SPEEDY BUS MASTERING PCI EXPRESS

Ray Bittner

Microsoft Research
One Microsoft Way
Redmond, WA 98072
email: raybit@microsoft.com

ABSTRACT

PCI Express is a ubiquitous bus interface providing the highest bandwidth connection in the PC platform. Sadly, support for it in FPGAs is limited and/or expensive. The Speedy PCIe core addresses this problem by bridging the gap from the bare bones interface to a user friendly, high performance design. This paper describes some of the fundamental design challenges and how they were addressed as well as giving detailed results. The hardware and software source code are available for free download from [12].

1. INTRODUCTION

FPGAs have been shown to provide significant computational acceleration for many problems of interest. These improved speeds often shift the bottleneck from computation to data movement as the IO subsystem is unable to keep pace with consumption. As a result, FPGA researchers are often forced into using a slow interface to download data to local memory and reporting performance numbers based on the relatively fast accesses that are available there. When this is not an option, great pains must be taken to create high speed interfaces to facilitate data throughput. For example, our embedded graphics project is in need of such a solution [11].

PCI Express (PCIe) is the fastest interface available to facilitate PC/FPGA communications. FPGA vendors have offered PCIe cores to harness this power for some time, but the cores are too rudimentary in nature to be of immediate use. For example, the Xilinx [2] and Altera [3] cores provide a split transmit (TX)/receive (RX) interface to the PCIe bus. However, the user must still encode or decode data to form packets while obeying the many rules of the PCIe specification for addressing, packet size, etc [1]. Once packets have been created, there are still several design hurdles and a great deal of code that must be written in order to turn these primitive interfaces into a useful core as discussed in [6][5].

Commercial solutions to this problem exist [7][8][9], and while custom support is available, they are costly. What is needed is a freely available, complete solution that can transfer data directly to/from arbitrary user data buffers at full speed while providing a user friendly bus access model to the hardware designer. This paper discusses difficulties and insights related to the implementation of the PCIe protocol on the PC platform in the form of the Speedy PCIe core and offers it as a solution or starting point for future research.

The Speedy PCIe core delivers a general purpose solution that solves the problems of high speed Direct Memory Access (DMA) while offering an interface that is generic and adaptable for a large number of applications. The complete solution consists of a Windows 32/64-bit driver, FPGA Verilog and a C++ test application. The following section contains a discussion of the hurdles that were encountered and overcome in its implementation.

2. CHALLENGES

There are several key insights that must be addressed in order to maintain high bus utilization and achieve full bandwidth transfers.

Parallel TX/RX Control Paths - The PCIe core interfaces provided from Xilinx [2] and Altera [3] both operate using split receive (RX) and transmit (TX) interfaces. These interfaces act independently and in parallel, though they are linked functionally (Fig. 1). For example, a Programmed I/O (PIO) read begins with a packet on the RX interface and is completed with one or more packets on the TX interface. Likewise, a bus mastering read operation begins with a packet on the TX interface and completes with one or more packets on the RX interface.

The parallel operation of the TX and RX paths must be maintained, leading to the creation of parallel state machines, each serving their respective interfaces. Bus master read operations are the best example of this, as the Speedy Core must maintain parallel master read requests outstanding on the TX interface as read data returns on the RX interface. The number of master read requests that may be in flight is determined by the Xilinx PCIe core as well as the host PC. This is often limited to 32 requests outstanding. The maximum size of master read requests is

determined by the root complex and dictated to the Speedy PCIe core at BIOS POST time. In a typical PCIe Gen 1.0 system, this is 512 Bytes, with 32 requests allowed in flight at any given time, and with a read completion latency of 2.4us. This mechanism could sustain a bandwidth of 512 Bytes * 32 Requests / 2.4us = 6.358 GByte/Sec if sufficient PCIe lanes were employed. In practice this number is reduced by address boundary effects imposed by the PCIe specification.

Interrupt Latency - The interrupt latency of non-real-time operating systems is highly variable and at times quite long. Interrupt latencies exceeding 1ms are common, which can have a large effect on bus utilization/bandwidth. While the hardware interrupt response is much faster than this, the structure of common driver models forces all meaningful activity to be postponed until the multitasking scheduler can create a timeslot. The implication is that interrupts should be as infrequent as possible, driving Speedy PCIe to use a large descriptor queue to buffer DMA scatter/gather requests for the typical 4 KByte virtual memory pages.

PIO Write Latency & Transaction Size - Intuitively, PIO might be used to write a new DMA descriptor into the FPGA after each contiguous physical memory page as part of servicing an interrupt. However, the observed sustained bandwidth of PIO is 2-4 MByte/Sec on all test platforms. Using the typical page size of 4 KBytes and minimally sized DMA descriptors of 20 bytes, a DMA data rate of 1.6 GByte/Sec for an x8 lane Gen 1.0 interface requires the DMA descriptor transfer rate to be at least 8.39 MByte/Sec.

