



# Direct Memory Access with FPGA.

Jan Marjanovic (DESY)

2018-12-04

Tutorial at MTCA Workshop 2018

**microTCA**  
TECHNOLOGY LAB

**HELMHOLTZ**  
RESEARCH FOR GRAND CHALLENGES



The objective of this tutorial session is to build a simple data acquisition system, in which the data is moved from an external device through an FPGA over PCIe into a CPU memory. The data will then be made available to user applications. We will use DAMC-TCK7 (and its Board Support Package) and a MicroTCA crate as a starting point. Then we will add a couple of Xilinx IP blocks and use Xilinx Linux driver to interface with FPGA modules to get to our final goal: having the data available in Python.

**Presenter:** Jan Marjanovic, MicroTCA Technology Lab

**Date:** Tuesday, 4 December 2018 15:00-17:00

**Venue:** CFEL, SR I-III

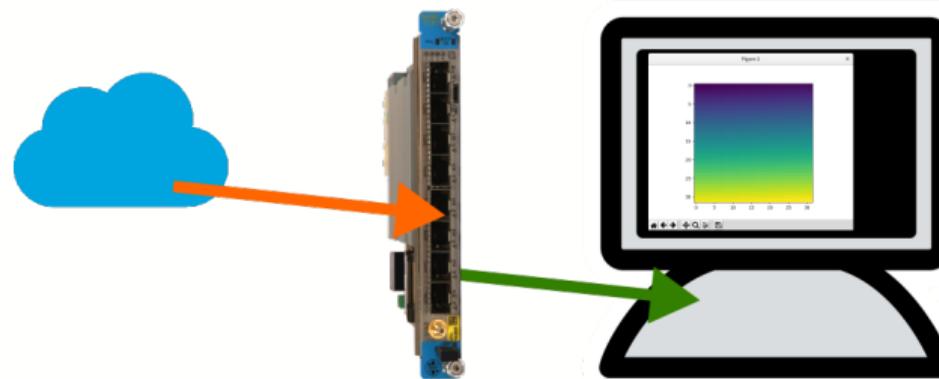
Jan Marjanovic

Firmware developer at MicroTCA Tech Lab

<https://www.desy.de/~jmarjan/>

Motivation for this talk:

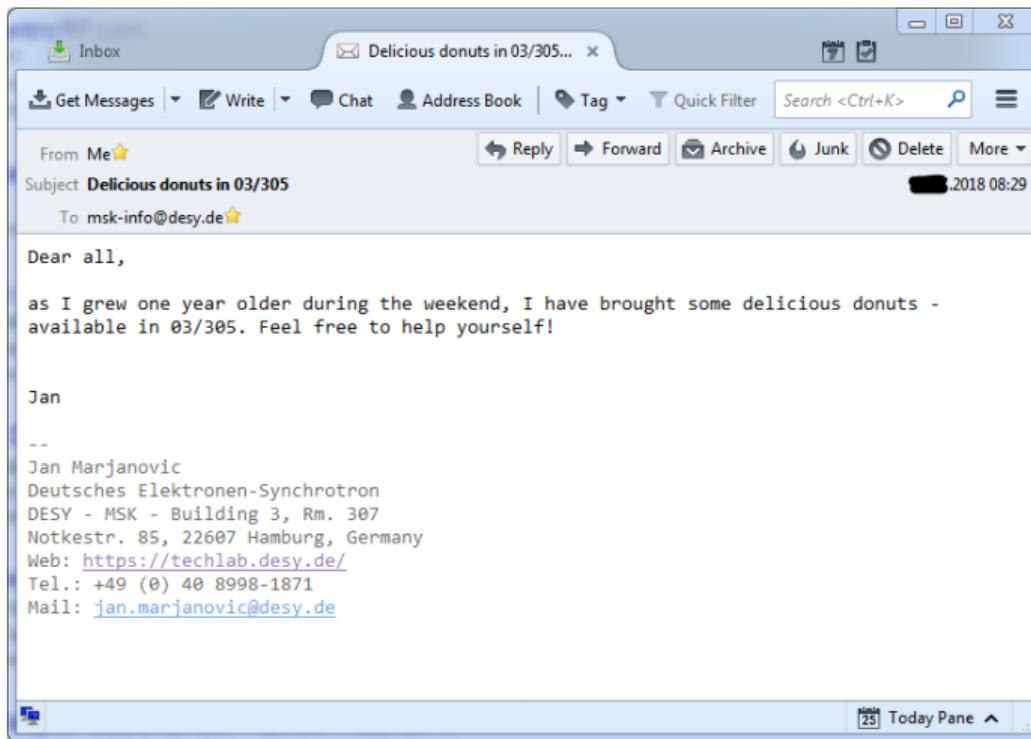
Student with a task to transfer data from a detector to his software/Matlab/...



From Wikipedia:

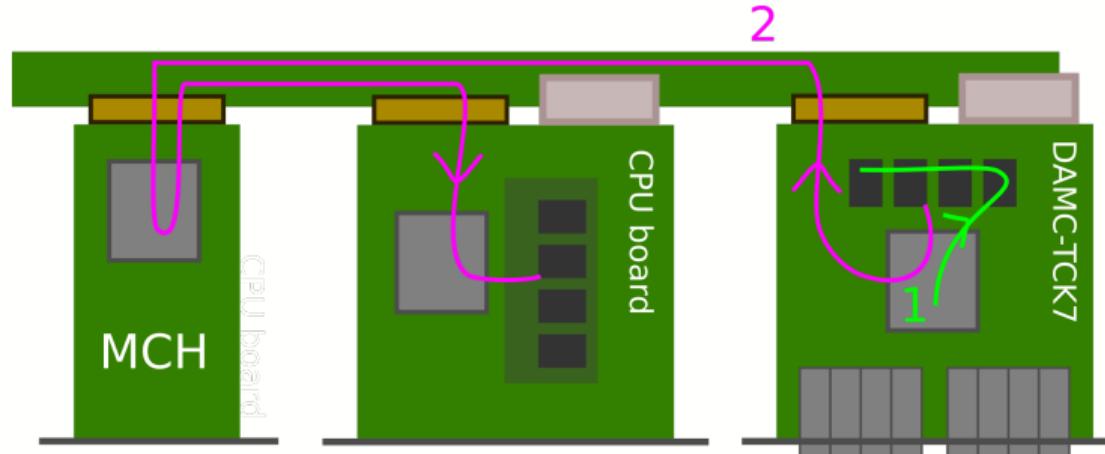
*Direct memory access (DMA) is a feature of computer systems that allows certain hardware subsystems [this is us] to access main system memory (random-access memory), independent of the central processing unit (CPU).*

