Jefferson Lab
Thomas Jefferson National Accelerator Facility

ML on FPGA for Event Selection

Sergey Furletov
Jefferson Lab

Workshop VIII on Streaming readout

29 Apr 2021

Outline

- Motivation
  - ✓ Examples from HEP experiments at LHC
  - ✓ EIC experiment
- ANN in FPGA
- ML in FPGA for physicists
  - ✓ Experimental setup
  - ✓ Offline PID analysis with ML (root / TMVA)
  - ✓ Moving on to FPGA
  - ✓ Run ML on FPGA
- Optimization ML in FPGA with hls4ml package
- Global PID with multiple PID detectors
- Hardware solutions
- Outlook

Team :
F. Barbosa, L. Belfore (ODU), C. Dickover, C. Fanelli (MIT),
Y. Furletova, L. Jokhovets (Jülich Research Centre, Germany),
D. Lawrence, D. Romanov

Motivation

- *With increase of luminosity for accelerator colliders as well as a granularity of detectors for particle physics, more challenges fall on the readout system and data transfer from detector front-end to computer farm and long term storage.*

- *Concepts of trigger-less readout and data streaming will produce large data volumes being read from the detectors.*

- *From a resource standpoint, it makes much more sense to perform data pre-processing and reduction at early stages of data streaming. Would allow to use information-rich data sets for event selection.*

Example from CMS

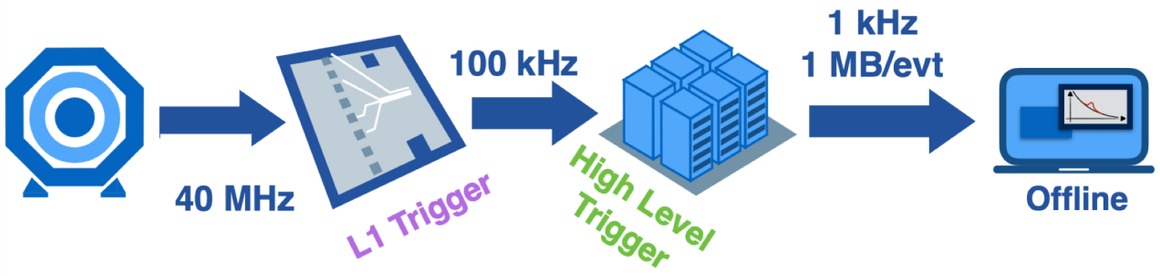
- CMS bandwidth: Phase-2 ~50 Tb/s (1.8TB/s in Phase-1)
- The task of the real-time processing is to filter events to reduce data rates to manageable levels for offline processing.

- Level-1 typically uses custom hardware with ASICs or FPGAs (decision ~4 μs)

- The second stage of triggering, High Level Trigger (HLT), uses commercial CPUs to process the filtered data in software. (decision ~100 000 μs)

Machine Learning Inference with FPGAs
Reconstruction, Trigger, and Machine Learning for the HL-LHC @ MIT
April 26th, 2018

Current CMS Data Processing



Level 1 Trigger (hardware)
Decision in ~4 us
99.75% rejected

High Level Trigger (software)
Decision in ~100 ms
99% rejected

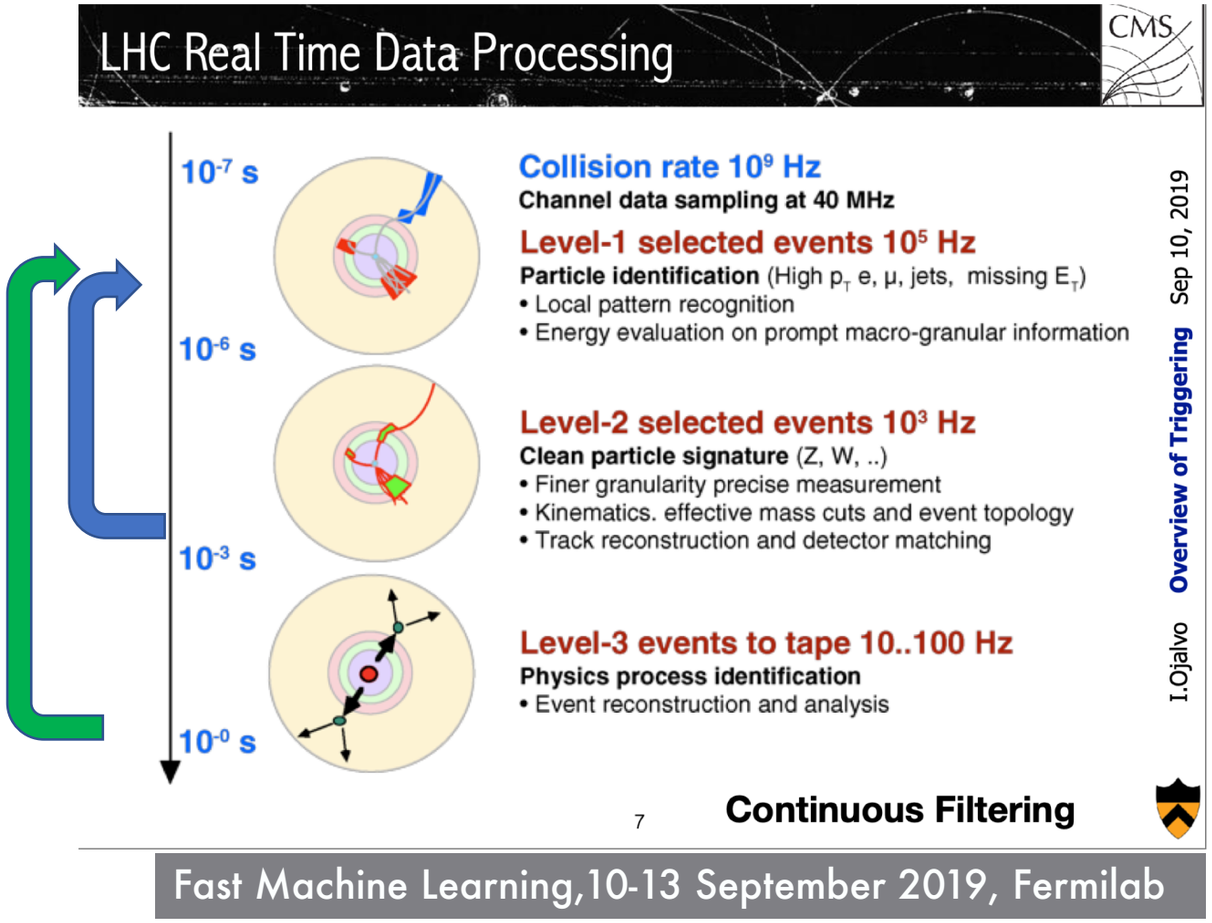
After trigger, 99.99975% of events are gone forever!

Motivation

- *The growing computational power of modern FPGA boards allows us to add more sophisticated algorithms for real time data processing.*
- *Many tasks could be solved using modern Machine Learning (ML) algorithms which are naturally suited for FPGA architectures.*

Level 1 works with Regional and sub-detector Trigger primitives

Using ML on FPGA many tasks from Level 2 and/or Level 3 can be performed at Level 1



ML in physics

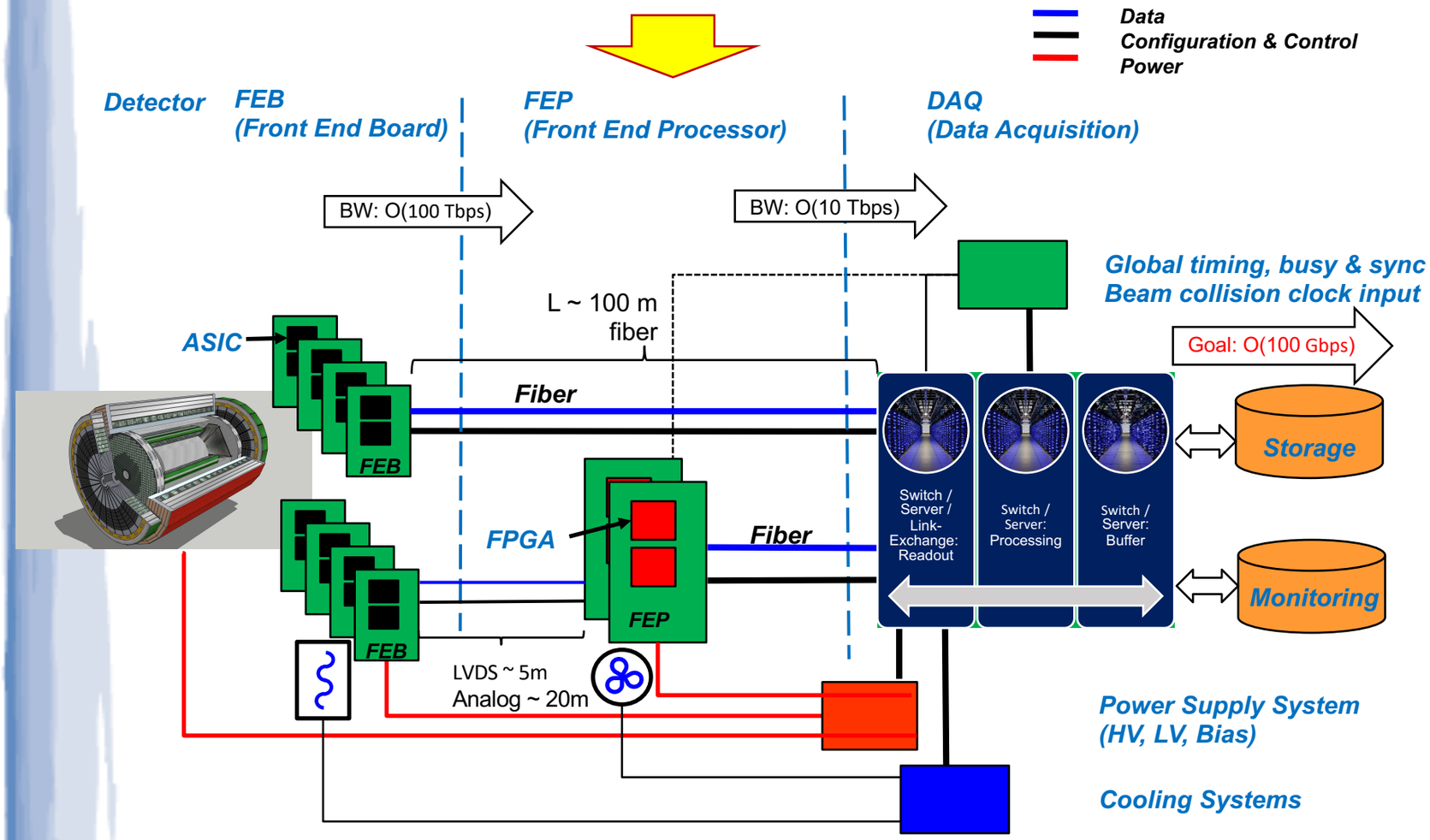
Jefferson Lab
Thomas Jefferson National Accelerator Facility

- Machine learning methods are widely used and have proven to be very powerful in particle physics.
- Although the methods of machine learning and artificial intelligence are developed by many groups and have a lot in common, nevertheless, the hardware used and performance is different:

  1) CPU only
  2) CPU and GPU accelerator
  3) CPU and FPGA accelerator
  4) pure FPGA

