PCI Express - Architecture, Implementation and Measurement.

Jan Marjanovic (MTCA Tech Lab/DESY)
@janmarjanovic

2019-10-22

Design and Validation of Multi-Gigabit Systems
DESY, Hamburg

MicroTCA TECHNOLOGY LAB

HELMHOLTZ RESEARCH FOR GRAND CHALLENGES

DESY.

Introduction

- MicroTCA and PCI Express
- PCI Express protocol
- PCI Express protocol - physical layer
- PCI Express protocol - configuration
- FPGA implementation
- Examples
- Further resources and summary

This is a hardware event - we focus on the hardware side

Other resources available on system level:

- MTCA training courses: Basic and Advanced
  `https://techlab.desy.de/services/training/index_eng.html`

- tutorial talks:

  MTCAWS 2018:
  `https://github.com/MicroTCA-Tech-Lab/damc-tck7-fpga-bsp/raw/demo-mtcaws2018/docs/demo/mtcaws2018_marjanovic_dma.pdf`

  MTCAWS China 2019:
  `https://indico.ihep.ac.cn/event/9651/contribution/24/material/slides/0.pdf`

MicroTCA and PCI Express

A typical ADC system in MicroTCA is shown here:

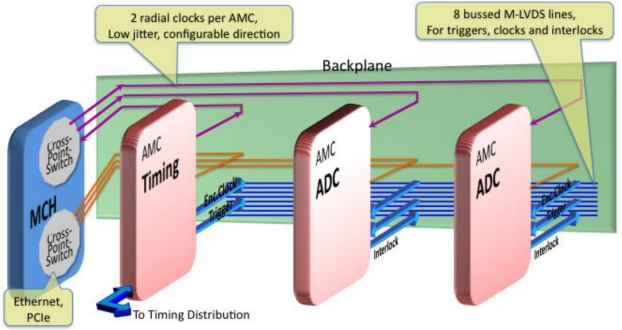


Figure 6-6: A 'typical' analog front-end.
CPU and further AMC's are not shown

from PICMG® MicroTCA.4 Enhancements for Rear I/O and Timing R1.0

A typical board in such a system is shown here:

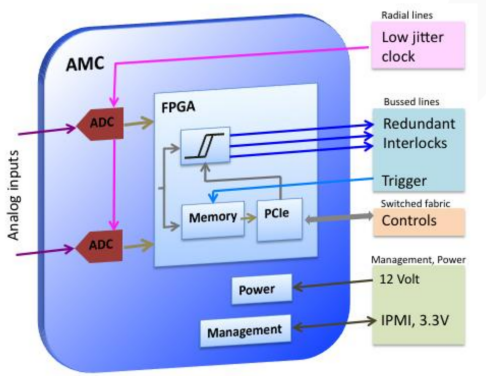
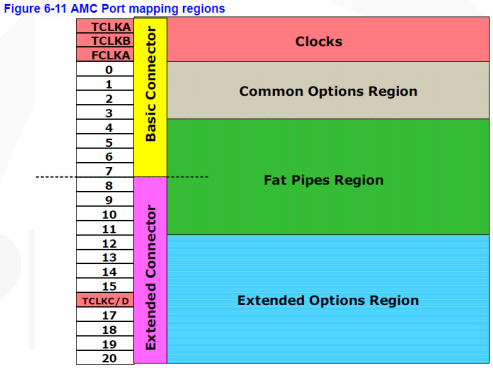


Figure 6-7: An example of a fast front-end (ADC AMC0)

from PICMG® MicroTCA.4 Enhancements for Rear I/O and Timing R1.0

Ports on an Advanced Mezzanine Cards

AMC standard provides some guidance on how the ports should be assigned:



from PICMG® Advanced Mezzanine Card AMC.O Specification R2.0

MicroTCA backplane connections as specified by MicroTCA.4 standard:

REQ 6-6 Port 0 **should** be used for base Ethernet interface. Port 1 is connected to the optional (redundant) MCH.

REQ 6-7 Port 4 to 7 **should** be used for PCIe.

REQ 6-8 Port 12 to 15 **may** be used for application specific wire-ring.

REQ 6-9 Ports 17- 20 **should** be wired as a bus according Section 6.4.

REQ 6-10 FCLKA (Clk3) **should** be used for PCIe clock distribution as defined in MTCA.O R1.0

...

from PICMG® Specification MTCA.4 Revision 1.0: MicroTCA Enhancements for Rear I/O and Precision Timing

MicroTCA backplane connections as specified by MicroTCA.4 standard:

REQ 6-6 Port 0 **should** be used for base Ethernet interface. Port 1 is connected to the optional (redundant) MCH.

REQ 6-7 Port 4 to 7 **should** be used for PCIe.

REQ 6-8 Port 12 to 15 **may** be used for application specific wire-ring.

REQ 6-9 Ports 17- 20 **should** be wired as a bus according Section 6.4.

REQ 6-10 FCLKA (Clk3) **should** be used for PCIe clock distribution as defined in MTCA.O R1.0

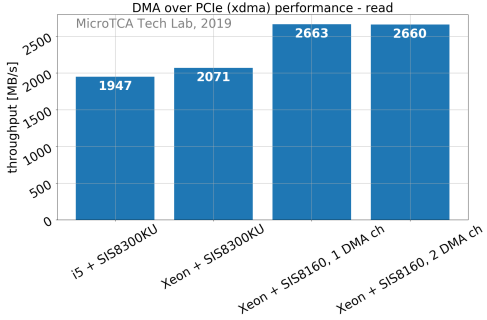
...

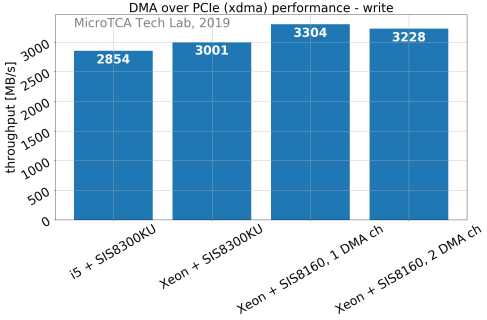
from PICMG® Specification MTCA.4 Revision 1.0: MicroTCA Enhancements for Rear I/O and Precision Timing

Reasons to use PCI Express

- High-throughput (up to 64 Gbps (gen 3, x8 link))
- Low-latency
- Software model (memory-based transaction, method to detect devices on the bus)

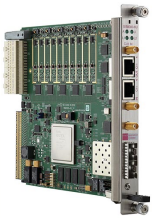
Example of the throughput achieved with two of the boards in MicroTCA:





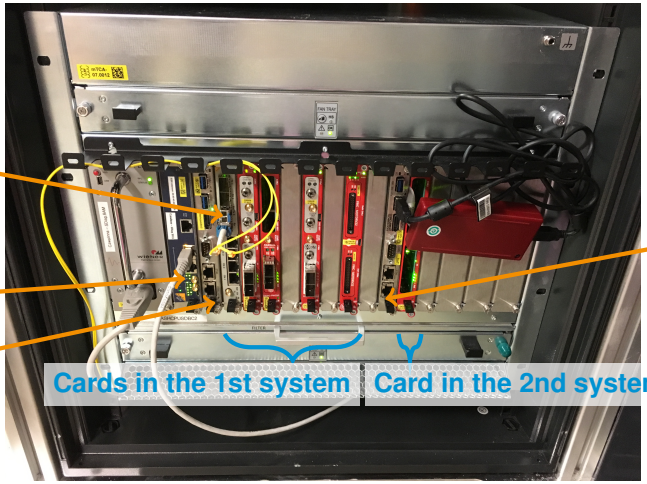
Reasons to use PCI Express

A lot of boards available on the market use PCI Express as main/only communication protocol.
FPGAs also allow implementation of other protocols (e.g. 10 and 40 Gb Ethernet and Serial Rapid IO).



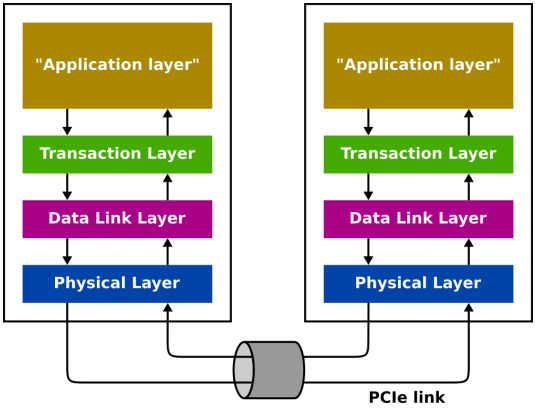



Example from FLASH: PCI Express switch enables splitting crate in two separate systems, but with shared timing and interlocks.



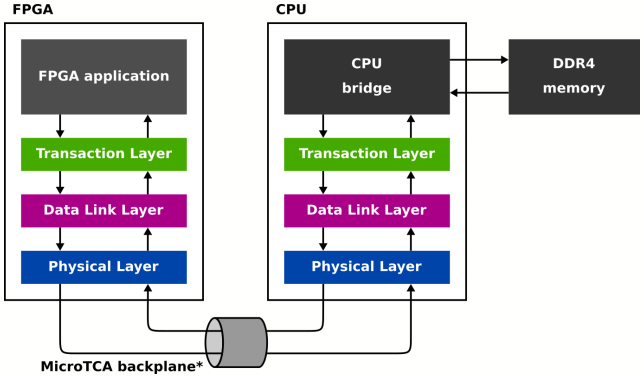
PCI Express protocol

PCIe express is high-speed protocol for PC extension cards. Different form-factors are also available. Link out of 1, 2, 4, 8 or 16 lanes, lane rate 2.5 GT/s, 5.0 GT/s, 8.0 GT/s and 16.0 GT/s.



inspired by Figure 2-12 from PCI Express Technology 3.0

Concrete example:



* real PCIe link in MicroTCA system includes a PCIe switch in MicroTCA Carrier Hub (MCH)

Physical Layer

- two differential pairs (TX and RX) per lane
- system clock (typical 100 MHz)
- 8b/10b (2.5 and 5.0 GT/s) or 128b/130b encoding (8.0 GT/s)
- Link Training and Status State Machine (LTSSM)
- in FPGAs implemented with transceivers + some logic

Data Link Layer

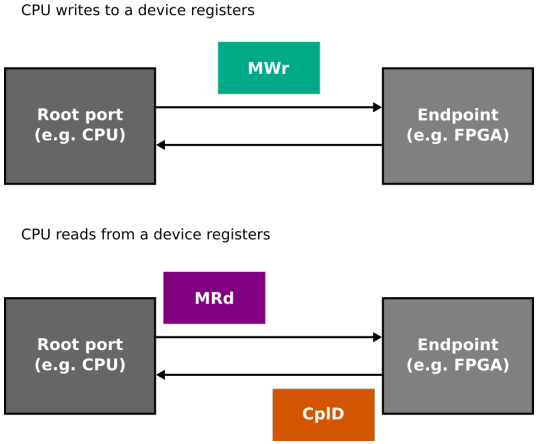
- Exhange of flow-control credits (= free buffers in receiver)
- CRC check, ACK/NAK, re-transmission

Transaction Layer

- ”Real application” packets - memory read and write, ...

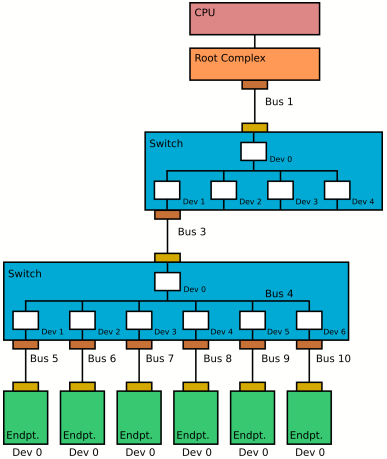
PCIe TLP transaction types

Packets can be one of several types: Memory Read, Memory Write, Completion, Completion with Data, Messages, Configuration Accesses and some legacy types.

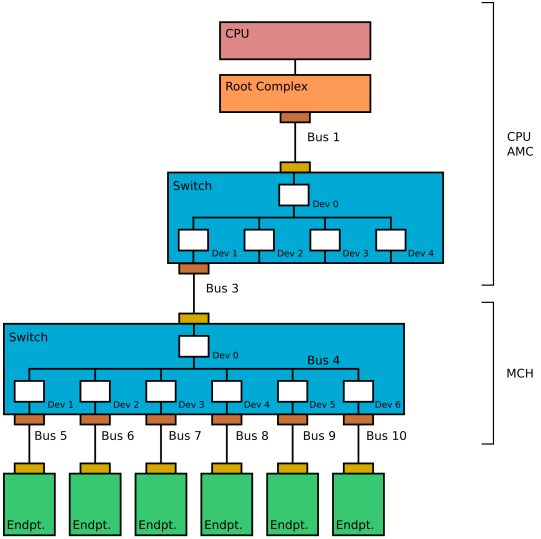


PCIe switches, buses, devices

Since PCIe is a point-to-point link, switches are used to connect more than 1 device to the CPU. Each PCIe device has its own address (bus:device.function), assigned during the enumeration process.



In the switch on the MCH one or more slots are configured as Upstream AMC.

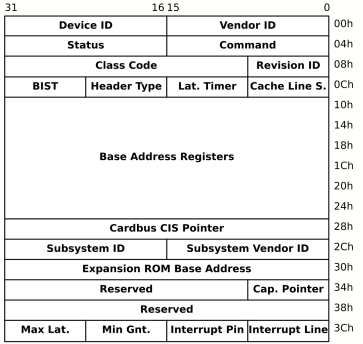


PCIe configuration space

Configuration Register Space defined for each Function. The first 256 bytes are the same as for PCI, a total of 4 kB for PCIe Extended Configuration Area.

The configuration space contains information such as: Vendor and Device ID, Link Status, Base Address Registers, ...

No driver [1] is needed to read this space. Drivers in Linux use Vendor ID and Device ID (and Subsystem Vendor ID and Subsystem Device ID) to claim devices.