# What is Direct Memory Access (DMA)?



# What is Direct Memory Access (DMA)?

Real example: move the data from a PCIe board into CPU memory

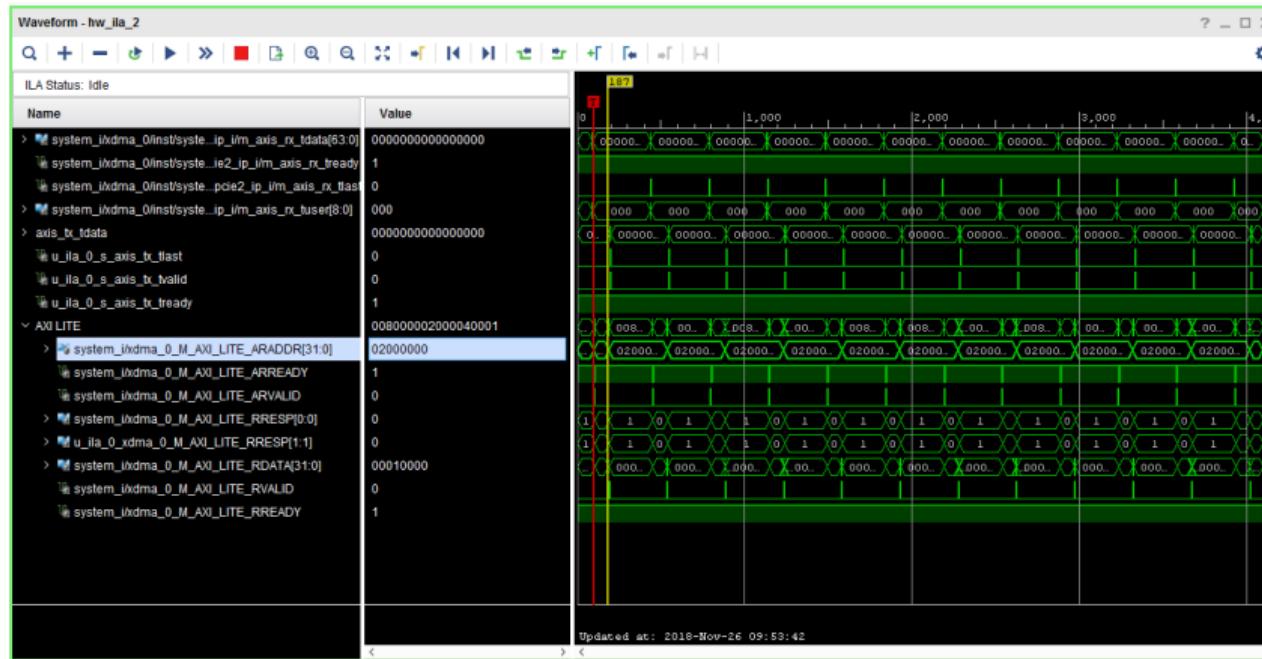


When we need to transfer large amounts of data quickly.

Alternative to DMA transfers is some kind of `memcpy()`:  
processor reads the data into its registers, and writes the data out of registers;  
several CPU cycles for each data word are needed

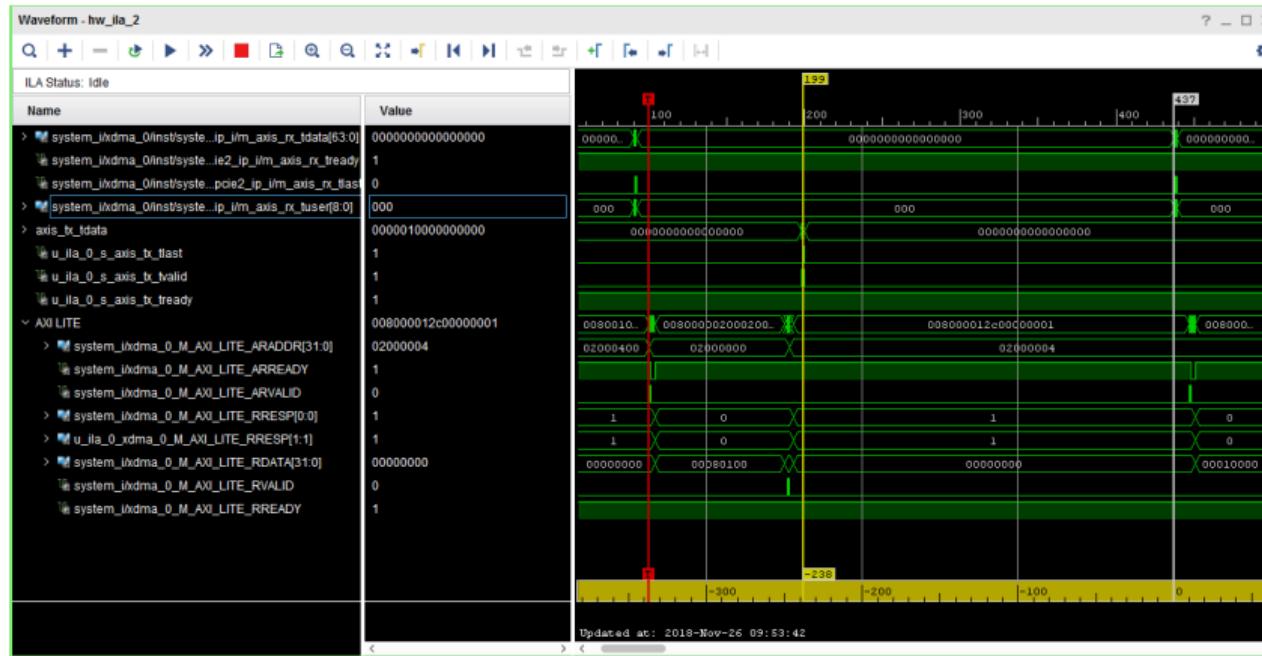
# When is DMA needed?

`memcpy()` over the PCIe in Linux kernel space results in  $\sim 2$  MB/s transfer rate



# When is DMA needed?

A large delay between new MRd request (@437) and completion (@199) can be noted; this is not under our control



We will be using Xilinx FPGA, Vivado, DAMC-TCK7. Crate is set-up in MTCA Tech Lab:



Quick tour after the tutorial of the Lab for all interested

DAMC-TCK7 is commercially available at N.A.T.:

<https://www.nateurope.com/products/NAT-AMC-TCK7.html>

and VadaTech:

[https://www.vadatech.com/product.php?product=353&catid\\_now=0&catid\\_prev=0](https://www.vadatech.com/product.php?product=353&catid_now=0&catid_prev=0)



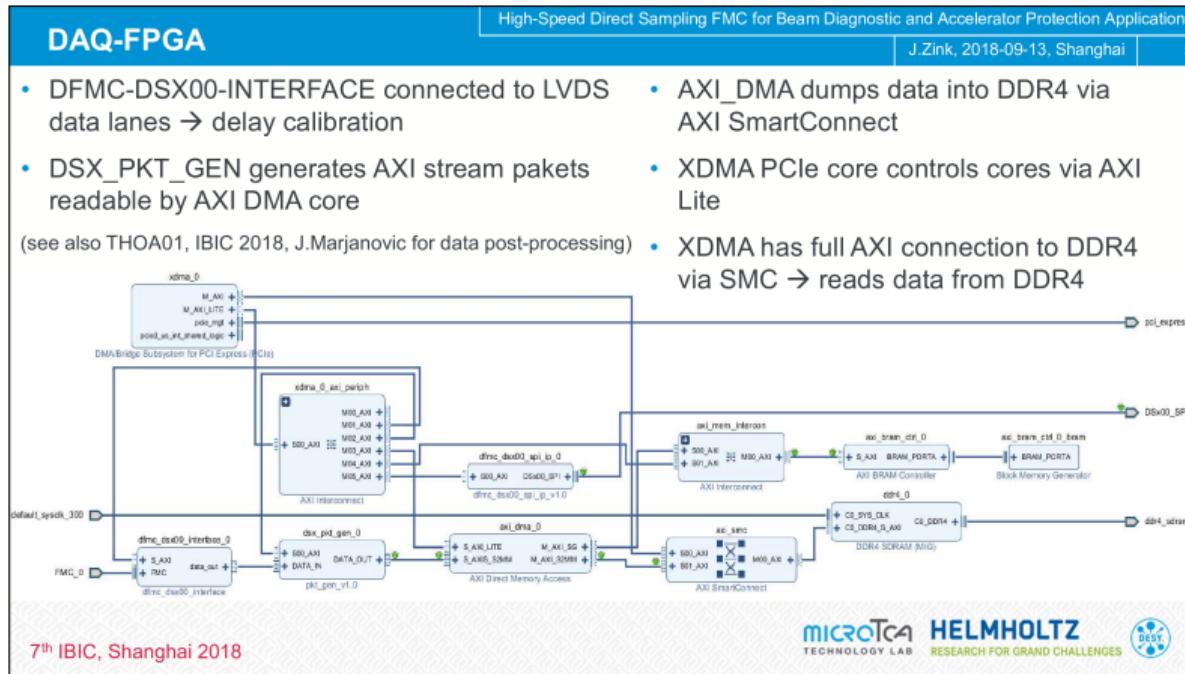
# Examples

- ▶ DFMC-DSx00 is an FMC mezzanine card with two-channel, 800 MSPS, 12-bit ADC
- ▶ Data rate is very high: 2400 MB/s
- ▶ DAQ application on Xilinx KCU105 eval kit

