

# **PCIeMini-ESCC**

## **2- Channels**

### **Serial Communication Controller**

## **PCIexpress Mini**

**927-10-003-4000**

**Software Manual**

Version 1.0  
08/06/2020

## **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.....	<b>Error! Bookmark not defined.</b>
Class Index.....	4
Class Documentation .....	5
AlphiBoard.....	5
BoardVersion.....	10
LinearAddress .....	11
LTC2872 .....	12
MINIPCIE_DEV_CTX.....	16
MINIPCIE_INT_RESULT .....	17
PCIEMini_ESCC.....	19
RxFifoData.....	24
SccChannel .....	25
SccFifo .....	35
UartChannelConfig.....	38

## Class Index

### Class List

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

<b>AlphiBoard (Base class implementing a PCI board connection to the board and the Jungo driver )</b> .....	9
<b>BoardVersion (Board Hardware identification and version )</b> .....	15
<b>LinearAddress (Memory Segment Descriptor )</b> .....	15
<b>LTC2872 (Implementation of the LTC2872 buffer )</b> .....	16
<b>MINIPCIE_DEV_CTX</b> .....	21
<b>MINIPCIE_INT_RESULT (Interrupt result information structure )</b> .....	21
<b>PCIeMini_ESCC (PCIeMini_ESCC controller board object )</b> .....	22
<b>SccChannel (Low-level SCC access class )</b> .....	30
<b>SccFifo (SCC FIFO access class )</b> .....	39
<b>UartChannelConfig (UART channel configuration structure )</b> Error! Bookmark not defined.	

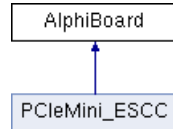
# Class Documentation

## 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**

<i>Status</i>	code
---------------	------

**DWORD AlphaBoard::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**

<i>Status</i>	code
---------------	------

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

Return a pointer to an object in BAR 0.

**Parameters**

<i>offset</i>	Offset in BAR0
---------------	----------------

**Return values**

<i>Pointer</i>	to the object
----------------	---------------

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

Return a pointer to an object in BAR 2.

**Parameters**

<i>offset</i>	Offset in BAR2
---------------	----------------

**Return values**

<i>Pointer</i>	to the object
----------------	---------------

**DWORD AlphaBoard::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**

<i>mask</i>	board dependent interrupt mask.
<i>uicr</i>	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 AlphaBoard::IsValidDevice (const CHAR \* sFunc)**

Validate a WDC device handle.

**Parameters**

<i>sFunc</i>	C-string with name of the function e.g. "IntEnable"
--------------	---

**Return values**

<i>true</i>	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**

<i>bar</i>	The number of the bar to access.
<i>addr</i>	The <b>LinearAddress</b> 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**

<i>brdNbr</i>	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**

<i>vb</i>	Verbosity level.
-----------	------------------

**DWORD AlphiBoard::UnhookMailboxInterrupt ()**

Disable the board interrupt.

**Parameters**

<i>mask</i>	board dependent interrupt mask.
<i>uicr</i>	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**

<i>WD_STATUS_SUCCESS</i>	
--------------------------	--

---

**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

<i>addr</i>	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

<i>devAddr</i>	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

<i>settings</i>	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 $\Omega$  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

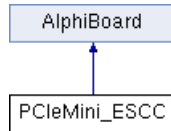


## PCIeMini\_ESCC Class Reference

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** (PCleMini\_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

**PCleMini\_ESCC** controller board object.

---

#### Constructor & Destructor Documentation

##### **PCleMini\_ESCC::PCleMini\_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

##### **PCleMini\_status PCleMini\_ESCC::close ()**

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

#### Returns

ERRCODE\_NO\_ERROR if successful.

#### PCleMini\_status PcleMini\_ESCC::disableInterrupts ()

Disable the interrupts from the board.

#### Returns

ERRCODE\_NO\_ERROR if successful.

#### void PcleMini\_ESCC::disableRxDma (int chan)[inline]

#### void PcleMini\_ESCC::disableTxDma (int chan)[inline]

#### PCleMini\_status PcleMini\_ESCC::enableInterrupts ()

Enable the interrupts from the board.

#### Returns

ERRCODE\_NO\_ERROR if successful.

