WP121 (v1.0) September 5, 2000

Benefits of Using Xilinx FPGAs with MIPS® Microprocessors

Author: Nick Price

Summary

This white paper discusses the benefits of placing Xilinx FPGAs next to MIPS microprocessors to increase system performance.

Background

With FPGA design changes being so simple compared to ASICs, FPGAs have become a valuable weapon in a designer's arsenal. FPGAs are ideal for fast time-to-market designs, or designs where upgrades/modifications maybe required.

As FPGAs become more of a system component, the need to interface with different devices becomes increasingly important. One of the more prevalent devices for FPGAs to interface with is the microprocessor.

Some common uses of an FPGA when interfaced with a processor are as follows:

- Providing I/O standard conversions
- Performing real time data manipulations
- Optimizing processor intensive functions
- Interfacing processor to backplane, or
- A combination of above

General Discussion

MIPS processors are very successful in the following markets: consumer electronics, digital entertainment, mobile computing, office automation, communications, and networking.

Figure 1 shows a simple example of a common MIPS system where each block is a separate element on the board.

As FPGAs become larger and more cost effective, it is natural that designers will use them to interface processors and FPGAs. The FPGA's speed, versatility, and flexibility will attract many designers. Some common FPGA uses when interfaced to a MIPS processor are as follows.

- Performing custom and/or complex CPU functions
- Interfacing the processor to a bus structure that requires data manipulation like PCI
- Interfacing to devices with different I/O standards
- Performing real time data manipulation, or
- Combination of the above situations.

Custom CPU Functions

Using a processor algorithm to perform complex functions, such as DSP functions (e.g., Fourier Transforms) requires multiple serial mathematical calculations (multiply and add) with each calculation requiring many clock cycles.

An FPGA can be designed to perform Fast Fourier Transforms (FFTs) in parallel, providing a hardware implementation that is many magnitudes faster than the serial processor algorithm.

