**PCIe-Mini-Synchro**

**16-BIT TRACKING**   
**RESOLVER-TO-DIGITAL CONVERTER**

**PCIexpress Mini**

**928-10-000-4000**

**Software MANUAL**

Revision 1.0

September 2020

**ALPHI TECHNOLOGY CORPORATION**

**1898 E. Southern Ave**

**Tempe, AZ 85282 USA**

**Tel : (480) 838-2428**

**Fax: (480) 838-4477**

**NOTICE**

The information in this document has been carefully checked and is believed to be entirely reliable. While all reasonable efforts to ensure accuracy have been taken in the preparation of this manual, ALPHI TECHNOLOGY assumes no responsibility resulting from omissions or errors in this manual, or from the use of information contained herein.

ALPHI TECHNOLOGY reserves the right to make any changes, without notice, to this or any of ALPHI TECHNOLOGY’s products to improve reliability, performance, function or design.

ALPHI TECHNOLOGY does not assume any liability arising out of the application or use of any product or circuit described herein; nor does ALPHI TECHNOLOGY convey any license under its patent rights or the rights of others.

**ALPHI TECHNOLOGY CORPORATION**

**All Rights Reserved**

This document shall not be duplicated, nor its contents used

for any purpose, unless express permission has been granted in advance.

# Table of Contents

Hierarchical Index 2

Class Index 3

Class Documentation 4

AlphiBoard 4

AlteraPio 11

AlteraSpi 13

BoardVersion 16

ControlRegister 18

LinearAddress 21

Ltc2754\_av 22

Ltc2754\_SPI 25

MINIPCIE\_DEV\_CTX 28

PcieCra 29

PCIeMini\_Synch 31

Rd19231 35

StatusRegister 38

Index 41

# Hierarchical Index

## Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

AlphiBoard

PCIeMini\_Synch

AlteraPio

ControlRegister

StatusRegister

AlteraSpi

Ltc2664\_SPI

BoardVersion

LinearAddress

Ltc2664\_av 22

MINIPCIE\_DEV\_CTX 28

PcieCra

Rd19231

# Class Index

## Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

**AlphiBoard (Base class implementing a PCIe board and the Jungo driver )**

**AlteraPio (Altera Avalon Pio controller class )** 11

**AlteraSpi (Low level SPI interface to the SPI hardware )**

**BoardVersion (Board Hardware identification and version )**

**ControlRegister (Status register class for the PCIeMini board )**

**LinearAddress (Memory Segment Descriptor )**

**Ltc2664\_av (Synchro simulator )**

**Ltc2664\_SPI (D/A controller SPI class definition )**

**MINIPCIE\_DEV\_CTX (Minipcie Device Information Structure )** 28

**PcieCra (PCIe controller class )** 29

**PCIeMini\_Synch (PCIeMini\_Synchro controller board object )** 31

**Rd19231 (DDC RD19231 controller class )** 35

**StatusRegister (Status register class for the PCIeMini board )**

# Class Documentation

## AlphiBoard Class Reference

Base class implementing a PCIe board and the Jungo driver.

#include <AlphiBoard.h>

Inheritance diagram for AlphiBoard:

### Public Member Functions

**AlphiBoard** (UINT16 vendorId, UINT16 deviceId)

**~AlphiBoard** (void)

*Destructor.*

**HRESULT** **Open** (int brdNbr)

*Open a board.*

DWORD **reset** ()

*reset some of the board resources*

uint32\_t **getFpgaID** ()

*Get the FPGA ID of.*

time\_t **getFpgaTimeStamp** ()

*Return the timestamp corresponding to when the FPGA was compiled.*

void **setVerbose** (int **verbose**)

*set the verbose flag*

bool **IsValidDevice** (const CHAR \*sFunc)

*Validate a WDC device handle.*

DWORD **hookInterruptServiceRoutine** (uint32\_t mask, **MINIPCIE\_INT\_HANDLER** uicr, void \*userData)

*Setup the interrupt of the board.*

DWORD **hookInterruptServiceRoutine** (**MINIPCIE\_INT\_HANDLER** uicr)

*Set an interrupt handling routine.*

DWORD **unhookInterruptServiceRoutine** ()

*Disable the board interrupt.*

DWORD **enableInterrupts** (uint16\_t mask=0xffff)

*Enable PCIe interrupts.*

DWORD **disableInterrupts** ()

*Disable PCIe interrupts.*

DWORD **Close** ()

*Close a device handle.*

volatile void \* **getBar0Address** (size\_t offset)

*Return a pointer to an object in BAR 0.*

volatile void \* **getBar2Address** (size\_t offset)

*Return a pointer to an object in BAR 2.*

### Static Public Member Functions

static void **MsSleep** (int ms)

*Millisecond Delay Function.*

### Public Attributes

**LinearAddress** **bar0**

*Memory descriptor for the BAR0 in user memory.*

**LinearAddress** **bar2**

*Memory descriptor for the BAR2 in user memory.*

**PcieCra** \* **cra**

*PCIe Interface instance.*

**BoardVersion** \* **sysid**

*Board identification.*

int **verbose**

*Flag used by various functions to determine the amount of messages to generate.*

DWORD **libStatus**

*Status returned when trying to open the Jungo library. If it is not WD\_STATUS\_SUCCESS, the initialization failed.*

### Detailed Description

Base class implementing a PCIe board and the Jungo driver.

### Constructor & Destructor Documentation

#### AlphiBoard::AlphiBoard (UINT16 *vendorId*, UINT16 *deviceId*)

#### AlphiBoard::~AlphiBoard (void )

Destructor.

Will close the connection to the board if needed.

### Member Function Documentation

#### DWORD AlphiBoard::Close ()

Close a device handle.

##### Returns

status, a Jungo status code

#### DWORD AlphiBoard::disableInterrupts ()

Disable PCIe interrupts.

Disable the generation of PCIe interrupts by the PCIe interface, and the reception by the Windows driver.

##### Return values

|  |  |
| --- | --- |
| *Status* | code |

#### DWORD AlphiBoard::enableInterrupts (uint16\_t *mask* = 0xffff)

Enable PCIe interrupts.

