output function for [IO_PWM_03](#) (shut off group 0)
Definition at line 1683 of file IO_Pin.h.

7.15.2.194 #define IO_DO_20 IO_PIN_159

alternative digital output function for [IO_PWM_04](#) (shut off group 0)
Definition at line 1686 of file IO_Pin.h.

7.15.2.195 #define IO_DO_21 IO_PIN_183

alternative digital output function for [IO_PWM_05](#) (shut off group 0)
Definition at line 1689 of file IO_Pin.h.

7.15.2.196 #define IO_DO_22 IO_PIN_186

alternative digital output function for [IO_PWM_06](#) (shut off group 0)
Definition at line 1692 of file IO_Pin.h.

7.15.2.197 #define IO_DO_23 IO_PIN_162

alternative digital output function for [IO_PWM_07](#) (shut off group 0)
Definition at line 1695 of file IO_Pin.h.

7.15.2.198 #define IO_DO_24 IO_PIN_189

alternative digital output function for [IO_PWM_08](#) (shut off group 0)
Definition at line 1698 of file IO_Pin.h.

7.15.2.199 #define IO_DO_25 IO_PIN_165

alternative digital output function for [IO_PWM_09](#) (shut off group 0)
Definition at line 1701 of file IO_Pin.h.

7.15.2.200 #define IO_DO_26 IO_PIN_192

alternative digital output function for [IO_PWM_10](#) (shut off group 0)
Definition at line 1704 of file IO_Pin.h.

7.15.2.201 #define IO_DO_27 IO_PIN_168

alternative digital output function for [IO_PWM_11](#) (shut off group 0)
Definition at line 1707 of file IO_Pin.h.

7.15.2.202 #define IO_DO_28 IO_PIN_195

alternative digital output function for [IO_PWM_12](#) (shut off group 0)
Definition at line 1710 of file IO_Pin.h.

7.15.2.203 #define IO_DO_29 IO_PIN_171

alternative digital output function for [IO_PWM_13](#) (shut off group 0)
Definition at line 1713 of file IO_Pin.h.

7.15.2.204 #define IO_DO_30 IO_PIN_154

alternative digital output function for [IO_PWM_14](#) (shut off group 1)
Definition at line 1716 of file IO_Pin.h.

7.15.2.205 #define IO_DO_31 IO_PIN_178

alternative digital output function for [IO_PWM_15](#) (shut off group 1)
Definition at line 1719 of file IO_Pin.h.

7.15.2.206 #define IO_DO_32 IO_PIN_157

alternative digital output function for [IO_PWM_16](#) (shut off group 1)
Definition at line 1722 of file IO_Pin.h.

7.15.2.207 #define IO_DO_33 IO_PIN_181

alternative digital output function for [IO_PWM_17](#) (shut off group 1)
Definition at line 1725 of file IO_Pin.h.

7.15.2.208 #define IO_DO_34 IO_PIN_160

alternative digital output function for [IO_PWM_18](#) (shut off group 1)
Definition at line 1728 of file IO_Pin.h.

7.15.2.209 #define IO_DO_35 IO_PIN_184

alternative digital output function for [IO_PWM_19](#) (shut off group 1)
Definition at line 1731 of file IO_Pin.h.

7.15.2.210 #define IO_DO_36 IO_PIN_187

alternative digital output function for [IO_PWM_20](#) (shut off group 1)
Definition at line 1734 of file IO_Pin.h.

7.15.2.211 #define IO_DO_37 IO_PIN_163

alternative digital output function for [IO_PWM_21](#) (shut off group 1)
Definition at line 1737 of file IO_Pin.h.

7.15.2.212 #define IO_DO_38 IO_PIN_190

alternative digital output function for [IO_PWM_22](#) (shut off group 1)
Definition at line 1740 of file IO_Pin.h.

7.15.2.213 #define IO_DO_39 IO_PIN_166

alternative digital output function for [IO_PWM_23](#) (shut off group 1)
Definition at line 1743 of file IO_Pin.h.

7.15.2.214 #define IO_DO_40 IO_PIN_193

alternative digital output function for [IO_PWM_24](#) (shut off group 1)
Definition at line 1746 of file IO_Pin.h.

7.15.2.215 #define IO_DO_41 IO_PIN_169

alternative digital output function for [IO_PWM_25](#) (shut off group 1)
Definition at line 1749 of file IO_Pin.h.

7.15.2.216 #define IO_DO_42 IO_PIN_196

alternative digital output function for [IO_PWM_26](#) (shut off group 1)
Definition at line 1752 of file IO_Pin.h.

7.15.2.217 #define IO_DO_43 IO_PIN_172

alternative digital output function for [IO_PWM_27](#) (shut off group 1)
Definition at line 1755 of file IO_Pin.h.

7.15.2.218 #define IO_DO_44 IO_PIN_101

alternative digital output function for [IO_PWM_28](#) (shut off group 2)
Definition at line 1758 of file IO_Pin.h.

7.15.2.219 #define IO_DO_45 IO_PIN_125

alternative digital output function for [IO_PWM_29](#) (shut off group 2)
Definition at line 1761 of file IO_Pin.h.

7.15.2.220 #define IO_DO_46 IO_PIN_150

alternative digital output function for [IO_PWM_30](#) (shut off group 2)
Definition at line 1764 of file IO_Pin.h.

7.15.2.221 #define IO_DO_47 IO_PIN_174

alternative digital output function for [IO_PWM_31](#) (shut off group 2)
Definition at line 1767 of file IO_Pin.h.

7.15.2.222 #define IO_DO_48 IO_PIN_102

alternative digital output function for [IO_PWM_32](#) (shut off group 2)
Definition at line 1770 of file IO_Pin.h.

7.15.2.223 #define IO_DO_49 IO_PIN_126

alternative digital output function for [IO_PWM_33](#) (shut off group 2)
Definition at line 1773 of file IO_Pin.h.

7.15.2.224 #define IO_DO_50 IO_PIN_151

alternative digital output function for [IO_PWM_34](#) (shut off group 2)
Definition at line 1776 of file IO_Pin.h.

7.15.2.225 #define IO_DO_51 IO_PIN_175

alternative digital output function for [IO_PWM_35](#) (shut off group 2)

Definition at line 1779 of file IO_Pin.h.

7.15.2.226 #define IO_DO_52 IO_PIN_161

alternative digital output function for [IO_PVG_00](#)

Definition at line 1783 of file IO_Pin.h.

7.15.2.227 #define IO_DO_53 IO_PIN_185

alternative digital output function for [IO_PVG_01](#)

Definition at line 1786 of file IO_Pin.h.

7.15.2.228 #define IO_DO_54 IO_PIN_188

alternative digital output function for [IO_PVG_02](#)

Definition at line 1789 of file IO_Pin.h.

7.15.2.229 #define IO_DO_55 IO_PIN_164

alternative digital output function for [IO_PVG_03](#)

Definition at line 1792 of file IO_Pin.h.

7.15.2.230 #define IO_DO_56 IO_PIN_191

alternative digital output function for [IO_PVG_04](#)

Definition at line 1795 of file IO_Pin.h.

7.15.2.231 #define IO_DO_57 IO_PIN_167

alternative digital output function for [IO_PVG_05](#)

Definition at line 1798 of file IO_Pin.h.

7.15.2.232 #define IO_DO_58 IO_PIN_194

alternative digital output function for [IO_PVG_06](#)

Definition at line 1801 of file IO_Pin.h.

7.15.2.233 #define IO_DO_59 IO_PIN_170

alternative digital output function for [IO_PVG_07](#)

Definition at line 1804 of file IO_Pin.h.

7.15.2.234 #define IO_GND IO_PIN_118

Definition at line 1259 of file IO_Pin.h.

7.15.2.235 #define IO_INT_PIN_1V2 122U

internal 1.2V supply voltage

Definition at line 1143 of file IO_Pin.h.

7.15.2.236 #define IO_INT_PIN_CAN_CH0 102U

internal for CAN Channel0

Definition at line 1120 of file IO_Pin.h.

7.15.2.237 #define IO_INT_PIN_CAN_CH1 103U

internal for CAN Channel1

Definition at line 1121 of file IO_Pin.h.

7.15.2.238 #define IO_INT_PIN_CAN_CH2 104U

internal for CAN Channel2

Definition at line 1122 of file IO_Pin.h.

7.15.2.239 #define IO_INT_PIN_CAN_CH3 105U

internal for CAN Channel3

Definition at line 1123 of file IO_Pin.h.

7.15.2.240 #define IO_INT_PIN_CAN_CH4 106U

internal for CAN Channel4

Definition at line 1124 of file IO_Pin.h.

7.15.2.241 #define IO_INT_PIN_CAN_CH5 107U

internal for CAN Channel5

Definition at line 1125 of file IO_Pin.h.

7.15.2.242 #define IO_INT_PIN_CAN_CH6 108U

internal for CAN Channel6

Definition at line 1126 of file IO_Pin.h.

7.15.2.243 #define IO_INT_PIN_ENDRV_CPU 113U

internal Pin for enable drive CPU, see [Driver for ECU Power functions](#) for details

Definition at line 1137 of file IO_Pin.h.

7.15.2.244 #define IO_INT_PIN_PWD 117U

internal for PWD module

Definition at line 1140 of file IO_Pin.h.

7.15.2.245 #define IO_INT_PIN_REF_2V5 121U

internal 2.5V reference voltage

Definition at line 1142 of file IO_Pin.h.

7.15.2.246 #define IO_INT_PIN_SAFETY_SW_0 109U

internal Pin for Safety Switch 0, see [Driver for ECU Power functions](#) for details

Definition at line 1127 of file IO_Pin.h.

7.15.2.247 #define IO_INT_PIN_SAFETY_SW_1 110U

internal Pin for Safety Switch 1, see [Driver for ECU Power functions](#) for details

Definition at line 1130 of file IO_Pin.h.

7.15.2.248 #define IO_INT_PIN_SAFETY_SW_2 111U

internal Pin for Safety Switch 2, see [Driver for ECU Power functions](#) for details

Definition at line 1133 of file IO_Pin.h.

7.15.2.249 #define IO_INT_PIN_SAFETY_SW_VP 112U

internal Pin for Safety Switch VP

Definition at line 1136 of file IO_Pin.h.

7.15.2.250 #define IO_INT_PIN_TEMP 120U

internal for board temperature

Definition at line 1141 of file IO_Pin.h.

7.15.2.251 #define IO_INT_PIN_VMON 123U

internal for voltage monitor

Definition at line 1144 of file IO_Pin.h.

7.15.2.252 #define IO_INT_POWERSTAGE_ENABLE IO_INT_PIN_ENDRV_CPU

Internal Pin for enable drive CPU

Definition at line 1832 of file IO_Pin.h.

7.15.2.253 #define IO_INT_SAFETY_SW_0 IO_INT_PIN_SAFETY_SW_0

Internal Pin for safety switch 0

Definition at line 1829 of file IO_Pin.h.

7.15.2.254 #define IO_INT_SAFETY_SW_1 IO_INT_PIN_SAFETY_SW_1

Internal Pin for safety switch 1

Definition at line 1830 of file IO_Pin.h.

7.15.2.255 #define IO_INT_SAFETY_SW_2 IO_INT_PIN_SAFETY_SW_2

Internal Pin for safety switch 2

Definition at line 1831 of file IO_Pin.h.

7.15.2.256 #define IO_K15 IO_PIN_207

Definition at line 1256 of file IO_Pin.h.

7.15.2.257 #define IO_PIN_101 80U

Pin Nr. 101

Definition at line 1005 of file IO_Pin.h.

7.15.2.258 #define IO_PIN_102 84U

Pin Nr. 102

Definition at line 1006 of file IO_Pin.h.

7.15.2.259 #define IO_PIN_103 0U

Pin Nr. 103

Definition at line 1007 of file IO_Pin.h.

7.15.2.260 #define IO_PIN_104 2U

Pin Nr. 104

Definition at line 1008 of file IO_Pin.h.

7.15.2.261 #define IO_PIN_105 4U

Pin Nr. 105

Definition at line 1009 of file IO_Pin.h.

7.15.2.262 #define IO_PIN_106 6U

Pin Nr. 106

Definition at line 1010 of file IO_Pin.h.

7.15.2.263 #define IO_PIN_107 8U

Pin Nr. 107

Definition at line 1011 of file IO_Pin.h.

7.15.2.264 #define IO_PIN_108 10U

Pin Nr. 108

Definition at line 1012 of file IO_Pin.h.

7.15.2.265 #define IO_PIN_109 12U

Pin Nr. 109

Definition at line 1013 of file IO_Pin.h.

7.15.2.266 #define IO_PIN_110 14U

Pin Nr. 110

Definition at line 1014 of file IO_Pin.h.

7.15.2.267 #define IO_PIN_111 16U

Pin Nr. 111

Definition at line 1015 of file IO_Pin.h.

7.15.2.268 #define IO_PIN_112 18U

Pin Nr. 112

Definition at line 1016 of file IO_Pin.h.

7.15.2.269 #define IO_PIN_113 20U

Pin Nr. 113

Definition at line 1017 of file IO_Pin.h.

7.15.2.270 #define IO_PIN_114 22U

Pin Nr. 114

Definition at line 1018 of file IO_Pin.h.

7.15.2.271 #define IO_PIN_115 24U

Pin Nr. 115

Definition at line 1019 of file IO_Pin.h.

7.15.2.272 #define IO_PIN_116 26U

Pin Nr. 116

Definition at line 1020 of file IO_Pin.h.

7.15.2.273 #define IO_PIN_117 28U

Pin Nr. 117

Definition at line 1021 of file IO_Pin.h.

7.15.2.274 #define IO_PIN_118 253U

Pin Nr. 118 (BAT-, ground)

Definition at line 1107 of file IO_Pin.h.

7.15.2.275 #define IO_PIN_122 30U

Pin Nr. 122

Definition at line 1022 of file IO_Pin.h.

7.15.2.276 #define IO_PIN_123 32U

Pin Nr. 123

Definition at line 1023 of file IO_Pin.h.

7.15.2.277 #define IO_PIN_124 34U

Pin Nr. 124

Definition at line 1024 of file IO_Pin.h.

7.15.2.278 #define IO_PIN_125 81U

Pin Nr. 125

Definition at line 1025 of file IO_Pin.h.

7.15.2.279 #define IO_PIN_126 85U

Pin Nr. 126

Definition at line 1026 of file IO_Pin.h.

7.15.2.280 #define IO_PIN_127 1U

Pin Nr. 127

Definition at line 1027 of file IO_Pin.h.

7.15.2.281 #define IO_PIN_128 3U

Pin Nr. 128

Definition at line 1028 of file IO_Pin.h.

7.15.2.282 #define IO_PIN_129 5U

Pin Nr. 129

Definition at line 1029 of file IO_Pin.h.

7.15.2.283 #define IO_PIN_130 7U

Pin Nr. 130

Definition at line 1030 of file IO_Pin.h.

7.15.2.284 #define IO_PIN_131 9U

Pin Nr. 131

Definition at line 1031 of file IO_Pin.h.

7.15.2.285 #define IO_PIN_132 11U

Pin Nr. 132

Definition at line 1032 of file IO_Pin.h.

7.15.2.286 #define IO_PIN_133 13U

Pin Nr. 133

Definition at line 1033 of file IO_Pin.h.

7.15.2.287 #define IO_PIN_134 15U

Pin Nr. 134

Definition at line 1034 of file IO_Pin.h.

7.15.2.288 #define IO_PIN_135 17U

Pin Nr. 135

Definition at line 1035 of file IO_Pin.h.

7.15.2.289 #define IO_PIN_136 19U

Pin Nr. 136

Definition at line 1036 of file IO_Pin.h.

7.15.2.290 #define IO_PIN_137 21U

Pin Nr. 137

Definition at line 1037 of file IO_Pin.h.

7.15.2.291 #define IO_PIN_138 23U

Pin Nr. 138

Definition at line 1038 of file IO_Pin.h.

7.15.2.292 #define IO_PIN_139 25U

Pin Nr. 139

Definition at line 1039 of file IO_Pin.h.

7.15.2.293 #define IO_PIN_140 27U

Pin Nr. 140

Definition at line 1040 of file IO_Pin.h.

7.15.2.294 #define IO_PIN_141 29U

Pin Nr. 141

Definition at line 1041 of file IO_Pin.h.

7.15.2.295 #define IO_PIN_146 31U

Pin Nr. 146

Definition at line 1042 of file IO_Pin.h.

7.15.2.296 #define IO_PIN_147 33U

Pin Nr. 147

Definition at line 1043 of file IO_Pin.h.

7.15.2.297 #define IO_PIN_148 35U

Pin Nr. 148

Definition at line 1044 of file IO_Pin.h.

7.15.2.298 #define IO_PIN_149 36U

Pin Nr. 149

Definition at line 1045 of file IO_Pin.h.

7.15.2.299 #define IO_PIN_150 82U

Pin Nr. 150

Definition at line 1046 of file IO_Pin.h.

7.15.2.300 #define IO_PIN_151 86U

Pin Nr. 151

Definition at line 1047 of file IO_Pin.h.

7.15.2.301 #define IO_PIN_152 38U

Pin Nr. 152

Definition at line 1048 of file IO_Pin.h.

7.15.2.302 #define IO_PIN_153 52U

Pin Nr. 153

Definition at line 1049 of file IO_Pin.h.

7.15.2.303 #define IO_PIN_154 66U

Pin Nr. 154

Definition at line 1050 of file IO_Pin.h.

7.15.2.304 #define IO_PIN_155 40U

Pin Nr. 155

Definition at line 1051 of file IO_Pin.h.

7.15.2.305 #define IO_PIN_156 54U

Pin Nr. 156

Definition at line 1052 of file IO_Pin.h.

7.15.2.306 #define IO_PIN_157 68U

Pin Nr. 157

Definition at line 1053 of file IO_Pin.h.

7.15.2.307 #define IO_PIN_158 42U

Pin Nr. 158

Definition at line 1054 of file IO_Pin.h.

7.15.2.308 #define IO_PIN_159 56U

Pin Nr. 159

Definition at line 1055 of file IO_Pin.h.

7.15.2.309 #define IO_PIN_160 70U

Pin Nr. 160

Definition at line 1056 of file IO_Pin.h.

7.15.2.310 #define IO_PIN_161 88U

Pin Nr. 161

Definition at line 1057 of file IO_Pin.h.

7.15.2.311 #define IO_PIN_162 59U

Pin Nr. 162

Definition at line 1058 of file IO_Pin.h.

7.15.2.312 #define IO_PIN_163 73U

Pin Nr. 163

Definition at line 1059 of file IO_Pin.h.

7.15.2.313 #define IO_PIN_164 91U

Pin Nr. 164

Definition at line 1060 of file IO_Pin.h.

7.15.2.314 #define IO_PIN_165 61U

Pin Nr. 165

Definition at line 1061 of file IO_Pin.h.

7.15.2.315 #define IO_PIN_166 75U

Pin Nr. 166

Definition at line 1062 of file IO_Pin.h.

7.15.2.316 #define IO_PIN_167 93U

Pin Nr. 167

Definition at line 1063 of file IO_Pin.h.

7.15.2.317 #define IO_PIN_168 63U

Pin Nr. 168

Definition at line 1064 of file IO_Pin.h.

7.15.2.318 #define IO_PIN_169 77U

Pin Nr. 169

Definition at line 1065 of file IO_Pin.h.

7.15.2.319 #define IO_PIN_170 95U

Pin Nr. 170

Definition at line 1066 of file IO_Pin.h.

7.15.2.320 #define IO_PIN_171 65U

Pin Nr. 171

Definition at line 1067 of file IO_Pin.h.

7.15.2.321 #define IO_PIN_172 79U

Pin Nr. 172

Definition at line 1068 of file IO_Pin.h.

7.15.2.322 #define IO_PIN_173 37U

Pin Nr. 173

Definition at line 1069 of file IO_Pin.h.

7.15.2.323 #define IO_PIN_174 83U

Pin Nr. 174

Definition at line 1070 of file IO_Pin.h.

7.15.2.324 #define IO_PIN_175 87U

Pin Nr. 175

Definition at line 1071 of file IO_Pin.h.

7.15.2.325 #define IO_PIN_176 39U

Pin Nr. 176

Definition at line 1072 of file IO_Pin.h.

7.15.2.326 #define IO_PIN_177 53U

Pin Nr. 177

Definition at line 1073 of file IO_Pin.h.

7.15.2.327 #define IO_PIN_178 67U

Pin Nr. 178

Definition at line 1074 of file IO_Pin.h.

7.15.2.328 #define IO_PIN_179 41U

Pin Nr. 179

Definition at line 1075 of file IO_Pin.h.

7.15.2.329 #define IO_PIN_180 55U

Pin Nr. 180

Definition at line 1076 of file IO_Pin.h.

7.15.2.330 #define IO_PIN_181 69U

Pin Nr. 181

Definition at line 1077 of file IO_Pin.h.

7.15.2.331 #define IO_PIN_182 43U

Pin Nr. 182

Definition at line 1078 of file IO_Pin.h.

7.15.2.332 #define IO_PIN_183 57U

Pin Nr. 183

Definition at line 1079 of file IO_Pin.h.

7.15.2.333 #define IO_PIN_184 71U

Pin Nr. 184

Definition at line 1080 of file IO_Pin.h.

7.15.2.334 #define IO_PIN_185 89U

Pin Nr. 185

Definition at line 1081 of file IO_Pin.h.

7.15.2.335 #define IO_PIN_186 58U

Pin Nr. 186

Definition at line 1082 of file IO_Pin.h.

7.15.2.336 #define IO_PIN_187 72U

Pin Nr. 187

Definition at line 1083 of file IO_Pin.h.

7.15.2.337 #define IO_PIN_188 90U

Pin Nr. 188

Definition at line 1084 of file IO_Pin.h.

7.15.2.338 #define IO_PIN_189 60U

Pin Nr. 189

Definition at line 1085 of file IO_Pin.h.

7.15.2.339 #define IO_PIN_190 74U

Pin Nr. 190

Definition at line 1086 of file IO_Pin.h.

7.15.2.340 #define IO_PIN_191 92U

Pin Nr. 191

Definition at line 1087 of file IO_Pin.h.

7.15.2.341 #define IO_PIN_192 62U

Pin Nr. 192

Definition at line 1088 of file IO_Pin.h.

7.15.2.342 #define IO_PIN_193 76U

Pin Nr. 193

Definition at line 1089 of file IO_Pin.h.

7.15.2.343 #define IO_PIN_194 94U

Pin Nr. 194

Definition at line 1090 of file IO_Pin.h.

7.15.2.344 #define IO_PIN_195 64U

Pin Nr. 195

Definition at line 1091 of file IO_Pin.h.

7.15.2.345 #define IO_PIN_196 78U

Pin Nr. 196

Definition at line 1092 of file IO_Pin.h.

7.15.2.346 #define IO_PIN_201 254U

Pin Nr. 201 (BAT+ Power)

Definition at line 1108 of file IO_Pin.h.

7.15.2.347 #define IO_PIN_207 97U

Pin Nr. 207

Definition at line 1093 of file IO_Pin.h.

7.15.2.348 #define IO_PIN_220 98U

Pin Nr. 220

Definition at line 1094 of file IO_Pin.h.

7.15.2.349 #define IO_PIN_221 101U

Pin Nr. 221

Definition at line 1095 of file IO_Pin.h.

7.15.2.350 #define IO_PIN_234 100U

Pin Nr. 234

Definition at line 1096 of file IO_Pin.h.

7.15.2.351 #define IO_PIN_238 45U

Pin Nr. 238

Definition at line 1097 of file IO_Pin.h.

7.15.2.352 #define IO_PIN_239 47U

Pin Nr. 239

Definition at line 1098 of file IO_Pin.h.

7.15.2.353 #define IO_PIN_240 49U

Pin Nr. 240

Definition at line 1099 of file IO_Pin.h.

7.15.2.354 #define IO_PIN_241 51U

Pin Nr. 241

Definition at line 1100 of file IO_Pin.h.

7.15.2.355 #define IO_PIN_246 96U

Pin Nr. 246

Definition at line 1101 of file IO_Pin.h.

7.15.2.356 #define IO_PIN_247 99U

Pin Nr. 247

Definition at line 1102 of file IO_Pin.h.

7.15.2.357 #define IO_PIN_251 44U

Pin Nr. 251

Definition at line 1103 of file IO_Pin.h.

7.15.2.358 #define IO_PIN_252 46U

Pin Nr. 252

Definition at line 1104 of file IO_Pin.h.

7.15.2.359 #define IO_PIN_253 48U

Pin Nr. 253

Definition at line 1105 of file IO_Pin.h.

7.15.2.360 #define IO_PIN_254 50U

Pin Nr. 254

Definition at line 1106 of file IO_Pin.h.

7.15.2.361 #define IO_PIN_NONE 255U

Undefined pin

Definition at line 1109 of file IO_Pin.h.

7.15.2.362 #define IO_PVG_00 IO_PIN_161

main function PVG output

Definition at line 1323 of file IO_Pin.h.

7.15.2.363 #define IO_PVG_01 IO_PIN_185

main function PVG output

Definition at line 1324 of file IO_Pin.h.

7.15.2.364 #define IO_PVG_02 IO_PIN_188

main function PVG output

Definition at line 1325 of file IO_Pin.h.

7.15.2.365 #define IO_PVG_03 IO_PIN_164

main function PVG output

Definition at line 1326 of file IO_Pin.h.

7.15.2.366 #define IO_PVG_04 IO_PIN_191

main function PVG output

Definition at line 1327 of file IO_Pin.h.

7.15.2.367 #define IO_PVG_05 IO_PIN_167

main function PVG output

Definition at line 1328 of file IO_Pin.h.

7.15.2.368 #define IO_PVG_06 IO_PIN_194

main function PVG output

Definition at line 1329 of file IO_Pin.h.

7.15.2.369 #define IO_PVG_07 IO_PIN_170

main function PVG output

Definition at line 1330 of file IO_Pin.h.

7.15.2.370 #define IO_PWD_00 IO_PIN_115

main function timer input

Definition at line 1263 of file IO_Pin.h.

7.15.2.371 #define IO_PWD_01 IO_PIN_139

main function timer input

Definition at line 1264 of file IO_Pin.h.

7.15.2.372 #define IO_PWD_02 IO_PIN_116

main function timer input

Definition at line 1265 of file IO_Pin.h.

7.15.2.373 #define IO_PWD_03 IO_PIN_140

main function timer input

Definition at line 1266 of file IO_Pin.h.

7.15.2.374 #define IO_PWD_04 IO_PIN_117

main function timer input

Definition at line 1267 of file IO_Pin.h.

7.15.2.375 #define IO_PWD_05 IO_PIN_141

main function timer input

Definition at line 1268 of file IO_Pin.h.

7.15.2.376 #define IO_PWD_06 IO_PIN_122

main function timer input

Definition at line 1269 of file IO_Pin.h.

7.15.2.377 #define IO_PWD_07 IO_PIN_146

main function timer input

Definition at line 1270 of file IO_Pin.h.

7.15.2.378 #define IO_PWD_08 IO_PIN_123

main function timer input

Definition at line 1271 of file IO_Pin.h.

7.15.2.379 #define IO_PWD_09 IO_PIN_147

main function timer input

Definition at line 1272 of file IO_Pin.h.

7.15.2.380 #define IO_PWD_10 IO_PIN_124

main function timer input

Definition at line 1273 of file IO_Pin.h.

7.15.2.381 #define IO_PWD_11 IO_PIN_148

main function timer input

Definition at line 1274 of file IO_Pin.h.

7.15.2.382 #define IO_PWD_12 IO_PIN_101

alternative timer function for [IO_PWM_28](#)

Definition at line 1275 of file IO_Pin.h.

7.15.2.383 #define IO_PWD_13 IO_PIN_125

alternative timer function for [IO_PWM_29](#)

Definition at line 1276 of file IO_Pin.h.

7.15.2.384 #define IO_PWD_14 IO_PIN_150

alternative timer function for [IO_PWM_30](#)

Definition at line 1277 of file IO_Pin.h.

7.15.2.385 #define IO_PWD_15 IO_PIN_174

alternative timer function for [IO_PWM_31](#)

Definition at line 1278 of file IO_Pin.h.

7.15.2.386 #define IO_PWD_16 IO_PIN_102

alternative timer function for [IO_PWM_32](#)

Definition at line 1279 of file IO_Pin.h.

7.15.2.387 #define IO_PWD_17 IO_PIN_126

alternative timer function for [IO_PWM_33](#)

Definition at line 1280 of file IO_Pin.h.

7.15.2.388 #define IO_PWD_18 IO_PIN_151

alternative timer function for [IO_PWM_34](#)

Definition at line 1281 of file IO_Pin.h.

7.15.2.389 #define IO_PWD_19 IO_PIN_175

alternative timer function for [IO_PWM_35](#)

Definition at line 1282 of file IO_Pin.h.

7.15.2.390 #define IO_PWM_00 IO_PIN_153

main function PWM output (shut off group 0)

Definition at line 1285 of file IO_Pin.h.

7.15.2.391 #define IO_PWM_01 IO_PIN_177

main function PWM output (shut off group 0)

Definition at line 1286 of file IO_Pin.h.

7.15.2.392 #define IO_PWM_02 IO_PIN_156

main function PWM output (shut off group 0)

Definition at line 1287 of file IO_Pin.h.

7.15.2.393 #define IO_PWM_03 IO_PIN_180

main function PWM output (shut off group 0)

Definition at line 1288 of file IO_Pin.h.

7.15.2.394 #define IO_PWM_04 IO_PIN_159

main function PWM output (shut off group 0)

Definition at line 1289 of file IO_Pin.h.

7.15.2.395 #define IO_PWM_05 IO_PIN_183

main function PWM output (shut off group 0)

Definition at line 1290 of file IO_Pin.h.

7.15.2.396 #define IO_PWM_06 IO_PIN_186

main function PWM output (shut off group 0)

Definition at line 1291 of file IO_Pin.h.

7.15.2.397 #define IO_PWM_07 IO_PIN_162

main function PWM output (shut off group 0)

Definition at line 1292 of file IO_Pin.h.

7.15.2.398 #define IO_PWM_08 IO_PIN_189

main function PWM output (shut off group 0)

Definition at line 1293 of file IO_Pin.h.

7.15.2.399 #define IO_PWM_09 IO_PIN_165

main function PWM output (shut off group 0)

Definition at line 1294 of file IO_Pin.h.

7.15.2.400 #define IO_PWM_10 IO_PIN_192

main function PWM output (shut off group 0)

Definition at line 1295 of file IO_Pin.h.

7.15.2.401 #define IO_PWM_11 IO_PIN_168

main function PWM output (shut off group 0)

Definition at line 1296 of file IO_Pin.h.

7.15.2.402 #define IO_PWM_12 IO_PIN_195

main function PWM output (shut off group 0)

Definition at line 1297 of file IO_Pin.h.

7.15.2.403 #define IO_PWM_13 IO_PIN_171

main function PWM output (shut off group 0)

Definition at line 1298 of file IO_Pin.h.

7.15.2.404 #define IO_PWM_14 IO_PIN_154

main function PWM output (shut off group 1)

Definition at line 1299 of file IO_Pin.h.

7.15.2.405 #define IO_PWM_15 IO_PIN_178

main function PWM output (shut off group 1)

Definition at line 1300 of file IO_Pin.h.

7.15.2.406 #define IO_PWM_16 IO_PIN_157

main function PWM output (shut off group 1)

Definition at line 1301 of file IO_Pin.h.

7.15.2.407 #define IO_PWM_17 IO_PIN_181

main function PWM output (shut off group 1)

Definition at line 1302 of file IO_Pin.h.

7.15.2.408 #define IO_PWM_18 IO_PIN_160

main function PWM output (shut off group 1)

Definition at line 1303 of file IO_Pin.h.

7.15.2.409 #define IO_PWM_19 IO_PIN_184

main function PWM output (shut off group 1)

Definition at line 1304 of file IO_Pin.h.

7.15.2.410 #define IO_PWM_20 IO_PIN_187

main function PWM output (shut off group 1)

Definition at line 1305 of file IO_Pin.h.

7.15.2.411 #define IO_PWM_21 IO_PIN_163

main function PWM output (shut off group 1)

Definition at line 1306 of file IO_Pin.h.

7.15.2.412 #define IO_PWM_22 IO_PIN_190

main function PWM output (shut off group 1)

Definition at line 1307 of file IO_Pin.h.

7.15.2.413 #define IO_PWM_23 IO_PIN_166

main function PWM output (shut off group 1)

Definition at line 1308 of file IO_Pin.h.

7.15.2.414 #define IO_PWM_24 IO_PIN_193

main function PWM output (shut off group 1)

Definition at line 1309 of file IO_Pin.h.

7.15.2.415 #define IO_PWM_25 IO_PIN_169

main function PWM output (shut off group 1)

Definition at line 1310 of file IO_Pin.h.

7.15.2.416 #define IO_PWM_26 IO_PIN_196

main function PWM output (shut off group 1)

Definition at line 1311 of file IO_Pin.h.

7.15.2.417 #define IO_PWM_27 IO_PIN_172

main function PWM output (shut off group 1)

Definition at line 1312 of file IO_Pin.h.

7.15.2.418 #define IO_PWM_28 IO_PIN_101

main function PWM output (shut off group 2)

Definition at line 1313 of file IO_Pin.h.

7.15.2.419 #define IO_PWM_29 IO_PIN_125

main function PWM output (shut off group 2)

Definition at line 1314 of file IO_Pin.h.

7.15.2.420 #define IO_PWM_30 IO_PIN_150

main function PWM output (shut off group 2)

Definition at line 1315 of file IO_Pin.h.

7.15.2.421 #define IO_PWM_31 IO_PIN_174

main function PWM output (shut off group 2)

Definition at line 1316 of file IO_Pin.h.

7.15.2.422 #define IO_PWM_32 IO_PIN_102

main function PWM output (shut off group 2)

Definition at line 1317 of file IO_Pin.h.

7.15.2.423 #define IO_PWM_33 IO_PIN_126

main function PWM output (shut off group 2)

Definition at line 1318 of file IO_Pin.h.

7.15.2.424 #define IO_PWM_34 IO_PIN_151

main function PWM output (shut off group 2)

Definition at line 1319 of file IO_Pin.h.

7.15.2.425 #define IO_PWM_35 IO_PIN_175

main function PWM output (shut off group 2)

Definition at line 1320 of file IO_Pin.h.

7.15.2.426 #define IO_SENSOR_SUPPLY_0 IO_PIN_247

Definition at line 1251 of file IO_Pin.h.

7.15.2.427 #define IO_SENSOR_SUPPLY_1 IO_PIN_234

Definition at line 1252 of file IO_Pin.h.

7.15.2.428 #define IO_SENSOR_SUPPLY_2 IO_PIN_221

Definition at line 1253 of file IO_Pin.h.

7.15.2.429 #define IO_UBAT IO_PIN_201

Definition at line 1260 of file IO_Pin.h.

7.15.2.430 #define IO_VOUT_00 IO_PIN_161

alternative voltage output function for [IO_PVG_00](#)

Definition at line 1333 of file IO_Pin.h.

7.15.2.431 #define IO_VOUT_01 IO_PIN_185

alternative voltage output function for [IO_PVG_01](#)

Definition at line 1336 of file IO_Pin.h.

7.15.2.432 #define IO_VOUT_02 IO_PIN_188

alternative voltage output function for [IO_PVG_02](#)

Definition at line 1339 of file IO_Pin.h.

7.15.2.433 #define IO_VOUT_03 IO_PIN_164

alternative voltage output function for [IO_PVG_03](#)

Definition at line 1342 of file IO_Pin.h.

7.15.2.434 #define IO_VOUT_04 IO_PIN_191

alternative voltage output function for [IO_PVG_04](#)

Definition at line 1345 of file IO_Pin.h.

7.15.2.435 #define IO_VOUT_05 IO_PIN_167

alternative voltage output function for [IO_PVG_05](#)

Definition at line 1348 of file IO_Pin.h.

7.15.2.436 #define IO_VOUT_06 IO_PIN_194

alternative voltage output function for [IO_PVG_06](#)

Definition at line 1351 of file IO_Pin.h.

7.15.2.437 #define IO_VOUT_07 IO_PIN_170

alternative voltage output function for [IO_PVG_07](#)

Definition at line 1354 of file IO_Pin.h.

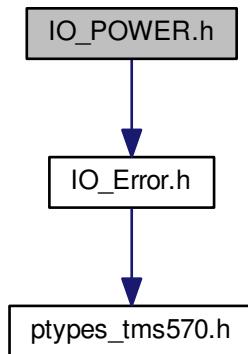
7.15.2.438 #define IO_WAKEUP IO_PIN_220

Definition at line 1257 of file IO_Pin.h.

7.16 IO_POWER.h File Reference

I/O Driver functions for Power IC control.

Include dependency graph for IO_POWER.h:



Functions

- `IO_ErrorType IO_POWER_EnableExtShutOff (ubyte1 group)`
Configures the external shut off inputs.
- `IO_ErrorType IO_POWER_Get (ubyte1 pin, ubyte1 *const state)`
Returns the current state of a POWER feature.
- `IO_ErrorType IO_POWER_GetExtShutOff (ubyte1 group, bool *const state)`
Reads the state of the external shut off signal.
- `IO_ErrorType IO_POWER_Set (ubyte1 pin, ubyte1 mode)`
Sets a certain mode of the Power IC.

Power values

Selects power configuration.

- `#define IO_POWER_OFF 0U`
- `#define IO_POWER_ON 1U`
- `#define IO_POWER_ON_5V 2U`
- `#define IO_POWER_ON_6V 3U`
- `#define IO_POWER_ON_7V 4U`
- `#define IO_POWER_ON_8V 5U`
- `#define IO_POWER_ON_9V 6U`
- `#define IO_POWER_ON_10V 7U`

7.16.1 Detailed Description

IO Driver functions for Power IC control.

The Power IC control functions allow to power down the ECU, switching on and off the sensor supply voltages of the ECU, enabling/disabling the power stage enable signal and the safety switches.

The ECU provides 2 separate inputs `IO_K15` and `IO_WAKEUP` for ECU power on and off functionalities.

Note

The ECU does not power down automatically when `IO_K15` and `IO_WAKEUP` is low. When `IO_K15` becomes low, the postrun is still active, and a power down can be now initiated by the application SW.

Remarks

- The ECU shall only be powered down with

```
IO_POWER_Set(IO_K15,  
            IO_POWER_OFF)
```

or

```
IO_POWER_Set(IO_WAKEUP,  
            IO_POWER_OFF)
```

if `IO_K15` is low in order to ensure a correct shutdown.

- It is the responsibility of the user to check for the right state of `IO_K15` directly before calling `IO_POWER_Set()`.

Attention

- `IO_K15` is level sensitive.
 - If `IO_K15` is low, the ECU will power down completely. Otherwise the ECU's reaction to `IO_POWER_Set(IO_K15, IO_POWER_OFF)` and `IO_POWER_Set(IO_WAKEUP, IO_POWER_OFF)` depends on multiple variables.
 - If `IO_K15` is high, and the resets were enabled in the safety configuration, and the watchdog device error counter has not reached the final threshold value, the ECU will perform a hard reset. The hard reset will reset the entire ECU, including all internal status variables and the watchdog error counter, and then power up the device again. (The current value of the watchdog error counter can be requested with the `DIAG_Status()` function.)
 - In all other cases the shutdown functions have no effect.
- `IO_WAKEUP` is edge sensitive.
 - If `IO_WAKEUP` is high and `IO_POWER_Set()` is called, the ECU will power down and remain off until the next rising edge on `IO_WAKEUP` (assuming that `IO_K15` is low).

Remarks

- Whenever `IO_POWER_Set()` with `IO_POWER_OFF` is called, the postrun will be disabled.
- If the ECU can not power down for some reason (e.g. `IO_K15` is high and a hard reset is not possible) a power down will occur as soon as `IO_K15` goes low.
- The application SW can reenable the postrun by calling `IO_POWER_Set()` with `IO_POWER_ON`.

7.16.2 Power Code Examples

Examples for using the Power API

7.16.2.1 Example for Power initialization

The Power driver does not need an explicit initialization. The Power driver is initialized by `IO_Driver_Init()`.

```
IO_Driver_Init(NULL);
```

7.16.2.2 Example for Sensor Supply

```
// switch on sensor supply 0
IO_POWER_Set (IO_SENSOR_SUPPLY_0,
               IO_POWER_ON);

// switch on variable sensor supply 2 with 8 volts
IO_POWER_Set (IO_SENSOR_SUPPLY_2,
               IO_POWER_ON_8V);

// switch off sensor supply 0
IO_POWER_Set (IO_SENSOR_SUPPLY_0,
               IO_POWER_OFF);
```

7.16.2.3 Example for Safety Switch

```
ubyte1 ssw2_state = IO_POWER_OFF;

// enable safety switch 0 (shut off group 0)
IO_POWER_Set (IO_INT_SAFETY_SW_0,
               IO_POWER_ON);

// get status of safety switch 2 (shut off group 2)
IO_POWER_Get (IO_INT_SAFETY_SW_2,
               &ssw2_state);
```

7.16.3 Macro Definition Documentation

7.16.3.1 #define IO_POWER_OFF 0U

switch off - turns off the respective power function

Definition at line 141 of file `IO_POWER.h`.

7.16.3.2 #define IO_POWER_ON 1U

switch on - turns on the respective power function

Definition at line 142 of file IO_POWER.h.

7.16.3.3 #define IO_POWER_ON_10V 7U

switch on with 10 volts - configures the respective power function for 10 volts and turns it on (only available for [IO_SENSOR_SUPPLY_2](#))

Definition at line 168 of file IO_POWER.h.

7.16.3.4 #define IO_POWER_ON_5V 2U

switch on with 5 volts - configures the respective power function for 5 volts and turns it on (only available for [IO_SENSOR_SUPPLY_2](#))

Definition at line 143 of file IO_POWER.h.

7.16.3.5 #define IO_POWER_ON_6V 3U

switch on with 6 volts - configures the respective power function for 6 volts and turns it on (only available for [IO_SENSOR_SUPPLY_2](#))

Definition at line 148 of file IO_POWER.h.

7.16.3.6 #define IO_POWER_ON_7V 4U

switch on with 7 volts - configures the respective power function for 7 volts and turns it on (only available for [IO_SENSOR_SUPPLY_2](#))

Definition at line 153 of file IO_POWER.h.

7.16.3.7 #define IO_POWER_ON_8V 5U

switch on with 8 volts - configures the respective power function for 8 volts and turns it on (only available for [IO_SENSOR_SUPPLY_2](#))

Definition at line 158 of file IO_POWER.h.

7.16.3.8 #define IO_POWER_ON_9V 6U

switch on with 9 volts - configures the respective power function for 9 volts and turns it on (only available for [IO_SENSOR_SUPPLY_2](#))

Definition at line 163 of file IO_POWER.h.

7.16.4 Function Documentation

7.16.4.1 IO_ErrorType IO_POWER_EnableExtShutOff (ubyte1 group)

Configures the external shut off inputs.

Configures and enables the external shut off inputs for a shut off group

Parameters

<i>group</i>	Group for which the external shut off inputs should be configured, one of: <ul style="list-style-type: none">• IO_INT_SAFETY_SW_0 (shut off group 0)• IO_INT_SAFETY_SW_1 (shut off group 1)• IO_INT_SAFETY_SW_2 (shut off group 2)
--------------	--

Returns

[IO_ErrorType](#)

Return values

IO_E_OK	everything fine
IO_E_INVALID_CHANNEL_ID	the given channel id does not exist or is not available on the used ECU variant
IO_E_CH_CAPABILITY	the given channel is not a valid shut off group
IO_E_CHANNEL_NOT_CONFIGURED	power module has not been initialized
IO_E_CHANNEL_BUSY	the external shut off inputs were already configured or the associated ADC input channels are already used by another function
IO_E_UNKNOWN	an unknown error occurred

Note

The external shut off functionality requires 2 analog inputs for each group. The following pins are used for each group:

- [IO_INT_SAFETY_SW_0](#) (shut off group 0):
 - [IO_ADC_18](#) (input 0)
 - [IO_ADC_19](#) (input 1)
- [IO_INT_SAFETY_SW_1](#) (shut off group 1):
 - [IO_ADC_20](#) (input 0)
 - [IO_ADC_21](#) (input 1)
- [IO_INT_SAFETY_SW_2](#) (shut off group 2):
 - [IO_ADC_22](#) (input 0)
 - [IO_ADC_23](#) (input 1)

Attention

- Both inputs need to switch in opposite direction, using a normally open and a normally close contact.
- The outputs can only be enabled, when input 0 reads a high level and input 1 reads a low level
- The outputs will be disabled, when input 0 reads a low level and input 1 reads a high level
- If both inputs read the same level, the diagnostic state machine will be informed, with the diagnostic error code being [DIAG_E_SSW_EXT_SHUTOFF](#).

Remarks

When the external shut off is applied, diagnostic is still possible on the affected outputs

7.16.4.2 IO_ErrorType IO_POWER_Get (ubyte1 *pin*, ubyte1 *const *state*)

Returns the current state of a POWER feature.