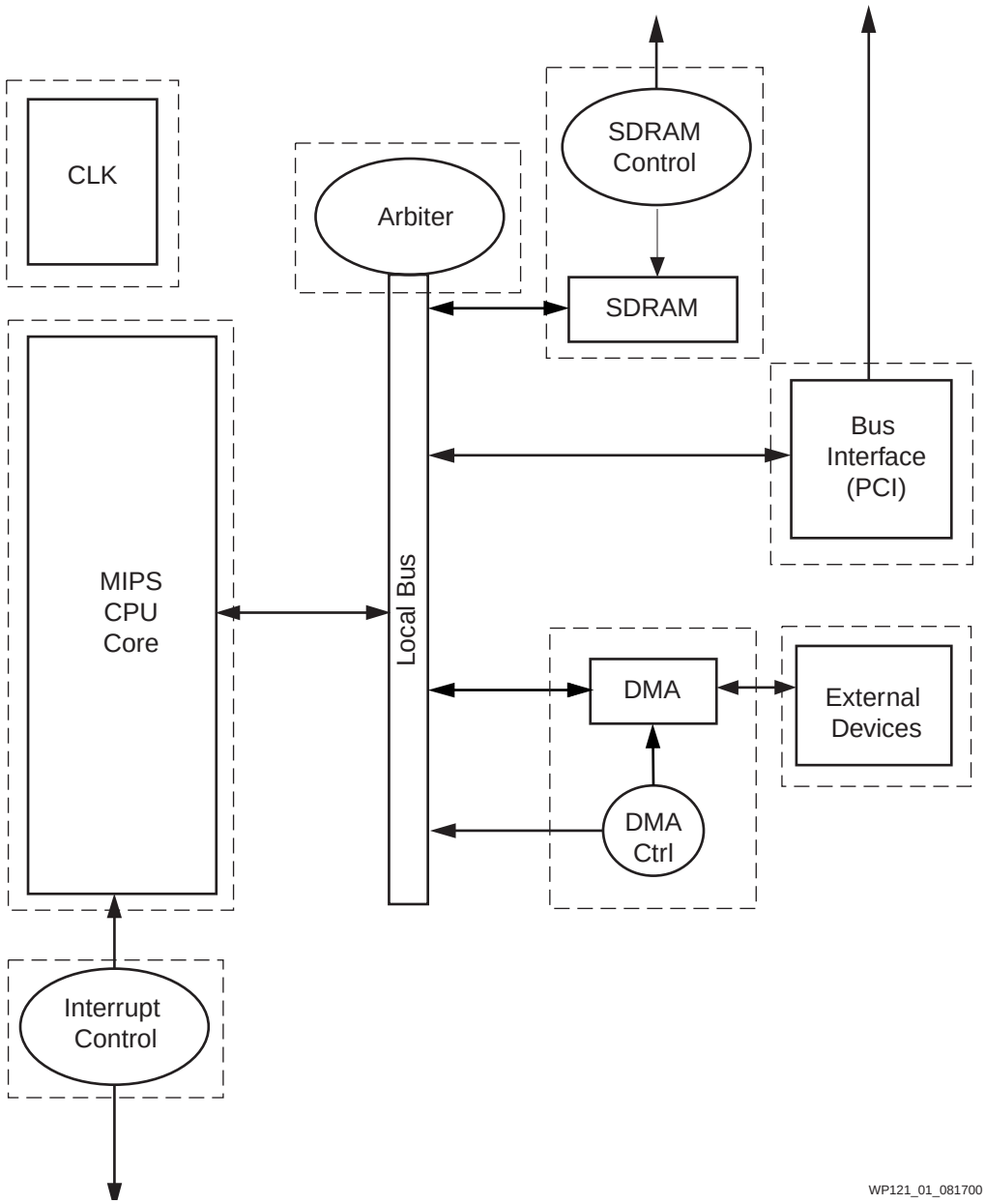

Figure 1: **Simple MIPS System**

Interfacing Bus Structures

FPGAs provide an excellent way to interface a processor to different bus structures like PCI. The Xilinx PCI core is fully verified and available in 32/33 MHz and 64/66 versions.

Using an FPGA with a Xilinx PCI core allows efficient interaction with the PCI bus, because the PCI core allows queuing of data to and from the PCI bus using First In First Outs (FIFOs) to avoid or reduce bus system idle time. A second advantage of using a Xilinx PCI core is that the core is able to interface with the PCI bus using zero wait states, achieving maximum throughput with the bus.

Interfacing Different I/O Standards

Xilinx FPGAs can be configured to interface to up to 20 different I/O standards, making it an excellent tool for converting between multiple standards.

For instance, one can interface the MIPS processor (using voltage level signal recognition) to Low Voltage Differential Signal (LVDS) via the Xilinx FPGA. One can have different banks configured to different voltage standards, allowing an easy transition between them. Xilinx FPGAs can support these different standards simultaneously.

Real-Time Data Manipulation

FPGAs can also be used to provide real-time data manipulation. They make an excellent tool for encrypting and decrypting data in real time. By placing the FPGA with the encryption/ decryption design in the processor's path, all data to and from the processor will be manipulated.

Combination of Above Situations

In most cases, a combination of the above situations occurs. For instance, part of the FPGA can be dedicated to performing special functions, while another part interacts with the PCI bus.

Figure 2 shows a simple MIPS system utilizing an FPGA for special bus functions and the PCI core.

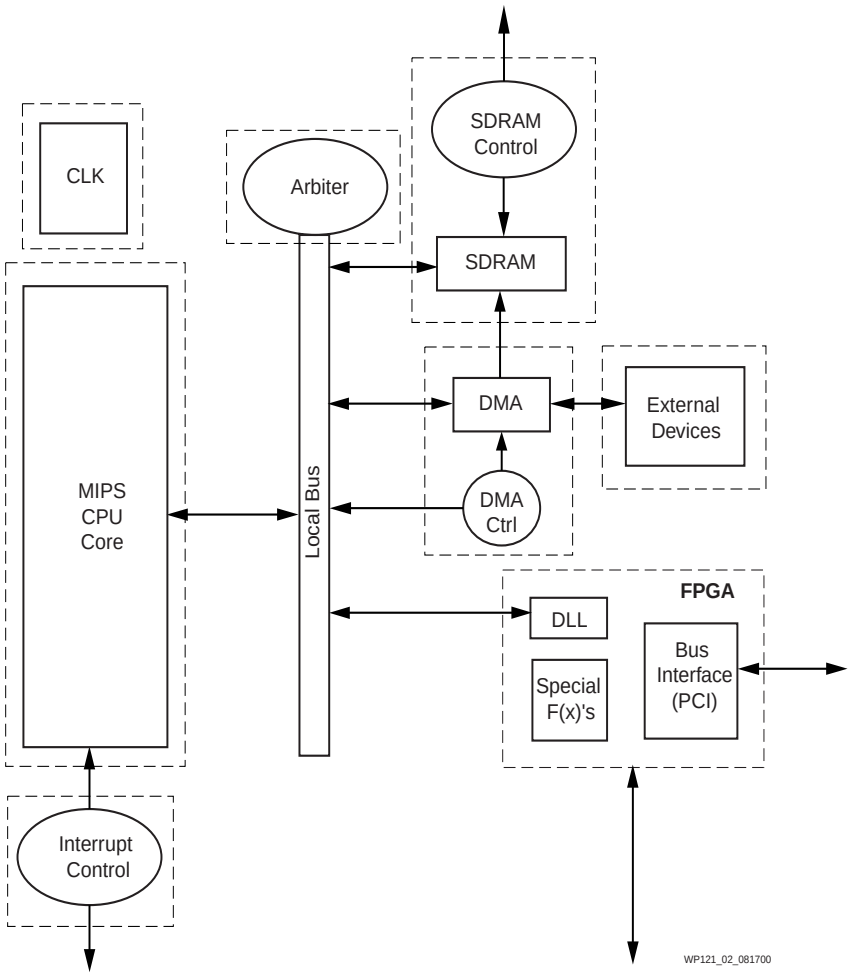

Figure 2: **Simple MIPS System Using an FPGA**

FPGAs are so flexible that accommodating different designs and function (possibly using different clocks) in a single FPGA is possible. Figure 3 shows a design that uses an FPGA for a majority of the system.

