

# **ADI\_TWI\_PSEUDO DEVICE DRIVER**

**DATE: 7 FEBRUARY 2006**

## Table of Contents

<b>1. Overview</b>	<b>5</b>
<b>2. Files</b>	<b>6</b>
2.1. Include Files	6
2.2. Source Files	6
<b>3. Lower Level Drivers</b>	<b>7</b>
<b>4. Resources Required</b>	<b>8</b>
4.1. Interrupts	8
4.2. DMA	8
4.3. Timers	8
4.4. Real-Time Clock	8
4.5. Programmable Flags	8
4.6. Pins	8
<b>5. Supported Features of the Device Driver</b>	<b>9</b>
5.1. Directionality	9
5.2. Dataflow Methods	9
5.3. Buffer Types	9
5.4. Command IDs	9
5.4.1. Device Manager Commands	9
5.4.2. Common Commands	10
5.4.3. Device Driver Specific Commands	10
5.5. Callback Events	11
5.5.1. Common Events	11
5.5.2. Device Driver Specific Events	12
5.6. Return Codes	12
5.6.1. Common Return Codes	13
5.6.2. Device Driver Specific Return Codes	13
<b>6. Opening and Configuring the Device Driver</b>	<b>15</b>
6.1. Entry Point	15
6.2. Default Settings	15
6.3. Additional Required Configuration Settings	15
<b>7. Hardware Considerations</b>	<b>16</b>
7.1. TWI Pseudo Port Structure	16

---

7.2.	TWI bit rate structure .....	16
7.3.	TWI Pseudo master configuration with arbitration.....	16
7.4.	TWI Pseudo master configuration without arbitration.....	17

## Table of Figures

Table 1 - Revision History.....	4
Table 2 - Supported Dataflow Directions .....	9
Table 3 - Supported Dataflow Methods.....	9
Table 4 - Default Settings .....	15
Table 5 – Additional Required Settings.....	15

**Document Revision History**

Date	Description of Changes
7/2/2006	Updated document format
5/18/06	Updated file locations

**Table 1 - Revision History**

## 1. Overview

The TWI device driver can operate in hardware or pseudo modes, this document only describes the pseudo operation of the TWI driver. For information on the hardware TWI please refer to the ADI\_TWI Driver Documentation manual. The pseudo TWI device driver supports only master mode of operation, and an error code will be returned when trying to start the driver (using ADI\_DEV\_CMD\_SET\_DATAFLOW) in slave mode. The pseudo TWI can run together with upto n other pseudo TWI devices and or TWI hardware devices (this is only limited by the hardware, note that hardware resources allocated to each pseudo TWI cannot be used by anything else while the device DATAFLOW is on).

## 2. Files

The files listed below comprise the device driver API and source files.

### 2.1. Include Files

The driver sources include the following include files:

- <services/services.h> This file contains all definitions, function prototypes etc. for all the System Services.
- <drivers/adi\_dev.h> This file contains all definitions, function prototypes etc. for the Device Manager and general device driver information.
- <driver/twi/adi\_twi.h>

### 2.2. Source Files

The driver sources are contained in the following files, as located in the default installation directory:

- <Blackfin/lib/src/drivers/twi/adi\_twi.c>
- <Blackfin/lib/src/drivers/twi/adi\_twi\_pseudo.c>

### **3. Lower Level Drivers**

This driver does not use any lower level driver.

## 4. Resources Required

Device drivers typically consume some amount of system resources. This section describes the resources required by the device driver.

Unless explicitly noted in the sections below, this device driver uses the System Services to access and control any required hardware. The information in this section may be helpful in determining the resources this driver requires, such as the number of interrupt handlers or number of DMA channels etc., from the System Services.

Because dynamic memory allocations are not used in the Device Drivers or System Services, all memory used by the Device Drivers and System Services must be supplied by the application. The Device Drivers and System Services supply macros that can be used by the application to size the amount of base memory and/or the amount of incremental memory required to support the needed functionality. Memory for the Device Manager and System Services is provided in the initialization functions (adi\_xxx\_Init()).

