PCIeMini-ESCC

2- Channels
Serial Communication Controller

PCIexpress Mini

927-10-003-4000

Software Manual

Version 1.0
08/06/2020

NOTICE

Table of Contents

Class Index

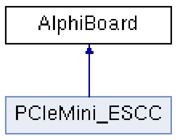
Class List

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

Class Documentation

AlphiBoard Class Reference

Base class implementing a PCI board connection to the 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.
- void **setVerbose** (int **verbose**)
  set the verbose flag
- bool **IsValidDevice** (const CHAR *sFunc)
  Validate a WDC device handle.
- DWORD **Map** (int bar, **LinearAddress** *addr)
  Establishes a connection to a board.
- DWORD **Unmap** (**LinearAddress** &Address)
  Release a memory segment.
- DWORD **HookMailboxInterrupt** (uint32_t mask, **MINIPCIE_INT_HANDLER** uicr, void *userData)
  Setup the interrupt of the board.
- DWORD **UnhookMailboxInterrupt** ()
  Disable the board interrupt.
- DWORD **DisableInterrupts** ()
  Disable PCIe interrupts.
- DWORD **EnableInterrupts** ()
  Enable 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

- int **verbose**

Protected Attributes

- **LinearAddress bar0**
  Memory descriptor for the BAR0 in user memory.

- **LinearAddress bar2**
  Memory descriptor for the BAR2 in user memory.

Detailed Description

Base class implementing a PCI board connection to the 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.

Return values

Status	code

DWORD AlphiBoard::EnableInterrupts ()

Enable PCIe interrupts.

Enable the generation of PCIe interrupts by the PCIe interface only. This is a low level function that does not do anything board specific for the interrupt generation.

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

DWORD AlphiBoard::HookMailboxInterrupt (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.

DWORD AlphiBoard::Map (int *bar*, LinearAddress * *addr*)

Establishes a connection to a board.

Returns a pointer to an address space. Only BAR 0 and 2 are recognized.

Parameters

bar	The number of the bar to access.
addr	The **LinearAddress** structure where to put the memory information.

Returns

WD_STATUS_SUCCESS when the bar is accessible WD_NO_RESOURCES_ON_DEVICE if there is no corresponding BAR.

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

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::UnhookMailboxInterrupt ()

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.

DWORD AlphiBoard::Unmap (LinearAddress & *Address*)

Release a memory segment.

Not used with the Jungo driver

Return values

WD_STATUS_SUCCESS	

Member Data Documentation

LinearAddress AlphiBoard::bar0 `[protected]`

Memory descriptor for the BAR0 in user memory.

LinearAddress AlphiBoard::bar2 `[protected]`

Memory descriptor for the BAR2 in user memory.

int AlphiBoard::verbose

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

- PCIe-Mini-SCC2/include/**AlphiBoard.h**
- PCIe-Mini-SCC2/PCIeMini_ESCC_lib/**AlphiBoard.cpp**
- PCIe-Mini-SCC2/PCIeMini_ESCC_lib/**AlphiBoard_irq.cpp**

BoardVersion Class Reference

Board Hardware identification and version.
`#include <PCIeMini_ESCC.h>`

Public Member Functions

- **BoardVersion** (volatile uint32_t *addr)
- 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*)

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:

- PCIe-Mini-SCC2/include/**PCIeMini_ESCC.h**
- PCIe-Mini-SCC2/PCIeMini_ESCC_lib/**PCIeMini_ESCC.cpp**

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:

- PCIe-Mini-SCC2/include/**AlphiBoard.h**

LTC2872 Class Reference

Implementation of the **LTC2872** buffer.

`#include <LTC2872.h>`

Public Member Functions

- **LTC2872** (volatile void *devAddr)
  Constructor.

- void **setBuffer** (uint32_t settings)
  Write the buffer configuration.

- uint32_t **getBuffer** ()
  Read the buffer configuration.

Public Attributes

- volatile uint32_t * **addr**
  Address of the buffer.

Static Public Attributes

- static const uint16_t **recvDisable** = 0x0001
  Receiver Disable A logic high disables RS232 and RS485 receivers in transceiver #1. A logic low enables the RS232 or RS485 receivers in the transceiver #1, depending on the state of the Interface Select Input 485/232_1.

- static const uint16_t **halfDuplexEnable** = 0x0002
  S485 Half-duplex Select Input.

- static const uint16_t **rs485Mode** = 0x0004
  Interface Select.

- static const uint16_t **fastMode** = 0x0008
  Fast mode enable.

- static const uint16_t **loopbackEnable** = 0x0010
  Loopback Enable.

- static const uint16_t **terminationEnable** = 0x0020
  RS485 Termination Enable for Transceiver.

