LPC USB Serial I/O (LPCUSBSIO)

1.00

Generated by Doxygen 1.8.5

Thu Oct 24 2013 16:48:25

Contents

Chapter 1

LPC USB Serial I/O Library

1.1 Introduction

The LPC USB serial I/O (LPCUSBSIO) is a generic API provided to PC applications programmer to develop applications to communicate with serial I/O devices connected to LPC micro-controller. LPCUSBIO library converts all API calls into USB messages which are transferred to LPC micro-controller, which in turn communicates with serial devices using I2C, SPI and GPIO interfaces. To make the USB device install free on host systems LPCUSBSIO uses USB-HID class as transport mechanism. HID class is natively supported by most commercially available host operating systems.

Note

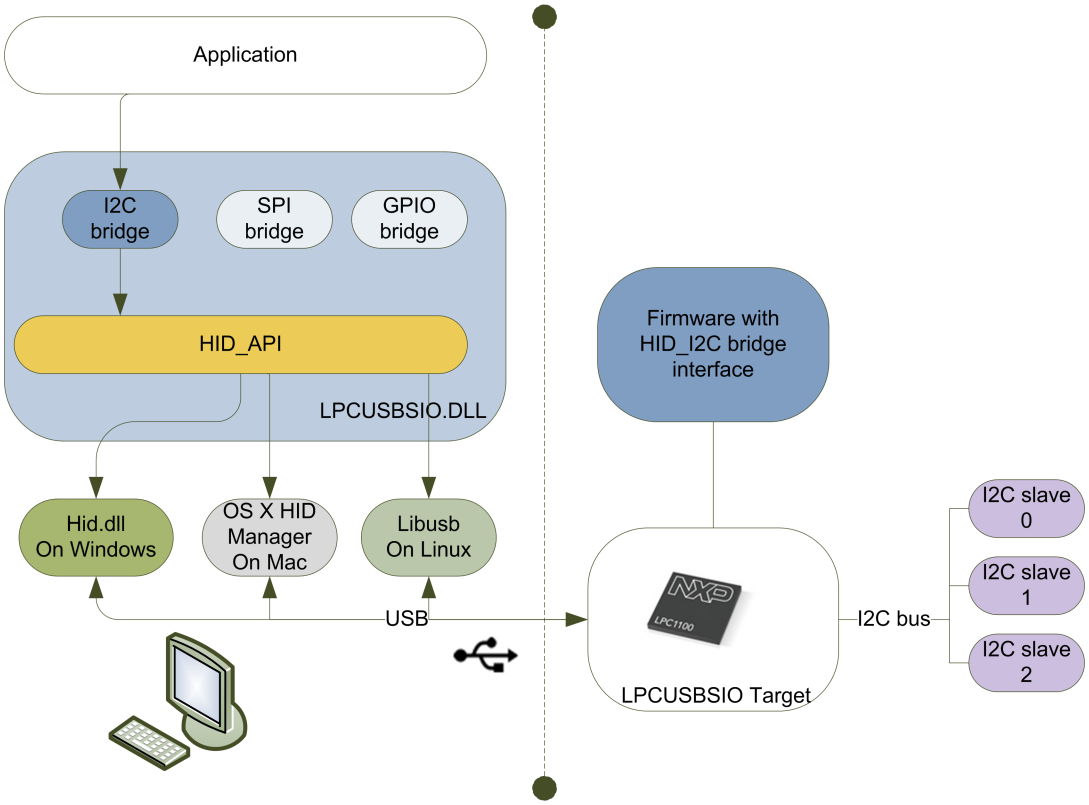
The current version of LPCUSBSIO allows communicating with I2C slave devices only.

See also:

- LPCUSBSIO Architecture
- Developing I2C application with LPCUSBSIO
- LPC USB serial I/O (LPCUSBSIO) I2C API interface

1.2 LPCUSBSIO Architecture

The following figure shows the architectural block diagram of the LPCUSBSIO framework. The framework uses `HID_API` , a multi-platform library which allows an application to interface with USB HID-Class devices on Windows, Linux, and Mac OS X.



The above figure shows how the components of the system are typically organized. The PC/Host may be desktop/laptop machine or an embedded system. The LPC micro-controller and the I2C device would usually be on the same PCB. The LPC micro-controller will operate in I2C master mode. All the I2C device connected to the bus should have unique address on bus. I2C devices that support configurable addresses will have pins which can be hard-wired to give a device an appropriate address; this information may be found in the datasheet of the I2C device chip.

Chapter 2

Developing I2C application with LPCUSBSIO

This section introduces you to application development using LPCUSBSIO library.

2.1 Prerequisites

- A host systems for which LPCUSBSIO library is available.
- A LPC target board running LPCUSBIO firmware.
- Download the LPCUSBSIO library package from `http://www.lpcware.com`
- C compiler/IDE for the host system.

