Overview

QuickUSB is capable of interfacing to a variety of different devices through use of a general-purpose interface (GPIF). The configuration of the GPIF is selected by choosing an IO Model to program into the EEPROM of a QuickUSB device. Determining which IO Model is best suited for a given application depends on many factors and needs to be made early in the design process to alleviate many headaches down the line. This application note discusses the available QuickUSB IO Models and how to choose the correct one for a given design.

The QuickUSB GPIF

QuickUSB essentially converts data transfers initiated by a host PC to a configurable high-speed parallel data interface. The interface is implemented using Cypress FX2's general-programmable interface (GPIF) and consists of a clock, a data bus, an address bus, control lines, and ready lines. The clock is used to synchronously clock over the data bus (except with asynchronous IO Models) and may be either sourced by the FX2 in the QuickUSB circuit (known as internally sourced) or sourced by external hardware (known as externally sourced). The data bus can be selected as either 16-bits wide (known as word-wide mode) or 8-bits wide (known as byte-wide mode) and carries the data moving over the GPIF. Use of the address bus is optional and, when enabled, holds the current address of each word of data (in word-wide mode) or byte of data (in byte-wide mode) as data moves over the GPIF. The control lines are outputs of the GPIF that indicate the current state of the GPIF such as read enable, write enable, full, and empty. The ready lines are inputs to the GPIF that indicate the state of the external interface hardware such as read enable, write enable, full, and empty. The specific use of the control lines and ready lines depend on the selected IO Model.

Data Transfers vs. Command Transfers

Both QuickUSB data transfers and command transfers use the GPIF. So what is the difference? Speed. Data transfers occur over USB bulk endpoints and can transfer large amounts of data very quickly. Command transfers occur over a USB control endpoint at a rate much lower than possible with data transfers. The benefit to using command transfers is that you have a way to perform simple reads and writes over USB on the GPIF without having to sacrifice the entire GPIF bus. An excellent example of a design that properly utilizes both the command bus and the data bus is when interfacing QuickUSB with an FPGA. The data bus is used to perform high-speed data transfers while the command bus is used to read and write registers in the FPGA (using the address lines to index specific registers). The GPIF indicates that it is performing either a data transfer or command transfer by setting the state of the CMD_DATA control line present on all IO Models that support command transfers.

Overview of the QuickUSB IO Models

Simple IO Model

The Simple IO Model should be regarded as the "go-to" IO Model and used whenever possible. Interfacing to and meeting the timing requirements of the Simple IO Model is easier than any other IO Model, and the Simple IO Model has the added benefit that it is the fastest and therefore will provide the highest data throughput out of all the IO Models.

The Simple IO Model transfers data over the GPIF as fast as the host can send/receive it without regard to the readiness of the target hardware, which is at most 48 MHz. This IO Model is suitable for target hardware that is always capable of reading or writing data when the host demands it. Since the GPIF does not need to know the current state of interface hardware no ready lines are used, and consequently the Simple IO Model is the fastest IO Model.

The Simple IO Model provides the following interface control lines:

- A read enable and a logically inverted read enable (REN/nREN) to indicate when the GPIF is performing data reads
- A write enable and a logically inverted write enable (WEN/nWEN) to indicate when the GPIF is performing writes
- A logically inverted output enable line (nOE) that indicates when it is safe for target hardware to drive the data bus for data reads since the data bus is bi-directional.

The clock (IFCLK) for the Simple IO Model may be either internally or externally sourced, and the address bus may optionally be used.

FIFOHS IO Model

The FIFO Handshake (FIFOHS) IO model is the second most common IO Model and is typically suited for applications where the Simple IO Model is not a good fit. Unlike the Simple IO Model, the FIFOHS IO Model uses ready pins that are inputs to the GPIF to allow target hardware indicate when it is ready for a data read or write to take place instead of assuming it is always ready. Because ready lines are used to handshake data transfers with target hardware, the FIFOHS IO Model is slightly slower than the Simple IO Model and therefore cannot reach the same data throughput rates that the Simple IO Model can. This limitation, however, can be alleviated if your design is capable of only transferring data in multiples of 512-byte (in High-Speed mode) or 64-byte (in Full-Speed mode) packets by using the BlockHS IO Model.

The FIFOHS IO Model is designed to interface to a synchronous or asynchronous FIFO IC, which may either be a physical FIFO IC, a FIFO present in an FPGA, or a FIFO contained within a more intelligent IC (microprocessor, DSP, etc.). Any interface that lends itself to transferring data using empty and full lines for handshaking should use this IO Model.