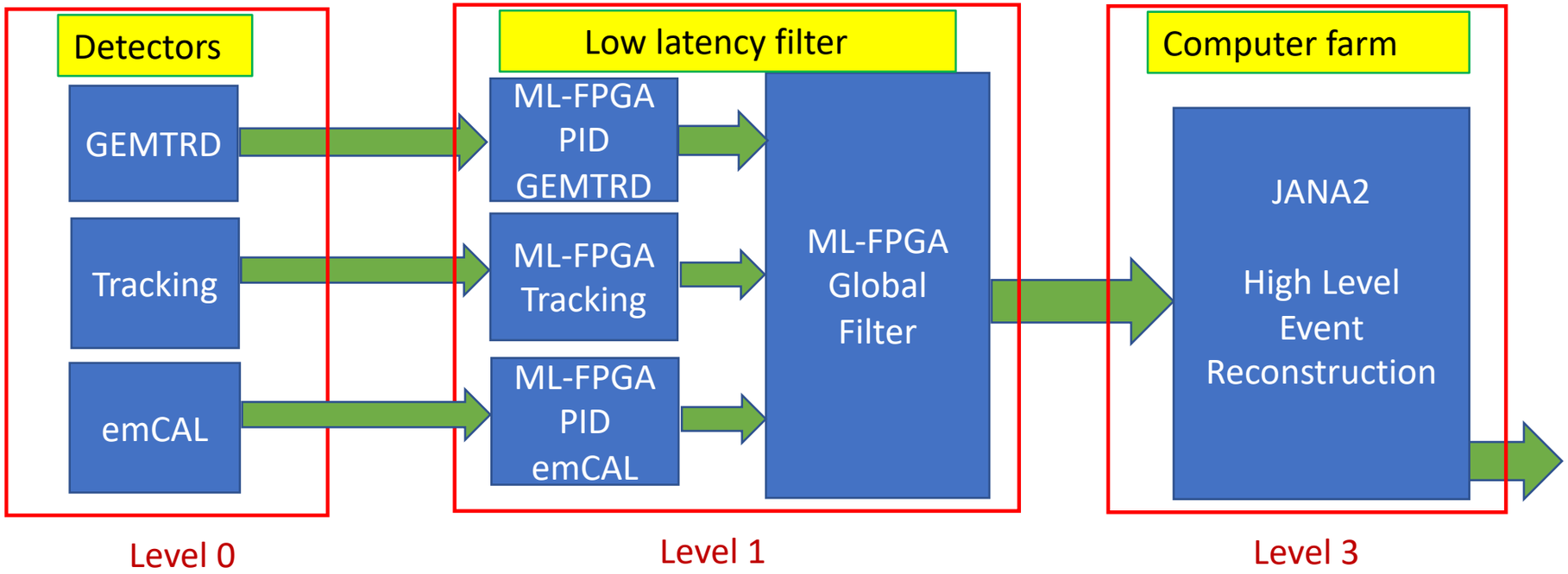
- While the large numerical processing capability of GPUs is attractive, these technologies are optimized for high throughput, not low latency.
- FPGA-based trigger and data acquisition systems have extremely low, sub-microsecond latency requirements that are unique to particle physics.
- Definitely FPGA can work on a computer farm as an ML accelerator, but the internal FPGA performance will be degraded due to slow I/O through the computer and the PCIe bus. Not to mention the latency, which will increase by 2-3 orders of magnitude.
- Therefore, the most effective would be the use of ML-FPGA directly between the front-end stream and a computer farm, on which it is already more efficient to use the CPU and GPU for ML/AI.

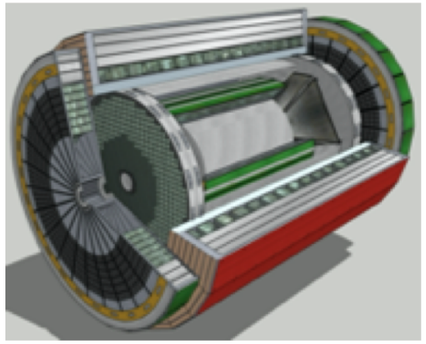
EIC readout

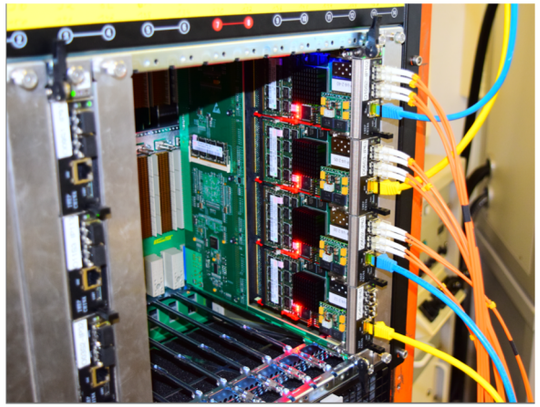


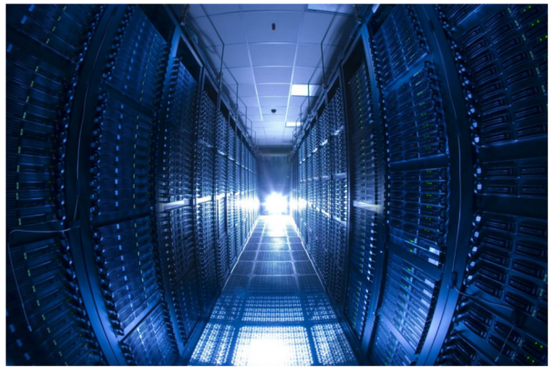
- *The correct location for the ML on the FPGA filter is called "FEP" in this figure.*
- *This gives us a chance to reduce traffic earlier.*
- *Allows us to touch physics: ML brings intelligence to L1.*
- *However, it is now unclear how far we can go with physics at the FPGA.*
- *Initially, we can start in pass-through mode.*
- *Then we can add background rejection.*
- *Later we can add filtering processes with the largest cross section.*
- *In case of problems with output traffic, we can add a selector for low cross section processes.*
- *The ML-on-FPGA solution complements the purely computer-based solution and mitigates DAQ performance risks.*

Filter design proposal







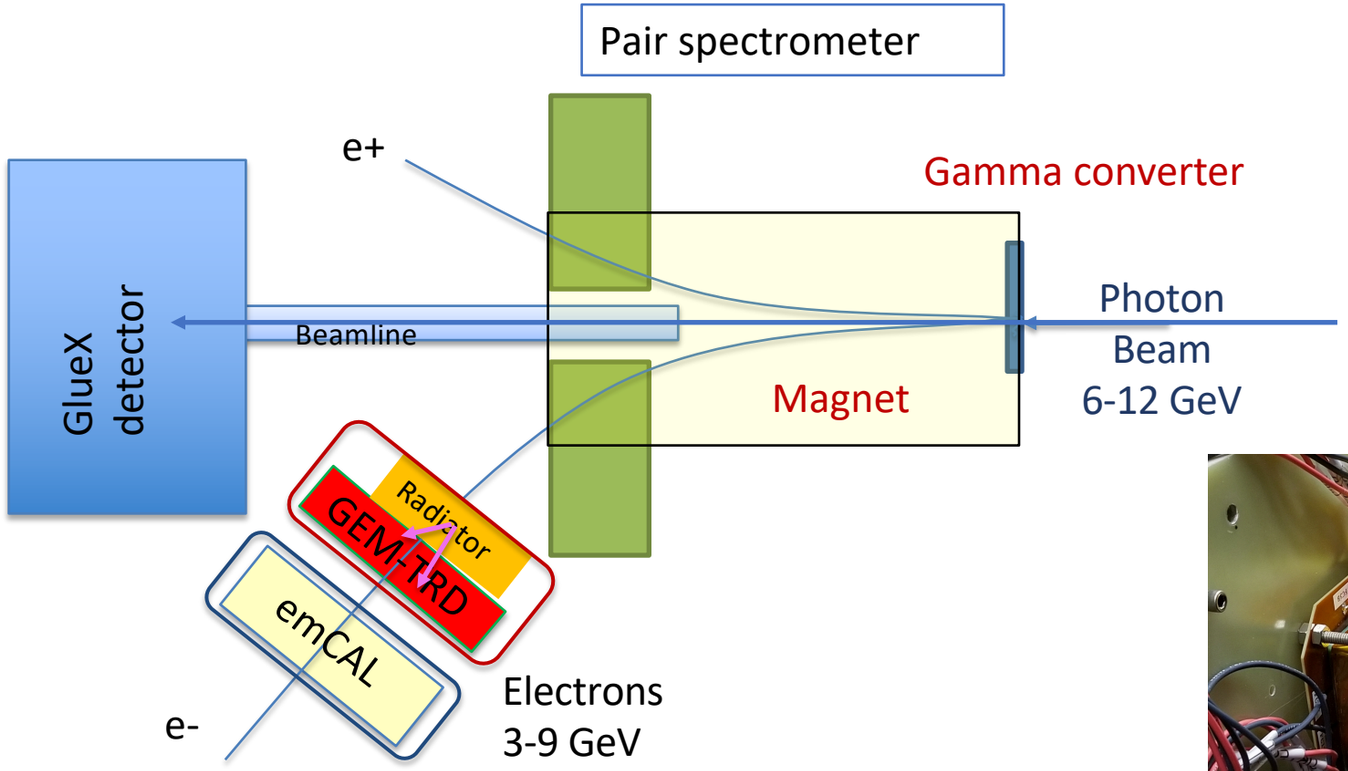


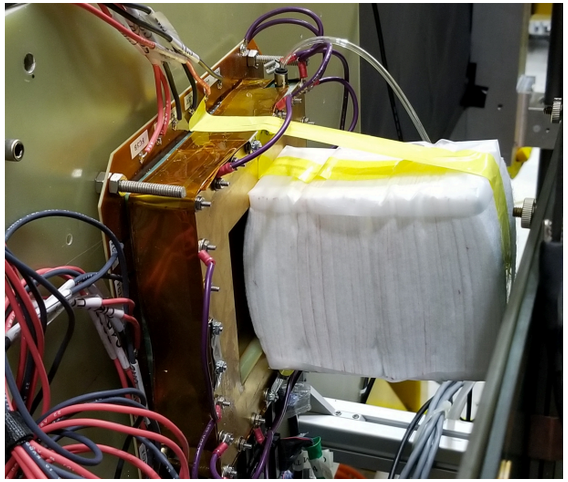
Images from the Internet are for illustration of scale only.

Beam test with GEMTRD and eCAL

Beam setup at JLab Hall-D

- *Tests were carried out using electrons with an energy of 3-6 GeV, produced in the converter of a pair spectrometer at the upstream of GlueX detector.*

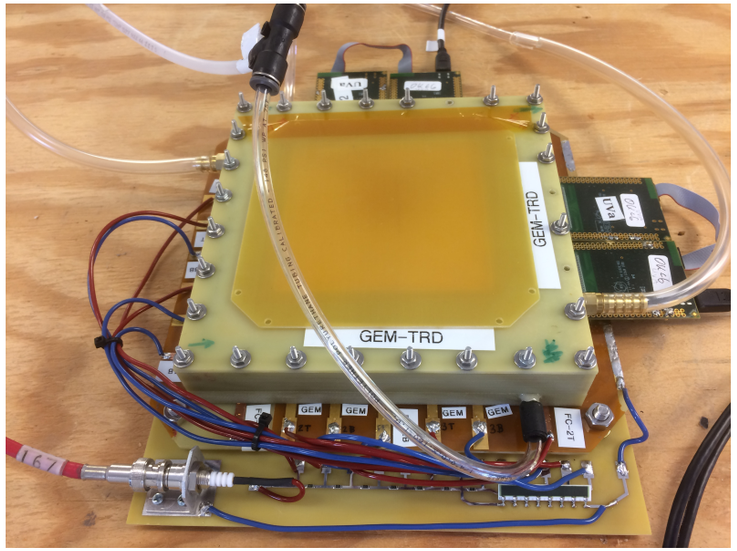


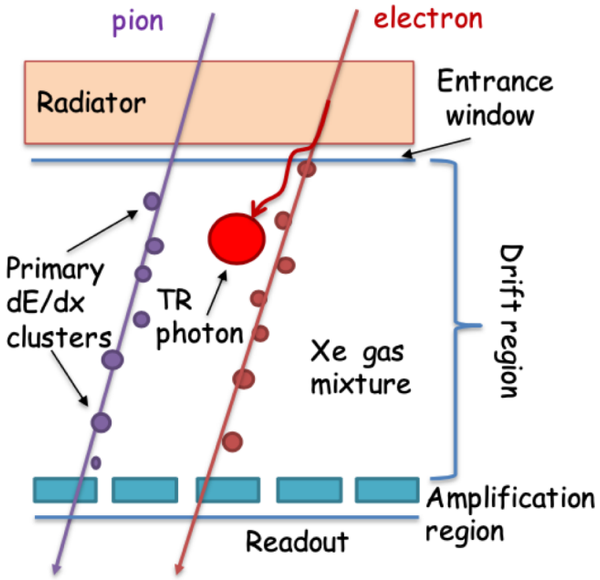


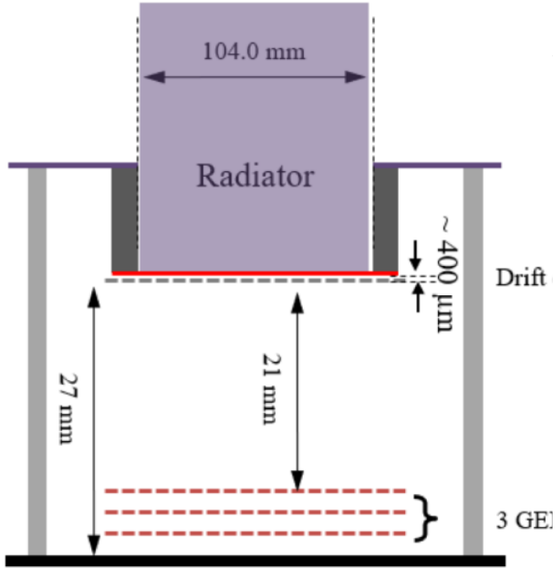
GEM-TRD prototype

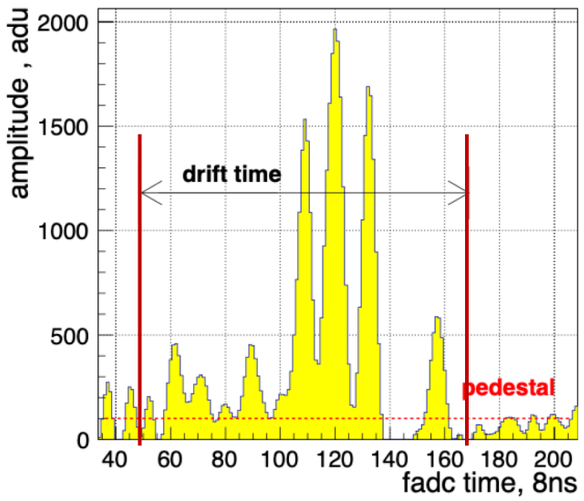
- A test module was built at the University of Virginia
- The prototype of GEMTRD/T module has a size of 10 cm × 10 cm with a corresponding to a total of 512 channels for X/Y coordinates.
- The readout is based on flash ADC system developed at JLAB (fADC125) @125 MHz sampling.