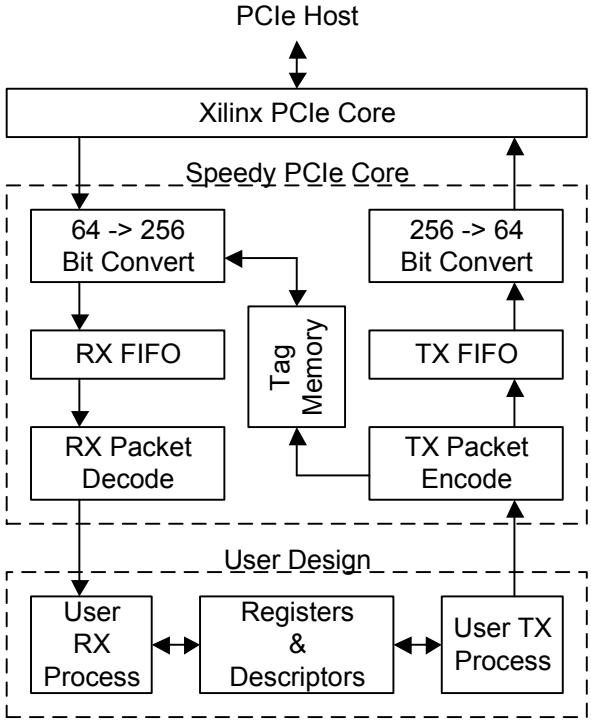

Fig. 1. Speedy PCIe Dataflow

That descriptor bandwidth can be guaranteed by storing descriptors in system memory on the PC host where latency is lower and bandwidth is higher, and then using DMA to transfer them to the FPGA.

3. DESIGN DESCRIPTION

Fig. 1 illustrates the major blocks in a Speedy PCIe hardware design. At the top is the Xilinx supplied core with a 64-bit split RX/TX interface. The section within the upper dotted line is the Speedy PCIe core consisting of connected RX and TX data paths. The lower dotted line is a user design that uses Speedy PCIe to enable a full bus mastering interface for the onboard DDR.

The Speedy PCIe core provides a memory-like interface to the user design. Slave read and write requests arrive on the RX interface, which the User RX Process decodes and either hands off to the User TX Process or writes into the appropriate memory, respectively. Master writes are initiated by the user TX Process and consist of a target PCIe address, data length and data payload. Master reads are also initiated by the User TX Process, which supplies a target PCIe address, data length and a user local address to which the returned data should be written. When the return data arrives on the RX interface, that user address is returned with it (via the Tag Memory), so that the User RX Process may simply decode and place it in the appropriate local memory in the same fashion that a slave write would be handled. The user application is not required to have knowledge of PCIe protocol rules, packet restrictions, etc.

The native word width of the Speedy PCIe core is 256-bits, which is needed due to the high bandwidth provided by PCIe and the relatively low achievable clock rates in the FPGA. At the full potential bandwidth of 3.2 GByte/Sec in a x8 lane Gen 2.0 system, the minimum system clock rate is 108MHz using 256-bit words. After allowing some overhead cycles for data processing, this still affords the user the ability to operate comfortably at 150-200MHz. While that extreme of operation is not currently supported due to the -1 speed grade part supplied with the ML605, the architecture was created with the future in mind.

The encode and decode phases of the Speedy PCIe core handle the packetization of user data, ensuring that all rules of the PCIe protocol are observed. The packets traverse the FIFOs as illustrated, while at the same time crossing clock domains. The clock rate above the FIFOs is dictated by the Xilinx PCIe core clock as negotiated at runtime to be in the range of 62.5MHz to 250MHz. Below the FIFOs, the Speedy PCIe core runs at a clock rate of the user's choosing; further reducing the difficulty of the design task imposed on the user.

In order to combat the interrupt and PIO effects discussed earlier, the driver allocates a 1 MByte physically contiguous block of system memory from the OS and writes the PCIe base address of this memory block to

