PCIe 3 FPGA Framework for stream processing

Malte Vesper
The University of Manchester
malte.vesper@postgrad.manchester.ac.uk

Dirk Koch
The University of Manchester
dirk.koch@manchester.ac.uk

Vipin Kizheppatt
Mahindra cole Centrale
vipin.kizheppatt@mechyd.edu.in

***Abstract*—Fast PC to FPGA communication is a requirment for heterogenous computing. While there are many frameworks to help with the creation of such a link, we present a solution optimized for streaming over PCIe 3 with multiple FPGA boards. We compare our solution to ¡LIST OF OTHERS¿, We achieve ¡XX%¿ of the theoretical ¡list stransfer types¿ and can achieve a continious throughput of ⟨¡VALUE¿⟩.**

Accelerating computing tasks with heterogenous

To achieve even better performance with the ever performance hungry world, heterogenous computing.

As science struggles to satisfy the evergrowing performance demand with traditional scaling, heterogenous computing becomes an option. While the idea of using the platform best suited to solve the problem seems natural, it requires efficient data transfer between the platforms in case the job is to be migrated or any benefit from the better platform is negated/eaten up by the transfer. The fact that many FPGA boards support PCIe, its easy integration and speed make it an obvious choice. Hence we surveyed the available PCIe 3 frameworks for FPGAs. Since all of them have shortcomings we developed our own. While looking for frameworks to support PCIe 3 connections to FPGAs we did not find any open source solutions and the comercial solutions are far from optimal performance. Despite PCIe 3.0 beeing available since late 2010 and PCIe 4 already on the horizon, we could not find any open source solutions. We present an easy to use framework to set up PCIe 3 on FPGAs, consisting of a linux driver and a configurable core which provides multiple channels.

As traditional methods for preformance gain are exceedingly exhausted heterogenous computing becomes more relevant despite its complexity. We surveyed the PCIe 3 solutions available and compare them to our own implementation, which overcomes performance and utilization shortcommings of the commercial solutions. Our solution supports multiple FPGAs in one system, is geared towards streaming and is the only open source solution currently available. We calculate the theoretical maximum throughput for single and bidirectional transfers and show how triple buffering helps to reduce the gap between practical and theoretical throughput.

I. Introduction

With FPGAs beeing used as accelarators for various tasks and beeing rather IO than compute bound, it becomes important to increase the IO bandwidth. Due to its theoretical bandwidth, presence on evaluation boards and its prevalence in PCs PCIe (PCIe) becomes a natural choice.

While the FPGA distributors provide cores for PCIe, these support only the physical and data link layer, the user still has to provide a driver and take care of generating the headers for the transaction layer. To ease development there are many academic and comercial solutions providing further abstraction, allowing the researcher to focus on his actual problem rather than the infrastructure required.

Unfortunatly most solutions fall short in one aspect or another. Most (¡list¿) do only support PCIe 2, which delivers only half the bandwidth[1] of PCIe 3. Xillybus, while supporting PCIe 3.0, is to slow, according to the website they achieve arround 800MB/s [1], [2], which is only around 10% of the theoretical maximum. Some comercial solutions are closed source, making it impossible to tweak them to specific needs. While the blackbox model works in the ideal case, there are valid reasons to tweak the cores. One last important feature is multi board support. To further increase performance it can be desirable to split the load accross more than one board. As ware as we are aware this is only supported by ¡list of products¿.

We would like to clarify, that with Gigabyt [GB] we refer to 1000^3 Bytes rather than 1024^3 Bytes. We refer to the later as *Gibibyte* [GiB] in accordance with [3]. Furthermore we

[1]Theoretical payload bandwidth.

assume the FPGA perspective for all uses of the word send or receive unless explicitly otherwise noted. In paticular this means that *send* refers to a data transfer from the FPGA to the host, and *receive* to a transfer in the opposite direction.

II. PCIe 3.0

PCIe uses differential pairs to transfer its data, a pair of these wires, one going to and one coming from the endpoint is called *lane*. One or more lanes form a *link*. The PCIe spec uses the layers 1, 2 and 7 of the OSI model, namely physical, data link and application layer. On the application layer, refered to as transaction layer, *transaction layer packages* (*TLPs*) are sent. The packages for flow control, power managment and (non) acknowledge which are exchanged by the data link layer are referred to as *data link layer packages* (*DLLPs*). PCIe knows two types of transfers, *non-posted transfers* require a response and *posted transfers*, which are fire and forget[2]. A send transfer is posted, while a receive transfer is non posted. The initial transfer is just a request for data, and the data is then received in the response. Since posted transfers do not have a turnarround delay they are much faster. One thing noteworthy for PCIe is that it does not require an ack for each package. If an ack is received all previous unacknowledged packages[3] are considered acknowledged. The ack factor represents the number of TLPs with a given payload that can be received before an ack is sent. This translates to the offerhead due to the ack DLLPs that have to be send. In the following subsection we will take a look at the overhead resulting overhead from the data link layer.

cite someone on this here

Theoretical limits

We calculate the theoretical maximum data througput assuming optimal conditions, i.e. we do assume there are always sufficent flow control credits available and enough packages queued, that there is zero latency between packages. Furthermore we assume that memory transfers are not split into multiple packets and addressing is done with 32 bits.