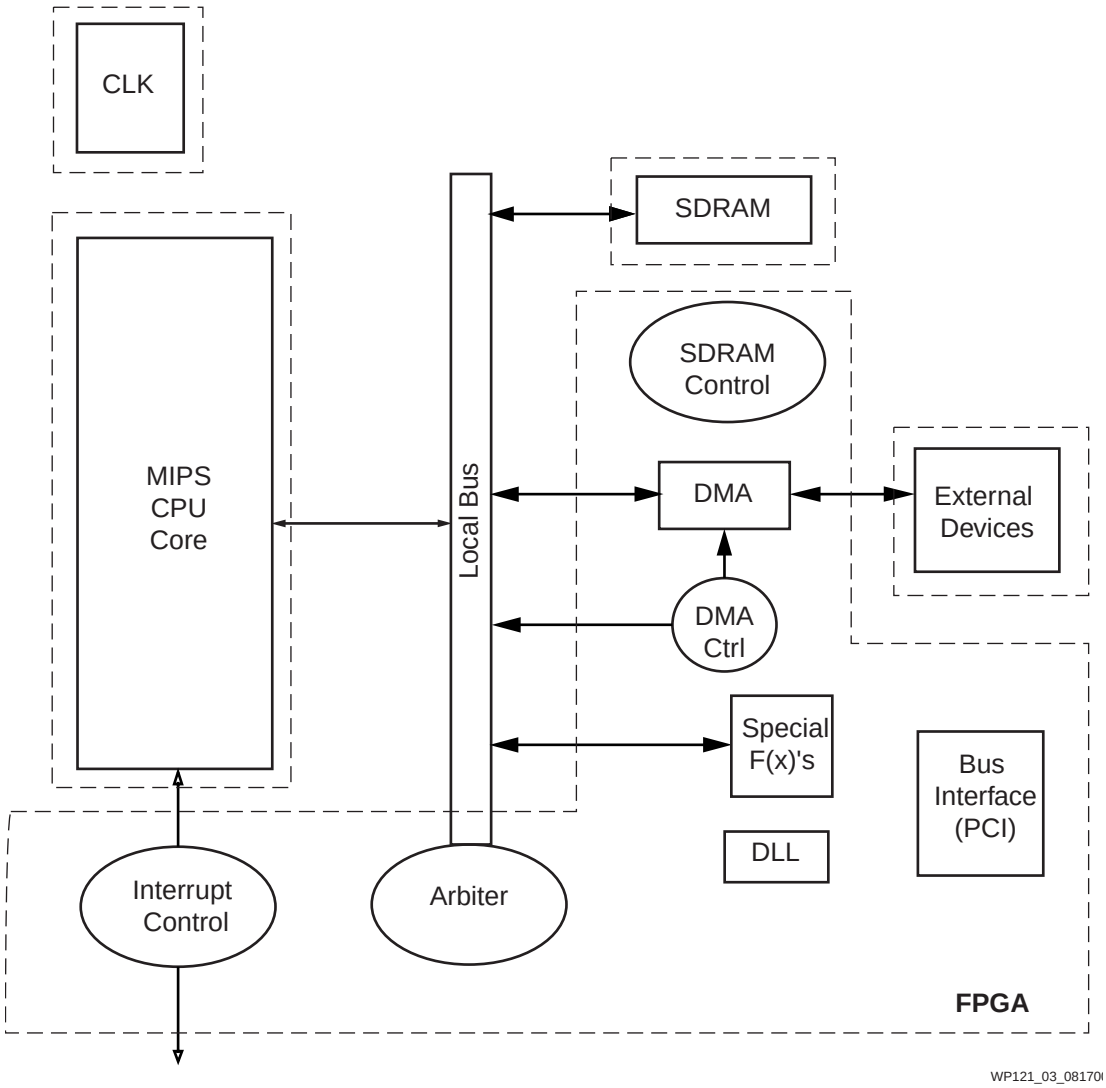

Figure 3: **MIPS System with Maximum FPGA Usage**

FPGA Design Considerations

General

When interfacing a MIPS processor to an FPGA, one of the primary questions is what will the FPGA do with the MIPS data? Is it going to manipulate data and return a value? Is it going to be a common memory medium that both will access independently? Or, is it going to be a required medium to another interface, such as a PCI bus?

Straight Data Manipulation

If the FPGA is only required to provide enhanced data manipulation, custom commands, or functions, such as providing an FFT for the processor to access as a special command, then, a simple state machine that looks for the appropriate MIPS command, feeds the data into the appropriate module (FFT), and returns the result onto the data bus when completed, is all that as needed. Figure 4 illustrates simple data manipulation.

