DIGITAL_IO

Here is a list of all modules:

- License Terms and Copyright Information
- Abbreviations and Definitions
- Overview
- Architecture Description
- APP Configuration Parameters
- Enumerations
- Data structures
- Methods
- Usage
- Release History
License Terms and Copyright Information

Copyright (c) 2015, Infineon Technologies AG All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of the copyright holders nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

To improve the quality of the software, users are encouraged to share modifications, enhancements or bug fixes with Infineon Technologies AG (dave@infineon.com).
## Abbreviations and Definitions

### Abbreviations:

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAVE™</td>
<td>Digital Application Virtual Engineer</td>
</tr>
<tr>
<td>APP</td>
<td>DAVE™ Application</td>
</tr>
<tr>
<td>API</td>
<td>Application Program Interface</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>MCU</td>
<td>Microcontroller Unit</td>
</tr>
<tr>
<td>SW</td>
<td>Software</td>
</tr>
<tr>
<td>HW</td>
<td>Hardware</td>
</tr>
<tr>
<td>LLD</td>
<td>Low Level Driver</td>
</tr>
<tr>
<td>IO</td>
<td>Input Output</td>
</tr>
<tr>
<td>GPIO</td>
<td>General Purpose Input Output</td>
</tr>
</tbody>
</table>

### Definitions:

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Singleton</td>
<td>Only single instance of the APP is permitted</td>
</tr>
<tr>
<td>Sharable</td>
<td>Resource sharing with other APPs permitted</td>
</tr>
<tr>
<td>initProvider</td>
<td>Provides the initialization routine</td>
</tr>
<tr>
<td>Physical connectivity</td>
<td>Hardware inter/intra peripheral (constant) signal connection</td>
</tr>
<tr>
<td>Conditional connectivity</td>
<td>Constrained hardware inter/intra peripheral signal connection</td>
</tr>
<tr>
<td>Aggregation</td>
<td>Indicates consumption of low level (dependent) DAVE™ APPs</td>
</tr>
</tbody>
</table>
The DIGITAL_IO APP provides the following functionalities:

- Configure and control a port pin as general purpose Input or Input/Output.
- Controls various IO pad characteristics.
- Connection of port pin to other peripherals (using the DAVE™ HW Signal Connections).
The red marked registers shown in above figures are modified by the DIGITAL_IO APP.

**Input Operation:**
Reads actual voltage level at the port pin and it can be inverted optionally.

- Configures internal weak pull-up or pull down devices in the pad.
- Configures input pad hysteresis.

**Input/Output Operation:**
The output driver is activated and drives the value supplied through the output multiplexer to the port pin.
- Configures between open-drain and push-pull.
- Configures pad driver strength.
- Please refer APP Configuration Parameters section for more information.

**Hardware Controlled operation:**
The pin will be controlled by the peripheral to which it is connected.

- Configures of HWSEL bitfield which selects the peripheral that drives the pin.
- Configures pad driver strength.
- Configures input pad hysteresis.
- Configures pin input data polarity.

Additionally **DIGITAL_IO** APP provides APIs to configure port pin.

Figure 3, shows how the APP is structured in DAVE™. XMC controllers provide the PORT module to configure pin. The LLD layer provides abstraction for these hardware modules. The **DIGITAL_IO** APP uses XMC_GPIO LLDs for the functionality.

![Diagram](image)

**Figure 3:** Hardware and Software connectivity of **DIGITAL_IO** APP

**Note:** This APP is intended to configure digital I/O ports, however for analog I/O operation refer **ANALOG_IO** APP.

**Supported Devices**
The APP supports below devices:

1. XMC4800 / XMC4700 / XMC4300 Series
2. XMC4500 Series
3. XMC4400 Series
4. XMC4200 / XMC4100 Series
5. XMC1400 Series
6. XMC1300 Series
7. XMC1200 Series
8. XMC1100 Series

References:

1. XMC4800 / XMC4700 / XMC4300 Reference Manual
2. XMC4500 Reference Manual
5. XMC1400 Reference Manual
6. XMC1300 Reference Manual

Limitations
None.
The above diagram represents the internal software architecture of the DIGITAL_IO APP. A DIGITAL_IO APP instance exists in a DAVE™ project with fixed attributes as shown. Each instance of this APP configures PORT pin in the MCU. The DIGITAL_IO APP also provides output signal for inter-peripheral connections.