The FIFO IO Model provides the following interface control and ready lines:

- A read enable and a logically inverted read enable (REN/nREN) to indicate when the GPIF is performing data reads. This control line toggles on and off for every word of data (in word-wide mode) or byte of data (in byte-wide mode). Doing so allows the control signal to be used either as a straightforward read enable line synchronously clocking out data with IFCLK, or used as an asynchronous strobe clocking out data without the need for IFCLK.
- A write enable and a logically inverted write enable (WEN/nWEN) to indicate when the GPIF is performing data writes. This control line toggles on and off for every word of data (in word-wide mode) or byte of data (in byte-wide mode). Doing so allows the control signal to be used either as a straightforward write enable line synchronously clocking out data with IFCLK, or used as an asynchronous strobe clocking out data without the need for IFCLK.
- A logically inverted output enable line (nOE) that indicates when it is safe for target hardware to drive the data bus for data reads since the data bus is bi-directional.

- A logically inverted full signal (nFULL) used by the GPIF to determine when the target hardware's FIFO buffer is full and writes should be suspended. This ready signal is an input to the GPIF and output of the target hardware's FIFO buffer.
- A logically inverted empty signal (nEMPTY) used by the GPIF to determine when the target hardware's FIFO buffer is empty and reads should be suspended. This ready signal is an input to the GPIF and output of the target hardware's FIFO buffer.

The clock (IFCLK) for the FIFOHS IO Model may be either internally or externally sourced, and the address bus may optionally be used.

BlockHS IO Model

The BlockHS IO Model has essentially the same interface as the FIFOHS IO Model. The only differences are:

1. All data requests made through the QuickUSB Software API must be multiples of 512 bytes (in High-Speed mode) or 64 bytes (in Full-Speed mode). This means that when using the QuickUSB Software API and calling a read data or write data function, you must always specify a valid request length. For example, in High-Speed mode some valid transfer sizes are 512 bytes, 1 KB, 64 KB, 1.5 MB, etc.
2. You cannot interface to asynchronous FIFOs since the REN/nREN and WEN/nWEN lines may no longer be used as data strobes.

If you interface to synchronous FIFOs and meet the requirements of the data request length then the FIFOHS and BlockHS IO Models may be used interchangeably, but the BlockHS IO Model will offer better data throughput rates—close to that of the Simple IO Model. For such designs, it is generally a good idea to first implement your design using the FIFOHS IO Model and get everything working properly, then switch over to using the BlockHS IO Model to increase your system throughput.

FullHS IO Model

The FullHS IO Model is ideal for interfacing to device that has a slow or variable transfer time. It is very similar to the Simple IO Model except that the GPIF uses a ready line to determine when it is okay to perform data transfers.

The FullHS IO Model provides the following interface control and ready lines:

- A read enable and a logically inverted read enable (REN/nREN) to indicate when the GPIF is performing data reads
- A write enable and a logically inverted write enable (WEN/nWEN) to indicate when the GPIF is performing writes
- A ready signal (READY) that is checked by the GPIF before transferring each word (in word-wide mode) or byte (in byte-wide mode) over the GPIF to ensure that the target hardware is ready for the transaction.

The clock (IFCLK) for the FullHS IO Model may be either internally or externally sourced, and the address bus may optionally be used.

Pipeline IO Model

The Pipeline IO Model has essentially the same interface as the Simple IO Model except that data is transferred over the GPIF one clock cycle after REN or WEN transitions to high. This allows target

hardware to have one additional clock cycle to prepare for the upcoming data transfer. If you need more than a one-cycle delay, you can set the number of pipeline cycle delays required by writing the lower byte of the SETTING_SLAVEFIFOFLAGS setting.

The clock (IFCLK) for the Pipeline IO Model may be either internally or externally sourced, and the address bus may optionally be used.

Synchronous Slave FIFO IO Model

The Synchronous Slave FIFO IO Model is enabled by programming your device with the Simple IO Model firmware and setting SETTING_FIFO_CONFIG bits 1 and 0 to 1 (high) and bit 3 to 0 (low). With this IO Model, external hardware directly controls the USB endpoint FIFOs contained directly in the Cypress FX2 and data transfers are synchronous to IFCLK. Unless you have very specific requirements that direct you to using this IO Model, it is recommended that you choose one of the other IO Models as the Slave FIFO IO Models are more difficult to interface to, more difficult to meet the timing requirements of, and offer lower data throughput rates than other IO Models.

The Synchronous Slave FIFO IO Model provides the following interface control and ready lines:

- A logically inverted full signal (nFULL) outputted by the GPIF to indicate when the endpoint buffer in the FX2 is full and writes should be suspended or data corruption will occur.
- A logically inverted empty signal (nEMPTY) outputted by the GPIF to indicate when the endpoint buffer in the FX2 is empty and reads should be suspended or data corruption will occur.
- Two FIFO address lines present on PA[5:4] indicating which endpoint buffer should be accessed by the FX2. These two pins must be configured as inputs to the FX2. QuickUSB currently implements two USB bulk endpoints: one for performing data reads located at endpoint 6 (EP6) and another for performing data writes on endpoint 2 (EP2). When using the Synchronous Slave FIFO IO Model you must indicate to the GPIF which endpoint buffer you would like to access by driving the values of PA[5:4].
- A logically inverted slave read line (nSLRD) used by the FX2 to determine when reads of the endpoint buffer should be enabled.
- A logically inverted slave write line (nSLWR) used by the FX2 to determine when writes to the endpoint buffer should be enabled.
- A logically inverted slave output enable line (nSLOE) use by the FX2 to determine when it is safe for the FX2 to drive the data bus for slave reads since the data bus is bi-directional. This signal is present on PA2 and must be configured as an input to the FX2.
- A logically inverted packet end signal used by the FX2 during slave writes to the endpoint buffer to determine the end of a data transfer. This signal is present on PA6 and must be configured as an input to the FX2.
- A logically inverted slave chip select line that used by the FX2 to determine when to allow access to the endpoint buffer. This signal is present on PA7 and must be configured as an input to the FX2.