Enable the generation of PCIe interrupts by the board's PCIe interface. Enable the reception of PCIe interrupts by the Windows driver.

##### Parameters

|  |  |
| --- | --- |
| *mask* | Optional bit map of which local interrupt line is enabled (board dependent.) If not used, default to 0xffff - all local interrupts allowed. |

##### Return values

|  |  |
| --- | --- |
| *Status* | code |

#### volatile void \* AlphiBoard::getBar0Address (size\_t *offset*)

Return a pointer to an object in BAR 0.

##### Parameters

|  |  |
| --- | --- |
| *offset* | Offset in BAR0 |

##### Return values

|  |  |
| --- | --- |
| *Pointer* | to the object |

#### volatile void \* AlphiBoard::getBar2Address (size\_t *offset*)

Return a pointer to an object in BAR 2.

##### Parameters

|  |  |
| --- | --- |
| *offset* | Offset in BAR2 |

##### Return values

|  |  |
| --- | --- |
| *Pointer* | to the object |

#### uint32\_t AlphiBoard::getFpgaID ()

Get the FPGA ID of.

##### Returns

The FPGA ID.

#### time\_t AlphiBoard::getFpgaTimeStamp ()

Return the timestamp corresponding to when the FPGA was compiled.

##### Returns

a timestamp.

#### DWORD AlphiBoard::hookInterruptServiceRoutine (MINIPCIE\_INT\_HANDLER *uicr*)

Set an interrupt handling routine.

##### Parameters

|  |  |
| --- | --- |
| *uicr* | user callback routine typedef void (\_\_stdcall \*UsersIntCompletionRoutine)(void \*, uint32\_t); |

##### Returns

ERRCODE\_NO\_ERROR if successful.

#### DWORD AlphiBoard::hookInterruptServiceRoutine (uint32\_t *mask*, MINIPCIE\_INT\_HANDLER *uicr*, void \* *userData*)

Setup the interrupt of the board.

Specify and interrupt service routine and enable the interrupts.

##### Parameters

|  |  |
| --- | --- |
| *mask* | board dependent interrupt mask. |
| *uicr* | pointer to the interrupt service routine. |

##### Returns

WD\_STATUS\_SUCCESS when the operation succeeded WD\_INVALID\_PARAMETER if the board is not opened WD\_OPERATION\_FAILED if the board does not have an interrupt resource WD\_OPERATION\_ALREADY\_DONE if there is already an isr active for the interrupt.

#### bool AlphiBoard::IsValidDevice (const CHAR \* *sFunc*)

Validate a WDC device handle.

##### Parameters

|  |  |
| --- | --- |
| *sFunc* | C-string with name of the function e.g. "IntEnable" |

##### Return values

|  |  |
| --- | --- |
| *true* | if the device context exists. |

#### static void AlphiBoard::MsSleep (int *ms*)[inline], [static]

Millisecond Delay Function.

#### HRESULT AlphiBoard::Open (int *brdNbr*)

Open a board.

Establishes a connection to a board.

##### Parameters

|  |  |
| --- | --- |
| *brdNbr* | the board index to open. |

##### Returns

WD\_DEVICE\_NOT\_FOUND if there is no board corresponding to the number

#### DWORD AlphiBoard::reset ()[inline]

reset some of the board resources

#### void AlphiBoard::setVerbose (int *vb*)

set the verbose flag

The verbose value is used to send more information to the log file or console. It is only partially implemented.

##### Parameters

|  |  |
| --- | --- |
| *vb* | Verbosity level. |

#### DWORD AlphiBoard::unhookInterruptServiceRoutine ()

Disable the board interrupt.

##### Parameters

|  |  |
| --- | --- |
| *mask* | board dependent interrupt mask. |
| *uicr* | pointer to the interrupt service routine. |

##### Returns

WD\_STATUS\_SUCCESS when the operation succeeded WD\_INVALID\_PARAMETER if the board is not opened WD\_OPERATION\_FAILED if the board does not have an interrupt resource WD\_OPERATION\_ALREADY\_DONE if there the interrupt is already disabled.

### Member Data Documentation

#### LinearAddress AlphiBoard::bar0

Memory descriptor for the BAR0 in user memory.

#### LinearAddress AlphiBoard::bar2

Memory descriptor for the BAR2 in user memory.

#### PcieCra\* AlphiBoard::cra

PCIe Interface instance.

#### DWORD AlphiBoard::libStatus

Status returned when trying to open the Jungo library. If it is not WD\_STATUS\_SUCCESS, the initialization failed.

#### BoardVersion\* AlphiBoard::sysid

Board identification.

#### int AlphiBoard::verbose

Flag used by various functions to determine the amount of messages to generate.

#### The documentation for this class was generated from the following files:

**AlphiBoard.h**

**AlphiBoard.cpp**

**AlphiBoard\_irq.cpp**

## AlteraPio Class Reference

Altera Avalon Pio controller class.

#include <AlteraPio.h>

Inheritance diagram for AlteraPio:

### Public Member Functions

**AlteraPio** (volatile void \*addr, uint16\_t capabilities)

**PCIeMini\_status** **reset** ()

*Reset the PIO.*

uint32\_t **getData** ()

**PCIeMini\_status** **setData** (uint32\_t data)

uint32\_t **getIrqMask** ()

**PCIeMini\_status** **setIrqMask** (uint32\_t mask)

uint32\_t **getEdgeCapture** ()

**PCIeMini\_status** **clearEdgeCapture** (uint32\_t mask)

### Static Public Attributes

static const uint16\_t **CAP\_INPUT** = 0x01

static const uint16\_t **CAP\_OUTPUT** = 0x02

static const uint16\_t **CAP\_INPUT\_OUTPUT** = 0x03

### Detailed Description

Altera Avalon Pio controller class.

### Constructor & Destructor Documentation

#### AlteraPio::AlteraPio (volatile void \* *addr*, uint16\_t *capabilities*)[inline]

### Member Function Documentation

#### PCIeMini\_status AlteraPio::clearEdgeCapture (uint32\_t *mask*)[inline]

#### uint32\_t AlteraPio::getData ()[inline]

#### uint32\_t AlteraPio::getEdgeCapture ()[inline]

#### uint32\_t AlteraPio::getIrqMask ()[inline]

#### PCIeMini\_status AlteraPio::reset ()[inline]

Reset the PIO.