- GEM-TRD provides e/hadron separation and tracking



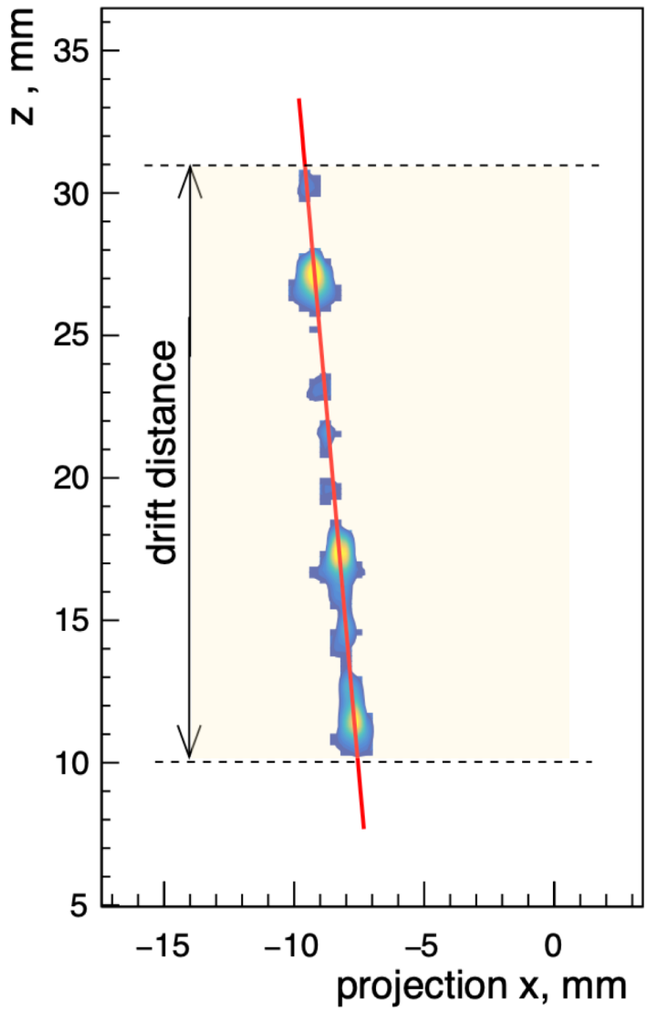


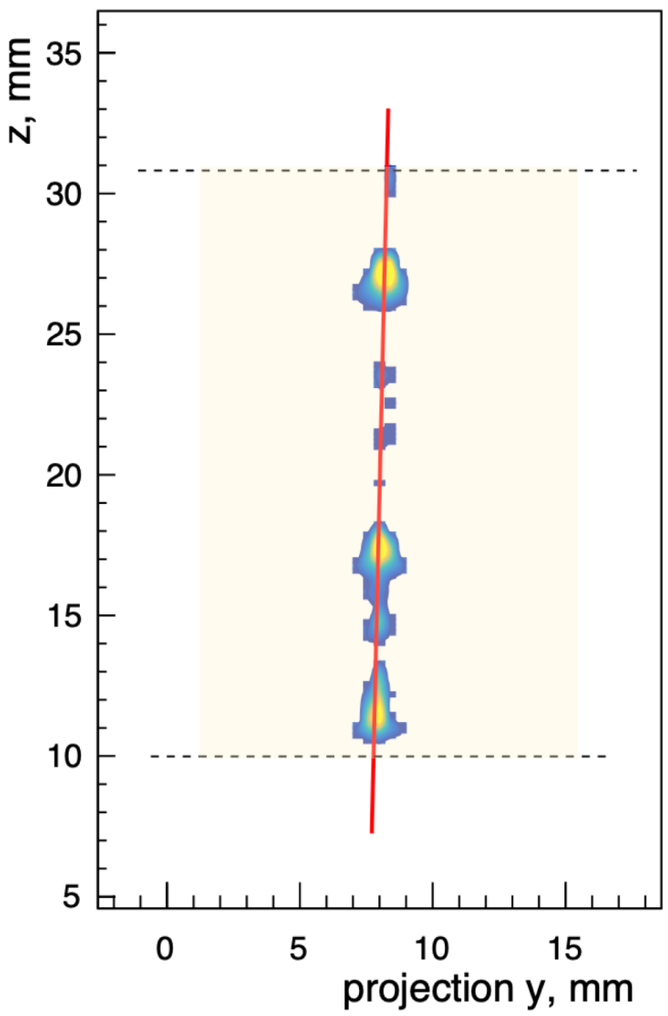




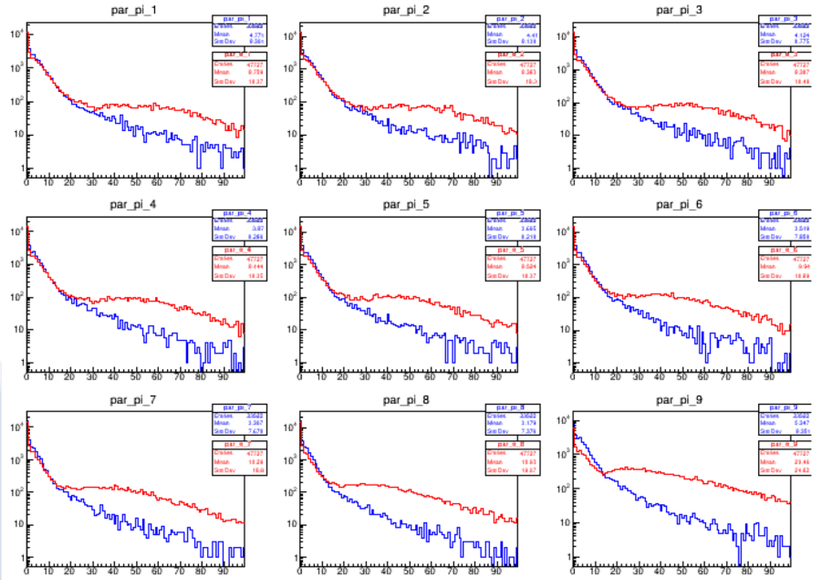
GEMTRD clusters on the track

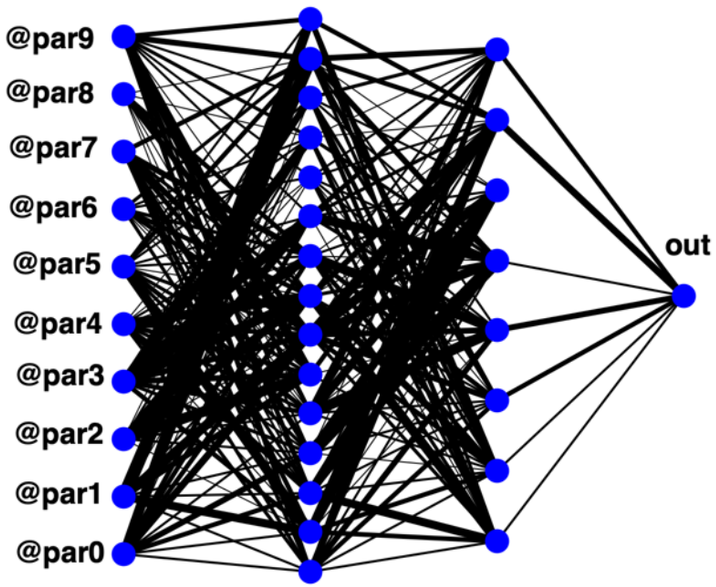
GEM-TRD can work as mini TPC, providing 3D track segments

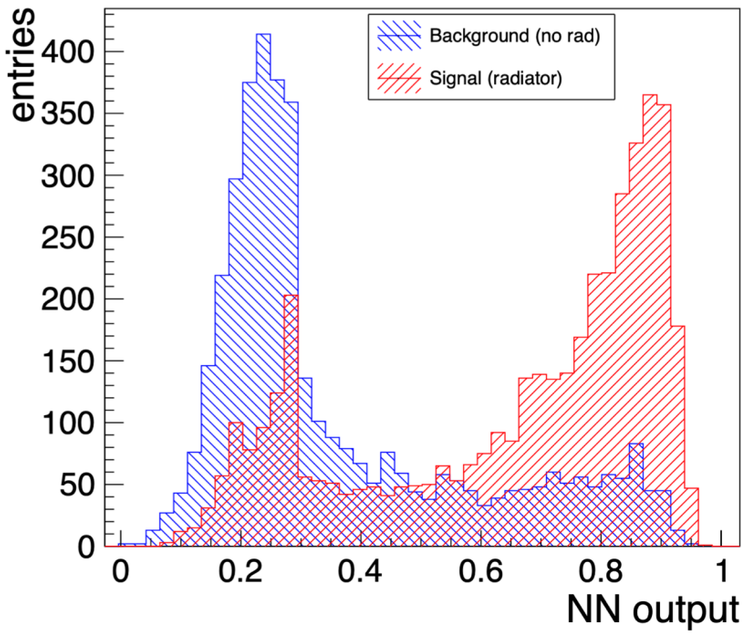




GEMTRD offline analysis







- *For data analysis we used a neural network library provided by root /TMVA package : MultiLayerPerceptron (MLP)*
- *All data was divided into 2 samples: training and test samples*
- *Top right plot shows neural network output for single module:*
  - Red - electrons with radiator
  - Blue – electrons without radiator

Moving forward

Jefferson Lab
Thomas Jefferson National Accelerator Facility

- *Offline analysis using ML looks promising.*
- *Can it be done in real time ?*
- *Here are some of the possible solutions :*
  - Computer farm.
  - CPU + GPU
  - CPU + FPGA
  - FPGA only

- *Steps to implement an FPGA solution:*
  - Select FPGA for application in ML
  - Export an offline trained neural network (NN) from root to C++ file.
  - Convert logical topology of NN coded in C++ to RTL structure of FPGA in VHDL or Verilog.
  - Optimize the NN for application in FPGA.
  - Create an I/O interface and configure FPGA.
  - Perform the test with hardware.

Artificial Neural Network

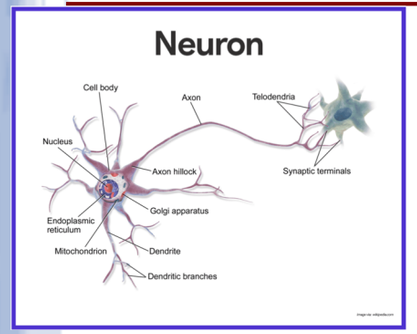
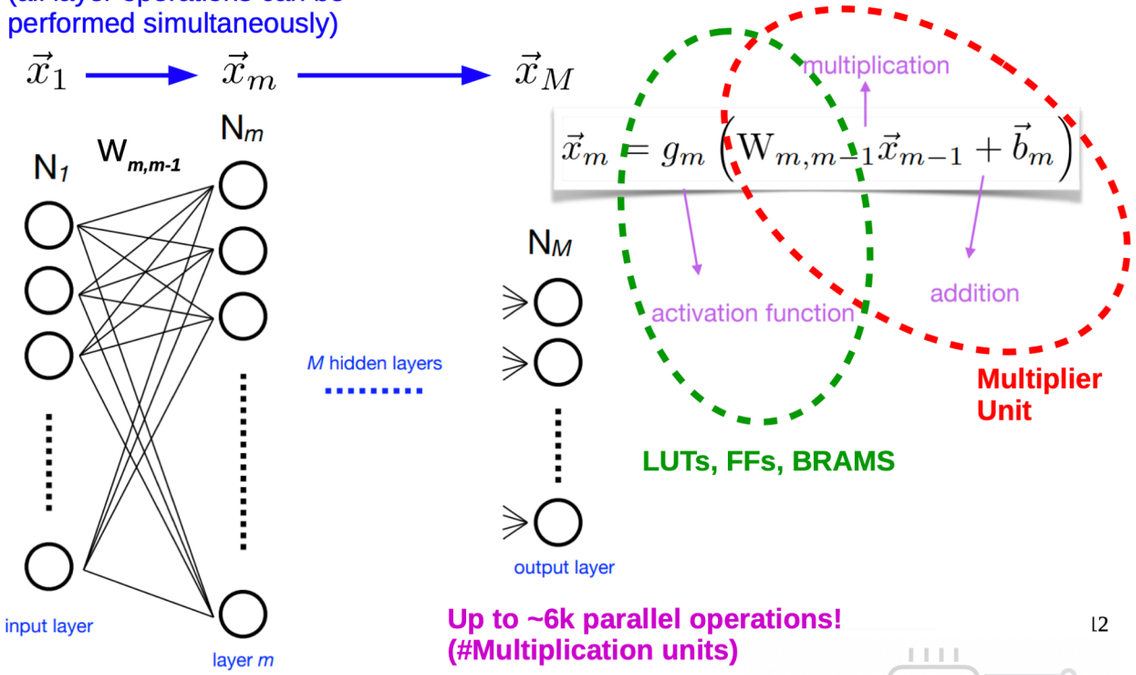