Each transfer is associated with a certain ammount of overhead. In the following the theoretical maximum throughput for links operating at 8GT/s is calculated. To minimize overhead the maximum payload size of 4096 Bytes is exploited and assumed for all further calculations. This overhead includes

[2]Actually buffered in the replay buffer till acknowledged

[3]for which no nack was received

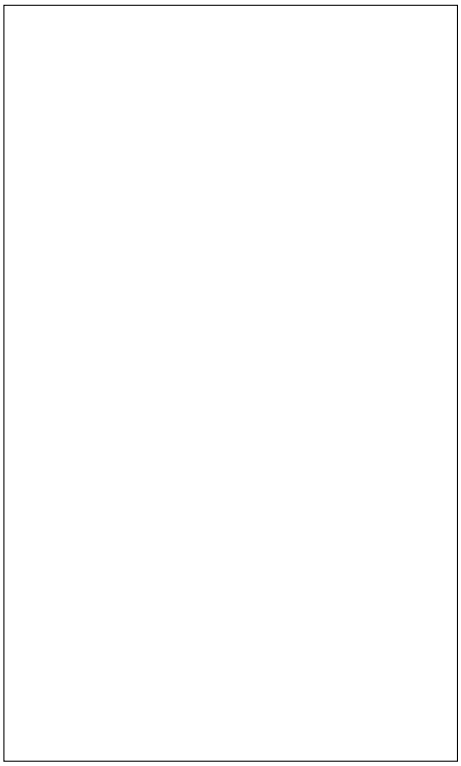

Fig. 1. Packages exchanged for a transfer, and latencys

	Bytes	
	from device	to device
Send	20[1]	24[3]
Receive	44[1,3]	32
Send & Receive	64[2]	56

[1] Add 4 Bytes for 64 bit addressing

[2] Add 4 Bytes for 64 bit addressing per direction

[3] Subtract 4 Bytes for Ack Factor 2.0

TABLE I
OVERHEAD IN BYTES PER PACKAGE

the 128/130 coding, CRC and LCRC check sums, framing, and flow control related to the transfer. shows the transaction layer packages and the the data link layer packages for a send transfer, for a receive. The protocol overhead in bytes transfered in either direction, excluding the coding, is summarized in Table I.

Figure + ref sub-fig a

Since the number of lanes[4] affects the number of ACKs that have to be sent and thus the efficency we show our results for all lane widths specified in the standard[4]. As and

Figure + ref sub-fig b

[4]and the package size

	Ack Factor	
	1.0	2.0
Send only	.980	.980
Receive only	.977	.977
Send & Receive		
Combined	.970	.971
Send	.969	.970
Receive	.971	.972
Symetric Speed	0.969	0.970

TABLE III
PERCENTUAL EFFECTIVNESS?

Table I show a transfer in either direction causes a transfer of some packets in the opposite direction. Thus the channels are not completly independent. We chose the send and receive dataflow in a way to maximize the overall throughput. The results in Table II are obtained with Equation 1.

$$\text{Throughput} = \frac{\text{Payload}}{\text{Payload} + \text{overhead}} * \text{Transferrate} * \text{Lanes} \quad (1)$$

$$\text{Throughput}_{\text{dir}} = \frac{4096\text{B}}{4096\text{B} + \text{overhead}_{\text{dir}}} * 8\text{GT/s} * \text{Lanes} \quad (2)$$

To determine the combined send and receive throughput we determine the maximum for Equation 3 under the constrains in Equation 4 and 5. For Ack Factor 2.0 the factors in the constraints are adjusted according to the overhead table (Table I).

$$\text{Tp} = (\text{Pkg}_{\text{send}} + \text{Pkg}_{\text{receive}}) * 4096\text{B} \quad (3)$$

$$1\text{GB} * \text{Lanes} >= (20\text{B} + 4096\text{B})\text{Pkg}_{\text{send}} + 44\text{BPkg}_{\text{receive}} \quad (4)$$

$$1\text{GB} * \text{Lanes} >= 24\text{BPkg}_{\text{send}} + (32\text{B} + 4096\text{B})\text{Pkg}_{\text{receive}} \quad (5)$$

For the symmetric speed we added the constraint that the amount of packages sent has to equal the amount received.

