Innovative data processing methods on field programmable gate arrays (FPGAs)

Dr Grzegorz Korcyl

Zakład Technologii Informatycznych

Uniwersytet Jagielloński, Kraków

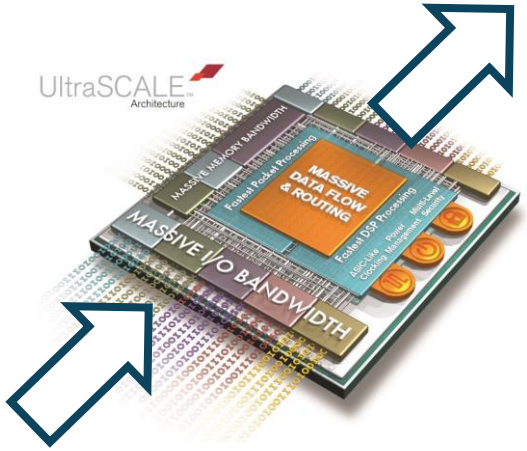
15 May 2026, Kraków

FPGA

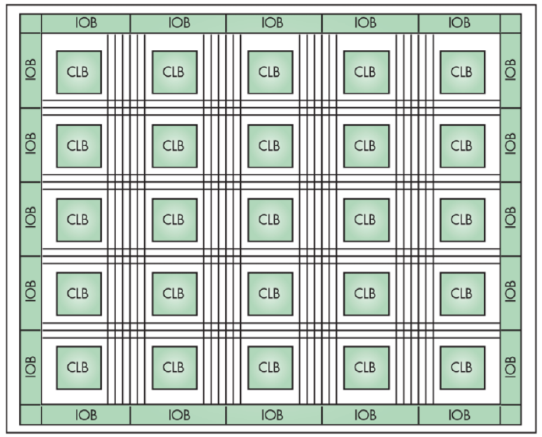
FPGA devices are arrays of Configurable Logic Blocks (GATE ARRAYS)

that one can reconfigure (PROGRAMMABLE)

at any time and place (FIELD)

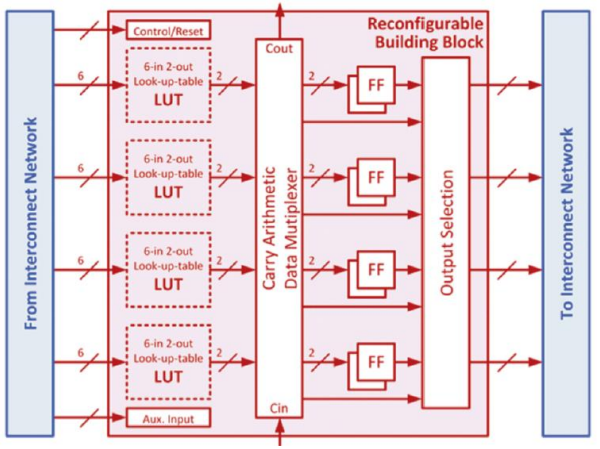


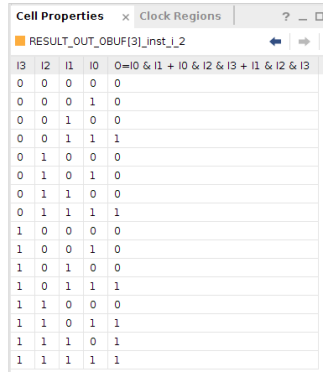
Arrays of Configurable Logic Blocks



www.electronicdesign.com

Basic Configurable Logic Block



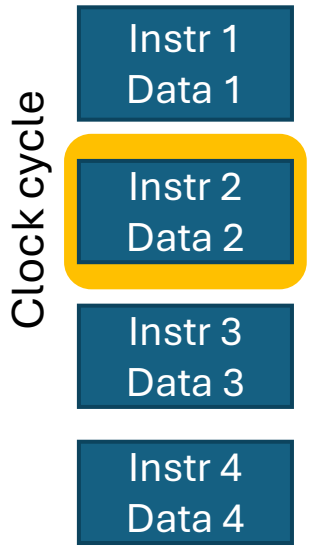


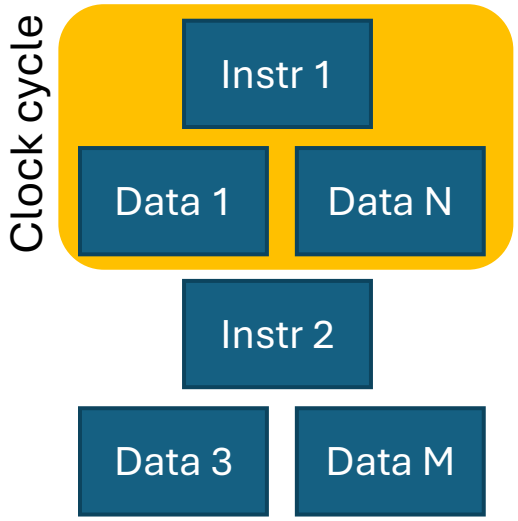
Cell Properties | Clock Regions

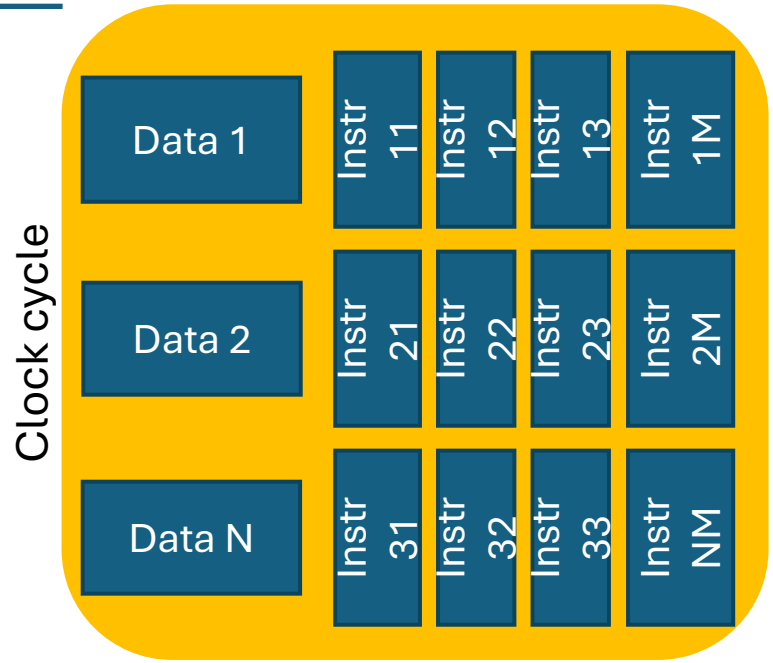
RESULT_OUT_OBUF[3]_inst_i_2

I3	I2	I1	I0	O=I0 & I1 + I0 & I2 & I3 + I1 & I2 & I3
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

CPU vs GPU vs FPGA





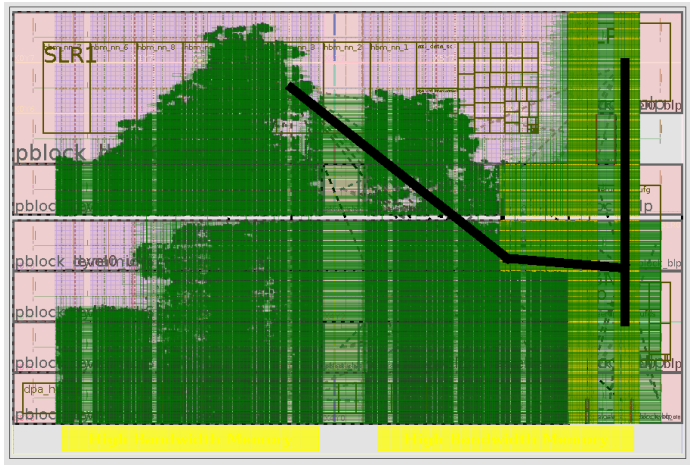


- CPU
  - Single Instruction Single Data per core
  - Fixed instruction set
  - Multiple cores
  - High clock freq.
  - Operating system
- GPU
  - Single Instruction Multiple Data
  - Fixed instruction set
  - High clock freq.
  - Memory access
  - Accelerates CPU
- FPGA
  - Flexible architecture
  - Massive parallelism
  - Streamlined processing
  - Low clock freq.
  - Instant memory access
  - Standalone platforms

FPGA

	Foundation						
Device Name	VU3P	VU5P	VU7P	VU9P	VU11P	VU13P	VU19P
System Logic Cells (K)	862	1,314	1,724	2,586	2,835	3,780	8,938
CLB Flip-Flops (K)	788	1,201	1,576	2,364	2,592	3,456	8,172
CLB LUTs (K)	394	601	788	1,182	1,296	1,728	4,086
Max. Dist. RAM (Mb)	12.0	18.3	24.1	36.1	36.2	48.3	58.4
Total Block RAM (Mb)	25.3	36.0	50.6	75.9	70.9	94.5	75.9
UltraRAM (Mb)	90.0	132.2	180.0	270.0	270.0	360.0	90.0
DSP Slices	2,280	3,474	4,560	6,840	9,216	12,288	3,840
Peak INT8 DSP (TOP/s)	7.1	10.8	14.2	21.3	28.7	38.3	10.4
PCIe® Gen3 x16	2	4	4	6	3	4	0
PCIe Gen3 x16/Gen4 x8 / CCIX[1]	–	–	–	–	–	–	8
150G Interlaken	3	4	6	9	6	8	0
100G Ethernet w/ KR4 RS-FEC	3	4	6	9	9	12	0
Max. Single-Ended HP I/Os	520	832	832	832	624	832	1,976
Max. Single-Ended HD I/Os	0	0	0	0	0	0	96
GTY 32.75Gb/s Transceivers	40	80	80	120	96	128	80
GTM 58Gb/s PAM4 Transceivers	–	–	–	–	–	–	–
100G / 50G KP4 FEC	–	–	–	–	–	–	–

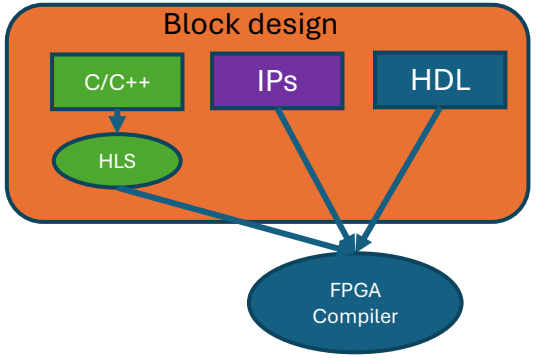
Xilinx.com



Example project:

- 150 lines of code
- 8 instances of NN kernels
- Device utilization 50% (Alveo U50)
- 500k FF
- 360k LUT

- Logic schematics
- Hardware Description Languages
- High Level Synthesis
- System builders



Example – source code

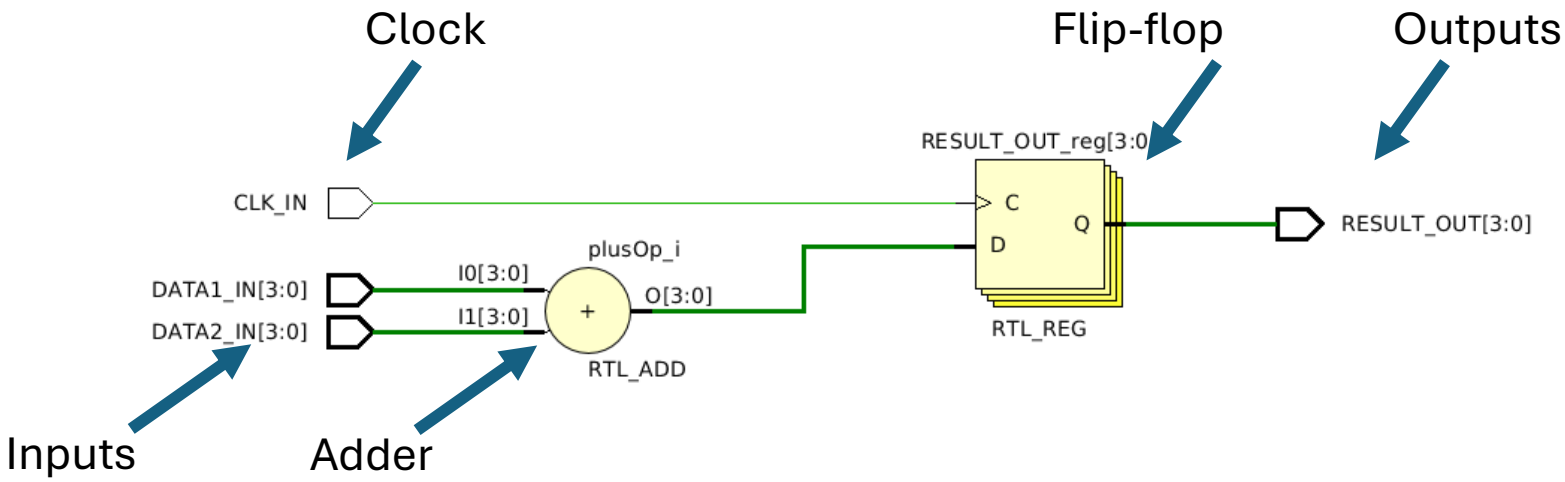
```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_unsigned.all;
4  use IEEE.NUMERIC_STD.ALL;
5  library UNISIM;
6  use UNISIM.VComponents.all;
7
8  entity top is
9  Port (
10     CLK_IN : in std_logic;
11     DATA1_IN : in std_logic_vector(3 downto 0);
12     DATA2_IN : in std_logic_vector(3 downto 0);
13     RESULT_OUT : out std_logic_vector(3 downto 0)
14 );
15 end top;
16 architecture Behavioral of top is
17 begin
18
19     process(CLK_IN)
20     begin
21         if rising_edge(CLK_IN) then
22             RESULT_OUT <= DATA1_IN + DATA2_IN;
23         end if;
24     end process;
25
26 end Behavioral;
```

TOP-level interface – matches the hardware PCB layout

Functional logic

Example – Elaborated Design

- Schematics of the logic



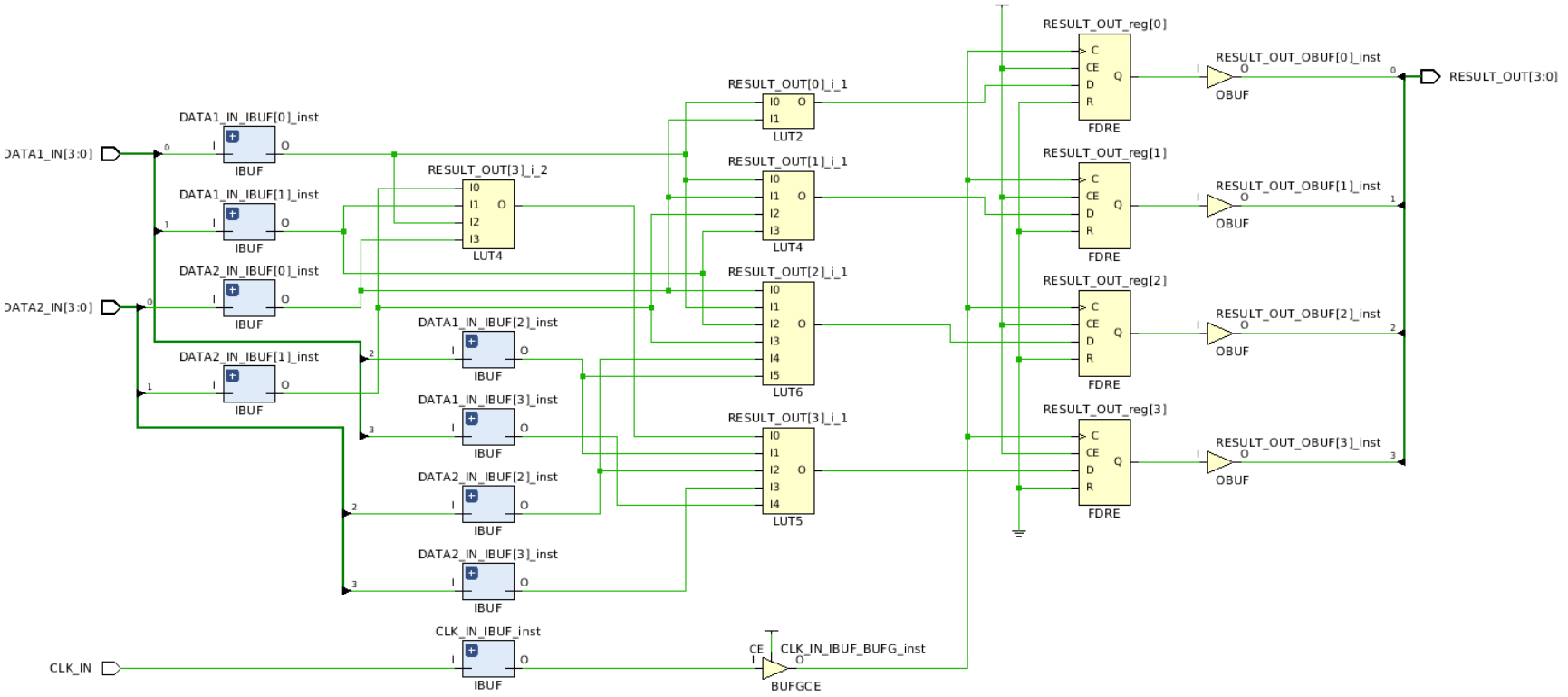
Function:

