



DW APB ARC EM Starter Kit GPIO User Manual

Version 6281-002 February 2015

Copyright Notice and Proprietary Information Notice

Copyright © 2015 Synopsys, Inc. All rights reserved. This software and documentation contain confidential and proprietary information that is the property of Synopsys, Inc. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Synopsys, Inc., or as expressly provided by the license agreement.

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at <http://www.synopsys.com/Company/Pages/Trademarks.aspx>.

All other product or company names may be trademarks of their respective owners.

Synopsys, Inc.
690 E. Middlefield Road
Mountain View, CA 94043
www.synopsys.com

Contents

Customer Support.....	3
Accessing SolvNet.....	3
Contacting the Synopsys Technical Support Center.....	3
1 Introduction	4
Definitions	4
2 DW APB GPIO Driver.....	5
Software Architecture.....	5
Error Codes.....	6
Functions	6
Generic GPIO APIs.....	6
Push-Button Driver APIs	14
LED Driver API's	16
Switch Driver API's	18
Data Types.....	21
dw_gpio_led_init.....	21
dw_gpio_switch_init.....	21
dw_button_init.....	22
dw_button_callback	22
dw_set_interrupt_mode	23
dw_get_interrupt_mode	23
I/O Controllers and I/O Structures.....	24
Push Buttons IOCtl's.....	24
LED Driver IOCtl's	25
Switch Driver IOCtl's.....	26
Default Configuration for DW APB GPIO	27
3 Test Application.....	28
Push-Button Test Application.....	28
LED Test Application	28
Switch Test Application.....	28

Customer support is available through SolvNet online customer support and through contacting the Synopsys Technical Support Center.

Accessing SolvNet

SolvNet includes an electronic knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. SolvNet also gives you access to a wide range of Synopsys online services, which include downloading software, viewing Documentation on the Web, and entering a call to the Support Center.

To access SolvNet:

1. Go to the SolvNet Web page at <http://solvnet.synopsys.com/>.
2. If prompted, enter your user name and password. (If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.)

If you need help using SolvNet, click SolvNet Help in the Support Resources section.

Contacting the Synopsys Technical Support Center

If you have problems, questions, or suggestions, you can contact the Synopsys Technical Support Center in the following ways:

- Open a call to your local support center from the Web by going to <http://solvnet.synopsys.com/> (Synopsys user name and password required), then clicking “Enter a Call to the Support Center.”
- Send an e-mail message to your local support center.
 - E-mail support_center@synopsys.com from within North America.
 - Find other local support center e-mail addresses at <http://www.synopsys.com/Support/GlobalSupportCenters>.
- Telephone your local support center.
 - Call (800) 245-8005 from within the continental United States.
 - Call (650) 584-4200 from Canada.
 - Find other local support center telephone numbers at <http://www.synopsys.com/Support/GlobalSupportCenters>.

1

Introduction

This document describes the usage of the GPIO driver for writing applications. It aims to make the development of embedded applications easier for developers. This document contains the descriptions of all the data types and API functions.

Definitions

Table 1 lists the definitions associated with this product.

Table 1 Definitions

Term	Definition
ARC 700	Family of 32-bit ARC processor cores with MMU support
AOM	Automated Overlay Manager
HLA	High-Level Architecture
MAS	Micro-Architecture Specification
ARChitect™	System-configuration tool developed by ARC. ARChitect can be run from the command line, or as a GUI to provide point-and-click definition of a processor architecture, and generates Verilog source code of the configured design.
HDL	Hardware Description Language — a programming language used to describe the hardware implementation of a system. HDL is usually synthesized to produce a chip (ASIC). Two common HDLs are Verilog and VHDL.
MetaWare	C/C++ development toolkit for ARC processors
ML509	Xilinx FPGA development platform used as reference platform by ARC.
MQX	RTOS supplied by ARC

This chapter describes the GPIO device drivers.