Returns the state of sensor supplies, K15, WakeUp, safety switches and power stage enable

Parameters

<i>pin</i>		pin for which the mode shall be set, one of: • IO_INT_POWERSTAGE_ENABLE • IO_SENSOR_SUPPLY_0 • IO_SENSOR_SUPPLY_1 • IO_SENSOR_SUPPLY_2 • IO_INT_SAFETY_SW_0 (shut off group 0) • IO_INT_SAFETY_SW_1 (shut off group 1) • IO_INT_SAFETY_SW_2 (shut off group 2) • IO_K15 (K15) • IO_WAKEUP (WakeUp)
<i>out</i>	<i>state</i>	pointer to ubyte1, returns the state of the selected feature, one of: • IO_POWER_ON • IO_POWER_OFF • IO_POWER_ON_5V • IO_POWER_ON_6V • IO_POWER_ON_7V • IO_POWER_ON_8V • IO_POWER_ON_9V • IO_POWER_ON_10V

Returns

[IO_ErrorType](#)

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist or is not available on the used ECU variant
<code>IO_E_CH_CAPABILITY</code>	the given channel is not a POWER channel
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	power module has not been initialized
<code>IO_E_NULL_POINTER</code>	a NULL pointer has been passed
<code>IO_E_SAFE_STATE</code>	the given channel is in a safe state

Remarks

- `IO_INT_POWERSTAGE_ENABLE` and `IO_INT_SAFETY_SW_0 .. IO_INT_SAFETY_SW_2` are internal pins.
- `IO_INT_POWERSTAGE_ENABLE` controls the internal powerstage enable signal. Without enabling this signal all power outputs remain low (switched off).
- `IO_INT_SAFETY_SW_0 .. IO_INT_SAFETY_SW_2` control the internal safety switches for PWM outputs and represent a shut off group. Without switching this signal to ON, the PWM power stages will not be supplied and the pin will therefore remain low. The safety switch assignment to the output stage is defined as following:
 - `IO_INT_SAFETY_SW_0` (shut off group 0): `IO_PWM_00 .. IO_PWM_13` (`IO_DO_16 .. IO_DO_29`)
 - `IO_INT_SAFETY_SW_1` (shut off group 1): `IO_PWM_14 .. IO_PWM_27` (`IO_DO_30 .. IO_DO_43`)
 - `IO_INT_SAFETY_SW_2` (shut off group 2): `IO_PWM_28 .. IO_PWM_35` (`IO_DO_44 .. IO_DO_51`)
- `IO_SENSOR_SUPPLY_0 .. IO_SENSOR_SUPPLY_2` control the 3 external sensor supply pins. `IO_SENSOR_SUPPLY_0` and `IO_SENSOR_SUPPLY_1` are fixed to 5 volts. Calling this function with this pins, the state can only return `IO_POWER_ON` or `IO_POWER_OFF`. `IO_SENSOR_SUPPLY_2` is a variable supply and can be configured in the range from 5 to 10 volts. Calling this function with this pin, the state can return `IO_POWER_OFF` or anything in between `IO_POWER_ON_5V` and `IO_POWER_ON_10V`.

7.16.4.3 `IO_ErrorType IO_POWER_GetExtShutOff(ubyte1 group, bool *const state)`

Reads the state of the external shut off signal.

Reads the evaluated signal of the external shut off inputs

Parameters

	<code>group</code>	Group for which the status shall be read, one of: • <code>IO_INT_SAFETY_SW_0</code> (shut off group 0) • <code>IO_INT_SAFETY_SW_1</code> (shut off group 1) • <code>IO_INT_SAFETY_SW_2</code> (shut off group 2)
<code>out</code>	<code>state</code>	pointer to <code>bool</code> , returns the state of the shut off group, one of: • <code>TRUE</code> if the outputs were disabled by the external shut off • <code>FALSE</code> if the outputs were not disabled by the external shut off

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist or is not available on the used ECU variant
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the external shut off has not been enabled
<code>IO_E_NULL_POINTER</code>	a NULL pointer has been passed
<code>IO_E_UNKNOWN</code>	an unknown error occurred

7.16.4.4 `IO_ErrorType IO_POWER_Set (ubyte1 pin, ubyte1 mode)`

Sets a certain mode of the Power IC.

Sets a certain mode for a sensor supply pin or the device power

Parameters

<code>pin</code>	pin for which the mode shall be set, one of: <ul style="list-style-type: none"> • <code>IO_INT_POWERSTAGE_ENABLE</code> • <code>IO_SENSOR_SUPPLY_0</code> • <code>IO_SENSOR_SUPPLY_1</code> • <code>IO_SENSOR_SUPPLY_2</code> • <code>IO_INT_SAFETY_SW_0</code> (shut off group 0) • <code>IO_INT_SAFETY_SW_1</code> (shut off group 1) • <code>IO_INT_SAFETY_SW_2</code> (shut off group 2) • <code>IO_K15</code> (K15, for power down) • <code>IO_WAKEUP</code> (WakeUp, for power down after a wake up)
<code>mode</code>	sets a certain mode, one of: <ul style="list-style-type: none"> • <code>IO_POWER_ON</code> • <code>IO_POWER_OFF</code> • <code>IO_POWER_ON_5V</code> • <code>IO_POWER_ON_6V</code> • <code>IO_POWER_ON_7V</code> • <code>IO_POWER_ON_8V</code> • <code>IO_POWER_ON_9V</code> • <code>IO_POWER_ON_10V</code>

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist or is not available on the used ECU variant
<code>IO_E_CH_CAPABILITY</code>	the given channel is not a POWER channel

<code>IO_E_INVALID_PARAMETER</code>	an invalid parameter has been passed
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	power module has not been initialized
<code>IO_E_SAFE_STATE</code>	the given channel is in a safe state
<code>IO_E_INTERNAL_CSM</code>	an internal error occurred
<code>IO_E_UNKNOWN</code>	an unknown error occurred

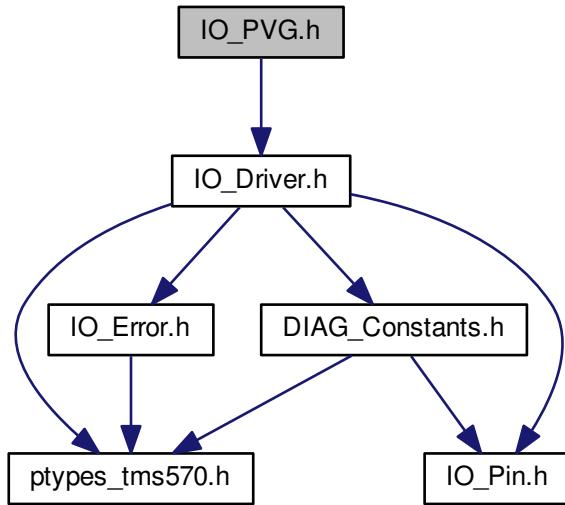
Remarks

- `IO_INT_POWERSTAGE_ENABLE` and `IO_INT_SAFETY_SW_0 .. IO_INT_SAFETY_SW_2` are internal pins.
- `IO_INT_POWERSTAGE_ENABLE` controls the internal powerstage enable signal. Without enabling this signal all power outputs remain low (switched off).
- `IO_INT_SAFETY_SW_0 .. IO_INT_SAFETY_SW_2` control the internal safety switches for PWM outputs and each represents a shut off group. Without switching this signal to ON, the PWM power * stages will not be supplied and the pins will therefore remain low. The safety switch assignment to the output stages is defined as follows:
 - `IO_INT_SAFETY_SW_0` (shut off group 0): `IO_PWM_00 .. IO_PWM_13` (`IO_DO_16 .. IO_DO_29`)
 - `IO_INT_SAFETY_SW_1` (shut off group 1): `IO_PWM_14 .. IO_PWM_27` (`IO_DO_30 .. IO_DO_43`)
 - `IO_INT_SAFETY_SW_2` (shut off group 2): `IO_PWM_28 .. IO_PWM_35` (`IO_DO_44 .. IO_DO_51`)
- `IO_SENSOR_SUPPLY_0 .. IO_SENSOR_SUPPLY_2` control the 3 external sensor supply pins. `IO_SENSOR_SUPPLY_0` and `IO_SENSOR_SUPPLY_1` are fixed to 5 volts. Calling this function with this pins, the mode can only be set to `IO_POWER_ON` or `IO_POWER_OFF`. `IO_SENSOR_SUPPLY_2` is a variable supply and can be configured in the range from 5 to 10 volts. Calling this function with this pin, the mode can only be set to `IO_POWER_OFF` or anything in between `IO_POWER_ON_5V` and `IO_POWER_ON_10V`.

7.17 IO_PVG.h File Reference

IO Driver functions for PVG channels.

Include dependency graph for IO_PVG.h:



Functions

- `IO_ErrorType IO_PVG_DelInit (ubyte1 pvg_channel)`
Deinitializes one PVG output.
- `IO_ErrorType IO_PVG_GetVoltage (ubyte1 pvg_channel, ubyte2 *const voltage, bool *const fresh)`
Get the voltage feedback of one PVG channel.
- `IO_ErrorType IO_PVG_Init (ubyte1 pvg_channel, ubyte2 output_value)`
Setup one PVG channel.
- `IO_ErrorType IO_PVG_ResetProtection (ubyte1 pvg_channel, ubyte1 *const reset_cnt)`
Reset the output protection for a PVG channel.
- `IO_ErrorType IO_PVG_SetOutput (ubyte1 pvg_channel, ubyte2 output_value)`
Sets the output value of one PVG channel.

7.17.1 Detailed Description

IO Driver functions for PVG channels.

Contains all service functions for the PVG (Proportional Valve Group) outputs. Up to 8 channels can be configured: `IO_PVG_00 .. IO_PVG_07`

PVG valves are active valves with integrated electronic control. The input impedance is defined with 12kOhm to 50% Ubat (2 * 24kOhm to Ubat and GND).

The PVG output stage is a push/pull PWM output with a well defined output resistance of 2.58kOhm. This resistance is necessary for operation, low pass filtering and over load protection.

Note

The correction of the PVG impedance is directly performed inside the module. Therefore a PVG valve is needed for correct operation of this outputs.

When configuring a PVG output, the associated voltage feedback channel will also be configured.

PVG-API Usage:

- [Examples for PVG API functions](#)

7.17.2 PVG output protection

Each PVG output is individually protected against malfunction. Whenever the difference between the measured output (`U_feedback`) and the configured output calculated by `U_diff = output_value / 100 * U_BAT - U_feedback` is greater than `abs(+/-18V)` for at least 100ms, the output protection is enabled latest within 12ms.

When entering the protection state, the PVG output has to remain in this state for at least 1s. After this wait time the PVG output can be reenabled via function `IO_PVG_ResetProtection()`. Note that the number of reenabling operations for a single PVG output is limited to 10. Refer to function `IO_PVG_ResetProtection()` for more information on how to reenable a PVG output.

7.17.3 PVG Code Examples

Examples for using the PVG API

7.17.3.1 PVG initialization example:

```
// Setup a PVG output with output value of 50%
IO_PVG_Init(IO_PVG_00,
             5000);           // output value is 5000 = 50%
```

7.17.3.2 PVG task examples:

```
IO_PVG_SetOutput(IO_PVG_00,
                  6200);           // set duty cycle to 62%
```

7.17.4 Function Documentation

7.17.4.1 IO_ErrorType IO_PVG_DelInit (ubyte1 pvg_channel)

Deinitializes one PVG output.

Parameters

<i>pvg_channel</i>	PVG output (IO_PVG_00 .. IO_PVG_07)
--------------------	---

Returns

[IO_ErrorType](#)

Return values

IO_E_OK	everything fine
IO_E_INVALID_CHANNEL_ID	the given channel id does not exist
IO_E_CHANNEL_NOT_CONFIGURED	the given channel is not configured
IO_E_CH_CAPABILITY	the given channel is not a PVG channel
IO_E_UNKNOWN	an unknown error occurred

7.17.4.2 [IO_ErrorType](#) [IO_PVG_GetVoltage](#) ([ubyte1](#) *pvg_channel*, [ubyte2](#) *const *voltage*, [bool](#) *const *fresh*)

Get the voltage feedback of one PVG channel.

Parameters

	<i>pvg_channel</i>	PVG channel (IO_PVG_00 .. IO_PVG_07)
out	<i>voltage</i>	Measured voltage in mV. Range: 0..32000 (0V..32.000V)
out	<i>fresh</i>	Indicates if new values are available since the last call. <ul style="list-style-type: none"> • TRUE: Value in "voltage" is valid • FALSE: No new value available.

Returns

[IO_ErrorType](#)

Return values

IO_E_OK	everything fine
IO_E_INVALID_CHANNEL_ID	the given channel id does not exist
IO_E_NULL_POINTER	a NULL pointer has been passed to the function
IO_E_CHANNEL_NOT_CONFIGURED	the given channel is not configured
IO_E_CH_CAPABILITY	the given channel is not a PVG channel
IO_E_STARTUP	the output is in the startup phase
IO_E_REFERENCE	the reference voltage is out of range
IO_E_UNKNOWN	an unknown error occurred

Remarks

If there is no new measurement value available (for example the function `IO_PVG_GetVoltage()` gets called more frequently than the AD sampling) the flag `fresh` will be set to `FALSE`.

7.17.4.3 IO_ErrorType `IO_PVG_Init(ubyte1 pvg_channel, ubyte2 output_value)`

Setup one PVG channel.

Parameters

<code>pvg_channel</code>	PVG channel (<code>IO_PVG_00 .. IO_PVG_07</code>)
<code>output_value</code>	Output value with which the PVG-channel will be initialized in percent * 100 (1000..9000)

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist or is not available on the used ECU variant
<code>IO_E_CH_CAPABILITY</code>	the given channel is not a PVG channel or the used ECU variant does not support this function
<code>IO_E_INVALID_PARAMETER</code>	parameter is out of range
<code>IO_E_CHANNEL_BUSY</code>	the PVG output channel or the voltage feedback channel is currently used by another function
<code>IO_E_DRIVER_NOT_INITIALIZED</code>	the common driver init function has not been called before
<code>IO_E_FPGA_NOT_INITIALIZED</code>	the FPGA has not been initialized
<code>IO_E_UNKNOWN</code>	an unknown error occurred

7.17.4.4 IO_ErrorType `IO_PVG_ResetProtection(ubyte1 pvg_channel, ubyte1 *const reset_cnt)`

Reset the output protection for a PVG channel.

Parameters

	<code>pvg_channel</code>	PVG channel (<code>IO_PVG_00 .. IO_PVG_07</code>)
<code>out</code>	<code>reset_cnt</code>	Protection reset counter. Indicates how often the application already reset the protection.

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist
<code>IO_E_FET_PROT_NOT_ACTIVE</code>	no output FET protection is active
<code>IO_E_FET_PROT_PERMANENT</code>	the output FET is protected permanently because it was already reset more than 10 times
<code>IO_E_FET_PROT_WAIT</code>	the output FET protection can not be reset, as the wait time of 1s is not already passed
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the given channel is not configured
<code>IO_E_CH_CAPABILITY</code>	the given channel is not a PVG channel
<code>IO_E_UNKNOWN</code>	an unknown error occurred

Attention

The protection can be reset 10 times, afterwards the output will remain permanently protected

Note

- After entering the output protection, a certain time has to pass before the output protection can be reset:
 - 1s for `IO_PVG_00 .. IO_PVG_07`
- This function will not set the output back again to the last setpoint. After calling this function the output has to be set to the intended level with `IO_PVG_SetOutput()`

Remarks

If the parameter `reset_cnt` is `NULL`, the parameter is ignored. The parameter `reset_cnt` returns the number of resets already performed.

7.17.4.5 IO_ErrorType IO_PVG_SetOutput(ubyte1 *pvg_channel*, ubyte2 *output_value*)

Sets the output value of one PVG channel.

Parameters

<code>pvg_channel</code>	PVG channel (<code>IO_PVG_00 .. IO_PVG_07</code>)
<code>output_value</code>	Output value in percent * 100 (1000..9000)

Returns

`IO_ErrorType`

Return values

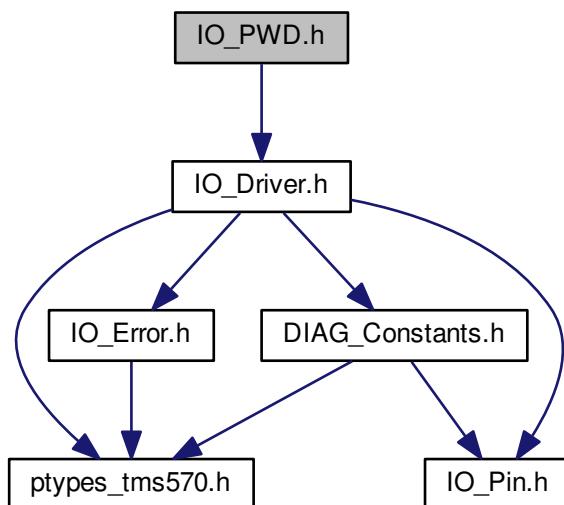
<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist
<code>IO_E_FET_PROT_ACTIVE</code>	the output FET protection is active and has set the output to 0%. It can be reset with <code>IO_PVG_ResetProtection()</code> after the wait time is passed
<code>IO_E_FET_PROT_REENABLE</code>	the output FET protection is ready to be reset with <code>IO_PVG_ResetProtection()</code>

<code>IO_E_FET_PROT_PERMANENT</code>	the output FET is protected permanently because it was already reset more than 10 times. The output will remain at 0%
<code>IO_E_INVALID_PARAMETER</code>	the setpoint is out of range
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the given channel is not configured
<code>IO_E_CH_CAPABILITY</code>	the given channel is not a PVG channel
<code>IO_E_UNKNOWN</code>	an unknown error occurred

7.18 IO_PWD.h File Reference

IO Driver functions for timer input channels.

Include dependency graph for IO_PWD.h:



Data Structures

- struct `io_pwd_cnt_conf_`
Edge counter configuration for the Universal PWD inputs.
- struct `io_pwd_cplx_conf_`
Complex configuration for the Universal PWD inputs.
- struct `io_pwd_cplx_safety_conf_`
Safety configuration for the Complex PWD inputs.
- struct `io_pwd_inc_conf_`
Incremental configuration for the Universal PWD inputs.

- struct `io_pwd_inc_safety_conf_`
Safety configuration for the Incremental or Counter PWD inputs.
- struct `io_pwd_pulse_samples_`
PWD pulse-width data structure.
- struct `io_pwd_universal_safety_conf_`
Safety configuration for the Universal PWD inputs.

Typedefs

- typedef struct `io_pwd_cnt_conf_ IO_PWD_CNT_CONF`
Edge counter configuration for the Universal PWD inputs.
- typedef struct `io_pwd_cplx_conf_ IO_PWD_CPLX_CONF`
Complex configuration for the Universal PWD inputs.
- typedef struct `io_pwd_cplx_safety_conf_ IO_PWD_CPLX_SAFETY_CONF`
Safety configuration for the Complex PWD inputs.
- typedef struct `io_pwd_inc_conf_ IO_PWD_INC_CONF`
Incremental configuration for the Universal PWD inputs.
- typedef struct `io_pwd_inc_safety_conf_ IO_PWD_INC_SAFETY_CONF`
Safety configuration for the Incremental or Counter PWD inputs.
- typedef struct `io_pwd_pulse_samples_ IO_PWD_PULSE_SAMPLES`
PWD pulse-width data structure.
- typedef struct `io_pwd_universal_safety_conf_ IO_PWD_UNIVERSAL_SAFETY_CONF`
Safety configuration for the Universal PWD inputs.

Functions

- `IO_ErrorType IO_PWD_ComplexDelInit (ubyte1 timer_channel)`
Deinitializes a complex PWD input.
- `IO_ErrorType IO_PWD_ComplexGet (ubyte1 timer_channel, ubyte4 *const frequency, ubyte4 *const pulse_width, bool *const pin_value, IO_PWD_PULSE_SAMPLES *const pulse_samples)`
Get the frequency and the pulse-width from the specified timer channel.
- `IO_ErrorType IO_PWD_ComplexInit (ubyte1 timer_channel, ubyte1 pulse_mode, ubyte1 freq_mode, ubyte1 capture_count, ubyte1 pupd, IO_PWD_CPLX_SAFETY_CONF const *const safety_conf)`
Setup single timer channel that measures frequency and pulse-width at the same time.
- `IO_ErrorType IO_PWD_CountDelInit (ubyte1 count_channel)`
Deinitialize a single counter channel.
- `IO_ErrorType IO_PWD_CountGet (ubyte1 count_channel, ubyte2 *const count, bool *const pin_value)`
Get the counter value of a counter channel.
- `IO_ErrorType IO_PWD_CountInit (ubyte1 count_channel, ubyte1 mode, ubyte1 direction, ubyte2 count_init, ubyte1 pupd, IO_PWD_INC_SAFETY_CONF const *const safety_conf)`
Setup a single counter channel.
- `IO_ErrorType IO_PWD_CountSet (ubyte1 count_channel, ubyte2 count)`
Set the counter value of a counter channel.
- `IO_ErrorType IO_PWD_GetCurrent (ubyte1 pwd_channel, ubyte2 *const current, bool *const fresh)`
Get the current feedback of a timer input channel.

- `IO_ErrorType IO_PWD_GetVoltage (ubyte1 pwd_channel, ubyte2 *const voltage, bool *const fresh)`
Get the voltage feedback of a timer input channel.
- `IO_ErrorType IO_PWD_IncDelInit (ubyte1 inc_channel)`
Deinitialize an incremental interface.
- `IO_ErrorType IO_PWD_IncGet (ubyte1 inc_channel, ubyte2 *const count, bool *const pin_value_0, bool *const pin_value_1)`
Get the counter value of an incremental interface.
- `IO_ErrorType IO_PWD_IncInit (ubyte1 inc_channel, ubyte1 mode, ubyte2 count_init, ubyte1 pupd, IO_PWD_INC_SAFETY_CONF const *const safety_conf)`
Setup a single incremental interface.
- `IO_ErrorType IO_PWD_IncSet (ubyte1 inc_channel, ubyte2 count)`
Set the counter value of an incremental interface.
- `IO_ErrorType IO_PWD_ResetProtection (ubyte1 pwd_channel, ubyte1 *const reset_cnt)`
Reset the FET protection for a timer input channel.
- `IO_ErrorType IO_PWD_UniversalDelInit (ubyte1 timer_channel)`
Deinitialize a single universal timer channel.
- `IO_ErrorType IO_PWD_UniversalGet (ubyte1 timer_channel, ubyte4 *const frequency, ubyte4 *const pulse_width, IO_PWD_PULSE_SAMPLES *const pulse_samples, ubyte2 *const edge_count, ubyte2 *const inc_count, bool *const primary_pin_value, bool *const secondary_pin_value)`
Get the measurement results from the specified universal timer channel.
- `IO_ErrorType IO_PWD_UniversalInit (ubyte1 timer_channel, IO_PWD_CPLX_CONF const *const cplx_conf, IO_PWD_CNT_CONF const *const cnt_conf, IO_PWD_INC_CONF const *const inc_conf, ubyte1 pupd, IO_PWD_UNIVERSAL_SAFETY_CONF const *const safety_conf)`
Setup a single universal timer channel.
- `IO_ErrorType IO_PWD_UniversalSet (ubyte1 timer_channel, ubyte2 const *const edge_count, ubyte2 const *const inc_count)`
Set the counter values of an universal timer channel.

High/low time

Specifies whether the high,low or both(period) time shall be captured

- `#define IO_PWD_LOW_TIME 0U`
- `#define IO_PWD_HIGH_TIME 1U`
- `#define IO_PWD_PERIOD_TIME 2U`

Variable edge

Specify the variable edge of the input signal.

If the rising edge is variable, the frequency is measured between the surrounding falling edges.

- `#define IO_PWD_RISING_VAR 2U`
- `#define IO_PWD_FALLING_VAR 3U`

Pull up / Pull down configuration for timer inputs

Configuration of the pull up or pull down resistors on the timer inputs. These defines can be used for the `pupd` parameter of the functions `IO_PWD_ComplexInit()`, `IO_PWD_IncInit()` and `IO_PWD_CountInit()`.

- `#define IO_PWD_NO_PULL 0x03U`
- `#define IO_PWD_PU_10K 0x01U`
- `#define IO_PWD_PD_10K 0x00U`
- `#define IO_PWD_PD_90 0x02U`

PWD maximum pulse pulse-width samples

Maximum pulse-width samples that can be stored in the data structure `IO_PWD_PULSE_SAMPLES`

- `#define IO_PWD_MAX_PULSE_SAMPLES 8U`

Counting mode

Specify the counting mode of a incremental interface.

- `#define IO_PWD_INC_2_COUNT 0x03U`
- `#define IO_PWD_INC_1_COUNT 0x01U`

edge count

Specify on which edge shall be counted

- `#define IO_PWD_RISING_COUNT 1U`
- `#define IO_PWD_FALLING_COUNT 2U`
- `#define IO_PWD_BOTH_COUNT 3U`

count direction

Specify the counting direction

- `#define IO_PWD_UP_COUNT 0U`
- `#define IO_PWD_DOWN_COUNT 1U`

7.18.1 Detailed Description

IO Driver functions for timer input channels.

Contains all service functions for the PWD (Pulse Width Demodulation). There are three groups of timer inputs available:

- `IO_PWD_00..IO_PWD_05`:

- Complex mode: Can be configured to measure frequency and pulse-width at the same time. Additionally, a pull up/down interface can be configured. These inputs can accumulate up to 8 pulse samples. An average value is calculated automatically, but all samples are available on demand. These inputs support voltage and current signals (7mA/14mA). For current signals an additional range check is done.
 - Incremental mode: Can be configured to read incremental (relative) encoders. In this case two inputs are reserved for one incremental encoder (clock and direction) interface. Additionally, a pull up/down interface can be configured. The incremental interface will decrement when the 1st channel is leading and increment when the 2nd channel is leading. These inputs support voltage signals only.
 - Count mode: Can be configured to count rising, falling or both edges. Additionally, a pull up/down interface can be configured. These inputs support voltage signals only.
 - Universal mode: Can be configured as a combination of complex, incremental and count mode.
- [IO_PWD_06..IO_PWD_11](#):
 - Complex mode: Can be configured to measure frequency and pulse-width at the same time. Additionally, a pull up/down interface can be configured. These inputs can accumulate up to 8 pulse samples. An average value is calculated automatically. The different pulse samples cannot be gathered. These inputs support voltage signals only.
 - Incremental mode: Can be configured to read incremental (relative) encoders. In this case two inputs are reserved for one incremental encoder (clock and direction) interface. Additionally, a pull up/down interface can be configured. The incremental interface will decrement when the 1st channel is leading and increment when the 2nd channel is leading. These inputs support voltage signals only.
 - Count mode: Can be configured to count rising, falling or both edges. Additionally, a pull up/down interface can be configured. These inputs support voltage signals only.
 - [IO_PWD_12..IO_PWD_19](#):
 - Complex mode: Can be configured to measure frequency and pulse-width at the same time. However, these inputs do not accumulate a number of samples. Instead every pulse sample has to be handled directly. These inputs support voltage signals only.

PWD-API Usage:

- [Examples for PWD API functions](#)

7.18.2 PWD input protection

Each PWD input that is configured for complex mode in combination with a current sensor is individually protected against overcurrent. Whenever two consecutive samples above 54326uA are measured, the input protection is enabled latest within 22ms.

When entering the protection state, the PWD input has to remain in this state for at least 1s. After this wait time the PWD input can be reenabled via function [IO_PWD_ResetProtection\(\)](#). Note that the number of reenabling operations for a single PWD input is limited to 10. Refer to function [IO_PWD_ResetProtection\(\)](#) for more information on how to reenable a PWD input.

Attention

PWD input protection is only applicable to PWD channels `IO_PWD_00 .. IO_PWD_05` when configured for complex mode in combination with current sensors.

7.18.3 PWD code examples

Examples for using the PWD API

7.18.3.1 PWD initialization example

```
// setup complex timer input (frequency and pulse measurement) for a voltage signal
IO_PWD_ComplexInit(IO_PWD_00,                                // timer input channel
                     IO_PWD_HIGH_TIME,                         // measure high time
                     IO_PWD_FALLING_VAR,                      // select falling edge as variable
                     8,                                         // set number of accumulations
                     IO_PWD_PU_10K,                           // select pull up resistor
                     NULL);                                    // not safety critical
```

7.18.3.2 PWD task example

```
ubyte4 freq_8_val;
ubyte4 pulse_8_val;
bool pin_value;

// read complex timer values (frequency and pulse value
IO_PWD_ComplexGet(IO_PWD_00,
                   &freq_8_val,
                   &pulse_8_val,
                   &pin_value,      // read digital pin value
                   NULL);          // pulse samples not needed
```

7.18.4 Macro Definition Documentation

7.18.4.1 #define IO_PWD_BOTH_COUNT 3U

count on both edges

Definition at line 221 of file IO_PWD.h.

7.18.4.2 #define IO_PWD_DOWN_COUNT 1U

count down

Definition at line 233 of file IO_PWD.h.

7.18.4.3 #define IO_PWD_FALLING_COUNT 2U

count on a falling edge

Definition at line 220 of file IO_PWD.h.

7.18.4.4 #define IO_PWD_FALLING_VAR 3U

falling edge of the input signal is the variable one
Definition at line 155 of file IO_PWD.h.

7.18.4.5 #define IO_PWD_HIGH_TIME 1U

capture the high time of the input signal
Definition at line 136 of file IO_PWD.h.

7.18.4.6 #define IO_PWD_INC_1_COUNT 0x01U

count on any edge of 1st input channel only:

- count on edges of `IO_PWD_00` for 1st incremental interface
- count on edges of `IO_PWD_02` for 2nd incremental interface
- count on edges of `IO_PWD_04` for 3rd incremental interface
- count on edges of `IO_PWD_06` for 4th incremental interface
- count on edges of `IO_PWD_08` for 5th incremental interface
- count on edges of `IO_PWD_10` for 6th incremental interface

Definition at line 196 of file IO_PWD.h.

7.18.4.7 #define IO_PWD_INC_2_COUNT 0x03U

count on any edge of the two input channels
Definition at line 195 of file IO_PWD.h.

7.18.4.8 #define IO_PWD_LOW_TIME 0U

capture the low time of the input signal
Definition at line 135 of file IO_PWD.h.

7.18.4.9 #define IO_PWD_MAX_PULSE_SAMPLES 8U

Definition at line 184 of file IO_PWD.h.

7.18.4.10 #define IO_PWD_NO_PULL 0x03U

fixed pull resistor
Definition at line 170 of file IO_PWD.h.

7.18.4.11 #define IO_PWD_PD_10K 0x00U

10 kOhm pull down

Definition at line 172 of file IO_PWD.h.

7.18.4.12 #define IO_PWD_PD_90 0x02U

90 Ohm pull down

Definition at line 173 of file IO_PWD.h.

7.18.4.13 #define IO_PWD_PERIOD_TIME 2U

capture the high and low time of the input signal

Definition at line 137 of file IO_PWD.h.

7.18.4.14 #define IO_PWD_PU_10K 0x01U

10 kOhm pull up

Definition at line 171 of file IO_PWD.h.

7.18.4.15 #define IO_PWD_RISING_COUNT 1U

count on a rising edge

Definition at line 219 of file IO_PWD.h.

7.18.4.16 #define IO_PWD_RISING_VAR 2U

rising edge of the input signal is the variable one

Definition at line 152 of file IO_PWD.h.

7.18.4.17 #define IO_PWD_UP_COUNT 0U

count up

Definition at line 232 of file IO_PWD.h.

7.18.5 Typedef Documentation**7.18.5.1 typedef struct io_pwd_cnt_conf_ IO_PWD_CNT_CONF**

Edge counter configuration for the Universal PWD inputs.

Stores all relevant configuration parameters for the edge counting mode of Universal PWD inputs.

7.18.5.2 `typedef struct io_pwd_cplx_conf_ IO_PWD_CPLX_CONF`

Complex configuration for the Universal PWD inputs.

Stores all relevant configuration parameters for the complex mode of Universal PWD inputs.

7.18.5.3 `typedef struct io_pwd_cplx_safety_conf_ IO_PWD_CPLX_SAFETY_CONF`

Safety configuration for the Complex PWD inputs.

Stores all relevant safety configuration parameters for the Complex PWD inputs. The internal checker modules verify that this input is in the valid range.

7.18.5.4 `typedef struct io_pwd_inc_conf_ IO_PWD_INC_CONF`

Incremental configuration for the Universal PWD inputs.

Stores all relevant configuration parameters for the incremental mode of Universal PWD inputs.

7.18.5.5 `typedef struct io_pwd_inc_safety_conf_ IO_PWD_INC_SAFETY_CONF`

Safety configuration for the Incremental or Counter PWD inputs.

Stores all relevant safety configuration parameters for the Incremental or counter PWD inputs. The internal checker modules verify that this input is in the valid range.

7.18.5.6 `typedef struct io_pwd_pulse_samples_ IO_PWD_PULSE_SAMPLES`

PWD pulse-width data structure.

stores each captured pulse-width for one measurement.

7.18.5.7 `typedef struct io_pwd_universal_safety_conf_ IO_PWD_UNIVERSAL_SAFETY_CONF`

Safety configuration for the Universal PWD inputs.

Stores all relevant safety configuration parameters for the Universal PWD inputs. The internal checker modules verify that this input is in the valid range.

7.18.6 Function Documentation**7.18.6.1 `IO_ErrorType IO_PWD_ComplexDelInit (ubyte1 timer_channel)`**

Deinitializes a complex PWD input.

Parameters

<i>timer_channel</i>	Timer channel: <ul style="list-style-type: none"> • <code>IO_PWD_00 .. IO_PWD_05</code> • <code>IO_PWD_06 .. IO_PWD_11</code> • <code>IO_PWD_12 .. IO_PWD_19</code>
----------------------	---

Returns

`IO_ErrorType`:

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the given channel is not configured
<code>IO_E_INTERNAL_CSM</code>	an internal error occurred
<code>IO_E_UNKNOWN</code>	an unknown error occurred

7.18.6.2 `IO_ErrorType IO_PWD_ComplexGet (ubyte1 timer_channel, ubyte4 *const frequency, ubyte4 *const pulse_width, bool *const pin_value, IO_PWD_PULSE_SAMPLES *const pulse_samples)`

Get the frequency and the pulse-width from the specified timer channel.

Parameters

	<i>timer_channel</i>	Timer channel: <ul style="list-style-type: none"> • <code>IO_PWD_00 .. IO_PWD_05</code> • <code>IO_PWD_06 .. IO_PWD_11</code> • <code>IO_PWD_12 .. IO_PWD_19</code>
out	<i>frequency</i>	Accumulated frequency in mHz (1/1000 Hz)
out	<i>pulse_width</i>	Accumulated pulse-width in us
out	<i>pin_value</i>	Digital value of the pin
out	<i>pulse_samples</i>	Contains each pulse-width measure sample

Returns

`IO_ErrorType`:

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the given channel is not configured
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist
<code>IO_E_PWD_NOT_FINISHED</code>	not enough edges to accumulate a result
<code>IO_E_PWD_OVERFLOW</code>	a timer overflow occurred
<code>IO_E_PWD_CURRENT_THRESH_HIGH</code>	the last measured current signal was above the threshold of 20 mA

<code>IO_E_PWD_CURRENT_THRESH_LOW</code>	the last measured current signal was below the threshold of 4 mA
<code>IO_E_FET_PROT_ACTIVE</code>	the input FET protection is active and has deactivated the internal switch. It can be reset with <code>IO_PWD_ResetProtection()</code> after the wait time is passed
<code>IO_E_FET_PROT_REENABLE</code>	the input FET protection is ready to be reset with <code>IO_PWD_ResetProtection()</code>
<code>IO_E_FET_PROT_PERMANENT</code>	the input FET is protected permanently because it was already reset more than 10 times
<code>IO_E_NULL_POINTER</code>	a NULL pointer has been passed to the function
<code>IO_E_STARTUP</code>	the input is in the startup phase
<code>IO_E_INTERNAL_CSM</code>	an internal error occurred
<code>IO_E_UNKNOWN</code>	an unknown error occurred

Note

The return values `IO_E_PWD_CURRENT_THRESH_HIGH`, `IO_E_PWD_CURRENT_THRESH_LOW`, `IO_E_FET_PROT_ACTIVE`, `IO_E_FET_PROT_REENABLE` and `IO_E_FET_PROT_PERMANENT` will only be returned in case of a current sensor configuration (`pupd == IO_PWD_PD_90`)

Remarks

- The maximum frequency that can be measured with `IO_PWD_00 .. IO_PWD_11` is 20kHz.
- The maximum frequency that can be measured with `IO_PWD_12 .. IO_PWD_19` is 10kHz.
- The parameter `pin_value` is optional. If not needed, the parameter can be set `NULL` to ignore them.
- If each individual measured pulse-width sample is not needed, the parameter `pulse_samples` should be set to `NULL`

7.18.6.3 `IO_ErrorType IO_PWD_ComplexInit (ubyte1 timer_channel, ubyte1 pulse_mode, ubyte1 freq_mode, ubyte1 capture_count, ubyte1 pupd, IO_PWD_CPLX_SAFETY_CONF const *const safety_conf)`

Setup single timer channel that measures frequency and pulse-width at the same time.

Parameters

<code>timer_channel</code>	Timer channel: <ul style="list-style-type: none"> • <code>IO_PWD_00 .. IO_PWD_05</code> • <code>IO_PWD_06 .. IO_PWD_11</code> • <code>IO_PWD_12 .. IO_PWD_19</code>
<code>pulse_mode</code>	Specifies the pulse mode <ul style="list-style-type: none"> • <code>IO_PWD_HIGH_TIME</code>: configuration to measure pulse-high-time • <code>IO_PWD_LOW_TIME</code>: configuration to measure pulse-low-time • <code>IO_PWD_PERIOD_TIME</code>: configuration to measure pulse-high and low-time (Period)

<i>freq_mode</i>	Specifies the variable edge <ul style="list-style-type: none"> • <code>IO_PWD_RISING_VAR</code>: rising edge is variable this means, frequency is measured on falling edges • <code>IO_PWD_FALLING_VAR</code>: falling edge is variable this means, frequency is measured on rising edges
<i>capture_count</i>	Number of frequency/pulse-width measurements that will be accumulated (1..8)
<i>pupd</i>	Pull up/down interface: <ul style="list-style-type: none"> • <code>IO_PWD_NO_PULL</code>: fixed pull resistor • <code>IO_PWD_PU_10K</code>: Pull up 10 kOhm • <code>IO_PWD_PD_10K</code>: Pull down 10 kOhm • <code>IO_PWD_PD_90</code>: Pull down 90 Ohm (for 7mA/14mA sensors)
in	<i>safety_conf</i>
	Relevant safety configurations for the checker modules

Returns

`IO_ErrorType`:

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_CHANNEL_BUSY</code>	the channel is currently used by another function
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist or is not available on the used ECU variant
<code>IO_E_CH_CAPABILITY</code>	the given channel is not a PWD channel or the used ECU variant does not support this function
<code>IO_E_INVALID_PARAMETER</code>	invalid parameter has been passed
<code>IO_E_DRIVER_NOT_INITIALIZED</code>	the common driver init function has not been called before
<code>IO_E_FPGA_NOT_INITIALIZED</code>	the FPGA has not been initialized
<code>IO_E_INVALID_SAFETY_CONFIG</code>	an invalid safety configuration has been passed
<code>IO_E_DRV_SAFETY_CONF_NOT_CONFIG</code>	global safety configuration is missing
<code>IO_E_DRV_SAFETY_CYCLE_RUNNING</code>	the init function was called after the task begin function
<code>IO_E_INTERNAL_CSM</code>	an internal error occurred
<code>IO_E_UNKNOWN</code>	an unknown error occurred

Remarks

- The supported features depend on the selected channel:
 - `IO_PWD_00 .. IO_PWD_05`:
 - * `pulse_mode`: `IO_PWD_HIGH_TIME`, `IO_PWD_LOW_TIME` or `IO_PWD_PERIOD_TIME`
 - * `freq_mode`: `IO_PWD_RISING_VAR` or `IO_PWD_FALLING_VAR`
 - * `capture_count`: 1..8
 - * `pupd`: `IO_PWD_PU_10K`, `IO_PWD_PD_10K` or `IO_PWD_PD_90`
 - * `safety_conf`: Safety configuration
 - `IO_PWD_06 .. IO_PWD_11`:

- * pulse_mode: `IO_PWD_HIGH_TIME`, `IO_PWD_LOW_TIME` or `IO_PWD_PERIOD_TIME`
- * freq_mode: `IO_PWD_RISING_VAR` or `IO_PWD_FALLING_VAR`
- * capture_count: 1..8
- * pupd: `IO_PWD_PU_10K` or `IO_PWD_PD_10K`
- `IO_PWD_12 .. IO_PWD_19`:
 - * pulse_mode: `IO_PWD_HIGH_TIME`, `IO_PWD_LOW_TIME` or `IO_PWD_PERIOD_TIME`
 - * freq_mode: `IO_PWD_RISING_VAR` or `IO_PWD_FALLING_VAR`

Attention

Passing `IO_PWD_PD_90` as pupd parameter configures the input for a current sensor (7mA/14mA). In this case an additional range check on the current signal will be performed.

Remarks

- A channel that is initialized with this function can retrieve frequency and duty cycle by calling the function: `IO_PWD_ComplexGet()`
- The timing measurement for channels `IO_PWD_00 .. IO_PWD_05` is based on a 27bit timer with a resolution of 0.5 us, therefore the period that shall be measured must be smaller than 67.108.863 us (~ 67 s).
- The timing measurement for channels `IO_PWD_06 .. IO_PWD_11` is hardware limited to 10.000.000 us (= 10 s) with a resolution of 0.5 us. Because those timers work in an accumulating fashion the product of the measured period and the `capture_count` must be below this time.
- The timing measurement for channels `IO_PWD_12 .. IO_PWD_19` is hardware limited to 10.000.000 us (= 10 s) with a resolution of 1 us.
- The maximum frequency that can be measured with `IO_PWD_00 .. IO_PWD_11` is 20kHz.
- The maximum frequency that can be measured with `IO_PWD_12 .. IO_PWD_19` is 10kHz.
- The driver captures exactly as many measurements are given in the parameter `capture_count`, except for `IO_PWD_12 .. IO_PWD_19`.

Note

Until not all configured samples are captured, the driver doesn't return a value.

7.18.6.4 IO_ErrorType IO_PWD_CountDelInit (`ubyte1 count_channel`)

Deinitialize a single counter channel.

Parameters

<code>count_channel</code>	Counter channel: <ul style="list-style-type: none"> • <code>IO_PWD_00 .. IO_PWD_05</code> • <code>IO_PWD_06 .. IO_PWD_11</code>
----------------------------	--

Returns

`IO_ErrorType:`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the given channel is not configured
<code>IO_E_INTERNAL_CSM</code>	an internal error occurred
<code>IO_E_UNKNOWN</code>	an unknown error occurred

7.18.6.5 `IO_ErrorType IO_PWD_CountGet (ubyte1 count_channel, ubyte2 *const count, bool *const pin_value)`

Get the counter value of a counter channel.

Parameters

	<code>count_channel</code>	Counter channel: <ul style="list-style-type: none"> • <code>IO_PWD_00 .. IO_PWD_05</code> • <code>IO_PWD_06 .. IO_PWD_11</code>
out	<code>count</code>	Value of the counter (0..65535)
out	<code>pin_value</code>	Digital value of the pin

Returns

`IO_ErrorType:`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the given channel is not configured
<code>IO_E_NULL_POINTER</code>	a NULL pointer has been passed
<code>IO_E_INTERNAL_CSM</code>	an internal error occurred
<code>IO_E_UNKNOWN</code>	an unknown error occurred

Remarks

The parameter `pin_value` is optional.

If not needed, the parameter can be set `NULL` to ignore them.

7.18.6.6 `IO_ErrorType IO_PWD_CountInit (ubyte1 count_channel, ubyte1 mode, ubyte1 direction, ubyte2 count_init, ubyte1 pupd, IO_PWD_INC_SAFETY_CONF const *const safety_conf)`

Setup a single counter channel.