- Initiation Interval = 1 clock cycle
- Latency = 1 clock cycle

In case CLK_IN = 100 MHz
The function achieves 100 MOps/s

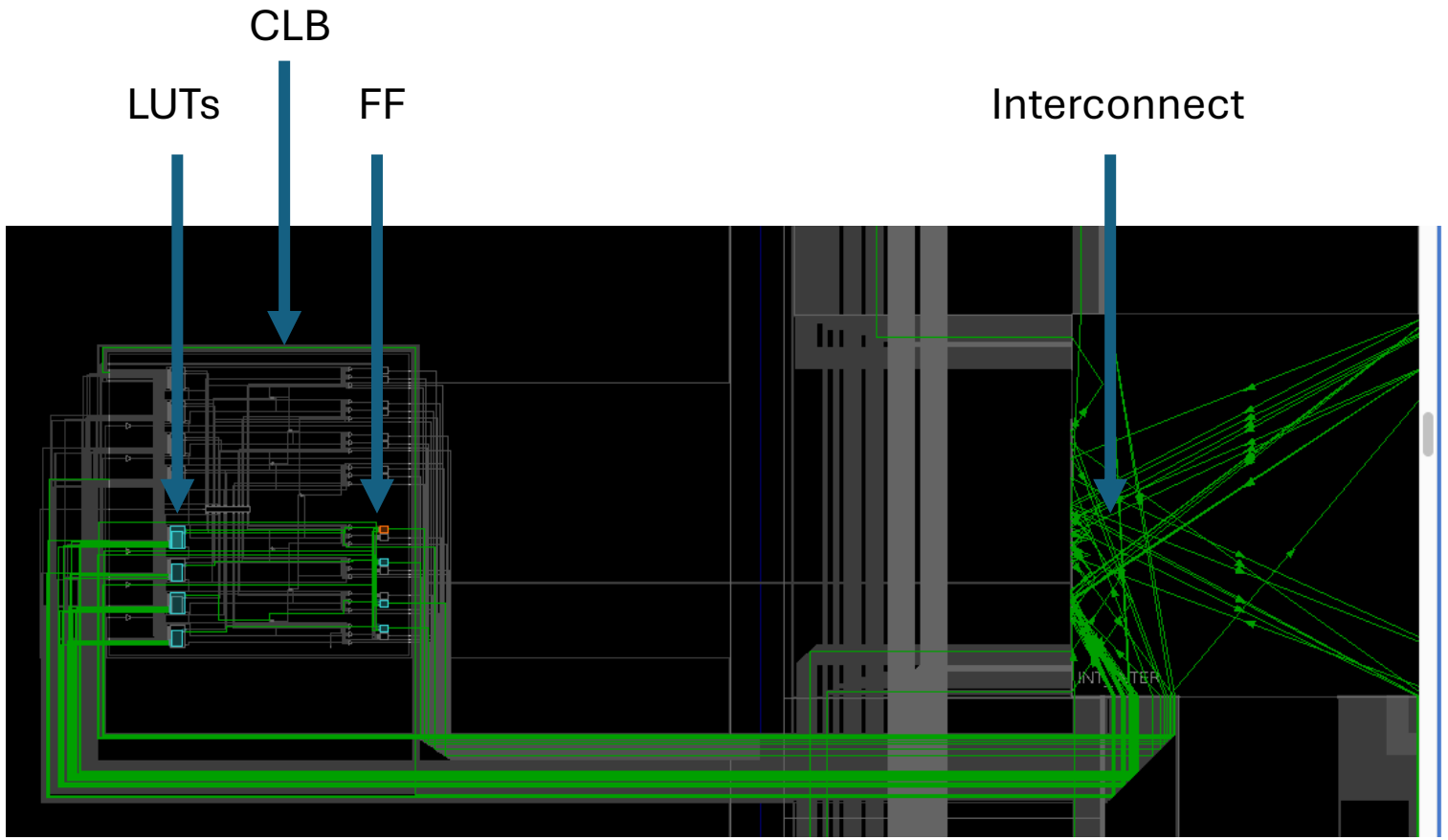
Example – synthesized design

- Schematics of the logic mapped into device resources

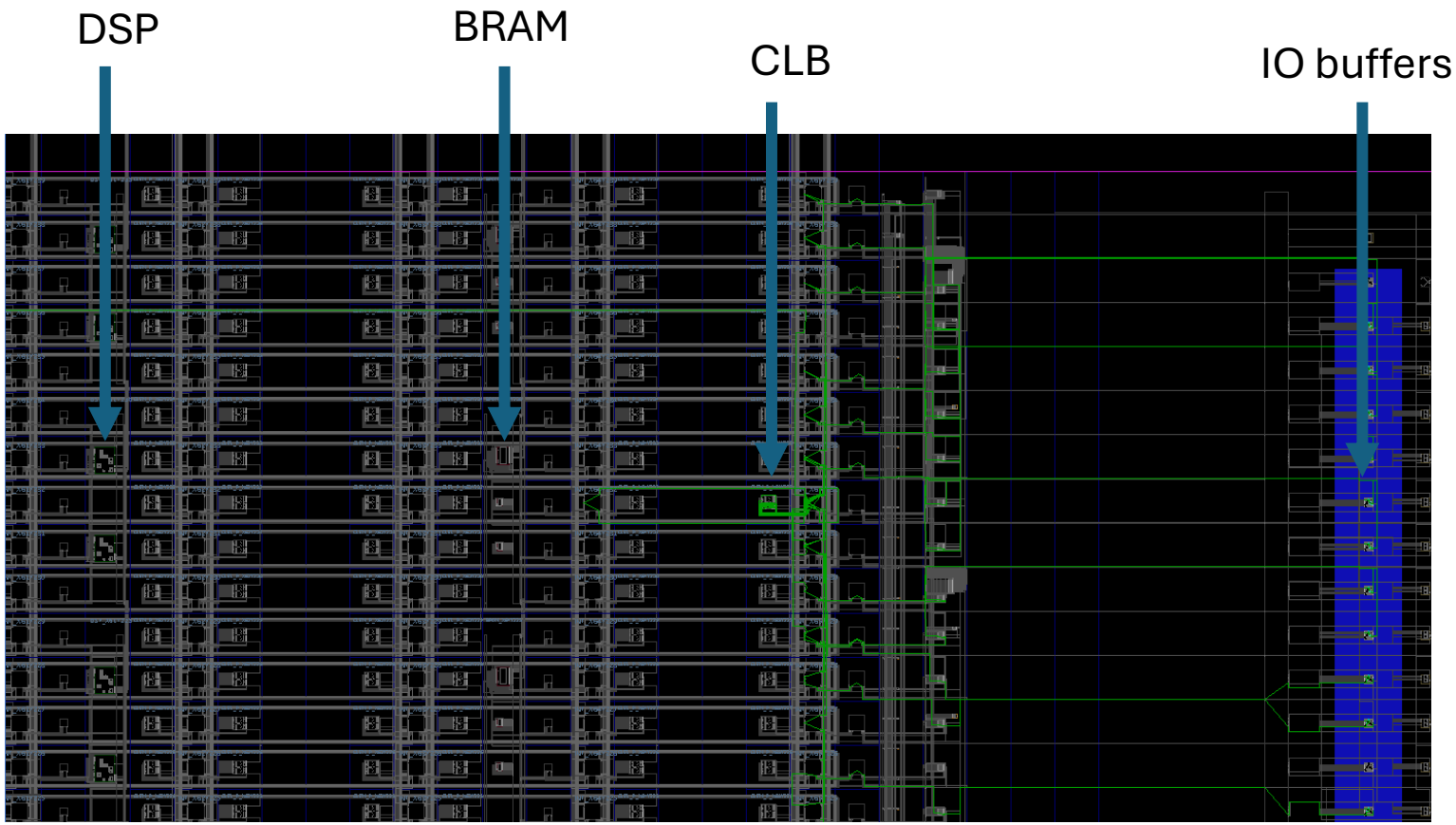


Example – implemented design

- Logic components placed within the device resources

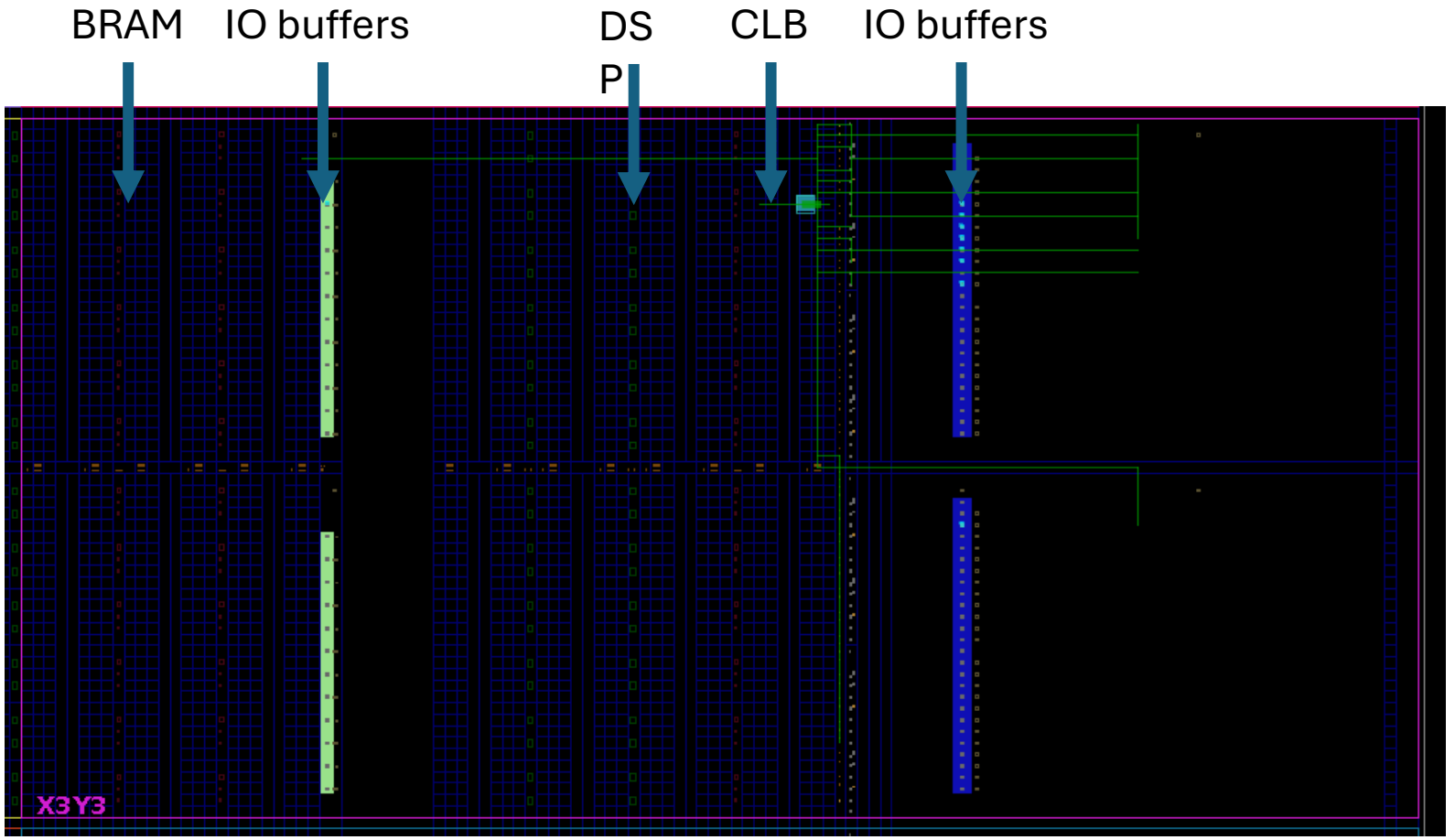


Example – implemented design

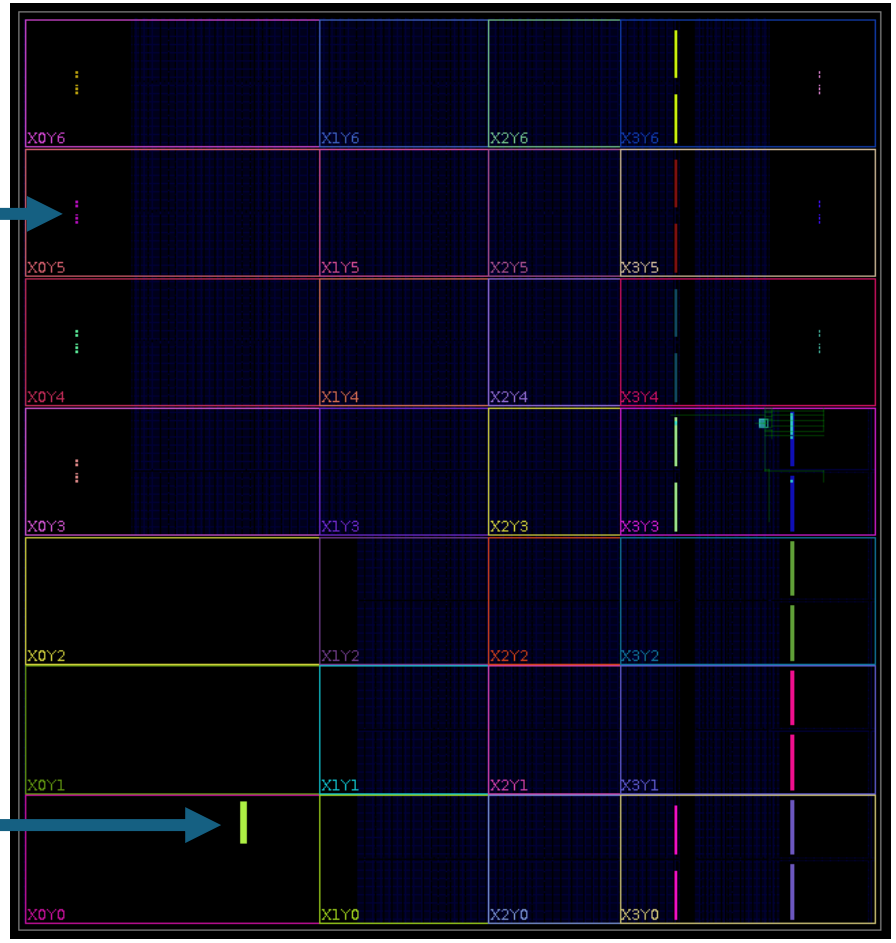


Example – implemented design

- Clock Region

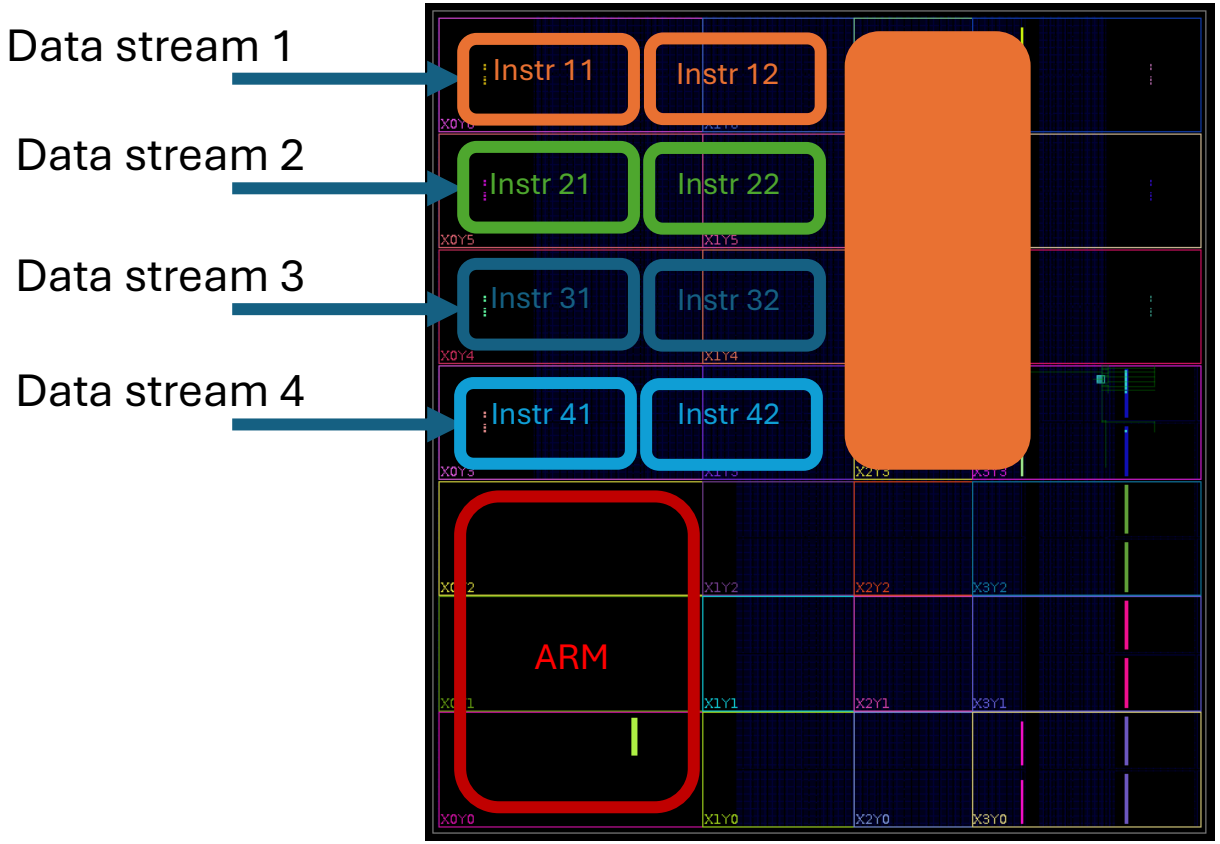


Example – implemented design



Utilization:
LUT: 4 out of 274 080
FF: 4 out of 548 169

Natural parallelism and streamlined processign



Research areas

Large scale pysics exp.

- Heterogeneous, distributed, streaming data sources
- True real-time regime
- Acceleration of online processing pipelines

High-Performance Computing Systems

- Complex algorithm acceleration
- No timing constraints
- Efficient resources usage

PET Tomography

- Uniform, distributed streaming data sources
- Continuous readout
- Full system: HW+GW+SW

Computer Networks

- Offloading packet processing
- Minimal and deterministic latencies
- Packet construction and payload manipulation

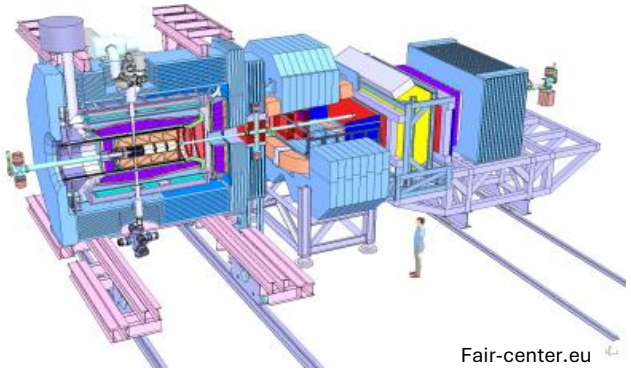
Large scale physics exp.

Data acquisition and processing systems

- Facility for Antiproton and Ion Research FAIR in Darmstadt

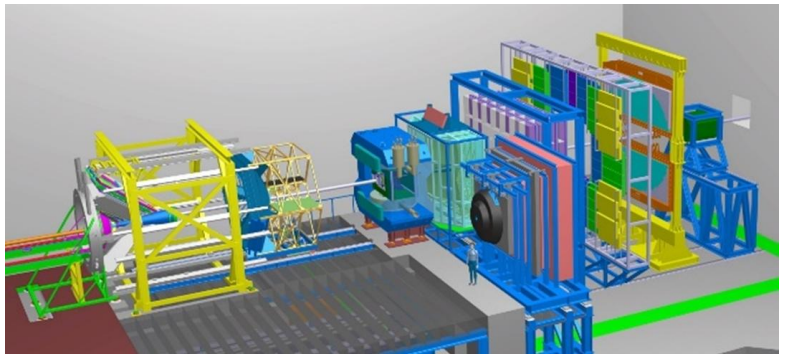
Large scale physics exp.

Data acquisition and processing systems



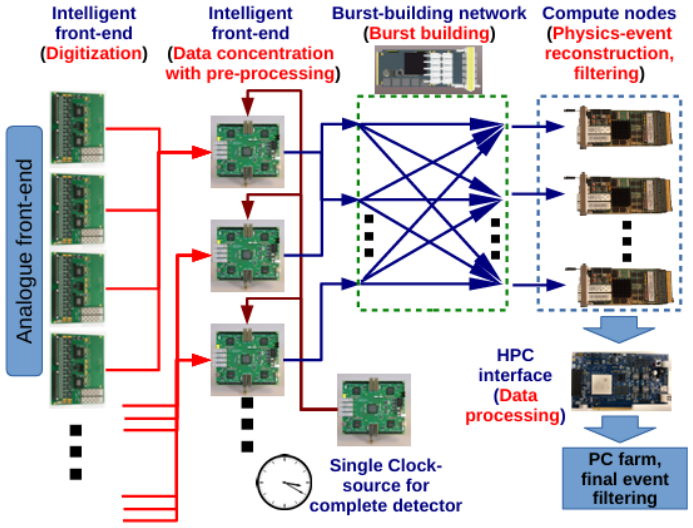
Fair-center.eu

PANDA

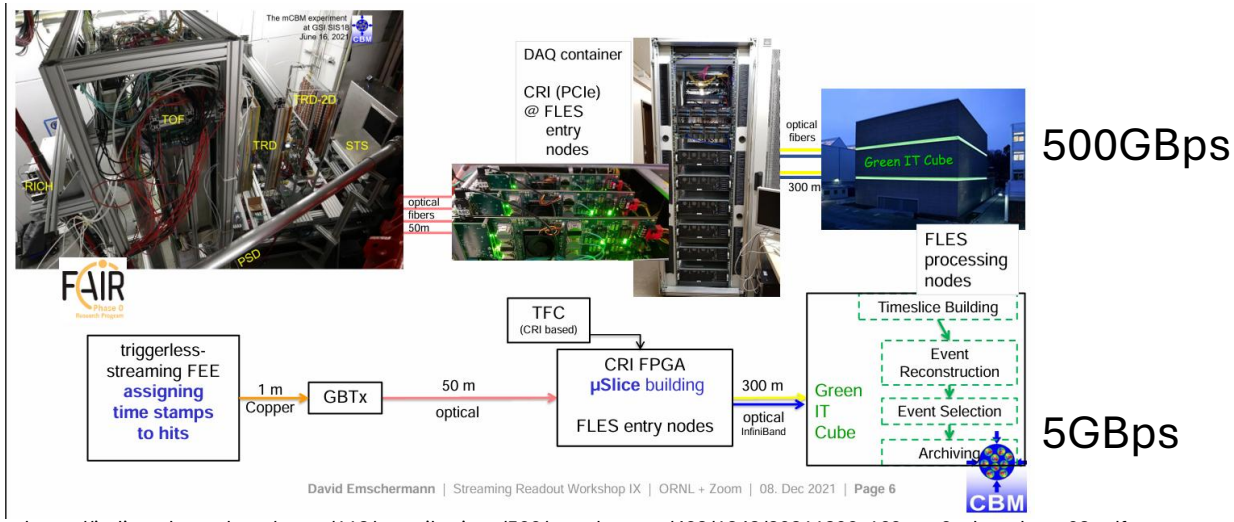


Fair-center.eu

CBM



M. Kavatsyuk



500GBps

5GBps

https://indico.phy.ornl.gov/event/112/contributions/566/attachments/492/1342/20211208_169_sro9_cbm_daq_v02.pdf

Large scale physics exp.

