# DMA transfer, PCIe Driver and FPGA Tools.

Jan Marjanovic (MTCA Tech Lab/DESY) 2019-06-24

**MTCA/ATCA Workshop China 2019 at IHEP, Beijing**

MicroTCA TECHNOLOGY LAB

HELMHOLTZ RESEARCH FOR GRAND CHALLENGES

DESY.

# Introduction

- MicroTCA and PCI Express
- PCI Express protocol
- FPGA side
- Software side
- Troubleshooting techniques
- Further resources and summary

# MicroTCA and PCI Express

A typical ADC system in MicroTCA is shown here:

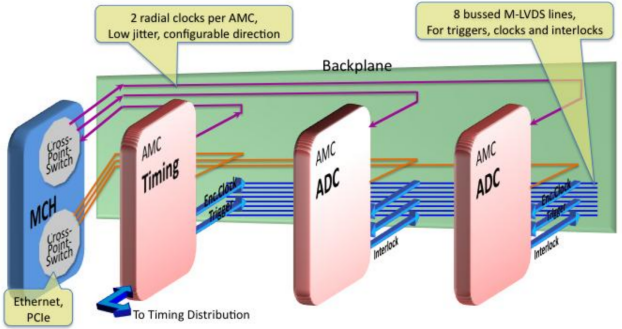



**Figure 6-6: A 'typical' analog front-end.**
**CPU and further AMC's are not shown**

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

A typical board in such a system is shown here:

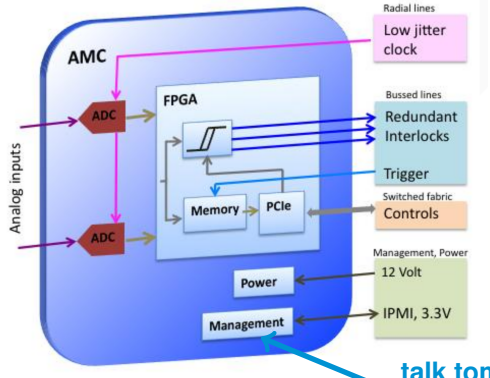



Figure 6-7: An example of a fast front-end (ADC AMC0)

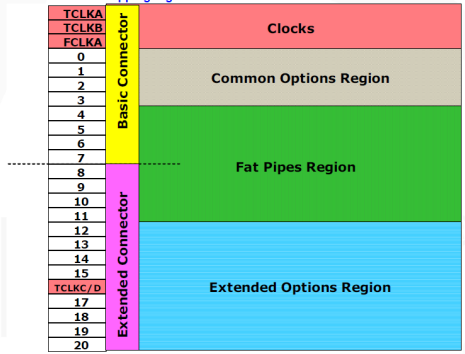
**talk tomorrow:**
**MMC Stamp and its applications**

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:

Figure 6-11 AMC Port mapping regions




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

# MicroTCA backplane

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

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

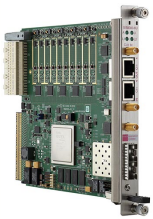
# Reasons to use PCI Express

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

# Reasons to use PCI Express

A lot of boards available on the market use PCI Express as main/only communication protocol.

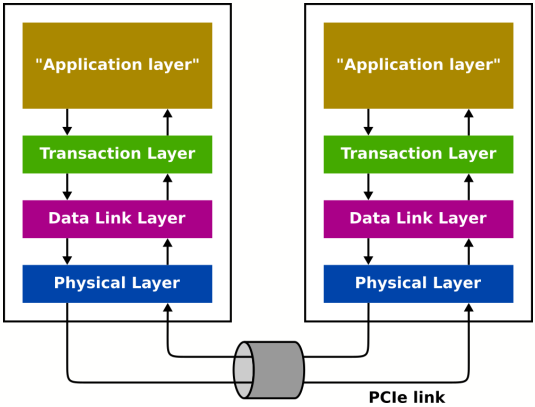
FPGAs also allow implementation of other protocols (e.g. 10 and 40 Gb Ethernet and Serial Rapid IO).





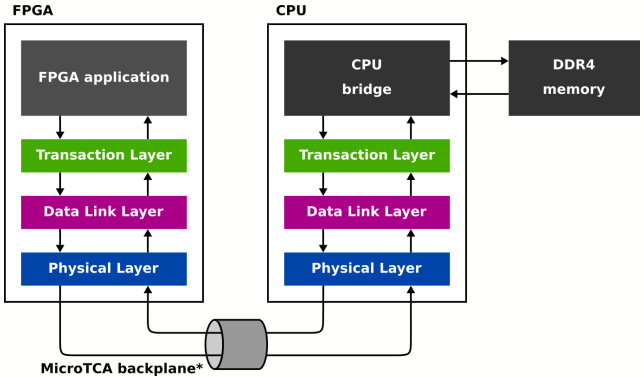

# PCI Express protocol

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)