2.2 Setting-up project environment

- Include lpcusbsio.h header available as part of distribution in application C files.
- On Windows systems,
  - Add bin/win32/lpcusbsio.lib as an input to the linker along with application object files.
  - LPCUSBSIO uses setupapi.dll hence add the corresponding library to the linker list.
    - If using MinGw include `-lsetupapi` as part of linker options.
- On Mac OSX,
  - Add bin/osx/lpcusbsio.a as an input to the linker along with application object files.
  - LPCUSBSIO uses "IOKit" and "CoreFoundation" frameworks hence add them to linker list.
    - Include `-framework IOKit -framework CoreFoundation` in linker options.
- On Linux,

  - Add bin/linux/lpcusbsio.a as an input to the linker along with application object files.
  - LPCUSBSIO uses "libusb-1.0" hence include it in linker list.
    * If unsure of the version of libusb-1.0 on your system use following linker flags
      · `pkg-config libusb-1.0 libudev --libs`

2.3 Initializing and obtaining device handle

- LPCUSBSIO library initialization and enumeration of all LPCUSBSIO target devices are combined into one API call of I2C_GetNumPorts().
  - All applications using the library should call this routine first.
  - If the target LPC controller has 2 HID_I2C interfaces then this routine reports them as 2 independent ports.
  - Multiple instances of the application linked with the library can be ran on the host. If an application claims a port by calling I2C_Open() then the corresponding port will not be reported in subsequent call of I2C_GetNumPorts().
  - Multiple LPC controller targets can be attached to the host. The current version of the library doesn't provide a mechanism to uniquely identify individual targets except with an index number assigned during enumeration.
- After finding the number of target LPC controllers connected to the host. Get a handle to the controller we are interested in by calling I2C_Open() with an index.
  - The handle returned by I2C_Open() should be used by all consequent calls targeted to the port.
  - If the port is claimed by a parallel thread or application a NULL handle is returned.
- After obtaining the handle, the I2C port should be initialized with bus speed using I2C_Init().
  - If bus speed is not set then the default speed or speed set by prior application is used by the controller to communicate with slave devices.
  - Note, this call also obtains the version string for the firmware running on the target controller. Calling I2C_GetVersion() prior to I2C_Init() call would return version string without firmware version.
- Now the port is ready for data transfers. Following code snippet shows the initialization steps.

```
int res;
I2C_PORTCONFIG_T cfgParam;
I2C_FAST_XFER_T xfer;

res = I2C_GetNumPorts();

if (res > 0) {
 printf("Total I2C devices: %d \r\n ", res);

 /* open device at index 0 */
 g_hI2CPort = I2C_Open(0);

 if (g_hI2CPort != NULL) {
  /* Init the I2C port for standard speed communication */
  cfgParam.ClockRate = I2C_CLOCK_STANDARD_MODE;
  cfgParam.Options = 0;
  res = I2C_Init(g_hI2CPort, &cfgParam);

  if (res < 0) {
   printf("Unable to init I2C port. %ls \r\n", I2C_Error(g_hI2CPort));
   I2C_Close(0);
  }
  else {
   .....
  }
 }
}
else {
 printf("Error: No free ports. \r\n");
}
```

2.4 Reading and Writing to an I2C slave device

The library provides two types of APIs for transferring data to and from I2C slave devices connected to the target LPC controller.

- Uni-directional routines : This group contains independent I2C_DeviceWrite() and I2C_DeviceRead() routines.
- Bi-directional routines : This group has single API I2C_FastXfer() routine to do read, write and read-after-write transfers.

To describe the I2C transactions following symbols are used in driver documentation.

- **S** (1 bit) : Start bit
- **P** (1 bit) : Stop bit
- **Rd**/**Wr** (1 bit) : Read/Write bit. **Rd** equals 1, **Wr** equals 0.
- **A**,NA (1 bit) : Acknowledge and Not-Acknowledge bit.
- **Addr** (7 bits): I2C 7 bit address. Note that this can be expanded as usual to get a 10 bit I2C address.
- **Data** (8 bits): A plain data byte. Sometimes, I write DataLow, DataHigh for 16 bit data.
- [..]: Data sent by I2C device, as opposed to data sent by the host adapter.

2.4.1 Uni-directional transfer routines

All data transfer API calls are translated into a single USB command and response message to reduce latencies. That means after successful transmission of START condition the data transfers happens without host involvement. In use cases where application code requires some processing in between a single transaction, the user code has to construct multiple I2C_DeviceWrite() and I2C_DeviceRead() calls with appropriate *option* parameters.

Note

In version 1.00 the library only supports transferring 56 bytes maximum per I2C_DeviceWrite() call and 60 bytes maximum per I2C_DeviceRead() call. The restriction comes from the underlying HID report size. Use *options* parameter to accomplish longer transfer in single I2C transaction.

The I2C transaction done I2C_DeviceWrite() and I2C_DeviceRead() routines can be tweaked by sending appropriate *option* parameter. Following user friendly options macros are defined in lpcusbsio.h header.