Data acquisition and processing systems

- TRB – digitizing cards streaming measurement data via UDP
- Framework for processing TRB data
- Particle track reconstruction algorithm
- SW – GW cross-validation, single codebase

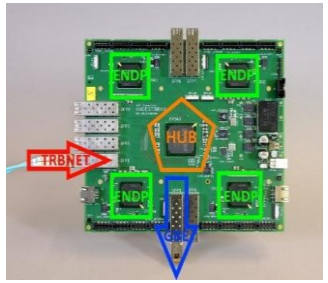
Journals & Magazines > IEEE Transactions on Nuclear ... > Volume: 69 Issue: 7

Real-Time Data Processing Pipeline for Trigger Readout Board-Based Data Acquisition Systems

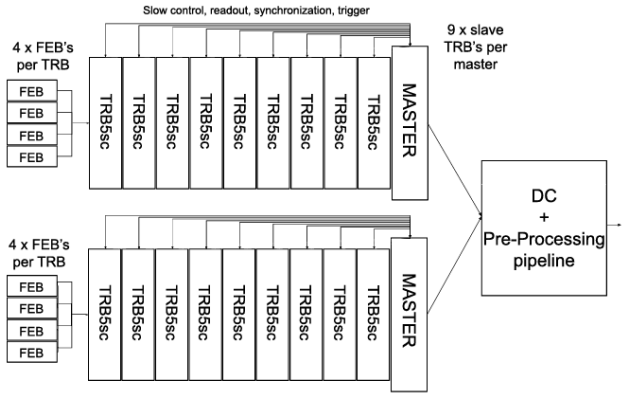
Publisher: IEEE | Cite This | PDF

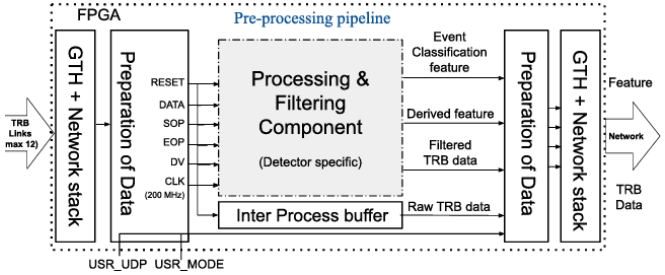
A. Malige ; G. Korcyl ; M. Firlej ; T. Fiutowski ; M. Idzik ; B. Korzeniak All Authors

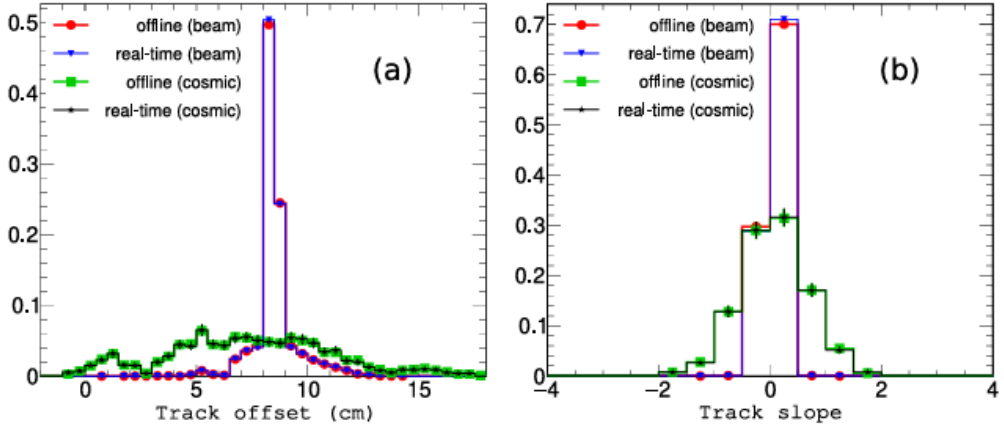
Real-Time Data Processing Pipeline for Trigger Readout Board-Based Data Acquisition Systems | IEEE Journals & Magazine | IEEE Xplore



Traxler, M.; Korcyl, G.; Bayer, E.; Maier, L.; Michel, J.; Palka, M.
„A compact system for high precision time measurements (<14 ps RMS) and integrated acquisition for a large number of channels”,
JINST 10.1088/1748-0221/6/12/C12004



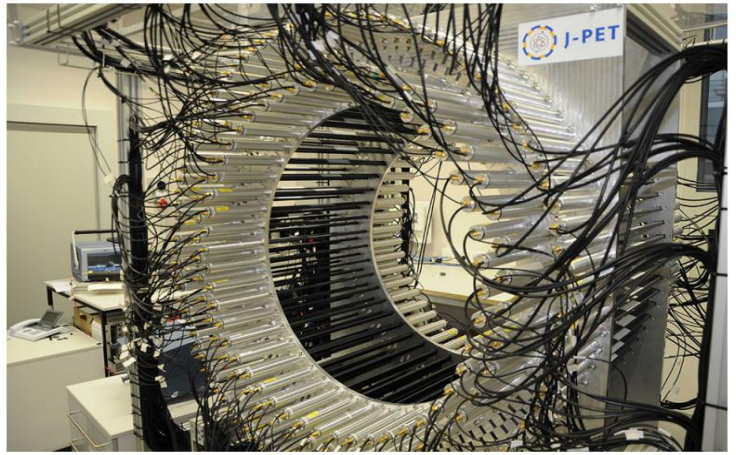


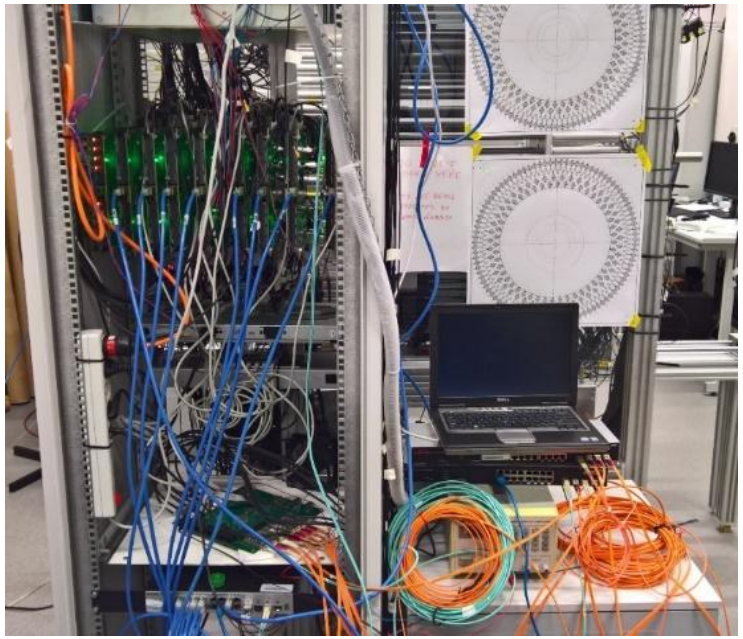


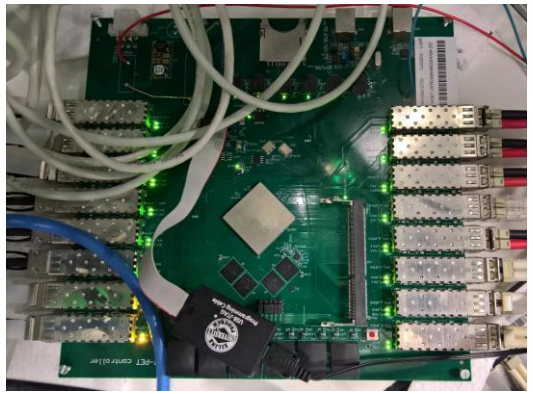
PET Tomography

J-PET prototype 2

- Spinoff of large scale systems towards medical imaging
  - Digitizing system TRB
- Platform for processing UDP data streams
  - System-On-Chip device
    - Programmable logic resources
    - Integrated ARM cores
  - 8 parallel data streams







PET Tomography

J-PET prototype 2

- Continuous readout
  - Unique in PET scanners
  - Enables in-depth analysis
  - Produces excessive amount of data

(12) **United States Patent**
Korcyl et al.

(10) **Patent No.: US 10,007,011 B2**
(45) **Date of Patent: Jun. 26, 2018**

(54) **SYSTEM FOR ACQUISITION OF TOMOGRAPHIC MEASUREMENT DATA**

(52) **U.S. Cl.**
CPC *G01T 1/2985* (2013.01)

US00000010007011B220180626

Journals & Magazines > IEEE Transactions on Medical ... > Volume: 37 Issue: 11

Evaluation of Single-Chip, Real-Time Tomographic Data Processing on FPGA SoC Devices

Publisher: IEEE | Cite This | PDF

G. Korcyl ; P. Białas ; C. Curceanu ; E. Czerwiński ; K. Dulski ; B. Flak All Authors

Evaluation of Single-Chip, Real-Time Tomographic Data Processing on FPGA SoC Devices | IEEE Journals & Magazine | IEEE Xplore

- SoC for online processing
  - LoR reconstruction on FPGA – real-time, high throughput
  - Control, monitoring, parameters, visualization on CPU

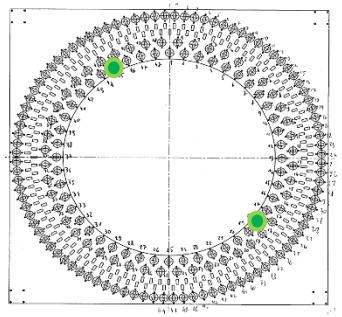
```
0000  ff ff ff ff ff ff 00 00
0010  01 1c 0b 00 00 00 ff 11
0020  ff ff c3 50 c3 50 01 08
0030  00 02 01 03 00 00 00 00
0040  00 02 06 01 00 02 06 02
```

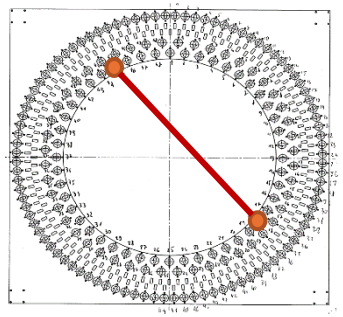
Hit1: ch 1, 115 ns, TOT 5 ns
Hit2: ch 2, 116 ns, TOT 7 ns
…

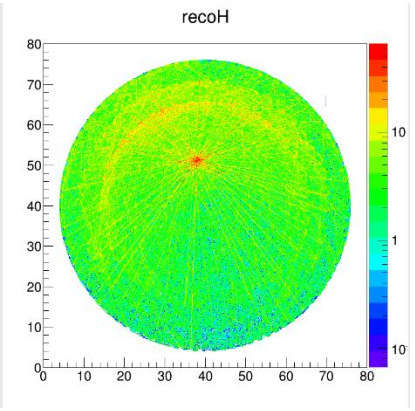
…

```
0000  ff ff ff ff ff ff 00 00
0010  01 1c 0b 00 00 00 ff 11
0020  ff ff c3 50 c3 50 01 08
0030  00 02 01 03 00 00 00 00
0040  00 02 06 01 00 02 06 02
```

Hit1: ch 1, 115 ns, TOT 5 ns
Hit2: ch 2, 116 ns, TOT 7 ns
…

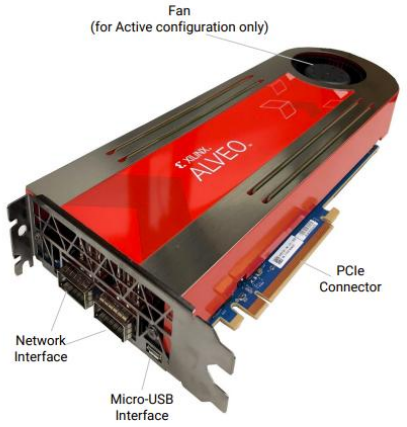


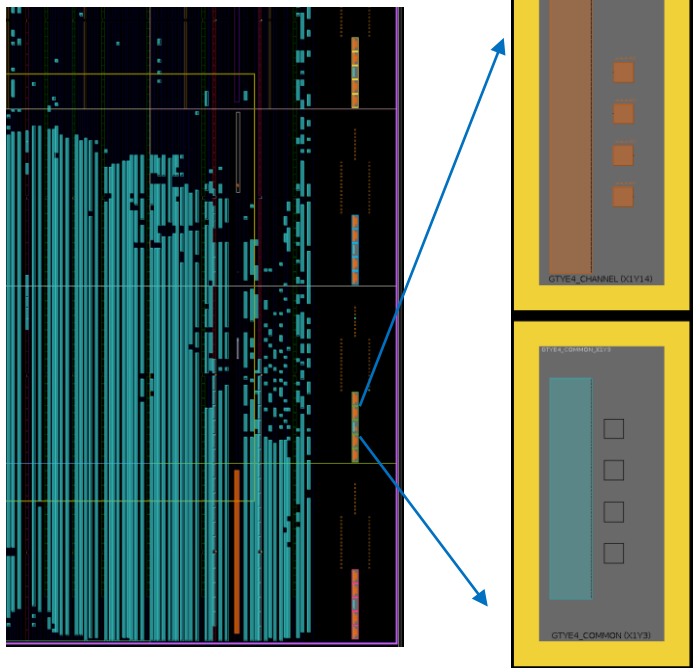


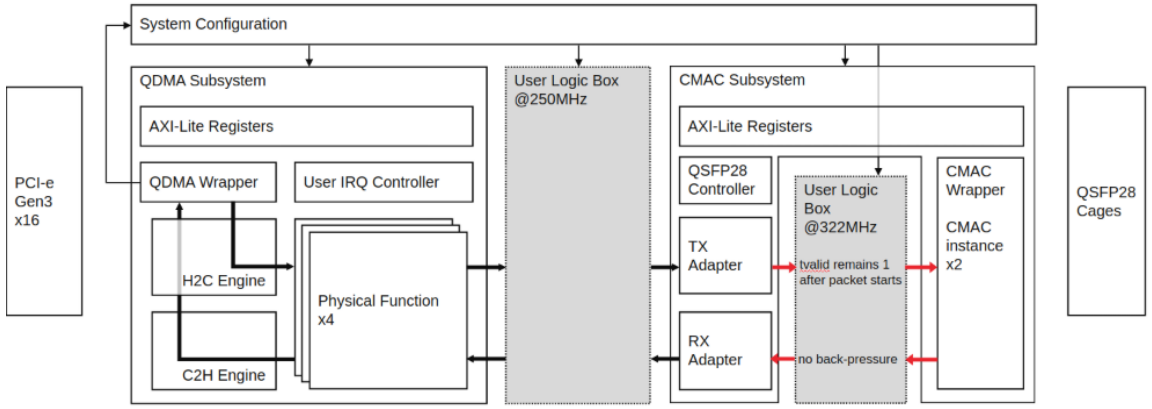


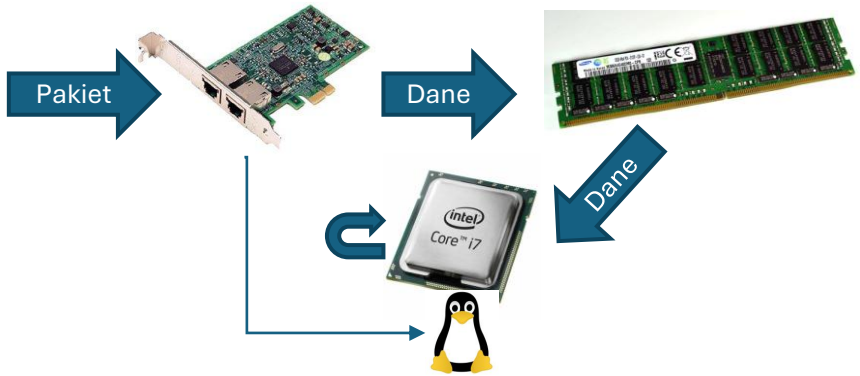
Latency measurements

- Processing of time-critical network traffic
  - Natural for FPGAs
  - Integrated Multi-Gigabit Transceivers
    - E.g.: 128 channels, 30Gbps each in VU13P
    - Fully parallel channel processing
    - Minimal and deterministic latency
  - Alveo platforms as SmartNIC
    - OpenNIC as GW and driver









GitHub - Xilinx/open-nic: AMD OpenNIC Project Overview · GitHub

Latency measurements

- OSI L1 100G Ethernet in FPGA fabric

- Stream of decoded words processed with HLS
  - C++ constructs to describe processing logic
  - Synthesized into HDL (VHDL and Verilog)
  - Stream of 512b words at 250MHz frequency
  - FPGA specifics and constraints
    - A pipelined loop symbolizes consecutive clock cycles
    - At each clock cycle a new data comes from the stream
    - Critical path in the implemented logic must fit within clock period

```
void ts_tx( hls::stream<word>& in, hls::stream<word>& out) {
    #pragma HLS INTERFACE mode=axis port=in
    #pragma HLS INTERFACE mode=axis port=out

    word w;

    while (true) {

        #pragma HLS PIPELINE

        in.read_nb(w);

        out.write(w);

    }
```

Computer Networks

Latency measurements

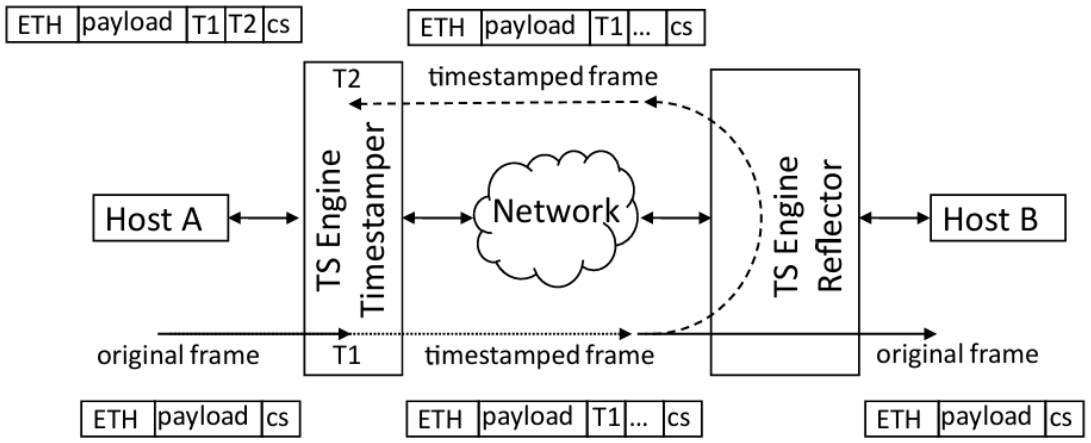
- Timestampers implemented on FPGA
  - Real traffic tagging instead of probe packets
  - Completely offloaded to Alveo accelerators
  - 250MHz -> 4 ns precision
  - No synchronization of timestampers -> round-trip

