

Hardware-Based PCI Express* Performance Measurement

Remote Lab Demonstration Tutorial 3

November 2010

Revision 001

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Intel, the Intel Atom family of processors, and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

*Other names and brands may be claimed as the property of others.

Copyright © 2010, Intel Corporation. All rights reserved.

Copyright © 2010, Gleichmann Electronics Research. All rights reserved.

Contents

1	Introduction	6
1.1	Goals	6
1.2	Referenced Documents.....	6
2	Quick Start	7
3	Theoretical throughput of PCIe	12
3.1	Channel coding.....	12
3.2	Protocol overhead	12
3.3	Data Payload size	13
3.4	The amount of non-data carrying Packets	15
3.5	Example Calculation.....	16
3.6	Summary.....	23
4	Measuring the throughput of CPU initiated PCIe transfers.....	24
4.1	Measurement setup	24
4.2	Synthesizing a design for the measurement setup	25
4.3	Test program	28
4.4	Test 1: Two Write Accesses.....	28
4.5	Test 2: Two Read Accesses	29
4.6	Test 3: Multiple Writes	30
4.7	Test 4: Multiple Reads.....	36
4.8	Test 5: Alternating Write/Read Accesses.....	41
4.9	Test 6 and 7: Examine Worst Case Behavior	41
4.10	Test 8: Multiple Writes, CPU Time	45
4.11	Test 9: Multiple Reads, CPU Time	46
4.12	Test 10: Multiple Writes, 64 Bits	47
4.13	Test 11: Multiple Reads, 64 Bits	48
4.14	Test 12: Multiple Writes, 64 Bits, CPU Time.....	49
4.15	Test 13: Multiple Reads, 64 Bits, CPU Time	49
4.16	Test 14: Write Combining	50
4.17	Test 15: Compare 1, 2, 4, 8Byte Payload size, read	52
4.18	Test 16: Timer Calibration	55
5	Measuring the throughput of DMA transfers	56
5.1	Test data generation	57
5.2	Performance Measurement.....	58
5.3	Measurement Setup	58
5.4	Address Mapping	60
5.5	DMA Buffer	61
5.6	Demo application.....	62
5.7	Test Procedures.....	63
5.8	Demo Application: Command Line Parameters.....	64
5.9	Results	64
6	Summary.....	66

Figures

Figure 1. PCIe protocol layers.....	12
Figure 2. Throughput for PCIe transmissions	15
Figure 3. Measurement setup	25
Figure 4. Delays between write accesses.....	32
Figure 5. Plot and histogram for 1000 AHB RAM write accesses.....	33
Figure 6. Delays between read accesses	36
Figure 7. Delays between write accesses (60 000 accesses)	42
Figure 8. Delays between read accesses (60 000 accesses)	43
Figure 9. Delays between read accesses with disabled kernel debug/trace features ..	44
Figure 10. Delays between read accesses, tickless kernel, disabled debug/trace/HRT	45
Figure 11. Throughput result of test 14 (AHB-RAM size = 32k)	51
Figure 12. Throughput result of test 15 with n=60000	53
Figure 13. Basic architecture of the Hpe_IRP	56
Figure 14. DMA demo design.....	57
Figure 15. Measurement setup for the DMA demo design	59
Figure 16. Contiguous virtual memory may be discontinuous in physical memory	60
Figure 17. Address mapping between virtual memory, physical memory and AHB addresses	61
Figure 18. The contents of /proc/iomem after reserving the upper 512MiB of RAM ...	62
Figure 19. The memory mappings of the dma-demo.....	63

Tables

Table 1. Throughputs for PCIe transmissions (without ECRC).....	14
Table 2: Command line options of dma-demo.....	64
Table 3: Results of the dma-demo	65

Revision History

Document Number	Revision Number	Description	Revision Date
324611	001	Initial release.	November 2010

§

1 Introduction

1.1 Goals

After this tutorial you will understand:

- The theoretical throughput of PCI Express* (PCIe) on the Hpe® Industrial Reference Platform (IRP)
- The protocol overhead involved in PCIe transactions
- The real-world performance of PCIe in terms of latency and throughput
- How to create Field Programmable Gate Array (FPGA) designs to perform your own, application-specific benchmarks

This document assumes familiarity with the environment of the Hpe_IRP which is described below.

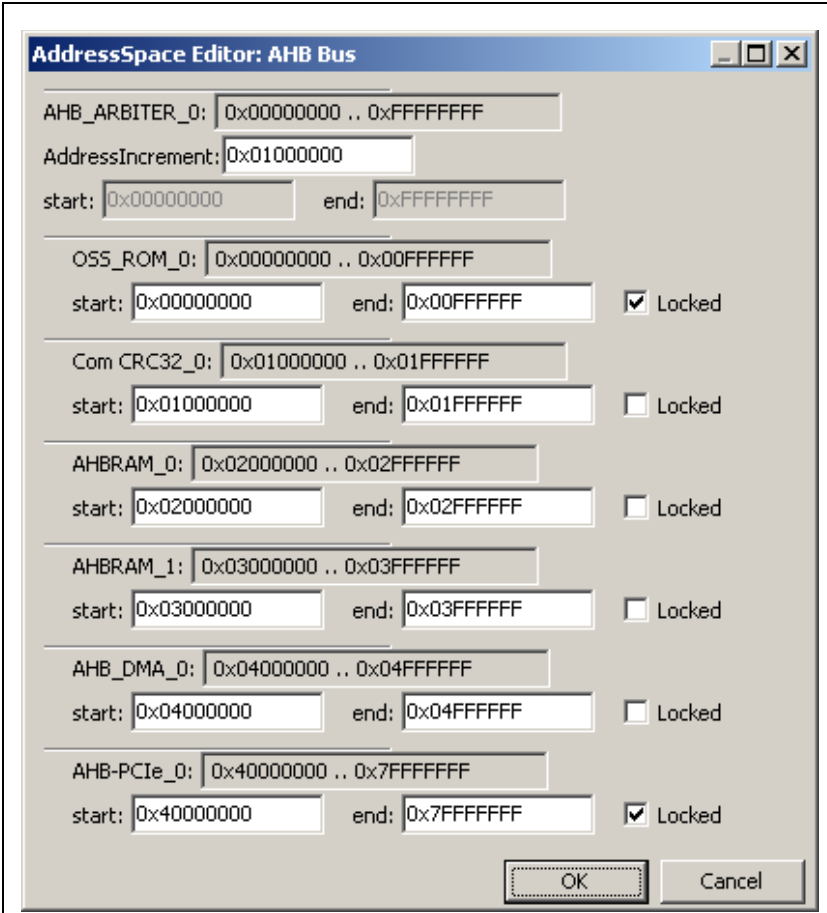
1.2 Referenced Documents

Ref	Document	Document Number/Location
1	Intel Remote Lab Access Guide	Contact your Intel representative
2	Hpe® Industrial Reference Platform (IRP) User Manual	http://www.ge-research.com/attach/UserManual-Hpe_IRP.pdf
3	Hpe® desk Basic Manual	http://www.ge-research.com/attach/UserManual_HpeDesk_basic.pdf
4	Remote Lab Demonstration	324614
5	Hardware Level IO Benchmarking of PCI Express	321071
6	Performance Analysis of the Intel® System Controller Hub (Intel® SCH) US15W	Intel® Technology Journal Volume 13, Issue 1, 2009
7	Intel® 64 and IA-32 Architectures, Software Developer's Manual, Volume 3A: System Programming Guide, Part 1	253668

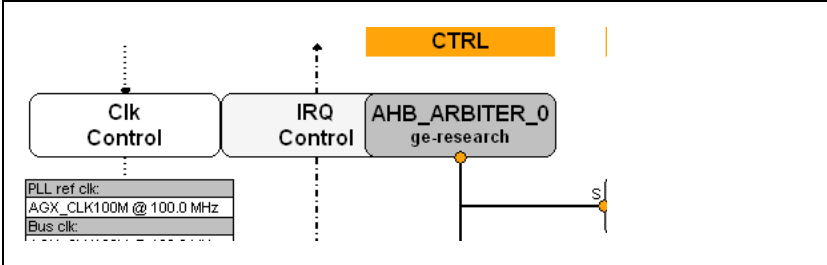
2 Quick Start

This section leads you directly to the practical benchmark results without going into detail. Details of this design are in Section 5.

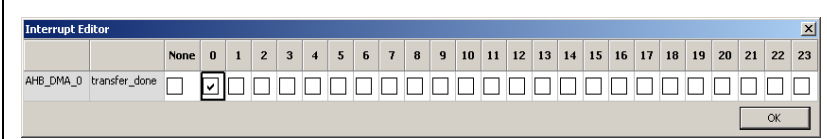
<p>Utilization Estimator: EP1AGX90EF1152</p> <table border="1"> <thead> <tr> <th></th> <th>used</th> <th>avail.</th> <th>[%]</th> </tr> </thead> <tbody> <tr> <td>ALLUs</td> <td>12,803</td> <td>72,176</td> <td>18%</td> </tr> <tr> <td>Registers</td> <td>8,785</td> <td>72,176</td> <td>12%</td> </tr> <tr> <td>MemoryBts</td> <td>1,613,824</td> <td>4,477,824</td> <td>36%</td> </tr> <tr> <td>LogicElements</td> <td>15,821</td> <td>72,176</td> <td>22%</td> </tr> </tbody> </table>		used	avail.	[%]	ALLUs	12,803	72,176	18%	Registers	8,785	72,176	12%	MemoryBts	1,613,824	4,477,824	36%	LogicElements	15,821	72,176	22%	<p>Open Hpe_desk and create an FPGA design using the following IPs:</p> <ul style="list-style-type: none"> • OSS_ROM • Com CRC32 • AHB RAM (2x) • AHB_DMA • AHB-PCIe.
	used	avail.	[%]																		
ALLUs	12,803	72,176	18%																		
Registers	8,785	72,176	12%																		
MemoryBts	1,613,824	4,477,824	36%																		
LogicElements	15,821	72,176	22%																		
	<p>Right-click the canvas and select "Edit AddressSpace"</p>																				



Configure the memory map. Note: Check the "Locked" checkbox of the AHB-PCIE **after** changing the start and end address.

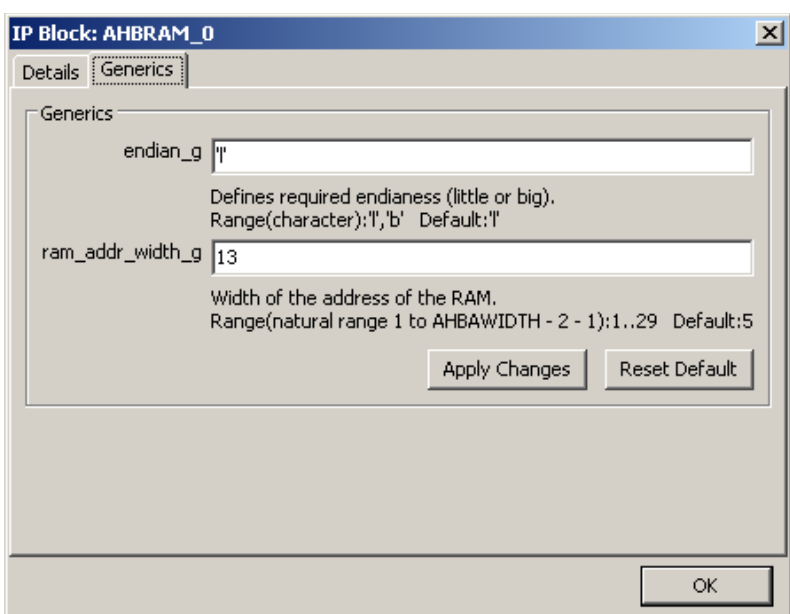
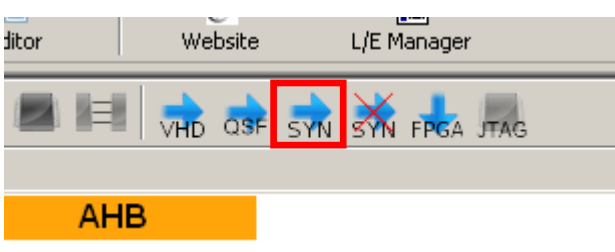
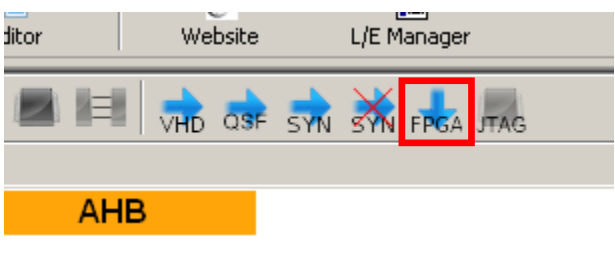


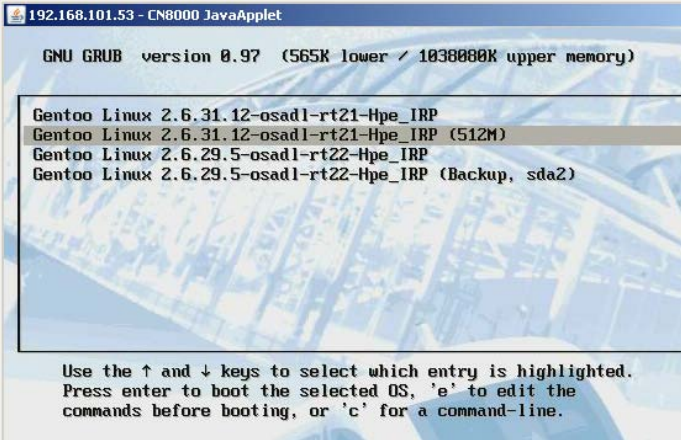
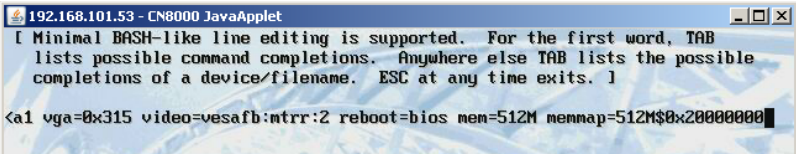
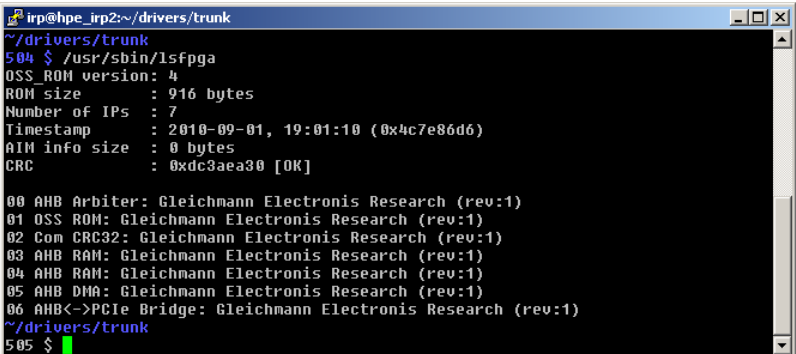
Double-click the "IRQ control" box.



Assign the interrupt of the DMA controller to IRQ 0.

Remote Lab Demonstration Tutorial
Quick Start

	<p>Configure the size of the two "AHBRAM" instances to 8k: Double-click AHBRAM_0, select the "Generics" tab and set the parameter "ram_addr_width_g" to 13. Repeat this for AHBRAM_1.</p>
 <p>AHB</p>	<p>Generate the top-level VHDL code, the Quartus® Settings File and synthesize the design by clicking the "SYN" button.</p> <p>Note: This step will take about 40 minutes. You can find a prebuilt design on the desktop in your remote lab environment.</p>
 <p>AHB</p>	<p>Download the resulting design to the FPGA by clicking the "FPGA"</p>

	<p>Reboot the Hpe_IRP. In the bootloader (GRUB) select the entry with the suffix "512M"</p>
	<p>Optional: If there is no such entry, interactively edit the kernel parameters: Type 'e', select the line starting with "kernel", type 'e' again and append the following parameters: "mem=512M memmap=512M\$0x20000000").</p>
	<p>Login after the boot process has finished. Use "lisfpga" to verify that the FPGA contains the correct design.</p>

Remote Lab Demonstration Tutorial

Quick Start

```
irp@hpe_irp2:~/dma-demo-1.0
~
550 $ tar xjf Hpe_IRP_support/Demo_designs/dma-demo-1.0.tar.bz2
~
551 $ cd dma-demo-1.0/
~/dma-demo-1.0
552 $ make
gcc -O2 -Wall -o dma-demo dma-demo.c uio-helper.c utils.c
cd kernel-module ; make
make[1]: Entering directory `/home/irp/dma-demo-1.0/kernel-module'
make EXTRA_CFLAGS='-DSUN_TAG=\"1.0\" -DSUN_REV=\"\"' \
      KBUILD_EXTRA_SYMBOLS=/lib/modules/2.6.31.12-osad1-rt21-Hpe_IRP/k
kernel/drivers/hpe_irp/Module.symvers \
      -C /lib/modules/2.6.31.12-osad1-rt21-Hpe_IRP/build M=/home/irp/d
dma-demo-1.0/kernel-module modules
make[2]: Entering directory `/usr/src/linux-2.6.31.12-osad1'
CC [M] /home/irp/dma-demo-1.0/kernel-module/dma_demo.o
Building modules, stage 2.
MODPOST 1 modules
CC /home/irp/dma-demo-1.0/kernel-module/dma_demo.mod.o
LD [M] /home/irp/dma-demo-1.0/kernel-module/dma_demo.ko
make[2]: Leaving directory `/usr/src/linux-2.6.31.12-osad1'
make[1]: Leaving directory `/home/irp/dma-demo-1.0/kernel-module'
~/dma-demo-1.0
553 $ ls
Makefile dma-demo.c uio-helper.c utils.c
dma-demo kernel-module uio-helper.h utils.h
~/dma-demo-1.0
554 $ ls kernel-module/
Makefile dma_demo.c dma_demo.mod.c dma_demo.o
Module.symvers dma_demo.ko dma_demo.mod.o modules.order
~/dma-demo-1.0
555 $ sudo insmod kernel-module/dma_demo.ko
~/dma-demo-1.0
```

Unpack the demo design and compile it: Type
"tar xjf
Hpe_IRP_support/Demo_designs/dma
-demo-1.0.tar.bz2"
cd dma-demo-1.0
make
The outcome should be an executable
called "dma-demo" and a kernel
module "kernel-mode/dma_demo.ko"
Load the kernel module. Type:
"sudo insmod kernel-
module/dma_demo.ko"

```
irp@hpe_irp2:~/dma-demo-1.0
~/dma-demo-1.0
538 $ ./dma-demo -3 -4
dma buf:0x97fb4000-0xb7673fff, size:527.17 MByte
Determining time stamp counter (TSC) frequency... DONE.
TSC frequency: 1.6 GHz

Test 1: DMA Mem->FPGA
=====
* Initializing dma_buf via CPU. seed=0x00c0ffee... DONE.
-> 221.74 MByte/sec
* Calculating CRC of dma_buf via CPU... DONE. ref_crc=0x1d7c118c.
-> 31.49 MByte/sec
* Starting transfer: Mem->FPGA... DONE. CRC [OK] (crc=0x1d7c118c).
* Duration of the write call in user-space (measured via TSC):
-> 10.93 sec, 48.23 MByte/sec.
* Duration the DMA controller was busy (measured in the FPGA):
-> 10.93 sec, 48.24 MByte/sec.

Test 2: DMA FPGA->Mem
=====
* Initializing dma_buf via CPU. seed=0xc01ce7ea... DONE.
-> 221.80 MByte/sec
* Setting random seed in the FPGA. seed=0xc0cac01a
* Starting transfer: FPGA->Mem... DONE.
* Duration of the read call in user-space (measured via TSC):
-> 4.76 sec, 110.83 MByte/sec.
* Duration the DMA controller was busy (measured in the FPGA):
-> 4.76 sec, 110.85 MByte/sec.
* Verifying memory... DONE. [OK]
~/dma-demo-1.0
539 $ █
```

Run the demo application. Type:
"./dma-demo -3 -4"
Note: If you get permission errors,
either run the dma-demo as root or
fix the file permissions of
/dev/dma_buf and /dev/uio*.
The demo application will run for
approx. 50 seconds and output the
read and write performance.

3 Theoretical throughput of PCIe

The throughput of a PCIe communication depends on several factors. The two most important factors are the number of the lanes and the PCIe version. On the Hpe IRP the FPGA is connected to the Intel® Atom™ Processor with one lane. This is called PCIe x1. The PCIe version is v1.1.

In addition to the physical bandwidth limitation, the most important throughput reducing factors are:

- Channel coding
- Protocol overhead
- Data payload size
- The amount of non-data carrying packets

3.1 Channel coding

The physical layer of PCIe v1.1 is full duplex and offers a bit data rate of 2.5 GBit/s per lane for each direction. For each direction a differential signaling scheme is used. The physical layer uses an 8b/10b encoding. The 8b/10b encoding maps every 8 bit symbol to a 10 bit symbol. This encoding is needed for DC-balancing, clock recovering and error detection. The encoding produces an over-head of 25%: Only 8 of 10 bits carry real information. Hence this limits the useable bandwidth from 2.5 GBit/s to 2 GBit/s = 250 MByte/s per lane and direction.

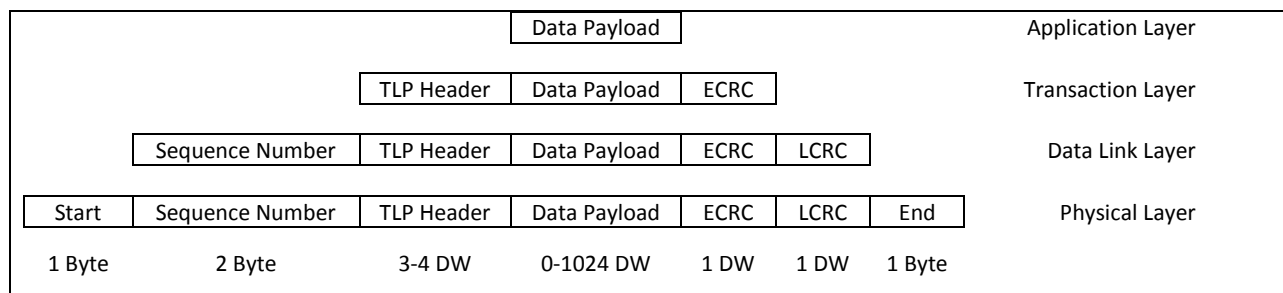
Note: Sometimes the bandwidth is given as 500 MByte/s, because the link is full duplex, i.e. it is possible to transfer 250 MByte/s in both directions at the same time. Of course this is only valid if the data streams in both directions are completely unrelated. To avoid confusion, we will always consider a single direction in the remainder of this document.

