LPC USB Serial I/O (LPCUSBSIO)

2.00

Generated by Doxygen 1.8.8

Thu Nov 20 2014 11:24:22

Contents

1 LPC USB Serial I/O Library **1**

1.1 Introduction 1

1.2 LPCUSBSIO Architecture 1

2 Developing Serial IO application with LPCUSBSIO **3**

2.1 Prerequisites 3

2.2 Setting-up project environment 3

2.3 Obtaining Serial IO device handle 4

2.4 Initializing and obtaining I2C port handle 5

2.5 Reading and Writing to an I2C slave device 6

2.5.1 Uni-directional transfer routines 6

2.5.1.1 Using I2C_DeviceWrite 7

2.5.1.2 Using I2C_DeviceRead 7

2.5.2 Bi-directional transfer routines 8

2.6 Initializing and obtaining SPI port handle 8

2.7 Data transfer with a SPI slave device 9

2.7.1 SPI transfer routine 9

2.8 GPIO Routines 10

2.9 Error propagation 10

2.10 Deploying LPCUSBSIO applications 10

3 Module Index **11**

3.1 Modules 11

4 Data Structure Index **13**

4.1 Data Structures 13

5 File Index **15**

5.1 File List 15

6 Module Documentation **17**

6.1 LPC USB serial I/O (LPCUSBSIO) API interface 17

6.1.1 Detailed Description 17

6.1.2 Macro Definition Documentation 19
6.1.2.1 I2C_FAST_XFER_OPTION_IGNORE_NACK 19
6.1.2.2 I2C_FAST_XFER_OPTION_LAST_RX_ACK 19
6.1.2.3 I2C_TRANSFER_OPTIONS_BREAK_ON_NACK 19
6.1.2.4 I2C_TRANSFER_OPTIONS_NACK_LAST_BYTE 19
6.1.2.5 I2C_TRANSFER_OPTIONS_NO_ADDRESS 20
6.1.2.6 I2C_TRANSFER_OPTIONS_START_BIT 20
6.1.2.7 I2C_TRANSFER_OPTIONS_STOP_BIT 20
6.1.2.8 LPCUSBSIO_GEN_SPI_DEVICE_NUM 20
6.1.2.9 LPCUSBSIO_PID 20
6.1.2.10 LPCUSBSIO_READ_TMO 20
6.1.2.11 LPCUSBSIO_VID 20
6.1.3 Typedef Documentation 20
6.1.3.1 LPC_HANDLE 20
6.1.4 Enumeration Type Documentation 20
6.1.4.1 I2C_CLOCKRATE_T 20
6.1.4.2 LPCUSBSIO_ERR_T 20
6.1.5 Function Documentation 21
6.1.5.1 GPIO_ClearPin 21
6.1.5.2 GPIO_ClearPort 21
6.1.5.3 GPIO_ConfigIOPin 22
6.1.5.4 GPIO_GetPin 23
6.1.5.5 GPIO_GetPortDir 23
6.1.5.6 GPIO_ReadPort 23
6.1.5.7 GPIO_SetPin 24
6.1.5.8 GPIO_SetPort 24
6.1.5.9 GPIO_SetPortInDir 24
6.1.5.10 GPIO_SetPortOutDir 25
6.1.5.11 GPIO_TogglePin 25
6.1.5.12 GPIO_WritePort 25
6.1.5.13 I2C_Close 26
6.1.5.14 I2C_DeviceRead 26
6.1.5.15 I2C_DeviceWrite 27
6.1.5.16 I2C_FastXfer 27
6.1.5.17 I2C_Open 28
6.1.5.18 I2C_Reset 28
6.1.5.19 LPCUSBSIO_Close 29
6.1.5.20 LPCUSBSIO_Error 29
6.1.5.21 LPCUSBSIO_GetLastError 29
6.1.5.22 LPCUSBSIO_GetMaxDataSize 29

6.1.5.23 LPCUSBSIO_GetNumGPIOPorts . . . 30
6.1.5.24 LPCUSBSIO_GetNumI2CPorts . . . 31
6.1.5.25 LPCUSBSIO_GetNumPorts . . . 31
6.1.5.26 LPCUSBSIO_GetNumSPIPorts . . . 31
6.1.5.27 LPCUSBSIO_GetVersion . . . 31
6.1.5.28 LPCUSBSIO_Open . . . 32
6.1.5.29 SPI_Close . . . 32
6.1.5.30 SPI_Open . . . 32
6.1.5.31 SPI_Reset . . . 33
6.1.5.32 SPI_Transfer . . . 34

7 Data Structure Documentation 35

7.1 I2C_FAST_XFER_T Struct Reference . . . 35
7.1.1 Detailed Description . . . 35
7.1.2 Field Documentation . . . 35
7.1.2.1 options . . . 35
7.1.2.2 rxBuff . . . 35
7.1.2.3 rxSz . . . 35
7.1.2.4 slaveAddr . . . 35
7.1.2.5 txBuff . . . 36
7.1.2.6 txSz . . . 36
7.2 I2C_PORTCONFIG_T Struct Reference . . . 36
7.2.1 Detailed Description . . . 36
7.2.2 Field Documentation . . . 36
7.2.2.1 ClockRate . . . 36
7.2.2.2 Options . . . 36
7.3 SPI_XFER_T Struct Reference . . . 36
7.3.1 Detailed Description . . . 36
7.3.2 Field Documentation . . . 37
7.3.2.1 device . . . 37
7.3.2.2 length . . . 37
7.3.2.3 options . . . 37
7.3.2.4 rxBuff . . . 37
7.3.2.5 txBuff . . . 37

8 File Documentation 39

8.1 DevelopingWith.dox File Reference . . . 39
8.1.1 Detailed Description . . . 39
8.2 MainPage.dox File Reference . . . 39
8.2.1 Detailed Description . . . 39
8.3 C:/Source/lpcopen/hosttools/lpcusbsio/inc/lpcusbsio.h File Reference . . . 39