Talk on Thursday at 12:00 by J. Zink:  
"Direct Sampling of RF Signals up to 3 GHz in MTCA.4"

# Example 1: DAQ for DFMC-DS800

Direct Memory Access with FPGA  
Jan Marjanovic (DESY), 2018-12-04 Page 15/58



from [http://ibic2018.vrws.de/talks/thoa02\\_talk.pdf](http://ibic2018.vrws.de/talks/thoa02_talk.pdf)

microTCA  
TECHNOLOGY LAB

HELMHOLTZ  
RESEARCH FOR GRAND CHALLENGES



DAQ-Software	High-Speed Direct Sampling FMC for Beam Diagnostic and Accelerator Protection Applications J.Zink, 2018-09-13, Shanghai   6
<ul style="list-style-type: none"><li>straight forward solution for DAQ</li><li>XDMA transfers sample data from DDR4 (KCU105) into host PC's RAM via PCIe root complex (DMA controller)</li><li>Xilinx DMA driver (kernel space) provides pointer for DMA transfers into RAM</li><li>Python library DSX00Lib can access transferred data in kernel space</li></ul>	<ul style="list-style-type: none"><li>data can be dumped into file on SSD/HDD</li><li>over 1 Mio. samples per channel can be stored on KCU105 (2GB DDR4)</li><li>8k samples can be stored on DAMC-FMC25 (256 MB DDR2)</li></ul>

**KCU105**  
FPGA  
XDMA PCIe core

**HOST PC**  
RAM  
CPU  
PCIe Root Complex

Xilinx DMA Driver (kernel space)  
Python DSX00Lib (user space)  
SSD (file system)  
file

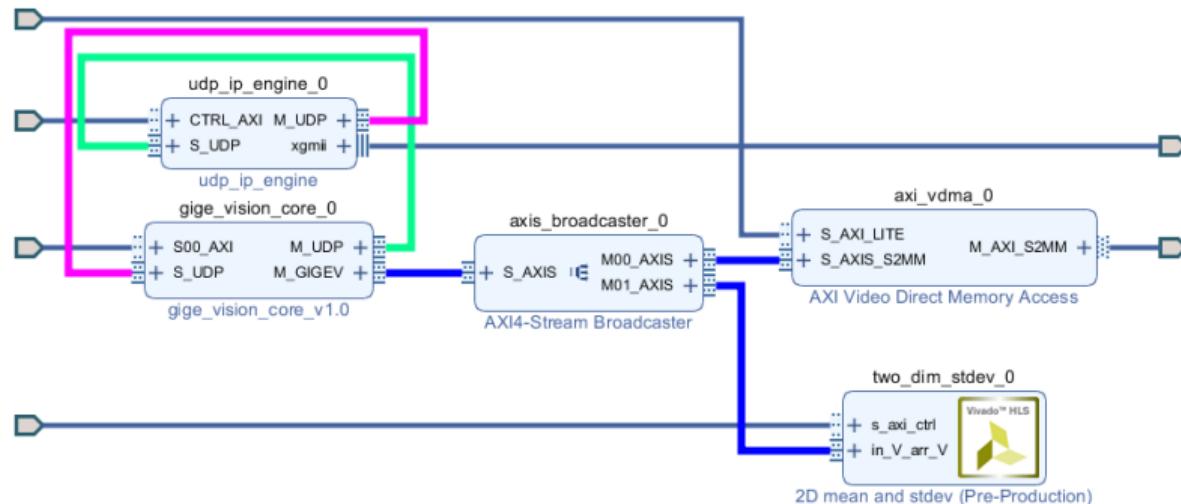
7<sup>th</sup> IBIC, Shanghai 2018

**microTCA** TECHNOLOGY LAB **HELMHOLTZ** RESEARCH FOR GRAND CHALLENGES

from [http://ibic2018.vrws.de/talks/thoa02\\_talk.pdf](http://ibic2018.vrws.de/talks/thoa02_talk.pdf)

## Example 2: Frame grabber for GigE Vision

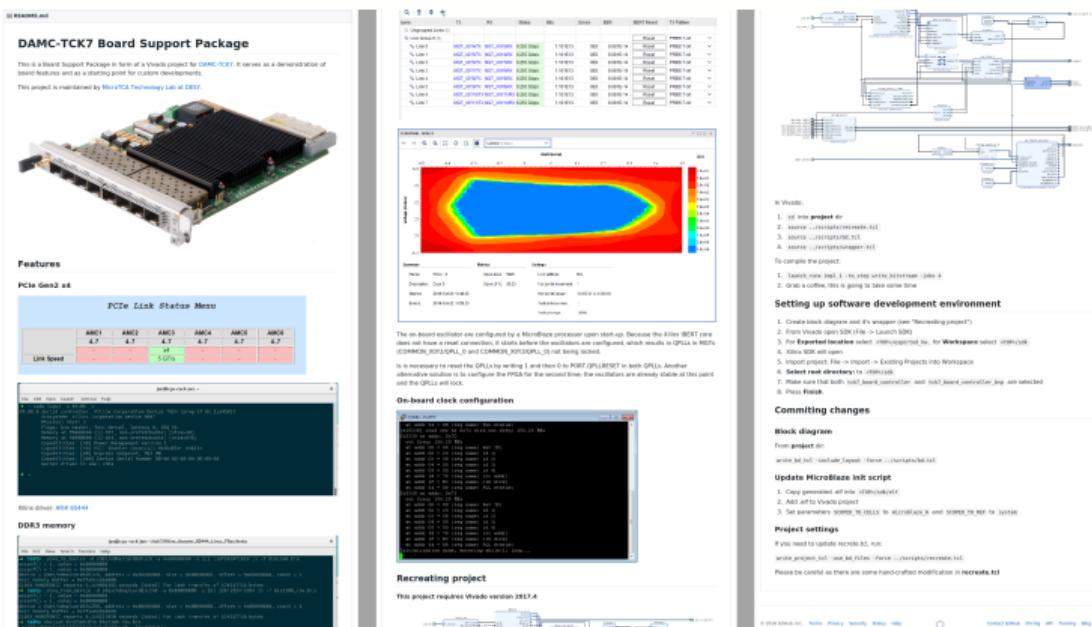
Missing on the image: DDR3 controller, DMA and PCIe interface



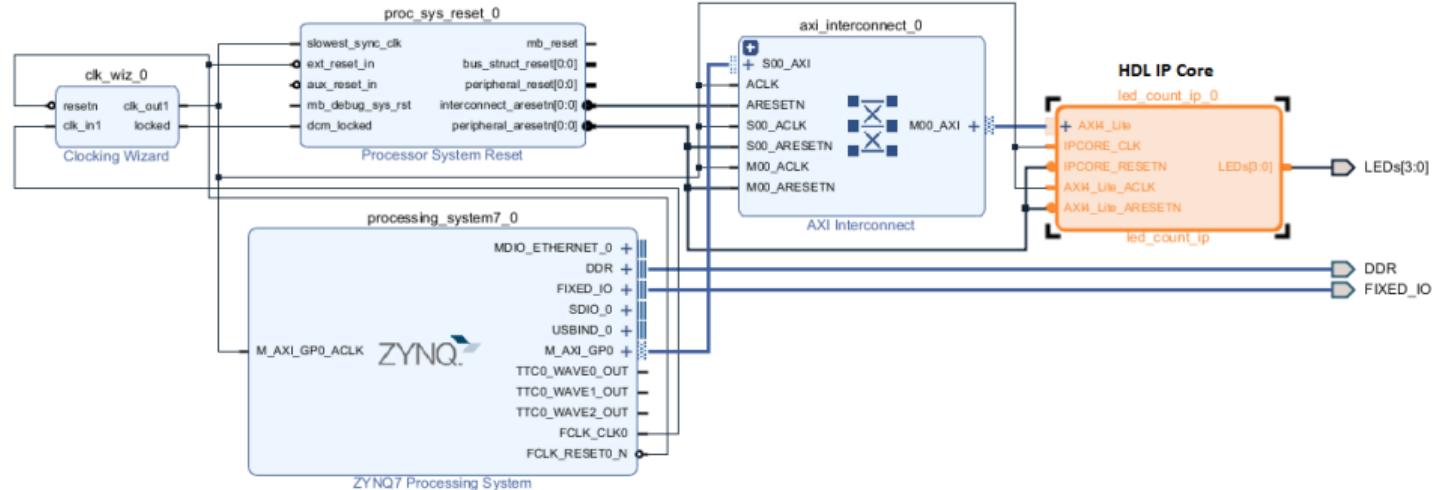
Talk on Thursday at 9:45 by S. Stubbe:  
"Implementation of GigE Vision standard and applications in MicroTCA"