- static const uint16_t **driverModeMask** = 0x00c0
  Drivers Enable.

- static const uint16_t **dcdInputDisable** = 0x0100
  DCD Input Disable.

- static const uint16_t **clockSelect** = 0x0200

SCC clock selection.

- static const uint16_t **RS_232_buffers** = 0x0348
  Set the output buffers to RS-232, Fast mode, driver 1 enable, DCD disable.

- static const uint16_t **RS_422_buffers** = 0x036c
  Set the output buffers to RS-422/485, Fast mode, driver 1 enable, DCD disable.

- static const uint16_t **RS_422_localLoopback** = 0x035c
  Set the output buffers to RS-422/485, Fast mode, driver 1 enable. Local Loop back mode, DCD disable.

- static const uint16_t **RS_422_pullups_buffers** = 0x37c
  Set the output buffers to RS-422/485, Fast mode, pull-ups enabled, driver 1 enable, DCD disable.

Detailed Description

Implementation of the **LTC2872** buffer.

The SCC use a transceiver to transform the serial interface signals to the proper voltages and format. This class allows to control the transceivers.

Constructor & Destructor Documentation

LTC2872::LTC2872 (volatile void * *devAddr*)`[inline]`

Constructor.

To be used only by the **SccChannel** constructor.

Parameters

devAddr	Pointer to the device mapped in user memory.

Member Function Documentation

uint32_t LTC2872::getBuffer ()`[inline]`

Read the buffer configuration.

Return values

A	16-bit unsigned containing the bit mapped settings for the buffer.

void LTC2872::setBuffer (uint32_t *settings*)`[inline]`

Write the buffer configuration.

Parameters

settings	A 16-bit unsigned containing the bit mapped settings for the buffer.

Member Data Documentation

volatile uint32_t* LTC2872::addr

Address of the buffer.

const uint16_t LTC2872::clockSelect = 0x0200 `[static]`

SCC clock selection.

when 1 the transmission clock is the local 14.7456MHz, when 0, it is the user provided clock.

const uint16_t LTC2872::dcdInputDisable = 0x0100 `[static]`

DCD Input Disable.

A logic high (1) force the DCD input of the ESCC to go low (active). When this bit is low the DCD input of the chip is the DCD input from the board interface.

const uint16_t LTC2872::driverModeMask = 0x00c0 `[static]`

Drivers Enable.

A logic low disables the RS232 and RS485 drivers, leaving their outputs in a Hi-Z state. A logic high enables the RS232 or RS485 drivers in transceiver #1, depending on the state of the Interface Select Input 485/232_1.

const uint16_t LTC2872::fastMode = 0x0008 `[static]`

Fast mode enable.

A logic high enable continuous voltage generation inside the buffer hardware. When the bit is 0 the DC/DC converters are on as needed which slows down slightly the performance, in exchange for a small gain in power consumption. This bit should be kept set.

const uint16_t LTC2872::halfDuplexEnable = 0x0002 `[static]`

S485 Half-duplex Select Input.

A logic low is used for full duplex operation where pins A and B are the receiver inputs and pins Y and Z are the driver outputs. A logic high is used for half duplex operation where pins Y and Z are both the receiver inputs and driver outputs and pins A and B do not serve as the receiver inputs. The impedance on A and B and state of differential termination between A and B is independent of the state of H/F. The H/F pin has no effect on RS232 operation.

const uint16_t LTC2872::loopbackEnable = 0x0010 `[static]`

Loopback Enable.

A logic high enables Logic Loopback diagnostic mode, internally routing the driver input logic levels to the receiver output pins within the same transceiver. This applies to both RS232 channels as well as the RS485 driver/receiver. The targeted receiver must be enabled for the loopback signal to be available on its output. A logic low disables loopback mode. In loopback mode, signals are not inverted from driver inputs to receiver outputs.

const uint16_t LTC2872::recvDisable = 0x0001 `[static]`

Receiver Disable A logic high disables RS232 and RS485 receivers in transceiver #1. A logic low enables the RS232 or RS485 receivers in the transceiver #1, depending on the state of the Interface Select Input 485/232_1.

const uint16_t LTC2872::rs485Mode = 0x0004 `[static]`

Interface Select.