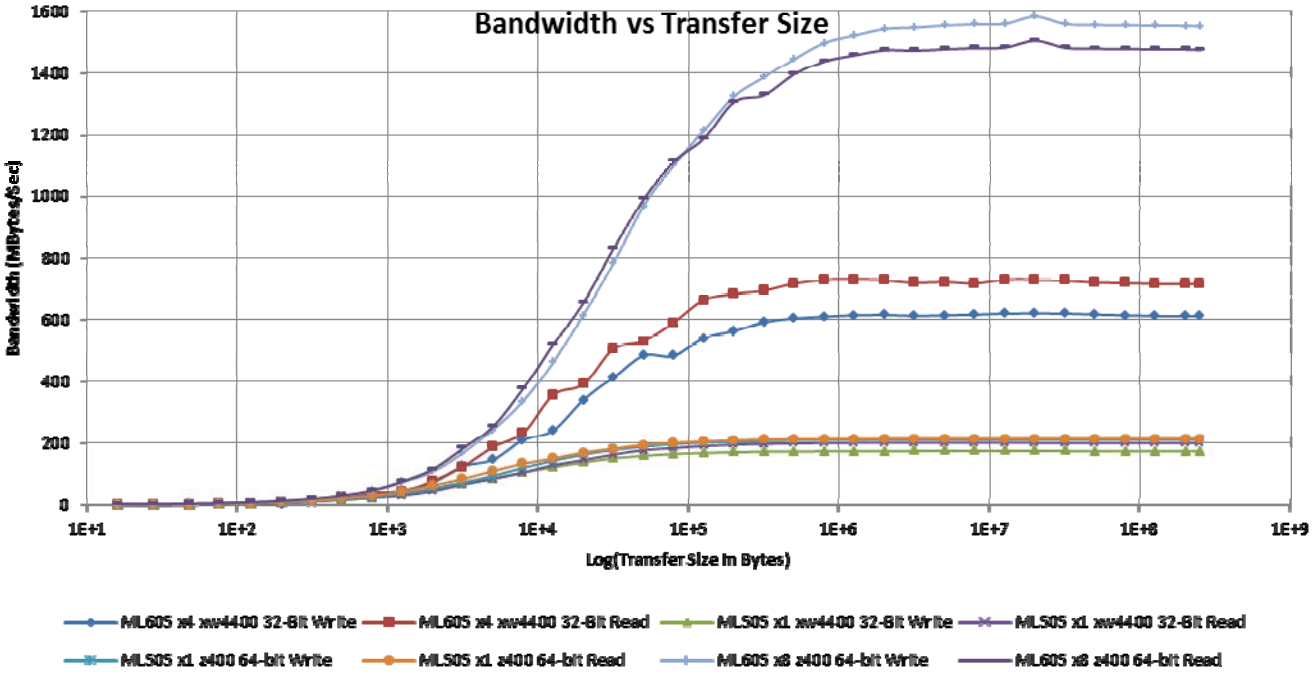

Fig. 2. Speedy PCIe Bandwidth vs transfer size for various PCIe gen 1.0 platforms

registers in the User Design. The memory block is set up as a circular queue with the driver adding descriptors to the tail, and the hardware user design removing descriptors from the head. At the start of a transfer, the driver writes DMA descriptors into this buffer and then updates the tail of the circular queue in the FPGA User Design. The User Design then generates PCIe master read requests to pre-fetch descriptors into the Registers & Descriptors block shown in Fig. 1 and uses those to initiate master reads or writes into or out of the DDR. Additional descriptors are fetched from system memory on the fly; staying ahead of the actual data transfers. This mechanism allows the host to have a relatively large DMA descriptor buffer in system memory, alleviating the problems of high interrupt latency and PIO bandwidth in one stroke.

4. RESULTS

Fig. 2 illustrates the transfer characteristics of the Speedy PCIe core in a number of configurations. An HP xw4400 workstation and an HP z400 workstation were used for testing. These were configured to run Windows 7 32-bit and 64-bit versions, while hosting two different Xilinx reference boards; an ML505 x1 lane card as well as an ML605 x8 lane card. Both are shown operating in Gen 1.0 mode, though the ML605 can also run in x1, x2 or x4 lane Gen 2.0 configurations as well. The supported Gen 2.0 configurations yield the 2x performance increase that is expected versus Gen 1.0 configurations with equal lane widths. Gen 2.0 x8 lane operation should yield 2x higher performance than the highest performer shown on the graph, but the required -2 speed grade silicon required for that mode is not supplied with the ML605 and so could not be tested.

Bandwidths were timed from a C++ application performing transfers of various sizes to/from an arbitrary 256 MB buffer allocated with the C++ new operator. Hence, these are true software attainable speeds including all associated overhead averaged over 100 transfers for each data point. In general, software writes (bus master reads) are ~5% slower than software reads (bus master writes). This is partially due to a smaller allowed DMA read request payload size than that for a bus mastering write on these platforms. Compounding the matter, these platforms only allowed 32 simultaneous bus master read requests to be in flight at any given time, due to a BIOS-imposed limitation on the number of tags, though both are typical.

Full speed operation is obtained with a transfer size of > 1 MB. The newer motherboard chipset supplied in the z400 host consistently turned in better results than the older xw4400 host. On the other hand, the choice of 32-bit or 64-bit OS had little influence on observed performance.

Table 1 shows latencies as measured by the elapsed time for a 1 DWORD (4 Byte) transfer from the point that the software API was called until it completed, averaged over 100,000 runs. As such, these show the latency as seen by the user in application software rather than the latency of the hardware alone.