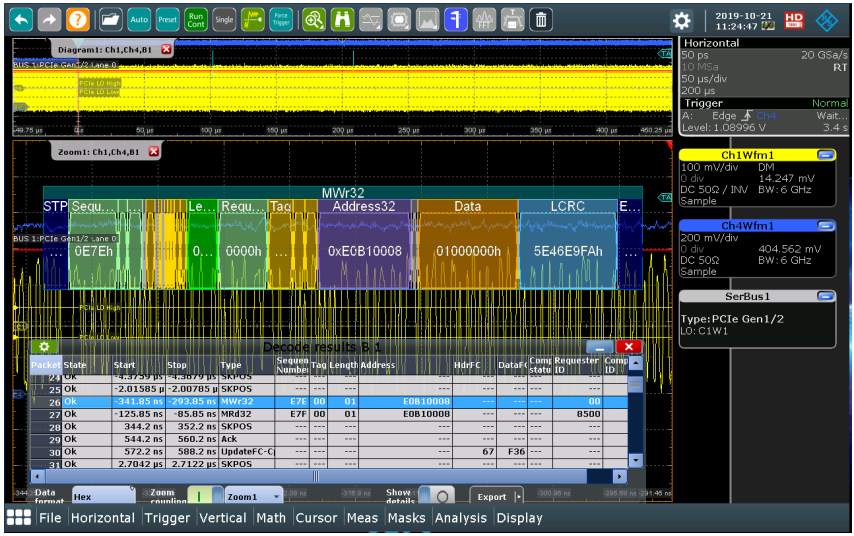


By Vijay Kumar Vijaykumar - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=3181779

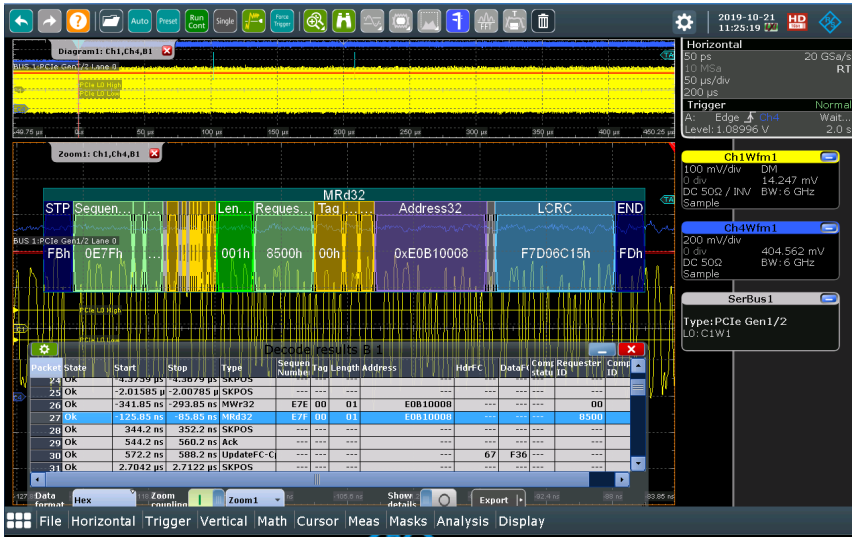
[1] there is a driver embedded in OS, usually, i.e. in GNU/Linux

TLP examples

MWr - Memory Write



MRd - Memory Read



CplD - Completion with Data

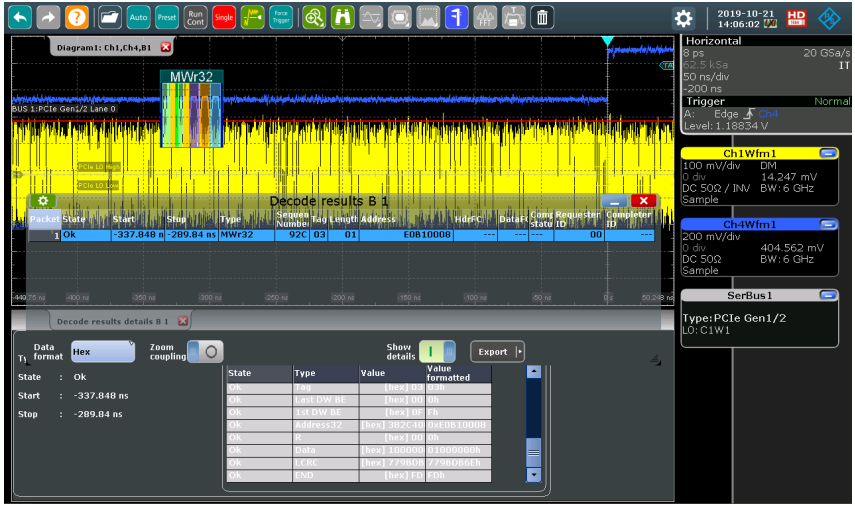
CfgRd - Configuration Read

CfgWr - Configuration Write

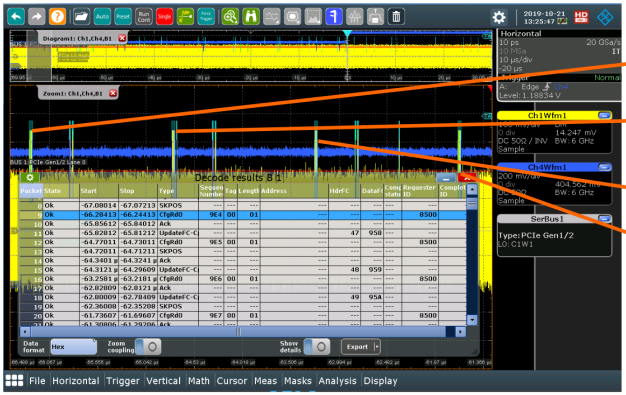
Packet	State	Start	Stop	Type	Sequen Numbe	Tag	Length	Address	HdrFC	DataFC	Comp Status	Requester ID	Complet ID
169	Ok	-1.58736 µ	-1.57136 µ	UpdateFC-C	---	---	---	---	71	97F	---	---	---
170	Ok	-999.35 ns	-991.35 ns	SKPOS	---	---	---	---	---	---	---	---	---
171	Ok	-533.34 ns	-485.34 ns	CfgWr0	A0F	00	01	---	---	---	---	---	8500
172	Ok	-107.34 ns	-91.34 ns	Ack	---	---	---	---	---	---	---	---	---
173	Ok	-79.34 ns	-63.34 ns	UpdateFC-C	---	---	---	---	72	97F	---	---	---
174	Ok	938.68 ns	978.68 ns	CfgRd0	A10	00	01	---	---	---	---	---	8500
175	Ok	1.36069 µs	1.36869 µs	SKPOS	---	---	---	---	---	---	---	---	---
176	Ok	1.37069 µs	1.38668 µs	Ack	---	---	---	---	---	---	---	---	---
177	Ok	1.39668 µs	1.41268 µs	UpdateFC-C	---	---	---	---	73	980	---	---	---

Memory write → LED

Turn on an LED with Memory Write



Software ”walking” Configuration Area



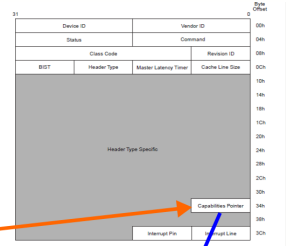


Figure 7-4: Common Configuration Space Header

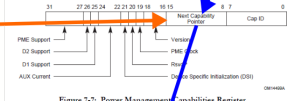


Figure 7-7: Power Management Capabilities Register

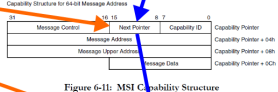
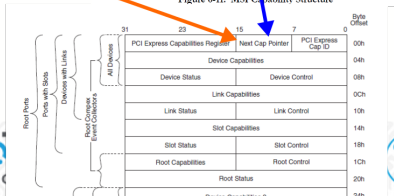


Figure 6-11: MSI Capability Structure

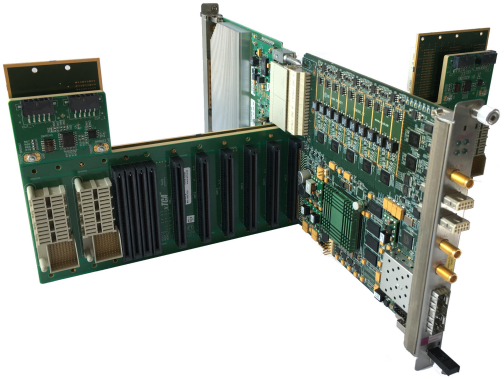


PCI Express protocol Physical Layer

Signal integrity

In a typical MicroTCA systems a signal integrity is an important topic.

- Each high speed signal passes 2 card-edge connectors.
- Components from different vendors are plugged together



Indication of poor signal integrity

Link not running at the maximum possible rate:

PCIe Link Status Menu

	AMC1	AMC2	AMC3	AMC4	AMC5	AMC6	AMC7	AMC8	AMC9	AMC10	AMC11	AMC12	OPT1	RTM
	4..7	4..7	4..7	4..7	4..7	4..7	4..7	4..7	4..7	4..7	4..7	4..7		
	-	-	-	-	-	-	-	-	x4	x4	-	-	x8	-
Link Speed	-	-	-	-	-	-	-	-	5 GT/s	2.5 GT/s	-	-	8 GT/s	-

High rate of DLLP errors (rule of thumb: 1 correctable (re-trans) error per minute is OK):

PCIe Error Counters Menu

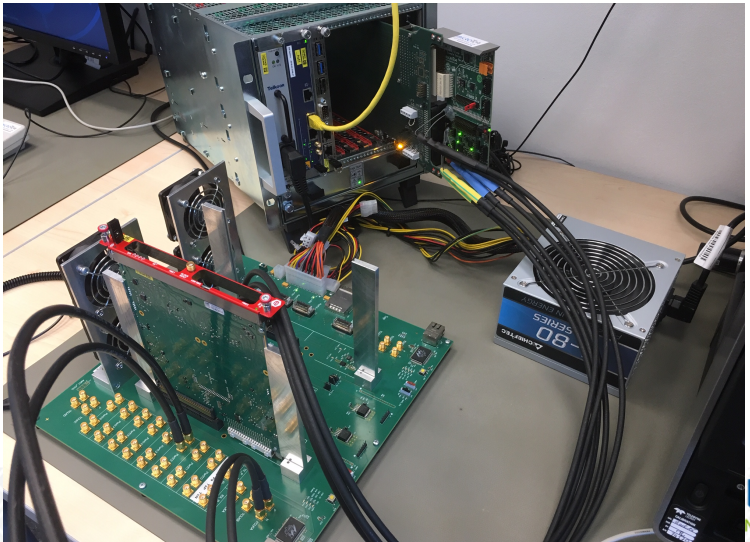
	AMC1	AMC2	AMC3	AMC4	AMC5	AMC
	4..7	4..7	4..7	4..7	4..7	4..
	x4	x4	x4	x4	x4	x4
RCV_CNT	0	255	0	0	0	0
Bad TLP	0	0	0	0	0	0
Bad DLLP	0	65	0	0	0	0

Reset Refresh

Signal integrity measurement

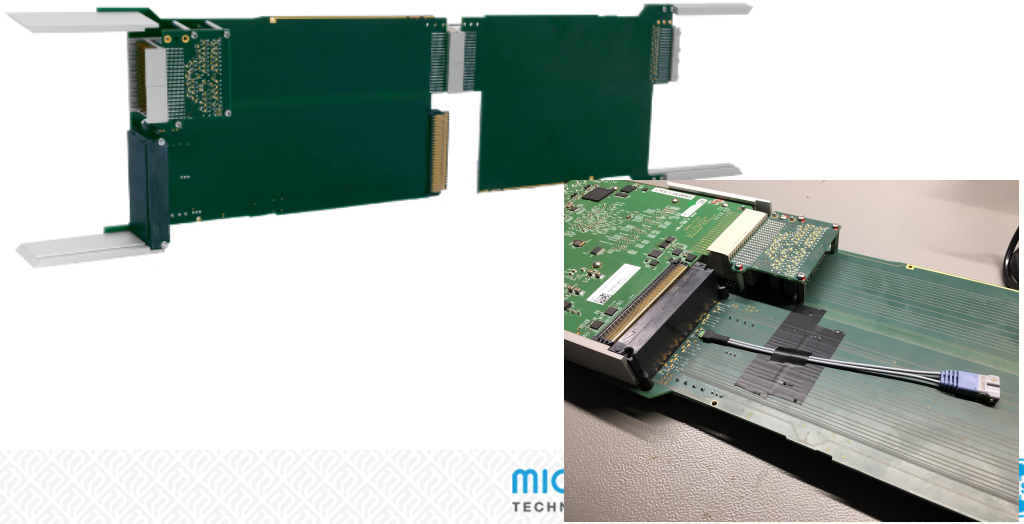
When developing or debugging boards in MicroTCA (or any other form factor) having test harnesses is extremely useful.