A logic low enables RS232 mode and a high enables RS485 mode. The mode determines which transceiver inputs and outputs are accessible at the **LTC2872** pins as well as which is controlled by the driver and receiver enable pins.

const uint16_t LTC2872::RS_232_buffers = 0x0348 `[static]`

Set the output buffers to RS-232, Fast mode, driver 1 enable, DCD disable.

const uint16_t LTC2872::RS_422_buffers = 0x036c `[static]`

Set the output buffers to RS-422/485, Fast mode, driver 1 enable, DCD disable.

const uint16_t LTC2872::RS_422_localLoopback = 0x035c `[static]`

Set the output buffers to RS-422/485, Fast mode, driver 1 enable. Local Loop back mode, DCD disable.

const uint16_t LTC2872::RS_422_pullups_buffers = 0x37c `[static]`

Set the output buffers to RS-422/485, Fast mode, pull-ups enabled, driver 1 enable, DCD disable.

const uint16_t LTC2872::terminationEnable = 0x0020 `[static]`

RS485 Termination Enable for Transceiver.

A logic high enables a 120? resistor between pins A1 and B1. If DZ1 is also high, a 120 Ohm resistor is enabled between pins Y1 and Z1. A logic low on TE485_1 opens the resistors, leaving A1/B1 and Y1/Z1 unterminated, independent of DZ1. The differential termination resistors are never enabled in RS232 mode.