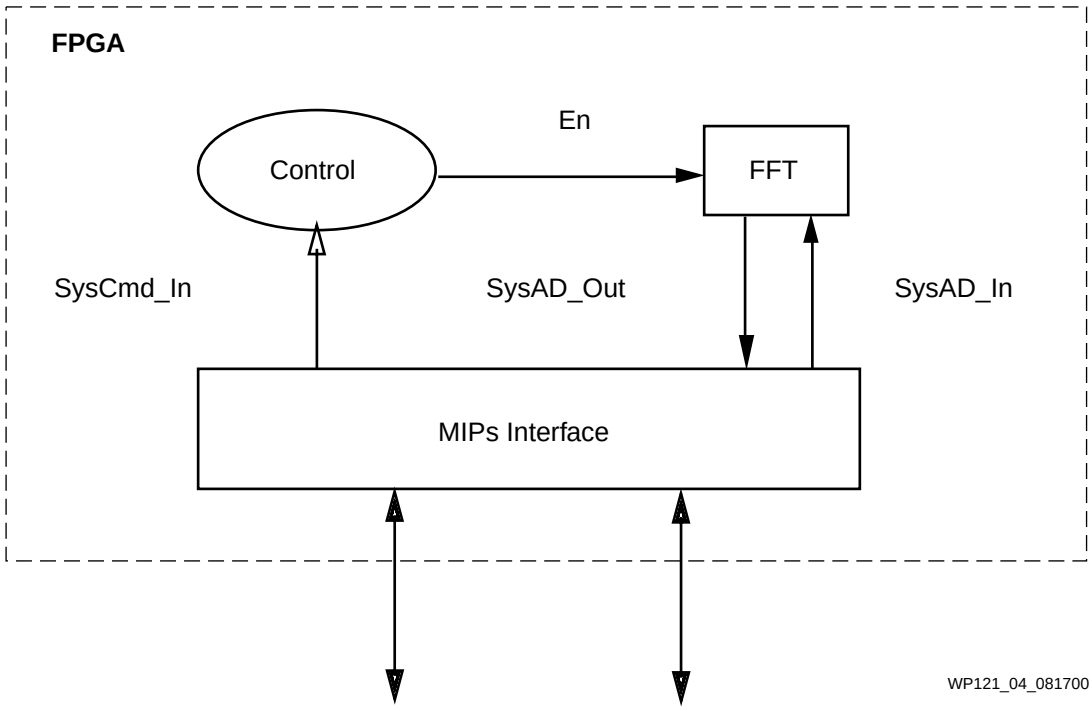

Figure 4: **Simple Data Manipulation**

Interacting Between Different Media

If interfacing to a specific bus system is required, such as a PCI bus, then a bridge design is required. Generally when interfacing two asynchronous media together, the bridge design will primarily consist of an asynchronous FIFO system to allow the temporary storage of a transmission between the two media. Figure 5 illustrates simple FIFO interface.