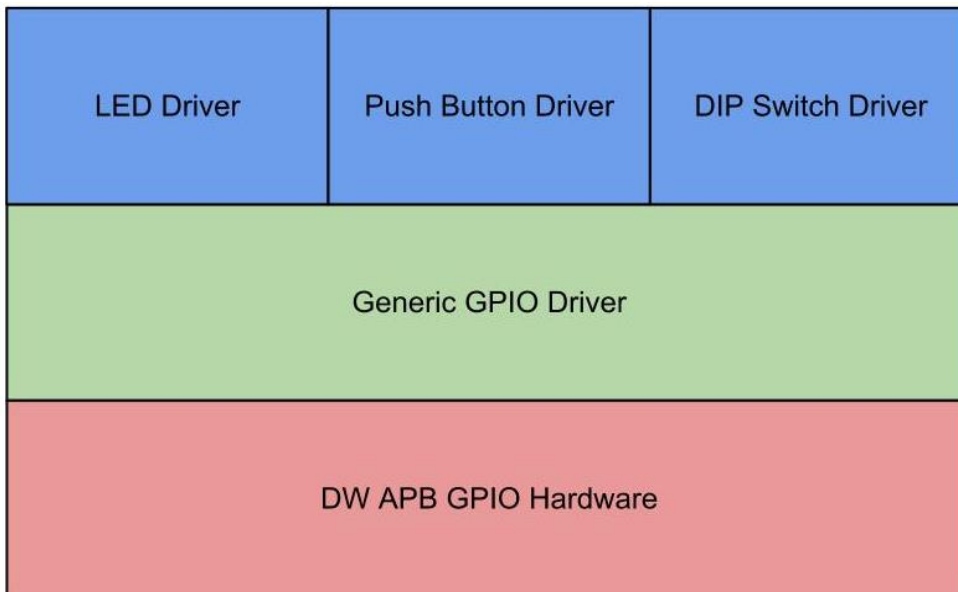
Software Architecture

The DW APB GPIO driver is written using Standard MQX GPIO APIs based on the I/O subsystem. Standard MQX GPIO APIs are responsible for data flow between the I/O subsystem and low-level driver.

All low-level functionality such as data transmission, reception, and interrupts is handled by the low-level driver. Standard MQX GPIO APIs provide support to the low-level driver to set up the data flow.

For more information on the driver structures and members, see the *MQX® RTOS I/O Drivers User's Guide*. Figure 1 shows the overview of the software architecture.

Figure 1 Software Architecture



Error Codes

Table 2 lists the error codes and their descriptions.

Table 2 Error Codes and Description

Error Codes	Description
MQX_OUT_OF_MEMORY	Memory allocation failed
IO_ERROR_INVALID_IOCTL_CMD	Invalid IOCTL
INVALID_INTR_MODE	Unknown interrupt-mode setting

Functions

Generic GPIO APIs

dw_gpio_port_init()

Description This function is used to initialize the port information structure.

Syntax

```
uint_32 dw_gpio_port_init(DW_APB_PORT_INFO_STRUCT_PTR dw_port,
uint_32 port)
```

Parameters `dw_port`
This variable is of type `DW_APB_PORT_INFO_STRUCT_PTR`, and holds the information about the specific GPIO port.

`port`
This variable indicates which port the `dw_port` variable belongs to.

dw_gpio_get_valid_bit_mask()

Description This function gets the valid bit mask of the port. It returns the bit-mask value.

Syntax

```
uint_32 dw_gpio_get_valid_bit_mask(uint_32 port)
```

Parameters `port`
This variable is the port value, and is sent as an input argument to get the valid bit mask of the port. For the `port` variable, use one of the following macros:

- `GPIO_PORT_A`
- `GPIO_PORT_B`
- `GPIO_PORT_C`

dw_gpio_write_data()

Description This writes the specified data to the data register of the port.