8.3.1 Macro Definition Documentation 42
8.3.1.1 LPCUSBSIO_API 42

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

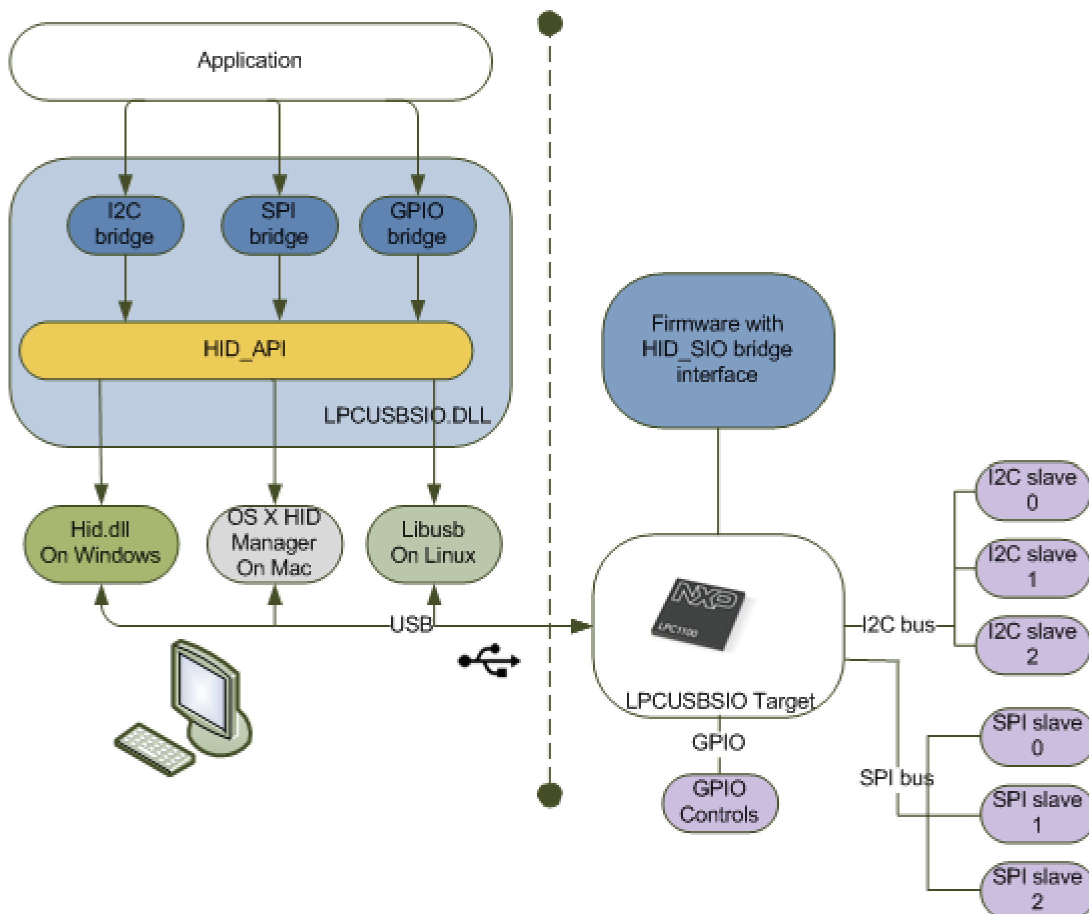
The current version of LPCUSBSIO allows communicating with I2C, SPI slave devices and GPIO.

See also:

- LPCUSBSIO Architecture
- Developing Serial IO application with LPCUSBSIO
- LPC USB serial I/O (LPCUSBSIO) 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 and connects to the LPC micro-controller through USB. The PC application calls API's from the LPCUSBSIO library which in turn uses the HID API to transact with the LPC micro-controller over HID USB interface. The LPC micro-controller and the Serial IO devices would usually be on the same PCB. When communicating over I2C or SPI, the LPC micro-controller will operate in 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. The LPC micro-controller uses the GPIO port pins to chip select the SPI slave devices. When doing a SPI transfer the slave device address is provided by the application as a combination of port and pin number.
Also any of the GPIO pins available on the LPC micro-controller can be configured as a GPIO Input or Output pin and can be controlled through the PC application.

Chapter 2

Developing Serial IO application with LPCUSBSIO

This section introduces you to application development using LPCUSBSIO library.

- Prerequisites
- Setting-up project environment
- Obtaining Serial IO device handle
- Initializing and obtaining I2C port handle
- Reading and Writing to an I2C slave device
- Initializing and obtaining SPI port handle
- Data transfer with a SPI slave device
- GPIO Routines
- Error propagation
- Deploying LPCUSBSIO applications

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.
  - The Visual studio project in the parent lpcusbsio directory contains both library and the test application. For convenience a project for the test application alone is created in apps/testapp directory.

- 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.
  - The makefile for compiling the LPCUSBSIO library is in the parent lpcusbsio directory and the makefile for compiling the test application is in the apps/testapp directory.
- 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`
  - The makefile for compiling the LPCUSBSIO library is in the parent lpcusbsio directory and the makefile for compiling the test application is in the apps/testapp directory.

2.3 Obtaining Serial IO device handle

- LPCUSBSIO library enumeration of all LPCUSBSIO target devices are combined into one API call of LPCUSBSIO_GetNumPorts().
  - All applications using the library should call this routine first. This API takes Vendor ID and Product ID as its parameters.
  - If the target LPC controller has 2 HID_SIO 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 Serial IO port by calling LPCUSBSIO_Open() then the corresponding port will not be reported in subsequent call of LPCUSBSIO_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 (starting from zero) 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 LPCUSBSIO_Open() with an index.
  - The handle returned by LPCUSBSIO_Open() should be used by all consequent calls targeted to the device.
  - If the port is claimed by a parallel thread or application a NULL handle is returned.
- After obtaining the handle for the Serial IO device, the number of I2C, SPI and GPIO ports available can be found out by calling LPCUSBSIO_GetNumI2CPorts(), LPCUSBSIO_GetNumSPIPorts() and LPCUSBSIO_GetNumGPIOPorts() respectively. These API's take the device handle as a parameter. Once the number of I2C, SPI and GPIO ports are known then the individual ports can be opened using the Serial IO device handle and port number to obtain a specific port handle which should be used for subsequent calls targeted to that port. The maximum data transfer size is specified by the LPC Serial IO firmware when the device is opened and this can be found out by calling LPCUSBSIO_GetMaxDataSize(). This API also takes the device handle as the parameter.
- The Serial IO device handle should be closed by calling LPCUSBSIO_Close() once it is no more required by the application. Below is a code snippet that explains the above steps.

```
#define LPCUSBSIO_VID 0x1FC9
#define LPCUSBSIO_PID 0x0090

LPC_HANDLE hSIOPort;
int res;
```

```
res = LPCUSBSIO_GetNumPorts(LPCUSBSIO_VID,
      LPCUSBSIO_PID);

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

 /*open device at index 0 */
 hSIOPort = LPCUSBSIO_Open(0);

 if(hSIOPort != NULL) {
  printf("Device version: %s \r\n ", LPCUSBSIO_GetVersion(hSIOPort));

  .....
  .....
  .....

  LPCUSBSIO_Close(hSIOPort);
 }
}
else {
 printf("Error: No free ports. \r\n");
}
```