Throughput after physical layer: 250 MByte/s

3.2 Protocol overhead

Four protocol layers are involved in a PCIe transmission (see Figure 1). The PCIe protocol itself comprises three layers: transaction layer, data link layer and physical layer. Data (payload) sent by an application will travel through these three protocol layers twice: from application to physical layer at the sender and from physical back to the application layer at the receiver. At each transition between the layers the additional fields shown in Figure 1 are added or removed – depending on the direction by which the data moves through the protocol stack.

Figure 1. PCIe protocol layers



A transaction layer packet (TLP) which is fed into the physical layer is shown in the fourth row of Figure 1. It consists of the start and end framing symbols, the sequence number, a TLP header, an optional end to end cyclic redundancy check (ECRC) and a link cyclic redundancy check (LCRC). The rest of the TLP contains 0 to 1024 dwords of data. This part is the data payload from the application layer. All other fields are protocol overhead. As you can see the efficiency of the transmission strongly depends on the payload size: The higher the payload size, the smaller the overhead compared to the overall packet size.

A single TLP has between five and seven dwords of protocol overhead, depending on the TLP header size and on whether or not the optional ECRC is included. A dword consists of 4 bytes, which means the overhead is between 20 and 28 bytes.

Most of the PCIe transmissions use TLP headers with 3 dwords in size. TLP headers with a size of 4 dwords are necessary mainly when 64 bit addresses are used. On the Hpe IRP 32 bit addresses are used, so almost all of the TLPs have a 3 dword header. The only exceptions are Message Requests. Message Requests always use a 4 dword header.

The efficiency can be calculated as follows:

$$\text{Efficiency [\%]} = 100\% * \text{payload size} / (\text{payload size} + \text{overhead})$$

overhead =	Start	Sequence Number	TLP Header		ECRC	LCRC	End
	1 Byte	2 Bytes	12 of 16 Bytes		4 Bytes	4 Bytes	1 Byte

The Hpe IRP does not use the optional ECRC, so the overhead of TLPs with a 3 dword header and a TLP with a 4 dword header can be calculated as follows:

TLP with 3 dword header:

overhead =	Start	Sequence Number	TLP Header		ECRC	LCRC	End
	1 Byte	2 Bytes	12 Bytes		0 Bytes	4 Bytes	1 Byte

= 20 bytes

TLP with 4 dword header:

overhead =	Start	Sequence Number	TLP Header		ECRC	LCRC	End
	1 Byte	2 Bytes	16 Bytes		0 Bytes	4 Bytes	1 Byte

= 24 bytes

Best case throughput with TLP overhead: 250Mbyte/s * 4096 / 4116 = 248.79 Mbyte/s

3.3 Data Payload size

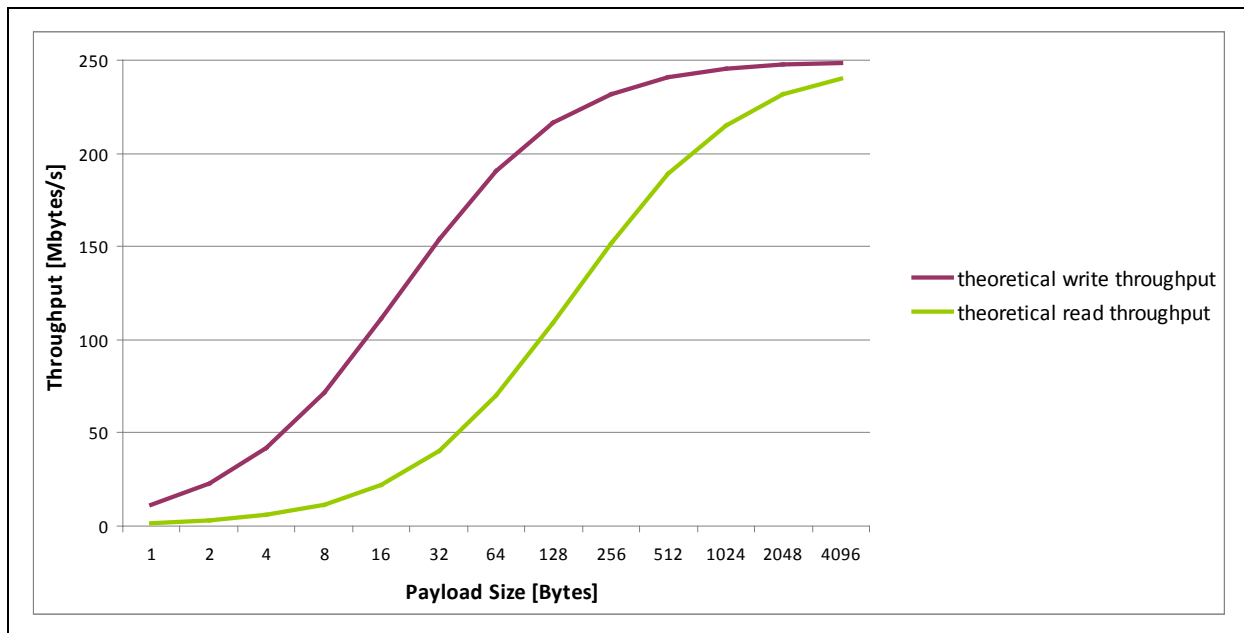
As already mentioned the data throughput depends strongly on the payload size. The maximum payload size defined by the PCIe standard is 4096 bytes. On the Hpe_IRP the maximum payload size is 128. The maximum payload size on the Hpe_IRP is limited by the PCIe root complex within the Intel® System Control Hub US15W.

The maximum throughput as a function of the payload size can be calculated. The results for some payload sizes are listed in the Table 1 and plotted in Figure 2.

Table 1. Throughputs for PCIe transmissions (without ECRC)

Payload size [Byte]	Efficiency [%] 3DW Header	Throughput [MByte/s] 3DW Header	Efficiency [%] 4DW Header	Throughput [MByte/s] 4DW Header
1	4.76	11.09	4.00	10
2	9.09	22.73	7.69	19.23
4	16.67	41.66	14.29	35.71
8	28.57	71.43	25.00	62.50
16	44.44	111.11	40.00	100.00
32	61.54	153.85	57.14	142.86
64	76.19	190.48	72.73	181.82
128	86.49	216.22	84.21	210.53
256	92.75	231.88	91.43	228.57
512	96.24	240.60	95.52	238.81
1024	98.08	245.21	97.71	244.27
2048	99.03	247.58	98.84	247.10
4096	99.51	248.79	99.42	248.54

Figure 2. Throughput for PCIe transmissions



PCIe uses flow control, so a TLP will not be sent until the receiver has enough buffer space. This means that the receiver must be fast enough or must have enough buffer space to reach the throughput as calculated in Table 1. Also keep in mind the influence of the DLLPs and PLPs explained in the next section is not considered in Table 1 and Figure 2.

The theoretical read throughput in Figure 2 was calculated with a read to completion time of 500ns and without the influence of DLLPs or PLPs.

3.4 The amount of non-data carrying Packets

In addition to the TLPs, data link layer packets (DLLPs) and physical layer packets (PLPs) are transmitted. Those packets do not contain any user data. DLLPs are used for acknowledgment, flow control and some power management functions. PLPs are sent regularly in order to ensure proper clock synchronization. DLLPs have a size of 8 bytes, PLPs a size of 4 bytes.

DLLPs originate and terminate in the data link layer, PLPs in the physical layer, respectively. These packets are not visible for the application.

PLPs are used to ensure proper clock synchronization. They have a total size of 4 bytes and transfer a so called SKIP ordered set. This PLP must be transmitted regularly every 1180th to 1538th transmitted byte. A PLP cannot be transmitted in the middle of a packet. If a PLP had to be transmitted during a long packet it will be scheduled after the end of this packet.

For every transmitted TLP the receiver must respond with an ACK or NACK. The transmitter must keep the TLP in its replay buffer until it received the corresponding ACK. So if many TLPs are transmitted, a large number of ACK/NACK DLLP may be produced. Note that it is possible for the receiver to acknowledge (or not acknowledge) multiple TLPs with a single ACK/NACK DLLP.

TLPs are only transmitted if the receiver has enough space in its receive buffer. Therefore flow control DLLPs are transmitted continuously in each direction to keep the communication partner informed about the available space of the own receive buffer. If the transmitter knows that its link partner has

enough space for example for 3 TLPs, the transmitter can send the 3 TLPs without waiting for flow control DLLPs from the receiver. After the receiver has processed a TLP and has freed the space in the receive buffer, it sends a new flow control update DLLP.

3.5 Example Calculation

Example 1: 100 memory writes with a payload size of 4 bytes.

This example reflects the situation where the Hpe_IRP writes 32 bit registers of (maybe multiple) IP cores in the FPGA in a short interval. This scenario is typical for IP cores where small amounts of data are handled. Examples for such IP cores are: UART, CAN, Timer/Counter, GPIO, and I²C.

- System parameters
 - Throughput after physical layer: 250 MByte/s
 - => 1 Byte is transmitted every 4ns
- Calculate the single packet transfer times
 - TLP with 4 bytes data, 32 bit addressing and no ECRC:
(4 bytes payload + 20 bytes overhead)*4ns = 96ns
 - DLLP
8 bytes * 4ns = 32ns
 - PLP
4 bytes * 4ns = 16ns
- Estimate the amount of non-data carrying packets over all packets
 - 2 DLLPs for each TLP: 1 ACK and 1 flow control update packet

Note: The DLLPs travel in the opposite direction compared to the TLP

- 1 PLP after every 1180th to 1538th byte
=> 3 PLPs over all 100 memory writes
- Calculate Throughput
 - Total payload
100 memory writes * 4 bytes = 400 bytes
 - Total transfer time
(100 * 96ns) + (3 * 16ns) = 9648ns
 - Net write throughput
400 bytes / 9648ns = 41.46 Mbytes/s
 - Allocated write bandwidth:
2412 bytes / 9648ns = 250 Mbytes/s (100%)
 - Allocated read bandwidth:
((8+8)*100) bytes / 9648ns = 165.8 Mbytes/s (66.3%)

Cycle	TX	RX
000:	D 00 K STP	MWr
001:	D 00 D 0F	SeqNr. : 4057
002:	D 00 D D9	
003:	D 00 D 40	RequesterID : 0 1c 0
004:	D 00 D 00	Tag : 0
005:	D 00 D 00	Last DW BE : 0x0
006:	D 00 D 01	First DW BE : 0xf
007:	D 00 D 00	Address : 0xc2000000

Remote Lab Demonstration Tutorial
Theoretical throughput of PCIe

008:		D	00	D	E0		
009:	UpdateFC_P	K	SDP	D	00	Attr	: RO- NS- EP-
010:		D	80	D	0F	Length	: 1 DWs
011:	VC : 0	D	0D	D	C2	Data	: 0x00 0x00 0x00 0x00
012:	HdrFC : 54	D	87	D	00	Digest(ECRC):	No ECRC
013:	DataFC : 1973	D	B5	D	00	LCRC	: 0x0684C516
014:		D	EF	D	00		
015:	CRC16 : 0xefc7	D	C7	D	00		
016:		K	END	D	00		
017:	Ack	K	SDP	D	00		
018:		D	00	D	00		
019:	SeqNum: 4057	D	00	D	06		
020:		D	0F	D	84		
021:	CRC16 : 0x40de	D	D8	D	C5		
022:		D	40	D	16		
023:		D	DE	K	END		
024:		K	END	K	STP		MWr
025:		D	00	D	0F	SeqNr.	: 4058
026:		D	00	D	DA		
027:		D	00	D	40	RequesterID	: 0 1c 0
028:		D	00	D	00	Tag	: 0
029:		D	00	D	00	Last DW BE	: 0x0
030:		D	00	D	01	First DW BE	: 0xf
031:	UpdateFC_P	K	SDP	D	00	Address	: 0xc2000004
032:		D	80	D	E0		
033:	VC : 0	D	0D	D	00	Attr	: RO- NS- EP-
034:	HdrFC : 55	D	C7	D	0F	Length	: 1 DWs
035:	DataFC : 1974	D	B6	D	C2	Data	: 0x01 0x00 0x00 0x00
036:		D	E0	D	00	Digest(ECRC):	No ECRC
037:	CRC16 : 0xe085	D	85	D	00	LCRC	: 0xE7B8E3FD
038:		K	END	D	04		
039:	Ack	K	SDP	D	01		
040:		D	00	D	00		
041:	SeqNum: 4058	D	00	D	00		
042:		D	0F	D	00		
043:	CRC16 : 0xe1c5	D	D9	D	E7		
044:		D	E1	D	B8		
045:		D	C5	D	E3		
046:		K	END	D	FD		
047:		D	00	K	END		
048:		D	00	K	STP		MWr
049:		D	00	D	0F	SeqNr.	: 4059
050:		D	00	D	DB		
051:		D	00	D	40	RequesterID	: 0 1c 0
052:		D	00	D	00	Tag	: 0
053:		D	00	D	00	Last DW BE	: 0x0
054:		D	00	D	01	First DW BE	: 0xf
055:		D	00	D	00	Address	: 0xc2000008
056:	UpdateFC_P	K	SDP	D	E0		
057:		D	80	D	00	Attr	: RO- NS- EP-
058:	VC : 0	D	0E	D	0F	Length	: 1 DWs
059:	HdrFC : 56	D	07	D	C2	Data	: 0x02 0x00 0x00 0x00
060:	DataFC : 1975	D	B7	D	00	Digest(ECRC):	No ECRC
061:		D	78	D	00	LCRC	: 0xF2632622
062:	CRC16 : 0x788e	D	8E	D	08		
063:		K	END	D	02		
064:	Ack	K	SDP	D	00		
065:		D	00	D	00		
066:	SeqNum: 4059	D	00	D	00		
067:		D	0F	D	F2		

068: CRC16 : 0x2e9	D	DA	D	63	
069:	D	02	D	26	
070:	D	E9	D	22	
071:	K	END	K	END	

This record goes over 288ns (72*4ns) and shows 3 memory write TLPs (MWr) with 4 Byte data payload each and the according DLLPs. In the 288ns, 12 bytes of user data are transmitted. This is a write throughput of 41.66MByte/s which is slightly better than calculated. This is because no PLP was transmitted during the recorded part of the transmission.

In addition to the TLPs, three DLLPs (UpdateFC_P) with 8 bytes each and three ACK DLLPs with 8 bytes each were transmitted in the TX direction. So 66.6% (48Byte/72Byte) of the read bandwidth is allocated. The slight difference to the calculation above is due to the absence of PLPs in the recorded part of the transmission. (PLPs occur every 1180th to 1538th byte.)

Example 2: 100 memory reads with a payload size of 4 bytes.

This example reflects the situation where the Hpe_IRP reads 32 bit registers of (maybe multiple) IP cores in the FPGA in a short interval. This scenario is typical for IP cores where small amounts of data are handled. Examples for such IP cores are: UART, CAN, Timer/Counter, GPIO, and I²C.

In this example we assume no new read request will be started before the previous one has completed.

- System parameters
 - **Throughput after physical layer:** 250 MByte/s
=> 1 Byte is transmitted every 4ns
- Calculate the single packet transfer times
 - Memory read request TLP 32 bit addressing
(0 Byte payload + 20 bytes overhead)*4ns = 80ns
Note: The request travels in "write" direction.
 - Completion TLP (answer to the read request) no ECRC
(4 Byte payload + 20 Byte overhead)*4ns = 96ns
 - DLLP
8 Byte * 4ns = 32ns
 - PLP
4 Byte * 4ns = 16ns
- Estimate the amount of non-data carrying packets over all packets
 - 2 DLLPs for each request TLP: 1 ACK and 1 flow control update packet
 - 1 DLLP for each completion TLP: 1 ACK
Note: The ACK travels in "write" direction.
 - 1 PLP after every 1180th to 1538th byte for each direction
→ 2x 3 PLPs over all 100 completion TLPs (3 in each direction)
 - Receiver incoming read to completion generation time: 500ns
→ During this time the DLLPs in the read direction and all PLPs can be transmitted
- Calculate Throughput
 - Total payload
100 memory reads * 4 bytes = 400 bytes
 - Total transfer time
(100 * 80ns) + (100 * 500ns) + (100*32ns) + (100 * 96 ns) = 70 800ns
 - **Net read throughput**
400 bytes / 70 800ns = **5.65 MByte/s**
Allocated read bandwidth:
(24*100 + (8+8)*100 + 4*3) bytes / 70 800ns = 56.67 MByte/s (22.67%)

Remote Lab Demonstration Tutorial
Theoretical throughput of PCIe

Allocated write bandwidth:
 (20*100 + 8*100 + 4*3) bytes / 70 800ns = 39.72 MByte/s (15.89%)

Cycle	TX		RX
00000:		D 00 K STP	MRd
00001:		D 00 D 08	SeqNr. : 2123
00002:		D 00 D 4B	
00003:		D 00 D 00	RequesterID: 0 1c 0
00004:		D 00 D 00	Tag : 0
00005:		D 00 D 00	Last DW BE : 0x0
00006:		D 00 D 01	First DW BE: 0xf
00007:		D 00 D 00	Address : 0xc2000000
00008:		D 00 D E0	
00009:		D 00 D 00	Attr : RO- NS- EP-
00010:		D 00 D 0F	Length : 1 DWs
00011:		D 00 D C2	Data : No data.
00012:		D 00 D 00	Digest(ECRC): No ECRC
00013:		D 00 D 00	LCRC : 0xF14A69F2
00014:		D 00 D 00	
00015:		D 00 D F1	
00016:		D 00 D 4A	
00017:		D 00 D 69	
00018:		D 00 D F2	
00019:		D 00 K END	
00020:		D 00 K COM	SKIP
00021:		D 00 K SKP	
00022:		D 00 K SKP	
00023:		D 00 K SKP	
00024:		D 00 D 00	
...			
00030:		D 00 D 00	
00031:	Ack	K SDP D 00	
00032:		D 00 D 00	
00033:	SeqNum: 2123	D 00 D 00	
00034:		D 08 D 00	
00035:	CRC16 : 0x8926	D 4B D 00	
00036:		D 89 D 00	
00037:		D 26 D 00	
00038:		K END D 00	
00039:	UpdateFC_NP	K SDP D 00	
00040:		D 90 D 00	
00041:	VC : 0	D 3C D 00	
00042:	HdrFC : 242	D 81 D 00	
00043:	DataFC : 511	D FF D 00	
00044:		D 20 D 00	
00045:	CRC16 : 0x202d	D 2D D 00	
00046:		K END D 00	
00047:		D 00 D 00	
...			
00144:		D 00 D 00	
00145:	Cp1D	K STP D 00	
00146:	SeqNr. : 1713	D 06 D 00	
00147:		D B1 D 00	
00148:	CompleterID : 2 0 0	D 4A D 00	
00149:	CompletionStatus: Successful	D 00 D 00	
00150:	BCM : 0x0	D 00 D 00	
00151:	ByteCount : 0x4	D 01 D 00	
00152:	RequesterID : 0 1c 0	D 02 D 00	
00153:	Tag : 0	D 00 D 00	

Remote Lab Demonstration Tutorial
Theoretical throughput of PCIe

00154:	LowerAddress	:	0x80	D	00	D	00	
00155:				D	04	D	00	
00156:	Attr	:	RO- NS- EP-	D	00	D	00	
00157:	Length	:	1 DWs	D	E0	D	00	
00158:	Data	:	0x63 0x00 0x0	D	00	D	00	
00159:	Digest(ECRC)	:	No ECRC	D	80	D	00	
00160:	LCRC	:	0x4CCCE640	D	63	D	00	
00161:				D	00	D	00	
00162:				D	00	D	00	
00163:				D	00	D	00	
00164:				D	4C	D	00	
00165:				D	CC	D	00	
00166:				D	E6	D	00	
00167:				D	40	D	00	
00168:				K	END	D	00	
00169:				D	00	K	SDP	Ack
00170:				D	00	D	00	
00172:				D	00	D	00	SeqNum: 1713
00173:				D	00	D	06	
00174:				D	00	D	B1	CRC16 : 0x8a66
00175:				D	00	D	8A	
00176:				D	00	D	66	
00177:				D	00	K	END	
00178:				D	00	K	STP	MRd
00179:				D	00	D	08	SeqNr. : 2124
00180:				D	00	D	4C	
00181:				D	00	D	00	RequesterID: 0 1c 0
00182:				D	00	D	00	Tag : 0
00183:				D	00	D	00	Last DW BE : 0x0
00184:				D	00	D	01	First DW BE: 0xf
00185:				D	00	D	00	Address : 0xc2000004
00186:				D	00	D	E0	
00187:				D	00	D	00	Attr : RO- NS- EP-
00188:				D	00	D	0F	Length : 1 DWs
00189:				D	00	D	C2	Data : No data.
00190:				D	00	D	00	Digest(ECRC): No ECRC
00191:				D	00	D	00	LCRC : 0xB0887788
00192:				D	00	D	04	
00193:				D	00	D	B0	
00194:				D	00	D	88	
00195:				D	00	D	77	
00196:				D	00	D	88	
00197:				D	00	K	END	
00198:				D	00	D	00	
...								
00206:				D	00	D	00	
00207:				K	SDP	D	00	Ack
00208:				D	00	D	00	
00209:	SeqNum:	:	2124	D	00	D	00	
00210:				D	08	D	00	
00211:	CRC16	:	0xee64	D	4C	D	00	
00212:				D	EE	D	00	
00213:				D	64	D	00	
00214:				K	END	D	00	
00215:				K	SDP	D	00	UpdateFC_NP
00216:				D	90	D	00	
00217:	VC	:	0	D	3C	D	00	
00218:	HdrFC	:	243	D	C1	D	00	
00219:	DataFC	:	511	D	FF	D	00	
00220:				D	CC	D	00	