Image: https://nurseslabs.com/nervous-system/

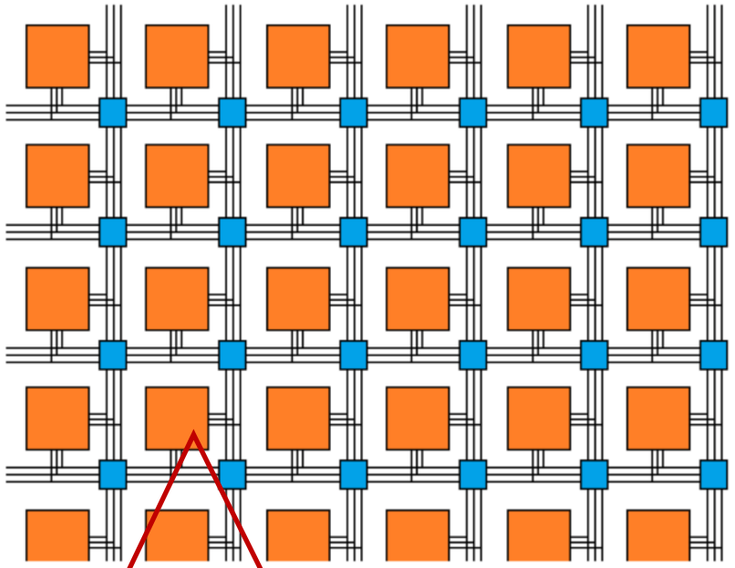
- FPGA Field Programmable Gate Array .
- It can perform logical operation in parallel

Inference on an FPGA

Every clock cycle
(all layer operations can be performed simultaneously)



Up to ~6k parallel operations!
(#Multiplication units)



"Clean slate" FPGA: programmable gates and routers

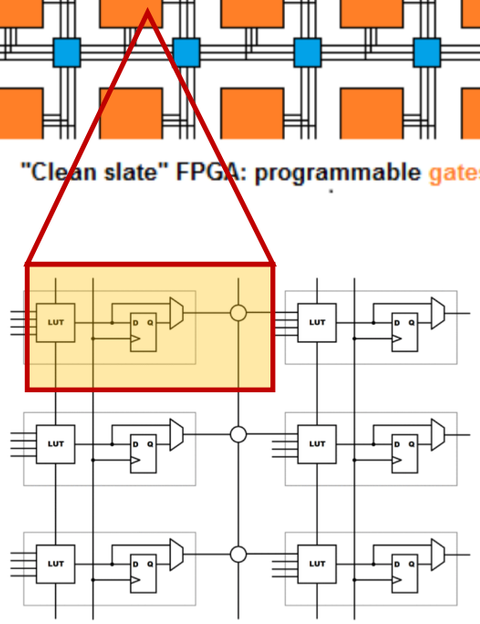


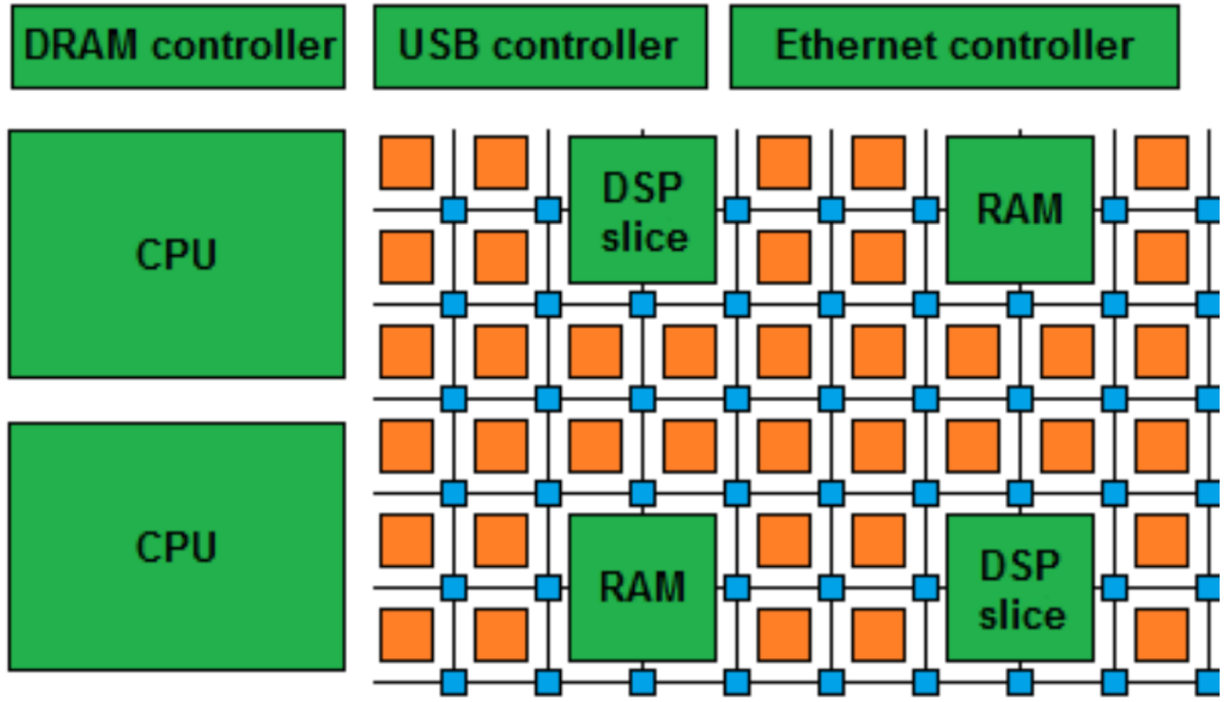
Image from: https://www.embeddedrelated.com/showarticle/195.php

IRIS-HEP th Febraury 13 , 2019 Dylan Rankin [MIT]

hls 4 ml

Modern FPGA

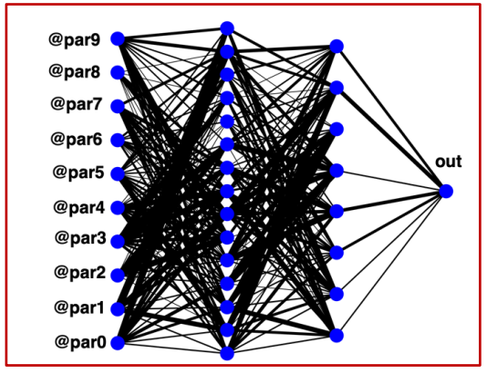
- Modern FPGAs have DSP slices - specialized hardware blocks placed between gateways and routers that perform mathematical calculations.
- The number of DSP slices can be up to 6000-12000 per chip.
- In addition, they often have ARM cores implemented using non-programmable gates.



Modern FPGA: lots of hard, not-field-programmable gates

Image from: https://www.embeddedrelated.com/showarticle/195.php

Xilinx HLS: C++ to Verilog



The C/C++ code of the trained network is used as input for Vivado_HLS.

The Xilinx Vivado HLS (High-Level Synthesis) tool provides a higher level of abstraction for the user by synthesizing functions written in C,C++ into IP blocks, by generating the appropriate ,low-level, VHDL and Verilog code. Then those blocks can be integrated into a real hardware system.

```
//----------------------------
// float_regex.sh:: converted to (tx_t)
//----------------------------
//---------- cxx file ---------
#include "trd_ann.h"
#include <cmath>
/*
fx_t ann(int index,fx_t in0,fx_t in1,fx_t in2,fx_t in3,fx_t in4,fx_t in5,fx_t in6,fx_t in7,
  input0 = (in0 - (fx_t)1.96805)/(fx_t)7.63362;
  input1 = (in1 - (fx_t)4.75766)/(fx_t)11.9138;
  input2 = (in2 - (fx_t)4.40589)/(fx_t)11.4831;
  input3 = (in3 - (fx_t)4.24519)/(fx_t)11.2533;
  input4 = (in4 - (fx_t)4.30175)/(fx_t)11.2252;
  input5 = (in5 - (fx_t)3.87414)/(fx_t)10.1781;
  input6 = (in6 - (fx_t)3.75959)/(fx_t)9.69367;
  input7 = (in7 - (fx_t)3.84352)/(fx_t)9.66213;
  input8 = (in8 - (fx_t)3.65047)/(fx_t)9.09565;
  input9 = (in9 - (fx_t)5.96775)/(fx_t)11.3203;
  switch(index) {
  case 0:
    return neuron0x32b4c90();
  default:
    return (fx_t)0.;
  }
}
*/
fout_t trdann(int index, finp_t input[10]) {
  input0 = (fx_t(input[0]) - (fx_t)1.96805)/(fx_t)7.63362;
  input1 = (fx_t(input[1]) - (fx_t)4.75766)/(fx_t)11.9138;
  input2 = (fx_t(input[2]) - (fx_t)4.40589)/(fx_t)11.4831;
  input3 = (fx_t(input[3]) - (fx_t)4.24519)/(fx_t)11.2533;
  input4 = (fx_t(input[4]) - (fx_t)4.30175)/(fx_t)11.2252;
  input5 = (fx_t(input[5]) - (fx_t)3.87414)/(fx_t)10.1781;
  input6 = (fx_t(input[6]) - (fx_t)3.75959)/(fx_t)9.69367;
  input7 = (fx_t(input[7]) - (fx_t)3.84352)/(fx_t)9.66213;
  input8 = (fx_t(input[8]) - (fx_t)3.65047)/(fx_t)9.09565;
  input9 = (fx_t(input[9]) - (fx_t)5.96775)/(fx_t)11.3203;
  switch(index) {
  case 0:
    return neuron0x32b4c90();
  default:
    return (fx_t)0.;
  }
}

fx_t neuron0x32bf850() {
  return input0;
}

fx_t neuron0x32bf190() {
  return input1;
}

fx_t neuron0x32bf4d0() {
  return input2;
}
```

C++

Note: fixed point calculation

Thanks to Ben Raydo for help.

```
// ==============================================================
// RTL generated by Vivado(TM) HLS - High-Level Synthesis from C, C++ and SystemC
// Version: 2019.1
// Copyright (C) 1986-2019 Xilinx, Inc. All Rights Reserved.
//
// ==============================================================

`timescale 1 ns / 1 ps

(* CORE_GENERATION_INFO="trdann,hls_ip_2019_1,{HLS_INPUT_TYPE=cxx,HLS_INPUT_FLOAT=1

module trdann (
        ap_clk,
        ap_rst_n,
        s_axi_AXILiteS_AWVALID,
        s_axi_AXILiteS_AWREADY,
        s_axi_AXILiteS_AWADDR,
        s_axi_AXILiteS_WVALID,
        s_axi_AXILiteS_WREADY,
        s_axi_AXILiteS_WDATA,
        s_axi_AXILiteS_WSTRB,
        s_axi_AXILiteS_ARVALID,
        s_axi_AXILiteS_ARREADY,
        s_axi_AXILiteS_ARADDR,
        s_axi_AXILiteS_RVALID,
        s_axi_AXILiteS_RREADY,
        s_axi_AXILiteS_RDATA,
        s_axi_AXILiteS_RRESP,
        s_axi_AXILiteS_BVALID,
        s_axi_AXILiteS_BREADY,
        s_axi_AXILiteS_BRESP,
        interrupt
);

parameter    ap_ST_fsm_state1 = 23'd1;
parameter    ap_ST_fsm_state2 = 23'd2;
parameter    ap_ST_fsm_state3 = 23'd4;
parameter    ap_ST_fsm_state4 = 23'd8;
parameter    ap_ST_fsm_state5 = 23'd16;
parameter    ap_ST_fsm_state6 = 23'd32;
parameter    ap_ST_fsm_state7 = 23'd64;
parameter    ap_ST_fsm_state8 = 23'd128;
parameter    ap_ST_fsm_state9 = 23'd256;
parameter    ap_ST_fsm_state10 = 23'd512;
parameter    ap_ST_fsm_state11 = 23'd1024;
parameter    ap_ST_fsm_state12 = 23'd2048;
parameter    ap_ST_fsm_state13 = 23'd4096;
parameter    ap_ST_fsm_state14 = 23'd8192;
parameter    ap_ST_fsm_state15 = 23'd16384;
parameter    ap_ST_fsm_state16 = 23'd32768;
parameter    ap_ST_fsm_state17 = 23'd65536;
parameter    ap_ST_fsm_state18 = 23'd131072;
parameter    ap_ST_fsm_state19 = 23'd262144;
parameter    ap_ST_fsm_state20 = 23'd524288;
parameter    ap_ST_fsm_state21 = 23'd1048576;
```