### 4.1. Interrupts

This driver uses one timer interrupt and if in master arbitration mode uses one flag interrupt.

### 4.2. DMA

This driver does not use DMA.

### 4.3. Timers

This driver uses one timer interrupt which is selected using the adi\_twi\_pseudo\_port structure that is passed to ADI\_TWI\_CMD\_SET\_PSEUDO.

### 4.4. Real-Time Clock

This driver does not use the real-time clock.

### 4.5. Programmable Flags

This driver uses two programmable flags that are defined in the adi\_twi\_pseudo\_port that is passed to ADI\_TWI\_CMD\_SET\_PSEUDO. If master arbitration is used the scl flag also uses a callback.

### 4.6. Pins

This driver uses two GPIO pins that can be configured using the ADI\_TWI\_CMD\_SET\_PSEUDO control. These vary depending to the Blackfin device used, please check hardware reference manual for the relevant pins used.



## 5. Supported Features of the Device Driver

This section describes what features are supported by the device driver.

### 5.1. Directionality

The driver supports the dataflow directions listed in the table below.

ADI_DEV_DIRECTION	Description
ADI_DEV_DIRECTION_BIDIRECTIONAL	Supports both the reception of data and transmission of data through the device.

**Table 2 - Supported Dataflow Directions**

### 5.2. Dataflow Methods

The driver supports the dataflow methods listed in the table below.

ADI_DEV_MODE	Description
ADI_DEV_MODE_SEQ_CHAINED	Supports the sequential I/O dataflow method. This puts the TWI device into master mode.

**Table 3 - Supported Dataflow Methods**

### 5.3. Buffer Types

The driver supports the buffer types listed in the table below.

- ADI\_DEV\_SEQ\_1D\_BUFFER
  - Linear one-dimensional sequential buffer
  - pAdditionalInfo – u32 (used to pass device address or ADI\_TWI\_REPEAT\_START in master mode)

### 5.4. Command IDs

This section enumerates the commands that are supported by the driver. The commands are divided into three sections. The first section describes commands that are supported directly by the Device Manager. The next section describes common commands that the driver supports. The remaining section describes driver specific commands.

Commands are sent to the device driver via the adi\_dev\_Control() function. The adi\_dev\_Control() function accepts three arguments:

- DeviceHandle – This parameter is a ADI\_DEV\_DEVICE\_HANDLE type that uniquely identifies the device driver. This handle is provided to the client in the adi\_dev\_Open() function call.
- CommandID – This parameter is a u32 data type that specifies the command ID.
- Value – This parameter is a void \* whose value is context sensitive to the specific command ID.

The sections below enumerate the command IDs that are supported by the driver and the meaning of the Value parameter for each command ID.

#### 5.4.1. Device Manager Commands

The commands listed below are supported and processed directly by the Device Manager. As such, all device drivers support these commands.

- ADI\_DEV\_CMD\_TABLE
  - Table of command pairs being passed to the driver
  - Value – ADI\_DEV\_CMD\_VALUE\_PAIR \*
- ADI\_DEV\_CMD\_END
  - Signifies the end of a command pair table
  - Value – ignored
- ADI\_DEV\_CMD\_PAIR
  - Single command pair being passed
  - Value – ADI\_DEV\_CMD\_PAIR \*
- ADI\_DEV\_CMD\_SET\_SYNCHRONOUS
  - Enables/disables synchronous mode for the driver
  - Value – TRUE/FALSE

### 5.4.2. Common Commands

The command IDs described in this section are common to many device drivers. The list below enumerates all common command IDs that are supported by this device driver.