## Example 3: Board Support Package for TCK7

Available at: <https://github.com/MicroTCA-Tech-Lab/damc-tck7-fpga-bsp>  
User can transfer the data from the CPU board to the DDR3 on DAMC-TCK7 board



# Basics



from <https://www.mathworks.com/help/hdlcoder/examples/define-and-register-custom-board-and-reference-design-for-zynq-workflow.html>

- ▶ (Almost) all Xilinx IPs use AMBA AXI protocols for interfacing between each other
- ▶ Provide good interface for testing

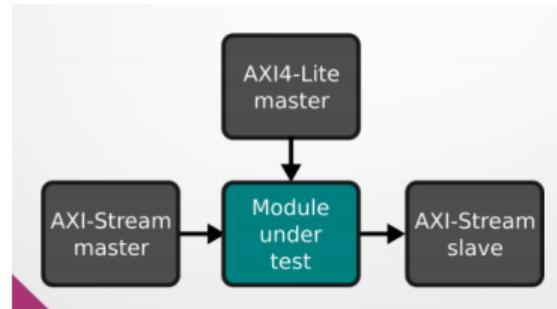


image from:

<https://indico.desy.de/indico/event/15974/session/6/contribution/15/material/slides/0.pdf>

another example (with Vivado):

[https://github.com/j-marjanovic/chisel-stuff/tree/master/example-3-vivado-ip/vivado\\_tb\\_project](https://github.com/j-marjanovic/chisel-stuff/tree/master/example-3-vivado-ip/vivado_tb_project)

- ▶ Save documentation and communication efforts

Very common in Xilinx IPs for data streams  
E.g.: FFT, 1/10/40/100 Gigabit Ethernet, high-perf PCIe

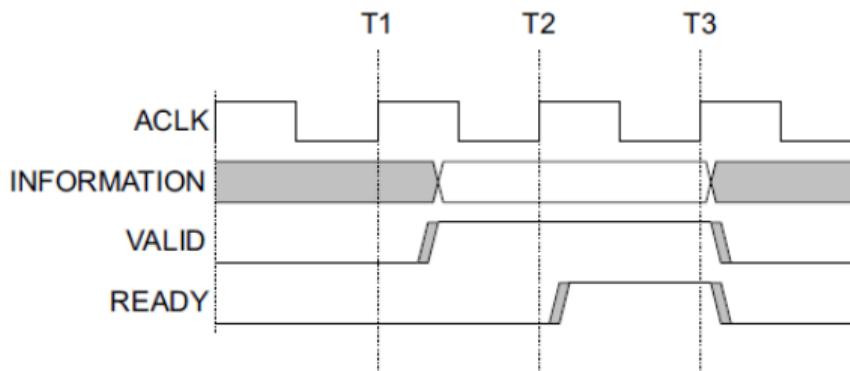


Figure A3-2 VALID before READY handshake

from AMBA® AXI™ and ACE™ Protocol Specification, available at

<https://www.arm.com/products/silicon-ip-system/embedded-system-design/amba-specifications>

Very common in Xilinx IPs for access to control/status registers

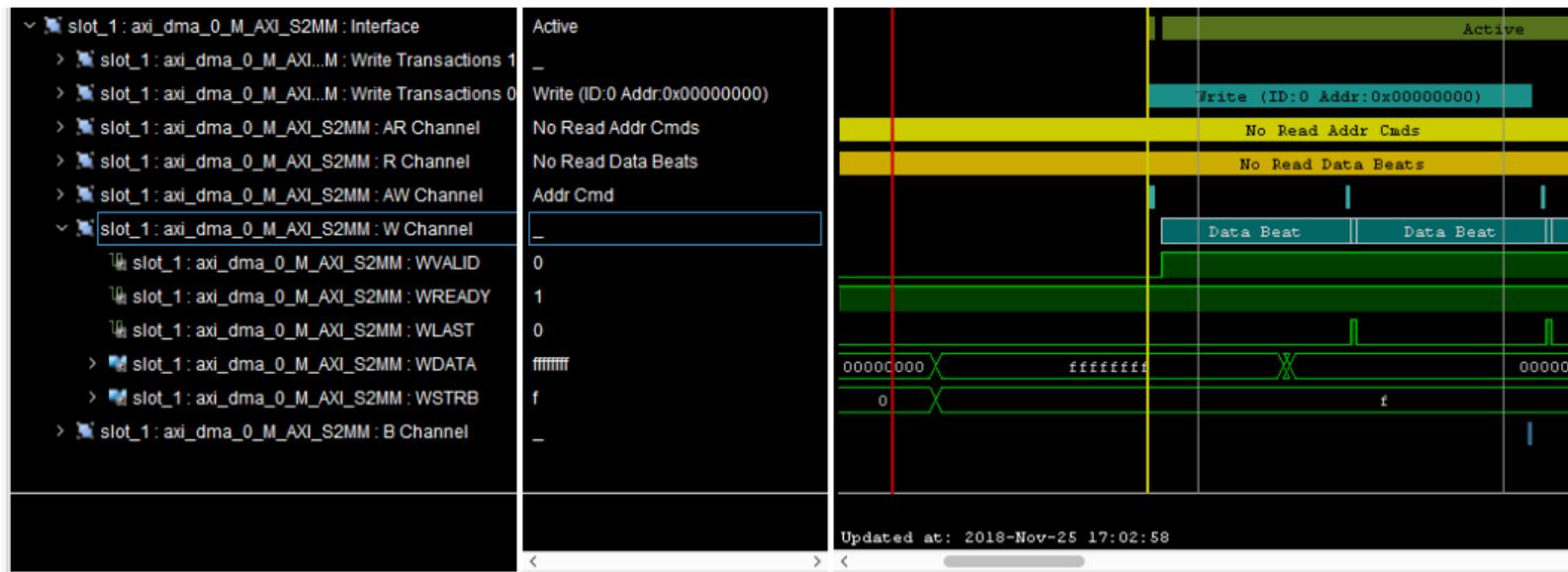
Composed of 5 independent channels:

- ▶ read address ( $M \rightarrow S$ )
- ▶ read data ( $S \rightarrow M$ )
- ▶ write address ( $M \rightarrow S$ )
- ▶ write data ( $M \rightarrow S$ )
- ▶ write response ( $S \rightarrow M$ )

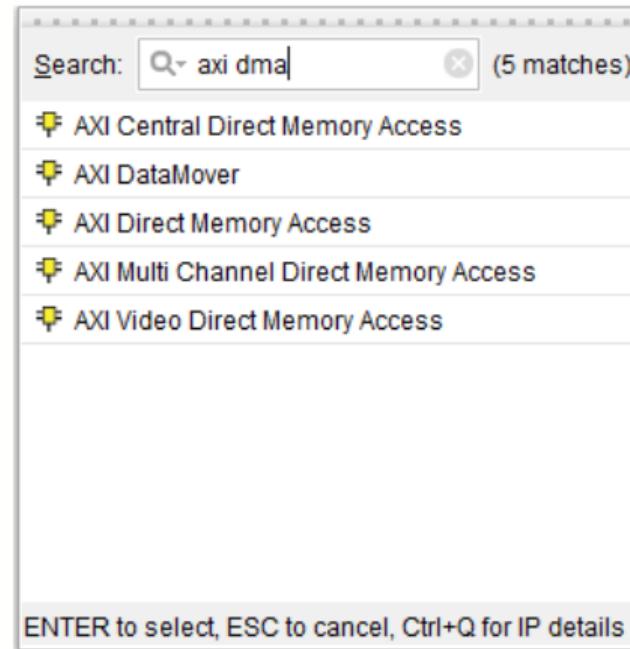
Three ways to implement AXI4-Lite interface:

- ▶ Vivado Wizard (Tools/Create and Package New IP)
- ▶ write the interface yourself (2 state machines)
- ▶ Vivado HLS (in C and C++, bundle all inputs/outputs into one interface)

Similar to AXI4-Lite, but support burst transfers (typically 16, 32 data beats per burst).  
Very common in Xilinx IP for high-throughput interfaces (DDR3/DDR4, PCIe DMA, DMA)



Vivado provides a large variety of AXI DMA:



## ► Altera mSGDMA

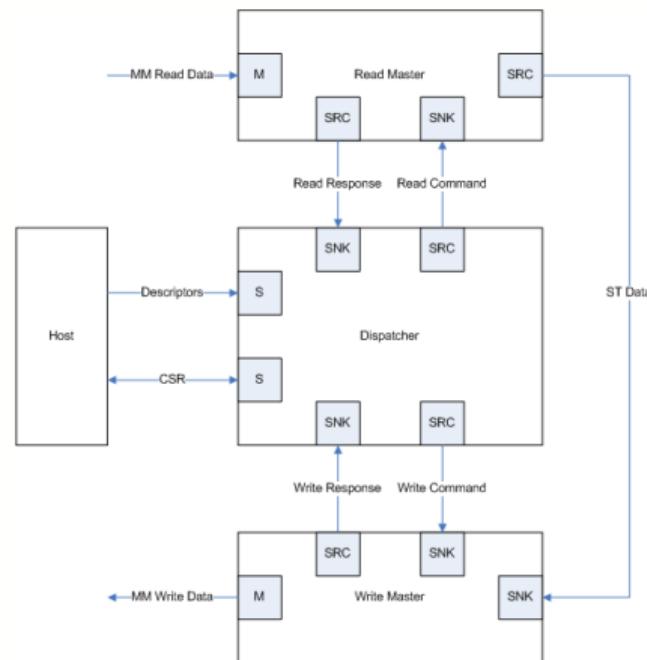
[https://fpgawiki.intel.com/wiki/Modular\\_SGDMA](https://fpgawiki.intel.com/wiki/Modular_SGDMA)

- IMHO the nicest SW interface
- uses FIFOs for commands and responses

## ► home-made DMA

- necessary in Virtex-5 times
- weeks to implements
- weeks to test all edge cases

## ► with Vivado HLS



Not really needed for this talk, but nice to understand what is going on behind the scenes.

Xilinx XDMA takes AXI transactions and creates PCIe TLP packets; these are passed to the Xilinx PCIe IP.

Figure 2-1: Non-Posted Read Transaction Protocol

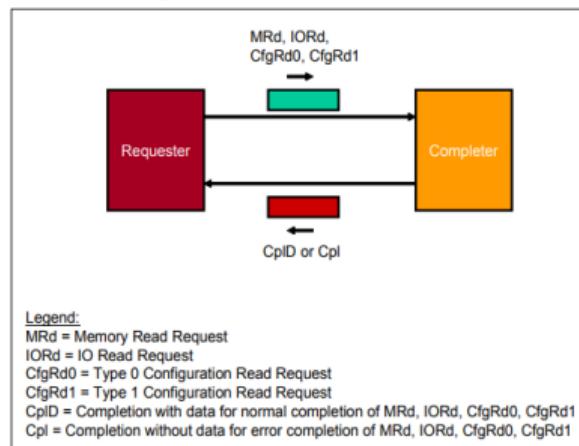
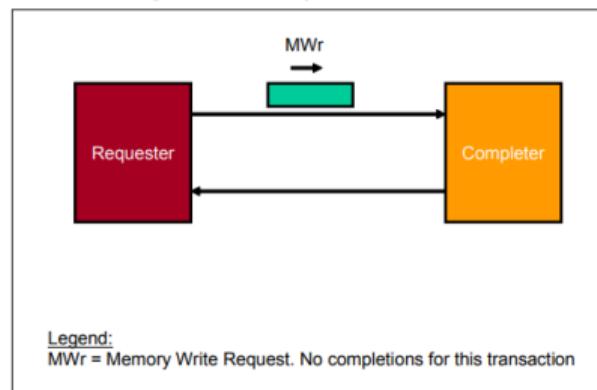


Figure 2-4: Posted Memory Write Transaction Protocol



from <https://www.mindshare.com/files/ebooks/PCI%20Express%20System%20Architecture.pdf>

# Demo 1

Available at <https://github.com/MicroTCA-Tech-Lab/damc-tck7-fpga-bsp>

The screenshot shows a GitHub repository page for "damc-tck7-fpga-bsp". The page includes the MicroTCA Technology Lab logo, location (Hamburg, Germany), and website link. The navigation bar shows 1 repository, 3 people, 0 teams, 0 projects, and settings. The repository details show it's a pinned repository for DAMC-TCK7, written in Tcl, and has 2 stars.

MicroTCA TECHNOLOGY LAB

Hamburg, Germany <https://techlab.desy.de>

Repositories 1 People 3 Teams 0 Projects 0 Settings

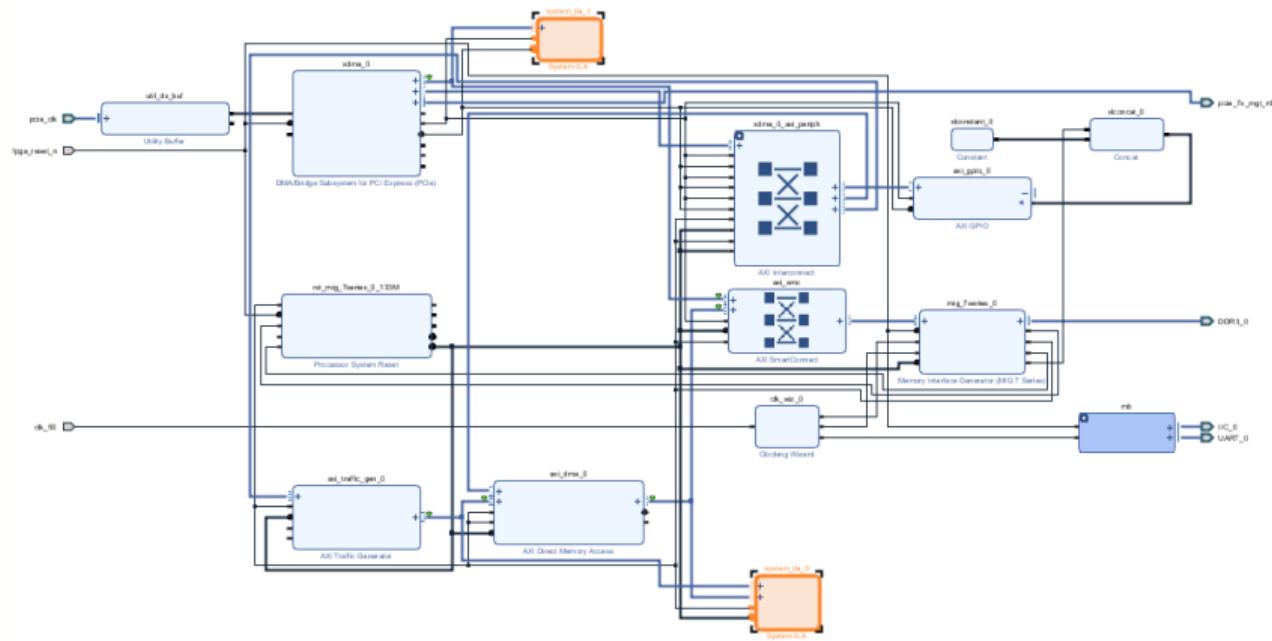
Pinned repositories

damc-tck7-fpga-bsp

Board Support Package for DAMC-TCK7

Tcl ★ 2

## Vivado IP Integrator overview

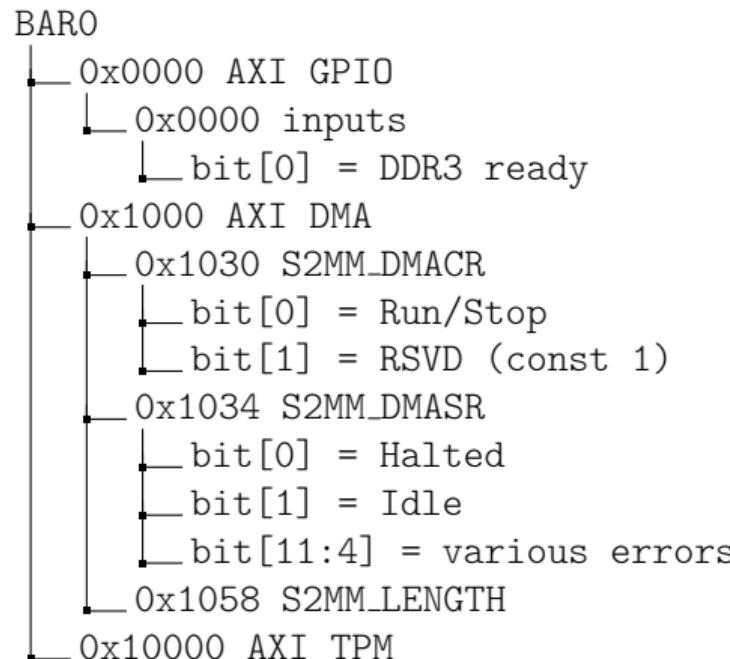


Two Integrated Logic Analyzers (ILAs) on important AXI connections are highlighted

At this point our design includes:

- ▶ DMA/Bridge Subsystem for PCI Express (from BSP)
- ▶ Memory Interface Generator (MIG) (from BSP)
- ▶ AXI DMA
- ▶ AXI Traffic Generator

Three modules accessible on AXI4-Lite Master (control and status interfaces):



Linux driver provided by Xilinx (AR# 65444) needs to be compiled and loaded.

```
> sudo lspci -s 08:00 -v
08:00.0 Serial controller: Xilinx Corporation Device 7024 (prog-if 01 [16450])
    Subsystem: Xilinx Corporation Device 0007
    Physical Slot: 3
    Flags: bus master, fast devsel, latency 0, IRQ 35
    Memory at ff500000 (32-bit, non-prefetchable) [size=1M]
    Memory at ff600000 (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
```

Use tool provided by Xilinx (`reg_rw`) to "peek" and "poke" registers.

## 1. Check DDR3 status

```
./reg_rw /dev/xdma/card0/user 0x0 w
```

Expected value: 0x1

## 2. Check if DMA is in idle

```
./reg_rw /dev/xdma/card0/user 0x1034 w
```

Expected value: 0x1

Get the data from AXI Traffic Generator into on-board DDR3 memory

## 1. Enable DMA

```
./reg_rw /dev/xdma/card0/user 0x1030 w 0x1
```

## 2. Check if DMA is not idle anymore

```
./reg_rw /dev/xdma/card0/user 0x1034 w
```

## 3. Start transfer

```
./reg_rw /dev/xdma/card0/user 0x1058 w 0x2000
```

## 4. Start AXI traffic gen

```
./trig.sh
```

We use the following script to trigger the AXI Traffic Generator.

```
#!/bin/bash
# script to configure and trigger AXI Traffic generator

AXI_TG_BASE=0x10000

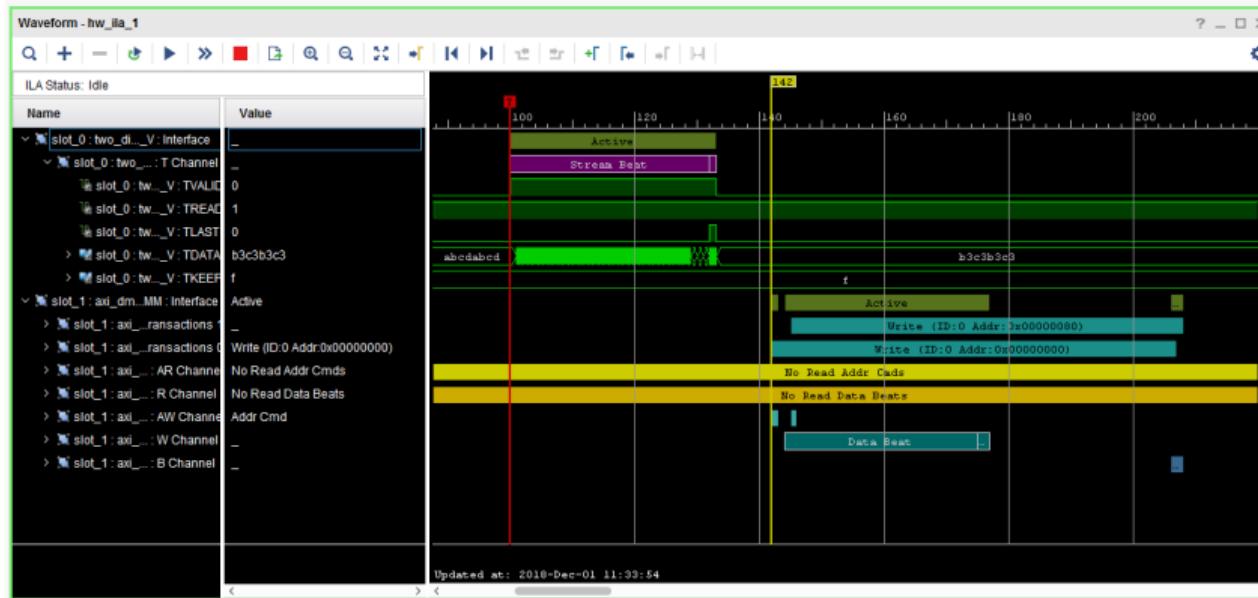
AXI_TG_ADDR_ST_CTRL=0x30
AXI_TG_ADDR_ST_CONF=0x34
AXI_TG_ADDR_TR_LEN=0x38

# set delay between packets
reg_rw /dev/xdma/card0/user $((AXI_TG_BASE + AXI_TG_ADDR_ST_CONF)) w 0x10000

# set packet len, number of packets
reg_rw /dev/xdma/card0/user $((AXI_TG_BASE + AXI_TG_ADDR_TR_LEN)) w 0x101ff

# trigger the traffic generation
reg_rw /dev/xdma/card0/user $((AXI_TG_BASE + AXI_TG_ADDR_ST_CTRL)) w 1
```