Verilog

ML FPGA Core for TRD

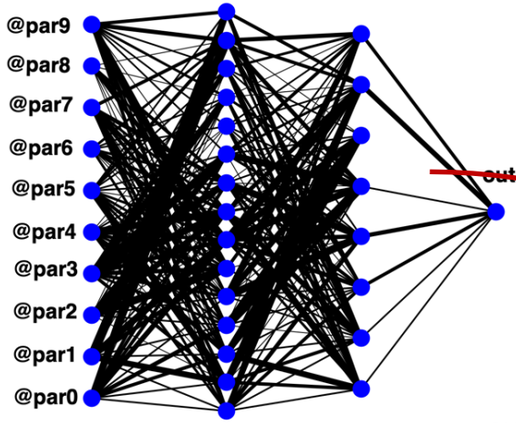
- *Using HLS significantly decreases development time. (at the cost of lower efficiency of use of FPGA resources)*

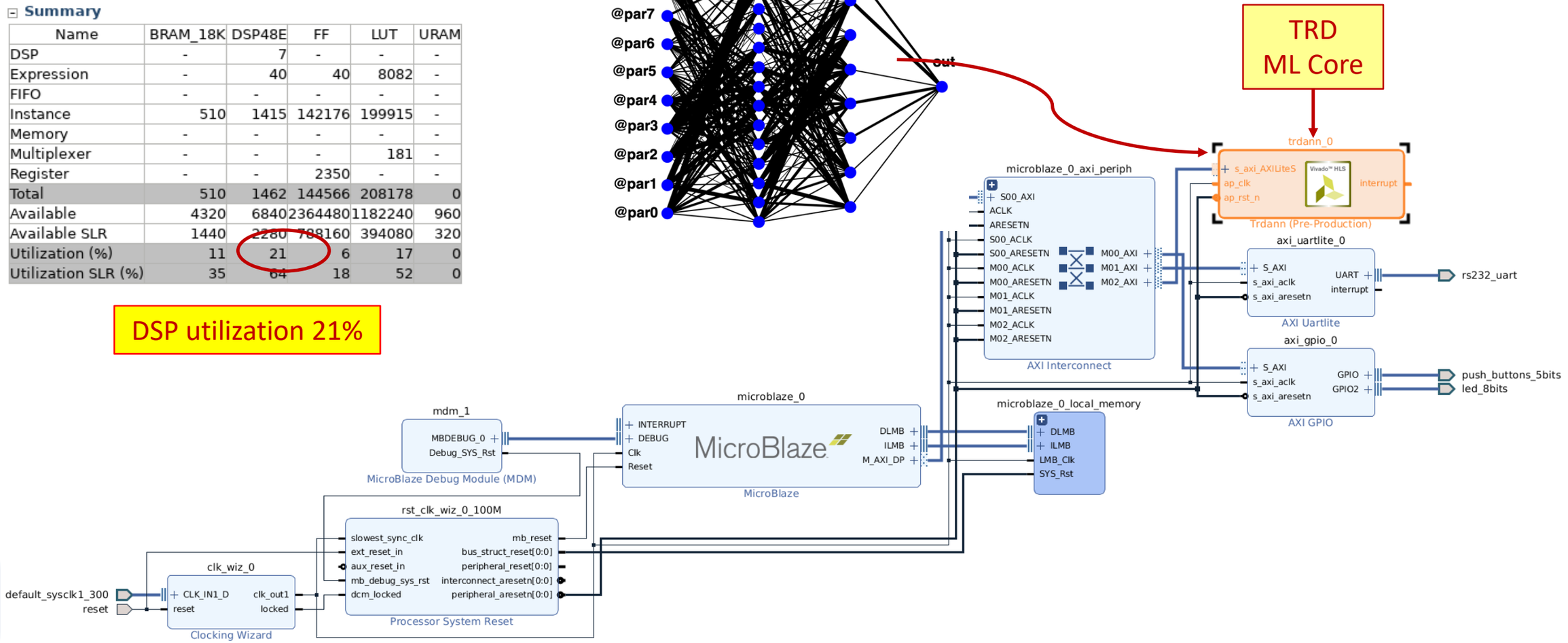
Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	7	-	-	-
Expression	-	40	40	8082	-
FIFO	-	-	-	-	-
Instance	510	1415	142176	199915	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	181	-
Register	-	-	2350	-	-
Total	510	1462	144566	208178	0
Available	4320	6840	2364480	1182240	960
Available SLR	1440	2280	788160	394080	320
Utilization (%)	11	21	6	17	0
Utilization SLR (%)	35	64	18	52	0

DSP utilization 21%



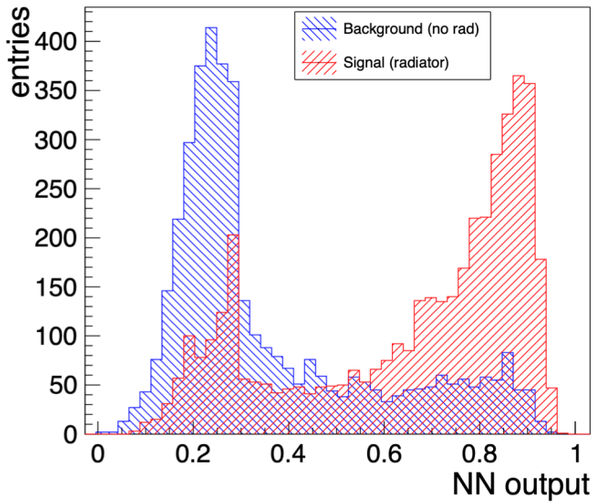


Test ML FPGA

Test tools:

1. Vivado SDK
2. Petalinux

```
ev=0 out=0.192 out0=0.197
ev=1 out=0.192 out0=0.197
ev=2 out=0.233 out0=0.236
ev=3 out=0.192 out0=0.197
ev=4 out=0.165 out0=0.169
ev=5 out=0.192 out0=0.196
ev=6 out=0.462 out0=0.470
ev=7 out=0.187 out0=0.191
```



C++ code for test :

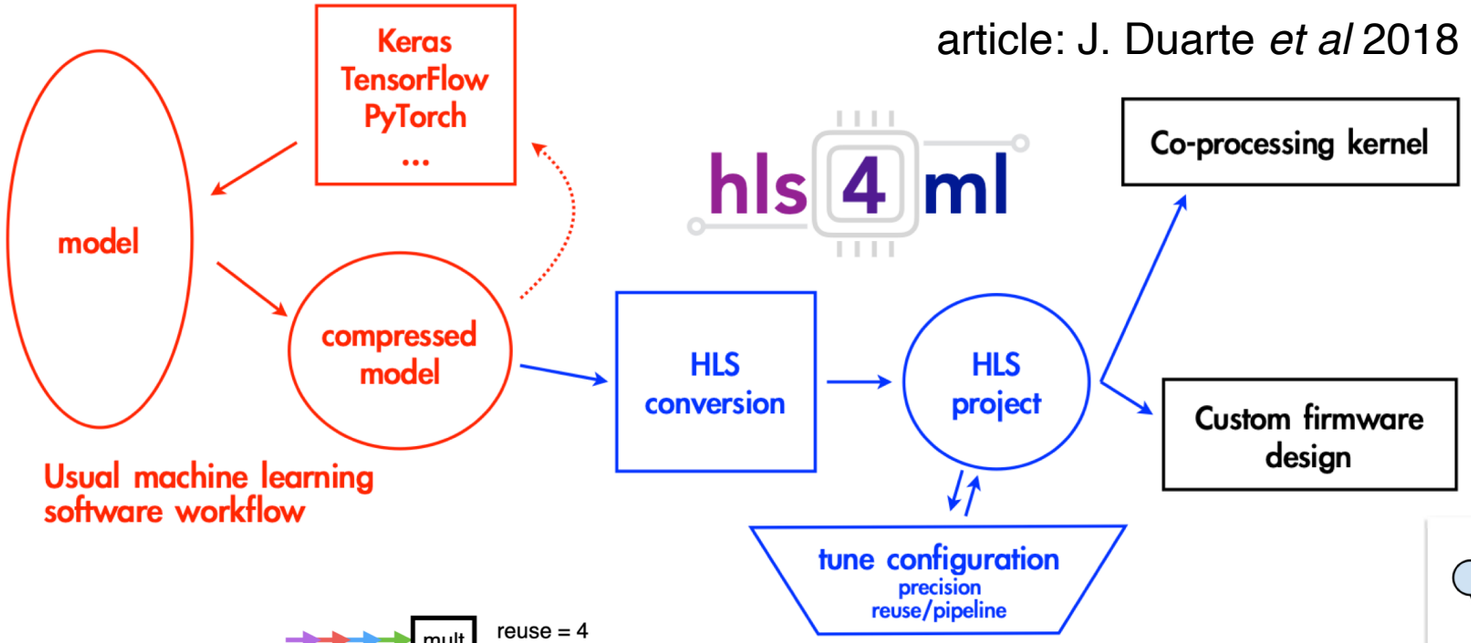
XTrdann ann; // create an instance of ML core.

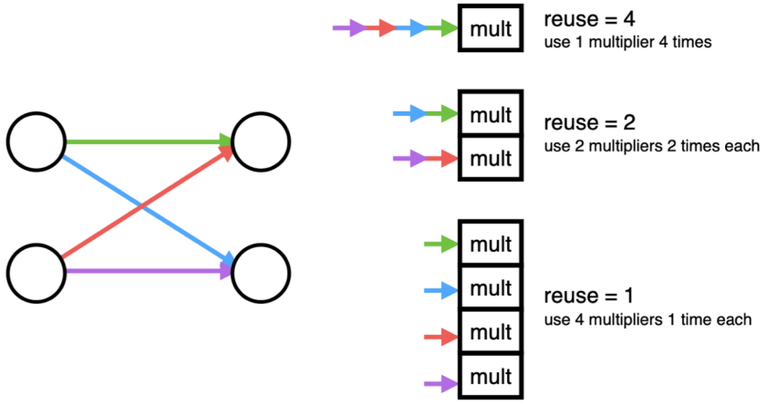
```
XTrdann ann;
int ret = XTrdann_Initialize(&ann, 0);

xil_printf(" XTrdann_Initialize =%d \n\r", ret);

XTrdann_Start(&ann);
xil_printf(" XTrdann_Started \n\r");

for (int i = 0; i < 8 ; i++ ) {


         for (int k=0; k<10; k++)
             params[k]=data[i][k];
         out0=data[i][10];

        ann_stat(&ann);

        int offset=0;
        int retw = XTrdann_Write_input_r_Words(&ann, offset, (u32*)&params[0], 10);
        xil_printf("Set Input ret=%d \n\r", retw);
        XTrdann_Set_index(&ann, 0);

        XTrdann_Start(&ann);

        while (!XTrdann_IsReady(&ann))
                ann_stat(&ann);
        ann_stat(&ann);

        int h1=out0;  int d1=(out0-h1)*1000;

        float *xout; //  *xin0, *xin1, *xin2;
        u32 iout = XTrdann_Get_return(&ann);
        xout = (float*) &iout;
        int whole = *xout;
        int thousandths = (*xout - whole) * 1000;
        if (whole==0 && thousandths<0)
                xil_printf("xout=-%d.%03d out0=%d.%03d\n\r", whole,-thousandths,h1,d1);
        else
                xil_printf("xout=+%d.%03d out0=%d.%03d\n\r", whole, thousandths,h1,d1);

        //u32 in0 = XTrdann_Get_in0(&ann); xin0 = (float*) &in0; int hin0 = *xin0 ; int din0=(*xin0-hin0)*1000;
        //u32 in1 = XTrdann_Get_in1(&ann); xin1 = (float*) &in1; int hin1 = *xin1 ; int din1=(*xin1-hin1)*1000;
        //u32 in2 = XTrdann_Get_in2(&ann); xin2 = (float*) &in2; int hin2 = *xin2 ; int din2=(*xin2-hin2)*1000;
        //xil_printf(" XTrdann in0=%d.%03d", hin0,din0);
        //xil_printf(" in1=%d.%03d ",hin1,din1);
        //xil_printf(" in2=%d.%03d ",hin2,din2);
        xil_printf(" ev=%d out=%d.%03d out0=%d.%03d\n\r",i,whole,thousandths,h1,d1);
}
```