Remote Lab Demonstration Tutorial
Theoretical throughput of PCIe

00221:	CRC16 : 0xcc43	D	43	D	00	
00222:		K	END	D	00	
00223:		D	00	D	00	
...						
00322:		D	00	D	00	
00323:		K	STP	D	00	
00324:	SeqNr. : 1714	D	06	D	00	
00325:		D	B2	D	00	
00326:	CompleterID : 2 0 0	D	4A	D	00	
00327:	CompletionStatus: Successful	D	00	D	00	
00328:	BCM : 0x0	D	00	D	00	
00329:	ByteCount : 0x4	D	01	D	00	
00330:	RequesterID : 0 1c 0	D	02	D	00	
00331:	Tag : 0	D	00	D	00	
00333:	LowerAddress : 0x84	D	00	D	00	
00334:		D	04	D	00	
00335:	Attr : RO- NS- EP-	D	00	D	00	
00336:	Length : 1 DWs	D	E0	D	00	
00337:	Data : 0x00 0x00 0x0	D	00	D	00	
00338:	Digest(ECRC): No ECRC	D	84	D	00	
00339:	LCRC : 0xE5696FCF	D	00	D	00	
00340:		D	00	D	00	
00341:		D	00	D	00	
00342:		D	00	D	00	
00343:		D	E5	D	00	
00344:		D	69	D	00	
00345:		D	6F	D	00	
00346:		D	CF	D	00	
00347:		K	END	D	00	
00348:		D	00	K	SDP	Ack
00349:		D	00	D	00	
00350:		D	00	D	00	SeqNum: 1714
00351:		D	00	D	06	
00352:		D	00	D	B2	CRC16 : 0x694a
00353:		D	00	D	69	
00354:		D	00	D	4A	
00355:		D	00	K	END	
00356:		D	00	K	STP	MRd
00357:		D	00	D	08	SeqNr. : 2125
00358:		D	00	D	4D	
00359:		D	00	D	00	RequesterID: 0 1c 0
00360:		D	00	D	00	Tag : 0
00361:		D	00	D	00	Last DW BE : 0x0
00362:		D	00	D	01	First DW BE: 0xf
00363:		D	00	D	00	Address : 0xc2000008
00364:		D	00	D	E0	
00365:		D	00	D	00	Attr : RO- NS- EP-
00366:		D	00	D	0F	Length : 1 DWs
00367:		D	00	D	C2	Data : No data.
00368:		D	00	D	00	Digest(ECRC): No ECRC
00369:		D	00	D	00	LCRC : 0x1E1D575C
00370:		D	00	D	08	
00371:		D	00	D	1E	
00372:		D	00	D	1D	
00373:		D	00	D	57	
00374:		D	00	D	5C	
00375:		D	00	K	END	
00376:		D	00	D	00	
...						
00381:		D	00	D	00	

Remote Lab Demonstration Tutorial
Theoretical throughput of PCIe

00382:	Ack	K	SDP	D	00	
00383:		D	00	D	00	
00384:	SeqNum: 2125	D	00	D	00	
00385:		D	08	D	00	
00386:	CRC16 : 0x4f7f	D	4D	D	00	
00387:		D	4F	D	00	
00388:		D	7F	D	00	
00389:		K	END	D	00	
00390:	UpdateFC_NP	K	SDP	D	00	
00391:		D	90	D	00	
00392:	VC : 0	D	3D	D	00	
00393:	HdrFC : 244	D	01	D	00	
00394:	DataFC : 511	D	FF	D	00	
00395:		D	0C	D	00	
00396:	CRC16 : 0xc0e	D	0E	D	00	
00397:		K	END	D	00	
...						
00495:		D	00	D	00	
00501:	Cp1D	K	STP	D	00	
00502:	SeqNr. : 1715	D	06	D	00	
00503:		D	B3	D	00	
00504:	CompleterID : 2 0 0	D	4A	D	00	
00505:	CompletionStatus: Successful	D	00	D	00	
00506:	BCM : 0x0	D	00	D	00	
00507:	ByteCount : 0x4	D	01	D	00	
00508:	RequesterID : 0 1c 0	D	02	D	00	
00509:	Tag : 0	D	00	D	00	
00510:	LowerAddress : 0x88	D	00	D	00	
00511:		D	04	D	00	
00512:	Attr : RO- NS- EP-	D	00	D	00	
00513:	Length : 1 Dws	D	E0	D	00	
00514:	Data : 0x00 0x00 0x0	D	00	D	00	
00515:	Digest(ECRC): No ECRC	D	88	D	00	
00516:	LCRC : 0xA74F398D	D	00	D	00	
00517:		D	00	D	00	
00518:		D	00	D	00	
00519:		D	00	D	00	
00520:		D	A7	D	00	
00521:		D	4F	D	00	
00522:		D	39	D	00	
00523:		D	8D	D	00	
00524:		K	END	D	00	
00525:		D	00	K	SDP	Ack
00526:		D	00	D	00	
00527:		D	00	D	00	SeqNum: 1715
00528:		D	00	D	06	
00529:		D	00	D	B3	CRC16 : 0xc851
00530:		D	00	D	C8	
00531:		D	00	D	51	
00532:		D	00	K	END	

This trace goes over 2128ns (523 * 4ns). In this time three memory request TLPs, the corresponding completion TLPs, DLLPs and one PLP in each direction are transmitted. Also the incoming read to completion generation time (500ns) can be seen.

In this 2128ns 12 bytes are read, which results in a throughput of 5.639 MByte/s. This is slightly slower than calculated, because a PLP occurs in this record.

Remote Lab Demonstration Tutorial

Theoretical throughput of PCIe

In read-direction three memory completion TLPs (CpID) with 24 Byte, three ACK DLLPs with 8 Byte, three Update DLLPs with 8 Byte and one PLP with 4 Byte were transmitted, this is a read bandwidth allocation of 23.26% (124/533).

In write-direction three memory read request TLPs (MRd) with 20 Byte, three ACK DLLPs with 8 Byte and 1 PLP with 4 Byte are sent. This is a write bandwidth allocation of 16.51 % (88/533).

Note: PLPs occur every 1180th to 1538th byte.

3.6 Summary

Note: The reflection of the theoretical throughput of PCIe shows that the bandwidth of the PCIe connection is 250 MByte/s. To reach a throughput close to this value it is essential to use large payload sizes. Otherwise the protocol overhead becomes dominant.

The theory shows that the read throughput is much slower than the write throughput. This is because a memory read consists of two TLPs: a memory read request and a completion TLP. Also the incoming read to completion generation time has a significant influence on the read throughput. In high performance configurations a new read request can be started before the last completion was received. This pipelining will increase the read throughput significantly.

4 Measuring the throughput of CPU initiated PCIe transfers

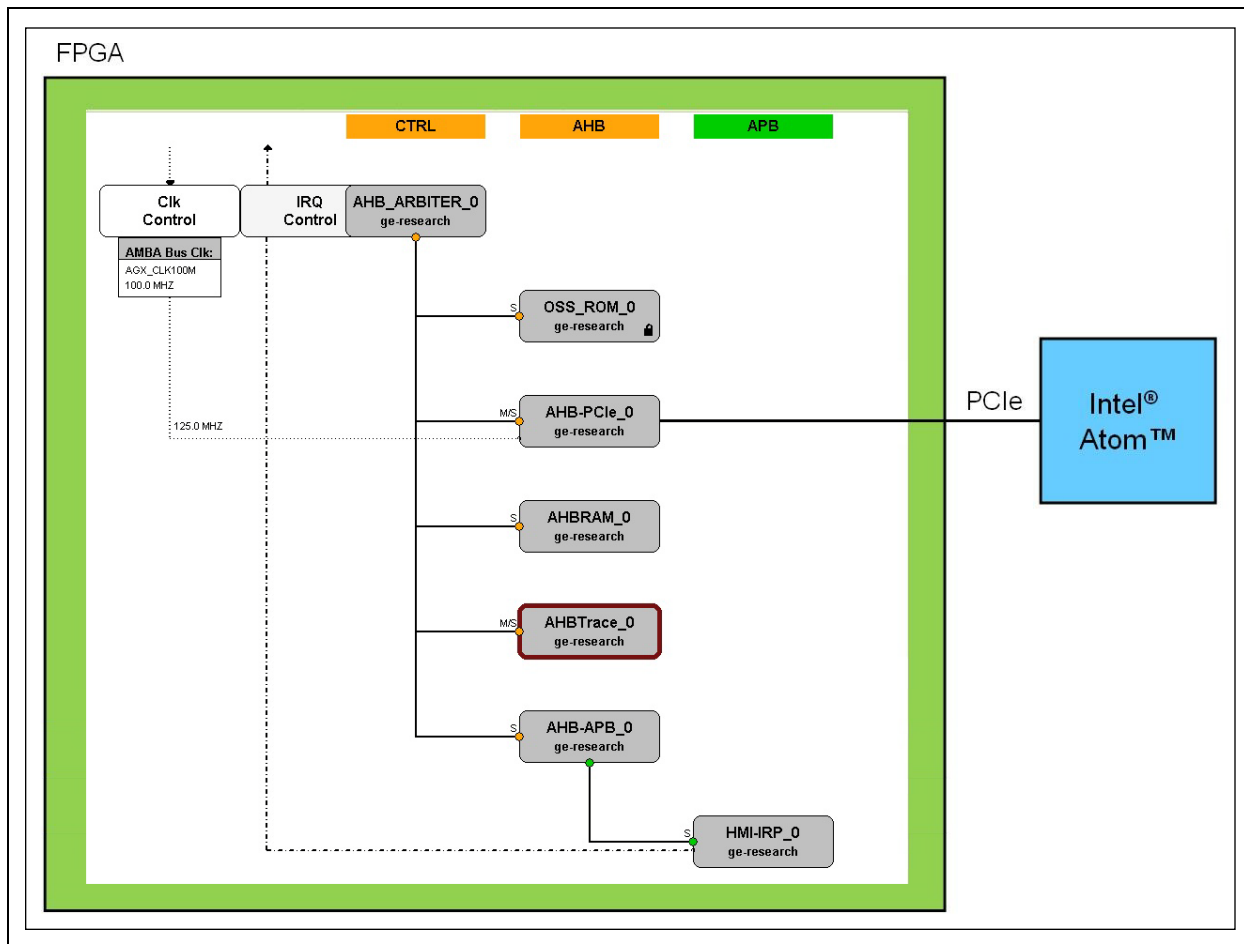
In this section we will measure the real-world performance of PCIe. We will use the PCIe connection between the Altera* FPGA and the Qseven* module on the Hpe_IRP. In this section all data transfers will be initiated by the CPU as assumed in the example shown in Section 3.5.

4.1 Measurement setup

The measurement setup consists of an AHB-to-PCIe bridge, an AHB RAM, an AHB-trace module and a test program running on the Intel® Atom™ processor. Figure 3 shows the measurement setup.

The test program can write to and read from the AHB RAM over PCIe and can control the AHB-trace module. The AHB-trace module traces all accesses to the AHB RAM. The AHB-trace module also stores a timestamp for every access to the AHB RAM. The HMI-IRP IP core is needed to have access to the LCD on the front panel of the Hpe_IRP and has no influence on the measured throughput.

Figure 3. Measurement setup



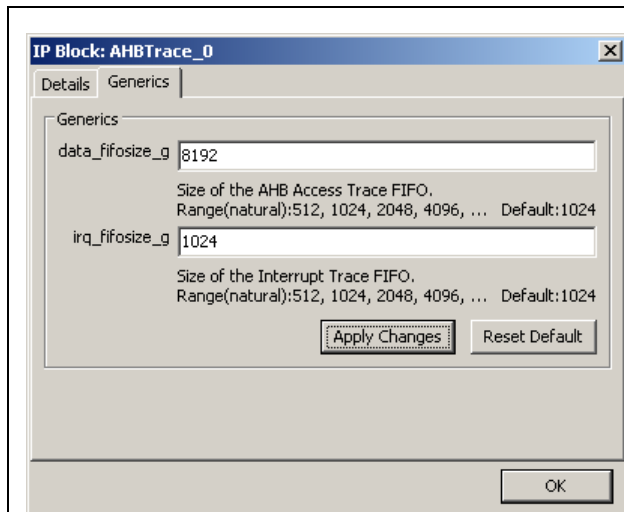
With this setup it is possible to record the delays between the accesses to the AHB RAM. This does not show the exact throughput of the PCIe transmission itself, but shows the throughput of the end-to-end connection from the processor to an IP core.

The test program maps the AHB RAM into the user space. The AHB RAM is then accessed as an unsigned int array or as double array. Writing to the int array produces a memory write TLP with a payload size of a 4 bytes (as in Example 1 of Section 3.5). Writing to the double array produces a memory write TLP with a payload size of 8 bytes. Reading from the arrays produces a memory read request TLP and waits for the completion TLP.

4.2 Synthesizing a design for the measurement setup

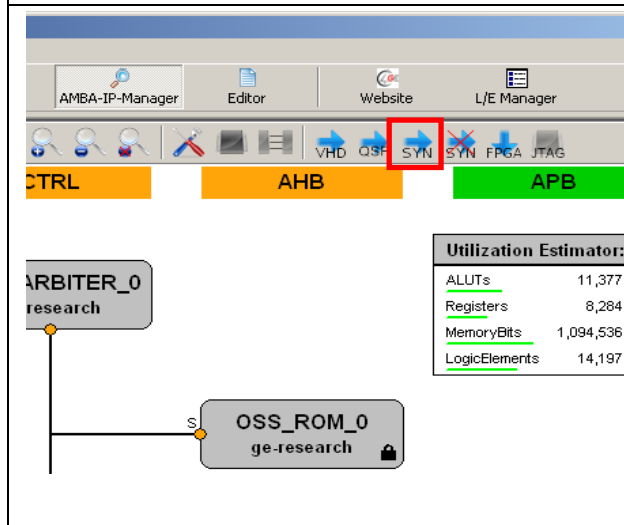
For details on how to create a design with Hpe_Desk, refer to Section 1.2 reference [4]. In this tutorial only the configuration of the IP cores is explained in detail.

Remote Lab Demonstration Tutorial
Measuring the throughput of CPU initiated PCIe transfers



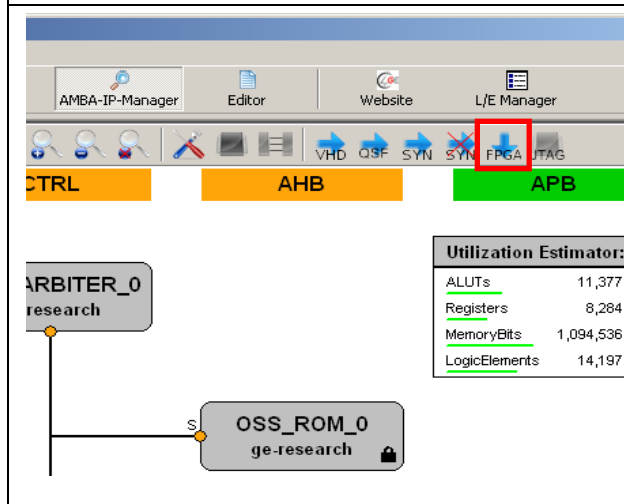
Step 3

Open the IP details menu of the AHB Trace module and choose the generics tab. Set the data_fifo_size_g to 8192 and the interrupt_fifo_size to 1024.
 (With 8192 AHB accesses an AHB RAM with 32768 Byte can be filled up.)
 Click "Apply Changes".



Step 4

Synthesize the design



Step 5

Download the design

4.3 Test program

The test program offers different tests to measure the throughput. The test program can be found under `Hpe_IRP_support/Demo_designs`. To unpack and compile it, type the following commands:

```
tar xjf Hpe_IRP_support/Demo_designs/throughput-demo-0.2.tar.bz2
cd throughput-demo-0.2
make
```

In this directory you can also find the source code of the test program. Feel free to change the code or to add your own code if you want to create your own test scenarios.

To start the test program, go to the directory from above and type:

```
./throughput
```

This brings you to the main menu of the test program

```
Throughput measurement menu:
-----
32 bit Accesses:
 [1] Test 1:      Two write accesses to the AHB RAM
 [2] Test 2:      Two read  accesses from the AHB RAM
 [3] Test 3:      n write accesses to the AHB RAM
 [4] Test 4:      n read  accesses from the AHB RAM
 [5] Test 5:      n write + read accesses to the AHB RAM
 [6] Test 6:      n write accesses to the first AHB RAM
 [7] Test 7:      n read  accesses from the first AHB RAM
 [8] Test 8:      Measures the time for writing (64 x n) 32 bit values to a AHB RAM
 [9] Test 9:      Measures the time for reading (64 x n) 32 bit values from a AHB RAM
64 bit Accesses:
 [10] Test 10:     n write accesses to the AHB RAM with 64 Bit
 [11] Test 11:     n read  accesses from the AHB RAM with 64 Bit
 [12] Test 12:     Measures the time for writing (64 x n) 64 bit values to a AHB RAM
 [13] Test 13:     Measures the time for reading (64 x n) 64 bit values from a AHB RAM
different payload sizes:
 [14] Test 14:     Test and compare write accesses with 1, 2, 4 and 8 Byte payload size
 [15] Test 15:     Test and compare read  accesses with 1, 2, 4 and 8 Byte payload size
Settings:
 [16] Test 16:     Tests and calibrates the stopwatch
 [17] Activate:    log file generation
 [18] Deactivate:  log file generation

Exit:
 [0] Exit
Choose:
```

4.4 Test 1: Two Write Accesses

Test 1 tries to produce two write accesses to the AHB RAM as fast as possible. Therefore following code is used:

```
RAM[0] = 0;
RAM[1] = 1;
```

When you run Test 1, you get an output like this:

```
Test1:
Two Write Accesses
-----
Press any key to start

timestamp:      543 [          0 ns] addr: 0x02000000, data: 0000000000, flags: 0x0105 WRITE
timestamp:      554 [        110 ns] addr: 0x02000004, data: 0000000001, flags: 0x0105 WRITE
```

Remote Lab Demonstration Tutorial

Measuring the throughput of CPU initiated PCIe transfers

```
Statistic for 2 AHB accesses:
-----
Every PCIe TLP Packet ends up in 1 AHB accesses
Number of Interrupts during the transmission: 0

average elapsed Time:      110.00 ns
min elapsed Time:         110.00 ns
max elapsed Time:         110.00 ns

average throughput:      36363636.36 Bytes/s
max throughput:          36363636.36 Bytes/s
min throughput:          36363636.36 Bytes/s

Press any key for next test.
```

The lines starting with "timestamp:" are the results from the AHB trace module. The first column gives the actual timestamp, i.e. the clock cycle in which this access happened. The value in the square brackets is the time difference between the current and the previous AHB access. The values of `addr` and `data` represent the address and data of the access. The flags following the data field are not relevant for this tutorial. The last field in each row indicates the direction of the access, i.e. read or write.

In the output shown above, the second AHB RAM write access starts 110ns after the first AHB RAM access. With every AHB RAM access 4 data bytes are written to the AHB RAM. When you run the test 1 several times, you would get different results. It is most likely to get a delay between 110ns and 130ns, but also delays between 140ns and 170ns, or delays between 240ns and 340ns or even delays with about 5 μ s, 13 μ s or 44 μ s are possible.

This brings up the following questions, which will be answered with the help of the following tests:

- How long can the shortest delay be (best case)?
- How long can the longest delay be (worst case)?
- Which delays are most likely to occur?
- What are the reasons for the different types of delays?
- Who often do long delays occur?
- Are there any periods in which different types of delays occur?

How long is the shortest delay between two write accesses?

Because in this test every AHB RAM access is the result of one TLP with a data payload size of 4 Bytes the theoretical throughput of PCIe is 41.66Mbyte/s (Example 1, Section 3.5). This means every 96ns a TLP packet arrives at the input of the PCIe IP core theoretically under best-case conditions.

The AHB runs with 100MHz. This is why only every 10ns an access to the AHB RAM is possible. The PCIe IP core runs with 125MHz. Therefore synchronization between these two clock domains is required. The synchronization is implemented with a FIFO.

The measurement shows that the shortest measured delay between two AHB RAM accesses is 110ns.

4.5 Test 2: Two Read Accesses

Test2 tries to produce two read accesses to the AHB RAM as fast as possible. Therefore following code is used.

```
ret = RAM[0];
ret = RAM[1];
```

When you run Test 2, you get an output like this:

```
Test2:
Two Read Accesses
-----
Press any key to start

timestamp: 7663260 [ 0.00 ns] addr: 0x02000000, data: 0000000000, flags: 0x0104 READ
timestamp: 7663422 [ 1520.00 ns] addr: 0x02000004, data: 0000000001, flags: 0x0104 READ

Statistic for 2 AHB accesses:
-----
Every PCIe TLP Paket ends up in 1 AHB accesses
Number of Interrupts during the transmission: 0

average elapsed Time: 1520.00 ns
min elapsed Time: 1520.00 ns
max elapsed Time: 1520.00 ns

average throughput: 2631578.95 Bytes/s
max throughput: 2631578.95 Bytes/s
min throughput: 2631578.95 Bytes/s

Press any key for next test.
```

When you run Test 2 several times you would get many different results. Test 2 shows that the delay between two read accesses is much longer than between two write accesses. This is because a memory read involves two TLPs. A read request TLP will be answered by a completion TLP.