Whenever supported, set the direction register to all input, the data register to 0, and disable interrupts.

##### Return values

|  |  |
| --- | --- |
| *Always* | success |

#### PCIeMini\_status AlteraPio::setData (uint32\_t *data*)[inline]

#### PCIeMini\_status AlteraPio::setIrqMask (uint32\_t *mask*)[inline]

### Member Data Documentation

#### const uint16\_t AlteraPio::CAP\_INPUT = 0x01[static]

#### const uint16\_t AlteraPio::CAP\_INPUT\_OUTPUT = 0x03[static]

#### const uint16\_t AlteraPio::CAP\_OUTPUT = 0x02[static]

#### The documentation for this class was generated from the following file:

**AlteraPio.h**

## AlteraSpi Class Reference

Low level SPI interface to the SPI hardware.

#include <AlteraSpi.h>

Inheritance diagram for AlteraSpi:

### Public Member Functions

**AlteraSpi** (volatile void \*addr)

*Constructor.*

int **sendSpiCommand** (uint32\_t slave, uint32\_t write\_length, const uint8\_t \*write\_data, uint32\_t read\_length, uint8\_t \*read\_data, uint32\_t flags)

*Send an SPI command.*

uint32\_t **getRxData** ()

*Get the content of the receive data register.*

uint32\_t **getStatus** ()

void **setTxData** (uint32\_t data)

void **setControl** (uint32\_t data)

void **selectSlave** (uint32\_t data)

### Static Public Attributes

static const uint32\_t **status\_ROE\_mask** = 0x0008

*Receive - overrun error.*

static const uint32\_t **status\_TOE\_mask** = 0x0010

*Transmitter-overrun error.*

static const uint32\_t **status\_TMT\_mask** = 0x0020

*Transmitter shift-register empty.*

static const uint32\_t **status\_TRDY\_mask** = 0x0040

*Transmitter ready.*

static const uint32\_t **status\_RRDY\_mask** = 0x0080

*Receiver ready.*

### Detailed Description

Low level SPI interface to the SPI hardware.

### Constructor & Destructor Documentation

#### AlteraSpi::AlteraSpi (volatile void \* *addr*)

Constructor.

Called only when the board is opened.

##### Parameters

|  |  |
| --- | --- |
| *addr* | Pointer to the device in user space. |

### Member Function Documentation

#### uint32\_t AlteraSpi::getRxData ()[inline]

Get the content of the receive data register.

##### Return values

|  |  |
| --- | --- |
| *Content* | of the receive data register |

#### uint32\_t AlteraSpi::getStatus ()[inline]

#### void AlteraSpi::selectSlave (uint32\_t *data*)[inline]

#### int AlteraSpi::sendSpiCommand (uint32\_t *slave*, uint32\_t *write\_length*, const uint8\_t \* *write\_data*, uint32\_t *read\_length*, uint8\_t \* *read\_data*, uint32\_t *flags*)

Send an SPI command.

This is a very simple routine which performs one SPI master transaction. It would be possible to implement a more efficient version using interrupts and sleeping threads but this is probably not worthwhile initially.

##### Parameters

|  |  |
| --- | --- |
| *slave* | Slave number select 0-31 |
| *write\_length* | Number of bytes to send |
| *write\_data* | A pointer to the buffer containing the data to write |
| *read\_length* | Number of bytes to receive |
| *read\_data* | A pointer to the buffer where the received data is going |
| *flags* | A bit mask, only ALT\_AVALON\_SPI\_COMMAND\_TOGGLE\_SS\_N is used. |

##### Return values

|  |  |
| --- | --- |
| *Number* | of bytes read - in SPI read and write are simultaneous so it cannot be 0. |

#### void AlteraSpi::setControl (uint32\_t *data*)[inline]

#### void AlteraSpi::setTxData (uint32\_t *data*)[inline]

### Member Data Documentation

#### const uint32\_t AlteraSpi::status\_ROE\_mask = 0x0008[static]

Receive - overrun error.

The ROE bit is set to 1 if new data is received while the rxdata register is full(that is, while the RRDY bit is 1).In this case, the new data overwrites the old.Writing to the status register clears the ROE bit to 0.

#### const uint32\_t AlteraSpi::status\_RRDY\_mask = 0x0080[static]

Receiver ready.

The RRDY bit is set to 1 when the rxdata register is full.

#### const uint32\_t AlteraSpi::status\_TMT\_mask = 0x0020[static]

Transmitter shift-register empty.

In master mode, the TMT bit is set to 0 when a transaction is in progress and set to 1 when the shift register is empty.

#### const uint32\_t AlteraSpi::status\_TOE\_mask = 0x0010[static]

Transmitter-overrun error.

The TOE bit is set to 1 if new data is written to the txdata register while it is still full (that is, while the TRDY bit is 0). In this case, the new data is ignored. Writing to the status register clears the TOE bit to 0.

#### const uint32\_t AlteraSpi::status\_TRDY\_mask = 0x0040[static]

Transmitter ready.

The TRDY bit is set to 1 when the txdata register is empty.

#### The documentation for this class was generated from the following files:

**AlteraSpi.h**

**AlteraSpi.cpp**

## BoardVersion Class Reference

Board Hardware identification and version.

#include <AlphiBoard.h>

### Public Member Functions

**BoardVersion** (volatile uint32\_t \*addr)

*constructor*

uint32\_t **getVersion** ()

*Version, if there is one programmed on the board hardware. Typically 0.*

time\_t **getTimeStamp** ()

*Date when the board firmware was compiled.*

### Detailed Description

Board Hardware identification and version.

### Constructor & Destructor Documentation

#### BoardVersion::BoardVersion (volatile uint32\_t \* *addr*)

constructor

This constructor reads the chip register to initialize the data. It is called by the open and should not be called by the user.

##### Parameters

|  |  |
| --- | --- |
| *addr* | Offset to the sysid controller in the BAR2 address space |

### Member Function Documentation

#### time\_t BoardVersion::getTimeStamp ()

Date when the board firmware was compiled.