An instantiated APP (after code generation) generates a specific data structure with the GUI configuration. The name of this data structure can be modified by changing the APP instance label (e.g. change label from default DIGITAL_IO_0 to INPUT_READ).
Signals:

The following table presents the signals provided by the APP for connection. It also gives the flexibility to configure and extend the connectivity to other APPs.

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Input/Output</th>
<th>Availability</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pin</td>
<td>Input/Output</td>
<td>When pin direction is <em>Input</em> or <em>Input/Output</em></td>
<td>GPIO functionality.</td>
</tr>
<tr>
<td>hardware pin control</td>
<td>Input/Output</td>
<td>When pin direction is <em>Hardware Controlled</em></td>
<td>Peripheral controlled functionality.</td>
</tr>
</tbody>
</table>
App Configuration Parameters

Figure 1: General Settings
### DIGITAL_IO

#### Enumerations

<table>
<thead>
<tr>
<th>Type</th>
<th>Declaration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>enum</code></td>
<td><code>DIGITAL_IO_STATUS</code> { <code>DIGITAL_IO_STATUS_OK</code> = 0U, <code>DIGITAL_IO_STATUS_FAILURE</code> = 1U }</td>
<td>Initialization status of <code>DIGITAL_IO</code> APP. More...</td>
</tr>
<tr>
<td><code>typedef enum</code></td>
<td><code>DIGITAL_IO_STATUS</code> <code>DIGITAL_IO_STATUS_t</code></td>
<td>Initialization status of <code>DIGITAL_IO</code> APP.</td>
</tr>
</tbody>
</table>
Enumeration Type Documentation

```c
enum DIGITAL_IO_STATUS

Initialization status of DIGITAL_IO APP.

Enumerator:

```
DIGITAL_IO_STATUS_OK           0=Status OK

DIGITAL_IO_STATUS_FAILURE      1=Status Failed
```

Definition at line 91 of file DIGITAL_IO.h.
### DIGITAL_IO

<table>
<thead>
<tr>
<th>Home</th>
<th>Data Structures</th>
</tr>
</thead>
</table>

#### Data structures
Data Structures

<table>
<thead>
<tr>
<th>struct</th>
<th>DIGITAL_IO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization data structure of DIGITAL_IO APP.</td>
<td>More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>typedef struct</th>
<th>DIGITAL_IO</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIGITAL_IO_t</td>
<td>Initialization data structure of DIGITAL_IO APP.</td>
</tr>
</tbody>
</table>
## DIGITAL_IO

### Methods

<table>
<thead>
<tr>
<th>Method Type</th>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAVE_APP_VERSION_t</td>
<td>DIGITAL_IO_GetAppVersion (void)</td>
<td>Get DIGITAL_IO APP version.</td>
</tr>
<tr>
<td>DIGITAL_IO_STATUS_t</td>
<td>DIGITAL_IO_Init (const DIGITAL_IO_t *const handler)</td>
<td>Function to initialize the port pin as per UI settings.</td>
</tr>
<tr>
<td>__STATIC_INLINE void</td>
<td>DIGITAL_IO_SetOutputHigh (const DIGITAL_IO_t *const handler)</td>
<td>Function to set port pin high.</td>
</tr>
<tr>
<td>__STATIC_INLINE void</td>
<td>DIGITAL_IO_SetOutputLow (const DIGITAL_IO_t *const handler)</td>
<td>Function to reset port pin.</td>
</tr>
<tr>
<td>__STATIC_INLINE void</td>
<td>DIGITAL_IO_ToggleOutput (const DIGITAL_IO_t *const handler)</td>
<td>Function to Toggle port pin.</td>
</tr>
<tr>
<td>__STATIC_INLINE uint32_t</td>
<td>DIGITAL_IO_GetInput (const DIGITAL_IO_t *const handler)</td>
<td>Function to read input level of port pin.</td>
</tr>
</tbody>
</table>
Function Documentation

DAVE_APP_VERSION_t DIGITAL_IO_GetAppVersion( void )

Get DIGITAL_IO APP version.

Returns:
DAVE_APP_VERSION_t APP version information (major, minor and patch number)

Description:
The function can be used to check application software compatibility with a specific version of the APP.

Example Usage:

```
#include <DAVE.h>

int main(void)
{
    DAVE_STATUS_t init_status;
    DAVE_APP_VERSION_t version;

    // Initialize DIGITAL_IO APP:
    // DIGITAL_IO_Init() is called from within DAVE_Init().
    init_status = DAVE_Init();
    if(init_status == DAVE_STATUS_SUCCESS)
    {
        version = DIGITAL_IO_GetAppVersion();
        if (version.major != 4U) {
            // Probably, not the right version.
        }
    }

    // More code here
```
while(1) {

} 
return (1);

Returns:
DAVE_APP_VERSION_t APP version information (major, minor and patch number)

Definition at line 77 of file DIGITAL_IO.c.

__STATIC_INLINE uint32_t DIGITAL_IO_GetInput ( const DIGITAL_IO_t 

Function to read input level of port pin.

Parameters:
handler Pointer pointing to APP data structure. Refer DIGITAL_IO_t for details.

Returns:
uint32_t input logic level. Range:0-1

Description:
This function reads the Pn_IN register and returns the current logical value at the GPIO pin.

Related APIs:
None

Example Usage:
#include <DAVE.h>//Declarations from DAVE Code Generation (includes SFR declaration)
int main(void)
{
    DAVE_STATUS_t status;
    uint32_t pin_status;
    status = DAVE_Init();  // (DAVE_STATUS_t)DIGITAL_IO_Init(&DIGITAL_IO_0) is called within DAVE_Init()
    if(status == DAVE_STATUS_SUCCESS)
    {
        XMC_DEBUG("DAVE Apps initialization success\n");
    }
    else
    {
        XMC_DEBUG("DAVE Apps initialization failed with status %d\n", status);
        while(1U)
        {
        }
    }
    // Placeholder for user application code. The while loop below can be replaced with user application code.
    while(1U)
    {
        pin_status = DIGITAL_IO_GetInput(&DIGITAL_IO_0);
        if(pin_status == 1)
        {
            // Add application code here
        }
        else
        {
            // Add application code here
        }
        return (1);
    }
}
**DIGITAL_IO_STATUS_t DIGITAL_IO_Init(const DIGITAL_IO_t *const handler)**

Function to initialize the port pin as per UI settings.

**Parameters:**
- **handler**  Pointer pointing to APP data structure. Refer [DIGITAL_IO_t](#) for details.

**Returns:**
- DIGITAL_IO_STATUS_t DIGITAL_IO APP status. Refer [DIGITAL_IO_STATUS_t](#) structure for details.

**Description:**
This function initializes GPIO port registers IOCR,PDISC,OMR,PDR/PHCR to configure pin direction, initial output level, and pad driver strength/hysteresis.

**Related APIs:**
 None

**Example Usage:**

```c
#include <DAVE.h>  //Declarations from DAVE Code Generation (includes SFR declaration)

int main(void)
{
  DAVE_STATUS_t status;
  status = DAVE_Init();  //DAVE_STATUS_t)DIGITAL_IO_Init(&DIGITAL_IO_0) is called within DAVE_Init()
  if(status == DAVE_STATUS_SUCCESS)
  {
    XMC_DEBUG("DAVE Apps initialization success\n    
    
```
Parameters:

**handler** Pointer pointing to APP data structure.

Returns:

**DIGITAL_IO APP status**.

Definition at line 95 of file **DIGITAL_IO.c**.

References **DIGITAL_IO_STATUS_OK**, **DIGITAL_IO::gpio_config**, **DIGITAL_IO::gpio_pin**, **DIGITAL_IO::gpio_port**, and **DIGITAL_IO::hwctrl**.