2.4 Initializing and obtaining I2C port handle

- Once the Serial IO handle is obtained and the number of I2C ports available on the LPC micro-controller is known then individual I2C ports can be opened and I2C transfers can be initiated.
- To obtain a I2C port handle the application should call I2C_Open() with the Serial IO device handle, port number and port configuration as the parameters. This also initialize the corresponding I2C port on the LPC microcontroller. Note that the port numbers start from 0. The port configuration sets the I2C bus speed. Note that this is the speed at which the I2C devices on the board can communicate.
- Once the I2C port handle is obtained i.e. the port handle is not NULL, the port is ready for data transfers. The I2C port handle should be closed by calling I2C_Close() once it is no more required by the application. Following code snippet shows the initialization steps.

```
#define LPCUSBSIO_VID 0x1FC9
#define LPCUSBSIO_PID 0x0090

LPC_HANDLE hSIOPort;
int res;
LPC_HANDLE hI2CPort = NULL;
I2C_PORTCONFIG_T cfgParam;

res = LPCUSBSIO_GetNumPorts(LPCUSBSIO_VID,
      LPCUSBSIO_PID);

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

 /*open device at index 0 */
 hSIOPort = LPCUSBSIO_Open(0);

 if(hSIOPort != NULL) {

  printf("Device version: %s \r\n ", LPCUSBSIO_GetVersion(hSIOPort));

  printf("Number of I2C ports available: %u \r\n ", LPCUSBSIO_GetNumI2CPorts(
      hSIOPort));

  if(LPCUSBSIO_GetNumI2CPorts(hSIOPort) > 0) {
   /*Init the I2C port for standard speed communication */
   cfgParam.ClockRate = I2C_CLOCK_STANDARD_MODE;
   cfgParam.Options = 0;
   /* open I2C0 port */
   hI2CPort = I2C_Open(hSIOPort, &cfgParam, 0);

   if(hI2CPort != NULL) {

    Perform I2C transfers
    .....
    .....
    .....

    I2C_Close(hI2CPort);
   }
  }
```

```
  LPCUSBSIO_Close(hSIOPort);
 }
}
else {
 printf("Error: No free ports. \r\n");
}
```

2.5 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 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.
- **Data** (8 bits): A plain data byte.
- [..]: Data sent by I2C device, as opposed to data sent by the host adapter.

2.5.1 Uni-directional transfer routines

The data transfer size supported by the transfer routines depend on the transfer size supported by the firmware which can be found out by calling LPCUSBSIO_GetMaxDataSize() and since this is typically a larger number (512/1024 bytes) and the HID report size is 64 bytes, each data transfer over USB is split into multiple packets of 64 bytes (header for book keeping + data bytes). Once the data is transferred to the LPC micro-controller the micro-controller performs the I2C transfer and returns a response plus data for I2C read. The response from the LPC micro-controller is also split into multiple USB packets for large data. Since the USB transfer is split into multiple packets, to maintain data integrity the USB transfers are protected by a mutex to maintain synchronization. So even though multiple threads can call the I2C transfer API's only one thread will get access to the HID USB transfer at a time.

The I2C transaction done through 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 : 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.

2.5.1.1 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 |
      I2C_TRANSFER_OPTIONS_BREAK_ON_NACK);
```

- 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]

2.5.1.2 Using I2C_DeviceRead

A sample code showing I2C_DeviceRead() call.

```
res = I2C_DeviceRead(
 g_hI2CPort,
 0x60,
 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

2.5.2 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.txBuff = &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);
  ```

2.6 Initializing and obtaining SPI port handle

- Once the Serial IO handle is obtained and the number of SPI ports available on the LPC micro-controller is known then individual SPI ports can be opened and SPI transfers can be initiated.
- To obtain a SPI port handle the application should call SPI_Open() with the Serial IO device handle, port number and port configuration as the parameters. This also initialize the corresponding SPI port on the LPC micro-controller. Note that the port numbers start from 0. The port configuration sets the SPI data width, clock rate, clock polarity/phase and pre/post delay between data transmission and chip select assert/deassert.

- Once the SPI port handle is obtained i.e. the port handle is not NULL, the port is ready for data transfers. The SPI port handle should be closed by calling SPI_Close() once it is no more required by the application. Following code snippet shows the initialization steps.

```
#define LPCUSBSIO_VID 0x1FC9
#define LPCUSBSIO_PID 0x0090

LPC_HANDLE hSIOPort;
int res;
LPC_HANDLE hSPIPort = NULL;
HID_SPI_PORTCONFIG_T cfgParam;

res = LPCUSBSIO_GetNumPorts(LPCUSBSIO_VID,
      LPCUSBSIO_PID);

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

 /*open device at index 0 */
 hSIOPort = LPCUSBSIO_Open(0);

 if(hSIOPort != NULL) {

  printf("Device version: %s \r\n ", LPCUSBSIO_GetVersion(hSIOPort));

  printf("Number of SPI ports available: %u \r\n ", LPCUSBSIO_GetNumSPIPorts(
      hSIOPort));

  if(LPCUSBSIO_GetNumSPIPorts(hSIOPort) > 0) {
   /*Init the SPI port for 1MHz communication */
   cfgParam.busSpeed = 1000000;
   cfgParam.Options = HID_SPI_CONFIG_OPTION_DATA_SIZE_8 | HID_SPI_CONFIG_OPTION_POL_0 |
      HID_SPI_CONFIG_OPTION_PHA_0 | HID_SPI_CONFIG_OPTION_PRE_DELAY(0) | HID_SPI_CONFIG_OPTION_POST_DELAY(0);
   /* open SPI0 port */
   hSPIPort = SPI_Open(hSIOPort, &cfgParam, 0);

   if(hSPIPort != NULL) {

    Perform SPI transfers
    .....
    .....
    .....