#### uint32\_t BoardVersion::getVersion ()

Version, if there is one programmed on the board hardware. Typically 0.

#### The documentation for this class was generated from the following files:

**AlphiBoard.h**

**AlphiBoard.cpp**

## ControlRegister Class Reference

Status register class for the PCIeMini board.

#include <MiniSynchStatusRegister.h>

Inheritance diagram for ControlRegister:

### Public Types

enum **Resolution** { **RES\_10\_BIT** = 0, **RES\_12\_BIT** = 1, **RES\_14\_BIT** = 2, **RES\_16\_BIT** = 3 }

enum **ShiftControl** { **VEL1** = 4, **VEL2** = 0 }

### Public Member Functions

**ControlRegister** (volatile void \*addr)

*Constructor.*

**PCIeMini\_status** **enableSpiDa** (bool enabled)

*Select the type of output of the D/A.*

### Static Public Attributes

static const uint32\_t **CTRL\_BITn\_MASK** = 0x01

static const uint32\_t **CTRL\_AQB\_mask** = 0x010

static const uint32\_t **CTRL\_DSR\_mask** = 0x020

static const uint32\_t **CTRL\_DA\_CLEAR\_mask** = 0x040

static const uint32\_t **CTRL\_DaMode\_mask** = 0x400

static const uint32\_t **CTRL\_LDAC\_mask** = 0x800

### Detailed Description

Status register class for the PCIeMini board.

### Member Enumeration Documentation

#### enum ControlRegister::Resolution

##### Enumerator:

|  |  |
| --- | --- |
| RES\_10\_BIT |  |
| RES\_12\_BIT |  |
| RES\_14\_BIT |  |
| RES\_16\_BIT |  |

#### enum ControlRegister::ShiftControl

##### Enumerator:

|  |  |
| --- | --- |
| VEL1 |  |
| VEL2 |  |

### Constructor & Destructor Documentation

#### ControlRegister::ControlRegister (volatile void \* *addr*)[inline]

Constructor.

Only called when the board is opened.

##### Parameters

|  |  |
| --- | --- |
| *addr* | Pointer to the device in user's space. |

### Member Function Documentation

#### PCIeMini\_status ControlRegister::enableSpiDa (bool *enabled*)[inline]

Select the type of output of the D/A.

##### Parameters

|  |  |
| --- | --- |
| *enabled* | True when the D/As are general purpose, False when they are a synchro simulator |

##### Return values

|  |  |
| --- | --- |
| *ERRCODE\_NO\_ERROR* |  |

### Member Data Documentation

#### const uint32\_t ControlRegister::CTRL\_AQB\_mask = 0x010[static]

#### const uint32\_t ControlRegister::CTRL\_BITn\_MASK = 0x01[static]

#### const uint32\_t ControlRegister::CTRL\_DA\_CLEAR\_mask = 0x040[static]

#### const uint32\_t ControlRegister::CTRL\_DaMode\_mask = 0x400[static]

#### const uint32\_t ControlRegister::CTRL\_DSR\_mask = 0x020[static]

#### const uint32\_t ControlRegister::CTRL\_LDAC\_mask = 0x800[static]

#### The documentation for this class was generated from the following file:

**MiniSynchStatusRegister.h**

## LinearAddress Struct Reference

Memory Segment Descriptor.

#include <AlphiBoard.h>

### Public Attributes

void \* **Address**

*Linear address.*

size\_t **Length**

*Length of the mapping.*

### Detailed Description

Memory Segment Descriptor.

### Member Data Documentation

#### void\* LinearAddress::Address

Linear address.

#### size\_t LinearAddress::Length

Length of the mapping.

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

**AlphiBoard.h**

## Ltc2664\_av Class Reference

Synchro simulator.

#include <Ltc2664\_av.h>

### Public Member Functions

**Ltc2664\_av** (volatile void \*addr)

*Constructor.*

void **setSimGain** (int chanNbr, int16\_t val)

*Set an individual gain register.*

int16\_t **getSimGain** (int chanNbr)

*Get an individual gain register value.*

void **setAngle** (double angle)

*configure the sine amplitudes for a given angle*

double **angleToRadian** (uint32\_t intVal)

*fixed point angle to double*

uint32\_t **radianToAngle** (double rad)

*radian angle to fixed point*

double **getFrequency** (void)

void **setFrequency** (double freq)

void **setAngularVelocity** (double rad)

double **getAngularVelocity** (void)

uint32\_t **getAngularVelocityRaw** (void)

### Detailed Description

Synchro simulator.

This class allows controlling the on board simulator. It consists of 4 DAs, generating sinewaves at the proper frequency and amplitude to simulate a synchro output.

### Constructor & Destructor Documentation

#### Ltc2664\_av::Ltc2664\_av (volatile void \* *addr*)

Constructor.

To be called only by the board open.

##### Parameters

|  |  |
| --- | --- |
| *addr* | Base address of the board registers in user space. |

### Member Function Documentation

#### double Ltc2664\_av::angleToRadian (uint32\_t *angle*)

fixed point angle to double

This function converts a 17-bit angle value as a radian value as double.

##### Parameters

|  |  |
| --- | --- |
| *angle* | 17-bit fixed point angle. |

##### Return values

|  |  |
| --- | --- |
| *Double* | value equivalent |

#### double Ltc2664\_av::getAngularVelocity (void )[inline]

#### uint32\_t Ltc2664\_av::getAngularVelocityRaw (void )[inline]

#### double Ltc2664\_av::getFrequency (void )

#### int16\_t Ltc2664\_av::getSimGain (int *chanNbr*)

Get an individual gain register value.

##### Parameters

|  |  |
| --- | --- |
| *chanNbr* | Gain number register (0 to 2). If the number is invalid, the function returns 0. |

##### Return values

|  |  |
| --- | --- |
| *16-bit* | signed integer value |

#### uint32\_t Ltc2664\_av::radianToAngle (double *rad*)

radian angle to fixed point

This function converts a radian value as double as a 17-bit angle value.

##### Parameters

|  |  |
| --- | --- |
| *rad* | A double between -PI and +PI . |

##### Return values

|  |  |
| --- | --- |
| *17-bit* | fixed point angle |

#### void Ltc2664\_av::setAngle (double *angle*)

configure the sine amplitudes for a given angle

For the synchro resolver, the angle is expressed as the amplitude of 3 sine waves.