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

- PCIe-Mini-SCC2/include/**LTC2872.h**

MINIPCIE_DEV_CTX Struct Reference

```
#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:

- PCIe-Mini-SCC2/include/**AlphiBoard.h**

MINIPCIE_INT_RESULT Struct Reference

Interrupt result information structure.
`#include <minipcie_arinc429_lib.h>`

Public Attributes

- DWORD **dwCounter**
  Number of interrupts received.
- DWORD **dwLost**
  Number of interrupts not yet handled.
- WD_INTERRUPT_WAIT_RESULT **waitResult**
  See WD_INTERRUPT_WAIT_RESULT values in windrvr.h.
- DWORD **dwEnabledIntType**
  Interrupt type that was actually enabled (MSI/MSI-X/Level Sensitive/Edge-Triggered)
- DWORD **dwLastMessage**

Detailed Description

Interrupt result information structure.

Member Data Documentation

DWORD MINIPCIE_INT_RESULT::dwCounter

Number of interrupts received.

DWORD MINIPCIE_INT_RESULT::dwEnabledIntType

Interrupt type that was actually enabled (MSI/MSI-X/Level Sensitive/Edge-Triggered)

DWORD MINIPCIE_INT_RESULT::dwLastMessage

Message data of the last received MSI/MSI-X (Windows Vista and higher); N/A to line-based interrupts)

DWORD MINIPCIE_INT_RESULT::dwLost

Number of interrupts not yet handled.

WD_INTERRUPT_WAIT_RESULT MINIPCIE_INT_RESULT::waitResult

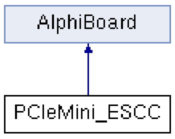
See WD_INTERRUPT_WAIT_RESULT values in windrvr.h.

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

- PCIe-Mini-SCC2/include/**minipcie_arinc429_lib.h**

PCIeMini_ESCC Class Reference

PCIeMini_ESCC controller board object.
`#include <PCIeMini_ESCC.h>`
Inheritance diagram for PCIeMini_ESCC:



Public Member Functions

- **PCIeMini_ESCC** ()
- **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 ARINC 429 controllers.
- uint32_t **getFpgaID** ()
  Get the FPGA ID of.
- time_t **getFpgaTimeStamp** ()
  Return the timestamp corresponding to when the FPGA was compiled.
- void **setLedPio** (uint32_t)
- uint32_t **getLedPio** ()
- **PCIeMini_status hookInterruptServiceRoutine** (**MINIPCIE_INT_HANDLER** uicr)
  Set an interrupt handling routine.
- **PCIeMini_status unhookInterruptServiceRoutine** ()
  Remove the connection to a interrupt handling routine.
- **PCIeMini_status enableInterrupts** ()
  Enable the interrupts from the board.
- **PCIeMini_status disableInterrupts** ()
  Disable the interrupts from the board.
- **SccChannel * getScc** (int channelNbr)
  get pointer to an instance serial channel object.
- void **sccRegisterRead** (int chan, uint8_t regNbr, volatile uint8_t *val)
- void **sccRegisterWrite** (int chan, uint8_t regNbr, uint8_t val)
- void **enableRxDma** (int chan)
- void **disableRxDma** (int chan)
- void **enableTxDma** (int chan)
- void **disableTxDma** (int chan)

- void **resetChannel** (int chan)
- void **resetChip** (int chan)
- int **initChannel** (int chan)

Static Public Member Functions

- static char * **getErrorMsg** (**PCIeMini_status** errorNbr)
  Return a text description corresponding to an error code.

Public Attributes

- **BoardVersion** * **sysid**
  Board identification.

- volatile uint32_t * **ledPio**
  LED control.

- **SccChannel** * **sccDevice_0**
  Descriptor for first channel.

- **SccChannel** * **sccDevice_1**
  Descriptor for second channel.

- **UartChannelConfig scc_config** [2]
- volatile uint32_t * **pcieIrqStatus**
  PCIe interface interrupt status.

- volatile uint32_t * **pcieIrqEnable**
  PCIe interface interrupt enable.

Additional Inherited Members

Detailed Description

PCIeMini_ESCC controller board object.

Constructor & Destructor Documentation

PCIeMini_ESCC::PCIeMini_ESCC ()

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_ESCC::close ()

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

Returns

ERRCODE_NO_ERROR if successful.

PCIeMini_status PCIeMini_ESCC::disableInterrupts ()

Disable the interrupts from the board.

Returns

ERRCODE_NO_ERROR if successful.

void PCIeMini_ESCC::disableRxDma (int *chan*)`[inline]`

void PCIeMini_ESCC::disableTxDma (int *chan*)`[inline]`

PCIeMini_status PCIeMini_ESCC::enableInterrupts ()

Enable the interrupts from the board.

Returns

ERRCODE_NO_ERROR if successful.

void PCIeMini_ESCC::enableRxDma (int *chan*)`[inline]`

void PCIeMini_ESCC::enableTxDma (int *chan*)`[inline]`

char * PCIeMini_ESCC::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_ESCC::getFpgaID ()

Get the FPGA ID of.

Returns

The FPGA ID.

time_t PCIeMini_ESCC::getFpgaTimeStamp ()

Return the timestamp corresponding to when the FPGA was compiled.

Returns

a timestamp.

uint32_t PCIeMini_ESCC::getLedPio ()

SccChannel * PCIeMini_ESCC::getScc (int *channelNbr*)

get pointer to an instance serial channel object.

Parameters

channelNbr	Channel number 0, or 1.

Returns

A pointer to a channel object if successful, else NULL.

PCIeMini_status PCIeMini_ESCC::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.

int PCIeMini_ESCC::initChannel (int *chan*)

PCIeMini_status PCIeMini_ESCC::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_ESCC::reset ()

Reset the board ARINC 429 controllers.

Returns

ERRCODE_NO_ERROR if successful.

void PCIeMini_ESCC::resetChannel (int *chan*)`[inline]`

void PCIeMini_ESCC::resetChip (int *chan*)`[inline]`

Reset the chip and FIFOs

void PCIeMini_ESCC::sccRegisterRead (int *chan*, uint8_t *regNbr*, volatile uint8_t * *val*)`[inline]`

void PCIeMini_ESCC::sccRegisterWrite (int *chan*, uint8_t *regNbr*, uint8_t *val*)`[inline]`

void PCIeMini_ESCC::setLedPio (uint32_t *val*)

PCIeMini_status PCIeMini_ESCC::unhookInterruptServiceRoutine ()

Remove the connection to a interrupt handling routine.

Returns

ERRCODE_NO_ERROR if successful.

Member Data Documentation

volatile uint32_t* PCIeMini_ESCC::ledPio

LED control.

volatile uint32_t* PCIeMini_ESCC::pcieIrqEnable

PCIe interface interrupt enable.

volatile uint32_t* PCIeMini_ESCC::pcieIrqStatus

PCIe interface interrupt status.

UartChannelConfig PCIeMini_ESCC::scc_config[2]

SccChannel* PCIeMini_ESCC::sccDevice_0

Descriptor for first channel.

SccChannel* PCIeMini_ESCC::sccDevice_1

Descriptor for second channel.

BoardVersion* PCIeMini_ESCC::sysid

Board identification.

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

- PCIe-Mini-SCC2/include/**PCIeMini_ESCC.h**
- PCIe-Mini-SCC2/PCIeMini_ESCC_lib/**PCIeMini_ESCC.cpp**

RxFifoData Class Reference