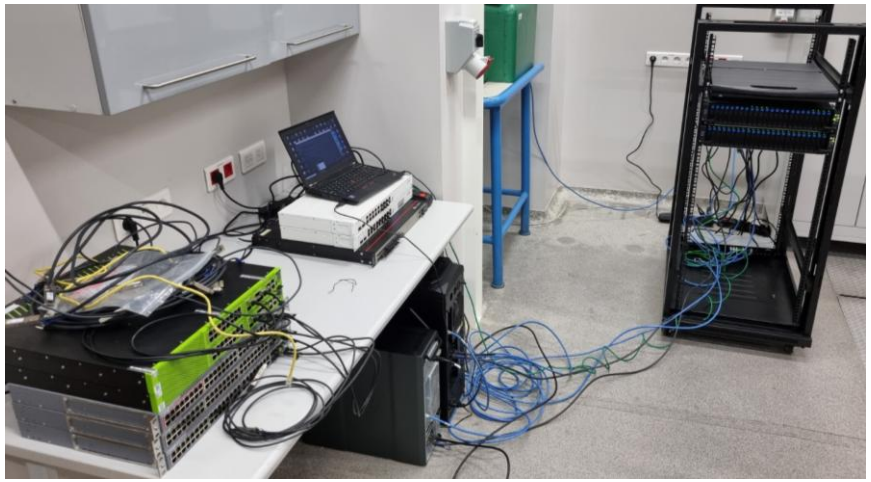
Computer Networks
Volume 279, April 2026, 112138

Passive latency measurement with nanosecond precision for time-critical applications

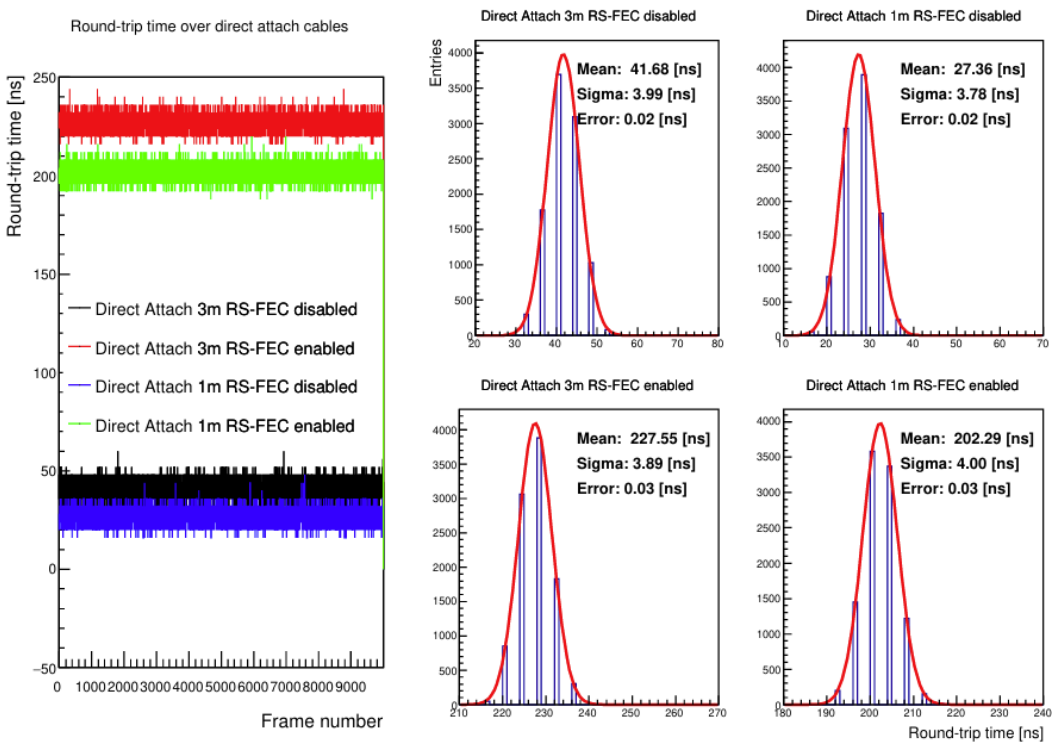
Grzegorz Korcyl [a], Zbigniew Duliński [a,b]

https://doi.org/10.1016/j.comnet.2026.112138

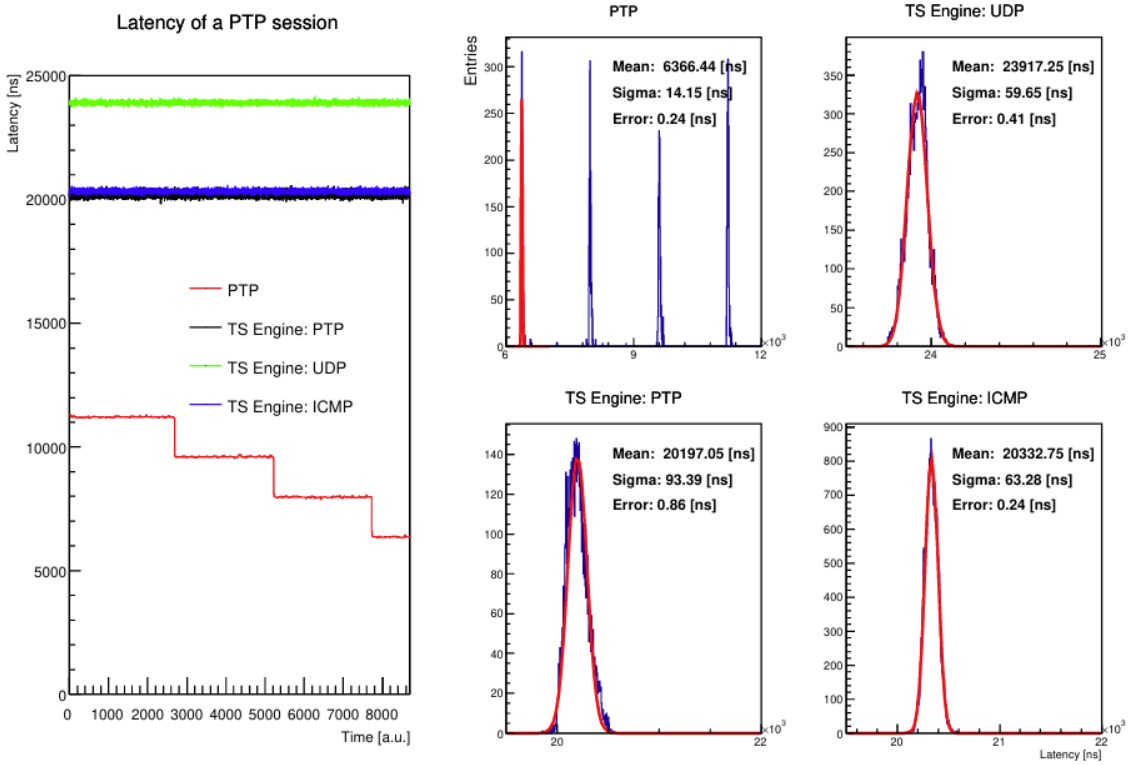




Latency measurements



Capability to benchmark network devices and their features



Real traffic vs Precision Time Protocol

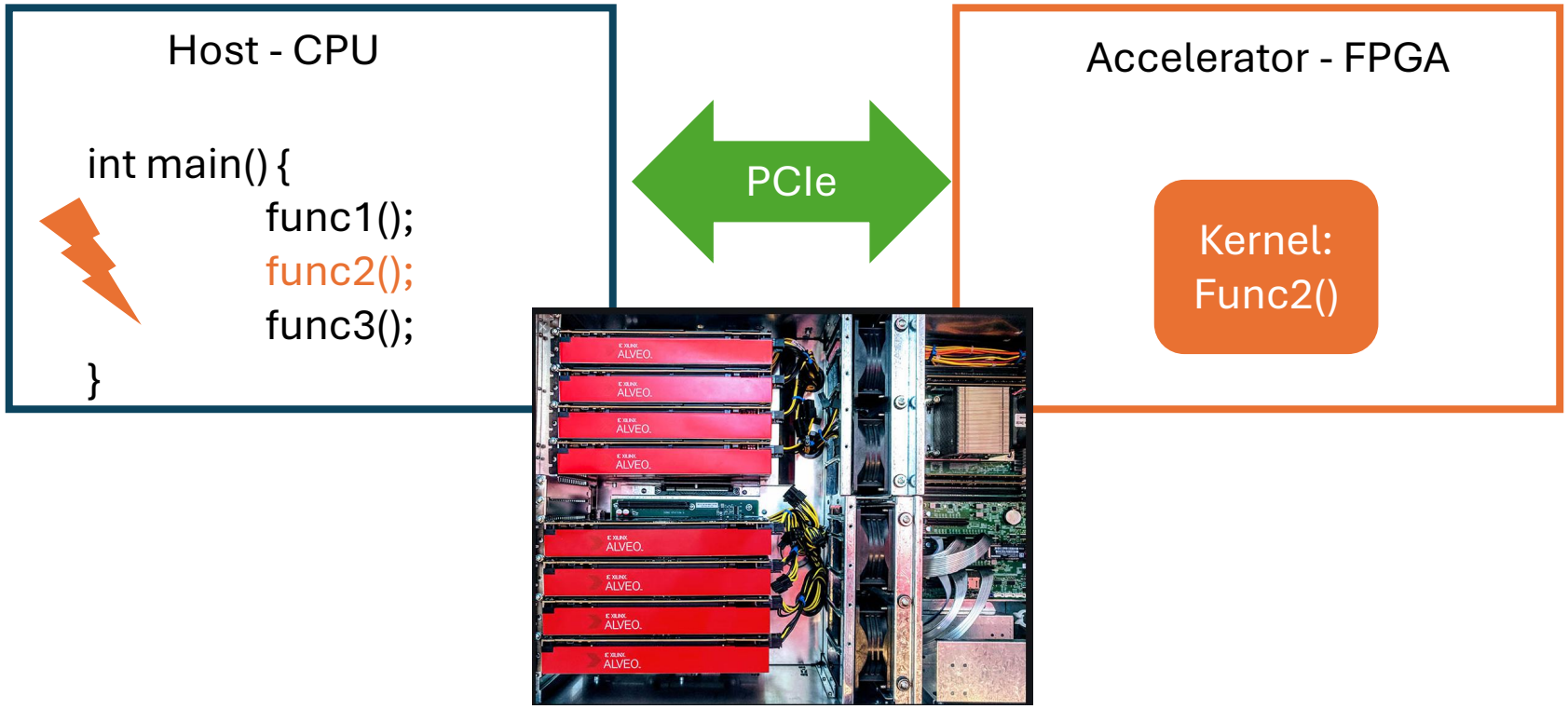
Latency measurements

- Round-trip is suboptimal
  - Asymetric paths
  - Rebound packets overhead
- One-way requires synchronization
  - GNSS system generates PPS signal
  - Easy to be integrated with FPGA logic
  - LTE modem
- Prototyping on ZCU102 platform
  - Embedded ARM
  - 10G interfaces
  - Continuous, passive measurement station

High-Performance Computing

FPGA as an accelerator

- FPGA + PCIe platforms to offload intensive computation tasks
  - Main thread on the host CPU
  - Selected functions delegated to FPGA



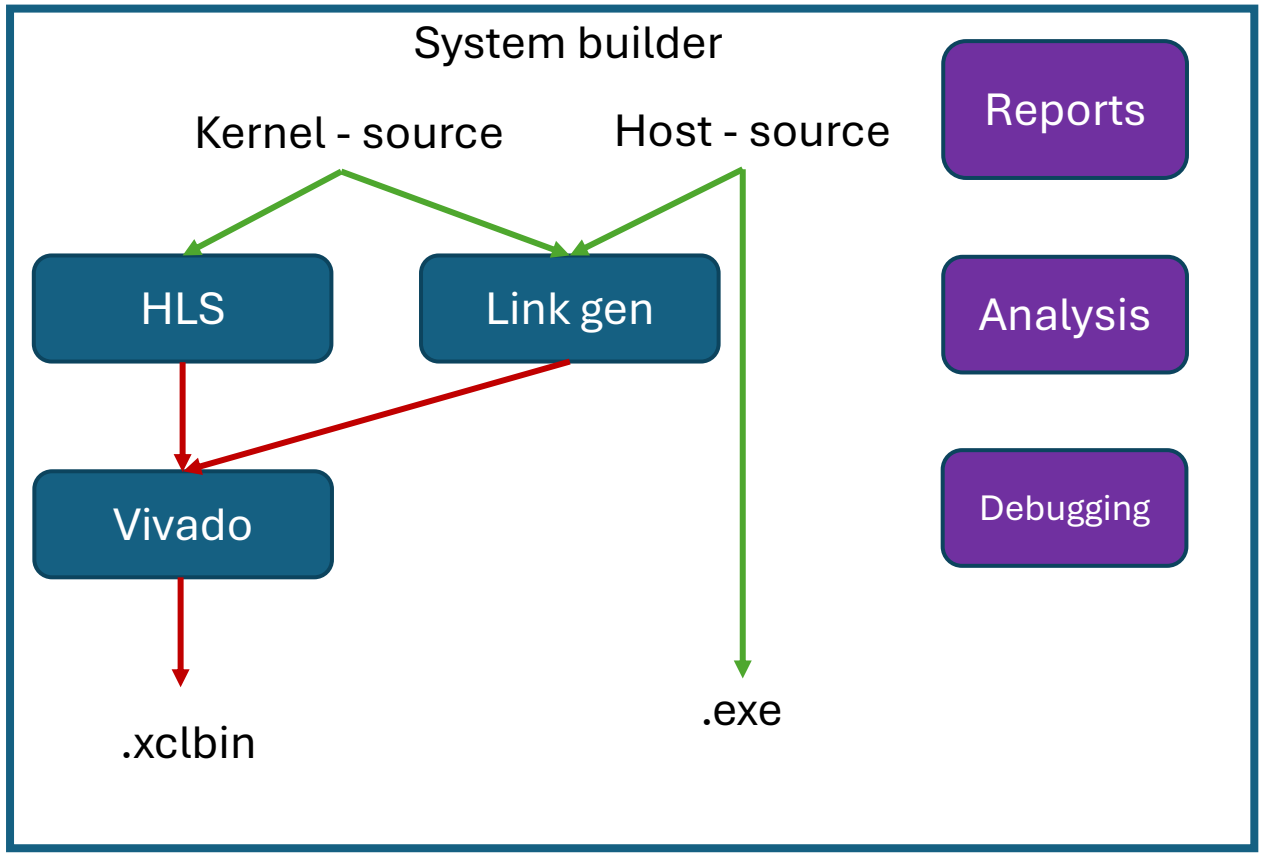
High-Performance Computing

FPGA as an accelerator

1. Host executable
   - Main() – start of the program
   - FPGA kard initialization and configuration
   - Main algorithm thread, data preparation
   - Accelerator tasks queueing

2. Kernel
   - Single, accelerated function
   - Function compiled with HLS

3. Link
   - Communication infrastructure
   - Ensures data exchange, control of kernels, profiling and debugging
   - Generated based on numer of kernels, their type, their argument types



High-Performance Computing

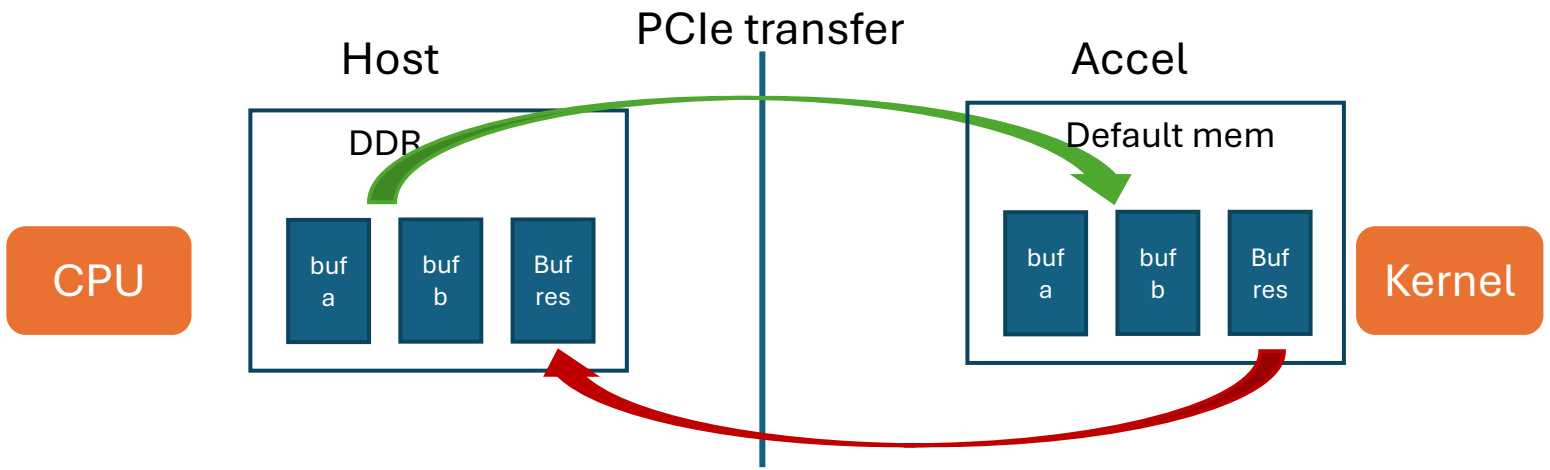
FPGA as an accelerator

```
void krnl_vadd(const unsigned int *in1, // Read-Only Vector 1
               const unsigned int *in2, // Read-Only Vector 2
               unsigned int *out_r,     // Output Result
               int size                 // Size in integer
)
```

C++ kernel compiled with HLS

Host code with OpenCL constructs

```
// Data will be migrated to kernel space
q.enqueueMigrateMemObjects({buffer_a,buffer_b},0/* 0 means from host*/);

//Launch the Kernel
q.enqueueTask(krnl_vector_add);

// The result of the previous kernel execution will need to be retrieved in
// order to view the results. This call will transfer the data from FPGA to
// source_results vector
q.enqueueMigrateMemObjects({buffer_result},CL_MIGRATE_MEM_OBJECT_HOST);

q.finish();
```



HPC benchmark on FPGA

High-Performance Computing

- Conjugate Gradient variant as HPC benchmark
- Project entirely implemented in C++
- First use of Alveos for compute intensive HPC
- Efficient use of available resources, interconnects and embedded memory

ELSEVIER
Computer Physics Communications
Volume 249, April 2020, 107029

Towards Lattice Quantum Chromodynamics on FPGA devices ☆

Grzegorz Korcyl [a], Piotr Korcyl [b c]

https://doi.org/10.1016/j.cpc.2019.107029

SUPERCOMPUTING FRONTIERS AND INNOVATIONS
An International Journal
Focus and Scope | Editorial Board | Current Issue | Archive

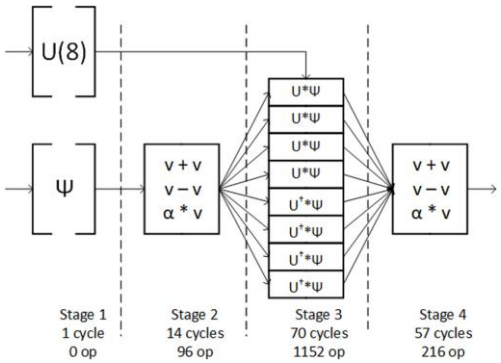
Home / Archive / Vol. 6 No. 2 (2019) / Articles

Investigating the Dirac Operator Evaluation with FPGAs

Grzegorz Korcyl
Department of Information Technologies, Faculty of Physics, Astronomy and Applied Computer Science, Jagiellonian University, Cracow, Poland
Piotr Korcyl
Institut fur Theoretische Physik, Universitat Regensburg, Regensburg, Germany
Institute of Physics, Jagiellonian University, Cracow, Poland

№2 2019 Volume 6
SUPERCOMPUTING FRONTIERS AND INNOVATIONS

https://doi.org/10.14529/jsfi190204



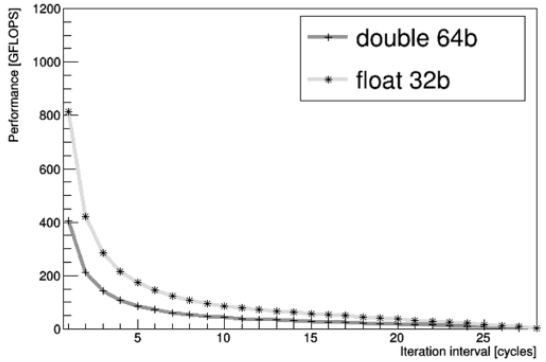


Figure 4. Performance as a function of the initiation interval

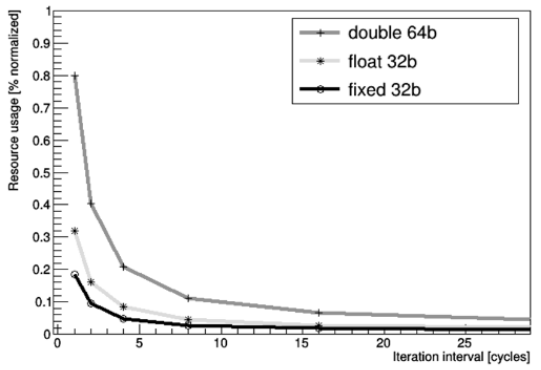


Figure 5. Resources consumption as a function of the initiation interval

```
from xrt_binding import *
import pyxrt

d = pyxrt.device(opt.index)
xbin = pyxrt.xclbin(opt.bitstreamFile)
uuid = d.load_xclbin(xbin)

simple = pyxrt.kernel(d, uuid, "simple")

boHandle1 = pyxrt.bo(d, 10, pyxrt.bo.normal, simple.group_id(0))
boHandle2 = pyxrt.bo(d, 10, pyxrt.bo.normal, simple.group_id(1))

boHandle1.sync(pyxrt.xclBOSyncDirection.XCL_BO_SYNC_BO_TO_DEVICE, 10, 0)
boHandle2.sync(pyxrt.xclBOSyncDirection.XCL_BO_SYNC_BO_TO_DEVICE, 10, 0)

run = simple(boHandle1, boHandle2, 0x10)
state = run.wait()

boHandle1.sync(pyxrt.xclBOSyncDirection.XCL_BO_SYNC_BO_FROM_DEVICE, 10, 0)
```

https://github.com/Xilinx/XRT/blob/master/tests/python/02_sim

High-Performance Computing

FPGA without HDL

- Heterogeneous programming models
- Common C++ codebase for any computing device
- SYCL constructs based on OpenCL
  - Code organization into kernels
  - SYCL builds dependency graph and schedules kernels
  - SYCL manages data buffers echangeable between kernels
  - Unified codebase
  - Limited support of FPGAs

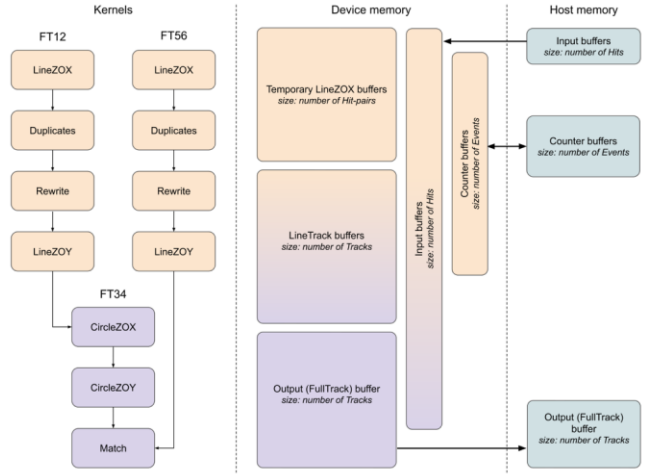


Fig. 4 Kernel data dependency and data flow graph

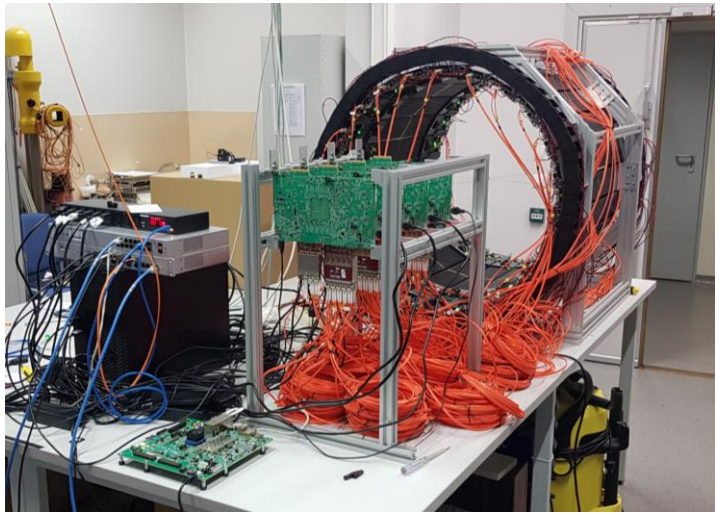
The successful compilation of the complex SYCL code for the FPGA platform is an achievement of its own. The measured performance of Alveo U280 was two to three orders of magnitude worse compared to other platforms, as FPGAs

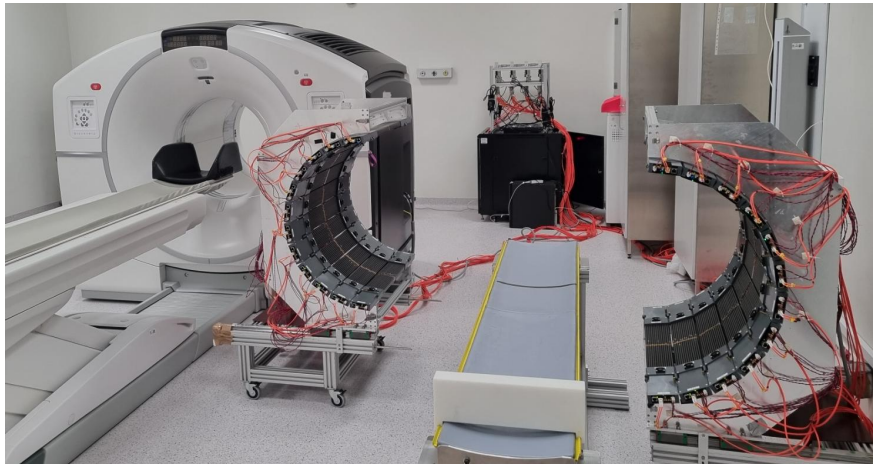
Complete DAQ system

PET Tomography

Large scale physics exp.

- Hardware layer
  - Uniezależnienie od ograniczonych standardów z HEP
  - Wykorzystanie komercyjnie dostępnych platform
  - Dedykowana elektronika czołowa
- Interconnect layer
  - Pojedyncze połączenie
    - Synchronizacja
    - Transport danych pomiarowych
    - Wymiana komunikatów sterująco-kontrolnych
- Software layer
  - Naturalna obsługa systemu z poziomu Python



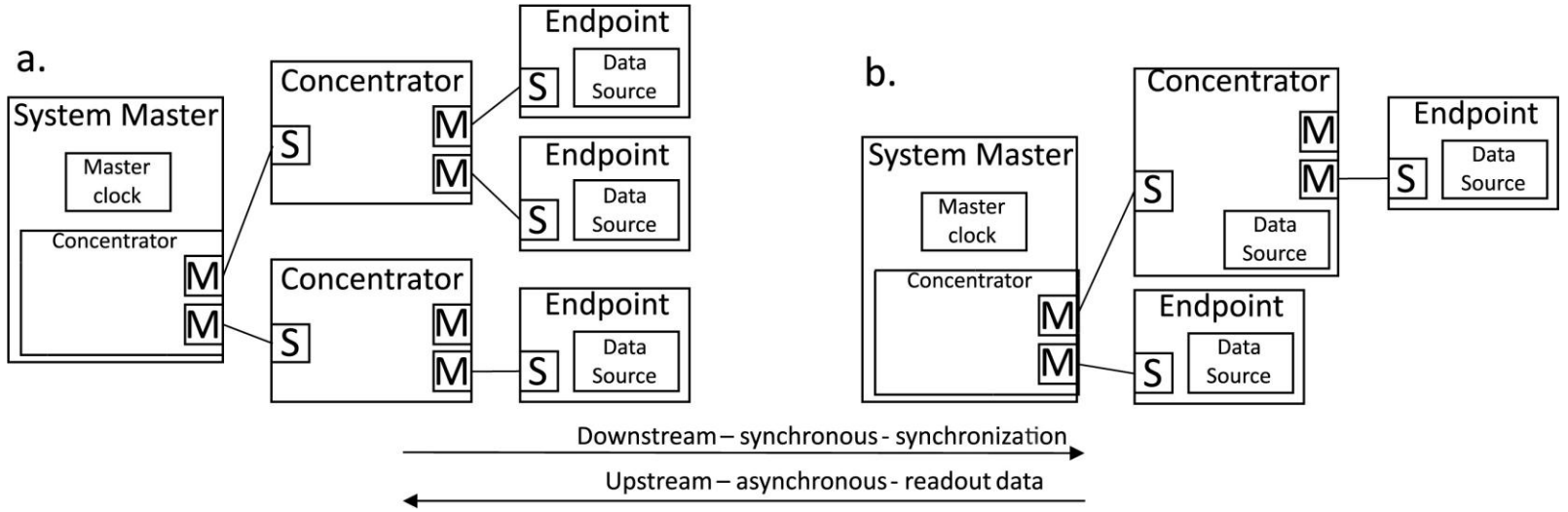


PET Tomography

Large scale physics exp.

AuroraSync

- Aurora
  - Low-level point-to-point communication protocol
- Sync
  - Synchronous connections
  - Recovered clock in fixed relation to the master clock
  - One source – hierarchical architecture – synchronization of endpoints

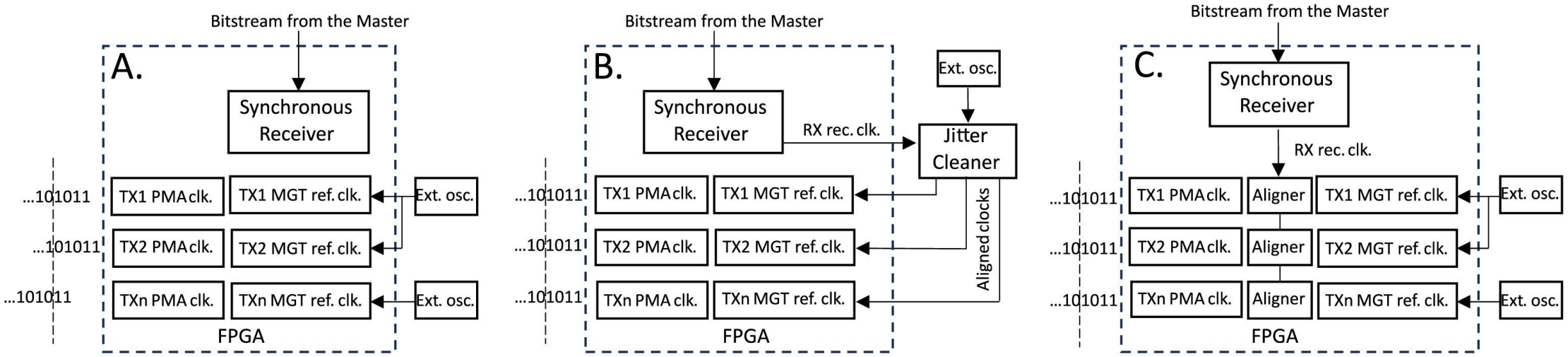


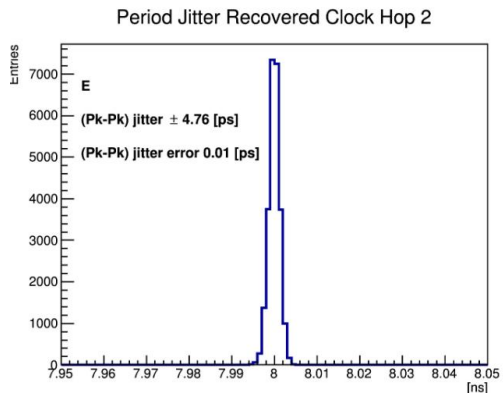
PET Tomography

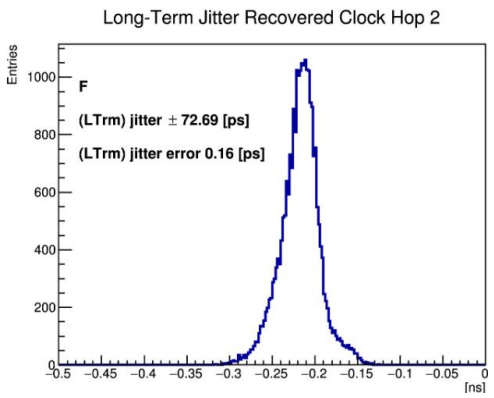
Large scale physics exp.

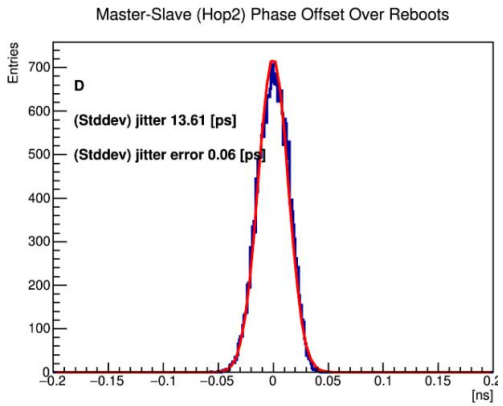
AuroraSync

- **Hardware-agnostic – no specific hardware features required**
- **Clock manipulation inside FPGA – no dedicated, external devices (jitter cleaners)**







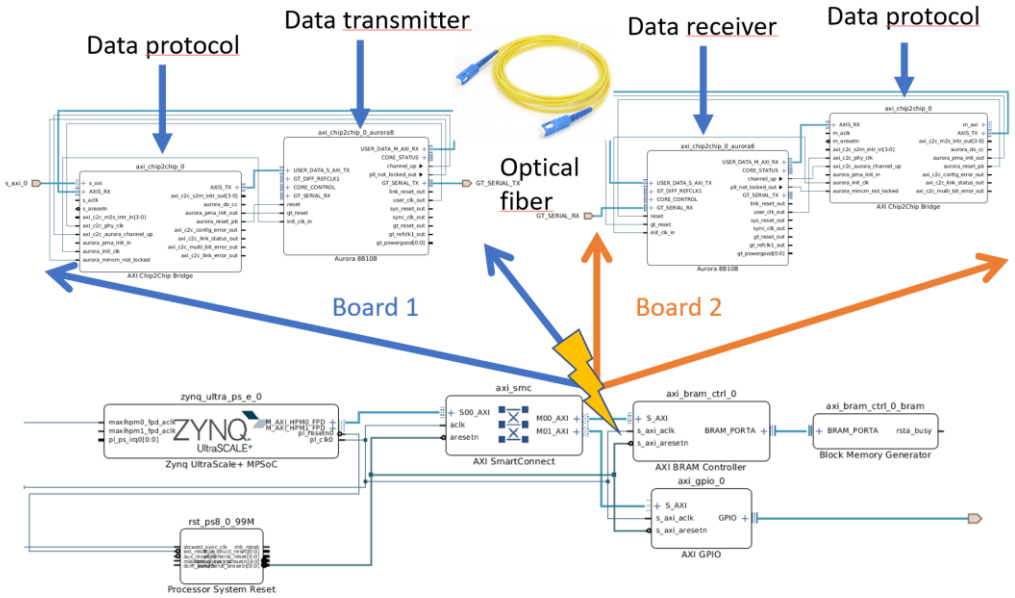


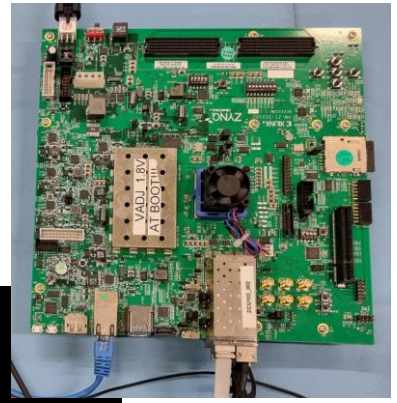
Complete DAQ system

PET Tomography

Large scale physics exp.

- Software layer
  - Linux on ARM embedded into SoC FPGA
  - FPGA logic components mapped into Linux memory (AXI)
  - Hierarchical addressing scheme
  - Transparent physical layout of the system
  - Python based control and monitoring





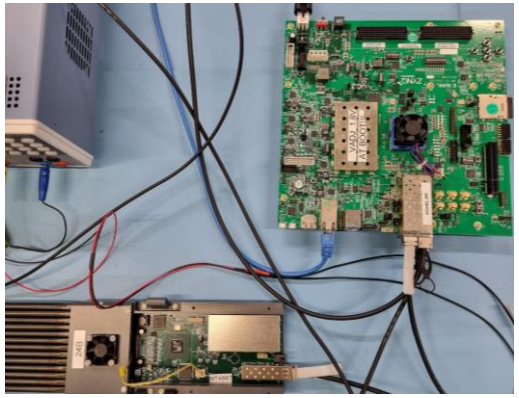
```
daq@djpetsrv01:~/MODv2/work$ ./daq_console.py

Welcome to DJPET cli.
Using MODULAR MTAB setup

<Setup> connecting to the server successfull
<Wiener Controller -0x1> connection to the device: True
<PDU Controller -0x1> connection to the device: True
>>> pdu.send_command([1, 2, 3, 4], 1)
<PDU Controller -0x1> sent http://1
=Apply
>>> enable_power_supply()
<Wiener Controller -0x1> connection
<Wiener Controller -0x1> sent outpu
<Wiener Controller -0x1> sent outpu
<Wiener Controller -0x1> sent outpu
<Wiener Controller -0x1> sent outpu
<Wiener Controller -0x1> sent outpu
<Wiener Controller -0x1> sent outpu
<Wiener Controller -0x1> sent outpu
<Wiener Controller -0x1> sent outpu
>>>
```

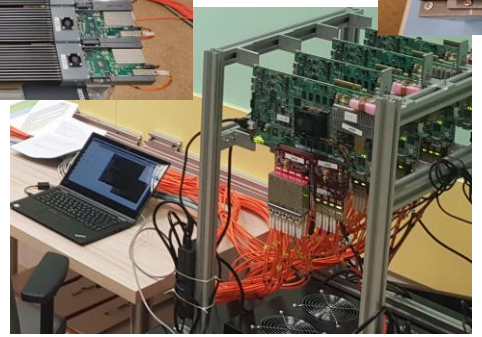
Complete DAQ system

PET Tomography

Large scale physics exp.

- Multiple development techniques
  - Pure HDL for low-level operations such as clock sync.
  - HLS for processing data streams
  - Block Design visual design of platform designs
  - Linux – Programmable resources interfacing
- Not only for J-PET
  - Prototyping of FAIR subsystems





ELSEVIER
Measurement
Volume 257, Part A, 15 January 2026, 118528

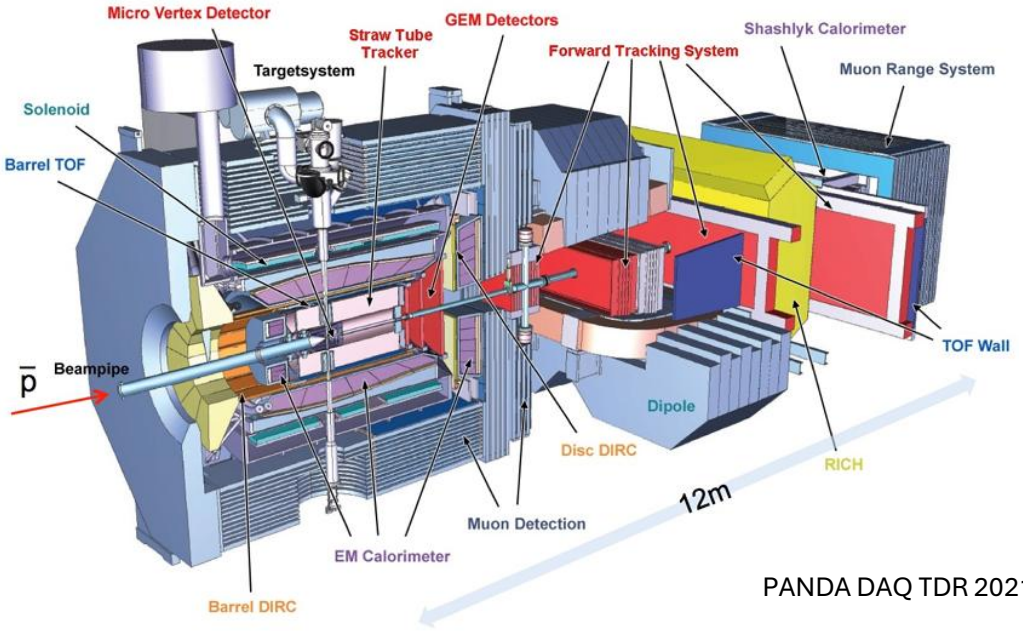
Hardware-agnostic framework for general-purpose data acquisition systems

Grzegorz Korcyl [a b], Maciej Bakalarek [c], Paweł Moskal [b c]

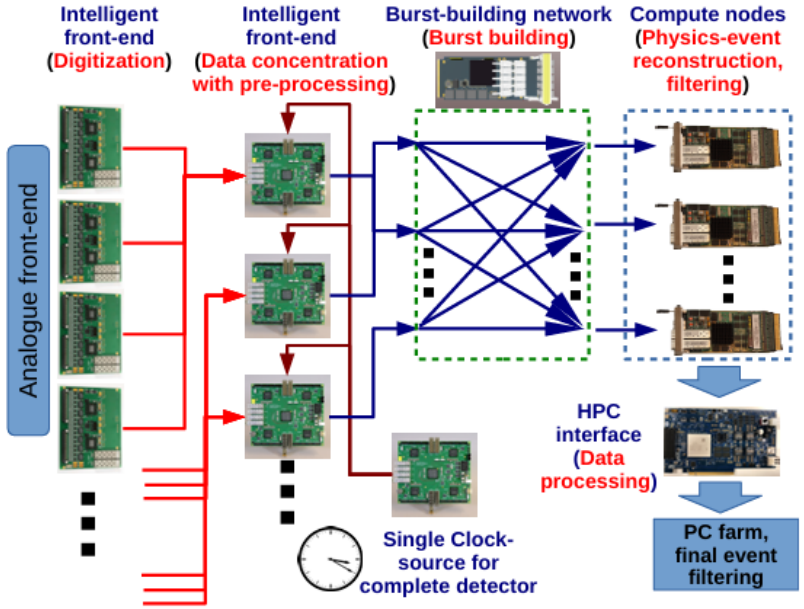
Hardware-agnostic framework for general-purpose data acquisition systems - ScienceDirect

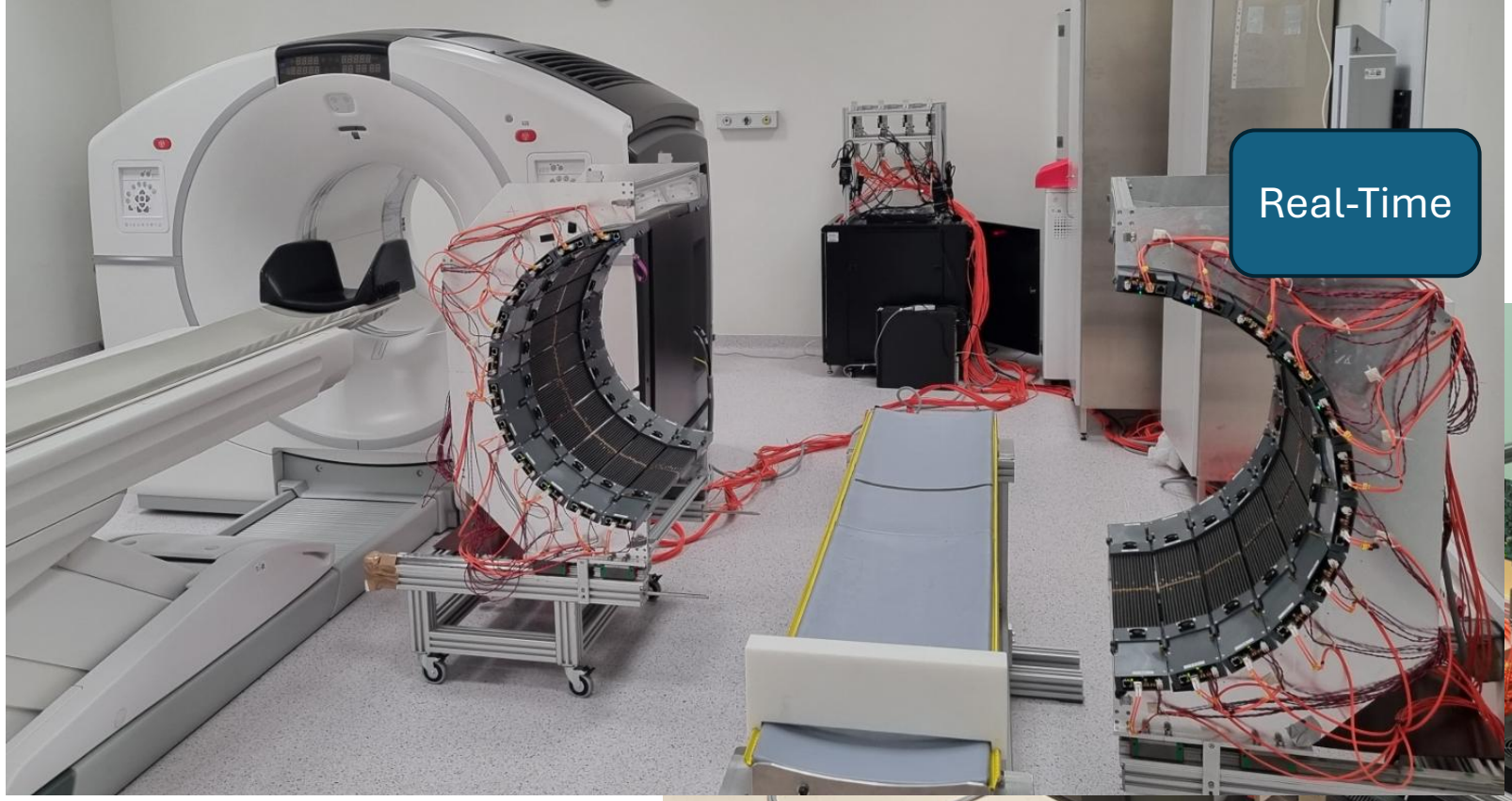
Systemy Przetwarzania

- Rozproszone źródła danych strumieniowych
  - Synchroniczne sieci
  - Dedykowane protokoły
- Agregacja danych
- Przetwarzanie wstępne w **czasie rzeczywistym**
- Transfer danych do HPC
- Przetwarzanie i selekcja danych w **trybie online**



PANDA DAQ TDR 2021

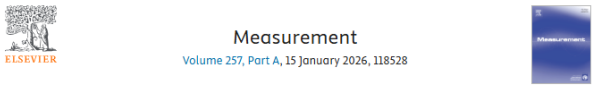




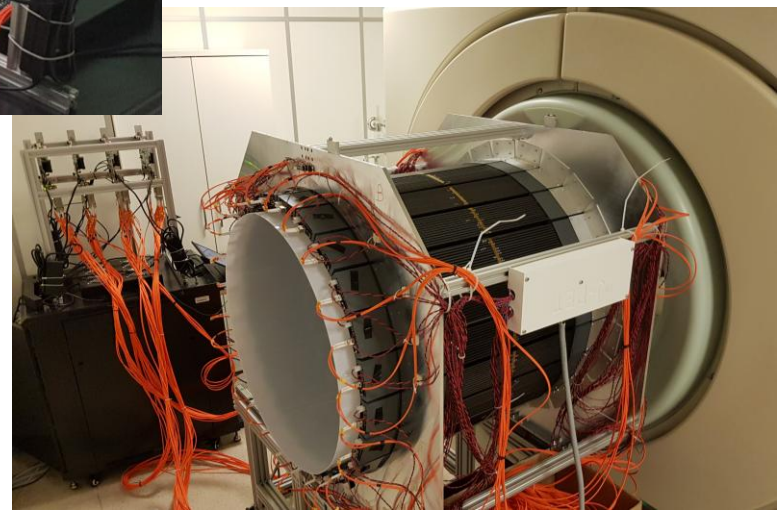
Szpital Uniwersytecki, Kraków 2024

CCB IFJ PAN, Kraków 2021

- Unikalny system:
  - Nowoczesne techniki rozwoju
  - Wykorzystanie uznanych standardów i protokołów
  - System niezależny od platformy sprzętowej



Hardware-agnostic framework for general-purpose data acquisition systems - ScienceDirect

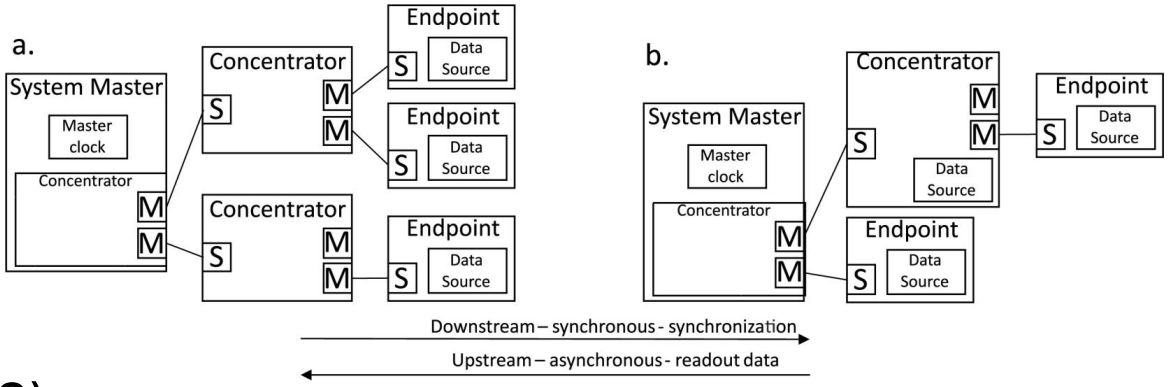


Szpital Banacha, Warszawa 2022

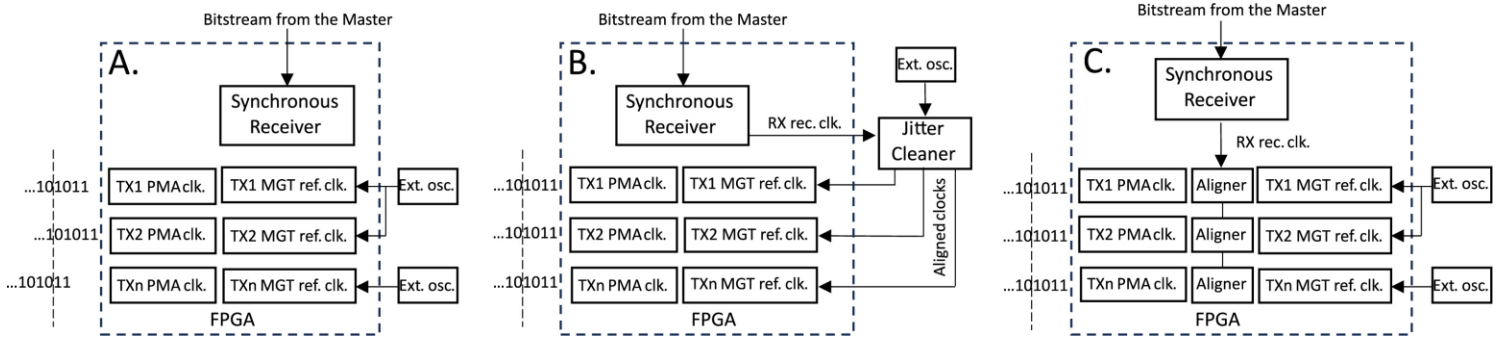
Aurora-Sync

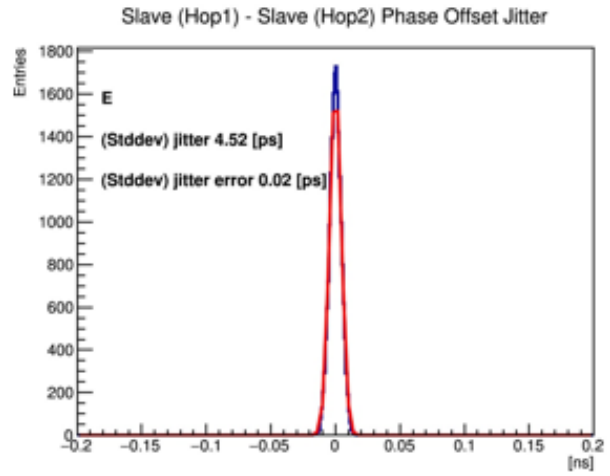
Real-Time

- System komunikacji i przetwarzania danych:
  - Konfiguracja układów FPGA (gateware)
    - Protokół komunikacji Aurora 5 Gbps
      - Odbiór strumieni danych od Endpointów
      - Dostarczanie synchronizacji do Endpointów
      - Wymiana komunikatów sterująco kontrolnych



  - Synchroniczne połączenia (warstwa L1, L2)

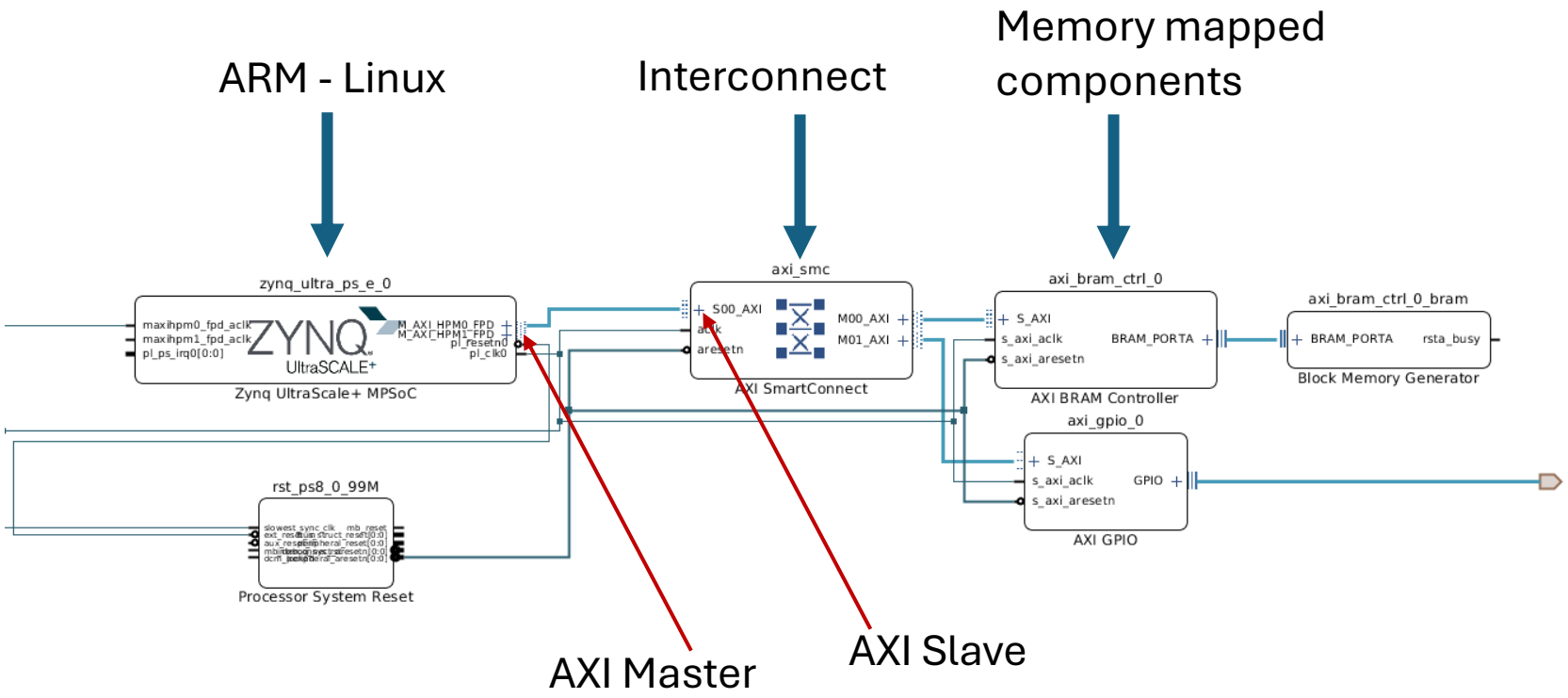




AXI and Aurora-Sync

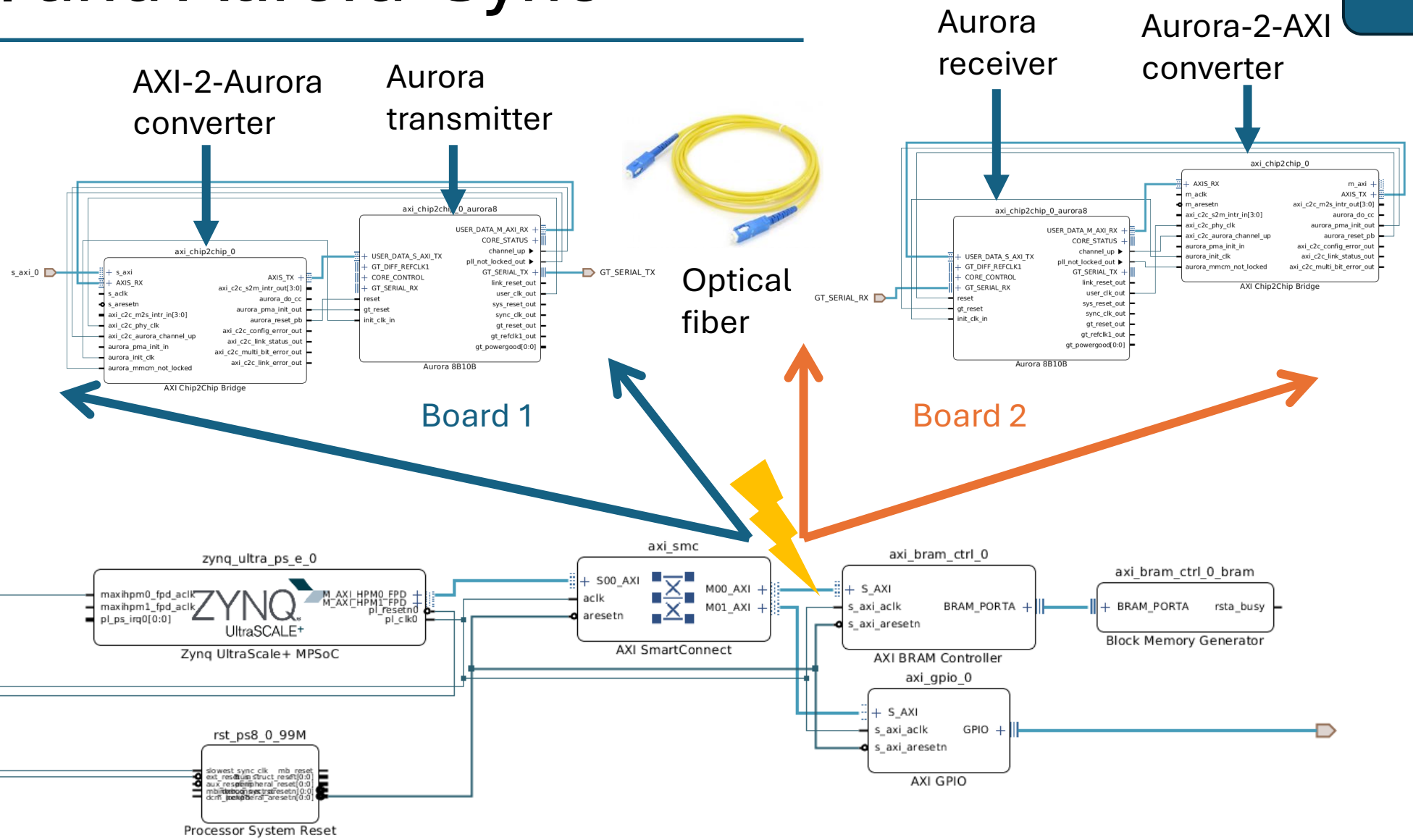
Real-Time

- Wymiana komunikatów sterująco-kontrolnych
  - Punkt centralny systemu: Petalinux
  - Każdy logiczny komponent AXI jest mapowany na adres w pamięci