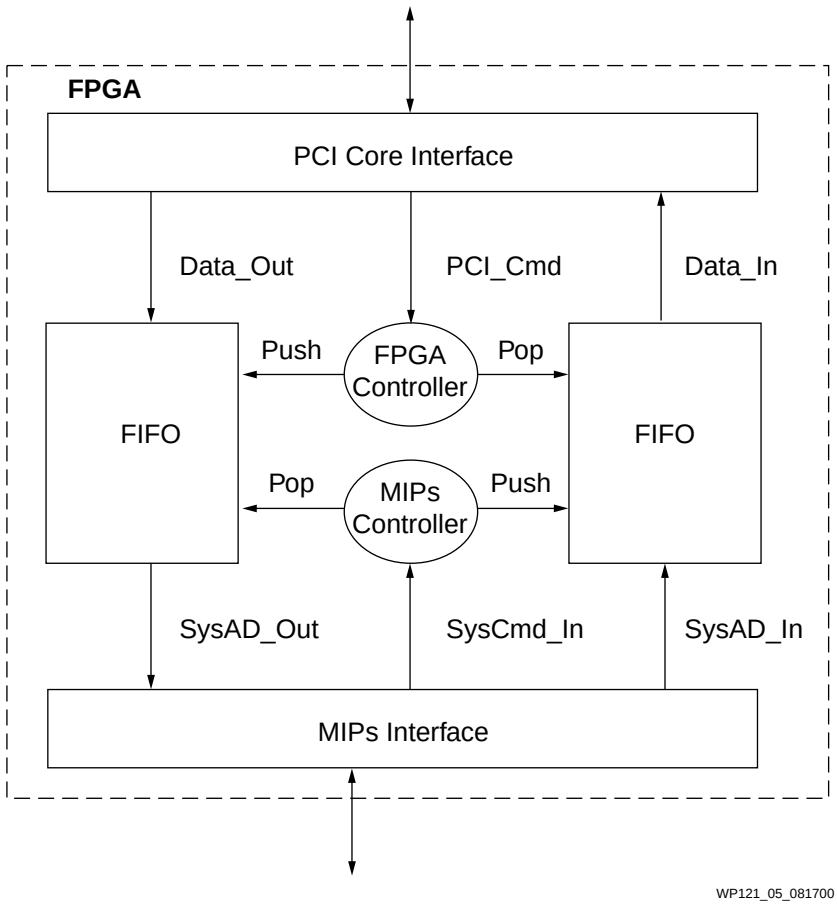

Figure 5: **Simple FIFO Interface Usage**

This type of interface lends itself well to the use of Dual Port Block RAMs. Xilinx does provide customizable IP Asynchronous FIFO cores for designers via CoreGen. If one needs to create a custom FIFO, refer to XAPP131: 170 MHz Synchronous and Asynchronous FIFOs Using the Virtex Block SelectRAM+ Features at , and XAPP205: Data-Width Conversion FIFOs using Virtex Block SelectRAM Memory at **http://www.xilinx.com/xapp/xapp205.pdf**.

Common Memory

Another common interface is to have both systems read and write to common memory. This allows both systems to access the memory in a non-linear fashion, i.e., just access a common memory block. Block Select RAM for Virtex™, Virtex-E, and Spartan®-II come in 4-Kbit chunks. Figure 6 illustrates simple communal memory usage.

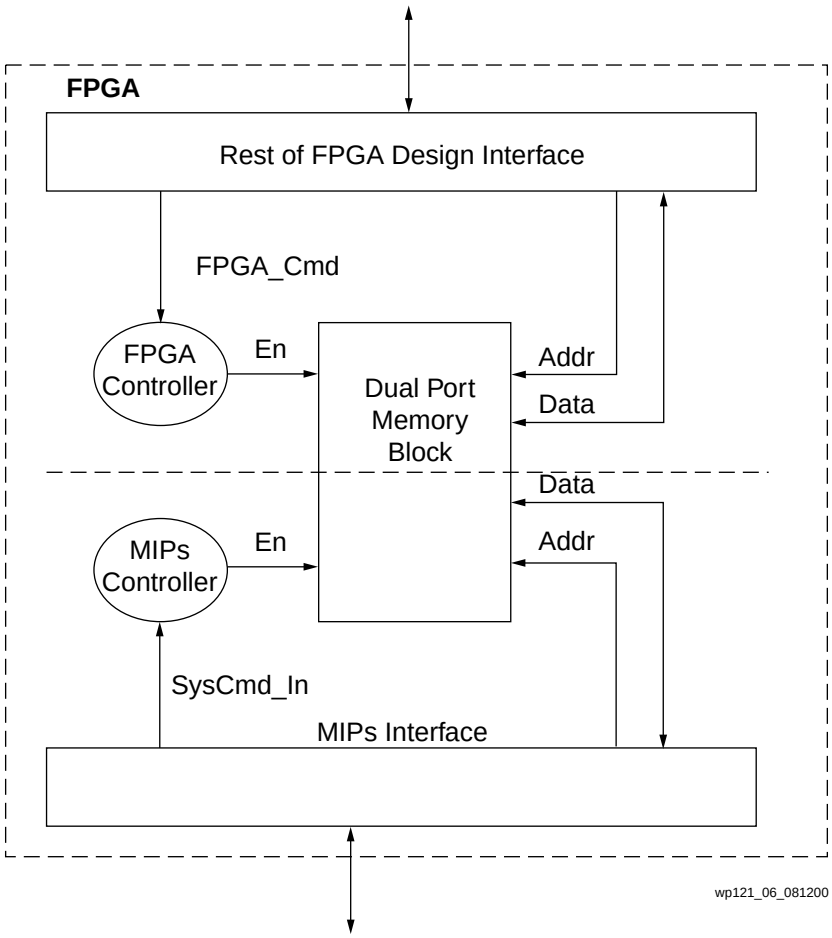

Figure 6: **Simple Communal Memory Usage**

Xilinx provides a simple interface for creating custom memory using Block RAM with CoreGen. If additional manipulation is required, refer to XAPP130: Using the Virtex Block SelectRAM+ Features at **lhttp://www.xilinx.com/xapp/xapp205.pdf**.

Interface Connections

The remainder of this paper will focus on a simple system that directly interfaces the MIPS processor and the FPGA. A simple example of such a system can be seen in Figure 7. This is also the basic interface structure used by the reference design provided in the appendix.