Even in the worst case over 96.9% of the PCI bandwidth is available for payload. While the ack factor is irrelevant for the send or receive only case, since the ack flow in the opposite direction to the large payload. However for the concurrent send and receive the ack factor leads to an increase of arround one percent. Eventhough it looks as if the symmetric throughput is the same as the throughput of the slower direction in the send an receive case, it is slightly better. Due to rounding this becomes only apparent for the last entry in Table II. If you compare the receive throughput from the send and receive column to the one way througput.

doublecheck if the ack factor

CHeck that all acronym are spelled out at least once

III. DMA

If the PCIe device does not supports 64 bit addressing, only the lower 4 GiB of memory are addressable. This can either be achieved through an *IOMMU*, a memory managment unit between the devices and main memory or through memory allocation mechanisms in that physical range. Since memory at fixed physical addresses can only be allocated by the operating system we have to either map it into the programs address space or copy the data from the userspace to the allocated memory. The former is known as *direct buffering*, the later as a *bounce buffer*. The downside of a bounce buffer is that it requires an extra copy of the data, thus doubling the stress on the memory.

Both, an IOMMU and mapping the DMA buffers allow for *zero copy DMA*; the data is directly transfered from its initial ram position without additional copies between buffers.

Buffering

On the driver side different bufferstrategies can be used. A good overview can be found in [5], we well go over the ones supported by our driver here briefly. While a single buffer is technically sufficient, *double buffering* allows to fill a second buffer while the first one is transfered. Since the PCIe transfer is the bottleneck it helps to ensure that PCIe is not stalled. A *buffer pool* is the generalization of a doulbe buffering, instead of two buffers there are N buffers available which can be filled and dispatched independently. By having multiple buffers dispatched simultaneously the transfer startup latency can be hidden.

A. Hardware

Our Design is based arround the PCIe 3 core supplied by Xilinx [6], which takes care of the physical layer and parts of the data link layer. We test on an Asus Rampage IV with an Intel Core i7-4930K with 64 GiB of Memory running at X. With the four channels this gives X GiB/s throughput from the memory.

	Lanes						
	AckFactor = 1				AckFactor = 2		
	1	2	4	8	12	16	32
Send only	0.980	1.960	3.919	7.839	11.758	15.677	31.355
Receive only	0.977	1.954	3.908	7.816	11.724	15.632	31.263
Send & Receive							
Combined	1.941	3.882	7.763	15.526	23.312	31.083	62.165
Send	0.969	1.939	3.878	7.756	11.645	15.526	31.052
Receive	0.971	1.943	3.885	7.771	11.667	15.556	31.113
Symetric Speed							
Combined	1.939	3.878	7.756	15.511	23.290	31.053	62.106
One way	0.969	1.939	3.878	7.756	11.645	15.526	31.053

TABLE II
THEORETICAL MAXIMUM THROUGHPUT IN GB/S

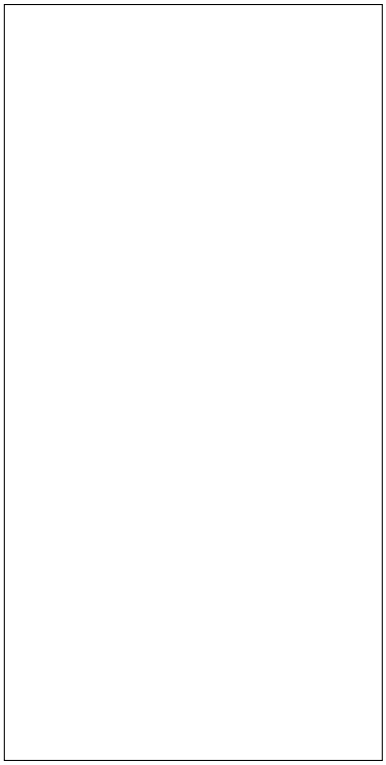

Fig. 2. Buffering strategies

IV. COMPARISON TO EXISTING SYSTEMS

While an IOMMU can coascle multiple scattered pages to one consistent region from the device view, and thus reduce addressing overhead, this does not eliminate the need for bank and rank switching. This addressing overhead causes the slowdown that we see in xxx.

add speed graph

	DMA engine(average)	PCIe Core	Entire System
LUT			
BRAMS			
DSPs			
FlipFlops			
Overall			

TABLE IV
RESOURCE USAGE

V. RESOURCE USAGE

The resource usage depends on the number of channels instantiated. shows the resource usage. We split the systems resource usage into the part for the logic generated by us and that used by the Xilinx core.

