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

Table of Contents

Hierarchical Index

Class Hierarchy

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

Class Index

Class List

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

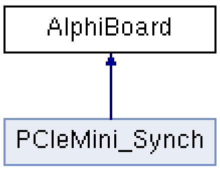
Class Documentation

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:

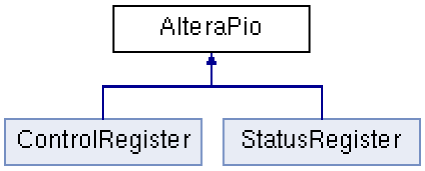
- **AlphiBoard.h**
- **AlphiBoard.cpp**
- **AlphiBoard_irq.cpp**

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:

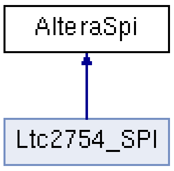
- **AlteraPio.h**

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:

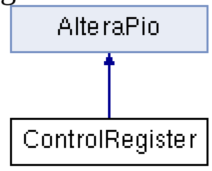
- **AlphiBoard.h**
- **AlphiBoard.cpp**

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:

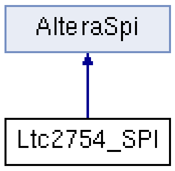
- **Ltc2664_av.h**
- **Ltc2664_av.cpp**

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:

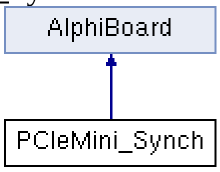
- **AlphiBoard.h**

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]`

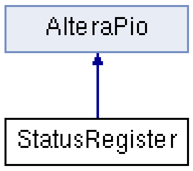
for compatibility only

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