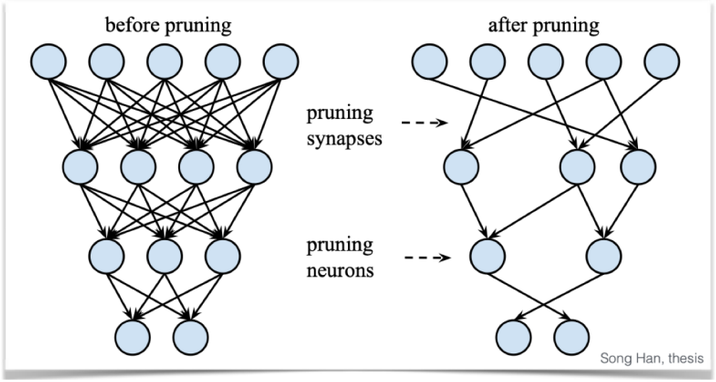
Optimization with hls4ml package

- A package hls4ml is developed based on High-Level Synthesis (HLS) to build machine learning models in FPGAs.

article: J. Duarte *et al* 2018 *JINST* **13** P07027







https://fastmachinelearning.org/hls4ml/

GEMTRD neural network optimization

Jefferson Lab
Thomas Jefferson National Accelerator Facility

Full size neural network, accuracy-optimized.

```
+ Timing (ns):
    * Summary:
    +--------+-------+----------+------------+
    |  Clock | Target| Estimated| Uncertainty|
    +--------+-------+----------+------------+
    |ap_clk  |   5.00|     3.968|        0.62|
    +--------+-------+----------+------------+

+ Latency (clock cycles):
    * Summary:
    +-----+-----+-----+-----+----------+
    |  Latency  |  Interval | Pipeline |
    | min | max | min | max |   Type   |
    +-----+-----+-----+-----+----------+
    |   15|   15|    1|    1| function |
    +-----+-----+-----+-----+----------+
```

Latency = 75ns

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	2	-	-	-
Expression	-	-	0	24	-
FIFO	-	-	-	-	-
Instance	19	692	3737	16446	-
Memory	2	-	0	0	-
Multiplexer	-	-	-	36	-
Register	-	-	1532	-	-
Total	21	694	5269	16506	0
Available SLR	1440	2280	788160	394080	320
Utilization SLR (%)	1	30	~0	4	0
Available	4320	6840	2364480	1182240	960
Utilization (%)	~0	10	~0	1	0

DSP utilization 10%

Size-optimized neural network

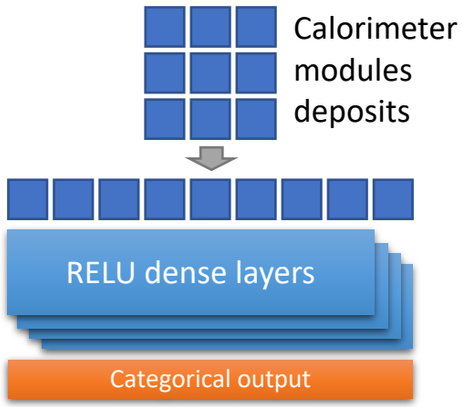
```
+ Timing (ns):
    * Summary:
    +--------+-------+----------+------------+
    |  Clock | Target| Estimated| Uncertainty|
    +--------+-------+----------+------------+
    |ap_clk  |   5.00|     3.883|        0.62|
    +--------+-------+----------+------------+

+ Latency (clock cycles):
    * Summary:
    +-----+-----+-----+-----+----------+
    |  Latency  |  Interval | Pipeline |
    | min | max | min | max |   Type   |
    +-----+-----+-----+-----+----------+
    |   17|   17|    3|    3| function |
    +-----+-----+-----+-----+----------+
```

Latency = 85ns

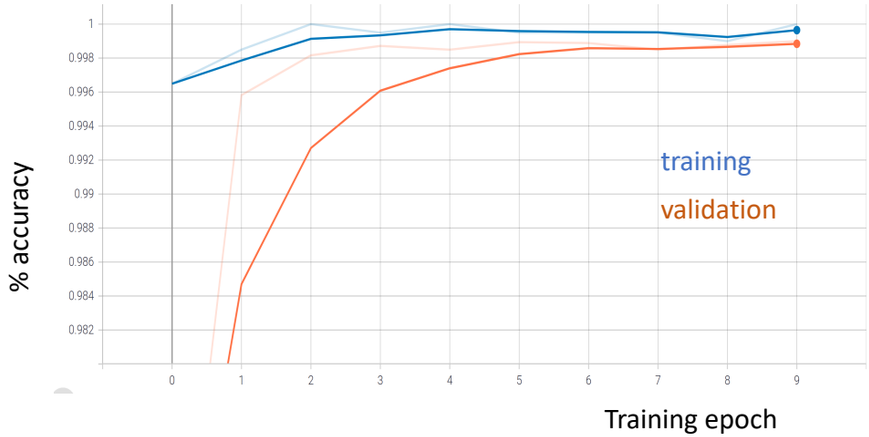
Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	2	-	-	-
Expression	-	-	0	24	-
FIFO	-	-	-	-	-
Instance	-	177	3132	10696	-
Memory	2	-	0	0	-
Multiplexer	-	-	-	81	-
Register	-	-	1423	-	-
Total	2	179	4555	10801	0
Available SLR	1440	2280	788160	394080	320
Utilization SLR (%)	~0	7	~0	2	0
Available	4320	6840	2364480	1182240	960
Utilization (%)	~0	2	~0	~0	0

DSP utilization 2%

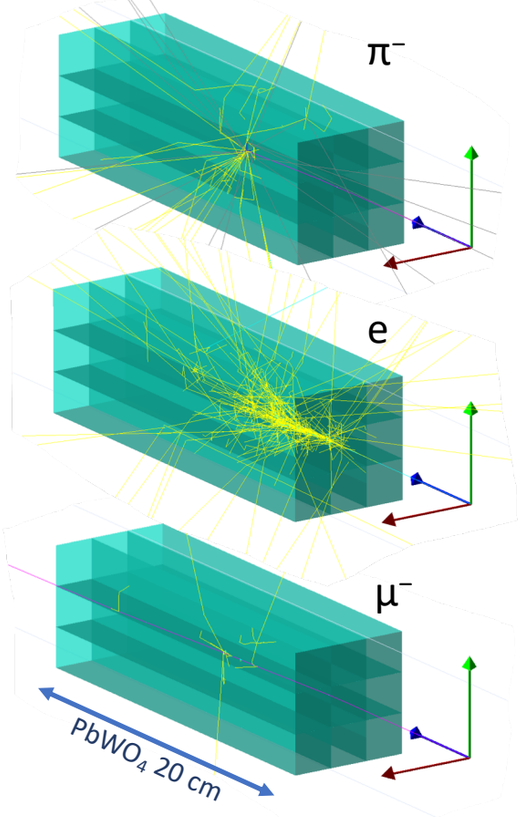
ML for Calorimeter e/pi separation



Classification	Last-layer activation	Loss function
single-label	softmax	categorical_crossentropy
multi-label (scores for candidates)	sigmoid	binary_crossentropy



Geant 4 simulation



Examples of events with e and π^- showers and μ^- passing through.

by D. Romanov

Calorimeter NN implementation report

Jefferson Lab
Thomas Jefferson National Accelerator Facility

```
+ Timing (ns):
    * Summary:
    +--------+-------+----------+------------+
    |  Clock | Target| Estimated| Uncertainty|
    +--------+-------+----------+------------+
    |ap_clk  |   5.00|     3.883|        0.62|
    +--------+-------+----------+------------+

+ Latency (clock cycles):
    * Summary:
    +-----+-----+-----+-----+----------+
    |  Latency  |  Interval | Pipeline |
    | min | max | min | max |   Type   |
    +-----+-----+-----+-----+----------+
    |   12|   12|    1|    1| function |
    +-----+-----+-----+-----+----------+
```

Latency = 60ns

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	3	-	-	-
Expression	-	-	0	42	-
FIFO	-	-	-	-	-
Instance	-	208	931	4426	-
Memory	3	-	0	0	-
Multiplexer	-	-	-	36	-
Register	-	-	946	-	-
Total	3	211	1877	4504	0
Available SLR	1344	3072	864000	432000	320
Utilization SLR (%)	~0	6	~0	1	0
Available	5376	12288	3456000	1728000	1280
Utilization (%)	~0	1	~0	~0	0

DSP utilization 1%

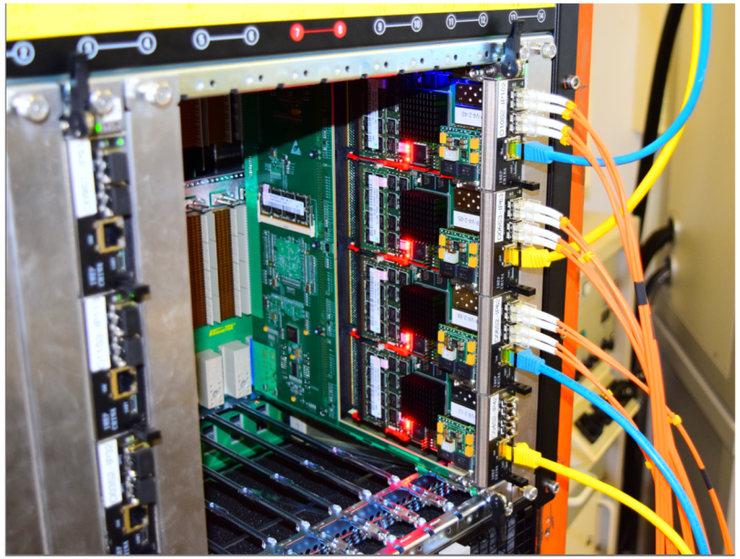
Possible hardware solutions for eic @ IP6

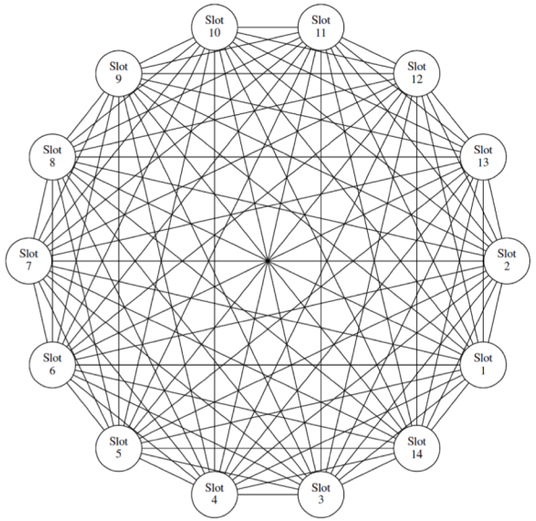
Jefferson Lab
Thomas Jefferson National Accelerator Facility

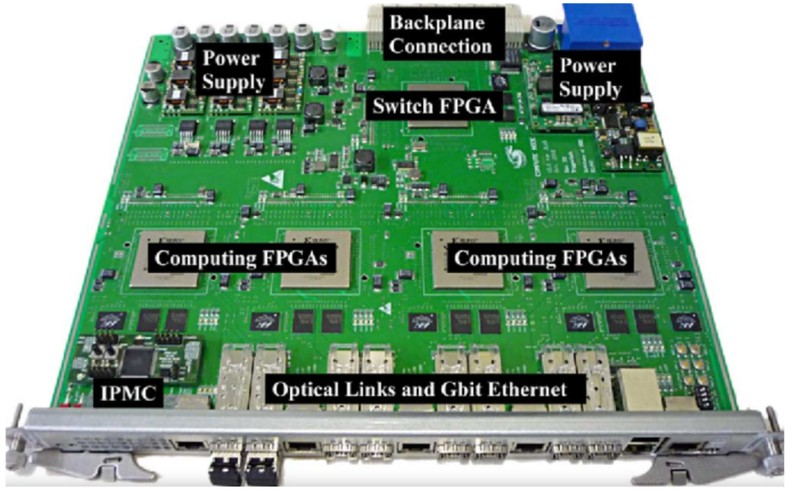
- *There a multiple way to implement ML-FPGA in DAQ.*
- *The main idea: to collect information from all detectors in one unit to process the full event.*
- *For data acquisition in physics, ATCA and OpenVPX standards have become widespread.*

Compute Node (PXD,Belle II)

- The pixel detector of Belle II with its ~ 8 million channels will deliver data at rate of 22 Gbytes/s for a trigger rate of 30 kHz
- A hardware platform capable of processing this amount of data is the ATCA based Compute Node. (Advanced Telecommunications Computing Architecture).
- A single ATCA crate can host up to 14 boards interconnected via a full mesh backplane.
- Each AMC board is equipped with 4 Xilinx Virtex-5 FX70T FPGA.