#### void PcleMini\_ESCC::enableRxDma (int chan)[inline]

#### void PcleMini\_ESCC::enableTxDma (int chan)[inline]

#### char \* PcleMini\_ESCC::getErrorMsg (PCleMini\_status errorNbr)[static]

Return a text description corresponding to an error code.

#### Returns

A pointer to a null terminated character string.

#### uint32\_t PcleMini\_ESCC::getFpgaID ()

Get the FPGA ID of.

#### Returns

The FPGA ID.

#### time\_t PcleMini\_ESCC::getFpgaTimeStamp ()

Return the timestamp corresponding to when the FPGA was compiled.

#### Returns

a timestamp.

#### uint32\_t PcleMini\_ESCC::getLedPio ()

#### SccChannel \* PcleMini\_ESCC::getScc (int channelNbr)

get pointer to an instance serial channel object.

#### Parameters

<i>channelNbr</i>	Channel number 0, or 1.
-------------------	-------------------------

#### Returns

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

**PCleMini\_status PCleMini\_ESCC::hookInterruptServiceRoutine  
(MINIPCIE\_INT\_HANDLER *uicr*)**

Set an interrupt handling routine.

#### Parameters

<i>uicr</i>	user callback routine typedef void (__stdcall *UsersIntCompletionRoutine)(void *, uint32_t);
-------------	---

#### Returns

ERRCODE\_NO\_ERROR if successful.

**int PCleMini\_ESCC::initChannel (int *chan*)**

**PCleMini\_status PCleMini\_ESCC::open (int *brdNbr*)**

Open: connect to an actual board.

#### Parameters

<i>brdNbr</i>	The board number is actually system dependent but if you have only one board, it should be 0.
---------------	--

#### Returns

ERRCODE\_NO\_ERROR if successful.

**PCleMini\_status PCleMini\_ESCC::reset ()**

Reset the board ARINC 429 controllers.

#### Returns

ERRCODE\_NO\_ERROR if successful.

**void PCleMini\_ESCC::resetChannel (int *chan*) [inline]**

**void PCleMini\_ESCC::resetChip (int *chan*) [inline]**

Reset the chip and FIFOs

**void PCleMini\_ESCC::sccRegisterRead (int *chan*, uint8\_t *regNbr*, volatile uint8\_t \*  
*val*) [inline]**

**void PCleMini\_ESCC::sccRegisterWrite (int *chan*, uint8\_t *regNbr*, uint8\_t  
*val*) [inline]**

**void PCleMini\_ESCC::setLedPio (uint32\_t *val*)**

**PCleMini\_status PCleMini\_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::pcielrqEnable**

PCIe interface interrupt enable.

### **volatile uint32\_t\* PCIeMini\_ESCC::pcielrqStatus**

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.*
- **PCleMini\_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.*
- **PCleMini\_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

<i>addr</i>	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

<i>rtable</i>	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

<i>config</i>	New channel configuration.
---------------	----------------------------

#### Returns

If successful, Scc\_Config returns 0. If the function fails, it returns a nonzero value.

### **PCleMini\_status SccChannel::disableIntSCC (uint8\_t *mask*)**

This subroutine disables ESCC interrupts.

**Parameters**

<i>mask</i>	Bits corresponding to interrupts to disable.
-------------	--

**Return values**

<i>Error</i>	status (always <code>ERRCODE_NO_ERROR</code> )
--------------	--

**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.

**PCleMini\_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**

<i>buffer</i>	Buffer receiving the characters
<i>n</i>	Size of the buffer

**Return values**

<i>Address</i>	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**

<i>pollDelay</i>	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**

<i>val</i>	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**

<i>val</i>	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**

<i>str</i>	Pointer to the string. *retval Returns the number of characters transmitted
------------	---

**PCleMini\_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**

<i>val</i>	The pointer to where to put the value.
------------	--

**Return values**

<i>Error</i>	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**

<i>val</i>	The value to write.
------------	---------------------

**Return values**

<i>Error</i>	status (always <code>ERRCODE_NO_ERROR</code> )
--------------	--

**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**

<i>regNbr</i>	The read register number to read.
<i>val</i>	The pointer to where to put the value.

**Return values**

<i>Error</i>	status (always <code>ERRCODE_NO_ERROR</code> )
--------------	--

**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**

<i>regNbr</i>	The read register number to read.
<i>val</i>	The value to write.