Syntax

```
uint_32 dw_gpio_write_data(DW_APB_PORT_INFO_STRUCT_PTR  
dw_port, uint_32 data)
```

Parameters `dw_port`
This variable holds the specific GPIO port information.

`data`
Data register value to be written.

dw_gpio_read_data()

Description This reads the data from data register of a specific GPIO port.

Syntax

```
uint_32 dw_gpio_read_data (DW_APB_PORT_INFO_STRUCT_PTR dw_port,  
uint_32_ptr data)
```

Parameters `dw_port`
This variable holds the specific GPIO port information.

`data`
The data register value read is stored in this pointer.

dw_gpio_read_ext()

Description This function reads the data from the external port register.

Syntax

```
uint_32 dw_gpio_read_ext (DW_APB_PORT_INFO_STRUCT_PTR dw_port,
uint_32_ptr data)
```

Parameters

`dw_port`
This variable holds the specific GPIO port information.

`data`
The port external register value read is stored in this pointer.

dw_gpio_write_direction()

Description This function writes specified data to the direction register of the port.

Syntax

```
uint_32 dw_gpio_write_direction( DW_APB_PORT_INFO_STRUCT_PTR
dw_port, uint_32 data)
```

Parameters

`dw_port`
This variable holds the specific GPIO port information.

`data`
Value to be written to GPIO port's direction register.

dw_gpio_read_direction()

Description This function reads the direction register value of a specific GPIO port.

Syntax

```
uint_32 dw_gpio_read_direction( DW_APB_PORT_INFO_STRUCT_PTR
dw_port, uint_32_ptr data)
```

Parameters

`dw_port`
This variable holds the specific GPIO port information.

`data`
The port direction register value read is stored in this pointer.

dw_gpio_set_bit_direction()

Description Set the direction of a GPIO pins/port.

Syntax

```
uint_32 dw_gpio_set_bit_direction( DW_APB_PORT_INFO_STRUCT_PTR  
dw_port, uint_32 bit_mask, bit_dir)
```

Parameters

`dw_port`
This variable holds the specific GPIO port information.

`bit_mask`
The value of the bit mask indicates the directions in which pins can be altered.

`bit_dir`
Pin direction, if `bit_dir` is set to `GPIO_PORT_OUT`, the pin directions are set to OUTPUT. Otherwise they are set to INPUT.

dw_gpio_set_bits()

Description This function clears specific bits in the data register of a GPIO port.

Syntax

```
uint_32 dw_gpio_clear_bits( DW_APB_PORT_INFO_STRUCT_PTR  
dw_port, uint_32 bit_mask)
```

Parameters

`dw_port`
This variable holds the specific GPIO port information.

`bit_mask`
The bit-mask value represents the bits to be cleared.

dw_gpio_toggle_bits()

Description This function toggles specific bits in the data register of a GPIO port.

Syntax

```
uint_32 dw_gpio_toggle_bits(DW_APB_PORT_INFO_STRUCT_PTR
dw_port, uint_32 bit_mask)
```

Parameters

`dw_port`
This variable holds the specific GPIO port information.

`bit_mask`
The port bit-mask value represents the bits to be toggled.

dw_gpio_intr_enable()

Description This function enables the specified interrupt bit of PORT A.

Syntax

```
uint_32 dw_gpio_intr_enable(DW_APB_PORT_INFO_STRUCT_PTR
dw_port, uint_32 bit_mask)
```

Parameters

`dw_port`
This variable holds the specific GPIO port information.

`bit_mask`
The bit-mask value represents the interrupt bit to be enabled.

dw_gpio_intr_disable()

Description This function disables the specified interrupt bit of PORT A.

Syntax

```
uint_32 dw_gpio_intr_disable(DW_APB_PORT_INFO_STRUCT_PTR
dw_port, uint_32 bit_mask)
```

Parameters

`dw_port`
This variable holds the specific GPIO port information.