Xilinx white paper WP350 gives a number of calculations for the theoretical performance of a PCIe

Table 1. Speedy PCIe Latency Running On Windows 7

Platform	OS	Lanes	Latency (us)	
			Read	Write
V5LX110T with HP xw4400	32	x1	33.4	34.8
V5LX110T with HP xw4400	64	x1	27.3	31.9
V5LX110T with HP z400	64	x1	22.8	25.1
V6LX240T with HP xw4400	32	x4	23.9	27.2
V6LX240T with HP xw4400	64	x4	18.1	23.3
V6LX240T with HP z400	64	x8	18.2	19.2

system [10] that take various system parameters into account, indicating that the performance of a Gen 1.0 PCIe system is 200 MByte/Sec per lane or less. The Speedy PCIe core compares favorably to those estimates in all configurations and does so while performing in the real world scenario of using scatter/gather transfers to move data to and from user memory.

5. DISCUSSION

Other techniques for overcoming the design challenges were attempted. For example, it is possible to allocate memory from Windows with a physical page size of 2 MBytes rather than 4 KBytes. Using those pages, high transfer rates can be achieved with a simpler architecture since the data is guaranteed to lie in larger contiguous memory regions and fewer pages/interrupts must be processed. However, the allocation of these large pages is not guaranteed by Windows, and in fact was found to be unreliable more than a few minutes after boot time.

The PCIe system is surprisingly fragile in that the software and hardware policies treat it as a monolithic bus rather than the packet-based switching fabric that it is in reality. This was most apparent in the handling of interrupts. Quite often hardware interrupts are shared in the PC platform, which requires a driver to poll status flags to determine an interrupt's ultimate cause. The PCIe designer must not only be careful to reply to each of these correctly, but also that the reply be sent within microseconds of reception. If not, the operating system will crash.

The resource requirements of the entire design are summarized in Table 2, broken out by sub block with the total resources available on a V6LX240T shown for comparison. Including DDR3 controller and all support logic, the complete design consumes 10-20% of the Virtex 6 device when compiled for a 200MHz user clock with full bus mastering capabilities enabled. The exact resource requirements fluctuate with the features implemented in the User Design as well as compiler settings.

6. CONCLUSION

While the PCIe protocol promises high bandwidth, the switching fabric introduces high latencies of 1us or more.

Table 2. Xilinx Virtex 6 Resource Requirements

Platform	Slices	FFs	LUTs	36Kb BRAMs
V6LX240T Available	37680	301440	150720	416
Speedy PCIe Core	2504	5892	6419	13
User Design	2216	8121	6559	9
Xilinx PCIe Core	561	941	954	16
MIG-based DDR3	3207	7807	5239	0

Moreover, the driver and interrupt structure of non-real-time operating systems can add even more latency. The Speedy PCIe core shows how these issues can be addressed, delivering full bus performance while presenting a simple memory-like interface to the user. It is currently used in several projects in our own lab and the complete design is available for download from [12].

7. REFERENCES

[1] PCI Express Base Specification, PCI SIG. Available: http://www.pcisig.com/specifications/pciexpress

[2] PCI Express, Xilinx Corporation. Available: http://www.xilinx.com/technology/protocols/pciexpress.htm

[3] PCI Express Compiler, Altera Corporation. Available: http://www.altera.com/products/ip/iup/pci-express/m-alt-pcie8.html

[4] XAPP1052 – Bus Master DMA Performance Demonstration Reference Design for the Xilinx Endpoint PCI Express Solutions, Xilinx Corporation, Available: http://www.xilinx.com/support/documentation/anbusinterfaceio_pciexpress.htm

[5] Qiang Wu, Jiamou Xu, Xuwen Li, Kebin Jia, "The research and implementation of interfacing based on PCI express," 9th International Conference on Electronic Measurement Instruments, pp. 3-116 - 3-121, 16-19 Aug. 2009.

[6] Ray Bittner, "Bus mastering PCI Express in an FPGA," Proceedings of the ACM international symposium on Field Programmable Gate Arrays, February 2009.

[7] Xillybus, http://xillybus.com/pcie-download

[8] PCIe 2.0 endpoint with DMA, Altera Corporation. Available: http://www.plda.com/prodetail.php?pid=103

[9] PCI Express Solution, Northwest Logic. Available: http://nwlogic.com/products/pci_express_solution.html

[10] Alex Goldhammer, John Ayer Jr, "Understanding Performance of PCI Express Systems," Xilinx WP350, Sept 4, 2008.

[11] Turner Whitted, Jim Kajiya, Erik Ruf, Ray Bittner, "Embedded Function Composition," Proceedings of the Conference on High Performance Graphics, 2009.

[12] The code will appear at the following website by the date of publication: http://research.microsoft.com/people/raybit