This function, if given an angle calculates the gains of the 3 DA channels and thus the amplitudes of the sine waves generated.

##### Parameters

|  |  |
| --- | --- |
| *angle* | A double between 0 and PI. |

#### void Ltc2664\_av::setAngularVelocity (double *rad*)[inline]

#### void Ltc2664\_av::setFrequency (double *freq*)

#### void Ltc2664\_av::setSimGain (int *chanNbr*, int16\_t *val*)

Set an individual gain register.

##### Parameters

|  |  |
| --- | --- |
| *chanNbr* | Gain number register (0 to 2). If the number is invalid, the function does nothing. |
| *val* | 16-bit value to put in the gain register. |

#### The documentation for this class was generated from the following files:

**Ltc2664\_av.h**

**Ltc2664\_av.cpp**

## Ltc2664\_SPI Class Reference

D/A controller SPI class definition.

#include <Ltc2664\_SPI.h>

Inheritance diagram for Ltc2664\_SPI:

### Public Types

enum **ChannelNbr** : uint8\_t { **CHANNEL\_0** = 0, **CHANNEL\_1** = 1, **CHANNEL\_2** = 2, **CHANNEL\_3** = 3, **CHANNEL\_ALL** = 15, **CHANNEL\_NONE** = 14 }

*Definition of channel number possible parameters.*

### Public Member Functions

**Ltc2664\_SPI** (volatile void \*addr, **ControlRegister** \*control)

*Constructor.*

bool **isChannelValid** (**ChannelNbr** c)

*Check that a channel is a valid selection.*

void **reset** ()

*Reset the LTC2664 chip.*

**PCIeMini\_status** **setCode** (int16\_t code, **ChannelNbr** channel, **ChannelNbr** update=**CHANNEL\_NONE**, bool echo=false)

*Set an output value to one or several D/A channels.*

### Public Attributes

uint8\_t **buff\_out** [4]

*output buffer, could be checked for verification*

uint8\_t **buff\_in** [4]

*input buffer, could be checked for verification*

**ControlRegister** \* **controlReg**

### Additional Inherited Members

### Detailed Description

D/A controller SPI class definition.

We interface to the LTC2664 chip through a standard SPI interface defined in **AlteraSpi**.

### Member Enumeration Documentation

#### enum Ltc2664\_SPI::ChannelNbr : uint8\_t

Definition of channel number possible parameters.

The commands to the LTC2664 have typically as an option the channel number, or all the channels, or if we want to disable part of a command, no channel.

##### Enumerator:

|  |  |
| --- | --- |
| CHANNEL\_0 | Channel #0, first channel. |
| CHANNEL\_1 | Channel #1, second channel. |
| CHANNEL\_2 | Channel #2, third channel. |
| CHANNEL\_3 | Channel #3, fourth channel. |
| CHANNEL\_ALL | All the channels are selected. |
| CHANNEL\_NONE | No channel, option disabled. |

### Constructor & Destructor Documentation

#### Ltc2664\_SPI::Ltc2664\_SPI (volatile void \* *addr*, ControlRegister \* *control*)[inline]

Constructor.

Only called by the board open function.

##### Parameters

|  |  |
| --- | --- |
| *addr* | Pointer to the SPI controller in user space |
| *control* | Pointer to the Control Register object |

### Member Function Documentation

#### bool Ltc2664\_SPI::isChannelValid (ChannelNbr *c*)[inline]

Check that a channel is a valid selection.

##### Parameters

|  |  |
| --- | --- |
| *c* | Channel number to check |

##### Return values

|  |  |
| --- | --- |
| *True* | if the channel is valid. |

#### void Ltc2664\_SPI::reset ()[inline]

Reset the LTC2664 chip.

#### PCIeMini\_status Ltc2664\_SPI::setCode (int16\_t *code*, ChannelNbr *channel*, ChannelNbr *update* = CHANNEL\_NONE, bool *echo* = false)

Set an output value to one or several D/A channels.

##### Parameters

|  |  |
| --- | --- |
| *code* | Value to convert to analog out. |
| *channel* | Channel number or all the channels. |
| *update* | Which output to update after updating the value. Default is CHANNEL\_NONE. |
| *echo* | Determine if we are sending a 32-bit command (echo is true) so that we can get an echo. Default is false, a 24-bit command, and no echo. |

##### Return values

|  |  |
| --- | --- |
| *ERRCODE\_NO\_ERROR* | if successful, ERRCODE\_INVALID\_VALUE if the parameters are incompatible. |

### Member Data Documentation

#### uint8\_t Ltc2664\_SPI::buff\_in[4]

input buffer, could be checked for verification

#### uint8\_t Ltc2664\_SPI::buff\_out[4]

output buffer, could be checked for verification

#### ControlRegister\* Ltc2664\_SPI::controlReg

#### The documentation for this class was generated from the following files:

**Ltc2664\_SPI.h**

**Ltc2664\_SPI.cpp**

## MINIPCIE\_DEV\_CTX Struct Reference

Minipcie Device Information Structure.

#include <AlphiBoard.h>

### Public Attributes

**MINIPCIE\_INT\_HANDLER** **funcDiagIntHandler**

*Interrupt handler routine.*

**MINIPCIE\_EVENT\_HANDLER** **funcDiagEventHandler**

*Event handler routine.*

### Detailed Description

Minipcie Device Information Structure.

### Member Data Documentation

#### MINIPCIE\_EVENT\_HANDLER MINIPCIE\_DEV\_CTX::funcDiagEventHandler

Event handler routine.

#### MINIPCIE\_INT\_HANDLER MINIPCIE\_DEV\_CTX::funcDiagIntHandler

Interrupt handler routine.

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

**AlphiBoard.h**

## PcieCra Class Reference

PCIe controller class.

#include <AlphiBoard.h>

### Public Member Functions

**PcieCra** (volatile void \*cra\_addr)

void **reset** ()

*Reset the CRA PCIe interface.*

uint32\_t **getIrqStatus** ()

*return the interrupt status of the local IRQ lines*

void **setIrqEnableMask** (uint32\_t mask)

*Enable/disable the interrupts.*

uint32\_t **getIrqEnableMask** ()

*return the interrupt enable mask*

### Detailed Description

PCIe controller class.