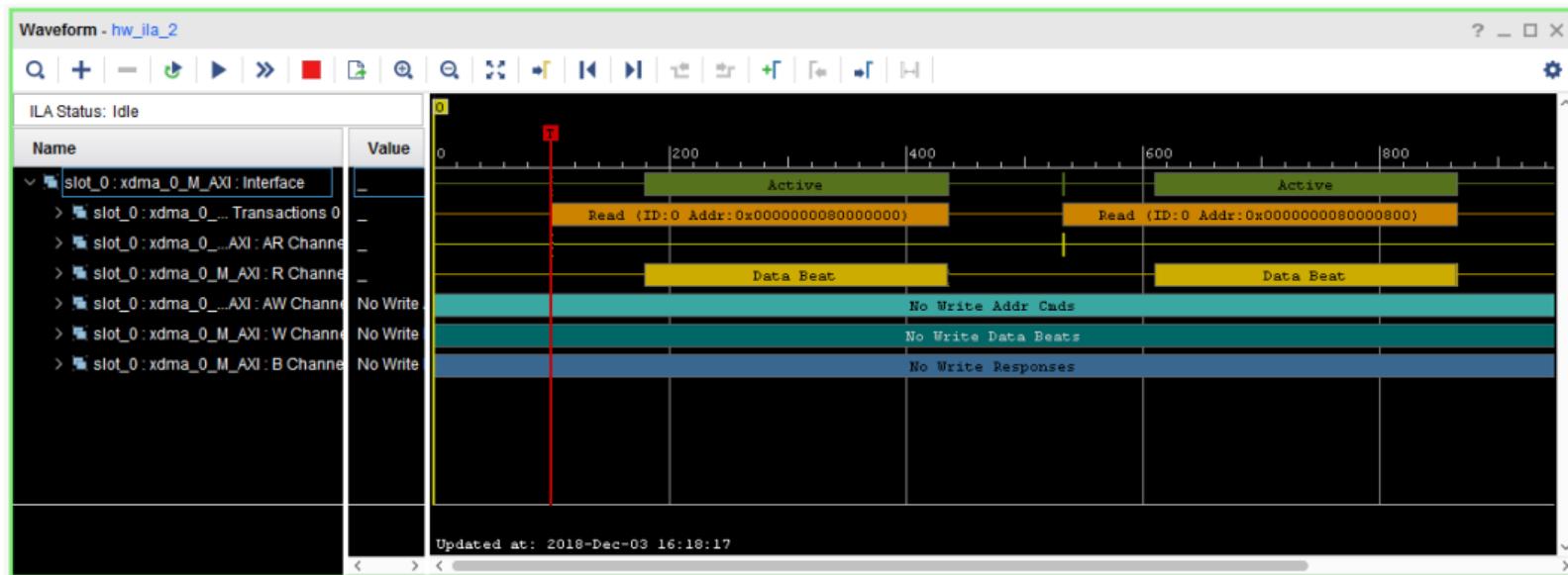
If we trigger ILA on AXI4-Stream VALID and READY being both '1' we are able to capture the DMA transfer into on-board DDR3 memory.



Data is not in on-board (DAMC-TCK7 board) DDR3 memory, we need to use the second DMA to bring it into CPU. We use Xilinx tool (`dma_from_device`) to transfer the data. The tool itself calls `seek()` and `read()` system calls to communicate to Linux driver to start the transfer.

```
./dma_from_device --device /dev/xdma/card0/c2h0 --address 0x80000000 \
    --size 0x200 --file out.bin
sscanf() = 1, value = 0x80000000
sscanf() = 1, value = 0x00000200
device = /dev/xdma/card0/c2h0, address = 0x80000000, size = 0x00000200,
    offset = 0x00000000, count = 1
host memory buffer = 0x56040427a000
CLOCK_MONOTONIC reports 0.000051682 seconds (total) for last transfer of 512
    bytes
```

Trigger on ARREADY and ARVALID to xdma/M\_AXI:



(Note the incrementing address)

We inspect the data transferred with `dma_from_device` and note the pattern which was generated by AXI Traffic Generator (2x16-bit values).

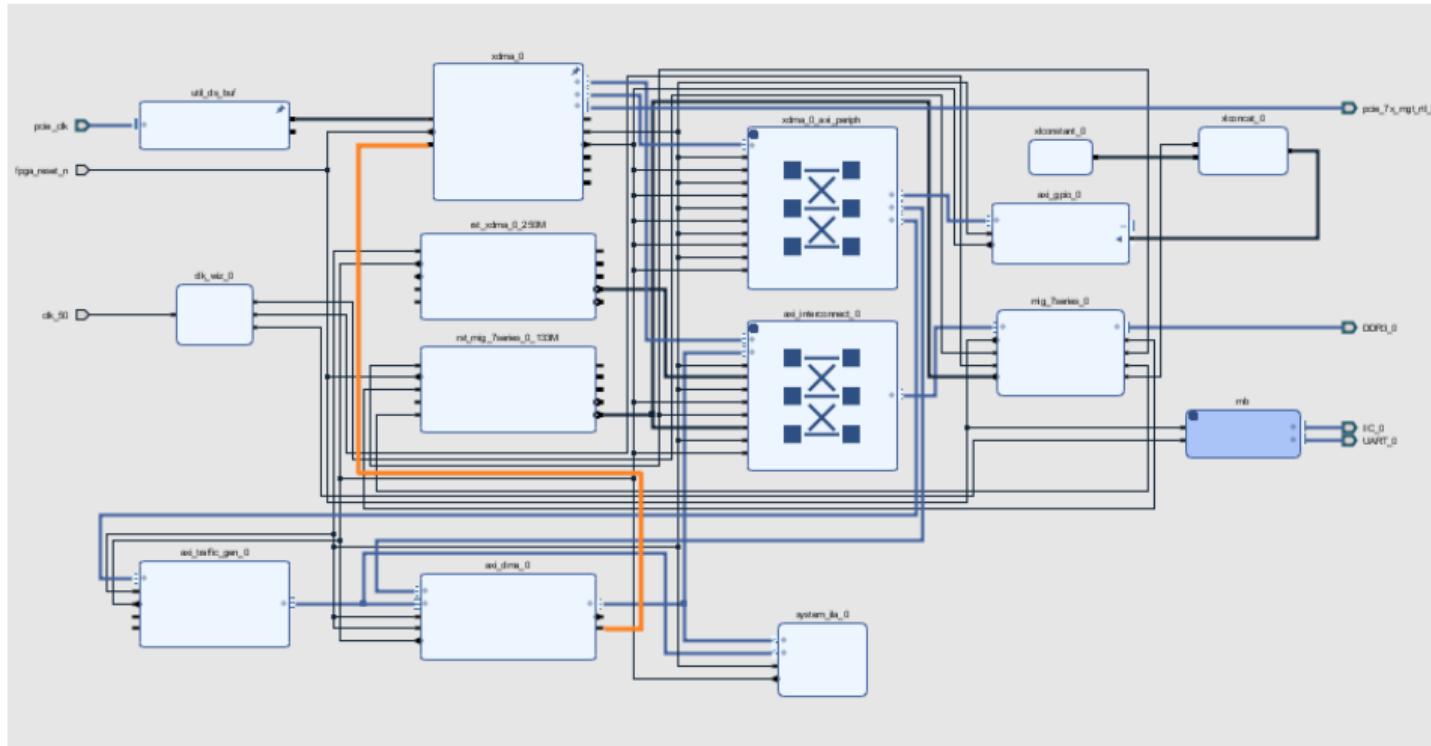
```
jan@cpu-110008: ~/dma_demo/driver/Xilinx_Answer_65444_Linux_Files/tests
File Edit View Search Terminal Help
→ tests hexdump -C jan.bin
00000000 cd ab cd ab e6 d5 e6 d5 f3 ea f3 ea 79 f5 79 f5 | .....y.y.|  
00000010 bc 7a bc 7a 5e bd 5e bd af 5e af 5e 57 af 57 af |.z.z^.^..^.^W.W.|  
00000020 ab d7 ab d7 d5 eb d5 eb ea 75 ea 75 f5 3a f5 3a |.....u.u.:.|  
00000030 7a 9d 7a 9d bd 4e bd 4e 5e a7 5e a7 af d3 af d3 |z.z..N.N^.^...|  
00000040 d7 e9 d7 e9 eb f4 eb f4 75 fa 75 fa 3a fd 3a fd |.....u.u.:.|  
00000050 9d 7e 9d 7e 4e 3f 4e 3f a7 1f a7 1f d3 0f d3 0f |.~.~N?N?....|  
00000060 e9 87 e9 87 f4 c3 f4 c3 fa e1 fa e1 fd f0 fd f0 |.....|  
00000070 7e 78 7e 78 3f 3c 3f 3c 1f 9e 1f 9e 0f cf 0f cf |~x-x?<?<.....|  
00000080 87 67 87 67 c3 b3 c3 b3 e1 59 e1 59 f0 ac f0 ac |.g.g....Y.Y....|  
00000090 78 d6 78 d6 3c eb 3c eb 9e 75 9e 75 cf 3a cf 3a |x.x.<.<..u.u.:.|  
000000a0 67 9d 67 9d b3 4e b3 4e 59 a7 59 a7 ac d3 ac d3 |g.g..N.NY.Y....|  
000000b0 d6 e9 d6 e9 eb 74 eb 74 75 ba 75 ba 3a dd 3a dd |.....t.tu.u.:.|  
000000c0 9d 6e 9d 6e 4e b7 4e b7 a7 5b a7 5b d3 2d d3 2d |.n.nN.N..[.|-.-|  
000000d0 e9 96 e9 96 74 4b 74 4b ba a5 ba a5 dd 2d dd 2d |....tKtK.....|  
000000e0 6e 69 6e 69 b7 b4 b7 b4 5b 5a 5b 5a 2d ad 2d ad |nini....[Z[Z-...|  
000000f0 96 d6 96 d6 4b eb 4b eb a5 75 a5 75 d2 ba d2 ba |....K.K..u.u....|  
00000100 69 dd 69 dd b4 6e b4 6e 5a b7 5a b7 ad 5b ad 5b |i.i..n.nZ.Z..[.|  
00000110 d6 2d 6d 2d eb 16 eb 16 75 8b 75 8b ba 45 ba 45 |.....u.u..E.E|  
00000120 dd a2 dd a2 6e d1 6e d1 b7 68 b7 68 5b b4 5b b4 |....n.n..h.h[.|  
00000130 2d da 2d da 16 6d 16 6d 8b 36 8b 36 45 9b 45 9b |....m.m.6.6E.E.|  
00000140 a2 cd a2 cd d1 66 d1 66 68 33 68 33 b4 99 b4 99 |.....f.fh3h3....|  
00000150 da 4c da 4c 6d a6 6d a6 36 d3 36 d3 9b e9 9b e9 |.L.Lm.m.6.6.....|  
00000160 cd 74 cd 74 66 3a 66 3a 33 9d 33 9d 99 4e 99 4e |.t.tf:f:3.3..N.N|
```