`bit_mask`
The bit-mask value represents the interrupt bit to be disabled.

dw_gpio_intr_mask()

Description This function masks the specified interrupt bits.

Syntax

```
uint_32 dw_gpio_intr_mask(DW_APB_PORT_INFO_STRUCT_PTR dw_port,
uint_32 bit_mask)
```

Parameters

`dw_port`
This variable holds the specific GPIO port information.

`bit_mask`
The bit-mask value represents the interrupt bit to be masked.

dw_gpio_intr_unmask()

Description This function unmaskes the specified interrupt bits.

Syntax

```
uint_32 dw_gpio_intr_unmask(DW_APB_PORT_INFO_STRUCT_PTR
dw_port, uint_32 bit_mask)
```

Parameters

`dw_port`
This variable holds the specific GPIO port information.

`bit_mask`
The bit-mask value indicates the interrupt bit to be unmasked.

dw_gpio_set_intr_level()

Description This function sets the level (level or edge) of the PORT A interrupt signals.

Syntax

```
uint_32 dw_gpio_set_intr_level( DW_APB_PORT_INFO_STRUCT_PTR
dw_port, uint_32 bit_mask, uint_32 level)
```

Parameters

`dw_port`
This variable holds the specific GPIO port information.

`bit_mask`

The bit-mask value indicates the signal to be changed.

`level`

Interrupt level value. The following are possible values:

- `LEVEL_SENSITIVE`
- `EDGE_SENSITIVE`.

`dw_gpio_get_intr_level()`

Description This function gets the interrupt-level register value.

Syntax

```
uint_32 dw_gpio_get_intr_level( DW_APB_PORT_INFO_STRUCT_PTR
dw_port)
```

Parameters `dw_port`
This variable holds the specific GPIO port information.

`dw_gpio_set_intr_polarity()`

Description This function sets the polarity sensitivity of the PORT A signals.

Syntax

```
uint_32 dw_gpio_set_intr_polarity( DW_APB_PORT_INFO_STRUCT_PTR
dw_port, uint_32 bit_mask, uint_32 pol)
```

Parameters `dw_port`
This variable holds the specific GPIO port information.

`bit_mask`

Bit mask value indicates which interrupt bit polarity should be changed.

`pol`

Interrupt-polarity value. The following are possible values:

- `ACTIVE_HIGH`
- `ACTIVE_LOW`

dw_gpio_get_intr_polarity()

Description This function gets the interrupt-polarity register value.

Syntax

```
uint_32 dw_gpio_get_intr_level(    DW_APB_PORT_INFO_STRUCT_PTR
dw_port)
```

Parameters `dw_port`
This variable holds the specific GPIO port information.

dw_gpio_intr_clear()

Description This function clears the interrupts.

Syntax

```
uint_32 dw_gpio_intr_clear(    DW_APB_PORT_INFO_STRUCT_PTR
dw_port, uint_32 bit_mask)
```

Parameters `dw_port`
This variable holds the specific GPIO port information.

`bit_mask`
The bit-mask value indicates the interrupt bit to be cleared.

dw_gpio_get_intr_status()

Description This function gets the interrupt status-register value.

Syntax

```
uint_32 dw_gpio_get_intr_status(DW_APB_PORT_INFO_STRUCT_PTR
dw_port)
```

Parameters `dw_port`
This variable holds the specific GPIO port information.

dw_gpio_debounce_reg_write()

Description This function sets the debounce-register value.

Syntax

```
uint_32 dw_gpio_debounce_reg_write(  
    DW_APB_PORT_INFO_STRUCT_PTR dw_port, uint_32 val)
```

Parameters

dw_port
This variable holds the specific GPIO port information.

val
This variable holds value to be written into debounce register.

Push-Button Driver APIs

_dw_apb_button_install()

Description This function registers the push-button driver with the MQX I/O system.