AXI and Aurora-Sync

Real-Time



Zagadnienia: J-PET i Aurora-Sync

Real-Time

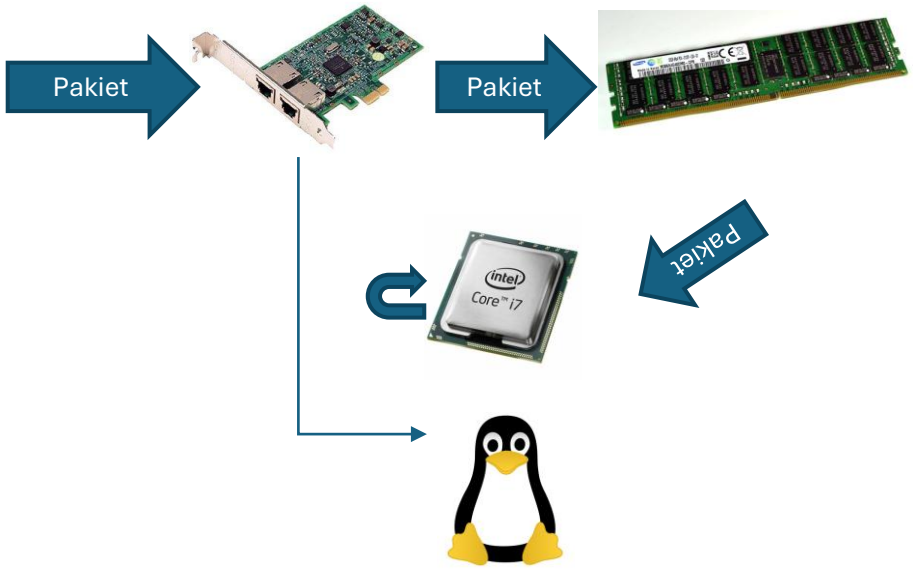
- Praca blisko FPGA
  - Metody wyliczania kalibracji i korekcji danych w locie
  - Metody przetwarzania wstępnego
    - Redukcja szumów
    - Selekcja interesujących zdarzeń
  - Mechanizmy kontroli przepływu danych przez system

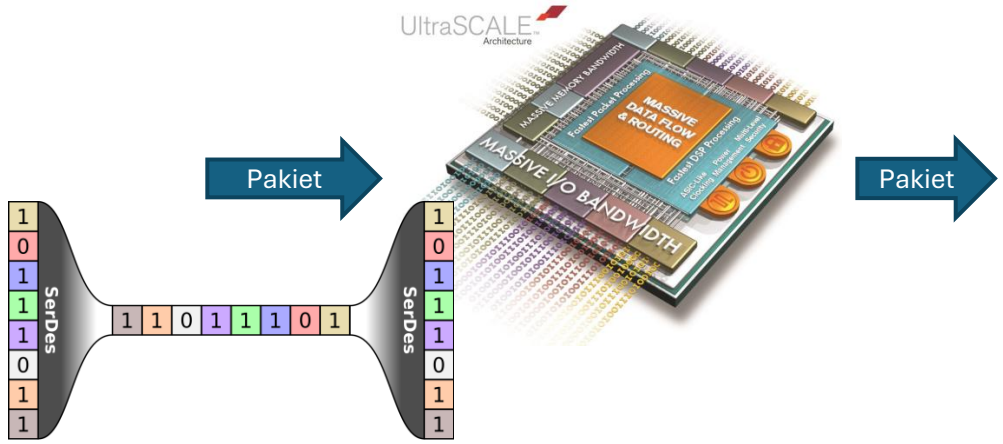
- Sterowanie
  - Logika – ARM – Petalinux – Python - Web
  - Usprawnienie mechanizmów czytania i pisania po rejestrach

Przetwarzanie ruchu sieciowego

Real-Time

- FPGA daje możliwość operowania najbliżej warstwy fizycznej L1





Wikipedia.org

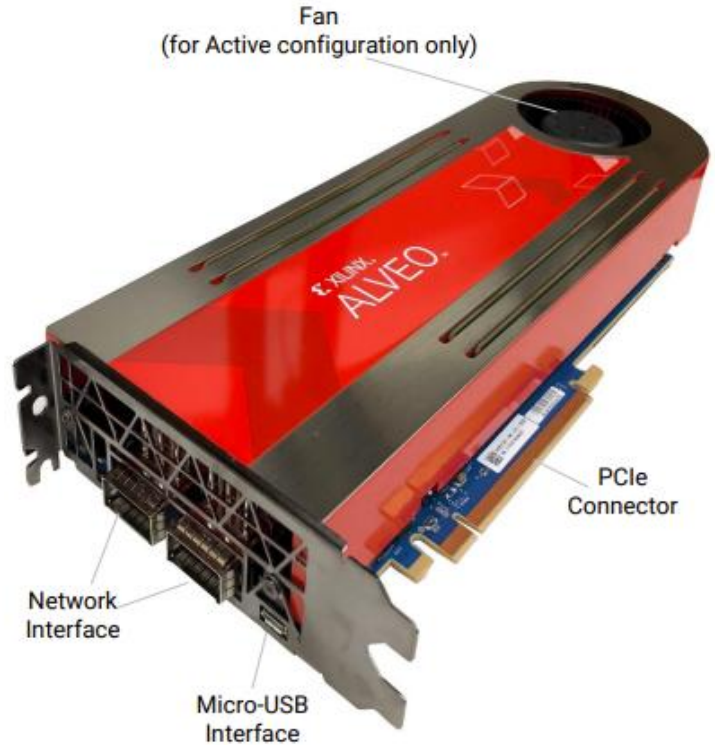
Przetwarzanie ruchu sieciowego

Real-Time

- Quality-of-Service w sieciach 5G+
  - Gwarancja opóźnienia dla krytycznych serwisów
  - Jak je zmierzyć?

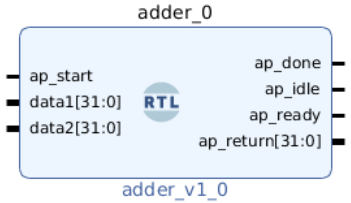
  →Jedynie FPGA jako SmartNIC

- Synteza Wysokiego Poziomu (HLS)
  - Konwersja C/C++ do HDL
  - Implementacja komponentu logicznego jako funkcji




```c
#include "adder.h"

int adder(int data1, int data2) {
    return data1 + data2;
}
```

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity adder is
port (
    ap_start : IN STD_LOGIC;
    ap_done : OUT STD_LOGIC;
    ap_idle : OUT STD_LOGIC;
    ap_ready : OUT STD_LOGIC;
    data1 : IN STD_LOGIC_VECTOR (31 downto 0);
    data2 : IN STD_LOGIC_VECTOR (31 downto 0);
    ap_return : OUT STD_LOGIC_VECTOR (31 downto 0) );
end;


architecture behav of adder is
    attribute CORE_GENERATION_INFO : STRING;
    attribute CORE_GENERATION_INFO of behav : architecture is
    "adder,hls_ip_2018_3,{HLS_INPUT_TYPE=cxx,HLS_INPUT_FLOAT=0,HLS_INPUT_FIXED=0,HLS_INPU
    constant ap_const_logic_1 : STD_LOGIC := '1';
    constant ap_const_logic_0 : STD_LOGIC := '0';
    constant ap_const_boolean_1 : BOOLEAN := true;



begin



    ap_done <= ap_start;
    ap_idle <= ap_const_logic_1;
    ap_ready <= ap_start;
    ap_return <= std_logic_vector(unsigned(data2) + unsigned(data1));
end behav;
```

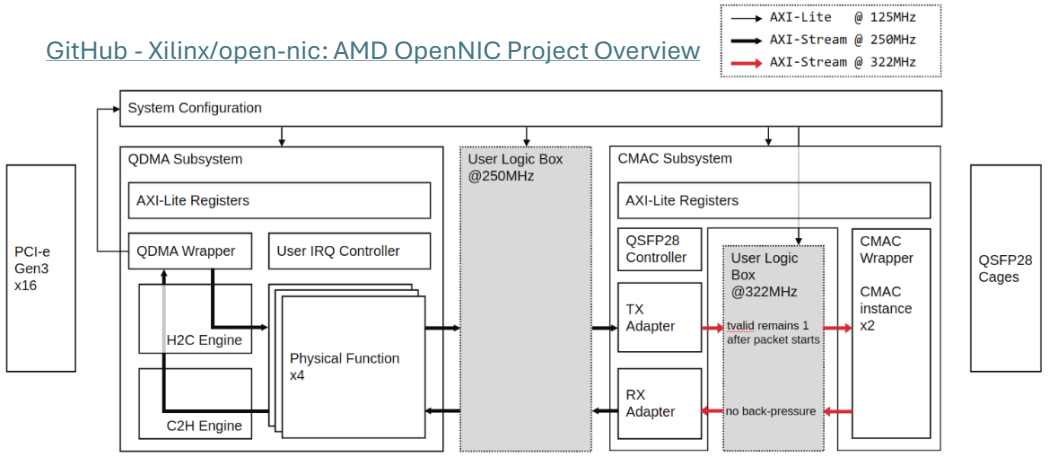


Przetwarzanie ruchu sieciowego

Real-Time

- Infrastruktura do interfejsowania z warstwami L1 i L2 (100G Ethernet)
  - Open-NIC framework (gateware + driver)
- 2 regiony do implementacji przetwarzania
  - Box322: bezpośrednio przy łączu, brak buforowania, wymagający zegar
  - Box250: pomiędzy buforami; mniej wymagający zegar
- Interfejsy AXI-Stream
  - Idealne do zastosowania HLS

GitHub - Xilinx/open-nic: AMD OpenNIC Project Overview



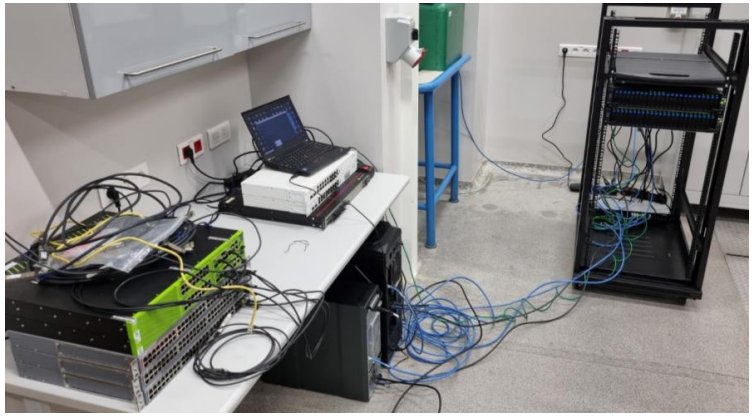

```
void ts_tx( hls::stream<word>& in, hls::stream<word>& out) {
    #pragma HLS INTERFACE mode=axis port=in
    #pragma HLS INTERFACE mode=axis port=out

    word w;

    while (true) {

        #pragma HLS PIPELINE

        in.read_nb(w);

        out.write(w);

    }
```

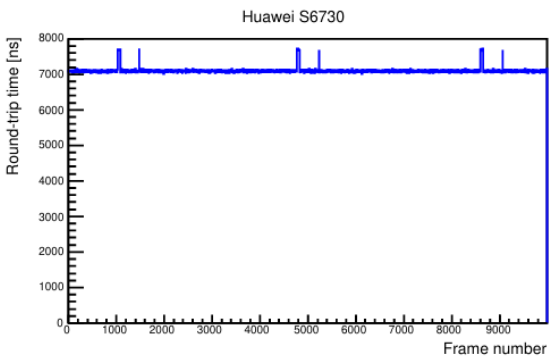
Pomiar opóźnienia w sieci

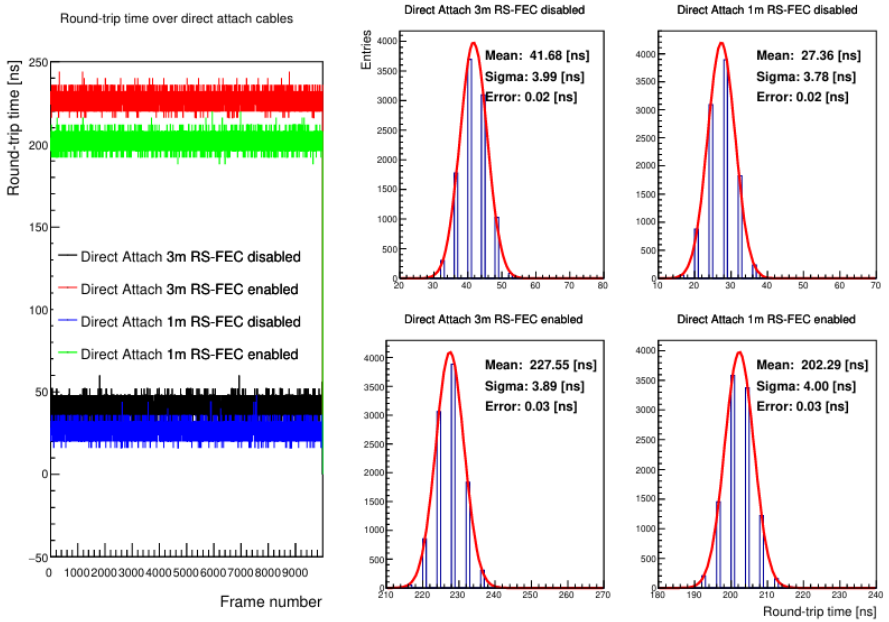
Real-Time

- Manipulacja strukturą pakietu
  - Dodanie sprzętowo generowanego timera
  - Przeliczenie nagłówków pakietu
  - Możliwość wyboru adresów i portów serwisu do monitorowania

- Rozwój w kierunku pomiarów one-way
- Dostarczanie na żywo metryk jakości





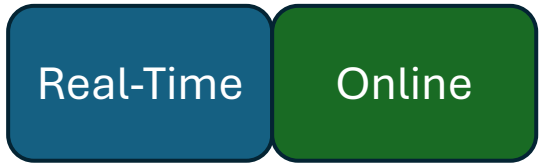


Zagadnienia: Pomiar opóźnienia w sieci

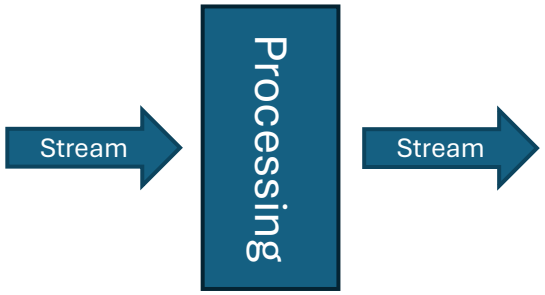
Real-Time

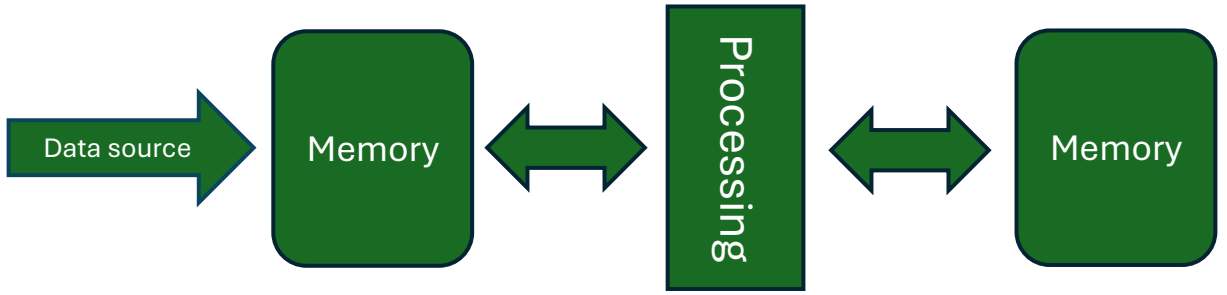
- Obsługa modułu GPS do odzyskania impulsu synchronizacyjnego PPS
  - RasberryPi/Arduino
- Obsługa modułu GSM do zdalnego sterowania i zbierania danych
  - RasberryPi/Arduino
- Implementacja infrastruktury do sterowania i prezentacji danych pomiarowych
  - C/Python/Web frameworks

Alveo jako akcelerator

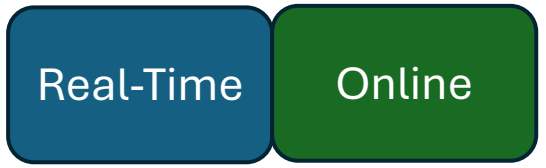


- HLS pozwala na implementację zaawansowanej algorytmiki
  - Komponenty w trybie Stream → real-time
    - Krytyczne wymagania czasowe
    - Sztywna częstotliwość zegara
    - Mniej złożona logika
  - Komponenty w trybie Memory-Mapped → online
    - Większa elastyczność
    - Bardziej złożone obliczeniowo operacje





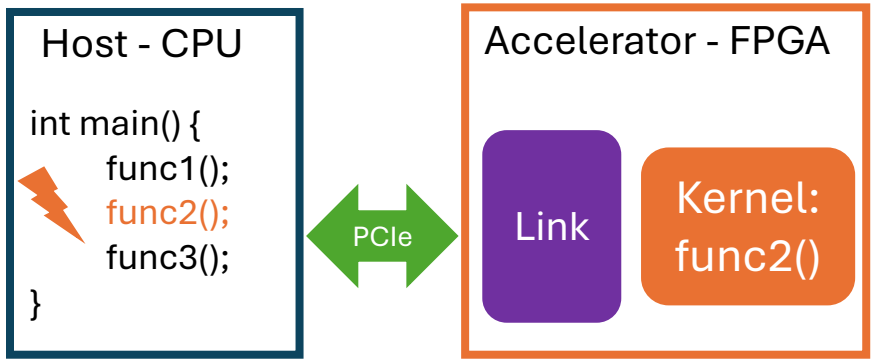
Akceleracja obliczeń

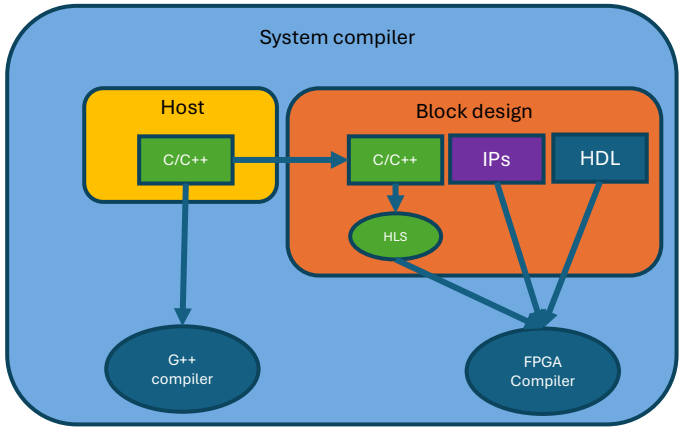


- Host-CPU
  - Konfiguruje akcelerator
  - Wykonuje główny wątek
  - Transferuje dane do i z akceleratora
  - Wywołuje akcelerator

  - Abstrakcje OpenCL
    - cl::Device, cl::Context, cl::Program
    - cl::Buffer, cl::CommandQueue, cl::Kernel
  - Abstrakcje Xilinx Runtime (XRT)
    - xrt::device, xrt::bo
    - pyxrt bindings

- Zintegrowane kompilatory
  - Konstruktory systemów

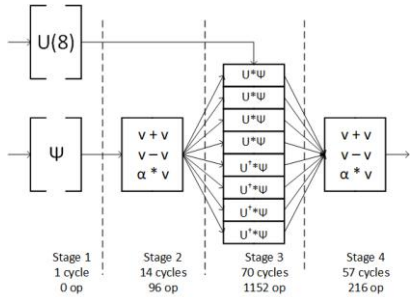





```python
from xrt_binding import *
import pyxrt

d = pyxrt.device(opt.index)
xbin = pyxrt.xclbin(opt.bitstreamFile)
uuid = d.load_xclbin(xbin)

simple = pyxrt.kernel(d, uuid, "simple")

boHandle1 = pyxrt.bo(d, 10, pyxrt.bo.normal, simple.group_id(0))
boHandle2 = pyxrt.bo(d, 10, pyxrt.bo.normal, simple.group_id(1))

boHandle1.sync(pyxrt.xclBOSyncDirection.XCL_BO_SYNC_BO_TO_DEVICE, 10, 0)
boHandle2.sync(pyxrt.xclBOSyncDirection.XCL_BO_SYNC_BO_TO_DEVICE, 10, 0)

run = simple(boHandle1, boHandle2, 0x10)
state = run.wait()

boHandle1.sync(pyxrt.xclBOSyncDirection.XCL_BO_SYNC_BO_FROM_DEVICE, 10, 0)
```

https://github.com/Xilinx/XRT/blob/master/tests/python/02_simple/main.py

Akceleracja obliczeń



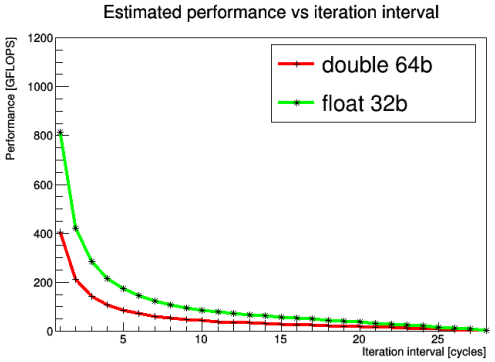
Online

- Benchmark HPCG
  - Implementacja w C++ algorytmu Gradientu Sprzężonego
  - Porównanie wydajność/Watt do GPU

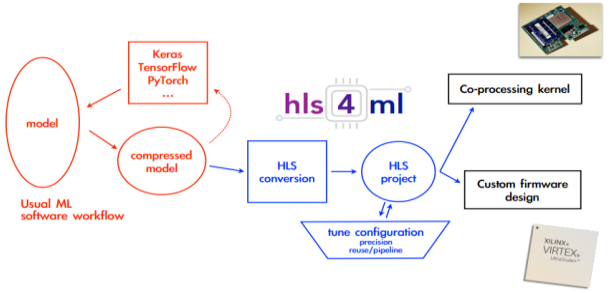
Towards Lattice Quantum Chromodynamics on FPGA devices - ScienceDirect

Towards Lattice Quantum Chromodynamics on FPGA devices ☆

Grzegorz Korcyl [a], Piotr Korcyl [b c]



- Sieci Neuronowe
  - Wykorzystanie HLS4ML
  - Poszukiwanie balansu zasoby/wydajność



- Systemy heterogeniczne
  - SYCL model programowania na OpenCL
  - Single source to any platform

```
#include <iostream>
#include <CL/sycl.hpp>

class vector_addition;

int main(int, char**) {
   cl::sycl::float4 a = { 1.0, 2.0, 3.0, 4.0 };
   cl::sycl::float4 b = { 4.0, 3.0, 2.0, 1.0 };
   cl::sycl::float4 c = { 0.0, 0.0, 0.0, 0.0 };

   cl::sycl::default_selector device_selector;

   cl::sycl::queue queue(device_selector);

   {
      cl::sycl::buffer<cl::sycl::float4, 1> a_sycl(&a, cl::sycl::range<1>(1));
      cl::sycl::buffer<cl::sycl::float4, 1> b_sycl(&b, cl::sycl::range<1>(1));
      cl::sycl::buffer<cl::sycl::float4, 1> c_sycl(&c, cl::sycl::range<1>(1));

      queue.submit([&] (cl::sycl::handler& cgh) {
         auto a_acc = a_sycl.get_access<cl::sycl::access::mode::read>(cgh);
         auto b_acc = b_sycl.get_access<cl::sycl::access::mode::read>(cgh);
         auto c_acc = c_sycl.get_access<cl::sycl::access::mode::discard_write>(cgh);

         cgh.single_task<class vector_addition>([=] () {
         c_acc[0] = a_acc[0] + b_acc[0];
         });
      });
   }

   return 0;
}
```

Zagadnienia: Akceleracja obliczeń

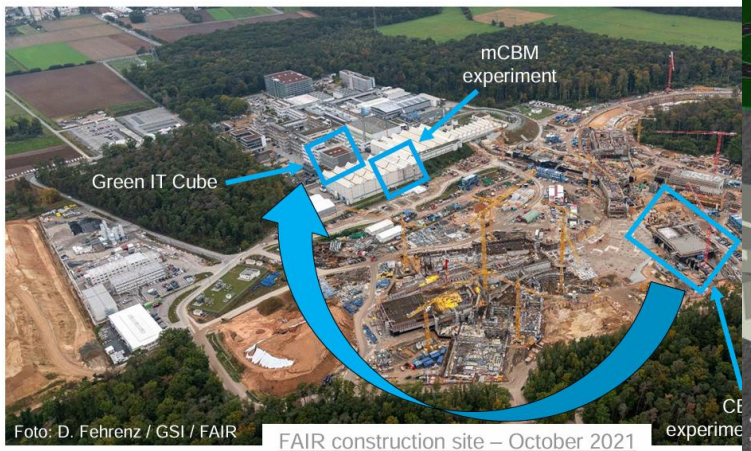
Online

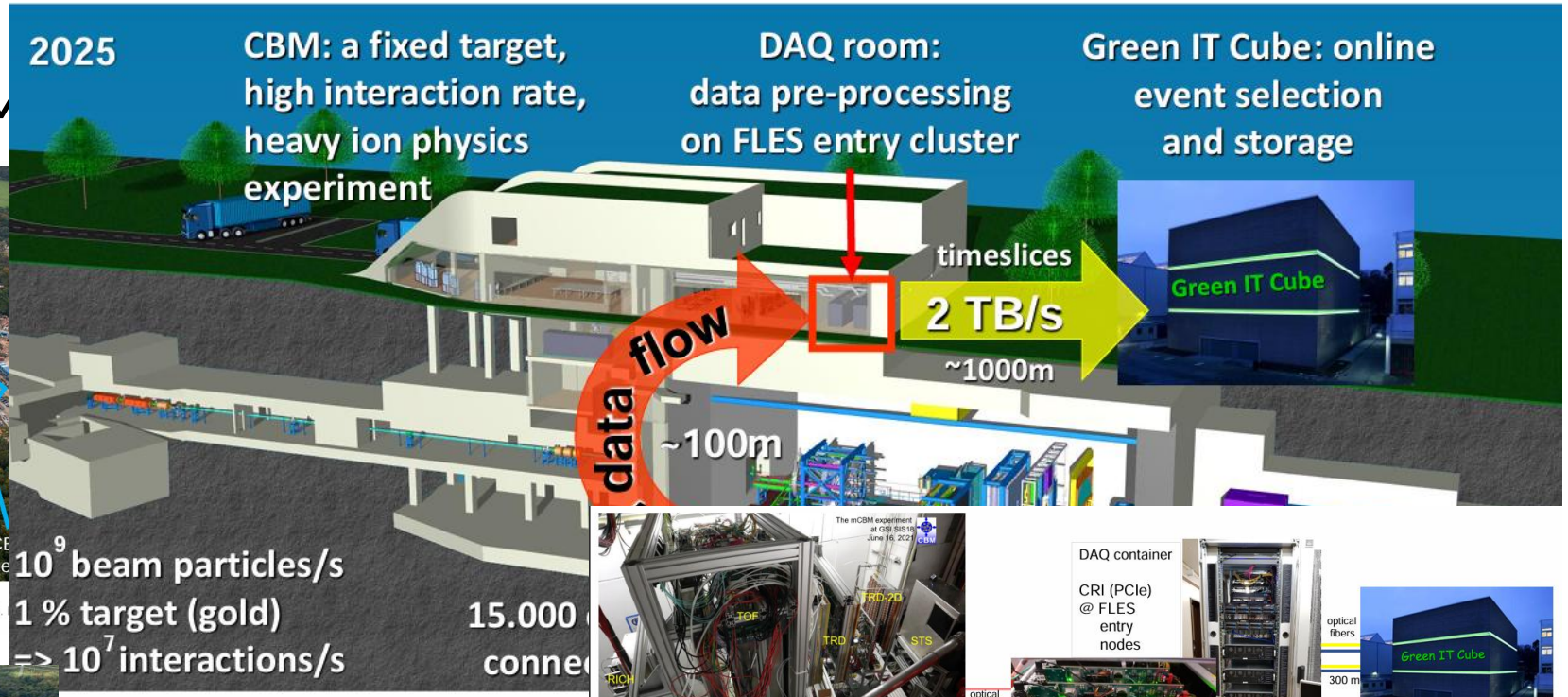
- Wykorzystanie zasobów HAC
  - 2 serwery z Alveo U280 i GPU
  - Infrastruktura sieciowa
- Zbadanie mechanizmu PCIe Peer-2-Peer
  - Bezpośrednie przesyłanie danych pomiędzy urządzeniami na magistrali
  - Odciążanie procesora
- Badanie akceleracji / optymalizacji energetycznej algorytmów
  - Inspirowane komputerami kwantowymi algorytmy implementowane na FPGA

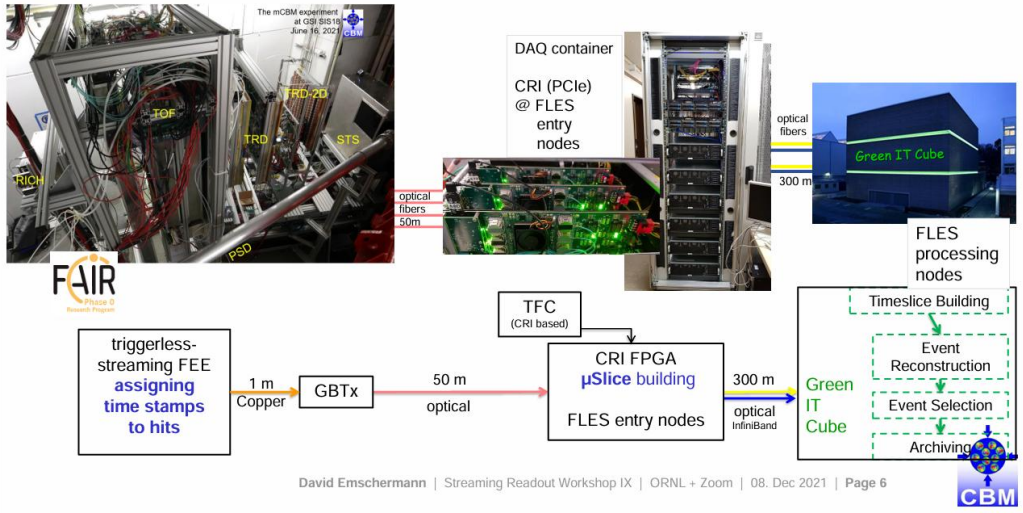
Masywne obliczenia

Real-Time

- Eksperyment CBM





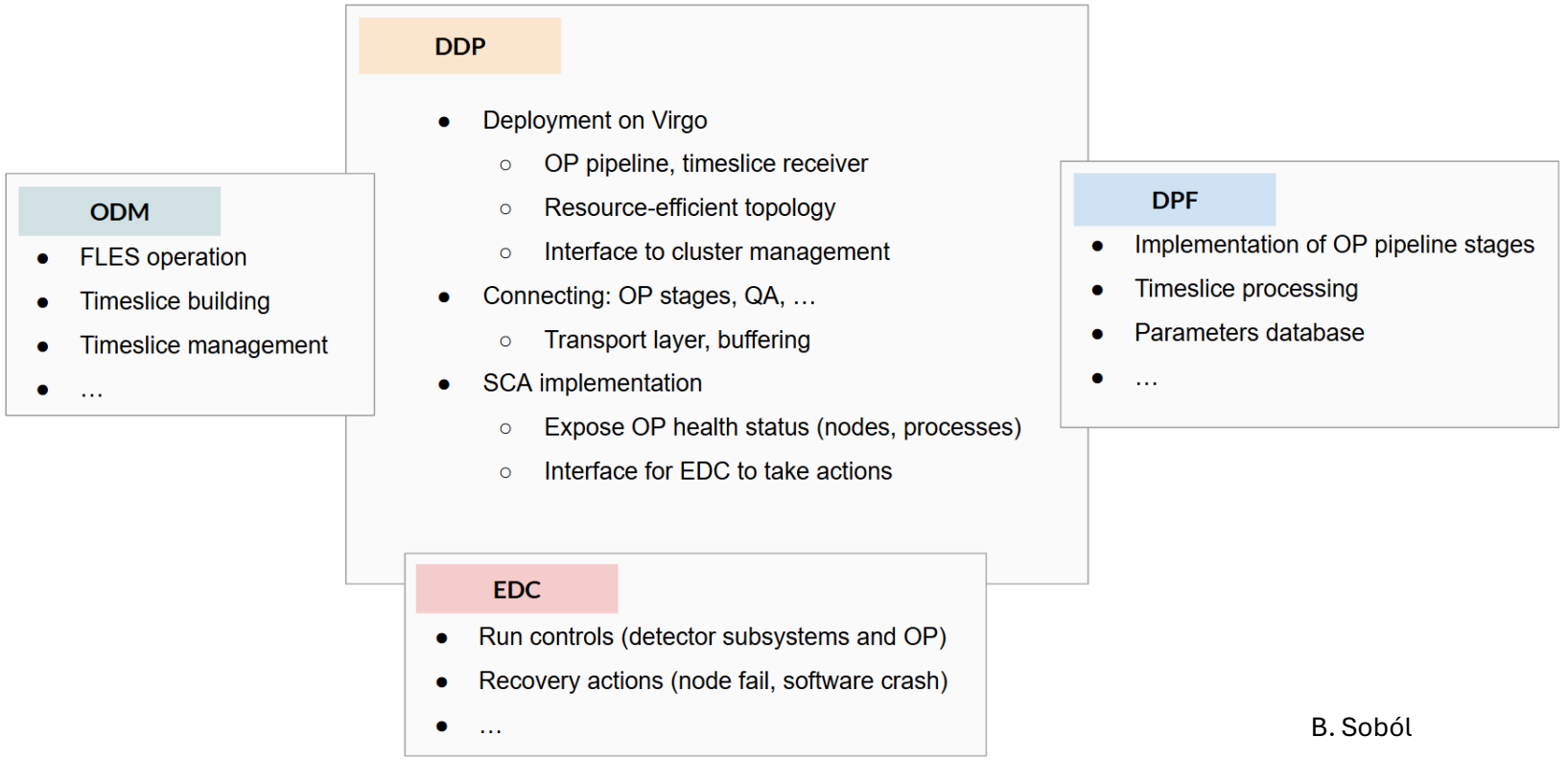


https://indico.phy.ornl.gov/event/112/contributions/566/attachments/492/1342/20211208_169_sro9_cbm_daq_v02.pdf

Masywne obliczenia

Real-Time

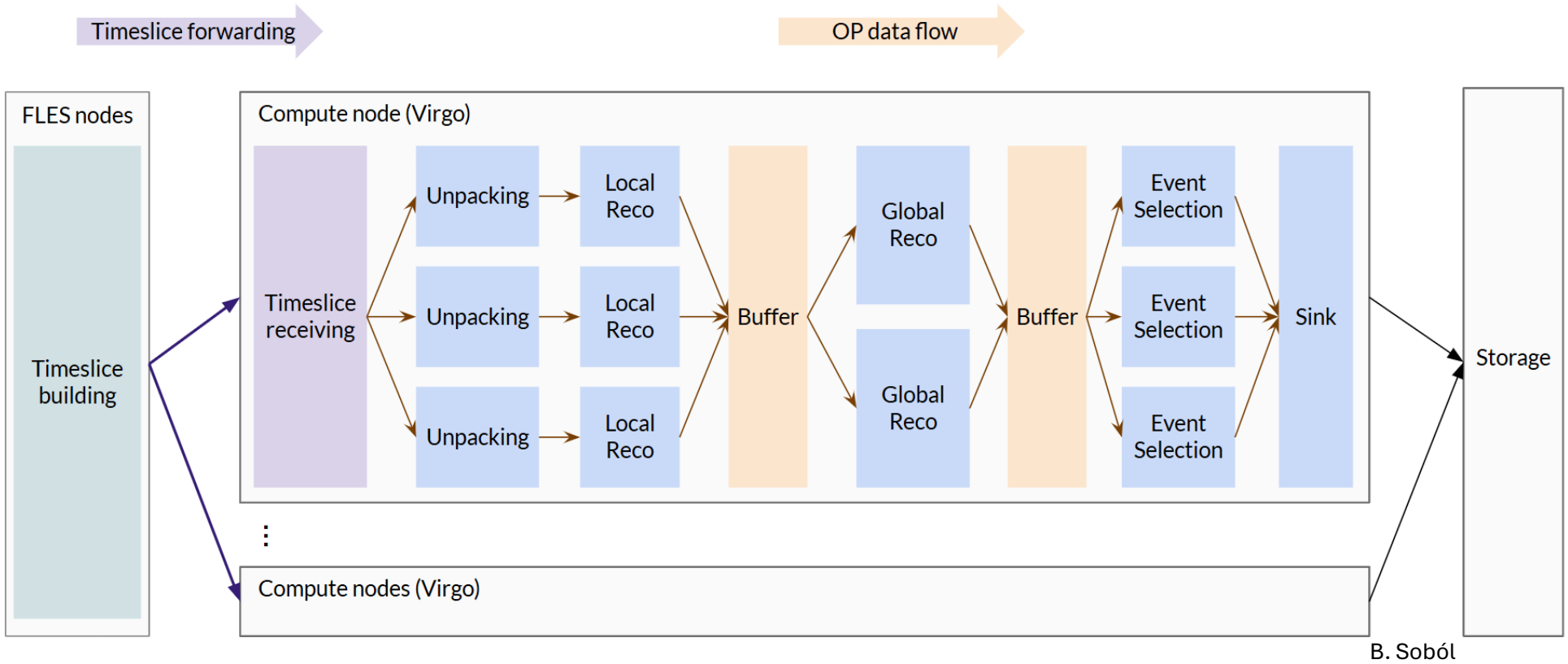
DDP in context



Masywne obliczenia

Real-Time

Main online data flow (concept)



B. Sobół

Zagadnienia: Masywne obliczenia

Real-Time

- CBM
  - RDMA over Infiniband vs RoCE
    - C/C++, porównanie praktyczne, benchmarking
  - Benchmarking sekcji algorytmów na zasobach HPC
    - C/C++, akceleracja GPU
  - Mechanizmy infrastrukturalne, sterowanie, monitorowanie procesów
    - C/C++
- FAIR
  - Systemy sterowania akceleratorem
  - Prace magisterskie/praktyki we współpracy z firmami