4.6 Test 3: Multiple Writes

Test 3 allows you to trace more than two write accesses to the AHB RAM. At the beginning of the test you can choose how many write accesses you want to test. Because of the limit of the RAM size you can only test $\text{RAM_SIZE} / 4$ (=8192 if you followed Section 4.2, Step 2) accesses. For writing to the AHB RAM following code is used:

```
for (int i=0; i < n; i++) {
    RAM[i] = i;
}
```

For 30 write accesses you would get an output like this:

```
timestamp: 1852 [ 0 ns] addr: 0x02000000, data: 0000000000, flags: 0x0105 WRITE
timestamp: 1873 [ 210 ns] addr: 0x02000004, data: 0000000001, flags: 0x0105 WRITE
timestamp: 1884 [ 110 ns] addr: 0x02000008, data: 0000000002, flags: 0x0105 WRITE
timestamp: 1897 [ 130 ns] addr: 0x0200000c, data: 0000000003, flags: 0x0105 WRITE
timestamp: 1908 [ 110 ns] addr: 0x02000010, data: 0000000004, flags: 0x0105 WRITE
timestamp: 1921 [ 130 ns] addr: 0x02000014, data: 0000000005, flags: 0x0105 WRITE
timestamp: 1932 [ 110 ns] addr: 0x02000018, data: 0000000006, flags: 0x0105 WRITE
timestamp: 1948 [ 160 ns] addr: 0x0200001c, data: 0000000007, flags: 0x0105 WRITE
timestamp: 1959 [ 110 ns] addr: 0x02000020, data: 0000000008, flags: 0x0105 WRITE
timestamp: 1972 [ 130 ns] addr: 0x02000024, data: 0000000009, flags: 0x0105 WRITE
timestamp: 1983 [ 110 ns] addr: 0x02000028, data: 0000000010, flags: 0x0105 WRITE
timestamp: 1996 [ 130 ns] addr: 0x0200002c, data: 0000000011, flags: 0x0105 WRITE
timestamp: 2008 [ 120 ns] addr: 0x02000030, data: 0000000012, flags: 0x0105 WRITE
timestamp: 2020 [ 120 ns] addr: 0x02000034, data: 0000000013, flags: 0x0105 WRITE
timestamp: 2032 [ 120 ns] addr: 0x02000038, data: 0000000014, flags: 0x0105 WRITE
timestamp: 2044 [ 120 ns] addr: 0x0200003c, data: 0000000015, flags: 0x0105 WRITE
timestamp: 2068 [ 240 ns] addr: 0x02000040, data: 0000000016, flags: 0x0105 WRITE
timestamp: 2080 [ 120 ns] addr: 0x02000044, data: 0000000017, flags: 0x0105 WRITE
timestamp: 2092 [ 120 ns] addr: 0x02000048, data: 0000000018, flags: 0x0105 WRITE
timestamp: 2104 [ 120 ns] addr: 0x0200004c, data: 0000000019, flags: 0x0105 WRITE
timestamp: 2117 [ 130 ns] addr: 0x02000050, data: 0000000020, flags: 0x0105 WRITE
timestamp: 2128 [ 110 ns] addr: 0x02000054, data: 0000000021, flags: 0x0105 WRITE
timestamp: 2141 [ 130 ns] addr: 0x02000058, data: 0000000022, flags: 0x0105 WRITE
timestamp: 2157 [ 160 ns] addr: 0x0200005c, data: 0000000023, flags: 0x0105 WRITE
```

Remote Lab Demonstration Tutorial

Measuring the throughput of CPU initiated PCIe transfers

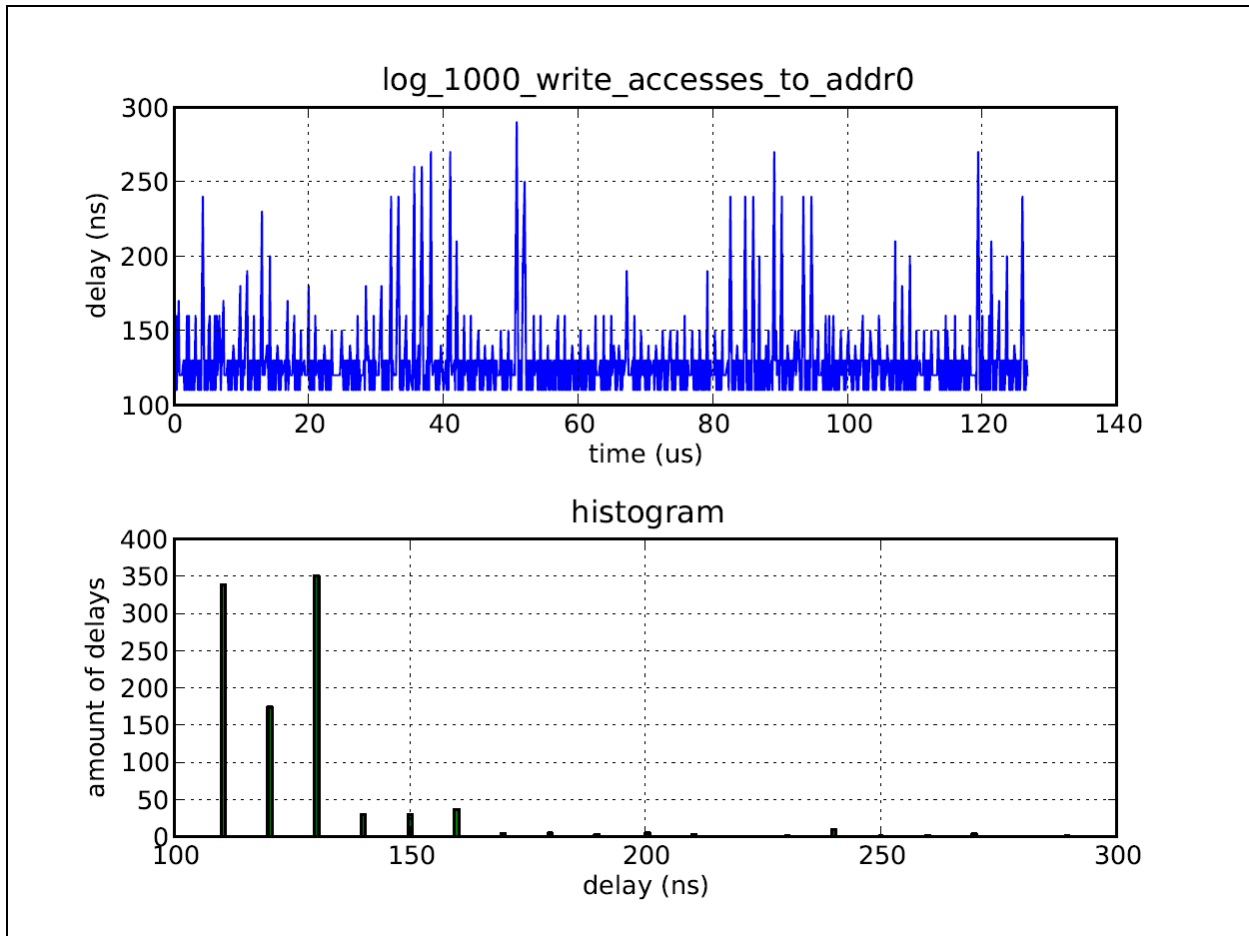
```
timestamp:    2171 [    140 ns] addr: 0x02000060, data: 0000000024, flags: 0x0105 WRITE
timestamp:    2182 [    110 ns] addr: 0x02000064, data: 0000000025, flags: 0x0105 WRITE
timestamp:    2195 [    130 ns] addr: 0x02000068, data: 0000000026, flags: 0x0105 WRITE
timestamp:    2206 [    110 ns] addr: 0x0200006c, data: 0000000027, flags: 0x0105 WRITE
timestamp:    2219 [    130 ns] addr: 0x02000070, data: 0000000028, flags: 0x0105 WRITE
timestamp:    2230 [    110 ns] addr: 0x02000074, data: 0000000029, flags: 0x0105 WRITE
Statistic for 30 AHB accesses:
-----
Every PCIe TLP Packet ends up in 1 AHB accesses
Number of Interrupts during the transmission: 0

average elapsed Time:      130.34 ns
min elapsed Time:         110.00 ns
max elapsed Time:         240.00 ns

average throughput:      30687830.69 Bytes/s
max throughput:         36363636.36 Bytes/s
min throughput:         16666666.67 Bytes/s
```

In the main menu you can activate the log file generation by choosing 17. For the example above the logfile would be named `log_30_write_accesses.log` and stored under `log`. It is also possible to generate a plot of the result. Therefore exit the test program and go to the `plot` directory. Plots for all log-files under `log` can be generated with `python logToPdf.py`. In this example a plot named `log_30_write_accesses.pdf` would be generated in the `plot` directory and would look like Figure 4 without the green graph plus a histogram of the different delays.

Figure 5. Plot and histogram for 1000 AHB RAM write accesses²



The following record shows three memory write TLPs. It is nearly the same record as in Example 1 in Section 3.5, but with small delays between consecutive packets.

Cycle	TX			RX		
57286:	D 00	K STP				MWr
57287:	D 00	D 05		SeqNr.	:	1281
57288:	D 00	D 01				
57289:	D 00	D 40		RequesterID:	0 1c 0	
57290:	D 00	D 00		Tag	:	0
57291:	D 00	D 00		Last DW BE	:	0x0
57292:	D 00	D 01		First DW BE:	:	0xf

² Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. Configurations: Refer to Section 2.2 of [4] for complete configuration details. For more information go to <http://www.intel.com/performance>

Intel does not control or audit the design or implementation of third party benchmark data or Web sites referenced in this document. Intel encourages all of its customers to visit the referenced Web sites or others where similar performance benchmark data are reported and confirm whether the referenced benchmark data are accurate and reflect performance of systems available for purchase.

Remote Lab Demonstration Tutorial
Measuring the throughput of CPU initiated PCIe transfers

57293:	D	00	D	00	Address	:	0xc2000000
57294:	D	00	D	E0			
57295:	D	00	D	00	Attr	:	RO- NS- EP-
57296:	D	00	D	0F	Length	:	1 DWs
57297:	D	00	D	C2	Data	:	0x01 0x00 0x0
57298:	D	00	D	00	Digest(ECRC)	:	No ECRC
57299:	D	00	D	00	LCRC	:	0xCF011C8F
57300:	D	00	D	00			
57301:	D	00	D	01			
57302:	D	00	D	00			
57303:	D	00	D	00			
57304:	Ack	K	SDP	D	00		
57305:		D	00	D	CF		
57306:	SeqNum: 1280		D	00	D	01	
57307:			D	05	D	1C	
57308:	CRC16 : 0xe123		D	00	D	8F	
57309:			D	E1	K	END	
57310:			D	23	D	00	
57311:			K	END	D	00	
57312:			D	00	D	00	
57313:			D	00	D	00	
57314:			D	00	D	00	
57315:			D	00	D	00	
57316:			D	00	K	STP	MWr
57317:			D	00	D	05	SeqNr. : 1282
57318:			D	00	D	02	
57319:			D	00	D	40	RequesterID: 0 1c 0
57320:			D	00	D	00	Tag : 0
57321:			D	00	D	00	Last DW BE : 0x0
57322:			D	00	D	01	First DW BE: 0xf
57323:			D	00	D	00	Address : 0xc2000004
57324:			D	00	D	E0	
57325:			D	00	D	00	Attr : RO- NS- EP-
57326:			D	00	D	0F	Length : 1 DWs
57327:			D	00	D	C2	Data : 0x02 0x00 0x0
57328:			D	00	D	00	Digest(ECRC): No ECRC
57329:			D	00	D	00	LCRC : 0xA5F533CE
57330:			D	00	D	04	
57331:			D	00	D	02	
57332:	Ack	K	SDP	D	00		
57333:			D	00	D	00	
57334:	SeqNum: 1281		D	00	D	00	
57335:			D	05	D	A5	
57336:	CRC16 : 0x4038		D	01	D	F5	
57337:			D	40	D	33	
57338:			D	38	D	CE	
57339:			K	END	K	END	
57340:			D	00	D	00	
57341:			D	00	D	00	
57342:			D	00	D	00	
57343:			D	00	D	00	
57344:			D	00	D	00	
57345:			D	00	D	00	
57346:			D	00	K	STP	MWr
57347:			D	00	D	05	SeqNr. : 1283
57348:			D	00	D	03	
57349:			D	00	D	40	RequesterID: 0 1c 0
57350:			D	00	D	00	Tag : 0
57351:			D	00	D	00	Last DW BE : 0x0
57352:			D	00	D	01	First DW BE: 0xf

Remote Lab Demonstration Tutorial
Measuring the throughput of CPU initiated PCIe transfers

57353:		D	00	D	00	Address	:	0xc2000008
57354:		D	00	D	E0			
57355:		D	00	D	00	Attr	:	RO- NS- EP-
57356:		D	00	D	0F	Length	:	1 DWs
57357:		D	00	D	C2	Data	:	0x03 0x00 0x0
57358:		D	00	D	00	Digest(ECRC)	:	No ECRC
57359:		D	00	D	00	LCRC	:	0x835929F1
57360:	UpdateFC_P	K	SDP	D	08			
57361:		D	80	D	03			
57362:	VC : 0	D	17	D	00			
57363:	HdrFC : 93	D	43	D	00			
57364:	DataFC : 988	D	DC	D	00			
57365:		D	61	D	83			
57366:	CRC16 : 0x61d5	D	D5	D	59			
57367:		K	END	D	29			
57368:	Ack	K	SDP	D	F1			
57369:		D	00	K	END			
57370:	SeqNum: 1282	D	00	D	00			
57371:		D	05	D	00			
57372:	CRC16 : 0xa314	D	02	D	00			
57373:		D	A3	D	00			
57374:		D	14	D	00			
57375:		K	END	D	00			
57376:		D	00	D	00			
57377:		D	00	D	00			
57378:		D	00	D	00			
57379:		D	00	D	00			
57380:		D	00	D	00			
57381:		D	00	D	00			
57382:		D	00	D	00			
57383:		D	00	D	00			
57384:		D	00	D	00			
57385:		D	00	D	00			
57386:		D	00	D	00			
57387:		D	00	D	00			
57388:	UpdateFC_P	K	SDP	D	00			
57389:		D	80	D	00			
57390:	VC : 0	D	17	D	00			
57391:	HdrFC : 94	D	83	D	00			
57392:	DataFC : 989	D	DD	D	00			
57393:		D	F4	D	00			
57394:	CRC16 : 0xf47d	D	7D	D	00			
57395:		K	END	D	00			
57396:	Ack	K	SDP	D	00			
57397:		D	00	D	00			
57398:	SeqNum: 1283	D	00	D	00			
57399:		D	05	D	00			
57400:	CRC16 : 0x20f	D	03	D	00			
57401:		D	02	D	00			
57402:		D	0F	D	00			
57403:		K	END	D	00			
57404:		D	00	D	00			
57405:		D	00	D	00			

The record shows a break of 24ns (=6 x 4ns) between every TLP which reduces the practical throughput (120ns between every TLP). The practical throughput in this case would be 33.33MByte/s (= 12 Byte/(3* 30 * 4ns)). The record also shows that the DLLPs were sent in the other direction as the memory write TLPs. This means the DLLPs do not influence the write throughput, but consume read bandwidth.

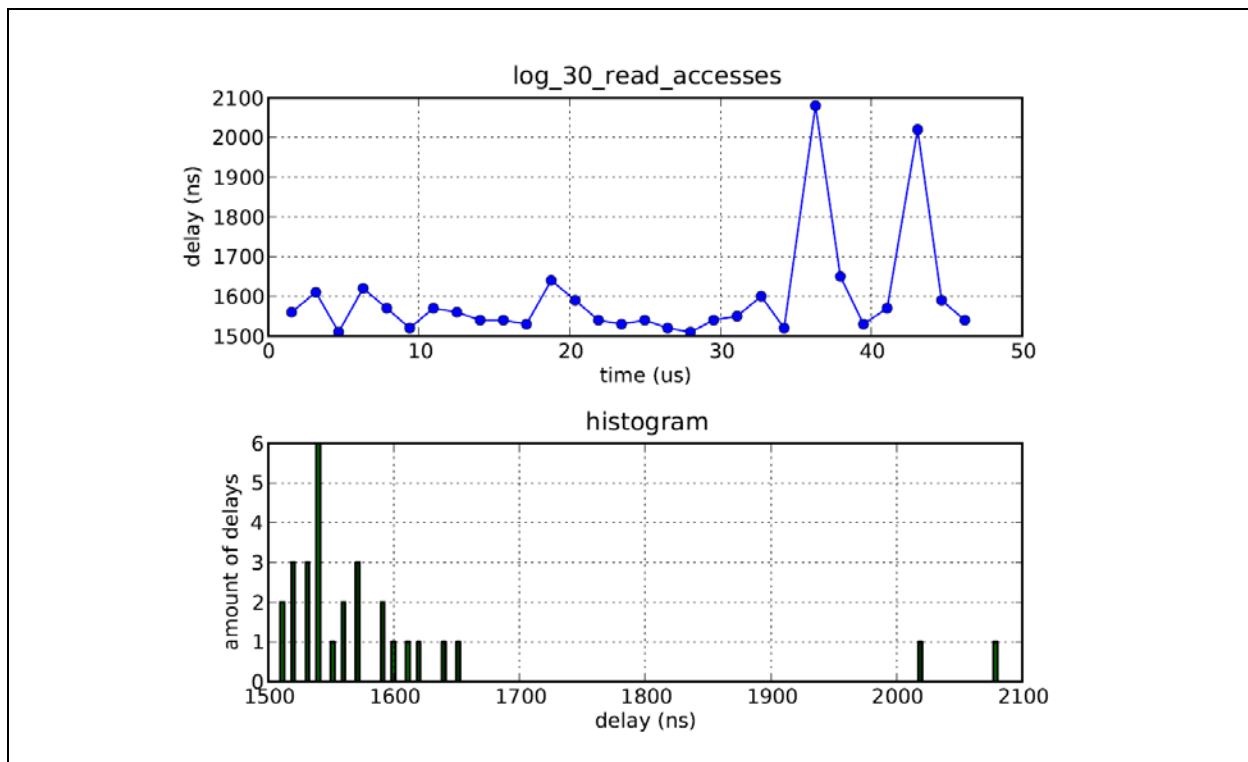
4.7 Test 4: Multiple Reads

Test 4 traces several read accesses to the AHB RAM. At the start of the test you can choose how many read accesses you want to test. Because of the limit of the RAM size you can only test $\text{RAM_SIZE} / 4$ (=8192 if you followed Section 4.2, Step 2) accesses. In this test the following code is used.

```
for (int i=0; i < n; i++) {  
    ret = RAM[i];  
}
```

When you run test 4 with $n=30$ you get a plot and a histogram like shown in Figure 6.

Figure 6. Delays between read accesses³



³ Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. Configurations: Refer to Section 2.2 of [4] for complete configuration details. For more information go to <http://www.intel.com/performance>

Intel does not control or audit the design or implementation of third party benchmark data or Web sites referenced in this document. Intel encourages all of its customers to visit the referenced Web sites or others where similar performance benchmark data are reported and confirm whether the referenced benchmark data are accurate and reflect performance of systems available for purchase.

Remote Lab Demonstration Tutorial

Measuring the throughput of CPU initiated PCIe transfers

The following record shows three memory read request TLPs and the corresponding completion TLPs and DLLPs. It is nearly the same record as in Example 2 in Section 3.5, but with delays between the packets.

To reduce the required space lines without any transactions are replaced by '...'. The cycle counter indicates the elapsed time, where every line records 4ns. SKIP packets are PLPs, which contain skip information for clock synchronization (see Section 3.4).

Cycle	Tx	Rx
57258:		MRd
57259:		SeqNr. : 2123
57260:		
57261:		RequesterID: 0 1c 0
57262:		Tag : 0
57263:		Last DW BE : 0x0
57264:		First DW BE: 0xf
57265:		Address : 0xc2000000
57266:		
57267:		Attr : RO- NS- EP-
57268:		Length : 1 DWs
57269:		Data : No data.
57270:		Digest(ECRC): No ECRC
57271:		LCRC : 0xF14A69F2
57272:		
57273:		
57274:		
57275:		
57276:		
57277:		
57278:		SKIP
57279:		
57280:		
57281:		
57282:		
...		
57301:		
57302:	Ack	
57303:		
57304:	SeqNum: 2123	
57305:		
57306:	CRC16 : 0x8926	
57307:		
57308:		
57309:		
57310:		
...		
57361:		
57362:	UpdateFC_NP	
57363:		
57364:	VC : 0	
57365:	HdrFC : 242	
57366:	DataFC : 511	
57367:		
57368:	CRC16 : 0x202d	
57369:		
57370:		
...		
57405:		
57406:	Cp1D	
57407:	SeqNr. : 1713	

Remote Lab Demonstration Tutorial
Measuring the throughput of CPU initiated PCIe transfers

57408:		D	B1	D	00	
57409: CompleterID	: 2 0 0	D	4A	D	00	
57410: CompletionStatus	: Successful	D	00	D	00	
57411: BCM	: 0x0	D	00	D	00	
57412: ByteCount	: 0x4	D	01	D	00	
57413: RequesterID	: 0 1c 0	D	02	D	00	
57414: Tag	: 0	D	00	D	00	
57415: LowerAddress	: 0x80	D	00	D	00	
57416:		D	04	D	00	
57417: Attr	: RO- NS- EP-	D	00	D	00	
57418: Length	: 1 DWs	D	E0	D	00	
57419: Data	: 0x63 0x00 0x0	D	00	D	00	
57420: Digest(ECRC)	: No ECRC	D	80	D	00	
57421: LCRC	: 0x4CCCE640	D	63	D	00	
57422:		D	00	D	00	
57423:		D	00	D	00	
57424:		D	00	D	00	
57425:		D	4C	D	00	
57426:		D	CC	D	00	
57427:		D	E6	D	00	
57428:		D	40	D	00	
57429:		K	END	D	00	
57430:		D	00	D	00	
...						
57501:		D	00	D	00	
57502:		D	00	K	SDP	Ack
57503:		D	00	D	00	
57504:		D	00	D	00	SeqNum: 1713
57505:		D	00	D	06	
57506:		D	00	D	B1	CRC16 : 0x8a66
57507:		D	00	D	8A	
57508:		D	00	D	66	
57509:		D	00	K	END	
57510:		D	00	D	00	
...						
57647:		D	00	D	00	
57648:		D	00	K	STP	MRd
57649:		D	00	D	08	SeqNr. : 2124
57650:		D	00	D	4C	
57651:		D	00	D	00	RequesterID: 0 1c 0
57652:		D	00	D	00	Tag : 0
57653:		D	00	D	00	Last DW BE : 0x0
57654:		D	00	D	01	First DW BE: 0xf
57655:		D	00	D	00	Address : 0xc2000004
57656:		D	00	D	E0	
57657:		D	00	D	00	Attr : RO- NS- EP-
57658:		D	00	D	0F	Length : 1 DWs
57659:		D	00	D	C2	Data : No data.
57660:		D	00	D	00	Digest(ECRC): No ECRC
57661:		D	00	D	00	LCRC : 0xB0887788
57662:		D	00	D	04	
57663:		D	00	D	B0	
57664:		D	00	D	88	
57665:		D	00	D	77	
57666:		D	00	D	88	
57667:		D	00	K	END	
57668:		D	00	D	00	
...						
57689:		D	00	D	00	
57690:	Ack	K	SDP	D	00	

Remote Lab Demonstration Tutorial
Measuring the throughput of CPU initiated PCIe transfers

57691:		D	00	D	00	
57692:	SeqNum: 2124	D	00	D	00	
57693:		D	08	D	00	
57694:	CRC16 : 0xee64	D	4C	D	00	
57695:		D	EE	D	00	
57696:		D	64	D	00	
57697:		K	END	D	00	
57698:	SKIP	K	COM	D	00	
57699:		K	SKP	D	00	
57700:		K	SKP	D	00	
57701:		K	SKP	D	00	
57702:		D	00	D	00	
...						
57749:		D	00	D	00	
57750:	UpdateFC_NP	K	SDP	D	00	
57751:		D	90	D	00	
57752:	VC : 0	D	3C	D	00	
57753:	HdrFC : 243	D	C1	D	00	
57754:	DataFC : 511	D	FF	D	00	
57755:		D	CC	D	00	
57756:	CRC16 : 0xcc43	D	43	D	00	
57757:		K	END	D	00	
57758:		D	00	D	00	
...						
57797:		D	00	D	00	
57798:	Cp1D	K	STP	D	00	
57799:	SeqNr. : 1714	D	06	D	00	
57800:		D	B2	D	00	
57801:	CompleterID : 2 0 0	D	4A	D	00	
57802:	CompletionStatus: Successful	D	00	D	00	
57803:	BCM : 0x0	D	00	D	00	
57804:	ByteCount : 0x4	D	01	D	00	
57805:	RequesterID : 0 1c 0	D	02	D	00	
57806:	Tag : 0	D	00	D	00	
57807:	LowerAddress : 0x84	D	00	D	00	
57808:		D	04	D	00	
57809:	Attr : RO- NS- EP-	D	00	D	00	
57810:	Length : 1 Dws	D	E0	D	00	
57811:	Data : 0x00 0x00 0x0	D	00	D	00	
57812:	Digest(ECRC): No ECRC	D	84	D	00	
57813:	LCRC : 0xE5696FCF	D	00	D	00	
57814:		D	00	D	00	
57815:		D	00	D	00	
57816:		D	00	D	00	
57817:		D	E5	D	00	
57818:		D	69	D	00	
57819:		D	6F	D	00	
57820:		D	CF	D	00	
57821:		K	END	D	00	
57822:		D	00	D	00	
...						
57893:		D	00	D	00	
57894:		D	00	K	SDP	Ack
57895:		D	00	D	00	
57896:		D	00	D	00	SeqNum: 1714
57897:		D	00	D	06	
57898:		D	00	D	B2	CRC16 : 0x694a
57899:		D	00	D	69	
57900:		D	00	D	4A	
57901:		D	00	K	END	