Syntax

```
uint_32 _dw_apb_button_install(char_ptr dev_name, pointer  
    init_data_ptr);
```

Parameters

dev_name
This variable is a `char_ptr` input parameter to the **dw_apb_button_install()** function and it holds string that identifies the device during **fopen()**.

init_data_ptr
This variable holds the device initialization values.

`_dw_apb_button_init()`

Description This function initializes the push-button (GPIO PORT A).

Syntax

```
uint_32 _dw_apb_button_init(pointer _PTR_ driver_info, pointer init_data);
```

Parameters `driver_info`
This variable is a structure pointer of the `DW_APB_BTN_INFO_STRUCT_PTR` type. This pointer is stored in intermediate GPIO subsystem, and memory for this structure is allocated in this function.

`init_data`
This variable is of type `DW_BUTTON_INIT_STRUCT_PTR`, and it holds the information related to the initial values of the driver.

`_dw_apb_button_deinit()`

Description This function de-allocates the driver's private data `struct` pointer and disables interrupts.

Syntax

```
uint_32 _dw_apb_button_deinit(pointer _PTR_ driver_info, pointer init_data);
```

Parameters `driver_info`
This variable is a structure pointer of the `DW_APB_BTN_INFO_STRUCT_PTR` type.

`init_data`
This variable is of type `DW_BUTTON_INIT_STRUCT_PTR`, and it holds the information related to the initial values of the driver.

`_dw_apb_button_ioctl()`

Description This function performs miscellaneous services for the I/O device.

The following are the actions performed:

- Set a particular hardware configuration
- Get the current configuration of the hardware

Syntax

```
uint_32 _dw_apb_button_ioctl(pointer _PTR_ driver_info,
                               _mqx_uint cmd, pointer param);
```

Parameters `driver_info`
This variable holds the specific GPIO port information.

`cmd`
The command value is passed by the application to perform a specific operation. If the command is invalid, driver returns the MQX error macro `IO_ERROR_INVALID_IOCTL_CMD`.

`param`
This parameter is taken as a void pointer and type-casted to one of the control data structures depending upon the command type.

LED Driver API's

`_dw_apb_led_install()`

Description This function registers the LED driver with the MQX IO system.

Syntax

```
uint_32 _dw_apb_led_install(char_ptr dev_name, pointer
                             init_data_ptr);
```

Parameters `dev_name`
This variable is a `char_ptr` input parameter to the `dw_apb_led_install()` function and holds a string that identifies the device during **`fopen()`**.

`init_data_ptr`
This variable holds the device initialization values.

`_dw_apb_led_init()`

Description This function initializes the LED (GPIO PORT B).

Syntax

```
uint_32 _dw_apb_led_init(pointer _PTR_ driver_info, pointer  
init_data);
```

`driver_info`

Parameters This variable is a structure pointer of the `DW_APB_LED_INFO_STRUCT_PTR` type. This pointer is stored in an intermediate GPIO subsystem and memory for this structure is allocated in this function.

`init_data_ptr`

This variable is of type `DW_GPIO_LED_INIT_STRUCT_PTR`, and it holds the information related to initial values of the driver.

`_dw_apb_led_deinit()`

Description This function de-allocates the driver's private data struct pointer.

Syntax

```
uint_32 _dw_apb_led_deinit(pointer _PTR_ driver_info, pointer  
init_data);
```

Parameters `driver_info`

This variable is a structure pointer of the `DW_APB_LED_INFO_STRUCT_PTR` type.

`init_data`

This variable is of type `DW_GPIO_LED_INIT_STRUCT_PTR`, and it holds the information related to initial values of the driver.

`_dw_apb_button_ioctl()`

Description This function performs miscellaneous services for the I/O device.

Following are the actions performed.

- Set a particular hardware configuration.
- Get the current configuration of the hardware.

Syntax

```
uint_32 _dw_apb_led_ioctl(pointer _PTR_ driver_info,  
_mqx_uint cmd, pointer param);
```

Parameters

`driver_info`

This variable is a structure pointer of the `DW_APB_LED_INFO_STRUCT_PTR` type.