```
#include <RxFifoData.h>
```

Public Member Functions

- uint32_t **getData** ()
- uint16_t **getRcvdValid** ()
- uint16_t **getCurrValid** ()

Member Function Documentation

uint16_t RxFifoData::getCurrValid () `[inline]`

uint32_t RxFifoData::getData () `[inline]`

uint16_t RxFifoData::getRcvdValid () `[inline]`

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

- PCIe-Mini-SCC2/include/**RxFifoData.h**

SccChannel Class Reference

Low-level SCC access class.
`#include <SccChannel.h>`

Public Member Functions

- **SccChannel** (volatile void ***baseAddress**)
  Constructor.
- **PCIeMini_status reset** ()
  Reset a channel pair.
- void **resetChannel** ()
  Reset FIFOs and disable DMAs.
- void **set_serialBuffers** (uint32_t val)
  specify the serial port buffer configuration, RS232, RS422, or RS485.
- int32_t **sccRegisterRead** (uint8_t regNbr, volatile uint8_t *val)
  Read a UART read register.
- int32_t **sccRegisterWrite** (uint8_t regNbr, uint8_t val)
  Write to a UART write register.
- int32_t **sccDataRead** (uint8_t *val)
  Read the UART receive register.
- int32_t **sccDataWrite** (uint8_t val)
  Write to the UART transmit register.
- int **channelLoad** (uint8_t *rtable)
  Load list or register number and value pairs in the 8530 after a reset.
- **PCIeMini_status config** (**UartChannelConfig** *config)
  Change an SCC channel configuration.
- uint32_t **setSccControlRegister** (uint32_t mask)
- uint32_t **resetSccControlRegister** (uint32_t mask)
- uint32_t **getSccControlRegister** ()
- int32_t **enableRxDma** ()
  Enable the Receive FIFO.
- int32_t **disableRxDma** ()
  Disable the Receive FIFO.
- int32_t **enableTxDma** ()
  Enable the Transmit FIFO.

- int32_t **disableTxDma** ()
  Disable the Transmit FIFO.

- int32_t **enableRTS** ()
  Asserts RTS.

- int32_t **disableRTS** ()
  Negate RTS.

- void **outch** (uint8_t val)
  Transmit a data byte.

- int **outchnw** (uint8_t val)
  Transmit a data byte, if possible.

- int **inchnw** ()
  Receive a data byte, if possible.

- int **inch** ()
  Receive a data byte.

- int **inch** (int pollDelay)
  Receive a data byte.

- char * **gets_s** (char ***buffer**, size_t n)
  Receive a line terminated by the EOL character.

- int **puts** (const char *str)
  Transmit a 0 terminated string.

- uint8_t **brgDivLow** (int **SCC_clocksource**, int div, int baudRate)
  Utility baud rate calculator.

- **PCIeMini_status enableIntSCC** (uint8_t mask)
- **PCIeMini_status disableIntSCC** (uint8_t mask)
  This subroutine disables ESCC interrupts.

- void **dump_regs** ()
  Dump the content of the SCC registers on stdout.

Public Attributes

- volatile void * **baseAddress**
  Pointer to the channel hardware.

- **UartChannelConfig scc_config**
  Copy of the SCC configuration parameters.

- **LTC2872 * buffer**

Pointer to the buffer control.

- **SccFifo * rxFifo**
  Receiver FIFO.

- **SccFifo * txFifo**
  Transmitter FIFO.

- uint8_t **cachedRegs** [32]

Static Public Attributes

- static const int **SCC_clocksource** = 16000000
  PCLK frequency used for baud rate calculations.

- static const int **RTxcFrequency** = 14745600
  TxC input, should be 14.745600 MHz. Used to calculate the value for the divider register.

- static const uint32_t **sccControlRxFifoIrqMask** = 0x040000
  status of receive FIFO interrupt request

- static const uint32_t **sccControlTxFifoIrqMask** = 0x020000
  status of transmit FIFO interrupt request

- static const uint32_t **sccControlSccIrqMask** = 0x010000
  status of the interrupt requests from the ESCC

- static const uint32_t **sccControlMainResetMask** = 0x008000
  When set, do a hard reset on the SCC logic. Must be manually set to 0 to allow operations.

- static const uint32_t **sccControlFilterOnMask** = 0x002000
  Filter the escaped bisync characters.

- static const uint32_t **sccControlTxDmaEnableMask** = 0x001000
  Use the FIFO to transmit characters.

- static const uint32_t **sccControlRxDmaEnableMask** = 0x000800
  Use the FIFO to receive characters.

- static const uint32_t **sccControlIrqEnableMask** = 0x000400
  General interrupt enable.

- static const uint32_t **sccControlStripcharRegMask** = 0x00ff
  Bisync character to strip.

Detailed Description

Low-level SCC access class.

This class contains low-level functions to access an SCC controller with a FIFO input and a FIFO output. It has no provision for DMA however the data can be either read or written directly to the SCC data registers, or to/from a FIFO that will send the data to the SCC using the SCC DMA mechanism. Going through the FIFO allows 32-bit read and write operations.

Constructor & Destructor Documentation

SccChannel::SccChannel (volatile void * *addr*)

Constructor.

Does a minimum of initialization. Does not reset the serial channel.

Parameters

addr	Base address of the channel interface in user space.

Member Function Documentation

uint8_t SccChannel::brgDivLow (int *SCC_clocksource*, int *div*, int *baudRate*)

Utility baud rate calculator.

Calculate a value based on the the requested baud rate and on the 8530 main divider. the function uses the SCC clock frequency to do the calculation.

Returns

The value to put in the divider low byte.

int SccChannel::channelLoad (uint8_t * *rtable*)

Load list or register number and value pairs in the 8530 after a reset.

Parameters

rtable	Zero-terminated list of character containing register number and value to be loaded. Please note that the alternate registers are indicated by adding 16 to the register number they shadow: for instance 7+16 refers to the register 7'. Please refer to the UART manual for more description.

errno_t SccChannel::config (UartChannelConfig * *config*)

Change an SCC channel configuration.

Scc_Config allows setting new UART parameters. placed in pSizeRead.

Parameters

config	New channel configuration.

Returns

If successful, Scc_Config returns 0. If the function fails, it returns a nonzero value.

PCIeMini_status SccChannel::disableIntSCC (uint8_t *mask*)

This subroutine disables ESCC interrupts.

Parameters

mask	Bits corresponding to interrupts to disable.

Return values

Error	status (always ERRCODE_NO_ERROR)

int SccChannel::disableRTS ()

Negate RTS.

int SccChannel::disableRxDma ()

Disable the Receive FIFO.