This is a very simplified interface that just allows to enable/disable the interrupt requests to the PCIe bus and check the status of the local interrupt request lines.

### Constructor & Destructor Documentation

#### PcieCra::PcieCra (volatile void \* *cra\_addr*)[inline]

@constructor

### Member Function Documentation

#### uint32\_t PcieCra::getIrqEnableMask ()[inline]

return the interrupt enable mask

#### uint32\_t PcieCra::getIrqStatus ()[inline]

return the interrupt status of the local IRQ lines

In order for the PCIe interface to request an interrupt on the PCIe bus, the bit needs to be set in this register, and the corresponding bit should be set in the interrupt mask register.

#### void PcieCra::reset ()[inline]

Reset the CRA PCIe interface.

Disable interrupts.

#### void PcieCra::setIrqEnableMask (uint32\_t *mask*)[inline]

Enable/disable the interrupts.

##### Parameters

|  |  |
| --- | --- |
| *mask* | bit mask of enabled interrupts |

#### The documentation for this class was generated from the following file:

**AlphiBoard.h**

## PCIeMini\_Synch Class Reference

PCIeMini\_Synchro controller board object.

#include <PCIeMini\_Synch.h>

Inheritance diagram for PCIeMini\_Synch:

### Public Member Functions

**PCIeMini\_Synch** ()

**PCIeMini\_status** **open** (int brdNbr)

*Open: connect to an actual board.*

**PCIeMini\_status** **close** ()

*Close the connection to a board object and free the resources.*

**PCIeMini\_status** **reset** ()

*Reset the board controllers.*

void **setLedPio** (uint32\_t)

uint32\_t **getLedPio** ()

### Static Public Member Functions

static char \* **getErrorMsg** (**PCIeMini\_status** errorNbr)

*Return a text description corresponding to an error code.*

### Public Attributes

volatile uint32\_t \* **ledPio**

*LED control.*

**ControlRegister** \* **controlRegister**

*Interface to the board control register.*

**StatusRegister** \* **statusRegister**

*Interface to the board status register.*

**Ltc2664\_av** \* **da**

*Interface to the DAs when used as synchro simulator.*

**Rd19231** \* **sync**

*Interface to the RD19231 Synchro resolver.*

**Ltc2664\_SPI** \* **spi\_da**

*SPI controller used by the DAs as general purpose DAs.*

### Static Public Attributes

static const uint16\_t **irq\_mask\_statusReg** = 0x0001

*Interrupt mask for the status register (used for BITn)*

static const uint16\_t **irq\_mask\_spiInterface** = 0x0004

*Interrupt mask for the SPI controller.*

### Detailed Description

PCIeMini\_Synchro controller board object.

### Constructor & Destructor Documentation

#### PCIeMini\_Synch::PCIeMini\_Synch ()

The constructor does not take any parameter. The board is not actually usable until the open method connects it to real hardware.

### Member Function Documentation

#### PCIeMini\_status PCIeMini\_Synch::close ()

Close the connection to a board object and free the resources.

##### Returns

ERRCODE\_NO\_ERROR if successful.

#### char \* PCIeMini\_Synch::getErrorMsg (PCIeMini\_status *errorNbr*)[static]

Return a text description corresponding to an error code.

##### Returns

A pointer to a null terminated character string.

#### uint32\_t PCIeMini\_Synch::getLedPio ()

#### PCIeMini\_status PCIeMini\_Synch::open (int *brdNbr*)

Open: connect to an actual board.

##### Parameters

|  |  |
| --- | --- |
| *brdNbr* | The board number is actually system dependent but if you have only one board, it should be 0. |

##### Returns

ERRCODE\_NO\_ERROR if successful.

#### PCIeMini\_status PCIeMini\_Synch::reset ()

Reset the board controllers.

##### Returns

ERRCODE\_NO\_ERROR if successful.

#### void PCIeMini\_Synch::setLedPio (uint32\_t *val*)

### Member Data Documentation

#### ControlRegister\* PCIeMini\_Synch::controlRegister

Interface to the board control register.

#### Ltc2664\_av\* PCIeMini\_Synch::da

Interface to the DAs when used as synchro simulator.

#### const uint16\_t PCIeMini\_Synch::irq\_mask\_spiInterface = 0x0004[static]

Interrupt mask for the SPI controller.

#### const uint16\_t PCIeMini\_Synch::irq\_mask\_statusReg = 0x0001[static]

Interrupt mask for the status register (used for BITn)

#### volatile uint32\_t\* PCIeMini\_Synch::ledPio

LED control.

#### Ltc2664\_SPI\* PCIeMini\_Synch::spi\_da

SPI controller used by the DAs as general purpose DAs.

#### StatusRegister\* PCIeMini\_Synch::statusRegister

Interface to the board status register.

#### Rd19231\* PCIeMini\_Synch::sync

Interface to the RD19231 Synchro resolver.

#### The documentation for this class was generated from the following files:

**PCIeMini\_Synch.h**

**PCIeMini\_Synch.cpp**

## Rd19231 Class Reference

DDC RD19231 controller class.

#include <Rd19231.h>

### Public Member Functions

**Rd19231** (volatile void \*rd\_addr, **StatusRegister** \*status, **ControlRegister** \*control)

*Constructor.*

uint32\_t **getRawPos** ()

*Read the Synchro raw binary output.*

double **getPos** ()

*Get angular position.*

uint32\_t **getCycleTime** ()

*Maintenance only.*

uint32\_t **getBusyTime** ()

*Maintenance only.*

bool **isBITnValid** ()

*Check if output is valid or if BIT error.*

uint32\_t **getEncoderCounter** ()

*Get the encoder counter.*

void **reset** ()

*for compatibility only*

### Detailed Description

DDC RD19231 controller class.

The board hardware reads regularly the position data output to make sure the read position function returns the freshest data possible.