One example is an AMC test harness (PCIe connected with coax cables - 6 in total):



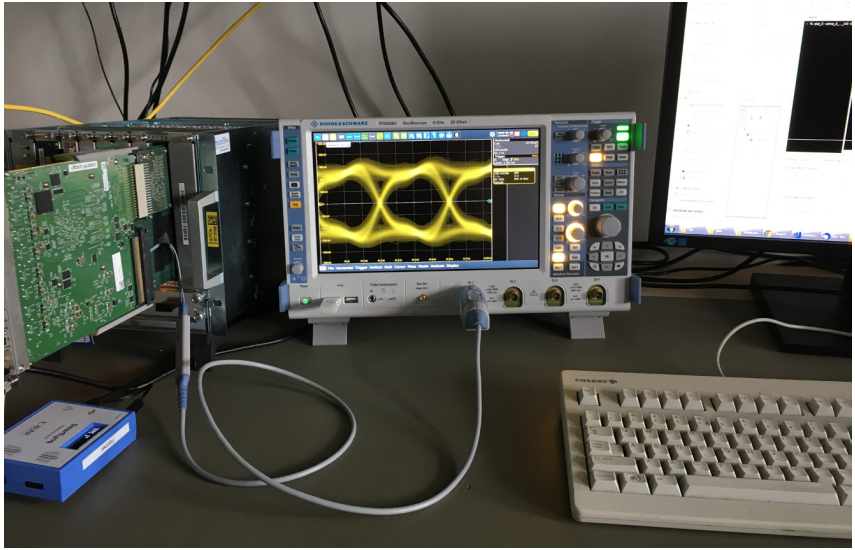
Signal integrity measurement

Extender boards (e.g. from N.A.T. - `https://www.nateurope.com/products/NAMC-EXT-RTM.html`) is a good equipment when we want to observe the data on the backplane:

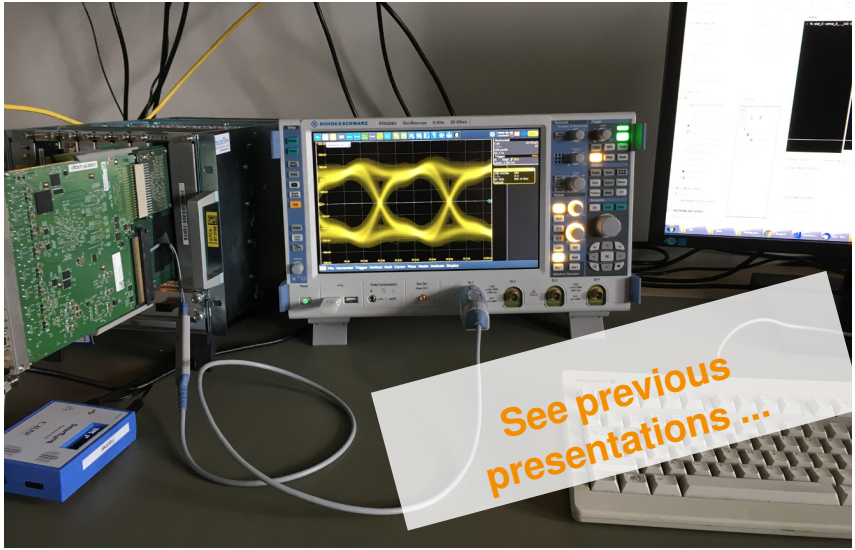


Signal integrity measurement

With oscilloscope, PCIe 2.5 GT/s:



With oscilloscope, PCIe 2.5 GT/s:



In System IBERT allows measuring link quality during operation (everything in grey is not needed for normal operation of the link and is there only to be used for eye scan):

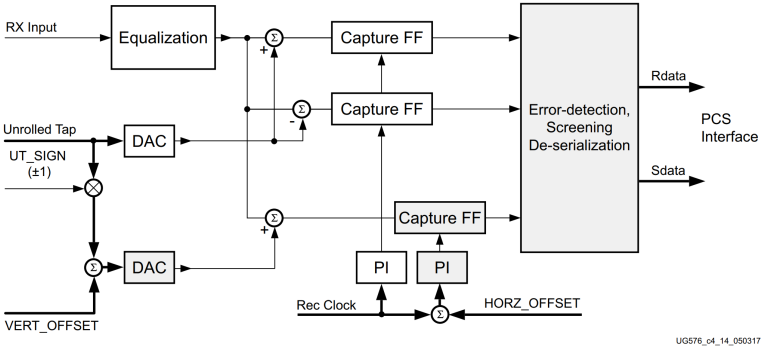
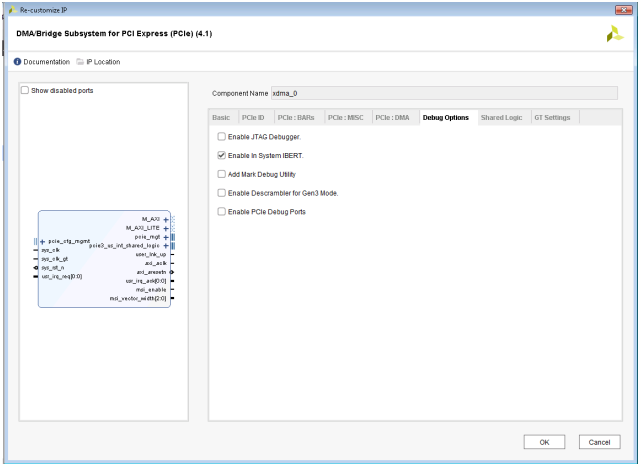


Figure 4-19: **PMA Architecture to Support Eye Scan**

from

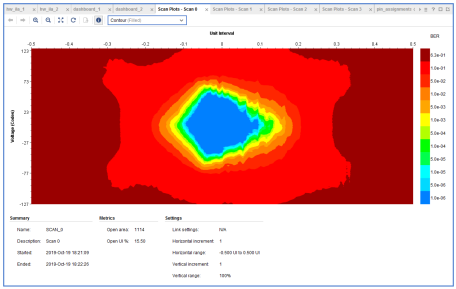
https://www.xilinx.com/support/documentation/user_guides/ug576-ultrascale-gth-transceivers.pdf

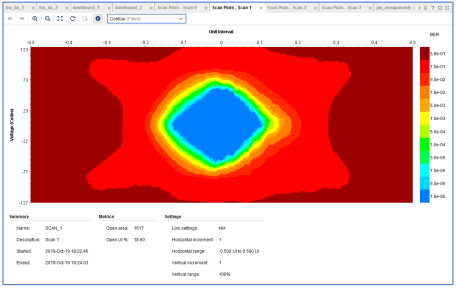
In System IBERT is enabled in IP configuration:

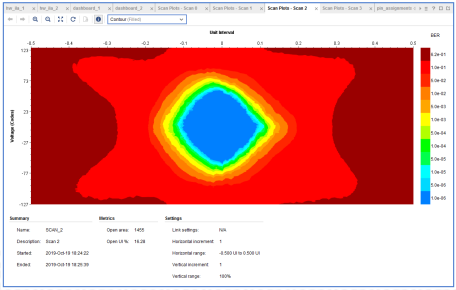


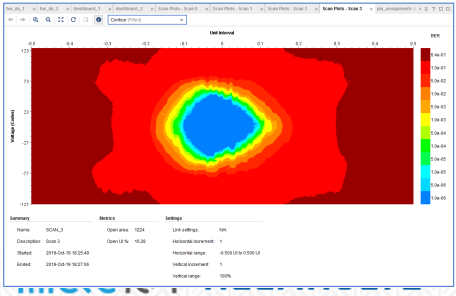
Measurements: IBERT

ELMA crate (12 slot) with SIS8300-KU at 8 GT/s



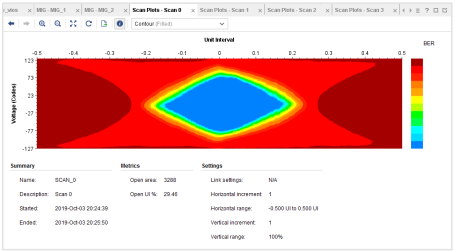


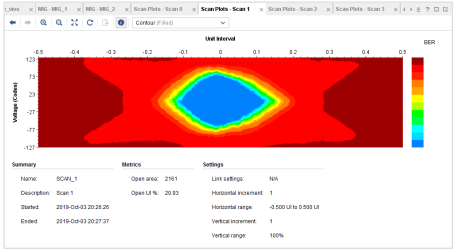


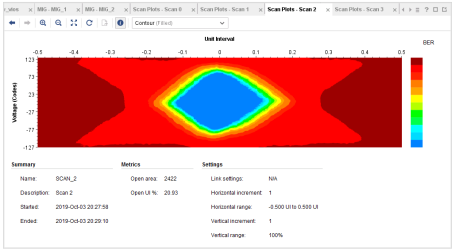


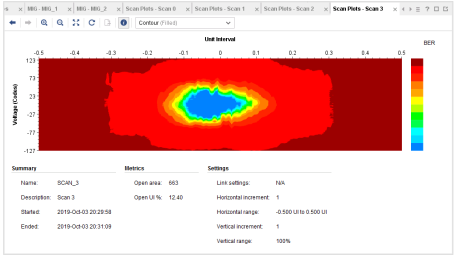
Measurements: IBERT

Schroff crate (12 slot) with SIS8160 at 8 GT/s



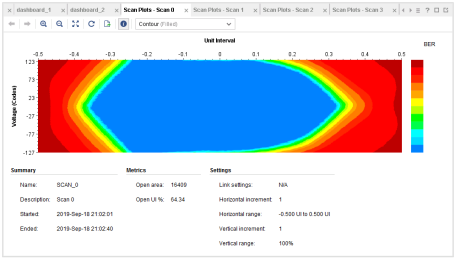


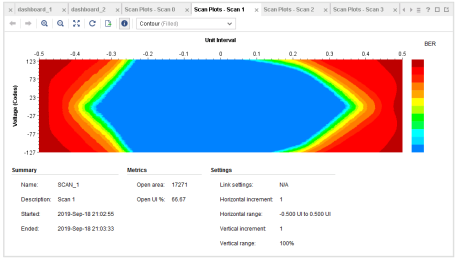


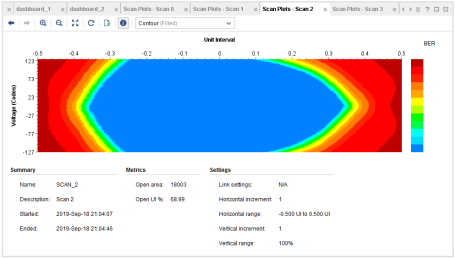


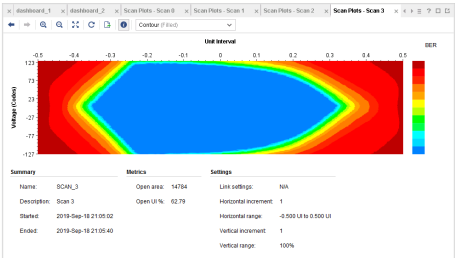
Measurements: IBERT

Schroff crate (7 slot) with SIS8300-KU at 5 GT/s (limited to 5 GT/s from slot configuration)

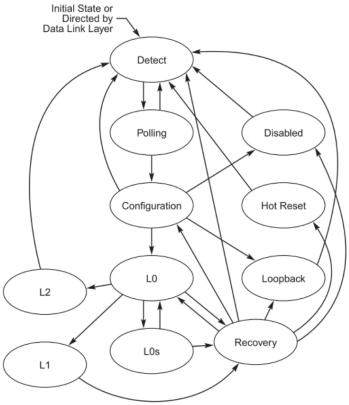








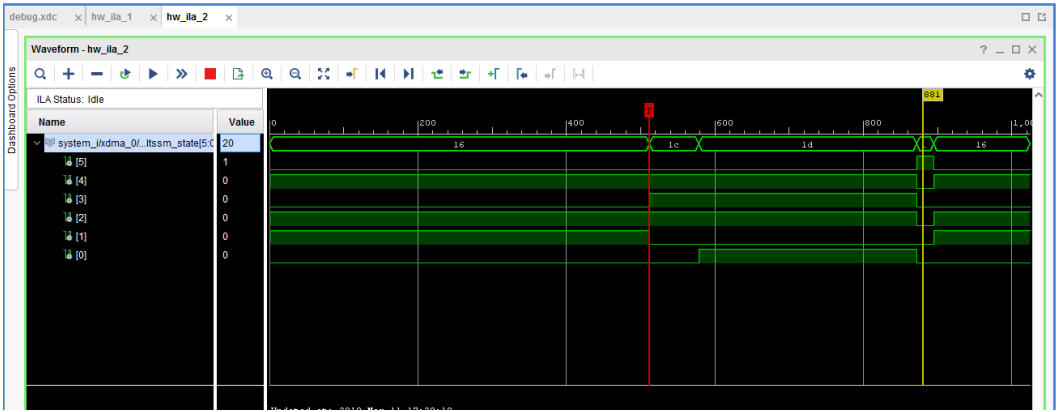
Link Training and State State Machine is a state machine running on both devices on the link. At the start up it negotiates link speed and link width, and removes the skew between lanes.



Very useful PCIe debug guide from Xilinx (recomends observing LTTSM state with Integrated Logic Analyzer - ILA):
`https://www.xilinx.com/Attachment/Xilinx_Answer_56616_7_Series_PCIe_Link_Training_Debug_Guide.pdf`

Here is an example of LTSSM state observed with ILA.

This example shows LTSSM leaving state L0 (the normal working state, encoded as 0x16) and cycling through states:
0x1C - Recovery Rcvrlock, 0x1D - Recovery Rcvrcfg, 0x20 - Recovery Idle.



PCI Express protocol Configuration

Tools

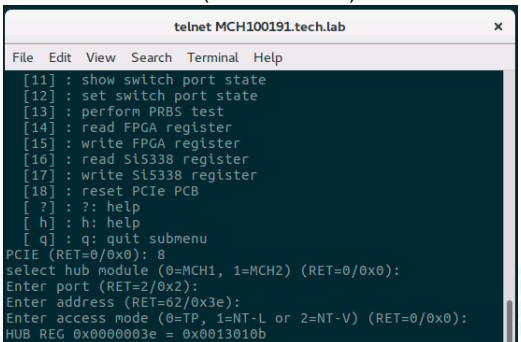
Two utilities from pciutils package (`https://github.com/pciutils/pciutils`) are useful to inspect and configure PCI Express system (on GNU/Linux):

- `lspci` - list PCI and PCIe devices, their organization and current status
- `setpci` - read and write individual registers in Configuration Space

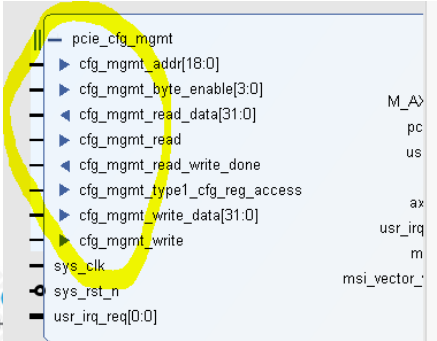
+

Tools to access the interface from the other (back) side:

N.A.T. MCH (PCIe switch) interface



Xilinx CFG interface

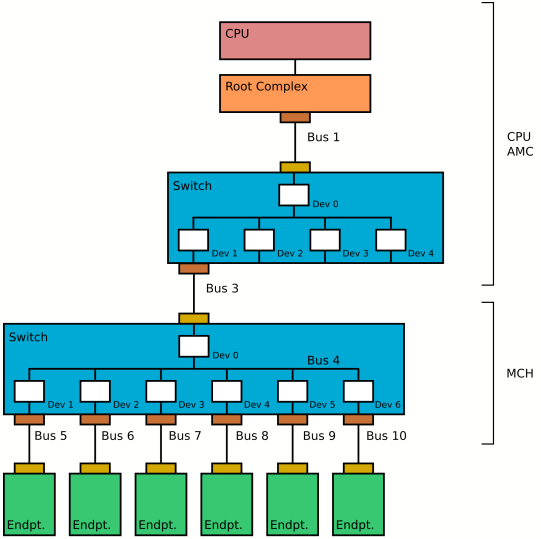


Example on Concurrent Tech AM G6x board:

```
$ lspci
00:00.0 Host bridge: Intel Corporation Xeon E3-1200 v6/7th Gen Core Processor Host Brid  /
00:01.0 PCI bridge: Intel Corporation Xeon E3-1200 v5/E3-1500 v5/6th Gen Core Processor
...
00:1f.3 Audio device: Intel Corporation CM238 HD Audio Controller (rev 31)
00:1f.4 SMBus: Intel Corporation 100 Series/C230 Series Chipset Family SMBus (rev 31)
01:00.0 PCI bridge: PLX Technology, Inc. Device 8725 (rev ca)
01:00.1 System peripheral: PLX Technology, Inc. Device 87d0 (rev ca)
...
02:01.0 PCI bridge: PLX Technology, Inc. Device 8725 (rev ca)
...
03:00.0 PCI bridge: PLX Technology, Inc. PEX 8748 48-Lane, 12-Port PCI Express Gen 3 (8    / )
04:01.0 PCI bridge: PLX Technology, Inc. PEX 8748 48-Lane, 12-Port PCI Express Gen 3 (8    / )
04:03.0 PCI bridge: PLX Technology, Inc. PEX 8748 48-Lane, 12-Port PCI Express Gen 3 (8    / )
...
05:00.0 Serial controller: Xilinx Corporation Device 7024
0d:00.0 Ethernet controller: Intel Corporation Ethernet Controller X710 for 10GbE SFP+ (    0
...
```

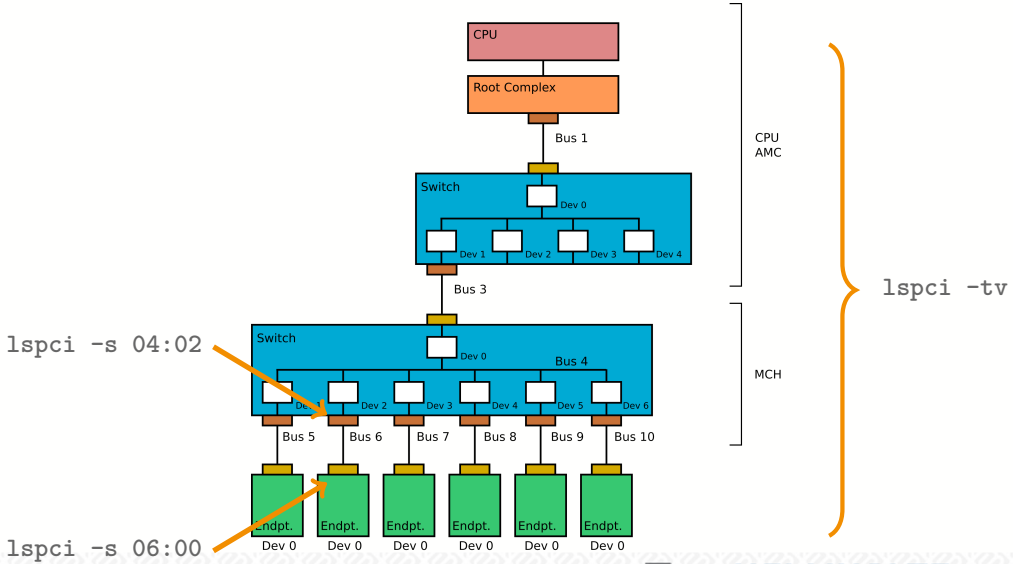
Various outputs from lspci

Several arguments can be passed to lspci to obtain different information from the system.



Various outputs from lspci

Several arguments can be passed to lspci to obtain different information from the system.



`lspci -tv` can be used to show the tree structure of the PCIe bus:

Example on Concurrent Tech AM G6x board:

```
$ lspci -tv
-[0000:00]-+-00.0  Intel Corporation Xeon E3-1200 v6/7th Gen Core Processor Host Bridge/DRAM Registers
           +-01.0-[01-0e]--+-00.0-[02-0e]--+-01.0-[03-0a]----00.0-[04-0a]--+-01.0-[05]--
           |               |                |                               +-03.0-[06]----00.0  Xilinx Corp [.] 7024
           |               |                |                               +-08.0-[07]--
           |               |                |                               +-0a.0-[08]--
           |               |                |                               +-11.0-[09]--
           |               |                |                               \-13.0-[0a]--
           |               |                +-02.0-[0b]--
           |               |                +-08.0-[0c]--
           |               |                +-09.0-[0d]--+-00.0  Intel Corporation Ethernet Controller X710 for 10GbE SFP+
           |               |                |            \-00.1  Intel Corporation Ethernet Controller X710 for 10GbE SFP+
           |               |                \-0a.0-[0e]--
           |               +-00.1  PLX Technology, Inc. Device 87d0
           |               +-00.2  PLX Technology, Inc. Device 87d0
           |               +-00.3  PLX Technology, Inc. Device 87d0
           |               \-00.4  PLX Technology, Inc. Device 87d0
           +-01.1-[0f]--
           +-02.0  Intel Corporation HD Graphics P630
           +-08.0  Intel Corporation Xeon E3-1200 v5/v6 / E3-1500 v5 / 6th/7th Gen Core Processor Gaussian Mixture Model
           +-14.0  Intel Corporation 100 Series/C230 Series Chipset Family USB 3.0 xHCI Controller
           +-14.2  Intel Corporation 100 Series/C230 Series Chipset Family Thermal Subsystem
           +-15.0  Intel Corporation 100 Series/C230 Series Chipset Family Serial IO I2C Controller #0
           +-15.1  Intel Corporation 100 Series/C230 Series Chipset Family Serial IO I2C Controller #1
           +-16.0  Intel Corporation 100 Series/C230 Series Chipset Family MEI Controller #1
           +-17.0  Intel Corporation Q170/Q150/B150/H170/H110/Z170/CM236 Chipset SATA Controller [AHCI Mode]
           +-1c.0-[10]----00.0  Intel Corporation I210 Gigabit Network Connection
           +-1c.1-[11]----00.0  Intel Corporation I210 Gigabit Backplane Connection
           +-1c.2-[12]----00.0  Intel Corporation I210 Gigabit Backplane Connection
           ...
```

lspci - slot

```
$ sudo lspci -s 04:03.0 -vv
04:03.0 PCI bridge: PLX Technology, Inc. PEX 8748 48-Lane, 12-Port PCI Express Gen 3 (8    / )
        Control: I/O+ Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr- Stepping- S    -
        [...]
        Bus: primary=04, secondary=06, subordinate=06, sec-latency=0
        [...]
        Capabilities: [68] Express (v2) Downstream Port (Slot+), MSI 00
                DevCap: MaxPayload 512 bytes, PhantFunc 0
                        ExtTag- RBE+
                DevCtl: Report errors: Correctable+ Non-Fatal+ Fatal+ Unsupported+
                        RlxdOrd+ ExtTag- PhantFunc- AuxPwr- NoSnoop+
                        MaxPayload 256 bytes, MaxReadReq 128 bytes
                DevSta: CorrErr+ UncorrErr- FatalErr- UnsuppReq+ AuxPwr- TransPend-
                LnkCap: Port #3, Speed 8GT/s, Width x4, ASPM L1, Exit Latency L0s <2us,      <4
                        ClockPM- Surprise+ LLActRep+ BwNot+ ASPMOptComp+
                LnkCtl: ASPM Disabled; Disabled- CommClk-
                        ExtSynch- ClockPM- AutWidDis- BWInt- AutBWInt-
                LnkSta: Speed 5GT/s, Width x4, TrErr- Train- SlotClk- DLActive+ BWMgmt-
                SltCap: AttnBtn+ PwrCtrl+ MRL+ AttnInd- PwrInd- HotPlug+ Surprise-
                        Slot #3, PowerLimit 25.000W; Interlock- NoCompl-
                SltCtl: Enable: AttnBtn+ PwrFlt- MRL- PresDet- CmdCplt+ HPIrq+ LinkChg+
                        Control: AttnInd Unknown, PwrInd Unknown, Power- Interlock-
                SltSta: Status: AttnBtn- PowerFlt- MRL- CmdCplt- PresDet+ Interlock-
                        Changed: MRL+ PresDet- LinkState-
                [...]
        [...]
        Kernel driver in use: pcieport
        Kernel modules: shpchp
```

lspci - endpoint

```
$ sudo lspci -s 06:00 -vv
06:00.0 Serial controller: Xilinx Corporation Device 7024 (prog-if 01 [16450])
        Subsystem: Xilinx Corporation Device 0007
        Physical Slot: 3
        Control: I/O- Mem- BusMaster- SpecCycle- MemWINV- VGASnoop- ParErr- Stepping- S    -
        Status: Cap+ 66MHz- UDF- FastB2B- ParErr- DEVSEL=fast >TAbort- <TAbort- <MAbort- >
        Interrupt: pin A routed to IRQ 16
        Region 0: Memory at 92800000 (32-bit, non-prefetchable) [size=8M]
        Region 1: Memory at 93000000 (32-bit, non-prefetchable) [size=64K]
        [...]
        Capabilities: [60] Express (v2) Endpoint, MSI 00
                DevCap: MaxPayload 512 bytes, PhantFunc 0, Latency L0s <64ns, L1 unlimi
                        ExtTag+ AttnBtn- AttnInd- PwrInd- RBE+ FLReset-
                DevCtl: Report errors: Correctable- Non-Fatal- Fatal- Unsupported-
                        RlxdOrd+ ExtTag+ PhantFunc- AuxPwr- NoSnoop+
                        MaxPayload 256 bytes, MaxReadReq 512 bytes
                DevSta: CorrErr- UncorrErr- FatalErr- UnsuppReq- AuxPwr- TransPend-
                LnkCap: Port #0, Speed 5GT/s, Width x4, ASPM L0s, Exit Latency L0s unli
                        ClockPM- Surprise- LLActRep- BwNot- ASPMOptComp-
                LnkCtl: ASPM Disabled; RCB 64 bytes Disabled- CommClk-
                        ExtSynch- ClockPM- AutWidDis- BWInt- AutBWInt-
                LnkSta: Speed 5GT/s, Width x4, TrErr- Train- SlotClk+ DLActive- BWMgmt-
                DevCap2: Completion Timeout: Range B, TimeoutDis-, LTR-, OBFF Not Suppo
                DevCtl2: Completion Timeout: 50us to 50ms, TimeoutDis-, LTR-, OBFF Disa
                LnkCtl2: Target Link Speed: 5GT/s, EnterCompliance- SpeedDis-
                         [...]
        Kernel driver in use: xdma
        Kernel modules: xdma
```

Using `setpci` to enter the compliance mode

Another system: PCIe up-link (Fujitsu server) and Struck SIS8300-KU AMC

```
$ lspci -tv
[...]
 +-[0000:85]-+-00.0-[86-8f]--+-00.0-[87-8f]--+-08.0-[88]--
 |           |                |               \-09.0-[89-8f]----00.0-[8a-8f]--+-00.0-[8b]--
 |           |                |                                               +-02.0-[8c]----00.0  Xilinx 8021
 |           |                |                                               +-04.0-[8d]--
 |           |                |                                               +-06.0-[8e]--
 |           |                |                                               \-08.0-[8f]--
 |           |                +-00.1  PLX Technology, Inc. Device 87d0
 |           |                +-00.2  PLX Technology, Inc. Device 87d0
 |           |                +-00.3  PLX Technology, Inc. Device 87d0
 |           |                \-00.4  PLX Technology, Inc. Device 87d0
[...]
```

Relevant registers

To enter the compliance mode, we must set Enter Compliance bit and then perform the Hot Reset.

PCI EXPRESS BASE SPECIFICATION, REV. 3.0

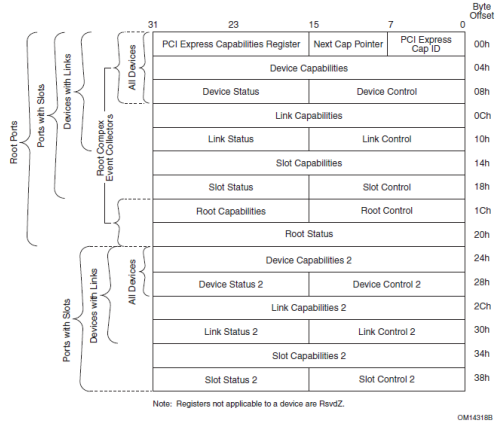


Figure 7-10: PCI Express Capability Structure

7.8.19. Link Control 2 Register (Offset 30h)

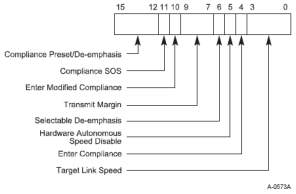


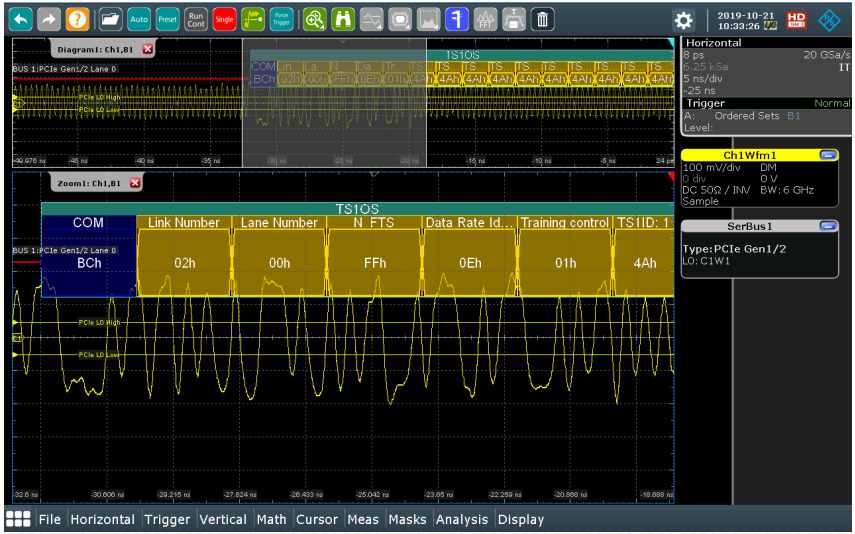
Figure 7-28: Link Control 2 Register

Commands to enter the compliance mode:

```
$ sudo setpci -s 8a:02 CAP_EXP+0x30.W
0003
$ sudo setpci -s 8c:00 CAP_EXP+0x30.W
0002
$ sudo setpci -s 8a:02 BRIDGE_CONTROL.W
0013

$ sudo setpci -s 8a:02 CAP_EXP+0x30.W=0x13
$ sudo setpci -s 8c:00 CAP_EXP+0x30.W=0x0012
$ sudo setpci -s 8a:02 BRIDGE_CONTROL.W=0x53
```

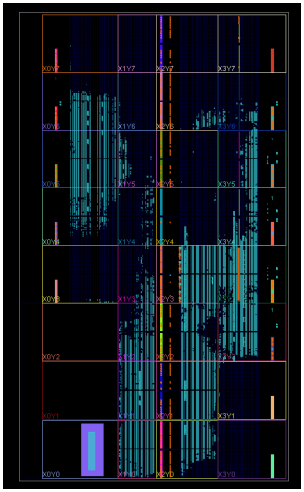
Compliance mode

We are now in compliance mode (both link partners are transmitting TS1 Ordered Sets):

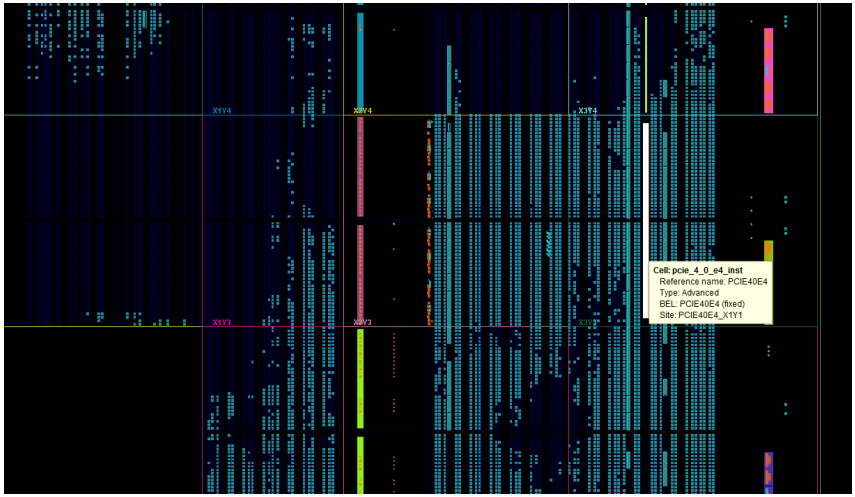


FPGA implementation

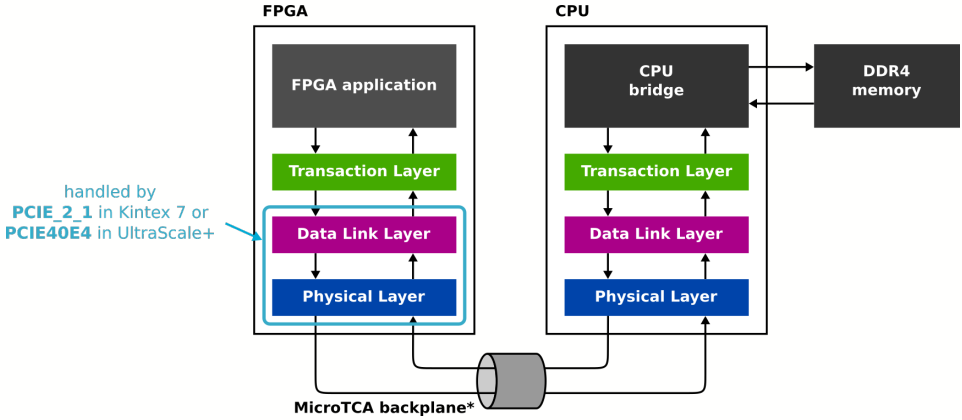
Modern FPGAs from Xilinx and Intel include hard IP cores to handle lower layers of protocol.



Implementation of PCIe in FPGA

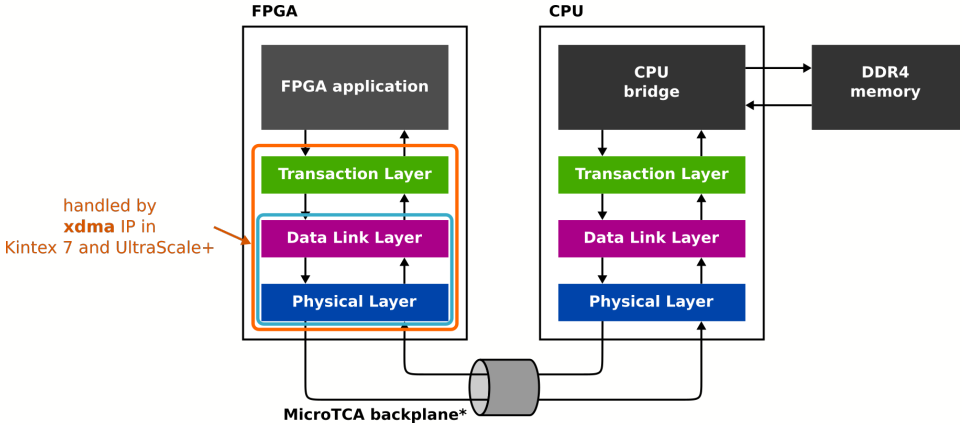


Implementation of PCIe in FPGA

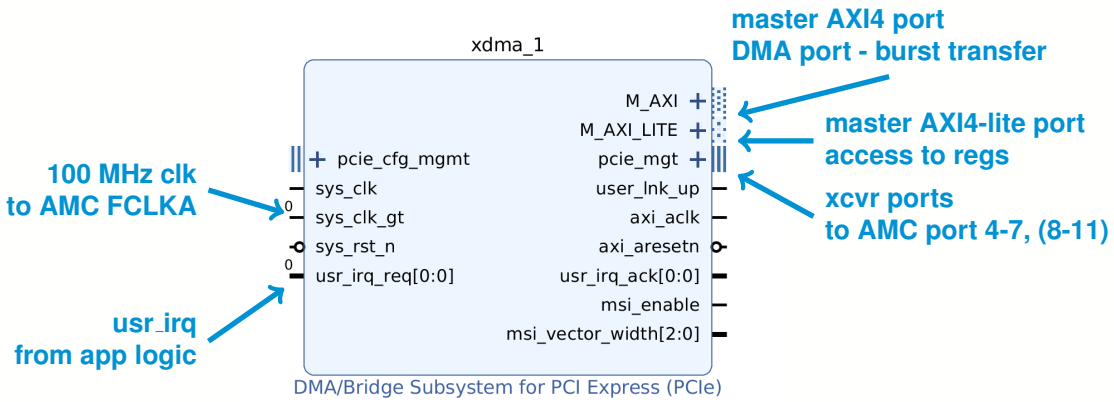


Implementation of PCIe in FPGA

On top of that, both Xilinx and Intel provide IP cores to also handle the Transaction Layer protocol



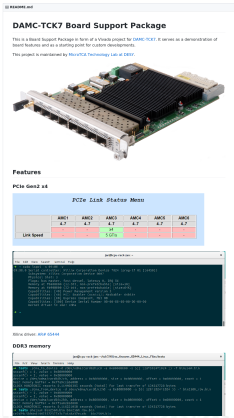
This is how the Xilinx DMA Subsystem for PCI Express looks in Vivado:

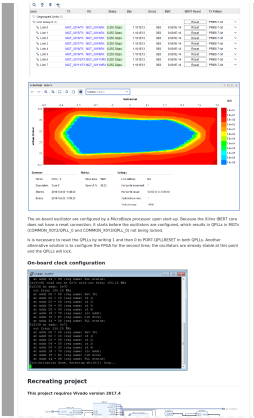


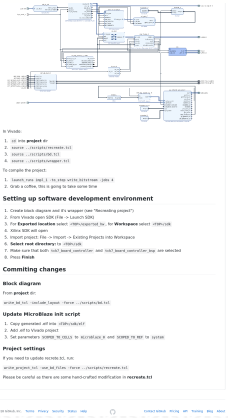
Reference designs for our boards

DAMC-TCK7: Vivado project including support for PCIe, DDR3, clock configuration, IBERT on SFP+ and UDP/IPv4 beacon on AMC port 0; available under permissive license (3-clause BSD)

Available on our GitHub page: `https://github.com/MicroTCA-Tech-Lab`



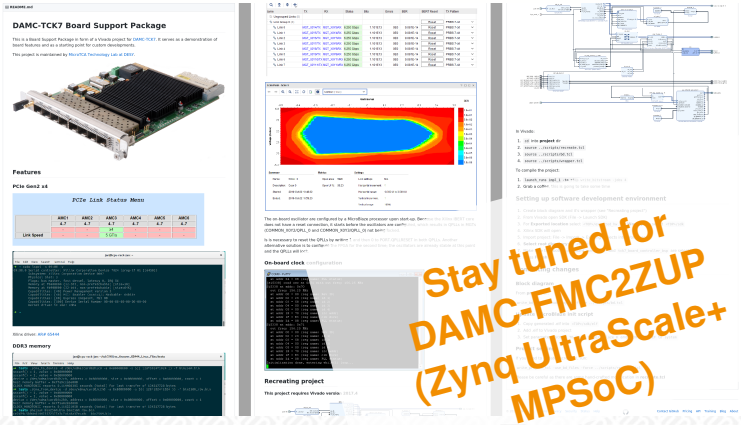




Reference designs for our boards

DAMC-TCK7: Vivado project including support for PCIe, DDR3, clock configuration, IBERT on SFP+ and UDP/IPv4 beacon on AMC port 0; available under permissive license (3-clause BSD)

Available on our GitHub page: `https://github.com/MicroTCA-Tech-Lab`



Driver for Xilinx DMA subsystem

- Xilinx provides a Linux driver for their DMA subsystem for PCIe
- Availabe on GitHub:

  `https://github.com/Xilinx/dma_ip_drivers/tree/master/XDMA/`
- Previously available in Xilinx Answer Record #65444:

  `https://www.xilinx.com/support/answers/65444.html`
- MicroTCA Tech Lab mantains an internal fork with some improvements → discussion with our legal department on licensing ongoing

Interface to user space

On the other side (towards the user space), Xilinx DMA driver provides several char devices:

```
$ ll /dev | grep xdma
drwxr-xr-x  3 root root           60 Jun 20 17:19 xdma
crw-rw-rw-  1 root root     238,  36 Jun 20 17:19 xdma0_c2h_0
crw-rw-rw-  1 root root     238,   1 Jun 20 17:19 xdma0_control
crw-rw-rw-  1 root root     238,  10 Jun 20 17:19 xdma0_events_0
crw-rw-rw-  1 root root     238,  11 Jun 20 17:19 xdma0_events_1
[...]
crw-rw-rw-  1 root root     238,  24 Jun 20 17:19 xdma0_events_14
crw-rw-rw-  1 root root     238,  25 Jun 20 17:19 xdma0_events_15
crw-rw-rw-  1 root root     238,  32 Jun 20 17:19 xdma0_h2c_0
crw-rw-rw-  1 root root     238,   0 Jun 20 17:19 xdma0_user
crw-rw-rw-  1 root root     238,   2 Jun 20 17:19 xdma0_xvc
```

Interface to user space

On the other side (towards the user space), Xilinx DMA driver provides several char devices:

```
$ ll /dev | grep xdma
drwxr-xr-x  3 root root          60 Jun 20 17:19 xdma
crw-rw-rw-  1 root root     238,  36 Jun 20 17:19 xdma0_c2h_0
crw-rw-rw-  1 root root     238,   1 Jun 20 17:19 xdma0_control
crw-rw-rw-  1 root root     238,  10 Jun 20 17:19 xdma0_events_0
crw-rw-rw-  1 root root     238,  11 Jun 20 17:19 xdma0_events_1
[...]
crw-rw-rw-  1 root root     238,  24 Jun 20 17:19 xdma0_events_14
crw-rw-rw-  1 root root     238,  25 Jun 20 17:19 xdma0_events_15
crw-rw-rw-  1 root root     238,  32 Jun 20 17:19 xdma0_h2c_0
crw-rw-rw-  1 root root     238,   0 Jun 20 17:19 xdma0_user
crw-rw-rw-  1 root root     238,   2 Jun 20 17:19 xdma0_xvc
```

c2h
AXI4 port (DMA from device)

h2c
AXI4 port (DMA to device)

user
AXI4-Lite port

events_N
interrupts

Examples

Example 1: CPU freezes on `memcpy()`

Using GPIO to trigger on software events

```
(gdb) x/i$pc
=> 0xffffffff717f820 <__memcpy_sse2_unaligned>:    mov    %rsi,%rax
   0xffffffff717f823 <__memcpy_sse2_unaligned+3>:  lea    (%rdx,%rdx,1),%rcx
   0xffffffff717f827 <__memcpy_sse2_unaligned+7>:  sub    %rdi,%rax
   0xffffffff717f82a <__memcpy_sse2_unaligned+10>: sub    %rdx,%rax
   0xffffffff717f82d <__memcpy_sse2_unaligned+13>: cmp    %rcx,%rax
(gdb) x/10i $pc
=> 0xffffffff717f820 <__memcpy_sse2_unaligned>:    mov    %rsi,%rax
   0xffffffff717f823 <__memcpy_sse2_unaligned+3>:  lea    (%rdx,%rdx,1),%rcx
   0xffffffff717f827 <__memcpy_sse2_unaligned+7>:  sub    %rdi,%rax
   0xffffffff717f82a <__memcpy_sse2_unaligned+10>: sub    %rdx,%rax
   0xffffffff717f82d <__memcpy_sse2_unaligned+13>: cmp    %rcx,%rax
   0xffffffff717f830 <__memcpy_sse2_unaligned+16>: jb     0xffffffff717f93d <__memcpy_sse2_unaligned+285>
   0xffffffff717f836 <__memcpy_sse2_unaligned+22>: cmp    $0x10,%rdx
   0xffffffff717f83a <__memcpy_sse2_unaligned+26>: jbe    0xffffffff717f9cb <__memcpy_sse2_unaligned+427>
   0xffffffff717f840 <__memcpy_sse2_unaligned+32>: movdqu (%rsi),%xmm8
   0xffffffff717f845 <__memcpy_sse2_unaligned+37>: cmp    $0x20,%rdx
```