```c
__STATIC_INLINE void DIGITAL_IO_SetOutputHigh ( const DIGITAL
```

Function to set port pin high.

**Parameters:**
**(handler** Pointer pointing to APP data structure. Refer **DIGITAL_IO_t** for details.

**Returns:**
None

**Description:**
This function configures port output modification register Pn_OMR, to make port pin to high level.

**Related APIs:**
**DIGITAL_IO_SetOutputLow()**

**Example Usage:**

```c
#include <DAVE.h> //Declarations from DAVE Code Generation (includes SFR declaration)
int main(void)
{
    DAVE_STATUS_t status;
    status = DAVE_Init(); // (DAVE_STATUS_t)DIGITAL_IO_Init(&DIGITAL_IO_0) is called within DAVE_Init()
    if(status == DAVE_STATUS_SUCCESS)
    {
        XMC_DEBUG("DAVE Apps initialization success\n");
    }
    else
    {
        XMC_DEBUG("DAVE Apps initialization failed with status %d\n", status);
        while(1U)
        {
        }
    }
    //Placeholder for user application code. The while loop below can be replaced with user applica
```
DIGITAL_IO_SetOutputHigh(&DIGITAL_IO_0);
while(1U)
{
    // Add application code here
}

return (1);

Definition at line 266 of file DIGITAL_IO.h.

References DIGITAL_IO::gpio_pin, and DIGITAL_IO::gpio_port.

__STATIC_INLINE void DIGITAL_IO_SetOutputLow (const DIGITAL_IO_t handler)

Function to reset port pin.

Parameters:
    handler Pointer pointing to APP data structure. Refer DIGITAL_IO_t for details.

Returns:
    None

Description:
    This function configures port output modification register Pn_OMR, to make port pin to low level.

Related APIs:
    DIGITAL_IO_SetOutputHigh()

Example Usage:

```
#include <DAVE.h>//Declarations from DAVE Code Generation (includes SFR declaration)
int main(void)
```
{  
    DAVE_STATUS_t status;
    status = DAVE_Init();  // (DAVE_STATUS_t) DIGITAL_IO_Init(&DIGITAL_IO_0) is called within DAVE_Init()
        if(status == DAVE_STATUS_SUCCESS)
            {
                XMC_DEBUG("DAVE Apps initialization success \n");
            }
        else
            {
                XMC_DEBUG("DAVE Apps initialization failed with status %d\n", status);
                while(1U)
                    {
                        
                    }
            }
    // Placeholder for user application code. The while loop below can be replaced with user application code.
    DIGITAL_IO_SetOutputLow(&DIGITAL_IO_0);
    while(1U)
        {
            // Add application code here
       }

    return (1);
}

Definition at line 313 of file DIGITAL_IO.h.

References DIGITAL_IO::gpio_pin, and DIGITAL_IO::gpio_port.

__STATIC_INLINE void DIGITAL_IO_ToggleOutput ( const DIGITAL_
Function to Toggle port pin.

**Parameters:**
- **handler** Pointer pointing to APP data structure. Refer `DIGITAL_IO_t` for details.

**Returns:**
None

**Description:**
This function configures port output modification register Pn_OMR, to toggle port pin.

**Related APIs:**
- `DIGITAL_IO_SetOutputLow()`, `DIGITAL_IO_SetOutputHigh()`

**Example Usage:**
```c
#include <DAVE.h> // Declarations from DAVE Code Generation (includes SFR declaration)
int main(void)
{
    DAVE_STATUS_t status;
    uint32_t delay_count;
    status = DAVE_Init();  //=DAVE_STATUS_t)DIGITAL_IO_Init(&DIGITAL_IO_0) is called within DAVE_Init()
    if(status == DAVE_STATUS_SUCCESS)
    {
        XMC_DEBUG("DAVE Apps initialization success\n");
    }
    else
    {
        XMC_DEBUG("DAVE Apps initialization failed with status %d\n", status));
        while(1U)
```
{ 
} 

// Placeholder for user application code. The while loop below can be replaced with user application code.
while (1U) {
    DIGITAL_IO_ToggleOutput(&DIGITAL_IO_0); // toggles : 1 -> 0 (if initial output level is logic 1)
    // Add application code here
    for (delay_count = 0; delay_count < 0xffffffff; delay_count++) {
        DIGITAL_IO_ToggleOutput(&DIGITAL_IO_0); // toggles : 0 -> 1
        // Add application code here
        for (delay_count = 0; delay_count < 0xffffffff; delay_count++) {
            return (1);
        }
    }
}

Definition at line 365 of file DIGITAL_IO.h.

References DIGITAL_IO::gpio_pin, and DIGITAL_IO::gpio_port.
DIGITAL_IO

Usage

The below example demonstrates LED Toggling.

Hardware requirement:

- XMC4000 Application Kit : CPU_45A-V3(CPU Board XMC4500 General Purpose).

**Step 1:** Instantiate APP and select Pin direction as Input/Output.
**Step 2:** Select *Mode* Push-pull.
**Step 3:** Select *Initial output level* to Low.
**Step 4:** Select Desired *Driver strength* to Weak Driver.(only available for XMC4X device).
**Step 5:** Select pin by using *Manual Pin Allocator.*(Figure shows for XMC4500 device).

*Figure 1:* Deployment of DIGITAL_IO APP for output mode
Generate code
Files are generated here: `\<project_name>/Dave/Generated/\` (`\project_name\` is the name chosen by the user during project creation). APP instance definitions and APIs are generated only after code generation.
**Note:** Code must be explicitly generated for every change in the GUI configuration.
**Important:** Any manual modification to APP specific files will be overwritten by a subsequent code generation operation.

Sample Application (main.c)