    SPI_Close(hSPIPort);
   }
  }

  LPCUSBSIO_Close(hSIOPort);
 }
}
else {
 printf("Error: No free ports. \r\n");
}
```

2.7 Data transfer with a SPI slave device

Since the SPI is a full duplex communication protocol, transmit and receive happen at the same time and both the transmit and receive length are the same. So the library provides a single transfer API SPI_Transfer() for transferring data to and from SPI slave devices connected to the target LPC controller. Note that the SPI device number is a combination of the GPIO port and pin number used for the chip select of the SPI slave device.

2.7.1 SPI transfer routine

The data transfer size supported by the transfer routines depend on the transfer size supported by the firmware which can be found out by calling LPCUSBSIO_GetMaxDataSize() and since this is typically a larger number (512/1024 bytes) and the HID report size is 64 bytes, each data transfer over USB is split into multiple packets of 64 bytes (header for book keeping + data bytes). Once the data is transferred to the LPC micro-controller the micro-controller performs the SPI transfer and returns received data. The response from the LPC micro-controller is also split into multiple USB packets for large data. Since the USB transfer is split into multiple packets, to maintain data integrity the USB transfers are protected by a mutex to maintain synchronization. So even though multiple threads can call the SPI transfer API only one thread will get access to the HID USB transfer at a time.

A sample code showing SPI transfer using SPI_Transfer() call.