- ADI\_DEV\_CMD\_SET\_DATAFLOW\_METHOD
  - Specifies the dataflow method the device is to use. The list of dataflow types supported by the device driver is specified in section 5.2.
  - Value – ADI\_DEV\_MODE enumeration
- ADI\_DEV\_CMD\_SET\_DATAFLOW
  - Enables/disables dataflow through the device
  - Value – TRUE/FALSE
- ADI\_DEV\_CMD\_GET\_PERIPHERAL\_DMA\_SUPPORT
  - Determines if the device driver is supported by peripheral DMA
  - Value – u32 \* (location where TRUE or FALSE is stored)

### 5.4.3. Device Driver Specific Commands

The command IDs listed below are supported and processed by the device driver. These command IDs are unique to this device driver.

- ADI\_TWI\_CMD\_SET\_HARDWARE
  - Sets the TWI hardware the driver uses, this is limited to the hardware available
  - Value – ADI\_INT\_PERIPHERAL\_ID (Interrupt id of TWI hardware)
- ADI\_TWI\_CMD\_SET\_PSEUDO
  - Sets the TWI in pseudo mode, please refer to TWIPseudoDeviceDriverDocumentation for more device operation in this mode.
  - Value – struct adi\_twi\_pseudo\_port \* (Pseudo port structure, defined in adi\_twi.h)
- ADI\_TWI\_CMD\_SET\_RATE
  - Sets the frequency of the TWI port
  - Value – struct adi\_twi\_bit\_rate \* (structure containig frequency and duty cycle, defined in adi\_twi\_h)
- ADI\_TWI\_CMD\_SET\_SLAVE\_ADDR
  - Sets the address used when the device driver is operating in slave mode
  - Value – u32 (Address, this is a 7bit value)
- ADI\_TWI\_CMD\_SET\_GCAL
  - Enables general call addressing in slave mode
  - Value – not used
- ADI\_TWI\_CMD\_SET\_SCCB
  - Enables SCCB compatibility in master mode.
  - Value – not used
- ADI\_TWI\_CMD\_SET\_SCLOVR
  - Overrides the SCL flag, when set the output on the SCL pin will be driven low

- Value – TRUE/FALSE
- ADI\_TWI\_CMD\_SET\_SDAOVR
  - Overrides the SDA flag, when set the output on the SDA pin will be driver low
  - Value – TRUE/FALSE
- ADI\_TWI\_CMD\_SET\_FIFO
  - Sets the FIFO buffer operating mode
  - Value – u32 (0x0008 enables receive interrupts on word boundary, 0x0004 enables word transfers on transmit interrupts, the driver automatically handles odd byte transfers)
- ADI\_TWI\_CMD\_SET\_LOSTARB
  - Set the number of retries for lost arbitration events before callback is run
  - Value – u32 (0=never generate callback, otherwise value indicates the number of events needed to trigger callback)
- ADI\_TWI\_CMD\_SET\_ANAK
  - Set the number of retries for address negative acknowledge events before callback is run
  - Value – u32 (0=never generate callback, otherwise value indicates the number of events needed to trigger callback)
- ADI\_TWI\_CMD\_SET\_DNAK
  - Set the number of retries for data negative acknowledge events before callback is run
  - Value – u32 (0=never generate callback, otherwise value indicates the number of events needed to trigger callback)
- ADI\_TWI\_CMD\_GET\_SENSE
  - Get the status of SDA, SCL and BUSBUSY flags
  - Value – u32 \* (Address of value holding status bits, 0x0100=BUSBUSY, 0x0080=SCL and 0x0040=SDA)

## 5.5. Callback Events

This section enumerates the callback events the device driver is capable of generating. The events are divided into two sections. The first section describes events that are common to many device drivers. The next section describes driver specific event IDs. The callback function of the client should be prepared to process each of the events in these sections.

The callback function is of the type ADI\_DCB\_CALLBACK\_FN. The callback function is passed three parameters. These parameters are:

- ClientHandle – This void \* parameter is the value that is passed to the device driver as a parameter in the adi\_dev\_Open() function.
- EventID – This is a u32 data type that specifies the event ID.
- Value – This parameter is a void \* whose value is context sensitive to the specific event ID.

The sections below enumerate the event IDs that the device driver can generate and the meaning of the Value parameter for each event ID.