Remote Lab Demonstration Tutorial
Measuring the throughput of CPU initiated PCIe transfers

57902:		D 00	D 00		
...					
58030:		D 00	D 00		
58031:		D 00	K STP		MRd
58032:		D 00	D 08	SeqNr.	: 2125
58033:		D 00	D 4D		
58034:		D 00	D 00	RequesterID:	0 1c 0
58035:		D 00	D 00	Tag	: 0
58036:		D 00	D 00	Last DW BE	: 0x0
58037:		D 00	D 01	First DW BE	: 0xf
58038:		D 00	D 00	Address	: 0xc2000008
58039:		D 00	D E0		
58040:		D 00	D 00	Attr	: RO- NS- EP-
58041:		D 00	D 0F	Length	: 1 DWs
58042:		D 00	D C2	Data	: No data.
58043:		D 00	D 00	Digest(ECRC)	: No ECRC
58044:		D 00	D 00	LCRC	: 0x1E1D575C
58045:		D 00	D 08		
58046:		D 00	D 1E		
58047:		D 00	D 1D		
58048:		D 00	D 57		
58049:		D 00	D 5C		
58050:		D 00	K END		
58051:		D 00	D 00		
...					
58073:		D 00	D 00		
58074:	Ack	K SDP	D 00		
58075:		D 00	D 00		
58076:	SeqNum: 2125	D 00	D 00		
58077:		D 08	D 00		
58078:	CRC16 : 0x4f7f	D 4D	D 00		
58079:		D 4F	D 00		
58080:		D 7F	D 00		
58081:		K END	D 00		
58082:		D 00	D 00		
...					
58133:		D 00	D 00		
58134:	UpdateFC_NP	K SDP	D 00		
58135:		D 90	D 00		
58136:	VC : 0	D 3D	D 00		
58137:	HdrFC : 244	D 01	D 00		
58138:	DataFC : 511	D FF	D 00		
58139:		D 0C	D 00		
58140:	CRC16 : 0xc0e	D 0E	D 00		
58141:		K END	D 00		
...					
58181:		D 00	D 00		
58182:	CplD	K STP	D 00		
58183:	SeqNr. : 1715	D 06	D 00		
58184:		D B3	D 00		
58185:	CompleterID : 2 0 0	D 4A	D 00		
58186:	CompletionStatus: Successful	D 00	D 00		
58187:	BCM : 0x0	D 00	D 00		
58188:	ByteCount : 0x4	D 01	D 00		
58189:	RequesterID : 0 1c 0	D 02	D 00		
58190:	Tag : 0	D 00	D 00		
58191:	LowerAddress : 0x88	D 00	D 00		
58192:		D 04	D 00		
58193:	Attr : RO- NS- EP-	D 00	D 00		
58194:	Length : 1 DWs	D E0	D 00		

58195:	Data	:	0x00 0x00 0x0		D 00	D 00	
58196:	Digest(ECRC):	No ECRC			D 88	D 00	
58197:	LCRC	:	0xA74F398D		D 00	D 00	
58198:					D 00	D 00	
58199:					D 00	D 00	
58200:					D 00	D 00	
58201:					D A7	D 00	
58202:					D 4F	D 00	
58203:					D 39	D 00	
58204:					D 8D	D 00	
58205:					K END	D 00	
58206:					D 00	D 00	
...							
58277:					D 00	D 00	
58278:					D 00	K SDP	Ack
58279:					D 00	D 00	
58280:					D 00	D 00	SeqNum: 1715
58281:					D 00	D 06	
58282:					D 00	D B3	CRC16 : 0xc851
58283:					D 00	D C8	
58284:					D 00	D 51	
58285:					D 00	K END	
58286:					D 00	D 00	
...							

The record shows that the receiver incoming read to completion time for these three transmissions are 512ns, 520ns and 524ns. (The read to completion time is the time between the end of a memory read request (MRd) and the start of the corresponding completion (CpID)). During this time the DLLPs the ACK and the flow control update for the request TLP can be transmitted.

The trace shows a significant delay between a completion and the next request. This delay was not considered in the theoretical calculation. In this case the delay is 872ns and 836ns. During this time the ACK DLLP for the completion can be transmitted.

The delay between the first and the second memory read request was 1560ns and between the second and the third memory read request 1532ns. This will be a throughput of 2.56MByte/s and 2.61MByte/s.

4.8 Test 5: Alternating Write/Read Accesses

Test 5 produces n write accesses to the AHB RAM where every write access is followed by a read access. This test is intended to test if the correct values are read when fast write read combinations are done. It is also intended to test if there are any unsuspected influences between read and write accesses.

Test 5 uses following code to generate the AHB RAM accesses:

```

for (int i=0; i < n; i++) {
    RAM[i] = i;
    ret = RAM[i];
}

```

4.9 Test 6 and 7: Examine Worst Case Behavior

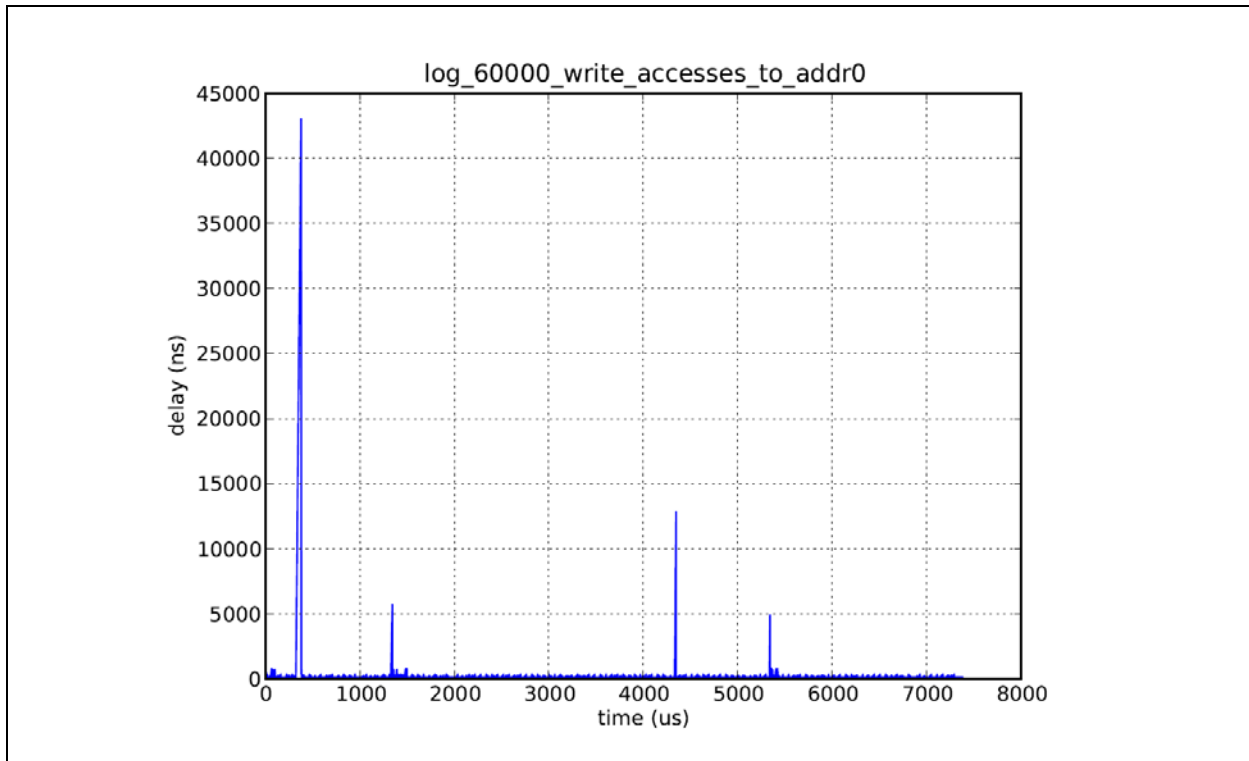
To examine the occasional peaks which show up in the results, longer traces are needed. Therefore a new design is used. This design consists of a very small RAM of only 16 bytes and a modified AHB-Trace IP core which records only the timestamp of every AHB access. With this design it is possible to

record the timestamps of 65536 AHB accesses. The .sof file of this design can be found in the remote lab environment. Open the AIM-Design and download the design **without** synthesizing it.

Test 6 and test 7 are modified versions of test 3 and test 4 and are specifically designed to work with the design mentioned above. The only difference is that all the n AHB RAM accesses address the first four bytes of the AHB RAM. When you have a look at the output of test 6 and test 7 it is important to keep in mind that only the timestamps and the delay times are correct. All addresses, data and flags shown in the trace have the values of the first recorded AHB access.

When you run test 6 you will get a plot like Figure 7.

Figure 7. Delays between write accesses (60 000 accesses)⁴



When you run test 7 you will get a plot like Figure 8.

⁴ Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. Configurations: Refer to Section 2.2 of [4] for complete configuration details. For more information go to <http://www.intel.com/performance>

Intel does not control or audit the design or implementation of third party benchmark data or Web sites referenced in this document. Intel encourages all of its customers to visit the referenced Web sites or others where similar performance benchmark data are reported and confirm whether the referenced benchmark data are accurate and reflect performance of systems available for purchase.

Figure 8. Delays between read accesses (60 000 accesses)⁵

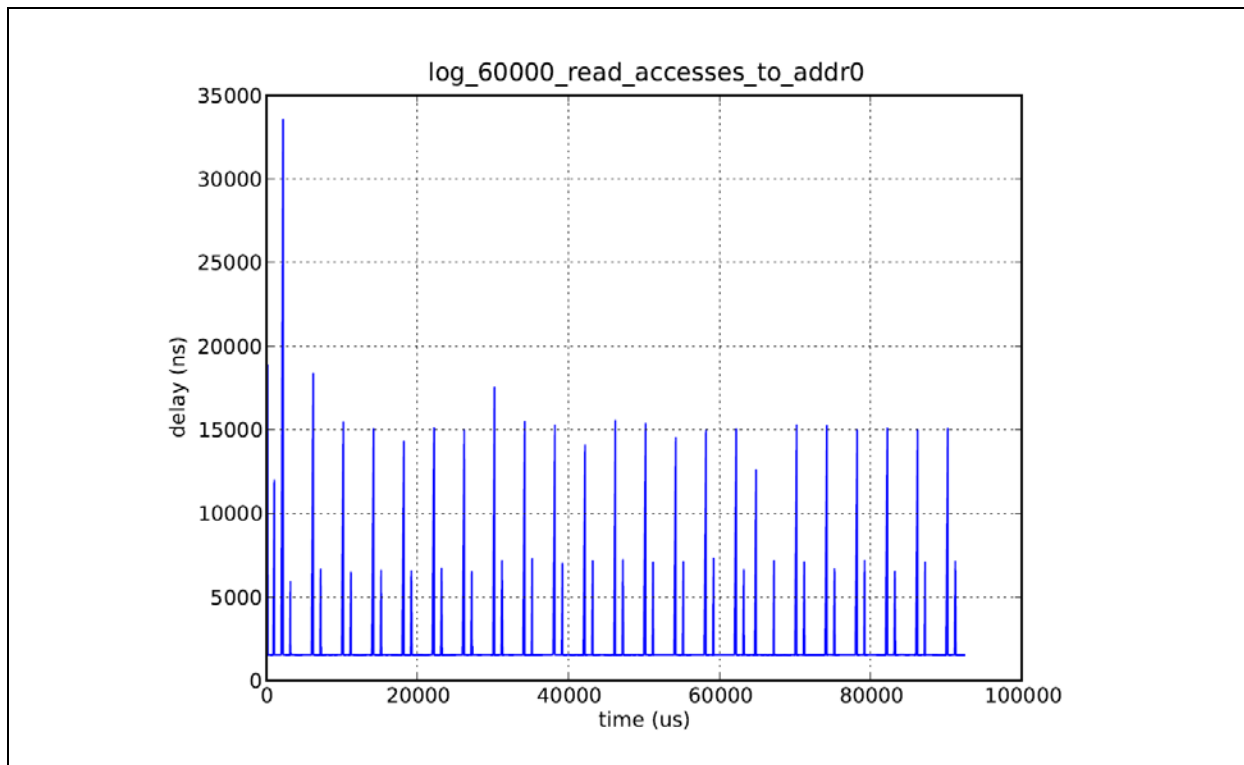


Figure 8 shows two types of longer delays occurring regularly. Both types of long delays occur every 4 μ s. Although the test program runs with the highest priority it is interruptible by interrupts. The long delays are caused by IO-APIC-edge timer interrupts, Local timer interrupt, and the Rescheduling interrupts.

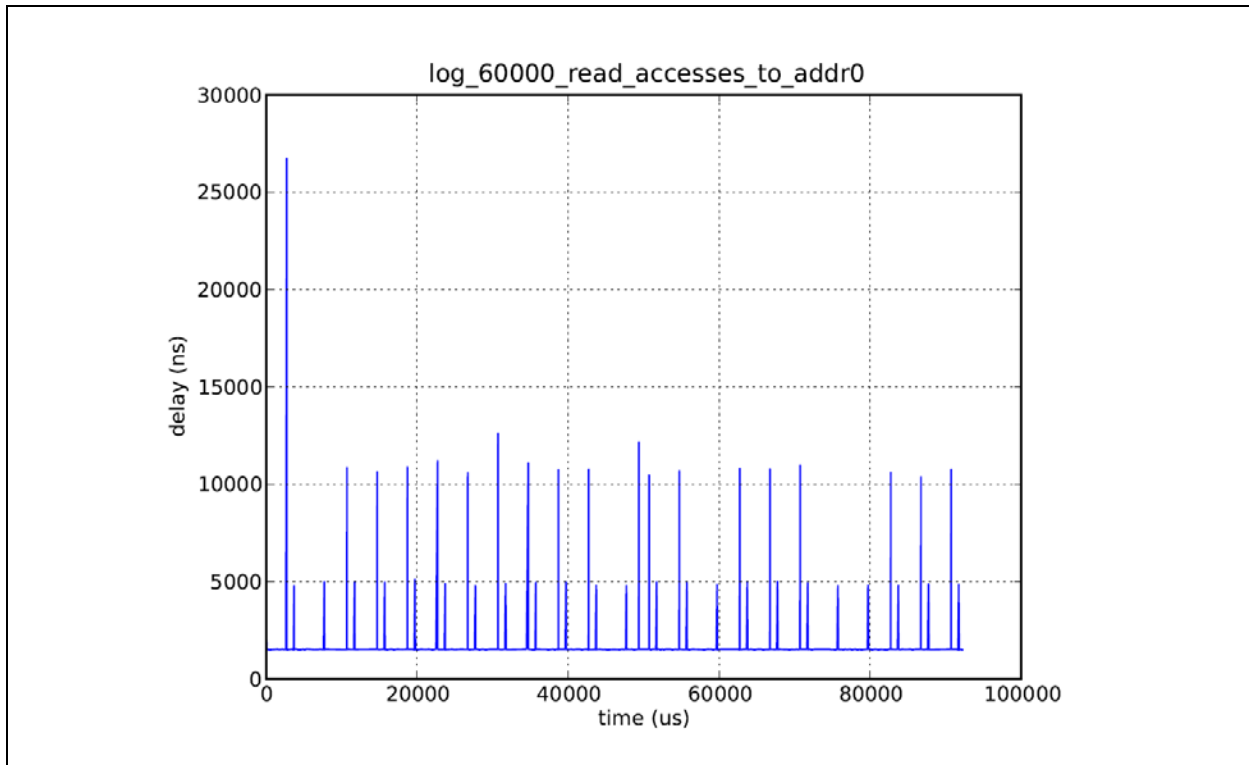
The processing time can be reduced by modifying the kernel configuration options. The optimal kernel configuration options depend on the requirements of the particular application.

To reduce the processing time of the timer interrupts it is recommended to turn off kernel debug/trace features. Figure 9 shows the delays of 60 000 read accesses with disabled kernel trace features.

⁵ Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. Configurations: Refer to Section 2.2 of [4] for complete configuration details. For more information go to <http://www.intel.com/performance>

Intel does not control or audit the design or implementation of third party benchmark data or Web sites referenced in this document. Intel encourages all of its customers to visit the referenced Web sites or others where similar performance benchmark data are reported and confirm whether the referenced benchmark data are accurate and reflect performance of systems available for purchase.

Figure 9. Delays between read accesses with disabled kernel debug/trace features⁶



To reduce the progressing time of the timer interrupts even further the 'tickless kernel feature' and the 'high resolution timer' were disabled. The result is shown in Figure 10.

⁶ Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. Configurations: Refer to Section 2.2 of [4] for complete configuration details. For more information go to <http://www.intel.com/performance>

Intel does not control or audit the design or implementation of third party benchmark data or Web sites referenced in this document. Intel encourages all of its customers to visit the referenced Web sites or others where similar performance benchmark data are reported and confirm whether the referenced benchmark data are accurate and reflect performance of systems available for purchase.

Figure 10. Delays between read accesses, tickless kernel, disabled debug/trace/HRT⁷

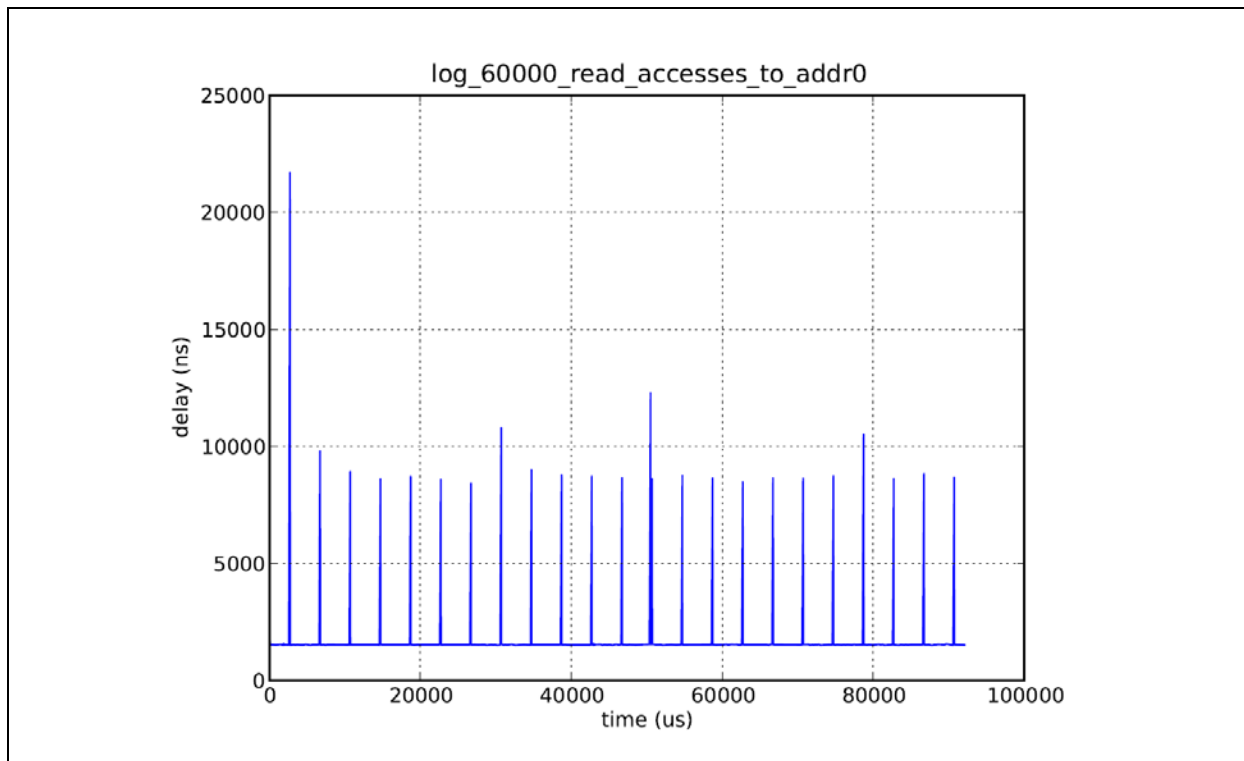


Figure 10 shows that disabling the 'tickless kernel' and the 'high resolution timer' reduce the amount and duration of interrupts during the data transfer.

4.10 Test 8: Multiple Writes, CPU Time

Test 8 measures the CPU time which is needed to execute n times 64 memory write accesses to the AHB RAM. The value of n can be chosen at the beginning of the test. The number of accesses can only be increased by 64 to reduce the overhead of the for-loop. In every loop iteration 64 accesses are executed. Every access writes to the first address of the AHB RAM to avoid limitation of the AHB RAM size.

The FIFO size of the Trace IP core does not limit the amount of accesses for this test. When you use the design explained in section 4.2 the first 8192 can be traced, but the time measurement also works with more accesses.

⁷ Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. Configurations: Refer to Section 2.2 of [4] for complete configuration details. For more information go to <http://www.intel.com/performance>

Intel does not control or audit the design or implementation of third party benchmark data or Web sites referenced in this document. Intel encourages all of its customers to visit the referenced Web sites or others where similar performance benchmark data are reported and confirm whether the referenced benchmark data are accurate and reflect performance of systems available for purchase.

The following code is used for this test:

```
const int intvalue_c = 10;
int i = 0;

cntOld = readTSC();
for(; i < n; i++) {
    RAM[0] = intvalue_c;
    ...
    RAM[0] = intvalue_c;
}
cnt = readTSC();
```

For time measuring the Time Stamp Counter (TSC) Register is used. TSC register runs at full processor speed and counts all clock cycles. This promises the highest accuracy for time measurement. For meeting highest accuracy for time measurement and real time behavior, CPU frequency scaling is disabled. Before running this test, it is recommended to run test 16 to calibrate the time measurement functions.

Test 8 shows similar results like test 3 and test 6. The only difference is that the measurement is done in the CPU and not in the FPGA. The fact that the results are consistent indicates the validity of the measurement.