`cmd`

The command value is passed by the application to perform a specific operation. If the command is invalid then driver returns the MQX error macro `IO_ERROR_INVALID_IOCTL_CMD`.

`param`

This parameter is taken as a `void` pointer and type casted to one of the control data structures depending on the command type.

Switch Driver API's

`_dw_apb_switch_install()`

Description This function registers the switch driver with the MQX I/O system.

Syntax

```
uint_32 _dw_apb_switch_install(char_ptr dev_name, pointer  
init_data_ptr);
```

Parameters

`dev_name`

This variable is a `char_ptr` input parameter to the `dw_apb_switch_install()` function. It holds a string that identifies the device during `fopen()`.

`init_data_ptr`

This variable holds the device initialization values.

`_dw_apb_switch_init()`

Description This function initializes the switch port (GPIO PORT C).

Syntax

```
uint_32 _dw_apb_switch_init(pointer _PTR_ driver_info, pointer
init_data);
```

Parameters `driver_info`
This variable is a structure pointer of the `DW_SWITCH_INFO_STRUCT_PTR` type. This pointer is stored in an intermediate GPIO subsystem, and memory for this structure is allocated in this function.

`init_data`
This variable is of type `DW_SWITCH_INIT_STRUCT_PTR`, and it holds the information related to initial values of the driver.

`_dw_apb_switch_deinit()`

Description This function de-allocates the driver's private data `struct` pointer.

Syntax

```
uint_32 _dw_apb_switch_deinit(pointer _PTR_ driver_info,
pointer init_data);
```

Parameters `driver_info`
This variable is a structure pointer of the `DW_SWITCH_INFO_STRUCT_PTR` type.

`init_data`
This variable is of type `DW_SWITCH_INIT_STRUCT_PTR`, and it holds the information related to initial values of the driver.

`_dw_apb_switch_ioctl()`

Description This function performs miscellaneous services for the I/O device.

The following are the actions performed.

- Set a particular hardware configuration.
- Get the current configuration of the hardware.

Syntax

```
uint_32 _dw_apb_switch_ioctl(pointer _PTR_ driver_info,  
_mqx_uint cmd, pointer param);
```

Parameters

`driver_info`

This variable is a structure pointer of the `DW_SWITCH_INFO_STRUCT_PTR` type.

`cmd`

The command value is passed by the application to perform a specific operation. If the command is invalid, the driver returns MQX error macro `IO_ERROR_INVALID_IOCTL_CMD`.

`param`

This parameter is taken as a `void` pointer and type casted to one of the control data structures depending on the command type.

Data Types

dw_gpio_led_init

The following is the GPIO LED initialization structure:

```
/* LED Init struct. */
typedef struct dw_gpio_led_init {
    uint_32 DIRECTION;
    uint_32 DEFAULT_VALUE;
} DW_GPIO_LED_INIT_STRUCT, _PTR_ DW_GPIO_LED_INIT_STRUCT_PTR;
```

Table 3 dw_gpio_led_init Structure

Field Name	Description
DIRECTION	Port direction
DEFAULT_VALUE	Default value for the GPIO port

dw_gpio_switch_init

The following is the GPIO Switch initialization structure:

```
typedef struct dw_switch_init {
    /* PORT C Default value. */
    uint_32 DEFAULT_VALUE;
} DW_SWITCH_INIT_STRUCT, _PTR_ DW_SWITCH_INIT_STRUCT_PTR;
```

Table 4 dw_gpio_switch_init Structure

Field Name	Description
DEFAULT_VALUE	Default value for the GPIO port

dw_button_init

The following is the GPIO Button initialization structure.

```
/* Button Init struct */
typedef struct dw_button_init {
    /* Interrupt mode. */
    uint_32 BUTTON_PORT;
    uint_32 INTR_MODE;
    uint_32 DEFAULT_VALUE;
    uint_32 BITMASK;
} DW_BUTTON_INIT_STRUCT, _PTR_ DW_BUTTON_INIT_STRUCT_PTR
```