- I2C_TRANSFER_OPTIONS_START_BIT : Tells API to generate start condition before transmitting.
- I2C_TRANSFER_OPTIONS_STOP_BIT : Tells API to generate stop condition at the end of transfer.
- I2C_TRANSFER_OPTIONS_BREAK_ON_NACK : Tells API to continue transmitting data in bulk without caring about Ack or nAck from device if this bit is not set. If this bit is set then stop transmitting the data in the buffer when the device nAcks.
- I2C_TRANSFER_OPTIONS_NACK_LAST_BYTE : lpcusbsio-I2C generates an ACKs for every byte read. Some I2C slaves require the I2C master to generate a nACK for the last data byte read. Setting this bit enables working with such I2C slaves.
- I2C_TRANSFER_OPTIONS_NO_ADDRESS : Setting this bit would mean that the address field should be ignored. The address is either a part of the data or this is a special I2C frame that doesn't require an address. For example when transferring a frame greater than the USB_HID packet this option can be used.

2.4.1.1 Using I2C_DeviceWrite

A sample code showing I2C_DeviceWrite() call.

```
res = I2C_DeviceWrite(g_hI2CPort,
       0x60,
       buff,
       5,
       I2C_TRANSFER_OPTIONS_START_BIT |
      I2C_TRANSFER_OPTIONS_STOP_BIT);
```

- When *options* is (I2C_TRANSFER_OPTIONS_START_BIT | I2C_TRANSFER_OPTIONS_STOP_BIT | I2C_TRANSFER_OPTIONS_BREAK_ON_NACK).

  S Addr Wr[A] txBuff0[A] txBuff1[A] ... txBuffN[A] P

  - If a NACK is received from slave the transfer is aborted in between and LPCUSBSIO_ERR_I2C_NAK is returned by routine.

- When *options* is (I2C_TRANSFER_OPTIONS_NO_ADDRESS | I2C_TRANSFER_OPTIONS_START_BIT | I2C_TRANSFER_OPTIONS_STOP_BIT | I2C_TRANSFER_OPTIONS_BREAK_ON_NACK).

  S txBuff0[A] ... txBuffN[A] P

- When *options* is (I2C_TRANSFER_OPTIONS_START_BIT | I2C_TRANSFER_OPTIONS_STOP_BIT).

  S Addr Wr[A] txBuff0[A or NA] ... txBuffN[A or NA] P

- When *options* is (I2C_TRANSFER_OPTIONS_START_BIT | I2C_TRANSFER_OPTIONS_BREAK_ON_NACK).

  S Addr Wr[A] txBuff0[A] txBuff1[A] ... txBuffN[A]

- When *options* is (I2C_TRANSFER_OPTIONS_BREAK_ON_NACK).

  txBuff0[A] ... txBuffN[A]

- When *options* is 0.

  txBuff0[A or NA] ... txBuffN[A or NA]

2.4.1.2 Using I2C_DeviceRead

A sample code showing I2C_DeviceRead() call.

```
res = I2C_DeviceRead(
 g_hI2CPort,
 i,
 buff,
 1,
 I2C_TRANSFER_OPTIONS_START_BIT |
      I2C_TRANSFER_OPTIONS_STOP_BIT |
 I2C_TRANSFER_OPTIONS_NACK_LAST_BYTE);
```

- When *options* is (I2C_TRANSFER_OPTIONS_START_BIT | I2C_TRANSFER_OPTIONS_STOP_BIT | I2C_TRANSFER_OPTIONS_NACK_LAST_BYTE).

  S Addr Rd [A] [rxBuff0] A [rxBuff1] A ...[rxBuffN] NA P

- When *options* is (I2C_TRANSFER_OPTIONS_NO_ADDRESS | I2C_TRANSFER_OPTIONS_START_BIT | I2C_TRANSFER_OPTIONS_STOP_BIT | I2C_TRANSFER_OPTIONS_NACK_LAST_BYTE).

  S [rxBuff0] A [rxBuff1] A ...[rxBuffN] NA P

- When *options* is (I2C_TRANSFER_OPTIONS_START_BIT | I2C_TRANSFER_OPTIONS_STOP_BIT).

  S Addr Rd [A] [rxBuff0] A [rxBuff1] A ...[rxBuffN] A P

- When *options* is (I2C_TRANSFER_OPTIONS_START_BIT).

  S Addr Rd [A] [rxBuff0] A [rxBuff1] A ...[rxBuffN] NA

- When *options* is 0.

  [rxBuff0] A [rxBuff1] A ...[rxBuffN] A

2.4.2 Bi-directional transfer routines

Most I2C read transaction are preceded with a write operation by using Uni-directional transfer routines we will be adding a round-trip delay between write and read operation. The LPC controller will hold the bus after write operation if a STOP signal is not transmitted on the bus. This will cause performance issues in multi-master scenarios. For those type of transaction I2C_FastXfer() is recommended.