Parameters

	<i>count_channel</i>	Counter channel: <ul style="list-style-type: none"> • IO_PWD_00 .. IO_PWD_05 • IO_PWD_06 .. IO_PWD_11
	<i>mode</i>	Specify on which edge shall be count <ul style="list-style-type: none"> • IO_PWD_RISING_COUNT: count on a rising edge • IO_PWD_FALLING_COUNT: count on a falling edge • IO_PWD_BOTH_COUNT: count on a both edges
	<i>direction</i>	Specify the counting direction <ul style="list-style-type: none"> • IO_PWD_UP_COUNT: counts up • IO_PWD_DOWN_COUNT: counts down
	<i>count_init</i>	Init value of the counter (0..65535)
	<i>pupd</i>	Pull up/down interface: <ul style="list-style-type: none"> • IO_PWD_PU_10K: Pull up 10 kOhm • IO_PWD_PD_10K: Pull down 10 kOhm
in	<i>safety_conf</i>	Relevant safety configurations for the checker modules

Returns

[IO_ErrorType](#):

Return values

IO_E_OK	everything fine
IO_E_CHANNEL_BUSY	the channel is currently used by another function
IO_E_INVALID_CHANNEL_ID	the given channel id does not exist or is not available on the used ECU variant
IO_E_CH_CAPABILITY	the given channel is not a PWD channel or the used ECU variant does not support this function
IO_E_INVALID_PARAMETER	invalid parameter has been passed
IO_E_DRIVER_NOT_INITIALIZED	the common driver init function has not been called before
IO_E_FPGA_NOT_INITIALIZED	the FPGA has not been initialized
IO_E_INVALID_SAFETY_CONFIG	an invalid safety configuration has been passed
IO_E_DRV_SAFETY_CONF_NOT_CONFIG	global safety configuration is missing
IO_E_DRV_SAFETY_CYCLE_RUNNING	the init function was called after the task begin function
IO_E_INTERNAL_CSM	an internal error occurred
IO_E_UNKNOWN	an unknown error occurred

Remarks

- The parameter `safety_conf` is only supported for the PWD channels `IO_PWD_00 .. IO_PWD_05`

7.18.6.7 IO_ErrorType IO_PWD_CountSet (`ubyte1 count_channel`, `ubyte2 count`)

Set the counter value of a counter channel.

Parameters

<code>count_channel</code>	Counter channel: • <code>IO_PWD_00 .. IO_PWD_05</code> • <code>IO_PWD_06 .. IO_PWD_11</code>
<code>count</code>	Value of the counter (0..65535)

Returns

`IO_ErrorType`:

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the given channel is not configured
<code>IO_E_INTERNAL_CSM</code>	an internal error occurred
<code>IO_E_UNKNOWN</code>	an unknown error occurred

7.18.6.8 IO_ErrorType IO_PWD_GetCurrent (`ubyte1 pwd_channel`, `ubyte2 *const current`, `bool *const fresh`)

Get the current feedback of a timer input channel.

Parameters

	<code>pwd_channel</code>	PWD channel: • <code>IO_PWD_00 .. IO_PWD_05</code>
out	<code>current</code>	Measured current in uA. Range: 0..20000 (0mA..20.000mA)
out	<code>fresh</code>	Indicates if a new value is available since the last call. • <code>TRUE</code> : Value in "current" is valid • <code>FALSE</code> : No new value available.

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist
<code>IO_E_NULL_POINTER</code>	a NULL pointer has been passed to the function
<code>IO_E_PWD_CURRENT_THRESH_HIGH</code>	the last measured current signal was above the threshold of 20 mA
<code>IO_E_PWD_CURRENT_THRESH_LOW</code>	the last measured current signal was below the threshold of 4 mA
<code>IO_E_FET_PROT_ACTIVE</code>	the input FET protection is active and has deactivated the internal switch. It can be reset with <code>IO_PWD_ResetProtection()</code> after the wait time is passed
<code>IO_E_FET_PROT_RESETABLE</code>	the input FET protection is ready to be reset with <code>IO_PWD_ResetProtection()</code>
<code>IO_E_FET_PROT_PERMANENT</code>	the input FET is protected permanently because it was already reset more than 10 times
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the given channel is not configured
<code>IO_E_CH_CAPABILITY</code>	the given channel is not a PWD channel
<code>IO_E_STARTUP</code>	the input is in the startup phase
<code>IO_E_CHANNEL_BUSY</code>	the channel is currently used by another function
<code>IO_E_REFERENCE</code>	the internal reference voltage is out of range
<code>IO_E_UNKNOWN</code>	an unknown error occurred

Note

The return value `IO_E_CHANNEL_BUSY` will only be returned in case of a voltage sensor configuration (`pupd != IO_PWD_PD_90`)

Remarks

If there is no new current value available (for example the function `IO_PWD_GetCurrent()` gets called more frequently than the AD sampling) the flag `fresh` will be set to `FALSE`.

7.18.6.9 `IO_ErrorType IO_PWD_GetVoltage(ubyte1 pwd_channel, ubyte2 *const voltage, bool *const fresh)`

Get the voltage feedback of a timer input channel.

Parameters

	<code>pwd_channel</code>	PWD channel: <ul style="list-style-type: none"> • <code>IO_PWD_00 .. IO_PWD_05</code> • <code>IO_PWD_06 .. IO_PWD_11</code>
out	<code>voltage</code>	Measured voltage in mV. Range: 0..32000 (0V..32.000V)
out	<code>fresh</code>	Indicates if a new value is available since the last call. <ul style="list-style-type: none"> • <code>TRUE</code>: Value in "voltage" is valid • <code>FALSE</code>: No new value available.

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist
<code>IO_E_NULL_POINTER</code>	a NULL pointer has been passed to the function
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the given channel is not configured
<code>IO_E_CH_CAPABILITY</code>	the given channel is not a PWD channel
<code>IO_E_STARTUP</code>	the input is in the startup phase
<code>IO_E_CHANNEL_BUSY</code>	the channel is currently used by another function
<code>IO_E_REFERENCE</code>	the internal reference voltage is out of range
<code>IO_E_UNKNOWN</code>	an unknown error occurred

Note

The return value `IO_E_CHANNEL_BUSY` will only be returned in case of a current sensor configuration (`pupd == IO_PWD_PD_90`)

Remarks

If there is no new voltage value available (for example the function `IO_PWD_GetVoltage()` gets called more frequently than the AD sampling) the flag `fresh` will be set to `FALSE`.

7.18.6.10 `IO_ErrorType IO_PWD_IncDeInit (ubyte1 inc_channel)`

Deinitialize an incremental interface.

Parameters

<code>inc_channel</code>	Channel of the incremental interface: <ul style="list-style-type: none"> • <code>IO_PWD_00 .. IO_PWD_05</code> • <code>IO_PWD_06 .. IO_PWD_11</code>
--------------------------	---

Returns

`IO_ErrorType`:

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the given channel is not configured
<code>IO_E_INTERNAL_CSM</code>	an internal error occurred
<code>IO_E_UNKNOWN</code>	an unknown error occurred

Note

- The function can be called with either the 1st or the 2nd channel of the incremental interface.

7.18.6.11 IO_ErrorType IO_PWD_IncGet (ubyte1 *inc_channel*, ubyte2 *const *count*, bool *const *pin_value_0*, bool *const *pin_value_1*)

Get the counter value of an incremental interface.

Parameters

	<i>inc_channel</i>	Channel of the incremental interface:
		<ul style="list-style-type: none"> • <code>IO_PWD_00 .. IO_PWD_05</code> • <code>IO_PWD_06 .. IO_PWD_11</code>
out	<i>count</i>	Value of the incremental counter (0..65535)
out	<i>pin_value_0</i>	Digital value of pin 0 (1st channel)
out	<i>pin_value_1</i>	Digital value of pin 1 (2nd channel)

Returns

`IO_ErrorType`:

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the given channel is not configured
<code>IO_E_NULL_POINTER</code>	a NULL pointer has been passed
<code>IO_E_INTERNAL_CSM</code>	an internal error occurred
<code>IO_E_UNKNOWN</code>	an unknown error occurred

Note

- The function can be called with either the 1st or the 2nd channel of the incremental interface.
- The incremental interface will decrement when the 1st channel is leading and increment when the 2nd channel is leading.

Remarks

- The parameters `pin_value_0` and `pin_value_1` are optional. If not needed, these parameters can be set `NULL` to ignore them.

7.18.6.12 IO_ErrorType IO_PWD_IncInit (ubyte1 *inc_channel*, ubyte1 *mode*, ubyte2 *count_init*, ubyte1 *pupd*, IO_PWD_INC_SAFETY_CONF const *const *safety_conf*)

Setup a single incremental interface.

Parameters

	<i>inc_channel</i>	Channel of the incremental interface: <ul style="list-style-type: none"> • IO_PWD_00 .. IO_PWD_05 • IO_PWD_06 .. IO_PWD_11
	<i>mode</i>	Defines the counter behavior <ul style="list-style-type: none"> • IO_PWD_INC_2_COUNT: Counts up/down on any edge of the two input channels • IO_PWD_INC_1_COUNT: Counts up/down on any edge of the 1st input channel only: <ul style="list-style-type: none"> – count on IO_PWD_00 for 1st incremental interface – count on IO_PWD_02 for 2nd incremental interface – count on IO_PWD_04 for 3rd incremental interface – count on IO_PWD_06 for 4th incremental interface – count on IO_PWD_08 for 5th incremental interface – count on IO_PWD_10 for 6th incremental interface
	<i>count_init</i>	Init value of the incremental counter (0..65535)
	<i>pupd</i>	Pull up/down interface: <ul style="list-style-type: none"> • IO_PWD_PU_10K: Pull up 10 kOhm • IO_PWD_PD_10K: Pull down 10 kOhm
in	<i>safety_conf</i>	Relevant safety configurations for the checker modules

Returns

[IO_ErrorType](#):

Return values

IO_E_OK	everything fine
IO_E_INVALID_CHANNEL_ID	the given channel id does not exist or is not available on the used ECU variant
IO_E_CH_CAPABILITY	the given channel is not capable for an incremental interface or the used ECU variant does not support this function
IO_E_CHANNEL_NOT_CONFIGURED	the given channel is not configured
IO_E_CHANNEL_BUSY	the channel is currently used by another function
IO_E_INVALID_PARAMETER	invalid parameter has been passed
IO_E_DRIVER_NOT_INITIALIZED	the common driver init function has not been called before
IO_E_FPGA_NOT_INITIALIZED	the FPGA has not been initialized
IO_E_INVALID_SAFETY_CONFIG	an invalid safety configuration has been passed
IO_E_DRV_SAFETY_CONF_NOT_CONFIG	global safety configuration is missing
IO_E_DRV_SAFETY_CYCLE_RUN-NING	the init function was called after the task begin function
IO_E_INTERNAL_CSM	an internal error occurred
IO_E_UNKNOWN	an unknown error occurred

Remarks

- The parameter `safety_conf` is only supported for the PWD channels `IO_PWD_00 .. IO_PWD_05`
- A channel that is initialized with this function can retrieve the counter value: `IO_PWD_IncGet()`
- Two PWD channels are used (needed) for one incremental interface.
The pairs are defined in the following order:
 - `IO_PWD_00` and `IO_PWD_01` define the 1st incremental interface.
 - `IO_PWD_02` and `IO_PWD_03` define the 2nd incremental interface.
 - `IO_PWD_04` and `IO_PWD_05` define the 3rd incremental interface.
 - `IO_PWD_06` and `IO_PWD_07` define the 4th incremental interface.
 - `IO_PWD_08` and `IO_PWD_09` define the 5th incremental interface.
 - `IO_PWD_10` and `IO_PWD_11` define the 6th incremental interface.

Note

- The function initializes both PWD channels of the incremental interface, independently if the function gets called with the 1st or 2nd channel belonging to an incremental interface.
- The incremental interface will decrement when the 1st channel is leading and increment when the 2nd channel is leading.

7.18.6.13 `IO_ErrorType IO_PWD_IncSet(ubyte1 inc_channel, ubyte2 count)`

Set the counter value of an incremental interface.

Parameters

<code>inc_channel</code>	Channel of the incremental interface: <ul style="list-style-type: none"> <code>IO_PWD_00 .. IO_PWD_05</code> <code>IO_PWD_06 .. IO_PWD_11</code>
<code>count</code>	Value of the incremental counter (0..65535)

Returns

`IO_ErrorType`:

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the given channel is not configured
<code>IO_E_INTERNAL_CSM</code>	an internal error occurred
<code>IO_E_UNKNOWN</code>	an unknown error occurred

Note

- The function can be called with either the 1st or the 2nd channel of the incremental interface.

7.18.6.14 IO_ErrorType IO_PWD_ResetProtection (ubyte1 *pwd_channel*, ubyte1 *const *reset_cnt*)

Reset the FET protection for a timer input channel.

Parameters

	<i>pwd_channel</i>	PWD channel: • <code>IO_PWD_00 .. IO_PWD_05</code>
out	<i>reset_cnt</i>	Protection reset counter. Indicates how often the application already reset the protection.

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist
<code>IO_E_FET_PROT_NOT_ACTIVE</code>	no input FET protection is active
<code>IO_E_FET_PROT_PERMANENT</code>	the input FET is protected permanently because it was already reset more than 10 times
<code>IO_E_FET_PROT_WAIT</code>	the input FET protection can not be reset, as the wait time of 1s is not already passed
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the given channel is not configured
<code>IO_E_CH_CAPABILITY</code>	the given channel is not a PWD channel
<code>IO_E_CHANNEL_BUSY</code>	the channel is currently used by another function
<code>IO_E_UNKNOWN</code>	an unknown error occurred

Note

The return value `IO_E_CHANNEL_BUSY` will only be returned in case of a voltage sensor configuration (`pupd != IO_PWD_PD_90`)

Remarks

When re-enabling the channel after it has been in protection state, it will do a transition to the startup state. Therefore the functions `IO_PWD_ComplexGet()` and `IO_PWD_GetCurrent()` will return `IO_E_STARTUP` for a certain time.

7.18.6.15 IO_ErrorType IO_PWD_UniversalDelInit (ubyte1 *timer_channel*)

Deinitialize a single universal timer channel.

Parameters

<i>timer_channel</i>	Timer channel: • IO_PWD_00 .. IO_PWD_05
----------------------	--

Returns

IO_ErrorType:

Return values

IO_E_OK	everything fine
IO_E_INVALID_CHANNEL_ID	the given channel id does not exist
IO_E_CHANNEL_NOT_CONFIGURED	the given channel is not configured
IO_E_INTERNAL_CSM	an internal error occurred
IO_E_UNKNOWN	an unknown error occurred

7.18.6.16 IO_ErrorType IO_PWD_UniversalGet (ubyte1 *timer_channel*, ubyte4 *const *frequency*, ubyte4 *const *pulse_width*, IO_PWD_PULSE_SAMPLES *const *pulse_samples*, ubyte2 *const *edge_count*, ubyte2 *const *inc_count*, bool *const *primary_pin_value*, bool *const *secondary_pin_value*)

Get the measurement results from the specified universal timer channel.

Parameters

	<i>timer_channel</i>	Timer channel: • IO_PWD_00 .. IO_PWD_05
out	<i>frequency</i>	Accumulated frequency in mHz (1/1000 Hz)
out	<i>pulse_width</i>	Accumulated pulse-width in us
out	<i>pulse_samples</i>	Contains each pulse-width measure sample
out	<i>edge_count</i>	Value of the edge counter (0..65535)
out	<i>inc_count</i>	Value of the incremental counter (0..65535)
out	<i>primary_pin_value</i>	Digital value of the primary pin
out	<i>secondary_pin_value</i>	Digital value of the secondary pin

Returns

IO_ErrorType:

Return values

IO_E_OK	everything fine
IO_E_CHANNEL_NOT_CONFIGURED	the given channel is not configured
IO_E_INVALID_CHANNEL_ID	the given channel id does not exist
IO_E_PWD_NOT_FINISHED	not enough edges to accumulate a result
IO_E_PWD_OVERFLOW	a timer overflow occurred
IO_E_PWD_CURRENT_THRESH_HIGH	the last measured current signal was above the threshold of 20 mA

<code>IO_E_PWD_CURRENT_THRESH_LOW</code>	the last measured current signal was below the threshold of 4 mA
<code>IO_E_FET_PROT_ACTIVE</code>	the input FET protection is active and has deactivated the internal switch. It can be reset with <code>IO_PWD_ResetProtection()</code> after the wait time is passed
<code>IO_E_FET_PROT_REENABLE</code>	the input FET protection is ready to be reset with <code>IO_PWD_ResetProtection()</code>
<code>IO_E_FET_PROT_PERMANENT</code>	the input FET is protected permanently because it was already reset more than 10 times
<code>IO_E_NULL_POINTER</code>	a NULL pointer has been passed to the function
<code>IO_E_STARTUP</code>	the input is in the startup phase
<code>IO_E_INTERNAL_CSM</code>	an internal error occurred
<code>IO_E_UNKNOWN</code>	an unknown error occurred

Note

- The function can be called with either the primary or the secondary channel of the incremental interface.
- The return values `IO_E_PWD_CURRENT_THRESH_HIGH`, `IO_E_PWD_CURRENT_THRESH_LOW`, `IO_E_FET_PROT_ACTIVE`, `IO_E_FET_PROT_REENABLE` and `IO_E_FET_PROT_PERMANENT` will only be returned in case of a current sensor configuration (`pupd == IO_PWD_PD_90`)

Remarks

- The maximum frequency that can be measured with `IO_PWD_00 .. IO_PWD_05` is 20kHz.
- The parameters `frequency`, `pulse_width`, `pulse_samples`, `edge_count`, `inc_count`, `primary_pin_value` and `secondary_pin_value` are optional. If not needed, these parameters can be set to `NULL` to ignore them.
- If each individual measured pulse-width sample is not needed, the parameter `pulse_samples` should be set to `NULL`.

7.18.6.17 `IO_ErrorType IO_PWD_UniversalInit (ubyte1 timer_channel, IO_PWD_CPLX_CONF const *const cplx_conf, IO_PWD_CNT_CONF const *const cnt_conf, IO_PWD_INC_CONF const *const inc_conf, ubyte1 pupd, IO_PWD_UNIVERSAL_SAFETY_CONF const *const safety_conf)`

Setup a single universal timer channel.

Parameters

	<code>timer_channel</code>	Timer channel: • <code>IO_PWD_00 .. IO_PWD_05</code>
in	<code>cplx_conf</code>	Complex configuration
in	<code>cnt_conf</code>	Edge counter configuration
in	<code>inc_conf</code>	Incremental counter configuration

	<i>pupd</i>	Pull up/down interface: <ul style="list-style-type: none"> • IO_PWD_PU_10K: Pull up 10 kOhm • IO_PWD_PD_10K: Pull down 10 kOhm • IO_PWD_PD_90: Pull down 90 Ohm (for 7mA/14mA sensors)
in	<i>safety_conf</i>	Relevant safety configurations for the checker modules

Returns

[IO_ErrorType](#):

Return values

IO_E_OK	everything fine
IO_E_CHANNEL_BUSY	the channel is currently used by another function
IO_E_INVALID_CHANNEL_ID	the given channel id does not exist or is not available on the used ECU variant
IO_E_CH_CAPABILITY	the given channel is not a PWD channel or the used ECU variant does not support this function
IO_E_INVALID_PARAMETER	invalid parameter has been passed
IO_E_DRIVER_NOT_INITIALIZED	the common driver init function has not been called before
IO_E_INVALID_SAFETY_CONFIG	an invalid safety configuration has been passed
IO_E_DRV_SAFETY_CONF_NOT_CONFIG	global safety configuration is missing
IO_E_DRV_SAFETY_CYCLE_RUNNING	the init function was called after the task begin function
IO_E_INTERNAL_CSM	an internal error occurred
IO_E_UNKNOWN	an unknown error occurred

Attention

Passing [IO_PWD_PD_90](#) as *pupd* parameter configures the input for a current sensor (7mA/14mA). In this case an additional range check on the current signal will be performed. Note that this setting is invalid if the input is also configured for edge and/or incremental counter mode. If a channel is configured for incremental mode, both the primary and secondary channel are redundantly configured using the provided complex, edge counter and safety configuration. Thus, the application must periodically call [IO_PWD_UniversalGet\(\)](#) for both channels if either the complex or edge counter mode is configured for safety.

Remarks

- The supported features depend on the selected channel:
 - [IO_PWD_00 .. IO_PWD_05](#):
 - * *pulse_mode*: [IO_PWD_HIGH_TIME](#), [IO_PWD_LOW_TIME](#) or [IO_PWD_PERIOD_TIME](#)
 - * *freq_mode*: [IO_PWD_RISING_VAR](#) or [IO_PWD_FALLING_VAR](#)
 - * *capture_count*: 1..8
 - * *pupd*: [IO_PWD_PU_10K](#), [IO_PWD_PD_10K](#) or [IO_PWD_PD_90](#)
 - * *safety_conf*: Safety configuration

- Parameter `safety_conf` can be set to NULL in order to initialize the input as non-safety-critical. To configure the input as safety-critical, parameter `safety_conf` must provide exactly one valid (i.e., not NULL) safety configuration. Providing zero or multiple safety configurations or a safety configuration for a mode that is not configured will fail with return code `IO_E_INVALID_SAFETY_CONFIG`.
- A channel that is initialized with this function can retrieve its measurement results by calling the function: `IO_PWD_UniversalGet()`
- The timing measurement for channels `IO_PWD_00 .. IO_PWD_05` is based on a 27bit timer with a resolution of 0.5 us, therefore the period that shall be measured must be smaller than 67.108.863 us (~ 67 s).
- The maximum frequency that can be measured with `IO_PWD_00 .. IO_PWD_05` is 20kHz.

Note

- For complex mode, the driver doesn't return a value as long as not all configured samples are captured.

7.18.6.18 `IO_ErrorType IO_PWD_UniversalSet (ubyte1 timer_channel, ubyte2 const *const edge_count, ubyte2 const *const inc_count)`

Set the counter values of an universal timer channel.

Parameters

	<i>timer_channel</i>	Timer channel: • <code>IO_PWD_00 .. IO_PWD_05</code>
in	<i>edge_count</i>	Value of the edge counter (0..65535)
in	<i>inc_count</i>	Value of the incremental counter (0..65535)

Returns

`IO_ErrorType`:

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the given channel is not configured
<code>IO_E_INTERNAL_CSM</code>	an internal error occurred
<code>IO_E_UNKNOWN</code>	an unknown error occurred

Note

- When configured for incremental mode, the function can be called with either the primary or the secondary channel.

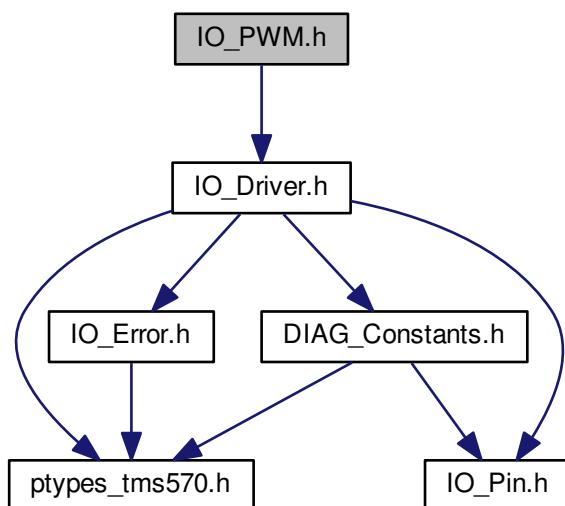
Remarks

- The parameters `edge_count` and `inc_count` are optional. If not needed, these parameters can be set to `NULL` to ignore them.

7.19 IO_PWM.h File Reference

IO Driver functions for PWM channels.

Include dependency graph for IO_PWM.h:



Data Structures

- struct `io_pwm_current_queue_`
PWM current measurement queue.
- struct `io_pwm_safety_conf_`
Safety configuration for the PWM outputs.

Typedefs

- typedef struct `io_pwm_current_queue_` `IO_PWM_CURRENT_QUEUE`
PWM current measurement queue.
- typedef struct `io_pwm_safety_conf_` `IO_PWM_SAFETY_CONF`
Safety configuration for the PWM outputs.

Functions

- `IO_ErrorType IO_PWM_DeInit (ubyte1 pwm_channel)`
Deinitializes a PWM output.
- `IO_ErrorType IO_PWM_GetCur (ubyte1 pwm_channel, ubyte2 *const current, bool *const fresh)`
Returns the measured current of the given channel.
- `IO_ErrorType IO_PWM_GetCurQueue (ubyte1 pwm_channel, IO_PWM_CURRENT_QUEUE *const current_queue)`
Returns the measured current values since the last call of the given channel.
- `IO_ErrorType IO_PWM_Init (ubyte1 pwm_channel, ubyte2 frequency, bool polarity, bool diag_margin, IO_PWM_SAFETY_CONF const *const safety_conf)`
Setup a single PWM output.
- `IO_ErrorType IO_PWM_InitWithLowside (ubyte1 pwm_channel, ubyte2 frequency, bool polarity, bool diag_margin, IO_PWM_SAFETY_CONF const *const safety_conf)`
Setup a single PWM output (which can be connected to a low side channel).
- `IO_ErrorType IO_PWM_ResetProtection (ubyte1 pwm_channel, ubyte1 *const reset_cnt)`
Reset the output protection for a PWM channel.
- `IO_ErrorType IO_PWM_ResolveOpenLoadShortCircuit (ubyte1 pwm_channel)`
Resolve an open load or short circuit to battery voltage condition for a PWM channel.
- `IO_ErrorType IO_PWM_SetDuty (ubyte1 pwm_channel, ubyte2 duty_cycle, ubyte2 *const high_time_fb, ubyte2 *const period_fb)`
Set the duty cycle for a PWM channel.

Size of current queue

Size of the queue holding the measured values of the equidistant current measurement

- `#define IO_PWM_CURRENT_QUEUE_MAX 6U`

7.19.1 Detailed Description

IO Driver functions for PWM channels.

Contains all service functions for the PWM (Pulse Width Modulation). Up to 36 channels can be configured: `IO_PWM_00 .. IO_PWM_35`

- All PWM outputs have timer feedback
- All PWM outputs have current measurement
- The PWM outputs will be switched off immediately if the continuous current is above 4A. The step function for retrieving the current is `IO_PWM_GetCur()` or `IO_PWM_GetCurQueue()`.

PWM-API Usage:

- [Examples for PWM API functions](#)

7.19.2 PWM output protection

Each PWM output is individually protected against overcurrent. Whenever a fatal overcurrent situation (i.e., a current greater than 5.5A) is detected on any safe or unsafe PWM output, the output

is disabled by software latest within 7ms. Note that disabled by software means that the duty cycle of the output signal is set to 0% but the states of the safety switch and PWM power stage remain unchanged.

When entering the protection state, the PWM output has to remain in this state for at least 10s. After this wait time the PWM output can be reenabled via function `IO_PWM_ResetProtection()`. Note that the number of reenabling operations for a single PWM output is limited to 10. Refer to function `IO_PWM_ResetProtection()` for more information on how to reenable a PWM output.

7.19.3 PWM code examples

Examples for using the PWM API

7.19.3.1 PWM initialization example

```
// Setup a PWM output
IO_PWM_Init(IO_PWM_00,
    100,           // frequency is 100 Hz
    TRUE,          // positive polarity
    FALSE,         // no diag margin
    NULL);         // not safety critical

// Setup a PWM output
IO_PWM_Init(IO_PWM_01,
    166,           // frequency is 166 Hz
    TRUE,          // positive polarity
    FALSE,         // no diag margin
    NULL);         // not safety critical

// Setup a PWM output
IO_PWM_Init(IO_PWM_12,
    200,           // frequency is 200 Hz
    TRUE,          // positive polarity
    FALSE,         // no diag margin
    NULL);         // not safety critical
```

7.19.3.2 PWM task example

```
ubyte2 curr;
bool new;

IO_PWM_SetDuty(IO_PWM_00,
    0x8000,        // set duty cycle to 50%
    NULL,          // feedback high time measurement ignored
    NULL);         // feedback period measurement ignored

IO_PWM_GetCur(IO_PWM_00,   // read current value of PWM output
    &curr,         // variable to store current
    &new);         // variable to store information if a current value is available
```

7.19.4 Macro Definition Documentation

7.19.4.1 #define IO_PWM_CURRENT_QUEUE_MAX 6U

maximum number of items in the current queue

Definition at line 112 of file IO_PWM.h.

7.19.5 Typedef Documentation

7.19.5.1 typedef struct io_pwm_current_queue_ IO_PWM_CURRENT_QUEUE

PWM current measurement queue.

Stores results of the equidistant current measurement.

The queue holds all current measurement since the last retrieval via the step function `IO_PWM_GetCur()` or `IO_PWM_GetCurQueue()`.

7.19.5.2 typedef struct io_pwm_safety_conf_ IO_PWM_SAFETY_CONF

Safety configuration for the PWM outputs.

Stores all relevant safety configuration parameters for the PWM outputs. The internal checker modules verifies that this inputs contain valid values.

7.19.6 Function Documentation

7.19.6.1 IO_ErrorType IO_PWM_DelInit (`ubyte1 pwm_channel`)

Deinitializes a PWM output.

Allows the re-initialization of the output by other functions

Parameters

<code>pwm_channel</code>	PWM channel, one of: • <code>IO_PWM_00 .. IO_PWM_35</code>
--------------------------	---

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the given channel is not configured
<code>IO_E_CH_CAPABILITY</code>	the given channel has no PWM capability
<code>IO_E_DRIVER_NOT_INITIALIZED</code>	the common driver init function has not been called before
<code>IO_E_UNKNOWN</code>	an unknown error occurred

7.19.6.2 IO_ErrorType IO_PWM_GetCur (ubyte1 *pwm_channel*, ubyte2 *const *current*, bool *const *fresh*)

Returns the measured current of the given channel.

Parameters

	<i>pwm_channel</i>	PWM channel, one of: • IO_PWM_00 .. IO_PWM_35
out	<i>current</i>	Measured current in mA Range: 0..7500 (0A .. 7.500A)
out	<i>fresh</i>	Indicates if new values are available since the last call. • TRUE : Value in "current" is valid • FALSE : No new value available.

Returns

[IO_ErrorType](#)

Return values

IO_E_OK	everything fine
IO_E_CHANNEL_NOT_CONFIGURED	the given channel is not configured
IO_E_INVALID_CHANNEL_ID	the given channel id does not exist
IO_E_CH_CAPABILITY	the given channel is not a PWM channel
IO_E_CM_CALIBRATION	the zero current calibration failed
IO_E_NULL_POINTER	a NULL pointer has been passed to the function
IO_E_DRIVER_NOT_INITIALIZED	the common driver init function has not been called before
IO_E_REFERENCE	the internal reference voltage is out of range
IO_E_UNKNOWN	an unknown error occurred

Attention

The current measurement is equidistant which means that the sampling happens synchronous to the PWM period. Every 1ms a current value will be captured. The captured current values will be averaged over the time of one period of the PWM signal and then provided to the application.

Remarks

- When the function [IO_PWM_GetCur\(\)](#) is called, the internal queue holding the values of the current measurement is flushed. If the function is called more than once in a cycle it may or may not deliver new values, depending on how many values the equidistant current measurement has sampled since the last call.
- If there is no new current value available (for example the function [IO_PWM_GetCur\(\)](#) gets called more frequently than the PWM period) the flag *fresh* will be set to [FALSE](#).
- If the functions [IO_PWM_GetCurQueue\(\)](#) and [IO_PWM_GetCur\(\)](#) are called after each other, only the first function will deliver a current value because both functions will empty the internal measurement queue.

7.19.6.3 IO_ErrorType IO_PWM_GetCurQueue (*ubyte1 pwm_channel*, *IO_PWM_CURRENT_QUEUE *const current_queue*)

Returns the measured current values since the last call of the given channel.

Parameters

	<i>pwm_channel</i>	PWM channel, one of: • IO_PWM_00 .. IO_PWM_35
out	<i>current_queue</i>	Queue holding the current values since the last call of the step function including a queue overrun flag and a counter for the quantity of available values

Returns

[IO_ErrorType](#)

Return values

IO_E_OK	everything fine
IO_E_CHANNEL_NOT_CONFIGURED	the given channel is not configured
IO_E_INVALID_CHANNEL_ID	the given channel id does not exist
IO_E_CH_CAPABILITY	the given channel is not a PWM channel
IO_E_CM_CALIBRATION	the zero current calibration failed
IO_E_NULL_POINTER	a NULL pointer has been passed to the function
IO_E_DRIVER_NOT_INITIALIZED	the common driver init function has not been called before
IO_E_REFERENCE	the internal reference voltage is out of range
IO_E_UNKNOWN	an unknown error occurred

Attention

The current measurement is equidistant which means that the sampling happens synchronous to the PWM period. Every 1ms a current value will be captured. The captured current values will be averaged over the time of one period of the PWM signal and then provided to the application in a queue. The queue storage was chosen to avoid the loss of any measurement value if the user application runs asynchronous to the current measurement or if its cycle time is lower than the PWM period. (Size of queue: [IO_PWM_CURRENT_QUEUE_MAX](#), Queue data type: [IO_PWM_CURRENT_QUEUE](#))

Remarks

- When the function [IO_PWM_GetCurQueue\(\)](#) is called, the internal queue holding the values of the current measurement is flushed. If the function is called more than once in a cycle it may or may not deliver new values, depending on how many values the equidistant current measurement has sampled since the last call.
- If there is no new current value available (for example the function [IO_PWM_GetCurQueue\(\)](#) gets called more frequently than the PWM period) the value `count` in the structure `current_queue` will be 0.

- If the functions `IO_PWM_GetCurQueue()` and `IO_PWM_GetCur()` are called after each other, only the first function will deliver a current value because both functions will empty the internal measurement queue.

7.19.6.4 IO_ErrorType IO_PWM_Init (`ubyte1 pwm_channel, ubyte2 frequency, bool polarity, bool diag_margin, IO_PWM_SAFETY_CONF const *const safety_conf`)

Setup a single PWM output.

Parameters

	<i>pwm_channel</i>	PWM channel, one of: • <code>IO_PWM_00 .. IO_PWM_35</code>
	<i>frequency</i>	PWM frequency (50Hz .. 1000Hz, only predefined frequencies with a period of an integral multiple of 1ms, 0.5ms or 0.25ms are possible)
	<i>polarity</i>	Polarity of output signal • <code>FALSE</code> : Low output signal is variable • <code>TRUE</code> : High output signal is variable
	<i>diag_margin</i>	Indicate if a margin should be applied or not. • <code>TRUE</code> : margin is on • <code>FALSE</code> : no margin will be applied • If a PWM channel is configured safety relevant this parameter has to be set to <code>TRUE</code>
in	<i>safety_conf</i>	Safety configuration of the PWM channel. The <code>low_side_channel</code> field is evaluated only with <code>IO_PWM_InitWithLowside()</code> . If a safety configuration is passed, the parameter <code>diag_margin</code> will be forced to <code>TRUE</code> .

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_CHANNEL_BUSY</code>	the PWM output channel or the timer input channel is currently used by another function
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist or is not available on the used ECU variant
<code>IO_E_CH_CAPABILITY</code>	the given channel is not a PWM channel or the used ECU variant does not support this function
<code>IO_E_INVALID_PARAMETER</code>	a given parameter is out of range
<code>IO_E_CM_CALIBRATION</code>	the zero current calibration failed
<code>IO_E_DRIVER_NOT_INITIALIZED</code>	the common driver init function has not been called before
<code>IO_E_FPGA_NOT_INITIALIZED</code>	the FPGA has not been initialized
<code>IO_E_INVALID_SAFETY_CONFIG</code>	an invalid safety configuration has been passed
<code>IO_E_DRV_SAFETY_CONF_NOT_CONFIG</code>	global safety configuration is missing

<i>IO_E_DRV_SAFETY_CYCLE_RUN-NING</i>	the init function was called after the task begin function
<i>IO_E_UNKNOWN</i>	an unknown error occurred

Remarks

- For the equidistant current measurement to work properly only certain frequencies are supported:

Available frequencies:		
Period[ms]	Frequency[Hz]	Current sample period[ms]
1	1000	1
1.25	800	5
1.5	666 (666.6666)	3
1.75	571 (571.4285)	7
2	500	2
2.25	444 (444.4444)	9
2.5	400	5
2.75	363 (363.6363)	11
3	333 (333.3333)	3
3.25	307 (307.6923)	13
3.5	285 (285.7142)	7
3.75	266 (266.6666)	15
4	250	4
4.25	235 (235.2941)	17
4.5	222 (222.2222)	9
4.75	210 (210.5263)	19
5	200	5
5.5	181 (181.8181)	11
6	166 (166.6666)	6
6.5	153 (153.8461)	13
7	142 (142.8571)	7
7.5	133 (133.3333)	15
8	125	8
8.5	117 (117.6470)	17
9	111 (111.1111)	9
9.5	105 (105.2631)	19
10	100	10
11	90 (90.9090)	11
12	83 (83.3333)	12
13	76 (76.9230)	13
14	71 (71.4285)	14
15	66 (66.6666)	15
16	62 (62.5)	16
17	58 (58.8235)	17
18	55 (55.5555)	18
19	52 (52.6315)	19
20	50	20

- If you select an unavailable frequency within the allowed range, the next higher available frequency will be used. For example, if 180 Hz is specified, 181.8 Hz will be used.
- The associated timer loopback channel will also be configured for open load and short circuit detection.
- The associated current measurement will also be configured.
- The duty cycle cannot exceed the margin of 100us(lower boundary) and 200us(upper boundary) used for diagnostic if the parameter diag_margin is **TRUE**. This mode is important for hydraulic coils. If the parameter diag_margin is **FALSE**, no duty cycle range margin will be applied.
- All PWM channels have their own frequency time base.
- If safety_conf != **NULL**, the internal checker modules check the given channels against the current parameter in safety_conf, and the period and duty cycle feedback against the output. For more details about the current checking refer to the definition of **IO_PWM_SAFETY_CONF**.
- If safety_conf != **NULL**, the parameter diag_margin is forced to **TRUE** to allow diagnostics
- Static friction and stiction can cause a hysteresis and make the control of a hydraulic valve erratic and unpredictable. In order to counteract these hysteresis effects, small vibrations about the desired position shall be created in the valve. This constantly breaks the static friction ensuring that it will move even with small input changes, and the effects of hysteresis are average out. A proper setting of PWM frequency according to the resonance frequency of the actuator allows to adjust this desired small vibration, low enough in amplitude to prevent noticeable oscillations on the hydraulic output but sufficient high to prevent friction. The PWM frequency can be set in the range of 50 .. 1000Hz, a typical range for hydraulic valves to operate without friction is 90 .. 160Hz.

7.19.6.5 IO_ErrorType IO_PWM_InitWithLowside (ubyte1 *pwm_channel*, ubyte2 *frequency*, bool *polarity*, bool *diag_margin*, IO_PWM_SAFETY_CONF const *const *safety_conf*)

Setup a single PWM output (which can be connected to a low side channel)

Parameters

<i>pwm_channel</i>	PWM channel, one of: • IO_PWM_00 .. IO_PWM_35
<i>frequency</i>	PWM frequency (50Hz .. 1000Hz, only predefined frequencies with a period of an integral multiple of 1ms, 0.5ms or 0.25ms are possible)
<i>polarity</i>	Polarity of output signal • FALSE : Low output signal is variable • TRUE : High output signal is variable

	<i>diag_margin</i>	Indicate if a margin should be applied or not. <ul style="list-style-type: none"> • TRUE: margin is on • FALSE: no margin will be applied • If a PWM channel is configured safety relevant this parameter has to be set to TRUE
in	<i>safety_conf</i>	Relevant safety configuration of the PWM channel. If a safety configuration is passed the parameter <i>diag_margin</i> will be forced to TRUE .

Returns

`IO_ErrorType`

Return values

The return values match exactly the [return values of `IO_PWM_Init\(\)`](#).

Remarks

- Unlike with [`IO_PWM_Init\(\)`](#), the `low_side_channel` field of the safety configuration *is* evaluated with this function.
- If configured, the low side switch of a PWM channel is switched on and off together with the high side safety switch of the channel.
- The [remarks for `IO_PWM_Init\(\)`](#) also apply here.

7.19.6.6 `IO_ErrorType IO_PWM_ResetProtection (ubyte1 pwm_channel, ubyte1 *const reset_cnt)`

Reset the output protection for a PWM channel.

Parameters

	<i>pwm_channel</i>	PWM channel, one of: <ul style="list-style-type: none"> • <code>IO_PWM_00 .. IO_PWM_35</code>
out	<i>reset_cnt</i>	Protection reset counter. Indicates how often the application already reset the protection.

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist
<code>IO_E_FET_PROT_NOT_ACTIVE</code>	no output FET protection is active
<code>IO_E_FET_PROT_PERMANENT</code>	the output FET is protected permanently because it was already reset more than 10 times

<i>IO_E_FET_PROT_WAIT</i>	the output FET protection can not be reset, as the wait time of 10s is not already passed
<i>IO_E_CHANNEL_NOT_CONFIGURED</i>	the given channel is not configured
<i>IO_E_CH_CAPABILITY</i>	the given channel is not a PWM channel
<i>IO_E_INTERNAL_CSM</i>	an internal error occurred
<i>IO_E_UNKNOWN</i>	an unknown error occurred

Attention

The protection can be reset 10 times, afterwards the output will remain permanently protected

Note

- After entering the output protection, a certain time has to pass before the output protection can be reset:
 - 10s for *IO_PWM_00 .. IO_PWM_35*
- This function will set the output back again to the minimum pulse if a diagnostic margin is configured. After calling this function the output has to be set to the intended duty cycle with *IO_PWM_SetDuty()*

Remarks

If the parameter *reset_cnt* is *NULL*, the parameter is ignored. The parameter *reset_cnt* returns the number of resets already performed.

7.19.6.7 IO_ErrorType *IO_PWM_ResolveOpenLoadShortCircuit (ubyte1 pwm_channel)*

Resolve an open load or short circuit to battery voltage condition for a PWM channel.

Parameters

<i>pwm_channel</i>	PWM channel, one of: • <i>IO_PWM_00 .. IO_PWM_35</i>
--------------------	---

Returns

IO_ErrorType

Return values

<i>IO_E_OK</i>	everything fine
<i>IO_E_DRIVER_NOT_INITIALIZED</i>	the common driver init function has not been called before
<i>IO_E_BUSY</i>	a previously started resolving operation is still ongoing
<i>IO_E_INVALID_CHANNEL_ID</i>	the given channel id does not exist
<i>IO_E_CH_CAPABILITY</i>	the given channel is not a PWM channel
<i>IO_E_CHANNEL_NOT_CONFIGURED</i>	the given channel is not configured
<i>IO_E_INVALID_OPERATION</i>	the given channel does not return <i>IO_E_OPEN_LOAD_OR_SHORT_BAT</i> or is not in an operational state

Attention

- The resolving of a PWM channel requires to shift an internal feedback signal that is used by all PWM channels. Although this does not influence other PWM channels, it can affect channels that are configured for the alternative functions digital input and timer input. Please refer to the safety manual for detailed information.
- The I/O driver does not provide any debouncing mechanism for the return value of function `IO_PWM_SetDuty()`. Thus, the application itself has to implement the debouncing mechanism before starting a resolving operation if required.

Note

- Resolving of a PWM channel can only be triggered if `IO_PWM_SetDuty()` returns `IO_E_OPEN_LOAD_OR_SHORT_BAT` for that channel.
- The result of the resolving is retrieved using the function `IO_PWM_SetDuty()`: During the resolving operation, `IO_PWM_SetDuty()` returns `IO_E_RESOLVING`. When the resolving completes, the returned value will be `IO_E_OPEN_LOAD` or `IO_E_SHORT_BAT`, depending on the detected condition.
- After resolving an open load or short circuit to battery condition the PWM channel will be automatically disabled and cannot be reenabled anymore.

7.19.6.8 IO_ErrorType IO_PWM_SetDuty (**ubyte1** *pwm_channel*, **ubyte2** *duty_cycle*, **ubyte2** *const *high_time_fb*, **ubyte2** *const *period_fb*)

Set the duty cycle for a PWM channel.

Parameters

	<i>pwm_channel</i>	PWM channel, one of: • <code>IO_PWM_00</code> .. <code>IO_PWM_35</code>
	<i>duty_cycle</i>	Duty cycle for the channel. Range: 0..65535 (0%..100%)
<code>out</code>	<i>high_time_fb</i>	High time feedback for the channels (optional). Returns high-time in us
<code>out</code>	<i>period_fb</i>	Period feedback for the channels (optional). Returns period in us

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the given channel is not configured
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist
<code>IO_E_CH_CAPABILITY</code>	the given channel has no PWM capability
<code>IO_E_DRIVER_NOT_INITIALIZED</code>	the common driver init function has not been called before
<code>IO_E_STARTUP</code>	the output is in the startup phase
<code>IO_E_PWM_NO_DIAG</code>	No diagnostic is possible currently. Either the set duty cycle is not within a diagnostic margin or not enough samples have been captured yet.