```
When the FIFO is disabled, The received characters must be retrieved by accessing
the 8530 SCC.
```

int SccChannel::disableTxDma ()

Disable the Transmit FIFO.

```
When the FIFO is disabled, the character must be transmitted by writing to the
8530 SCC.
```

void SccChannel::dump_regs ()

Dump the content of the SCC registers on stdout.

PCIeMini_status SccChannel::enableIntSCC (uint8_t *mask*)

int SccChannel::enableRTS ()

Asserts RTS.

int SccChannel::enableRxDma ()

Enable the Receive FIFO.

```
    When the FIFO is enabled, any character available will be copied through DMA
to the associated receive FIFO.
    The character must then be retrieved by accessing the FIFO instead of the 8530
SCC.
```

int SccChannel::enableTxDma ()

Enable the Transmit FIFO.

```
When the FIFO is enabled, whenever the transmitter is ready, any character
available in the transmit FIFO will be copied through DMA to the SCC.
```

```
The character must then be transmitted by writing to the FIFO instead of the 8530
SCC.
```

char * SccChannel::gets_s (char * *buffer*, size_t *n*)

Receive a line terminated by the EOL character.

The EOL character is defined in the SccConfig structure. The string is always 0 terminated.

Parameters

buffer	Buffer receiving the characters
n	Size of the buffer

Return values

Address	of the buffer if the operation succeeded or NULL

uint32_t SccChannel::getSccControlRegister ()

int SccChannel::inch ()

Receive a data byte.

receive a character either directly from the SCC if the DMA is disabled, or from the FIFO if the DMA is enabled. Wait until a character is available.

Returns

This function returns the character.

int SccChannel::inch (int *pollDelay*)

Receive a data byte.

receive a character either directly from the SCC if the DMA is disabled, or from the FIFO if the DMA is enabled. Wait until a character is available.

Parameters

pollDelay	set the pollDelay to be used

Returns

This function returns the character.

int SccChannel::inchnw ()

Receive a data byte, if possible.

receive a character either directly from the SCC if the DMA is disabled, or from the FIFO if the DMA is enabled. Does not wait if there is no character available.

Returns

This function returns -1 if no character is available, and the character if the character was received successfully.

void SccChannel::outch (uint8_t *val*)

Transmit a data byte.

Transmit a character directly to the SCC if the DMA is disabled, or using the FIFO if the DMA is enabled. Returns when the character has been written to its destination, even if the actual transmission has not actually started.

Parameters

val	The character to transmit.

int SccChannel::outchnw (uint8_t *val*)

Transmit a data byte, if possible.

Transmit a character directly to the SCC if the DMA is disabled, or using the FIFO if the DMA is enabled. Does not wait if the transmitter logic is not ready.

Parameters

val	The character to transmit.

Returns

This function returns -1 if the character has not been written, and 0 if it has been written successfully

int SccChannel::puts (const char * *str*)

Transmit a 0 terminated string.

Parameters

str	Pointer to the string. *retval Returns the number of characters transmitted

PCIeMini_status SccChannel::reset ()

Reset a channel pair.

Reset a channel. Please note that because of hardware limitations, the two channels of the on board ESCC are reset at the same time

void SccChannel::resetChannel ()

Reset FIFOs and disable DMAs.

uint32_t SccChannel::resetSccControlRegister (uint32_t *mask*)

int32_t SccChannel::sccDataRead (uint8_t * *val*)

Read the UART receive register.

This function should not be used if the DMA is enabled, since it might create a race condition.

Parameters

val	The pointer to where to put the value.

Return values

Error	status (always ERRCODE_NO_ERROR)

int32_t SccChannel::sccDataWrite (uint8_t *val*)

Write to the UART transmit register.

This function should not be used if the DMA is enabled.

Parameters

val	The value to write.

Return values

Error	status (always ERRCODE_NO_ERROR)

int32_t SccChannel::sccRegisterRead (uint8_t *regNbr*, volatile uint8_t * *val*)