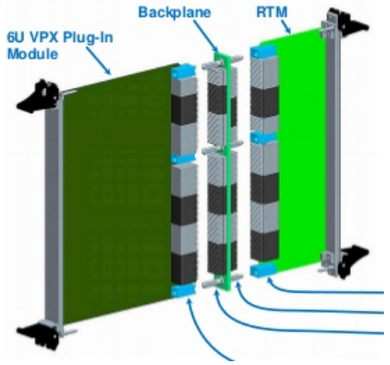
ADC based DAQ for PANDA STT

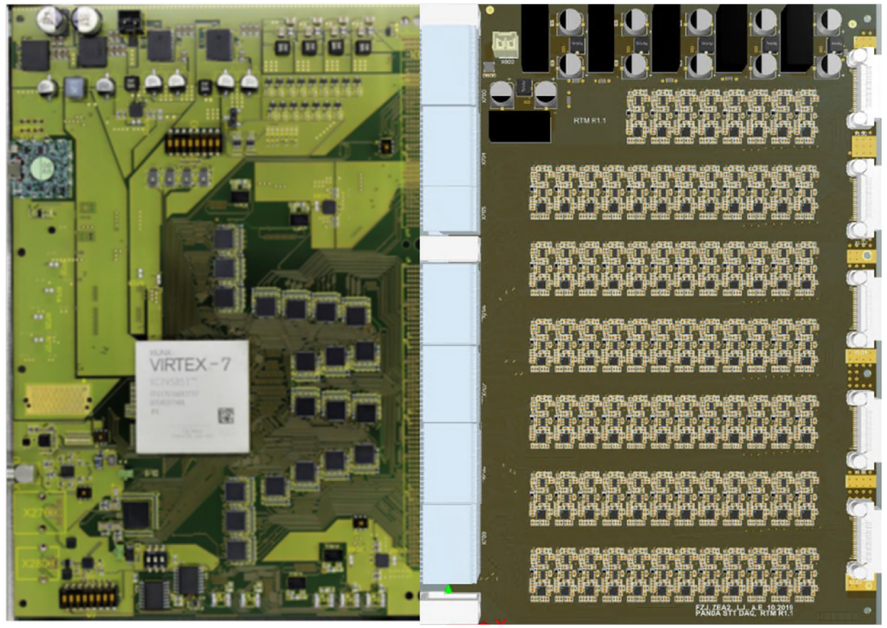
Jefferson Lab
Thomas Jefferson National Accelerator Facility

Level 0 Open VPX Crate

ADC based DAQ for PANDA STT (one of approaches):

- 160 channels (shaping, sampling and processing) per payload slot, 14 payload slots+2 controllers;
- **totally 2200 channels per crate**;
- time sorted output data stream (arrival time, energy,...)
- noise rejection, pile up resolution, base line correction, ..

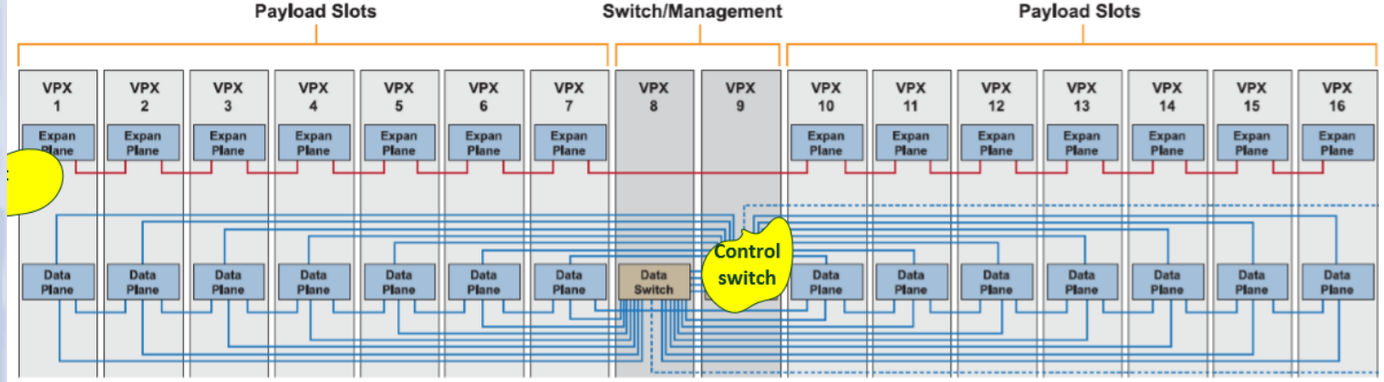




- 40 4-channel ADCs (configurable up to 1 GSPS);
- Single Virtex7 FPGA

- 160 Amplifiers;
- 5 connectors for 32-pins samtec cables

- *All information from the straw tube tracker is processed in one unit.*
- *Allows to build a complete STT event.*
- *This unit can also be used for calorimeters readout and processing.*

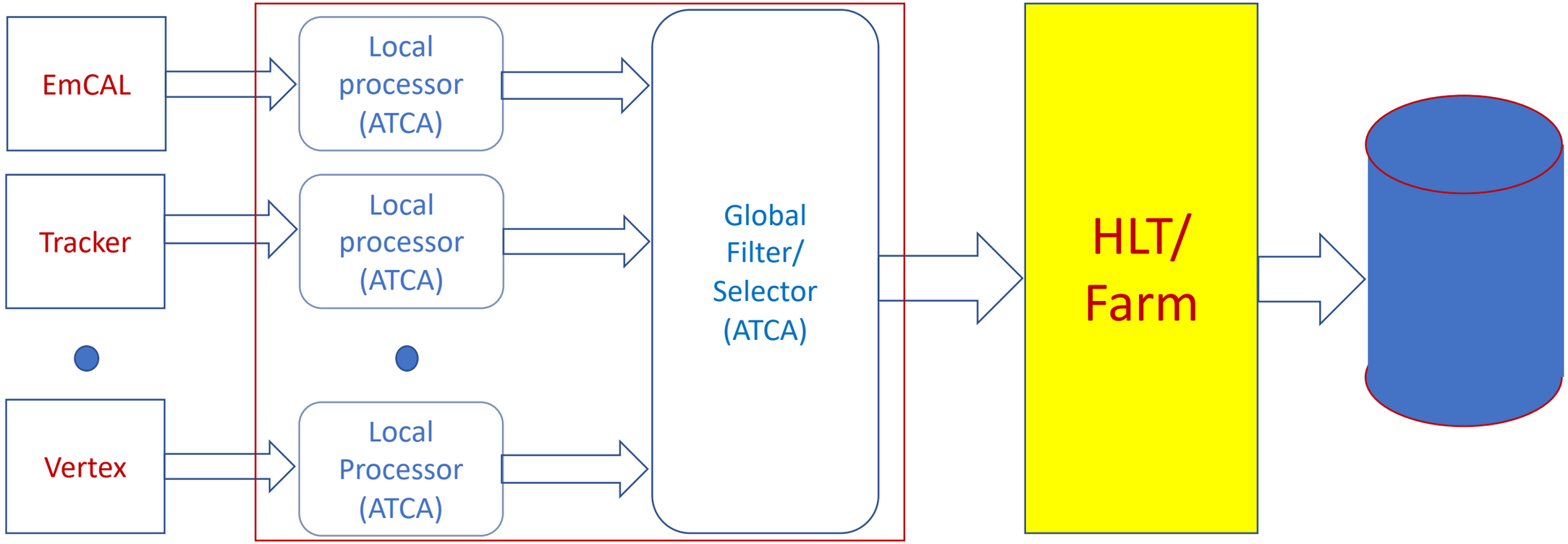


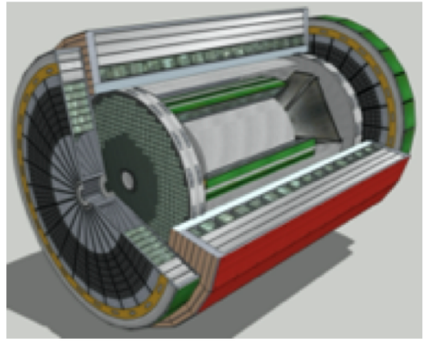
Powerful Backplane up to 670 GBs

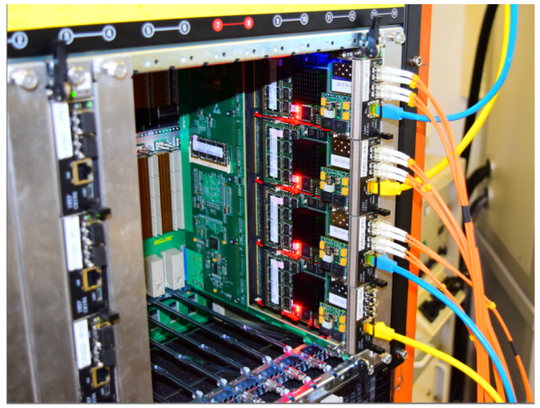
L. Jokhovets, P Kulessa ..

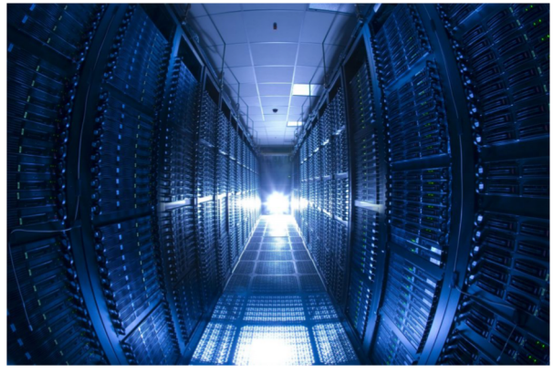
JÜLICH Forschungszentrum

Unified hardware solution (ATCA or OpenVPX)









Outlook

- *An FPGA-based Neural Network application would offer online event preprocessing and allow for data reduction based on physics at the early stage of data processing.*
- *The ML-on-FPGA solution complements the purely computer-based solution and mitigates DAQ performance risks.*
- *FPGA provides extremely low-latency neural-network inference on the order of 100 nanoseconds.*
- *Open-source hls4ml software tool with Xilinx® Vivado® High Level Synthesis (HLS) accelerates machine learning neural network algorithm development.*

- *The ultimate goal is to build a real-time event filter based on physics signatures.*

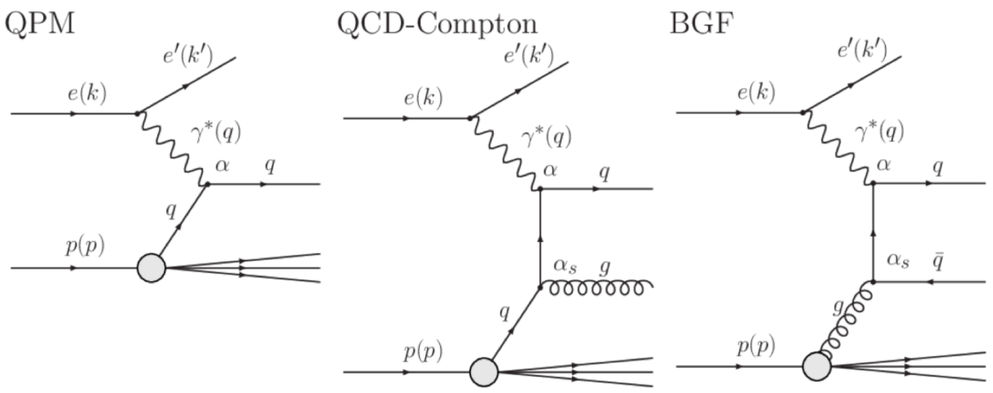


Figure 2.1: Feynman diagrams of the Quark Parton Model, QCD-Compton and Boson Gluon Fusion processes in NC DIS.

