

*Quick***USB**<sup>®</sup>  
*User Guide*

**Bitwise**<sup>™</sup>  
..... **systems**

Bitwise Systems  
6489 Calle Real, Suite E  
Goleta, CA 93117  
Voice (805) 683-6469  
Fax (805) 683-4833  
Toll Free (800) 224-1633  
Web Site [www.bitwisesys.com](http://www.bitwisesys.com)  
Information [info@bitwisesys.com](mailto:info@bitwisesys.com)  
Technical Support [support@bitwisesys.com](mailto:support@bitwisesys.com)

Version 2.15.1  
January 27, 2011

Copyright © 2012 Bitwise Systems. All rights reserved. This document contains confidential information and trade secrets of Bitwise Systems, and is protected by United States and international copyright laws. Use, disclosure, or reproduction is prohibited without the prior express written permission of Bitwise Systems, except as agreed in the License Agreement. Use, duplication or disclosure by the U.S. Government is subject to restrictions as provided in DFARS 227.7202-1(a) and 227.7202-3(a) (1998), and FAR 12.212, as applicable.

# Table of Contents

|   |           |
|---|-----------|
| <b>Table of Contents</b>                    | <b>3</b>  |
| <b>Introduction</b>                         | <b>6</b>  |
| <b>QuickUSB and the Big USB Picture</b>     | <b>6</b>  |
| USB Nomenclature                            | 6         |
| USB System Architecture                     | 6         |
| I/O Subsystem Latency and Throughput        | 7         |
| USB Interpacket Delay                       | 7         |
| <b>Designing Hardware for QuickUSB</b>      | <b>8</b>  |
| The Cypress EZ USB FX2                      | 8         |
| Power and Ground                            | 8         |
| I/O Levels                                  | 9         |
| Unused I/O Pins                             | 9         |
| Default I/O State                           | 9         |
| High-Speed Parallel Port                    | 9         |
| Overview                                    | 9         |
| IFCLK Sourcing                              | 10        |
| GPIF Master Mode                            | 10        |
| GPIF Master Mode I/O Models                 | 11        |
| Slave FIFO I/O Models                       | 21        |
| General-Purpose I/O Pins                    | 26        |
| RS-232                                      | 26        |
| I2C   | 26        |
| SPI   | 27        |
| FPGA Configuration                          | 30        |
| Altera FPGA Sample Schematics               | 31        |
| Storage                                     | 33        |
| QuickUSB Pin Definitions                    | 34        |
| <b>Using the QuickUSB Library</b>           | <b>49</b> |
| Overview                                    | 49        |
| How to Communicate with a Module            | 49        |
| Data Types                                  | 50        |
| Data Structures                             | 50        |
| QBULKSTREAM                                 | 50        |
| Blocking versus Non-blocking Data Transfers | 51        |
| Deploying your Application                  | 51        |
| Base API                                    | 53        |
| QuickUsbFindModules                         | 53        |
| QuickUsbOpen                                | 53        |
| QuickUsbOpenEx                              | 54        |
| QuickUsbClose                               | 54        |
| QuickUsbGetStringDescriptor                 | 55        |
| QuickUsbSetTimeout                          | 55        |
| QuickUsbGetDriverVersion                    | 56        |
| QuickUsbGetDIIVersion                       | 56        |
| QuickUsbGetFirmwareVersion                  | 57        |
| QuickUsbGetLastError                        | 57        |
| QuickUsbGetLastDriverError                  | 59        |
| QuickUSB Settings                           | 60        |
| QuickUsbReadSetting                         | 66        |

|                                  |     |
|----------------------------------|-----|
| QuickUsbWriteSetting             | 67  |
| QuickUsbReadDefault              | 67  |
| QuickUsbWriteDefault             | 67  |
| FPGA Configuration               | 68  |
| QuickUsbConfigureFpga            | 68  |
| QuickUsbStartFpgaConfiguration   | 69  |
| QuickUsbWriteFpgaData            | 69  |
| QuickUsbIsFpgaConfigured         | 70  |
| High-Speed Parallel Port         | 71  |
| QuickUsbReadCommand              | 71  |
| QuickUsbWriteCommand             | 72  |
| QuickUsbReadData                 | 73  |
| QuickUsbWriteData                | 74  |
| QuickUsbReadDataAsync            | 75  |
| QuickUsbWriteDataAsync           | 76  |
| QuickUsbAsyncWait                | 77  |
| Asynchronous Data API            | 78  |
| QuickUsbAllocateDataBuffer       | 79  |
| QuickUsbFreeDataBuffer           | 79  |
| QuickUsbReadBulkDataAsync        | 80  |
| QuickUsbWriteBulkDataAsync       | 81  |
| QuickUsbBulkWait                 | 82  |
| QuickUsbSetNumAsyncThreads       | 83  |
| QuickUsbGetNumAsyncThreads       | 84  |
| Streaming Data API               | 85  |
| QuickUsbReadBulkDataStartStream  | 86  |
| QuickUsbWriteBulkDataStartStream | 87  |
| QuickUsbBulkDataStopStream       | 88  |
| QuickUsbProcessStream            | 88  |
| QuickUsbPauseStream              | 89  |
| QuickUsbResumeStream             | 89  |
| General-Purpose I/O              | 90  |
| QuickUsbReadPortDir              | 90  |
| QuickUsbWritePortDir             | 90  |
| QuickUsbReadPort                 | 91  |
| QuickUsbWritePort                | 91  |
| RS-232 Port                      | 92  |
| QuickUsbSetRs232BaudRate         | 92  |
| QuickUsbGetNumRS232              | 92  |
| QuickUsbFlushRS232               | 93  |
| QuickUsbReadRS232                | 93  |
| QuickUsbWriteRS232               | 94  |
| I2C-Compatible Port              | 95  |
| QuickUsbReadI2C                  | 95  |
| QuickUsbWriteI2C                 | 95  |
| QuickUsbCachedWriteI2C           | 96  |
| SPI-Compatible Port              | 97  |
| QuickUsbReadSpi                  | 97  |
| QuickUsbWriteSpi                 | 97  |
| QuickUsbWriteReadSpi             | 98  |
| Storage API                      | 99  |
| QuickUsbReadStorage              | 99  |
| QuickUsbWriteStorage             | 99  |
| Programming API                  | 100 |
| QuickUsbWriteFirmware            | 100 |

|                        |     |
|------------------------|-----|
| QuickUsbVerifyFirmware | 101 |
| EPCS API               | 102 |
| QuickUsbIdentifyEpcs   | 102 |
| QuickUsbConfigureEpcs  | 103 |
| QuickUsbVerifyEpcs     | 104 |
| QuickUsbEraseEpcs      | 105 |

## Introduction

Thank you for choosing QuickUSB®. QuickUSB makes your product a well-connected USB device quickly and with a minimum of hassle. Not only is QuickUSB a quick way to get connected to USB, it also offers great Hi-Speed USB 2.0 performance with a wide variety of target interface options.

We hope this guide will answer all of your questions about QuickUSB. However, if you have a question that you cannot find an answer for in this guide, please check the web site at [www.quickusb.com](http://www.quickusb.com) and/or contact the QuickUSB support team at [support@quickusb.com](mailto:support@quickusb.com). Our support team will do our best get you the answer you need.

## QuickUSB and the Big USB Picture

Please take some time to understand the big picture as it relates to USB connections. USB has gained the success it has because it is a well-designed bus specifically designed to easily, and reliably connect peripherals to a computer. Part of that design defines the relationship between your PC and the device. Although with QuickUSB you do not need to learn the inner workings of USB, you do need to understand the basics of USB. We will explain the basics here and if you want to learn more, you can browse [www.usb.org](http://www.usb.org) and learn just about everything that there is to know about USB. Just be careful, because you can easily get distracted from what you really need to accomplish.

## USB Nomenclature

Conveying the big picture requires defining some key words. The first is USB and it is an acronym for Universal Serial Bus. Host means your PC. Device means the QuickUSB module and/or the subsystem you need to connect to the PC. A pipe is a unidirectional virtual connection between a host and the device. Every pipe has a direction attribute of either IN or OUT to indicate the direction of data flow *with respect to the host*. An endpoint is the device side connection of a pipe. When a device is connected to a host, the host automatically senses this and enumerates the bus to find it.

## USB System Architecture

The USB is a master/slave bus. This means that the master initiates all traffic on the bus and the slave can only respond to the master. For the USB, the master is the host computer (your PC) and the slave is the device. This master/slave relationship means that interrupts are not possible on the USB. The USB supports a pseudo-interrupt scheme involving low-latency *interrupt endpoints* so the host can perform low-latency device polling to emulate an interrupt. QuickUSB does not currently support interrupt endpoints.

### I/O Subsystem Latency and Throughput

The period of time between the start of a transfer and the time that it actually occurs is the *transfer latency*. USB transfer latency is the result of several factors. First is the fact that the USB is a frame oriented bus and that all packets must be scheduled to a time base of either 1ms (full speed) or 125us (Hi-Speed). Secondly, the operating system generally assesses a software latency penalty when switching from user mode to kernel mode.

*Throughput* is a measure of data transfer speed and generally expressed in megabytes per second (MB/s). Transfer latency affects throughput because it increases the amount of time a transfer takes regardless of the connection speed.

However, as the data transfer size becomes larger, the transfer latency becomes a smaller fraction of the total transfer time thereby diminishing its effect. When the transfer size is small, the transfer latency will seriously degrade throughput.

Therefore, for applications that require the highest throughput, transfer sizes of at least 64KB are recommended.

Another way to mitigate transfer latency issues is to minimize the amount of time that the USB subsystem waits to schedule USB packets. Issuing multiple requests all at once via asynchronous functions calls allows you to minimize this USB subsystem latency. With asynchronous function calls, transfers are scheduled when the function is called, but the function returns immediately without waiting for the transfer to complete. Using this mechanism, one can concurrently schedule enough USB transfers to assure that the USB bus will not remain idle waiting for data to be transferred to or from your device.

The simplest and most reliable technique for this is to employ multiple transfer buffers and rotate them on an as-needed basis.

### USB Interpacket Delay

In certain circumstances, the USB target interface bandwidth is greater than the USB bus bandwidth. This is the case with the Cypress EZ-USB FX2LP microcontroller. In word-wide mode, the FX2LP can transfer data at up to 96MB/sec. The maximum theoretical throughput of a Hi-Speed USB 2.0 pipe is 54MB/sec. Because the FX2LP can go faster than the USB pipe, the target interface is subject to periods of bus inactivity ('gaps') between data packets. Your system design should take into consideration the strong possibility that there will be gaps between data packets and deal with them accordingly.

# Designing Hardware for QuickUSB

Connecting QuickUSB to your hardware is simple. First, decide on the type of connection you need. If you need to transfer large amounts of data very quickly, then you should use the High-Speed Parallel Port. If you only need to turn some I/O pins on and off, you can just use the general-purpose I/O pins.

## The Cypress EZ USB FX2

QuickUSB is based on the Cypress EZ-USB FX2LP microcontroller. The FX2LP is a powerful, single-chip USB microcontroller that offers an unparalleled capability to interface subsystems to a PC with a high-speed USB 2.0 connection. QuickUSB unleashes the power of the FX2LP to high-level hardware and software designers by abstracting its capabilities as library of dataflow oriented function calls. In addition, chip-specific capabilities are supported via 'Settings' that allow the user to customize the behavior of the FX2LP to suit the target application.

## Power and Ground

QuickUSB supplies unregulated +5V at up to 400 mA max on the VBUS pins to power your circuitry. For modules Rev A1 and above, a FET on the QuickUSB module controls power. Power is off by default and then turned on once the host configures the module. This behavior is required by the USB specification. The QuickUSB module incorporates a current limiting circuit that will shut down the VBUS pins on an over-current condition. In addition the entire module may be powered down by the host or a USB hub if it draws more than the 500 mA allotted by the USB.

If your circuit draws less than 400 mA, you may power it from the unregulated 5V provided on the VBUS pins. However, if your circuit will draw more than 50 mA, you should design your circuit with either a downstream power switch (such as the TPS2051A) or an active high enable logic switched voltage regulator. Connect the enable signal to SW\_PG (pin 76). This signal will enable your circuit's voltage regulator once the VBUS switch is turned on and the output voltage has stabilized to  $\geq 93\%$  of the voltage supplied by the USB. For more information about the QuickUSB VBUS switch, consult the datasheet for the Texas Instruments TPS2150.

If your circuit draws more than 400 mA, do not power it from VBUS. It should be powered with an external power supply and connect the digital ground of your circuit to GND. In this case, you might want to connect an unused I/O pin to the external power supply through a current limiting resistor (10K) so you can read the pin to determine the state of the external power supply.

**NOTE:** If you have an external power supply powering your design and you wish to power the FX2 from that power supply instead of from the USB VBUS power, it is essential to keep the WAKEUP pin pulled high to VBUS and not to your external power supply. Failing to do so causes the FX2 to violate the USB specification and may cause your USB hardware to enumerate inconsistently.



## I/O Levels

The I/O pins on the QuickUSB module have the following characteristics:

| Parameter       | Description           | Conditions   | Min  | Typ | Max    | Units |
|-----------------|-----------------------|--------------|------|-----|--------|-------|
| V <sub>IH</sub> | Input HIGH Voltage    |              | 2    |     | 5.25   | V     |
| V <sub>IL</sub> | Input LOW Voltage     |              | -0.5 |     | 0.8    | V     |
| I <sub>L</sub>  | Input Leakage Current |              |      |     | +/- 10 | uA    |
| V <sub>OH</sub> | Output Voltage HIGH   | IOUT = 4 mA  | 2.4  |     |        |       |
| V <sub>OL</sub> | Output Voltage LOW    | IOUT = -4 mA |      |     | 0.4    |       |
| I <sub>OH</sub> | Output Current HIGH   |              |      |     | 4      | mA    |
| I <sub>OL</sub> | Output Current LOW    |              |      |     | 4      | mA    |

Table 1 - I/O Characteristics  
(From Cypress Document# 38-08032 Rev. \*K)

## Unused I/O Pins

Some I/O pins are reserved for future use and may be activated by a new version of the module or a new firmware release. Therefore, you must not connect unused QuickUSB I/O pins to any signals or power supplies. **DO NOT DIRECTLY GROUND UNUSED QUICKUSB I/O PINS IN YOUR CIRCUIT.** You may use a 10k resistor to tie unused pins to a known level, but do not connect them directly.

## Default I/O State

With the QuickUSB Library (including firmware) v2.11 and above, QuickUSB supports non-volatile default settings. The default settings are programmed and read using the [QuickUsbWriteDefault](#) and [QuickUsbReadDefault](#) functions, respectively. On device power-up, the default settings are read into their corresponding runtime settings, written to and read from with the [QuickUsbWriteSetting](#) and [QuickUsbReadSetting](#) functions. That is, settings affect the runtime operation of the device and defaults are the power-on settings.

## High-Speed Parallel Port

The High-Speed parallel port is a truly outstanding feature of the QuickUSB module. It is the fastest connection on the QuickUSB module and can transfer very large blocks of data to and from your device with ease. It provides both master and slave mode transfers with several types of transfer handshaking models.

## Overview

The high-speed parallel port (HSPP) is an 8- or 16-bit port that is used to transfer high-speed data between the host PC and your device. The WORDWIDE setting controls the data element width. If WORDWIDE = 1, the transfers are 16-bits wide and if 0, 8-bits wide. For more information about WORDWIDE, see the [SETTING\\_WORDWIDE](#) setting in the 'Settings' section of this document. If the HSPP is in 8-bit mode, the upper 8 bits may be used as general purpose I/O.

In addition, there is a 9-bit address bus which increments each time a data element is transferred. The address bus can be set to a fixed address to allow multiple writes to the same address. The address bus can also be disabled and the address bus bits reused as general purpose I/O. See the

## Designing Hardware for QuickUSB

[SETTING\\_DATAADDRESS](#) setting in the 'Settings' section of this manual for more information

There are two modes of HSPP operation, master and slave. The HSPP mode is automatically selected by the QuickUSB firmware, but it may be changed at any time using the [SETTING\\_FIFO\\_CONFIG](#) setting. Typically, your hardware will be configured for either master or slave mode and the requirements of your application will determine which mode is best for you.

### **IFCLK Sourcing**

The IFCLK data clock may optionally be sourced from an internally selectable 30 or 48 MHz clock, or from an external clock. When sourcing externally, it is important that the clock be free running, between 5 and 48 MHz, and present on the IFCLK pin before IFCLKSRC (IFCONFIG[7]) is set to zero to select external IFCLK sourcing. The LSB of the [SETTING\\_FIFO\\_CONFIG](#) setting is the IFCONFIG setting, which is used to control the behavior of the IFCLK pin.

### **GPIF Master Mode**

In GPIF master mode, the QuickUSB module controls all aspects of the HSPP and the host PC initiates all data transfers through the QuickUSB module. This mode is implemented using the GPIF programmable DMA engine built into the FX2. All GPIF master mode HSPP transfers are synchronous with IFCLK and are controlled by CMD\_DATA, REN, WEN, and OE. CMD\_DATA indicates whether the HSPP transfer was initiated by the command or data functions. REN indicates read a transfer and WEN indicates a write transfer. OE indicates a read transfer prior to actually asserting the REN signal so that the peripheral can prepare to execute a read transfer.

#### **Command Transfers**

Command transfers are low-speed transfers that use the data bus (FD) and the address bus (GPIFADR) to read and write data to and from the target hardware. The [QuickUsbReadCommand](#) and [QuickUsbWriteCommand](#) functions are used to perform command transfers. They transfer data one element at a time with the CMD\_DATA line set high ('1'). Command transfers were designed to control registers in a peripheral connected to the HSPP, but they can be used for any type of bi-directional low speed parallel I/O.

#### **Data Transfers**

Data transfers are high-speed block-oriented data transfers that use the data bus (FD) and the address bus (GPIFADR) to read and write data to either a FIFO or a memory in the target hardware. The [QuickUsbReadData](#) and [QuickUsbWriteData](#) functions are used to perform high-speed data transfers. They transfer data in a burst of data blocks with the CMD\_DATA line set low ('0'). A single call from QuickUsbReadData or QuickUsbWriteData will be broken down into a series of data blocks transferred over the HSPP.

## GPIF Master Mode I/O Models

The QuickUSB module interfaces to target hardware by implementing a number of I/O models that provide enough flexibility to interface to a wide variety target hardware. The I/O models are selected by reprogramming the firmware of the QuickUSB module using the **QuickUSB Programmer**. Each firmware load implements a different I/O model. The timing diagrams for each I/O model are given below.

| QuickUSB Signal                             | I/O Model    | Direction                           |
|---|--------------|-------------------------------------|
| FD[15:0] (Word Wide) or FD[7:0] (Byte Wide) | All          | Bidirectional                       |
| IFCLK                                       | All          | OUT or IN<br>(Programmable default) |
| CMD_DATA                                    | All          | OUT                                 |
| REN or nREN                                 | All          | OUT                                 |
| WEN or nWEN                                 | All          | OUT                                 |
| nOE   | FIFO & Block | OUT                                 |
| nEMPTY                                      | FIFO & Block | IN                                  |
| nFULL                                       | FIFO & Block | IN                                  |

Table 2 – GPIF I/O Connections

## GPIF Master Mode Timing Parameters

| Parameter | Description                                 | Internally Sourced IFCLK |     | Externally Sourced IFCLK |      | Unit |
|-----------|---|--------------------------|-----|--------------------------|------|------|
|           |   | Min                      | Max | Min                      | Max  |      |
| tIFCLK    | IFCLK Period                                | 20.83                    |     | 20.83                    | 200  | ns   |
| tSRY      | RDYX to Clock Set-up Time                   | 8.9                      |     | 2.9                      |      | ns   |
| tRYH      | Clock to RDYX Hold Time                     | 0                        |     | 3.7                      |      | ns   |
| tSGD      | GPIF Data to Clock Set-up Time              | 9.2                      |     | 3.2                      |      | ns   |
| tDAH      | GPIF Data Hold Time                         | 0                        |     | 4.5                      |      | ns   |
| tSGA      | Clock to GPIF Address Propagation Delay     |                          | 7.4 |                          | 11.4 | ns   |
| tXGD      | Clock to GPIF Data Output Propagation Delay |                          | 11  |                          | 15   | ns   |
| tXCTL     | Clock to CTLX Output Propagation Delay      |                          | 6.7 |                          | 10.7 | ns   |

Table 3 – GPIF Master Mode Timing Parameters

## Designing Hardware for QuickUSB

### Simple I/O Model

This I/O model performs transfers without regard to the readiness of the target hardware. This model is suitable for hardware that is always ready and that can transfer data as fast as the host can deliver it. This is the fastest QuickUSB I/O model available.

The Simple I/O model is backwards compatible with all default QuickUSB firmware files.

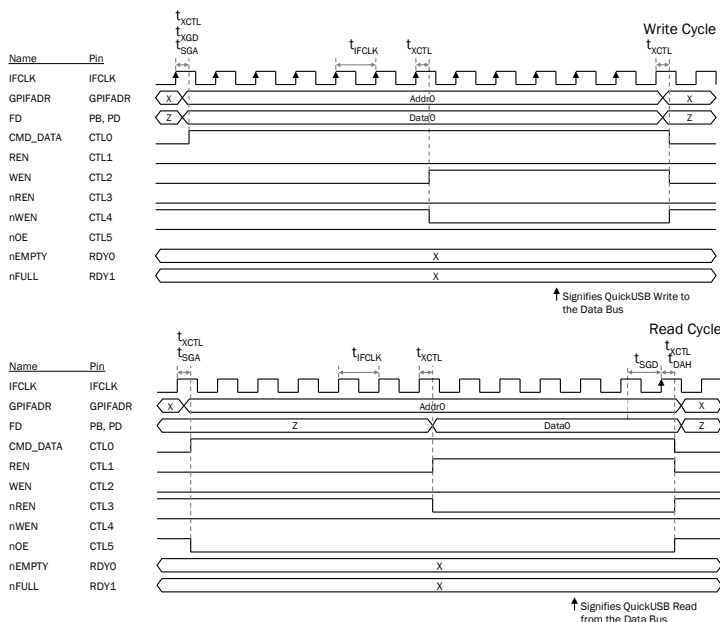
This I/O model is implemented in the QuickUSB firmware file 'quickusb-simple vX.XX.qusb' where X.XX is the firmware version number.

The Simple I/O model is designed to provide the highest possible data rate that the hardware can provide.

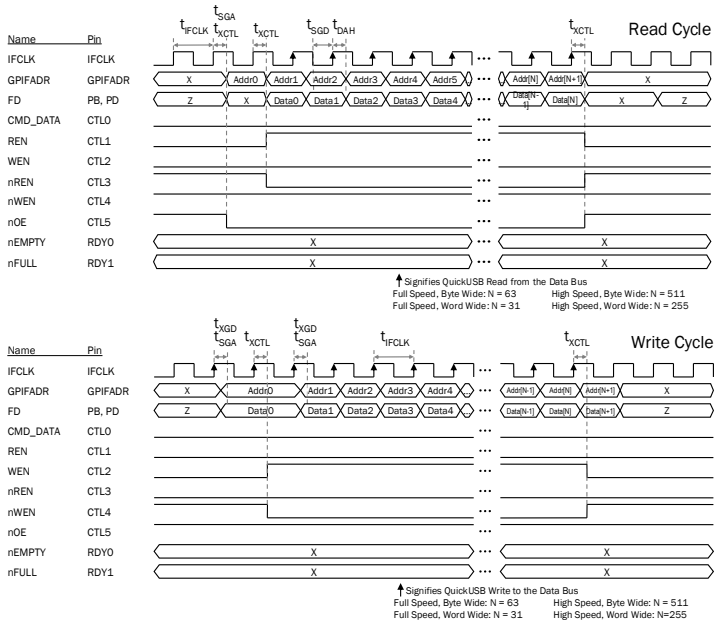
**AS A RESULT, THERE ARE CERTAIN INVALID TRANSFER LENGTHS THAT MAY RESULT IN INCORRECT SYSTEM BEHAVIOR.**

The simplest valid transfer length calculation is to request data transfer lengths in multiples of 512 bytes for Hi-Speed mode or 64 bytes for Full-Speed mode. For applications that cannot use this simplified method, see the Notes section below.

### Command Transfers



## Data Transfers



## Notes

The valid data transfer length for the Simple I/O model can be calculated with the following pseudo code:

```

if (Firmware IO Model is Simple IO or BlockHS) {
    QULONG packetSize = (highspeed) ? 512 : 64;
    QULONG preRead = (wordwide) ? 4 : 2;
    QULONG mod = (Length % packetSize);
    QULONG odd = (Length % 2);
    QBOOL ValidLength = !((wordwide && odd) && ((mod == 0) || (mod
    >= preRead)));
}
    
```

Where:

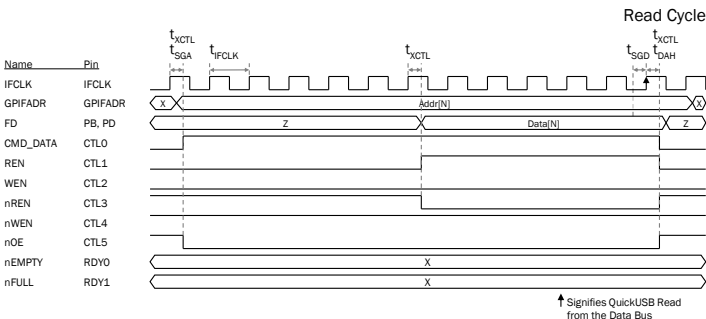
Length = The desired transfer length

ValidLength = A bool indicating if the transfer length is valid

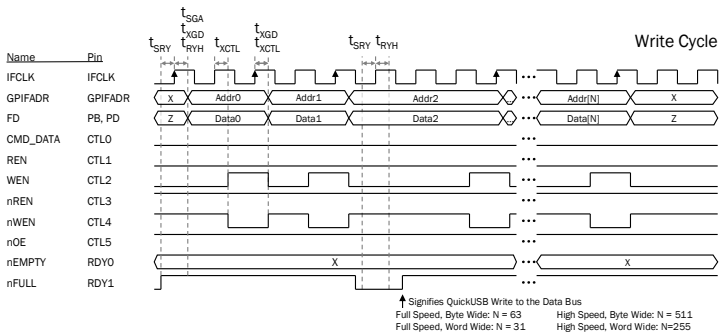
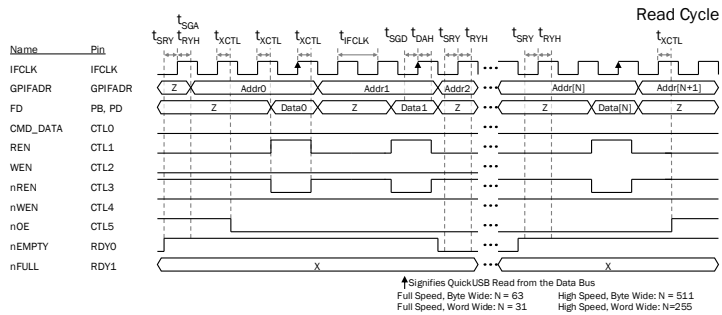
## FIFO Handshake I/O Model

With this I/O model, you simply instantiate a FIFO in the FPGA and connect the data and control lines to the QuickUSB module. Please note the FIFO timing requirements given in the 'Data Transfers' section.

## Command Transfers



## Data Transfers

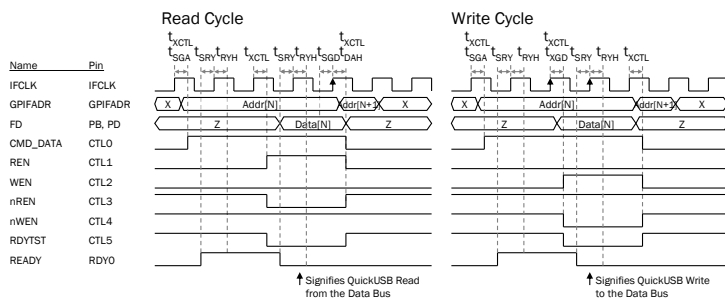


## Full Handshake I/O Model

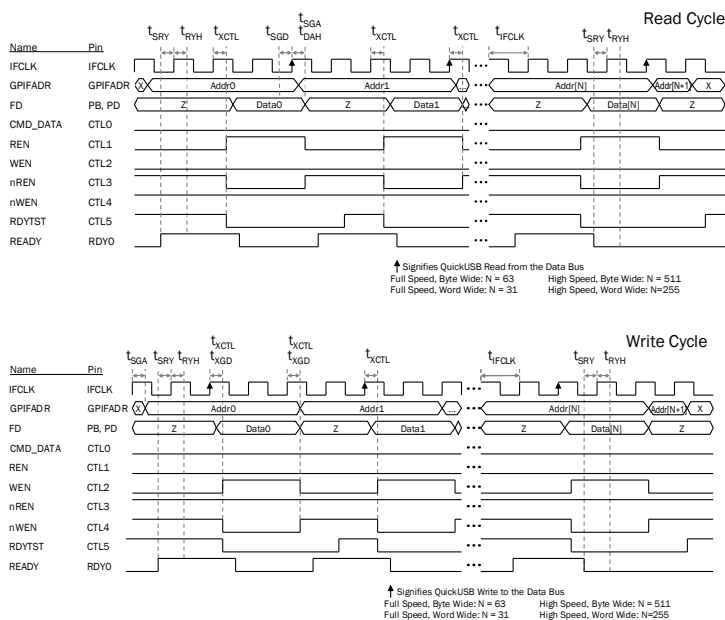
The full handshake I/O model is ideal for connecting a device that has a very slow or variable transfer time. The module checks the state of the READY signal before each state transition and thereby guarantees that the module and target will be properly synchronized at all times.

This I/O model is implemented in the QuickUSB firmware file 'quickusb-fullhs vX.XX.qusb' where X.XX is the firmware version number.

## Command Transfers



## Data Transfers





## Block Handshake I/O Model

Block handshake I/O model is best used for targets that need the benefits of FIFO handshake but always transfer either 64 (Full Speed) or 512 (Hi-Speed) byte blocks for each transaction. This I/O model checks the FIFO flags just once at the beginning of the block and assume that it can transfer the entire block.

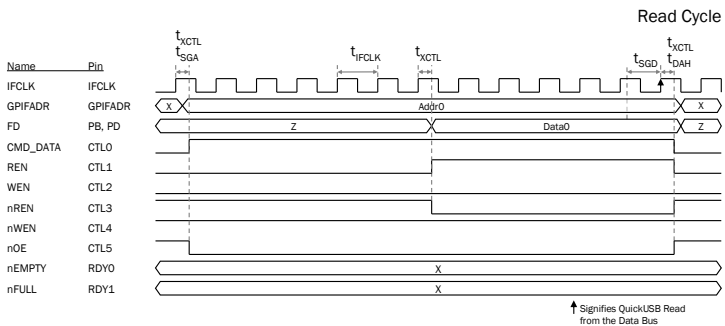
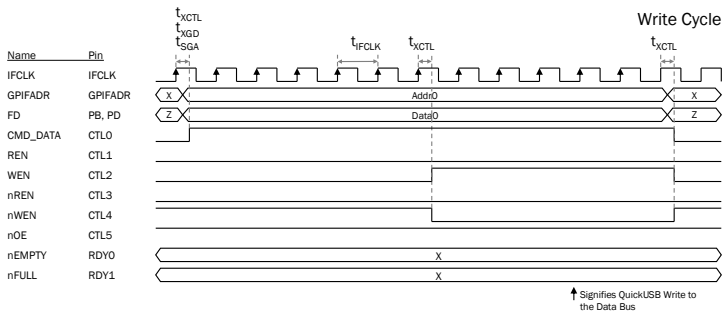
This I/O model is implemented in the QuickUSB firmware file 'quickusb-blockhs vX.XX.qusb' where X.XX is the firmware version number.

The Block I/O model is designed to provide the highest possible data rate possible with hardware handshaking.

**AS A RESULT, THERE ARE CERTAIN INVALID TRANSFER LENGTHS THAT MAY RESULT IN INCORRECT SYSTEM BEHAVIOR.**

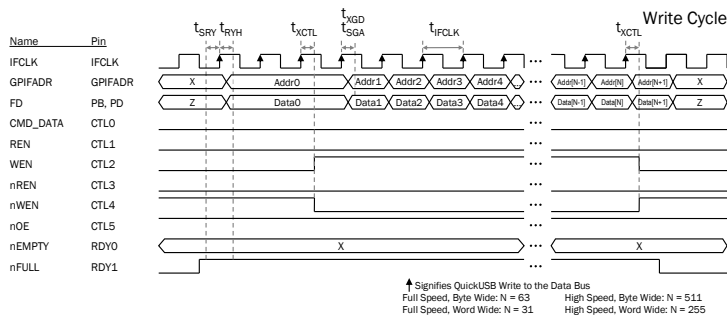
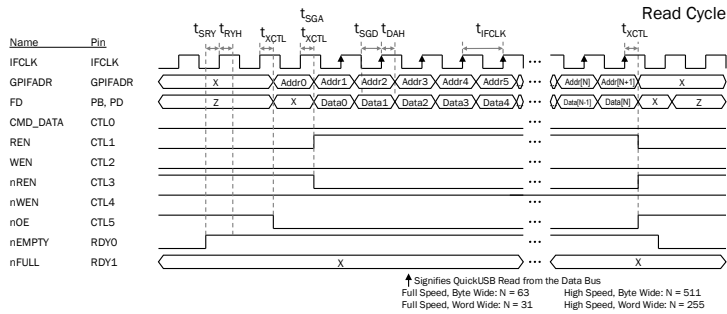
The simplest valid transfer length calculation is to request data transfer lengths in multiples of 512 bytes for Hi-Speed mode or 64 bytes for Full-Speed mode. For applications that cannot use this simplified method, see the Notes section below.

## Command Transfers



# Designing Hardware for QuickUSB

## Data Transfers



## Notes

The valid data transfer length for the Block HS I/O model can be calculated with the following pseudo code:

```
if (Firmware IO Model is Simple IO or BlockHS) {
    ULONG packetSize = (highspeed) ? 512 : 64;
    ULONG preRead = (wordwide) ? 4 : 2;
    ULONG mod = (Length % packetSize);
    ULONG odd = (Length % 2);
    BOOL ValidLength = !((wordwide && odd) && ((mod == 0) || (mod
    >= preRead)));
}
```

Where:

Length = The desired transfer length

ValidLength = A bool indicating if the transfer length is valid

## Pipeline I/O Model

This I/O model implements a one-stage read pipeline. It performs transfers without regard to the readiness of the target hardware. This model is suitable for hardware that is always ready and that can transfer data as fast as the host can deliver it. With this I/O model, the data is transferred one clock cycle after the transfer is made via REN or WEN. The pipeline may be extended for one cycle to up to 255 cycles by writing the desired pipeline delay to the lower byte of the [SETTING\\_SLAVEFIFOFLAGS](#) setting.

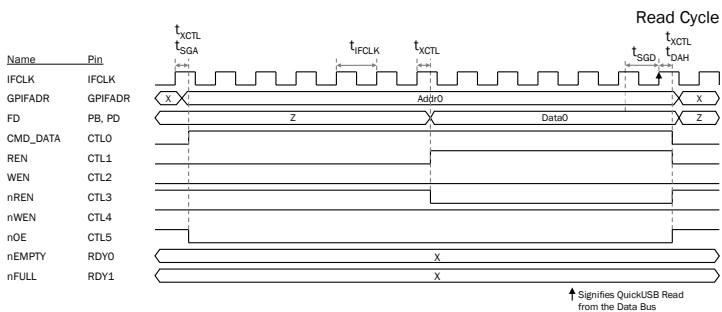
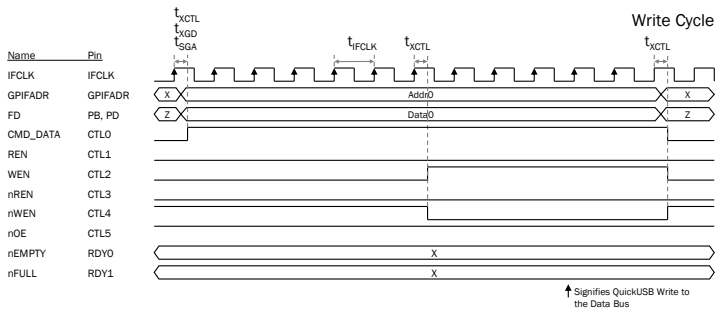
This I/O model is implemented in the QuickUSB firmware file 'quickusb-pipe1 vX.XX.qusb' where X.XX is the firmware version number.

The Pipeline I/O model is designed to provide the highest possible data rate possible with hardware handshaking.

### AS A RESULT, THERE ARE CERTAIN INVALID TRANSFER LENGTHS THAT MAY RESULT IN INCORRECT SYSTEM BEHAVIOR.

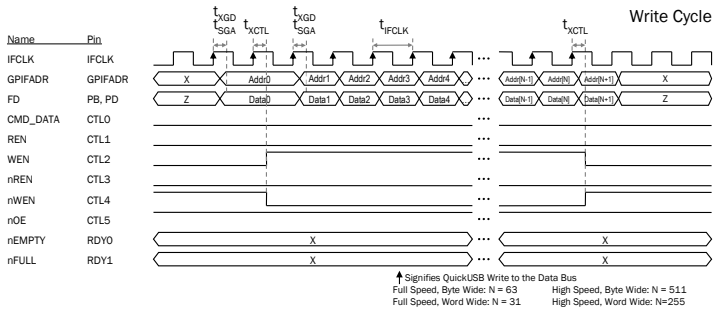
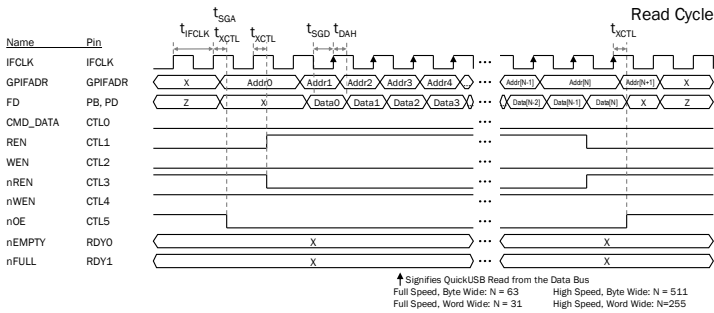
The simplest valid transfer length calculation is to request data transfer lengths in multiples of 512 bytes for Hi-Speed mode or 64 bytes for Full-Speed mode. For applications that cannot use this simplified method, see the Notes section below.

## Command Transfers



# Designing Hardware for QuickUSB

## Data Transfers



## Notes

The valid data transfer length for the Pipeline I/O model can be calculated with the following pseudo code:

```

if (Firmware IO Model is Simple IO or BlockHS) {
    ULONG packetSize = (highspeed) ? 512 : 64;
    ULONG preRead = (wordwide) ? 4 : 2;
    ULONG mod = (Length % packetSize);
    ULONG odd = (Length % 2);
    BOOL ValidLength = !((wordwide && odd) && ((mod == 0) || (mod
    >= preRead)));
}
    
```

Where:

Length = The desired transfer length

ValidLength = A bool indicating if the transfer length is valid

## Slave FIFO I/O Models

The HSPP may also be operated in 'Slave FIFO' mode. In this mode, the GPIF programmable DMA engine is disabled and the QuickUSB FIFOs are controlled directly by external logic signals. GPIF master mode command/data transfers are not applicable in slave FIFO mode since the GPIF programmable DMA engine is disabled. Slave FIFO mode is selected by changing bits 1-0 of the [SETTING\\_FIFO\\_CONFIG](#) setting and may be changed at any time. In slave FIFO mode, data is transferred to and from the QuickUSB FIFOs using the standard [QuickUsbReadData](#) (EP6) & [QuickUsbWriteData](#) (EP2) functions. These endpoints are double-buffered by default. The QuickUSB module can be configured to perform slave FIFO transfers either synchronously or asynchronously by changing Bit 3 of the [SETTING\\_FIFO\\_CONFIG](#) setting. The slave FIFO flags may be queried using the [SETTING\\_SLAVEFIFOFLAGS](#) setting.

The following signals are required for slave mode operation:

| Pin Name | Alternate Name   | Description  | Dir    | Default Config          |
|----------|------------------|--|--------|-------------------------|
| IFCLK    | IFCLK            | Clock for synchronous I/O  | In/Out | Rising edge             |
| FD[15:0] | PD[7:0], PB[7:0] | Bi-directional FIFO data bus   | Bidir  | N/A                     |
| CTL0     | FLAGA            | Programmable level flag (Half-full)                                      | Out    | Indexed mode            |
| CTL1     | FLAGB            | FIFO Full Status Flag  | Out    | Indexed mode            |
| CTL2     | FLAGC            | FIFO Empty Status Flag   | Out    | Indexed Mode            |
| PA2      | nSLOE            | Enables the FD outputs for the selected OUT FIFO                         | In     | Synchronous, Active low |
| RDY0     | nSLRD            | FIFO read enable/clock   | In     | Synchronous, Active low |
| RDY1     | nSLWR            | FIFO write enable/clock  | In     | Synchronous, Active low |
| PA6      | nPKTEND          | Indicates the end of a short IN packet                                   | In     | Synchronous, Active low |
| PA7      | nSLCS            | FIFO Chip Select   | In     | Synchronous, Active low |
| PA5:PA4  | FIFOADR[1:0]     | Selects the active FIFO for FD and flags. 00=EP2, 01=EP4, 10=EP6, 11=EP8 | In     | N/A                     |

Table 4 - Slave FIFO Mode I/O Connections

There are three Slave FIFO I/O Models available with QuickUSB. For the Synchronous and Asynchronous Slave FIFO IO Modes, no special firmware file is required. Simply use the Simple I/O Model firmware file and program the [SETTING\\_FIFO\\_CONFIG](#) setting appropriately to implement the desired I/O model.

## Designing Hardware for QuickUSB

### Synchronous Slave FIFO I/O Model

This I/O Model performs all transfers synchronously with IFCLK. This model is configured by setting the [SETTING\\_FIFO\\_CONFIG](#) bits IFCFG (bits 0-1) to '11' to go to Slave FIFO Mode and ASYNC (bit 3) to '0' to make all transactions synchronous to IFCLK.

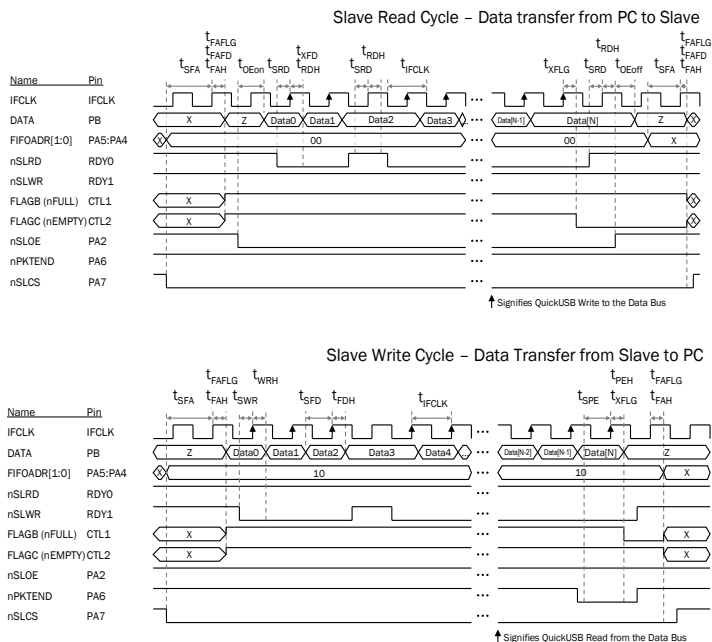


Figure 1 - Synchronous Slave FIFO I/O Timing Diagrams

## Designing Hardware for QuickUSB

| Parameter | Description                                 | Internally Sourced IFCLK |      | Externally Sourced IFCLK |      | Unit |
|-----------|---|--------------------------|------|--------------------------|------|------|
|           |   | Min                      | Max  | Min                      | Max  |      |
| tIFCLK    | IFCLK Period                                | 20.83                    |      | 20.83                    | 200  | ns   |
| tSRD      | SLRD to Clock Set-up Time                   | 18.7                     |      | 12.7                     |      | ns   |
| tRDH      | Clock to SLRD Hold Time                     | 0                        |      | 3.7                      |      | ns   |
| tOEon     | SLOE Turn-on to FIFO Data Valid             |                          | 10.5 |                          | 10.5 | ns   |
| tOEoff    | SLOE Turn-off to FIFO Data Hold             |                          | 10.5 |                          | 10.5 | ns   |
| tXFLG     | Clock to FLAGS Output Propagation Delay     |                          | 9.5  |                          | 13.5 | ns   |
| tXFD      | Clock to FIFO Data Output Propagation Delay |                          | 11   |                          | 15   | ns   |
| tSWR      | SLWR to Clock Set-up Time                   | 18.1                     |      | 12.1                     |      | ns   |
| tWRH      | Clock to SLWR Hold Time                     | 0                        |      | 3.6                      |      | ns   |
| tSFD      | FIFO Data to Clock Set-up Time              | 9.2                      |      | 3.2                      |      | ns   |
| tFDH      | Clock to FIFO Data Hold Time                | 0                        |      | 4.5                      |      | ns   |
| tSFA      | FIFOADR and nSLCS to Clock Set-up Time      | 25                       |      |                          |      | ns   |
| tFAH      | Clock to FIFOADR and nSLCS Hold Time        | 10                       |      |                          |      | ns   |
| tFAFLG    | FIFOADR to FLAGS Output Propagation Delay   |                          | 10.7 |                          |      | ns   |
| tFAFD     | FIFOADR to FIFO Data Bus Propagation Delay  |                          | 14.3 |                          |      | ns   |
| tSPE      | PKTEND to Clock Set-up Time                 | 14.6                     |      | 8.6                      |      | ns   |
| tPEH      | Clock to PKTEND Hold Time                   | 0                        |      | 2.5                      |      | ns   |

*Table 5 – Synchronous Slave FIFO Timing Parameters*

## Designing Hardware for QuickUSB

### Asynchronous Slave FIFO I/O Model

This I/O Model performs all transfers asynchronously using only the rising/falling edge of the nSLRD or nSLWR line to perform the transfer. This model is configured by setting the [SETTING\\_FIFO\\_CONFIG](#) bits IFCFG (bits 0-1) to '11' to go to Slave FIFO Mode and ASYNC (bit 3) to '1' to make all transactions asynchronous.

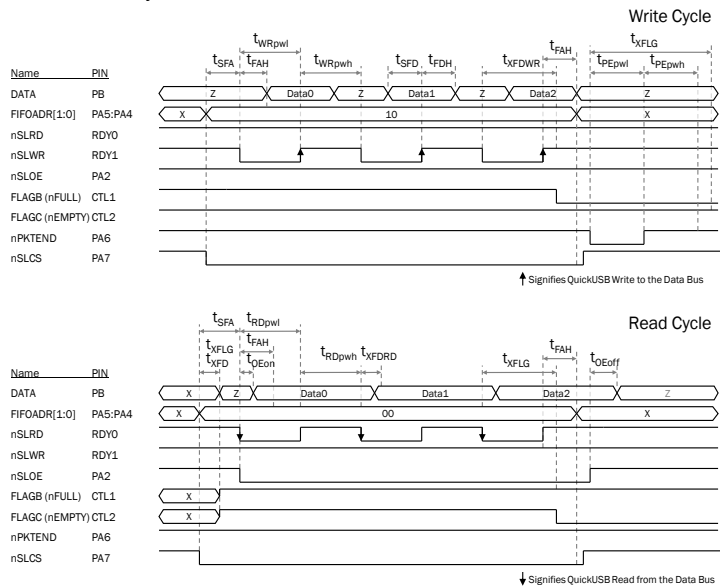


Figure 2 - Asynchronous Slave FIFO I/O Model Timing Diagrams

| Parameter | Description                                   | Min | Max  | Unit |
|-----------|---|-----|------|------|
| tRDpwl    | SLRD Pulse Width Low                          | 50  |      | ns   |
| tRDpwh    | SLRD Pulse Width High                         | 50  |      | ns   |
| tWRpwl    | SLWR Pulse Width Low                          | 50  |      | ns   |
| tWRpwh    | SLWR Pulse Width High                         | 70  |      | ns   |
| tSFD      | SLWR to FIFO DATA Set-up Time                 | 10  |      | ns   |
| tFDH      | FIFO DATA to SLWR Hold Time                   | 10  |      | ns   |
| tXFLG     | SLRD to Flags Output Propagation Delay        |     | 70   | ns   |
| tXFDWR    | SLWR to Flags Output Propagation Delay        |     | 70   | ns   |
| tXFDRD    | SLRD to FIFO Data Output Propagation Delay    |     | 15   | ns   |
| tOEon     | SLOE On to FIFO Data Valid                    |     | 10.5 | ns   |
| tOEoff    | SLOE Off to FIFO Data Hold                    |     | 10.5 | ns   |
| tSFA      | FIFOADR/nSLCS to SLRD/SLWR/PKTEND Set-up Time | 10  |      | ns   |
| tFAH      | SLRD/SLWR/PKTEND to FIFOADR/nSLCS Hold Time   | 10  |      | ns   |
| tPEpwl    | PKTEND Pulse Width Low                        | 50  |      | ns   |
| tPEpwh    | PKTEND Pulse Width High                       | 50  |      | ns   |
| tXFL      | PKTEND to Flags Output Propagation Delay      |     | 115  | ns   |

Table 6 - Asynchronous Slave FIFO Timing Parameters



## Slave245 I/O Model

The Slave245 I/O Model allows QuickUSB to duplicate the functionality of the FTDI245BM I/O waveforms with a speed increase of up to 10X. This I/O model is implemented in the QuickUSB firmware file 'quickusb-245 vX.XX.qusb' where X.XX is the firmware version number. The following QuickUSB I/O connections are used for 245BM targets:

| QuickUSB Signal                 | 245BM Signal |
|---------------------------------|--------------|
| RDY0 (nSLRD) and PA2 (SLOE)     | RD#          |
| RDY1 (nSLWR) and PA5 (FIFOADR1) | WR           |
| CTL2 (FLAGC)                    | RXF#         |
| CTL1 (FLAGB)                    | TXE#         |
| PB[7:0]                         | D[7:0]       |
| PA4                             | GND          |

Figure 3 - QuickUSB to 245BM Connection Table

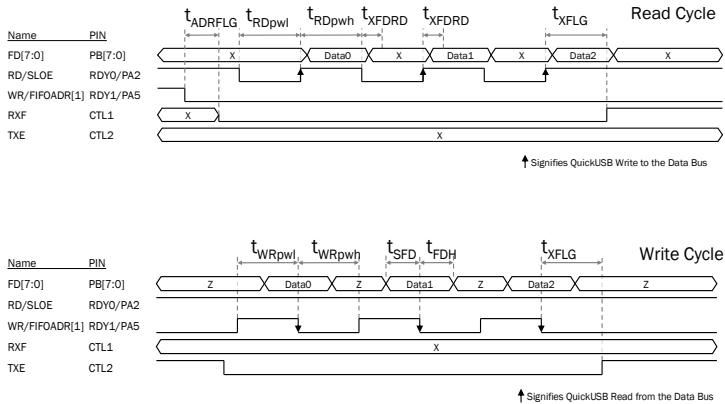


Figure 4 - Slave245 I/O Model Timing Diagrams

| Parameter    | Description                                       | Min | Max  | Unit |
|--------------|---|-----|------|------|
| $t_{RDpwl}$  | SLRD Pulse Width Low                              | 50  |      | ns   |
| $t_{RDpwh}$  | SLRD Pulse Width High                             | 50  |      | ns   |
| $t_{WRpwl}$  | SLWR Pulse Width Low                              | 50  |      | ns   |
| $t_{WRpwh}$  | SLWR Pulse Width High                             | 70  |      | ns   |
| $t_{SFD}$    | SLWR to FIFO DATA Set-up Time                     | 10  |      | ns   |
| $t_{FDH}$    | FIFO DATA to SLWR Hold Time                       | 10  |      | ns   |
| $t_{XFLG}$   | SLRD/SLWR to Flags Output Propagation Delay       |     | 70   | ns   |
| $t_{ADRFLG}$ | SLWR/FIFOADR[1] to Flags Output Propagation Delay |     | 10.7 | ns   |
| $t_{XFDRD}$  | SLRD to FIFO Data Output Propagation Delay        |     | 15   | ns   |

Figure 5 - Slave245 I/O Model Timing Parameters

## General-Purpose I/O Pins

The QuickUSB Module implements General Purpose I/O Pins on Ports A, B, C, D, and E when not using the alternate functions for those ports. Please see the [QuickUSB Pin Definitions](#) section of this user guide for information on each port and their alternate functions.

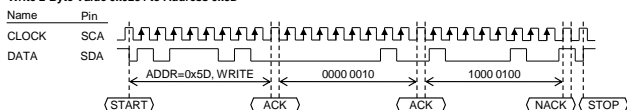
## RS-232

The QuickUSB Module's RS-232 Ports provide standard asynchronous, full-duplex communications. The RS-232 ports operate with no parity, eight data bits, and one stop bit (N81). RS-232 data is received using interrupt-driven receive routines in the module. Both ports operate at the same baud rate.

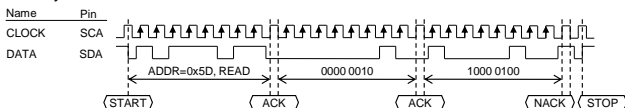
## I2C

The QuickUSB I<sup>2</sup>C-compatible port is a master-only bus controller that can operate in Standard Mode (100 kHz) or Fast Mode (400 kHz) with 7-bit addressing. The bus speed is selectable using Bit 0 of [SETTING\\_I2CTL](#). Addresses 81 (decimal) and 1 are reserved. The R/W bit is automatically inserted, so it does not need to be included in the address. The address is automatically shifted to accommodate the R/W bit.

Write 2-Byte Value 0x0284 to Address 0x5D



Read 2 Bytes from Address 0x5D – Returned Value 0x0284



Read 1 Byte from Address 0x5D with 1 Byte Cached-Write of 0x09 – Returned Value 0x5C

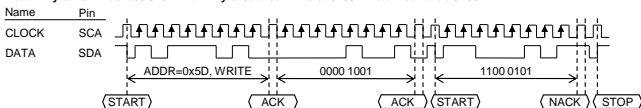


Figure 6 – I<sup>2</sup>C Timing Diagrams

## SPI

The QuickUSB module implements a 'soft' SPI port using pins on Port E or Port A. These routines support up to 10 devices with individual active-low slave select lines for each device. By default, data is shifted in and out MSB to LSB. The bit shift order, clock phase, and clock polarity can all be configured through the [SETTINGS\\_SPICONFIG](#) setting. The SPI bus writes at a little over 600 Kbps and reads at almost 500 Kbps.

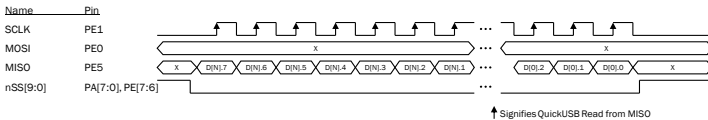
The SPI signals may be mapped to one of four different pin-out configurations depending on the requirements of the design. Mapping A ([SETTINGS\\_SPICONFIG\[3\]='0'](#) and [SETTINGS\\_SPICONFIG\[5\]='0'](#)) and Mapping C ([SETTINGS\\_SPICONFIG\[3\]='0'](#) and [SETTINGS\\_SPICONFIG\[5\]='1'](#)) allow for up to 10 nSS lines, but do not allow for SPI or FPGA configuration on the 56-pin FX2 due to the absence of Port E. Mappings B ([SETTINGS\\_SPICONFIG\[3\]='1'](#) and [SETTINGS\\_SPICONFIG\[5\]='0'](#)) and D ([SETTINGS\\_SPICONFIG\[3\]='1'](#) and [SETTINGS\\_SPICONFIG\[5\]='1'](#)) place the SPI pins on Port A, sacrificing nSS lines, but do allow for SPI on the 56-pin FX2. Additionally, Mapping D is the only pin assignment that allows for both SPI and FPGA configuration on 56-pin FX2's.

| Signal Name | Mapping A | Mapping B     | Mapping C | Mapping D     |
|-------------|-----------|---------------|-----------|---------------|
| SCLK        | PE1       | PA1           | PE1       | PA1           |
| MOSI        | PE0       | PA0           | PE0       | PA0           |
| MISO        | PE5       | PA5           | PE2       | PA2           |
| nSS9        | PA7       | PA7           | PA7       | PA7           |
| nSS8        | PA6       | PA6           | PA6       | PA6           |
| nSS7        | PA5       | Not available | PA5       | PA5           |
| nSS6        | PA4       | PA4           | PA4       | PA4           |
| nSS5        | PA3       | PA3           | PA3       | PA3           |
| nSS4        | PA2       | PA2           | PA2       | Not available |
| nSS3        | PA1       | Not available | PA1       | Not available |
| nSS2        | PA0       | Not available | PA0       | Not available |
| nSS1        | PE7       | PE7           | PE7       | PE7           |
| nSS0        | PE6       | PE6           | PE6       | PE6           |

Table 7: SPI Pin-out Mappings

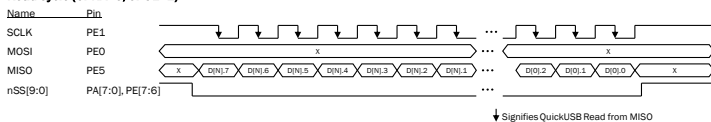
The timing diagrams below show SPI reads and SPI writes with every combination of clock phase (CPHA) and clock polarity (CPOL). The given pin-outs are for Mapping A, but all waveforms are valid for all pin mappings.

Read Cycle (CPHA=0, CPOL=0)

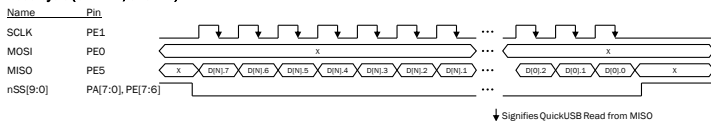


# Designing Hardware for QuickUSB

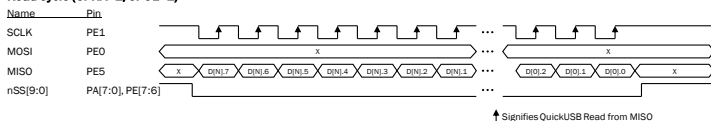
## Read Cycle (CPHA=0, CPOL=1)



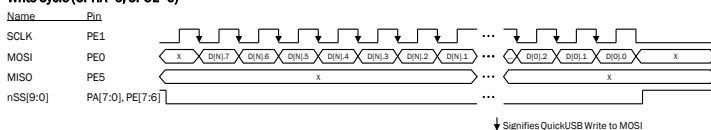
## Read Cycle (CPHA=1, CPOL=0)



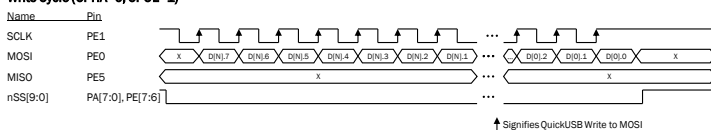
## Read Cycle (CPHA=1, CPOL=1)



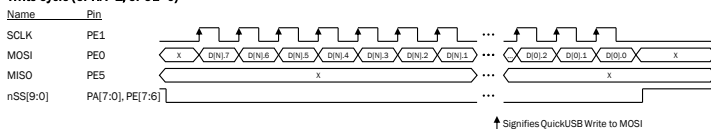
## Write Cycle (CPHA=0, CPOL=0)



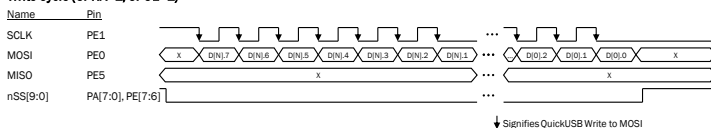
## Write Cycle (CPHA=1, CPOL=1)



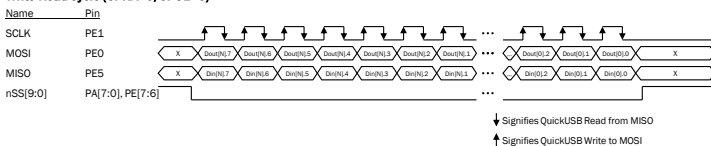
## Write Cycle (CPHA=1, CPOL=0)



## Write Cycle (CPHA=1, CPOL=1)

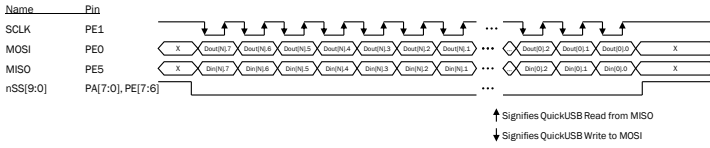


## Write-Read Cycle (CPHA=0, CPOL=0)

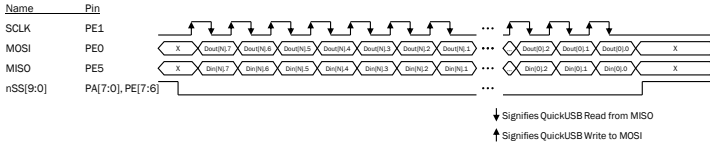


# Designing Hardware for QuickUSB

## Write-Read Cycle (CPHA=0, CPOL=1)



## Write-Read Cycle (CPHA=1, CPOL=0)



## Write-Read Cycle (CPHA=1, CPOL=1)

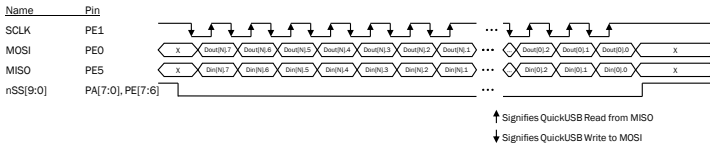


Figure 7 - SPI Timing Diagrams for Pin Mapping A

If you use any SPI signals on Port A (nSS2-9), Port A will convert to the alternate SPI signal functionality for the entire port. Please ensure you do not use Port A for General Purpose I/O if using the slave select nSS2-nSS9 signals.

If you use any of the slave select signals on Port E (nSS0-1), PE6 and PE7 will convert to the alternate slave select functionality for the entire port. Please ensure you do not use PE6 or PE7 for General Purpose I/O if using the nSS0-nSS1 signals.

## FPGA Configuration

The QuickUSB Plug-In module can configure SRAM-based FPGA devices over the USB. The configuration method that QuickUSB uses is based on the [SETTING\\_FPGATYPE](#) setting of the 'Settings' section of this document. Currently, two configuration schemes are supported: Altera Passive Serial and Xilinx Slave Serial.

The QuickUSB module uses 3.3V I/O, so make sure your device can handle 3.3V on the configuration pins. Then, connect the FPGA as shown in Table 8. You must be sure to add pull-up/pull-down resistors as required by the FPGA manufacturer. Refer to the FPGA manufacturer's documentation for the proper configuration connection pin out, signal level and device configuration mode. QuickUSB can transfer an unlimited number of configuration data blocks, so multiple daisy-chained devices can be configured using QuickUSB.

| QuickUSB Signal | Altera PS | Xilinx Slave Serial |
|-----------------|-----------|---------------------|
| DATA0           | DATA0     | DIN                 |
| DCLK            | DCLK      | CCLK                |
| nCONFIG         | nCONFIG   | PROG_B              |
| nSTATUS         | nSTATUS   | INIT_B              |
| CONF_DONE       | CONF_DONE | DONE                |

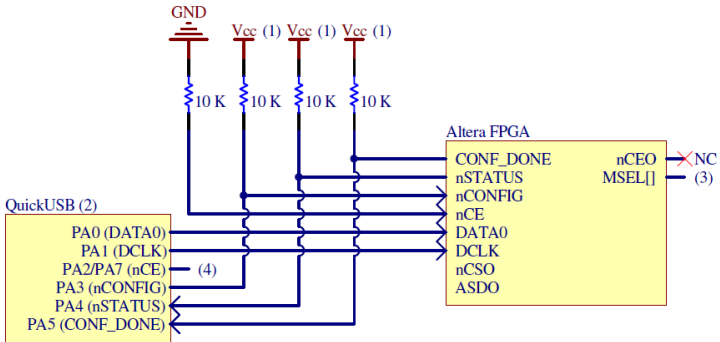
*Table 8 - FPGA Configuration Signals*

The pin-out for the FPGA configuration signals is the same for the SPI pin-out, where DCLK=SCLK and DATA0=MOSI. Note that in order to use perform SPI and FPGA configuration in 56-pin FX2 devices, you must use SPI Pin Mapping D (See Table 7). Additionally, nCE must be placed on Port A Pin 7 by setting SETTING\_SPICONFIG[4]='1', though its use is optional (nCE is currently only used when configuring Altera EPCS devices with the [EPCS API](#)).

## Altera FPGA Sample Schematics

These sample schematics should only be used as a starting point. Always consult your FPGA's documentation before committing to a design.

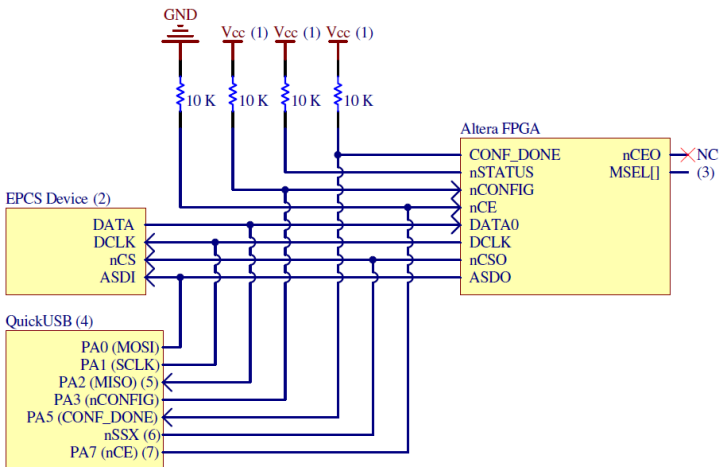
### Port A FPGA Configuration (Passive Serial)



#### NOTES

- (1) For the VCC value, refer to the respective FPGA family handbook Configuration chapter.
- (2) SPI/FPGA ports must be configured for use on Port A by setting SPICONFIG[3]=1. SPI signals DCLK, MOSI, MISO, nSS[9:5], nSS[1:0] remain available (nSS[1:0] not available on 56-Pin FX2).
- (3) Connect the FPGA MSEL[] input pins to select the PS configuration mode.
- (4) nCE on QuickUSB may optionally be connected to nCE on the FPGA. SPICONFIG[4]=0 places nCE on PA[2] and SPICONFIG[4]=1 places nCE on PA[7].

### Port A FPGA Configuration with EPCS Device (Active Serial)

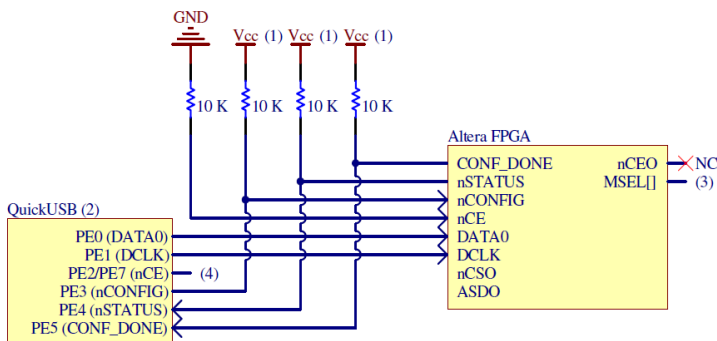


#### NOTES

- (1) For the VCC value, refer to the respective FPGA family handbook Configuration chapter.
- (2) Serial configuration devices cannot be cascaded.
- (3) Connect the FPGA MSEL[] input pins to select the AS configuration mode.
- (4) SPI/FPGA ports must be configured for use on Port A by setting SPICONFIG[3]=1. SPI signals DCLK, MOSI, MISO, nSS8, nSS6, and nSS[1:0] remain available, except for nSSX (nSS[1:0] not available on 56-Pin FX2).
- (5) MISO must be configured for use on PA[2] by setting SPICONFIG[5]=1.
- (6) nSSX may be any one of nSS8 (PA[6]), nSS6 (PA[4]), or nSS[1:0] (PE[7:6]). On 56-Pin FX2, nSSX may only be nSS8 (PA[6]) or nSS6 (PA[4]).
- (7) nCE must be configured for use on PA[7] by setting SPICONFIG[4]=1.

## Designing Hardware for QuickUSB

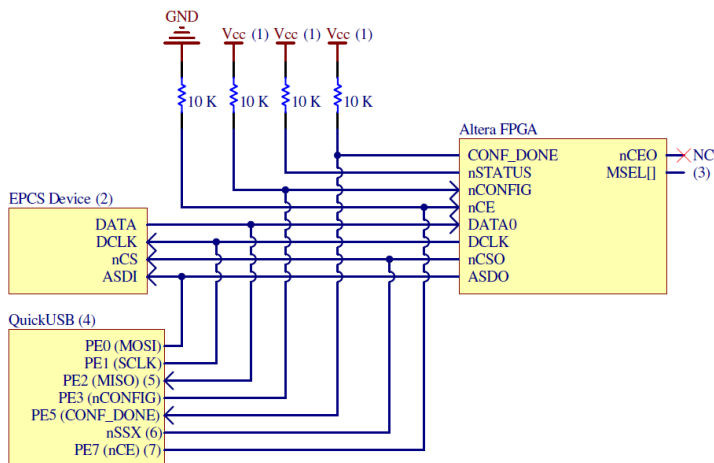
### Port E FPGA Configuration (Passive Serial)



#### NOTES

- (1) For the VCC value, refer to the respective FPGA family handbook Configuration chapter.
- (2) SPI/FPGA ports must be configured for use on Port E by setting SPICONFIG[3]=0. SPI signals DCLK, MOSI, MISO, and nSS[9:0] remain available.
- (3) Connect the FPGA MSEL[] input pins to select the PS configuration mode.
- (4) nCE on QuickUSB may optionally be connected to nCE on the FPGA. SPICONFIG[4]=0 places nCE on PE[2] and SPICONFIG[4]=1 places nCE on PE[7].

### Port E FPGA Configuration with EPCS Device (Active Serial)



#### NOTES

- (1) For the VCC value, refer to the respective FPGA family handbook Configuration chapter.
- (2) Serial configuration devices cannot be cascaded.
- (3) Connect the FPGA MSEL[] input pins to select the AS configuration mode.
- (4) SPI/FPGA ports must be configured for use on Port E by setting SPICONFIG[3]=1. SPI signals DCLK, MOSI, MISO, nSS[9:2], and nSS0 remain available, except for nSSX.
- (5) MISO must be configured for use on PE[2] by setting SPICONFIG[5]=1.
- (6) nSSX may be any one of nSS[9:2] (PA[7:0]) or nSS0 (PE[6]).
- (7) nCE must be configured for use on PE[7] by setting SPICONFIG[4]=1.



### Storage

QuickUSB contains a reserved 2 KB section of non-volatile EEPROM memory for application use. This memory may be read and written using the [QuickUsbReadStorage](#) and [QuickUsbWriteStorage](#) API functions. This memory is intended to store information to uniquely identify the QuickUSB hardware and store configuration information, though the memory may be used for any purpose.

## QuickUSB Pin Definitions

| FX2128 Pin | QUSB2 Pin | Name                              | Dir    | Description        | Function  |
|------------|-----------|-----------------------------------|--------|--------------------|---|
| N/A        | 76        | SW_EN                             | Output | VBUS Switch Enable | SW_EN is high when the QuickUSB Module has successfully enumerated and power can be drawn from VBUS. Used to control the VBUS switch on the QuickUSB Module.  |
| 82         | 3         | PA0 / nSS2 / DATA0 / MOSI / nINT0 | I/O    | Port A, Bit 0      | Multifunction Pin<br><b>PA0</b> (default) is a bi-directional general purpose I/O pin.<br><b>nSS2</b> (When <a href="#">SETTING_SPICONFIG</a> [3] = '0') is the SPI slave select signal for Address 2. Automatically switches functionality when using the SPI commands.<br><i>Note: If using nSS2-nSS9, all of Port A is converted to slave select functionality, so ensure that Port A is not used for GPIO if using nSS2-nSS9.</i><br><b>DATA0</b> (When <a href="#">SETTING_SPICONFIG</a> [3] = '1') is the data output signal for serial FPGA configuration. Automatically switches functionality when using the FPGA configuration commands.<br><b>MOSI</b> (When <a href="#">SETTING_SPICONFIG</a> [3] = '1') is the Master-Out Slave-In data signal for the SPI port. Automatically switches functionality when using the SPI commands.<br><b>nINT0</b> is an active low interrupt input signal. This function is currently unused. |
| 83         | 5         | PA1 / nSS3 / DCLK / SCLK / nINT1  | I/O    | Port A, Bit 1      | Multifunction Pin<br><b>PA1</b> (default) is a bi-directional general purpose I/O pin.<br><b>nSS3</b> (When <a href="#">SETTING_SPICONFIG</a> [3] = '0') is the SPI slave select signal for Address 3. Automatically switches functionality when using the SPI commands. See <i>Note</i> for nSS2.<br><b>DCLK</b> (When <a href="#">SETTING_SPICONFIG</a> [3] = '1') is the clock output signal for serial FPGA configuration. Automatically switches functionality when using the FPGA configuration commands.<br><b>SCLK</b> (When <a href="#">SETTING_SPICONFIG</a> [3] = '1') is the clock output signal for the SPI port. Automatically switches functionality when using the SPI commands.<br><b>nINT1</b> is an active low interrupt input signal. This function is currently unused.  |
| 84         | 7         | PA2 / nSS4 / nSLOE /              | I/O    | Port A, Bit 2      | Multifunction Pin whose function is selected by <a href="#">SETTING_FIFO_CONFIG</a> [1:0]:  |

## Designing Hardware for QuickUSB

| FX2128<br>Pin | QUSB2<br>Pin | Name                                      | Dir | Description   | Function   |
|---------------|--------------|---|-----|---------------|--|
|               |              | MISO / nCE                                |     |               | <p><b>PA2</b> (default) is a bi-directional general purpose I/O pin. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '00'</code> or <code>'10'</code>.</p> <p><b>nSS4</b> is the SPI slave select signal for Address 4. <i>See Note for nSS2.</i></p> <p><b>nSLOE</b> is an input-only active low output enable when operating in slave mode. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '11'</code>.</p> <p><b>MISO</b> (When <a href="#">SETTING_SPICONFIG</a> [3] = '1' and <a href="#">SETTING_SPICONFIG</a> [5] = '1') is the Master-In Slave-Out data signal for the SPI port. Automatically switches functionality when using the SPI commands.</p> <p><b>nCE</b> (When <a href="#">SETTING_SPICONFIG</a> [4:3] = '01') is available for use as a Chip Enable signal for FPGA/SPI commands controlled using the normal GPIO function calls or automatically by the <a href="#">EPCS API</a>.</p> |
| 85            | 9            | PA3 / nSS5 / nCONFIG                      | I/O | Port A, Bit 3 | <p>Multifunction Pin</p> <p><b>PA3</b> (default) is a bi-directional general purpose I/O pin.</p> <p><b>nSS5</b> is the SPI slave select signal for Address 5. <i>See Note for nSS2.</i></p> <p><b>nCONFIG</b> (When <a href="#">SETTING_SPICONFIG</a> [3] = '1') is the Configure/Program output signal for serial FPGA configuration. Automatically switches functionality when using the FPGA configuration commands.</p>   |
| 89            | 11           | PA4 / nSS6 / nSTATUS / FIFOADR0           | I/O | Port A, Bit 4 | <p>Multifunction Pin whose function is selected by <a href="#">SETTING_FIFO_CONFIG</a>[1:0]:</p> <p><b>PA4</b> (default) is a bi-directional general purpose I/O pin. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '00'</code> or <code>'10'</code>.</p> <p><b>nSS6</b> is the SPI slave select signal for Address 6. <i>See Note for nSS2.</i></p> <p><b>nSTATUS</b> (When <a href="#">SETTING_SPICONFIG</a> [3] = '1') is the Status input signal for serial FPGA configuration. Automatically switches functionality when using the FPGA configuration commands.</p> <p><b>FIFOADR0</b> is an input-only address select for slave devices when operating in slave mode. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '11'</code>. <i>See Note for SLOE.</i></p>   |
| 90            | 13           | PA5 / nSS7 / FIFOADR1 / CONF_DO NE / MISO | I/O | Port A, Bit 5 | <p>Multifunction Pin whose function is selected by <a href="#">SETTING_FIFO_CONFIG</a>[1:0]:</p> <p><b>PA5</b> (default) is a bi-directional general purpose I/O pin. Enabled</p>  |

## Designing Hardware for QuickUSB

| FX2128 Pin | QUSB2 Pin | Name                             | Dir | Description   | Function   |
|------------|-----------|----------------------------------|-----|---------------|--|
|            |           |                                  |     |               | <p>when <code>SETTING_FIFO_CONFIG[1:0] = '00'</code> or <code>'10'</code>.</p> <p><b>nSS7</b> is the SPI slave select signal for Address 7. <i>See Note for nSS2.</i></p> <p><b>FIFOADR1</b> is an input-only address select for slave devices when operating in slave mode. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '11'</code>. <i>See Note for SLOE.</i></p> <p><b>CONF_DONE</b> (When <a href="#">SETTING_SPICONFIG</a> [3] = '1') is the Done output signal for serial FPGA configuration. Automatically switches functionality when using the FPGA configuration commands.</p> <p><b>MISO</b> (When <a href="#">SETTING_SPICONFIG</a> [3] = '1' and <a href="#">SETTING_SPICONFIG</a> [5] = '0') is the Master-In Slave-Out data signal for the SPI port. Automatically switches functionality when using the SPI commands.</p>  |
| 91         | 15        | PA6 / nSS8 / nPKTEND             | I/O | Port A, Bit 6 | <p>Multifunction Pin whose function is selected by <a href="#">SETTING_FIFO_CONFIG</a>[1:0]:</p> <p><b>PA6</b> (default) is a bi-directional general purpose I/O pin. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '00'</code> or <code>'10'</code>.</p> <p><b>nSS8</b> is the SPI slave select signal for Address 8. <i>See Note for nSS2.</i></p> <p><b>nPKTEND</b> is an input-only active low signal used to commit the FIFO packet data for slave devices when operating in slave mode. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '11'</code>. <i>See Note for SLOE.</i></p>   |
| 92         | 17        | PA7 / nSS9 / nSLCS / FLAGD / nCE | I/O | Port A, Bit 7 | <p>Multifunction Pin whose function is selected by <a href="#">SETTING_FIFO_CONFIG</a>[1:0]:</p> <p><b>PA7</b> (default) is a bi-directional general purpose I/O pin. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '00'</code> or <code>'10'</code>.</p> <p><b>nSS9</b> is the SPI slave select signal for Address 9. <i>See Note for nSS2.</i></p> <p><b>nSLCS</b> is an input-only active low chip select for slave devices. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '11'</code> and <code>SETTING_PORTACCFG = '0x80'</code>. <i>See Note for SLOE.</i></p> <p><b>FLAGD</b>: EP2 Programmable Flag status. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '11'</code> and <code>SETTING_PORTACCFG = '0x40'</code>. <i>See Note for SLOE.</i></p> <p><b>nCE</b> (When <a href="#">SETTING_SPICONFIG</a> [4:3] = '11') is available for use as a Chip Enable signal for FPGA/SPI commands controlled using the normal GPIO function calls or</p> |

## Designing Hardware for QuickUSB

| FX2128 Pin | QUSB2 Pin | Name      | Dir | Description   | Function  |
|------------|-----------|-----------|-----|---------------|---|
|            |           |           |     |               | automatically by the <a href="#">EPCS API</a> .   |
| 44         | 21        | PB0 / FD0 | I/O | Port B, Bit 0 | Multifunction Pin whose function is selected by <a href="#">SETTING_FIFO_CONFIG[1:0]</a> :<br><b>PB0</b> is a bi-directional general purpose I/O pin. Enabled when SETTING_FIFO_CONFIG[1:0] = '00'.<br><b>FD0</b> (default) is the bi-directional GPIF data bus low byte. Enabled when SETTING_FIFO_CONFIG[1:0] = '10'. |
| 45         | 23        | PB1 / FD1 | I/O | Port B, Bit 1 | Multifunction Pin whose function is selected by <a href="#">SETTING_FIFO_CONFIG[1:0]</a> :<br><b>PB1</b> is a bi-directional general purpose I/O pin. Enabled when SETTING_FIFO_CONFIG[1:0] = '00'.<br><b>FD1</b> (default) is the bi-directional GPIF data bus low byte. Enabled when SETTING_FIFO_CONFIG[1:0] = '10'. |
| 46         | 25        | PB2 / FD2 | I/O | Port B, Bit 2 | Multifunction Pin whose function is selected by <a href="#">SETTING_FIFO_CONFIG[1:0]</a> :<br><b>PB2</b> is a bi-directional general purpose I/O pin. Enabled when SETTING_FIFO_CONFIG[1:0] = '00'.<br><b>FD2</b> (default) is the bi-directional GPIF data bus low byte. Enabled when SETTING_FIFO_CONFIG[1:0] = '10'. |
| 47         | 27        | PB3 / FD3 | I/O | Port B, Bit 3 | Multifunction Pin whose function is selected by <a href="#">SETTING_FIFO_CONFIG[1:0]</a> :<br><b>PB3</b> is a bi-directional general purpose I/O pin. Enabled when SETTING_FIFO_CONFIG[1:0] = '00'.<br><b>FD3</b> (default) is the bi-directional GPIF data bus low byte. Enabled when SETTING_FIFO_CONFIG[1:0] = '10'. |
| 54         | 29        | PB4 / FD4 | I/O | Port B, Bit 4 | Multifunction Pin whose function is selected by <a href="#">SETTING_FIFO_CONFIG[1:0]</a> :<br><b>PB4</b> is a bi-directional general purpose I/O pin. Enabled when SETTING_FIFO_CONFIG[1:0] = '00'.<br><b>FD4</b> (default) is the bi-directional GPIF data bus low byte. Enabled when SETTING_FIFO_CONFIG[1:0] = '10'. |
| 55         | 31        | PB5 / FD5 | I/O | Port B, Bit 5 | Multifunction Pin whose function is selected by <a href="#">SETTING_FIFO_CONFIG[1:0]</a> :<br><b>PB5</b> is a bi-directional general purpose I/O pin. Enabled when SETTING_FIFO_CONFIG[1:0] = '00'.<br><b>FD5</b> (default) is the bi-directional GPIF data bus low byte. Enabled                                       |

## Designing Hardware for QuickUSB

| FX2128 Pin | QUSB2 Pin | Name           | Dir | Description   | Function  |
|------------|-----------|----------------|-----|---------------|---|
|            |           |                |     |               | when SETTING_FIFO_CONFIG[1:0] = '10'.   |
| 56         | 33        | PB6 / FD6      | I/O | Port B, Bit 6 | Multifunction Pin whose function is selected by <a href="#">SETTING_FIFO_CONFIG[1:0]</a> :<br><b>PB6</b> is a bi-directional general purpose I/O pin. Enabled when SETTING_FIFO_CONFIG[1:0] = '00'.<br><b>FD6</b> (default) is the bi-directional GPIF data bus low byte. Enabled when SETTING_FIFO_CONFIG[1:0] = '10'. |
| 57         | 35        | PB7 / FD7      | I/O | Port B, Bit 7 | Multifunction Pin whose function is selected by <a href="#">SETTING_FIFO_CONFIG[1:0]</a> :<br><b>PB7</b> is a bi-directional general purpose I/O pin. Enabled when SETTING_FIFO_CONFIG[1:0] = '00'.<br><b>FD7</b> (default) is the bi-directional GPIF data bus low byte. Enabled when SETTING_FIFO_CONFIG[1:0] = '10'. |
| 72         | 39        | PC0 / GPIFADR0 | I/O | Port C, Bit 0 | Multifunction Pin whose function is selected by <a href="#">SETTING_FIFO_CONFIG[1:0]</a> :<br><b>PC0</b> is a bi-directional general purpose I/O pin. Enabled when SETTING_FIFO_CONFIG[1:0] = '00'.<br><b>GPIFADR0</b> (default) is a GPIF address output pin. Enabled when SETTING_FIFO_CONFIG[1:0] = '10'.            |
| 73         | 41        | PC1 / GPIFADR1 | I/O | Port C, Bit 1 | Multifunction Pin whose function is selected by <a href="#">SETTING_FIFO_CONFIG[1:0]</a> :<br><b>PC1</b> is a bi-directional general purpose I/O pin. Enabled when SETTING_FIFO_CONFIG[1:0] = '00'.<br><b>GPIFADR1</b> (default) is a GPIF address output pin. Enabled when SETTING_FIFO_CONFIG[1:0] = '10'.            |
| 74         | 43        | PC2 / GPIFADR2 | I/O | Port C, Bit 2 | Multifunction Pin whose function is selected by <a href="#">SETTING_FIFO_CONFIG[1:0]</a> :<br><b>PC2</b> is a bi-directional general purpose I/O pin. Enabled when SETTING_FIFO_CONFIG[1:0] = '00'.<br><b>GPIFADR2</b> (default) is a GPIF address output pin. Enabled when SETTING_FIFO_CONFIG[1:0] = '10'.            |
| 75         | 45        | PC3 / GPIFADR3 | I/O | Port C, Bit 3 | Multifunction Pin whose function is selected by <a href="#">SETTING_FIFO_CONFIG[1:0]</a> :<br><b>PC3</b> is a bi-directional general purpose I/O pin. Enabled when SETTING_FIFO_CONFIG[1:0] = '00'.<br><b>GPIFADR3</b> (default) is a GPIF address output pin. Enabled when SETTING_FIFO_CONFIG[1:0] = '10'.            |
| 76         | 47        | PC4 /          | I/O | Port C, Bit 4 | Multifunction Pin whose function is   |

## Designing Hardware for QuickUSB

| FX2128 Pin | QUSB2 Pin | Name           | Dir | Description   | Function  |
|------------|-----------|----------------|-----|---------------|---|
|            |           | GPIFADR4       |     |               | selected by <a href="#">SETTING_FIFO_CONFIG[1:0]</a> :<br><b>PC4</b> is a bi-directional general purpose I/O pin. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '00'</code> .<br><b>GPIFADR4</b> (default) is a GPIF address output pin. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '10'</code> .  |
| 77         | 49        | PC5 / GPIFADR5 | I/O | Port C, Bit 5 | Multifunction Pin whose function is selected by <a href="#">SETTING_FIFO_CONFIG[1:0]</a> :<br><b>PC5</b> is a bi-directional general purpose I/O pin. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '00'</code> .<br><b>GPIFADR5</b> (default) is a GPIF address output pin. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '10'</code> .  |
| 78         | 51        | PC6 / GPIFADR6 | I/O | Port C, Bit 6 | Multifunction Pin whose function is selected by <a href="#">SETTING_FIFO_CONFIG[1:0]</a> :<br><b>PC6</b> is a bi-directional general purpose I/O pin. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '00'</code> .<br><b>GPIFADR6</b> (default) is a GPIF address output pin. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '10'</code> .  |
| 79         | 53        | PC7 / GPIFADR7 | I/O | Port C, Bit 7 | Multifunction Pin whose function is selected by <a href="#">SETTING_FIFO_CONFIG[1:0]</a> :<br><b>PC7</b> is a bi-directional general purpose I/O pin. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '00'</code> .<br><b>GPIFADR7</b> (default) is a GPIF address output pin. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '10'</code> .  |
| 102        | 57        | PD0 / FD8      | I/O | Port D, Bit 0 | Multifunction Pin whose function is selected by <a href="#">SETTING_FIFO_CONFIG[1:0]</a> :<br><b>PD0</b> is a bi-directional general purpose I/O pin. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '00'</code> or when <code>SETTING_WORDWIDE = '0'</code> .<br><b>FD8</b> (default) is the bi-directional GPIF data bus. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '10'</code> and when <code>SETTING_WORDWIDE = '1'</code> . |
| 103        | 59        | PD1 / FD9      | I/O | Port D, Bit 1 | Multifunction Pin whose function is selected by <a href="#">SETTING_FIFO_CONFIG[1:0]</a> :<br><b>PD1</b> is a bi-directional general purpose I/O pin. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '00'</code> or when <code>SETTING_WORDWIDE = '0'</code> .<br><b>FD9</b> (default) is the bi-directional GPIF data bus. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '10'</code> and when <code>SETTING_WORDWIDE = '1'</code> . |

## Designing Hardware for QuickUSB

| FX2128 Pin | QUSB2 Pin | Name       | Dir | Description   | Function   |
|------------|-----------|------------|-----|---------------|--|
| 104        | 61        | PD2 / FD10 | I/O | Port D, Bit 2 | Multifunction Pin whose function is selected by <a href="#">SETTING_FIFO_CONFIG[1:0]</a> :<br><b>PD2</b> is a bi-directional general purpose I/O pin. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '00'</code> or when <code>SETTING_WORDWIDE = '0'</code> .<br><b>FD10</b> (default) is the bi-directional GPIF data bus. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '10'</code> and when <code>SETTING_WORDWIDE = '1'</code> . |
| 105        | 63        | PD3 / FD11 | I/O | Port D, Bit 3 | Multifunction Pin whose function is selected by <a href="#">SETTING_FIFO_CONFIG[1:0]</a> :<br><b>PD3</b> is a bi-directional general purpose I/O pin. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '00'</code> or when <code>SETTING_WORDWIDE = '0'</code> .<br><b>FD11</b> (default) is the bi-directional GPIF data bus. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '10'</code> and when <code>SETTING_WORDWIDE = '1'</code> . |
| 121        | 65        | PD4 / FD12 | I/O | Port D, Bit 4 | Multifunction Pin whose function is selected by <a href="#">SETTING_FIFO_CONFIG[1:0]</a> :<br><b>PD4</b> is a bi-directional general purpose I/O pin. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '00'</code> or when <code>SETTING_WORDWIDE = '0'</code> .<br><b>FD12</b> (default) is the bi-directional GPIF data bus. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '10'</code> and when <code>SETTING_WORDWIDE = '1'</code> . |
| 122        | 67        | PD5 / FD13 | I/O | Port D, Bit 5 | Multifunction Pin whose function is selected by <a href="#">SETTING_FIFO_CONFIG[1:0]</a> :<br><b>PD5</b> is a bi-directional general purpose I/O pin. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '00'</code> or when <code>SETTING_WORDWIDE = '0'</code> .<br><b>FD13</b> is the bi-directional GPIF data bus. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '10'</code> and when <code>SETTING_WORDWIDE = '1'</code> .           |
| 123        | 69        | PD6 / FD14 | I/O | Port D, Bit 6 | Multifunction Pin whose function is selected by <a href="#">SETTING_FIFO_CONFIG[1:0]</a> :<br><b>PD6</b> is a bi-directional general purpose I/O pin. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '00'</code> or when <code>SETTING_WORDWIDE = '0'</code> .<br><b>FD14</b> is the bi-directional GPIF data bus. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '10'</code> and when <code>SETTING_WORDWIDE = '1'</code> .           |



## Designing Hardware for QuickUSB

| FX2128<br>Pin | QUSB2<br>Pin | Name               | Dir | Description   | Function   |
|---------------|--------------|--------------------|-----|---------------|--|
| 124           | 71           | PD7 / FD15         | I/O | Port D, Bit 7 | Multifunction Pin whose function is selected by <a href="#">SETTING_FIFO_CONFIG</a> [1:0]:<br><b>PD7</b> is a bi-directional general purpose I/O pin. Enabled when <a href="#">SETTING_FIFO_CONFIG</a> [1:0] = '00' or when <a href="#">SETTING_WORDWIDE</a> = '0'.<br><b>FD15</b> (default) is the bi-directional GPIF data bus. Enabled when <a href="#">SETTING_FIFO_CONFIG</a> [1:0] = '10' and when <a href="#">SETTING_WORDWIDE</a> = '1'.   |
| 108           | 58           | PE0 / DATA0 / MOSI | I/O | Port E, Bit 0 | Multifunction Pin<br><b>PE0</b> (default) is a bi-directional general purpose I/O pin. Enabled when <a href="#">SETTING_FIFO_CONFIG</a> [1:0] = '00', '10', or '11'.<br><b>DATA0</b> (When <a href="#">SETTING_SPICONFIG</a> [3] = '0') is the data output signal for serial FPGA configuration. Automatically switches functionality when using the FPGA configuration commands.<br><b>MOSI</b> (When <a href="#">SETTING_SPICONFIG</a> [3] = '0') is the Master-Out Slave-In data signal for the SPI port. Automatically switches functionality when using the SPI commands.                                       |
| 109           | 60           | PE1 / DCLK / SCK   | I/O | Port E, Bit 1 | Multifunction Pin<br><b>PE1</b> (default) is a bi-directional general purpose I/O pin. Enabled when <a href="#">SETTING_FIFO_CONFIG</a> [1:0] = '00', '10', or '11'.<br><b>DCLK</b> (When <a href="#">SETTING_SPICONFIG</a> [3] = '0') is the clock output signal for serial FPGA configuration. Automatically switches functionality when using the FPGA configuration commands.<br><b>SCLK</b> (When <a href="#">SETTING_SPICONFIG</a> [3] = '0') is the clock output signal for the SPI port. Automatically switches functionality when using the SPI commands.   |
| 110           | 62           | PE2 / nCE / MISO   | I/O | Port E, Bit 2 | Multifunction Pin<br><b>PE2</b> is a bi-directional general purpose I/O pin. Enabled when <a href="#">SETTING_FIFO_CONFIG</a> [1:0] = '00', '10', or '11'.<br><b>nCE</b> (When <a href="#">SETTING_SPICONFIG</a> [4:3] = '00') is available for use as a Chip Enable signal for FPGA/SPI commands controlled using the normal GPIO function calls or automatically by the <a href="#">EPCS API</a> .<br><b>MISO</b> (When <a href="#">SETTING_SPICONFIG</a> [3] = '0' and <a href="#">SETTING_SPICONFIG</a> [5] = '1') is the Master-In Slave-Out data signal for the SPI port. Automatically switches functionality |

## Designing Hardware for QuickUSB

| FX2128 Pin | QUSB2 Pin | Name                                 | Dir | Description   | Function   |
|------------|-----------|--------------------------------------|-----|---------------|--|
|            |           |                                      |     |               | when using the SPI commands.   |
| 111        | 64        | PE3 /<br>nCONFIG                     | I/O | Port E, Bit 3 | Multifunction Pin<br><b>PE3</b> (default) is a bi-directional general purpose I/O pin. Enabled when <a href="#">SETTING_FIFO_CONFIG</a> [1:0] = '00', '10', or '11'.<br><b>nCONFIG</b> (When <a href="#">SETTING_SPICONFIG</a> [3] = '0') is the Configure/Program output signal for serial FPGA configuration. Automatically switches functionality when using the FPGA configuration commands.   |
| 112        | 66        | PE4 /<br>nSTATUS                     | I/O | Port E, Bit 4 | Multifunction Pin<br><b>PE4</b> (default) is a bi-directional general purpose I/O pin. Enabled when <a href="#">SETTING_FIFO_CONFIG</a> [1:0] = '00', '10', or '11'.<br><b>nSTATUS</b> (When <a href="#">SETTING_SPICONFIG</a> [3] = '0') is the Status input signal for serial FPGA configuration. Automatically switches functionality when using the FPGA configuration commands.   |
| 113        | 68        | PE5 /<br>CONF_DO<br>NE / MISO        | I/O | Port E, Bit 5 | Multifunction Pin<br><b>PE5</b> (default) is a bi-directional general purpose I/O pin. Enabled when <a href="#">SETTING_FIFO_CONFIG</a> [1:0] = '00', '10', or '11'.<br><b>CONF_DONE</b> (When <a href="#">SETTING_SPICONFIG</a> [3] = '0') is the Done output signal for serial FPGA configuration. Automatically switches functionality when using the FPGA configuration commands.<br><b>MISO</b> (When <a href="#">SETTING_SPICONFIG</a> [3] = '0' and <a href="#">SETTING_SPICONFIG</a> [5] = '0') is the Master-In Slave-Out data signal for the SPI port. Automatically switches functionality when using the SPI commands. |
| 114        | 70        | PE6 / nSS0                           | I/O | Port E, Bit 6 | Multifunction Pin<br><b>PE6</b> (default) is a bi-directional general purpose I/O pin. Enabled when <a href="#">SETTING_FIFO_CONFIG</a> [1:0] = '00', '10', or '11'.<br><b>nSS0</b> is the active low slave select signal for Address 0 of the SPI port. Automatically switches functionality when using the SPI commands.<br><i>Note: If using nSS0-nSS1, PE6 and PE7 convert to SPI slave select functionality, so ensure that PE6 and PE7 are not used for GPIO if using SPI nSS0-nSS1.</i>   |
| 115        | 72        | PE7 /<br>GPIFADR8<br>/ nSS1 /<br>nCE | I/O | Port E, Bit 7 | Multifunction Pin<br><b>PE7</b> is a bi-directional general purpose I/O pin. Enabled when <a href="#">SETTING_FIFO_CONFIG</a> [1:0] = '00', or '11'.   |

## Designing Hardware for QuickUSB

| FX2128 Pin | QUSB2 Pin | Name    | Dir    | Description             | Function   |
|------------|-----------|---------|--------|-------------------------|--|
|            |           |         |        |                         | <p><b>GPIFADR8</b> (default) is a GPIF address output pin. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '10'</code>.</p> <p><b>nSS1</b> is the active low slave select signal for Address 1 of the SPI port. Automatically switches functionality when using the SPI commands. See <i>Note for nSS0</i>.</p> <p><b>nCE</b> (When <code>SETTING_SPICONFIG[4:3] = '10'</code>) is available for use as a Chip Enable signal for FPGA/SPI commands controlled using the normal GPIO function calls or automatically by the <a href="#">EPCS API</a>.</p> |
| 36         | 73        | SCL     | Output | Clock for I2C interface | Clock for I2C Interface. <i>Connect to VCC with a 2.2K resistor, even if no I2C peripheral is attached.</i> Connected to VCC with 2.2K resistor on QuickUSB Module.  |
| 37         | 75        | SDA     | OD     | Data for I2C interface  | Data for I2C Interface. <i>Connect to VCC with a 2.2K resistor, even if no I2C peripheral is attached.</i> Connected to VCC with 2.2K resistor on QuickUSB Module.   |
| 29         | 77        | T0      | Input  | Input for Timer0        | Active High Input to Timer0. Not needed for general operation.   |
| 30         | 78        | T1      | Input  | Input for Timer1        | Active High input to Timer1. Not needed for general operation.   |
| 31         | N/A       | T2      | Input  | Input for Timer2        | Active High input to Timer2. Connected to VCC through 2.2k resistor on QuickUSB Module.  |
| 99         | 4         | RESET_B | OD     | FX2 reset, Active low.  | Active Low Reset. Resets the entire chip. Connected to VCC through a 10k resistor on QuickUSB Module.  |
| 1          | 6         | CLKOUT  | Output | 48MHz CPU Clock         | A 12, 24, or 48MHz output clock, phase locked to the 24MHz input clock. CLKOUT settings can be configured through <code>SETTING_CPUCONFIG[4:1]</code> . Default frequency is 48MHz.  |
| 32         | 8         | IFCLK   | Output | 48MHz GPIO Clock        | GPIF Interface Clock. Used to synchronously clock data in or out of the on-chip FIFOs. Also used as a timing reference for all control, data, and address signals on the GPIF. IFCLK settings can be configured through <a href="#">SETTING_FIFO_CONFIG[7:4]</a> .   |
| 28         | 10        | INT4    | Input  | INT4 Interrupt Request  | FX2 internal interrupt request input signal. Active High, Edge Sensitive. This function is currently unused.   |
| N/A        | 12        | RXD_0   | Input  | Serial Port 0 Rx/D      | Serial Port 0 Rx/D   |
| N/A        | 14        | TXD_0   | Output | Serial Port 0 Tx/D      | Serial Port 0 Tx/D   |
| N/A        | 16        | TXD_1   | Output | Serial Port 1 Tx/D      | Serial Port 1 Tx/D   |
| N/A        | 18        | RXD_1   | Input  | Serial Port 1 Rx/D      | Serial Port 1 Rx/D   |

## Designing Hardware for QuickUSB

| FX2128 Pin | QUSB2 Pin | Name                                 | Dir    | Description              | Function   |
|------------|-----------|--------------------------------------|--------|--------------------------|--|
| 69         | 22        | CTL0 /<br>CMD_DATA<br>/ FLAGA        | Output | GPIF Control<br>Output 0 | Multifunction Pin whose function is selected by <a href="#">SETTING_FIFO_CONFIG[1:0]</a> :<br><b>CTL0</b> (default) is a GPIF output signal whose function (CMD_DATA) is waveform specific. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '10'</code> .<br><b>CMD_DATA</b> is the active high Command Enable output signal for the GPIF. Implemented in the Simple, FIFO Handshake, Full Handshake, and Block Handshake I/O Models. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '10'</code> .<br><b>FLAGA</b> is the Slave FIFO Half-Full Flag. Gives half-full status of the FIFO selected by <code>FIFOADR[1:0]</code> in slave mode. Output only, active low. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '11'</code> . |
| 70         | 24        | CTL1 / REN<br>/ FLAGB /<br>nFULL     | Output | GPIF Control<br>Output 1 | Multifunction Pin whose function is selected by <a href="#">SETTING_FIFO_CONFIG[1:0]</a> :<br><b>CTL1</b> (default) is a GPIF output signal whose function (REN) is waveform specific. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '10'</code> .<br><b>REN</b> is the active high Read Enable output signal for the GPIF. Implemented in the Simple, FIFO Handshake, Full Handshake, and Block Handshake I/O Models. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '10'</code> .<br><b>FLAGB / nFULL</b> is the Slave FIFO Full Flag. Gives Full status of the FIFO selected by <code>FIFOADR[1:0]</code> in slave mode. Output only, active low. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '11'</code> .                |
| 71         | 26        | CTL2 /<br>WEN /<br>FLAGC /<br>nEMPTY | Output | GPIF Control<br>Output 2 | Multifunction Pin whose function is selected by <a href="#">SETTING_FIFO_CONFIG[1:0]</a> :<br><b>CTL2</b> (default) is a GPIF output signal whose function (WEN) is waveform specific. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '10'</code> .<br><b>WEN</b> is the active high Write Enable output signal for the GPIF. Implemented in the Simple, FIFO Handshake, Full Handshake, and Block Handshake I/O Models. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '10'</code> .<br><b>FLAGC / nEMPTY</b> is the Slave FIFO Empty Flag. Gives Empty status of the FIFO selected by <code>FIFOADR[1:0]</code> in slave mode. Output only, active low. Enabled when <code>SETTING_FIFO_CONFIG[1:0] = '11'</code> .            |

## Designing Hardware for QuickUSB

| FX2128<br>Pin | QUSB2<br>Pin | Name                   | Dir    | Description               | Function   |
|---------------|--------------|------------------------|--------|---------------------------|--|
| 66            | 28           | CTL3 /<br>nREN         | Output | GPIF Control<br>Output 3  | <b>CTL3</b> is a GPIF output signal whose function (nREN) is waveform specific. Enabled when <a href="#">SETTING_FIFO_CONFIG[1:0] = '10'</a> . <b>nREN</b> is the active low Read Enable output signal for the GPIF. Implemented in the Simple, FIFO Handshake, Full Handshake, and Block Handshake I/O Models. Enabled when <a href="#">SETTING_FIFO_CONFIG[1:0] = '10'</a> .   |
| 67            | 30           | CTL4 /<br>nWEN         | Output | GPIF Control<br>Output 4  | <b>CTL4</b> is a GPIF output signal whose function (nWEN) is waveform specific. Enabled when <a href="#">SETTING_FIFO_CONFIG[1:0] = '10'</a> . <b>nWEN</b> is the active low Write Enable output signal for the GPIF. Implemented in the Simple, FIFO Handshake, Full Handshake, and Block Handshake I/O Models. Enabled when <a href="#">SETTING_FIFO_CONFIG[1:0] = '10'</a> .  |
| 98            | 32           | CTL5 / nOE<br>/ RDYTST | Output | GPIF Control<br>Output 5  | <b>CTL5</b> is a GPIF output signal whose function (nOE or RDYTST) is waveform specific. Enabled when <a href="#">SETTING_FIFO_CONFIG[1:0] = '10'</a> . <b>nOE</b> is the active low Output Enable output signal for the GPIF. Implemented in the Simple, FIFO Handshake, and Block Handshake I/O Models. Enabled when <a href="#">SETTING_FIFO_CONFIG[1:0] = '10'</a> . <b>RDYTST</b> is the Ready Test output signal for the Full Handshake I/O Model. RDYTST outputs the correct handshake waveform for the READY line, so it can be connected to READY to test the Full Handshake functionality. Implemented in the Full Handshake I/O Model. Enabled when <a href="#">SETTING_FIFO_CONFIG[1:0] = '10'</a> . |
| 51            | 34           | RXD0                   | Input  | Serial Port 0<br>TTL Rx/D | <b>RXD0</b> is the receive signal for the 8051 UART0. Active High, Input only. Do not use if U1 is populated on the QuickUSB Module.   |
| 50            | 36           | TXD0                   | Output | Serial Port 0<br>TTL Rx/D | <b>TXD0</b> is the transmit signal for the 8051 UART0. Active High, Output only. Do not use if U1 is populated on the QuickUSB Module.   |
| 53            | 52           | RXD1                   | Input  | Serial Port 1<br>TTL Rx/D | <b>RXD1</b> is the receive signal for the 8051 UART1. Active High, Input only. Do not use if U1 is populated on the QuickUSB Module.   |
| 52            | 54           | TXD1                   | Output | Serial Port 1<br>TTL Rx/D | <b>TXD1</b> is the transmit signal for the 8051 UART1. Active High, Output only. Do not use if U1 is populated on the QuickUSB Module.   |
| 4             | 40           | RDY0 /<br>nEMPTY /     | Input  | GPIF input<br>signal 0    | Multifunction Pin whose function is selected by  |

## Designing Hardware for QuickUSB

| FX2128 Pin | QUSB2 Pin | Name                 | Dir   | Description         | Function  |
|------------|-----------|----------------------|-------|---------------------|---|
|            |           | READY / nSLRD        |       |                     | <p><a href="#">SETTING_FIFO_CONFIG[1:0]</a>: <b>RDY0</b> (default) is a GPIF input signal whose function (nEMPTY or READY) is waveform specific. Enabled when <a href="#">SETTING_FIFO_CONFIG[1:0]</a> = '10'. <b>nEMPTY</b> is an active low input that checks the status of the EMPTY flag of a connected FIFO. Implemented in the FIFO Handshake and Block Handshake I/O Models. Enabled when <a href="#">SETTING_FIFO_CONFIG[1:0]</a> = '10'.</p> <p><b>READY</b> is an input that checks the status of the READY flag of a slave device. Implemented in the Full Handshake I/O Model. Enabled when <a href="#">SETTING_FIFO_CONFIG[1:0]</a> = '10'. <b>nSLRD</b> is the Slave Read Strobe. Input only, active low. Enabled when <a href="#">SETTING_FIFO_CONFIG[1:0]</a> = '11'.</p> |
| 5          | 42        | RDY1 / nFULL / nSLWR | Input | GPIF input signal 1 | <p>Multifunction Pin whose function is selected by <a href="#">SETTING_FIFO_CONFIG[1:0]</a>: <b>RDY1</b> (default) is a GPIF input whose function (nFULL) is waveform specific. Enabled when <a href="#">SETTING_FIFO_CONFIG[1:0]</a> = '10'. <b>nFULL</b> is an active low input that checks the status of the FULL flag of a connected FIFO. Implemented in the FIFO Handshake and Block Handshake waveforms. Enabled when <a href="#">SETTING_FIFO_CONFIG[1:0]</a> = '10'. <b>nSLWR</b> is the Slave Write Strobe. Input only, active low. Enabled when <a href="#">SETTING_FIFO_CONFIG[1:0]</a> = '11'.</p>   |
| 6          | 44        | RDY2                 | Input | GPIF input signal 2 | <b>RDY2</b> is a GPIF input signal. Enabled when <a href="#">SETTING_FIFO_CONFIG[1:0]</a> = '10'. Not currently used by QuickUSB.   |
| 7          | 46        | RDY3                 | Input | GPIF input signal 3 | <b>RDY3</b> is a GPIF input signal. Enabled when <a href="#">SETTING_FIFO_CONFIG[1:0]</a> = '10'. Not currently used by QuickUSB.   |
| 8          | 48        | RDY4                 | Input | GPIF input signal 4 | <b>RDY4</b> is a GPIF input signal. Enabled when <a href="#">SETTING_FIFO_CONFIG[1:0]</a> = '10'. Not currently used by QuickUSB.   |
| 9          | 50        | RDY5                 | Input | GPIF input signal 5 | <b>RDY5</b> is a GPIF input signal. Enabled when <a href="#">SETTING_FIFO_CONFIG[1:0]</a> = '10'. Not currently used by QuickUSB.   |
| 101        | 74        | WAKEUP_B             | Input | USB Wakeup          | USB Wakeup. Asserting pin brings FX2 out of suspend mode. Active Low. Connected to VCC through a 10k resistor on QuickUSB Module.   |
| 19         | N/A       | DMINUS               | I/O   | USB D- Signal       | Connect to the USB D- Signal  |

## Designing Hardware for QuickUSB

| FX2128<br>Pin | QUSB2<br>Pin | Name    | Dir    | Description                        | Function  |
|---------------|--------------|---------|--------|------------------------------------|---|
| 18            | N/A          | DPLUS   | I/O    | USB D+<br>Signal                   | Connect to the USB D+ Signal  |
| 94            | N/A          | A0      | Output | 8051 Address<br>Bus                | 8051 Address Bus. Driven at all times. Reflects internal address when 8051 is addressing RAM. Not connected on QuickUSB Module.   |
| 95            | N/A          | A1      | Output |                                    |   |
| 96            | N/A          | A2      | Output |                                    |   |
| 97            | N/A          | A3      | Output |                                    |   |
| 117           | N/A          | A4      | Output |                                    |   |
| 118           | N/A          | A5      | Output |                                    |   |
| 119           | N/A          | A6      | Output |                                    |   |
| 120           | N/A          | A7      | Output |                                    |   |
| 126           | N/A          | A8      | Output |                                    |   |
| 127           | N/A          | A9      | Output |                                    |   |
| 128           | N/A          | A10     | Output |                                    |   |
| 21            | N/A          | A11     | Output |                                    |   |
| 22            | N/A          | A12     | Output |                                    |   |
| 23            | N/A          | A13     | Output |                                    |   |
| 24            | N/A          | A14     | Output |                                    |   |
| 25            | N/A          | A15     | Output |                                    |   |
| 59            | N/A          | D0      | I/O    | 8051 Data<br>Bus                   | 8051 Data Bus. Bi-directional bus used for external 8051 program and data memory. High Impedance when inactive. Active only for external bus accesses. Driven low in suspend. Not connected on QuickUSB Module. |
| 60            | N/A          | D1      | I/O    |                                    |   |
| 61            | N/A          | D2      | I/O    |                                    |   |
| 62            | N/A          | D3      | I/O    |                                    |   |
| 63            | N/A          | D4      | I/O    |                                    |   |
| 86            | N/A          | D5      | I/O    |                                    |   |
| 87            | N/A          | D6      | I/O    |                                    |   |
| 88            | N/A          | D7      | I/O    |                                    |   |
| 39            | N/A          | PSEN#   | Output | Program<br>Store Enable            | Indicates an 8051 code fetch from external memory. Active low.  |
| 34            | N/A          | BKPT    | Output | Breakpoint                         | Used as SW_EN on the QuickUSB module to control the onboard VBUS switch.  |
| 35            | N/A          | EA      | Input  | External<br>Access                 | Determines where the 8051 fetches code from RAM. If EA=0, 8051 fetches from internal Ram. If EA=1, 8051 fetches from external RAM. Tied to GND through 10k Resistor on QuickUSB Module.                         |
| 12            | N/A          | XTALIN  | Input  | Crystal Input                      | Connect to 24MHz parallel resonant, fundamental mode crystal and connect the parallel load capacitor to GND.  |
| 11            | N/A          | XTALOUT | Output | Crystal<br>Output                  | Connect to 24MHz parallel resonant, fundamental mode crystal and connect the parallel load capacitor to GND.  |
| 42            | N/A          | CS#     | Output | External<br>Memory Chip<br>Select  | External Memory Chip Select. Active Low. Not connected on QuickUSB Module   |
| 41            | N/A          | WR#     | Output | External<br>Memory Write<br>Strobe | External Memory Write Strobe. Active Low. Not connected on QuickUSB Module  |
| 40            | N/A          | RD#     | Output | External<br>Memory Read<br>Strobe  | External Memory Read Strobe. Active Low. Not connected on QuickUSB Module   |
| 38            | N/A          | OE#     | Output | External<br>Memory<br>Output       | External Memory Output Enable. Active Low. Not connected on QuickUSB Module   |

## Designing Hardware for QuickUSB

| FX2128 Pin | QUSB2 Pin | Name     | Dir   | Description     | Function   |
|------------|-----------|----------|-------|-----------------|--|
|            |           |          |       | Enable          |  |
| 33         | N/A       | Reserved | Input | Reserved        | Reserved. Connect to Ground  |
| 2          | N/A       | VCC      | Power | VCC             | <b>VCC</b> Connect VCC to 3.3V Power Source  |
| 26         | N/A       | VCC      | Power | VCC             | <b>VCC</b> Connect VCC to 3.3V Power Source  |
| 43         | N/A       | VCC      | Power | VCC             | <b>VCC</b> Connect VCC to 3.3V Power Source  |
| 48         | N/A       | VCC      | Power | VCC             | <b>VCC</b> Connect VCC to 3.3V Power Source  |
| 64         | N/A       | VCC      | Power | VCC             | <b>VCC</b> Connect VCC to 3.3V Power Source  |
| 68         | N/A       | VCC      | Power | VCC             | <b>VCC</b> Connect VCC to 3.3V Power Source  |
| 81         | N/A       | VCC      | Power | VCC             | <b>VCC</b> Connect VCC to 3.3V Power Source  |
| 100        | N/A       | VCC      | Power | VCC             | <b>VCC</b> Connect VCC to 3.3V Power Source  |
| 107        | N/A       | VCC      | Power | VCC             | <b>VCC</b> Connect VCC to 3.3V Power Source  |
| 10         | N/A       | AVCC     | N/A   | Analog VCC      | <b>ANALOG VCC</b> connect to 3.3V power source. Provides power to the analog side of the FX2 |
| 17         | N/A       | AVCC     | N/A   | Analog VCC      | <b>ANALOG VCC</b> connect to 3.3V power source. Provides power to the analog side of the FX2 |
| 13         | N/A       | AGND     | N/A   | Analog GND      | <b>ANALOG GND</b> Connect to ground with the shortest lead possible                          |
| 20         | N/A       | AGND     | N/A   | Analog GND      | <b>ANALOG GND</b> Connect to ground with the shortest lead possible                          |
| N/A        | 2         | +5V      | N/A   | Unregulated +5V | <b>Unregulated +5V</b> from the USB bus (250mA total)  |
| N/A        | 20        | +5V      | N/A   | Unregulated +5V | <b>Unregulated +5V</b> from the USB bus (250mA total)  |
| N/A        | 38        | +5V      | N/A   | Unregulated +5V | <b>Unregulated +5V</b> from the USB bus (250mA total)  |
| N/A        | 56        | +5V      | N/A   | Unregulated +5V | <b>Unregulated +5V</b> from the USB bus (250mA total)  |
| N/A        | 80        | +5V      | N/A   | Unregulated +5V | <b>Unregulated +5V</b> from the USB bus (250mA total)  |
| 3          | 1         | GND      | N/A   | Ground          | <b>Ground</b>  |
| 27         | 19        | GND      | N/A   | Ground          | <b>Ground</b>  |
| 49         | 37        | GND      | N/A   | Ground          | <b>Ground</b>  |
| 58         | 55        | GND      | N/A   | Ground          | <b>Ground</b>  |
| 65         | 79        | GND      | N/A   | Ground          | <b>Ground</b>  |
| 80         | N/A       | GND      | N/A   | Ground          | <b>Ground</b>  |
| 93         | N/A       | GND      | N/A   | Ground          | <b>Ground</b>  |
| 116        | N/A       | GND      | N/A   | Ground          | <b>Ground</b>  |
| 125        | N/A       | GND      | N/A   | Ground          | <b>Ground</b>  |
| 14         | N/A       | NC       | N/A   | No Connect      | This pin must be left open.  |
| 15         | N/A       | NC       | N/A   | No Connect      | This pin must be left open.  |
| 16         | N/A       | NC       | N/A   | No Connect      | This pin must be left open.  |

Table 9 - QuickUSB Pin Definitions



# Using the QuickUSB Library

## Overview

The QuickUSB® Library API gives programmers a cohesive programming interface to the QuickUSB family of products. The same QuickUSB Library API works for all QuickUSB products on all platforms, so you write your software once and all your QuickUSB based products will work on any supported platform.

The QuickUSB library offers support for many programming languages. The QuickUSB Library includes support for the following programming languages (as well as many others not listed here):

- C/C++
- Microsoft .NET Languages (C#, VB.NET, etc.)
- Python
- Microsoft Visual Basic 6
- LabView
- Matlab
- Delphi

This API documentation is mostly language independent, though there may be some differences depending on the requirements on the language and the interface to the QuickUSB API. To find the exact parameter function definition for a specific language, please consult the QuickUSB Library include files for the programming language in question.

There are three basic methods of interfacing with the QuickUSB Library: Using a function-based API, a class-based API, or using a QuickUSB component.

The function-based API makes calls to the QuickUSB Library directly from your programming language. Your application will (typically) include a header file that declares all of the QuickUSB function and link to an appropriate library. Then, from your code you call the appropriate functions to implement the behavior you need.

The class-based API makes the QuickUSB Library level calls from an object-oriented interface. Your code instantiates one or more QuickUSB class objects and calls the methods to implement the required behavior. Like the function-based method, you (typically) include a header file and link to the appropriate library.

The component-based approach displays available modules and lets the user interact with the component GUI to make that selection, and then the app calls component methods to implement the required behavior.

## How to Communicate with a Module

The general procedure to use a QuickUSB module is given below:

1. Call [QuickUsbFindModules](#) to get a list of available modules. The list will be returned as a NULL-delimited string that must be parsed.
2. Parse the list of available modules and provide a means to select the desired module.
3. Then, for each data transaction (send or receive):
  - a. Call [QuickUsbOpen](#) and pass in the device name. A new device id is returned on success.
  - b. Call the data transfer functions needed by your application.
  - c. Call [QuickUsbClose](#) to close the handle to the USB module.

### Data Types

This document uses the following parameter data type naming convention:

|          |   |
|----------|---|
| QBYTE    | an 8-bit unsigned character                 |
| PQBYTE   | a pointer to a QBYTE or an array of BYTES   |
| QCHAR    | an 8-bit signed character                   |
| PQCHAR   | a pointer to a QCHAR or an array of CHARs   |
| QBOOL    | an 8-bit unsigned character                 |
| PQBOOL   | a pointer to a QBOOL or an array of BOOLs   |
| QWORD    | a 16-bit unsigned integer                   |
| PQWORD   | a pointer to a QWORD or an array of WORDs   |
| QULONG   | a 32-bit unsigned integer                   |
| PQULONG  | a pointer to a QULONG or an array of ULONGs |
| QHANDLE  | a QLONG                                     |
| PQHANDLE | a pointer to a QHANDLE                      |

### Data Structures

#### QBULKSTREAM

The following structure fields are intended to be publicly accessible and should only ever be read and never written. All other fields are used internally and should not be accessed.

##### **Handle**

The handle to the QuickUSB device.

##### **Buffer**

A pointer to the data buffer.

##### **CompletionRoutine**

A pointer to the completion routine. The completion routine returns nothing and takes a single pointer to a QBULKSTREAM object as a parameter (i.e. the C function prototype is of the form 'QVOIDRETURN func(PQBULKSTREAM BulkStream)'). On Windows, it is especially important to declare the return value as QVOIDRETURN, defined as 'void \_\_stdcall', or the program may crash unpredictably. On other platforms, QVOIDRETURN is simply defined as 'void'.

##### **Tag**

A pointer to user specific data.

##### **BytesRequested**

The number of bytes requested.

##### **BytesTransferred**

The number of bytes successfully transferred.

##### **Error**

The QuickUSB error code that applies to this transaction. See [QuickUsbGetLastError](#) for descriptions of error codes this value may represent.

##### **StreamID**

The ID of the stream (as returned when the stream was started from [QuickUsbReadBulkDataStartStream](#) or [QuickUsbWriteBulkDataStartStream](#)).

##### **RequestID**

The ID of the request. This ID is unique for each request, begins at zero, and counts upwards for each issued request.

## Blocking versus Non-blocking Data Transfers

The QuickUSB Library allows a user to make both blocking ([QuickUsbReadData](#) and [QuickUsbWriteData](#)) and non-blocking ([QuickUsbReadBulkDataAsync](#), [QuickUsbWriteBulkDataAsync](#)) data transfer calls. When called, the blocking functions will initiate a data transfer, and will return from the function once that data transfer has completed. A non-blocking function, when called, will initiate a data transfer and return to the program without waiting for the data transfer to complete. The user can then call [QuickUsbBulkWait](#) to get the status of the data transfer or wait for the transfer to complete. For a user concerned with transferring data and processing it as quickly as possible, they will want to implement the asynchronous non-blocking function calls. This will allow the user to process already transferred data while collecting more data.

The QuickUSB Library also includes streaming data functions to further ease development and maximize data throughput in applications where asynchronous data reads/writes are repeatedly issued, services, and re-issued. In these applications, such as video imaging, use the [QuickUsbReadBulkDataStartStream](#), [QuickUsbWriteBulkDataStartStream](#), and [QuickUsbBulkDataStopStream](#) API functions.

Currently, all non-blocking data transfer functions (i.e. the asynchronous and streaming APIs) are only supported on Windows.

## Deploying your Application

NOTE: This section is specific to Windows machines.

Once you have developed application using QuickUSB and wish to deploy it to end-users, your software will have a dependency on the QuickUSB driver and DLLs. All you need to do on your customers target machine is to install the QuickUSB Driver Package. The QuickUSB Driver Package installs the device driver, the 32/64-bit DLLs, and the 32/64-bit Microsoft Visual C Runtime v9.0 (since the QuickUSB DLLs have a dependency on the MSVCRT90). To perform the install run the "setup.exe" executable. The setup program will automatically handle registering the driver with Windows and copying the DLLs to the proper location for both 32- and 64-bit versions of Windows. The driver setup package also creates an entry in the programs list of the control panel with an entry of the form "Windows Driver Package – Bitwise Systems QuickUSB (01/12/2012 2.15.1.0)" so that an end-user may easily uninstall the QuickUSB driver from their computer just like any other software.

If your software uses the QuickUSB .NET Assembly, you will also have to install the QuickUSB.NET.dll file into the Global Assembly Cache (GAC) of your customers PCs. This is most easily accomplished by using an installer/setup application that registers the assembly with the GAC. If you wish to not install the QuickUSB .NET Assembly in the GAC, you may simply install a copy of the assembly file alongside your application executable.

For developers creating a windows installer/setup application we provide a standard Windows Merge Module which performs the same driver install tasks as running "setup.exe", with the exception of installing the MSVCRT90 DLL dependencies which must be performed manually as an additional step. If you are creating a Windows installer/setup application to deploy your software, simply add the Merge Module found in the "QuickUsb\Drivers\Windows\Merge Modules" directory to your project and the QuickUSB driver and DLLs will automatically install with your software. To install the MSVCRT90 DLL dependencies you will need to execute the "vcredist\_x86.exe" file found in the

## Using the QuickUSB Library

driver's folder. Additionally, on x64 systems you will need to execute the "vcredist\_x64.exe" file.

If you wish to manually install the QuickUSB DLLs, it is important to keep in mind that there is a 32-bit QuickUSB.dll file and a 64-bit QuickUSB.dll file, as well as a dependency on the MSVCRT90. On 32-bit systems, you must place the 32-bit QuickUSB.dll file in the \Windows\System32 system directory. On 64-bit systems, you must place the 32-bit QuickUsb.dll file in the \Windows\SysWOW64 system directory and place the 64-bit QuickUsb.dll in the \Windows\System32 directory.

## Base API

General purpose functions to manage the operation of the QuickUSB module and the Library.

### QuickUsbFindModules

#### Purpose

Build a list of all QuickUSB modules connected to the host.

#### Parameters

**nameList:** A PQCHAR that points to a buffer in which to store a of QuickUSB module names found by the library. Device names are of the form 'QUSB-XXX' where XXX is the device address (0-126) in decimal. 'nameList' must be large enough to contain all the device names + 1 character.

**length:** A QLONG containing the length of the nameList buffer in CHARs.

#### Returns

A QLONG that is either non-zero on success or zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#). If successful, nameList will contain a NULL ('\0' or CHR(0)) delimited list of QuickUSB module names found by the library. The final entry is designated by two consecutive NULL characters. For example, after executing this function with one module connected, nameList will contain "QUSB-0\0\0". If there are two devices plugged in, nameList will contain "QUSB-0\0QUSB-1\0\0".

#### Notes

- This routine will return a list of all connected QuickUSB devices, whether they are currently opened and being used by the QuickUSB API or not.

### QuickUsbOpen

#### Purpose

Open a QuickUSB device for use by the library

#### Parameters

**hDevice:** A QHANDLE that points to a QHANDLE in which to place the new device ID. If successful, hDevice will contain the new QHANDLE.

**devName:** A PQCHAR that points to a null-terminated QCHAR array containing the name of the device. Device names are of the form 'QUSB-XXX' where XXX is the device address (0-126) in decimal ("QUSB-0", for example). The device name should be parsed from the response from [QuickUsbFindModules](#).

#### Returns

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

#### Notes

- This function will open both closed and already opened devices. Use [QuickUsbOpenEx](#) to open a module only if it is closed (i.e. not already opened).
- On Mac, a device may only be opened if it is not already opened in another process. Attempting to do so will return an error.

### QuickUsbOpenEx

#### **Purpose**

Open a QuickUSB device for use by the library, with additional options.

#### **Parameters**

**hDevice:** A PQHANDLE that points to a QHANDLE in which to place the new device ID. If successful, hDevice will contain the new QHANDLE.

**devName:** A PQCHAR that points to a null-terminated QCHAR array containing the name of the device. Device names are of the form 'QUSB-XXX' where XXX is the device address (0-126) in decimal ("QUSB-0", for example). The device name should be parsed from the response from [QuickUsbFindModules](#).

**flags:** A QWORD containing additional flags. Pass QUICKUSB\_OPEN\_NORMAL (0x0000) to perform the same operation as [QuickUsbOpen](#). Pass QUICKUSB\_OPEN\_IF\_CLOSED (0x0001) to only open a module if it is not already opened by the QuickUSB API.

#### **Returns**

A QLONG that is either non-zero on success or zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

#### **Notes**

- If QUICKUSB\_OPEN\_IF\_CLOSED is specified and the indicated module is already opened, this function will fail with error code QUICKUSB\_ERROR\_ALREADY\_OPENED.
- On Mac, a device may only be opened if it is not already opened in another process. Attempting to do so will return an error.

### QuickUsbClose

#### **Purpose**

Close a QuickUSB device.

#### **Parameters**

**hDevice:** A QHANDLE that was returned from a call to [QuickUsbOpen](#).

#### **Returns**

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

#### **Notes**

- This function will only close devices that are opened and will return an error for devices that are not open, have already been closed, or are no longer connected to the host computer.

## QuickUsbGetStringDescriptor

### Purpose

Returns the string descriptor for the selected QuickUSB module.

### Parameters

hDevice: A QHANDLE that was returned from a call to [QuickUsbOpen](#).

index: The QBYTE string descriptor index given in the following table:

buffer: A PQCHAR that points to a buffer in which to place the string descriptor. The buffer should be at least 128 bytes long.

length: A QWORD that contains the length of the buffer in bytes.

| Symbol          | Index | Description   |
|-----------------|-------|---|
| QUICKUSB_MAKE   | 1     | Manufacturer String                                       |
| QUICKUSB_MODEL  | 2     | Device ID string<br>(including firmware type and version) |
| QUICKUSB_SERIAL | 3     | Serial Number   |

*Table 10 - QuickUSB String Descriptors*

### Returns

A QLONG that is either non-zero on success or zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

### Notes

- This function writes a NULL terminated string to 'buffer', if successful.