Read a UART read register.

This function read the register not the cached copy of the write register. It does not have provisions to read alternate registers. The difference between this function and registerRead8 is that it is intended to access the SCC registers and handle the data/address multiplexed access.

Parameters

regNbr	The read register number to read.
val	The pointer to where to put the value.

Return values

Error	status (always ERRCODE_NO_ERROR)

int32_t SccChannel::sccRegisterWrite (uint8_t *regNbr*, uint8_t *val*)

Write to a UART write register.

This function write to the UART register and store a cached copy. It does not have provisions to read alternate registers. The difference between this function and registerWrite8 is that it is intended to access the SCC registers and handle the data/address multiplexed access.

Parameters

regNbr	The read register number to read.
val	The value to write.

Return values

Error	status (always ERRCODE_NO_ERROR)

void SccChannel::set_serialBuffers (uint32_t *val*)

specify the serial port buffer configuration, RS232, RS422, or RS485.

Parameters

val	The value to put in the buffer configuration register, as defined in the **LTC2872** class.

uint32_t SccChannel::setSccControlRegister (uint32_t *mask*)

Member Data Documentation

volatile void* SccChannel::baseAddress

Pointer to the channel hardware.

LTC2872* SccChannel::buffer

Pointer to the buffer control.

uint8_t SccChannel::cachedRegs[32]

const int SccChannel::RTxcFrequency = 14745600 `[static]`

TxC input, should be 14.745600 MHz. Used to calculate the value for the divider register.

SccFifo* SccChannel::rxFifo

Receiver FIFO.

const int SccChannel::SCC_clocksource = 16000000 `[static]`

PCLK frequency used for baud rate calculations.

UartChannelConfig SccChannel::scc_config

Copy of the SCC configuration parameters.

const uint32_t SccChannel::sccControlFilterOnMask = 0x002000 `[static]`

Filter the escaped bisync characters.

const uint32_t SccChannel::sccControlIrqEnableMask = 0x000400 `[static]`

General interrupt enable.

const uint32_t SccChannel::sccControlMainResetMask = 0x008000 `[static]`

When set, do a hard reset on the SCC logic. Must be manually set to 0 to allow operations.

const uint32_t SccChannel::sccControlRxDmaEnableMask = 0x000800 `[static]`

Use the FIFO to receive characters.

const uint32_t SccChannel::sccControlRxFifoIrqMask = 0x040000 `[static]`

status of receive FIFO interrupt request

const uint32_t SccChannel::sccControlSccIrqMask = 0x010000 `[static]`

status of the interrupt requests from the ESCC

const uint32_t SccChannel::sccControlStripcharRegMask = 0x00ff `[static]`

Bisync character to strip.

const uint32_t SccChannel::sccControlTxDmaEnableMask = 0x001000 `[static]`

Use the FIFO to transmit characters.

const uint32_t SccChannel::sccControlTxFifoIrqMask = 0x020000 [static]

status of transmit FIFO interrupt request

SccFifo* SccChannel::txFifo

Transmitter FIFO.

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

- PCIe-Mini-SCC2/include/**SccChannel.h**
- PCIe-Mini-SCC2/PCIeMini_ESCC_lib/**SccChannel.cpp**
- PCIe-Mini-SCC2/PCIeMini_ESCC_lib/**SccDma.cpp**
- PCIe-Mini-SCC2/PCIeMini_ESCC_lib/**UartChannel.cpp**

SccFifo Class Reference

SCC FIFO access class.
`#include <SccFifo.h>`

Public Member Functions

- **SccFifo** (uint32_t *ctrlAddress, **RxFifoData** *fifoOut=NULL)
  Constructor.
- int **usage** ()
  Returns the number of bytes in the FIFO.
- void **reset** ()
  reset the FIFO.
- bool **isFifoEmpty** ()
  Check if the fifo is empty.
- bool **isFifoAlmostEmpty** ()
  Check if the fifo is almost empty.
- bool **isFifoAlmostFull** ()
  Check if the fifo is almost full.
- bool **isFifoFull** ()
  Check if the fifo is full.
- int **fifoSpace** ()
  Returns the space available in the FIFO in bytes.
- uint32_t **getCtrlReg** ()
- uint32_t **FifoGetWord** ()
  Check the 8 to 32 bit FIFO adapter.
- int **getByte** ()

Static Public Attributes

- static const int **fifoSize** = 2048
  Total FIFO size in bytes.

Detailed Description

SCC FIFO access class.