### Constructor & Destructor Documentation

#### Rd19231::Rd19231 (volatile void \* *rd\_addr*, StatusRegister \* *status*, ControlRegister \* *control*)[inline]

Constructor.

only called by the board open function

### Member Function Documentation

#### uint32\_t Rd19231::getBusyTime ()[inline]

Maintenance only.

#### uint32\_t Rd19231::getCycleTime ()[inline]

Maintenance only.

#### uint32\_t Rd19231::getEncoderCounter ()[inline]

Get the encoder counter.

Counting the A and B output pulses. Refer to the Synchro manual

#### double Rd19231::getPos ()[inline]

Get angular position.

##### Return values

|  |  |
| --- | --- |
| *Position* | in degree, from 0 to 360 degrees |

#### uint32\_t Rd19231::getRawPos ()[inline]

Read the Synchro raw binary output.

See **getPos()** for the output translated in degree

##### Return values

|  |  |
| --- | --- |
| *A* | 16-bit binary number |

#### bool Rd19231::isBITnValid ()[inline]

Check if output is valid or if BIT error.

The BIT error is triggered if any of the following conditions exist : ~180 LSBs of positive error, ~180 LSBs of negative error, Loss of Signal(LOS), or Loss of Reference(LOR) is less than 500 mVp, or a false null occurs when the phase detect circuitry causes a BIT and corrects the error.Logic 0 for fault condition.

##### Return values

|  |  |
| --- | --- |
| *True* | if valid read, False if there is a BIT error |

#### void Rd19231::reset ()[inline]

for compatibility only

#### The documentation for this class was generated from the following file:

**Rd19231.h**

## StatusRegister Class Reference

Status register class for the PCIeMini board.

#include <MiniSynchStatusRegister.h>

Inheritance diagram for StatusRegister:

### Public Member Functions

**StatusRegister** (volatile void \*addr)

bool **isBITnValid** ()

*Check the synchro error flag (BIT)*

void **enableBitValidIrq** ()

*Enable the synchro error interrupt.*

void **disableBitValidIrq** ()

*Disable the synchro error interrupt.*

### Static Public Attributes

static const uint32\_t **STATUS\_BIT\_VALID\_MASK** = 0x01

### Detailed Description

Status register class for the PCIeMini board.

### Constructor & Destructor Documentation

#### StatusRegister::StatusRegister (volatile void \* *addr*)[inline]

### Member Function Documentation

#### void StatusRegister::disableBitValidIrq ()[inline]

Disable the synchro error interrupt.

#### void StatusRegister::enableBitValidIrq ()[inline]

Enable the synchro error interrupt.

The interrupt is generated on the error signal becoming active (BITn going low)

#### bool StatusRegister::isBITnValid ()[inline]

Check the synchro error flag (BIT)