**Return values**

<i>Error</i>	status (always <code>ERRCODE_NO_ERROR</code> )
--------------	--

**void SccChannel::set\_serialBuffers (uint32\_t *val*)**

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

**Parameters**

<i>val</i>	The value to put in the buffer configuration register, as defined in the <b>LTC2872</b> 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::sccControlScclrqMask = 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**

<i>ctrlAddress</i>	Pointer to the FIFO structure in user space.
<i>isReadFifo</i>	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::getBytes ()****uint32\_t SccFifo::getCtrlReg ()****bool SccFifo::isFifoAlmostEmpty ()**

Check if the fifo is almost empty.

**Return values**

<i>True</i>	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**

<i>True</i>	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**

<i>True</i>	if the FIFO is empty Returns true if the fifo is empty
-------------	--

**bool SccFifo::isFifoFull ()**

Check if the fifo is full.

**Return values**

<i>True</i>	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/PCleMini\_ESCC\_lib/**SccChannel.cpp**



# Index

- ~AlphiBoard
  - AlphiBoard, 6
- addr
  - LTC2872, 14
- Address
  - LinearAddress, 11
- AlphiBoard, 5
  - ~AlphiBoard, 6
  - AlphiBoard, 6
  - bar0, 9
  - bar2, 9
  - Close, 6
  - DisableInterrupts, 6
  - EnableInterrupts, 7
  - getBar0Address, 7
  - getBar2Address, 7
  - HookMailboxInterrupt, 7
  - IsValidDevice, 7
  - Map, 8
  - MsSleep, 8
  - Open, 8
  - setVerbose, 8
  - UnhookMailboxInterrupt, 8
  - Unmap, 8
  - verbose, 9
- bar0
  - AlphiBoard, 9
- bar2
  - AlphiBoard, 9
- baseAddress
  - SccChannel, 32
- baudRate
  - UartChannelConfig, 39
- BoardVersion, 10
  - BoardVersion, 10
  - getTimeStamp, 10
  - getVersion, 10
- brgDivLow
  - SccChannel, 28
- buffer
  - SccChannel, 33
- bufferConfig
  - UartChannelConfig, 39
- cachedRegs
  - SccChannel, 33
- channelLoad
  - SccChannel, 28
- clockSelect
  - LTC2872, 14
- close
  - PCIEMini\_ESCC, 20
- Close
  - AlphiBoard, 6
- config
  - SccChannel, 28
- dataBits
  - UartChannelConfig, 39
- dcdInputDisable
  - LTC2872, 14
- disableInterrupts
  - PCIEMini\_ESCC, 21
- DisableInterrupts
  - AlphiBoard, 6
- disableIntSCC
  - SccChannel, 28
- disableRTS
  - SccChannel, 29
- disableRxDma
  - PCIEMini\_ESCC, 21
  - SccChannel, 29
- disableTxDma
  - PCIEMini\_ESCC, 21
  - SccChannel, 29
- driverModeMask
  - LTC2872, 14
- dump\_regs
  - SccChannel, 29
- dwCounter
  - MINIPCIE\_INT\_RESULT, 17
- dwEnabledIntType
  - MINIPCIE\_INT\_RESULT, 17
- dwLastMessage
  - MINIPCIE\_INT\_RESULT, 17
- dwLost
  - MINIPCIE\_INT\_RESULT, 17
- enableInterrupts
  - PCIEMini\_ESCC, 21
- EnableInterrupts
  - AlphiBoard, 7
- enableIntSCC
  - SccChannel, 29
- enableRTS
  - SccChannel, 29
- enableRxDma
  - PCIEMini\_ESCC, 21
  - SccChannel, 29
- enableTxDma
  - PCIEMini\_ESCC, 21
  - SccChannel, 29
- eolChar
  - UartChannelConfig, 39
- fastMode
  - LTC2872, 14
- FifoGetWord
  - SccFifo, 36
- fifoSize
  - SccFifo, 37
- fifoSpace