Following types of transfers are possible :

- Write-only transfer : When *rxSz* member of *xfer* is set to 0. And *options* member of *xfer* is set to 0.

  S Addr Wr[A] txBuff0[A] txBuff1[A] ... txBuffN[A] P

  - If I2C_FAST_XFER_OPTION_IGNORE_NACK is set in *options* member

    S Addr Wr[A] txBuff0[A or NA] ... txBuffN[A or NA] P

    A sample code showing Write-only transfer using I2C_FastXfer() call.

```
xfer.options = 0;
xfer.txSz = 4;
xfer.rxSz = 0;
xfer.rxBuff = &buff[0];
xfer.slaveAddr = 0x60;
res = I2C_FastXfer(g_hI2CPort, &xfer);
```

- Read-only transfer : When *txSz* member of *xfer* is set to 0. And *options* member of *xfer* is set to 0.

  S Addr Rd[A][rxBuff0] A[rxBuff1] A ...[rxBuffN] NA P

  - If I2C_FAST_XFER_OPTION_LAST_RX_ACK is set in *options* member

    S Addr Rd[A][rxBuff0] A[rxBuff1] A ...[rxBuffN] A P

A sample code showing Read-only transfer using I2C_FastXfer() call.

```
xfer.options = 0;
xfer.txSz = 0;
xfer.rxSz = 4;
xfer.rxBuff = &buff[0];
xfer.slaveAddr = 0x60;
res = I2C_FastXfer(g_hI2CPort, &xfer);
```

- Read-Write transfer : When *rxSz* and @ txSz members of *xfer* are non - zero.

  S Addr Wr[A] txBuff0[A] txBuff1[A] ... txBuffN[A]

  S Addr Rd[A][rxBuff0] A[rxBuff1] A ...[rxBuffN] NA P

A sample code showing Read-Write transfer using I2C_FastXfer() call.

```
xfer.options = 0;
xfer.txSz = 4;
xfer.rxSz = 4;
xfer.rxBuff = &rxBuff[0];
xfer.txBuff = &txBuff[0];
xfer.slaveAddr = 0x60;
res = I2C_FastXfer(g_hI2CPort, &xfer);
```

2.5 Error propagation

Most APIs return zero or positive number on success while return negative numbers on error. The error types are defined by LPCUSBSIO_ERR_T enum. The library also provides I2C_Error() routine to obtain end-user presentable uni-code string. Application can use this routine to obtain more description of the last error.

2.6 Deploying LPCUSBSIO applications

For Windows hosts lpcusbsio.dll should be included with executable in distribution and be located in the same directory as executable. For OSx and Linux the library is linked statically hence no LIBUSBSIO components needs to be included with final distribution.

Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Module Documentation

6.1 LPC USB serial I/O (LPCUSBSIO) I2C API interface

6.1.1 Detailed Description

API description

The LPCUSBSIO-I2C APIs can be divided into two broad sets. The first set consists of five control APIs and the second set consists of two data transferring APIs. On error most APIs return an LPCUSBSIO_ERR_T code. Application code can call I2C_Error() routine to get user presentable uni-code string corresponding to the last error.

The current version of LPCUSBSIO allows communicating with I2C slave devices only.

Data Structures

- struct I2C_PORTCONFIG_T

  Port configuration information.
- struct I2C_FAST_XFER_T

  Fast transfer parameter structure.

Macros

- #define LPCUSBSIO_VID 0x1FC9
- #define LPCUSBSIO_PID 0x0088
- #define LPCUSBSIO_READ_TMO 500

Typedefs

- typedef void * LPC_HANDLE

  Handle type.

Enumerations

- enum LPCUSBSIO_ERR_T {
  LPCUSBSIO_OK = 0, LPCUSBSIO_ERR_HID_LIB = -1, LPCUSBSIO_ERR_BAD_HANDLE = -2, LPCUSBSIO_ERR_FATAL = -0x11,
  LPCUSBSIO_ERR_I2C_NAK = -0x12, LPCUSBSIO_ERR_I2C_BUS = -0x13, LPCUSBSIO_ERR_I2C_SLAVE_NAK = -0x14, LPCUSBSIO_ERR_I2C_ARBLOST = -0x15,
  LPCUSBSIO_ERR_TIMEOUT = -0x20, LPCUSBSIO_ERR_INVALID_CMD = -0x21, LPCUSBSIO_ERR_INVALID_PARAM = -0x22, LPCUSBSIO_ERR_PARTIAL_DATA = -0x23 }

Error types returned by LPCUSBSIO APIs.

- enum I2C_CLOCKRATE_T { I2C_CLOCK_STANDARD_MODE = 100000, I2C_CLOCK_FAST_MODE = 400000, I2C_CLOCK_FAST_MODE_PLUS = 1000000 }

  I2C clock rates.