4.11 Test 9: Multiple Reads, CPU Time

Test 9 measures the time which is needed to execute n times 64 memory read accesses to the AHB RAM. The value of n can be chosen at the beginning of the test. The number of accesses can only be increased by 64 to reduce the overhead of the for-loop. In every loop iteration 64 accesses are executed. Every access writes to the first address of the AHB RAM to avoid limitation of the AHB RAM size.

The FIFO size of the Trace IP core does not limit the amount of accesses for this test. When you use the design explained in Section 4.2 the first 8192 can be traced, but the time measurement also works with more accesses.

The following code is used for this test:

```
u32 ret;
int i = 0;

cntOld = readTSC();
for(; i < n; i++) {
    ret = RAM[0];

    ret = RAM[0];
}
cnt = readTSC();
```

Before running this test, it is recommended to run test 16 to calibrate the time measurement functions.

Test 9 shows similar results like test 4 and test 7. The only difference is that the measurement is done in CPU and not in the FPGA. The fact that the results are consistent indicates the validity of the measurement.

4.12 Test 10: Multiple Writes, 64 Bits

Test 10 generates n memory writes with a data payload size of 8 bytes. So every TLP causes 2 write accesses to the AHB RAM. So test 10 will end up with a higher throughput than the previous tests. The throughput is about twice as high as the throughput in test 3 where TLPs with a data payload size of 4 bytes are used. The value of n can be chosen at the beginning of test 10.

Note that the number of the possible TLPs is limited by the used AHB RAM size.

To generate TLPs with a data payload size of 8 bytes the memory-mapped integer array used in the previous tests is cast to a double array. In test 10 the following code is used to generate the TLPs:

```
const double doublevalue_c = 1.2;
volatile double *p64;
p64 = (volatile double *) RAM;
for (int i=0; i < n; i++) {
    p64[i] = doublevalue_c;
}
```

When you run test 12 with 10 iterations you will get an output like the following:

```
timestamp: 1301230 [ 0.00 ns] addr: 0x02000000, data: 0858993459, flags: 0x0105 WRITE
timestamp: 1301232 [ 20.00 ns] addr: 0x02000004, data: 1072902963, flags: 0x0105 WRITE
timestamp: 1301243 [ 110.00 ns] addr: 0x02000008, data: 0858993459, flags: 0x0105 WRITE
timestamp: 1301245 [ 20.00 ns] addr: 0x0200000c, data: 1072902963, flags: 0x0105 WRITE
timestamp: 1301256 [ 110.00 ns] addr: 0x02000010, data: 0858993459, flags: 0x0105 WRITE
timestamp: 1301258 [ 20.00 ns] addr: 0x02000014, data: 1072902963, flags: 0x0105 WRITE
timestamp: 1301269 [ 110.00 ns] addr: 0x02000018, data: 0858993459, flags: 0x0105 WRITE
timestamp: 1301271 [ 20.00 ns] addr: 0x0200001c, data: 1072902963, flags: 0x0105 WRITE
timestamp: 1301281 [ 100.00 ns] addr: 0x02000020, data: 0858993459, flags: 0x0105 WRITE
timestamp: 1301283 [ 20.00 ns] addr: 0x02000024, data: 1072902963, flags: 0x0105 WRITE
timestamp: 1301294 [ 110.00 ns] addr: 0x02000028, data: 0858993459, flags: 0x0105 WRITE
timestamp: 1301296 [ 20.00 ns] addr: 0x0200002c, data: 1072902963, flags: 0x0105 WRITE
timestamp: 1301307 [ 110.00 ns] addr: 0x02000030, data: 0858993459, flags: 0x0105 WRITE
timestamp: 1301309 [ 20.00 ns] addr: 0x02000034, data: 1072902963, flags: 0x0105 WRITE
timestamp: 1301320 [ 110.00 ns] addr: 0x02000038, data: 0858993459, flags: 0x0105 WRITE
timestamp: 1301322 [ 20.00 ns] addr: 0x0200003c, data: 1072902963, flags: 0x0105 WRITE
timestamp: 1301333 [ 110.00 ns] addr: 0x02000040, data: 0858993459, flags: 0x0105 WRITE
timestamp: 1301335 [ 20.00 ns] addr: 0x02000044, data: 1072902963, flags: 0x0105 WRITE
timestamp: 1301346 [ 110.00 ns] addr: 0x02000048, data: 0858993459, flags: 0x0105 WRITE
timestamp: 1301348 [ 20.00 ns] addr: 0x0200004c, data: 1072902963, flags: 0x0105 WRITE
```

Statistic for 20 AHB accesses:

Every PCIe TLP Packet ends up in 2 AHB accesses
Number of Interrupts during the transmission: 0

```
average elapsed Time: 128.89 ns
min elapsed Time: 120.00 ns
max elapsed Time: 130.00 ns
```

```
average throughput: 62068965.52 Bytes/s
max throughput: 66666666.67 Bytes/s
min throughput: 61538461.54 Bytes/s
```

This output shows the trace of the AHB bus. The time between two TLPs can be estimated by the sum of two delays of AHB RAM accesses. So the TLPs with a payload size of 8 bytes arrive in about the same time interval as TLPs with a payload size of 4 bytes.

The second AHB RAM access always follows 20ns after the first AHB RAM access caused by same TLP. This is because of the 100MHz of the AHB. In the first 10ns the AHB address is written to the AHB and in the next 10ns the data is written to the AHB.

Overall, test 10 shows a doubling of the throughput compared to test 3.

4.13 Test 11: Multiple Reads, 64 Bits

Test 11 generates n memory reads with a data payload size of 8 bytes. So every TLP causes 2 reads from the AHB RAM. So test 11 will end up with a higher read throughput than the previous tests. The throughput is about twice as high as the throughput in test 4 where TLPs with a data payload size of 4 bytes are used. The value of n can be chosen at the beginning of test 11.

Note that the number of the possible TLPs is limited by the used AHB RAM size.

The following code is used to generate memory read TLPs with a data payload size of 8 bytes:

```
volatile double * p64;
double readvalue;
p64 = (volatile double *) RAM;
for (int i=0; i < n; i++) {
    readvalue = p64[i];
}
```

When you run test 13 with 10 iterations you will get an output like the following.

```
timestamp:    32434 [    0.00 ns] addr: 0x02000000, data: 0858993459, flags: 0x0104 READ
timestamp:    32436 [   20.00 ns] addr: 0x02000004, data: 1072902963, flags: 0x0104 READ
timestamp:    32599 [  1630.00 ns] addr: 0x02000008, data: 0858993459, flags: 0x0104 READ
timestamp:    32601 [   20.00 ns] addr: 0x0200000c, data: 1072902963, flags: 0x0104 READ
timestamp:    32759 [  1580.00 ns] addr: 0x02000010, data: 0858993459, flags: 0x0104 READ
timestamp:    32761 [   20.00 ns] addr: 0x02000014, data: 1072902963, flags: 0x0104 READ
timestamp:    32918 [  1570.00 ns] addr: 0x02000018, data: 0858993459, flags: 0x0104 READ
timestamp:    32920 [   20.00 ns] addr: 0x0200001c, data: 1072902963, flags: 0x0104 READ
timestamp:    33076 [  1560.00 ns] addr: 0x02000020, data: 0858993459, flags: 0x0104 READ
timestamp:    33078 [   20.00 ns] addr: 0x02000024, data: 1072902963, flags: 0x0104 READ
timestamp:    33233 [  1550.00 ns] addr: 0x02000028, data: 0858993459, flags: 0x0104 READ
timestamp:    33235 [   20.00 ns] addr: 0x0200002c, data: 1072902963, flags: 0x0104 READ
timestamp:    33392 [  1570.00 ns] addr: 0x02000030, data: 0858993459, flags: 0x0104 READ
timestamp:    33394 [   20.00 ns] addr: 0x02000034, data: 1072902963, flags: 0x0104 READ
timestamp:    33552 [  1580.00 ns] addr: 0x02000038, data: 0858993459, flags: 0x0104 READ
timestamp:    33554 [   20.00 ns] addr: 0x0200003c, data: 1072902963, flags: 0x0104 READ
timestamp:    33711 [  1570.00 ns] addr: 0x02000040, data: 0858993459, flags: 0x0104 READ
timestamp:    33713 [   20.00 ns] addr: 0x02000044, data: 1072902963, flags: 0x0104 READ
timestamp:    33869 [  1560.00 ns] addr: 0x02000048, data: 0858993459, flags: 0x0104 READ
timestamp:    33871 [   20.00 ns] addr: 0x0200004c, data: 1072902963, flags: 0x0104 READ
```

Statistic for 20 AHB accesses:

Every PCIe TPL Packet ends up in 2 AHB accesses
Number of Interrupts during the transmission: 0

```
average elapsed Time:    1594.44 ns
min elapsed Time:       1570.00 ns
max elapsed Time:       1650.00 ns
```

```
average throughput:     5017421.60 Bytes/s
max throughput:         5095541.40 Bytes/s
min throughput:         4848484.85 Bytes/s
```

This output shows that memory read TLPs with a payload size of 8 bytes arrive in about the same time intervals as memory read TLPs with a payload size of 4 bytes. So test 11 shows a doubling of the throughput compared to test 4.

4.14 Test 12: Multiple Writes, 64 Bits, CPU Time

Test 12 measures the CPU time which is needed to execute n times 64 memory write TLPs with a payload size of 8 Byte. Every TLP will end up in two AHB accesses to the AHB RAM. The value of n can be chosen at the beginning of the test. The number of accesses can only be increased by 64 to reduce the overhead of the for-loop. In every loop iteration 64 accesses are executed. Every access writes to the first address of the AHB RAM to avoid limitations of the AHB RAM size.

The FIFO size of the Trace IP core does not limit the amount of accesses for this test. When you use the design explained in Section 4.2 the first 8192 AHB accesses can be traced, but the time measurement also works with more accesses.

The following code is used for this test:

```
volatile double * p64;
p64 = (volatile double *) RAM;
const double doublevalue_c = 0.0;
int i=0;

cntOld = readTSC();
for (; i < n; i++) {
    p64[i] = doublevalue_c;
    ...
    p64[i] = doublevalue_c;
}
cnt = readTSC();
```

Before running this test, it is recommended to run test 16 to calibrate the time measurement functions.

Test 12 shows similar results like test 10 the only difference is that the measurement is done in CPU and not in the FPGA.

4.15 Test 13: Multiple Reads, 64 Bits, CPU Time

Test 13 measures the CPU time which is needed to execute n times 64 memory read TLPs with a payload size of 8 bytes. Every TLP will end up in two AHB accesses to the AHB RAM. The value of n can be chosen at the beginning of the test. The number of accesses can only be increased by 64 to reduce the overhead of the for-loop. In every loop iteration 64 accesses are executed. Every access writes to the first address of the AHB RAM to avoid limitations of the AHB RAM size.

The FIFO size of the Trace IP core does not limit the amount of accesses for this test. When you use the design explained in Section 4.2 the first 8192 AHB accesses can be traced, but the time measurement also works with more accesses.

Therefore the following code is used.

```
volatile double * p64;
p64 = (volatile double *) RAM;
double ret;
int i=0;

cntOld = readTSC();
for (; i < n; i++) {
    ret = p64[i];
    ...
}
```

```

ret = p64[i];
}
cnt = readTSC();

```

Before running this test, it is recommended to run test 16 to calibrate the time measurement functions.

Test 13 shows similar results like test 11 the only difference is that the measurement is done in CPU and not in the FPGA.

4.16 Test 14: Write Combining

Test 14 produces sequential write accesses of varying lengths to the AHB-RAM. The test starts with a single byte-write to the start of the AHB-RAM, then with two bytes starting at the AHB-RAM, then four bytes and so on. Before the test starts, the CPU's write combining feature is enabled. The write combining mechanism collects (i.e. buffers) multiple sequential memory accesses and combines them into a single, larger, memory access. In conjunction with the sequential write access pattern this will cause PCIe write transactions of varying payload size.

To generate multiple accesses of a size N, this test generates N sequential byte-wise write accesses followed by an `sfence` instruction. This will cause a (partially) filled write combining buffer to be evicted. For details refer to [7], p. 11-9.

The write combining feature can be activated via the memory type range registers (MTRR). Under Linux a user-space application can set the MTRRs by writing to `/proc/mtrr`. Root permissions are required to do this. Hence this test will not work without root permissions. For more details refer to the source code of the demo application (`mtrr.c`) as well as the documentation of the Linux kernel (`/usr/src/linux/Documentation/x86/mtrr.txt`).

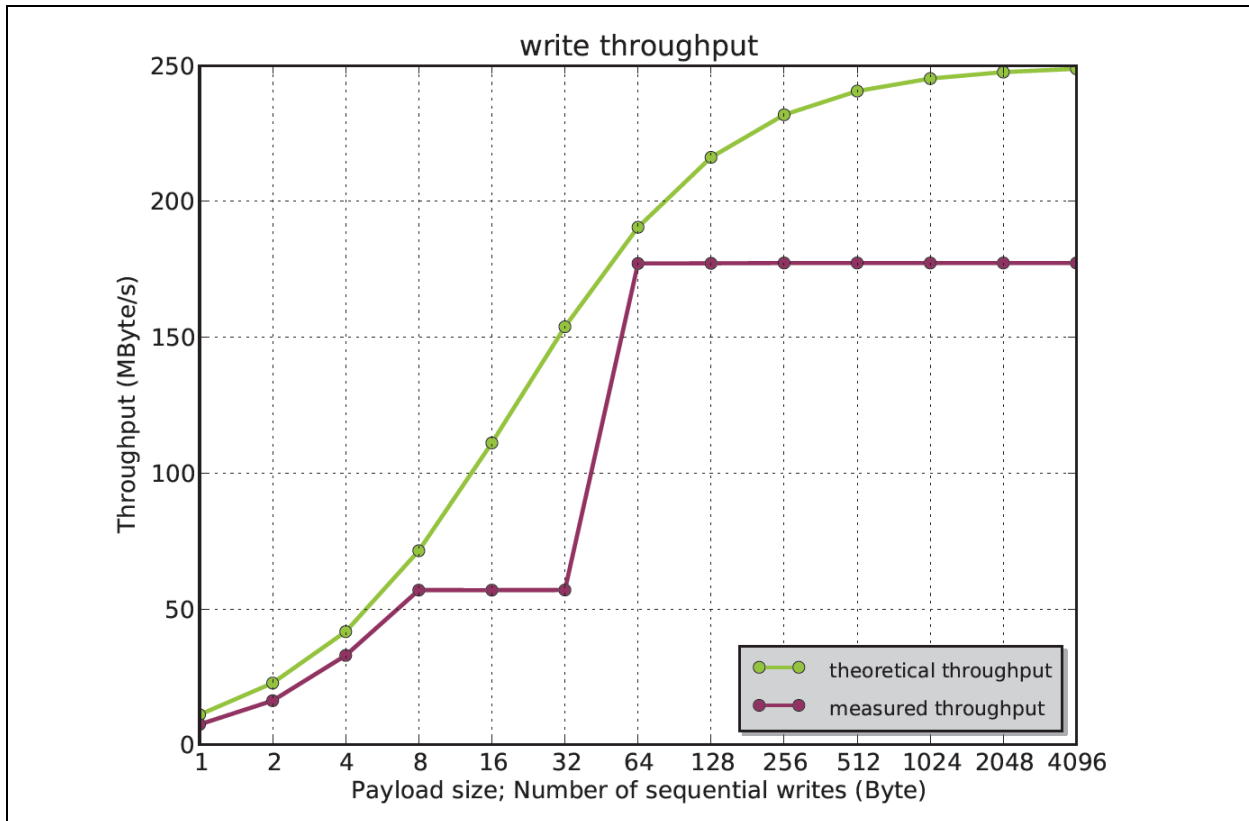
The design explained in Section 4.2 will produce an output similar to this one:

Throughput overview:		
Write accesses:		
Sequential writes	Max PCIe throughput	avg. measured throughput
1 Byte	11.9048 MByte/s	7.50511 MByte/s
2 Byte	22.7273 MByte/s	16.2375 MByte/s
4 Byte	41.6667 MByte/s	32.9131 MByte/s
8 Byte	71.4286 MByte/s	56.9373 MByte/s
16 Byte	111.111 MByte/s	56.9283 MByte/s
32 Byte	153.846 MByte/s	56.9748 MByte/s
64 Byte	190.476 MByte/s	177.142 MByte/s
128 Byte	216.216 MByte/s	177.208 MByte/s
256 Byte	231.884 MByte/s	177.332 MByte/s
512 Byte	240.602 MByte/s	177.338 MByte/s
1024 Byte	245.211 MByte/s	177.331 MByte/s
2048 Byte	247.582 MByte/s	177.336 MByte/s

4096 Byte	248.785 MByte/s	177.338 MByte/s
-----------	-----------------	-----------------

When the log file generation is activated a plot of this result can be generated. This plot will look like the following:

Figure 11. Throughput result of test 14 (AHB-RAM size = 32k)⁸



As can be seen in Figure 11, for 1 to 8 bytes of sequential writes the measured throughput follows the theoretical PCIe throughput of the corresponding payload size. For 8, 16 and 32 bytes the throughput remains constant which indicates that in these cases PCIe write transactions with 8 bytes are issued. For 64 bytes the measured throughput again closely matches the theoretical maximum which indicates that PCIe write transactions with 64 bytes are issued. This behavior is caused by the way the write combining works. The write combining buffers are of 64 bytes size. Writes are collected in these

⁸ Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. Configurations: Refer to Section 2.2 of [4] for complete configuration details. For more information go to <http://www.intel.com/performance>

Intel does not control or audit the design or implementation of third party benchmark data or Web sites referenced in this document. Intel encourages all of its customers to visit the referenced Web sites or others where similar performance benchmark data are reported and confirm whether the referenced benchmark data are accurate and reflect performance of systems available for purchase.

buffers. A full buffer is written out all at once, whereas partial buffers are written out in chunks of 8 bytes. Details can be found in [7], Section 11.3.1 Buffering of Write Combining Memory Locations.

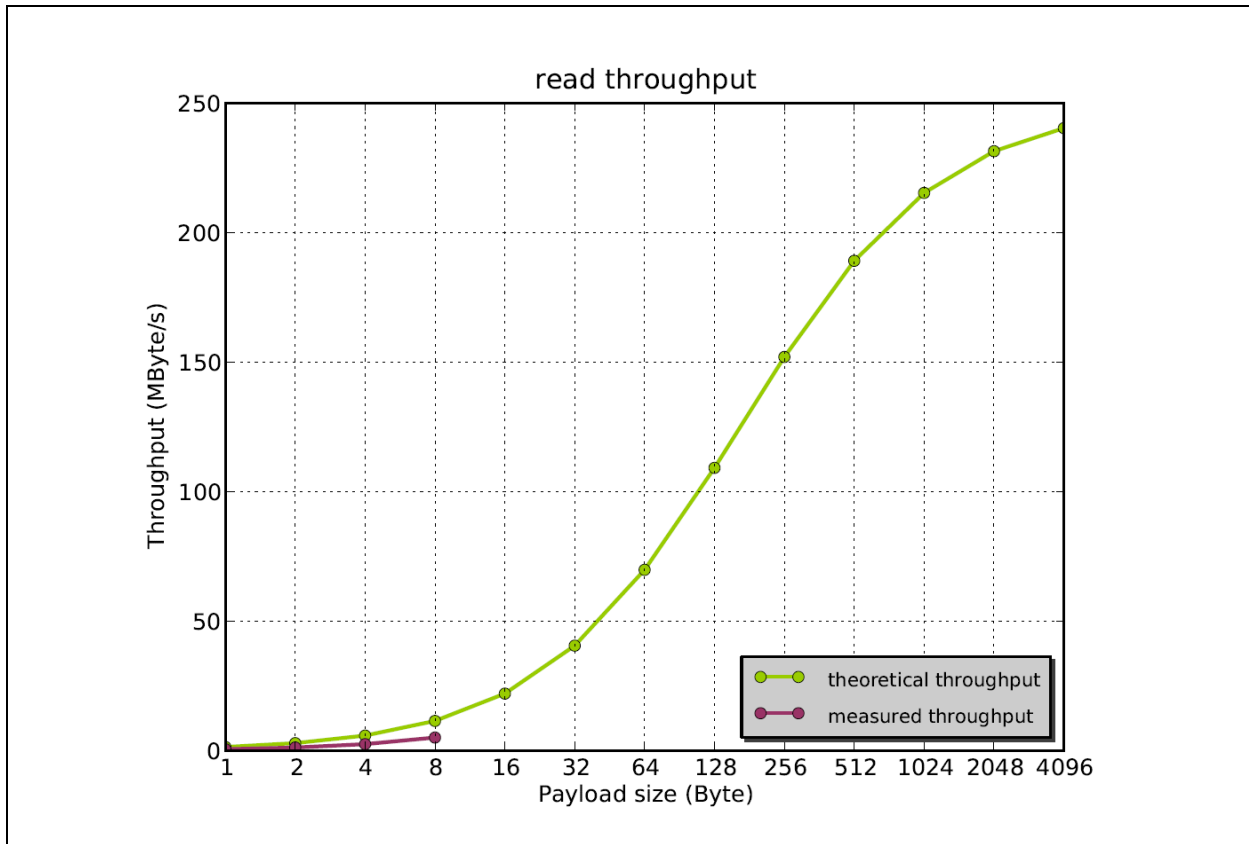
4.17 Test 15: Compare 1, 2, 4, 8Byte Payload size, read

Test 15 produces n read accesses to the AHB-RAM which are initiated from TLPs with a payload size of 1, 2, 4 and 8 bytes. At the end a statistic over the different throughputs will be presented. Every read access will address the first bytes in the AHB-RAM, so the AHB-RAM has to have a size of at least 8 bytes. It is recommended that the value of n does not exceed the value of traceable AHB accesses. With the design explained in Section 4.2 the maximum for n will be 8192, with the design explained in Section 4.9 the maximum for n will be 65384.

Throughput overview:		
Read accesses:		
Payload size:	theo. troughput	avg measured throughput
1 Byte	1.51 MByte/s	0.65 MByte/s
2 Byte	2.99 MByte/s	1.30 MByte/s
4 Byte	5.98 MByte/s	2.61 MByte/s
8 Byte	11.56 MByte/s	5.15 MByte/s

When the log file generation is activated a plot of this result can be generated. This plot will look like the following:

Figure 12. Throughput result of test 15 with n=60000⁹



As opposed to the write-case, there is no such thing like "read combining". However, there is a possibility to generate requests for more than 8 bytes: prefetching. For memory regions marked as cachable the CPU may read data in advance, i.e. before it is actually needed. When data is fetched into the cache, whole cache lines are read. This operation is called cache line fill. A cache line is 64 bytes on the platform used in this tutorial.

Note that caching by default is disabled for all I/O regions which are mapped into memory. The following trace shows what happens if caching is enabled. The trace results from a single byte-read from the first address of the AHB-RAM. Note how chunks of 64 bytes are read each with a gap of approx. 1650 ns. Also note that the addresses are not strictly increasing: 0x020000c0 is read before 0x02000080.