```c
#include <DAVE.h> //Declarations from DAVE Code Generation (includes SFR declaration)
int main(void)
{
    DAVE_STATUS_t status;
    uint32_t delay_count;
    status = DAVE_Init(); //((DAVE_STATUS_t)DIGITAL_IO_Init(&DIGITAL_IO_0) is called within DAVE_Init()
    if(status == DAVE_STATUS_SUCCESS)
    {
```
XMC_DEBUG("DAVE Apps initialization success\n");
}
else
{
    XMC_DEBUG("DAVE Apps initialization failed with status %d\n", status);
    while(1U)
    {
    }
}
//Placeholder for user application code. The while loop below can be replaced with user application code.
while(1U)
{
    DIGITAL_IO_ToggleOutput(&DIGITAL_IO_0); //toggles level at pin
    for(delay_count = 0; delay_count<0xffffffff; delay_count++);
    }
return 1U;
}

Build and Run the Project

Observation:
Observe LED1(P3.9) is blinking.
DIGITAL_IO

Data Structures

Here are the data structures with brief descriptions:

| DIGITAL_IO | Initialization data structure of DIGITAL_IO APP |

---
### DIGITAL_IO

<table>
<thead>
<tr>
<th>Home</th>
<th>Data Structures</th>
<th>Data Structure Index</th>
<th>Data Fields</th>
<th>Data Fields</th>
</tr>
</thead>
</table>

**DIGITAL_IO Struct Reference**

Data structures
Detailed Description

Initialization data structure of **DIGITAL_IO** APP.

Definition at line 112 of file **DIGITAL_IO.h**.

#include <DIGITAL_IO.h>
**Data Fields**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>XMC_GPIO_PORT_t *const</td>
<td><strong>gpio_port</strong></td>
<td></td>
</tr>
<tr>
<td>const XMC_GPIO_CONFIG_t</td>
<td><strong>gpio_config</strong></td>
<td></td>
</tr>
<tr>
<td>const uint8_t</td>
<td><strong>gpio_pin</strong></td>
<td></td>
</tr>
<tr>
<td>const XMC_GPIO_HWCTRL_t</td>
<td><strong>hwctrl</strong></td>
<td></td>
</tr>
</tbody>
</table>
### Field Documentation

**const XMC_GPIO_CONFIG_t** `DIGITAL_IO::gpio_config`  

Mode, initial output level and pad driver strength / hysteresis  
Definition at line 115 of file `DIGITAL_IO.h`.  
Referenced by `DIGITAL_IO_Init()`.  

**const uint8_t** `DIGITAL_IO::gpio_pin`  

Pin number  
Definition at line 116 of file `DIGITAL_IO.h`.  
Referenced by `DIGITAL_IO_GetInput()`, `DIGITAL_IO_Init()`, `DIGITAL_IO_SetOutputHigh()`, `DIGITAL_IO_SetOutputLow()`, and `DIGITAL_IO_ToggleOutput()`.  

**XMC_GPIO_PORT_t** `const DIGITAL_IO::gpio_port`  

Port number  
Definition at line 114 of file `DIGITAL_IO.h`.  
Referenced by `DIGITAL_IO_GetInput()`, `DIGITAL_IO_Init()`, `DIGITAL_IO_SetOutputHigh()`, `DIGITAL_IO_SetOutputLow()`, and `DIGITAL_IO_ToggleOutput()`.  

**const XMC_GPIO_HWCTRL_t** `DIGITAL_IO::hwctrl`  

Hardware port control
Definition at line 117 of file DIGITAL_IO.h.

Referenced by DIGITAL_IO_Init().

The documentation for this struct was generated from the following file:

- DIGITAL_IO.h
# DIGITAL_IO

<table>
<thead>
<tr>
<th>Home</th>
<th>Data Structures</th>
<th>Data Structure Index</th>
<th>Data Fields</th>
</tr>
</thead>
</table>

## Data Structure Index

| D |

<table>
<thead>
<tr>
<th>DIGITAL_IO</th>
</tr>
</thead>
</table>

| D |

---
Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- gpio_config : DIGITAL_IO
- gpio_pin : DIGITAL_IO
- gpio_port : DIGITAL_IO
- hwctrl : DIGITAL_IO