Functions

- LPCUSBSIO_API const char * I2C_GetVersion (LPC_HANDLE handle)

  Get version string of the library.
- LPCUSBSIO_API int I2C_GetNumPorts (void)

  Get number I2C ports available on the LPC controller.
- LPCUSBSIO_API LPC_HANDLE I2C_Open (uint32_t index)

  Opens the indexed port.
- LPCUSBSIO_API int32_t I2C_Init (LPC_HANDLE handle, I2C_PORTCONFIG_T *config)

  Initialize the port.
- LPCUSBSIO_API int32_t I2C_Close (LPC_HANDLE handle)

  Closes a port.
- LPCUSBSIO_API int32_t I2C_Reset (LPC_HANDLE handle)

  Reset I2C Controller.
- LPCUSBSIO_API const wchar_t * I2C_Error (LPC_HANDLE handle)

  Get a string describing the last error which occurred.
- LPCUSBSIO_API int32_t I2C_DeviceRead (LPC_HANDLE handle, uint8_t deviceAddress, uint8_t *buffer, uint16_t sizeToTransfer, uint8_t options)

  Read from an addressed I2C slave.
- LPCUSBSIO_API int32_t I2C_DeviceWrite (LPC_HANDLE handle, uint8_t deviceAddress, uint8_t *buffer, uint16_t sizeToTransfer, uint8_t options)

  Writes to the addressed I2C slave.
- LPCUSBSIO_API int32_t I2C_FastXfer (LPC_HANDLE handle, I2C_FAST_XFER_T *xfer)

  Transmit and Receive data in master mode.

6.1.2 Macro Definition Documentation

6.1.2.1 #define I2C_CONFIG_OPTIONS_FORCE_INIT 0x00000001

I2C_CFG_OPTIONS Configuration options flags used by I2C_PORTCONFIG_TGenerate start condition before transmitting

6.1.2.2 #define I2C_FAST_XFER_OPTION_IGNORE_NACK 0x01

I2C_FAST_TRANSFER_OPTIONS I2C master faster transfer optionsIgnore NACK during data transfer. By default transfer is aborted.

6.1.2.3 #define I2C_FAST_XFER_OPTION_LAST_RX_ACK 0x02

ACK last byte received. By default we NACK last byte we receive per I2C specification.

6.1.2.4 #define I2C_TRANSFER_OPTIONS_BREAK_ON_NACK 0x0004

Continue transmitting data in bulk without caring about Ack or nAck from device if this bit is not set. If this bit is set then stop transmitting the data in the buffer when the device nAcks

6.1.2.5 #define I2C_TRANSFER_OPTIONS_NACK_LAST_BYTE 0x0008

lpcusbsio-I2C generates an ACKs for every byte read. Some I2C slaves require the I2C master to generate a nACK for the last data byte read. Setting this bit enables working with such I2C slaves

6.1.2.6 #define I2C_TRANSFER_OPTIONS_NO_ADDRESS 0x00000040

6.1.2.7 #define I2C_TRANSFER_OPTIONS_START_BIT 0x0001

I2C_IO_OPTIONS Options to I2C_DeviceWrite & I2C_DeviceRead routinesGenerate start condition before transmitting

6.1.2.8 #define I2C_TRANSFER_OPTIONS_STOP_BIT 0x0002

Generate stop condition at the end of transfer

6.1.2.9 #define LPCUSBSIO_PID 0x0088

USB-IF product ID for LPCUSBSIO devices.

6.1.2.10 #define LPCUSBSIO_READ_TMO 500

Read time-out value in milliseconds used by the library. If a response is not received

6.1.2.11 #define LPCUSBSIO_VID 0x1FC9

NXP USB-IF vendor ID.

6.1.3 Typedef Documentation

6.1.3.1 typedef void* LPC_HANDLE

Handle type.

6.1.4 Enumeration Type Documentation

6.1.4.1 enum I2C_CLOCKRATE_T

I2C clock rates.

Enumerator

I2C_CLOCK_STANDARD_MODE 100kb/sec

I2C_CLOCK_FAST_MODE 400kb/sec

I2C_CLOCK_FAST_MODE_PLUS 1000kb/sec

6.1.4.2 enum LPCUSBSIO_ERR_T

Error types returned by LPCUSBSIO APIs.

Enumerator

LPCUSBSIO_OK All API return positive number for success

LPCUSBSIO_ERR_HID_LIB HID library error.

LPCUSBSIO_ERR_BAD_HANDLE Handle passed to the function is invalid.

LPCUSBSIO_ERR_FATAL Fatal error occurred

LPCUSBSIO_ERR_I2C_NAK Transfer aborted due to NACK

LPCUSBSIO_ERR_I2C_BUS Transfer aborted due to bus error

LPCUSBSIO_ERR_I2C_SLAVE_NAK NAK received after SLA+W or SLA+R

LPCUSBSIO_ERR_I2C_ARBLOST I2C bus arbitration lost to other master

LPCUSBSIO_ERR_TIMEOUT Transaction timed out

LPCUSBSIO_ERR_INVALID_CMD Invalid HID_I2C Request or Request not supported in this version.