⁹ Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. Configurations: Refer to Section 2.2 of [4] for complete configuration details. For more information go to <http://www.intel.com/performance>

Intel does not control or audit the design or implementation of third party benchmark data or Web sites referenced in this document. Intel encourages all of its customers to visit the referenced Web sites or others where similar performance benchmark data are reported and confirm whether the referenced benchmark data are accurate and reflect performance of systems available for purchase.

Remote Lab Demonstration Tutorial
Measuring the throughput of CPU initiated PCIe transfers

In this trace 320 bytes are transferred in 8030ns which results in a throughput of roughly 40 MByte/s.

timestamp:	6617486	[0 ns]	addr:	0x02000000,	data:	0303174162
timestamp:	6617488	[20 ns]	addr:	0x02000004,	data:	0303174162
timestamp:	6617490	[20 ns]	addr:	0x02000008,	data:	0303174162
timestamp:	6617492	[20 ns]	addr:	0x0200000c,	data:	0303174162
timestamp:	6617494	[20 ns]	addr:	0x02000010,	data:	0303174162
timestamp:	6617496	[20 ns]	addr:	0x02000014,	data:	0303174162
timestamp:	6617498	[20 ns]	addr:	0x02000018,	data:	0303174162
timestamp:	6617500	[20 ns]	addr:	0x0200001c,	data:	0303174162
timestamp:	6617502	[20 ns]	addr:	0x02000020,	data:	0303174162
timestamp:	6617504	[20 ns]	addr:	0x02000024,	data:	0303174162
timestamp:	6617506	[20 ns]	addr:	0x02000028,	data:	0303174162
timestamp:	6617508	[20 ns]	addr:	0x0200002c,	data:	0303174162
timestamp:	6617510	[20 ns]	addr:	0x02000030,	data:	0303174162
timestamp:	6617512	[20 ns]	addr:	0x02000034,	data:	0303174162
timestamp:	6617514	[20 ns]	addr:	0x02000038,	data:	0303174162
timestamp:	6617516	[20 ns]	addr:	0x0200003c,	data:	0303174162
timestamp:	6617680	[1640 ns]	addr:	0x02000040,	data:	0303174162
timestamp:	6617682	[20 ns]	addr:	0x02000044,	data:	0303174162
timestamp:	6617684	[20 ns]	addr:	0x02000048,	data:	0303174162
timestamp:	6617686	[20 ns]	addr:	0x0200004c,	data:	0303174162
timestamp:	6617688	[20 ns]	addr:	0x02000050,	data:	0303174162
timestamp:	6617690	[20 ns]	addr:	0x02000054,	data:	0303174162
timestamp:	6617692	[20 ns]	addr:	0x02000058,	data:	0303174162
timestamp:	6617694	[20 ns]	addr:	0x0200005c,	data:	0303174162
timestamp:	6617696	[20 ns]	addr:	0x02000060,	data:	0303174162
timestamp:	6617698	[20 ns]	addr:	0x02000064,	data:	0303174162
timestamp:	6617700	[20 ns]	addr:	0x02000068,	data:	0303174162
timestamp:	6617702	[20 ns]	addr:	0x0200006c,	data:	0303174162
timestamp:	6617704	[20 ns]	addr:	0x02000070,	data:	0303174162
timestamp:	6617706	[20 ns]	addr:	0x02000074,	data:	0303174162
timestamp:	6617708	[20 ns]	addr:	0x02000078,	data:	0303174162
timestamp:	6617710	[20 ns]	addr:	0x0200007c,	data:	0303174162
timestamp:	6617875	[1650 ns]	addr:	0x020000c0,	data:	0303174162
timestamp:	6617877	[20 ns]	addr:	0x020000c4,	data:	0303174162
timestamp:	6617879	[20 ns]	addr:	0x020000c8,	data:	0303174162
timestamp:	6617881	[20 ns]	addr:	0x020000cc,	data:	0303174162
timestamp:	6617883	[20 ns]	addr:	0x020000d0,	data:	0303174162
timestamp:	6617885	[20 ns]	addr:	0x020000d4,	data:	0303174162
timestamp:	6617887	[20 ns]	addr:	0x020000d8,	data:	0303174162
timestamp:	6617889	[20 ns]	addr:	0x020000dc,	data:	0303174162
timestamp:	6617891	[20 ns]	addr:	0x020000e0,	data:	0303174162
timestamp:	6617893	[20 ns]	addr:	0x020000e4,	data:	0303174162
timestamp:	6617895	[20 ns]	addr:	0x020000e8,	data:	0303174162
timestamp:	6617897	[20 ns]	addr:	0x020000ec,	data:	0303174162
timestamp:	6617899	[20 ns]	addr:	0x020000f0,	data:	0303174162
timestamp:	6617901	[20 ns]	addr:	0x020000f4,	data:	0303174162
timestamp:	6617903	[20 ns]	addr:	0x020000f8,	data:	0303174162
timestamp:	6617905	[20 ns]	addr:	0x020000fc,	data:	0303174162
timestamp:	6618067	[1620 ns]	addr:	0x02000080,	data:	0303174162
timestamp:	6618069	[20 ns]	addr:	0x02000084,	data:	0303174162
timestamp:	6618071	[20 ns]	addr:	0x02000088,	data:	0303174162
timestamp:	6618073	[20 ns]	addr:	0x0200008c,	data:	0303174162
timestamp:	6618075	[20 ns]	addr:	0x02000090,	data:	0303174162
timestamp:	6618077	[20 ns]	addr:	0x02000094,	data:	0303174162
timestamp:	6618079	[20 ns]	addr:	0x02000098,	data:	0303174162
timestamp:	6618081	[20 ns]	addr:	0x0200009c,	data:	0303174162
timestamp:	6618083	[20 ns]	addr:	0x020000a0,	data:	0303174162
timestamp:	6618085	[20 ns]	addr:	0x020000a4,	data:	0303174162
timestamp:	6618087	[20 ns]	addr:	0x020000a8,	data:	0303174162

timestamp:	6618089	[20 ns]	addr:	0x020000ac,	data:	0303174162
timestamp:	6618091	[20 ns]	addr:	0x020000b0,	data:	0303174162
timestamp:	6618093	[20 ns]	addr:	0x020000b4,	data:	0303174162
timestamp:	6618095	[20 ns]	addr:	0x020000b8,	data:	0303174162
timestamp:	6618097	[20 ns]	addr:	0x020000bc,	data:	0303174162
timestamp:	6618259	[1620 ns]	addr:	0x02000180,	data:	0303174162
timestamp:	6618261	[20 ns]	addr:	0x02000184,	data:	0303174162
timestamp:	6618263	[20 ns]	addr:	0x02000188,	data:	0303174162
timestamp:	6618265	[20 ns]	addr:	0x0200018c,	data:	0303174162
timestamp:	6618267	[20 ns]	addr:	0x02000190,	data:	0303174162
timestamp:	6618269	[20 ns]	addr:	0x02000194,	data:	0303174162
timestamp:	6618271	[20 ns]	addr:	0x02000198,	data:	0303174162
timestamp:	6618273	[20 ns]	addr:	0x0200019c,	data:	0303174162
timestamp:	6618275	[20 ns]	addr:	0x020001a0,	data:	0303174162
timestamp:	6618277	[20 ns]	addr:	0x020001a4,	data:	0303174162
timestamp:	6618279	[20 ns]	addr:	0x020001a8,	data:	0303174162
timestamp:	6618281	[20 ns]	addr:	0x020001ac,	data:	0303174162
timestamp:	6618283	[20 ns]	addr:	0x020001b0,	data:	0303174162
timestamp:	6618285	[20 ns]	addr:	0x020001b4,	data:	0303174162
timestamp:	6618287	[20 ns]	addr:	0x020001b8,	data:	0303174162
timestamp:	6618289	[20 ns]	addr:	0x020001bc,	data:	0303174162

4.18 Test 16: Timer Calibration

Test 16 is used to calibrate the time measurement functions and to get an idea of the accuracy of the time measurement functions. The time measurement functions are used to measure CPU time in test 8, test 9, test 12 and test 13.

For CPU time measuring in test 8, test 9 test 12 and test 13 the Time Stamp Counter (TSC) Register is used. TSC register runs at full processor speed and counts all clock cycles. This promises the highest accuracy for time measurement. For meeting highest accuracy for time measurement and real time behavior CPU frequency scaling is disabled.

Test 16 tries to read out the Time Stamp Counter (TSC) register as fast as possible. To calibrate the time measurement functions, it is useful to run test 16 with about 100 000 iterations. At the end of the test the maximum, the minimum and the average measured time is reported and a calibration value can be set. This calibration value will be subtracted from the measured time in test 8, test 9, test 12 and test 13. It is recommended to use the minimum measured time as calibration value. The minimum measured time should be about 150ns.

5 Measuring the throughput of DMA transfers

In this chapter we will measure the performance of data transfers which are initiated by the FPGA. More specifically we deal with large transfers initiated by a DMA controller in an FPGA design. The data is transferred between the CPU's RAM and the FPGA. Recall the architecture of the Hpe_IRP as shown in Figure 13. The FPGA design we use is the one shown in Section 2 and in Figure 14.

Figure 13. Basic architecture of the Hpe_IRP

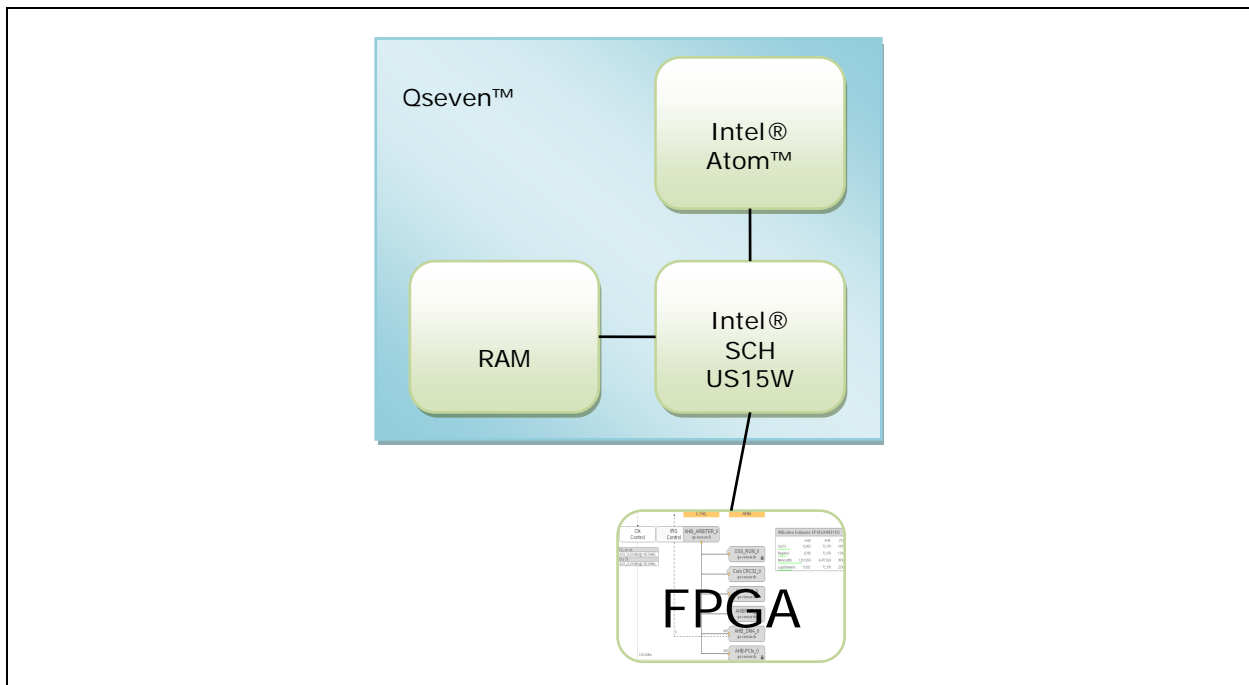
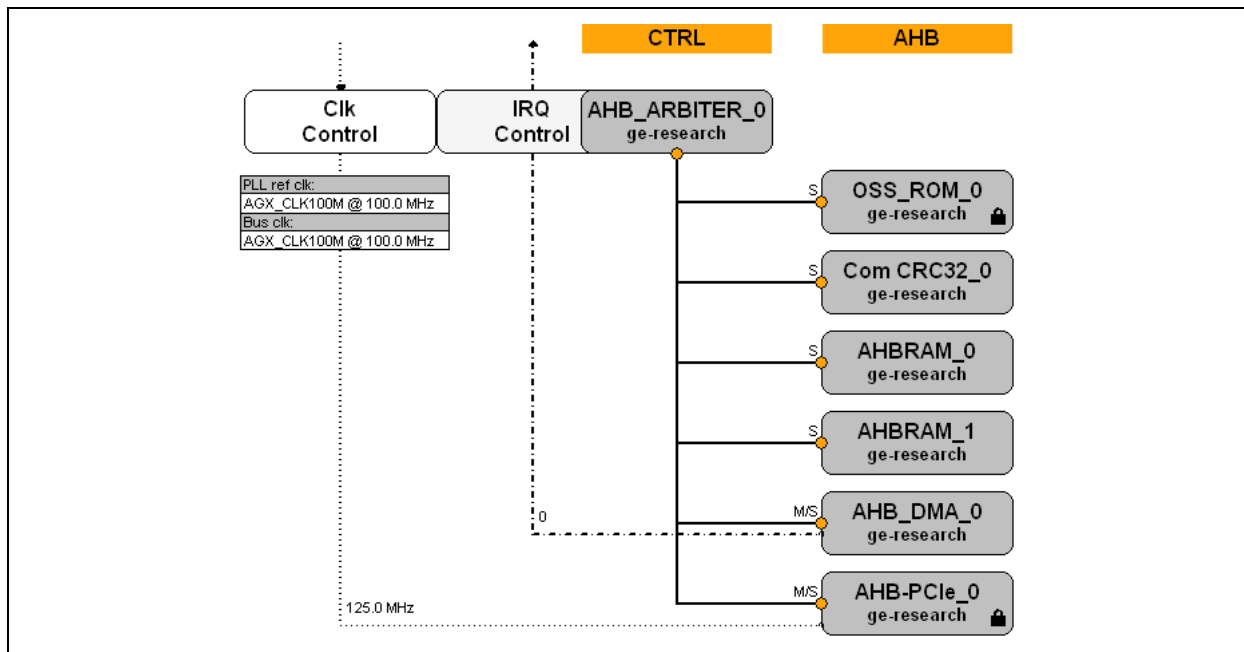


Figure 14. DMA demo design



In this design the FPGA will directly read and write the CPU's RAM with almost no work for the CPU. The general sequence of events in this scenario is as follows:

- The CPU instructs the DMA controller to copy data by setting source, destination and length of the transfer. The CPU may continue with other tasks not depending on the data to be transferred.
- The DMA controller starts copying the data. Accesses to the CPU's RAM are done through the AHB slave port of the AHB-PCIe IP. So in this scenario the DMA controller becomes a bus master and the AHB-PCIe IP a slave. The AHB-PCIe slave maps (parts) of the CPU's RAM to the AHB. Hence accessing the CPU's RAM from within the FPGA is no different than accessing any other IP in the FPGA.
- Upon completion the DMA controller triggers an interrupt to notify the CPU that the transfer is done.

5.1 Test data generation

We already saw in Section 4 that the memory resources within the FPGA are too limited to store large amounts of data. In order to transfer large amounts of data we will generate data on the fly instead of storing it. This is what the Com CRC IP in Figure 14 is used for: When transferring data from the FPGA into CPU's RAM the Com CRC IP will generate pseudo random numbers. When transferring data from the CPU's RAM into the FPGA, the Com CRC IP will calculate a CRC value over all the data. Since the Com CRC IP is just an AHB slave (as indicated by the 'S' next to its bus connection in Figure 14) it cannot initiate any bus accesses. This is what the DMA controller does. For FPGA to RAM transfers, the DMA controller copies data from the Com CRC IP to the AHB-PCIe IP. For RAM to FPGA transfers, the DMA controller copies data from the AHB-PCIe IP to the Com CRC IP.

In order to verify the correctness of the transmitted data two different things - depending on the data direction - have to be done. For FPGA to RAM transfers, the CPU recalculates the pseudo random numbers and compares these recalculated values with the result of the DMA transfer. For RAM to

FPGA transfers, the CPU calculates the CRC of the data to be transmitted and compares this value with the CRC calculated by the Com CRC IP.

The pseudo random number generation is implemented by a linear feedback shift register (LFSR). Before the DMA transfer is started the software initializes the LFSR with a known seed value. This allows the recalculation of the random numbers.

The CRC (cyclic redundancy check) calculates a kind of fingerprint of the transmitted data. If data is corrupted or gets lost during the transfer, the CRC will most likely be different from the CRC of the original data. The CRC itself is a 32 bit value. It is computed with the polynomial which is also used in Ethernet

$(x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + x^0)$.

5.2 Performance Measurement

In order to determine the throughput of the transfer we measure the time it takes for the transfer to complete. We measure the time using two different approaches. The measurement takes place in the FPGA and in the CPU. Inside the FPGA the DMA controller will count the number of clock cycles it takes to copy all the data. In the CPU the time stamp counter register (TSC) will be used to measure the duration of the whole transfer.

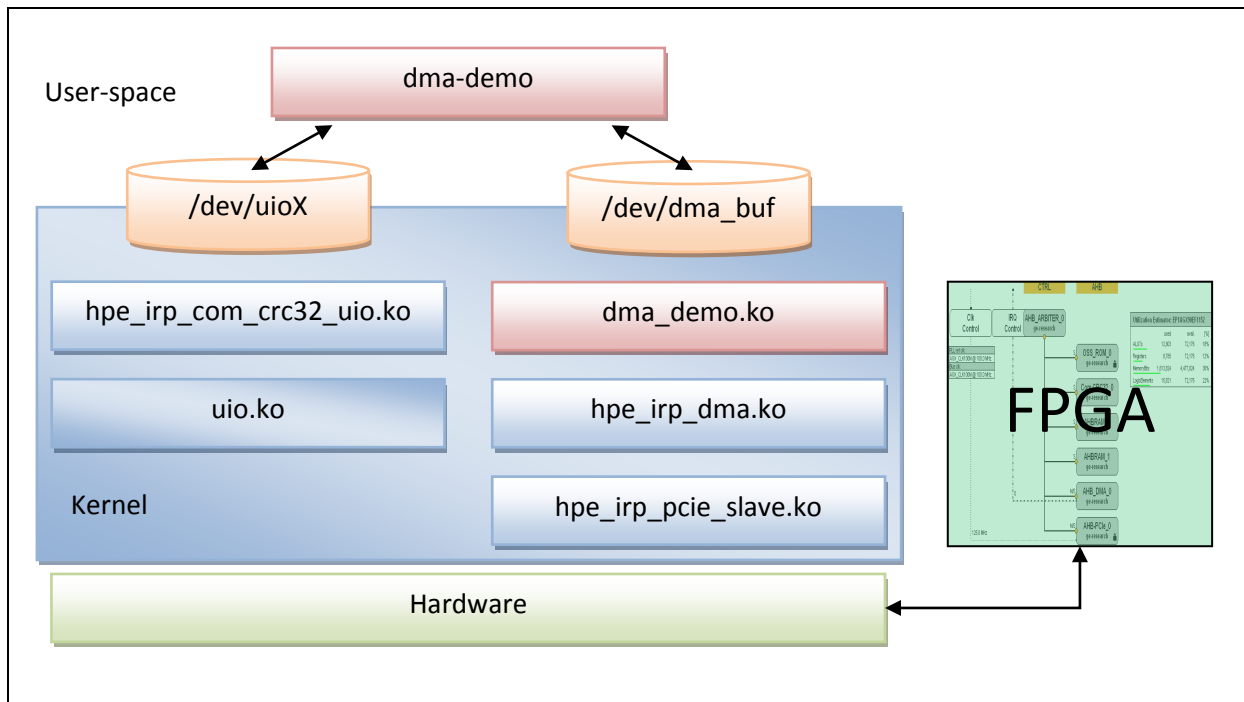
The two approaches are used for two reasons. First, it is possible to validate the accuracy of the measurement. Second, we can do the same transfers using the CPU instead of the DMA controller and compare the results to the relatively small transfers done in Section 4. In the latter case we cannot use the DMA controller's performance counter since it is not involved in this particular scenario.

Even if both clocks, i.e. the TSC and DMA's performance counter, were perfectly accurate we can expect the TSC to report a slightly longer duration. The reason is that the DMA's performance counter only measures the time the DMA controller is busy. More precisely, it measures the time from the first to the last AHB access. From the software point of view, however, further things have to happen until the transfer is considered complete. First of all, immediately after the DMA controller finished copying data to the AHB-PCIe core, this data will be stored in internal buffers of the IP core. Depending on the amount of available flow control credits (see Section 3) and the amount of previously buffered data, it will take some time until the data actually leaves the FPGA. This issue is only relevant for FPGA to RAM transfers, because in the opposite direction there are no buffering effects of this kind. The second reason why the TSC value can be expected higher than the DMA's performance counter has to do with the interrupt latency. By the time the DMA controller completes the transfer it stops its performance counter and simultaneously raises an interrupt. It will take some time until the software actually recognizes this interrupt. We can expect these time differences to be in the order of 100µs. For large DMA transfers which take multiple seconds as intended in this Chapter, these differences can be neglected. However, when performing small transfers, e.g. from or to small AHB-RAMs, these differences have to be considered.

5.3 Measurement Setup

Figure 15 shows the software part of the measurement setup.

Figure 15. Measurement setup for the DMA demo design



The overall test is executed by the user-space application `dma-demo`. This is the executable which is started in the last step in Section 2. The user-space application accesses the FPGA via the two character devices `/dev/uioX` and `/dev/dma_buf`.

The character device `/dev/uioX` is required for the software to access the Com CRC IP. Access to the Com CRC IP is necessary to set the random seed and to initialize and to read the CRC value. This character device is provided by the standard UIO kernel modules.

The second character device `/dev/dma_buf` serves two purposes. First, it lets the user-space application initiate DMA transfers. This can be done via standard `read()` and `write()` calls. The `read()` call reads the random numbers from the Com CRC IP and stores them in the RAM. The `write()` call writes data from the RAM into the Com CRC IP. The second purpose of this character device is to let the user-space software directly access the portion of the RAM which is used as DMA buffer, i.e. the source or destination in DMA transfers. In Section 5.5 more details are given why it is necessary to take special care of the DMA buffer.