<p>SccFifo, 36</p> <p>funcDiagEventHandler MINIPCI_DEV_CTX, 16</p> <p>funcDiagIntHandler MINIPCI_DEV_CTX, 16</p> <p>getBar0Address AlphiBoard, 7</p> <p>getBar2Address AlphiBoard, 7</p> <p>getBuffer LTC2872, 13</p> <p>getByte SccFifo, 36</p> <p>getCtrlReg SccFifo, 36</p> <p>getCurrValid RxFifoData, 24</p> <p>getData RxFifoData, 24</p> <p>getErrorMsg PCIEMini_ESCC, 21</p> <p>getFpgaID PCIEMini_ESCC, 21</p> <p>getFpgaTimeStamp PCIEMini_ESCC, 21</p> <p>getLedPio PCIEMini_ESCC, 21</p> <p>getRcvdValid RxFifoData, 24</p> <p>gets_s SccChannel, 30</p> <p>getScc PCIEMini_ESCC, 21</p> <p>getSccControlRegister SccChannel, 30</p> <p>getTimeStamp BoardVersion, 10</p> <p>getVersion BoardVersion, 10</p> <p>halfDuplexEnable LTC2872, 14</p> <p>hookInterruptServiceRoutine PCIEMini_ESCC, 22</p> <p>HookMailboxInterrupt AlphiBoard, 7</p> <p>inch SccChannel, 30</p> <p>inchnw SccChannel, 30</p> <p>initChannel PCIEMini_ESCC, 22</p> <p>isFifoAlmostEmpty SccFifo, 36</p> <p>isFifoAlmostFull SccFifo, 36</p> <p>isFifoEmpty SccFifo, 36</p> <p>isFifoFull SccFifo, 37</p> <p>IsValidDevice</p>	<p>AlphiBoard, 7</p> <p>ledPio PCIEMini_ESCC, 23</p> <p>Length LinearAddress, 11</p> <p>LinearAddress, 11 Address, 11 Length, 11</p> <p>localLoopbackMode UartChannelConfig, 39</p> <p>loopbackEnable LTC2872, 14</p> <p>LTC2872, 12 addr, 14 clockSelect, 14 dcdInputDisable, 14 driverModeMask, 14 fastMode, 14 getBuffer, 13 halfDuplexEnable, 14 loopbackEnable, 14 LTC2872, 13 recvDisable, 15 RS_232_buffers, 15 RS_422_buffers, 15 RS_422_localLoopback, 15 RS_422_pullups_buffers, 15 rs485Mode, 15 setBuffer, 13 terminationEnable, 15</p> <p>Map AlphiBoard, 8</p> <p>MINIPCI_DEV_CTX, 16 funcDiagEventHandler, 16 funcDiagIntHandler, 16</p> <p>MINIPCI_INT_RESULT, 17 dwCounter, 17 dwEnabledIntType, 17 dwLastMessage, 17 dwLost, 17 waitResult, 17</p> <p>MsSleep AlphiBoard, 8</p> <p>open PCIEMini_ESCC, 22</p> <p>Open AlphiBoard, 8</p> <p>outch SccChannel, 30</p> <p>outchnw SccChannel, 31</p> <p>parity UartChannelConfig, 39</p> <p>parityEnable UartChannelConfig, 39</p> <p>pcieIrqEnable PCIEMini_ESCC, 23</p> <p>pcieIrqStatus PCIEMini_ESCC, 23</p> <p>PCIEMini_ESCC, 19</p>
---	--