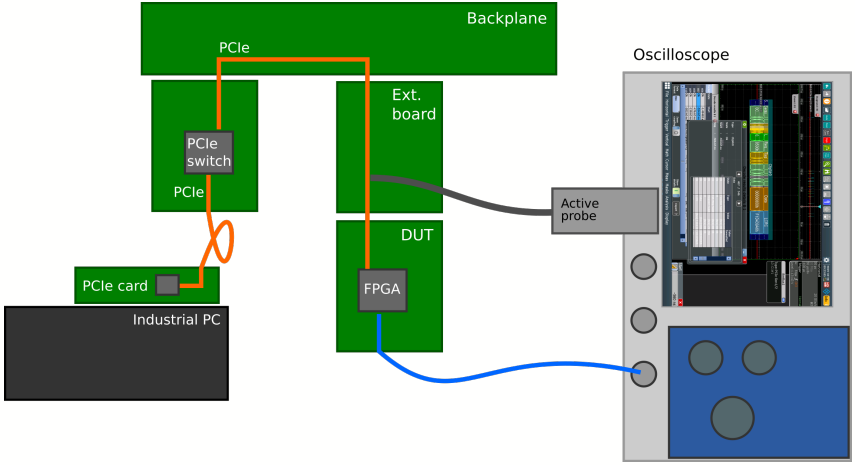
Example 2: CPU crashes on DMA (after some time)

Using FPGA to help with the trigger

```
[ 1555.337708] pcieport 0000:89:00.0: AER:    [15] HeaderOF
[ 1555.365082] pcieport 0000:89:00.0: AER: aer_layer=Physical Layer, aer_agent=Receiver ID
[ 1923.983565] {3}[Hardware Error]: Hardware error from APEI Generic Hardware Error Source: 0
[ 1923.983568] {3}[Hardware Error]: It has been corrected by h/w and requires no further action
[ 1923.983570] {3}[Hardware Error]: event severity: corrected
[ 1923.983572] {3}[Hardware Error]:  Error 0, type: corrected
[ 1923.983574] {3}[Hardware Error]:   section_type: PCIe error
[ 1923.983576] {3}[Hardware Error]:   port_type: 5, upstream switch port
[ 1923.983577] {3}[Hardware Error]:   version: 3.0
[ 1923.983579] {3}[Hardware Error]:   command: 0x0147, status: 0x0010
[ 1923.983581] {3}[Hardware Error]:   device_id: 0000:89:00.0
[ 1923.983582] {3}[Hardware Error]:   slot: 0
[ 1923.983584] {3}[Hardware Error]:   secondary_bus: 0x8a
[ 1923.983585] {3}[Hardware Error]:   vendor_id: 0x10b5, device_id: 0x8780
[ 1923.983587] {3}[Hardware Error]:   class_code: 000406
[ 1923.983588] {3}[Hardware Error]:   bridge: secondary_status: 0x0000, control: 0x0013
[ 1923.983823] pcieport 0000:89:00.0: AER: aer_status: 0x00004001, aer_mask: 0x0000200D
[ 1923.992489] pcieport 0000:89:00.0: AER:    [ 0] RxErr
[ 1923.999792] pcieport 0000:89:00.0: AER:    [15] HeaderOF
[ 1924.007086] pcieport 0000:89:00.0: AER: aer_layer=Physical Layer, aer_agent=Receiver ID
[ 1985.423861] {4}[Hardware Error]: Hardware error from APEI Generic Hardware Error Source: 0
[ 1985.423864] {4}[Hardware Error]: It has been corrected by h/w and requires no further action
```

Test setup

In both examples a similar setup is used: active probe monitors PCIe differential pair, and another signal is connected to LVDS or LVCMOS output for trigger.



Example 1: basic overview

Introduction:

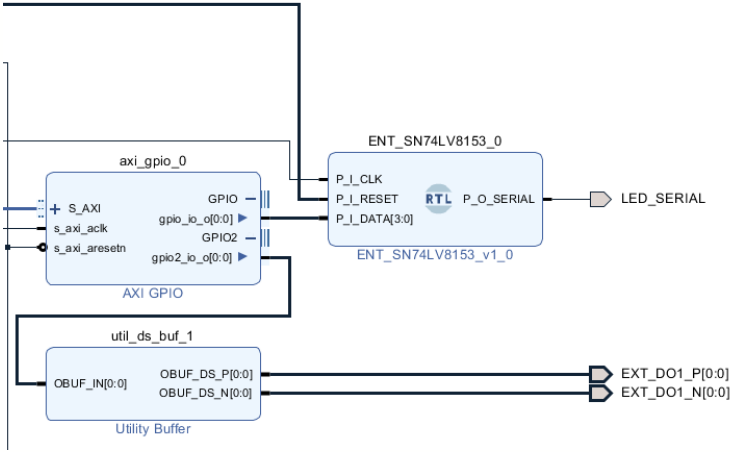
- To copy large amounts of data one usually uses DMA
- But sometimes in our applications we have to copy just a couple of registers (e.g. 32 bytes in total) $\rightarrow$ setting up DMA makes no sense
- In this case it would convenient (from SW point of view) to use `memcpy()`

Issue:

- Performing `memcpy()` on more than 4 bytes freezes the CPU, reboot is required to recover (luckily we have IPMI)
- Reading registers one-by-one (4 bytes at the time) works
- `memcpy()` also works on Kintex UltraScale, but does not work on Kintex 7-series (with the same IP - xdma)

Example 1: triggering

- ...
- The most convenient way to trigger on such event is to set GPIO pin high before starting the relevant SW test case



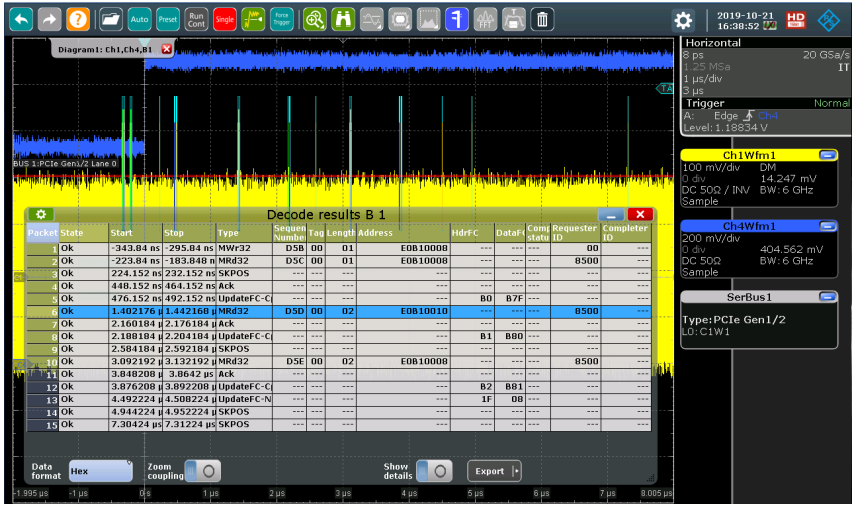
Example 1: code to recreate the issue

We first set GPIO pin high to trigger the oscilloscope, then perform a 32-bit read (for check) and then perform `memcpy()`.

```
// virt_addr from mmap(), points to a memory in FPGA

gpio_control(GPIO0, 1);

uint32_t tmp = *((uint32_t*)virt_addr);

char buf[16];
memcpy(buf, (void*)virt_addr, sizeof(buf));
```

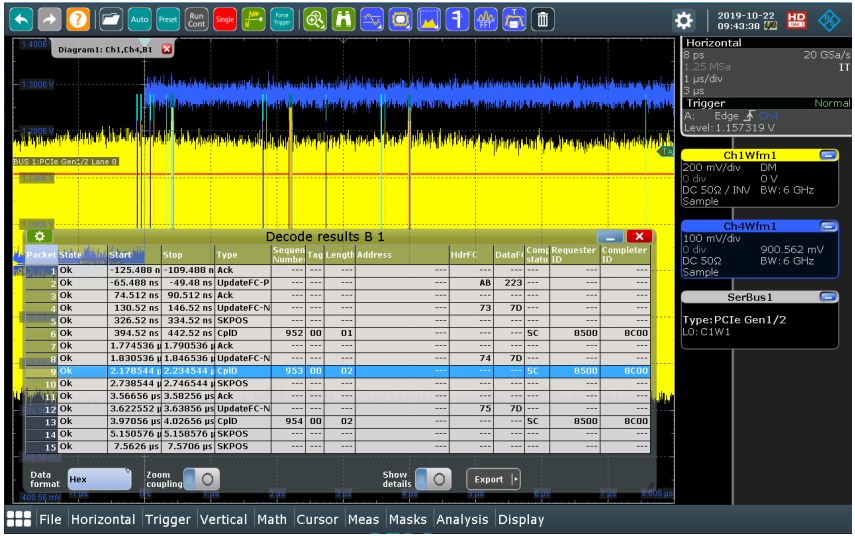
Example 1: measurement

TX side on SIS8300-KU (with Kintex UltraScale)



Example 1: measurement

RX side on SIS8300-KU (with Kintex UltraScale)

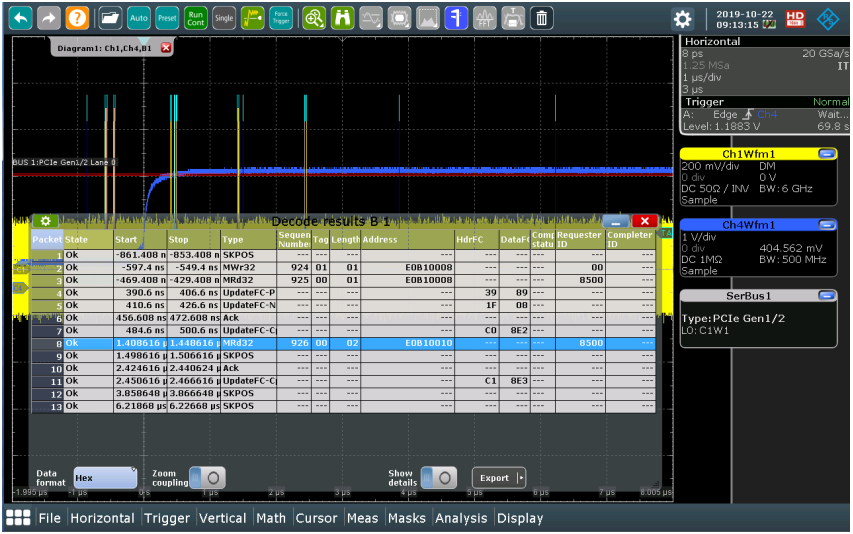


Example 1: measurement

RX side on SIS8300-KU (with Kintex UltraScale)

Example 1: measurement

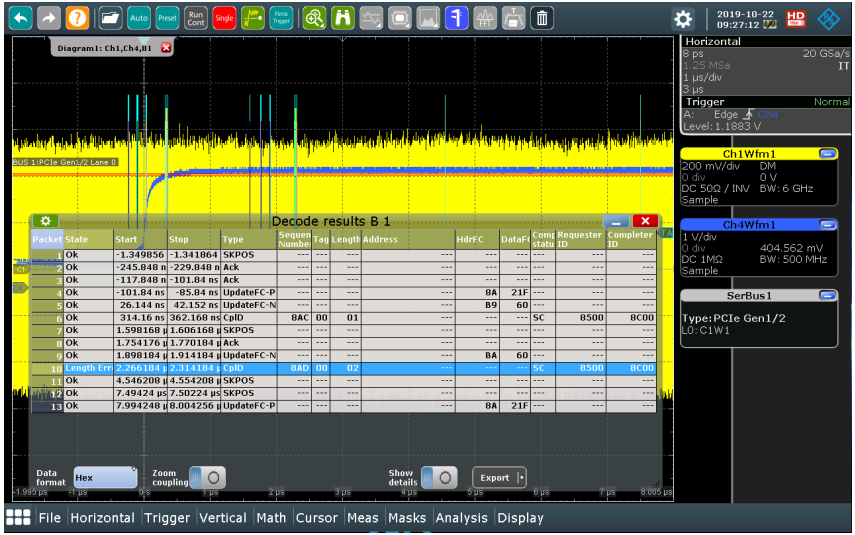
TX side on DAMC-TCK7 (with Kintex 7)



Packet	State	Start	Stop	Type	Sequen Numbe	Tag	Length	Address	HdrFC	DataFC	Comp Status	Requester ID	Completer ID
1	Ok	-861.408 n	-853.408 n	SKPOS	---	---	---	---	---	---	---	---	---
2	Ok	-597.4 ns	-549.4 ns	MWr32	924	01	01	E0B10008	---	---	---	00	---
3	Ok	-469.408 n	-429.408 n	MRd32	925	00	01	E0B10008	---	---	---	8500	---
4	Ok	390.6 ns	406.6 ns	UpdateFC-P	---	---	---	---	39	89	---	---	---
5	Ok	410.6 ns	426.6 ns	UpdateFC-N	---	---	---	---	1F	08	---	---	---
6	Ok	456.608 ns	472.608 ns	Ack	---	---	---	---	---	---	---	---	---
7	Ok	484.6 ns	500.6 ns	UpdateFC-C	---	---	---	---	C0	8E2	---	---	---
8	Ok	1.408616 µ	1.448616 µ	MRd32	926	00	02	E0B10010	---	---	---	8500	---
9	Ok	1.498616 µ	1.506616 µ	SKPOS	---	---	---	---	---	---	---	---	---
10	Ok	2.424616 µ	2.440624 µ	Ack	---	---	---	---	---	---	---	---	---
11	Ok	2.450616 µ	2.466616 µ	UpdateFC-C	---	---	---	---	C1	8E3	---	---	---
12	Ok	3.858648 µ	3.866648 µ	SKPOS	---	---	---	---	---	---	---	---	---
13	Ok	6.21868 µs	6.22668 µs	SKPOS	---	---	---	---	---	---	---	---	---