Note that the user-space application itself does not directly access the DMA controller. Neither does the kernel module `dma_demo.ko`. This kernel module makes use of the kernel's DMA infrastructure to implement its `read()` and `write()` functions. It is the kernel module `hpe_irp_dma.ko` which provides a generic DMA engine to the DMA infrastructure which in turn is used by `dma_demo.ko`.

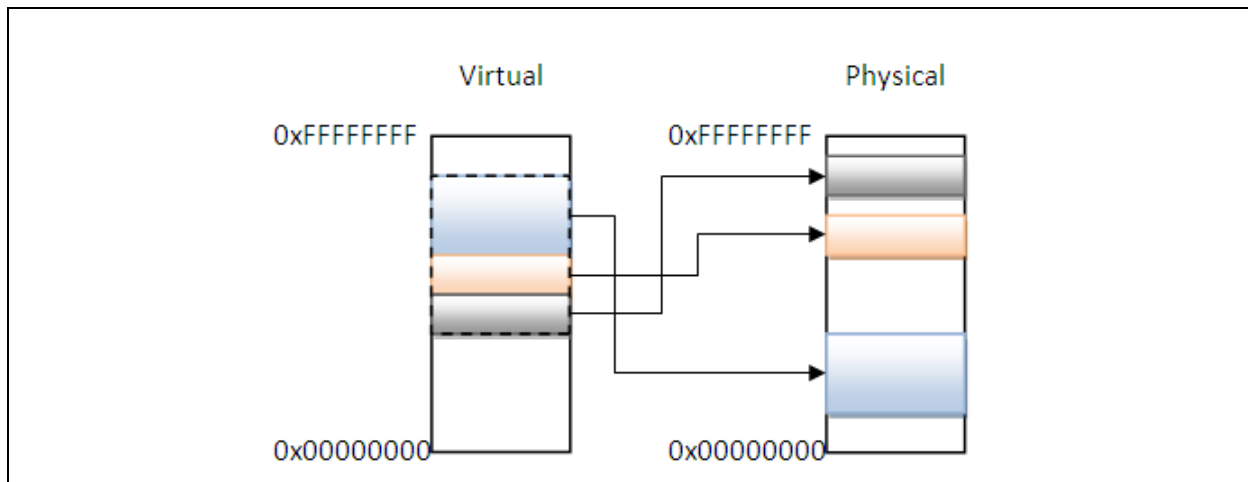
The user-space also does not directly notice the interrupts which are signaled by the DMA controller upon completion of the transfer. Instead the application experiences blocking `read()` and `write()` calls. The kernel module `dma_demo.ko` causes the calling application to sleep until all data is transferred, i.e. the CPU is free to run other tasks during this time. An interrupt from the DMA controller causes the interrupt service routine in the module `hpe_irp_dma.ko` to run. This will in turn cause a callback to `dma_demo.ko` which finally causes the completion of the `read()` or `write()` call.

The kernel module `hpe_irp_pcie_slave.ko` shown in Figure 15 handles the slave part of the AHB-PCIe IP core. It provides functions to map a portion of the CPU's RAM into the AHB address space inside the FPGA.

5.4 Address Mapping

To achieve maximum performance this demo is designed to transfer large amounts of data directly into the RAM which can be accessed by user-space applications. One difficulty that arises with this approach is the fact that every user-space application has its own virtual address space. The operating system sets up an individual mapping between virtual addresses seen by the user-space application and the actual locations in the physical RAM. This poses two problems. First, the mapping between the AHB addresses in the FPGA and an application's virtual address space is unknown. Second, memory ranges which are contiguous in an application's virtual address space may be discontinuous in physical RAM. The latter is caused by the fact that the mapping is done on a per page basis. Different ranges in a process' virtual address space may be mapped to completely different ranges of physical memory. This is depicted in Figure 16. Contiguous virtual memory may be discontinuous in physical memory. A further consequence of this mapping is the fragmentation of the physical memory. The more often these mappings are established and destroyed the less likely it is to find a large contiguous range of physical memory.

Figure 16. Contiguous virtual memory may be discontinuous in physical memory

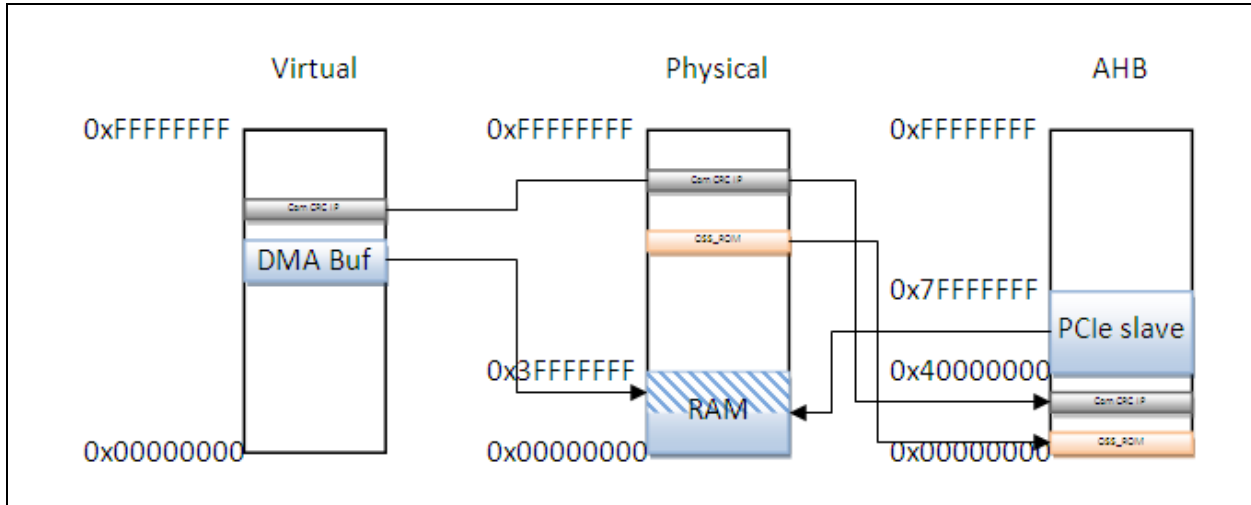


When the FPGA accesses the CPU's RAM via PCIe it needs to supply an address of the location it wants to access. These PCIe addresses are the same as the physical ones (on this platform), but they are not the same as the ones seen on the AHB inside the FPGA. This is depicted in Figure 17. For instance the middle of the RAM with the physical address `0x20000000` has the AHB address `0x60000000`. Of course the exact addresses depend on various configuration options. The addresses shown in Figure 17 represent the configuration given in Section 2.

Note that these mappings are unidirectional. If some AHB master accesses AHB address `0x60000000` it will end up accessing the CPU's RAM at physical address `0x20000000`. If the CPU accesses the physical address `0x20000000` it will also end up in its own RAM and **not** in the FPGA. If some AHB master accesses AHB address `0x00000000` it will access the OSS_ROM no matter if and where this memory is mapped in the physical address range. Such accesses are internal to the FPGA and not visible to the CPU. Only those AHB accesses within the PCIe slave range (`0x40000000-0x7FFFFFFF` in the example) will be forwarded via PCIe. The corresponding memory range in the physical memory

does not necessarily have to start at 0x00000000. It can be mapped at different locations and can be reconfigured at run time. However, since the platform has 1GiB ($=2^{30}=0x40000000$) of RAM this is the most convenient setting. All the RAM can be accessed without changing the mapping.

Figure 17. Address mapping between virtual memory, physical memory and AHB addresses



5.5 DMA Buffer

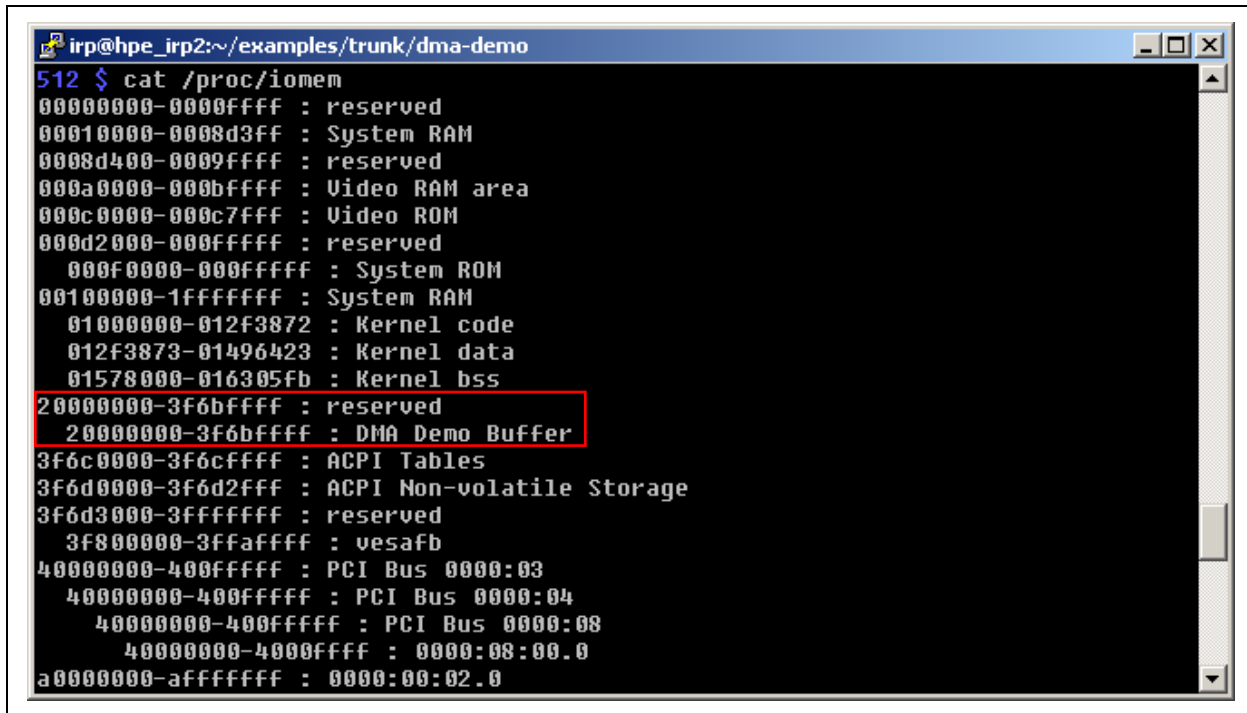
In addition to the addressing and fragmentation issue described above, (by default) the Linux kernel will not provide DMA capable buffers of more than 4MiB. To perform DMA transfers with more than 500MB en bloc without intermediate copies, we use a little trick in our demo design.

At boot time we instruct the Linux kernel to reserve the upper 512MiB of the 1GiB of RAM. This is indicated in Figure 17 by the shaded area in the physical memory from 0x20000000 to 0x3FFFFFFF. This is what the kernel command line parameters `mem=` and `memmap=` are good for in Section 2. These parameters ensure that the upper half of the RAM stays unused and that nothing else will be mapped into this memory range.

The kernel module `dma_demo.ko` (see Section 5.3) takes care of this memory. The user-space application `dma-demo` can get access to this memory, i.e. map it into its virtual address space, by calling `mmap()` on the character device `/dev/dma_buf`. As indicated in Figure 15, this character device is provided by `dma_demo.ko` which finally implements the `mmap()` call.

To verify that the memory reservation succeeded we can have a look at the file `/proc/iomem` as show in Figure 18.

Figure 18. The contents of /proc/iomem after reserving the upper 512MiB of RAM



```
irp@hpe_irp2:~/examples/trunk/dma-demo
512 $ cat /proc/iomem
00000000-0000ffff : reserved
00010000-0008d3ff : System RAM
0008d400-0009ffff : reserved
000a0000-000bffff : Video RAM area
000c0000-000c7fff : Video ROM
000d2000-000fffff : reserved
000f0000-000fffff : System ROM
00100000-1fffffff : System RAM
01000000-012f3872 : Kernel code
012f3873-01496423 : Kernel data
01578000-016305fb : Kernel bss
20000000-3f6bffff : reserved
20000000-3f6bffff : DMA Demo Buffer
3f6c0000-3f6cffff : ACPI Tables
3f6d0000-3f6d2fff : ACPI Non-volatile Storage
3f6d3000-3fffffff : reserved
3f800000-3ffaffff : vesafb
40000000-400fffff : PCI Bus 0000:03
40000000-400fffff : PCI Bus 0000:04
40000000-400fffff : PCI Bus 0000:08
40000000-400fffff : 0000:08:00.0
a0000000-afffffff : 0000:00:02.0
```

As we can see in Figure 18 slightly less than 512MiB ($=2^{29}=0x20000000=536870912=536.870912\text{MB}$) are reserved. The upper most part is used by ACPI and vesafb. The kernel module `dma_demo.ko` takes care of this and only provides the part which is unused. This is about 527MB.

5.6 Demo application

The demo application `dma-demo` sets up the DMA buffer using `mmap()` as described in Section 5.5 and UIO access to the Com CRC IP core. In principle UIO uses the same mechanism, namely `mmap()` to get access to the CRC core. After mapping these two memory ranges, its virtual address space may look like the one shown in Figure 17. The actual mappings can be examined through `/proc/<PID>/maps` (where `<PID>` is the process ID). In Figure 19 we can see the mappings for the DMA buffer and CRC IP (UIO) as well as further mappings, e.g. for the code and data of the executable itself (`dma-demo`), the program memory (heap and stack) and the C library (`libc`).

Figure 19. The memory mappings of the dma-demo

```
irp@hpe_irp2:~  
515 $ sudo cat /proc/4491/maps  
08048000-0804c000 r-xp 00000000 08:02 124169 /home/irp/examples/trunk/dma-de  
mo/dma-demo  
0804c000-0804d000 r--p 00003000 08:02 124169 /home/irp/examples/trunk/dma-de  
mo/dma-demo  
0804d000-0804e000 rw-p 00004000 08:02 124169 /home/irp/examples/trunk/dma-de  
mo/dma-demo  
0804e000-08071000 rw-p 00000000 00:00 0 [heap]  
97f04000-b75c4000 rw-s 20000000 00:0e 6826 /dev/dma_buf  
b75c4000-b75c5000 rw-p 00000000 00:00 0  
b75c5000-b7703000 r-xp 00000000 08:02 708893 /lib/libc-2.10.1.so  
b7703000-b7705000 r--p 0013e000 08:02 708893 /lib/libc-2.10.1.so  
b7705000-b7706000 rw-p 00140000 08:02 708893 /lib/libc-2.10.1.so  
b7706000-b770a000 rw-p 00000000 00:00 0  
b770e000-b770f000 rw-s c1000000 00:0e 6795 /dev/uio0  
b770f000-b7711000 rw-p 00000000 00:00 0  
b7711000-b7712000 r-xp 00000000 00:00 0 [vdso]  
b7712000-b772e000 r-xp 00000000 08:02 708672 /lib/ld-2.10.1.so  
b772e000-b772f000 r--p 0001c000 08:02 708672 /lib/ld-2.10.1.so  
b772f000-b7730000 rw-p 0001d000 08:02 708672 /lib/ld-2.10.1.so  
bf9fb000-bfa10000 rw-p 00000000 00:00 0 [stack]  
516 $
```

The last prerequisite for the demo application is the access to the DMA controller's performance counter. The performance counter is exposed as sysfs attribute by the kernel module `hpe_irp_dma.ko`. So reading and writing it can be done through ordinary file operations on `/sys/bus/hpe_irp/devices/.../perf_cnt`.

5.7 Test Procedures

The demo application `dma-demo` performs two tests in two variants. The first test is a data transfer from the CPU's RAM to the FPGA. The second test is a data transfer from the FPGA to the CPU's RAM. Both tests are done once with the DMA controller and once with the CPU.

The first test consists of the following steps:

- The CPU initializes the DMA buffer with random numbers.
- The CPU computes the CRC of the DMA buffer contents.
- The CPU initializes the CRC value in the Com CRC IP.
- The CPU starts a DMA transfer: The contents of the DMA buffer are copied to the Com CRC IP.
- After the transfer is complete the CPU stores the measured duration(s) of the transfer.
- The CPU reads back the CRC value from the Com CRC IP and compares it to the one calculated in step 2.
- The resulting performance is printed.

The second test consists of the following steps:

- The CPU initializes the DMA buffer with random numbers. These are intended to be overwritten and to potentially detect missing data.
- The CPU initializes the random number generator in the FPGA with a different seed than the one used in step 1.
- The CPU starts a DMA transfer: Random numbers from the FPGA are copied into the DMA buffer.
- After the transfer is complete the CPU stores the measured duration(s) of the transfer.
- The CPU verifies the contents of the DMA buffer by recalculating the random numbers with the seed used in step 2.
- The resulting performance is printed.

Note: The performance of the intermediate steps is also printed out. If you carefully examine the results of the first test, you will find out that copying the data to the FPGA and computing the CRC there is more efficient than computing the CRC locally in software - even though the bandwidth between CPU and RAM is greater than the bandwidth between FPGA and RAM.

5.8 Demo Application: Command Line Parameters

The demo application `dma-demo` provides several command line options to skip certain parts of the test. The parameters are given in Table 2.

If the `dma-demo` is started without any parameters it will take approximately 5 minutes.

Table 2: Command line options of `dma-demo`

Option	Description
-c	Skip timer calibration. Assume TSC frequency = 1.6GHz.
-k	Skip kernel command line check. Assume that required kernel command line settings (mem=..., mmap=...) are correct.
-m	Skip loaded modules check. Assume that all required kernel modules are loaded.
-b <n>	Transfer <n> MB (10 ⁶) in each test. If omitted the entire reserved buffer is used. The size of the actually used buffer size is printed at the beginning of the test.
-<N>	Skip test number <N> (<N>=1..4). For instance if you want to skip the CPU-based tests, you can specify "-3 -4" as shown in Chapter 2.

5.9 Results

Table 3 shows the results of running the `dma-demo`. We can see that the use of DMA increases the performance. The performance gain is mainly achieved by the use of larger payloads (128 bytes). The use of larger payloads is possible, because the transactions are initiated by the FPGA and the AHB-PCIe core can generate optimum sized packets. The second performance advantage is offloading the copying work from the CPU to the DMA controller thereby freeing the CPU for other tasks.

As we can see in the first row of Table 3 the DMA performance is still significantly less than the theory predicts. A more detailed analysis of this can be found in [6].

The second row of Table 3 shows that the measurements are consistent with the ones in Chapter 4.

Table 3: Results of the dma-demo¹⁰

	MEM→FPGA	FPGA→MEM
DMA	48 MB/s	110 MB/s
CPU	32 MB/s	2.5 MB/s

Note that neither the write combing (shown in Section 4.16) nor the prefetching (shown in Section 4.17) is applicable in this test. The reason is that data is written to and read from a single address. Activating write combining would cause test 3 to fail with a wrong CRC. Successive writes to the same address would be collapsed to a single write with the last written data. Activating prefetching requires enabling caching. Successive reads from the random number generator would be cached resulting always in the same number instead of generating new ones.

¹⁰ Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. Configurations: Refer to Section 2.2 of [4] for complete configuration details. For more information go to <http://www.intel.com/performance>. Intel does not control or audit the design or implementation of third party benchmark data or Web sites referenced in this document. Intel encourages all of its customers to visit the referenced Web sites or others where similar performance benchmark data are reported and confirm whether the referenced benchmark data are accurate and reflect performance of systems available for purchase.

6 Summary

In the course of this tutorial we have seen how to use the AMBA-IP-Manager to build FPGA designs which allow a precise analysis of the performance. Especially the AHB trace module with its capability to record cycle accurate timing information allows for very detailed analyses.

We have seen that the PCIe connection between the Intel® Atom™ and the FPGA is capable of transferring up to 250 Mbytes/s. To get close to the theoretical maximum it is important to use sufficiently large payloads. Otherwise the protocol overhead becomes dominant, especially for read transactions. What seems to be a high speed bus can be surprisingly slow.

In this tutorial we considered two scenarios: data transfers initiated by the CPU and data transfers initiated by the DMA controller. This results in four possible types of transfers:

- The CPU reads from the FPGA
- The CPU writes to the FPGA
- The FPGA (DMA controller) reads from main memory
- The FPGA (DMA controller) writes to main memory

In general reads are less efficient than writes. Reads involve more overhead, because two PCIe packets per read are required: a request and a completion. Additionally there is a delay between the reception of a request and the transmission of the completion.

The first scenario is typical for many simple IP cores. As the theory predicts, the performance is in the order of 32 MB/s¹¹ for writing and 2.5 MB/s¹² for reading. This is far below the theoretical maximum, but more than enough for IP cores like UART, I²C and CAN. The relatively low performance compared to the capabilities of the PCIe connection is mainly caused by the small payloads. For instance a CAN message is only 8 bytes. Using DMA in this case would be even less efficient. Setting up a DMA transfer usually involves writing 4*4 bytes: source address, destination address, length and control flags.

We saw how to improve the performance for CPU initiated transfers by using write combining and prefetching. The results in the Sections 4.16 and 4.17 show a throughput of 177 MB/s¹³ for writing and 40 MB/s¹⁴ for reading. However, great care has to be taken when using these mechanisms on I/O memory.

The second scenario (DMA), is typical for high bandwidth applications like video processing. Chapter 5 shows a write performance in the order of 110MB/s and a read performance in the order of 50MB/s.

¹¹ Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. Configurations: Refer to Section 2.2 of [4] for complete configuration details. For more information go to <http://www.intel.com/performance>

Intel does not control or audit the design or implementation of third party benchmark data or Web sites referenced in this document. Intel encourages all of its customers to visit the referenced Web sites or others where similar performance benchmark data are reported and confirm whether the referenced benchmark data are accurate and reflect performance of systems available for purchase.

¹² Ibid

¹³ Ibid

¹⁴ Ibid

Remote Lab Demonstration Tutorial

Summary

Note that the actual data flow direction is opposite to the one above: When the DMA controller writes data, this data leaves the FPGA. When the CPU writes data it enters the FPGA.

The latency of a write access is difficult to measure directly, because there are no synchronous timestamps available in the FPGA and the INTEL™ Atom™ Processor. Therefore this time can only be estimated. According to Test 2 the round trip time of a single read access from the AHB RAM with a TLP with a payload size of 4 bytes is in the order of 1500ns. The latency of a write access to the AHB RAM can be estimated with the half of the roundtrip time of a read access from the AHB RAM. This means the latency of a write access to the AHB RAM would in the order of 750ns¹⁵. A simulation of the AHB-PCIe IP core shows a latency of about 300ns¹⁶ through the IP core itself.

¹⁵ Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. Configurations: Refer to Section 2.2 of [4] for complete configuration details. For more information go to <http://www.intel.com/performance>

Intel does not control or audit the design or implementation of third party benchmark data or Web sites referenced in this document. Intel encourages all of its customers to visit the referenced Web sites or others where similar performance benchmark data are reported and confirm whether the referenced benchmark data are accurate and reflect performance of systems available for purchase.

¹⁶ Ibid