LPCUSBSIO_ERR_INVALID_PARAM Invalid parameters are provided for the given Request.

LPCUSBSIO_ERR_PARTIAL_DATA Partial transfer completed.

6.1.5 Function Documentation

6.1.5.1 LPCUSBSIO_API int32_t I2C_Close (LPC_HANDLE *handle*)

Closes a port.

Closes a port and frees all resources that were used by it.

Parameters

handle	: Handle of the port.

Returns

This function returns LPCUSBSIO_OK on success and negative error code on failure. Check LPCUSBSIO_-ERR_T for more details on error code.

6.1.5.2 LPCUSBSIO_API int32_t I2C_DeviceRead (LPC_HANDLE *handle*, uint8_t *deviceAddress*, uint8_t * *buffer*, uint16_t *sizeToTransfer*, uint8_t *options*)

Read from an addressed I2C slave.

This function reads the specified number of bytes from an addressed I2C slave. The *options* parameter effects the transfers. Some example transfers are shown below :

- When I2C_TRANSFER_OPTIONS_START_BIT, I2C_TRANSFER_OPTIONS_STOP_BIT and I2C_TRANSFER_OPTIONS_NACK_LAST_BYTE are set.

  S Addr Rd [A] [rxBuff0] A [rxBuff1] A ...[rxBuffN] NA P

- If I2C_TRANSFER_OPTIONS_NO_ADDRESS is also set.

S [rxBuff0] A [rxBuff1] A ...[rxBuffN] NA P

- if I2C_TRANSFER_OPTIONS_NACK_LAST_BYTE is not set

  S Addr Rd [A] [rxBuff0] A [rxBuff1] A ...[rxBuffN] A P

- If I2C_TRANSFER_OPTIONS_STOP_BIT is not set.

  S Addr Rd [A] [rxBuff0] A [rxBuff1] A ...[rxBuffN] NA

Parameters

handle	: Handle of the port.
deviceAddress	: Address of the I2C slave. This is a 7bit value and it should not contain the data direction bit, i.e. the decimal value passed should be always less than 128
buffer	: Pointer to the buffer where data is to be read
sizeToTransfer,:	Number of bytes to be read
options,:	This parameter specifies data transfer options. Check I2C_TRANSFER_OPTIONS_ macros.

Returns

This function returns number of bytes read on success and negative error code on failure. Check LPCUSBSIO_ERR_T for more details on error code.

6.1.5.3 LPCUSBSIO_API int32_t I2C_DeviceWrite (LPC_HANDLE *handle*, uint8_t *deviceAddress*, uint8_t * *buffer*, uint16_t *sizeToTransfer*, uint8_t *options*)

Writes to the addressed I2C slave.

This function writes the specified number of bytes to an addressed I2C slave. The *options* parameter effects the transfers. Some example transfers are shown below :

- When I2C_TRANSFER_OPTIONS_START_BIT, I2C_TRANSFER_OPTIONS_STOP_BIT and I2C_TRANSFER_OPTIONS_BREAK_ON_NACK are set.

  S Addr Wr[A] txBuff0[A] txBuff1[A] ... txBuffN[A] P

  – If I2C_TRANSFER_OPTIONS_NO_ADDRESS is also set.

  S txBuff0[A] ... txBuffN[A] P

  – if I2C_TRANSFER_OPTIONS_BREAK_ON_NACK is not set
  S Addr Wr[A] txBuff0[A or NA] ... txBuffN[A or NA] P
  – If I2C_TRANSFER_OPTIONS_STOP_BIT is not set.
  S Addr Wr[A] txBuff0[A] txBuff1[A] ... txBuffN[A]

Parameters

handle	: Handle of the port.
deviceAddress	: Address of the I2C slave. This is a 7bit value and it should not contain the data direction bit, i.e. the decimal value passed should be always less than 128
sizeToTransfer,:	Number of bytes to be written
buffer	: Pointer to the buffer where data is to be read
options	: This parameter specifies data transfer options. Check I2C_TRANSFER_OPTIONS_- macros.

Returns

This function returns number of bytes written on success and negative error code on failure. Check LPCUSBSIO_ERR_T for more details on error code.

6.1.5.4 LPCUSBSIO_API const wchar_t* I2C_Error (LPC_HANDLE *handle*)

Get a string describing the last error which occurred.

Parameters

handle	: A device handle returned from I2C_OpenPort().

Returns

This function returns a string containing the last error which occurred or NULL if none has occurred.

6.1.5.5 LPCUSBSIO_API int32_t I2C_FastXfer (LPC_HANDLE *handle*, I2C_FAST_XFER_T * *xfer*)

Transmit and Receive data in master mode.