<code>IO_E_SHORT_GND</code>	short circuit has been detected
<code>IO_E_OPEN_LOAD_OR_SHORT_BAT</code>	open load or short circuit to battery has been detected
<code>IO_E_OPEN_LOAD</code>	open load has been detected
<code>IO_E_SHORT_BAT</code>	short circuit to battery has been detected
<code>IO_E_FET_PROT_ACTIVE</code>	the output FET protection is active and has set the output to a constant low level. It can be reset with <code>IO_PWM_ResetProtection()</code> after the wait time is passed
<code>IO_E_FET_PROT_RESETABLE</code>	the output FET protection is ready to be reset with <code>IO_PWM_ResetProtection()</code>
<code>IO_E_FET_PROT_PERMANENT</code>	the output FET is protected permanently because it was already reset more than 10 times. The output will remain at a constant low level
<code>IO_E_SAFETY_SWITCH_DISABLED</code>	The safety switch of the corresponding output is disabled. The output is currently forced to low.
<code>IO_E_SAFE_STATE</code>	the PWM channel is in a safe state
<code>IO_E_INTERNAL_CSM</code>	an internal error occurred
<code>IO_E_UNKNOWN</code>	an unknown error occurred
<code>IO_E_RESOLVING</code>	a previously started resolving operation for this channel is still ongoing
<code>IO_E_RESOLVING_FAILED</code>	a previously started resolving operation for this channel has failed to resolve the actual error (open load or short circuit to battery)

Remarks

- The duty cycle cannot exceed a margin of 100us(lower boundary) and 200us(upper boundary) used for diagnostic if the parameter diag_margin was set `TRUE` (via the function `IO_PWM_Init()`). This mode is important for hydraulic coils. If the parameter diag_margin is `FALSE`, no duty cycle range margin will be applied.
- If one of the parameter `high_time_fb` or `period_fb` is `NULL`, the parameter is ignored. The parameter `high_time_fb` returns the measured pulse-width (high time) of the PWM signal in the last round in us. The parameter `period_fb` returns the measured period of the PWM signal in the last round in us. If the measurement is not finished yet, the parameter `high_time_fb` or `period_fb` holds the value 0.
- `IO_E_OPEN_LOAD` and `IO_E_SHORT_BAT` may be resolved by calling `IO_PWM_ResolveOpenLoadShort()` once and then continue calling `IO_PWM_SetDuty()`.

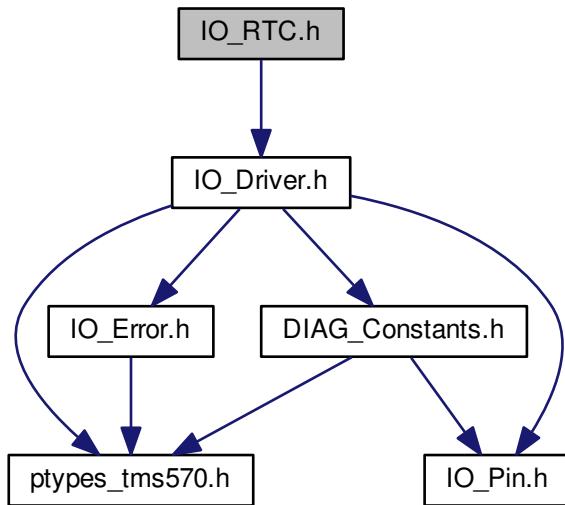
Attention

At the time when a PWM channel is initialized with `IO_PWM_Init()`, it has to be ensured that the actor is completely discharged and no current is induced to the PWM output. Otherwise the PWM current measurement calibration will fail.

7.20 IO_RTC.h File Reference

RTC functions, provides exact timing functions and services.

Include dependency graph for IO_RTC.h:



Macros

- `#define IO_RTC_Process IO_RTC_GetDateAndTimeStatus`
The Macro maintains backwards compatibility for applications using the `IO_RTC_Process()` from I/O Driver 3.1.

TypeDefs

- `typedef void(* IO_RTC_EVENT_HANDLER) (void)`
function pointer for event handler of the periodic interrupt handler

Functions

- `IO_ErrorType IO_RTC_DeInitDateAndTime (void)`
Deinitializes the external RTC device.
- `IO_ErrorType IO_RTC_GetDateAndTime (ubyte1 *const seconds, ubyte1 *const minutes, ubyte1 *const hours, ubyte1 *const days, ubyte1 *const months, ubyte1 *const years)`
Returns the date and time of the real time clock.
- `IO_ErrorType IO_RTC_GetDateAndTimeStatus (void)`
Gets status of the RTC Date and Time module and triggers the internal processing.
- `ubyte4 IO_RTC_GetTimeUS (ubyte4 timestamp)`

Returns the passed time.

- `IO_ErrorType IO_RTC_InitDateAndTime (void)`
Initializes the external RTC device for date and time operations.
- `IO_ErrorType IO_RTC_PeriodicDeInit (void)`
Deinitializes the Periodic Timer and stops it.
- `IO_ErrorType IO_RTC_PeriodicInit (ubyte2 period, IO_RTC_EVENT_HANDLER event_handler)`
Initializes the Periodic Timer.
- `IO_ErrorType IO_RTC_SetDateAndTime (ubyte1 seconds, ubyte1 minutes, ubyte1 hours, ubyte1 days, ubyte1 months, ubyte1 years)`
Sets the date and time of the real time clock.
- `IO_ErrorType IO_RTC_StartTime (ubyte4 *const timestamp)`
Returns a timestamp.

7.20.1 Detailed Description

RTC functions, provides exact timing functions and services.

Three different timing related functionalities are accessible:

- Microsecond based time stamp and time difference
- Periodic timer call
- Real Time Clock for date and time information

Timestamp functions

- `IO_RTC_StartTime ()`
- `IO_RTC_GetTimeUS ()`

The function `IO_RTC_StartTime()` returns a timestamp. The function `IO_RTC_GetTimeUS()` returns the time which has passed since the timestamp-value passed as an argument to this function. The application can use these two functions as often as it needs to. For different timing tasks only different timestamp variables need to be used.

Periodic functions

- `IO_RTC_PeriodicInit ()`
- `IO_RTC_PeriodicDeInit ()`

The function `IO_RTC_PeriodicInit()` can be used to register a periodic callback to the provided function which has to be passed as an argument. The function `IO_RTC_PeriodicDeInit()` will unregister this periodic call again.

Real Time Clock (date and time) functions

- `IO_RTC_InitDateAndTime ()`
- `IO_RTC_GetDateAndTime ()`
- `IO_RTC_SetDateAndTime ()`
- `IO_RTC_GetDateAndTimeStatus ()`
- `IO_RTC_DeInitDateAndTime ()`

The Real Time Clock functions provide a service for date and time keeping. It can be initialized by calling `IO_RTC_InitDateAndTime()`. After a power loss the RTC must be configured by calling

`IO_RTC_SetDateAndTime()` with the desired date and time. The application can request the current date and time by calling `IO_RTC_GetDateAndTime()`. Both setting and requesting need to be checked for completeness by using the function `IO_RTC_GetDateAndTimeStatus()`. This service can be uninitialized again by calling `IO_RTC_DeInitDateAndTime()`.

Attention

If the date and time are requested from a situation where no interrupts are available, such as from within the error or notification callback, the user application must ensure that the function `IO_RTC_GetDateAndTimeStatus()` is called periodically to trigger the RTC processing.

7.20.2 RTC Code Examples

Examples for using the RTC API

7.20.2.1 Example for RTC timestamp usage

The example initializes the RTC driver, and implements a loop which is executed every 5ms:

```
ubyte4 time_stamp;

IO_Driver_Init(NULL);           // initialize driver

IO_RTC_StartTime (&time_stamp); // start time (get timestamp)

while (1)
{
    task();      // user task function

    while (IO_RTC_GetTimeUS (time_stamp) < 5000); // wait until 5ms have passed
    time_stamp += 5000;                         // increase time stamp by cycle time

    // Note: If one cycle takes longer than the configured cycle time, for the next
    // cycle less time is available. This method helps to prevent a phase shift between
    // application runtime and hardware runtime
}
```

7.20.2.2 Example for RTC date and time usage

The example initializes the RTC driver, and setup the date and time

```
ubyte1 seconds;
ubyte1 minutes;
ubyte1 hours;
ubyte1 days;
ubyte1 months;
ubyte1 years;

// initialize real time clock
IO_RTC_InitDateAndTime();

// wait for real time clock to be ready
while(IO_RTC_GetDateAndTimeStatus() == IO_E_BUSY);
```

```

// set time to 11:30:00 and date to 05.07.13
IO_RTC_SetDateAndTime(00,          // seconds
                      30,          // minutes
                      11,          // hours
                      05,          // days
                      07,          // months
                      13);         // years

// wait for real time clock to be ready
while(IO_RTC_GetDateAndTimeStatus() == IO_E_BUSY);

// get date and time from real time clock
IO_RTC_GetDateAndTime(&seconds,   // seconds
                      &minutes,    // minutes
                      &hours,      // hours
                      &days,       // days
                      &months,     // months
                      &years);     // years

// wait for transfer to be complete
while(IO_RTC_GetDateAndTimeStatus() == IO_E_BUSY);

// check if data is valid
if (IO_RTC_GetDateAndTimeStatus() == IO_E_OK)
{
    // process data
}
else
{
    // error
}

```

7.20.3 Macro Definition Documentation

7.20.3.1 #define IO_RTC_Process IO_RTC_GetDateAndTimeStatus

The Macro maintains backwards compatibility for applications using the `IO_RTC_Process()` from I/O Driver 3.1.

Definition at line 168 of file `IO_RTC.h`.

7.20.4 Typedef Documentation

7.20.4.1 typedef void(* IO_RTC_EVENT_HANDLER)(void)

function pointer for event handler of the periodic interrupt handler

Definition at line 161 of file `IO_RTC.h`.

7.20.5 Function Documentation

7.20.5.1 IO_ErrorType IO_RTC_DeinitDateAndTime (void)

Deinitializes the external RTC device.

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the module has not been initialized

7.20.5.2 `IO_ErrorType IO_RTC_GetDateAndTime (ubyte1 *const seconds, ubyte1 *const minutes, ubyte1 *const hours, ubyte1 *const days, ubyte1 *const months, ubyte1 *const years)`

Returns the date and time of the real time clock.

The function triggers the reception from the RTC.

The reception can take several cycles to be completed, depending on the SPI load.

The date and time information is available on the addresses where the parameters point to as soon as the operation is finished. The state can be polled using the `IO_RTC_GetDateAndTimeStatus()` function.

Parameters

out	<code>seconds</code>	Seconds (0..59)
out	<code>minutes</code>	Minutes (0..59)
out	<code>hours</code>	Hours (0..23)
out	<code>days</code>	Days (1..31)
out	<code>months</code>	Months (1..12)
out	<code>years</code>	Years (0..99)

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_BUSY</code>	a get or set transfer is still ongoing, wait to be complete
<code>IO_E_NULL_POINTER</code>	a null pointer was passed
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the module has not been initialized

Remarks

This function cannot be called before `IO_RTC_InitDateAndTime()`.

7.20.5.3 `IO_ErrorType IO_RTC_GetDateAndTimeStatus (void)`

Gets status of the RTC Date and Time module and triggers the internal processing.

Remarks

Periodic polling using this function is necessary to trigger the processing of the RTC operations in situations where interrupts are not available, such as within the error or notification callbacks. If used from the error/notification callback, the nature of the failure will effect whether the RTC device is usable. In these cases it is important to include a timeout on any operations.

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	Everything fine.
<code>IO_E_BUSY</code>	A get or set transfer is still ongoing, wait to be complete.
<code>IO_E_RTC_CLOCK_INTEGRITY_FAILED</code>	The clock integrity has failed due to a power loss.
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	The module has not been initialized.
<code>IO_E_UNKNOWN</code>	An error occurred converting the data from the RTC chip.

Remarks

This function cannot be called before `IO_RTC_InitDateAndTime()`.

7.20.5.4 ubyte4 IO_RTC_GetTimeUS (`ubyte4 timestamp`)

Returns the passed time.

The function returns the time in us which has passed since the timestamp-value passed as an argument to this function.

Parameters

<code>timestamp</code>	Timestamp value (Could be received from a call of <code>IO_RTC_StartTime()</code>)
------------------------	---

Returns

`ubyte4`

Remarks

- If the RTC module has not been initialized, the function will return 0
- Please keep in mind that the time between `IO_RTC_StartTime()` and `IO_RTC_GetTimeUS()` for one timestamp should not exceed 71min (overflow)

7.20.5.5 `IO_ErrorType IO_RTC_InitDateAndTime (void)`

Initializes the external RTC device for date and time operations.

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	RTC is not supported by the used ECU variant
<code>IO_E_CHANNEL_BUSY</code>	the module has been initialized before
<code>IO_E_DRIVER_NOT_INITIALIZED</code>	the common driver init function has not been called before
<code>IO_E_UNKNOWN</code>	an unknown error occurred

Remarks

This function cannot be called before `IO_Driver_Init()` because the SPI driver needs to be initialized.

7.20.5.6 `IO_ErrorType IO_RTC_PeriodicDelInit(void)`

Deinitializes the Periodic Timer and stops it.

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_PERIODIC_NOT_CONFIGURED</code>	the periodic timer was not configured
<code>IO_E_UNKNOWN</code>	an unknown error occurred

7.20.5.7 `IO_ErrorType IO_RTC_PeriodicInit(ubyte2 period, IO_RTC_EVENT_HANDLER event_handler)`

Initializes the Periodic Timer.

Parameters

	<code>period</code>	Period on which the event handler should be called. unit: us (500..65535)
<code>in</code>	<code>event_handler</code>	Function pointer to the periodic event handler

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	RTC is not supported by the used ECU variant
<code>IO_E_NULL_POINTER</code>	a null pointer was passed
<code>IO_E_INVALID_PARAMETER</code>	an invalid parameter was passed
<code>IO_E_DRIVER_NOT_INITIALIZED</code>	the common driver init function has not been called before
<code>IO_E_CHANNEL_BUSY</code>	the periodic timer is already used
<code>IO_E_UNKNOWN</code>	an unknown error occurred

Remarks

This function cannot be called before `IO_Driver_Init()` because the RTC driver needs to be initialized.

Attention

- It is not allowed to call any I/O driver function out of the periodic interrupt
- The execution time of the periodic interrupt shall not exceed a time of 200us.

7.20.5.8 `IO_ErrorType IO_RTC_SetDateAndTime (ubyte1 seconds, ubyte1 minutes, ubyte1 hours, ubyte1 days, ubyte1 months, ubyte1 years)`

Sets the date and time of the real time clock.

The function triggers the configuration of the RTC.

The operation can take several cycles to be completed, depending on the SPI load.

The configuration is completed as soon as the SPI communication has been finished. The state can be polled using the `IO_RTC_GetDateAndTimeStatus()` function.

Parameters

<code>seconds</code>	Seconds (0..59)
<code>minutes</code>	Minutes (0..59)
<code>hours</code>	Hours (0..23)
<code>days</code>	Days (1..31)
<code>months</code>	Months (1..12)
<code>years</code>	Years (0..99)

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_BUSY</code>	a get or set transfer is still ongoing, wait to be complete
<code>IO_E_INVALID_PARAMETER</code>	an invalid parameter has been passed
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the module has not been initialized

Remarks

This function cannot be called before `IO_RTC_InitDateAndTime()`.

7.20.5.9 IO_ErrorType IO_RTC_StartTime (`ubyte4 *const timestamp`)

Returns a timestamp.

Returns a timestamp which can be used for RTC timing functions

Parameters

<code>out</code>	<code>timestamp</code>	Pointer for the returned timestamp value
------------------	------------------------	--

Returns

`IO_ErrorType`

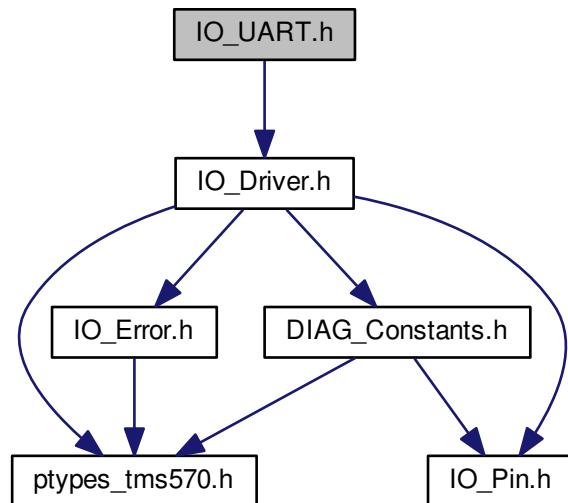
Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_NULL_POINTER</code>	a null pointer has been passed
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the module has not been initialized

7.21 IO_UART.h File Reference

IO Driver functions for UART communication.

Include dependency graph for IO_UART.h:



Functions

- **IO_ErrorType IO_UART_Delinit (void)**
Deinitialization of the UART module.
- **IO_ErrorType IO_UART_GetRxStatus (ubyte2 *const rx_len)**
Retrieve the status of the reception buffer.
- **IO_ErrorType IO_UART_GetTxStatus (ubyte2 *const tx_len)**
Retrieve the status of the transmission buffer.
- **IO_ErrorType IO_UART_Init (ubyte4 baudrate, ubyte1 dbits, ubyte1 par, ubyte1 sbits)**
Initialization of the UART module.
- **IO_ErrorType IO_UART_Read (ubyte1 *const data, ubyte2 len, ubyte2 *const rx_len)**
Read data from the serial interface.
- **IO_ErrorType IO_UART_Write (const ubyte1 *const data, ubyte2 len, ubyte2 *const tx_len)**
Write data to the serial interface.

Baudrate configuration

Defines the minimum and maximum possible UART baudrate

- #define **IO_UART_BAUDRATE_MIN** 1200U
- #define **IO_UART_BAUDRATE_MAX** 115200U

UART buffer length

Length (in bytes) of the internal buffers for transmission and reception

- #define `IO_UART_BUFFER_LEN` 512U

Parity configuration

Defines for the UART parity configuration

- #define `IO_UART_PARITY_NONE` 0x0U
- #define `IO_UART_PARITY_EVEN` 0x2U
- #define `IO_UART_PARITY_ODD` 0x3U

7.21.1 Detailed Description

IO Driver functions for UART communication.

The UART driver uses the SCI module of the TMS570 CPU. The interface supports baud rates up to 115.200 bit/s.

UART-API Usage:

- [Examples for UART API functions](#)

7.21.2 UART Code Examples

Examples for using the UART API

7.21.2.1 Example for UART initialization

```
// init UART driver
IO_UART_Init(115200,           // baud rate
             8,                // 8 data bits
             IO_UART_PARITY_NONE, // no parity bit
             1);               // 1 stop bit
```

7.21.2.2 Example for UART task function call

```
char tx_data[20] = { '\0' };
char rx_data[20] = { '\0' };
ubyte2 tx_len = 0;
ubyte2 rx_len = 0;

// get number of bytes in the transmit buffer
IO_UART_GetTxStatus(&tx_len);

// check, if there's enough space in the transmit buffer
if ((tx_len + sizeof(tx_data)) < IO_UART_BUFFER_LEN)
{
    // transmit some data
    IO_UART_Write(tx_data,           // data buffer to be transmitted
                  sizeof(tx_data), // size of the given data buffer
```

```

        &tx_len);           // successfully queued data bytes
    }

// get number of bytes in the receive buffer
IO_UART_GetRxStatus(&rx_len);

// check, if there's data in the receive buffer
if (rx_len > 0)
{
    // try to read some data
    IO_UART_Read(rx_data,           // destination data buffer
                 sizeof(rx_data),   // size of the given data buffer
                 &rx_len);          // number of read data bytes
}

```

7.21.3 Macro Definition Documentation

7.21.3.1 #define IO_UART_BAUDRATE_MAX 115200U

Maximum UART baudrate

Definition at line 95 of file IO_UART.h.

7.21.3.2 #define IO_UART_BAUDRATE_MIN 1200U

Minimum UART baudrate

Definition at line 94 of file IO_UART.h.

7.21.3.3 #define IO_UART_BUFFER_LEN 512U

512 bytes

Definition at line 105 of file IO_UART.h.

7.21.3.4 #define IO_UART_PARITY_EVEN 0x2U

even parity

Definition at line 116 of file IO_UART.h.

7.21.3.5 #define IO_UART_PARITY_NONE 0x0U

no parity

Definition at line 115 of file IO_UART.h.

7.21.3.6 #define IO_UART_PARITY_ODD 0x3U

odd parity

Definition at line 117 of file IO_UART.h.

7.21.4 Function Documentation

7.21.4.1 IO_ErrorType IO_UART_Delnit (void)

Deinitialization of the UART module.

The UART module can be initialized again by calling [IO_UART_Init\(\)](#).

Returns

[IO_ErrorType](#)

Return values

IO_E_OK	Everything fine.
IO_E_CHANNEL_NOT_CONFIGURED	Channel has not been initialized.

7.21.4.2 IO_ErrorType IO_UART_GetRxStatus (ubyte2 *const rx_len)

Retrieve the status of the reception buffer.

This function returns the number of bytes in the Rx buffer and the last error detected during the reception ([IO_E_UART_...](#)) since the last call of this function (i.e., it *reads and clears* the errors).

Note

The UART reception stores the last detected error. When multiple errors occurred since the last call of this function, all except the last one have been discarded.

Parameters

out	rx_len	Number of received data bytes in reception buffer.
-----	------------------------	--

Returns

[IO_ErrorType](#)

Return values

IO_E_OK	Everything fine.
IO_E_UART_BUFFER_FULL	The Rx buffer overflowed. The data received after the overflow has been lost.
IO_E_UART_FRAMING	UART framing error has been detected.
IO_E_UART_OVERFLOW	UART hardware reception buffer overrun.
IO_E_UART_PARITY	UART parity check failed.
IO_E_UART_DMA	Error in the Rx DMA data processing. Data has been lost.
IO_E_NULL_POINTER	NULL pointer has been passed.
IO_E_CHANNEL_NOT_CONFIGURED	Channel has not been initialized.

7.21.4.3 IO_ErrorType IO_UART_GetTxStatus (**ubyte2 *const tx_len**)

Retrieve the status of the transmission buffer.

Parameters

out	<i>tx_len</i>	Number of remaining data bytes in transmission buffer.
-----	---------------	--

Returns

[IO_ErrorType](#)

Return values

IO_E_OK	Everything fine.
IO_E_NULL_POINTER	NULL pointer has been passed.
IO_E_CHANNEL_NOT_CONFIGURED	Channel has not been initialized.

7.21.4.4 IO_ErrorType IO_UART_Init (**ubyte4 baudrate, ubyte1 dbits, ubyte1 par, ubyte1 sbits**)

Initialization of the UART module.

The UART module can only be initialized after [IO_Driver_Init\(\)](#). The function [IO_UART_Init\(\)](#) needs to be called before any UART function.

Parameters

<i>baudrate</i>	Baud rate in baud/s (1200 ... 115200)
<i>dbits</i>	Number of data bits per frame (1 ... 8)
<i>par</i>	Parity configuration, one of: <ul style="list-style-type: none"> • IO_UART_PARITY_NONE • IO_UART_PARITY_EVEN • IO_UART_PARITY_ODD
<i>sbits</i>	Number of stop bits per frame (1 .. 2)

Returns

[IO_ErrorType](#)

Return values

IO_E_OK	Everything fine.
IO_E_INVALID_CHANNEL_ID	The UART interface is not available on this ECU variant.
IO_E_INVALID_PIN_CONFIG	Error in the ECU variant configuration data.
IO_E_INVALID_VARIANT	The detected ECU variant is not supported.
IO_E_INVALID_PARAMETER	A parameter is out of range.
IO_E_CHANNEL_BUSY	The UART interface is already initialized.
IO_E_DRIVER_NOT_INITIALIZED	The I/O Driver has not been initialized.

Remarks

Module is initialized only once. To re-initialize the module, the function [IO_UART_DelInit\(\)](#) needs to be called.

7.21.4.5 IO_ErrorType IO_UART_Read (ubyte1 *const data, ubyte2 len, ubyte2 *const rx_len)

Read data from the serial interface.

This function reads the received data from the Rx buffer. It also reports the errors detected during the reception (IO_E_UART_...) since the last call to [IO_UART_GetRxStatus\(\)](#). The errors are *not cleared* in this function. See [IO_UART_GetRxStatus\(\)](#) for more details.

Parameters

out	<i>data</i>	Address where the read data will be stored.
	<i>len</i>	Maximum size of data that can be stored.
out	<i>rx_len</i>	Actually read bytes.

Returns

[IO_ErrorType](#)

Return values

IO_E_OK	Everything fine.
IO_E_UART_BUFFER_FULL	The Rx buffer overflowed. The data received after the overflow has been lost.
IO_E_UART_FRAMING	UART framing error has been detected.
IO_E_UART_OVERFLOW	UART hardware reception buffer overrun.
IO_E_UART_PARITY	UART parity check failed.
IO_E_UART_DMA	Error in the Rx DMA data processing. Data has been lost.
IO_E_NULL_POINTER	NULL pointer has been passed.
IO_E_CHANNEL_NOT_CONFIGURED	Channel has not been initialized.

7.21.4.6 IO_ErrorType IO_UART_Write (const ubyte1 *const data, ubyte2 len, ubyte2 *const tx_len)

Write data to the serial interface.

This function writes the data to the Tx buffer and starts the transmission.

Parameters

in	<i>data</i>	Address to the data to be written.
	<i>len</i>	Number of bytes to be written.
out	<i>tx_len</i>	Actually written bytes.

Returns

`IO_ErrorType`

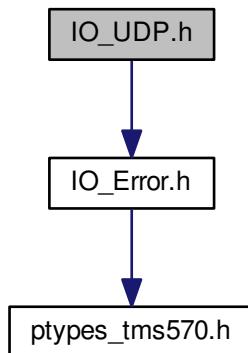
Return values

<code>IO_E_OK</code>	Everything fine.
<code>IO_E_UART_BUFFER_FULL</code>	The free space in the Tx buffer is less than <code>len</code> . No data is written in this case.
<code>IO_E_NULL_POINTER</code>	NULL pointer has been passed.
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	Channel has not been initialized.

7.22 IO_UDP.h File Reference

IO Driver functions for UDP communication.

Include dependency graph for IO_UDP.h:



Functions

- `IO_ErrorType IO_UDP_CreateSocket (ubyte1 ubReadWrite, ubyte2 port, const ubyte1 *IPAddr, ubyte2 *socket_id)`
Create a socket for transmission or reception.
- `IO_ErrorType IO_UDP_DelInit (void)`
Deinitialize the Ethernet interface and free all sockets.
- `IO_ErrorType IO_UDP_FreeSocket (ubyte2 socket_id)`
Free socket.
- `IO_ErrorType IO_UDP_GetStatus (ubyte2 socket_id, ubyte2 *socket_status)`
Get status info for specified socket.

- **IO_ErrorType IO_UDP_Init (ubyte2 baudrate, const ubyte1 *local_ip, const ubyte1 *pc_ip)**
Initialize the Ethernet interface and the UDP server.
- **IO_ErrorType IO_UDP_Read (ubyte2 socket_id, ubyte2 *port, ubyte1 *buf, ubyte2 *len, ubyte2 maxlen)**
Read data from the socket and process ARP requests.
- **IO_ErrorType IO_UDP_WriteTo (ubyte2 socket_id, const ubyte1 *buf, ubyte2 len, ubyte2 port, const ubyte1 *target_ip)**
Send data to the receiver defined by target_addr and port.

Data transfer direction

- #define **IO_UDP_READ 1U**
- #define **IO_UDP_WRITE 2U**

Ethernet speed

- #define **IO_ETH_SPEED_10_MB 1u**
- #define **IO_ETH_SPEED_100_MB 2u**

7.22.1 Detailed Description

IO Driver functions for UDP communication.

7.22.2 Limitations

This preliminary implementation contains a very limited handling of the ARP packets:

- No ARP requests are sent.
- No filtering on received ARP packets - all ARP packets are responded to.

7.22.3 Macro Definition Documentation

7.22.3.1 #define IO_ETH_SPEED_100_MB 2u

100 Mbit/s

Definition at line 62 of file IO_UDP.h.

7.22.3.2 #define IO_ETH_SPEED_10_MB 1u

10 Mbit/s

Definition at line 61 of file IO_UDP.h.

7.22.3.3 #define IO_UDP_READ 1U

used to set up a socket for receiving
 Definition at line 46 of file IO_UDP.h.

7.22.3.4 #define IO_UDP_WRITE 2U

used to set up a socket for transmitting
 Definition at line 47 of file IO_UDP.h.

7.22.4 Function Documentation

7.22.4.1 IO_ErrorType IO_UDP_CreateSocket (ubyte1 *ubReadWrite*, ubyte2 *port*, const ubyte1 * *IPAddr*, ubyte2 * *socket_id*)

Create a socket for transmission or reception.

Parameters

<i>ubReadWrite</i>	direction of transfer, one of <ul style="list-style-type: none"> • IO_UDP_READ socket for receive • IO_UDP_WRITE socket for transmit
<i>port</i>	local (source) port
<i>IPAddr</i>	ASCII string of local IP address (ipv4)
<i>out</i>	handle of socket created

Returns

[IO_ErrorType](#)

Return values

IO_E_OK	everything fine
IO_E_CHANNEL_NOT_CONFIGURED	interface was not configured
IO_E_ETH_MDIO_READ	an MDIO read transfer error occurred
IO_E_ETH_MDIO_TIMEOUT	a timeout error occurred when performing an MDIO transfer
IO_E_BUSY	auto-negotiation is not finished
IO_E_ETH_NO_LINK	no valid link has been established on the Ethernet interface
IO_E_UDP_NOMORESOCKETS	no more UDP sockets are available
IO_E_SOCKET_NOT_INITIALIZED	socket initialization failed due to illegal IP address

7.22.4.2 IO_ErrorType IO_UDP_DelInit (void)

Deinitialize the Ethernet interface and free all sockets.

Returns**IO_ErrorType****Return values**

<i>IO_E_OK</i>	everything fine
<i>IO_E_CHANNEL_NOT_CONFIGURED</i>	interface was not configured
<i>IO_E_INTERNAL_CSM</i>	internal CSM error
<i>IO_E_ETH_DEINIT_TIMEOUT</i>	timeout on de-initialization of the Ethernet interface

7.22.4.3 IO_ErrorType IO_UDP_FreeSocket (ubyte2 socket_id)

Free socket.

Parameters

<i>socket_id</i>	handle of socket to be freed
------------------	------------------------------

Returns**IO_ErrorType****Return values**

<i>IO_E_OK</i>	everything fine
<i>IO_E_UDP_WRONG_HANDLE</i>	socket handle invalid
<i>IO_E_SOCKET_NOT_INITIALIZED</i>	socket not initialized

7.22.4.4 IO_ErrorType IO_UDP_GetStatus (ubyte2 socket_id, ubyte2 * socket_status)

Get status info for specified socket.

Parameters

	<i>socket_id</i>	handle of the socket
out	<i>socket_status</i>	status of socket, one of:
		<ul style="list-style-type: none"> • <i>IO_E_SOCKET_NOT_INITIALIZED</i> socket not used • <i>IO_E_OK</i> socket initialized

Returns**IO_ErrorType****Return values**

<i>IO_E_OK</i>	everything fine
<i>IO_E_NULL_POINTER</i>	null pointer was passed
<i>IO_E_UDP_WRONG_HANDLE</i>	invalid handle has been passed

7.22.4.5 IO_ErrorType IO_UDP_Init (ubyte2 *baudrate*, const ubyte1 * *local_ip*, const ubyte1 * *pc_ip*)

Initialize the Ethernet interface and the UDP server.

Parameters

<i>baudrate</i>	baudrate parameter, one of <ul style="list-style-type: none"> • IO_ETH_SPEED_10_MB • IO_ETH_SPEED_100_MB
<i>local_ip</i>	ASCII string of local IP address (ipv4), format "www.xxx.yyy.zzz"
<i>pc_ip</i>	ASCII string of host (PC) IP address (ipv4), format "www.xxx.yyy.zzz"

Returns

[IO_ErrorType](#)

Return values

IO_E_OK	everything fine
IO_E_INVALID_CHANNEL_ID	the ECU variant doesn't support this feature
IO_E_CHANNEL_BUSY	the channel has already been initialized
IO_E_ETH_INIT_TIMEOUT	a timeout error occurred during Ethernet initialization
IO_E_ETH_INIT_FAIL	an error during Ethernet initialization occurred
IO_E_ETH_MAC_INVALID	the configured MAC address is invalid
IO_E_ETH_MDIO_TIMEOUT	a timeout error occurred when performing an MDIO transfer
IO_E_ETH_MDIO_READ	an MDIO read transfer error occurred
IO_E_INTERNAL_CSM	an internal error occurred

7.22.4.6 IO_ErrorType IO_UDP_Read (ubyte2 *socket_id*, ubyte2 * *port*, ubyte1 * *buf*, ubyte2 * *len*, ubyte2 *maxlen*)

Read data from the socket and process ARP requests.

Note

This function needs to be called periodically in order to handle the received ARP requests: When an ARP requests is received, the function automatically sends the response. In this case, [IO_E_UDP_ARP RECEIVED](#) is returned.

Parameters

	<i>socket_id</i>	handle of socket to be used for reception
out	<i>port</i>	source port of received message
	<i>buf</i>	buffer for received data
out	<i>len</i>	number of bytes actually received
	<i>maxlen</i>	maximum size of receive buffer

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_UDP_WRONG_HANDLE</code>	invalid handle has been passed
<code>IO_E_NULL_POINTER</code>	null pointer was passed
<code>IO_E_UDP_INVALID_BUFFER</code>	buffer invalid: null pointer or len = 0
<code>IO_E_ETH_NO_LINK</code>	no ethernet link available
<code>IO_E_BUSY</code>	link busy (possibly download in progress)
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	interface was not configured
<code>IO_E_ETH_READ_FAIL</code>	error in a read operation on the Ethernet interface
<code>IO_E_ETH_WRITE_FAIL</code>	error in a write operation on the Ethernet interface
<code>IO_E_UDP_ARP RECEIVED</code>	Address Resolution Package received
<code>IO_E_UDP_OVERFLOW</code>	received data length greater than maximum buffer size
<code>IO_E_WRONG_ADDRESS</code>	IP address of received message does not match
<code>IO_E_UDP_WRONG_PORT</code>	port number of received message does not match

7.22.4.7 `IO_ErrorType IO_UDP_WriteTo (ubyte2 socket_id, const ubyte1 * buf, ubyte2 len, ubyte2 port, const ubyte1 * target_ip)`

Send data to the receiver defined by target_addr and port.

Note

The transmitted MAC address will be set to the remote MAC address in the last received packet.

Parameters

<code>socket_id</code>	handle of socket to be used for transmit
<code>buf</code>	buffer containing the data to be sent
<code>len</code>	number of bytes to be sent
<code>port</code>	target port for transmitted message
<code>target_ip</code>	ASCII string of target IP address (ipv4)

Returns

`IO_ErrorType`

Return values

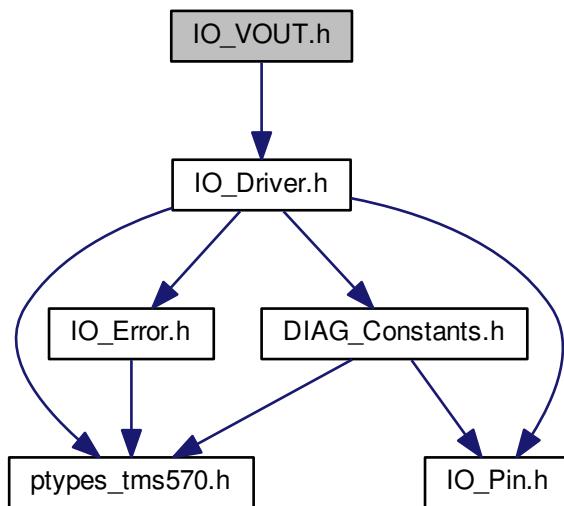
<code>IO_E_OK</code>	everything fine
<code>IO_E_UDP_OVERFLOW</code>	packet too long
<code>IO_E_UDP_WRONG_HANDLE</code>	invalid handle has been passed
<code>IO_E_NULL_POINTER</code>	null pointer has been passed
<code>IO_E_WRONG_ADDRESS</code>	illegal target IP address
<code>IO_E_INVALID_PARAMETER</code>	invalid header parameter
<code>IO_E_BUSY</code>	link busy

IO_E_ETH_NO_LINK	no ethernet link available
IO_E_ETH_WRITE_FAIL	error in a write operation on the Ethernet interface

7.23 IO_VOUT.h File Reference

IO Driver functions for voltage outputs.

Include dependency graph for IO_VOUT.h:



Functions

- [`IO_ErrorType IO_VOUT_Delinit \(ubyte1 vout_channel\)`](#)
Deinitializes one voltage output.
- [`IO_ErrorType IO_VOUT_GetVoltage \(ubyte1 vout_channel, ubyte2 *const voltage, bool *const fresh\)`](#)
Get the voltage feedback of one voltage output channel.
- [`IO_ErrorType IO_VOUT_Init \(ubyte1 vout_channel\)`](#)
Setup one voltage output channel.
- [`IO_ErrorType IO_VOUT_ResetProtection \(ubyte1 vout_channel, ubyte1 *const reset_cnt\)`](#)
Reset the output protection for a VOUT channel.
- [`IO_ErrorType IO_VOUT_SetVoltage \(ubyte1 vout_channel, ubyte2 output_voltage\)`](#)
Sets the output voltage of one voltage output channel.

7.23.1 Detailed Description

IO Driver functions for voltage outputs.

Contains all service functions for the voltage outputs. Up to 8 channels can be configured: [IO_VOUT_00 .. IO_VOUT_07](#)

The voltage output stage is a push/pull PWM output with a well defined output resistance of 2.58kOhm. This resistance is necessary for operation, low pass filtering and over load protection.

Note

The voltage output is limited to resistive loads to ground with 10kOhm or higher.

The outputs will be activated after setting them via [IO_VOUT_SetVoltage\(\)](#).

When configuring a voltage output, the associated voltage feedback channel will also be configured.

Note

Deviations on the power supply are automatically corrected by the module.

VOUT-API Usage:

- [Examples for VOUT API functions](#)

7.23.2 VOUT output protection

Each voltage output is individually protected against malfunction. Whenever the difference between the measured output (**U_feedback**) and the configured output calculated by **U_diff = output_value / 100 * U_BAT - U_feedback** is greater than **abs(+/-18V)** for at least 100ms, the output protection is enabled latest within 12ms.

When entering the protection state, the voltage output has to remain in this state for at least 1s. After this wait time the voltage output can be reenabled via function [IO_VOUT_ResetProtection\(\)](#). Note that the number of reenabling operations for a single voltage output is limited to 10. Refer to function [IO_VOUT_ResetProtection\(\)](#) for more information on how to reenable a voltage output.

7.23.3 VOUT Code Examples

Examples for using the VOUT API

7.23.3.1 VOUT initialization example:

```
IO_VOUT_Init(IO_VOUT_00);           // setup a voltage output
```

7.23.3.2 VOUT task function example:

```
ubyte2 voltage = 0;
bool fresh = FALSE;
```

```

IO_VOUT_SetVoltage(IO_VOUT_00,
                    6200);           // set output voltage to 6200mV

IO_VOUT_GetVoltage(IO_VOUT_00,
                    &voltage,
                    &fresh);        // get voltage feedback
    
```

7.23.4 Function Documentation

7.23.4.1 IO_ErrorType IO_VOUT_DeInit (ubyte1 vout_channel)

Deinitializes one voltage output.

Parameters

vout_channel	VOUT output (IO_VOUT_00 .. IO_VOUT_07)
--------------	--

Returns

IO_ErrorType

Return values

IO_E_OK	everything fine
IO_E_INVALID_CHANNEL_ID	the given channel id does not exist
IO_E_CHANNEL_NOT_CONFIGURED	the given channel is not configured
IO_E_CH_CAPABILITY	the given channel is not a VOUT channel
IO_E_UNKNOWN	an unknown error occurred

7.23.4.2 IO_ErrorType IO_VOUT_GetVoltage (ubyte1 vout_channel, ubyte2 *const voltage, bool *const fresh)

Get the voltage feedback of one voltage output channel.

Parameters

vout_channel	VOUT channel (IO_VOUT_00 .. IO_VOUT_07)
out	voltage
out	fresh

Measured voltage in mV. Range: 0..32000 (0V..32.000V)

Indicates if new values are available since the last call.

- **TRUE**: Value in "voltage" is valid
- **FALSE**: No new value available.

Returns

IO_ErrorType

Return values

IO_E_OK	everything fine
IO_E_INVALID_CHANNEL_ID	the given channel id does not exist

<code>IO_E_NULL_POINTER</code>	a NULL pointer has been passed to the function
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the given channel is not configured
<code>IO_E_CH_CAPABILITY</code>	the given channel is not a VOUT channel
<code>IO_E_STARTUP</code>	the output is in the startup phase
<code>IO_E_REFERENCE</code>	the reference voltage is out of range
<code>IO_E_UNKNOWN</code>	an unknown error occurred

Remarks

If there is no new measurement value available (for example the function `IO_VOUT_GetVoltage()` gets called more frequently than the AD sampling) the flag `fresh` will be set to `FALSE`.

7.23.4.3 IO_ErrorType IO_VOUT_Init (`ubyte1 vout_channel`)

Setup one voltage output channel.

Parameters

<code>vout_channel</code>	VOUT channel (<code>IO_VOUT_00 .. IO_VOUT_07</code>)
---------------------------	--

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist or is not available on the used ECU variant
<code>IO_E_CH_CAPABILITY</code>	the given channel is not a VOUT channel or the used ECU variant does not support this function
<code>IO_E_CHANNEL_BUSY</code>	the VOUT output channel or the voltage feedback channel is currently used by another function
<code>IO_E_DRIVER_NOT_INITIALIZED</code>	the common driver init function has not been called before
<code>IO_E_FPGA_NOT_INITIALIZED</code>	the FPGA has not been initialized
<code>IO_E_UNKNOWN</code>	an unknown error occurred

7.23.4.4 IO_ErrorType IO_VOUT_ResetProtection (`ubyte1 vout_channel, ubyte1 *const reset_cnt`)

Reset the output protection for a VOUT channel.

Parameters

	<code>vout_channel</code>	VOUT channel (<code>IO_VOUT_00 .. IO_VOUT_07</code>)
<code>out</code>	<code>reset_cnt</code>	Protection reset counter. Indicates how often the application already reset the protection.

Returns

[IO_ErrorType](#)

Return values

IO_E_OK	everything fine
IO_E_INVALID_CHANNEL_ID	the given channel id does not exist
IO_E_FET_PROT_NOT_ACTIVE	no output FET protection is active
IO_E_FET_PROT_PERMANENT	the output FET is protected permanently because it was already reset more than 10 times
IO_E_FET_PROT_WAIT	the output FET protection can not be reset, as the wait time of 1s is not already passed
IO_E_CHANNEL_NOT_CONFIGURED	the given channel is not configured
IO_E_CH_CAPABILITY	the given channel is not a VOUT channel
IO_E_UNKNOWN	an unknown error occurred

Attention

The protection can be reset 10 times, afterwards the output will remain permanently protected

Note

- After entering the output protection, a certain time has to pass before the output protection can be reset:
 - 1s for [IO_VOUT_00 .. IO_VOUT_07](#)
- This function will not set the output back again to the last voltage. After calling this function the output has to be set to the intended voltage with [IO_VOUT_SetVoltage\(\)](#)

Remarks

If the parameter `reset_cnt` is `NULL`, the parameter is ignored. The parameter `reset_cnt` returns the number of resets already performed.

7.23.4.5 [IO_ErrorType IO_VOUT_SetVoltage \(ubyte1 vout_channel, ubyte2 output_voltage \)](#)

Sets the output voltage of one voltage output channel.

Parameters

<code>vout_channel</code>	VOUT channel (IO_VOUT_00 .. IO_VOUT_07)
<code>output_voltage</code>	Output voltage in mV (0..32000)

Returns

[IO_ErrorType](#)

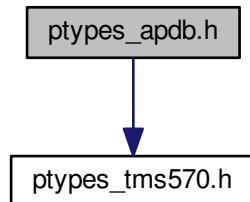
Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist
<code>IO_E_FET_PROT_ACTIVE</code>	the output FET protection is active and has set the output to 0V. It can be reset with <code>IO_VOUT_ResetProtection()</code> after the wait time is passed
<code>IO_E_FET_PROT_REENABLE</code>	the output FET protection is ready to be reset with <code>IO_VOUT_ResetProtection()</code>
<code>IO_E_FET_PROT_PERMANENT</code>	the output FET is protected permanently because it was already reset more than 10 times. The output will remain at 0V
<code>IO_E_INVALID_PARAMETER</code>	the setpoint is out of range
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the given channel is not configured
<code>IO_E_CH_CAPABILITY</code>	the given channel is not a VOUT channel
<code>IO_E_UNKNOWN</code>	an unknown error occurred

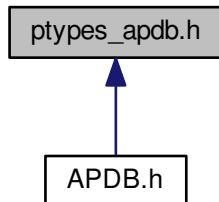
7.24 ptypes_apdb.h File Reference

APDB target abstraction.

Include dependency graph for ptypes_apdb.h:



This graph shows which files directly or indirectly include this file:



Macros

- `#define APPL_START ((ubyte4)_c_int00)`
Application entry point.

APDB default date defines

Default defines for the date structure in the APDB. These values should be normally set during the compilation process.

- `#define RTS_TTC_FLASH_DATE_YEAR 0UL`
- `#define RTS_TTC_FLASH_DATE_MONTH 0UL`
- `#define RTS_TTC_FLASH_DATE_DAY 0UL`
- `#define RTS_TTC_FLASH_DATE_HOUR 0UL`
- `#define RTS_TTC_FLASH_DATE_MINUTE 0UL`

Startup entry point

- `void _c_int00 (void)`

7.24.1 Detailed Description

APDB target abstraction.