# 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.

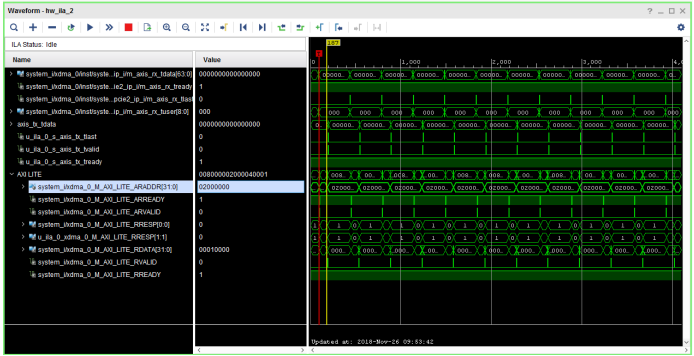
CPU writes to a device registers


CPU reads from a device registers

# Getting the data from FPGA to CPU

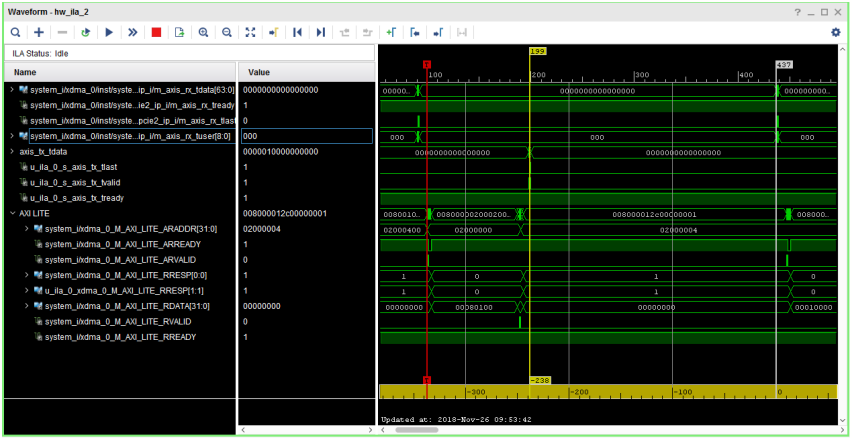
- Often we need to access a large quantity of data (e.g. readout from ADCs)
- One option is to use memcpy() - CPU fetches data 64 bit at a time (using MRd command)
- Experimental results show that we can get around 2.5 MB/s with kernel-space memcpy() (see backup slides for the method and the results)

memcpy() as seen from FPGA:



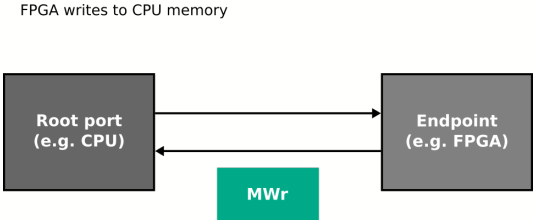
# Getting the data from FPGA to CPU

memcpy() as seen from FPGA (single 64 bit read):



# Bus Mastering and DMA

Bus Mastering means that endpoint (device) can send read and write packets.




Usually driver enables bus mastering on a device when the device gets configured. Result can be observed with lspci:

```
$ sudo lspci -s 0c:00 -vv
0c:00.0 Signal processing controller: Xilinx Corporation Device 0038 (rev 25)
    Subsystem: Xilinx Corporation Device 0007
    Physical Slot: 12
    Control: I/O- Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr- Stepping-
        SERR- FastB2B- DisINTx+
    Status: Cap+ 66MHz- UDF- FastB2B- ParErr- DEVSEL=fast >TAbort- <TAbort-
        <MAbort- >SERR- <PERR- INTx-
    [...]
```

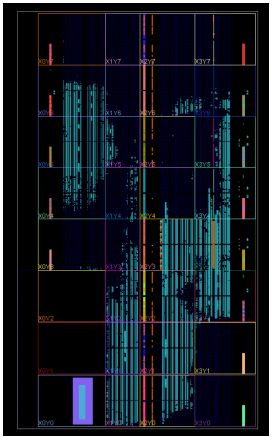
# PCIe protocol summary

- Packet-oriented protocol, memory-based
- Three important packet types to transfer data
  - Memory Read
  - Memory Write
  - Completion with Data
- Root port (i.e. CPU) and endpoints (FPGA devices, typically)
- Bus Mastering needed for DMA from device (endpoint)
- lspci - Linux command for inspecting PCI and PCIe devices

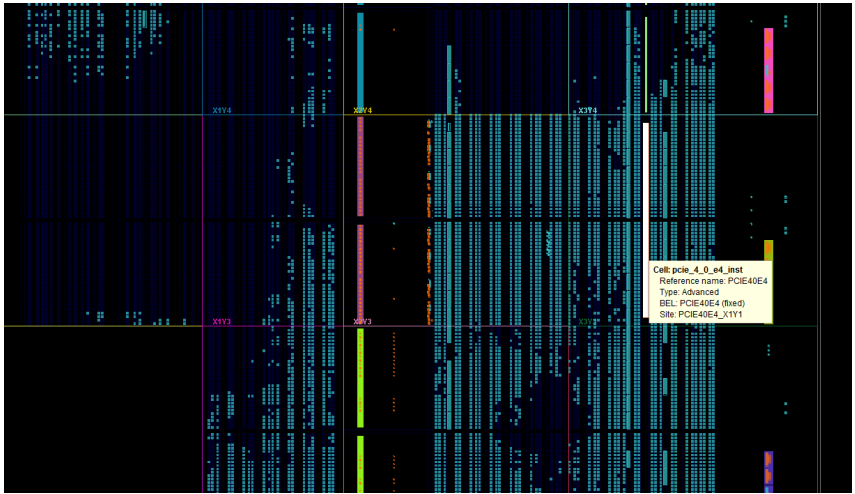
# FPGA side

# Implementation of PCIe in FPGA

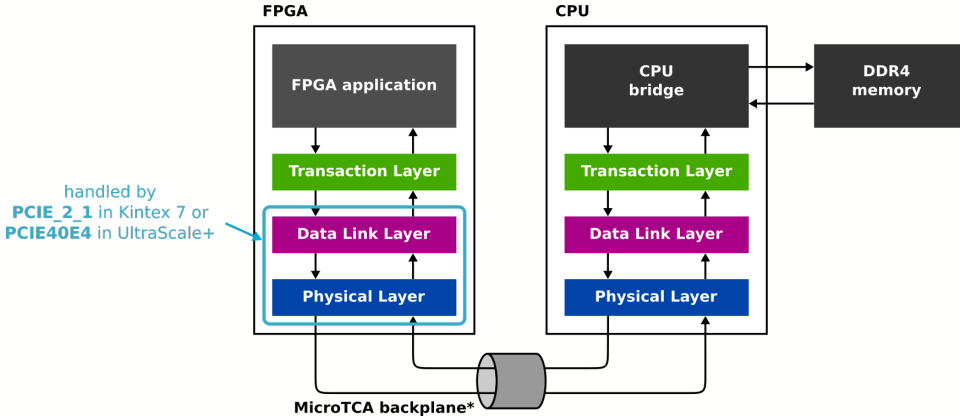
Modern FPGAs from Xilinx and Intel include hard IP cores to handle lower layers of protocol.



# Implementation of PCIe in FPGA

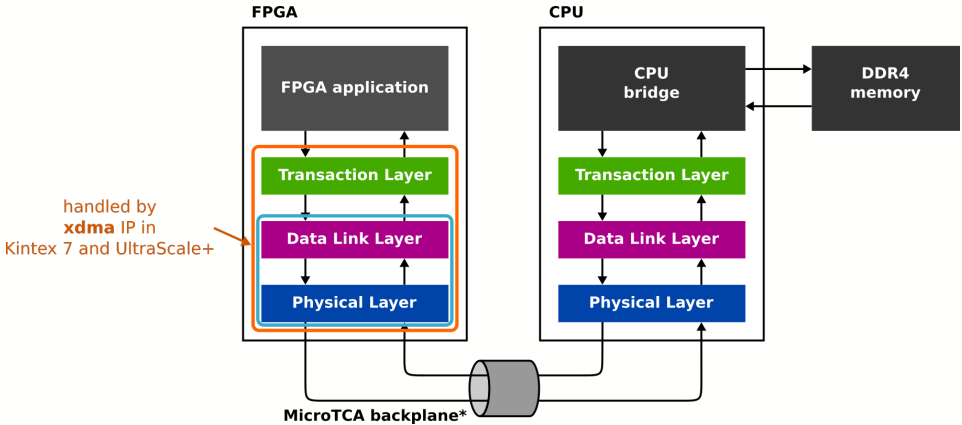


# Implementation of PCIe in FPGA

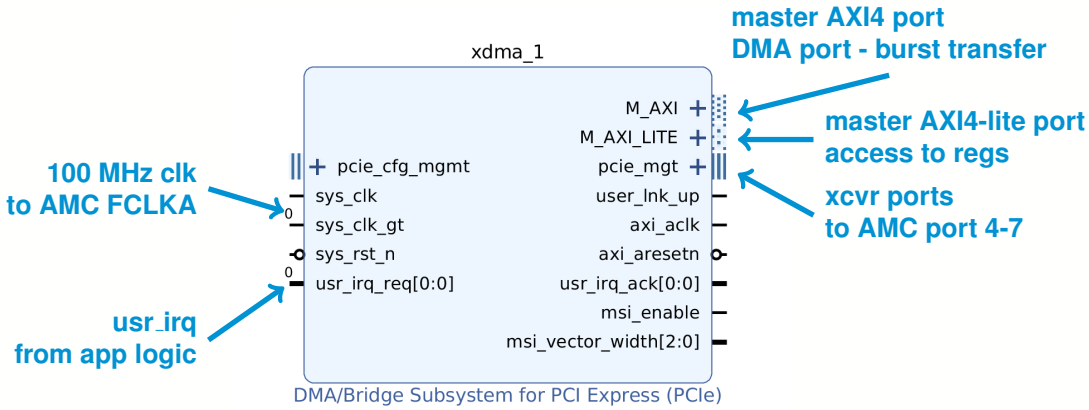


# Implementation of PCIe in FPGA

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

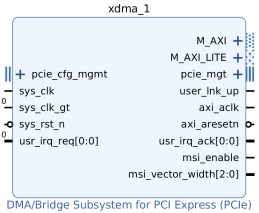


This is how the Xilinx DMA Subsystem for PCI Express looks in Vivado:



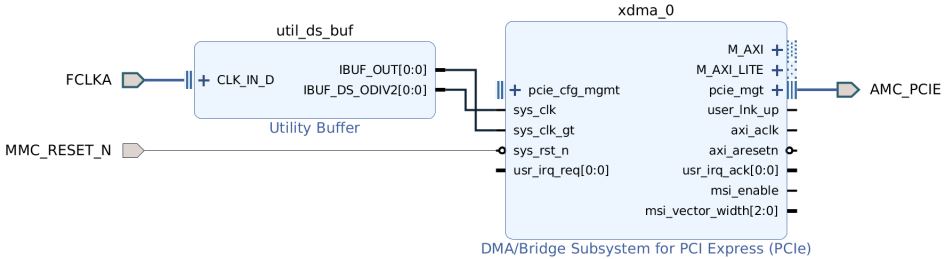
# Let's build a system, step by step

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



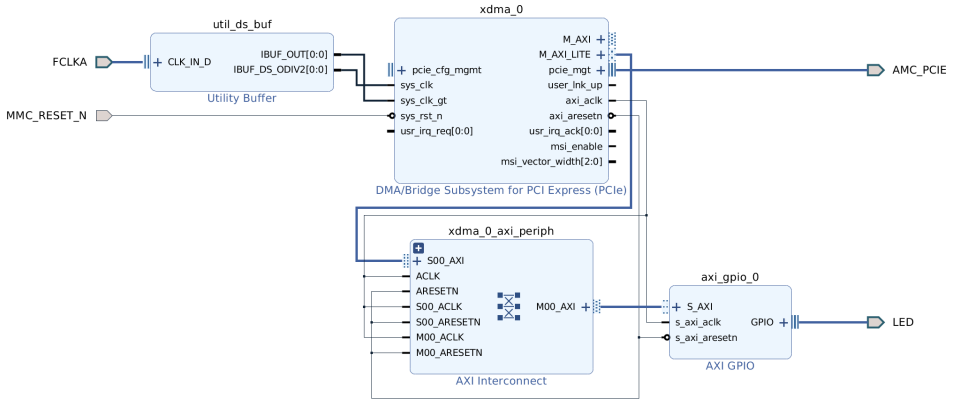
# Let's build a system, step by step

2. We then connect external connections:



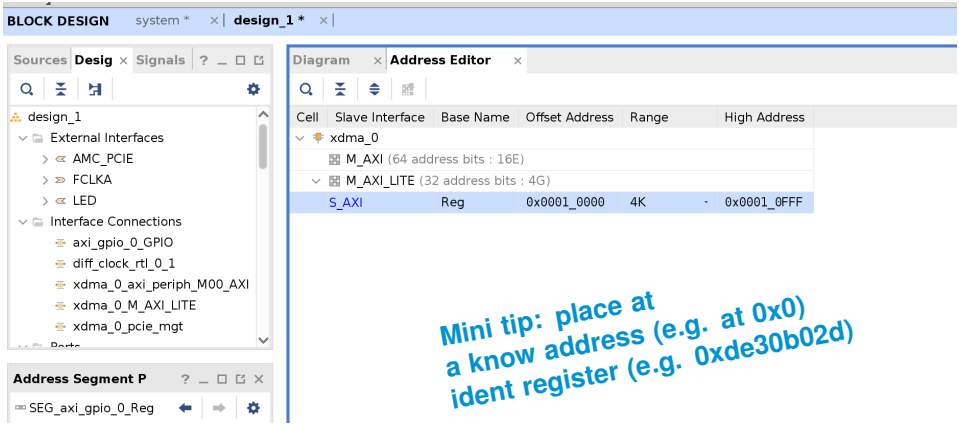
# Let's build a system, step by step

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



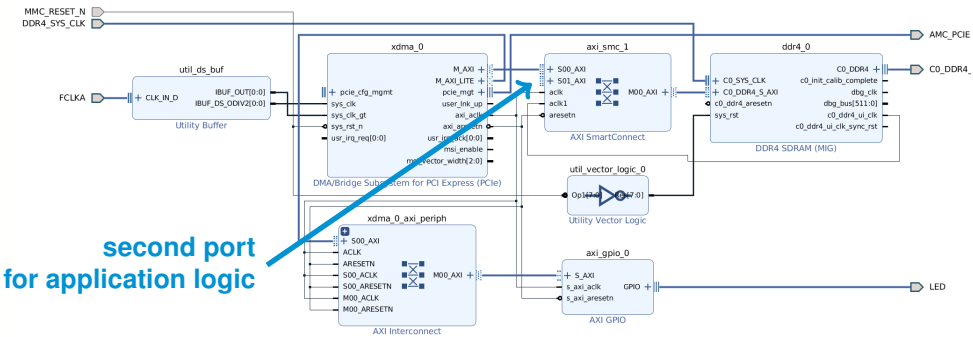
# Let's build a system, step by step

4. Let's not forget to assign the addresses



# Let's build a system, step by step

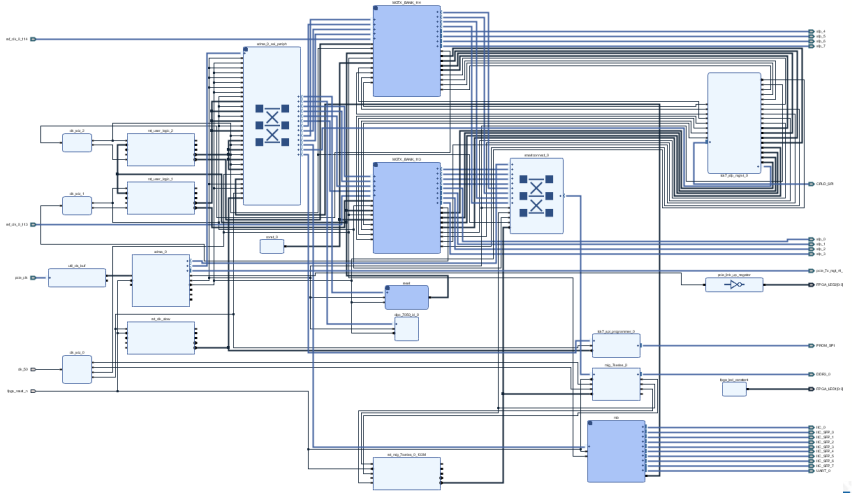
5. We add a DDR4 controller (not a trivial step, a lot of configuration) and connect the DMA port of PCIe to DDR4 controller:



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

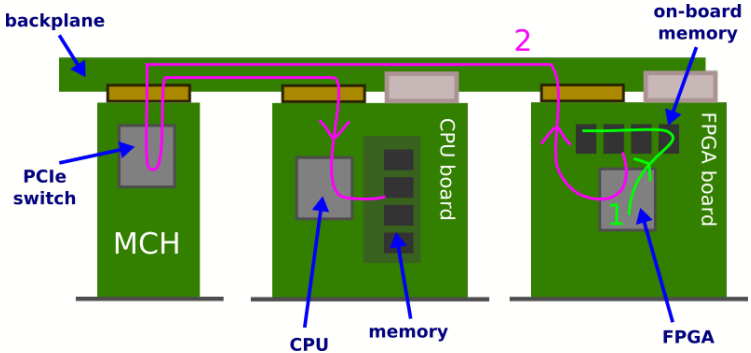
# Typical PCIe-based system in FPGA

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



# Summary of the FPGA part

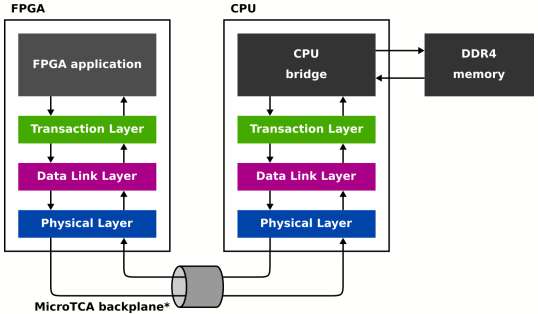
- Modern FPGAs provide a lot of hard IP
- Typical flow involves configuring IPs and connecting them together
- Custom logic required for application-specific part
- Two-step transfer: first to on-board memory, then to CPU



# 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/`

# 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

# 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

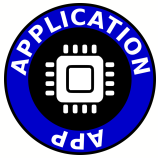
# Troubleshooting techniques

# Three possible roles

- Not all techniques are needed in all cases
- Following slides are marked with icons to indicate the target roles
- Here we distinguish between three different roles

**Hardware Developer**


**Application Developer**




**System Integrator**

# Troubleshooting configuration

Linux command lspci, expected output:


```
$ lspci
00:00.0 Host bridge: Intel Corporation Xeon E3-1200 v6/7th Gen Core Processor Host Br
00:01.0 PCI bridge: Intel Corporation Xeon E3-1200 v5/E3-1500 v5/6th Gen Cor ess
...
01:00.1 System peripheral: PLX Technology, Inc. Device 87d0 (rev ca)
01:00.2 System peripheral: PLX Technology, Inc. Device 87d0 (rev ca)
01:00.3 System peripheral: PLX Technology, Inc. Device 87d0 (rev ca)
01:00.4 System peripheral: PLX Technology, Inc. Device 87d0 (rev ca)
02:01.0 PCI bridge: PLX Technology, Inc. Device 8725 (rev ca)
02:02.0 PCI bridge: PLX Technology, Inc. Device 8725 (rev ca)
02:08.0 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
04:01.0 PCI bridge: PLX Technology, Inc. PEX 8748 48-Lane, 12-Port PCI Express Gen 3
...
07:00.0 Serial controller: Xilinx Corporation Device 7024
0d:00.0 Ethernet controller: Intel Corporation Ethernet Controller X710 for 10GbE SFP
...
```

# Troubleshooting configuration

Linux command lspci
Expected output:

```
$ lspci
...
07:00.0 Serial controller: Xilinx Corporation Device 7024
...
```



# Troubleshooting configuration

Linux command lspci
More information




```
$ sudo lspci -s 07:00.0 -v
07:00.0 Serial controller: Xilinx Corporation Device 7024 (prog-if 01 [16450])
  Subsystem: Xilinx Corporation Device 0007
  Physical Slot: 5
  Flags: bus master, fast devsel, latency 0, IRQ 17
  Memory at 92800000 (32-bit, non-prefetchable) [size=8M]
  Memory at 93000000 (32-bit, non-prefetchable) [size=64K]
  Capabilities: [40] Power Management version 3
  Capabilities: [48] MSI: Enable- Count=1/1 Maskable- 64bit+
  Capabilities: [60] Express Endpoint, MSI 00
  Capabilities: [100] Device Serial Number 00-00-00-00-00-00-00-00
  Kernel driver in use: xdma
```

# xdma driver

To enable more verbose output from the driver, un-comment line with LIBXDMA_DEBUG in xdma/Makefile:

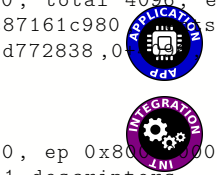
```
EXTRA_CFLAGS := -I$(topdir)/include $(XVC_FLAGS)
#EXTRA_CFLAGS += -D__LIBXDMA_DEBUG__
#EXTRA_CFLAGS += -DINTERNAL_TESTING
```



or fine-grained control in libxdma/libxdma.h (enable printout for individual fuction, e.g. interrupt handler)

## Dmesg output:

```
[ 1332.118599] xdma:char_sgdma_llseek: char_sgdma_llseek: pos=2147483648
[ 1332.125051] xdma:char_sgdma_read_write: file 0xffffffc87a6ea900, priv 0xffffffc875
[ 1332.138547] xdma:xdma_request_cb_dump: request 0xffffffc87a6a5e00, total 4096, ep
[ 1332.150105] xdma:sgt_dump: sgt 0xffffff800fecbd68, sgl 0xffffffc87161c980 s 1
[ 1332.157929] xdma:sgt_dump: 0, 0xffffffc87161c980, pg 0xffffffbf1d772838,0 dm
[ 1332.167141] xdma:xdma_request_cb_dump: 0/1, 0x86b301000, 4096.
[ 1332.172967] xdma:xdma_xfer_submit: 0-C2H0-MM, len 4096 sg cnt 1.
[ 1332.178964] xdma:transfer_init: transfer->desc_bus = 0x701b0000.
[ 1332.184964] xdma:transfer_build: sw desc 0/1: 0x86b301000, 0x1000, ep 0x80001000.
[ 1332.192437] xdma:transfer_init: transfer 0xffffffc87a6a5e18 has 1 descriptors
[ 1332.199566] xdma:xdma_xfer_submit: xfer, 4096, ep 0x80001000, done 0, sg 1/1.
[ 1332.206692] xdma:transfer_dump: xfer 0xffffffc87a6a5e18, state 0x0, f 0x1, dir 2,
[ 1332.215729] xdma:transfer_dump: transfer 0xffffffc87a6a5e18, desc 1, bus 0x701b000
[ 1332.224070] xdma:dump_desc: 0xffffff800fbc2000/0x00: 0xad4b0013 0xad4b0013 magic|e
[ 1332.233455] xdma:dump_desc: 0xffffff800fbc2004/0x04: 0x00001000 0x00001000 bytes
[ 1332.240841] xdma:dump_desc: 0xffffff800fbc2008/0x08: 0x80000000 0x80000000 src_add
[ 1332.248756] xdma:dump_desc: 0xffffff800fbc200c/0x0c: 0x00000000 0x00000000 src_add
```



# MCH configuration

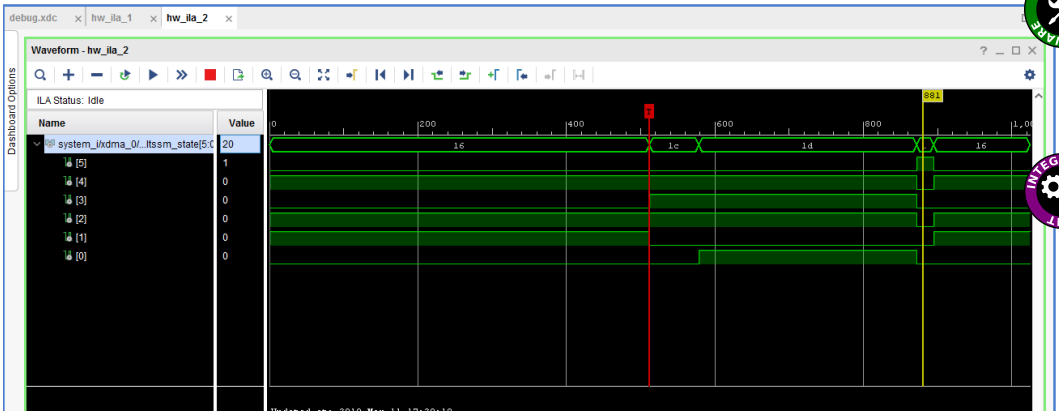
MCH provides an overview of the link stati per individual port:

**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 | - |

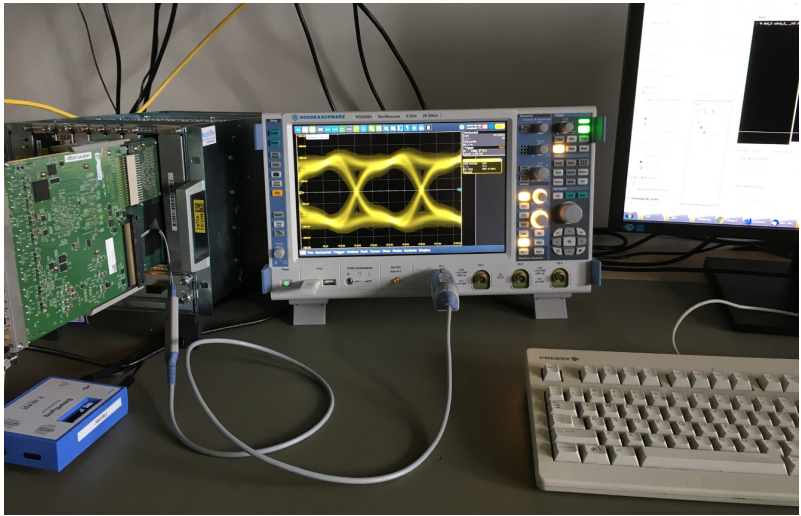

# Link state

Monitor LTSSM with Integrated Logic Analyzer (ILA)




Shown above is the behavior of a faulty link - link should stay in state L0 (0x16).

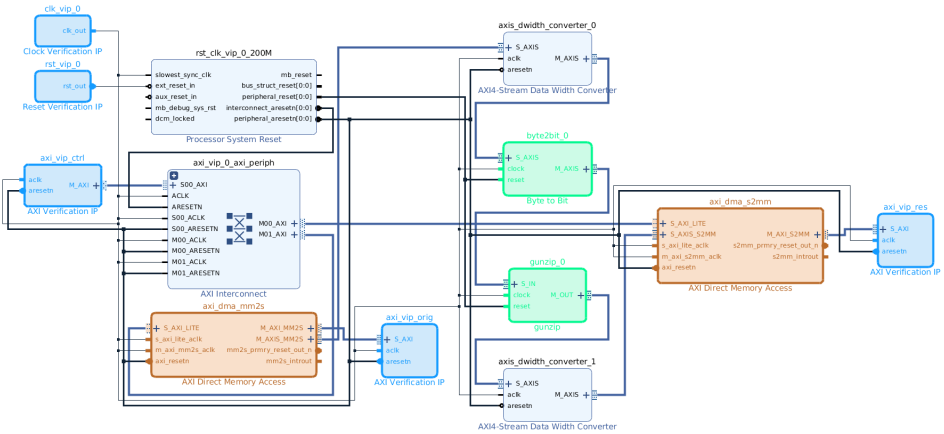
# Eye diagram with oscilloscope



# Protocol decoding with oscilloscope

# Application logic - unit tests

From `https://github.com/j-marjanovic/chisel-stuff/tree/master/example-5-gunzip/vivado_tb_project`



# 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, ...
- Modern FPGA development is focused on connecting IP cores together
- There are a lot of ”tools” to troubleshoot PCIe-based systems
- MicroTCA Tech Lab provides BSP with PCIe for some of the boards, more BSPs (e.g. for Zynq UltraScale MPSoc) to follow


## 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
# 谢谢

https://techlab.desy.de

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

# Backup slides

# memcpy meas: memcpy implementation

Inspired by `drivers/net/wireless/broadcom/brcm80211/brcmfmac/pcie.c#L486`

```
static void xdma_memcpy_from_dev (
        struct xdma_dev *lro,
        u32 offs,
        void *dstaddr,
        u32 len
) {
        void __iomem *address = lro->bar[lro->user_bar_idx] + offs;
        u32* dst32;

        len = len / 4;
        dst32 = (u32*)dstaddr;

        while (len) {
                *dst32 = ioread32(address);
                address += 4;
                dst32++;
                len--;
        }
}
```

# memcpy meas: time measurement

see also https://www.kernel.org/doc/html/v4.15/driver-api/device-io.html

```
long char_ctrl_ioctl(struct file *filp, unsigned int cmd, unsigned long arg) {
        struct xdma_dev *lro;
        struct xdma_char *lro_char = (struct xdma_char *)filp->private_data;
        u64 t0_ns, t1_ns, duration_ns;

        BUG_ON(!lro_char);
        BUG_ON(lro_char->magic != MAGIC_CHAR);
        lro = lro_char->lro;
        BUG_ON(!lro);
        BUG_ON(lro->magic != MAGIC_DEVICE);

        printk(KERN_DEBUG "JAN: memcpy started, size: %lu\n", sizeof(tmp));
        t0_ns = ktime_get_ns();
        xdma_memcpy_from_dev(lro, USR_DDR3_OFFS, (void*)tmp, sizeof(tmp));
        t1_ns = ktime_get_ns();
        printk(KERN_DEBUG "JAN: memcpy done\n");
        duration_ns = t1_ns - t0_ns;
        printk(KERN_DEBUG "JAN: duration %llu ns (t1 = %llu, t0 = %llu)\n",
        duration_ns, t1_ns, t0_ns);
        return 0;
}
```

# memcpy meas: CPU 1

On Concurrent Tech AM900x
kernel 4.4.0-139-generic
Intel(R) Core(TM) i7-3612QE CPU @ 2.10GHz

```
[55775.132977] JAN: memcpy started, size: 1048576
[55775.498411] JAN: memcpy done
[55775.498414] JAN: duration 560049231 ns (t1 = 111312945419962, t0 = 111312385370731)

[55793.478833] JAN: memcpy started, size: 1048576
[55793.844333] JAN: memcpy done
[55793.844335] JAN: duration 468041144 ns (t1 = 111352256875650, t0 = 111351788834506)

[55819.083870] JAN: memcpy started, size: 1048576
[55819.449562] JAN: memcpy done
[55819.449564] JAN: duration 520045715 ns (t1 = 111401605213633, t0 = 111401085167918)
```

# memcpy meas: CPU 2

On ADLINK AMC-1000
kernel 4.15.0-39-generic
Intel(R) Core(TM)2 Duo CPU L7400 @ 1.50GHz

```
[  357.333248] JAN: memcpy started, size: 1048576
[  357.792152] JAN: memcpy done
[  357.792157] JAN: duration 458898891 ns (t1 = 357792150659, t0 = 357333251768)

[  360.061151] JAN: memcpy started, size: 1048576
[  360.520068] JAN: memcpy done
[  360.520073] JAN: duration 458910762 ns (t1 = 360520066218, t0 = 360061155456)

[  362.647225] JAN: memcpy started, size: 1048576
[  363.106262] JAN: memcpy done
[  363.106268] JAN: duration 459030051 ns (t1 = 363106259232, t0 = 362647229181)
```

# memcpy meas: CPU 3

On external CPU
kernel 4.15.0-39-generic
Intel(R) Core(TM) i5-2500 CPU @ 3.30GHz

```
[  225.678066] JAN: memcpy started, size: 1048576
[  226.128774] JAN: memcpy done
[  226.128775] JAN: duration 450702965 ns (t1 = 225831915216, t0 = 225381212251)

[  229.493646] JAN: memcpy started, size: 1048576
[  229.944742] JAN: memcpy done
[  229.944743] JAN: duration 451090885 ns (t1 = 229647854461, t0 = 229196763576)

[  232.285649] JAN: memcpy started, size: 1048576
[  232.736626] JAN: memcpy done
[  232.736627] JAN: duration 450971647 ns (t1 = 232439716846, t0 = 231988745199)
```

# xdma tools: reg_rw

reg_rw tool, e.g. read

```
$ reg_rw /dev/xdma/card0/user 0x10038 w
argc = 4
device: /dev/xdma/card0/user
address: 0x00010038
access type: write
access width given.
access width: word (32-bits)
character device /dev/xdma/card0/user opened.
Memory mapped at address 0x7f34356fb000.
Read 32-bit value at address 0x00010038 (0x7f343570b038): 0x000107ff
```

# xdma tools: reg_rw

reg_rw tool, e.g. write

```
$ reg_rw /dev/xdma/card0/user 0x10038 w 0x107ff
argc = 5
device: /dev/xdma/card0/user
address: 0x00010038
access type: write
access width given.
access width: word (32-bits)
character device /dev/xdma/card0/user opened.
Memory mapped at address 0x7f1675e65000.
Write 32-bits value 0x000107ff to 0x00010038 (0x0x7f1675e75038)
```

