



DW APB ARC EM Starter Kit I2C User Manual

Version 6282-001 March 2013

Copyright Notice and Proprietary Information Notice

Copyright © 2013 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.
700 E. Middlefield Road
Mountain View, CA 94043
www.synopsys.com

Contents

Customer Support.....	4
Accessing SolvNet.....	4
Contacting the Synopsys Technical Support Center.....	4
1 Introduction	5
2 DW APB I2C Driver	6
Block Diagram of DW_apb_i2c	6
Software Architecture.....	7
Error Codes.....	8
Functions	8
_dw_apb_i2c_poll_install().....	8
_dw_apb_i2c_poll_init()	9
_dw_apb_i2c_poll_deinit()	9
_dw_apb_i2c_poll_read().....	10
_dw_apb_i2c_poll_write()	10
_dw_apb_i2c_ioctl()	11
_dw_apb_i2c_int_install()	11
_dw_apb_i2c_int_init()	12
_dw_apb_i2c_int_deinit()	12
_dw_apb_i2c_int_read()	13
_dw_apb_i2c_int_write()	13
Data Types.....	14
dw_apb_i2c_init.....	14
I/O Controllers and I/O Structures.....	15
IO_IOCTL_I2C_ENABLE_DEVICE / IO_IOCTL_I2C_DISABLE_DEVICE	15
IO_IOCTL_I2C_SET_MASTER_MODE / IO_IOCTL_I2C_SET_SLAVE_MODE /	
IO_IOCTL_GET_MODE	16
IO_IOCTL_I2C_SET_DESTINATION_ADDRESS/ IO_IOCTL_I2C_GET_DESTINATION_ADDRESS	16
IO_IOCTL_I2C_SET_STATION_ADDRESS / IO_IOCTL_I2C_GET_STATION_ADDRESS	17
IO_IOCTL_I2C_GET_STATISTICS / IO_IOCTL_I2C_CLEAR_STATISTICS.....	17
IO_IOCTL_I2C_GET_BUS_AVAILABILITY	18

IO_IOCTL_I2C_STOP	18
IO_IOCTL_I2C_GET_LAST_TX_STATUS	19
DW_APB_I2C_SLAVE_STATE	19
DW_APB_I2C_DATA_COUNT	20
Default Configuration for DW APB I2C	20
3 Test Application.....	21
I2C_ADC demo.....	21

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 I2C driver for writing applications. It aims to make the development of embedded application easier for developers. This document contains the descriptions of all the data types and API functions.

The following definitions can be 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 for 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

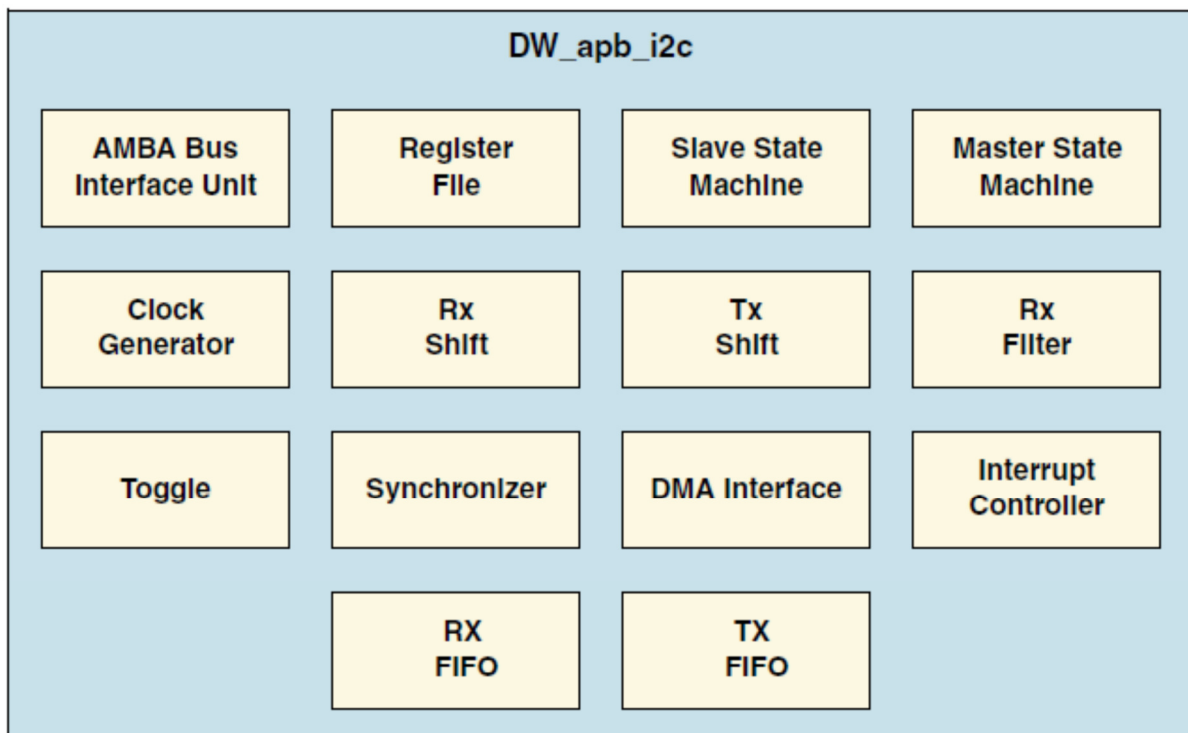
This chapter describes the I2C device driver.