### 5.5.1. Common Events

The events described in this section are common to many device drivers. The list below enumerates all common event IDs that are supported by this device driver.

- ADI\_DEV\_EVENT\_BUFFER\_PROCESSED
  - Notifies callback function that a chained or sequential I/O buffer has been processed by the device driver. This event is also used to notify that an entire circular buffer has been processed if the driver was directed to generate a callback upon completion of an entire circular buffer.
  - Value – For chained or sequential I/O dataflow methods, this value is the CallbackParameter value that was supplied in the buffer that was passed to the adi\_dev\_Read(), adi\_dev\_Write() or adi\_dev\_SequentialIO() function. For the circular dataflow method, this value is the address of the buffer provided in the adi\_dev\_Read() or adi\_dev\_Write() function.

### 5.5.2. Device Driver Specific Events

The events listed below are supported and processed by the device driver. These event IDs are unique to this device driver.

Master mode events

- ADI\_TWI\_EVENT\_BUFWRERR
  - Receive buffer overflow, usually indicates no read buffer available
  - Value - CallbackParameter value that was supplied in the buffer that was passed to the adi\_dev\_SequentialIO() function.
- ADI\_TWI\_EVENT\_BUFRDERR
  - Transmit buffer underflow, usually indicates to write buffer available
  - Value - CallbackParameter value that was supplied in the buffer that was passed to the adi\_dev\_SequentialIO() function.
- ADI\_TWI\_EVENT\_DNAK
  - Data Negative Acknowledge has been received x times (where x is value set by ADI\_TWI\_CMD\_SET\_DNAK)
  - Value - CallbackParameter value that was supplied in the buffer that was passed to the adi\_dev\_SequentialIO() function.
- ADI\_TWI\_EVENT\_ANAK
  - Address Negative Acknowledge has been received x times (where x is value set by ADI\_TWI\_CMD\_SET\_ANAK)
  - Value - CallbackParameter value that was supplied in the buffer that was passed to the adi\_dev\_SequentialIO() function.
- ADI\_TWI\_EVENT\_LOSTARB
  - TWI has Lost Bus Arbitration x times (where x is value set by ADI\_TWI\_CMD\_SET\_LOSTARB)
  - Value - CallbackParameter value that was supplied in the buffer that was passed to the adi\_dev\_SequentialIO() function.
- ADI\_TWI\_EVENT\_YANKED
  - Buffer has been removed from device transmit or receive queue, this occurs if there is an error sending/receiving a buffer and there are subsequent buffers that are chaing with repeat start address
  - Value - Buffer that has been removed

## 5.6. Return Codes

All API functions of the device driver return status indicating either successful completion of the function or an indication that an error has occurred. This section enumerates the return codes that the device driver is capable of returning to the client. A return value of ADI\_DEV\_RESULT\_SUCCESS indicates success, while any other value indicates an error or some other informative result. The value ADI\_DEV\_RESULT\_SUCCESS is always equal to the value zero. All other return codes are a non-zero value.

The return codes are divided into two sections. The first section describes return codes that are common to many device drivers. The next section describes driver specific return codes. Wherever functions in the device driver API are called, the application should be prepared to process any of these return codes.

Typically the application should check the return code for ADI\_DEV\_RESULT\_SUCCESS, taking appropriate corrective action if ADI\_DEV\_RESULT\_SUCCESS is not returned. For example:

```
if (adi_dev_Xxxx(...) == ADI_DEV_RESULT_SUCCESS) {
    // normal processing
} else {
    // error processing
}
```

### 5.6.1. Common Return Codes

The return codes described in this section are common to many device drivers. The list below enumerates all common return codes that are supported by this device driver.

- ADI\_DEV\_RESULT\_SUCCESS
  - The function executed successfully.
- ADI\_DEV\_RESULT\_NOT\_SUPPORTED
  - The function is not supported by the driver.
- ADI\_DEV\_RESULT\_DEVICE\_IN\_USE
  - The requested device is already in use.
- ADI\_DEV\_RESULT\_NO\_MEMORY
  - There is insufficient memory available.
- ADI\_DEV\_RESULT\_BAD\_DEVICE\_NUMBER
  - The device number is invalid.
- ADI\_DEV\_RESULT\_DIRECTION\_NOT\_SUPPORTED
  - The device cannot be opened in the direction specified.
- ADI\_DEV\_RESULT\_BAD\_DEVICE\_HANDLE
  - The handle to the device driver is invalid.
- ADI\_DEV\_RESULT\_BAD\_MANAGER\_HANDLE
  - The handle to the Device Manager is invalid.
- ADI\_DEV\_RESULT\_BAD\_PDD\_HANDLE
  - The handle to the physical driver is invalid.
- ADI\_DEV\_RESULT\_INVALID\_SEQUENCE
  - The action requested is not within a valid sequence.
- ADI\_DEV\_RESULT\_ATTEMPTED\_READ\_ON\_OUTBOUND\_DEVICE
  - The client attempted to provide an inbound buffer for a device opened for outbound traffic only.
- ADI\_DEV\_RESULT\_ATTEMPTED\_WRITE\_ON\_INBOUND\_DEVICE
  - The client attempted to provide an outbound buffer for a device opened for inbound traffic only.
- ADI\_DEV\_RESULT\_DATAFLOW\_UNDEFINED
  - The dataflow method has not yet been declared.
- ADI\_DEV\_RESULT\_DATAFLOW\_INCOMPATIBLE
  - The dataflow method is incompatible with the action requested.
- ADI\_DEV\_RESULT\_BUFFER\_TYPE\_INCOMPATIBLE
  - The device does not support the buffer type provided.
- ADI\_DEV\_RESULT\_CANT\_HOOK\_INTERRUPT
  - The Interrupt Manager failed to hook an interrupt handler.
- ADI\_DEV\_RESULT\_CANT\_UNHOOK\_INTERRUPT
  - The Interrupt Manager failed to unhook an interrupt handler.
- ADI\_DEV\_RESULT\_NON\_TERMINATED\_LIST
  - The chain of buffers provided is not NULL terminated.
- ADI\_DEV\_RESULT\_NO\_CALLBACK\_FUNCTION\_SUPPLIED
  - No callback function was supplied when it was required.
- ADI\_DEV\_RESULT\_REQUIRES\_UNIDIRECTIONAL\_DEVICE
  - Requires the device be opened for either inbound or outbound traffic only.
- ADI\_DEV\_RESULT\_REQUIRES\_BIDIRECTIONAL\_DEVICE
  - Requires the device be opened for bidirectional traffic only.

### 5.6.2. Device Driver Specific Return Codes

The return codes listed below are supported and processed by the device driver. These event IDs are unique to this device driver.

- ADI\_TWI\_RESULT\_BAD\_RATE

- The data rate passed using ADI\_TWI\_CMD\_SET\_RATE is outside valid values, the frequency must be within 20 to 400 and duty cycle must be within 1 to 99. Depending on the clock speed of the Blackfin other values might generate this error.
- ADI\_TWI\_RESULT\_BAD\_ADDR
  - The address pass in either slave mode (with ADI\_TWI\_CMD\_SET\_SLAVE) or in master mode (in pAdditionalInfo) is invalid. The address must be within 8 to 123 or ADI\_TWI\_RSTART\_ADDR.
- ADI\_TWI\_RESULT\_NO\_DATA
  - The read/write/sequential buffer submitted contained no data.
- ADI\_TWI\_RESULT\_SLAVE\_DCB
  - The device has been configured for slave mode and uses deferred callbacks. This could cause problems because whilst the driver is waiting for another buffer to transmit/receive data from/too the bus is being held by the twi driver.
- ADI\_TWI\_RESULT\_BAD\_PSEUDO
  - The pseudo port configuration cannot be setup, because the values supplied are invalid or the timers and/or flags are already being used.

## 6. Opening and Configuring the Device Driver

This section describes the default configuration settings for the device driver and any additional configuration settings required from the client application.

### 6.1. Entry Point

When opening the device driver with the `adi_dev_Open()` function call, the client passes a parameter to the function that identifies the specific device driver that is being opened. This parameter is called the entry point. The entry point for this driver is listed below.

- `<ADITWIEnterPoint>`

### 6.2. Default Settings

The table below describes the default configuration settings for the device driver. If the default values are inappropriate for the given system, the application should use the command IDs listed in the table to configure the device driver appropriately. Any configuration settings not listed in the table below are undefined.

Item	Default Value	Possible Values	Command ID
Hardware configuration	ADI_INT_TWI	Depends on Blakfin device used	ADI_TWI_CMD_SET_HARDWARE
Rate	Frequency=100 Duty cycle=50	Frequency 20-400 Duty cycle 1-99	ADI_TWI_CMD_SET_RATE
FIFO	0x000C	0x0000, 0x0004, 0x0008 and 0x000C	ADI_TWI_CMD_SET_FIFO
Lost arbitration	3	0 - 255	ADI_TWI_CMD_SET_LOSTARB
ANAK	1	0 - 255	ADI_TWI_CMD_SET_ANAK
DNAK	1	0 - 255	ADI_TWI_CMD_SET_DNAK
SCL override	0	0 - 1	ADI_TWI_CMD_SET_SCLOVR
SDA override	0	0 - 1	ADI_TWI_CMD_SET_SDAOVR
General call	0	1 (Once set cannot be reset)	ADI_TWI_CMD_SET_GCALL
SCCB mode	0	1 (Once set cannot be reset)	ADI_TWI_CMD_SET_SCCB
Pseudo configuration	0	See section 7.3 for example	ADI_TWI_CMD_SET_PSEUDO

**Table 4 - Default Settings**

### 6.3. Additional Required Configuration Settings

In addition to the possible overrides of the default driver settings, the device driver requires the application to specify the additional configuration information listed in the table below.

Item	Possible Values	Command ID
Dataflow method	See section 5.2	ADI_DEV_CMD_SET_DATAFLOW_METHOD
Pseudo configuration	adi_twi_pseudo_port structure defining hardware to use, see section 7.3 for example	ADI_TWI_CMD_SET_PSEUDO

**Table 5 – Additional Required Settings**

## 7. Hardware Considerations

The pseudo TWI device driver does not require any configur, however to use the hardware TWI device the Blackfin device used must support hardware TWI (ie BF537). Listed below are the twi data structures that are used to configure the pseudo port and the twi data rate along with some typical configuration options for the TWI.

The frequency in pseudo mode may not be accurate depending on processor clock speed and frequency selected, due to performace issues the driver imposes a minimum timer speed of 10,000 clock cycles. This is configured in adi\_twi.c by altering the definition ADI\_TWI\_PSEUDO\_PERIOD.

Note: the SCL and SDA pins need an external pull-up resistor, please refer to the TWI specifications for more information about this, the ADSP-BF537 EZ-Kit Lite has these resistors on the board.

### 7.1. TWI Pseudo Port Structure

Parameter passed with ADI\_TWI\_CMD\_SET\_PSEUDO.

```
// structure for software based TWI port
typedef struct
{
    ADI_FLAG_ID scl;           // Flag that is to be used as SCL port
    ADI_FLAG_ID sda;           // Flag that is to be used as SDA port
    u32 timer;                 // Timer that is used to clock the pseudo TWI
    ADI_INT_PERIPHERAL_ID scl_pid; // Clock interrupt id
} adi_twi_pseudo_port;       // submit as ADI_TWI_CMD_SET_PSEUDO control data
```

### 7.2. TWI bit rate structure

Parameter passed with ADI\_TWI\_CMD\_SET\_RATE.

```
// structure for TWI timing info
typedef struct
{
    u16 frequency;           // The TWI bit rate per second in kHz
    u16 duty_cycle;          // The clock duty cycle (not used in pseudo mode)
} adi_twi_bit_rate;         // submit as ADI_TWI_CMD_SET_RATE control data
```

### 7.3. TWI Pseudo master configuration with arbitration

Configure the TWI as a pseudo port with SCL on flag PG1 and SDA on flag PG0. The TWI is clocked from ADI\_TMR\_GP\_TIMER\_2 and flag interrupts occur on ADI\_INT\_PORTG\_B for master bus arbitration.

Note: Care must be taken when using arbitration, the flag interrupt occurs on a bank of flags and if not configured correctly other pins may trigger the flag interrupt, this will not cause the pseudo TWI flag interrupt handler to be called but it could cause the processor to 'lockup'. This can occur if another flag on the same bank being processed by the flag interrupt is switching very fast (ie like using 8bit PPIO on BF537 when using ADI\_INT\_PORTFG\_A or ADI\_INT\_PORTG\_B).

```
adi_twi_pseudo_port port={
    ADI_FLAG_PG1,
    ADI_FLAG_PG0,
    ADI_TMR_GP_TIMER_2,
    ADI_INT_PORTG_B
};
ADI_DEV_CMD_VALUE_PAIR config[]={           // configuration table for the TWI driver
    {ADI_TWI_CMD_SET_PSEUDO,(void *)&port},
    {ADI_DEV_CMD_SET_DATAFLOW_METHOD,(void *)ADI_DEV_MODE_SEQ_CHAINED},
    {ADI_TWI_CMD_SET_FIFO,(void *)0x0000},
    {ADI_TWI_CMD_SET_LOSTARB,(void *)1},
    {ADI_TWI_CMD_SET_ANAK,(void *)0},
    {ADI_TWI_CMD_SET_DNAK,(void *)0},
    {ADI_DEV_CMD_SET_DATAFLOW,(void *)TRUE},
```



```

        {ADI_DEV_CMD_END,NULL}
};
adi_dev_Open(
    ManagerHandle,
    &ADITWIEnterPoint,
    DeviceNumber,
    DeviceHandle,
    &PhysicalDevice,
    ADI_DEV_DIRECTION_BIDIRECTIONAL,
    NULL,
    DCBHandle,
    DMCallback);
adi_dev_Control(device,ADI_DEV_CMD_TABLE,config);

```

## 7.4. TWI Pseudo master configuration without arbitration

Configure the TWI as a pseudo port with SCL on flag PG1 and SDA on flag PG0. The TWI is clocked from ADI\_TMR\_GP\_TIMER\_2.

```

adi_twi_pseudo_port port={
    ADI_FLAG_PG1,
    ADI_FLAG_PG0,
    ADI_TMR_GP_TIMER_2,
    (ADI_INT_PERIPHERAL_ID)NULL
};
ADI_DEV_CMD_VALUE_PAIR config[]={          // configuration table for the TWI driver
    {ADI_TWI_CMD_SET_PSEUDO,(void *)&port},
    {ADI_DEV_CMD_SET_DATAFLOW_METHOD,(void *)ADI_DEV_MODE_SEQ_CHAINED},
    {ADI_TWI_CMD_SET_FIFO,(void *)0x0000},
    {ADI_TWI_CMD_SET_LOSTARB,(void *)1},
    {ADI_TWI_CMD_SET_ANAK,(void *)0},
    {ADI_TWI_CMD_SET_DNAK,(void *)0},
    {ADI_DEV_CMD_SET_DATAFLOW,(void *)TRUE},
    {ADI_DEV_CMD_END,NULL}
};
adi_dev_Open(
    ManagerHandle,
    &ADITWIEnterPoint,
    DeviceNumber,
    DeviceHandle,
    &PhysicalDevice,
    ADI_DEV_DIRECTION_BIDIRECTIONAL,
    NULL,
    DCBHandle,
    DMCallback);
adi_dev_Control(device,ADI_DEV_CMD_TABLE,config);

```