The parameter *xfer* should have its member *slaveAddr* initialized to the 7 - Bit slave address to which the master will do the xfer, Bit0 to bit6 should have the address and Bit8 is ignored.During the transfer no code(like event handler) must change the content of the memory pointed to by *xfer.The* member of *xfer*, *txBuff* and *txSz* be initialized to the memory from which the I2C must pick the data to be transferred to slave and the number of bytes to send respectively, similarly *rxBuff* and *rxSz* must have pointer to memory where data received from slave be stored and the number of data to get from slave respectively.

Following types of transfers are possible :

- Write-only transfer : When *rxSz* member of *xfer* is set to 0.

  S Addr Wr[A] txBuff0[A] txBuff1[A] ... txBuffN[A] P

  - If I2C_FAST_XFER_OPTION_IGNORE_NACK is set in *options* member
    S Addr Wr[A] txBuff0[A or NA] ... txBuffN[A or NA] P

- Read-only transfer : When *txSz* member of *xfer* is set to 0.

  S Addr Rd[A][rxBuff0] A[rxBuff1] A ...[rxBuffN] NA P

  - If I2C_FAST_XFER_OPTION_LAST_RX_ACK is set in *options* member
    S Addr Rd[A][rxBuff0] A[rxBuff1] A ...[rxBuffN] A P

- Read-Write transfer : When *rxSz* and @ txSz members of *xfer* are non - zero.

  S Addr Wr[A] txBuff0[A] txBuff1[A] ... txBuffN[A]

  S Addr Rd[A][rxBuff0] A[rxBuff1] A ...[rxBuffN] NA P

Parameters

handle	: Handle of the port.
xfer	: Pointer to a I2C_FAST_XFER_T structure.

Returns

This function returns number of bytes read on success and negative error code on failure. Check LPCUSBSIO_ERR_T for more details on error code.

6.1.5.6 LPCUSBSIO_API int I2C_GetNumPorts (void)

Get number I2C ports available on the LPC controller.

This function gets the number of I2C ports that are available on the LPC controller. The number of ports available in each of these chips is different.

Returns

The number of ports available on the LPC controller.

6.1.5.7 LPCUSBSIO_API const char* I2C_GetVersion (LPC_HANDLE *handle*)

Get version string of the library.

Parameters

handle	: A device handle returned from I2C_OpenPort().

Returns

This function returns a string containing the version of the library. If the device handle passed is not NULL then the firmware version of the connected device is appended to the string.

6.1.5.8 LPCUSBSIO_API int32_t I2C_Init (LPC_HANDLE *handle*, I2C_PORTCONFIG_T * *config*)

Initialize the port.

This function initializes the port and the communication parameters associated with it.

Parameters

handle	: Handle of the port.
config	: Pointer to I2C_PORTCONFIG_T structure. Members of I2C_PORTCONFIG_T structure contains the values for I2C master clock, latency timer and Options

Returns

This function returns LPCUSBSIO_OK on success and negative error code on failure. Check LPCUSBSIO_ERR_T for more details on error code.

6.1.5.9 LPCUSBSIO_API LPC_HANDLE I2C_Open (uint32_t *index*)

Opens the indexed port.

This function opens the indexed port and provides a handle to it. Valid values for the index of port can be from 0 to the value obtained using I2C_GetNumPorts – 1).

Parameters

index	: Index of the port to be opened.

Returns

This function returns a handle to I2C port object on success or NULL on failure.

6.1.5.10 LPCUSBSIO_API int32_t I2C_Reset (LPC_HANDLE *handle*)

Reset I2C Controller.

Parameters

handle	: A device handle returned from I2C_OpenPort().

Returns

This function returns LPCUSBSIO_OK on success and negative error code on failure. Check LPCUSBSIO_ERR_T for more details on error code.

Chapter 7

Data Structure Documentation

7.1 I2C_FAST_XFER_T Struct Reference

7.1.1 Detailed Description

Fast transfer parameter structure.

`#include "lpcusbsio.h"`

Data Fields

- uint8_t txSz
- uint8_t rxSz
- uint8_t options
- uint8_t slaveAddr
- const uint8_t * txBuff
- uint8_t * rxBuff

7.1.2 Field Documentation

7.1.2.1 uint8_t options

Fast transfer options

7.1.2.2 uint8_t* rxBuff

Pointer memory where bytes received from I2C be stored

7.1.2.3 uint8_t rxSz

```
Number of bytes to received,
```

if 0 only transmission we be carried on

7.1.2.4 uint8_t slaveAddr

7-bit I2C Slave address

7.1.2.5 const uint8_t* txBuff

Pointer to array of bytes to be transmitted

7.1.2.6 uint8_t txSz

```
Number of bytes in transmit array,
```

if 0 only receive transfer will be carried on

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

- /Users/pdurgesh/git/lpcopen/hosttools/lpcusbsio/inc/lpcusbsio.h

7.2 I2C_PORTCONFIG_T Struct Reference

7.2.1 Detailed Description

Port configuration information.

```
#include "lpcusbsio.h"
```

Data Fields

- I2C_CLOCKRATE_T ClockRate
- uint32_t Options

7.2.2 Field Documentation

7.2.2.1 I2C_CLOCKRATE_T ClockRate

I2C Clock speed

7.2.2.2 uint32_t Options

Configuration options

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

- /Users/pdurgesh/git/lpcopen/hosttools/lpcusbsio/inc/lpcusbsio.h

Chapter 8

File Documentation

8.1 DevelopingWith.dox File Reference

8.1.1 Detailed Description

This file contains special DoxyGen information for the generation of the main page and other special documentation pages. It is not a project source file.