This file defines the interface between the generic APDB format and the platform specific includes and definitions.

7.24.2 Macro Definition Documentation

7.24.2.1 #define APPL_START ((ubyte4)_c_int00)

Application entry point.

The entry point of the user application, which is called by the bootloader (must be written to the MainAddress field of the APDB to ensure proper application startup).

Definition at line 46 of file ptypes_apdb.h.

7.24.2.2 #define RTS_TTC_FLASH_DATE_DAY 0UL

day of the date

Definition at line 66 of file ptypes_apdb.h.

7.24.2.3 #define RTS_TTC_FLASH_DATE_HOUR 0UL

hour of the date

Definition at line 70 of file ptypes_apdb.h.

7.24.2.4 #define RTS_TTC_FLASH_DATE_MINUTE 0UL

minute of the date

Definition at line 74 of file ptypes_apdb.h.

7.24.2.5 #define RTS_TTC_FLASH_DATE_MONTH 0UL

month of the date

Definition at line 62 of file ptypes_apdb.h.

7.24.2.6 #define RTS_TTC_FLASH_DATE_YEAR 0UL

year of the date

Definition at line 58 of file ptypes_apdb.h.

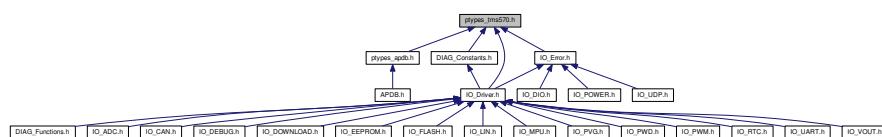
7.24.3 Function Documentation

7.24.3.1 void _c_int00 (void)

7.25 ptypes_tms570.h File Reference

Primitive data types.

This graph shows which files directly or indirectly include this file:



Macros

- #define **FALSE** ((bool)0U)
*bool definition for FALSE (deprecated, please use **TT_FALSE** instead)*
- #define **NULL** ((void *)0U)
NULL definition.
- #define **NULL_PTR** ((void *)0U)
NULL pointer definition.
- #define **TRUE** ((bool)1U)
*bool definition for TRUE (deprecated, please use **TT_TRUE** instead)*
- #define **TT_FALSE** ((bool)0U)
bool definition for FALSE
- #define **TT_TRUE** ((bool)1U)
bool definition for TRUE
- #define **UBYTE1_MAX_VALUE** ((ubyte1)0xFFU)
maximum value of the ubyte1 type
- #define **UBYTE1_MIN_VALUE** ((ubyte1)0x00U)
minimum value of the ubyte1 type
- #define **UBYTE2_MAX_VALUE** ((ubyte2)0xFFFFU)
maximum value of the ubyte2 type
- #define **UBYTE2_MIN_VALUE** ((ubyte2)0x0000U)
minimum value of the ubyte2 type
- #define **UBYTE4_MAX_VALUE** ((ubyte4)0xFFFFFFFFU)
maximum value of the ubyte4 type
- #define **UBYTE4_MIN_VALUE** ((ubyte4)0x00000000U)
minimum value of the ubyte4 type

Typedefs

- typedef unsigned char **bool**
boolean type
- typedef unsigned char **BOOL**
boolean type, obsolete definition (for compatibility reasons only)
- typedef float **float4**
32bit float type
- typedef double **float8**
64bit float type
- typedef signed char **sbyte1**
signed 8bit integer
- typedef signed short int **sbyte2**

- *signed 16bit integer*
- `typedef signed long sbyte4`
signed 32bit integer
- `typedef signed long long sbyte8`
signed 64bit integer
- `typedef unsigned char ubyte1`
unsigned 8bit integer
- `typedef unsigned short int ubyte2`
unsigned 16bit integer
- `typedef unsigned long ubyte4`
unsigned 32bit integer
- `typedef unsigned long long ubyte8`
unsigned 64bit integer

7.25.1 Detailed Description

Primitive data types.

This file defines the primitive data types used for the IO Driver

7.25.2 Macro Definition Documentation

7.25.2.1 #define FALSE ((bool)0U)

bool definition for FALSE (deprecated, please use `TT_FALSE` instead)

Definition at line 76 of file ptypes_tms570.h.

7.25.2.2 #define NULL ((void *)0U)

NULL definition.

Definition at line 61 of file ptypes_tms570.h.

7.25.2.3 #define NULL_PTR ((void *)0U)

NULL pointer definition.

Definition at line 68 of file ptypes_tms570.h.

7.25.2.4 #define TRUE ((bool)1U)

bool definition for TRUE (deprecated, please use `TT_TRUE` instead)

Definition at line 83 of file ptypes_tms570.h.

7.25.2.5 #define TT_FALSE ((bool)0U)

bool definition for FALSE

Definition at line 90 of file ptypes_tms570.h.

7.25.2.6 #define TT_TRUE ((bool)1U)

bool definition for TRUE

Definition at line 95 of file ptypes_tms570.h.

7.25.2.7 #define UBYTE1_MAX_VALUE ((ubyte1)0xFFU)

maximum value of the ubyte1 type

Definition at line 120 of file ptypes_tms570.h.

7.25.2.8 #define UBYTE1_MIN_VALUE ((ubyte1)0x00U)

minimum value of the ubyte1 type

Definition at line 125 of file ptypes_tms570.h.

7.25.2.9 #define UBYTE2_MAX_VALUE ((ubyte2)0xFFFFU)

maximum value of the ubyte2 type

Definition at line 110 of file ptypes_tms570.h.

7.25.2.10 #define UBYTE2_MIN_VALUE ((ubyte2)0x0000U)

minimum value of the ubyte2 type

Definition at line 115 of file ptypes_tms570.h.

7.25.2.11 #define UBYTE4_MAX_VALUE ((ubyte4)0xFFFFFFFFU)

maximum value of the ubyte4 type

Definition at line 100 of file ptypes_tms570.h.

7.25.2.12 #define UBYTE4_MIN_VALUE ((ubyte4)0x00000000U)

minimum value of the ubyte4 type

Definition at line 105 of file ptypes_tms570.h.

7.25.3 Typedef Documentation**7.25.3.1 typedef unsigned char bool**

boolean type

Definition at line 47 of file ptypes_tms570.h.

7.25.3.2 typedef unsigned char BOOL

boolean type, obsolete definition (for compatibility reasons only)

Definition at line 51 of file ptypes_tms570.h.

7.25.3.3 typedef float float4

32bit float type

Definition at line 41 of file ptypes_tms570.h.

7.25.3.4 typedef double float8

64bit float type

Definition at line 42 of file ptypes_tms570.h.

7.25.3.5 typedef signed char sbyte1

signed 8bit integer

Definition at line 36 of file ptypes_tms570.h.

7.25.3.6 typedef signed short int sbyte2

signed 16bit integer

Definition at line 37 of file ptypes_tms570.h.

7.25.3.7 typedef signed long sbyte4

signed 32bit integer

Definition at line 38 of file ptypes_tms570.h.

7.25.3.8 typedef signed long long sbyte8

signed 64bit integer

Definition at line 39 of file ptypes_tms570.h.

7.25.3.9 typedef unsigned char ubyte1

unsigned 8bit integer

Definition at line 31 of file ptypes_tms570.h.

7.25.3.10 `typedef unsigned short int ubyte2`

unsigned 16bit integer

Definition at line 32 of file ptypes_tms570.h.

7.25.3.11 `typedef unsigned long ubyte4`

unsigned 32bit integer

Definition at line 33 of file ptypes_tms570.h.

7.25.3.12 `typedef unsigned long long ubyte8`

unsigned 64bit integer

Definition at line 34 of file ptypes_tms570.h.

Index

— Symbols —	
_c_int00	
_c_int00	
ptypes_apdb.h	361
— A —	
ABRDTTimeout	
ABRDTTimeout	
bl_apdb_	15
adc_val_lower	
adc_val_lower	
io_adc_safety_conf_	22
adc_val_upper	
adc_val_upper	
io_adc_safety_conf_	22
APDB.h	
APDB.h	38
APDB_FLAGS_ABRD_ENABLE	40
APDB_FLAGS_CRC64_ENABLE	40
APDB_FLAGS_MULTI_APP	40
Apdb_t	41
APDB_VERSION	40
BL_APDB	41
BL_T_CAN_ID	41
BL_T_DATE	41
APDB_FLAGS_ABRD_ENABLE	
APDB_FLAGS_ABRD_ENABLE	
APDB.h	40
APDB_FLAGS_CRC64_ENABLE	
APDB_FLAGS_CRC64_ENABLE	
APDB.h	40
APDB_FLAGS_MULTI_APP	
APDB_FLAGS_MULTI_APP	
APDB.h	40
Apdb_t	
Apdb_t	
APDB.h	41
APDB_VERSION	
APDB_VERSION	
APDB.h	40
APDBVersion	
APDBVersion	
bl_apdb_	15
APPL_START	
APPL_START	
ptypes_apdb.h	361
ApplicationCRC	
ApplicationCRC	
bl_apdb_	15
ApplicationID	
ApplicationID	
bl_apdb_	15
ApplicationVersion	
ApplicationVersion	
bl_apdb_	15
— B —	
BL_APDB	
BL_APDB	
APDB.h	41
bl_apdb_	
bl_apdb_	14
ABRDTTimeout	15
APDBVersion	15
ApplicationCRC	15
ApplicationID	15
ApplicationVersion	15
BuildDate	16
CANBaudrate	16
CANChannel	16
CANDownloadID	16
CANUploadID	16
CodeSize	16
CRCSeed	16
CRCStartAddress	17
DebugKey	17
DLMulticastIPAddress	17
Flags	17
FlashDate	17
HeaderCRC	17
Hook1	17
Hook2	18
Hook3	18
LegacyApplicationCRC	18
LegacyHeaderCRC	18
MagicSeed	18
MainAddress	18
ManufacturerID	18
NodeNumber	18
NodeType	19
Password	19

Reserved.....	19	count	
SubnetMask.....	19	io_pwm_current_queue_.....	35
TargetIPAddress.....	19	CRCSeed	
BL_T_CAN_ID		CRCSeed	
BL_T_CAN_ID		bl_apdb_.....	16
APDB.h	41	CRCStartAddress	
bl_t_can_id_		CRCStartAddress	
bl_t_can_id_.....	19	bl_apdb_.....	17
extended.....	20	current_limit	
ID.....	20	current_limit	
BL_T_DATE		io_pwm_safety_conf_.....	36
BL_T_DATE			
APDB.h	41		
bl_t_date_		— D —	
bl_t_date_.....	20	data	
date.....	20	data	
BOOL		io_can_data_frame_.....	23
BOOL		io_lin_data_frame_.....	28
ptypes_tms570.h.....	364	date	
bool		date	
bool		bl_t_date_.....	20
ptypes_tms570.h.....	364	DebugKey	
BuildDate		DebugKey	
BuildDate		bl_apdb_.....	17
bl_apdb_.....	16	device_num	
— C —		device_num	
CANBaudrate		diag_errorcode_.....	21
CANBaudrate		DIAG_Constants.h	
bl_apdb_.....	16	DIAG_Constants.h	
CANChannel		DIAG_DEV_1V2.....	49
CANChannel		DIAG_DEV_2MODE_CONF_0.....	49
bl_apdb_.....	16	DIAG_DEV_2MODE_CONF_1.....	49
CANDownloadID		DIAG_DEV_2MODE_CONF_2.....	49
CANDownloadID		DIAG_DEV_2MODE_CONF_3.....	50
bl_apdb_.....	16	DIAG_DEV_2MODE_CONF_4.....	50
CANUploadID		DIAG_DEV_2MODE_CONF_5.....	50
CANUploadID		DIAG_DEV_2MODE_CONF_6.....	50
bl_apdb_.....	16	DIAG_DEV_2MODE_CONF_7.....	50
capture_count		DIAG_DEV_ADC.....	50
capture_count		DIAG_DEV_DIO.....	50
io_pwd_cplx_conf_.....	30	DIAG_DEV_DMA.....	50
CodeSize		DIAG_DEV_DO_CONF_0.....	51
CodeSize		DIAG_DEV_DO_CONF_1.....	51
bl_apdb_.....	16	DIAG_DEV_DO_CONF_2.....	51
command_period		DIAG_DEV_ESM.....	51
command_period		DIAG_DEV_ESM_CCM_R4_SLFTST.....	51
io_driver_safety_conf_.....	27	DIAG_DEV_ESM_CCMR4_COMPARE.....	51
count		DIAG_DEV_ESM_CLOCK_MONITOR.....	51
		DIAG_DEV_ESM_CPU_SLFTST.....	51
		DIAG_DEV_ESM_DCC1_ERROR.....	52

DIAG_DEV_ESM_DCC2	52
DIAG_DEV_ESM_DMA_DMM_IMPR_READ	52
DIAG_DEV_ESM_DMA_DMM_IMPR_WRITE	52
DIAG_DEV_ESM_DMA_MPUMPU_VIOLATION	52
DIAG_DEV_ESM_DMA_PARITY	52
DIAG_DEV_ESM_E_FUSE_AUTOLOAD	52
DIAG_DEV_ESM_E_FUSE_CNTL	52
DIAG_DEV_ESM_E_FUSE_CNTL_SLFTST	53
DIAG_DEV_ESM_FLASH_ECC_LIVE_LOCK	53
DIAG_DEV_ESM_FMC_B1_B2_UNC_ERR	53
DIAG_DEV_ESM_FMC_B1_UNC_ERR	53
DIAG_DEV_ESM_FMC_CFG_FLASH_UNC_ERR	53
DIAG_DEV_ESM_HET_TU1_2_PARITY	53
DIAG_DEV_ESM_IOMM_MUX_CONFIG	53
DIAG_DEV_ESM_MIBADC1_PARITY ..	53
DIAG_DEV_ESM_MIBADC2_PARITY ..	54
DIAG_DEV_ESM_MIBSPI1_PARITY ..	54
DIAG_DEV_ESM_MIBSPI3_PARITY ..	54
DIAG_DEV_ESM_N2HET1_2_PARITY	54
DIAG_DEV_ESM_PLL1_SLIP	54
DIAG_DEV_ESM_PLL2_SLIP	54
DIAG_DEV_ESM_PWR_DOM_CNTL_COMP	54
DIAG_DEV_ESM_PWR_DOM_CNTL_SLFTST	54
DIAG_DEV_ESM_RAM_B0_ADDR_PARITY	55
DIAG_DEV_ESM_RAM_B0_ECC_UNC_ERR	55
DIAG_DEV_ESM_RAM_B0_UNC_ERR	55
DIAG_DEV_ESM_RAM_B1_ADDR_PARITY	55
DIAG_DEV_ESM_RAM_B1_ECC_UNC_ERR	55
DIAG_DEV_ESM_RAM_B1_UNC_ERR	55
DIAG_DEV_ESM_RTI_WDD_NMI	55
DIAG_DEV_ESM_UNKNOWN	55
DIAG_DEV_ESM_VIM_RAM_PARITY ..	56
DIAG_DEV_EXT_SHUTOFF_0	56
DIAG_DEV_EXT_SHUTOFF_1	56
DIAG_DEV_EXT_SHUTOFF_2	56
DIAG_DEV_MAIN_CPU	56
DIAG_DEV_MAX	56
DIAG_DEV MCU	56
DIAG_DEV_NHET	56
DIAG_DEV_NONE	57
DIAG_DEV_PWD	57
DIAG_DEV_PWD_CONF_0	57
DIAG_DEV_PWD_CONF_1	57
DIAG_DEV_PWD_CONF_2	57
DIAG_DEV_PWD_CONF_3	57
DIAG_DEV_PWD_CONF_4	57
DIAG_DEV_PWD_CONF_5	57
DIAG_DEV_REF_2V5	58
DIAG_DEV_RTC	58
DIAG_DEV_SAFETY_SW_VP	58
DIAG_DEV_SPI	58
DIAG_DEV_VIM	58
DIAG_DEV_VMON	58
DIAG_DEV_WATCHDOG_CPU	58
DIAG_E_ADC_1V2	58
DIAG_E_ADC_2MODE_RED_CHANNEL_TEST	59
DIAG_E_ADC_2V5_REF	59
DIAG_E_ADC_3MODE_SWITCH_PERIODIC	59
DIAG_E_ADC_3MODE_SWITCH_TEST	59
DIAG_E_ADC_BOARD_TEMP	59
DIAG_E_ADC_RANGE	60
DIAG_E_ADC_SENSOR_SUPPLY ..	60
DIAG_E_ADC_SR_CONF_CHECK ..	60
DIAG_E_ADC_UBAT	60
DIAG_E_ADC_VPGATE	60
DIAG_E_APPL_SAFE_STATE	60
DIAG_E_CORE_READBACK	61
DIAG_E_DATA_ABORT	61
DIAG_E_DO_FEEDBACK	61
DIAG_E_DO_OPEN_LOAD	61
DIAG_E_DO_SHORT_CIRCUIT	61
DIAG_E_DRIVER_INIT	62
DIAG_E_ENABLE_TREE_TEST	62
DIAG_E_ERROR_CALLBACK_RECURSION	62
DIAG_E_ESM_HLI	62
DIAG_E_ESM_LLI	62
DIAG_E_ESM_LLI_CALLBACK	62
DIAG_E_INIT_CORE_ADD_BUS_PAR_B0	63
DIAG_E_INIT_CORE_ADD_BUS_PAR_B1	63

DIAG_E_INIT_CORE_ADD_DECODE_B0	63	DIAG_E_PWM_OPEN_LOAD	69
DIAG_E_INIT_CORE_ADD_DECODE_B1	63	DIAG_E_PWM_SHORT_CIRCUIT.....	69
DIAG_E_INIT_CORE_CCM_SELFTEST	63	DIAG_E_SSW_EXT_SHUTOFF	69
DIAG_E_INIT_CORE_CFG_FLASH_ECC	63	DIAG_E_SSW_PERIODIC	69
DIAG_E_INIT_CORE_DCC1_SELFTEST	63	DIAG_E_SSW_TEST	69
DIAG_E_INIT_CORE_DCC2_SELFTEST	63	DIAG_E_UNDEF_INSTRUCTION	70
DIAG_E_INIT_CORE_DMA_BASIC_TEST	64	DIAG_E_VMON_PERIODIC.....	70
DIAG_E_INIT_CORE_EFUSE_ECC ...	64	DIAG_E_VMON_TEST	70
DIAG_E_INIT_CORE_ERROR_PIN_TEST	64	DIAG_E_WD_ACTIVATION.....	70
DIAG_E_INIT_CORE_FLASH_BUS1_PAR	64	DIAG_E_WD_INIT	70
DIAG_E_INIT_CORE_FLASH_DATA_ECC	64	DIAG_E_WD_TRIGGER	70
DIAG_E_INIT_CORE_FLASH_WR_ECC	64	DIAG_ERR_DISABLE_HS00	71
DIAG_E_INIT_CORE_IOMM_LOCK...	64	DIAG_ERR_DISABLE_HS01	71
DIAG_E_INIT_CORE_IOMM_PROT_TEST	64	DIAG_ERR_DISABLE_HS02	71
DIAG_E_INIT_CORE_L2L3	65	DIAG_ERR_DISABLE_HS03	71
DIAG_E_INIT_CORE_MPUS	65	DIAG_ERR_DISABLE_HS04	71
DIAG_E_INIT_CORE_OSC_FAIL_TEST	65	DIAG_ERR_DISABLE_HS05	71
DIAG_E_INIT_CORE_PBIST_TEST...	65	DIAG_ERR_DISABLE_HS06	71
DIAG_E_INIT_CORE_PLL1_SLIP_TEST	65	DIAG_ERR_DISABLE_HS07	71
DIAG_E_INIT_CORE_PLL2_SLIP_TEST	65	DIAG_ERR_DISABLE_SSW0	72
DIAG_E_INIT_CORE_PSCON_SELFTEST	65	DIAG_ERR_DISABLE_SSW1	72
DIAG_E_INIT_CORE_RAM_ECC_B0 .	65	DIAG_ERR_DISABLE_SSW2	72
DIAG_E_INIT_CORE_RAM_ECC_B1 .	66	DIAG_ERR_NOACTION.....	72
DIAG_E_INIT_CORE_RAM_PARITY_TEST	66	DIAG_ERR_SAFESTATE.....	72
DIAG_E_INIT_CORE_RAM_PBIST ...	66	DIAG_ERROR_CB	74
DIAG_E_INIT_CORE_SELFTEST	66	DIAG_ERRORCODE	75
DIAG_E_INIT_CORE_STC_TEST	66	DIAG_NOTIFY_CB	75
DIAG_E_INVALID_DIAG_STATE	66	DIAG_STATE_CONFIG.....	72
DIAG_E_INVALID_IRQ	66	DIAG_STATE_DISABLED	72
DIAG_E_MAIN_LOOP	66	DIAG_STATE_INIT	72
DIAG_E_NOERROR	67	DIAG_STATE_MAIN	73
DIAG_E_PARITY_FBACK.....	67	DIAG_STATE_SAFE	73
DIAG_E_PREFETCH_ABORT	67	DIAG_WD_STATE_ACTIVE	73
DIAG_E_PRG_OVERFLOW.....	67	DIAG_WD_STATE_DIAGNOSTIC	73
DIAG_E_PWD_CURRENT	67	DIAG_WD_STATE_RESET	73
DIAG_E_PWD_RANGE	67	DIAG_WD_STATE_SAFE	73
DIAG_E_PWD_THRESH.....	68	DIAG_WD_STATE_STANDBY	73
DIAG_E_PWM_CURRENT.....	68	DIAG_WD_STATE_UNKNOWN.....	73
DIAG_E_PWM_FEEDBACK.....	68	DIAG_DEV_1V2	
		DIAG_DEV_1V2	
		DIAG_Constants.h	49
		DIAG_DEV_2MODE_CONF_0	
		DIAG_DEV_2MODE_CONF_0	
		DIAG_Constants.h	49
		DIAG_DEV_2MODE_CONF_1	
		DIAG_DEV_2MODE_CONF_1	
		DIAG_Constants.h	49
		DIAG_DEV_2MODE_CONF_2	
		DIAG_DEV_2MODE_CONF_2	

DIAG_Constants.h	49
DIAG_DEV_2MODE_CONF_3	
DIAG_DEV_2MODE_CONF_3	
DIAG_Constants.h	50
DIAG_DEV_2MODE_CONF_4	
DIAG_DEV_2MODE_CONF_4	
DIAG_Constants.h	50
DIAG_DEV_2MODE_CONF_5	
DIAG_DEV_2MODE_CONF_5	
DIAG_Constants.h	50
DIAG_DEV_2MODE_CONF_6	
DIAG_DEV_2MODE_CONF_6	
DIAG_Constants.h	50
DIAG_DEV_2MODE_CONF_7	
DIAG_DEV_2MODE_CONF_7	
DIAG_Constants.h	50
DIAG_DEV_ADC	
DIAG_DEV_ADC	
DIAG_Constants.h	50
DIAG_DEV_DIO	
DIAG_DEV_DIO	
DIAG_Constants.h	50
DIAG_DEV_DMA	
DIAG_DEV_DMA	
DIAG_Constants.h	50
DIAG_DEV_DO_CONF_0	
DIAG_DEV_DO_CONF_0	
DIAG_Constants.h	51
DIAG_DEV_DO_CONF_1	
DIAG_DEV_DO_CONF_1	
DIAG_Constants.h	51
DIAG_DEV_DO_CONF_2	
DIAG_DEV_DO_CONF_2	
DIAG_Constants.h	51
DIAG_DEV_ESM	
DIAG_DEV_ESM	
DIAG_Constants.h	51
DIAG_DEV_ESM_CCM_R4_SLFTST	
DIAG_DEV_ESM_CCM_R4_SLFTST	
DIAG_Constants.h	51
DIAG_DEV_ESM_CCMR4_COMPARE	
DIAG_DEV_ESM_CCMR4_COMPARE	
DIAG_Constants.h	51
DIAG_DEV_ESM_CLOCK_MONITOR	
DIAG_DEV_ESM_CLOCK_MONITOR	
DIAG_Constants.h	51
DIAG_DEV_ESM_CPU_SLFTST	
DIAG_DEV_ESM_CPU_SLFTST	
DIAG_Constants.h	51
DIAG_DEV_ESM_DCC1_ERROR	
DIAG_DEV_ESM_DCC1_ERROR	
DIAG_Constants.h	52
DIAG_DEV_ESM_DCC2	
DIAG_DEV_ESM_DCC2	
DIAG_Constants.h	52
DIAG_DEV_ESM_DMA_DMM_IMPR_READ	
DIAG_DEV_ESM_DMA_DMM_IMPR_READ	
DIAG_Constants.h	52
DIAG_DEV_ESM_DMA_DMM_IMPR_WRITE	
DIAG_DEV_ESM_DMA_DMM_IMPR_WRITE	
DIAG_Constants.h	52
DIAG_DEV_ESM_DMA_MPUI_VIOLATION	
DIAG_DEV_ESM_DMA_MPUI_VIOLATION	
DIAG_Constants.h	52
DIAG_DEV_ESM_DMA_PARITY	
DIAG_DEV_ESM_DMA_PARITY	
DIAG_Constants.h	52
DIAG_DEV_ESM_E_FUSE_AUTOLOAD	
DIAG_DEV_ESM_E_FUSE_AUTOLOAD	
DIAG_Constants.h	52
DIAG_DEV_ESM_E_FUSE_CNTL	
DIAG_DEV_ESM_E_FUSE_CNTL	
DIAG_Constants.h	52
DIAG_DEV_ESM_E_FUSE_CNTL_SLFTST	
DIAG_DEV_ESM_E_FUSE_CNTL_SLFTST	
DIAG_Constants.h	53
DIAG_DEV_ESM_FLASH_ECC_LIVE_LOCK	
DIAG_DEV_ESM_FLASH_ECC_LIVE_LOCK	
DIAG_Constants.h	53
DIAG_DEV_ESM_FMC_B1_B2_UNC_ERR	
DIAG_DEV_ESM_FMC_B1_B2_UNC_ERR	
DIAG_Constants.h	53
DIAG_DEV_ESM_FMC_B1_UNC_ERR	
DIAG_DEV_ESM_FMC_B1_UNC_ERR	
DIAG_Constants.h	53
DIAG_DEV_ESM_FMC_CFG_FLASH_UNC_ERR	
DIAG_DEV_ESM_FMC_CFG_FLASH_UNC_	
ERR	
DIAG_Constants.h	53
DIAG_DEV_ESM_HET_TU1_2_PARITY	
DIAG_DEV_ESM_HET_TU1_2_PARITY	
DIAG_Constants.h	53
DIAG_DEV_ESM_IOMM_MUX_CONFIG	
DIAG_DEV_ESM_IOMM_MUX_CONFIG	
DIAG_Constants.h	53
DIAG_DEV_ESM_MIBADC1_PARITY	
DIAG_DEV_ESM_MIBADC1_PARITY	
DIAG_Constants.h	53

DIAG_DEV_ESM_MIBADC2_PARITY	
DIAG_DEV_ESM_MIBADC2_PARITY	
DIAG_Constants.h	54
DIAG_DEV_ESM_MIBSPI1_PARITY	
DIAG_DEV_ESM_MIBSPI1_PARITY	
DIAG_Constants.h	54
DIAG_DEV_ESM_MIBSPI3_PARITY	
DIAG_DEV_ESM_MIBSPI3_PARITY	
DIAG_Constants.h	54
DIAG_DEV_ESM_N2HET1_2_PARITY	
DIAG_DEV_ESM_N2HET1_2_PARITY	
DIAG_Constants.h	54
DIAG_DEV_ESM_PLL1_SLIP	
DIAG_DEV_ESM_PLL1_SLIP	
DIAG_Constants.h	54
DIAG_DEV_ESM_PLL2_SLIP	
DIAG_DEV_ESM_PLL2_SLIP	
DIAG_Constants.h	54
DIAG_DEV_ESM_PWR_DOM_CNTL_COMP	
DIAG_DEV_ESM_PWR_DOM_CNTL_COMP	
DIAG_Constants.h	54
DIAG_DEV_ESM_PWR_DOM_CNTL_SLFTST	
DIAG_DEV_ESM_PWR_DOM_CNTL_SLFTST	
DIAG_Constants.h	54
DIAG_DEV_ESM_RAM_B0_ADDR_PARITY	
DIAG_DEV_ESM_RAM_B0_ADDR_PARITY	
DIAG_Constants.h	55
DIAG_DEV_ESM_RAM_B0_ECC_UNC_ERR	
DIAG_DEV_ESM_RAM_B0_ECC_UNC_ERR	
DIAG_Constants.h	55
DIAG_DEV_ESM_RAM_B0_UNC_ERR	
DIAG_DEV_ESM_RAM_B0_UNC_ERR	
DIAG_Constants.h	55
DIAG_DEV_ESM_RAM_B1_ADDR_PARITY	
DIAG_DEV_ESM_RAM_B1_ADDR_PARITY	
DIAG_Constants.h	55
DIAG_DEV_ESM_RAM_B1_ECC_UNC_ERR	
DIAG_DEV_ESM_RAM_B1_ECC_UNC_ERR	
DIAG_Constants.h	55
DIAG_DEV_ESM_RAM_B1_UNC_ERR	
DIAG_DEV_ESM_RAM_B1_UNC_ERR	
DIAG_Constants.h	55
DIAG_DEV_ESM_RTI_WDD_NMI	
DIAG_DEV_ESM_RTI_WDD_NMI	
DIAG_Constants.h	55
DIAG_DEV_ESM_UNKNOWN	
DIAG_DEV_ESM_UNKNOWN	
DIAG_Constants.h	55
DIAG_DEV_VIM_RAM_PARITY	
DIAG_DEV_VIM_RAM_PARITY	
DIAG_Constants.h	56
DIAG_DEV_EXT_SHUTOFF_0	
DIAG_DEV_EXT_SHUTOFF_0	
DIAG_Constants.h	56
DIAG_DEV_EXT_SHUTOFF_1	
DIAG_DEV_EXT_SHUTOFF_1	
DIAG_Constants.h	56
DIAG_DEV_EXT_SHUTOFF_2	
DIAG_DEV_EXT_SHUTOFF_2	
DIAG_Constants.h	56
DIAG_DEV_MAIN_CPU	
DIAG_DEV_MAIN_CPU	
DIAG_Constants.h	56
DIAG_DEV_MAX	
DIAG_DEV_MAX	
DIAG_Constants.h	56
DIAG_DEV MCU	
DIAG_DEV MCU	
DIAG_Constants.h	56
DIAG_DEV_NHET	
DIAG_DEV_NHET	
DIAG_Constants.h	56
DIAG_DEV_NONE	
DIAG_DEV_NONE	
DIAG_Constants.h	57
DIAG_DEV_PWD	
DIAG_DEV_PWD	
DIAG_Constants.h	57
DIAG_DEV_PWD_CONF_0	
DIAG_DEV_PWD_CONF_0	
DIAG_Constants.h	57
DIAG_DEV_PWD_CONF_1	
DIAG_DEV_PWD_CONF_1	
DIAG_Constants.h	57
DIAG_DEV_PWD_CONF_2	
DIAG_DEV_PWD_CONF_2	
DIAG_Constants.h	57
DIAG_DEV_PWD_CONF_3	
DIAG_DEV_PWD_CONF_3	
DIAG_Constants.h	57
DIAG_DEV_PWD_CONF_4	
DIAG_DEV_PWD_CONF_4	
DIAG_Constants.h	57
DIAG_DEV_PWD_CONF_5	
DIAG_DEV_PWD_CONF_5	
DIAG_Constants.h	57
DIAG_DEV_REF_2V5	
DIAG_DEV_REF_2V5	

DIAG_Constants.h	58	DIAG_E_ADC_VPGATE	
DIAG_DEV_RTC		DIAG_E_ADC_VPGATE	
DIAG_DEV_RTC		DIAG_Constants.h	60
DIAG_Constants.h	58	DIAG_E_APPL_SAFE_STATE	
DIAG_DEV_SAFETY_SW_VP		DIAG_E_APPL_SAFE_STATE	
DIAG_DEV_SAFETY_SW_VP		DIAG_Constants.h	60
DIAG_Constants.h	58	DIAG_E_CORE_READBACK	
DIAG_DEV_SPI		DIAG_E_CORE_READBACK	
DIAG_DEV_SPI		DIAG_Constants.h	61
DIAG_Constants.h	58	DIAG_E_DATA_ABORT	
DIAG_DEV_VIM		DIAG_E_DATA_ABORT	
DIAG_DEV_VIM		DIAG_Constants.h	61
DIAG_Constants.h	58	DIAG_E_DO_FEEDBACK	
DIAG_DEV_VMON		DIAG_E_DO_FEEDBACK	
DIAG_DEV_VMON		DIAG_Constants.h	61
DIAG_Constants.h	58	DIAG_E_DO_OPEN_LOAD	
DIAG_DEV_WATCHDOG_CPU		DIAG_E_DO_OPEN_LOAD	
DIAG_DEV_WATCHDOG_CPU		DIAG_Constants.h	61
DIAG_Constants.h	58	DIAG_E_DO_SHORT_CIRCUIT	
DIAG_E_ADC_1V2		DIAG_E_DO_SHORT_CIRCUIT	
DIAG_E_ADC_1V2		DIAG_Constants.h	61
DIAG_Constants.h	58	DIAG_E_DRIVER_INIT	
DIAG_E_ADC_2MODE_RED_CHANNEL_TEST		DIAG_E_DRIVER_INIT	
DIAG_E_ADC_2MODE_RED_CHANNEL_TEST		DIAG_Constants.h	62
DIAG_Constants.h	59	DIAG_E_ENABLE_TREE_TEST	
DIAG_E_ADC_2V5_REF		DIAG_E_ENABLE_TREE_TEST	
DIAG_E_ADC_2V5_REF		DIAG_Constants.h	62
DIAG_Constants.h	59	DIAG_E_ERROR_CALLBACK_RECURSION	
DIAG_E_ADC_3MODE_SWITCH_PERIODIC		DIAG_E_ERROR_CALLBACK_RECURSION	
DIAG_E_ADC_3MODE_SWITCH_PERIODIC		DIAG_Constants.h	62
DIAG_Constants.h	59	DIAG_E_ESM_HLI	
DIAG_E_ADC_3MODE_SWITCH_TEST		DIAG_E_ESM_HLI	
DIAG_E_ADC_3MODE_SWITCH_TEST		DIAG_Constants.h	62
DIAG_Constants.h	59	DIAG_E_ESM_LLI	
DIAG_E_ADC_BOARD_TEMP		DIAG_E_ESM_LLI	
DIAG_E_ADC_BOARD_TEMP		DIAG_Constants.h	62
DIAG_Constants.h	59	DIAG_E_ESM_LLI_CALLBACK	
DIAG_E_ADC_RANGE		DIAG_E_ESM_LLI_CALLBACK	
DIAG_E_ADC_RANGE		DIAG_Constants.h	62
DIAG_Constants.h	60	DIAG_E_INIT_CORE_ADD_BUS_PAR_B0	
DIAG_E_ADC_SENSOR_SUPPLY		DIAG_E_INIT_CORE_ADD_BUS_PAR_B0	
DIAG_E_ADC_SENSOR_SUPPLY		DIAG_Constants.h	63
DIAG_Constants.h	60	DIAG_E_INIT_CORE_ADD_BUS_PAR_B1	
DIAG_E_ADC_SR_CONF_CHECK		DIAG_E_INIT_CORE_ADD_BUS_PAR_B1	
DIAG_E_ADC_SR_CONF_CHECK		DIAG_Constants.h	63
DIAG_Constants.h	60	DIAG_E_INIT_CORE_ADD_DECODE_B0	
DIAG_E_ADC_UBAT		DIAG_E_INIT_CORE_ADD_DECODE_B0	
DIAG_E_ADC_UBAT		DIAG_Constants.h	63
DIAG_Constants.h	60	DIAG_E_INIT_CORE_ADD_DECODE_B1	

DIAG_E_INIT_CORE_ADD_DECODE_B1	DIAG_Constants.h	65
DIAG_E_INIT_CORE_CCM_SELFTEST	DIAG_E_INIT_CORE_CCM_SELFTEST	
DIAG_E_INIT_CORE_CCM_SELFTEST	DIAG_Constants.h	63
DIAG_E_INIT_CORE_CFG_FLASH_ECC	DIAG_E_INIT_CORE_CFG_FLASH_ECC	
DIAG_E_INIT_CORE_CFG_FLASH_ECC	DIAG_Constants.h	63
DIAG_E_INIT_CORE_DCC1_SELFTEST	DIAG_E_INIT_CORE_DCC1_SELFTEST	
DIAG_E_INIT_CORE_DCC1_SELFTEST	DIAG_Constants.h	63
DIAG_E_INIT_CORE_DCC2_SELFTEST	DIAG_E_INIT_CORE_DCC2_SELFTEST	
DIAG_E_INIT_CORE_DCC2_SELFTEST	DIAG_Constants.h	63
DIAG_E_INIT_CORE_DMA_BASIC_TEST	DIAG_E_INIT_CORE_DMA_BASIC_TEST	
DIAG_E_INIT_CORE_DMA_BASIC_TEST	DIAG_Constants.h	64
DIAG_E_INIT_CORE_EFUSE_ECC	DIAG_E_INIT_CORE_EFUSE_ECC	
DIAG_E_INIT_CORE_EFUSE_ECC	DIAG_Constants.h	64
DIAG_E_INIT_CORE_ERROR_PIN_TEST	DIAG_E_INIT_CORE_ERROR_PIN_TEST	
DIAG_E_INIT_CORE_ERROR_PIN_TEST	DIAG_Constants.h	64
DIAG_E_INIT_CORE_FLASH_BUS1_PAR	DIAG_E_INIT_CORE_FLASH_BUS1_PAR	
DIAG_E_INIT_CORE_FLASH_BUS1_PAR	DIAG_Constants.h	64
DIAG_E_INIT_CORE_FLASH_DATA_ECC	DIAG_E_INIT_CORE_FLASH_DATA_ECC	
DIAG_E_INIT_CORE_FLASH_DATA_ECC	DIAG_Constants.h	64
DIAG_E_INIT_CORE_FLASH_WR_ECC	DIAG_E_INIT_CORE_FLASH_WR_ECC	
DIAG_E_INIT_CORE_FLASH_WR_ECC	DIAG_Constants.h	64
DIAG_E_INIT_CORE_IOMM_LOCK	DIAG_E_INIT_CORE_IOMM_LOCK	
DIAG_E_INIT_CORE_IOMM_LOCK	DIAG_Constants.h	64
DIAG_E_INIT_CORE_IOMM_PROT_TEST	DIAG_E_INIT_CORE_IOMM_PROT_TEST	
DIAG_E_INIT_CORE_IOMM_PROT_TEST	DIAG_Constants.h	64
DIAG_E_INIT_CORE_L2L3	DIAG_E_INIT_CORE_L2L3	
DIAG_E_INIT_CORE_L2L3	DIAG_Constants.h	65
DIAG_E_INIT_CORE_MPUS_TEST	DIAG_E_INIT_CORE_MPUS_TEST	
DIAG_E_INIT_CORE_MPUS_TEST	DIAG_Constants.h	65
DIAG_E_INIT_CORE_OSC_FAIL_TEST	DIAG_E_INIT_CORE_OSC_FAIL_TEST	
DIAG_E_INIT_CORE_OSC_FAIL_TEST	DIAG_Constants.h	65
DIAG_E_INIT_CORE_PBIST_TEST	DIAG_E_INIT_CORE_PBIST_TEST	
DIAG_E_INIT_CORE_PBIST_TEST	DIAG_Constants.h	65
DIAG_Constants.h	DIAG_Constants.h	65
DIAG_E_INIT_CORE_PLL1_SLIP_TEST	DIAG_E_INIT_CORE_PLL1_SLIP_TEST	
DIAG_E_INIT_CORE_PLL1_SLIP_TEST	DIAG_Constants.h	65
DIAG_E_INIT_CORE_PLL2_SLIP_TEST	DIAG_E_INIT_CORE_PLL2_SLIP_TEST	
DIAG_E_INIT_CORE_PLL2_SLIP_TEST	DIAG_Constants.h	65
DIAG_E_INIT_CORE_PSCON_SELFTEST	DIAG_E_INIT_CORE_PSCON_SELFTEST	
DIAG_E_INIT_CORE_PSCON_SELFTEST	DIAG_Constants.h	65
DIAG_E_INIT_CORE_RAM_ECC_B0	DIAG_E_INIT_CORE_RAM_ECC_B0	
DIAG_E_INIT_CORE_RAM_ECC_B0	DIAG_Constants.h	65
DIAG_E_INIT_CORE_RAM_ECC_B1	DIAG_E_INIT_CORE_RAM_ECC_B1	
DIAG_E_INIT_CORE_RAM_ECC_B1	DIAG_Constants.h	66
DIAG_E_INIT_CORE_RAM_PARITY_TEST	DIAG_E_INIT_CORE_RAM_PARITY_TEST	
DIAG_E_INIT_CORE_RAM_PARITY_TEST	DIAG_Constants.h	66
DIAG_E_INIT_CORE_RAM_PBIST	DIAG_E_INIT_CORE_RAM_PBIST	
DIAG_E_INIT_CORE_RAM_PBIST	DIAG_Constants.h	66
DIAG_E_INIT_CORE_SELFTEST	DIAG_E_INIT_CORE_SELFTEST	
DIAG_E_INIT_CORE_SELFTEST	DIAG_Constants.h	66
DIAG_E_INIT_CORE_STC_TEST	DIAG_E_INIT_CORE_STC_TEST	
DIAG_E_INIT_CORE_STC_TEST	DIAG_Constants.h	66
DIAG_E_INVALID_DIAG_STATE	DIAG_E_INVALID_DIAG_STATE	
DIAG_E_INVALID_DIAG_STATE	DIAG_Constants.h	66
DIAG_E_INVALID_IRQ	DIAG_E_INVALID_IRQ	
DIAG_E_INVALID_IRQ	DIAG_Constants.h	66
DIAG_E_MAIN_LOOP	DIAG_E_MAIN_LOOP	
DIAG_E_MAIN_LOOP	DIAG_Constants.h	66
DIAG_E_NOERROR	DIAG_E_NOERROR	
DIAG_E_NOERROR	DIAG_Constants.h	67
DIAG_E_PARITY_FALLBACK	DIAG_E_PARITY_FALLBACK	
DIAG_E_PARITY_FALLBACK	DIAG_Constants.h	67
DIAG_E_PREFETCH_ABORT	DIAG_E_PREFETCH_ABORT	
DIAG_E_PREFETCH_ABORT	DIAG_Constants.h	67
DIAG_E_PRG_OVERFLOW	DIAG_E_PRG_OVERFLOW	
DIAG_E_PRG_OVERFLOW	DIAG_Constants.h	67