## QuickUsbSetTimeout

### Purpose

Set the timeout for QuickUSB requests.

### Parameters

hDevice: A QHANDLE that was returned from a call to [QuickUsbOpen](#).

timeout: A QLONG that specifies the new timeout value in milliseconds.

### Returns

A QLONG that is either non-zero on success or zero (0) on failure.

### Notes

- The default timeout for requests is 1000 ms.

### QuickUsbGetDriverVersion

#### **Purpose**

Determine the version of the QuickUSB driver.

#### **Parameters**

- |        |   |
|--------|---|
| major: | A PWORD that points to a variable in which to place the major version number. |
| minor: | A PWORD that points to a variable in which to place the minor version number. |
| build: | A PWORD that points to a variable in which to place the build number.         |

#### **Returns**

A QLONG that is either non-zero on success or zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

#### **Notes**

- On Mac systems, this function returns the same value as QuickUsbGetDllVersion, as the driver and library version are the same.

### QuickUsbGetDllVersion

#### **Purpose**

Determine the version of the QuickUSB Library file.

#### **Parameters**

- |        |  |
|--------|--|
| major: | A PWORD pointing to variable in which to place the major version number. |
| minor: | A PWORD pointing to variable in which to place the minor version number. |
| build: | A PWORD pointing to variable in which to place the build number.         |

#### **Returns**

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

#### **Notes**

- On Windows systems, this function returns the .dll version.
- On Linux systems, this function returns the .a/.so version.
- On Mac systems, this function returns the .DyLib version. This function returns the same value as QuickUsbGetDriverVersion, as the driver and library version are the same.



## QuickUsbGetFirmwareVersion

### Purpose

Determine the version of the QuickUSB Firmware is currently in the QuickUSB Module.

### Parameters

major: A PWORD pointing to variable in which to place the major version number.  
 minor: A PWORD pointing to variable in which to place the minor version number.  
 build: A PWORD pointing to variable in which to place the build number.

### Returns

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

### Notes

- None

## QuickUsbGetLastError

### Purpose

Return extended error information for the last API call that returned a value of FALSE (0).

### Parameters

error: A PQLONG pointing to a variable in which to place the error code.

### Returns

A QLONG containing one '1', indicating success. This function never fails.

### Notes

| Error Code - Error Name                   | Error Description  |
|---|--|
| 0 - QUICKUSB_ERROR_NO_ERROR               | No error.  |
| 1 - QUICKUSB_ERROR_OUT_OF_MEMORY          | Out of memory. Please free some memory and try again.  |
| 2 - QUICKUSB_ERROR_CANNOT_OPEN_MODULE     | Cannot open module.  |
| 3 - QUICKUSB_ERROR_CANNOT_FIND_DEVICE     | Cannot find specified QuickUSB module. Please check the specified module name.   |
| 4 - QUICKUSB_ERROR_IOCTL_FAILED           | IOCTL failed. Caused when an error is returned by the QuickUSB driver to the QuickUSB Library.   |
| 5 - QUICKUSB_ERROR_INVALID_PARAMETER      | Cannot read or write data length of zero or greater than 16 megabytes. Cannot read or write I2C or SPI data length of zero or greater than 64 bytes. Try breaking the transfer up into smaller blocks. |
| 6 - QUICKUSB_ERROR_TIMEOUT                | Timeout occurred while attempting to read or write data.   |
| 7 - QUICKUSB_ERROR_FUNCTION_NOT_SUPPORTED | This function is not supported by the version of the QuickUSB driver you are using. Please update your driver to the latest version.   |
| 8 - QUICKUSB_ERROR_I2C_BUS_ERROR          | I2C bus error.   |

## Using the QuickUSB Library

|   |  |
|---|--|
| 9 - QUICKUSB_ERROR_I2C_NO_ACK               | No ACK received from I2C device.   |
| 10 - QUICKUSB_ERROR_I2C_SLAVE_WAIT          | An I2C slave device is holding SCL low.  |
| 11 - QUICKUSB_ERROR_I2C_TIMEOUT             | Timeout on the I2C bus.  |
| 12 - QUICKUSB_ERROR_UNKNOWN_DRIVER_TYPE     | Unknown driver.  |
| 13 - QUICKUSB_ERROR_ALREADY_OPENED          | The QuickUSB device has already been opened.   |
| 14 - QUICKUSB_ERROR_CANNOT_CLOSE_MODULE     | The QuickUSB device has already been closed or was never opened.                                       |
| 15 - QUICKUSB_ERROR_FPGA_INIT_FAILED        | The FPGA could not initialize. Ensure power is applied to the FPGA.                                    |
| 16 - QUICKUSB_ERROR_PACKET_NOT_MULTIPLE_512 | The packet size must be a multiple of 512.   |
| 17 - QUICKUSB_ERROR_PACKET_NOT_MULTIPLE_64  | The packet size must be a multiple of 64.  |
| 18 - QUICKUSB_ERROR_UNKNOWN_SYSTEM_ERROR    | Unknown system error.  |
| 19 - QUICKUSB_ERROR_ABORTED                 | The issued request was aborted.  |
| 20 - QUICKUSB_ERROR_DEPRECATED              | The function is deprecated.  |
| 21 - QUICKUSB_ERROR_INVALID_SERIAL          | An invalid serial number was provided.   |
| 22 - QUICKUSB_ERROR_CANNOT_OPEN_FILE        | The specified file cannot be opened, either because it does not exist or requires special permissions. |
| 23 - QUICKUSB_ERROR_VERIFY_FAILED           | The verify failed.   |
| 24 - QUICKUSB_ERROR_FIRMWARE_ERROR          | The specified firmware file is corrupt or invalid.   |
| 25 - QUICKUSB_ERROR_ALREADY_COMPLETED       | The specified request has already completed.   |
| 26 - QUICKUSB_ERROR_NOT_COMPLETED           | The specified request has not yet completed.   |
| 27 - QUICKUSB_ERROR_FPGA_CONFIG_FAILED      | The FPGA configuration failed.   |
| 28 - QUICKUSB_ERROR_INVALID_OPERATION       | You are trying to perform an invalid operation.  |
| 29 - QUICKUSB_ERROR_TOO_MANY_REQUESTS       | There are too many pending requests.   |
| 30 - QUICKUSB_ERROR_EPCS_NOT_FOUND          | An EPCS device was not found.  |
| 31 - QUICKUSB_ERROR_EPCS_TOO_SMALL          | The EPCS device does not contain enough memory to store the requested data file.                       |
| 32 - QUICKUSB_ERROR_NOT_STREAMING           | The request could not be completed because the device is currently not streaming data.                 |
| 33 - QUICKUSB_ERROR_BUFFER_NOT_ALIGNED      | On Linux, the provided data buffer was not 512-byte aligned.   |
| 34 - QUICKUSB_ERROR_INTERNAL_ERROR          | An internal error occurred.  |
| 35 - QUICKUSB_ERROR_DEVICE_IS_CLOSING       | The operation could not be completed because the device is currently being closed.                     |
| 36 - QUICKUSB_ERROR_PROTECTION              | The operation could not be completed because it would corrupt the device.                              |

Table 11 - QuickUsbGetLastError Error Codes

### **QuickUsbGetLastDriverError**

#### **Purpose**

Return extended driver error information for the last API call that returned a value of FALSE (0).

#### **Parameters**

error:           A PQLONG pointing to a variable in which to place the error code.

#### **Returns**

A QLONG containing one '1', indicating success. This function never fails.

#### **Notes**

This function currently only returns non-zero error values on Windows when the driver reports an error.

## QuickUSB Settings

The QuickUSB module has certain settings that control the behavior of the module. These functions manipulate those settings in order to customize the module's behavior for your particular needs. A list of settings follows:

| Addr | Name               | Description            | Values  |
|------|--------------------|------------------------|---|
| 0    | SETTING_EP26CONFIG | Endpoint configuration | <p>MSB=EP2CFG – Bit definitions:</p> <p>Bit 15: Valid – Activates EP2<br/>0 = EP2 Endpoint not activated<br/>1 = EP2 Endpoint activated</p> <p>Bit 14: Direction – Sets EP2 Direction<br/>0 = Output<br/>1 = Input</p> <p>Bit 13-12: Type – Defines EP2 Type<br/>00 = Invalid<br/>01 = Isochronous<br/>10 = Bulk<br/>11 = Interrupt</p> <p>Bit 11: Size – Sets Size of EP2 Buffer<br/>0 = 512 Bytes<br/>1 = 1024 Bytes</p> <p>Bit 10: Unused R/O = 0</p> <p>Bit 9-8: Buf – EP2 Buffer Type/Amount<br/>00 = Quad<br/>01 = Invalid<br/>10 = Double<br/>11 = Triple</p> <p>LSB=EP6CFG – Bit definitions:</p> <p>Bit 7: Valid – Activates EP6<br/>0 = EP6 Endpoint not activated<br/>1 = EP6 Endpoint activated (default)</p> <p>Bit 6: Direction – Sets EP6 Direction<br/>0 = Output<br/>1 = Input</p> <p>Bit 5-4: Type – Defines EP6 Type<br/>00 = Invalid<br/>01 = Isochronous<br/>10 = Bulk<br/>11 = Interrupt</p> <p>Bit 3: Size – Sets Size of EP6 Buffer<br/>0 = 512 Bytes<br/>1 = 1024 Bytes</p> <p>Bit 2: Unused R/O = 0</p> <p>Bit 1-0: Buf – EP6 Buffer Type/Amount<br/>00 = Quad<br/>01 = Invalid<br/>10 = Double<br/>11 = Triple</p> |

## Using the QuickUSB Library

| Addr | Name                | Description   | Values   |
|------|---------------------|---|--|
| 1    | SETTING_WORDWIDE    | High-speed port data width  | MSB=Unused, Reserved for future use.<br>LSB - Bit definitions:<br>Bit 7-1: Reserved<br>Bit 0: WORDWIDE – HSPP data width<br>0 = 8 bits<br>1 = 16 bits  |
| 2    | SETTING_DATAADDRESS | Data bus starting address and flags to enable and disable 1) the address bus and 2) the feature that auto increments the bus address after each data transaction. | Bit 15:<br>0=Increment address bus<br>1=Don't increment address bus<br>Bit 14:<br>0=enable address bus<br>1=disable address bus (port C[7:0] and E[7] may be used as general purpose I/O)<br>Bit 13-9: Unused<br>Bits 8-0:<br>HSPP address value   |
| 3    | SETTING_FIFO_CONFIG | Sets the FIFO configuration. Controls the FX2 IFCONFIG register.  | MSB=FIFOINPOLAR – Slave FIFO Interface Pins Polarity Bit<br>Definitions:<br>Bit 15-14: Unused R/O = 0<br>Bit 13: PKTEND – FIFO Packet End Polarity<br>0 = Active Low<br>1 = Active High<br>Bit 12: SLOE – FIFO Output Enable Polarity<br>0 = Active Low<br>1 = Active High<br>Bit 11: SLRD – FIFO Read Polarity<br>0 = Active Low<br>1 = Active High<br>Bit 10: SLWR – FIFO Write Polarity<br>0 = Active Low<br>1 = Active High<br>Bit 9: EF – FIFO Empty Flag Polarity<br>0 = Active Low<br>1 = Active High<br>Bit 8: FF – FIFO Full Flag Polarity<br>0 = Active Low<br>1 = Active High<br>LSB=IFCONFIG – Interface Configuration Bit definitions:<br>Bit 7: IFCLKSRC – IFCLK source select<br>0=External clock<br>1=Internal clock<br>Bit 6: 3048MHZ – IFCLK speed select<br>0=30Mhz<br>1=48MHz<br>Bit 5: IFCLKOE – IFCLK output enable<br>0=Tri-state the IFCLK pin<br>1=Drive the IFCLK pin<br>Bit 4: IFCLKPOL – IFCLK polarity select<br>0=Normal |

## Using the QuickUSB Library

| Addr | Name              | Description  | Values  |
|------|-------------------|--|---|
|      |                   |  | 1=Inverted<br>Bit 3: ASYNC – GPIF clock mode select<br>0=Synchronous GPIF<br>1=Asynchronous GPIF<br>Bit 2: Reserved (do not change)<br>Bit 1-0: IFCFG- HSPP Configuration<br>00=I/O ports<br>01=Reserved<br>10=GPIF Master mode<br>11=Slave FIFO mode   |
| 4    | SETTING_FPGATYPE  | Sets the FPGA configuration scheme                           | MSB=Reserved (Reads 0, Ignored if set),<br>LSB – FPGATYPE<br>Bit 7-1 : Reserved<br>Bit 0 : FPGATYPE<br>0 = Altera Passive Serial<br>1 = Xilinx Slave Serial   |
| 5    | SETTING_CPUCONFIG | Sets the CPU configuration. Controls the FX2 CPUCS register. | MSB= Misc Settings<br>Bit 15: USB Bus Speed<br>0=Force full speed (12Mbps)<br>1=Allow high-speed (480Mbps)<br>Bit 14-8: Reserved for future use<br>LSB=CPUCS – Bit definitions:<br>Bit 7-6: Unused R/O = 0<br>Bit 5: Reserved (do not change)<br>Bit 4-3: CLKSPD – CPU clock speed<br>00=12MHz<br>01=24MHz<br>10=48MHz<br>11=Reserved<br>Bit 2: CLKINV – Invert CLKOUT<br>0=Normal<br>1=Invert CLKOUT<br>Bit 1: CLKOE – CLKOUT output enable<br>0=Tri-state the CLKOUT pin<br>1=Drive the CLKOUT pin  |
| 6    | SETTING_SPICONFIG | Configures the SPI interface                                 | Bit 15: GPIFA8 – Enable GPIF Address Pin<br>0 = Configure PE7 as GPIO<br>1 = Configure PE7 as GPIFADR[8] output<br>Bit 14-8: Reserved, do not change<br>Bit 7-6: Reserved.<br>Bit 5: MISOPIN<br>0 = MISO is on pin 5 of the port selected by Bit 3 of this setting (SPIPORT)<br>1 = MISO is on pin 2 of the port selected by Bit 3 of this setting (SPIPORT)<br>Bit 4: NCEPIN<br>0 = nCE is on pin 2 of the port selected by Bit 3 of this setting (SPIPORT)<br>1 = nCE is on pin 7 of the port selected by Bit 3 of this setting (SPIPORT) |

## Using the QuickUSB Library

| Addr | Name                   | Description  | Values   |
|------|------------------------|--|--|
|      |                        |  | <p>Bit 3: SPIPORT – Selects either Port A or Port E for SPI/FPGA communication.<br/> 0 = Use Port E<br/> 1 = Use Port A</p> <p>Bit 2: SPICPHA – Sets SPI clock phase for input sampling.<br/> 0=Sample then clock<br/> 1=Clock then sample</p> <p>Bit 1: SPICPOL – Sets SPI clock polarity.<br/> 0=Normal clock<br/> 1=Inverted clock</p> <p>Bit 0: SPIENDIAN – Sets SPI bit order<br/> 0=LSBit to MSBit<br/> 1=MSBit to LSBIt</p>   |
| 7    | SETTING_SLAVEFIFOFLAGS | <p>Returns Slave FIFO flag status.<br/> Note: These flags are only significant when the FX2 is in slave FIFO mode. When using the Pipeline IO Model, the lower byte represents the pipeline delay.<br/> These flags do not reflect the polarity of the Slave Output Flags on FLAGA, FLAGB, FLAGC, and FLAGD since the polarity can be changed with the SETTING_FIFO_CONFIG register.</p> | <p>Bit 15-12: Reserved for future use<br/> Bit 11: RDY0 Pin Status<br/> Bit 10: Reserved for future use<br/> Bit 9: Empty Flag for EP6 (QuickUsbReadData) FIFO. Active high signal<br/> Bit 8: Full Flag for EP6 (QuickUsbReadData) FIFO. Active high signal<br/> Bit 7-4: Reserved for future use<br/> Bit 3: RDY1 pin status<br/> Bit 2: Reserved for future use<br/> Bit 1: Empty flag for EP2 (QuickUsbWriteData) FIFO. Active high signal<br/> Bit 0: Full Flag for EP2 (QuickUsbWriteData) FIFO. Active high signal.</p>                               |
| 8    | SETTING_I2CTL          | Configures I2C peripheral  | <p>MSB = Last I2C I/O status, read only<br/> Bits 15-8:<br/> 00000110 Bus error<br/> 00000111 No Ack<br/> 00001000 Normal completion<br/> 00001010 Slave wait<br/> 00001011 Timeout<br/> LSB=I2CTL – I2C Compatible Bus Control Bit definitions<br/> Bit 7: IgnoreACK<br/> 0=Handle ACK for normal I2C traffic.<br/> 1=Process I2C traffic even if the device does not supply an ACK (necessary for some I2C peripherals).<br/> Bit 6-1: Reserved for future use<br/> Bit 0: 400KHz – Sets I2C bus clock speed<br/> 0=Approx 100KHz<br/> 1=Approx 400KHz</p> |
| 9    | SETTING_PORTA          | Configures Port A default state. Reading a bit from IOA returns the logic level of the port pin that is two  | <p>MSB = OEA – Port A Output Enable Bit Definitions<br/> 0 = Disables output buffer<br/> 1 = Enables output buffer</p>   |

## Using the QuickUSB Library

| Addr | Name              | Description  | Values  |
|------|-------------------|--|---|
|      |                   | CLKOUT-clocks old.<br>Writing a register bit to IOA writes to the port pin latch. The port latch value appears on the I/O pin if the corresponding OEA bit is set high.  | LSB = IOA – Port A I/O<br>0 = Low logic level<br>1 = High logic level   |
| 10   | SETTING_PORTB     | Configures Port B default state. Reading a bit from IOB returns the logic level of the port pin that is two CLKOUT-clocks old. Writing a register bit to IOB writes to the port pin latch. The port latch value appears on the I/O pin if the corresponding OEB bit is set high. | MSB = OEB – Port B Output Enable Bit Definitions<br>0 = Disables output buffer<br>1 = Enables output buffer<br>LSB = IOB – Port B I/O<br>0 = Low logic level<br>1 = High logic level  |
| 11   | SETTING_PORTC     | Configures Port C default state. Reading a bit from IOC returns the logic level of the port pin that is two CLKOUT-clocks old. Writing a register bit to IOC writes to the port pin latch. The port latch value appears on the I/O pin if the corresponding OEC bit is set high. | MSB = OEC – Port C Output Enable Bit Definitions<br>0 = Disables output buffer<br>1 = Enables output buffer<br>LSB = IOC – Port C I/O<br>0 = Low logic level<br>1 = High logic level  |
| 12   | SETTING_PORTD     | Configures Port D default state. Reading a bit from IOD returns the logic level of the port pin that is two CLKOUT-clocks old. Writing a register bit to IOD writes to the port pin latch. The port latch value appears on the I/O pin if the corresponding OED bit is set high. | MSB = OED – Port D Output Enable Bit Definitions<br>0 = Disables output buffer<br>1 = Enables output buffer<br>LSB = IOD – Port D I/O<br>0 = Low logic level<br>1 = High logic level  |
| 13   | SETTING_PORTE     | Configures Port E default state. Reading a bit from IOE returns the logic level of the port pin that is two CLKOUT-clocks old. Writing a register bit to IOE writes to the port pin latch. The port latch value appears on the I/O pin if the corresponding OEE bit is set high. | MSB = OEE – Port E Output Enable Bit Definitions<br>0 = Disables output buffer<br>1 = Enables output buffer<br>LSB = IOE – Port E I/O<br>0 = Low logic level<br>1 = High logic level  |
| 14   | SETTING_PORTACCFG | Sets Port A & C configuration  | MSB=PORTACFG – I/O Port A Alternate Configuration Pin Definitions<br>Bit 15: FLAGD – Flag D Alternate Configuration<br>1 = PA7 gives FLAGD status when in Slave Mode<br>0 = PA7 does not give FLAGD status in Slave Mode<br><i>Note: If both Bit 15 (FLAGD)</i> |



## Using the QuickUSB Library

| Addr | Name             | Description                      | Values   |
|------|------------------|----------------------------------|--|
|      |                  |                                  | <p>and Bit 14 (SLCS) are set, PA7 will be configured to give the FLAGD status.</p> <p>Bit 14: SLCS – Slave FIFO Chip Select Alternate Configuration<br/>1 = PA7 configured as SLCS input in Slave Mode<br/>0 = PA7 not configured as SLCS input in Slave Mode<br/><i>Note: If both Bit 15 (FLAGD) and Bit 14 (SLCS) are set, PA7 will be configured to give the FLAGD status.</i></p> <p>Bit 13-10: Unused</p> <p>Bit 9: INT1 – Interrupt 1 Alternate Configuration<br/>1 = PA1 configured as interrupt input<br/>0 = PA1 not configured as interrupt input (default)<br/><i>Note: INT1 is not currently used</i></p> <p>Bit 8: INT0 – Interrupt 0 Alternate Configuration<br/>1 = PA0 configured as interrupt input<br/>0 = PA0 not configured as interrupt input (default)<br/><i>Note: INT0 is not currently used</i></p> <p>LSB=PORTCCFG – I/O Port C Alternate Configuration Pin Definitions</p> <p>Bit 7-0: GPIFA7:0 – Enable GPIF Address Pins<br/>1 = Set these pins to "1" to configure this port to output the GPIF Address<br/>0 = Set these pins to "0" to configure this port as Port C</p> |
| 15   | SETTING_PINFLAGS | Sets FIFO pin flag configuration | <p>MSB=PINFLAGSAB – Slave FIFO FLAGA and FLAGB Pin Configuration</p> <p>Bit 15-12: FLAGB – FLAGB Show the status of the FIFO Flag selected by programming these bits with the code given below.</p> <p>Bit 11-8: FLAGA – FLAGA shows the status of the FIFO Flag selected by programming these bits with the code given below.</p> <p>LSB=PINFLAGSCD</p> <p>Bit 7-4: FLAGD – FLAGD shows the status of the FIFO Flag selected by programming these bits with the code given below.</p> <p>Bit 3-0: FLAGC – FLAGC shows the status of the FIFO Flag selected by programming these bits with the code given below.</p> <p>FIFO Flag Select Codes:<br/>'0000' = FLAGA = PF, FLAGB = FF, FLAGC = EF, FLAGD =</p>   |

## Using the QuickUSB Library

| Addr | Name                 | Description  | Values   |
|------|----------------------|--|--|
|      |                      |  | EP2PF. The Endpoint FIFO of the PF, FF, or EF is selected by the FIFOADR[1:0] pins.<br>'0001', '0010', '0011' = Reserved<br>'0100' = EP2PF<br>'0101' = EP4PF<br>'0110' = EP6PF<br>'0111' = EP8PF<br>'1000' = EP2EF<br>'1001' = EP4EF<br>'1010' = EP6EF<br>'1011' = EP8EF<br>'1100' = EP2FF<br>'1101' = EP4FF<br>'1110' = EP6FF<br>'1111' = EP7FF |
| 16   | Reserved.            | Reserved.  | Reserved.  |
| 17   | SETTING_VERSIONSPEED | Returns the CY7C68013 hardware revision and USB bus speed. | MSB= Hardware revision<br>Bits 15-8:<br>00000000=CY7C68013 Rev A/B<br>00000001=CY7C68013A Rev A<br>00000010=CY7C68013 Rev C/D<br>00000100=CY7C68013 Rev E<br>LSB= USB bus speed<br>Bit 7:<br>0=Full Speed (12Mbps)<br>1=High-Speed (480Mbps)<br>Bit 6-0: Reserved for future use   |
| 18   | SETTING_TIMEOUT_HIGH | Access the firmware timeout value.                         | MSB=Firmware Timeout High<br>Bits 15-0: The upper word of the firmware timeout, in milliseconds.   |
| 19   | SETTING_TIMEOUT_LOW  | Access the firmware timeout value.                         | LSB=Firmware Timeout Low<br>Bits 15-0: The lower word of the firmware timeout, in milliseconds.  |

Table 12 - QuickUSB Settings

## QuickUsbReadSetting

### Purpose

Read QuickUSB module settings.

### Parameters

hDevice: A QHANDLE that was returned from a call to [QuickUsbOpen](#).  
 address: A QWORD containing the setting address (number).  
 setting: A PQWORD pointing to a variable in which to place the value of the setting if successful.

### Returns

A QLONG that is non-zero on success, zero (0) otherwise. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

### Notes

- None.

## QuickUsbWriteSetting

### **Purpose**

Write QuickUSB module settings.

### **Parameters**

hDevice: A QHANDLE that was returned from a call to [QuickUsbOpen](#).  
 address: A QWORD containing the setting address (number).  
 setting: A QWORD containing the new setting value.

### **Returns**

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

### **Notes**

- None

## QuickUsbReadDefault

### **Purpose**

Read QuickUSB module defaults. The defaults are non-volatile and are read into the settings table on power up.

### **Parameters**

hDevice: A QHANDLE that was returned from a call to [QuickUsbOpen](#).  
 address: A QWORD containing the default address (number).  
 setting: A PWORD pointing to a variable in which to place the value of the default if successful.

### **Returns**

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

### **Notes**

- None.

## QuickUsbWriteDefault

### **Purpose**

Write QuickUSB module defaults. The defaults are non-volatile and are read into the settings table on power up.

### **Parameters**

hDevice: A QHANDLE that was returned from a call to [QuickUsbOpen](#).  
 address: A QWORD containing the default address (number).  
 setting: A QWORD containing the new default value.

### **Returns**

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

### **Notes**

- None.

# FPGA Configuration

The QuickUSB Plug-In module can configure SRAM-based FPGA devices over the USB. The default FPGATYPE is Passive Serial. For more information on changing the FPGA type, see the [SETTING\\_FPGATYPE](#) setting of the 'Settings' section of this document.