8.2 MainPage.dox File Reference

8.2.1 Detailed Description

This file contains special DoxyGen information for the generation of the main page and other special documentation pages. It is not a project source file.

8.3 /Users/pdurgesh/git/lpcopen/hosttools/lpcusbsio/inc/lpcusbsio.h File Reference

```
#include <stdint.h>
```

Data Structures

- struct I2C_PORTCONFIG_T

  Port configuration information.
- struct I2C_FAST_XFER_T

  Fast transfer parameter structure.

Macros

- #define LPCUSBSIO_API __declspec(dllimport)
- #define LPCUSBSIO_VID 0x1FC9
- #define LPCUSBSIO_PID 0x0088
- #define LPCUSBSIO_READ_TMO 500

- #define I2C_CONFIG_OPTIONS_FORCE_INIT 0x00000001

- #define I2C_TRANSFER_OPTIONS_START_BIT 0x0001
- #define I2C_TRANSFER_OPTIONS_STOP_BIT 0x0002
- #define I2C_TRANSFER_OPTIONS_BREAK_ON_NACK 0x0004
- #define I2C_TRANSFER_OPTIONS_NACK_LAST_BYTE 0x0008
- #define I2C_TRANSFER_OPTIONS_NO_ADDRESS 0x00000040

- #define I2C_FAST_XFER_OPTION_IGNORE_NACK 0x01
- #define I2C_FAST_XFER_OPTION_LAST_RX_ACK 0x02

Typedefs

- typedef void * LPC_HANDLE

  Handle type.

Enumerations

- enum LPCUSBSIO_ERR_T {
  LPCUSBSIO_OK = 0, LPCUSBSIO_ERR_HID_LIB = -1, LPCUSBSIO_ERR_BAD_HANDLE = -2, LPCUSBSIO_ERR_FATAL = -0x11,
  LPCUSBSIO_ERR_I2C_NAK = -0x12, LPCUSBSIO_ERR_I2C_BUS = -0x13, LPCUSBSIO_ERR_I2C_SLAVE_NAK = -0x14, LPCUSBSIO_ERR_I2C_ARBLOST = -0x15,
  LPCUSBSIO_ERR_TIMEOUT = -0x20, LPCUSBSIO_ERR_INVALID_CMD = -0x21, LPCUSBSIO_ERR_INVALID_PARAM = -0x22, LPCUSBSIO_ERR_PARTIAL_DATA = -0x23 }

  Error types returned by LPCUSBSIO APIs.
- enum I2C_CLOCKRATE_T { I2C_CLOCK_STANDARD_MODE = 100000, I2C_CLOCK_FAST_MODE = 400000, I2C_CLOCK_FAST_MODE_PLUS = 1000000 }

  I2C clock rates.

Functions

- LPCUSBSIO_API const char * I2C_GetVersion (LPC_HANDLE handle)

  Get version string of the library.
- LPCUSBSIO_API int I2C_GetNumPorts (void)

  Get number I2C ports available on the LPC controller.
- LPCUSBSIO_API LPC_HANDLE I2C_Open (uint32_t index)

  Opens the indexed port.
- LPCUSBSIO_API int32_t I2C_Init (LPC_HANDLE handle, I2C_PORTCONFIG_T *config)

  Initialize the port.
- LPCUSBSIO_API int32_t I2C_Close (LPC_HANDLE handle)

  Closes a port.
- LPCUSBSIO_API int32_t I2C_Reset (LPC_HANDLE handle)

  Reset I2C Controller.
- LPCUSBSIO_API const wchar_t * I2C_Error (LPC_HANDLE handle)

  Get a string describing the last error which occurred.
- LPCUSBSIO_API int32_t I2C_DeviceRead (LPC_HANDLE handle, uint8_t deviceAddress, uint8_t *buffer, uint16_t sizeToTransfer, uint8_t options)

  Read from an addressed I2C slave.
- LPCUSBSIO_API int32_t I2C_DeviceWrite (LPC_HANDLE handle, uint8_t deviceAddress, uint8_t *buffer, uint16_t sizeToTransfer, uint8_t options)

  Writes to the addressed I2C slave.
- LPCUSBSIO_API int32_t I2C_FastXfer (LPC_HANDLE handle, I2C_FAST_XFER_T *xfer)

  Transmit and Receive data in master mode.

8.3.1 Macro Definition Documentation

8.3.1.1 #define LPCUSBSIO_API __declspec(dllimport)

Index