```
xfer.options = 0;
xfer.length = length;
xfer.txBuff = &tx_buff[0];
xfer.rxBuff = &rx_buff[0];
xfer.device = (uint8_t)LPCUSBSIO_GEN_SPI_DEVICE_NUM(1, 2);
res = SPI_Transfer(hSPIPort, &xfer);
```

2.8 GPIO Routines

The LPCUSBSIO library provides different API's to set the direction of a GPIO pin, set the GPIO pin state individually as well as writing or reading the entire port. Refer to the Module documentation for a description of each API. But before an IO pin is to be used as a GPIO it should be configure to do by calling GPIO_ConfigIOPin() by passing the device handle, IO port number, IO pin number and IO mode. The IO mode should be set according to the user manual of the LPC micro-controller. The API LPCUSBSIO_GetNumGPIOPorts() returns the number of GPIO ports supported by the micro-controller and each port usually has 32 pins.

2.9 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 LPCUSBSIO_Error() routine to obtain end-user presentable uni-code string. Application can use this routine to obtain more description of the last error.

2.10 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:

LPC USB serial I/O (LPCUSBSIO) API interface 17

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

I2C_FAST_XFER_T
I2C Fast transfer parameter structure 35
I2C_PORTCONFIG_T
I2C Port configuration information 36
SPI_XFER_T
SPI transfer parameter structure 36

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

C:/Source/lpcopen/hosttools/lpcusbsio/inc/lpcusbsio.h 39

Chapter 6

Module Documentation

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

6.1.1 Detailed Description

API description
The LPCUSBSIO 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 LPCUSBSIO_Error() routine to get user presentable uni-code string corresponding to the last error.
The current version of LPCUSBSIO allows communicating with I2C, SPI slave devices and GPIO.

Data Structures

- struct I2C_PORTCONFIG_T

  I2C Port configuration information.
- struct I2C_FAST_XFER_T

  I2C Fast transfer parameter structure.
- struct SPI_XFER_T

  SPI transfer parameter structure.

Macros

- #define LPCUSBSIO_VID 0x1FC9
- #define LPCUSBSIO_PID 0x0090
- #define LPCUSBSIO_READ_TMO 500
- #define LPCUSBSIO_GEN_SPI_DEVICE_NUM(port, pin) ((((uint8_t)(port) & 0x07) << 5) | ((pin) & 0x1F))

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_SYNCHRONIZATION = -3,
  LPCUSBSIO_ERR_MEM_ALLOC = -4, LPCUSBSIO_ERR_MUTEX_CREATE = -5, LPCUSBSIO_ERR_F↩

ATAL = -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 int LPCUSBSIO_GetNumPorts (uint32_t vid, uint32_t pid)

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

  Opens the indexed Serial IO port.
- LPCUSBSIO_API int32_t LPCUSBSIO_Close (LPC_HANDLE hUsbSio)

  Closes a LPC Serial IO port.
- LPCUSBSIO_API const char ∗ LPCUSBSIO_GetVersion (LPC_HANDLE hUsbSio)

  Get version string of the LPCUSBSIO library.
- LPCUSBSIO_API const wchar_t ∗ LPCUSBSIO_Error (LPC_HANDLE hUsbSio)

  Get a string describing the last error which occurred.
- LPCUSBSIO_API uint32_t LPCUSBSIO_GetNumI2CPorts (LPC_HANDLE hUsbSio)

  Returns the number of I2C ports supported by Serial IO device.
- LPCUSBSIO_API uint32_t LPCUSBSIO_GetNumSPIPorts (LPC_HANDLE hUsbSio)

  Returns the number of SPI ports supported by Serial IO device.
- LPCUSBSIO_API uint32_t LPCUSBSIO_GetNumGPIOPorts (LPC_HANDLE hUsbSio)

  Returns the number of GPIO ports supported by Serial IO device.
- LPCUSBSIO_API uint32_t LPCUSBSIO_GetMaxDataSize (LPC_HANDLE hUsbSio)

  Returns the max number of bytes supported for I2C/SPI transfers by the Serial IO device.
- LPCUSBSIO_API int32_t LPCUSBSIO_GetLastError (void)

  Returns the last error seen by the Library.
- LPCUSBSIO_API LPC_HANDLE I2C_Open (LPC_HANDLE hUsbSio, I2C_PORTCONFIG_T ∗config, uint8_t portNum)

  Initialize a I2C port.
- LPCUSBSIO_API int32_t I2C_Close (LPC_HANDLE hI2C)

  Closes a I2C port.
- LPCUSBSIO_API int32_t I2C_Reset (LPC_HANDLE hI2C)

  Reset I2C Controller.
- LPCUSBSIO_API int32_t I2C_DeviceRead (LPC_HANDLE hI2C, 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 hI2C, 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 hI2C, I2C_FAST_XFER_T ∗xfer)

  Transmit and Receive data in I2C master mode.
- LPCUSBSIO_API LPC_HANDLE SPI_Open (LPC_HANDLE hUsbSio, HID_SPI_PORTCONFIG_T ∗config, uint8_t portNum)

  Initialize a SPI port.
- LPCUSBSIO_API int32_t SPI_Close (LPC_HANDLE hSPI)

Closes a SPI port.
- LPCUSBSIO_API int32_t SPI_Transfer (LPC_HANDLE hSPI, SPI_XFER_T ∗xfer)
  Transmit and Receive data in SPI master mode.
- LPCUSBSIO_API int32_t SPI_Reset (LPC_HANDLE hSPI)
  Reset SPI Controller.
- LPCUSBSIO_API int32_t GPIO_ReadPort (LPC_HANDLE hUsbSio, uint8_t port, uint32_t ∗status)
  Read a GPIO port.
- LPCUSBSIO_API int32_t GPIO_WritePort (LPC_HANDLE hUsbSio, uint8_t port, uint32_t ∗status)
  Write to a GPIO port.
- LPCUSBSIO_API int32_t GPIO_SetPort (LPC_HANDLE hUsbSio, uint8_t port, uint32_t pins)
  Set GPIO port bits.
- LPCUSBSIO_API int32_t GPIO_ClearPort (LPC_HANDLE hUsbSio, uint8_t port, uint32_t pins)
  Clear GPIO port bits.
- LPCUSBSIO_API int32_t GPIO_GetPortDir (LPC_HANDLE hUsbSio, uint8_t port, uint32_t ∗pPins)
  Read GPIO port direction bits.
- LPCUSBSIO_API int32_t GPIO_SetPortOutDir (LPC_HANDLE hUsbSio, uint8_t port, uint32_t pins)
  Sets GPIO port pins direction to output.
- LPCUSBSIO_API int32_t GPIO_SetPortInDir (LPC_HANDLE hUsbSio, uint8_t port, uint32_t pins)
  Sets GPIO port pins direction to input.
- LPCUSBSIO_API int32_t GPIO_SetPin (LPC_HANDLE hUsbSio, uint8_t port, uint8_t pin)
  Sets a specific GPIO port pin value to high.
- LPCUSBSIO_API int32_t GPIO_GetPin (LPC_HANDLE hUsbSio, uint8_t port, uint8_t pin)
  Reads the state of a specific GPIO port pin.
- LPCUSBSIO_API int32_t GPIO_ClearPin (LPC_HANDLE hUsbSio, uint8_t port, uint8_t pin)
  Clears a specific GPIO port pin.
- LPCUSBSIO_API int32_t GPIO_TogglePin (LPC_HANDLE hUsbSio, uint8_t port, uint8_t pin)
  Toggles the state of a specific GPIO port pin.
- LPCUSBSIO_API int32_t GPIO_ConfigIOPin (LPC_HANDLE hUsbSio, uint8_t port, uint8_t pin, uint32_↩
  t mode)
  Configures the IO mode for a specific GPIO port pin.

6.1.2 Macro Definition Documentation

6.1.2.1 #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.

6.1.2.2 #define I2C_FAST_XFER_OPTION_LAST_RX_ACK 0x02

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

6.1.2.3 #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

6.1.2.4 #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

6.1.2.5 #define I2C_TRANSFER_OPTIONS_NO_ADDRESS 0x00000040

6.1.2.6 #define I2C_TRANSFER_OPTIONS_START_BIT 0x0001

I2C_IO_OPTIONS Options to I2C_DeviceWrite & I2C_DeviceRead routinesGenerate start condition before transmitting

6.1.2.7 #define I2C_TRANSFER_OPTIONS_STOP_BIT 0x0002

Generate stop condition at the end of transfer

6.1.2.8 #define LPCUSBSIO_GEN_SPI_DEVICE_NUM(*port*, *pin*) ((((uint8_t)(port) & 0x07) << 5) | ((pin) & 0x1F))

Macro to generate SPI device number from port and pin

6.1.2.9 #define LPCUSBSIO_PID 0x0090

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_SYNCHRONIZATION Thread Synchronization error.

LPCUSBSIO_ERR_MEM_ALLOC Memory Allocation error.

LPCUSBSIO_ERR_MUTEX_CREATE Mutex Creation error.

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_SIO 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 GPIO_ClearPin (LPC_HANDLE *hUsbSio*, uint8_t *port*, uint8_t *pin*)

Clears a specific GPIO port pin.

Clears a specific pin indicated by *pin* for the GPIO port mentioned by *port*. Each port has 32 pins associated with it.

Parameters

hUsbSio	Handle to LPCUSBSIO port.
port	: GPIO port number.
pin	: The pin number which needs to be cleared (0 - 31).

Returns

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

6.1.5.2 LPCUSBSIO_API int32_t GPIO_ClearPort (LPC_HANDLE *hUsbSio*, uint8_t *port*, uint32_t *pins*)

Clear GPIO port bits.

Clears the selected pins of the GPIO port mentioned by *port*. Each port has 32 pins associated with it. The pins selected are indicated by the corresponding high bits of *pins*

Parameters

hUsbSio	Handle to LPCUSBSIO port.
port	: GPIO port number.
pins	: Indicates which pins need to be cleared by setting the corresponding bit in this variable.

Returns

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

6.1.5.3 LPCUSBSIO_API int32_t GPIO_ConfigIOPin (LPC_HANDLE *hUsbSio*, uint8_t *port*, uint8_t *pin*, uint32_t *mode*)

Configures the IO mode for a specific GPIO port pin.

Configures the IO mode of a specific pin indicated by *pin* for the GPIO port mentioned by *port* to the value mentioned by *mode*. Each port has 32 pins associated with it.

Parameters

hUsbSio	Handle to LPCUSBSIO port.
port	: GPIO port number.
pin	: The pin number which needs to be toggled (0 - 31).
mode	: The 32 bit IO mode value that needs to updated.

Returns

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

6.1.5.4 LPCUSBSIO_API int32_t GPIO_GetPin (LPC_HANDLE *hUsbSio,* uint8_t *port,* uint8_t *pin*)

Reads the state of a specific GPIO port pin.

Read the state of a specific pin indicated by *pin* for the GPIO port mentioned by *port*. Each port has 32 pins associated with it.

Parameters

hUsbSio	Handle to LPCUSBSIO port.
port	: GPIO port number.
pin	: The pin number which needs to be read (0 - 31).

Returns

This function returns negative error code on failure and returns the pin state (0 or 1) upon on success. Check LPCUSBSIO_ERR_T for more details on error code.

6.1.5.5 LPCUSBSIO_API int32_t GPIO_GetPortDir (LPC_HANDLE *hUsbSio,* uint8_t *port,* uint32_t ∗ *pPins*)

Read GPIO port direction bits.

Reads the direction status for all pins of the GPIO port mentioned by *port*. Each port has 32 pins associated with it.

Parameters

hUsbSio	Handle to LPCUSBSIO port.
port	: GPIO port number.
pPins	: Pointer to GPIO port direction status, which is updated by the function.

Returns

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

6.1.5.6 LPCUSBSIO_API int32_t GPIO_ReadPort (LPC_HANDLE *hUsbSio,* uint8_t *port,* uint32_t ∗ *status*)

Read a GPIO port.

Reads the pin status of the GPIO port mentioned by *port*. Each port has 32 pins associated with it.

Parameters

hUsbSio	Handle to LPCUSBSIO port.
port	: GPIO port number.
status	: Pointer to GPIO port status, which is updated by the function.

Returns

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

6.1.5.7 LPCUSBSIO_API int32_t GPIO_SetPin (LPC_HANDLE *hUsbSio*, uint8_t *port*, uint8_t *pin*)

Sets a specific GPIO port pin value to high.

Sets a specific pin indicated by *pin* to high value for the GPIO port mentioned by *port*. Each port has 32 pins associated with it.

Parameters

hUsbSio	Handle to LPCUSBSIO port.
port	: GPIO port number.
pin	: The pin number which needs to be set (0 - 31).

Returns

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

6.1.5.8 LPCUSBSIO_API int32_t GPIO_SetPort (LPC_HANDLE *hUsbSio*, uint8_t *port*, uint32_t *pins*)

Set GPIO port bits.

Sets the selected pins of the GPIO port mentioned by *port*. Each port has 32 pins associated with it. The pins selected are indicated by the corresponding high bits of *pins*

Parameters

hUsbSio	Handle to LPCUSBSIO port.
port	: GPIO port number.
pins	: Indicates which pins need to be set high by setting the corresponding bit in this variable.

Returns

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

6.1.5.9 LPCUSBSIO_API int32_t GPIO_SetPortInDir (LPC_HANDLE *hUsbSio*, uint8_t *port*, uint32_t *pins*)

Sets GPIO port pins direction to input.

Sets the direction of selected pins as input for the GPIO port mentioned by *port*. Each port has 32 pins associated with it. The pins selected are indicated by the corresponding high bits of *pins*

Parameters

hUsbSio	Handle to LPCUSBSIO port.
port	: GPIO port number.
pins	: Indicates which pins are input pins by setting the corresponding bit in this variable.

Returns

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

6.1.5.10 LPCUSBSIO_API int32_t GPIO_SetPortOutDir (LPC_HANDLE *hUsbSio,* uint8_t *port,* uint32_t *pins*)

Sets GPIO port pins direction to output.

Sets the direction of selected pins as output for the GPIO port mentioned by *port*. Each port has 32 pins associated with it. The pins selected are indicated by the corresponding high bits of *pins*

Parameters

hUsbSio	Handle to LPCUSBSIO port.
port	: GPIO port number.
pins	: Indicates which pins are output pins by setting the corresponding bit in this variable.

Returns

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

6.1.5.11 LPCUSBSIO_API int32_t GPIO_TogglePin (LPC_HANDLE *hUsbSio,* uint8_t *port,* uint8_t *pin*)

Toggles the state of a specific GPIO port pin.

Toggles the state of a specific pin indicated by *pin* for the GPIO port mentioned by *port*. Each port has 32 pins associated with it.

Parameters

hUsbSio	Handle to LPCUSBSIO port.
port	: GPIO port number.
pin	: The pin number which needs to be toggled (0 - 31).

Returns

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

6.1.5.12 LPCUSBSIO_API int32_t GPIO_WritePort (LPC_HANDLE *hUsbSio,* uint8_t *port,* uint32_t * *status*)

Write to a GPIO port.

Write the pin status of the GPIO port mentioned by *port*. Each port has 32 pins associated with it.

Parameters

hUsbSio	Handle to LPCUSBSIO port.
port	: GPIO port number.
status	: Pointer GPIO port status to be written. After writing into the GPIO port this value is updated with the read back from that port.

Returns

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