The clock (IFCLK) for the Synchronous Slave FIFO IO Model may be either internally or externally sourced. The address bus is not used with this IO Model. Also note that the Slave FIFO IO Models do not support command data transfers.

Asynchronous Slave FIFO IO Model

The Asynchronous Slave FIFO IO Model is enabled by programming your device with the Simple IO Model firmware and setting SETTING_FIFO_CONFIG bits 1 and 0 to 1 (high) and bit 3 to 1 (high). With this IO Model, external hardware directly controls the USB endpoint FIFOs contained directly in the Cypress FX2 and data transfers are clocked use slave read and slave write signal strobes. This IO Model is very similar to the Synchronous Slave FIFO IO Model except that instead of synchronously clocking data transactions with IFCLK, the slave read and slave write lines strobe out data. Unless you have very specific requirements that direct you to using this IO Model, it is recommended that you choose one of the other IO Models as the Slave FIFO IO Models are more difficult to interface to, more difficult to meet the timing requirements of, and offer lower data throughput rates than other IO Models.

The clock (IFCLK) for the Asynchronous Slave FIFO IO Model may be either internally or externally sourced. The address bus is not used with this IO Model. Also note that the Slave FIFO IO Models do not support command data transfers.

Application Examples

Interfacing QuickUSB with an FPGA

FPGAs are easy to interface with QuickUSB since FPGAs by definition they allow for a lot of flexibility in a design. Many FPGAs even contain on chip memories such as FIFOs so that external FIFO ICs do not need to be included in designs that require buffering data between QuickUSB and an FPGA.

Because of the flexibility of FPGAs, an FPGA is able to interface to QuickUSB using any of the QuickUSB IO Models. Typically, the quickest and easiest way to interface to an FPGA is to use either the Simple IO Model or FIFOHS/BlockHS IO Model, depending on if data in the design needs to be buffered and if any handshaking needs to occur with the FPGA.

Interfacing QuickUSB with a FIFO Buffer

The FIFOHS and BlockHS IO Models are designed to make interfacing directly to either a synchronous or an asynchronous FIFO buffer easy and without the need for any additional interfacing hardware. Using FIFO buffers between QuickUSB and other target hardware is generally a good idea as it separates data sources/sinks and smoothes out the bursts of USB data, providing for a more stable design with increased data throughput.

Interfacing QuickUSB with a Microprocessor

Interfacing QuickUSB with a microprocessor requires a little more work and care in the design. This is because microprocessors have their own dedicated data and address bus for which they are the master of and QuickUSB has its own data and address bus for which it is the master of. The most reliable way to interface the two is to place a FIFO buffers between them. This means that QuickUSB only needs to interface to FIFO buffers, which is pretty straight-forward using the FIFOHS and BlockHS IO Models, and the microprocessor only needs to interface with FIFO buffers. Keep in mind that in order to perform both data reads and data writes you will need two interface FIFO buffers—one for data reads and one for data writes. This design works best because USB is a host-driven protocol, which means USB data reads and data writes are always initiated by the host PC. When a read or write request is issued on the host PC, it makes its way down to the QuickUSB firmware where the data is transferred between the FX2 and FIFO buffer over the GPIF. While this is occurring, the microprocessor may

independently check the FIFO buffers to determine if data is ready to be read in or written out. An important factor that will influence data throughput in this design will be the depth of the FIFO buffers. You must make sure to select a depth that will allow the QuickUSB GPIF to read and write the FIFOs with minimal halting because they are either empty or full. This is a balance between how quickly the microprocessor can sink/source data to the FIFOs and the desired USB data throughput.

You may also interface to microprocessor using one of the Slave FIFO IO Models. Doing so allows you to directly access the endpoint FIFOs present in the FX2 allowing you to not have to design in additional FIFO buffers between the GPIF and the microprocessor. This implementation, however, should be avoided if possible because the Slave FIFO IO Models are more difficult to interface with and meet the timing requirements of than other IO Models, and as a result offer lower data throughput than other implementations. The depth of the endpoint FIFOs are also limited and can degrade the performance of the overall design. Be sure that you verify that you can achieve your desired throughput rates using one of the Slave FIFO IO Models before committing to a design.

Bitwise Systems
6489 Calle Real, Suite E
Goleta, CA 93117
V (805) 683-6469
F (805) 683-4833
info@bitwisesys.com