Block Diagram of DW_apb_i2c

DW APB I2C is a configurable and programmable control bus that provides support for the communications link between integrated circuits in a system. It is a simple two-wire bus with a software-defined protocol for system control, and is used in temperature sensors and voltage-level translators to EEPROMs, general-purpose I/O, A/D and D/A converters, CODECs, and many types of microprocessors.

Figure 1 shows the functional block diagram of DW_apb_i2c.

Figure 1 Functional Block Diagram of DW_apb_i2c



Software Architecture

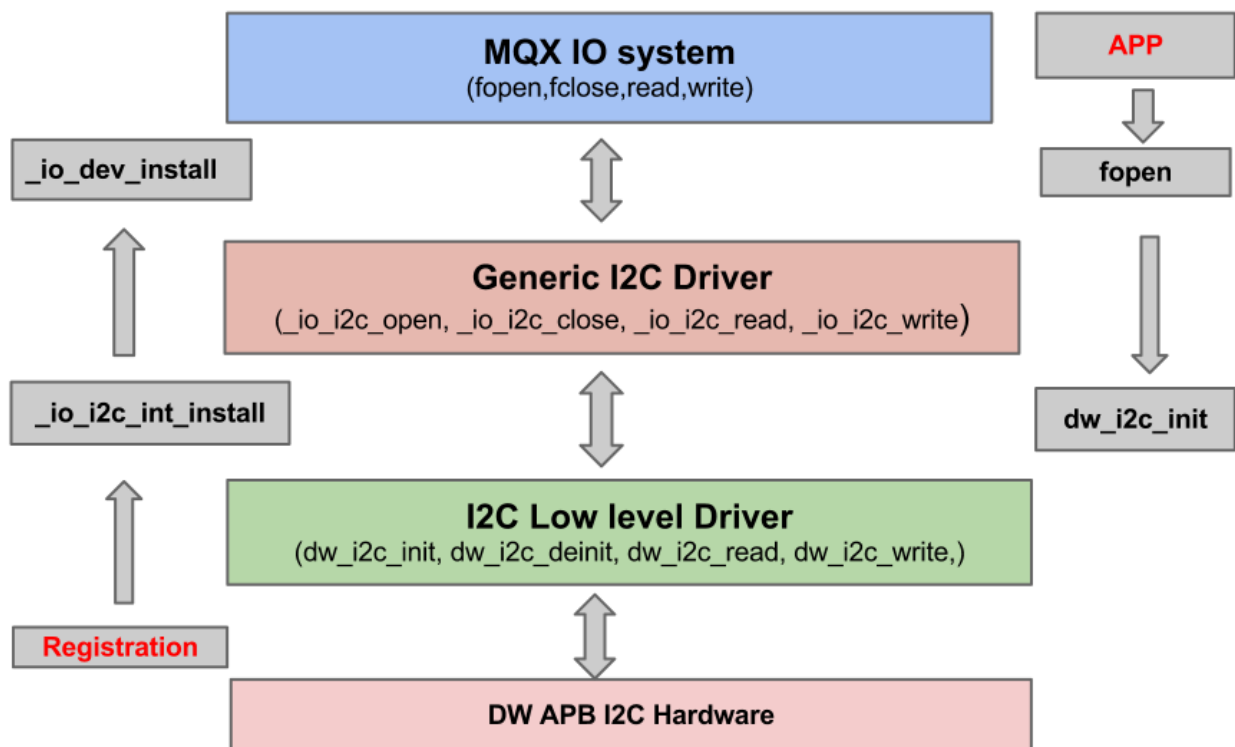
The DW APB driver is written using standard MQX I2C APIs based on the I/O subsystem. Standard MQX I2C APIs are responsible for data flow between the I/O subsystem and low-level driver.

All low level functionalities such as data transmission, reception and interrupts are handled by the low level driver. Standard MQX I2C APIs are responsible for maintaining the data provided by the low level driver.

For more information on the structures and its members, see the *MQX® RTOS I/O Drivers User's Guide*.

Figure 2 shows the overview of the software architecture.

Figure 2 Software Architecture



Error Codes

The DW APB I2C driver returns standard MQX-defined error codes for each error condition handled in the driver. The following are the standard MQX error defines used in driver.

Table 2 Error Codes and its Description

Error Codes	Description
I2C_IOCTL_ERR_NULL_PARAM	Parameter pointer passed to IOCTL is NULL.
MQX_OUT_OF_MEMORY	Memory allocation failed
IO_ERROR_INVALID_IOCTL_CMD	Invalid IOCTL

Functions

`_dw_apb_i2c_poll_install()`

Description This function installs the polling mode of the driver, it can be called either from the BSP or from the application.

Syntax

```
uint_32 _dw_apb_i2c_poll_install(char_ptr dev_name,  
pointer init_data);
```

Parameters

`dev_name`
This variable is a `char_ptr` input parameter to the `_dw_apb_i2c_poll_install()` function and holds a string that identifies the device during `fopen()`.

`init_data`
This variable is an input parameter to the `_dw_apb_i2c_poll_install()` function. The default configuration of the device is provided through this parameter, including vector number, base address, and station address.

`_dw_apb_i2c_poll_init()`

Description This function initializes the DW APB I2C hardware and provides a handle to the user. It also initializes required resources for the device. It must be called prior to the use of any other functions.

Syntax

```
uint_32 _dw_apb_i2c_poll_init( pointer _PTR_ driver_info,  
DW_APB_I2C_INIT_STRUCT_PTR init_data);
```

Parameters `driver_info`
This variable is a structure pointer of the `DW_APB_INFO_STRUCT_PTR` type. This pointer is stored in an intermediate I2C subsystem, and memory for this structure is allocated by this function.

`init_data`
This variable is of type `DW_APB_I2C_INIT_STRUCT_PTR`, and it holds the information related to initial values of the driver.

`_dw_apb_i2c_poll_deinit()`

Description This function de-initializes the DW APB I2C hardware and releases all obtained resources. After calling this function, users have no control over the I2C device.

Syntax

```
uint_32 _dw_apb_i2c_poll_deinit (DW_APB_I2C_INFO_STRUCT_PTR  
pdata);
```

Parameters `pdata`
This variable is a structure pointer of type `DW_APB_I2C_INFO_STRUCT`, which is the driver's private data structure.

`_dw_apb_i2c_poll_read()`

Description This function is used to read the data from the device. This function call is a blocking call: It never returns until it reads `length` number of bytes, which is passed to this function as one of its arguments. In case of error, this function returns and the returned value represents the number of bytes transmitted.

Syntax

```
uint_32 _dw_apb_i2c_poll_read (DW_APB_I2C_INFO_STRUCT_PTR
pdata, char_ptr buf, uint_32 length);
```

Parameters `Pdata`
This variable is a structure pointer of the `DW_APB_I2C_INFO_STRUCT_PTR` type, which is a private data structure.

`Buf`
This variable is an input-buffer pointer; the driver uses this buffer pointer to store the received bytes.

`length`
Number of bytes to be read.

`_dw_apb_i2c_poll_write()`

Description This function writes the specified number of bytes to the transmit FIFO. It also checks for an error. In of error, this function returns, and the returned value represents the number of bytes transmitted.

Syntax

```
uint_32 _dw_apb_i2c_poll_write (DW_APB_I2C_INFO_STRUCT_PTR
pdata, char_ptr buf, uint_32 length);
```

Parameters `pdata`
This variable is a structure pointer of the `DW_APB_I2C_INFO_STRUCT_PTR` type, which is a private data structure.

`buf`
This variable is an input-buffer pointer, and points to the buffer from which the data is transmitted. This buffer is filled by the application.

`length`
Number of bytes to be transmitted.

`_dw_apb_i2c_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
- Get the statistical information of the device

Syntax

```
uint_32 _dw_apb_i2c_ioctl(DW_APB_I2C_INFO_STRUCT_PTR pdata,
                          _mqx_uint cmd, pointer param);
```

Parameters `pdata`
This variable is a structure pointer of the `DW_APB_I2C_INFO_STRUCT_PTR` type, which is a private data structure.

`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. The type of `param` is determined according to the command value.

`_dw_apb_i2c_int_install()`

Description This function installs the interrupt mode of the driver. It can be called either from the BSP or from the application.

Syntax

```
uint_32 _dw_apb_i2c_int_install(char_ptr dev_name, pointer
init_data);
```

Parameters `dev_name`
This variable is a `char_ptr` that holds a string that identifies the device during `fopen()`.

`init_data`
This parameter passes the default configuration of the device, including the vector number, base address, and station address.

`_dw_apb_i2c_int_init()`

Description This function initializes the DW APB I2C hardware for interrupt mode and provides a handle to the user. This function also initializes required resources for the device. It must be called prior to the use of any other functions.

Syntax

```
uint_32 _dw_apb_i2c_int_init (IO_I2C_INT_INFO_STRUCT_PTR io_i2c,  
char_ptr dev_name);
```

Parameters `io_i2c`
This variable is a pointer of type `IO_I2C_INT_DEVICE_STRUCT_PTR`. It holds the interrupt I/O initialization information.

`dev_name`

This variable is a `char_ptr` that holds a string to identify the device during **fopen()**.

`_dw_apb_i2c_int_deinit()`

Description This function de-initializes the DW APB I2C hardware, disables all interrupts, and releases all obtained resources. After calling this function, users have no control over the I2C device.

Syntax

```
uint_32 _dw_apb_i2c_int_deinit (DW_APB_I2C_INFO_STRUCT_PTR  
pdata);
```

Parameters `pdata`
This variable is a structure pointer of type `DW_APB_I2C_INFO_STRUCT_PTR`, which is a private data structure of the driver.

`_dw_apb_i2c_int_read()`

Description This function is used to read the data from the device. This function call is a blocking call, meaning that this function returns only after it reads `length` number of bytes or if an error occurs during read. This function returns the number of bytes read.

Syntax

```
uint_32 _dw_apb_i2c_int_read(DW_APB_I2C_INFO_STRUCT_PTR
                             pdata, char_ptr buf, uint_32 length);
```

Parameters `pdata`
This variable is a structure pointer of the `DW_APB_I2C_INFO_STRUCT_PTR` type, which is a private data structure.

`buf`
This variable is an input-buffer pointer. The driver uses this buffer pointer to store the received bytes.

`length`
Number of bytes to be read.

`_dw_apb_i2c_int_write()`

Description This function writes input data to the transmit FIFO. It also checks for an error. If an error occurs, this function returns, and the returned value represents the number of bytes transmitted.

Syntax

```
uint_32 _dw_apb_i2c_int_write(DW_APB_I2C_INFO_STRUCT_PTR
                               pdata, char_ptr buf, uint_32 length);
```

Parameters `pdata`
This variable is a structure pointer of the `DW_APB_I2C_INFO_STRUCT_PTR` type, which is a private data structure.

`buf`
This variable is an input buffer pointer, and points to the buffer from which the data is transmitted. This buffer is filled by the application.

`length`
Number of bytes to be transmitted.

Data Types

dw_apb_i2c_init

```
typedef struct dw_apb_i2c_init {
    volatile uint_32  BASE_ADDR;
    /* Slave (station) address of the device. */
    uint_32  STATION_ADDRESS;
    /* ISR Number of the device. */
    uint_32  VECTOR;
} DW_APB_I2C_INIT_STRUCT, _PTR_ DW_APB_I2C_INIT_STRUCT_PTR;
```

Table 3 dw_apb_i2c_init_struct Structure Description

Field Name	Description
BASE_ADDR	Base address of the I2C memory where the device is mapped. This value is used by the driver to access the registers.
STATION_ADDRESS	Slave address of the I2C device.
VECTOR	Interrupt vector number for the I2C device.

I/O Controllers and I/O Structures

IOCTLS are used to interact with the device. They check the status and change any settings of the device if required.

Following are the IOCTS supported by the driver

- `IO_IOCTL_I2C_DISABLE_DEVICE`
- `IO_IOCTL_I2C_ENABLE_DEVICE`
- `IO_IOCTL_I2C_SET_MASTER_MODE`
- `IO_IOCTL_I2C_SET_SLAVE_MODE`
- `IO_IOCTL_I2C_GET_MODE`
- `IO_IOCTL_I2C_SET_DESTINATION_ADDRESS`
- `IO_IOCTL_I2C_GET_DESTINATION_ADDRESS`
- `IO_IOCTL_I2C_SET_STATION_ADDRESS`
- `IO_IOCTL_I2C_GET_STATION_ADDRESS`
- `IO_IOCTL_I2C_GET_STATE`
- `IO_IOCTL_I2C_GET_STATISTICS`
- `IO_IOCTL_I2C_CLEAR_STATISTICS`
- `IO_IOCTL_I2C_GET_BUS_AVAILABILITY`
- `IO_IOCTL_I2C_STOP`
- `IO_IOCTL_I2C_GET_LAST_TX_STATUS`
- `DW_APB_I2C_SLAVE_STATE`
- `DW_APB_I2C_DATA_COUNT`

IO_IOCTL_I2C_ENABLE_DEVICE / IO_IOCTL_I2C_DISABLE_DEVICE

This IOCTL is used to enable and disable the I2C. Driver enables and disables the device only if the device is in the idle state. Otherwise it polls the status bit continuously until the device becomes idle.

IO_IOCTL_I2C_SET_MASTER_MODE / IO_IOCTL_I2C_SET_SLAVE_MODE / IO_IOCTL_GET_MODE

This IOCTL is used to get and set the I2C device mode. `IO_IOCTL_SET_MASTER_MODE` is used to set the device in master mode, and `IO_IOCTL_I2C_SET_SLAVE_MODE` is used to set the device in slave mode. The driver sets this mode only if the device is in the idle state. Otherwise it polls the status bit continuously until the device becomes idle.

`IO_IOCTL_GET_MODE` returns the current mode of the device.

uint_32_ptr param

Field Name	Description
Param	This field contains the value of I2C mode for <code>IO_IOCTL_GET_MODE</code> . • <code>I2C_MODE_MASTER</code> • <code>I2C_MODE_SLAVE</code>

IO_IOCTL_I2C_SET_DESTINATION_ADDRESS/ IO_IOCTL_I2C_GET_DESTINATION_ADDRESS

This IOCTL is used to get and set the destination of the slave I2C device (slave station address).

uint_32_ptr param

Field Name	Description
Param	This field is used to pass the destination address as an argument.

IO_IOCTL_I2C_SET_STATION_ADDRESS / IO_IOCTL_I2C_GET_STATION_ADDRESS

This IOCTL is used to get and set the station address of the I2C device.

uint_32_ptr param

Field Name	Description
Param	This field is used to pass the station address as an argument and also it can be an empty buffer to hold the returned station address.