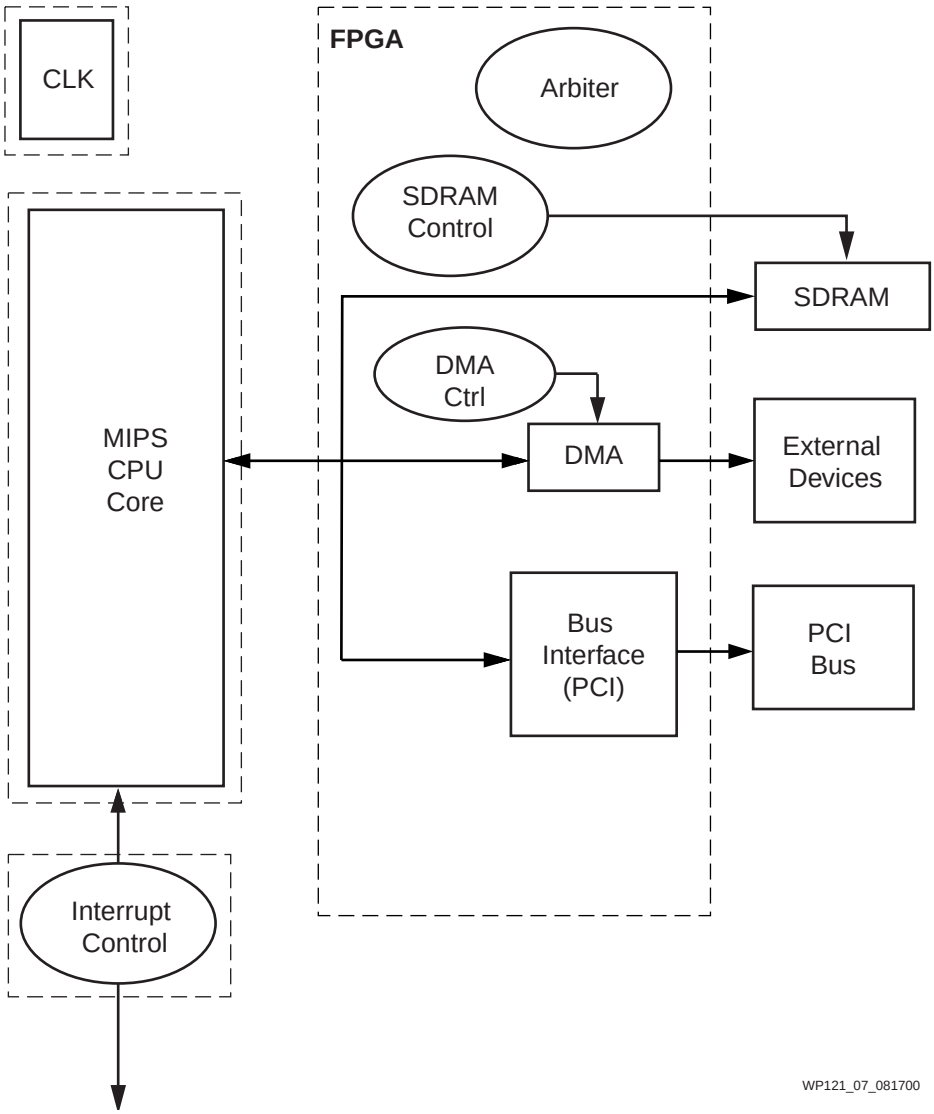

Figure 7: **Direct MIPS to FPGA Example Interface**

Figure 8 shows a more detailed example of how to directly connect a Xilinx FPGA to a QED 7000 MIPS microprocessor. It uses the standard MIPS R4000 interface and can be used as a starting point for most designs. This interface uses the most common portions to allow for greatest portability amongst the R4000 MIPS family. Notice that in this example there are no other devices. No interrupt circuitry is required and, therefore, is ignored.