close, 20	getCurrValid, 24
disableInterrupts, 21	getData, 24
disableRxDma, 21	getRcvdValid, 24
disableTxDma, 21	SCC_clocksource
enableInterrupts, 21	SccChannel, 33
enableRxDma, 21	scc_config
enableTxDma, 21	PCIEMini_ESCC, 23
getErrorMsg, 21	SccChannel, 33
getFpgaID, 21	SccChannel, 25
getFpgaTimeStamp, 21	baseAddress, 32
getLedPio, 21	brgDivLow, 28
getScc, 21	buffer, 33
hookInterruptServiceRoutine, 22	cachedRegs, 33
initChannel, 22	channelLoad, 28
ledPio, 23	config, 28
open, 22	disableIntSCC, 28
pcieIrqEnable, 23	disableRTS, 29
pcieIrqStatus, 23	disableRxDma, 29
PCIEMini_ESCC, 20	disableTxDma, 29
reset, 22	dump_regs, 29
resetChannel, 22	enableIntSCC, 29
resetChip, 22	enableRTS, 29
scc_config, 23	enableRxDma, 29
sccDevice_0, 23	enableTxDma, 29
sccDevice_1, 23	gets_s, 30
sccRegisterRead, 22	getSccControlRegister, 30
sccRegisterWrite, 22	inch, 30
setLedPio, 22	inchnw, 30
sysid, 23	outch, 30
unhookInterruptServiceRoutine, 22	outchnw, 31
puts	puts, 31
SccChannel, 31	reset, 31
recvDisable	resetChannel, 31
LTC2872, 15	resetSccControlRegister, 31
reset	RTxcFrequency, 33
PCIEMini_ESCC, 22	rxFifo, 33
SccChannel, 31	SCC_clocksource, 33
SccFifo, 37	scc_config, 33
resetChannel	SccChannel, 28
PCIEMini_ESCC, 22	sccControlFilterOnMask, 33
SccChannel, 31	sccControlIrqEnableMask, 33
resetChip	sccControlMainResetMask, 33
PCIEMini_ESCC, 22	sccControlRxDmaEnableMask, 33
resetSccControlRegister	sccControlRxFifoIrqMask, 33
SccChannel, 31	sccControlSccIrqMask, 33
RS_232_buffers	sccControlStripcharRegMask, 33
LTC2872, 15	sccControlTxDmaEnableMask, 33
RS_422_buffers	sccControlTxFifoIrqMask, 34
LTC2872, 15	sccDataRead, 31
RS_422_localLoopback	sccDataWrite, 31
LTC2872, 15	sccRegisterRead, 32
RS_422_pullups_buffers	sccRegisterWrite, 32
LTC2872, 15	set_serialBuffers, 32
rs485Mode	setSccControlRegister, 32
LTC2872, 15	txFifo, 34
RTxcFrequency	sccControlFilterOnMask
SccChannel, 33	SccChannel, 33
rxFifo	sccControlIrqEnableMask
SccChannel, 33	SccChannel, 33
RxFifoData, 24	sccControlMainResetMask

SccChannel, 33	LTC2872, 13
sccControlRxDmaEnableMask	setLedPio
SccChannel, 33	PCIEMini_ESCC, 22
sccControlRxFifoIrqMask	setSccControlRegister
SccChannel, 33	SccChannel, 32
sccControlSccIrqMask	setVerbose
SccChannel, 33	AlphiBoard, 8
sccControlStripcharRegMask	stopBits
SccChannel, 33	UartChannelConfig, 39
sccControlTxDmaEnableMask	sysid
SccChannel, 33	PCIEMini_ESCC, 23
sccControlTxFifoIrqMask	terminationEnable
SccChannel, 34	LTC2872, 15
sccDataRead	txFifo
SccChannel, 31	SccChannel, 34
sccDataWrite	UartChannelConfig, 38
SccChannel, 31	baudRate, 39
sccDevice_0	bufferConfig, 39
PCIEMini_ESCC, 23	dataBits, 39
sccDevice_1	eolChar, 39
PCIEMini_ESCC, 23	localLoopbackMode, 39
SccFifo, 35	parity, 39
FifoGetWord, 36	parityEnable, 39
fifoSize, 37	stopBits, 39
fifoSpace, 36	UartChannelConfig, 39
getBytes, 36	useRxFifo, 39
getCtrlReg, 36	useTxFifo, 39
isFifoAlmostEmpty, 36	unhookInterruptServiceRoutine
isFifoAlmostFull, 36	PCIEMini_ESCC, 22
isFifoEmpty, 36	UnhookMailboxInterrupt
isFifoFull, 37	AlphiBoard, 8
reset, 37	Unmap
SccFifo, 36	AlphiBoard, 8
usage, 37	usage
sccRegisterRead	SccFifo, 37
PCIEMini_ESCC, 22	useRxFifo
SccChannel, 32	UartChannelConfig, 39
sccRegisterWrite	useTxFifo
PCIEMini_ESCC, 22	UartChannelConfig, 39
SccChannel, 32	verbose
set_serialBuffers	AlphiBoard, 9
SccChannel, 32	waitResult
setBuffer	MINIPCIE_INT_RESULT, 17