Example 1: measurement

RX side on DAMC-TCK7 (with Kintex 7)



Example 1: measurement

RX side on DAMC-TCK7 (with Kintex 7) - length error

Example 1: conclusion

- Triggering with GPIO pin is a convenient way to ”isolate” relevant events
- Difference between 7-series and UltraScale xdma
- Work-around for 7-series xdma: `mempcy()` based on 32-bit reads

Example 2: basic overview

It was observed that after some time (usually less than an hour) the DMA stops working - every time we attempt to perform a DMA transfer the CPU crashes. It was soon notice that

everything works OK until first error report appears in `dmesg`. This error report itself is nothing unusual - HW handled the error on its own.

```
[ 1555.357788] pcieport 0000:89:00.0: AER:    [15] HeaderOF
[ 1555.365082] pcieport 0000:89:00.0: AER: aer_layer=Physical Layer, aer_agent=Receiver ID
[ 1923.983565] {3}[Hardware Error]: Hardware error from APEI Generic Hardware Error Source: 0
[ 1923.983568] {3}[Hardware Error]: It has been corrected by h/w and requires no further action
[ 1923.983570] {3}[Hardware Error]: event severity: corrected
[ 1923.983572] {3}[Hardware Error]:  Error 0, type: corrected
[ 1923.983574] {3}[Hardware Error]:   section_type: PCIe error
[ 1923.983576] {3}[Hardware Error]:   port_type: 5, upstream switch port
[ 1923.983577] {3}[Hardware Error]:   version: 3.0
[ 1923.983579] {3}[Hardware Error]:   command: 0x0147, status: 0x0010
[ 1923.983581] {3}[Hardware Error]:   device_id: 0000:89:00.0
[ 1923.983582] {3}[Hardware Error]:   slot: 0
[ 1923.983584] {3}[Hardware Error]:   secondary_bus: 0x8a
[ 1923.983585] {3}[Hardware Error]:   vendor_id: 0x10b5, device_id: 0x8780
[ 1923.983587] {3}[Hardware Error]:   class_code: 000406
[ 1923.983588] {3}[Hardware Error]:   bridge: secondary_status: 0x0000, control: 0x0013
[ 1923.983823] pcieport 0000:89:00.0: AER: aer_status: 0x0000a001, aer_mask: 0x00002000
[ 1923.992489] pcieport 0000:89:00.0: AER:    [ 0] RxErr
[ 1923.999792] pcieport 0000:89:00.0: AER:    [15] HeaderOF
[ 1924.007086] pcieport 0000:89:00.0: AER: aer_layer=Physical Layer, aer_agent=Receiver ID
[ 1985.423861] {4}[Hardware Error]: Hardware error from APEI Generic Hardware Error Source: 0
[ 1985.423864] {4}[Hardware Error]: It has been corrected by h/w and requires no further action
```

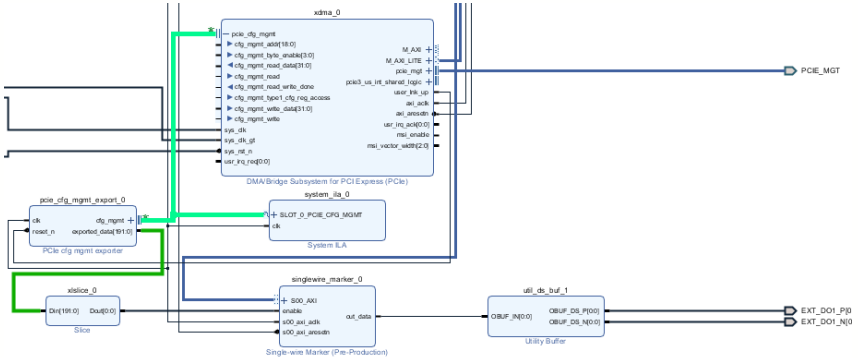
Example 2: initial observations

We noticed that MSI vector gets corrupted (we performed `diff` on `lspci` output before and after first ”Hardware Error” message.

```
$ sudo lspci -s 8c:00 -vv
8c:00.0 Serial controller: Xilinx Corporation Device 8021 (prog-if 01 [16450])
        Subsystem: Xilinx Corporation Device 0007
        [...]
        NUMA node: 1
        Region 0: Memory at e0b00000 (32-bit, non-prefetchable) [size=1M]
        Region 1: Memory at e0c00000 (32-bit, non-prefetchable) [size=64K]
        Capabilities: [80] Power Management version 3
                Flags: PMEClk- DSI- D1- D2- AuxCurrent=0mA PME(D0-,D1-,D2-,D3hot-,D3col -)
                Status: D0 NoSoftRst+ PME-Enable- DSel=0 DScale=0 PME-
        Capabilities: [90] MSI: Enable+ Count=1/1 Maskable- 64bit+
                Address: 00080000fee00198  Data: 0000
        [...]
        Capabilities: [100 v2] Advanced Error Reporting
                UESta:  DLP- SDES- TLP- FCP- CmpltTO- CmpltAbrt- UnxCmplt- RxOF- MalfTL -
                UEMsk:  DLP- SDES- TLP- FCP- CmpltTO- CmpltAbrt- UnxCmplt- RxOF- MalfTL -
                UESvrt: DLP+ SDES+ TLP- FCP+ CmpltTO- CmpltAbrt- UnxCmplt- RxOF+ MalfTL +
                CESta:  RxErr- BadTLP- BadDLLP- Rollover- Timeout- AdvNonFatalErr-
                CEMsk:  RxErr- BadTLP- BadDLLP- Rollover- Timeout- AdvNonFatalErr-
                AERCap: First Error Pointer: 00, ECRCGenCap- ECRCGenEn- ECRCChkCap- ECR
                        MultHdrRecCap- MultHdrRecEn- TLPPfxPres- HdrLogCap-
                HeaderLog: 04000001 85000003 8c010000 00000000
        Kernel driver in use: xdma
```

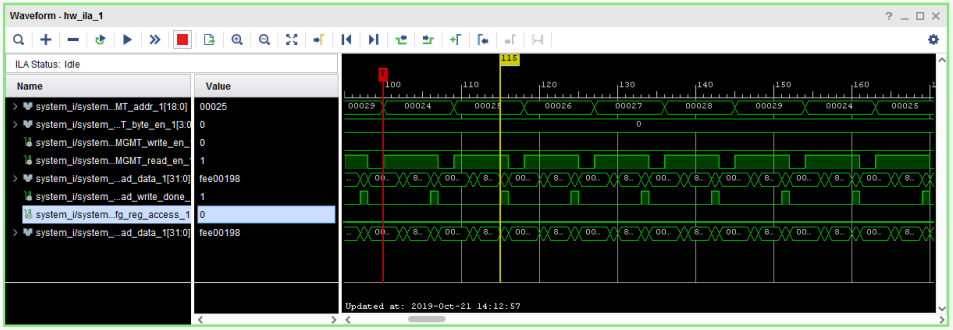
Example 2: debug setup

We prepared a test setup to trigger the scope on this event - an IP (PCIe config mgmt exporter) continuously copies the values from the PCIe endpoint configuration area to a vector. This vector is then used to drive LVDS output on the front panel.



Example 2: debug setup - export

Here we observe the connection between PCIe config mgmt exporter and PCIe IP core - we see that exporter is cyclically reading relevant addresses.



Example 2: capture overview

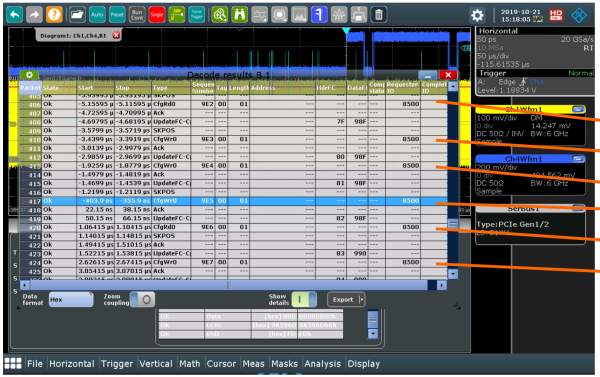
We arm the trigger and wait. After some time, the oscilloscope triggers:

Example 2: the packet

This is the packet which corrupts MSI vector! It means that the source of this issue is on the software side - probably in Linux driver.

Example 2: analysis

We can analyze the configuration packets and compare them with software - this allows us pin-pointing the source of the erroneous packet.




```
rd @ 0x0      [-15.8 us]
rd @ 0x4      [-14.3 us]
rd @ 0x4      [-12.7 us]
rd @ 0x34     [-11.2 us]
rd @ 0x80     [- 9.7 us]
rd @ 0x90     [- 8.2 us]
rd @ 0xC0     [- 6.6 us]
rd @ 0x100    [- 5.1 us]
wr @ 0x104    [- 3.5 us]
wr @ 0x110    [- 1.9 us]
wr @ 0x98     [- 0.4 us]
rd @ 0x108    [  1.1 us]
wr @ 0x108    [  2.6 us]
rd @ 0x114    [  4.1 us]
wr @ 0x114    [  5.6 us]
rd @ 0xC      [  7.2 us]
rd @ 0x104    [  8.8 us]
rd @ 0x104    [ 10.4 us]
```

Example 2: conclusion

- Using a clever triggering method we were able to observe the relevant packets
- Protocol decode enables us to understand software better/easier
- Long buffers in oscilloscope (10 M samples) allows capturing large SW routines
- This debug is on-going, but we are very close to resolving the problem

Summary

Additional resources

- L. Petrosyan, ”MicroTCA and PCIe HotSwap under Linux”,
  `http://webcast.desy.de/?m=201312&lang=en`,
  at MTCA Workshop for Industry and Research 2013, Hamburg
- J. Marjanovic, ”Direct Memory Access with FPGA”,
  `https://github.com/MicroTCA-Tech-Lab/damc-tck7-fpga-bsp/blob/demo-mtcaws2018/docs/demo/mtcaws2018_marjanovic_dma.pdf`
  at MTCA Workshop for Industry and Research 2018, Hamburg
  (similar to this one, but more hands-on - Vivado project and SW available)
- J. Corbet, A. Rubini, G. Kroah-Hartman, ”Linux Device Drivers, 3rd Edition”,
  (quite old at this point, 4th Edition long over-due)
- M. Jackson, R. Budruk, ”PCI Express Technology 3.0”

Summary

- PCI Express is a protocol of choice for many institutes and vendors
- Provides low latency and high throughput
- FPGA vendors provide a lot of IP, drivers, ...
- MicroTCA Tech Lab provides BSP with PCIe for some of the boards, more BSPs (e.g. for Zynq UltraScale MPSoc) to follow
- There are a lot of tools to troubleshoot PCIe-based systems

TRANSFER MTCA TO RESEARCH AND INDUSTRY

- Custom developments
- High-end test & measurement services
- System configuration & integration
- LLRF design

Marketing.
Services & Support.
Tech-Shop.

Thank you
Vielen Dank

https://techlab.desy.de

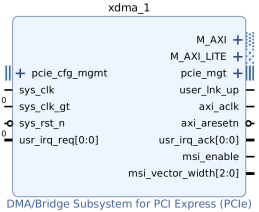
Deutsches Elektronen-Synchrotron DESY
A Research Centre of the Helmholtz Association
Notkestr. 85, 22607 Hamburg, Germany

Backup slides

FPGA side

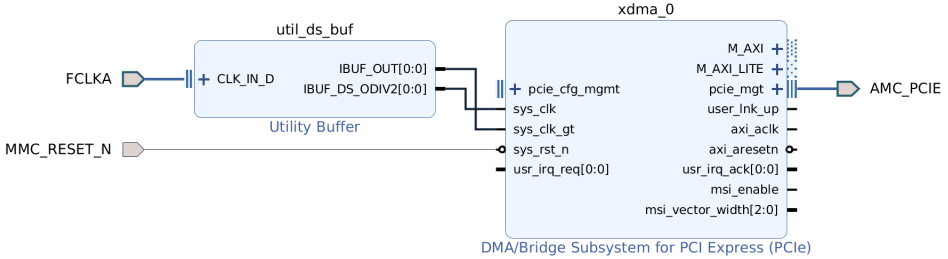
Let's build a system, step by step

1. We open Vivado, start a new project, create a Block Diagram and place an xdma IP:

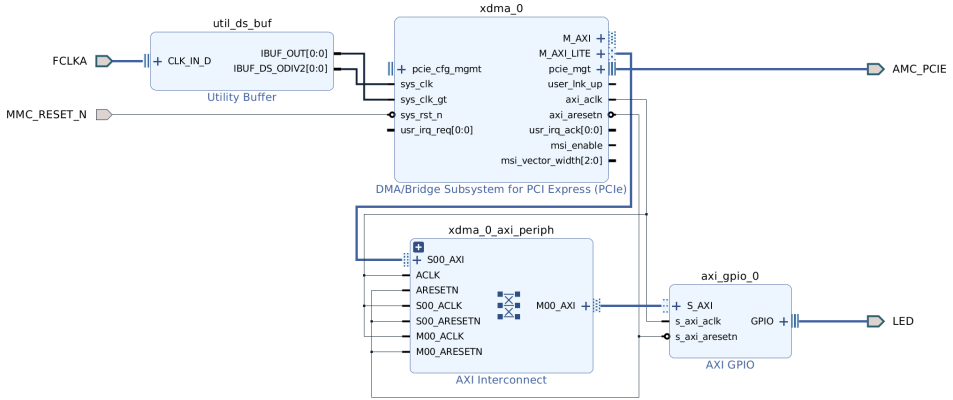


Let's build a system, step by step

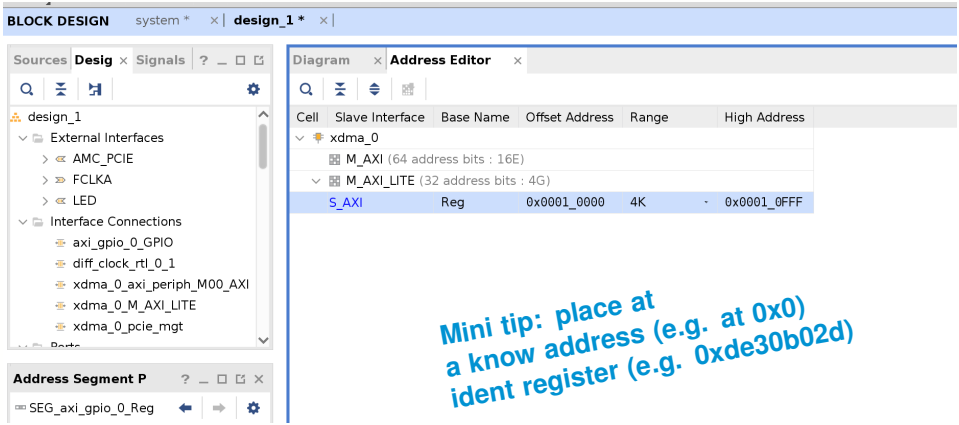
2. We then connect external connections:



3. We then proceed to add a GPIO, to blink the LEDs

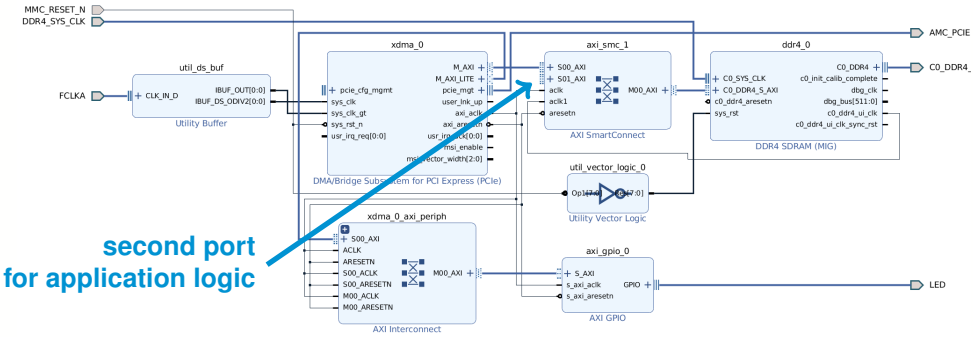


4. Let's not forget to assign the addresses



Let's build a system, step by step

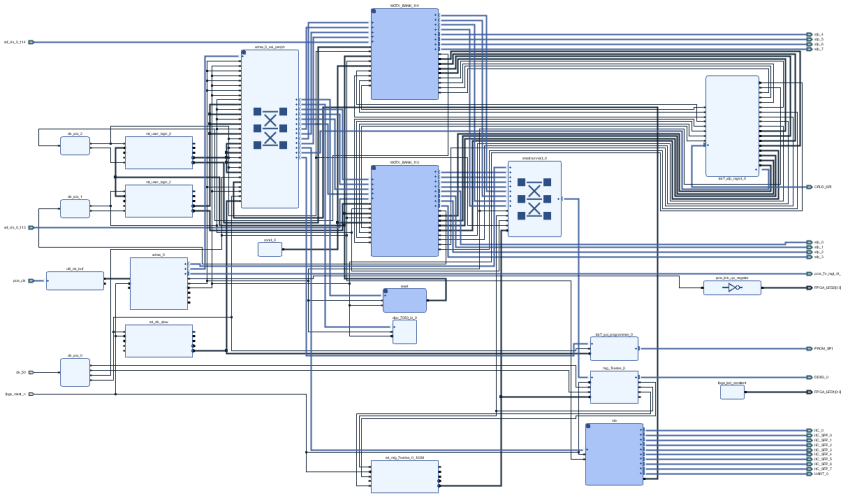
5. We add a DDR4 controller (not a trivial step, a lot of configuration) and connect the DMA port of PCIe to DDR4 controller:



6. some magic (aka months of work on application logic)

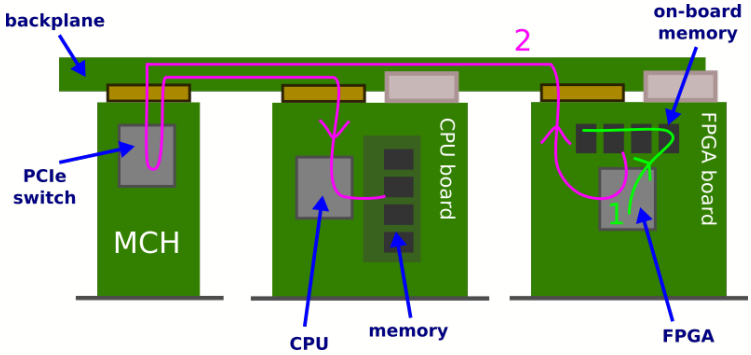
Typical PCIe-based system in FPGA

7. This is how a typical PCIe-based system looks



Summary of the FPGA part

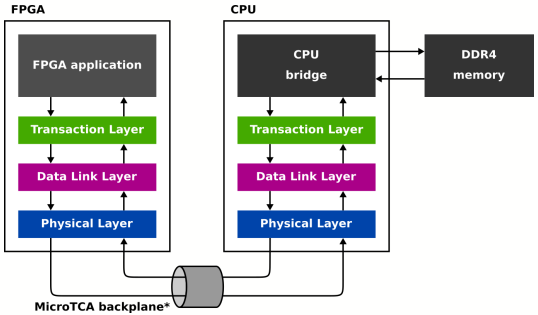
- Modern FPGAs provide a lot of hard IP
- Typical flow involves configuring IPs and connecting them together
- Custom logic required for application-specific part
- Two-step transfer: first to on-board memory, then to CPU



Software side

CPU board

From ”FPGA side” we have solved the left part, now it is time to look at the right part:



- Typically there is one (or more) CPU AMCs is MicroTCA systems
- Usually (in experimental physics) the CPU runs GNU/Linux
- DESY runs Ubuntu 16.04 LTS, Ubuntu 18.04 LTS is also in use
- PCIe hotplug was modernized in Linux kernel 4.19[1] → available in Ubuntu 20.04 LTS

[1] `https://lwn.net/Articles/767885/`

Linux pci subsystem

- Linux has a subsystem responsible for managing PCI/PCIe device driver
- Hot-swap module (pciehp) is also part of Linux kernel
- Device-specific driver register by providing pci_driver structure to pci_register_driver() function

Linux pci subsystem

https://elixir.bootlin.com/linux/v4.15/source/include/linux/pci.h#L744

```
struct pci_driver {
    struct list_head node;
    const char *name;
    const struct pci_device_id *id_table;   /* must be non-NULL for probe to be called */
    int  (*probe)  (struct pci_dev *dev, const struct pci_device_id *id);   /* New device
         inserted */
    void (*remove) (struct pci_dev *dev);   /* Device removed (NULL if not a hot-plug
         capable driver) */
    int  (*suspend) (struct pci_dev *dev, pm_message_t state);  /* Device suspended */
    int  (*suspend_late) (struct pci_dev *dev, pm_message_t state);
    int  (*resume_early) (struct pci_dev *dev);
    int  (*resume) (struct pci_dev *dev);                    /* Device woken up */
    void (*shutdown) (struct pci_dev *dev);
    int (*sriov_configure) (struct pci_dev *dev, int num_vfs); /* PF pdev */
    const struct pci_error_handlers *err_handler;
    const struct attribute_group **groups;
    struct device_driver    driver;
    struct pci_dynids dynids;
};
```

Xilinx driver

Xilinx driver only provides probe() and remove() functions - enough to perform hot-swap.

from https://github.com/Xilinx/dma_ip_drivers/blob/8b8c70b697f049649d5fa99be9c6bc4302d89ac9/XDMA/linux-kernel/xdma/xdma_mod.c#L316:

```
static struct pci_driver pci_driver = {
        .name = DRV_MODULE_NAME,
        .id_table = pci_ids,
        .probe = probe_one,
        .remove = remove_one,
        .err_handler = &xdma_err_handler,
};
```

Interface to user space

On the other side (towards the user space), Xilinx DMA driver provides several char devices:

```
$ ll /dev | grep xdma
drwxr-xr-x  3 root root           60 Jun 20 17:19 xdma
crw-rw-rw-  1 root root     238,  36 Jun 20 17:19 xdma0_c2h_0
crw-rw-rw-  1 root root     238,   1 Jun 20 17:19 xdma0_control
crw-rw-rw-  1 root root     238,  10 Jun 20 17:19 xdma0_events_0
crw-rw-rw-  1 root root     238,  11 Jun 20 17:19 xdma0_events_1
[...]
crw-rw-rw-  1 root root     238,  24 Jun 20 17:19 xdma0_events_14
crw-rw-rw-  1 root root     238,  25 Jun 20 17:19 xdma0_events_15
crw-rw-rw-  1 root root     238,  32 Jun 20 17:19 xdma0_h2c_0
crw-rw-rw-  1 root root     238,   0 Jun 20 17:19 xdma0_user
crw-rw-rw-  1 root root     238,   2 Jun 20 17:19 xdma0_xvc
```

Interface to user space

On the other side (towards the user space), Xilinx DMA driver provides several char devices:

```
$ ll /dev | grep xdma
drwxr-xr-x  3 root root           60 Jun 20 17:19 xdma
crw-rw-rw-  1 root root     238,  36 Jun 20 17:19 xdma0_c2h_0
crw-rw-rw-  1 root root     238,   1 Jun 20 17:19 xdma0_control
crw-rw-rw-  1 root root     238,  10 Jun 20 17:19 xdma0_events_0
crw-rw-rw-  1 root root     238,  11 Jun 20 17:19 xdma0_events_1
[...]
crw-rw-rw-  1 root root     238,  24 Jun 20 17:19 xdma0_events_14
crw-rw-rw-  1 root root     238,  25 Jun 20 17:19 xdma0_events_15
crw-rw-rw-  1 root root     238,  32 Jun 20 17:19 xdma0_h2c_0
crw-rw-rw-  1 root root     238,   0 Jun 20 17:19 xdma0_user
crw-rw-rw-  1 root root     238,   2 Jun 20 17:19 xdma0_xvc
```

c2h
AXI4 port (DMA from device)

h2c
AXI4 port (DMA to device)

user
AXI4-Lite port

events_N
AXI4 port (DMA from device)

Interface to user application

To access the status and control registers (on AXI4-Lite interface), C variant:

```c
#define ADDR_STATUS_REG          (0x1020)

int fd = open("/dev/xdma/card0/user", O_RDWR | O_SYNC);
if (fd < 0) {
        perror("open()");
        return EXIT_FAILURE;
}

void* mem = mmap(0, mmap_size, PROT_READ | PROT_WRITE, MAP_SHARED, fd, mmap_offset);
if (mem = MAP_FAILED) {
        perror("mmap()");
        return EXIT_FAILURE;
}

uint32_t status = *(uint32_t*)((char*)mmap_base + ADDR_STATUS_REG);
```

Interface to user application

To access the status and control registers (on AXI4-Lite interface), Python variant

```python
class HwAccessPcie(object):

        def __init__(self):
                user_filename = "/dev/xdma/card0/user"
                self.fd_user = os.open(user_filename, os.O_RDWR)
                self.mem = mmap.mmap(self.fd_user, 4 * 1024 * 1024)

        def __close__(self):
                self.mem.close()
                os.close(self.fd_user)

        def _rd32(self, addr):
                bs = self.mem[addr:addr + 4]
                return struct.unpack("I", bs)[0]

        def _wr32(self, addr, data):
                bs = struct.pack("I", data)
                self.mem[addr:addr + 4] = bs
```

Hot-plug and hot-swap

- Hot-plug requires that virtual memory is additional kernel arguments:

```
pci=realloc,assign-busses,hpmemsize=32M
```

- Listing of a hot-plug procedure is shown on the next slide

Hot-plug

```
[   11.719854] xdma: loading out-of-tree module taints kernel.
[   11.719902] xdma: module verification failed: signature and/or required key missing - tainting
[   11.720630] xdma:xdma_mod_init: Xilinx XDMA Reference Driver xdma v2018.3.50
[   11.720631] xdma:xdma_mod_init: desc_blen_max: 0xfffffff/268435455, sgdma_timeout: 10 sec.
[   12.938964] JAN: plug in AMC and push in the handle
[   26.347956] pciehp 0000:04:08.0:pcie204: Slot(5): Attention button pressed
[   26.347978] pciehp 0000:04:08.0:pcie204: Slot(5) Powering on due to button press
[   26.505506] pciehp 0000:04:08.0:pcie204: Slot(5): Link Up
[   27.420194] pci 0000:07:00.0: [10ee:7024] type 00 class 0x070001
...
[   27.420471] pci 0000:07:00.0: BAR 0: assigned [mem 0xa6000000-0xa67fffff]
[   27.420477] pci 0000:07:00.0: BAR 1: assigned [mem 0xa6800000-0xa680ffff]
[   27.420482] pcieport 0000:04:08.0: PCI bridge to [bus 07]
[   27.420485] pcieport 0000:04:08.0:   bridge window [io  0x4000-0x4fff]
[   27.420489] pcieport 0000:04:08.0:   bridge window [mem 0xa6000000-0xa9ffffff]
[   27.420492] pcieport 0000:04:08.0:   bridge window [mem 0x96000000-0x97ffffff 64bit pref]
...
[   27.420944] xdma:map_single_bar: BAR0 at 0xa6000000 mapped at 0x000000004c4894e6, length=83886
[   27.420977] xdma:map_single_bar: BAR1 at 0xa6800000 mapped at 0x00000000b314fa42, length=65536
[   27.420980] xdma:map_bars: config bar 1, pos 1.
[   27.420981] xdma:identify_bars: 2 BARs: config 1, user 0, bypass -1.
[   27.421038] xdma:probe_one: 0000:07:00.0 xdma0, pdev 0x00000000a8508428, xdev 0x00000000affcab
[   27.421833] xdma:cdev_xvc_init: xcdev 0x00000000b85138f3, bar 0, offset 0x40000.
```

Summary of the software part

- Drivers are provided by FPGA vendors
- Easy-to-use user-space interface (for register access, DMA and interrupts)
- Hot-swap facilitates development and improves up-time