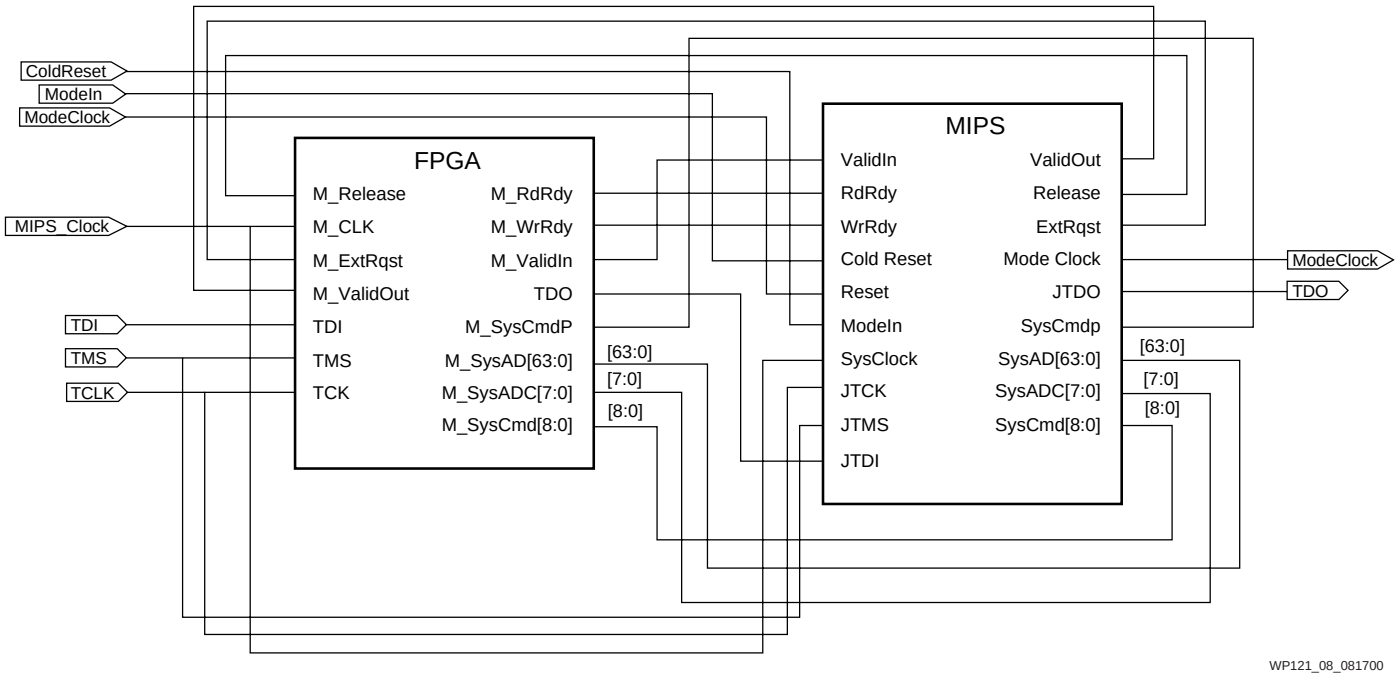

Figure 8: **FPGA to MIPS Interface Example**

Achieving Maximum I/O Throughput

General

Achieving maximum I/O throughput is beneficial to speed-sensitive systems. I/O speed tends to be the bottleneck for most systems; achieving maximum I/O throughput is generally desired for today's high-speed systems.

There are two areas of focus when achieving maximum I/O throughput: chip issues and board issues. For chips, the main focus is to get on and off the chip as quickly as possible. For the board, the focus is to provide a clean signal from any chip to any other chip as quickly as possible.

Techniques to Aid I/O Throughput

For these two devices, there are two different styles for optimizing the I/O throughputs. For the MIPS processor, one can only adjust the MIPS interface settings by changing the configuration mode bits to the most optimal setting. On the other hand, FPGAs are very flexible, and there are many ways to improve their performance with design techniques and I/O standards.

MIPS Settings to Aid I/O Throughput

Maximizing the MIPS I/O speed can be done by using the appropriate Mode settings, using optimal clock operating conditions, and using the MIPS phase locked loop (PLL) to reduce clock delay within the MIPS processor. Since maximum clock speed and operating conditions are different for different MIPS processors, refer to the appropriate processor data sheet and user manual.

To maximize the I/O throughput, use the following Mode settings:

- Set write-back data rate to continuous (DDDD). FPGA interfaces directly to the MIPS. No need to worry about creating bus IDLE time.
- Set Master/SysClock multiplication to the optimal setting.
- Set output drive strength to 100 percent (fastest).
- Set external bus width to 64 bit (Max data per cycle).

Notes:

1. Check MIPS data sheet for correct mode bits to set while loading mode configuration.

Write-Back Data Rate Setting

The write-back data rate is a setting on the MIPS device that controls data/idle pattern on the bus. This allows one to control the maximum data rate to/from the MIPS processor from one double word per cycle to one double word per four cycles. To achieve maximum data throughput, having a continuous write back data is desired.

Detailed information can found in the system interface protocol section of the user manual at **http://www.qedinc.com**.

Master/System Clock Setting

The goal is to have a system with the highest system frequency. The MIPS devices drive the master clock used internally by multiplying the system frequency by a value provided in the mode settings. At present, the fastest system speed available for a QED device is 125 MHz. This device also uses an internal clock speed of 250 MHz. Therefore, the master/system clock needs to be set to two.

As newer devices arrive, this ratio (master/system clock) for achieving the greatest I/O throughput may change.

Output Drive Strength

To achieve the fastest I/O speed, set the drive strength of the MIPS processor to 100 percent.

Maximize Bus Widths

To maximize data throughput, use the largest bus settings available (currently 64-bit). This will allow the most data to be processed per clock cycle.

FPGA Techniques to Aid I/O Throughput

Some common FPGA design techniques to increase I/O speed are as follows:

- Use a DLL to virtually eliminate clock delay for registers that deal with I/O.
- Register the I/Os to get the minimum distance between the register and the I/O pad.
- Use Select I/O to increase the I/O Port speed.

An example of these techniques is seen in the reference design, which demonstrates the FPGA techniques discussed to maximize MIPS to FPGA throughput.

Using Internal DLL

To increase I/O data throughput, use a DLL to eliminate clock delay among registers communicating with I/O. This gives the user more of the clock period to utilize, because there is less than 25 ps clock delay.