You must convert your FPGA configuration file to binary image format for use with QuickUSB. Altera binary files are of type RBF and Xilinx are of type BIT. If you are configuring multiple devices, they must be daisy-chained and the configuration files combined in the conversion process into a single binary file.

To configure an FPGA, first follow these three steps:

1. (Applies when using the QuickUsb Evaluation Board) Set Port A bit 7 to output high to turn on power to the FPGA. Go to step 3.
2. Apply power to the FPGA.
3. Wait an ample amount of time for the FPGA power rails to stabilize and for the FPGA to power-up.

Then, to configure an FPGA automatically, follow this single step:

4. Call [QuickUsbConfigureFpga](#) with the path to the binary data file to configure the FPGA with. This function automatically performs the processes described in steps 5 – 8 of manually configuring an FPGA.

Otherwise, to configure an FPGA manually, follow these four steps after step 3.

5. Call [QuickUsbStartFpgaConfiguration](#). This resets the FPGA and starts the configuration process.
6. Open the binary FPGA configuration file and read a block into a buffer.
7. Call [QuickUsbWriteFpgaData](#) and pass in the data from the file. Repeat this process until the entire file is written. If you need to restart the configuration process for any reason, you may go to step 5 at any time.
8. Call [QuickUsbIsFpgaConfigured](#). If the 'isConfigured' parameter gets set to '1', the FPGA was correctly configured. If not, try again starting at step 5.

## QuickUsbConfigureFpga

### Purpose

Start the process of FPGA configuration. If the FPGA is in the process of being configured, the process will restart. If the FPGA is already configured, it will be reset and reconfigured.

### Parameters

hDevice: A QHANDLE that was returned from a call to [QuickUsbOpen](#).  
filePath: The path to the binary data file.

### Returns

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

### Notes

- Be sure that the FPGA power rails have stabilized and that the FPGA has had sufficient time to power on before attempting to call this function.
- If the function failed to configure the FPGA, the function fails with error QUICKUSB\_ERROR\_FPGA\_CONFIG\_FAILED.

## QuickUsbStartFpgaConfiguration

### **Purpose**

Start the process of FPGA configuration. If the FPGA is in the process of being configured, the process will restart. If the FPGA is already configured, it will be reset and reconfigured.

### **Parameters**

hDevice: A QHANDLE that was returned from a call to [QuickUsbOpen](#).

### **Returns**

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

### **Notes**

- Be sure that the FPGA power rails have stabilized and that the FPGA has had sufficient time to power on before attempting to call this function.
- You do not need to use this function if you are automatically configuring the FPGA through use of the [QuickUsbConfigureFpga](#) function.

## QuickUsbWriteFpgaData

### **Purpose**

Sends FPGA configuration data to the FPGA using the QuickUSB FPGA configuration port.

### **Parameters**

hDevice: A QHANDLE that was returned from a call to [QuickUsbOpen](#).

data: A PQBYTE pointing to a QBYTE buffer containing the FPGA configuration data.

length: A QULONG containing the length of the data in bytes.

### **Returns**

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

### **Notes**

- A call to [QuickUsbStartFpgaConfiguration](#) must precede FPGA configuration.
- You do not need to use this function if you are automatically configuring the FPGA through use of the [QuickUsbConfigureFpga](#) function.

### QuickUsbIsFpgaConfigured

#### **Purpose**

Check to see if the FPGA is configured.

#### **Parameters**

hDevice: A QHANDLE that was returned from a call to [QuickUsbOpen](#).  
isConfigured: A PQWORD pointing to a QWORD in which to write the configuration status of the FPGA connected to the QuickUSB FPGA configuration port. 1 = the FPGA is configured (CONF\_DONE = '1'), 0 = the FPGA is not configured (CONF\_DONE = '0').

#### **Returns**

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

#### **Notes**

- The value of the CONF\_DONE line is returned as bit 0 of 'isConfigured'.

## High-Speed Parallel Port

### QuickUsbReadCommand

#### Purpose

Read a block of command values from the high-speed parallel port using the QuickUSB module.

#### Parameters

hDevice: A QHANDLE that was returned from a call to [QuickUsbOpen](#).  
 address: A QWORD containing the address. This address is the starting value of the HSPP address bus. If address bit 14 is set (1), then the address bus will not be driven. If address bit 15 is set (1), then the address will not be incremented after each read.  
 data: A pointer to a buffer in which to place data read from the high-speed parallel port. *See notes*.  
 length: A PQWORD pointing to a QWORD containing the number of bytes to read from the high-speed parallel port on input and the number of bytes read on return. *See notes*.

#### Returns

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

#### Notes

- The maximum length is 64 bytes.
- The data buffer can contain data values of any type.
- Commands are transferred over the high-speed parallel port with the CMD\_DATA line set to '1'.
- The address bus behavior for data transfers is defined by the [SETTING\\_DATAADDRESS](#) setting.

### QuickUsbWriteCommand

#### **Purpose**

Write a block of command values to the high-speed parallel port using the QuickUSB module.

#### **Parameters**

hDevice: A QHANDLE that was returned from a call to [QuickUsbOpen](#).  
address: A QWORD containing the address. This address is the starting value of the HSPP address bus. If address bit 14 is set (1), then the address bus will not be driven. If address bit 15 is set (1), then the address will not be incremented after each write.  
data: A pointer to a buffer containing the data to write to the high-speed parallel port. See notes.  
length: A QWORD containing the number of bytes to write to the high-speed parallel port. See notes.

#### **Returns**

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

#### **Notes**

- The maximum length is 64 bytes.
- The data buffer can receive data values of any type.
- Commands are transferred over the high-speed parallel port with the CMD\_DATA line set to '1'.
- The address bus behavior for data transfers is defined by the [SETTING\\_DATAADDRESS](#) setting.



## **QuickUsbReadData**

### **Purpose**

Read a block of data values from the high-speed parallel port using the QuickUSB module.

### **Parameters**

hDevice: A QHANDLE that was returned from a call to [QuickUsbOpen](#).  
 data: A pointer to a buffer in which to place data values read from the HSPP. *See notes*.  
 length: A PQULONG to a QULONG containing the number of bytes to read from the HSPP. Additionally, length is overwritten with the number of bytes actually read. *See notes*.

### **Returns**

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

### **Notes**

- The maximum length is 16 megabytes (16777216 bytes).
- The data buffer can receive data values of any type.
- This function has a minimum call latency of approximately 1 ms for full-speed connections and 250 us for hi-speed connections.
- In order to obtain the maximum performance, call this function with the largest appropriate length value. Each call to this function will incur one call latency delay regardless of the transfer size. So to minimize delays that erode the data transfer rate, make the transfers as large as possible for your application.
- In master mode, data values are transferred over the high-speed parallel port with the CMD\_DATA line set to '0'.
- In slave mode, the USB will wait for 'length' transfers to occur before returning. If the target interface does not write length transfers to the slave FIFOs, the call will return with a value of 0 and set the last error status to QUICKUSB\_ERROR\_TIMEOUT after the timeout value specified by the [QuickUsbSetTimeout](#) function.
- The address bus behavior for data transfers is defined by the [SETTING\\_DATAADDRESS](#) setting.
- On Mac, the length must be a multiple of 512 for Hi-Speed transfers or a multiple of 64 for Low-Speed transfers.

### QuickUsbWriteData

#### Purpose

Write a block of data values to the high-speed parallel port using the QuickUSB module.

#### Parameters

hDevice: A QHANDLE that was returned from a call to [QuickUsbOpen](#).  
data: A pointer to a block of data values to write to the HSPP. See *notes*.  
length: A QULONG containing the number of bytes to write to the HSPP. See *notes*.

#### Returns

A QULONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

#### Notes

- The maximum length is 16 megabytes (16777216 bytes).
- The data buffer can contain data values of any type.
- This function has a minimum call latency of approximately 1 ms for full-speed connections and 250 us for hi-speed connections.
- In order to obtain the maximum performance, call this function with the largest appropriate length value. Each call to this function will incur one call latency delay regardless of the transfer size. So to minimize delays that erode the data transfer rate, make the transfers as large as possible for your application.
- In master mode, data values are transferred over the high-speed parallel port with the CMD\_DATA line set to '0'.
- In slave mode, the USB will hang and wait for 'length' transfers to occur before returning. If the target interface does not read length transfers from the slave FIFOs, the call will return with a value of 0 and set the last error status to QUICKUSB\_ERROR\_TIMEOUT after the timeout value specified by the [QuickUsbSetTimeout](#) function.
- The address bus behavior for data transfers is defined by the [SETTING\\_DATAADDRESS](#) setting.

## **QuickUsbReadDataAsync**

### **Purpose**

Read a block of data values from the high-speed parallel port using an asynchronous function call. Consider using the [Asynchronous Data API](#) or [Streaming Data API](#) in place of this function.

### **Parameters**

hDevice: A QHANDLE that was returned from a call to [QuickUsbOpen](#).  
 data: A pointer to a buffer in which to place data read from the HSPP. See *notes*.  
 length: A PQULONG containing the number of bytes to read from the HSPP. See *notes*.  
 transaction: A PQBYTE to a QBYTE in which to place the transaction identifier required by [QuickUsbAsyncWait](#).

### **Returns**

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

### **Notes**

- This function is deprecated. Please use the [Asynchronous Data API](#) or [Streaming Data API](#) in place of this function.
- QuickUSB asynchronous function calls return immediately and must be followed by a call to [QuickUsbAsyncWait](#) to determine when the transaction actually completes and to free internal buffers used by the operating system. Failure to follow this procedure will result in a memory leak and an eventual system crash.
- The maximum length is 16 megabytes (16777216 bytes).
- The length parameter is not overwritten with the number of bytes actually read as it is with [QuickUsbReadData](#). The length parameter of [QuickUsbAsyncWait](#) is used to retrieve the number of bytes actually read.
- The data buffer can receive data values of any type.
- In master mode, data values are transferred over the high-speed parallel port with the CMD\_DATA line set to '0'.
- In slave mode, the USB will hang and wait for 'length' transfers to occur before returning. If the target interface does not write length transfers to the slave FIFOs, the call will return with a value of 0 and set the last error status to QUICKUSB\_ERROR\_TIMEOUT after the timeout value specified by the [QuickUsbSetTimeout](#) function.
- The address bus behavior for data transfers is defined by the [SETTING\\_DATAADDRESS](#) setting.
- There is a global maximum of 253 asynchronous reads and writes outstanding at any time for all QuickUSB modules in the system.
- This function is not supported on Linux or Mac. For platform independent asynchronous data requests, use the [Asynchronous Data API](#) or [Streaming Data API](#) in place of this function.

### QuickUsbWriteDataAsync

#### Purpose

Write a block of data values to the high-speed parallel port using an asynchronous function call. Consider using the [Asynchronous Data API](#) or [Streaming Data API](#) in place of this function.

#### Parameters

hDevice: A QHANDLE that was returned from a call to [QuickUsbOpen](#).  
data: A pointer to a block of data values to write to the HSPP. See *notes*.  
length: A QULONG containing the number of bytes to write to the HSPP. See *notes*.  
transaction: A PQBYTE to a QBYTE in which to place the transaction identifier required by [QuickUsbAsyncWait](#).

#### Returns

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

#### Notes

- This function is deprecated. Please use the [Asynchronous Data API](#) or [Streaming Data API](#) in place of this function.
- QuickUSB asynchronous function calls return immediately and must be followed by a call to [QuickUsbAsyncWait](#) to determine when the transaction actually completes and to free internal buffers used by the operating system. Failure to follow this procedure will result in a memory leak and an eventual system crash.
- The maximum length is 16 megabytes (16777216 bytes).
- The data buffer can receive data values of any type.
- In master mode, data values are transferred over the high-speed parallel port with the CMD\_DATA line set to '0'.
- In slave mode, the USB will hang and wait for 'length' transfers to occur before returning. If the target interface does not read length transfers from the slave FIFOs, the call will return with a value of 0 and set the last error status to QUICKUSB\_ERROR\_TIMEOUT after the timeout value specified by the [QuickUsbSetTimeout](#) function.
- The address bus behavior for data transfers is defined by the [SETTING\\_DATAADDRESS](#) setting.
- There is a global maximum of 253 asynchronous reads and writes outstanding at any time for all QuickUSB modules in the system.
- This function is not supported on Linux and Mac. For platform independent asynchronous data requests, consider using the [Asynchronous Data API](#) or [Streaming Data API](#) in place of this function

## **QuickUsbAsyncWait**

### **Purpose**

Wait for an asynchronous transfer to complete. Consider using the [Asynchronous Data API](#) or [Streaming Data API](#) in place of this function.

### **Parameters**

hDevice: A QHANDLE that was returned from a call to [QuickUsbOpen](#).

length: An PQULONG that returns the number of bytes that were transferred as a result of the asynchronous function call. If the asynchronous function call is still pending, 'length' will be set to zero '0'. This function must be called until 'length' is non-zero otherwise, the driver will not release its internal buffers, thus causing a memory leak and an eventual system crash (Blue Screen of Death). If the asynchronous function call has completed, the number of bytes requested will be stored in 'length' and all internal buffers will be released.

transaction: A QBYTE transaction identifier returned by [QuickUsbReadDataAsync](#) or [QuickUsbWriteDataAsync](#).

immediate: A QBYTE value. If nonzero, the driver will not wait the default timeout value for the transaction to complete. If zero, the driver will wait the default timeout period for the transaction to complete.

### **Returns**

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#). If the asynchronous function call is still pending, 'length' will be set to zero '0'. This function must be called until 'length' is non-zero. If the asynchronous function call has completed, the number of bytes requested will be stored in 'length'.

### **Notes**

- This function is deprecated. Please use the [Asynchronous Data API](#) or [Streaming Data API](#) in place of this function.
- This function is not supported on Linux and Mac. For platform independent asynchronous data requests, consider using the [Asynchronous Data API](#) or [Streaming Data API](#) in place of this function

### Asynchronous Data API

The Asynchronous Data API is used to perform asynchronous data transfers. The Asynchronous Data API functions are the successors of the [QuickUsbReadDataAsync](#), [QuickUsbWriteDataAsync](#), and [QuickUsbAsyncWait](#) functions and should be used in place of them for all new development (in fact, on Windows beginning with v2.15.0 the older asynchronous functions have been replaced with wrappers that simply call the new Asynchronous Data API functions).

When an asynchronous data request is issued, the issuing call returns immediately without waiting for the transaction to complete. This allows additional processing to take place while a data transaction completes in the background. Multiple asynchronous transactions may be issued at once to help reduce software latencies in issuing and scheduling data requests allowing for maximum data throughput and data processing performance. If you need to continually perform a number of successive data requests, look at using the [Streaming Data API](#), which automatically handles issuing and re-issuing data requests for continuous data transfers.

Once an asynchronous request has been issued, there are two ways to determine when that request has completed (either successfully or not). The first is to test if the transaction has completed (or wait for the transaction to complete) via the [QuickUsbBulkWait](#) function. The second is to specify a completion routine (callback function) to be executed when the transaction has completed.

To further improve performance, the Asynchronous Data API functions may optionally be multithreaded via the [QuickUsbSetNumAsyncThreads](#) and [QuickUsbGetNumAsyncThreads](#) functions. When multithreading, completion routines execute asynchronously on threads internally managed by the QuickUSB API. This allows for fast processing of data requests as they complete on multi-core/multiprocessor systems, especially when time consuming data processing needs to occur in real time without hindering data throughput. If the Asynchronous Data API is used without multithreading (default behavior), completion routines are called from the main application thread after a call to [QuickUsbBulkWait](#) indicates that the request has completed. However, when multithreading completion routines are asynchronously called by internally allocated worker threads and the use of [QuickUsbBulkWait](#) becomes optional.

Note that when multithreading, completion routines execute on threads other than the main application thread. Because of this, global data/variables and data/variables shared between the main application thread and accessed in the completion routines must be appropriately protected as to be thread-safe.

## QuickUsbAllocateDataBuffer

### Purpose

Allocate a data buffer that may be used with the [Asynchronous Data API](#) or the [Streaming Data API](#).

### Parameters

|         |  |
|---------|--|
| buffer: | A pointer to the data buffer pointer (PQBYTE *). |
| bytes:  | The number of bytes to allocate for the buffer.  |

### Returns

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

### Notes

- You must free the data buffer with a call to [QuickUsbFreeDataBuffer](#) when you are done using the data buffer.
- This function ensures that the data buffer is properly aligned on platforms that require buffer alignment for asynchronous requests, such as Linux.

## QuickUsbFreeDataBuffer

### Purpose

Free a data buffer that was allocated with [QuickUsbAllocateDataBuffer](#).

### Parameters

|         |  |
|---------|--|
| buffer: | A pointer to the data buffer pointer (PQBYTE *). |
|---------|--|

### Returns

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

### Notes

- Once the buffer is freed, the buffer pointer is set to zero to prevent access to the freed memory.

### QuickUsbReadBulkDataAsync

#### Purpose

Issue an asynchronous bulk data read request.

#### Parameters

- hDevice: A QHANDLE that was returned from a call to [QuickUsbOpen](#).
- buffer: The data buffer. This buffer must remain valid until the request completes.
- bytes: The number of bytes to read.
- bulkStream: A pointer to a user-allocated QBULKSTREAM variable used to store all the information about the request. This data must remain valid until the request completes.
- cRoutine: The [completion routine](#) (callback function) to call when the request completes. If zero (0) or NULL is specified, then no completion routine is called at the completion of the transaction and [QuickUsbBulkWait](#) must be called to determine when the transaction has completed. When multithreading, this routine executes from another thread and multiple threads may execute this routine at the same time. When single-threaded (not multithreading), the [QuickUsbBulkWait](#) must be called to determine when the transaction has completed, which will then execute the completion routine.
- tag: A user pointer passed on to the completion routine used to associate user information with the request.

#### Returns

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

#### Notes

- On Linux, the data buffer must be 512-byte aligned. Using the [QuickUsbAllocateDataBuffer](#) and [QuickUsbFreeDataBuffer](#) API functions ensures that the data buffer is properly aligned on all supported platforms.



## QuickUsbWriteBulkDataAsync

### **Purpose**

Issue an asynchronous bulk data write request.

### **Parameters**

- hDevice: A QHANDLE that was returned from a call to [QuickUsbOpen](#).
- buffer: The data buffer. This buffer must remain valid until the request completes.
- bytes: The number of bytes to write.
- bulkStream: A pointer to a user-allocated QBULKSTREAM variable used to store all the information about the request. This data must remain valid until the request completes.
- cRoutine: The [completion routine](#) (callback function) to call when the request completes. If zero (0) or NULL is specified, then no completion routine is called at the completion of the transaction and [QuickUsbBulkWait](#) must be called to determine when the transaction has completed. When multithreading, this routine executes from another thread and multiple threads may execute this routine at the same time. When single-threaded (not multithreading), the [QuickUsbBulkWait](#) must be called to determine when the transaction has completed, which will then execute the completion routine.
- tag: A user pointer passed on to the completion routine used to associate user information with the request.

### **Returns**

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

### **Notes**

- On Linux, the data buffer must be 512-byte aligned. Using the [QuickUsbAllocateDataBuffer](#) and [QuickUsbFreeDataBuffer](#) API functions ensures that the data buffer is properly aligned on all supported platforms.

### QuickUsbBulkWait

#### **Purpose**

Determine if an asynchronous transaction has completed, wait for an asynchronous transaction to complete, or wait for all pending asynchronous transactions to complete.

#### **Parameters**

**hDevice:** A QHANDLE that was returned from a call to [QuickUsbOpen](#).  
**bulkStream:** A pointer to a user-allocated QBULKSTREAM variable that has been associated with a data transaction from a call to [QuickUsbReadBulkDataAsync](#) or [QuickUsbWriteBulkDataAsync](#). If zero (0) or NULL is specified, then this function will wait until ALL pending requests complete and execute their completion routines.  
**immediate:** If false or zero, this function will wait for the specified request to complete and execute its completion routine. If true or non-zero, this function will return immediately and indicate whether the specified transaction has completed (or all transactions if bulkStream is zero or NULL) by either returning true (1) or by returning zero (0) with a call to [QuickUsbGetLastError](#) indicating QUICKUSB\_ERROR\_NOT\_COMPLETED.

#### **Returns**

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

#### **Notes**

- None.

## QuickUsbSetNumAsyncThreads

### **Purpose**

Set the number of threads and the thread concurrency for the [Asynchronous Data API](#).

### **Parameters**

- hDevice: A QHANDLE that was returned from a call to [QuickUsbOpen](#).
- numThreads: A word indicating the number of threads to allocate for the Asynchronous API. If numThreads is zero, no threads are created and the Asynchronous API will be single-threaded. If numThreads is greater than zero, the Asynchronous API is multithreaded—the indicated number of threads will be created and dedicated to asynchronously processing completion routines. When multithreading, the recommended number of threads to maximize multithreading performance is equal to the number of processors/cores in the system times two.
- concurrency: A word indicating the number of threads that are allowed to execute completion routines simultaneously. Specify zero if numThreads is zero. When multithreading, the number of threads allowed to execute completion routines concurrently will be limited by the value of 'concurrency', which must be at least one. The recommended concurrency to maximize multithreading performance is equal to the total number of processors/cores in the system.

### **Returns**

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

### **Notes**

- The concurrency value has no meaning when not multithreading (numThreads = 0).
- When multithreading (numThreads > 0), thread concurrency must be at least one.
- The number of threads and thread concurrency are reset back to their default value of zero once the device is closed with a call to [QuickUsbClose](#).

### QuickUsbGetNumAsyncThreads

#### **Purpose**

Get the number of threads and the thread concurrency for the [Asynchronous Data API](#).

#### **Parameters**

hDevice: A QHANDLE that was returned from a call to [QuickUsbOpen](#).

numThreads: A pointer to a word indicating the number of threads allocated for the Asynchronous API.

concurrency: A pointer to a word indicating the number of threads that are allowed to execute completion routines simultaneously.

#### **Returns**

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

#### **Notes**

- The concurrency value has no meaning when not multithreading (numThreads = 0).
- When multithreading (numThreads > 0), thread concurrency will be at least one.

## Streaming Data API

The streaming data API may be used to continuously receive or send data with maximum performance and ease-of-use. To ensure the highest performance, the streaming data APIs work on arrays of data buffers. As each buffer of data is sent/received, a completion routine (callback function) is executed to notify your application that, for data reads, a buffer of data has been received and now may be processed or for data writes, a buffer of data has been written out and the next packet of data may be loaded into the buffer. Asynchronous data requests are issued for each data buffer in the stream to minimize software latencies and the data buffers are used sequentially in a circular manner.

To further improve performance, the Streaming Data API functions may be multithreaded. When multithreading, completion routines execute asynchronously on threads internally managed by the QuickUSB API. This allows for fast processing of data requests as they complete on multi-core/multiprocessor systems, especially when time consuming data processing needs to occur in real time without hindering data throughput. If the Streaming Data API is used without multithreading, completion routines are called from the main application thread as [QuickUsbProcessStream](#) is called. The number of threads and the thread concurrency for the Streaming Data API is set through parameters to the [QuickUsbReadBulkDataStartStream](#) and [QuickUsbWriteBulkDataStartStream](#).

Note that when multithreading, completion routines execute on threads other than the main application thread. Because of this, global data/variables and data/variables shared between the main application thread and accessed in the completion routines must be appropriately protected as to be thread-safe.

### QuickUsbReadBulkDataStartStream

#### **Purpose**

Start streaming data into the computer from the QuickUSB device.

#### **Parameters**

- hDevice:** A QHANDLE that was returned from a call to [QuickUsbOpen](#).
- buffers:** An array of pointers to the user allocated buffers used to store the read data. These buffers must remain valid until the read stream has been stopped. If this parameter is zero (0) or NULL then the API will automatically allocate and manage the memory for the data buffers.
- numBuffers:** The number of data buffers to use.
- bufferSize:** The size of each buffer, in bytes.
- cRoutine:** The [completion routine](#) (callback function) that is called each time a buffer is filled with data. This routine executes from another thread and multiple threads may execute this routine at the same time.
- tag:** A pointer containing user-specific data that is passed into the completion routine. This parameter may optionally be zero (0) or NULL.
- streamID:** A pointer to a QBYTE storing the ID of the stream.
- numThreads:** A word indicating the number of threads to allocate for this stream. If numThreads is zero, no threads are created and the stream will be single-threaded. If numThreads is greater than zero, the stream is multithreaded—the indicated number of threads will be created and dedicated to asynchronously processing completion routines. When multithreading, the recommended number of threads to maximize multithreading performance is equal to the number of processors/cores in the system times two.
- concurrency:** A word indicating the number of threads that are allowed to execute completion routines simultaneously. Specify zero if numThreads is zero. When multithreading, the number of threads allowed to execute completion routines concurrently will be limited by the value of 'concurrency', which must be at least one. The recommended concurrency to maximize multithreading performance is equal to the total number of processors/cores in the system.

#### **Returns**

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

#### **Notes**

- To stop the data stream, you must call [QuickUsbBulkDataStopStream](#).
- If the 'buffers' parameter is null then the API will internally allocate and manage the memory for the data buffers.
- The concurrency value has no meaning when not multithreading (numThreads = 0).
- When multithreading (numThreads > 0), thread concurrency will be at least one.
- On Linux, the data buffers must be 512-byte aligned. Using the [QuickUsbAllocateDataBuffer](#) and [QuickUsbFreeDataBuffer](#) API functions ensures that the data buffer is properly aligned on all supported platforms.

## QuickUsbWriteBulkDataStartStream

### Purpose

Start streaming data from the computer to the QuickUSB device.