ref table resource usage

VI. PERFORMANCE

While section II PCIe 3.0 shows the maximal theoretical throughput for 4096 byte Payloads, the Xilinx PCIe core is limited to 1024 byte [6]. Applying our throughput calculations to this we get the results in .

uebersch richtig setzen zahlen

As outlined in section II PCIe 3.0 the flowcontrol affects the (reverse?)opposite direction. Therefore we measured the throughput for unidirectional and concurrent send and receive. The results for this in regard to the different buffering techniques we support are shown in TBL. In the ideal configuration we achieve X% for send, X% for receive and X% for

tbl

uebersch richtig set-

	throughput			
	theoretical	single board	4 boards	% of theoretical throughput
Send				
Receive				
Send & receive				

TABLE V
RESOURCE USAGE

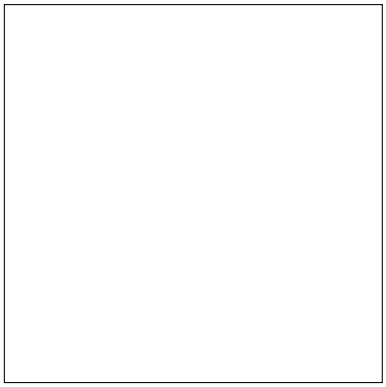

Fig. 3. Throughput send

bidirectional asyn transfers.

Graph X shows the performance of the different techniques with different ammounts of boards. As can be clearly seen, by the drop in the average performance per device plot copy DMA becomes unacceptable once more than X Boards/Links are used. This throughput plateau causing the drop of per device throughput is explained by the systems memory bandwidth of X now becoming the bottleneck instead of the PCIe bus.

We measured the test data generation to occur at up to X GiB/s, ruling out that this is the bottleneck.

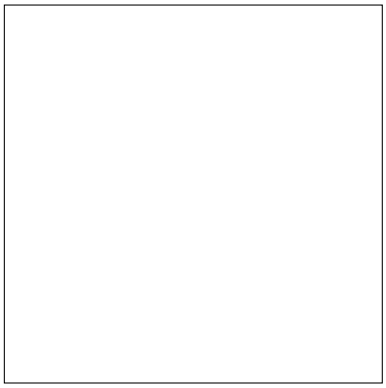

Fig. 4. Throughput receive

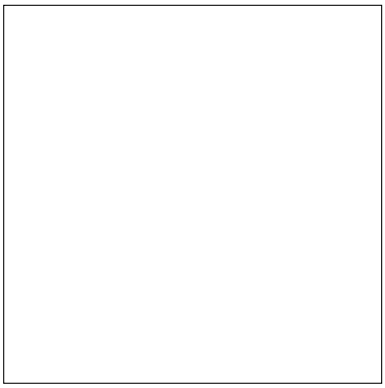

Fig. 5. Combined Throughput

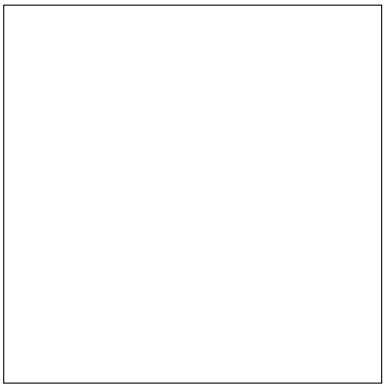

Fig. 6. Througput depending on buffering strategy

VII. CONCLUSION

We have calculated the theoretical throughput limitations for PCIe 3.0 and presented our design. The design supports PCIe transfers to multiple boards at average transfer speeds equal to % of the theroetical limit. Furthermore we measured different buffer strategis and their impact on performance. Since our system contains no device specific code besides Xilinx PCIe core it should be easily adoptable to different boards.

REFERENCES

[1] Xillybus. (2015) Download xillybus for pcie. [Online]. Available: http://xillybus.com/pcie-download

[2] ——. (2015) Xillybus data bandwidth. [Online]. Available: http://xillybus.com/doc/xillybus-bandwidth

[3] ISO, *Quantities and units Part 13: Information science and technolog*, International Organization for Standardization ISO IEC 80 000-13, 2008.

[4] PCI-SIG, *PCI Express Base Specification, Revision 3.0*, PCI-SIG Std., 2010.

[5] G. M. Martinez *et al.*, "On buffer management strategies for high performance computing with reconfigurable hardware," in *International Conference on Field Programmable Logic and Applications*, 2006.

[6] Xilinx, *Virtex-7 FPGA Gen3 Integrated Block for PCI Express v3.0*, Nov 2014.