DIGITAL_IO

- gpio_config : DIGITAL_IO
- gpio_pin : DIGITAL_IO
- gpio_port : DIGITAL_IO
- hwctrl : DIGITAL_IO
Here is a list of all documented files with brief descriptions:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>DIGITAL_IO.c</code></td>
<td>[code]</td>
</tr>
<tr>
<td><code>DIGITAL_IO.h</code></td>
<td>[code]</td>
</tr>
</tbody>
</table>
## DIGITAL_IO

<table>
<thead>
<tr>
<th>Home</th>
<th>File List</th>
<th>Globals</th>
<th>Functions</th>
</tr>
</thead>
</table>

**DIGITAL_IO.c File Reference**
Detailed Description

Date:
   2015-08-25

NOTE: This file is generated by DAVE. Any manual modification done to this file will be lost when the code is regenerated.

Definition in file DIGITAL_IO.c.

#include "digital_io.h"
### Functions

<table>
<thead>
<tr>
<th>Type</th>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>DAVE_APP_VERSION_t</code></td>
<td><code>DIGITAL_IO_GetAppVersion</code></td>
<td>(void) Get <code>DIGITAL_IO</code> APP version.</td>
</tr>
<tr>
<td><code>DIGITAL_IO_STATUS_t</code></td>
<td><code>DIGITAL_IO_Init</code></td>
<td>(const <code>DIGITAL_IO_t</code> *const handler) Function to initialize the port pin as per UI settings.</td>
</tr>
</tbody>
</table>
Function Documentation

DIGITAL_IO_STATUS_t DIGITAL_IO_Init (const DIGITAL_IO_t *const handler)

Function to initialize the port pin as per UI settings.

Parameters:
   handler Pointer pointing to APP data structure.

Returns:
   DIGITAL_IO_STATUS_t DIGITAL_IO APP status.

Definition at line 95 of file DIGITAL_IO.c.

References DIGITAL_IO_STATUS_OK, DIGITAL_IO::gpio_config, DIGITAL_IO::gpio_pin, DIGITAL_IO::gpio_port, and DIGITAL_IO::hwctrl.

Go to the source code of this file.
Here is a list of all documented functions, variables, defines, enums, and typedefs with links to the documentation:

- DIGITAL_IO_GetAppVersion() : [DIGITAL_IO.c](DIGITAL_IO.c)
- DIGITAL_IO_Init() : [DIGITAL_IO.c](DIGITAL_IO.c)
DIGITAL_IO

- DIGITAL_IO_GetAppVersion() : DIGITAL_IO.c
- DIGITAL_IO_Init() : DIGITAL_IO.c
/*
 * @file digital_io.h
 * @date 2015-12-22
 *
 * NOTE:
 * This file is generated by DAVE. Any manual modification done to this file will be lost when
 * the code is regenerated.
 *
 * DIGITAL_IO v4.0.14 - The DIGITAL_IO APP is used to configure a port pin as digital Input/Output.
 *
 * Copyright (c) 2015, Infineon Technologies AG
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without modification, are permitted
 * provided that the following conditions are met:
 *
 * Redistributions of source code must retain
ain the above copyright notice, this list of conditions and the following

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of the copyright holders nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
To improve the quality of the software, users are encouraged to share modifications, enhancements or bug fixes with Infineon Technologies AG (dave@infineon.com).

Change History
--------------
2015-02-16 - Initial version
2015-04-22 - XMC_ASSERT is added in static inline functions.<br>
2015-06-20 - Version check added for XMCLib dependency.<br>
2015-12-22 - Added hardware controlled IO feature.
@endcond

HEADER FILES

*/