### Parameters

- hDevice: A QHANDLE that was returned from a call to [QuickUsbOpen](#).
- buffers: An array of pointers to the user allocated buffers used to store the read data. These buffers must remain valid until the read stream has been stopped.
- numBuffers: The number of data buffers to use.
- bufferSize: The size of each buffer, in bytes.
- cRoutine: The [completion routine](#) (callback function) that is called each time a buffer has been written to the device. The buffer must be refilled with new data to write from within the completion routine. This routine executes from another thread and multiple threads may execute this routine at the same time.
- tag: A pointer containing user-specific data that is passed into the completion routine. tag - A pointer containing user-specific data that is passed into the completion routine. This parameter may optionally be zero (0) or NULL.
- numThreads: A word indicating the number of threads to allocate for this stream. If numThreads is zero, no threads are created and the stream will be single-threaded. If numThreads is greater than zero, the stream is multithreaded—the indicated number of threads will be created and dedicated to asynchronously processing completion routines. When multithreading, the recommended number of threads to maximize multithreading performance is equal to the number of processors/cores in the system times two.
- concurrency: A word indicating the number of threads that are allowed to execute completion routines simultaneously. Specify zero if numThreads is zero. When multithreading, the number of threads allowed to execute completion routines concurrently will be limited by the value of 'concurrency', which must be at least one. The recommended concurrency to maximize multithreading performance is equal to the total number of processors/cores in the system.

### Returns

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

### Notes

- To stop the data stream, you must call [QuickUsbBulkDataStopStream](#).
- The concurrency value has no meaning when not multithreading (numThreads = 0).
- When multithreading (numThreads > 0), thread concurrency will be at least one.
- On Linux, the data buffer must be 512-byte aligned. Using the [QuickUsbAllocateDataBuffer](#) and [QuickUsbFreeDataBuffer](#) API functions ensures that the data buffer is properly aligned on all supported platforms.

### QuickUsbBulkDataStopStream

#### Purpose

Stop a previously started read or write bulk data stream.

#### Parameters

- hDevice: A QHANDLE that was returned from a call to [QuickUsbOpen](#).
- streamID: The stream ID, as returned from a call to [QuickUsbReadBulkDataStartStream](#) or [QuickUsbWriteBulkDataStartStream](#).
- immediate: If zero (or false), this function will block until the stream has completely stopped and any remaining completion routines have executed. If non-zero (or true), the stream will be marked to shutdown and will return immediately without waiting for any remaining completion routines to execute.

#### Returns

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

#### Notes

- If wait is zero (or false), you must ensure that the stream has fully stopped before freeing the stream buffers (if they were user-allocated).
- This function may be called from within a completion routine, but the 'wait' parameter must be zero (or false) or deadlock may occur.

### QuickUsbProcessStream

#### Purpose

For single-threaded streams, this function is called to process and re-issue completed requests.

#### Parameters

- hDevice: A QHANDLE that was returned from a call to [QuickUsbOpen](#).
- streamID: The ID of the stream to process, as returned from a call to [QuickUsbReadBulkDataStartStream](#) or [QuickUsbWriteBulkDataStartStream](#).
- milliseconds: The number of milliseconds the caller is willing to wait until a data request on the indicated stream has completed and is processed. If no data request on the indicated stream has completed within the specified amount of time, the function will return unsuccessfully and a call to [QuickUsbGetLastError](#) will indicate the error QUICKUSB\_ERROR\_TIMEOUT. If milliseconds is zero, the function will process any requests that are completed or timeout immediately.

#### Returns

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

#### Notes

- This function does not need to be called on multithreaded streams, but must be called on single-threaded streams (typically repeatedly from within a main application loop) in order to execute completion routines and re-issue requests.



## QuickUsbPauseStream

### Purpose

Pause the automatic re-issuing of data requests on a stream.

### Parameters

- hDevice: A QHANDLE that was returned from a call to [QuickUsbOpen](#).
- streamID: The ID of the stream to process, as returned from a call to [QuickUsbReadBulkDataStartStream](#) or [QuickUsbWriteBulkDataStartStream](#).
- immediate: If zero (or false), this function will block until the stream has paused and all in-process requests have completed. If non-zero (or true), the stream will be marked to enter the pause state and will return immediately without waiting for any in-process requests to complete.

### Returns

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

### Notes

- None.

## QuickUsbResumeStream

### Purpose

Resume the automatic re-issuing of data requests on a stream.

### Parameters

- hDevice: A QHANDLE that was returned from a call to [QuickUsbOpen](#).
- streamID: The ID of the stream to process, as returned from a call to [QuickUsbReadBulkDataStartStream](#) or [QuickUsbWriteBulkDataStartStream](#).

### Returns

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

### Notes

- None.

### General-Purpose I/O

There are five 8-bit general-purpose I/O ports on the QuickUSB module named A through E. Many of the pin functions on each port serve multiple functions and may be not be available for GPIO. For example, Port A or Port E may be configured for FPGA configuration and/or SPI communication and therefore some of the pins on Port A/E will be consumed. Note that the 56-Pin version of the FX2 does not have Port C or Port E.

Other than that, the general-purpose I/O ports are just like I/O ports you would find on a microcontroller. The functions provided by the QuickUSB Library give you the capability to set the direction of each pin, and read/write to the ports on a byte wide basis.

#### QuickUsbReadPortDir

##### **Purpose**

Read the data direction of each data port bit for the specified port.

##### **Parameters**

hDevice: A QHANDLE that was returned from a call to [QuickUsbOpen](#).  
address: A QBYTE containing the port address. Ports are addressed 0 to 4 corresponding to port A-E.  
data: A PQBYTE to a QBYTE in which to place the data direction bit values. Each bit in data corresponds to data bits of the specified port. A data direction bit value of 0=input and 1=output (i.e. 0x03 means that bits 0 and 1 are outputs and bits 2-7 are inputs).

##### **Returns**

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

##### **Notes**

- None

#### QuickUsbWritePortDir

##### **Purpose**

Set the data direction of each data port bit for the specified port.

##### **Parameters**

hDevice: A QHANDLE that was returned from a call to [QuickUsbOpen](#).  
address: The port address. Ports are addressed 0 to 4 corresponding to port A-E.  
data: A byte that contains the data direction bit values. Each bit in data corresponds to data bits of the specified port. A data direction bit value of 0=input and 1=output.

##### **Returns**

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

##### **Notes**

- None

## QuickUsbReadPort

### Purpose

Read a series of bytes from the specified port.

### Parameters

hDevice: A QHANDLE that was returned from a call to [QuickUsbOpen](#).  
 address: The port address. Ports are addressed 0 to 4 corresponding to port A-E.  
 data: A pointer to an array of bytes in which to place the data. This buffer must be at least 'length' bytes long.  
 length: A pointer to the number of bytes to read from the port. The bytes are read sequentially. The maximum length is 64 bytes.

### Returns

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

### Notes

- This function reads an array of values from the specified port. There is no synchronization mechanism provided so the values are read as fast as the microcontroller can read them.

## QuickUsbWritePort

### Purpose

Write a series of bytes to the specified port.

### Parameters

hDevice: A QHANDLE that was returned from a call to [QuickUsbOpen](#).  
 address: The port address. Ports are addressed 0 to 4 corresponding to port A-E.  
 data: A pointer to an array of bytes to send out the port. This buffer must be at least 'length' bytes long.  
 length: A pointer to the number of bytes to write to the port. The bytes are written sequentially. The maximum length is 64 bytes.

### Returns

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

### Notes

- This function writes an array of values from the specified port. Of course, the most common array is an array of 1 byte. Writing multiple bytes out makes it possible to clock out a series of bits if one of the bits is used as the clock. Up to 32 clock cycles can be generated using this technique.

### RS-232 Port

The RS-232 ports are interrupt-driven for transmit and receive. The QuickUSB RS232 receive buffers are 128 bytes deep, so your software just needs to service these buffers often enough to make sure they do not overflow.

#### QuickUsbSetRs232BaudRate

##### Purpose

Set the baud rate for both serial ports. Baud rates are programmable from 4800 to 230k baud. This function sets the baud rate of both serial ports. It is not possible to set the baud rate of each serial port independently.

##### Parameters

hDevice: A QHANDLE that was returned from a call to [QuickUsbOpen](#).  
baudRate: An unsigned long integer (32-bits) containing the baud rate in bits per second.

##### Returns

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

##### Notes

- The default baud rate is 9600 baud.

#### QuickUsbGetNumRS232

##### Purpose

Read the number of characters waiting in the receive buffer.

##### Parameters

hDevice: A QHANDLE that was returned from a call to [QuickUsbOpen](#).  
portNum: The serial port number. Serial port 0 (P1) = 0, serial port 1 (P2) = 1.  
length: A pointer to the number of characters to read. Set to the number of characters actually read on return.

##### Returns

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

##### Notes

- None.

## QuickUsbFlushRS232

### Purpose

Flush the RS232 port transmit and receive buffers.

### Parameters

hDevice: A QHANDLE that was returned from a call to [QuickUsbOpen](#).  
 portNum: The serial port number. Serial port 0 (P1) = 0, serial port 1 (P2) = 1.

### Returns

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

### Notes

- None.

## QuickUsbReadRS232

### Purpose

Read a block of characters from the interrupt receive buffer of the specified QuickUSB serial port.

### Parameters

hDevice: A QHANDLE that was returned from a call to [QuickUsbOpen](#).  
 portNum: The serial port number. Serial port 0 (P1) = 0, serial port 1 (P2) = 1.  
 data: A pointer to a buffer in which to place the data. The buffer must be at least 128 bytes long.  
 length: A pointer to the number of characters to read. Set to the number of characters actually read on return.

### Returns

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

### Notes

- The interrupt buffer is 128 bytes deep. If length is set to more than 128, the routine will hang and wait for the specified number of characters to be read from the port before returning.
- Do not request a read for more data than is reported as available by the [QuickUsbGetNumRS232](#) function. Doing so will place the firmware in a loop waiting for data, causing the firmware to essentially lock up until data arrives.

### **QuickUsbWriteRS232**

#### **Purpose**

Write a block of characters to the specified QuickUSB serial port.

#### **Parameters**

hDevice: A QHANDLE that was returned from a call to [QuickUsbOpen](#).  
portNum: The serial port number. Serial port 0 (P1) = 0, serial port 1 (P2) = 1.  
data: A pointer to a buffer containing the data.  
length: The number of characters to write.

#### **Returns**

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

#### **Notes**

- None

## I2C-Compatible Port

The QuickUSB I<sup>2</sup>C-compatible port is a master-only bus controller with 7-bit addressing. The bus speed is selectable via Bit 0 of [SETTING\\_I2CTL](#) and can run at 100kHz or 400kHz. The R/W bit is automatically inserted, so it should not need to be included in the address. The address is automatically shifted to accommodate the R/W bit. Addresses 81 (decimal) and 1 are reserved.

### QuickUsbReadI2C

#### Purpose

Read for the I2C port.

#### Parameters

|          |   |
|----------|---|
| hDevice: | A QHANDLE that was returned from a call to <a href="#">QuickUsbOpen</a> . |
| address: | The device address.   |
| data:    | A pointer to a buffer in which to place the data.                         |
| length:  | The length of the data buffer in bytes. The maximum length is 64 bytes.   |

#### Returns

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

#### Notes

- Some devices require that a write occur before a read in a single read transaction by issuing a second I2C START command after the write instead of a STOP command. To do this use the [QuickUsbCachedWriteI2C](#) function.

### QuickUsbWriteI2C

#### Purpose

Write to the I2C port

#### Parameters

|          |   |
|----------|---|
| hDevice: | A QHANDLE that was returned from a call to <a href="#">QuickUsbOpen</a> . |
| address: | The device address.   |
| data:    | A pointer to the data to send.  |
| length:  | The length of the data buffer in bytes. The maximum length is 64 bytes.   |

#### Returns

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

#### Notes

- Some devices require that a write occur before a read in a single read transaction by issuing a second I2C START command after the write instead of a STOP command. To do this use the [QuickUsbCachedWriteI2C](#) function.

### **QuickUsbCachedWriteI2C**

#### **Purpose**

Cache a write to the I2C port, which will be performed in following I2C read transaction. This allows an operation, sometimes referred to as a repeated start, where you may perform a write and then a read in a single I2C transaction.

#### **Parameters**

|          |   |
|----------|---|
| hDevice: | A QHANDLE that was returned from a call to <a href="#">QuickUsbOpen</a> . |
| address: | The device address.   |
| data:    | A pointer to the data to send.  |
| length:  | The length of the data buffer in bytes. The maximum length is 64 bytes.   |

#### **Returns**

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

#### **Notes**

- Some devices require that a write occur before a read in a single read transaction by issuing a second I2C START command after the write instead of a STOP command. The above read and write functions normally only issue a single I2C START and STOP command. To perform a write and then a read in a single transaction, you must first cache a write and then issue a read.
- This function does not execute the write and instead cache the write transaction. Following this function call you should issue an I2C read command as normal. The read will be executed following the cached write in a single transaction.



## SPI-Compatible Port

The QuickUSB module implements a ‘soft’ SPI port using pins on Port E and optionally Port A. These routines support from up to 10 devices with individual active-low slave select lines for each device. The signals names are MOSI, SCK, MISO, and nSS0-9 and may be found in [Table 9 - QuickUSB Pin Definitions](#). By default, data is shifted in and out MSB to LSB. The bit shift order, clock phase, and clock polarity can all be configured through the [SETTINGS\\_SPICONFIG](#) setting. The SPI bus writes at a little over 600 Kbps and reads at almost 500 Kbps. You can learn more about QuickUSB SPI interface in the [SPI](#) section of this document.

### QuickUsbReadSpi

#### Purpose

Read a block of bytes from the specified SPI slave port.

#### Parameters

hDevice: A QHANDLE that was returned from a call to [QuickUsbOpen](#).  
 portNum: The SPI device address (nSS line) to read from.  
 data: A pointer to a buffer in which to place the received data.  
 length: A pointer to the number of bytes to read. The maximum length is 64 bytes.

#### Returns

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#). In addition, the data buffer is filled with the received data and the length is set to the number of bytes actually received on success. Both data and length are left unchanged if the function failed.

#### Notes

- None

### QuickUsbWriteSpi

#### Purpose

Write a block of bytes to the specified SPI slave port.

#### Parameters

hDevice: A QHANDLE that was returned from a call to [QuickUsbOpen](#).  
 portNum: The SPI device address (nSS line) to write to.  
 data: A pointer to the data to send.  
 length: The number of bytes to send. The maximum length is 64 bytes.

#### Returns

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

#### Notes

- This function ignores data received at the MISO pin while writing. If you need to capture MISO data while writing, use the [QuickUsbWriteReadSpi](#) function.

### **QuickUsbWriteReadSpi**

#### **Purpose**

Simultaneously write and read a block of data to and from the specified SPI slave port.

#### **Parameters**

|          |   |
|----------|---|
| hDevice: | A QHANDLE that was returned from a call to <a href="#">QuickUsbOpen</a> .                       |
| portNum: | The SPI device address (nSS line) to write to and read from.                                    |
| data:    | A pointer to the buffer that contains the data to send and in which to place the received data. |
| length:  | The number of bytes to send and receive. The maximum length is 64 bytes.                        |

#### **Returns**

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#). In addition, the data buffer is filled with the received data and the length is set to the number of bytes actually received on success. Both data and length are left unchanged if the function failed.

#### **Notes**

- This function uses the data buffer for both writing data to the SPI and to store read data from the SPI. Therefore, the data buffer will always be overwritten on each call to this function.

## Storage API

The QuickUSB module reserves 2 KB (2048 bytes) of user accessible memory. This memory may be used to store information within the QuickUSB module that is preserved during power-cycles.

### QuickUsbReadStorage

#### Purpose

Read a block of bytes from memory.

#### Parameters

|          |   |
|----------|---|
| hDevice: | A QHANDLE that was returned from a call to <a href="#">QuickUsbOpen</a> .   |
| address: | A QWORD indicating the byte offset into memory where the read should begin. |
| data:    | A pointer to a buffer in which to place the received data.                  |
| bytes:   | A QWORD indicating the number of bytes to read.                             |

#### Returns

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

#### Notes

- The address plus the number of bytes to write must not exceed 2048 or the function will fail.

### QuickUsbWriteStorage

#### Purpose

Write a block of bytes to memory.

#### Parameters

|          |  |
|----------|--|
| hDevice: | A QHANDLE that was returned from a call to <a href="#">QuickUsbOpen</a> .    |
| address: | A QWORD indicating the byte offset into memory where the write should begin. |
| data:    | A pointer to the data buffer to write to memory.                             |
| bytes:   | A QWORD indicating the number of bytes to write.                             |

#### Returns

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

#### Notes

- The address plus the number of bytes to write must not exceed 2048 or the function will fail.

# Programming API

The programming API allows you to program different QuickUSB firmware into a QuickUSB device if, for example, you wish to programmatically upgrade devices to newer firmware or change the firmware IO model. This API cannot be used to program blank firmware—it may be used only to program firmware already containing valid QuickUSB firmware. If you are using QuickUSB iChipPacks, you must use the QuickUSB Programmer to perform the initial firmware load.

## QuickUsbWriteFirmware

### Purpose

Write QuickUSB firmware to a device.

### Parameters

|           |   |
|-----------|---|
| hDevice:  | A QHANDLE that was returned from a call to <a href="#">QuickUsbOpen</a> .   |
| fileName: | The path and file name of the QuickUSB firmware (QUSB) file.  |
| options:  | Specify QUICKUSB_PRESERVE_CUSTOMIZATION to preserve customizations present in the current firmware, QUICKUSB_PRESERVE_DEFAULTS to preserve default settings present in the current firmware, QUICKUSB_PRESERVE_GPIF to preserve GPIF customizations, and/or QUICKUSB_PRESERVE_SERIAL to preserve the serial number. To specify a new serial number, pass QUICKUSB_PROGRAM_SERIAL bitwise or'ed ( ) together with the new serial number (must be between 1 and 65535). |
| callback: | A pointer to a QPROGRESS_CALLBACK function which is called during the programming process to report the percentage completed. This may be used to perform background processes and GUI updates while the programming process takes place.   |
| tag:      | A PQVOID variable to store user data that is passed onto the callback function.   |

### Returns

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

### Notes

- The 'callback' parameter may be zero (NULL) if no callback operation is desired.
- Do not attempt to access the device while it is being programmed.
- Multiple option flags may be bitwise or'ed (|) together.
- If either the QuickUSB library version ([QuickUsbGetDllVersion](#)) or the QuickUSB firmware version ([QuickUsbGetFirmwareVersion](#)) are v2.15.0 (or later), the write process will take approximately 4 seconds. Otherwise, the write process will take approximately 35 seconds.
- After the current firmware has been overwritten with new firmware, the device requires a power-cycle to load the new firmware and execute it.
- If the programming process is interrupted in any way, ensure that good firmware is present in the EEPROM through use of the [QuickUsbVerifyFirmware](#) function before power cycling the device, or the device may fail to enumerate.

**QuickUsbVerifyFirmware****Purpose**

Verify that the firmware contained in the specified device matches that of the supplied firmware file.

**Parameters**

|           |   |
|-----------|---|
| hDevice:  | A QHANDLE that was returned from a call to <a href="#">QuickUsbOpen</a> .   |
| fileName: | The path and file name of the QuickUSB firmware (QUSB) file.  |
| callback: | A pointer to a QPROGRESS_CALLBACK function which is called during the verify process to report the percentage completed. This may be used to perform background processes and GUI updates while the verify process takes place. |
| tag:      | A PQVOID variable to store user data that is passed onto the callback function.   |

**Returns**

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

**Notes**

- The 'callback' parameter may be zero (NULL) if no callback operation is desired.
- Do not attempt to access the device while it is being verified.
- Multiple option flags may be bitwise or'ed (|) together.
- The verify process takes approximately 2 seconds to complete.

### EPCS API

The EPCS API allows you to interact with Altera FPGA serial configuration devices using the SPI bus. The API currently supports EPCS1 (1 Mbit), EPCS4 (4 Mbit), EPCS16 (16 Mbit), and EPCS64 (64 Mbit) devices that serially configure Arria® series, Cyclone® series, all device families in the Stratix® series except the Stratix device family, and FPGAs using the active serial (AS) configuration scheme.

### QuickUsbIdentifyEpcs

#### Purpose

Identify the EPCS device connected to the SPI bus on the specified nSS line.

#### Parameters

|           |  |
|-----------|--|
| hDevice:  | A QHANDLE that was returned from a call to <a href="#">QuickUsbOpen</a> .  |
| nSS:      | The nSS SPI line that the EPCS device is connected to (0-9).   |
| epcsId:   | A pointer to QWORD used to store the ID of the EPCS found. This will be one of QUICKUSB_EPCS1_ID (0x10), QUICKUSB_EPCS4_ID (0x12), QUICKUSB_EPCS16_ID (0x14), QUICKUSB_EPCS64_ID (0x16), or QUICKUSB_EPCS_ID_UNKNOWN (0x00).                                 |
| epcsSize: | A pointer to a QULONG used to store the size of the EPCS device, in bytes.   |
| flags:    | Additional flags controlling the operation of this function. Specify QUICKUSB_EPCS_IGNORE_NCE to prevent the function from automatically asserting the nCE line of the FPGA at the beginning of the function are de-asserting it at the end of the function. |

#### Returns

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

#### Notes

- If QUICKUSB\_EPCS\_IGNORE\_NCE is not specified, the nCE line of QuickUSB must be connected to the nCE line of the FPGA.
- If QUICKUSB\_EPCS\_IGNORE\_NCE is specified, then the nCE line of the FPGA must be asserted before calling this function to disable the FPGA allowing communication access to the EPCS device.

## QuickUsbConfigureEpcs

### Purpose

Write the specified FPGA data file to the EPCS device.

### Parameters

|           |  |
|-----------|--|
| hDevice:  | A QHANDLE that was returned from a call to <a href="#">QuickUsbOpen</a> .  |
| nSS:      | The nSS SPI line that the EPCS device is connected to (0-9).   |
| filePath: | The path and file name of the FPGA data file.  |
| flags:    | Additional flags controlling the operation of this function. Specify QUICKUSB_EPCS_IGNORE_NCE to prevent the function from automatically asserting the nCE line of the FPGA at the beginning of the function are de-asserting it at the end of the function. |
| callback: | A pointer to a QPROGRESS_CALLBACK function which is called during the configuring process to report the percentage completed. This may be used to perform background processes and GUI updates while the configure process takes place.                      |
| tag:      | A PQVOID variable to store user data that is passed onto the callback function.  |

### Returns

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

### Notes

- If QUICKUSB\_EPCS\_IGNORE\_NCE is not specified, the nCE line of QuickUSB must be connected to the nCE line of the FPGA.
- If QUICKUSB\_EPCS\_IGNORE\_NCE is specified, then the nCE line of the FPGA must be asserted before calling this function to disable the FPGA allowing communication access to the EPCS device.
- Unless QUICKUSB\_EPCS\_SKIP\_ERASE is specified, this function will automatically perform a bulk erase prior to configuring the EPCS device.

### QuickUsbVerifyEpcs

#### **Purpose**

Verify that the data contained in the EPCS device matches that of the specified FPGA data file.

#### **Parameters**

|           |  |
|-----------|--|
| hDevice:  | A QHANDLE that was returned from a call to <a href="#">QuickUsbOpen</a> .  |
| nSS:      | The nSS SPI line that the EPCS device is connected to (0-9).   |
| filePath: | The path and file name of the FPGA data file.  |
| flags:    | Additional flags controlling the operation of this function. Specify QUICKUSB_EPCS_IGNORE_NCE to prevent the function from automatically asserting the nCE line of the FPGA at the beginning of the function are de-asserting it at the end of the function. |
| callback: | A pointer to a QPROGRESS_CALLBACK function which is called during the verify process to report the percentage completed. This may be used to perform background processes and GUI updates while the verify process takes place.                              |
| tag:      | A PQVOID variable to store user data that is passed onto the callback function.  |

#### **Returns**

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

#### **Notes**

- If QUICKUSB\_EPCS\_IGNORE\_NCE is not specified, the nCE line of QuickUSB must be connected to the nCE line of the FPGA.
- If QUICKUSB\_EPCS\_IGNORE\_NCE is specified, then the nCE line of the FPGA must be asserted before calling this function to disable the FPGA allowing communication access to the EPCS device.



## **QuickUsbEraseEpcs**

### **Purpose**

Perform a bulk erase action of the EPCS device to erase the entire device to 0xFF.

### **Parameters**

|           |  |
|-----------|--|
| hDevice:  | A QHANDLE that was returned from a call to <a href="#">QuickUsbOpen</a> .  |
| nSS:      | The nSS SPI line that the EPCS device is connected to (0-9).   |
| flags:    | Additional flags controlling the operation of this function. Specify QUICKUSB_EPCS_IGNORE_NCE to prevent the function from automatically asserting the nCE line of the FPGA at the beginning of the function are de-asserting it at the end of the function. |
| callback: | A pointer to a QPROGRESS_CALLBACK function which is called during the erase process to report the percentage completed. This may be used to perform background processes and GUI updates while the erase process takes place.                                |
| tag:      | A PQVOID variable to store user data that is passed onto the callback function.  |

### **Returns**

A QLONG that is non-zero on success, zero (0) on failure. Extra error information may be retrieved with a call to [QuickUsbGetLastError](#).

### **Notes**

- If QUICKUSB\_EPCS\_IGNORE\_NCE is not specified, the nCE line of QuickUSB must be connected to the nCE line of the FPGA.
- If QUICKUSB\_EPCS\_IGNORE\_NCE is specified, then the nCE line of the FPGA must be asserted before calling this function to disable the FPGA allowing communication access to the EPCS device.
- The amount of time it takes to perform a bulk erase depends on the EPCS device. EPCS1 devices take 3-6 seconds, EPCS4 devices take 5-10 seconds, EPCS16 device take 17-40 seconds, and EPCS64 devices take 68-160 seconds.
- The percentage complete reported by the callback function always indicates 0 percent. Since the erase operation does not take an exact amount of time to complete, the percentage complete is unknown and therefore not reported. The callback may still be used to perform background processes and GUI updates while the erase process takes place.