Table 5 dw_button_init Structure

Field Name	Description
INTR_MODE	Push-button interrupt mode

dw_button_callback

This structure is used in setting the callback function for the push button. The assigned function is called by the driver whenever a change in the status of the push button occurs.

```
typedef struct dw_button_callback {
    /* Call back function, called from ISR. */
    void (_CODE_PTR_ CALL_BACK)(uint_32);
} DW_BUTTON_CALLBACK_STRUCT, _PTR_ DW_BUTTON_CALLBACK_STRUCT_PTR;
```

Table 6 dw_button_callback Structure

Field Name	Description
CALL_BACK	Function pointer to the callback function
DEFAULT_VALUE	Default value for the GPIO port

dw_set_interrupt_mode

This structure is used in setting the interrupt mode for GPIO push buttons.

```
typedef struct dw_set_interrupt_mode {
    uint_32 BIT_MASK;
    uint_32 MODE;
    uint_32 ERRORS;
} DW_SET_INTERRUPT_MODE_STRUCT, _PTR_ DW_SET_INTERRUPT_MODE_STRUCT_PTR;
```

Table 7 dw_set_interrupt_mode Structure

Field Name	Description
BIT_MASK	Bit-mask value
MODE	Interrupt mode
ERRORS	Error value for any error occurring during interrupt-mode setting

dw_get_interrupt_mode

This structure is used to get the interrupt mode of GPIO push buttons.

```
typedef struct dw_get_interrupt_mode {
    /* Interrupt mode for each button. */
    uint_32 INTR_MODE[NUMBER_OF_BUTTONS];
} DW_GET_INTERRUPT_MODE_STRUCT, _PTR_ DW_GET_INTERRUPT_MODE_STRUCT_PTR;
```

Table 8 dw_get_interrupt_mode Structure

Field Name	Description
INTR_MODE	Interrupt mode of the push buttons

I/O Controllers and I/O Structures

IOCTLS are used to interact with the device, that is, to check the status and to change any settings if required.

Push Buttons IOCtl's

The following IOCTls are used in the application to configure push buttons:

- `IO_IOCTL_BUTTON_GET_CALLBACK`
- `IO_IOCTL_BUTTON_SET_CALLBACK`
- `IO_IOCTL_BUTTON_SET_INTR_MODE`
- `IO_IOCTL_BUTTON_GET_INTR_MODE`
- `IO_IOCTL_BUTTON_ENABLE`
- `IO_IOCTL_BUTTON_DISABLE`

The following are DW_APB GPIO-specific IOCTls:

- `IO_IOCTL_BUTTON_DW_GET_STATE`
- `IO_IOCTL_BUTTON_DW_SET_DEBOUNCE_REG`

`IO_IOCTL_BUTTON_GET_CALLBACK/ IO_IOCTL_BUTTON_SET_CALLBACK`

This IOCTL is used to get and set the callback function for the push-button port. The assigned function is called by the driver whenever a change in the status of the push button occurs.

`IO_IOCTL_BUTTON_GET_INTR_MODE/ IO_IOCTL_BUTTON_SET_INTR_MODE`

This IOCTL is used to get and set the interrupt mode for the push buttons.

`uint_32 INTR_MODE`

Table 9 `IO_IOCTL_BUTTON_GET_INTR_MODE/IO_IOCTL_BUTTON_SET_INTR_MODE` IOCtl Description

Field Name	Description
<code>INTR_MODE</code>	This field can have the following values to set the interrupt mode: <pre>#define BUTTON_INTR_HIGH_LEVEL (1) #define BUTTON_INTR_LOW_LEVEL (2) #define BUTTON_INTR_RISING_EDGE (3) #define BUTTON_INTR_FALLING_EDGE (4)</pre>

IO_IOCTL_BUTTON_ENABLE / IO_IOCTL_BUTTON_DISABLE

This IOCTL is used to enable and disable the push-button interrupts.