## Dmesg output:

```
[ 1332.118599] xdma:char_sgdma_llseek: char_sgdma_llseek: pos=2147483648
[ 1332.125051] xdma:char_sgdma_read_write: file 0xffffffc87a6ea900, priv 0xffffffc875f3e0e8, buf 0x0000000020d21000,4096, pos 214748
[ 1332.138547] xdma:xdma_request_cb_dump: request 0xffffffc87a6a5e00, total 4096, ep 0x80000000, sw_desc 1, sgt 0xffffff800fecbd68.
[ 1332.150105] xdma:sgt_dump: sgt 0xffffff800fecbd68, sgl 0xffffffc87161c980, nents 1/1.
[ 1332.157929] xdma:sgt_dump: 0, 0xffffffc87161c980, pg 0xffffffbf1d772838,0+4096, dma 0x86b301000,4096.
[ 1332.167141] xdma:xdma_request_cb_dump: 0/1, 0x86b301000, 4096.
[ 1332.172967] xdma:xdma_xfer_submit: 0-C2H0-MM, len 4096 sg cnt 1.
[ 1332.178964] xdma:transfer_init: transfer->desc_bus = 0x701b0000.
[ 1332.184964] xdma:transfer_build: sw desc 0/1: 0x86b301000, 0x1000, ep 0x80000000.
[ 1332.192437] xdma:transfer_init: transfer 0xffffffc87a6a5e18 has 1 descriptors
[ 1332.199566] xdma:xdma_xfer_submit: xfer, 4096, ep 0x80001000, done 0, sg 1/1.
[ 1332.206692] xdma:transfer_dump: xfer 0xffffffc87a6a5e18, state 0x0, f 0x1, dir 2, len 4096, last 1.
[ 1332.215729] xdma:transfer_dump: transfer 0xffffffc87a6a5e18, desc 1, bus 0x701b0000, adj 1.
[ 1332.224070] xdma:dump_desc: 0xffffff800fbc2000/0x00: 0xad4b0013 0xad4b0013 magic|extra_adjacent|control
[ 1332.233455] xdma:dump_desc: 0xffffff800fbc2004/0x04: 0x00001000 0x00001000 bytes
[ 1332.240841] xdma:dump_desc: 0xffffff800fbc2008/0x08: 0x80000000 0x80000000 src_addr_lo
[ 1332.248756] xdma:dump_desc: 0xffffff800fbc200c/0x0c: 0x00000000 0x00000000 src_addr_hi
[ 1332.256667] xdma:dump_desc: 0xffffff800fbc2010/0x00: 0x6b301000 0x6b301000 dst_addr_lo
[ 1332.264578] xdma:dump_desc: 0xffffff800fbc2014/0x04: 0x00000008 0x00000008 dst_addr_hi
[ 1332.272492] xdma:dump_desc: 0xffffff800fbc2018/0x08: 0x00000000 0x00000000 next_addr
[ 1332.280228] xdma:dump_desc: 0xffffff800fbc201c/0x0c: 0x00000000 0x00000000 next_addr_pad
[ 1332.288310] xdma:dump_desc:
[ 1332.291183] xdma:transfer_queue: transfer_queue(transfer=0xffffffc87a6a5e18).
[ 1332.298308] xdma:transfer_queue: transfer_queue(): starting 0-C2H0-MM engine.
[ 1332.305430] xdma:engine_start: engine_start(0-C2H0-MM): transfer=0xffffffc87a6a5e18.
[ 1332.313156] xdma:engine_start: iowrite32(0x701b0000 to 0xffffff800fba5080) (first_desc_lo)
[ 1332.321404] xdma:__write_register: engine_start: w reg 0xffffffb7958c66e0(0xffffff800fba5080), 0x701b0000.
[ 1332.331039] xdma:engine_start: iowrite32(0x00000000 to 0xffffff800fba5084) (first_desc_hi)
```


## Dmesg output:

```
[ 1332.339286] xdma:__write_register: engine_start: w reg 0xffffffb7958c66e4(0xffffff800fba5084), 0x0.
[ 1332.348314] xdma:engine_start: iowrite32(0x00000000 to 0xffffff800fba5088) (first_desc_adjacent)
[ 1332.357082] xdma:__write_register: engine_start: w reg 0xffffffb7958c66e8(0xffffff800fba5088), 0x0.
[ 1332.366109] xdma:engine_start: ioread32(0xffffff800fba1040) (dummy read flushes writes).
[ 1332.374183] xdma:engine_start_mode_config: iowrite32(0x00f83e1f to 0xffffff800fba1004) (control)
[ 1332.382951] xdma:__write_register: engine_start_mode_config: w reg 0xffffffb7958c266c(0xffffff800fba1004), 0xf83e1f.
[ 1332.393457] xdma:engine_start_mode_config: ioread32(0xffffff800fba1040) = 0x00000001 (dummy read flushes writes).
[ 1332.393477] xdma:xdma_isr: (irq=50, dev 0xffffffc87a2de000) <<<< ISR.
[ 1332.393482] xdma:xdma_isr: ch_irq = 0x00000002
[ 1332.393485] xdma:__write_register: channel_interrupts_disable: w reg 0x2000(0xffffff800fba2018), 0x2.
[ 1332.393489] xdma:xdma_isr: user_irq = 0x00000000
[ 1332.393491] xdma:xdma_isr: schedule_work, 0-C2H0-MM.
[ 1332.433310] xdma:engine_start: 0-C2H0-MM engine 0xffffffc87a2de978 now running
[ 1332.440514] xdma:transfer_queue: transfer=0xffffffc87a6a5e18 started 0-C2H0-MM engine with transfer 0xffffffc87a6a5e
[ 1332.451277] xdma:transfer_queue: engine->running = 1
[ 1332.456226] xdma:engine_service_work: engine_service() for 0-C2H0-MM engine ffffffc87a2de978
[ 1332.464649] xdma:engine_service_shutdown: engine just went idle, resetting RUN_STOP.
[ 1332.472381] xdma:xdma_engine_stop: xdma_engine_stop(engine=ffffffc87a2de978)
[ 1332.479413] xdma:xdma_engine_stop: Stopping SG DMA 0-C2H0-MM engine; writing 0x00f83e1e to 0xffffff800fba1004.
[ 1332.489396] xdma:__write_register: xdma_engine_stop: w reg 0xffffffb7958c266c(0xffffff800fba1004), 0xf83e1e.
[ 1332.499204] xdma:xdma_engine_stop: xdma_engine_stop(0-C2H0-MM) done
[ 1332.505457] xdma:engine_service: desc_count = 1
[ 1332.509969] xdma:engine_service: head of queue transfer 0xffffffc87a6a5e18 has 1 descriptors
[ 1332.518389] xdma:engine_service: Engine completed 1 desc, 1 not yet dequeued
[ 1332.525421] xdma:engine_service_final_transfer: engine 0-C2H0-MM completed transfer
[ 1332.533060] xdma:engine_service_final_transfer: Completed transfer ID = 0xffffffc87a6a5e18
[ 1332.541307] xdma:engine_service_final_transfer: *pdesc_completed=1, transfer->desc_num=1
[ 1332.541311] xdma:engine_service_resume: no pending transfers, 0-C2H0-MM engine stays idle.
[ 1332.557636] xdma:__write_register: channel_interrupts_enable: w reg 0x2000(0xffffff800fba2014), 0x2.
[ 1332.566759] xdma:xdma_xfer_submit: transfer ffffffc87a6a5e18, 4096, ep 0x80000000 compl, +0.
```