6.1.5.13 LPCUSBSIO_API int32_t I2C_Close (LPC_HANDLE *hI2C*)

Closes a I2C port.

Deinitializes I2C port and frees all resources that were used by it.

Parameters

hI2C	: Handle of the I2C 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.

6.1.5.14 LPCUSBSIO_API int32_t I2C_DeviceRead (LPC_HANDLE *hI2C,* 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_TRAN←SFER_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

hI2C	: Handle of the I2C 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 the read data is to be stored
sizeToTransfer	Number of bytes to be read
options	This parameter specifies data transfer options. Check HID_I2C_TRANSFER_OPTIONS_↩ macros.

Returns

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

6.1.5.15 LPCUSBSIO_API int32_t I2C_DeviceWrite (LPC_HANDLE *hI2C,* 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_TRAN↩ SFER_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

hI2C	: Handle of the I2C 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 the data to be written is stored
sizeToTransfer	Number of bytes to be written
options	: This parameter specifies data transfer options. Check HID_I2C_TRANSFER_OPTIONS_ macros.

Returns

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

6.1.5.16 LPCUSBSIO_API int32_t I2C_FastXfer (LPC_HANDLE *hI2C,* I2C_FAST_XFER_T ∗ *xfer*)

Transmit and Receive data in I2C 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

hI2C	: Handle of the I2C port.
xfer	: Pointer to a I2C_FAST_XFER_T structure.

Returns

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

6.1.5.17 LPCUSBSIO_API LPC_HANDLE I2C_Open (LPC_HANDLE *hUsbSio*, I2C_PORTCONFIG_T ∗ *config*, uint8_t *portNum*)

Initialize a I2C port.

This function initializes the I2C port and the communication parameters associated with it.

Parameters

hUsbSio	: Handle of the LPSUSBSIO port.
config	: Pointer to I2C_PORTCONFIG_T structure. Members of I2C_PORTCONFIG_T structure contains the values for I2C master clock and Options
portNum	: I2C port number.

Returns

This function returns a handle to I2C port object on success or NULL on failure. Use LPCUSBSIO_Error() function to get last error.

6.1.5.18 LPCUSBSIO_API int32_t I2C_Reset (LPC_HANDLE *hI2C*)

Reset I2C Controller.

Parameters

hI2C	: A device handle returned from I2C_Open().

Returns

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

6.1.5.19 LPCUSBSIO_API int32_t LPCUSBSIO_Close (LPC_HANDLE *hUsbSio*)

Closes a LPC Serial IO port.

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

Parameters

hUsbSio	: Handle of the LPSUSBSIO 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.

6.1.5.20 LPCUSBSIO_API const wchar_t* LPCUSBSIO_Error (LPC_HANDLE *hUsbSio*)

Get a string describing the last error which occurred.

Parameters

hUsbSio	: A device handle returned from LPCUSBSIO_Open().

Returns

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

6.1.5.21 LPCUSBSIO_API int32_t LPCUSBSIO_GetLastError (void)

Returns the last error seen by the Library.

Returns

This function returns the last error seen by the library. Check LPCUSBSIO_ERR_T for more details on error code.

6.1.5.22 LPCUSBSIO_API uint32_t LPCUSBSIO_GetMaxDataSize (LPC_HANDLE *hUsbSio*)

Returns the max number of bytes supported for I2C/SPI transfers by the Serial IO device.

Parameters

hUsbSio	: A device handle returned from LPCUSBSIO_Open().

Returns

This function returns the max data transfer size on success and negative error code on failure. Check LPCUSBSIO_ERR_T for more details on error code.

6.1.5.23 LPCUSBSIO_API uint32_t LPCUSBSIO_GetNumGPIOPorts (LPC_HANDLE *hUsbSio*)

Returns the number of GPIO ports supported by Serial IO device.

Parameters

hUsbSio	: A device handle returned from LPCUSBSIO_Open().

Returns

This function returns the number of GPIO ports on success and negative error code on failure. Check LPC↵USBSIO_ERR_T for more details on error code.

6.1.5.24 LPCUSBSIO_API uint32_t LPCUSBSIO_GetNumI2CPorts (LPC_HANDLE *hUsbSio*)

Returns the number of I2C ports supported by Serial IO device.

Parameters

hUsbSio	: A device handle returned from LPCUSBSIO_Open().

Returns

This function returns the number of I2C ports on success and negative error code on failure. Check LPCUS↵BSIO_ERR_T for more details on error code.

6.1.5.25 LPCUSBSIO_API int LPCUSBSIO_GetNumPorts (uint32_t *vid*, uint32_t *pid*)

Get number LPCUSBSIO ports available on the LPC controller.

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

Parameters

vid	: Vendor ID.
pid	: Product ID.

Returns

The number of ports available on the LPC controller.

6.1.5.26 LPCUSBSIO_API uint32_t LPCUSBSIO_GetNumSPIPorts (LPC_HANDLE *hUsbSio*)

Returns the number of SPI ports supported by Serial IO device.

Parameters

hUsbSio	: A device handle returned from LPCUSBSIO_Open().

Returns

This function returns the number of SPI ports on success and negative error code on failure. Check LPCU↵SBSIO_ERR_T for more details on error code.

6.1.5.27 LPCUSBSIO_API const char∗ LPCUSBSIO_GetVersion (LPC_HANDLE *hUsbSio*)

Get version string of the LPCUSBSIO library.

Parameters

hUsbSio	: A device handle returned from LPCUSBSIO_Open().

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.

6.1.5.28 LPCUSBSIO_API LPC_HANDLE LPCUSBSIO_Open (uint32_t *index*)

Opens the indexed Serial IO 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 LPCUSBSIO_GetNumPorts – 1).

Parameters

index	: Index of the port to be opened.

Returns

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

6.1.5.29 LPCUSBSIO_API int32_t SPI_Close (LPC_HANDLE *hSPI*)

Closes a SPI port.

Deinitializes SPI port and frees all resources that were used by it.

Parameters

hSPI	: Handle of the SPI 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.

6.1.5.30 LPCUSBSIO_API LPC_HANDLE SPI_Open (LPC_HANDLE *hUsbSio*, HID_SPI_PORTCONFIG_T ∗ *config*, uint8_t *portNum*)

Initialize a SPI port.

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

Parameters

hUsbSio	: Handle of the LPSUSBSIO port.
config	: Pointer to HID_SPI_PORTCONFIG_T structure. Members of HID_SPI_PORTCONFIG_T structure contains the values for SPI data size, clock polarity and phase
portNum	: SPI port number.

Returns

This function returns a handle to SPI port object on success or NULL on failure. Use LPCUSBSIO_Error() function to get last error.

6.1.5.31 LPCUSBSIO_API int32_t SPI_Reset (LPC_HANDLE *hSPI*)

Reset SPI Controller.

Parameters

hSPI	: A device handle returned from SPI_Open().

Returns

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

6.1.5.32 LPCUSBSIO_API int32_t SPI_Transfer (LPC_HANDLE *hSPI*, SPI_XFER_T * *xfer*)

Transmit and Receive data in SPI master mode.