DIAG_E_PWD_CURRENT	
DIAG_E_PWD_CURRENT	
DIAG_Constants.h	67
DIAG_E_PWD_RANGE	
DIAG_E_PWD_RANGE	
DIAG_Constants.h	67
DIAG_E_PWD_THRESH	
DIAG_E_PWD_THRESH	
DIAG_Constants.h	68
DIAG_E_PWM_CURRENT	
DIAG_E_PWM_CURRENT	
DIAG_Constants.h	68
DIAG_E_PWM_FEEDBACK	
DIAG_E_PWM_FEEDBACK	
DIAG_Constants.h	68
DIAG_E_PWM_OPEN_LOAD	
DIAG_E_PWM_OPEN_LOAD	
DIAG_Constants.h	69
DIAG_E_PWM_SHORT_CIRCUIT	
DIAG_E_PWM_SHORT_CIRCUIT	
DIAG_Constants.h	69
DIAG_E_SSW_EXT_SHUTOFF	
DIAG_E_SSW_EXT_SHUTOFF	
DIAG_Constants.h	69
DIAG_E_SSW_PERIODIC	
DIAG_E_SSW_PERIODIC	
DIAG_Constants.h	69
DIAG_E_SSW_TEST	
DIAG_E_SSW_TEST	
DIAG_Constants.h	69
DIAG_E_UNDEF_INSTRUCTION	
DIAG_E_UNDEF_INSTRUCTION	
DIAG_Constants.h	70
DIAG_E_VMON_PERIODIC	
DIAG_E_VMON_PERIODIC	
DIAG_Constants.h	70
DIAG_E_VMON_TEST	
DIAG_E_VMON_TEST	
DIAG_Constants.h	70
DIAG_E_WD_ACTIVATION	
DIAG_E_WD_ACTIVATION	
DIAG_Constants.h	70
DIAG_E_WD_INIT	
DIAG_E_WD_INIT	
DIAG_Constants.h	70
DIAG_E_WD_TRIGGER	
DIAG_E_WD_TRIGGER	
DIAG_Constants.h	70
DIAG_EnterSafestate	
DIAG_EnterSafestate	
DIAG_Functions.h	77
DIAG_ERR_DISABLE_HS00	
DIAG_ERR_DISABLE_HS00	
DIAG_Constants.h	71
DIAG_ERR_DISABLE_HS01	
DIAG_ERR_DISABLE_HS01	
DIAG_Constants.h	71
DIAG_ERR_DISABLE_HS02	
DIAG_ERR_DISABLE_HS02	
DIAG_Constants.h	71
DIAG_ERR_DISABLE_HS03	
DIAG_ERR_DISABLE_HS03	
DIAG_Constants.h	71
DIAG_ERR_DISABLE_HS04	
DIAG_ERR_DISABLE_HS04	
DIAG_Constants.h	71
DIAG_ERR_DISABLE_HS05	
DIAG_ERR_DISABLE_HS05	
DIAG_Constants.h	71
DIAG_ERR_DISABLE_HS06	
DIAG_ERR_DISABLE_HS06	
DIAG_Constants.h	71
DIAG_ERR_DISABLE_HS07	
DIAG_ERR_DISABLE_HS07	
DIAG_Constants.h	71
DIAG_ERR_DISABLE_SSW0	
DIAG_ERR_DISABLE_SSW0	
DIAG_Constants.h	72
DIAG_ERR_DISABLE_SSW1	
DIAG_ERR_DISABLE_SSW1	
DIAG_Constants.h	72
DIAG_ERR_DISABLE_SSW2	
DIAG_ERR_DISABLE_SSW2	
DIAG_Constants.h	72
DIAG_ERR_NOACTION	
DIAG_ERR_NOACTION	
DIAG_Constants.h	72
DIAG_ERR_SAFESTATE	
DIAG_ERR_SAFESTATE	
DIAG_Constants.h	72
DIAG_ERROR_CB	
DIAG_ERROR_CB	
DIAG_Constants.h	74
DIAG_ERRORCODE	
DIAG_ERRORCODE	
DIAG_Constants.h	75
diag_errorcode_	
diag_errorcode_	21

device_num	21	DIAG_Constants.h	73
error_code	21	DIAG_WD_STATE_RESET	
faulty_value	21	DIAG_WD_STATE_RESET	
DIAG_Functions.h		DIAG_Constants.h	73
DIAG_Functions.h	75	DIAG_WD_STATE_SAFE	
DIAG_EnterSafestate.....	77	DIAG_WD_STATE_SAFE	
DIAG_GetCfgFlashErrors.....	77	DIAG_Constants.h	73
DIAG_GetFlashErrors	77	DIAG_WD_STATE_STANDBY	
DIAG_GetRamB0Errors	78	DIAG_WD_STATE_STANDBY	
DIAG_GetRamB1Errors	78	DIAG_Constants.h	73
DIAG_Status	79	DIAG_WD_STATE_UNKNOWN	
DIAG_GetCfgFlashErrors		DIAG_WD_STATE_UNKNOWN	
DIAG_GetCfgFlashErrors		DIAG_Constants.h	73
DIAG_Functions.h.....	77	direction	
DIAG_GetFlashErrors		direction	
DIAG_GetFlashErrors		io_pwd_cnt_conf_.....	29
DIAG_Functions.h.....	77	DLMulticastIPAddress	
DIAG_GetRamB0Errors		DLMulticastIPAddress	
DIAG_GetRamB0Errors		bl_apdb_.....	17
DIAG_Functions.h.....	78		
DIAG_GetRamB1Errors		— E —	
DIAG_GetRamB1Errors		enable_current_check	
DIAG_Functions.h.....	78	enable_current_check	
DIAG_NOTIFY_CB		io_pwr_safety_conf_.....	36
DIAG_NOTIFY_CB		error_callback	
DIAG_Constants.h	75	error_callback	
DIAG_STATE_CONFIG		io_driver_safety_conf_.....	27
DIAG_STATE_CONFIG		error_code	
DIAG_Constants.h	72	error_code	
DIAG_STATE_DISABLED		diag_errorcode_.....	21
DIAG_STATE_DISABLED		extended	
DIAG_Constants.h	72	extended	
DIAG_STATE_INIT		bl_t_can_id_.....	20
DIAG_STATE_INIT			
DIAG_Constants.h	72	— F —	
DIAG_STATE_MAIN		FALSE	
DIAG_STATE_MAIN		FALSE	
DIAG_Constants.h	73	ptypes_tms570.h.....	363
DIAG_STATE_SAFE		faulty_value	
DIAG_STATE_SAFE		faulty_value	
DIAG_Constants.h	73	diag_errorcode_.....	21
DIAG_Status		Flags	
DIAG_Status		Flags	
DIAG_Functions.h.....	79	bl_apdb_.....	17
DIAG_WD_STATE_ACTIVE		FlashDate	
DIAG_WD_STATE_ACTIVE		FlashDate	
DIAG_Constants.h	73	bl_apdb_.....	17
DIAG_WD_STATE_DIAGNOSTIC		float4	
DIAG_WD_STATE_DIAGNOSTIC			

float4		io_pwd_cnt_conf.....	29
ptypes_tms570.h	io_pwd_inc_conf.....	32
float8		IO_ADC.h	
float8		IO_ADC.h.....	80
ptypes_tms570.h	IO_ADC_ABSOLUTE	84
freq_mode		IO_ADC_BoardTempFloat.....	86
freq_mode		IO_ADC_BoardTempSbyte	86
io_pwd_cplx_conf_.....	30	IO_ADC_ChannelDelInit	87
— G —		IO_ADC_ChannellInit.....	88
glitch_filter_time		IO_ADC_CURRENT	84
glitch_filter_time		IO_ADC_Get.....	91
io_driver_safety_conf_.....	27	IO_ADC_NO_PULL.....	84
— H —		IO_ADC_NO_RANGE.....	84
HeaderCRC		IO_ADC_PD_10K	84
HeaderCRC		IO_ADC_PU_10K	84
bl_apdb_.....	17	IO_ADC_RANGE_10V	84
high_thresh1		IO_ADC_RANGE_32V	85
high_thresh1		IO_ADC_RANGE_5V	85
io_dio_limits_.....	24	IO_ADC_RATIOMETRIC	85
high_thresh2		IO_ADC_ResetProtection	93
high_thresh2		IO_ADC_RESISTIVE.....	85
io_dio_limits_.....	24	IO_ADC_SAFETY_CONF	85
Hook1		IO_ADC_00	
Hook1		IO_ADC_00	
bl_apdb_.....	17	IO_Pin.h	225
Hook2		IO_ADC_01	
Hook2		IO_ADC_01	
bl_apdb_.....	18	IO_Pin.h	225
Hook3		IO_ADC_02	
Hook3		IO_ADC_02	
bl_apdb_.....	18	IO_Pin.h	225
— I —		IO_ADC_03	
ID		IO_ADC_03	
ID		IO_Pin.h	225
bl_t_can_id_.....	20	IO_ADC_04	
id		IO_ADC_04	
id		IO_Pin.h	225
io_can_data_frame_.....	23	IO_ADC_05	
io_lin_data_frame_.....	28	IO_ADC_05	
id_format		IO_Pin.h	225
id_format		IO_ADC_06	
io_can_data_frame_.....	23	IO_ADC_06	
init		IO_Pin.h	226
init		IO_ADC_07	
		IO_ADC_07	
		IO_Pin.h	226
		IO_ADC_08	
		IO_ADC_08	
		IO_Pin.h	226

IO_ADC_09	
IO_ADC_09	
IO_Pin.h	226
IO_ADC_10	
IO_ADC_10	
IO_Pin.h	226
IO_ADC_11	
IO_ADC_11	
IO_Pin.h	226
IO_ADC_12	
IO_ADC_12	
IO_Pin.h	226
IO_ADC_13	
IO_ADC_13	
IO_Pin.h	226
IO_ADC_14	
IO_ADC_14	
IO_Pin.h	227
IO_ADC_15	
IO_ADC_15	
IO_Pin.h	227
IO_ADC_16	
IO_ADC_16	
IO_Pin.h	227
IO_ADC_17	
IO_ADC_17	
IO_Pin.h	227
IO_ADC_18	
IO_ADC_18	
IO_Pin.h	227
IO_ADC_19	
IO_ADC_19	
IO_Pin.h	227
IO_ADC_20	
IO_ADC_20	
IO_Pin.h	227
IO_ADC_21	
IO_ADC_21	
IO_Pin.h	227
IO_ADC_22	
IO_ADC_22	
IO_Pin.h	228
IO_ADC_23	
IO_ADC_23	
IO_Pin.h	228
IO_ADC_24	
IO_ADC_24	
IO_Pin.h	228
IO_ADC_25	
IO_ADC_25	
IO_Pin.h	228
IO_ADC_26	
IO_ADC_26	
IO_Pin.h	228
IO_ADC_27	
IO_ADC_27	
IO_Pin.h	228
IO_ADC_28	
IO_ADC_28	
IO_Pin.h	228
IO_ADC_29	
IO_ADC_29	
IO_Pin.h	228
IO_ADC_30	
IO_ADC_30	
IO_Pin.h	229
IO_ADC_31	
IO_ADC_31	
IO_Pin.h	229
IO_ADC_32	
IO_ADC_32	
IO_Pin.h	229
IO_ADC_33	
IO_ADC_33	
IO_Pin.h	229
IO_ADC_34	
IO_ADC_34	
IO_Pin.h	229
IO_ADC_35	
IO_ADC_35	
IO_Pin.h	229
IO_ADC_36	
IO_ADC_36	
IO_Pin.h	229
IO_ADC_37	
IO_ADC_37	
IO_Pin.h	229
IO_ADC_38	
IO_ADC_38	
IO_Pin.h	230
IO_ADC_39	
IO_ADC_39	
IO_Pin.h	230
IO_ADC_40	
IO_ADC_40	
IO_Pin.h	230
IO_ADC_41	
IO_ADC_41	

IO_Pin.h	230	IO_ADC_58	
IO_ADC_42		IO_ADC_58	
IO_ADC_42		IO_Pin.h	232
IO_Pin.h	230	IO_ADC_59	
IO_ADC_43		IO_ADC_59	
IO_ADC_43		IO_Pin.h	232
IO_Pin.h	230	IO_ADC_ABSOLUTE	
IO_ADC_44		IO_ADC_ABSOLUTE	
IO_ADC_44		IO_ADC.h	84
IO_Pin.h	230	IO_ADC_BOARD_TEMP	
IO_ADC_45		IO_ADC_BOARD_TEMP	
IO_ADC_45		IO_Pin.h	232
IO_Pin.h	230	IO_ADC_BoardTempFloat	
IO_ADC_46		IO_ADC_BoardTempFloat	
IO_ADC_46		IO_ADC.h	86
IO_Pin.h	231	IO_ADC_BoardTempSbyte	
IO_ADC_47		IO_ADC_BoardTempSbyte	
IO_ADC_47		IO_ADC.h	86
IO_Pin.h	231	IO_ADC_ChannelDelInit	
IO_ADC_48		IO_ADC_ChannelDelInit	
IO_ADC_48		IO_ADC.h	87
IO_Pin.h	231	IO_ADC_ChannellInit	
IO_ADC_49		IO_ADC_ChannellInit	
IO_ADC_49		IO_ADC.h	88
IO_Pin.h	231	IO_ADC_CURRENT	
IO_ADC_50		IO_ADC_CURRENT	
IO_ADC_50		IO_ADC.h	84
IO_Pin.h	231	IO_ADC_Get	
IO_ADC_51		IO_ADC_Get	
IO_ADC_51		IO_ADC.h	91
IO_Pin.h	231	IO_ADC_K15	
IO_ADC_52		IO_ADC_K15	
IO_ADC_52		IO_Pin.h	232
IO_Pin.h	231	IO_ADC_NO_PULL	
IO_ADC_53		IO_ADC_NO_PULL	
IO_ADC_53		IO_ADC.h	84
IO_Pin.h	231	IO_ADC_NO_RANGE	
IO_ADC_54		IO_ADC_NO_RANGE	
IO_ADC_54		IO_ADC.h	84
IO_Pin.h	232	IO_ADC_PD_10K	
IO_ADC_55		IO_ADC_PD_10K	
IO_ADC_55		IO_ADC.h	84
IO_Pin.h	232	IO_ADC_PU_10K	
IO_ADC_56		IO_ADC_PU_10K	
IO_ADC_56		IO_ADC.h	84
IO_Pin.h	232	IO_ADC_RANGE_10V	
IO_ADC_57		IO_ADC_RANGE_10V	
IO_ADC_57		IO_ADC.h	84
IO_Pin.h	232	IO_ADC_RANGE_32V	

IO_ADC_RANGE_32V		100
IO_ADC.h	85	
IO_ADC_RANGE_5V		100
IO_ADC_RANGE_5V		100
IO_ADC.h	85	
IO_ADC_RATIOMETRIC		100
IO_ADC_RATIOMETRIC		100
IO_ADC.h	85	
IO_ADC_ResetProtection		100
IO_ADC_ResetProtection		100
IO_ADC.h	93	
IO_ADC_RESISTIVE		100
IO_ADC_RESISTIVE		100
IO_ADC.h	85	
IO_ADC_SAFETY_CONF		100
IO_ADC_SAFETY_CONF		100
IO_ADC.h	85	
io_adc_safety_conf		100
io_adc_safety_conf_		22
adc_val_lower		22
adc_val_upper		22
redundant_channel		22
IO_ADC_SAFETY_SW_0		100
IO_ADC_SAFETY_SW_0		100
IO_Pin.h	232	
IO_ADC_SAFETY_SW_1		100
IO_ADC_SAFETY_SW_1		100
IO_Pin.h	233	
IO_ADC_SAFETY_SW_2		100
IO_ADC_SAFETY_SW_2		100
IO_Pin.h	233	
IO_ADC_SENSOR_SUPPLY_0		100
IO_ADC_SENSOR_SUPPLY_0		100
IO_Pin.h	233	
IO_ADC_SENSOR_SUPPLY_1		100
IO_ADC_SENSOR_SUPPLY_1		100
IO_Pin.h	233	
IO_ADC_SENSOR_SUPPLY_2		100
IO_ADC_SENSOR_SUPPLY_2		100
IO_Pin.h	233	
IO_ADC_UBAT		100
IO_ADC_UBAT		100
IO_Pin.h	233	
IO_ADC_WAKE_UP		100
IO_ADC_WAKE_UP		100
IO_Pin.h	233	
IO_CAN.h		95
IO_CAN.h	95	
IO_CAN_BIT_1000_KB		100
IO_CAN_BIT_100_KB		100
IO_CAN_BIT_125_KB		100
IO_CAN_BIT_250_KB		100
IO_CAN_BIT_500_KB		100
IO_CAN_BIT_50_KB		100
IO_CAN_BIT_USER		101
IO_CAN_ConfigFIFO		102
IO_CAN_ConfigMsg		103
IO_CAN_DATA_FRAME		101
IO_CAN_DelInit		104
IO_CAN_DelInitHandle		105
IO_CAN_EXT_FRAME		101
IO_CAN_FIFOStatus		105
IO_CAN_Init		105
IO_CAN_MSG_READ		101
IO_CAN_MSG_WRITE		101
IO_CAN_MsgStatus		107
IO_CAN_ReadFIFO		108
IO_CAN_ReadMsg		109
IO_CAN_Status		109
IO_CAN_STD_FRAME		101
IO_CAN_WriteMsg		110
IO_CAN_BIT_1000_KB		100
IO_CAN_BIT_1000_KB		100
IO_CAN.h	100	
IO_CAN_BIT_100_KB		100
IO_CAN_BIT_100_KB		100
IO_CAN.h	100	
IO_CAN_BIT_125_KB		100
IO_CAN_BIT_125_KB		100
IO_CAN.h	100	
IO_CAN_BIT_250_KB		100
IO_CAN_BIT_250_KB		100
IO_CAN.h	100	
IO_CAN_BIT_500_KB		100
IO_CAN_BIT_500_KB		100
IO_CAN.h	100	
IO_CAN_BIT_50_KB		100
IO_CAN_BIT_50_KB		100
IO_CAN.h	100	
IO_CAN_BIT_USER		101
IO_CAN_BIT_USER		101
IO_CAN.h	101	
IO_CAN_CHANNEL_0		100
IO_CAN_CHANNEL_0		100
IO_Pin.h	233	
IO_CAN_CHANNEL_1		100
IO_CAN_CHANNEL_1		100
IO_Pin.h	233	

IO_CAN_CHANNEL_2	IO_CAN_MSG_WRITE
IO_CAN_CHANNEL_2	IO_CAN.h.....
IO_Pin.h	101
IO_CAN_CHANNEL_3	IO_CAN_MsgStatus
IO_CAN_CHANNEL_3	IO_CAN_MsgStatus
IO_CAN.h.....	107
IO_CAN_CHANNEL_4	IO_CAN_ReadFIFO
IO_CAN_CHANNEL_4	IO_CAN_ReadFIFO
IO_CAN.h.....	108
IO_CAN_CHANNEL_5	IO_CAN_ReadMsg
IO_CAN_CHANNEL_5	IO_CAN_ReadMsg
IO_CAN.h.....	109
IO_CAN_CHANNEL_6	IO_CAN_Status
IO_CAN_CHANNEL_6	IO_CAN_Status
IO_CAN.h.....	109
IO_CAN_ConfigFIFO	IO_CAN_STD_FRAME
IO_CAN_ConfigFIFO	IO_CAN_STD_FRAME
IO_CAN.h.....	101
IO_CAN_ConfigMsg	IO_CAN_WriteMsg
IO_CAN_ConfigMsg	IO_CAN_WriteMsg
IO_CAN.h.....	110
IO_CAN_DATA_FRAME	IO_DEBUG.h
IO_CAN_DATA_FRAME	IO_DEBUG.h.....
IO_CAN.h.....	111
io_can_data_frame_	IO_DEBUG_GetTxStatus
io_can_data_frame_.....	114
data.....	IO_DEBUG_GetWatchdogState
id.....	114
id_format.....	IO_DEBUG_OUTPUT_PIN_0
length.....	113
IO_CAN_DelInit	IO_DEBUG_OUTPUT_PIN_1
IO_CAN_DelInit	113
IO_CAN.h.....	IO_DEBUG_OUTPUT_PIN_2
IO_CAN_DelInitHandle	113
IO_CAN_DelInitHandle	IO_DEBUG_SetOutputPin.....
IO_CAN.h.....	115
IO_CAN_EXT_FRAME	IO_DEBUG_StdioDelInit.....
IO_CAN_EXT_FRAME	116
IO_CAN.h.....	IO_DEBUG_StdioInit.....
IO_CAN_FIFOStatus	116
IO_CAN_FIFOStatus	IO_DEBUG_WD_NORMAL
IO_CAN.h.....	113
IO_CAN_Init	IO_DEBUG_WD_STATE_ACTIVE....
IO_CAN_Init	114
IO_CAN.h.....	IO_DEBUG_WD_STATE_PREPARED
IO_CAN_MSG_READ	114
IO_CAN_MSG_READ	IO_DEBUG_WD_STATE_SAFE.....
IO_CAN.h.....	114
IO_CAN_MSG_WRITE	IO_DEBUG_WD_STATE_UNKNOWN
	114
	IO_DEBUG_GetTxStatus
	IO_DEBUG_GetTxStatus
	IO_DEBUG.h
	114
	IO_DEBUG_GetWatchdogState
	IO_DEBUG_GetWatchdogState
	IO_DEBUG.h
	114
	IO_DEBUG_OUTPUT_PIN_0
	IO_DEBUG_OUTPUT_PIN_0
	IO_DEBUG.h
	113
	IO_DEBUG_OUTPUT_PIN_1
	IO_DEBUG_OUTPUT_PIN_1
	IO_DEBUG.h
	113
	IO_DEBUG_OUTPUT_PIN_2
	IO_DEBUG_OUTPUT_PIN_2
	IO_DEBUG_OUTPUT_PIN_2

IO_DEBUG.h	113	IO_DI_08	
IO_DEBUG_SetOutputPin		IO_DI_08	
IO_DEBUG_SetOutputPin		IO_Pin.h	235
IO_DEBUG.h	115	IO_DI_09	
IO_DEBUG_StdioDelInit		IO_DI_09	
IO_DEBUG_StdioDelInit		IO_Pin.h	235
IO_DEBUG.h	116	IO_DI_10	
IO_DEBUG_StdioInit		IO_DI_10	
IO_DEBUG_StdioInit		IO_Pin.h	235
IO_DEBUG.h	116	IO_DI_11	
IO_DEBUG_WD_NORMAL		IO_DI_11	
IO_DEBUG_WD_NORMAL		IO_Pin.h	235
IO_DEBUG.h	113	IO_DI_12	
IO_DEBUG_WD_STATE_ACTIVE		IO_DI_12	
IO_DEBUG_WD_STATE_ACTIVE		IO_Pin.h	236
IO_DEBUG.h	114	IO_DI_13	
IO_DEBUG_WD_STATE_PREPARED		IO_DI_13	
IO_DEBUG_WD_STATE_PREPARED		IO_Pin.h	236
IO_DEBUG.h	114	IO_DI_14	
IO_DEBUG_WD_STATE_SAFE		IO_DI_14	
IO_DEBUG_WD_STATE_SAFE		IO_Pin.h	236
IO_DEBUG.h	114	IO_DI_15	
IO_DEBUG_WD_STATE_UNKNOWN		IO_DI_15	
IO_DEBUG_WD_STATE_UNKNOWN		IO_Pin.h	236
IO_DEBUG.h	114	IO_DI_16	
IO_DI_00		IO_DI_16	
IO_DI_00		IO_Pin.h	236
IO_Pin.h	234	IO_DI_17	
IO_DI_01		IO_DI_17	
IO_DI_01		IO_Pin.h	236
IO_Pin.h	234	IO_DI_18	
IO_DI_02		IO_DI_18	
IO_DI_02		IO_Pin.h	236
IO_Pin.h	234	IO_DI_19	
IO_DI_03		IO_DI_19	
IO_DI_03		IO_Pin.h	236
IO_Pin.h	234	IO_DI_20	
IO_DI_04		IO_DI_20	
IO_DI_04		IO_Pin.h	237
IO_Pin.h	235	IO_DI_21	
IO_DI_05		IO_DI_21	
IO_DI_05		IO_Pin.h	237
IO_Pin.h	235	IO_DI_22	
IO_DI_06		IO_DI_22	
IO_DI_06		IO_Pin.h	237
IO_Pin.h	235	IO_DI_23	
IO_DI_07		IO_DI_23	
IO_DI_07		IO_Pin.h	237
IO_Pin.h	235	IO_DI_24	

IO_DI_24	IO_Pin.h	239
IO_Pin.h	237	
IO_DI_25	IO_Pin.h	237
IO_Pin.h	237	
IO_DI_26	IO_Pin.h	237
IO_Pin.h	237	
IO_DI_27	IO_Pin.h	237
IO_Pin.h	237	
IO_DI_28	IO_Pin.h	238
IO_Pin.h	238	
IO_DI_29	IO_Pin.h	238
IO_Pin.h	238	
IO_DI_30	IO_Pin.h	238
IO_Pin.h	238	
IO_DI_31	IO_Pin.h	238
IO_Pin.h	238	
IO_DI_32	IO_Pin.h	238
IO_Pin.h	238	
IO_DI_33	IO_Pin.h	238
IO_Pin.h	238	
IO_DI_34	IO_Pin.h	238
IO_Pin.h	238	
IO_DI_35	IO_Pin.h	238
IO_Pin.h	238	
IO_DI_36	IO_Pin.h	239
IO_Pin.h	239	
IO_DI_37	IO_Pin.h	239
IO_Pin.h	239	
IO_DI_38	IO_Pin.h	239
IO_Pin.h	239	
IO_DI_39	IO_Pin.h	239
IO_Pin.h	239	
IO_DI_40	IO_Pin.h	240
IO_Pin.h	240	
IO_DI_41	IO_Pin.h	239
IO_Pin.h	239	
IO_DI_42	IO_Pin.h	239
IO_Pin.h	239	
IO_DI_43	IO_Pin.h	239
IO_Pin.h	239	
IO_DI_44	IO_Pin.h	240
IO_Pin.h	240	
IO_DI_45	IO_Pin.h	240
IO_Pin.h	240	
IO_DI_46	IO_Pin.h	240
IO_Pin.h	240	
IO_DI_47	IO_Pin.h	240
IO_Pin.h	240	
IO_DI_48	IO_Pin.h	240
IO_Pin.h	240	
IO_DI_49	IO_Pin.h	240
IO_Pin.h	240	
IO_DI_50	IO_Pin.h	240
IO_Pin.h	240	
IO_DI_51	IO_Pin.h	240
IO_Pin.h	240	
IO_DI_52	IO_Pin.h	241
IO_Pin.h	241	
IO_DI_53	IO_Pin.h	241
IO_Pin.h	241	
IO_DI_54	IO_Pin.h	241
IO_Pin.h	241	
IO_DI_55	IO_Pin.h	241
IO_Pin.h	241	
IO_DI_56	IO_Pin.h	241
IO_Pin.h	241	

IO_DI_57	IO_DI_73
IO_DI_57	IO_Pin.h
IO_Pin.h	243
IO_DI_58	IO_DI_74
IO_DI_58	IO_Pin.h
IO_Pin.h	243
IO_DI_59	IO_DI_75
IO_DI_59	IO_Pin.h
IO_Pin.h	243
IO_DI_60	IO_DI_76
IO_DI_60	IO_Pin.h
IO_Pin.h	244
IO_DI_61	IO_DI_77
IO_DI_61	IO_Pin.h
IO_Pin.h	244
IO_DI_62	IO_DI_78
IO_DI_62	IO_Pin.h
IO_Pin.h	244
IO_DI_63	IO_DI_79
IO_DI_63	IO_Pin.h
IO_Pin.h	244
IO_DI_64	IO_DI_80
IO_DI_64	IO_Pin.h
IO_Pin.h	244
IO_DI_65	IO_DI_81
IO_DI_65	IO_Pin.h
IO_Pin.h	244
IO_DI_66	IO_DI_82
IO_DI_66	IO_Pin.h
IO_Pin.h	244
IO_DI_67	IO_DI_83
IO_DI_67	IO_Pin.h
IO_Pin.h	244
IO_DI_68	IO_DI_84
IO_DI_68	IO_Pin.h
IO_Pin.h	245
IO_DI_69	IO_DI_85
IO_DI_69	IO_Pin.h
IO_Pin.h	245
IO_DI_70	IO_DI_86
IO_DI_70	IO_Pin.h
IO_Pin.h	245
IO_DI_71	IO_DI_87
IO_DI_71	IO_Pin.h
IO_Pin.h	245
IO_DI_72	IO_DI_88
IO_DI_72	IO_Pin.h
IO_Pin.h	245
IO_DI_73	IO_DI_89
	IO_DI_89

IO_Pin.h	245
IO_DI_90	
IO_DI_90	
IO_Pin.h	245
IO_DI_91	
IO_DI_91	
IO_Pin.h	245
IO_DI_92	
IO_DI_92	
IO_Pin.h	246
IO_DI_93	
IO_DI_93	
IO_Pin.h	246
IO_DI_94	
IO_DI_94	
IO_Pin.h	246
IO_DI_95	
IO_DI_95	
IO_Pin.h	246
IO_DI_DelInit	
IO_DI_DelInit	
IO_DIO.h	121
IO_DI_Get	
IO_DI_Get	
IO_DIO.h	121
IO_DI_Init	
IO_DI_Init	
IO_DIO.h	122
IO_DI_NO_PULL	
IO_DI_NO_PULL	
IO_DIO.h	119
IO_DI_PD_10K	
IO_DI_PD_10K	
IO_DIO.h	119
IO_DI_PU_10K	
IO_DI_PU_10K	
IO_DIO.h	120
IO_DIO.h	
IO_DIO.h	116
IO_DI_DelInit.....	121
IO_DI_Get.....	121
IO_DI_Init.....	122
IO_DI_NO_PULL.....	119
IO_DI_PD_10K.....	119
IO_DI_PU_10K.....	120
IO_DIO_LIMITS.....	120
IO_DO_DelInit.....	124
IO_DO_GetCur.....	124
IO_DO_GetVoltage	125
IO_DO_Init.....	126
IO_DO_ResetProtection.....	127
IO_DO_SAFETY_CONF.....	120
IO_DO_Set.....	128
IO_DIO_LIMITS	
IO_DIO_LIMITS	
IO_DIO.h	120
io_dio_limits_	
io_dio_limits_	24
high_thresh1	24
high_thresh2	24
low_thresh1	24
low_thresh2	25
IO_DO_00	
IO_DO_00	
IO_Pin.h	246
IO_DO_01	
IO_DO_01	
IO_Pin.h	246
IO_DO_02	
IO_DO_02	
IO_Pin.h	246
IO_DO_03	
IO_DO_03	
IO_Pin.h	246
IO_DO_04	
IO_DO_04	
IO_Pin.h	247
IO_DO_05	
IO_DO_05	
IO_Pin.h	247
IO_DO_06	
IO_DO_06	
IO_Pin.h	247
IO_DO_07	
IO_DO_07	
IO_Pin.h	247
IO_DO_08	
IO_DO_08	
IO_Pin.h	247
IO_DO_09	
IO_DO_09	
IO_Pin.h	247
IO_DO_10	
IO_DO_10	
IO_Pin.h	247
IO_DO_11	
IO_DO_11	
IO_Pin.h	247

IO_DO_12		IO_DO_28	
IO_DO_12	IO_Pin.h	IO_Pin.h	250
IO_Pin.h	248		
IO_DO_13		IO_DO_29	
IO_DO_13	IO_Pin.h	IO_Pin.h	250
IO_Pin.h	248		
IO_DO_14		IO_DO_30	
IO_DO_14	IO_Pin.h	IO_Pin.h	250
IO_Pin.h	248		
IO_DO_15		IO_DO_31	
IO_DO_15	IO_Pin.h	IO_Pin.h	250
IO_Pin.h	248		
IO_DO_16		IO_DO_32	
IO_DO_16	IO_Pin.h	IO_Pin.h	250
IO_Pin.h	248		
IO_DO_17		IO_DO_33	
IO_DO_17	IO_Pin.h	IO_Pin.h	250
IO_Pin.h	248		
IO_DO_18		IO_DO_34	
IO_DO_18	IO_Pin.h	IO_Pin.h	250
IO_Pin.h	248		
IO_DO_19		IO_DO_35	
IO_DO_19	IO_Pin.h	IO_Pin.h	250
IO_Pin.h	248		
IO_DO_20		IO_DO_36	
IO_DO_20	IO_Pin.h	IO_Pin.h	251
IO_Pin.h	249		
IO_DO_21		IO_DO_37	
IO_DO_21	IO_Pin.h	IO_Pin.h	251
IO_Pin.h	249		
IO_DO_22		IO_DO_38	
IO_DO_22	IO_Pin.h	IO_Pin.h	251
IO_Pin.h	249		
IO_DO_23		IO_DO_39	
IO_DO_23	IO_Pin.h	IO_Pin.h	251
IO_Pin.h	249		
IO_DO_24		IO_DO_40	
IO_DO_24	IO_Pin.h	IO_Pin.h	251
IO_Pin.h	249		
IO_DO_25		IO_DO_41	
IO_DO_25	IO_Pin.h	IO_Pin.h	251
IO_Pin.h	249		
IO_DO_26		IO_DO_42	
IO_DO_26	IO_Pin.h	IO_Pin.h	251
IO_Pin.h	249		
IO_DO_27		IO_DO_43	
IO_DO_27	IO_Pin.h	IO_Pin.h	251
IO_Pin.h	249		
IO_DO_28		IO_DO_44	
		IO_DO_44	

IO_Pin.h	252	IO_DO_GetCur	
IO_DO_45		IO_DO_GetCur	
IO_DO_45		IO_DIO.h	124
IO_Pin.h	252	IO_DO_GetVoltage	
IO_DO_46		IO_DO_GetVoltage	
IO_DO_46		IO_DIO.h	125
IO_Pin.h	252	IO_DO_Init	
IO_DO_47		IO_DO_Init	
IO_DO_47		IO_DIO.h	126
IO_Pin.h	252	IO_DO_ResetProtection	
IO_DO_48		IO_DO_ResetProtection	
IO_DO_48		IO_DIO.h	127
IO_Pin.h	252	IO_DO_SAFETY_CONF	
IO_DO_49		IO_DO_SAFETY_CONF	
IO_DO_49		IO_DIO.h	120
IO_Pin.h	252	io_do_safety_conf_	
IO_DO_50		io_do_safety_conf_	25
IO_DO_50		low_side_channel	25
IO_Pin.h	252	IO_DO_Set	
IO_DO_51		IO_DO_Set	
IO_DO_51		IO_DIO.h	128
IO_Pin.h	252	IO_DOWNLOAD.h	
IO_DO_52		IO_DOWNLOAD.h	130
IO_DO_52		IO_DOWNLOAD_CheckRequest	132
IO_Pin.h	253	IO_DOWNLOAD_Init	132
IO_DO_53		IO_DOWNLOAD_Launch	132
IO_DO_53		IO_DOWNLOAD_CheckRequest	
IO_Pin.h	253	IO_DOWNLOAD_CheckRequest	
IO_DO_54		IO_DOWNLOAD.h	132
IO_DO_54		IO_DOWNLOAD_Init	
IO_Pin.h	253	IO_DOWNLOAD_Init	
IO_DO_55		IO_DOWNLOAD.h	132
IO_DO_55		IO_DOWNLOAD_Launch	
IO_Pin.h	253	IO_DOWNLOAD_Launch	
IO_DO_56		IO_DOWNLOAD.h	132
IO_DO_56		IO_Driver.h	
IO_Pin.h	253	IO_Driver.h	133
IO_DO_57		IO_DRIVER_ECU_MAC_ADD_LENGTH	141
IO_DO_57		IO_DRIVER_ECU_PROD_CODE_LENGTH	
IO_Pin.h	253	142	
IO_DO_58		IO_DRIVER_ECU_SERIAL_LENGTH	142
IO_DO_58		IO_DRIVER_FPU_HANDLER	145
IO_Pin.h	253	IO_Driver_GetMacAddress	146
IO_DO_59		IO_Driver_GetProdCode	146
IO_DO_59		IO_Driver_GetSerialNumber	147
IO_Pin.h	253	IO_Driver_GetVersionOfBootloader ..	148
IO_DO_Delnit		IO_Driver_GetVersionOfDriver	149
IO_DO_Delnit		IO_Driver_GetVersionOfFPGA	149
IO_DIO.h	124	IO_Driver_Init	149

IO_Driver_Reset	150	IO_Driver.h.....	147
IO_DRIVER_SAFETY_CONF	145	IO_Driver_GetVersionOfBootloader	
IO_Driver_SetFPUHandler	151	IO_Driver_GetVersionOfBootloader	
IO_Driver_SetIntegerDivisionByZeroException	151	IO_Driver.h.....	148
IO_Driver_TaskBegin.....	152	IO_Driver_GetVersionOfDriver	
IO_Driver_TaskEnd	152	IO_Driver_GetVersionOfDriver	
SAFETY_CONF_RESETS_1.....	142	IO_Driver.h.....	149
SAFETY_CONF_RESETS_2.....	142	IO_Driver_GetVersionOfFPGA	
SAFETY_CONF_RESETS_3.....	142	IO_Driver_GetVersionOfFPGA	
SAFETY_CONF_RESETS_4.....	142	IO_Driver.h.....	149
SAFETY_CONF_RESETS_5.....	142	IO_Driver_Init	
SAFETY_CONF_RESETS_6.....	143	IO_Driver_Init	
SAFETY_CONF_RESETS_7.....	143	IO_Driver.h.....	149
SAFETY_CONF_RESETS_8.....	143	IO_Driver_Reset	
SAFETY_CONF_RESETS_9.....	143	IO_Driver_Reset	
SAFETY_CONF_RESETS_DISABLED	143	IO_Driver.h.....	150
SAFETY_CONF_WINDOW_SIZE_100_PERCENT	143	IO_DRIVER_SAFETY_CONF	
SAFETY_CONF_WINDOW_SIZE_12_5_PERCENT	144	IO_DRIVER_SAFETY_CONF	
SAFETY_CONF_WINDOW_SIZE_25_PERCENT	144	IO_Driver.h.....	145
SAFETY_CONF_WINDOW_SIZE_3_125_PERCENT	144	io_driver_safety_conf_	
SAFETY_CONF_WINDOW_SIZE_50_PERCENT	144	io_driver_safety_conf_.....	26
SAFETY_CONF_WINDOW_SIZE_6_25_PERCENT	144	command_period.....	27
IO_DRIVER_ECU_MAC_ADD_LENGTH		error_callback.....	27
IO_DRIVER_ECU_MAC_ADD_LENGTH		glitch_filter_time.....	27
IO_Driver.h.....	141	notify_callback.....	27
IO_DRIVER_ECU_PROD_CODE_LENGTH		reset_behavior.....	27
IO_DRIVER_ECU_PROD_CODE_LENGTH		window_size.....	27
IO_Driver.h.....	142	IO_Driver_SetFPUHandler	
IO_DRIVER_ECU_SERIAL_LENGTH		IO_Driver_SetFPUHandler	
IO_DRIVER_ECU_SERIAL_LENGTH		IO_Driver.h.....	151
IO_Driver.h.....	142	IO_Driver_SetIntegerDivisionByZeroException	
IO_DRIVER_FPU_HANDLER		IO_Driver_SetIntegerDivisionByZeroException	
IO_DRIVER_FPU_HANDLER		IO_Driver.h.....	151
IO_Driver.h.....	145	IO_Driver_TaskBegin	
IO_Driver_GetMacAddress		IO_Driver_TaskBegin	
IO_Driver_GetMacAddress		IO_Driver.h.....	152
IO_Driver.h.....	146	IO_Driver_TaskEnd	
IO_Driver_GetProdCode		IO_Driver_TaskEnd	
IO_Driver_GetProdCode		IO_Driver.h.....	152
IO_Driver.h.....	146	IO_E_BUSY	
IO_Driver_GetSerialNumber		IO_E_BUSY	
IO_Driver_GetSerialNumber		IO_Error.h.....	163

IO_E_CAN_ERROR_WARNING	IO_Error.h.....	166
IO_Error.h.....		163
IO_E_CAN_FIFO_FULL	IO_E_CAN_FIFO_FULL	
IO_Error.h.....		163
IO_E_CAN_MAX_HANDLES_REACHED	IO_E_CAN_MAX_HANDLES_REACHED	
IO_Error.h.....		163
IO_E_CAN_MAX_MO_REACHED	IO_E_CAN_MAX_MO_REACHED	
IO_Error.h.....		163
IO_E_CAN_OLD_DATA	IO_E_CAN_OLD_DATA	
IO_Error.h.....		164
IO_E_CAN_OVERFLOW	IO_E_CAN_OVERFLOW	
IO_Error.h.....		164
IO_E_CAN_TIMEOUT	IO_E_CAN_TIMEOUT	
IO_Error.h.....		164
IO_E_CAN_WRONG_HANDLE	IO_E_CAN_WRONG_HANDLE	
IO_Error.h.....		164
IO_E_CH_CAPABILITY	IO_E_CH_CAPABILITY	
IO_Error.h.....		164
IO_E_CHANNEL_BUSY	IO_E_CHANNEL_BUSY	
IO_Error.h.....		165
IO_E_CHANNEL_NOT_CONFIGURED	IO_E_CHANNEL_NOT_CONFIGURED	
IO_Error.h.....		165
IO_E_CM_CALIBRATION	IO_E_CM_CALIBRATION	
IO_Error.h.....		165
IO_E_CORE_TEST_FAILED	IO_E_CORE_TEST_FAILED	
IO_Error.h.....		165
IO_E_DOWNLOAD_HANDSHAKE	IO_E_DOWNLOAD_HANDSHAKE	
IO_Error.h.....		165
IO_E_DOWNLOAD_NO_REQ	IO_E_DOWNLOAD_NO_REQ	
IO_Error.h.....		165
IO_E_DOWNLOAD_TIMEOUT	IO_E_DOWNLOAD_TIMEOUT	
IO_Error.h.....		165
IO_E_DRIVER_NOT_INITIALIZED	IO_E_DRIVER_NOT_INITIALIZED	
IO_Error.h.....		165
IO_E_DRV_SAFETY_CONF_NOT_CONFIG	IO_E_DRV_SAFETY_CONF_NOT_CONFIG	
IO_Error.h.....		166
IO_E_DRV_SAFETY_CYCLE_RUNNING	IO_E_DRV_SAFETY_CYCLE_RUNNING	
IO_Error.h.....		166
IO_E EEPROM_RANGE	IO_E EEPROM_RANGE	
IO_Error.h.....		166
IO_E EEPROM_SPI	IO_E EEPROM_SPI	
IO_Error.h.....		166
IO_E_ERROR_PIN_TEST_FAILED	IO_E_ERROR_PIN_TEST_FAILED	
IO_Error.h.....		167
IO_E_ERROR_PIN_TEST_TIMEOUT	IO_E_ERROR_PIN_TEST_TIMEOUT	
IO_Error.h.....		167
IO_E_ETH_DEINIT_TIMEOUT	IO_E_ETH_DEINIT_TIMEOUT	
IO_Error.h.....		167
IO_E_ETH_INIT_FAIL	IO_E_ETH_INIT_FAIL	
IO_Error.h.....		167
IO_E_ETH_INIT_TIMEOUT	IO_E_ETH_INIT_TIMEOUT	
IO_Error.h.....		167
IO_E_ETH_MAC_INVALID	IO_E_ETH_MAC_INVALID	
IO_Error.h.....		167
IO_E_ETH_MDIO_READ	IO_E_ETH_MDIO_READ	
IO_Error.h.....		167
IO_E_ETH_MDIO_TIMEOUT	IO_E_ETH_MDIO_TIMEOUT	
IO_Error.h.....		167
IO_E_ETH_NO_LINK	IO_E_ETH_NO_LINK	
IO_Error.h.....		168
IO_E_ETH_READ_FAIL	IO_E_ETH_READ_FAIL	
IO_Error.h.....		168
IO_E_ETH_WRITE_FAIL	IO_E_ETH_WRITE_FAIL	
IO_Error.h.....		168
IO_E_FET_PROT_ACTIVE	IO_E_FET_PROT_ACTIVE	
IO_Error.h.....		168

IO_E_FET_PROT_NOT_ACTIVE		IO_E_INVALID_CHANNEL_ID	
IO_E_FET_PROT_NOT_ACTIVE		IO_Error.h.....	171
IO_Error.h.....	168		
IO_E_FET_PROT_PERMANENT		IO_E_INVALID_ESM_INIT_STATUS	
IO_E_FET_PROT_PERMANENT		IO_E_INVALID_ESM_INIT_STATUS	
IO_Error.h.....	168	IO_Error.h.....	171
IO_E_FET_PROT_REENABLE		IO_E_INVALID_LIMITS	
IO_E_FET_PROT_REENABLE		IO_E_INVALID_LIMITS	
IO_Error.h.....	169	IO_Error.h.....	171
IO_E_FET_PROT_WAIT		IO_E_INVALID_OPERATION	
IO_E_FET_PROT_WAIT		IO_E_INVALID_OPERATION	
IO_Error.h.....	169	IO_Error.h.....	171
IO_E_FLASH_BLANK_CHECK_FAILED		IO_E_INVALID_PARAMETER	
IO_E_FLASH_BLANK_CHECK_FAILED		IO_E_INVALID_PARAMETER	
IO_Error.h.....	169	IO_Error.h.....	172
IO_E_FLASH_OP_FAILED		IO_E_INVALID_PIN_CONFIG	
IO_E_FLASH_OP_FAILED		IO_E_INVALID_PIN_CONFIG	
IO_Error.h.....	169	IO_Error.h.....	172
IO_E_FLASH_OP_TIMEOUT		IO_E_INVALID_PROD_DATA	
IO_E_FLASH_OP_TIMEOUT		IO_E_INVALID_PROD_DATA	
IO_Error.h.....	169	IO_Error.h.....	172
IO_E_FLASH_SUSPENDED		IO_E_INVALID_SAFETY_CONFIG	
IO_E_FLASH_SUSPENDED		IO_E_INVALID_SAFETY_CONFIG	
IO_Error.h.....	169	IO_Error.h.....	172
IO_E_FLASH_WRONG_DEVICE_ID		IO_E_INVALID_SERIAL_NUMBER	
IO_E_FLASH_WRONG_DEVICE_ID		IO_E_INVALID_SERIAL_NUMBER	
IO_Error.h.....	170	IO_Error.h.....	172
IO_E_FPGA_CRC_ERROR		IO_E_INVALID_VARIANT	
IO_E_FPGA_CRC_ERROR		IO_E_INVALID_VARIANT	
IO_Error.h.....	170	IO_Error.h.....	172
IO_E_FPGA_IMAGE		IO_E_INVALID_VOLTAGE	
IO_E_FPGA_IMAGE		IO_E_INVALID_VOLTAGE	
IO_Error.h.....	170	IO_Error.h.....	172
IO_E_FPGA_NOT_INITIALIZED		IO_E_LIN_BIT	
IO_E_FPGA_NOT_INITIALIZED		IO_E_LIN_BIT	
IO_Error.h.....	170	IO_Error.h.....	173
IO_E_FPGA_TIMEOUT		IO_E_LIN_CHECKSUM	
IO_E_FPGA_TIMEOUT		IO_E_LIN_CHECKSUM	
IO_Error.h.....	170	IO_Error.h.....	173
IO_E_FPGA_VERSION		IO_E_LIN_FRAMING	
IO_E_FPGA_VERSION		IO_E_LIN_FRAMING	
IO_Error.h.....	170	IO_Error.h.....	173
IO_E_INTERNAL_CSM		IO_E_LIN_INCONSISTENT_SYNCH_FIELD	
IO_E_INTERNAL_CSM		IO_E_LIN_INCONSISTENT_SYNCH_FIELD	
IO_Error.h.....	171	IO_Error.h.....	173
IO_E_INTERNAL_MEM FAILED		IO_E_LIN_NO_RESPONSE	
IO_E_INTERNAL_MEM FAILED		IO_E_LIN_NO_RESPONSE	
IO_Error.h.....	171	IO_Error.h.....	173
IO_E_INVALID_CHANNEL_ID		IO_E_LIN_OVERRUN	
		IO_E_LIN_OVERRUN	

IO_Error.h.....	173
IO_E_LIN_PARITY	
IO_E_LIN_PARITY	
IO_Error.h.....	173
IO_E_LIN_PHYSICAL_BUS	
IO_E_LIN_PHYSICAL_BUS	
IO_Error.h.....	174
IO_E_LIN_TIMEOUT	
IO_E_LIN_TIMEOUT	
IO_Error.h.....	174
IO_E_MPU_REGION_DISABLED	
IO_E_MPU_REGION_DISABLED	
IO_Error.h.....	174
IO_E_MPU_REGION_ENABLED	
IO_E_MPU_REGION_ENABLED	
IO_Error.h.....	174
IO_E_NO_DIAG	
IO_E_NO_DIAG	
IO_Error.h.....	174
IO_E_NULL_POINTER	
IO_E_NULL_POINTER	
IO_Error.h.....	175
IO_E_OK	
IO_E_OK	
IO_Error.h.....	175
IO_E_OPEN_LOAD	
IO_E_OPEN_LOAD	
IO_Error.h.....	175
IO_E_OPEN_LOAD_OR_SHORT_BAT	
IO_E_OPEN_LOAD_OR_SHORT_BAT	
IO_Error.h.....	175
IO_E_OPTION_NOT_SUPPORTED	
IO_E_OPTION_NOT_SUPPORTED	
IO_Error.h.....	176
IO_E_PERIODIC_NOT_CONFIGURED	
IO_E_PERIODIC_NOT_CONFIGURED	
IO_Error.h.....	176
IO_E_PWD_CURRENT_THRESH_HIGH	
IO_E_PWD_CURRENT_THRESH_HIGH	
IO_Error.h.....	176
IO_E_PWD_CURRENT_THRESH_LOW	
IO_E_PWD_CURRENT_THRESH_LOW	
IO_Error.h.....	176
IO_E_PWD_NOT_FINISHED	
IO_E_PWD_NOT_FINISHED	
IO_Error.h.....	176
IO_E_PWD_OVERFLOW	
IO_E_PWD_OVERFLOW	
IO_Error.h.....	176
IO_E_PWM_NO_DIAG	
IO_E_PWM_NO_DIAG	
IO_Error.h.....	176
IO_E_REFERENCE	
IO_E_REFERENCE	
IO_Error.h.....	177
IO_E_RESOLVING	
IO_E_RESOLVING	
IO_Error.h.....	177
IO_E_RESOLVING_FAILED	
IO_E_RESOLVING_FAILED	
IO_Error.h.....	177
IO_E_RTC_CLOCK_INTEGRITY_FAILED	
IO_E_RTC_CLOCK_INTEGRITY_FAILED	
IO_Error.h.....	177
IO_E_SAFE_STATE	
IO_E_SAFE_STATE	
IO_Error.h.....	177
IO_E_SAFETY_SWITCH_DISABLED	
IO_E_SAFETY_SWITCH_DISABLED	
IO_Error.h.....	177
IO_E_SHM_INTEGRITY	
IO_E_SHM_INTEGRITY	
IO_Error.h.....	178
IO_E_SHORT_BAT	
IO_E_SHORT_BAT	
IO_Error.h.....	178
IO_E_SHORT_GND	
IO_E_SHORT_GND	
IO_Error.h.....	178
IO_E_SOCKET_NOT_INITIALIZED	
IO_E_SOCKET_NOT_INITIALIZED	
IO_Error.h.....	179
IO_E_STARTUP	
IO_E_STARTUP	
IO_Error.h.....	179
IO_E_UART_BUFFER_EMPTY	
IO_E_UART_BUFFER_EMPTY	
IO_Error.h.....	180
IO_E_UART_BUFFER_FULL	
IO_E_UART_BUFFER_FULL	
IO_Error.h.....	180
IO_E_UART_DMA	
IO_E_UART_DMA	
IO_Error.h.....	180
IO_E_UART_FRAMING	
IO_E_UART_FRAMING	
IO_Error.h.....	180
IO_E_UART_OVERFLOW	

IO_E_UART_OVERFLOW	IO_Error.h.....	183
IO_Error.h.....		180
IO_E_UART_PARITY	IO_E_UART_PARITY	
IO_Error.h.....		180
IO_E_UDP_ARP_RECEIVED	IO_E_UDP_ARP_RECEIVED	
IO_Error.h.....		181
IO_E_UDP_INVALID_BUFFER	IO_E_UDP_INVALID_BUFFER	
IO_Error.h.....		181
IO_E_UDP_NOMORESOCKETS	IO_E_UDP_NOMORESOCKETS	
IO_Error.h.....		181
IO_E_UDP_OVERFLOW	IO_E_UDP_OVERFLOW	
IO_Error.h.....		181
IO_E_UDP_WRONG_HANDLE	IO_E_UDP_WRONG_HANDLE	
IO_Error.h.....		181
IO_E_UDP_WRONG_PORT	IO_E_UDP_WRONG_PORT	
IO_Error.h.....		181
IO_E_UNKNOWN	IO_E_UNKNOWN	
IO_Error.h.....		181
IO_E_WD_ACTIVATION	IO_E_WD_ACTIVATION	
IO_Error.h.....		181
IO_E_WD_DEBUGGING_PREPARED	IO_E_WD_DEBUGGING_PREPARED	
IO_Error.h.....		182
IO_E_WD_INITIALIZATION	IO_E_WD_INITIALIZATION	
IO_Error.h.....		182
IO_E_WD_PRECISION	IO_E_WD_PRECISION	
IO_Error.h.....		182
IO_E_WD_RANGE	IO_E_WD_RANGE	
IO_Error.h.....		182
IO_E_WD_SAFE_LOCK	IO_E_WD_SAFE_LOCK	
IO_Error.h.....		182
IO_E_WD_SELF_MONITORING	IO_E_WD_SELF_MONITORING	
IO_Error.h.....		182
IO_E_WD_STATUS_INVALID	IO_E_WD_STATUS_INVALID	
IO_Error.h.....		183
IO_E_WD_TRIGGER	IO_E_WD_TRIGGER	
IO_Error.h.....		183
IO_E_WD_VICE_VERSA_MONITORING	IO_E_WD_VICE_VERSA_MONITORING	
IO_Error.h.....		183
IO_E_WRONG_ADDRESS	IO_E_WRONG_ADDRESS	
IO_Error.h.....		183
IO_EEPROM.h	IO_EEPROM.h.....	153
IO_EEPROM_DelInit	IO_EEPROM_DelInit.....	157
IO_EEPROM_GetStatus	IO_EEPROM_GetStatus.....	157
IO_EEPROM_Init	IO_EEPROM_Init.....	157
IO_EEPROM_Process	IO_EEPROM_Process.....	156
IO_EEPROM_Read	IO_EEPROM_Read.....	158
IO_EEPROM_Write	IO_EEPROM_Write.....	159
IO_EEPROM_DelInit	IO_EEPROM_DelInit	
IO_EEPROM_h.....		157
IO_EEPROM_GetStatus	IO_EEPROM_GetStatus	
IO_EEPROM_h.....		157
IO_EEPROM_Init	IO_EEPROM_Init	
IO_EEPROM_h.....		157
IO_EEPROM_Process	IO_EEPROM_Process	
IO_EEPROM_h.....		156
IO_EEPROM_Read	IO_EEPROM_Read	
IO_EEPROM_h.....		158
IO_EEPROM_Write	IO_EEPROM_Write	
IO_EEPROM_h.....		159
IO_Error.h	IO_Error.h.....	159
IO_E_BUSY	IO_E_BUSY.....	163
IO_E_CAN_BUS_OFF	IO_E_CAN_BUS_OFF.....	163
IO_E_CAN_ERROR_PASSIVE	IO_E_CAN_ERROR_PASSIVE.....	163
IO_E_CAN_ERROR_WARNING	IO_E_CAN_ERROR_WARNING.....	163
IO_E_CAN_FIFO_FULL	IO_E_CAN_FIFO_FULL.....	163
IO_E_CAN_MAX_HANDLES_REACHED	IO_E_CAN_MAX_HANDLES_REACHED.....	163
IO_E_CAN_MAX_MO_REACHED	IO_E_CAN_MAX_MO_REACHED ..	163
IO_E_CAN_OLD_DATA	IO_E_CAN_OLD_DATA.....	164
IO_E_CAN_OVERFLOW	IO_E_CAN_OVERFLOW.....	164
IO_E_CAN_TIMEOUT	IO_E_CAN_TIMEOUT.....	164
IO_E_CAN_WRONG_HANDLE	IO_E_CAN_WRONG_HANDLE ..	164

IO_E_CH_CAPABILITY	164
IO_E_CHANNEL_BUSY	165
IO_E_CHANNEL_NOT_CONFIGURED	165
IO_E_CM_CALIBRATION.....	165
IO_E_CORE_TEST_FAILED.....	165
IO_E_DOWNLOAD_HANDSHAKE...	165
IO_E_DOWNLOAD_NO_REQ.....	165
IO_E_DOWNLOAD_TIMEOUT.....	165
IO_E_DRIVER_NOT_INITIALIZED...	166
IO_E_DRV_SAFETY_CONF_NOT_CONFIG	166
IO_E_DRV_SAFETY_CYCLE_RUNNING	166
IO_E EEPROM_RANGE	166
IO_E EEPROM_SPI.....	166
IO_E_ERROR_PIN_TEST_FAILED ..	167
IO_E_ERROR_PIN_TEST_TIMEOUT	167
IO_E_ETH_DEINIT_TIMEOUT.....	167
IO_E_ETH_INIT_FAIL.....	167
IO_E_ETH_INIT_TIMEOUT.....	167
IO_E_ETH_MAC_INVALID.....	167
IO_E_ETH_MDIO_READ	167
IO_E_ETH_MDIO_TIMEOUT	167
IO_E_ETH_NO_LINK.....	168
IO_E_ETH_READ_FAIL.....	168
IO_E_ETH_WRITE_FAIL.....	168
IO_E_FET_PROT_ACTIVE	168
IO_E_FET_PROT_NOT_ACTIVE ...	168
IO_E_FET_PROT_PERMANENT ...	168
IO_E_FET_PROT_REENABLE	169
IO_E_FET_PROT_WAIT	169
IO_E_FLASH_BLANK_CHECK_FAILED	169
IO_E_FLASH_OP_FAILED.....	169
IO_E_FLASH_OP_TIMEOUT	169
IO_E_FLASH_SUSPENDED	169
IO_E_FLASH_WRONG_DEVICE_ID.	170
IO_E_FPGA_CRC_ERROR.....	170
IO_E_FPGA_IMAGE	170
IO_E_FPGA_NOT_INITIALIZED	170
IO_E_FPGA_TIMEOUT	170
IO_E_FPGA_VERSION	170
IO_E_INTERNAL_CSM	171
IO_E_INTERNAL_MEM_FAILED....	171
IO_E_INVALID_CHANNEL_ID	171
IO_E_INVALID_ESM_INIT_STATUS.	171
IO_E_INVALID_LIMITS	171
IO_E_INVALID_OPERATION.....	171
IO_E_INVALID_PARAMETER.....	172
IO_E_INVALID_PIN_CONFIG.....	172
IO_E_INVALID_PROD_DATA	172
IO_E_INVALID_SAFETY_CONFIG...	172
IO_E_INVALID_SERIAL_NUMBER ..	172
IO_E_INVALID_VARIANT	172
IO_E_INVALID_VOLTAGE.....	172
IO_E LIN_BIT	173
IO_E LIN_CHECKSUM.....	173
IO_E LIN_FRAMING	173
IO_E LIN_INCONSISTENT_SYNCH_FIELD	173
IO_E LIN_NO_RESPONSE	173
IO_E LIN_OVERRUN.....	173
IO_E LIN_PARITY	173
IO_E LIN_PHYSICAL_BUS.....	174
IO_E LIN_TIMEOUT.....	174
IO_E MPU_REGION_DISABLED ..	174
IO_E MPU_REGION_ENABLED ..	174
IO_E_NO_DIAG	174
IO_E_NULL_POINTER	175
IO_E_OK	175
IO_E_OPEN_LOAD	175
IO_E_OPEN_LOAD_OR_SHORT_BAT	175
IO_E_OPTION_NOT_SUPPORTED..	176
IO_E_PERIODIC_NOT_CONFIGURED	176
IO_E_PWD_CURRENT_THRESH_HIGH	176
IO_E_PWD_CURRENT_THRESH_LOW	176
IO_E_PWD_NOT_FINISHED	176
IO_E_PWD_OVERFLOW	176
IO_E_PWM_NO_DIAG	176
IO_E_REFERENCE	177
IO_E_RESOLVING	177
IO_E_RESOLVING_FAILED	177
IO_E_RTC_CLOCK_INTEGRITY_FAILED	177
IO_E_SAFE_STATE	177
IO_E_SAFETY_SWITCH_DISABLED	177
IO_E_SHM_INTEGRITY	178
IO_E_SHORT_BAT	178
IO_E_SHORT_GND	178
IO_E_SOCKET_NOT_INITIALIZED ..	179
IO_E_STARTUP	179
IO_E_UART_BUFFER_EMPTY	180
IO_E_UART_BUFFER_FULL	180
IO_E_UART_DMA	180
IO_E_UART_FRAMING	180
IO_E_UART_OVERFLOW	180
IO_E_UART_PARITY	180
IO_E_UDP_ARP RECEIVED	181
IO_E_UDP_INVALID_BUFFER.....	181
IO_E_UDP_NOMORESOCKETS	181

IO_E_UDP_OVERFLOW.....	181	IO_FLASH_ChipErase IO_FLASH.h	190
IO_E_UDP_WRONG_HANDLE	181	IO_FLASH_DelInit IO_FLASH_DelInit IO_FLASH.h	191
IO_E_UDP_WRONG_PORT	181	IO_FLASH_GetBank IO_FLASH_GetBank IO_FLASH.h	191
IO_E_UNKNOWN.....	181	IO_FLASH_GetStatus IO_FLASH_GetStatus IO_FLASH.h	192
IO_E_WD_ACTIVATION	181	IO_FLASH_Init IO_FLASH_Init IO_FLASH.h	192
IO_E_WD_DEBUGGING_PREPARED	182	IO_FLASH_Read IO_FLASH_Read IO_FLASH.h	193
IO_E_WD_INITIALIZATION	182	IO_FLASH_SetBusy IO_FLASH_SetBusy IO_FLASH.h	189
IO_E_WD_PRECISION.....	182	IO_FLASH_Suspend IO_FLASH_Suspend IO_FLASH.h	194
IO_E_WD_RANGE	182	IO_FLASH_Write IO_FLASH_Write IO_FLASH.h	195
IO_E_WD_SAFE_LOCK	182	IO_GND IO_GND IO_Pin.h	254
IO_E_WD_SELF_MONITORING	182	IO_INT_PIN_1V2 IO_INT_PIN_1V2 IO_Pin.h	254
IO_E_WD_STATUS_INVALID	183	IO_INT_PIN_CAN_CH0 IO_INT_PIN_CAN_CH0 IO_Pin.h	254
IO_E_WD_TRIGGER	183	IO_INT_PIN_CAN_CH1 IO_INT_PIN_CAN_CH1 IO_Pin.h	254
IO_E_WD_VICE_VERSA_MONITORING	183	IO_INT_PIN_CAN_CH2 IO_INT_PIN_CAN_CH2 IO_Pin.h	254
IO_E_WRONG_ADDRESS	183	IO_INT_PIN_CAN_CH3 IO_INT_PIN_CAN_CH3 IO_Pin.h	254
IO_ErrorType.....	183	IO_INT_PIN_CAN_CH4 IO_INT_PIN_CAN_CH4 IO_Pin.h	254
IO_ErrorType		IO_INT_PIN_CAN_CH5 IO_INT_PIN_CAN_CH5 IO_INT_PIN_CAN_CH5	
IO_ErrorType			
IO_Error.h.....	183		
IO_ETH_SPEED_100_MB			
IO_ETH_SPEED_100_MB			
IO_UDP.h.....	349		
IO_ETH_SPEED_10_MB			
IO_ETH_SPEED_10_MB			
IO_UDP.h.....	349		
IO_FLASH.h			
IO_FLASH.h.....	183		
IO_FLASH_BankSelect.....	189		
IO_FLASH_BlockErase.....	189		
IO_FLASH_BYT_SIZE.....	189		
IO_FLASH_ChipErase.....	190		
IO_FLASH_DelInit.....	191		
IO_FLASH_GetBank.....	191		
IO_FLASH_GetStatus.....	192		
IO_FLASH_Init.....	192		
IO_FLASH_Read.....	193		
IO_FLASH_SetBusy.....	189		
IO_FLASH_Suspend.....	194		
IO_FLASH_Write.....	195		
IO_FLASH_BankSelect			
IO_FLASH_BankSelect			
IO_FLASH.h.....	189		
IO_FLASH_BlockErase			
IO_FLASH_BlockErase			
IO_FLASH.h.....	189		
IO_FLASH_BYT_SIZE			
IO_FLASH_BYT_SIZE			
IO_FLASH.h.....	189		
IO_FLASH_ChipErase			

IO_Pin.h	254	IO_LIN_BAUDRATE_MIN	198
IO_INT_PIN_CAN_CH6		IO_LIN_CLASSIC	198
IO_INT_PIN_CAN_CH6		IO_LIN_DATA_FRAME	199
IO_Pin.h	255	IO_LIN_Delnit	199
IO_INT_PIN_ENDRV_CPU		IO_LIN_ENHANCED	198
IO_INT_PIN_ENDRV_CPU		IO_LIN_GetStatus	199
IO_Pin.h	255	IO_LIN_Init	200
IO_INT_PIN_PWD		IO_LIN_Read	200
IO_INT_PIN_PWD		IO_LIN_Write	201
IO_Pin.h	255	IO_LIN_BAUDRATE_MAX	
IO_INT_PIN_REF_2V5		IO_LIN_BAUDRATE_MAX	
IO_INT_PIN_REF_2V5		IO_LIN.h	198
IO_Pin.h	255	IO_LIN_BAUDRATE_MIN	
IO_INT_PIN_SAFETY_SW_0		IO_LIN_BAUDRATE_MIN	
IO_INT_PIN_SAFETY_SW_0		IO_LIN.h	198
IO_Pin.h	255	IO_LIN_CLASSIC	
IO_INT_PIN_SAFETY_SW_1		IO_LIN_CLASSIC	
IO_INT_PIN_SAFETY_SW_1		IO_LIN.h	198
IO_Pin.h	255	IO_LIN_DATA_FRAME	
IO_INT_PIN_SAFETY_SW_2		IO_LIN_DATA_FRAME	
IO_INT_PIN_SAFETY_SW_2		IO_LIN.h	199
IO_Pin.h	255	io_lin_data_frame_	
IO_INT_PIN_SAFETY_SW_VP		io_lin_data_frame_	28
IO_INT_PIN_SAFETY_SW_VP		data	28
IO_Pin.h	255	id	28
IO_INT_PIN_TEMP		length	28
IO_INT_PIN_TEMP		IO_LIN_Delnit	
IO_Pin.h	256	IO_LIN_Delnit	
IO_INT_PIN_VMON		IO_LIN.h	199
IO_INT_PIN_VMON		IO_LIN_ENHANCED	
IO_Pin.h	256	IO_LIN_ENHANCED	
IO_INT_POWERSTAGE_ENABLE		IO_LIN.h	198
IO_INT_POWERSTAGE_ENABLE		IO_LIN_GetStatus	
IO_Pin.h	256	IO_LIN_GetStatus	
IO_INT_SAFETY_SW_0		IO_LIN.h	199
IO_INT_SAFETY_SW_0		IO_LIN_Init	
IO_Pin.h	256	IO_LIN_Init	
IO_INT_SAFETY_SW_1		IO_LIN.h	200
IO_INT_SAFETY_SW_1		IO_LIN_Read	
IO_Pin.h	256	IO_LIN_Read	
IO_INT_SAFETY_SW_2		IO_LIN.h	200
IO_INT_SAFETY_SW_2		IO_LIN_Write	
IO_Pin.h	256	IO_LIN_Write	
IO_K15		IO_LIN.h	201
IO_K15		IO_MPU.h	
IO_Pin.h	256	IO_MPU.h	201
IO_LIN.h		IO_MPU_ACCESS_ANY	206
IO_LIN.h	195	IO_MPU_ACCESS_NONE	206
IO_LIN_BAUDRATE_MAX	198	IO_MPU_ACCESS_READ	206

IO_MPUM_ACCESS_READ_EXECUTE	206
IO_MPUM_ACCESS_READ_WRITE...	206
IO_MPUM_Disable	211
IO_MPUM_DisableAll	212
IO_MPUM_Enable	212
IO_MPUM_ENABLE_ALL_SUBREGIONS	206
IO_MPUM_ENABLE_SUBREGION_0..	206
IO_MPUM_ENABLE_SUBREGION_1..	206
IO_MPUM_ENABLE_SUBREGION_2..	206
IO_MPUM_ENABLE_SUBREGION_3..	207
IO_MPUM_ENABLE_SUBREGION_4..	207
IO_MPUM_ENABLE_SUBREGION_5..	207
IO_MPUM_ENABLE_SUBREGION_6..	207
IO_MPUM_ENABLE_SUBREGION_7..	207
IO_MPUM_EnableAll	212
IO_MPUM_Init	212
IO_MPUM_Policy	213
IO_MPUM_POLICY_ALLREGIONS ..	207
IO_MPUM_POLICY_OFF	207
IO_MPUM_POLICY_REGION0	208
IO_MPUM_REGION_0	208
IO_MPUM_REGION_1	208
IO_MPUM_REGION_2	208
IO_MPUM_REGION_3	208
IO_MPUM_SIZE_128_B	208
IO_MPUM_SIZE_128_KB	208
IO_MPUM_SIZE_16_KB	209
IO_MPUM_SIZE_16_MB	209
IO_MPUM_SIZE_1_KB	209
IO_MPUM_SIZE_1_MB	209
IO_MPUM_SIZE_256_B	209
IO_MPUM_SIZE_256_KB	209
IO_MPUM_SIZE_2_KB	209
IO_MPUM_SIZE_2_MB	209
IO_MPUM_SIZE_32_B	210
IO_MPUM_SIZE_32_KB	210
IO_MPUM_SIZE_32_MB	210
IO_MPUM_SIZE_4_KB	210
IO_MPUM_SIZE_4_MB	210
IO_MPUM_SIZE_512_B	210
IO_MPUM_SIZE_512_KB	210
IO_MPUM_SIZE_64_B	210
IO_MPUM_SIZE_64_KB	211
IO_MPUM_SIZE_64_MB	211
IO_MPUM_SIZE_8_KB	211
IO_MPUM_SIZE_8_MB	211
IO_MPUM_ACCESS_ANY	
IO_MPUM_ACCESS_ANY	
IO_MPUM.h	206
IO_MPUM_ACCESS_NONE	
IO_MPUM_ACCESS_NONE	
IO_MPUM.h	206
IO_MPUM_ACCESS_READ	
IO_MPUM_ACCESS_READ	
IO_MPUM.h	206
IO_MPUM_ACCESS_READ_EXECUTE	
IO_MPUM_ACCESS_READ_EXECUTE	
IO_MPUM.h	206
IO_MPUM_ACCESS_READ_WRITE	
IO_MPUM_ACCESS_READ_WRITE	
IO_MPUM.h	206
IO_MPUM_Disable	
IO_MPUM_Disable	
IO_MPUM.h	211
IO_MPUM_DisableAll	
IO_MPUM_DisableAll	
IO_MPUM.h	212
IO_MPUM_Enable	
IO_MPUM_Enable	
IO_MPUM.h	212
IO_MPUM_ENABLE_ALL_SUBREGIONS	
IO_MPUM_ENABLE_ALL_SUBREGIONS	
IO_MPUM.h	206
IO_MPUM_ENABLE_SUBREGION_0	
IO_MPUM_ENABLE_SUBREGION_0	
IO_MPUM.h	206
IO_MPUM_ENABLE_SUBREGION_1	
IO_MPUM_ENABLE_SUBREGION_1	
IO_MPUM.h	206
IO_MPUM_ENABLE_SUBREGION_2	
IO_MPUM_ENABLE_SUBREGION_2	
IO_MPUM.h	206
IO_MPUM_ENABLE_SUBREGION_3	
IO_MPUM_ENABLE_SUBREGION_3	
IO_MPUM.h	207
IO_MPUM_ENABLE_SUBREGION_4	
IO_MPUM_ENABLE_SUBREGION_4	
IO_MPUM.h	207
IO_MPUM_ENABLE_SUBREGION_5	
IO_MPUM_ENABLE_SUBREGION_5	
IO_MPUM.h	207
IO_MPUM_ENABLE_SUBREGION_6	
IO_MPUM_ENABLE_SUBREGION_6	
IO_MPUM.h	207
IO_MPUM_ENABLE_SUBREGION_7	
IO_MPUM_ENABLE_SUBREGION_7	
IO_MPUM.h	207
IO_MPUM_EnableAll	

IO_MPU_EnableAll	IO_MPU.h	209
IO_MPU.h	212	
IO_MPU_Init	IO_MPU_Init	
IO_MPU.h	212	
IO_MPU_Policy	IO_MPU_Policy	
IO_MPU.h	213	
IO_MPU_POLICY_ALLREGIONS	IO_MPU_POLICY_ALLREGIONS	
IO_MPU.h	207	
IO_MPU_POLICY_OFF	IO_MPU_POLICY_OFF	
IO_MPU.h	207	
IO_MPU_POLICY_REGION0	IO_MPU_POLICY_REGION0	
IO_MPU.h	208	
IO_MPU_REGION_0	IO_MPU_REGION_0	
IO_MPU.h	208	
IO_MPU_REGION_1	IO_MPU_REGION_1	
IO_MPU.h	208	
IO_MPU_REGION_2	IO_MPU_REGION_2	
IO_MPU.h	208	
IO_MPU_REGION_3	IO_MPU_REGION_3	
IO_MPU.h	208	
IO_MPU_SIZE_128_B	IO_MPU_SIZE_128_B	
IO_MPU.h	208	
IO_MPU_SIZE_128_KB	IO_MPU_SIZE_128_KB	
IO_MPU.h	208	
IO_MPU_SIZE_16_KB	IO_MPU_SIZE_16_KB	
IO_MPU.h	209	
IO_MPU_SIZE_16_MB	IO_MPU_SIZE_16_MB	
IO_MPU.h	209	
IO_MPU_SIZE_1_KB	IO_MPU_SIZE_1_KB	
IO_MPU.h	209	
IO_MPU_SIZE_1_MB	IO_MPU_SIZE_1_MB	
IO_MPU.h	209	
IO_MPU_SIZE_256_B	IO_MPU_SIZE_256_B	
IO_MPU.h	209	
IO_MPU.h	209	
IO_MPU_SIZE_256_KB	IO_MPU_SIZE_256_KB	
IO_MPU.h	209	
IO_MPU_SIZE_2_KB	IO_MPU_SIZE_2_KB	
IO_MPU.h	209	
IO_MPU_SIZE_2_MB	IO_MPU_SIZE_2_MB	
IO_MPU.h	209	
IO_MPU_SIZE_32_B	IO_MPU_SIZE_32_B	
IO_MPU.h	210	
IO_MPU_SIZE_32_KB	IO_MPU_SIZE_32_KB	
IO_MPU.h	210	
IO_MPU_SIZE_32_MB	IO_MPU_SIZE_32_MB	
IO_MPU.h	210	
IO_MPU_SIZE_4_KB	IO_MPU_SIZE_4_KB	
IO_MPU.h	210	
IO_MPU_SIZE_4_MB	IO_MPU_SIZE_4_MB	
IO_MPU.h	210	
IO_MPU_SIZE_512_B	IO_MPU_SIZE_512_B	
IO_MPU.h	210	
IO_MPU_SIZE_512_KB	IO_MPU_SIZE_512_KB	
IO_MPU.h	210	
IO_MPU_SIZE_64_B	IO_MPU_SIZE_64_B	
IO_MPU.h	210	
IO_MPU_SIZE_64_KB	IO_MPU_SIZE_64_KB	
IO_MPU.h	211	
IO_MPU_SIZE_64_MB	IO_MPU_SIZE_64_MB	
IO_MPU.h	211	
IO_MPU_SIZE_8_KB	IO_MPU_SIZE_8_KB	
IO_MPU.h	211	
IO_MPU_SIZE_8_MB	IO_MPU_SIZE_8_MB	
IO_MPU.h	211	
IO_Pin.h	IO_Pin.h	
IO_Pin.h	214	
IO_ADC_00	225	

IO_ADC_01.....	225	IO_ADC_50.....	231
IO_ADC_02.....	225	IO_ADC_51.....	231
IO_ADC_03.....	225	IO_ADC_52.....	231
IO_ADC_04.....	225	IO_ADC_53.....	231
IO_ADC_05.....	225	IO_ADC_54.....	232
IO_ADC_06.....	226	IO_ADC_55.....	232
IO_ADC_07.....	226	IO_ADC_56.....	232
IO_ADC_08.....	226	IO_ADC_57.....	232
IO_ADC_09.....	226	IO_ADC_58.....	232
IO_ADC_10.....	226	IO_ADC_59.....	232
IO_ADC_11.....	226	IO_ADC_BOARD_TEMP.....	232
IO_ADC_12.....	226	IO_ADC_K15.....	232
IO_ADC_13.....	226	IO_ADC_SAFETY_SW_0.....	232
IO_ADC_14.....	227	IO_ADC_SAFETY_SW_1.....	233
IO_ADC_15.....	227	IO_ADC_SAFETY_SW_2.....	233
IO_ADC_16.....	227	IO_ADC_SENSOR_SUPPLY_0.....	233
IO_ADC_17.....	227	IO_ADC_SENSOR_SUPPLY_1.....	233
IO_ADC_18.....	227	IO_ADC_SENSOR_SUPPLY_2.....	233
IO_ADC_19.....	227	IO_ADC_UBAT.....	233
IO_ADC_20.....	227	IO_ADC_WAKE_UP.....	233
IO_ADC_21.....	227	IO_CAN_CHANNEL_0.....	233
IO_ADC_22.....	228	IO_CAN_CHANNEL_1.....	233
IO_ADC_23.....	228	IO_CAN_CHANNEL_2.....	233
IO_ADC_24.....	228	IO_CAN_CHANNEL_3.....	234
IO_ADC_25.....	228	IO_CAN_CHANNEL_4.....	234
IO_ADC_26.....	228	IO_CAN_CHANNEL_5.....	234
IO_ADC_27.....	228	IO_CAN_CHANNEL_6.....	234
IO_ADC_28.....	228	IO_DI_00.....	234
IO_ADC_29.....	228	IO_DI_01.....	234
IO_ADC_30.....	229	IO_DI_02.....	234
IO_ADC_31.....	229	IO_DI_03.....	234
IO_ADC_32.....	229	IO_DI_04.....	235
IO_ADC_33.....	229	IO_DI_05.....	235
IO_ADC_34.....	229	IO_DI_06.....	235
IO_ADC_35.....	229	IO_DI_07.....	235
IO_ADC_36.....	229	IO_DI_08.....	235
IO_ADC_37.....	229	IO_DI_09.....	235
IO_ADC_38.....	230	IO_DI_10.....	235
IO_ADC_39.....	230	IO_DI_11.....	235
IO_ADC_40.....	230	IO_DI_12.....	236
IO_ADC_41.....	230	IO_DI_13.....	236
IO_ADC_42.....	230	IO_DI_14.....	236
IO_ADC_43.....	230	IO_DI_15.....	236
IO_ADC_44.....	230	IO_DI_16.....	236
IO_ADC_45.....	230	IO_DI_17.....	236
IO_ADC_46.....	231	IO_DI_18.....	236
IO_ADC_47.....	231	IO_DI_19.....	236
IO_ADC_48.....	231	IO_DI_20.....	237
IO_ADC_49.....	231	IO_DI_21.....	237

IO_DI_22	237	IO_DI_71	243
IO_DI_23	237	IO_DI_72	243
IO_DI_24	237	IO_DI_73	243
IO_DI_25	237	IO_DI_74	243
IO_DI_26	237	IO_DI_75	243
IO_DI_27	237	IO_DI_76	244
IO_DI_28	238	IO_DI_77	244
IO_DI_29	238	IO_DI_78	244
IO_DI_30	238	IO_DI_79	244
IO_DI_31	238	IO_DI_80	244
IO_DI_32	238	IO_DI_81	244
IO_DI_33	238	IO_DI_82	244
IO_DI_34	238	IO_DI_83	244
IO_DI_35	238	IO_DI_84	245
IO_DI_36	239	IO_DI_85	245
IO_DI_37	239	IO_DI_86	245
IO_DI_38	239	IO_DI_87	245
IO_DI_39	239	IO_DI_88	245
IO_DI_40	239	IO_DI_89	245
IO_DI_41	239	IO_DI_90	245
IO_DI_42	239	IO_DI_91	245
IO_DI_43	239	IO_DI_92	246
IO_DI_44	240	IO_DI_93	246
IO_DI_45	240	IO_DI_94	246
IO_DI_46	240	IO_DO_00	246
IO_DI_47	240	IO_DO_01	246
IO_DI_48	240	IO_DO_02	246
IO_DI_49	240	IO_DO_03	246
IO_DI_50	240	IO_DO_04	247
IO_DI_51	240	IO_DO_05	247
IO_DI_52	241	IO_DO_06	247
IO_DI_53	241	IO_DO_07	247
IO_DI_54	241	IO_DO_08	247
IO_DI_55	241	IO_DO_09	247
IO_DI_56	241	IO_DO_10	247
IO_DI_57	241	IO_DO_11	247
IO_DI_58	241	IO_DO_12	248
IO_DI_59	241	IO_DO_13	248
IO_DI_60	242	IO_DO_14	248
IO_DI_61	242	IO_DO_15	248
IO_DI_62	242	IO_DO_16	248
IO_DI_63	242	IO_DO_17	248
IO_DI_64	242	IO_DO_18	248
IO_DI_65	242	IO_DO_19	248
IO_DI_66	242	IO_DO_20	249
IO_DI_67	242	IO_DO_21	249
IO_DI_68	243	IO_DO_22	249
IO_DI_69	243	IO_DO_23	249
IO_DI_70	243		

IO_DO_24	249	IO_INT_PIN_SAFETY_SW_1	255
IO_DO_25	249	IO_INT_PIN_SAFETY_SW_2	255
IO_DO_26	249	IO_INT_PIN_SAFETY_SW_VP	255
IO_DO_27	249	IO_INT_PIN_TEMP	256
IO_DO_28	250	IO_INT_PIN_VMON	256
IO_DO_29	250	IO_INT_POWERSTAGE_ENABLE ..	256
IO_DO_30	250	IO_INT_SAFETY_SW_0	256
IO_DO_31	250	IO_INT_SAFETY_SW_1	256
IO_DO_32	250	IO_INT_SAFETY_SW_2	256
IO_DO_33	250	IO_K15	256
IO_DO_34	250	IO_PIN_101	256
IO_DO_35	250	IO_PIN_102	257
IO_DO_36	251	IO_PIN_103	257
IO_DO_37	251	IO_PIN_104	257
IO_DO_38	251	IO_PIN_105	257
IO_DO_39	251	IO_PIN_106	257
IO_DO_40	251	IO_PIN_107	257
IO_DO_41	251	IO_PIN_108	257
IO_DO_42	251	IO_PIN_109	257
IO_DO_43	251	IO_PIN_110	258
IO_DO_44	252	IO_PIN_111	258
IO_DO_45	252	IO_PIN_112	258
IO_DO_46	252	IO_PIN_113	258
IO_DO_47	252	IO_PIN_114	258
IO_DO_48	252	IO_PIN_115	258
IO_DO_49	252	IO_PIN_116	258
IO_DO_50	252	IO_PIN_117	258
IO_DO_51	252	IO_PIN_118	259
IO_DO_52	253	IO_PIN_122	259
IO_DO_53	253	IO_PIN_123	259
IO_DO_54	253	IO_PIN_124	259
IO_DO_55	253	IO_PIN_125	259
IO_DO_56	253	IO_PIN_126	259
IO_DO_57	253	IO_PIN_127	259
IO_DO_58	253	IO_PIN_128	259
IO_DO_59	253	IO_PIN_129	260
IO_GND	254	IO_PIN_130	260
IO_INT_PIN_1V2	254	IO_PIN_131	260
IO_INT_PIN_CAN_CH0	254	IO_PIN_132	260
IO_INT_PIN_CAN_CH1	254	IO_PIN_133	260
IO_INT_PIN_CAN_CH2	254	IO_PIN_134	260
IO_INT_PIN_CAN_CH3	254	IO_PIN_135	260
IO_INT_PIN_CAN_CH4	254	IO_PIN_136	260
IO_INT_PIN_CAN_CH5	254	IO_PIN_137	261
IO_INT_PIN_CAN_CH6	255	IO_PIN_138	261
IO_INT_PIN_ENDRV_CPU	255	IO_PIN_139	261
IO_INT_PIN_PWD	255	IO_PIN_140	261
IO_INT_PIN_REF_2V5	255	IO_PIN_141	261
IO_INT_PIN_SAFETY_SW_0	255	IO_PIN_146	261

IO_PIN_147.....	261	IO_PIN_196.....	267
IO_PIN_148.....	261	IO_PIN_201.....	268
IO_PIN_149.....	262	IO_PIN_207.....	268
IO_PIN_150.....	262	IO_PIN_220.....	268
IO_PIN_151.....	262	IO_PIN_221.....	268
IO_PIN_152.....	262	IO_PIN_234.....	268
IO_PIN_153.....	262	IO_PIN_238.....	268
IO_PIN_154.....	262	IO_PIN_239.....	268
IO_PIN_155.....	262	IO_PIN_240.....	268
IO_PIN_156.....	262	IO_PIN_241.....	269
IO_PIN_157.....	263	IO_PIN_246.....	269
IO_PIN_158.....	263	IO_PIN_247.....	269
IO_PIN_159.....	263	IO_PIN_251.....	269
IO_PIN_160.....	263	IO_PIN_252.....	269
IO_PIN_161.....	263	IO_PIN_253.....	269
IO_PIN_162.....	263	IO_PIN_254.....	269
IO_PIN_163.....	263	IO_PIN_NONE.....	269
IO_PIN_164.....	263	IO_PVG_00.....	270
IO_PIN_165.....	264	IO_PVG_01.....	270
IO_PIN_166.....	264	IO_PVG_02.....	270
IO_PIN_167.....	264	IO_PVG_03.....	270
IO_PIN_168.....	264	IO_PVG_04.....	270
IO_PIN_169.....	264	IO_PVG_05.....	270
IO_PIN_170.....	264	IO_PVG_06.....	270
IO_PIN_171.....	264	IO_PVG_07.....	270
IO_PIN_172.....	264	IO_PWD_00.....	271
IO_PIN_173.....	265	IO_PWD_01.....	271
IO_PIN_174.....	265	IO_PWD_02.....	271
IO_PIN_175.....	265	IO_PWD_03.....	271
IO_PIN_176.....	265	IO_PWD_04.....	271
IO_PIN_177.....	265	IO_PWD_05.....	271
IO_PIN_178.....	265	IO_PWD_06.....	271
IO_PIN_179.....	265	IO_PWD_07.....	271
IO_PIN_180.....	265	IO_PWD_08.....	272
IO_PIN_181.....	266	IO_PWD_09.....	272
IO_PIN_182.....	266	IO_PWD_10.....	272
IO_PIN_183.....	266	IO_PWD_11.....	272
IO_PIN_184.....	266	IO_PWD_12.....	272
IO_PIN_185.....	266	IO_PWD_13.....	272
IO_PIN_186.....	266	IO_PWD_14.....	272
IO_PIN_187.....	266	IO_PWD_15.....	272
IO_PIN_188.....	266	IO_PWD_16.....	273
IO_PIN_189.....	267	IO_PWD_17.....	273
IO_PIN_190.....	267	IO_PWD_18.....	273
IO_PIN_191.....	267	IO_PWD_19.....	273
IO_PIN_192.....	267	IO_PWM_00.....	273
IO_PIN_193.....	267	IO_PWM_01.....	273
IO_PIN_194.....	267	IO_PWM_02.....	273
IO_PIN_195.....	267	IO_PWM_03.....	273

IO_PWM_04	274	IO_PIN_102 IO_Pin.h	257
IO_PWM_05	274	IO_PIN_103 IO_PIN_103 IO_Pin.h	257
IO_PWM_06	274	IO_PIN_104 IO_PIN_104 IO_Pin.h	257
IO_PWM_07	274	IO_PIN_105 IO_PIN_105 IO_Pin.h	257
IO_PWM_08	274	IO_PIN_106 IO_PIN_106 IO_Pin.h	257
IO_PWM_09	274	IO_PIN_107 IO_PIN_107 IO_Pin.h	257
IO_PWM_10	274	IO_PIN_108 IO_PIN_108 IO_Pin.h	257
IO_PWM_11	274	IO_PIN_109 IO_PIN_109 IO_Pin.h	257
IO_PWM_12	275	IO_PIN_110 IO_PIN_110 IO_Pin.h	258
IO_PWM_13	275	IO_PIN_111 IO_PIN_111 IO_Pin.h	258
IO_PWM_14	275	IO_PIN_112 IO_PIN_112 IO_Pin.h	258
IO_PWM_15	275	IO_PIN_113 IO_PIN_113 IO_Pin.h	258
IO_PWM_16	275	IO_PIN_114 IO_PIN_114 IO_Pin.h	258
IO_PWM_17	275	IO_PIN_115 IO_PIN_115 IO_Pin.h	258
IO_PWM_18	275	IO_PIN_116 IO_PIN_116 IO_Pin.h	258
IO_PWM_19	275	IO_PIN_117 IO_PIN_117 IO_Pin.h	258
IO_PWM_20	276	IO_PIN_118 IO_PIN_118	
IO_PWM_21	276		
IO_PWM_22	276		
IO_PWM_23	276		
IO_PWM_24	276		
IO_PWM_25	276		
IO_PWM_26	276		
IO_PWM_27	276		
IO_PWM_28	277		
IO_PWM_29	277		
IO_PWM_30	277		
IO_PWM_31	277		
IO_PWM_32	277		
IO_PWM_33	277		
IO_PWM_34	277		
IO_PWM_35	277		
IO_SENSOR_SUPPLY_0	278		
IO_SENSOR_SUPPLY_1	278		
IO_SENSOR_SUPPLY_2	278		
IO_UBAT	278		
IO_VOUT_00	278		
IO_VOUT_01	278		
IO_VOUT_02	278		
IO_VOUT_03	278		
IO_VOUT_04	278		
IO_VOUT_05	279		
IO_VOUT_06	279		
IO_VOUT_07	279		
IO_WAKEUP	279		
IO_PIN_101 IO_PIN_101 IO_Pin.h	256		
IO_PIN_102			

IO_Pin.h	259
IO_PIN_122	
IO_PIN_122	
IO_Pin.h	259
IO_PIN_123	
IO_PIN_123	
IO_Pin.h	259
IO_PIN_124	
IO_PIN_124	
IO_Pin.h	259
IO_PIN_125	
IO_PIN_125	
IO_Pin.h	259
IO_PIN_126	
IO_PIN_126	
IO_Pin.h	259
IO_PIN_127	
IO_PIN_127	
IO_Pin.h	259
IO_PIN_128	
IO_PIN_128	
IO_Pin.h	259
IO_PIN_129	
IO_PIN_129	
IO_Pin.h	260
IO_PIN_130	
IO_PIN_130	
IO_Pin.h	260
IO_PIN_131	
IO_PIN_131	
IO_Pin.h	260
IO_PIN_132	
IO_PIN_132	
IO_Pin.h	260
IO_PIN_133	
IO_PIN_133	
IO_Pin.h	260
IO_PIN_134	
IO_PIN_134	
IO_Pin.h	260
IO_PIN_135	
IO_PIN_135	
IO_Pin.h	260
IO_PIN_136	
IO_PIN_136	
IO_Pin.h	260
IO_PIN_137	
IO_PIN_137	
IO_Pin.h	261
IO_PIN_138	
IO_PIN_138	
IO_Pin.h	261
IO_PIN_139	
IO_PIN_139	
IO_Pin.h	261
IO_PIN_140	
IO_PIN_140	
IO_Pin.h	261
IO_PIN_141	
IO_PIN_141	
IO_Pin.h	261
IO_PIN_142	
IO_PIN_142	
IO_Pin.h	261
IO_PIN_143	
IO_PIN_143	
IO_Pin.h	261
IO_PIN_144	
IO_PIN_144	
IO_Pin.h	261
IO_PIN_145	
IO_PIN_145	
IO_Pin.h	261
IO_PIN_146	
IO_PIN_146	
IO_Pin.h	261
IO_PIN_147	
IO_PIN_147	
IO_Pin.h	261
IO_PIN_148	
IO_PIN_148	
IO_Pin.h	261
IO_PIN_149	
IO_PIN_149	
IO_Pin.h	262
IO_PIN_150	
IO_PIN_150	
IO_Pin.h	262
IO_PIN_151	
IO_PIN_151	
IO_Pin.h	262
IO_PIN_152	
IO_PIN_152	
IO_Pin.h	262
IO_PIN_153	
IO_PIN_153	
IO_Pin.h	262
IO_PIN_154	
IO_PIN_154	
IO_Pin.h	262
IO_PIN_155	
IO_PIN_155	
IO_Pin.h	262
IO_PIN_156	
IO_PIN_156	
IO_Pin.h	262
IO_PIN_157	
IO_PIN_157	
IO_Pin.h	263
IO_PIN_158	

IO_PIN_158	IO_Pin.h	265
IO_PIN_158	IO_Pin.h	263
IO_PIN_159	IO_PIN_159	
IO_PIN_159	IO_Pin.h	263
IO_PIN_160	IO_PIN_160	
IO_PIN_160	IO_Pin.h	263
IO_PIN_161	IO_PIN_161	
IO_PIN_161	IO_Pin.h	263
IO_PIN_162	IO_PIN_162	
IO_PIN_162	IO_Pin.h	263
IO_PIN_163	IO_PIN_163	
IO_PIN_163	IO_Pin.h	263
IO_PIN_164	IO_PIN_164	
IO_PIN_164	IO_Pin.h	263
IO_PIN_165	IO_PIN_165	
IO_PIN_165	IO_Pin.h	264
IO_PIN_166	IO_PIN_166	
IO_PIN_166	IO_Pin.h	264
IO_PIN_167	IO_PIN_167	
IO_PIN_167	IO_Pin.h	264
IO_PIN_168	IO_PIN_168	
IO_PIN_168	IO_Pin.h	264
IO_PIN_169	IO_PIN_169	
IO_PIN_169	IO_Pin.h	264
IO_PIN_170	IO_PIN_170	
IO_PIN_170	IO_Pin.h	264
IO_PIN_171	IO_PIN_171	
IO_PIN_171	IO_Pin.h	264
IO_PIN_172	IO_PIN_172	
IO_PIN_172	IO_Pin.h	264
IO_PIN_173	IO_PIN_173	
IO_PIN_173	IO_Pin.h	265
IO_PIN_174	IO_PIN_174	
IO_PIN_174	IO_Pin.h	265
IO_PIN_175	IO_PIN_175	
IO_PIN_175	IO_Pin.h	265
IO_PIN_176	IO_PIN_176	
IO_PIN_176	IO_Pin.h	265
IO_PIN_177	IO_PIN_177	
IO_PIN_177	IO_Pin.h	265
IO_PIN_178	IO_PIN_178	
IO_PIN_178	IO_Pin.h	265
IO_PIN_179	IO_PIN_179	
IO_PIN_179	IO_Pin.h	265
IO_PIN_180	IO_PIN_180	
IO_PIN_180	IO_Pin.h	265
IO_PIN_181	IO_PIN_181	
IO_PIN_181	IO_Pin.h	266
IO_PIN_182	IO_PIN_182	
IO_PIN_182	IO_Pin.h	266
IO_PIN_183	IO_PIN_183	
IO_PIN_183	IO_Pin.h	266
IO_PIN_184	IO_PIN_184	
IO_PIN_184	IO_Pin.h	266
IO_PIN_185	IO_PIN_185	
IO_PIN_185	IO_Pin.h	266
IO_PIN_186	IO_PIN_186	
IO_PIN_186	IO_Pin.h	266
IO_PIN_187	IO_PIN_187	
IO_PIN_187	IO_Pin.h	266
IO_PIN_188	IO_PIN_188	
IO_PIN_188	IO_Pin.h	266
IO_PIN_189	IO_PIN_189	
IO_PIN_189	IO_Pin.h	267
IO_PIN_190	IO_PIN_190	
IO_PIN_190	IO_Pin.h	267

IO_PIN_191	IO_PIN_247
IO_PIN_191	IO_Pin.h
IO_Pin.h	269
IO_PIN_192	IO_PIN_251
IO_PIN_192	IO_PIN_251
IO_Pin.h	IO_Pin.h
267	269
IO_PIN_193	IO_PIN_252
IO_PIN_193	IO_PIN_252
IO_Pin.h	IO_Pin.h
267	269
IO_PIN_194	IO_PIN_253
IO_PIN_194	IO_PIN_253
IO_Pin.h	IO_Pin.h
267	269
IO_PIN_195	IO_PIN_254
IO_PIN_195	IO_PIN_254
IO_Pin.h	IO_Pin.h
267	269
IO_PIN_196	IO_PIN_NONE
IO_PIN_196	IO_PIN_NONE
IO_Pin.h	IO_Pin.h
267	269
IO_PIN_201	IO_POWER.h
IO_PIN_201	IO_POWER.h
IO_Pin.h	279
268	IO_POWER_EnableExtShutOff
IO_PIN_207	IO_POWER_Get
IO_PIN_207	IO_POWER_GetExtShutOff
IO_Pin.h	IO_POWER_OFF
268	IO_POWER_ON
IO_PIN_220	IO_POWER_ON_10V
IO_PIN_220	IO_POWER_ON_5V
IO_Pin.h	IO_POWER_ON_6V
268	IO_POWER_ON_7V
IO_PIN_221	IO_POWER_ON_8V
IO_PIN_221	IO_POWER_ON_9V
IO_Pin.h	IO_POWER_Set
268	IO_POWER_EnableExtShutOff
IO_PIN_234	IO_POWER_EnableExtShutOff
IO_PIN_234	IO_POWER.h
IO_Pin.h	284
268	IO_POWER_Get
IO_PIN_238	IO_POWER.h
IO_PIN_238	IO_POWER_GetExtShutOff
IO_Pin.h	IO_POWER_OFF
268	IO_POWER_ON
IO_PIN_239	IO_POWER_ON_10V
IO_PIN_239	IO_POWER_ON_10V
IO_Pin.h	IO_POWER_ON_10V
268	IO_POWER_ON_10V
IO_PIN_240	IO_POWER_ON_10V
IO_PIN_240	IO_POWER_ON_10V
IO_Pin.h	IO_POWER_ON_10V
268	IO_POWER_ON_10V
IO_PIN_241	IO_POWER_ON_10V
IO_PIN_241	IO_POWER_ON_10V
IO_Pin.h	IO_POWER_ON_10V
269	IO_POWER_ON_10V
IO_PIN_246	IO_POWER_ON_10V
IO_PIN_246	IO_POWER_ON_10V
IO_Pin.h	IO_POWER_ON_10V
269	IO_POWER_ON_10V
IO_PIN_247	IO_POWER_ON_10V

IO_POWER_ON_5V		IO_PVG_DeInit	
IO_POWER_ON_5V		IO_PVG_DeInit	
IO_POWER.h	283	IO_PVG.h	290
IO_POWER_ON_6V		IO_PVG_GetVoltage	
IO_POWER_ON_6V		IO_PVG_GetVoltage	
IO_POWER.h	283	IO_PVG.h	291
IO_POWER_ON_7V		IO_PVG_Init	
IO_POWER_ON_7V		IO_PVG_Init	
IO_POWER.h	283	IO_PVG.h	292
IO_POWER_ON_8V		IO_PVG_ResetProtection	
IO_POWER_ON_8V		IO_PVG_ResetProtection	
IO_POWER.h	283	IO_PVG.h	292
IO_POWER_ON_9V		IO_PVG_SetOutput	
IO_POWER_ON_9V		IO_PVG_SetOutput	
IO_POWER.h	283	IO_PVG.h	293
IO_POWER_Set		IO_PWD.h	
IO_POWER_Set		IO_PWD.h	294
IO_POWER.h	287	IO_PWD_BOTH_COUNT	299
IO_PVG.h		IO_PWD_CNT_CONF	301
IO_PVG.h	288	IO_PWD_ComplexDeInit	302
IO_PVG_DeInit	290	IO_PWD_ComplexGet	303
IO_PVG_GetVoltage	291	IO_PWD_ComplexInit	304
IO_PVG_Init	292	IO_PWD_CountDeInit	306
IO_PVG_ResetProtection	292	IO_PWD_CountGet	307
IO_PVG_SetOutput	293	IO_PWD_CountInit	307
IO_PVG_00		IO_PWD_CountSet	309
IO_PVG_00		IO_PWD_CPLX_CONF	301
IO_Pin.h	270	IO_PWD_CPLX_SAFETY_CONF	302
IO_PVG_01		IO_PWD_DOWN_COUNT	299
IO_PVG_01		IO_PWD_FALLING_COUNT	299
IO_Pin.h	270	IO_PWD_FALLING_VAR	299
IO_PVG_02		IO_PWD_GetCurrent	309
IO_PVG_02		IO_PWD_GetVoltage	310
IO_Pin.h	270	IO_PWD_HIGH_TIME	300
IO_PVG_03		IO_PWD_INC_1_COUNT	300
IO_PVG_03		IO_PWD_INC_2_COUNT	300
IO_Pin.h	270	IO_PWD_INC_CONF	302
IO_PVG_04		IO_PWD_INC_SAFETY_CONF	302
IO_PVG_04		IO_PWD_IncDeInit	311
IO_Pin.h	270	IO_PWD_IncGet	312
IO_PVG_05		IO_PWD_IncInit	312
IO_PVG_05		IO_PWD_IncSet	314
IO_Pin.h	270	IO_PWD_LOW_TIME	300
IO_PVG_06		IO_PWD_MAX_PULSE_SAMPLES ..	300
IO_PVG_06		IO_PWD_NO_PULL	300
IO_Pin.h	270	IO_PWD_PD_10K	300
IO_PVG_07		IO_PWD_PD_90	301
IO_PVG_07		IO_PWD_PERIOD_TIME	301
IO_Pin.h	270	IO_PWD_PU_10K	301

IO_PWD_PULSE_SAMPLES	302
IO_PWD_ResetProtection.....	315
IO_PWD_RISING_COUNT.....	301
IO_PWD_RISING_VAR.....	301
IO_PWD_UNIVERSAL_SAFETY_CONF	302
IO_PWD_UniversalDelInit	315
IO_PWD_UniversalGet.....	316
IO_PWD_UniversalInit.....	317
IO_PWD_UniversalSet.....	319
IO_PWD_UP_COUNT.....	301
IO_PWD_00	
IO_PWD_00	
IO_Pin.h	271
IO_PWD_01	
IO_PWD_01	
IO_Pin.h	271
IO_PWD_02	
IO_PWD_02	
IO_Pin.h	271
IO_PWD_03	
IO_PWD_03	
IO_Pin.h	271
IO_PWD_04	
IO_PWD_04	
IO_Pin.h	271
IO_PWD_05	
IO_PWD_05	
IO_Pin.h	271
IO_PWD_06	
IO_PWD_06	
IO_Pin.h	271
IO_PWD_07	
IO_PWD_07	
IO_Pin.h	271
IO_PWD_08	
IO_PWD_08	
IO_Pin.h	272
IO_PWD_09	
IO_PWD_09	
IO_Pin.h	272
IO_PWD_10	
IO_PWD_10	
IO_Pin.h	272
IO_PWD_11	
IO_PWD_11	
IO_Pin.h	272
IO_PWD_12	
IO_PWD_12	
IO_Pin.h	272
IO_PWD_13	
IO_PWD_13	
IO_Pin.h	272
IO_PWD_14	
IO_PWD_14	
IO_Pin.h	272
IO_PWD_15	
IO_PWD_15	
IO_Pin.h	272
IO_PWD_16	
IO_PWD_16	
IO_Pin.h	273
IO_PWD_17	
IO_PWD_17	
IO_Pin.h	273
IO_PWD_18	
IO_PWD_18	
IO_Pin.h	273
IO_PWD_19	
IO_PWD_19	
IO_Pin.h	273
IO_PWD_BOTH_COUNT	
IO_PWD_BOTH_COUNT	
IO_PWD.h	299
IO_PWD_CNT_CONF	
IO_PWD_CNT_CONF	
IO_PWD.h	301
io_pwd_cnt_conf	
io_pwd_cnt_conf	29
direction	29
init	29
mode	29
IO_PWD_ComplexDelInit	
IO_PWD_ComplexDelInit	
IO_PWD.h	302
IO_PWD_ComplexGet	
IO_PWD_ComplexGet	
IO_PWD.h	303
IO_PWD_ComplexInit	
IO_PWD_ComplexInit	
IO_PWD.h	304
IO_PWD_CountDelInit	
IO_PWD_CountDelInit	
IO_PWD.h	306
IO_PWD_CountGet	
IO_PWD_CountGet	
IO_PWD.h	307
IO_PWD_CountInit	
IO_PWD_CountInit	

IO_PWD.h	307
IO_PWD_CountSet	
IO_PWD_CountSet	
IO_PWD.h	309
IO_PWD_CPLX_CONF	
IO_PWD_CPLX_CONF	
IO_PWD.h	301
io_pwd_cplx_conf_	
io_pwd_cplx_conf_	29
capture_count	30
freq_mode	30
pulse_mode	30
IO_PWD_CPLX_SAFETY_CONF	
IO_PWD_CPLX_SAFETY_CONF	
IO_PWD.h	302
io_pwd_cplx_safety_conf_	
io_pwd_cplx_safety_conf_	30
pwd_freq_val_lower	31
pwd_freq_val_upper	31
pwd_pulse_val_lower	31
pwd_pulse_val_upper	31
IO_PWD_DOWN_COUNT	
IO_PWD_DOWN_COUNT	
IO_PWD.h	299
IO_PWD_FALLING_COUNT	
IO_PWD_FALLING_COUNT	
IO_PWD.h	299
IO_PWD_FALLING_VAR	
IO_PWD_FALLING_VAR	
IO_PWD.h	299
IO_PWD_GetCurrent	
IO_PWD_GetCurrent	
IO_PWD.h	309
IO_PWD_GetVoltage	
IO_PWD_GetVoltage	
IO_PWD.h	310
IO_PWD_HIGH_TIME	
IO_PWD_HIGH_TIME	
IO_PWD.h	300
IO_PWD_INC_1_COUNT	
IO_PWD_INC_1_COUNT	
IO_PWD.h	300
IO_PWD_INC_2_COUNT	
IO_PWD_INC_2_COUNT	
IO_PWD.h	300
IO_PWD_INC_CONF	
IO_PWD_INC_CONF	
IO_PWD.h	302
io_pwd_inc_conf_	
io_pwd_inc_conf_	31
init	32
mode	32
IO_PWD_INC_SAFETY_CONF	
IO_PWD_INC_SAFETY_CONF	
IO_PWD.h	302
io_pwd_inc_safety_conf_	
io_pwd_inc_safety_conf_	32
pwd_cnt_val_lower	33
pwd_cnt_val_upper	33
IO_PWD_IncDelInit	
IO_PWD_IncDelInit	
IO_PWD.h	311
IO_PWD_IncGet	
IO_PWD_IncGet	
IO_PWD.h	312
IO_PWD_InclInit	
IO_PWD_InclInit	
IO_PWD.h	312
IO_PWD_IncSet	
IO_PWD_IncSet	
IO_PWD.h	314
IO_PWD_LOW_TIME	
IO_PWD_LOW_TIME	
IO_PWD.h	300
IO_PWD_MAX_PULSE_SAMPLES	
IO_PWD_MAX_PULSE_SAMPLES	
IO_PWD.h	300
IO_PWD_NO_PULL	
IO_PWD_NO_PULL	
IO_PWD.h	300
IO_PWD_PD_10K	
IO_PWD_PD_10K	
IO_PWD.h	300
IO_PWD_PD_90	
IO_PWD_PD_90	
IO_PWD.h	301
IO_PWD_PERIOD_TIME	
IO_PWD_PERIOD_TIME	
IO_PWD.h	301
IO_PWD_PU_10K	
IO_PWD_PU_10K	
IO_PWD.h	301
IO_PWD_PULSE_SAMPLES	
IO_PWD_PULSE_SAMPLES	
IO_PWD.h	302
io_pwd_pulse_samples_	
io_pwd_pulse_samples_	33
pulse_sample	33

pulse_samples_count	33
IO_PWD_ResetProtection	
IO_PWD_ResetProtection	
IO_PWD.h	315
IO_PWD_RISING_COUNT	
IO_PWD_RISING_COUNT	
IO_PWD.h	301
IO_PWD_RISING_VAR	
IO_PWD_RISING_VAR	
IO_PWD.h	301
IO_PWD_UNIVERSAL_SAFETY_CONF	
IO_PWD_UNIVERSAL_SAFETY_CONF	
IO_PWD.h	302
io_pwd_universal_safety_conf_	
io_pwd_universal_safety_conf_	34
pwd_cnt_safety_conf.....	34
pwd_cplx_safety_conf.....	34
pwd_inc_safety_conf.....	35
IO_PWD_UniversalDelInit	
IO_PWD_UniversalDelInit	
IO_PWD.h	315
IO_PWD_UniversalGet	
IO_PWD_UniversalGet	
IO_PWD.h	316
IO_PWD_UniversalInit	
IO_PWD_UniversalInit	
IO_PWD.h	317
IO_PWD_UniversalSet	
IO_PWD_UniversalSet	
IO_PWD.h	319
IO_PWD_UP_COUNT	
IO_PWD_UP_COUNT	
IO_PWD.h	301
IO_PWM.h	
IO_PWM.h	320
IO_PWM_CURRENT_QUEUE	323
IO_PWM_CURRENT_QUEUE_MAX	323
IO_PWM_DelInit.....	323
IO_PWM_GetCur	324
IO_PWM_GetCurQueue	324
IO_PWM_Init	326
IO_PWM_InitWithLowSide.....	328
IO_PWM_ResetProtection	329
IO_PWM_ResolveOpenLoadShortCircuit	330
IO_PWM_SAFETY_CONF	323
IO_PWM_SetDuty	331
IO_PWM_00	
IO_PWM_00	
IO_Pin.h	273
IO_PWM_01	
IO_PWM_01	
IO_Pin.h	273
IO_PWM_02	
IO_PWM_02	
IO_Pin.h	273
IO_PWM_03	
IO_PWM_03	
IO_Pin.h	273
IO_PWM_04	
IO_PWM_04	
IO_Pin.h	274
IO_PWM_05	
IO_PWM_05	
IO_Pin.h	274
IO_PWM_06	
IO_PWM_06	
IO_Pin.h	274
IO_PWM_07	
IO_PWM_07	
IO_Pin.h	274
IO_PWM_08	
IO_PWM_08	
IO_Pin.h	274
IO_PWM_09	
IO_PWM_09	
IO_Pin.h	274
IO_PWM_10	
IO_PWM_10	
IO_Pin.h	274
IO_PWM_11	
IO_PWM_11	
IO_Pin.h	274
IO_PWM_12	
IO_PWM_12	
IO_Pin.h	275
IO_PWM_13	
IO_PWM_13	
IO_Pin.h	275
IO_PWM_14	
IO_PWM_14	
IO_Pin.h	275
IO_PWM_15	
IO_PWM_15	
IO_Pin.h	275
IO_PWM_16	
IO_PWM_16	
IO_Pin.h	275
IO_PWM_17	

IO_PWM_17	IO_Pin.h	275	IO_Pin.h	277
IO_Pin.h	275			
IO_PWM_18	IO_PWM_18		IO_PWM_34	277
IO_Pin.h	275		IO_Pin.h	277
IO_PWM_19	IO_PWM_19		IO_PWM_35	
IO_Pin.h	275		IO_PWM_35	
IO_PWM_20	IO_PWM_20		IO_Pin.h	277
IO_Pin.h	276		IO_PWM_CURRENT_QUEUE	
IO_PWM_21	IO_PWM_21		IO_PWM_CURRENT_QUEUE	
IO_Pin.h	276		IO_PWM.h	323
IO_PWM_22	IO_PWM_22		io_pwm_current_queue_	
IO_Pin.h	276		io_pwm_current_queue_	35
IO_PWM_23	IO_PWM_23		count.....	35
IO_Pin.h	276		overrun.....	35
IO_PWM_24	IO_PWM_24		values.....	35
IO_Pin.h	276		IO_PWM_CURRENT_QUEUE_MAX	
IO_PWM_25	IO_PWM_25		IO_PWM_CURRENT_QUEUE_MAX	
IO_Pin.h	276		IO_PWM.h	323
IO_PWM_26	IO_PWM_26		IO_PWM_Delnit	
IO_Pin.h	276		IO_PWM_Delnit	
IO_PWM_27	IO_PWM_27		IO_PWM.h	323
IO_Pin.h	276		IO_PWM_GetCur	
IO_PWM_28	IO_PWM_28		IO_PWM_GetCur	
IO_Pin.h	277		IO_PWM.h	324
IO_PWM_29	IO_PWM_29		IO_PWM_GetCurQueue	
IO_Pin.h	277		IO_PWM_GetCurQueue	
IO_PWM_30	IO_PWM_30		IO_PWM.h	324
IO_Pin.h	277		IO_PWM_Init	
IO_PWM_31	IO_PWM_31		IO_PWM_Init	
IO_Pin.h	277		IO_PWM.h	326
IO_PWM_32	IO_PWM_32		IO_PWM_InitWithLowside	
IO_Pin.h	277		IO_PWM_InitWithLowside	
IO_PWM_33	IO_PWM_33		IO_PWM.h	328
IO_PWM.h			IO_PWM_ResetProtection	
			IO_PWM_ResetProtection	
			IO_PWM.h	329
			IO_PWM_ResolveOpenLoadShortCircuit	
			IO_PWM_ResolveOpenLoadShortCircuit	
			IO_PWM.h	330
			IO_PWM_SAFETY_CONF	
			IO_PWM_SAFETY_CONF	
			IO_PWM.h	323
			io_pwm_safety_conf_	
			io_pwm_safety_conf_	36
			current_limit.....	36
			enable_current_check.....	36
			low_side_channel.....	36
			IO_PWM_SetDuty	
			IO_PWM_SetDuty	

IO_PWM.h.....	331	IO_Pin.h	278
IO_RTC.h		IO_SENSOR_SUPPLY_1	
IO_RTC.h	332	IO_SENSOR_SUPPLY_1	
IO_RTC_DeInitDateAndTime.....	336	IO_Pin.h	278
IO_RTC_EVENT_HANDLER.....	336	IO_SENSOR_SUPPLY_2	
IO_RTC_GetDateAndTime.....	337	IO_SENSOR_SUPPLY_2	
IO_RTC_GetDateAndTimeStatus ...	337	IO_Pin.h	278
IO_RTC_GetTimeUS.....	338	IO_UART.h	
IO_RTC_InitDateAndTime.....	338	IO_UART.h	341
IO_RTC_PeriodicDeInit.....	339	IO_UART_BAUDRATE_MAX.....	344
IO_RTC_PeriodicInit.....	339	IO_UART_BAUDRATE_MIN.....	344
IO_RTC_Process	336	IO_UART_BUFFER_LEN.....	344
IO_RTC_SetDateAndTime	340	IO_UART_DeInit.....	345
IO_RTC_StartTime.....	341	IO_UART_GetRxStatus	345
IO_RTC_DeInitDateAndTime		IO_UART_GetTxStatus	345
IO_RTC_DeInitDateAndTime		IO_UART_Init	346
IO_RTC.h	336	IO_UART_PARITY_EVEN.....	344
IO_RTC_EVENT_HANDLER		IO_UART_PARITY_NONE	344
IO_RTC_EVENT_HANDLER		IO_UART_PARITY_ODD	344
IO_RTC.h	336	IO_UART_Read.....	347
IO_RTC_GetDateAndTime		IO_UART_Write	347
IO_RTC_GetDateAndTime		IO_UART_BAUDRATE_MAX	
IO_RTC.h	337	IO_UART_BAUDRATE_MAX	
IO_RTC_GetDateAndTimeStatus		IO_UART.h	344
IO_RTC_GetDateAndTimeStatus		IO_UART_BAUDRATE_MIN	
IO_RTC.h	337	IO_UART.h	344
IO_RTC_GetTimeUS		IO_UART_BUFFER_LEN	
IO_RTC_GetTimeUS		IO_UART.h	344
IO_RTC.h	338	IO_UART_DeInit	
IO_RTC_InitDateAndTime		IO_UART_DeInit	
IO_RTC_InitDateAndTime		IO_UART.h	345
IO_RTC.h	338	IO_UART_GetRxStatus	
IO_RTC_PeriodicDeInit		IO_UART_GetRxStatus	
IO_RTC_PeriodicDeInit		IO_UART.h	345
IO_RTC.h	339	IO_UART_GetTxStatus	
IO_RTC_PeriodicInit		IO_UART_GetTxStatus	
IO_RTC_PeriodicInit		IO_UART.h	345
IO_RTC.h	339	IO_UART_Init	
IO_RTC_Process		IO_UART_Init	
IO_RTC_Process		IO_UART.h	346
IO_RTC.h	336	IO_UART_PARITY_EVEN	
IO_RTC_SetDateAndTime		IO_UART_PARITY_EVEN	
IO_RTC_SetDateAndTime		IO_UART.h	344
IO_RTC.h	340	IO_UART_PARITY_NONE	
IO_RTC_StartTime		IO_UART_PARITY_NONE	
IO_RTC_StartTime		IO_UART.h	344
IO_RTC.h	341	IO_UART_PARITY_ODD	
IO_SENSOR_SUPPLY_0			
IO_SENSOR_SUPPLY_0			

IO_UART_PARITY_ODD	
IO_UART.h.....	344
IO_UART_Read	
IO_UART_Read	
IO_UART.h.....	347
IO_UART_Write	
IO_UART_Write	
IO_UART.h.....	347
IO_UBAT	
IO_UBAT	
IO_Pin.h	278
IO_UDP.h	
IO_UDP.h.....	348
IO_ETH_SPEED_100_MB	349
IO_ETH_SPEED_10_MB	349
IO_UDP_CreateSocket.....	350
IO_UDP_DelInit	350
IO_UDP_FreeSocket.....	351
IO_UDP_GetStatus	351
IO_UDP_Init	351
IO_UDP_READ	349
IO_UDP_Read.....	352
IO_UDP_WRITE	350
IO_UDP_WriteTo.....	353
IO_UDP_CreateSocket	
IO_UDP_CreateSocket	
IO_UDP.h.....	350
IO_UDP_DelInit	
IO_UDP_DelInit	
IO_UDP.h.....	350
IO_UDP_FreeSocket	
IO_UDP_FreeSocket	
IO_UDP.h.....	351
IO_UDP_GetStatus	
IO_UDP_GetStatus	
IO_UDP.h.....	351
IO_UDP_Init	
IO_UDP_Init	
IO_UDP.h.....	351
IO_UDP_READ	
IO_UDP_READ	
IO_UDP.h.....	349
IO_UDP_Read	
IO_UDP_Read	
IO_UDP.h.....	352
IO_UDP_WRITE	
IO_UDP_WRITE	
IO_UDP.h.....	350
IO_UDP_WriteTo	
IO_UDP_WriteTo	
IO_UDP_WriterTo	
IO_UDP.h.....	353
IO_VOUT.h	
IO_VOUT.h.....	354
IO_VOUT_DeInit.....	356
IO_VOUT_GetVoltage.....	356
IO_VOUT_Init.....	357
IO_VOUT_ResetProtection.....	357
IO_VOUT_SetVoltage.....	358
IO_VOUT_00	
IO_VOUT_00	
IO_Pin.h	278
IO_VOUT_01	
IO_VOUT_01	
IO_Pin.h	278
IO_VOUT_02	
IO_VOUT_02	
IO_Pin.h	278
IO_VOUT_03	
IO_VOUT_03	
IO_Pin.h	278
IO_VOUT_04	
IO_VOUT_04	
IO_Pin.h	278
IO_VOUT_05	
IO_VOUT_05	
IO_Pin.h	279
IO_VOUT_06	
IO_VOUT_06	
IO_Pin.h	279
IO_VOUT_07	
IO_VOUT_07	
IO_Pin.h	279
IO_VOUT_DelInit	
IO_VOUT_DelInit	
IO_VOUT.h	356
IO_VOUT_GetVoltage	
IO_VOUT_GetVoltage	
IO_VOUT.h	356
IO_VOUT_Init	
IO_VOUT_Init	
IO_VOUT.h	357
IO_VOUT_ResetProtection	
IO_VOUT_ResetProtection	
IO_VOUT.h	357
IO_VOUT_SetVoltage	
IO_VOUT_SetVoltage	
IO_VOUT.h	358
IO_WAKEUP	

IO_WAKEUP	
IO_Pin.h	279
 — L —	
LegacyApplicationCRC	
LegacyApplicationCRC	
bl_apdb_.....	18
LegacyHeaderCRC	
LegacyHeaderCRC	
bl_apdb_.....	18
length	
length	
io_can_data_frame_.....	23
io_lin_data_frame_.....	28
low_side_channel	
low_side_channel	
io_do_safety_conf_.....	25
io_pwm_safety_conf_.....	36
low_thresh1	
low_thresh1	
io_dio_limits_.....	24
low_thresh2	
low_thresh2	
io_dio_limits_.....	25
 — M —	
MagicSeed	
MagicSeed	
bl_apdb_.....	18
MainAddress	
MainAddress	
bl_apdb_.....	18
ManufacturerID	
ManufacturerID	
bl_apdb_.....	18
mode	
mode	
io_pwd_cnt_conf_.....	29
io_pwd_inc_conf_.....	32
 — N —	
NodeNumber	
NodeNumber	
bl_apdb_.....	18
NodeType	
NodeType	
bl_apdb_.....	19
notify_callback	
notify_callback	
io_driver_safety_conf_.....	27
NULL	
NULL	
ptypes_tms570.h.....	363
NULL_PTR	
NULL_PTR	
ptypes_tms570.h.....	363
 — O —	
overrun	
overrun	
io_pwm_current_queue_.....	35
 — P —	
Password	
Password	
bl_apdb_.....	19
ptypes_apdb.h	
ptypes_apdb.h.....	359
_c_int00	361
APPL_START	361
RTS_TTC_FLASH_DATE_DAY.....	361
RTS_TTC_FLASH_DATE_HOUR.....	361
RTS_TTC_FLASH_DATE_MINUTE ..	361
RTS_TTC_FLASH_DATE_MONTH..	361
RTS_TTC_FLASH_DATE_YEAR....	361
ptypes_tms570.h	
ptypes_tms570.h	361
BOOL.....	364
bool.....	364
FALSE	363
float4	365
float8	365
NULL	363
NULL_PTR	363
sbyte1	365
sbyte2	365
sbyte4	365
sbyte8	365
TRUE	363
TT_FALSE.....	363
TT_TRUE.....	363
ubyte1	365
UBYTE1_MAX_VALUE	364
UBYTE1_MIN_VALUE	364
ubyte2	365

UBYTE2_MAX_VALUE	364	bl_apdb_	19
UBYTE2_MIN_VALUE	364	reset_behavior	
ubyte4	366	reset_behavior	
UBYTE4_MAX_VALUE	364	io_driver_safety_conf_	27
UBYTE4_MIN_VALUE	364	RTS_TTC_FLASH_DATE_DAY	
ubyte8	366	RTS_TTC_FLASH_DATE_DAY	
pulse_mode		ptypes_apdb.h	361
pulse_mode		RTS_TTC_FLASH_DATE_HOUR	
io_pwd_cplx_conf_	30	RTS_TTC_FLASH_DATE_HOUR	
pulse_sample		ptypes_apdb.h	361
pulse_sample		RTS_TTC_FLASH_DATE_MINUTE	
io_pwd_pulse_samples_	33	RTS_TTC_FLASH_DATE_MINUTE	
pulse_samples_count		ptypes_apdb.h	361
pulse_samples_count		RTS_TTC_FLASH_DATE_MONTH	
io_pwd_pulse_samples_	33	RTS_TTC_FLASH_DATE_MONTH	
pwd_cnt_safety_conf		ptypes_apdb.h	361
pwd_cnt_safety_conf		RTS_TTC_FLASH_DATE_YEAR	
io_pwd_universal_safety_conf_	34	RTS_TTC_FLASH_DATE_YEAR	
pwd_cnt_val_lower		ptypes_apdb.h	361
pwd_cnt_val_lower			
io_pwd_inc_safety_conf_	33		
pwd_cnt_val_upper			
pwd_cnt_val_upper			
io_pwd_inc_safety_conf_	33		
pwd_cplx_safety_conf			
pwd_cplx_safety_conf			
io_pwd_universal_safety_conf_	34		
pwd_freq_val_lower			
pwd_freq_val_lower			
io_pwd_cplx_safety_conf_	31		
pwd_freq_val_upper			
pwd_freq_val_upper			
io_pwd_cplx_safety_conf_	31		
pwd_inc_safety_conf			
pwd_inc_safety_conf			
io_pwd_universal_safety_conf_	35		
pwd_pulse_val_lower			
pwd_pulse_val_lower			
io_pwd_cplx_safety_conf_	31		
pwd_pulse_val_upper			
pwd_pulse_val_upper			
io_pwd_cplx_safety_conf_	31		
— R —			
redundant_channel			
redundant_channel			
io_adc_safety_conf_	22		
Reserved			
Reserved			
		— S —	
SAFETY_CONF_RESETS_1			
SAFETY_CONF_RESETS_1			
IO_Driver.h	142		
SAFETY_CONF_RESETS_2			
SAFETY_CONF_RESETS_2			
IO_Driver.h	142		
SAFETY_CONF_RESETS_3			
SAFETY_CONF_RESETS_3			
IO_Driver.h	142		
SAFETY_CONF_RESETS_4			
SAFETY_CONF_RESETS_4			
IO_Driver.h	142		
SAFETY_CONF_RESETS_5			
SAFETY_CONF_RESETS_5			
IO_Driver.h	142		
SAFETY_CONF_RESETS_6			
SAFETY_CONF_RESETS_6			
IO_Driver.h	143		
SAFETY_CONF_RESETS_7			
SAFETY_CONF_RESETS_7			
IO_Driver.h	143		
SAFETY_CONF_RESETS_8			
SAFETY_CONF_RESETS_8			
IO_Driver.h	143		
SAFETY_CONF_RESETS_9			
SAFETY_CONF_RESETS_9			
IO_Driver.h	143		
SAFETY_CONF_RESETS_DISABLED			

SAFETY_CONF_RESETS_DISABLED		ptypes_tms570.h.....	363																																																																																																																																																																						
IO_Driver.h.....	143																																																																																																																																																																								
SAFETY_CONF_WINDOW_SIZE_100_PERCENT		TT_TRUE																																																																																																																																																																							
SAFETY_CONF_WINDOW_SIZE_100_PERCENT		TT_TRUE																																																																																																																																																																							
IO_Driver.h.....	143	ptypes_tms570.h.....	363																																																																																																																																																																						
SAFETY_CONF_WINDOW_SIZE_12_5_PERCENT																																																																																																																																																																									
SAFETY_CONF_WINDOW_SIZE_12_5_PERCENT		— U —																																																																																																																																																																							
IO_Driver.h.....	144	ubyte1																																																																																																																																																																							
SAFETY_CONF_WINDOW_SIZE_25_PERCENT		ubyte1																																																																																																																																																																							
SAFETY_CONF_WINDOW_SIZE_25_PERCENT		ptypes_tms570.h.....	365																																																																																																																																																																						
IO_Driver.h.....	144	SAFETY_CONF_WINDOW_SIZE_3_125_PERCENT		UBYTE1_MAX_VALUE		SAFETY_CONF_WINDOW_SIZE_3_125_PERCENT		UBYTE1_MAX_VALUE		IO_Driver.h.....	144	ptypes_tms570.h.....	364	SAFETY_CONF_WINDOW_SIZE_50_PERCENT		UBYTE1_MIN_VALUE		SAFETY_CONF_WINDOW_SIZE_50_PERCENT		UBYTE1_MIN_VALUE		IO_Driver.h.....	144	ptypes_tms570.h.....	364	SAFETY_CONF_WINDOW_SIZE_6_25_PERCENT		ubyte2		SAFETY_CONF_WINDOW_SIZE_6_25_PERCENT		ubyte2		IO_Driver.h.....	144	ptypes_tms570.h.....	365	sbyte1		UBYTE2_MAX_VALUE		sbyte1		UBYTE2_MAX_VALUE		ptypes_tms570.h.....	365	ptypes_tms570.h.....	364	sbyte2		UBYTE2_MIN_VALUE		sbyte2		UBYTE2_MIN_VALUE		ptypes_tms570.h.....	365	ptypes_tms570.h.....	364	sbyte4		ubyte4		sbyte4		ubyte4		ptypes_tms570.h.....	365	ptypes_tms570.h.....	366	sbyte8		UBYTE4_MAX_VALUE		sbyte8		UBYTE4_MAX_VALUE		ptypes_tms570.h.....	365	ptypes_tms570.h.....	364	SubnetMask		UBYTE4_MIN_VALUE		SubnetMask		UBYTE4_MIN_VALUE		bl_apdb_.....	19	ptypes_tms570.h.....	364			ubyte8				ubyte8				ptypes_tms570.h.....	366							— V —				values				values				io_pwm_current_queue_.....	35							— W —				window_size				window_size				io_driver_safety_conf_.....	27	TRUE				TRUE				ptypes_tms570.h.....	363			TT_FALSE				TT_FALSE			
SAFETY_CONF_WINDOW_SIZE_3_125_PERCENT		UBYTE1_MAX_VALUE																																																																																																																																																																							
SAFETY_CONF_WINDOW_SIZE_3_125_PERCENT		UBYTE1_MAX_VALUE																																																																																																																																																																							
IO_Driver.h.....	144	ptypes_tms570.h.....	364																																																																																																																																																																						
SAFETY_CONF_WINDOW_SIZE_50_PERCENT		UBYTE1_MIN_VALUE																																																																																																																																																																							
SAFETY_CONF_WINDOW_SIZE_50_PERCENT		UBYTE1_MIN_VALUE																																																																																																																																																																							
IO_Driver.h.....	144	ptypes_tms570.h.....	364																																																																																																																																																																						
SAFETY_CONF_WINDOW_SIZE_6_25_PERCENT		ubyte2																																																																																																																																																																							
SAFETY_CONF_WINDOW_SIZE_6_25_PERCENT		ubyte2																																																																																																																																																																							
IO_Driver.h.....	144	ptypes_tms570.h.....	365																																																																																																																																																																						
sbyte1		UBYTE2_MAX_VALUE																																																																																																																																																																							
sbyte1		UBYTE2_MAX_VALUE																																																																																																																																																																							
ptypes_tms570.h.....	365	ptypes_tms570.h.....	364																																																																																																																																																																						
sbyte2		UBYTE2_MIN_VALUE																																																																																																																																																																							
sbyte2		UBYTE2_MIN_VALUE																																																																																																																																																																							
ptypes_tms570.h.....	365	ptypes_tms570.h.....	364																																																																																																																																																																						
sbyte4		ubyte4																																																																																																																																																																							
sbyte4		ubyte4																																																																																																																																																																							
ptypes_tms570.h.....	365	ptypes_tms570.h.....	366																																																																																																																																																																						
sbyte8		UBYTE4_MAX_VALUE																																																																																																																																																																							
sbyte8		UBYTE4_MAX_VALUE																																																																																																																																																																							
ptypes_tms570.h.....	365	ptypes_tms570.h.....	364																																																																																																																																																																						
SubnetMask		UBYTE4_MIN_VALUE																																																																																																																																																																							
SubnetMask		UBYTE4_MIN_VALUE																																																																																																																																																																							
bl_apdb_.....	19	ptypes_tms570.h.....	364																																																																																																																																																																						
		ubyte8																																																																																																																																																																							
		ubyte8																																																																																																																																																																							
		ptypes_tms570.h.....	366																																																																																																																																																																						
		— V —																																																																																																																																																																							
		values																																																																																																																																																																							
		values																																																																																																																																																																							
		io_pwm_current_queue_.....	35																																																																																																																																																																						
		— W —																																																																																																																																																																							
		window_size																																																																																																																																																																							
		window_size																																																																																																																																																																							
		io_driver_safety_conf_.....	27																																																																																																																																																																						
TRUE																																																																																																																																																																									
TRUE																																																																																																																																																																									
ptypes_tms570.h.....	363																																																																																																																																																																								
TT_FALSE																																																																																																																																																																									
TT_FALSE																																																																																																																																																																									

Disposal

NOTICE

If disposal has to be undertaken after the life span of the device, the respective applicable country-specific regulations are to be observed.

Legal Disclaimer

THE INFORMATION GIVEN IN THIS DOCUMENT IS GIVEN AS A SUPPORT FOR THE USAGE OF THE PRODUCT AND SHALL NOT BE REGARDED AS ANY DESCRIPTION OR WARRANTY OF A CERTAIN FUNCTIONALITY, CONDITION OR QUALITY OF THE PRODUCT. THE RECIPIENT OF THIS DOCUMENT MUST VERIFY ANY FUNCTION DESCRIBED HEREIN IN THE REAL APPLICATION. THIS DOCUMENT WAS MADE TO THE BEST OF KNOWLEDGE OF TTCONTROL GMBH. NEVERTHELESS AND DESPITE GREATEST CARE, IT CANNOT BE EXCLUDED THAT MISTAKES COULD HAVE CREPT IN. TTCONTROL GMBH PROVIDES THE DOCUMENT FOR THE PRODUCT "AS IS" AND WITH ALL FAULTS AND HEREBY DISCLAIMS ALL WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ACCURACY OR COMPLETENESS, OR OF RESULTS TO THE EXTENT PERMITTED BY APPLICABLE LAW. THE ENTIRE RISK, AS TO THE QUALITY, USE OR PERFORMANCE OF THE DOCUMENT, REMAINS WITH THE RECIPIENT. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW TTCONTROL GMBH SHALL IN NO EVENT BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING BUT NOT LIMITED TO LOSS OF DATA, DATA BEING RENDERED INACCURATE, BUSINESS INTERRUPTION OR ANY OTHER PECUNIARY OR OTHER LOSS WHATSOEVER) ARISING OUT OF THE USE OR INABILITY TO USE THE DOCUMENT EVEN IF TTCONTROL GMBH HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

IF THE PRODUCT IS MARKED AS "PROTOTYPE", THE DELIVERED PRODUCT IS A DEVELOPMENT SAMPLE ("SAMPLE"). THE RECIPIENT ACKNOWLEDGES THAT THEY ARE ALLOWED TO USE THE SAMPLE ONLY IN A LABORATORY FOR THE PURPOSE OF DEVELOPMENT. IN NO EVENT IS THE RECIPIENT ALLOWED TO USE THE SAMPLE FOR THE PURPOSE OF SERIES MANUFACTURING.

TTCONTROL GMBH PROVIDES NO WARRANTY FOR ITS PRODUCTS OR ITS SAMPLES, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW DISCLAIMS ALL LIABILITIES FOR DAMAGES RESULTING FROM OR ARISING OUT OF THE APPLICATION OR USE OF THESE PRODUCTS OR SAMPLES. THE EXCLUSION OF LIABILITY DOES NOT APPLY IN CASES OF INTENT AND GROSS NEGLIGENCE. MOREOVER, IT DOES NOT APPLY TO DEFECTS WHICH HAVE BEEN DECEITFULLY CONCEALED OR WHOSE ABSENCE HAS BEEN GUARANTEED, NOR IN CASES OF CULPABLE HARM TO LIFE, PHYSICAL INJURY AND DAMAGE TO HEALTH. CLAIMS DUE TO STATUTORY PROVISIONS OF PRODUCT LIABILITY SHALL REMAIN UNAFFECTED.

ANY DISPUTES ARISING OUT OF OR IN CONNECTION WITH THIS DOCUMENT SHALL BE GOVERNED SOLELY BY AUSTRIAN LAW, EXCLUDING ITS CONFLICT OF LAW RULES AND THE UNITED NATIONS CONVENTION ON CONTRACTS FOR THE INTERNATIONAL SALE OF GOODS. SUCH DISPUTES SHALL BE DECIDED EXCLUSIVELY BY THE COURTS OF VIENNA, AUSTRIA.

ALL PRODUCT NAMES AND TRADEMARKS MENTIONED IN THIS I/O DRIVER MANUAL ARE THE PROPERTY OF THEIR RESPECTIVE OWNERS, WHICH ARE IN NO WAY ASSOCIATED OR AFFILIATED WITH TTCONTROL GMBH.