#ifndef DIGITAL_IO_H
#define DIGITAL_IO_H

#include <xmc_gpio.h>
#include <DAVE_common.h>
#include "digital_io_conf.h"

/*******************************************
**************************************************
*************************
* MACROS
*******************************************
**************************************************
*************************/

#if (!((XMC_LIB_MAJOR_VERSION == 2U) && 
       (XMC_LIB_MINOR_VERSION >= 0U) && 
       (XMC_LIB_PATCH_VERSION >= 0U)))
#error "DIGITAL_IO requires XMC Peripheral Library v2.0.0 or higher"
#endif

/******************************************
**************************************************
**************************
* ENUMS
******************************************
**************************************************
*************************/

typedef enum DIGITAL_IO_STATUS
{ DIGITAL_IO_STATUS_OK = 0U,
  DIGITAL_IO_STATUS_FAILURE = 1U
} DIGITAL_IO_STATUS_t;
typedef struct DIGITAL_IO
{
    XMC_GPIO_PORT_t *const gpio_port;
    const XMC_GPIO_CONFIG_t gpio_config;
    const uint8_t gpio_pin;
    const XMC_GPIO_HWCTRL_t hwctrl;
} DIGITAL_IO_t;

#define __cplusplus
extern "C"
{
#define _cplusplus
    DAVE_APP_VERSION_t DIGITAL_IO_GetAppVersion(void);
}
DIGITAL_IO_STATUS_t DIGITAL_IO_Init(const DIGITAL_IO_t *const handler);

__STATIC_INLINE void DIGITAL_IO_SetOutputHigh(const DIGITAL_IO_t *const handler) {
    XMC_ASSERT("DIGITAL_IO_SetOutputHigh: handler null pointer", handler != NULL);
    XMC_GPIO_SetOutputHigh(handler->gpio_port, handler->gpio_pin);
}

__STATIC_INLINE void DIGITAL_IO_SetOutputLow(const DIGITAL_IO_t *const handler) {
    XMC_ASSERT("DIGITAL_IO_SetOutputLow: handler null pointer", handler != NULL);
    XMC_GPIO_SetOutputLow(handler->gpio_port, handler->gpio_pin);
}

__STATIC_INLINE void DIGITAL_IO_ToggleOutput(const DIGITAL_IO_t *const handler) {
    XMC_ASSERT("DIGITAL_IO_ToggleOutput: handler null pointer", handler != NULL);
    XMC_GPIO_ToggleOutput(handler->gpio_port, handler->gpio_pin);
}

__STATIC_INLINE uint32_t DIGITAL_IO_GetInput(const DIGITAL_IO_t *const handler) {
    XMC_ASSERT("DIGITAL_IO_GetInput: handler null pointer", handler != NULL);
    return XMC_GPIO_GetInput(handler->gpio_port, handler->gpio_pin);
}  
#endif
  
#ifdef __cplusplus
}  
#endif
  
/* Include APP extern file */
#include "digital_ioExtern.h"

#include "digital_ioExtern.h"

#endif /* DIGITAL_IO_H */
Go to the documentation of this file.

```c
#include "digital_io.h"
```
DAVE_APP_VERSION_t DIGITAL_IO_GetAppVersion(void)
{
    DAVE_APP_VERSION_t version;
    version.major = (uint8_t)DIGITAL_IO_MAJOR_VERSION;
    version.minor = (uint8_t)DIGITAL_IO_MINOR_VERSION;
    version.patch = (uint8_t)DIGITAL_IO_PATCH_VERSION;
    return (version);
}

DIGITAL_IO_STATUS_t DIGITAL_IO_Init(const DIGITAL_IO_t *const handler)
{
    XMC_ASSERT("DIGITAL_IO_Init: handler null pointer", handler != NULL);
}
/* Initializes input / output characteristics */

XMC_GPIO_Init(handler->gpio_port, handler->gpio_pin, &handler->gpio_config);

/* Configure hardware port control */
XMC_GPIO_SetHardwareControl(handler->gpio_port, handler->gpio_pin, handler->hwctrl);

return (DIGITAL_IO_STATUS_OK);