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 *length* be initialized to the memory from which the SPI must pick the data to be transferred to slave and the number of bytes to send respectively, similarly *rxBuff* and *length* must have pointer to memory where data received from slave be stored and the number of data to get from slave respectively. Since SPI is full duplex transmission transmit length and receive length are the same and represented by the member of *xfer*, *length*.

Parameters

hSPI	: Handle of the SPI port.
xfer	: Pointer to a SPI_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.

Chapter 7

Data Structure Documentation

7.1 I2C_FAST_XFER_T Struct Reference

7.1.1 Detailed Description

I2C Fast transfer parameter structure.

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

Data Fields

- uint16_t txSz
- uint16_t rxSz
- uint16_t options
- uint16_t slaveAddr
- const uint8_t * txBuff
- uint8_t * rxBuff

7.1.2 Field Documentation

7.1.2.1 uint16_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 uint16_t rxSz

Number of bytes to received, if 0 only transmission we be carried on

7.1.2.4 uint16_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 uint16_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:

- C:/Source/lpcopen/hosttools/lpcusbsio/inc/lpcusbsio.h

7.2 I2C_PORTCONFIG_T Struct Reference

7.2.1 Detailed Description

I2C 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:

- C:/Source/lpcopen/hosttools/lpcusbsio/inc/lpcusbsio.h

7.3 SPI_XFER_T Struct Reference

7.3.1 Detailed Description

SPI transfer parameter structure.

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

Data Fields

- uint16_t length
- uint8_t options
- uint8_t device

- const uint8_t * txBuff
- uint8_t * rxBuff

7.3.2 Field Documentation

7.3.2.1 uint8_t device

SPI slave device, use LPCUSBSIO_GEN_SPI_DEVICE_NUM macro to derive device number from a GPIO port and pin number

7.3.2.2 uint16_t length

Number of bytes to transmit and receive

7.3.2.3 uint8_t options

Transfer options

7.3.2.4 uint8_t* rxBuff

Pointer memory where bytes received from SPI be stored

7.3.2.5 const uint8_t* txBuff

Pointer to array of bytes to be transmitted

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

- C:/Source/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 C:/Source/lpcopen/hosttools/lpcusbsio/inc/lpcusbsio.h File Reference

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

Data Structures

- struct I2C_PORTCONFIG_T
  I2C Port configuration information.
- struct I2C_FAST_XFER_T
  I2C Fast transfer parameter structure.
- struct SPI_XFER_T
  SPI transfer parameter structure.

Macros

- #define LPCUSBSIO_API
- #define LPCUSBSIO_VID 0x1FC9
- #define LPCUSBSIO_PID 0x0090
- #define LPCUSBSIO_READ_TMO 500

- #define LPCUSBSIO_GEN_SPI_DEVICE_NUM(port, pin) ((((uint8_t)(port) & 0x07) << 5) | ((pin) & 0x1F))

- #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_SYNCHRONIZATION = -3,
  LPCUSBSIO_ERR_MEM_ALLOC = -4, LPCUSBSIO_ERR_MUTEX_CREATE = -5, 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 int LPCUSBSIO_GetNumPorts (uint32_t vid, uint32_t pid)

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

  Opens the indexed Serial IO port.
- LPCUSBSIO_API int32_t LPCUSBSIO_Close (LPC_HANDLE hUsbSio)

  Closes a LPC Serial IO port.
- LPCUSBSIO_API const char * LPCUSBSIO_GetVersion (LPC_HANDLE hUsbSio)

  Get version string of the LPCUSBSIO library.
- LPCUSBSIO_API const wchar_t * LPCUSBSIO_Error (LPC_HANDLE hUsbSio)

  Get a string describing the last error which occurred.
- LPCUSBSIO_API uint32_t LPCUSBSIO_GetNumI2CPorts (LPC_HANDLE hUsbSio)

  Returns the number of I2C ports supported by Serial IO device.
- LPCUSBSIO_API uint32_t LPCUSBSIO_GetNumSPIPorts (LPC_HANDLE hUsbSio)

  Returns the number of SPI ports supported by Serial IO device.
- LPCUSBSIO_API uint32_t LPCUSBSIO_GetNumGPIOPorts (LPC_HANDLE hUsbSio)

  Returns the number of GPIO ports supported by Serial IO device.
- LPCUSBSIO_API uint32_t LPCUSBSIO_GetMaxDataSize (LPC_HANDLE hUsbSio)

  Returns the max number of bytes supported for I2C/SPI transfers by the Serial IO device.

- LPCUSBSIO_API int32_t LPCUSBSIO_GetLastError (void)

    Returns the last error seen by the Library.
- LPCUSBSIO_API LPC_HANDLE I2C_Open (LPC_HANDLE hUsbSio, I2C_PORTCONFIG_T ∗config, uint8_t portNum)

    Initialize a I2C port.
- LPCUSBSIO_API int32_t I2C_Close (LPC_HANDLE hI2C)

    Closes a I2C port.
- LPCUSBSIO_API int32_t I2C_Reset (LPC_HANDLE hI2C)

    Reset I2C Controller.
- LPCUSBSIO_API int32_t I2C_DeviceRead (LPC_HANDLE hI2C, 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 hI2C, 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 hI2C, I2C_FAST_XFER_T ∗xfer)

    Transmit and Receive data in I2C master mode.
- LPCUSBSIO_API LPC_HANDLE SPI_Open (LPC_HANDLE hUsbSio, HID_SPI_PORTCONFIG_T ∗config, uint8_t portNum)

    Initialize a SPI port.
- LPCUSBSIO_API int32_t SPI_Close (LPC_HANDLE hSPI)

    Closes a SPI port.
- LPCUSBSIO_API int32_t SPI_Transfer (LPC_HANDLE hSPI, SPI_XFER_T ∗xfer)

    Transmit and Receive data in SPI master mode.
- LPCUSBSIO_API int32_t SPI_Reset (LPC_HANDLE hSPI)

    Reset SPI Controller.
- LPCUSBSIO_API int32_t GPIO_ReadPort (LPC_HANDLE hUsbSio, uint8_t port, uint32_t ∗status)

    Read a GPIO port.
- LPCUSBSIO_API int32_t GPIO_WritePort (LPC_HANDLE hUsbSio, uint8_t port, uint32_t ∗status)

    Write to a GPIO port.
- LPCUSBSIO_API int32_t GPIO_SetPort (LPC_HANDLE hUsbSio, uint8_t port, uint32_t pins)

    Set GPIO port bits.
- LPCUSBSIO_API int32_t GPIO_ClearPort (LPC_HANDLE hUsbSio, uint8_t port, uint32_t pins)

    Clear GPIO port bits.
- LPCUSBSIO_API int32_t GPIO_GetPortDir (LPC_HANDLE hUsbSio, uint8_t port, uint32_t ∗pPins)

    Read GPIO port direction bits.
- LPCUSBSIO_API int32_t GPIO_SetPortOutDir (LPC_HANDLE hUsbSio, uint8_t port, uint32_t pins)

    Sets GPIO port pins direction to output.
- LPCUSBSIO_API int32_t GPIO_SetPortInDir (LPC_HANDLE hUsbSio, uint8_t port, uint32_t pins)

    Sets GPIO port pins direction to input.
- LPCUSBSIO_API int32_t GPIO_SetPin (LPC_HANDLE hUsbSio, uint8_t port, uint8_t pin)

    Sets a specific GPIO port pin value to high.
- LPCUSBSIO_API int32_t GPIO_GetPin (LPC_HANDLE hUsbSio, uint8_t port, uint8_t pin)

    Reads the state of a specific GPIO port pin.
- LPCUSBSIO_API int32_t GPIO_ClearPin (LPC_HANDLE hUsbSio, uint8_t port, uint8_t pin)

    Clears a specific GPIO port pin.
- LPCUSBSIO_API int32_t GPIO_TogglePin (LPC_HANDLE hUsbSio, uint8_t port, uint8_t pin)

    Toggles the state of a specific GPIO port pin.
- LPCUSBSIO_API int32_t GPIO_ConfigIOPin (LPC_HANDLE hUsbSio, uint8_t port, uint8_t pin, uint32_↩ t mode)

    Configures the IO mode for a specific GPIO port pin.

8.3.1 Macro Definition Documentation

8.3.1.1 #define LPCUSBSIO_API