IO_IOCTL_I2C_GET_STATISTICS / IO_IOCTL_I2C_CLEAR_STATISTICS

This IOCTL is used to get and clear the current statistical information of the I2C device. The statistical information is maintained in a private data structure of the I2C driver.

The following are the members of the `DW_APB_I2C_INFO_STRUCT` structure used to maintain the statistical information.

```
typedef struct dw_apb_i2c_info_struct
{
    /* other members ... */
    uint_32    INTERRUPTS;
    uint_32    RX_BYTES;
    uint_32    TX_BYTES;
    uint_32    TX_OVER;
    uint_32    RX_OVER;
    uint_32    RX_UNDER;
} DW_APB_I2C_INFO_STRUCT, _PTR_ DW_APB_I2C_INFO_STRUCT_PTR;
```

Field Name	Description
INTERRUPTS	Number of Interrupts occurred
RX_BYTES	Total number of received bytes
TX_BYTES	Total number of transmitted bytes
TX_OVER	Total number of TX overrun errors
RX_OVER	Total number of RX overrun errors
RX_UNDER	Total number of RX underrun errors

IO_IOCTL_I2C_GET_BUS_AVAILABILITY

This IOCTL is used to get the bus availability of the I2C device. It returns `I2C_BUSY` if the I2C bus is busy. Otherwise it returns `I2C_READY` if the I2C bus is available. The return value is passed through the user-specified `param` argument.

uint_32_ptr param

Field Name	Description
Param	This field can have the following values:
	<code>#define I2C_BUSY (0)</code>
	<code>#define I2C_READY (1)</code>

IO_IOCTL_I2C_STOP

This IOCTL is used to send the stop signal on the I2C bus.

IO_IOCTL_I2C_GET_LAST_TX_STATUS

This IOCTL is used to get the status of the last transfer (receive or transmit). Call this IOCTL to check the last transfer status after calling the read/write API. This IOCTL returns the error macro if any error occurred; otherwise it returns 0.

uint_32_ptr param

Table 4 IO_IOCTL_I2C_GET_LAST_TX_STATUS I/O Control Field Descriptions

Field Name	Description
Param	This field can have the following values: <pre> #define DW_ABRT_TXDATA_NOACK (0x0501) #define DW_ABRT_MASTER_DIS (0x0502) #define DW_ARB_LOST (0x0503) #define DW_ABRT_7B_ADDR_NOACK (0x0504) #define DW_ABRT_USER_ABRT (0x0505) #define DW_TX_ABRT (0x0506) #define DW_TX_OVER (0x0507) #define DW_RX_OVER (0x0508) #define DW_RX_UNDER (0x0509) #define DW_UNKNOWN_ERROR (0xFFFF) </pre>

DW_APB_I2C_SLAVE_STATE

This IOCTL is used to check whether I2C device has received any bytes or any read requests from the I2C master device. This IOCTL is used when the I2C device is configured in slave mode.

- `I2C_STATE_RECEIVE` indicates the slave has received a byte.
- `I2C_STATE_TRANSMIT` indicates the slave has a read request.
- `I2C_STATE_FINISHED` indicates the master device has sent a stop signal to end the data transfer.

uint_32_ptr param

Table 5 IO_IOCTL_I2C_GET_SLAVE_STATE I/O Control Field Descriptions

Field Name	Description
Param	This field can have the following values: #define I2C_STATE_RECEIVE (2) #define I2C_STATE_TRANSMIT (3) #define I2C_STATE_FINISHED (4)

DW_APB_I2C_DATA_COUNT

This IOCTL is used to get the number of valid data items available in the receive FIFO.

Default Configuration for DW APB I2C

The default values with which the DW APB I2C is configured during the initialization are explained in the table below. These values are passed to the driver through the `struct dw_apb_i2c_init`.

3

Test Application

This chapter describes the test application for I2C.

I2C_ADC demo

Application is written to interact with ADC device connected. DW APB I2C is configured as I2C master and ADC device is I2C slave. The test application read 16 bits data from each of 4 channels of slave device.