IO_IOCTL_BUTTON_DW_SET_DEBOUNCE_REG

This IOCTL is used to assign a value to the debounce register.

All IOCTLs can take the parameter values specified below:

```
#define BUTTON_NORTH      (0x00000001)
#define BUTTON_EAST       (0x00000002)
#define BUTTON_SOUTH      (0x00000004)
#define BUTTON_WEST       (0x00000008)
#define BUTTON_CENTRE     (0x00000010)
#define BUTTON_ALL        (0x0000001F)
#define SWITCH_6          (0x00000040)
#define SWITCH_7          (0x00000080)
#define SWITCH_ALL        (0x000000FF)
```

LED Driver IOCTLs

Following IOCTLs are used in the application to configure LEDs (GPIO PORT B).

- IO_IOCTL_LED_TURN_ON
- IO_IOCTL_LED_TURN_OFF
- IO_IOCTL_LED_GET_STATE
- IO_IOCTL_LED_TOGGLE_STATE

IO_IOCTL_LED_TURN_ON / IO_IOCTL_LED_TURN_OFF

This IOCTL is used to turn the LEDs on and off.

IO_IOCTL_LED_GET_STATE

This IOCTL is used to get the status of GPIO LED port.

IO_IOCTL_LED_TOGGLE_STATE

This IOCTL is used to toggle the status of the GPIO LED port.

All IOCTLs can take the parameter values specified below:

```
#define LED_0                (0x00000001)
#define LED_1                (0x00000002)
#define LED_2                (0x00000004)
#define LED_3                (0x00000008)
#define LED_4                (0x00000010)
#define LED_5                (0x00000020)
#define LED_6                (0x00000040)
#define LED_7                (0x00000080)
#define LED_ALL              (0x00007FFF)
#define LED_NONE             (0x00000000)
```

Switch Driver IOCTLs

The following IOCTLs are used in the application to configure or get the status of switches (GPIO PORT C):

- IO_IOCTL_SWITCH_GET_STATE
- IO_IOCTL_SWITCH_ENABLE
- IO_IOCTL_SWITCH_DISABLE

IO_IOCTL_SWITCH_GET_STATE

This IOCTL is used to get the current value of the switch port (PORT C) data register. The current value is available as part of the `param` variable.

IO_IOCTL_SWITCH_ENABLE / IO_IOCTL_SWITCH_DISABLE

This IOCTL is used to enable and disable specific switches. The switch position is specified as part of the input argument to the IOCTL.

All IOCTLs can take the parameter values specified below:

```
#define SWITCH_0          (0x00000001)
#define SWITCH_1          (0x00000002)
#define SWITCH_2          (0x00000004)
#define SWITCH_3          (0x00000008)
#define SWITCH_4          (0x00000010)
#define SWITCH_5          (0x00000020)
#define SWITCH_6          (0x00000040)
#define SWITCH_7          (0x00000080)
#define SWITCH_ALL        (0x000000FF)
```

Default Configuration for DW APB GPIO

The default configuration of the GPIO ports depends on the user input because GPIO drivers are always installed from the user-level application.

3

Test Application

This chapter describes the test application is provided with the GPIO driver.

Push-Button Test Application

This test application configures the push button in one of following interrupt modes. It reads the input from the user and represents the status of the push-button on LEDs.

The interrupt modes are as follows:

- `BUTTON_INTR_HIGH_LEVEL`
- `BUTTON_INTR_LOW_LEVEL`
- `BUTTON_INTR_RISING_EDGE`
- `BUTTON_INTR_FALLING_EDGE`

LED Test Application

This test application configures the LED port and changes the status of the LED port.

Switch Test Application

This test application configures the switch in input mode, read the input from the user, and display the status on LED port.