##### Return values

|  |  |
| --- | --- |
| *true* | if the sysnchro output is not in error |

### Member Data Documentation

#### const uint32\_t StatusRegister::STATUS\_BIT\_VALID\_MASK = 0x01[static]

#### The documentation for this class was generated from the following file:

**MiniSynchStatusRegister.h**

# Index

~AlphiBoard

AlphiBoard, 6

Address

LinearAddress, 21

AlphiBoard, 4

~AlphiBoard, 6

AlphiBoard, 6

bar0, 9

bar2, 9

Close, 6

cra, 9

disableInterrupts, 6

enableInterrupts, 6

getBar0Address, 7

getBar2Address, 7

getFpgaID, 7

getFpgaTimeStamp, 7

hookInterruptServiceRoutine, 7, 8

IsValidDevice, 8

libStatus, 9

MsSleep, 8

Open, 8

reset, 8

setVerbose, 9

sysid, 9

unhookInterruptServiceRoutine, 9

verbose, 9

AlteraPio, 11

AlteraPio, 11

CAP\_INPUT, 12

CAP\_INPUT\_OUTPUT, 12

CAP\_OUTPUT, 12

clearEdgeCapture, 12

getData, 12

getEdgeCapture, 12

getIrqMask, 12

reset, 12

setData, 12

setIrqMask, 12

AlteraSpi, 13

AlteraSpi, 14

getRxData, 14

getStatus, 14

selectSlave, 14

sendSpiCommand, 14

setControl, 14

setTxData, 14

status\_ROE\_mask, 15

status\_RRDY\_mask, 15

status\_TMT\_mask, 15

status\_TOE\_mask, 15

status\_TRDY\_mask, 15

angleToRadian

Ltc2664\_av, 23

bar0

AlphiBoard, 9

bar2

AlphiBoard, 9

BoardVersion, 16

BoardVersion, 16

getTimeStamp, 16

getVersion, 16

buff\_in

Ltc2664\_SPI, 27

buff\_out

Ltc2664\_SPI, 27

CAP\_INPUT

AlteraPio, 12

CAP\_INPUT\_OUTPUT

AlteraPio, 12

CAP\_OUTPUT

AlteraPio, 12

CHANNEL\_0

Ltc2664\_SPI, 26

CHANNEL\_1

Ltc2664\_SPI, 26

CHANNEL\_2

Ltc2664\_SPI, 26

CHANNEL\_3

Ltc2664\_SPI, 26

CHANNEL\_ALL

Ltc2664\_SPI, 26

CHANNEL\_NONE

Ltc2664\_SPI, 26

ChannelNbr

Ltc2664\_SPI, 26

clearEdgeCapture

AlteraPio, 12

close

PCIeMini\_Synch, 32

Close

AlphiBoard, 6

controlReg

Ltc2664\_SPI, 27

controlRegister

PCIeMini\_Synch, 33

ControlRegister, 18

ControlRegister, 19

CTRL\_AQB\_mask, 19

CTRL\_BITn\_MASK, 19

CTRL\_DA\_CLEAR\_mask, 19

CTRL\_DaMode\_mask, 19

CTRL\_DSR\_mask, 19

CTRL\_LDAC\_mask, 19

enableSpiDa, 19

RES\_10\_BIT, 18

RES\_12\_BIT, 18

RES\_14\_BIT, 18

RES\_16\_BIT, 18

Resolution, 18

ShiftControl, 19

VEL1, 19

VEL2, 19

cra

AlphiBoard, 9

CTRL\_AQB\_mask

ControlRegister, 19

CTRL\_BITn\_MASK

ControlRegister, 19

CTRL\_DA\_CLEAR\_mask

ControlRegister, 19

CTRL\_DaMode\_mask

ControlRegister, 19

CTRL\_DSR\_mask

ControlRegister, 19

CTRL\_LDAC\_mask

ControlRegister, 19

da

PCIeMini\_Synch, 33

disableBitValidIrq

StatusRegister, 38

disableInterrupts

AlphiBoard, 6

enableBitValidIrq

StatusRegister, 38

enableInterrupts

AlphiBoard, 6

enableSpiDa

ControlRegister, 19

funcDiagEventHandler

MINIPCIE\_DEV\_CTX, 28

funcDiagIntHandler

MINIPCIE\_DEV\_CTX, 28

getAngularVelocity

Ltc2664\_av, 23

getAngularVelocityRaw

Ltc2664\_av, 23

getBar0Address

AlphiBoard, 7

getBar2Address

AlphiBoard, 7

getBusyTime

Rd19231, 36

getCycleTime

Rd19231, 36

getData

AlteraPio, 12

getEdgeCapture

AlteraPio, 12

getEncoderCounter

Rd19231, 36

getErrorMsg

PCIeMini\_Synch, 32

getFpgaID

AlphiBoard, 7

getFpgaTimeStamp

AlphiBoard, 7

getFrequency

Ltc2664\_av, 23

getIrqEnableMask

PcieCra, 29

getIrqMask

AlteraPio, 12

getIrqStatus

PcieCra, 29

getLedPio

PCIeMini\_Synch, 32

getPos

Rd19231, 36

getRawPos

Rd19231, 36

getRxData

AlteraSpi, 14

getSimGain

Ltc2664\_av, 23

getStatus

AlteraSpi, 14

getTimeStamp

BoardVersion, 16

getVersion

BoardVersion, 16

hookInterruptServiceRoutine

AlphiBoard, 7, 8

irq\_mask\_spiInterface

PCIeMini\_Synch, 33

irq\_mask\_statusReg

PCIeMini\_Synch, 33

isBITnValid

Rd19231, 36

StatusRegister, 39

isChannelValid

Ltc2664\_SPI, 26

IsValidDevice

AlphiBoard, 8

ledPio

PCIeMini\_Synch, 33

Length

LinearAddress, 21

libStatus

AlphiBoard, 9

LinearAddress, 21

Address, 21

Length, 21

Ltc2664\_av, 22

angleToRadian, 23

getAngularVelocity, 23

getAngularVelocityRaw, 23

getFrequency, 23

getSimGain, 23

Ltc2664\_av, 22

radianToAngle, 23

setAngle, 23

setAngularVelocity, 24

setFrequency, 24

setSimGain, 24

Ltc2664\_SPI, 25

buff\_in, 27

buff\_out, 27

CHANNEL\_0, 26

CHANNEL\_1, 26

CHANNEL\_2, 26

CHANNEL\_3, 26

CHANNEL\_ALL, 26

CHANNEL\_NONE, 26

ChannelNbr, 26

controlReg, 27

isChannelValid, 26

Ltc2664\_SPI, 26

reset, 27

setCode, 27

MINIPCIE\_DEV\_CTX, 28

funcDiagEventHandler, 28

funcDiagIntHandler, 28

MsSleep

AlphiBoard, 8

open

PCIeMini\_Synch, 32

Open

AlphiBoard, 8

PcieCra, 29

getIrqEnableMask, 29

getIrqStatus, 29

PcieCra, 29

reset, 30

setIrqEnableMask, 30

PCIeMini\_Synch, 31

close, 32

controlRegister, 33

da, 33

getErrorMsg, 32

getLedPio, 32

irq\_mask\_spiInterface, 33

irq\_mask\_statusReg, 33

ledPio, 33

open, 32

PCIeMini\_Synch, 32

reset, 33

setLedPio, 33

spi\_da, 33

statusRegister, 33

sync, 34

radianToAngle

Ltc2664\_av, 23

Rd19231, 35

getBusyTime, 36

getCycleTime, 36

getEncoderCounter, 36

getPos, 36

getRawPos, 36

isBITnValid, 36

Rd19231, 35

reset, 36

RES\_10\_BIT

ControlRegister, 18

RES\_12\_BIT

ControlRegister, 18

RES\_14\_BIT

ControlRegister, 18

RES\_16\_BIT

ControlRegister, 18

reset

AlphiBoard, 8

AlteraPio, 12

Ltc2664\_SPI, 27

PcieCra, 30

PCIeMini\_Synch, 33

Rd19231, 36

Resolution

ControlRegister, 18

selectSlave

AlteraSpi, 14

sendSpiCommand

AlteraSpi, 14

setAngle

Ltc2664\_av, 23

setAngularVelocity

Ltc2664\_av, 24

setCode

Ltc2664\_SPI, 27

setControl

AlteraSpi, 14

setData

AlteraPio, 12

setFrequency

Ltc2664\_av, 24

setIrqEnableMask

PcieCra, 30

setIrqMask

AlteraPio, 12

setLedPio

PCIeMini\_Synch, 33

setSimGain

Ltc2664\_av, 24

setTxData

AlteraSpi, 14

setVerbose

AlphiBoard, 9

ShiftControl

ControlRegister, 19

spi\_da

PCIeMini\_Synch, 33

STATUS\_BIT\_VALID\_MASK

StatusRegister, 39

status\_ROE\_mask

AlteraSpi, 15

status\_RRDY\_mask

AlteraSpi, 15

status\_TMT\_mask

AlteraSpi, 15

status\_TOE\_mask

AlteraSpi, 15

status\_TRDY\_mask

AlteraSpi, 15

statusRegister

PCIeMini\_Synch, 33

StatusRegister, 38

disableBitValidIrq, 38

enableBitValidIrq, 38

isBITnValid, 39

STATUS\_BIT\_VALID\_MASK, 39

StatusRegister, 38

sync

PCIeMini\_Synch, 34

sysid

AlphiBoard, 9

unhookInterruptServiceRoutine

AlphiBoard, 9

VEL1

ControlRegister, 19

VEL2

ControlRegister, 19

verbose

AlphiBoard, 9