This class allows resetting and checking the state of the byte oriented FIFOs used by the transmitter and by the receiver.

Constructor & Destructor Documentation

SccFifo::SccFifo (uint32_t * *ctrlAddress*, RxFifoData * *fifo8to32* = `NULL`)

Constructor.

This is called during the board open process. It should not be called by the user.

Parameters

ctrlAddress	Pointer to the FIFO structure in user space.
isReadFifo	Read FIFO have a 4 byte buffer that needs to be taken in count when calculating FIFO usage

---

Member Function Documentation

uint32_t SccFifo::FifoGetWord ()

Check the 8 to 32 bit FIFO adapter.

If there are no byte left in the cache, try to read from the adapter.

Returns

The number of valid bytes in the cache.

int SccFifo::fifoSpace ()

Returns the space available in the FIFO in bytes.

Check the FIFO space left.

Returns

The number of characters that can still be added to the FIFO.

int SccFifo::getByte ()

uint32_t SccFifo::getCtrlReg ()

bool SccFifo::isFifoAlmostEmpty ()

Check if the fifo is almost empty.

Return values

True	if the FIFO is almost empty Returns true if the fifo is almost empty

bool SccFifo::isFifoAlmostFull ()

Check if the fifo is almost full.

Return values

True	if the FIFO is almost full Returns true if the fifo is almost full

bool SccFifo::isFifoEmpty ()

Check if the fifo is empty.

Return values

True	if the FIFO is empty Returns true if the fifo is empty

bool SccFifo::isFifoFull ()

Check if the fifo is full.

Return values

True	if the FIFO is full Returns true if the fifo is full

void SccFifo::reset ()

reset the FIFO.

Resets the FIFO.

The pointers are reset and the FIFO is emptied.

int SccFifo::usage ()

Returns the number of bytes in the FIFO.

Check the FIFO usage.

Returns

The number of characters in the FIFO.

Member Data Documentation

const int SccFifo::fifoSize = 2048 `[static]`

Total FIFO size in bytes.

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

- PCIe-Mini-SCC2/include/**SccFifo.h**
- PCIe-Mini-SCC2/PCIeMini_ESCC_lib/**SccFifo.cpp**

UartChannelConfig Class Reference

UART channel configuration structure.
`#include <SccChannel.h>`

Public Member Functions

- **UartChannelConfig** ()
  Configuration structure constructor.

Public Attributes

- int **baudRate**
  300 to 38400 bps
- uint8_t **dataBits**
  7 or 8 data bits
- uint8_t **stopBits**
  1 or 2 stop bits
- bool **parityEnable**
  0 for parity disabled, 1 for parity enabled
- uint8_t **parity**
  0 for even parity, 1 for odd parity
- uint8_t **eolChar**
  byte value used as an end of line for the Scc_gets_s function. Default is 0x0a. If eolChar contains 0xff, the feature is disabled.
- int **bufferConfig**
  buffer selection
- bool **useRxFifo**
  use external FIFO to store received data
- bool **useTxFifo**
  use external FIFO to store data to transmit
- bool **localLoopbackMode**

Detailed Description

UART channel configuration structure.

This structure contains the configuration values for the UART channel.

Constructor & Destructor Documentation

UartChannelConfig::UartChannelConfig ()

Configuration structure constructor.

Initialize the structure with default values: 9600bps, 8-bit, '
' as end of line, parity disabled, 1 stop bit, FIFOs enabled, RS422.

Member Data Documentation

int UartChannelConfig::baudRate

300 to 38400 bps

int UartChannelConfig::bufferConfig

buffer selection

uint8_t UartChannelConfig::dataBits

7 or 8 data bits

uint8_t UartChannelConfig::eolChar

byte value used as an end of line for the Scc_gets_s function. Default is 0x0a. If eolChar contains 0xff, the feature is disabled.

bool UartChannelConfig::localLoopbackMode

uint8_t UartChannelConfig::parity

0 for even parity, 1 for odd parity

bool UartChannelConfig::parityEnable

0 for parity disabled, 1 for parity enabled

uint8_t UartChannelConfig::stopBits

1 or 2 stop bits

bool UartChannelConfig::useRxFifo

use external FIFO to store received data

bool UartChannelConfig::useTxFifo

use external FIFO to store data to transmit

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

- PCIe-Mini-SCC2/include/**SccChannel.h**
- PCIe-Mini-SCC2/PCIeMini_ESCC_lib/**SccChannel.cpp**

Index