For more detailed information, refer to XAPP 132 Using the Virtex Delay-Locked Loop, **http://www.xilinx.com/xapp/xapp132.pdf**

Registering I/O

Another technique used by designers to achieve higher throughput is to utilize the Input-Output Block (IOB) Registers. These registers are dedicated for the I/Os that are placed on the edge of the die to be as close to the I/O pad as possible. By using I/O registers, one guarantees the shortest path between an I/O and a register. Again aiding to achieve the best I/O speed.

Refer to Development System Reference Guide at **http://toolbox.xilinx.com/docsan/3_1i/** for information on how to register the I/O.

Using Select I/O and Drive Strength

If using the default I/O standard (LVTTL) does not produce adequate results (even in fast mode), try increasing the drive strength or using a different I/O standard. Increasing the drive will help increase the I/O speeds by adding more current/power to the I/O. If increasing the drive strength is not adequate, a different I/O standard may be appropriate.

For the Virtex families, there are a large variety of different I/O standards available. A listing of I/O available standards can be found in the Xilinx data book at http://support.xilinx.com/partinfo/databook.htm. By selecting different I/O standards, each with its own benefits, one can have the I/O behave differently. For achieving optimal I/O throughput, choosing an HSTL IV as the I/O standard greatly reduces the time on and off the chip, thereby, achieving a very fast I/O throughput.

For more information on using Select I/O, refer to XAPP133: Using the Virtex Select I/O Resource at http://support.xilinx.com/xapp/xapp133.pdf.

Board Techniques to Aid I/O Throughput

General Board Techniques

General board techniques to aid in I/O throughput include the following:

- Placing the MIPS and FPGA as close as possible
- Reducing trace length
- Using adequate decoupling capacitors to prevent brown outs and reduce ground bounce
- Using ample V_{CC} and Ground pairs to reduce ground bounce

FPGA Board Techniques

In addition, with FPGAs try to keep all data/address lines running horizontally and all control and clock lines vertically into the chip. This is because the FPGA's routing is set up to utilize data in a horizontal fashion, by having routes that directly connect the data of a Configuration Logic Block (CLB) to the CLB horizontally beside it. The FPGA also has dedicated routing that runs in vertical columns (long lines). These long lines are ideal for control signals like clock enables or clocks that need to control large quantities of data.

To reduce ground bounce, try to spread or separate bus data lines, preferably while maintaining groupings around ground connections. Spreading the signal will help to spread out the bouncing ground around the chip. Placing the signals close to ground connections will help to absorb or lessen the effects of the ground bounce.

To reduce board skew, use a second DLL in the FPGA to de-skew the board clock. Try to keep all the I/O and feedback trace lines equal in length to ensure proper clock de-skewing. For more detailed information, refer to XAPP132 Using the Virtex Delay-Locked Loop,

http://www.xilinx.com/xapp/xapp132.pdf

To run at optimal performance, make sure the device loading on the local bus is within specification. If there is too much loading from too many devices, one will have to slow down the system clock frequency or add latency. One way to reduce bus loading is to interface multiple components (memory and PCI) via the FPGA, thereby, only having loading from the FPGA rather than from all the other devices.

Recommendations

When interfacing a MIPS processor to a Xilinx FPGA, I/O throughput is generally the system bottleneck. Here is a list of recommendations to help maximize the I/O throughput between the MIPS processor and the Xilinx FPGA:

- MIPS - Optimal I/O mode
- FPGA - Registered I/O
- FPGA - Select I/O
- FPGA - DLL (internal and externally)

- Board - Place chips as close together
- Board - Enough Decoupling capacitors

By using the above techniques, one should be able to utilize the maximum I/O throughput available from the MIPS processor with an FPGA.

References

XAPP130: Using the Virtex Block SelectRAM+ Features at **http://www.xilinx.com/xapp/xapp130.pdf**

XAPP131: 170 MHz Synchronous and Asynchronous FIFOs Using the Virtex Block SelectRAM+ Features at **http://www.xilinx.com/xapp/xapp131.pdf**

XAPP132: Using the Virtex Delay-Locked Loop,**http://www.xilinx.com/xapp/xapp132.pdf**

- XAPP133: Using the Virtex Select I/O Resource at **http://www.xilinx.com/xapp/xapp133.pdf**
  XAPP205: Data-Width Conversion FIFOs using Virtex Block SelectRAM Memory at **http://www.xilinx.com/xapp/xapp205.pdf**
- Development System Reference Guide at **http://toolbox.xilinx.com/docsan/3_1i**
- Xilinx data book at **http://support.xilinx.com/partinfo/databook.htm**
- QED: RM7000/RM7000A Data Sheet at **http://www.qedinc.com**
- MIPS web site: **http://www.mips.com/**

Appendix

Example Reference Design

Link to MIPS to Virtex-E reference design.TBD

MIPS Line Card Providers

http://www.mips.com/Documentation/LineCard.pdf

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
09/05/00	1.0	Initial Xilinx release.