# Demo 2

A more concrete application:

1. AXI DMA will always be ready to accept data
2. when an entire packet is received, AXI DMA raises interrupt request
3. The user-space application monitors the interrupt request (When the AXI DMA has finished storing data into on-board DDR3, the user-space application is alerted)
4. The user-space application then copies the data from on-board DDR3 to its internal buffer
5. The content of the buffer is compared to the SW implementation of LFSR
6. A software script is used to re-trigger AXI Traffic Generator

# Adding interrupt requests



## **LFSR Implementation used for Random Generation**

The below linear feed-back shift register (LFSR) is used for random address or data generation:

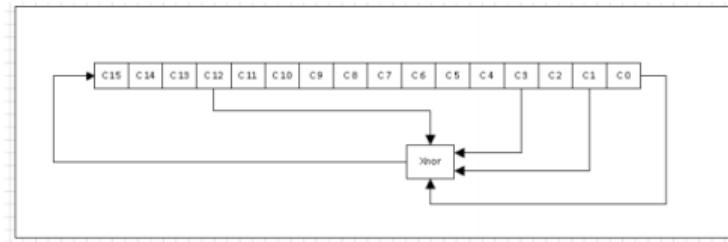


Figure 2-5: LFSR

Initially, the C0 to C15 flops are loaded with the input seed value. Later, it behaves as a shift register as per the architecture.

from Xilinx AXI Traffic Generator v3.0

[https://www.xilinx.com/support/documentation/ip\\_documentation/axi\\_traffic\\_gen/v3\\_0/pg125-axi-traffic-gen.pdf](https://www.xilinx.com/support/documentation/ip_documentation/axi_traffic_gen/v3_0/pg125-axi-traffic-gen.pdf)

## Content of App.cpp

```
/// implements LFSR as described in "AXI Traffic Generator v3.0"
class AxiTrafficGenLfsr;

/// interface to Xilinx DMA driver
class XdmaIsr;

/// general interface for register access (using wr32 and rd32 methods)
class XdmaRegAccess;

/// AXI DMA controller
class HwAxiDmaControl;

/// data consumer
class DataConsumer;

/// application capturing data and checking data integrity
int main();
```

# Output from application

```
./app
File Edit View Search Terminal Tabs Help
./app  x  jan@jan-Vi...  x  telnet MCH...  x  jan@jan-Vi...  x  jan@cpu-1...  x  +  ▾
→ app git:(master) X ./app
HwAxiDmaControl: status after run bit set = 0
XdmaIsr: received an event, value: 1
HwAxiDmaControl: status = 1002
HwAxiDmaControl: status = 2
DataConsumer::check, bytes_len = 2048
at  0 expect abcd, got: abcd | abcd |
at  1 expect d5e6, got: d5e6 | d5e6 |
at  2 expect eaf3, got: eaf3 | eaf3 |
at  3 expect f579, got: f579 | f579 |
at  4 expect 7abc, got: 7abc | 7abc |
at 1fc expect 33d0, got: 33d0 | 33d0 |
at 1fd expect 19e8, got: 19e8 | 19e8 |
at 1fe expect 8cf4, got: 8cf4 | 8cf4 |
at 1ff expect c67a, got: c67a | c67a |
DataConsumer: check, ok = 400, tot = 400 (100%)
XdmaIsr: received an event, value: 1
HwAxiDmaControl: status = 1002
HwAxiDmaControl: status = 2
DataConsumer::check, bytes_len = 2048
at  0 expect e33d, got: e33d | e33d |
at  1 expect f19e, got: f19e | f19e |
at  2 expect 78cf, got: 78cf | 78cf |
at  3 expect bc67, got: bc67 | bc67 |
at  4 expect 5e33, got: 5e33 | 5e33 |
at 1fc expect 2eb5, got: 2eb5 | 2eb5 |
at 1fd expect 175a, got: 175a | 175a |
at 1fe expect bad, got: bad | bad |
at 1ff expect 85d6, got: 85d6 | 85d6 |
DataConsumer: check, ok = 400, tot = 400 (100%)
```

# Conclusion

- ▶ Xilinx provides a lot of infrastructure to help start the development
- ▶ MTCA Tech Lab is committed to provide open-source Board Support Packages for new boards (DAMC-TCK7, Zynq MPSoC FMC+ carrier, ...)
- ▶ Linux drivers are also provided by Xilinx  
(<https://www.xilinx.com/support/answers/65444.html>)  
Servers are a huge market and require better support for drivers/interfaces. Xilinx DMA provides everything out of the box.

## 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

<https://techlab.desy.de>

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

# Backup slides

Before we start with example, let's check if everything is OK with the system:

- FPGA programmed
- PCIe link is up (**show\_link\_state**, **show\_ekey**, front-panel LED)
- PCIe device enumerated (**lspci**)
- Driver loaded (**lspci -v**)

```
system_i/xdma_0/inst/system_xdma_0_0_pcie2_to_pcie3_wrapper_i/
  pcie2_ip_i/s_axis_tx_tdata[*]
  pcie2_ip_i/s_axis_tx_tuser[*]
  pcie2_ip_i/s_axis_tx_tkeep[*]
  pcie2_ip_i/s_axis_tx_tvalid
  pcie2_ip_i/s_axis_tx_tready
  pcie2_ip_i/s_axis_tx_tlast
  pcie2_ip_i/m_axis_rx_tdata[*]
  pcie2_ip_i/m_axis_rx_tuser[*]
  pcie2_ip_i/m_axis_rx_tlast
  pcie2_ip_i/m_axis_rx_tready
  pcie2_ip_i/m_axis_rx_tvalid
```

Inspired by [drivers/net/wireless/broadcom/brcm80211/brcmfmac/pcie.c#L486](https://github.com/broadcom/linux-brcm80211/blob/master/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--;
    }
}
```

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;
}
```



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)
```

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)
```

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)
```