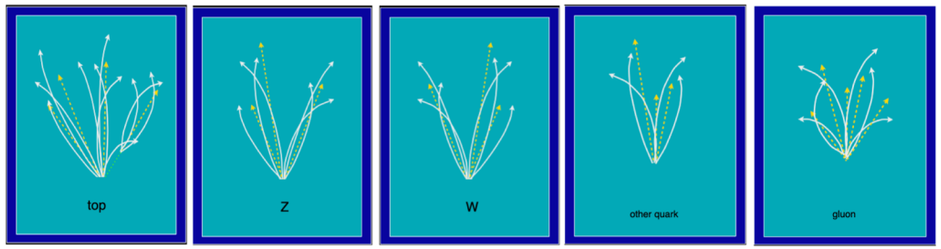
Published in 2007

Measurement of multijet events at low x_{Bj} and low Q^2 with the ZEUS detector at HERA

T. Gosau

Case study: jet tagging

Study a multi-classification task: discrimination between highly energetic (boosted) ***q, g, W, Z, t*** initiated jets



t→bW→bqq	Z→qq	W→qq	q/g background
3-prong jet	2-prong jet	2-prong jet	no substructure and/or mass ~ 0

Signal: reconstructed as one massive jet with substructure

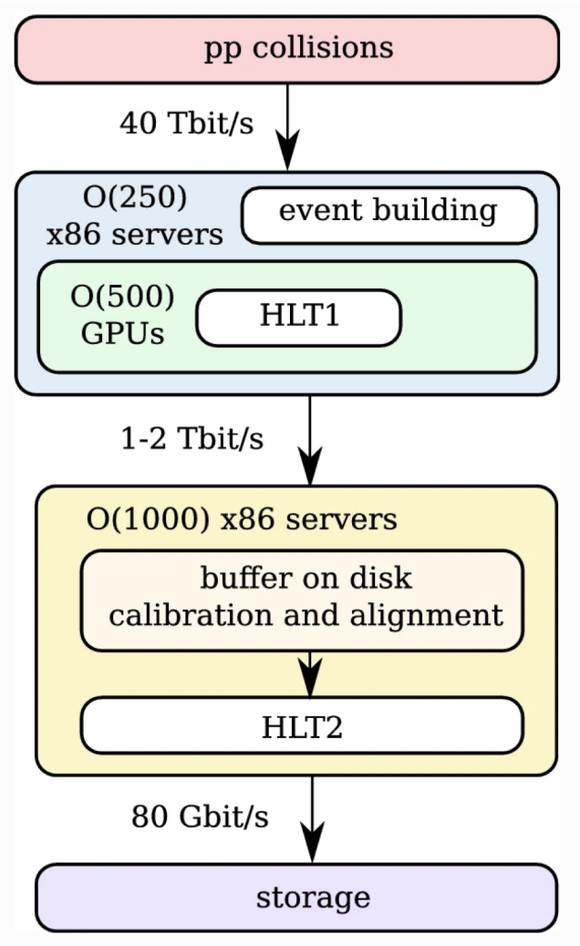
Jet substructure observables used to distinguish signal vs background [*]

[*] D. Guest at al. PhysRevD.94.112002, G. Kasieczka et al. JHEP05(2017)006, J. M. Butterworth et al. PhysRevLett.100.242001, etc..

11.01.2019 Jennifer Ngadiuba - hls4ml: deep neural networks in FPGAs 25

Backup

Example from LHCb



R. Aaij *et al* 2019 *JINST* **14** P04013

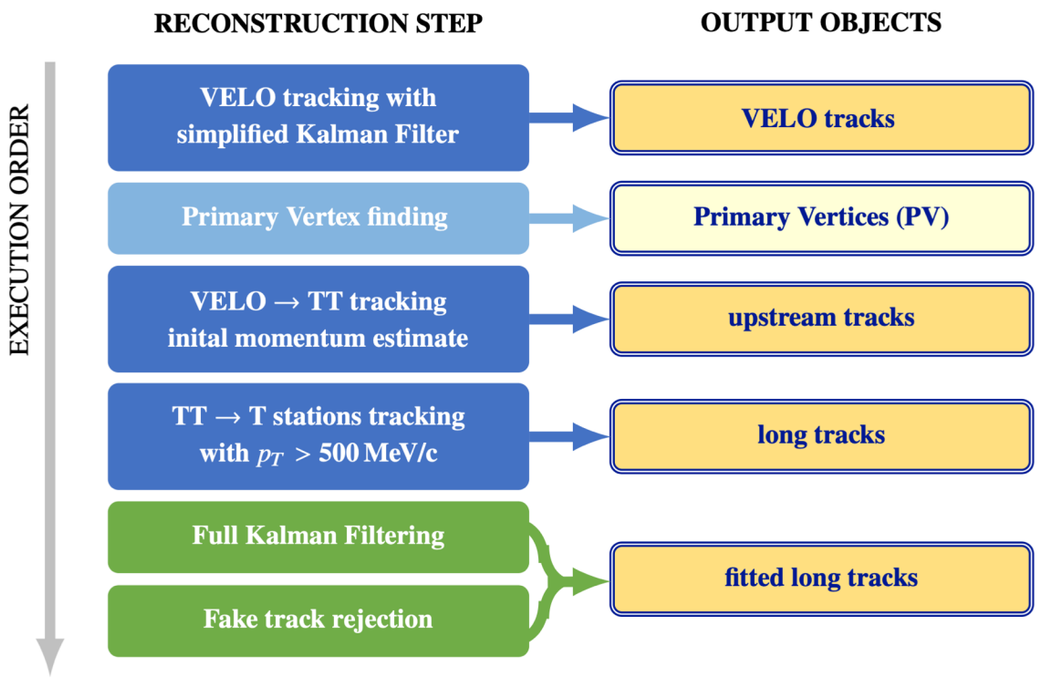
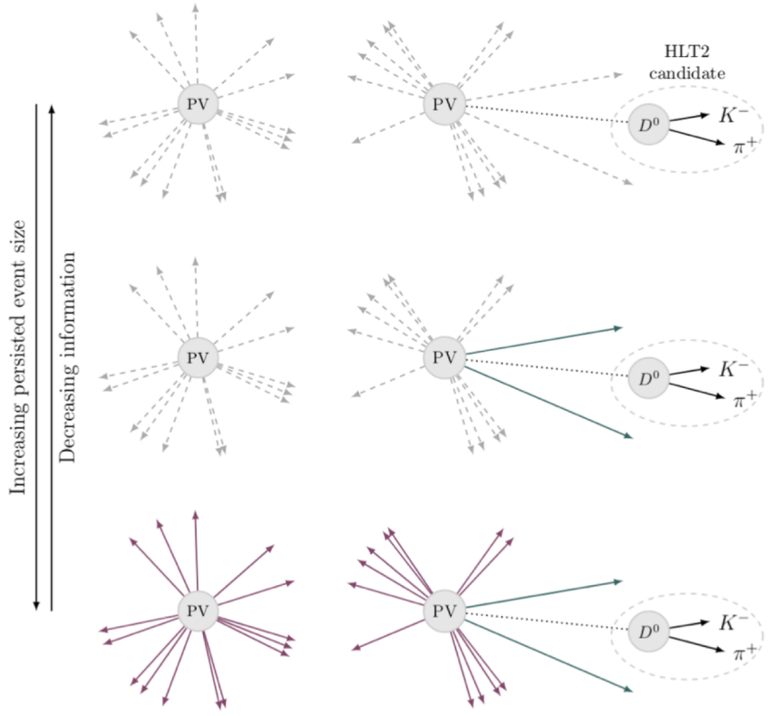


Figure 4. Sketch of the HLT1 track and vertex reconstruction.

LHCb 'Tubo'

Jefferson Lab
Thomas Jefferson National Accelerator Facility

Turbo



- Turbo is the LHCb paradigm for reduced event format data[4]
- High degree of flexibility: Save only as much of the event as is needed for analysis
  - Keep all reconstructed objects, drop the raw event: 70kB
  - Keep only objects used to trigger: 15kB
  - 'Selective Persistence' objects used to trigger + user-defined selection: 15 → 70kB

[4] arXiv:1604.05596, NEW arXiv:1903.01360

C. Fitzpatrick

March 12, 2019

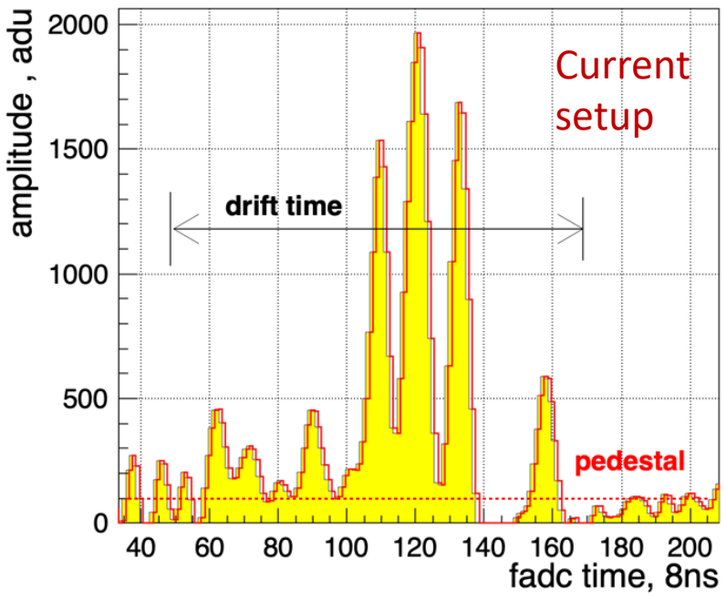
CERN

11 / 20

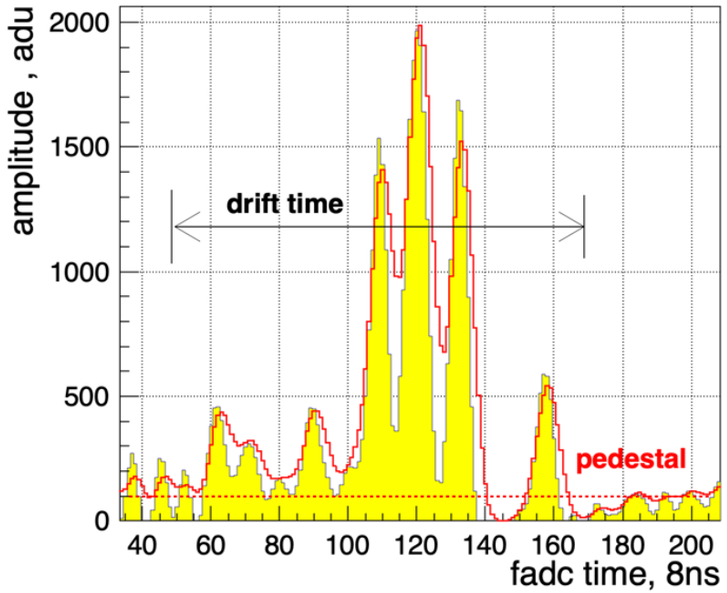
Readout electronics, shaping time optimization

Current setup: preamplifiers (GAS2 ASIC chip) with shaping times of~10-12ns.
The flash ADC has a sampling rate of 125 MHz and 12 bit resolution but provides only pipe-lined triggered readout (price ~50$/channel)

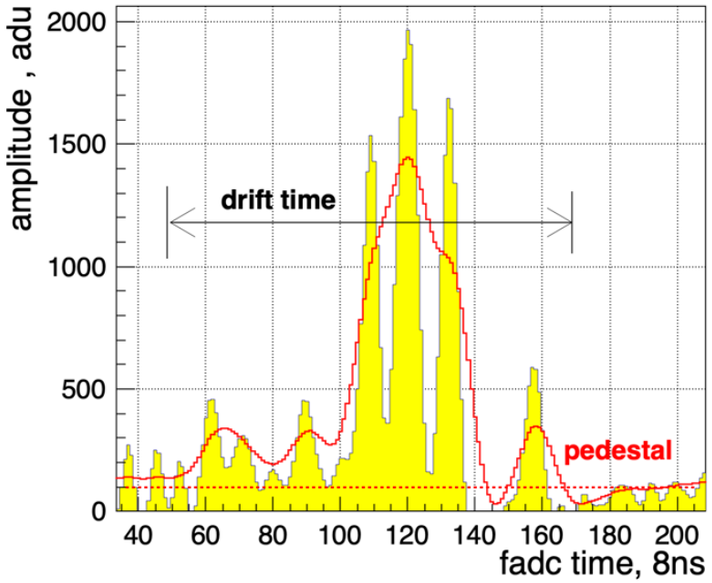
~ 20ns shaping time



~ 40ns shaping time

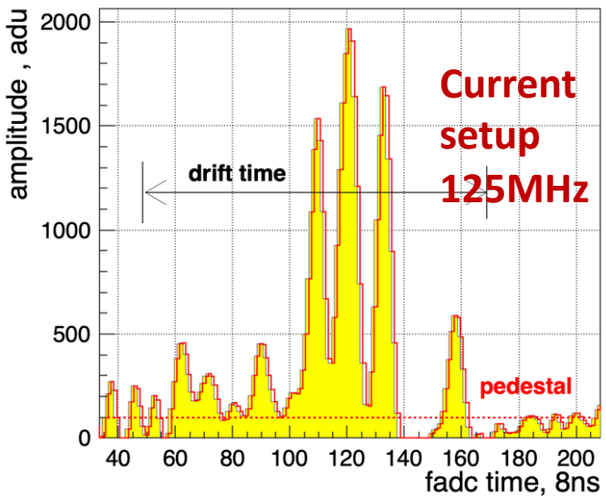


80ns (Sampa) shaping time



Readout electronics, sampling rate optimization

a preamplifier (GAS2 ASIC chip) with shaping times of~10-12ns.
The flash ADC has a sampling rate of 125 MHz and 12 bit resolution but provides only pipe-lined triggered readout (price ~50$/channel)



Sampling frequency: