Jefferson Lab
Thomas Jefferson National Accelerator Facility

Machine Learning on FPGA

Sergey Furletov

Jefferson Lab

Streaming Readout VI meeting

14 May 2020

Outline

- Motivation
- ML FPGA(*) project
- Tests of ML with Xilinx FPGA
- Conclusion

(*) Field Programmable Gate Array

Introduction

Jefferson Lab
Thomas Jefferson National Accelerator Facility

- *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.*
- *Modern (triggered) data acquisition systems (LHC, KEK, Fair) employ several stages for data reduction.*
  - The CMS experiment at LHC has a Level 1 trigger that makes a decision in ~4 μs and rejects 99.75% of events.
  - Their High Level Trigger (software), decision ~100 ms , rejects 99.9% of the data from Level 1.
- *Concepts of trigger-less readout and data streaming will produce large data volumes being read from the detectors. Most of this will be uninteresting and ultimately discarded.*
- *From a resource standpoint, it makes much more sense to perform data pre-processing and reduction at early stages of data streaming.*
- *Our project mostly inspired by work carried out at CERN , and progress in ML application on FPGA*
- *At the LHC, data rates at the CMS and ATLAS , are of the order of hundreds of terabytes per second.*

Example from CMS

Jefferson Lab
Thomas Jefferson National Accelerator Facility

- The task of the real-time processing is to filter events to reduce data rates to manageable levels for offline processing called triggering.

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

- Level-1 typically uses custom hardware with ASICs or FPGAs.
- The second stage of triggering, High Level Trigger (HLT), uses commercial CPUs to process the filtered data in software.

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!

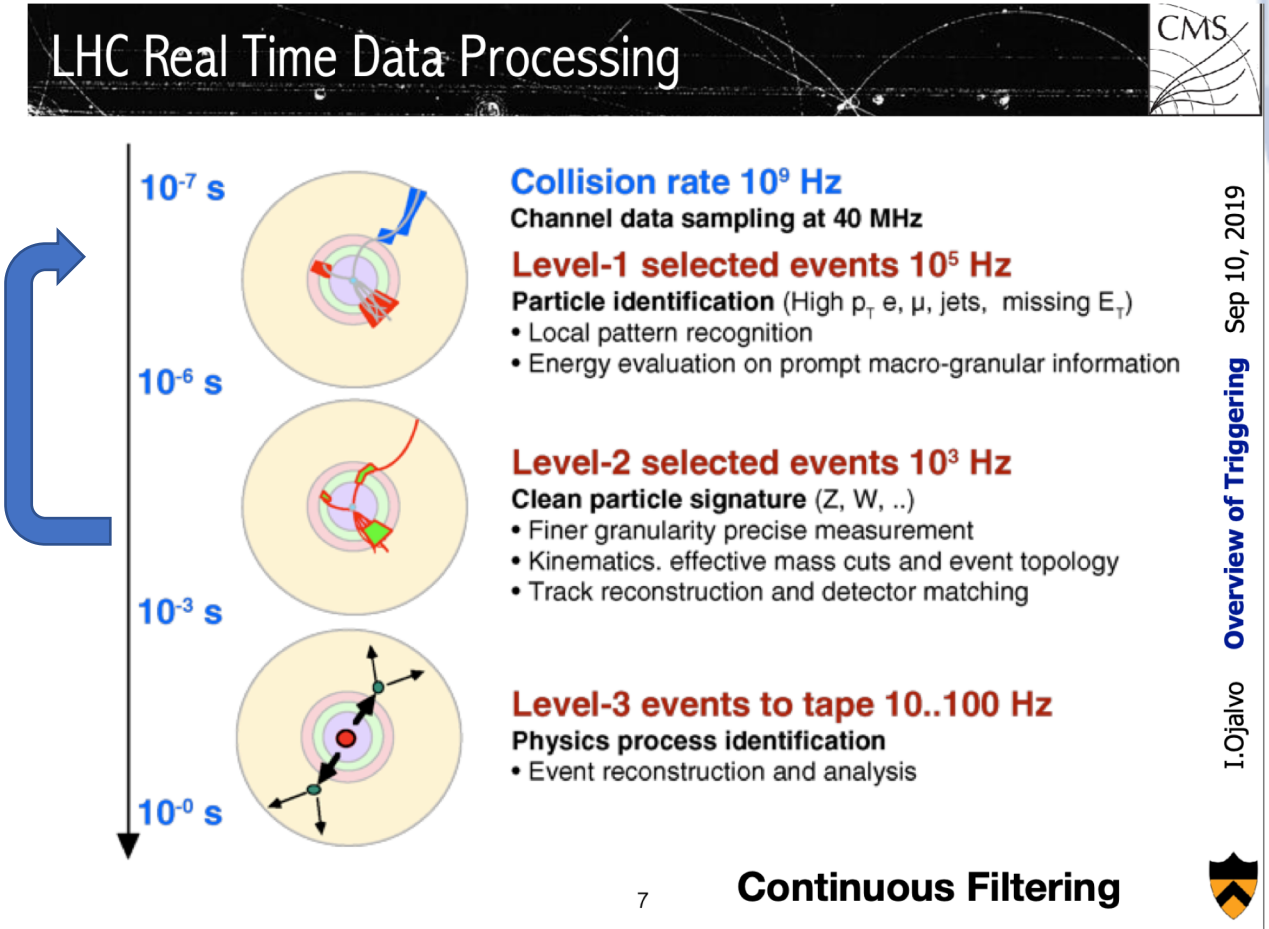
Rejection is mostly defined by cross section of interesting physics processes.

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 can be performed at Level 1



Fast Machine Learning,10-13 September 2019, Fermilab

ML on Xilinx 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.*
- *Machine learning methods are widely used and have proven to be very powerful in particle physics.*
- *However, exploration of the use of such techniques in low-latency FPGA hardware has only just begun.*

Figure 2: Compared to alternative devices such as GPUs and ASICs, FPGAs are the only viable choice for the event trigger processing because they provide extremely low latency. While the large numerical processing capability of GPUs is attractive, these technologies are optimized for high throughput, not low latency.

https://www.xilinx.com/publications/powered-by-xilinx/cerncasestudy-final.pdf

EIC rates estimation

Jefferson Lab
Thomas Jefferson National Accelerator Facility

Jin Huang <jhuang@bnl.gov> YR kick-off meeting

BROOKHAVEN NATIONAL LABORATORY

Despite the fact that it is possible to record everything, it's better to be able to filter or prescale events with a large cross section

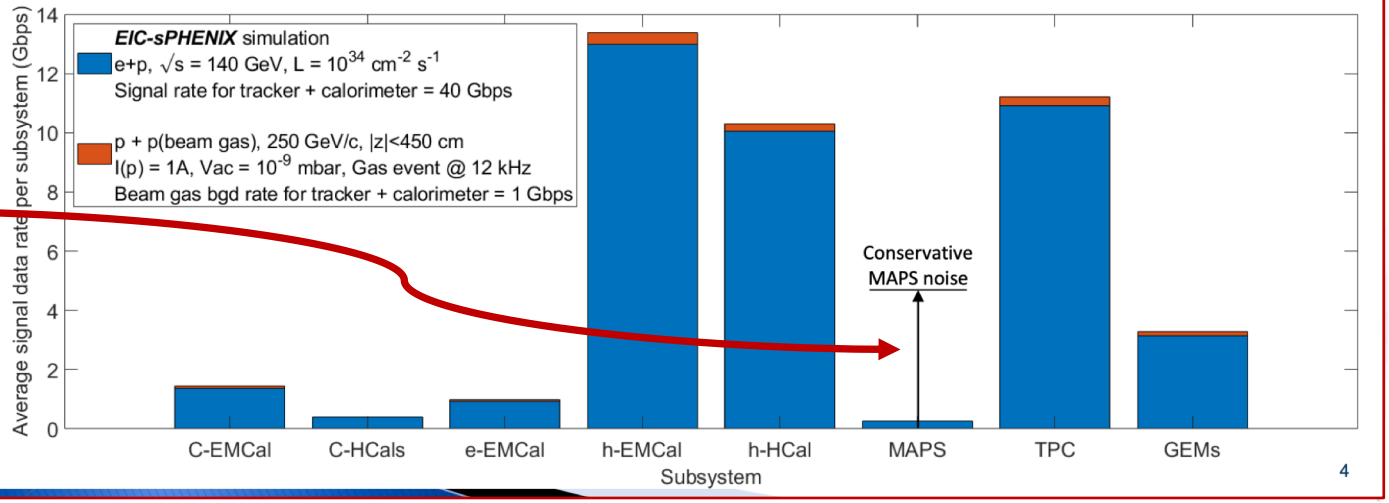
Need a large computer farm to handle streaming data. In terms of resources, it makes sense to perform data pre-processing and reduction at the early stages of data streaming.

Belle II , 8 megapixels PXD produces ~200 Gbps @ 30 kHz trigger rate. (beam background, noise and synchrotron)

Signal data rate -> DAQ strategy

Note sPH-cQCD-2018-001: https://indico.bnl.gov/event/5283/ , Simulation: https://eic-detector.github.io/

- What we want to record: total collision signal ~ 100 Gbps @ 10^{34} cm^{-2} s^{-1}
  - Assumption: sPHENIX data format, 100% noise
  - Less than sPHENIX peak disk rate. 10^{-4} comparing to LHC collision
- Therefore, we could choose to stream out all EIC collisions data
- In addition, DAQ may need to filter out excessive beam background and electronics noise, if they become dominant.
  - Very different from LHC, where it is necessary to filter out uninteresting p+p collisions (CMS/ATLAS/LHCb) or highly compress collision data (ALICE)
  - Such filtering does not require real-time event reconstruction



4

ML-FPGA project

F. Barbosa, C. Dickover, Y. Furletova, D. Romanov. L. Belfore (ODU)

Physics signatures example

There are many physics signatures which could benefit from jet substructure and electron identification in real-time.
In our project, we focus on the electron/hadron identification and on the classification of jets as either a quark (*q*)) (light and heavy) or gluon (g).

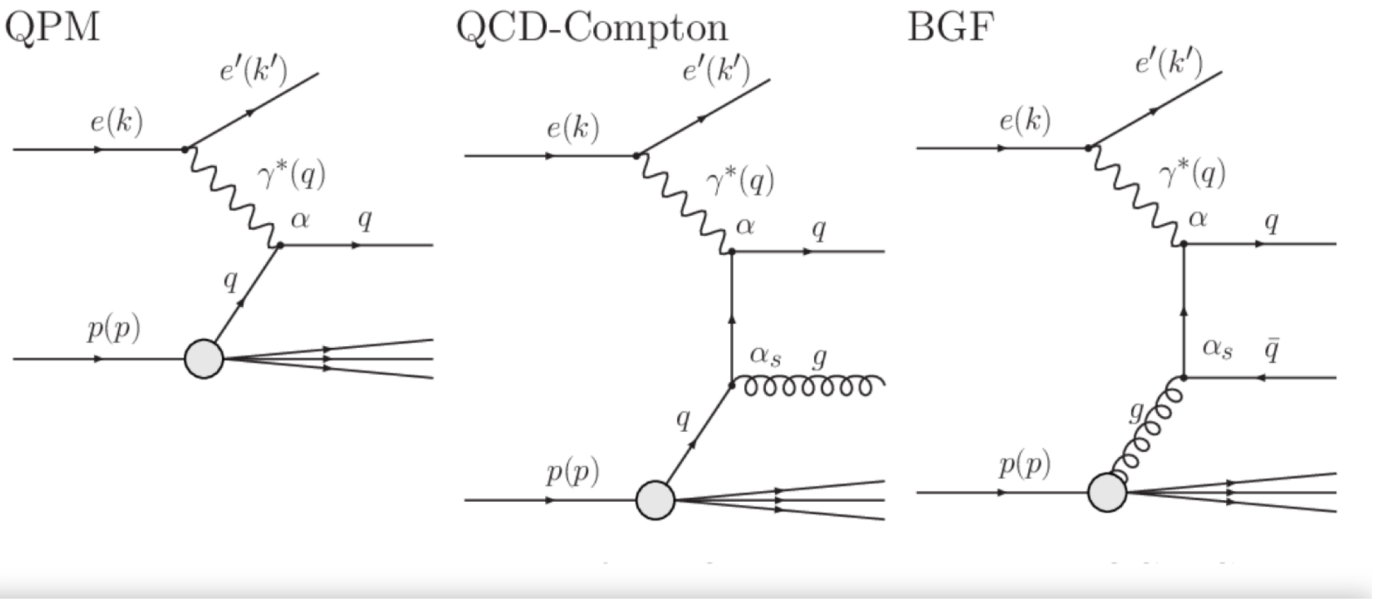


Figure 2.1: Feynman diagrams of the Quark Parton Model, QCD-Compton and Boson Gluon Fusion processes in NC DIS.

Published in 2007

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

T. Gosau

ML FPGA test setup

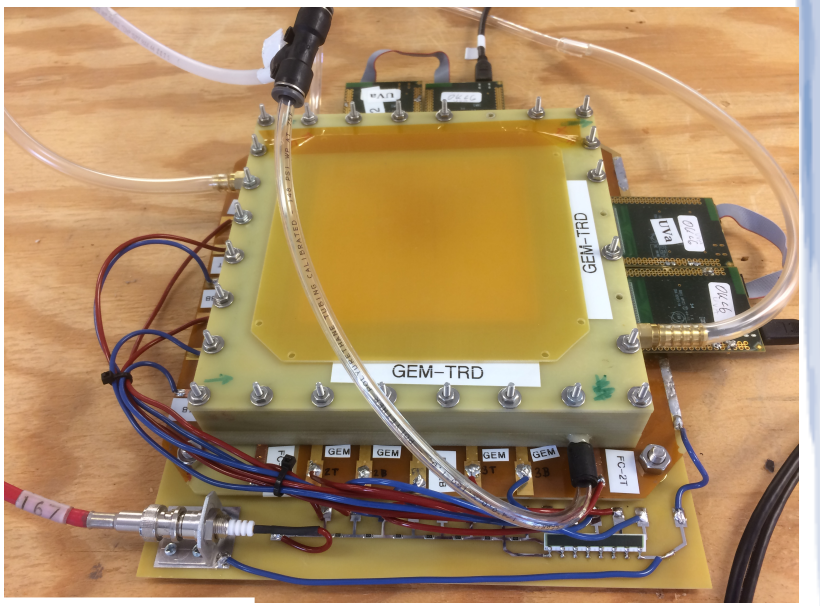
Jefferson Lab
Thomas Jefferson National Accelerator Facility

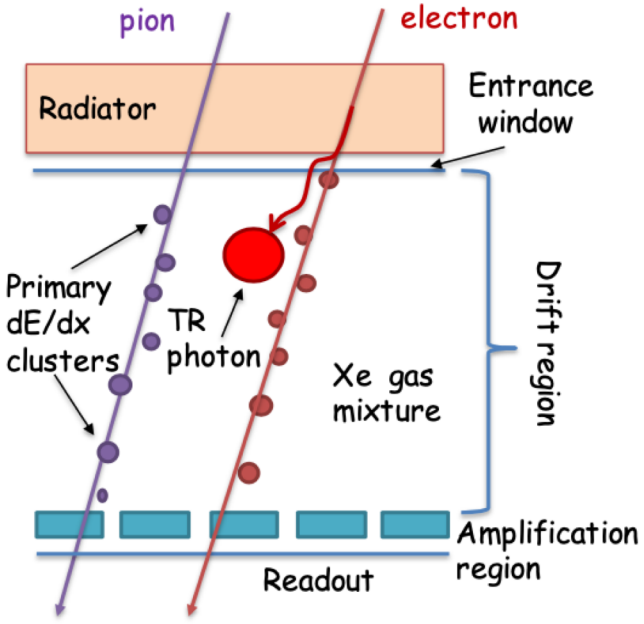
- *To demonstrate the operating principle of the ML FPGA, we propose to use the existing setup of the ongoing EIC detector R&D project (eRD22) "GEM based Transition radiation detector (TRD) and tracker".*
- *A small 10x10 cm GEM-TRD prototype and fADC125 can generate up to 128 GB/s of raw data traffic.*
- *This detector, in addition to a track coordinate (μTPC mode), has capabilities of electron identification or electron/hadron separation, which is highly important for EIC physics.*
- *For the GEM-TRD project we already use offline Machine Learning tools (JETNET, ROOT-based TMVA), and the results can be used for validation of the proposed implementation of FPGA-based neural networks .*
- *A FPGA-based Neural Network application would offer online particle identification and allow for data reduction based on physics at the early stage of data processing.*
- *Another important part of the project is evaluation of advantages of "global PID" compared to the standalone PID from each detector. To test the global PID performance we plan to integrate the EIC calorimeter prototype (3x3 modules) into the ML-FPGA setup.*
- *Preprocessed data from both detectors including decision on the particle type will be transferred to another ML-FPGA board with neural network for global PID decision.*
- *Real beam testing is planned in Hall D, where there is already a test beam site that can be used for testing the prototype GEM-TRD, ECAL and Modular RICH detectors.*

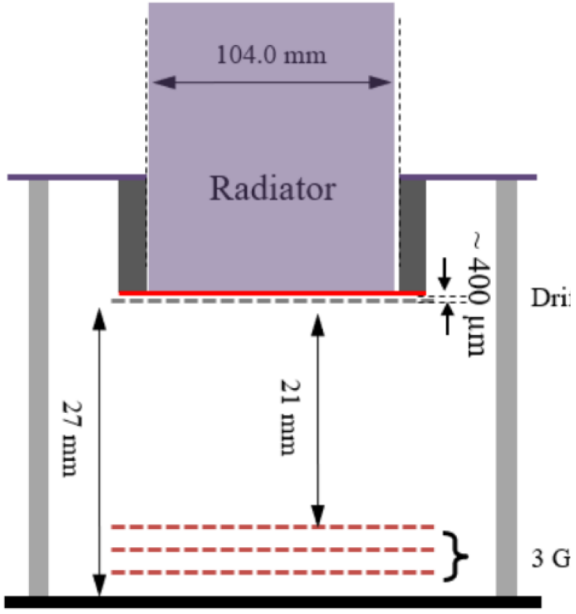
GEM-TRD prototype

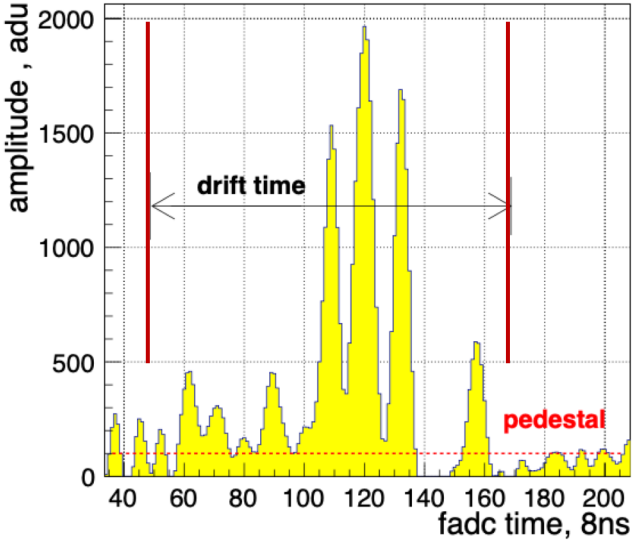
Jefferson Lab
Thomas Jefferson National Accelerator Facility

- 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).
- Still need to modify a FADC125 board with serial streaming interface (in progress).
- GEM-TRD provides e/hadron separation and tracking



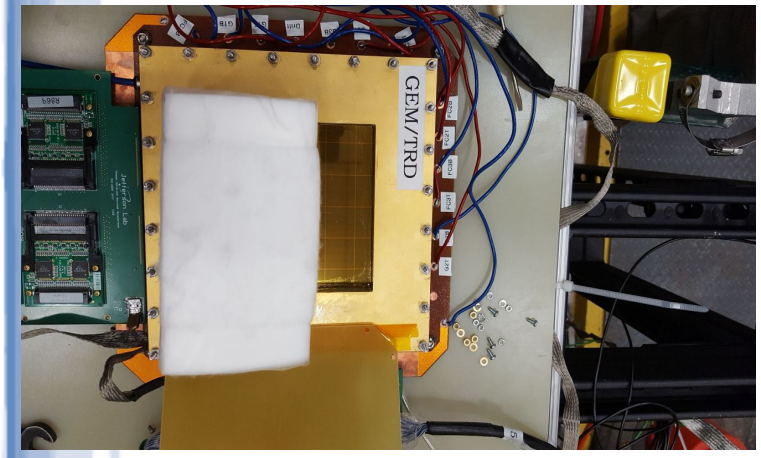


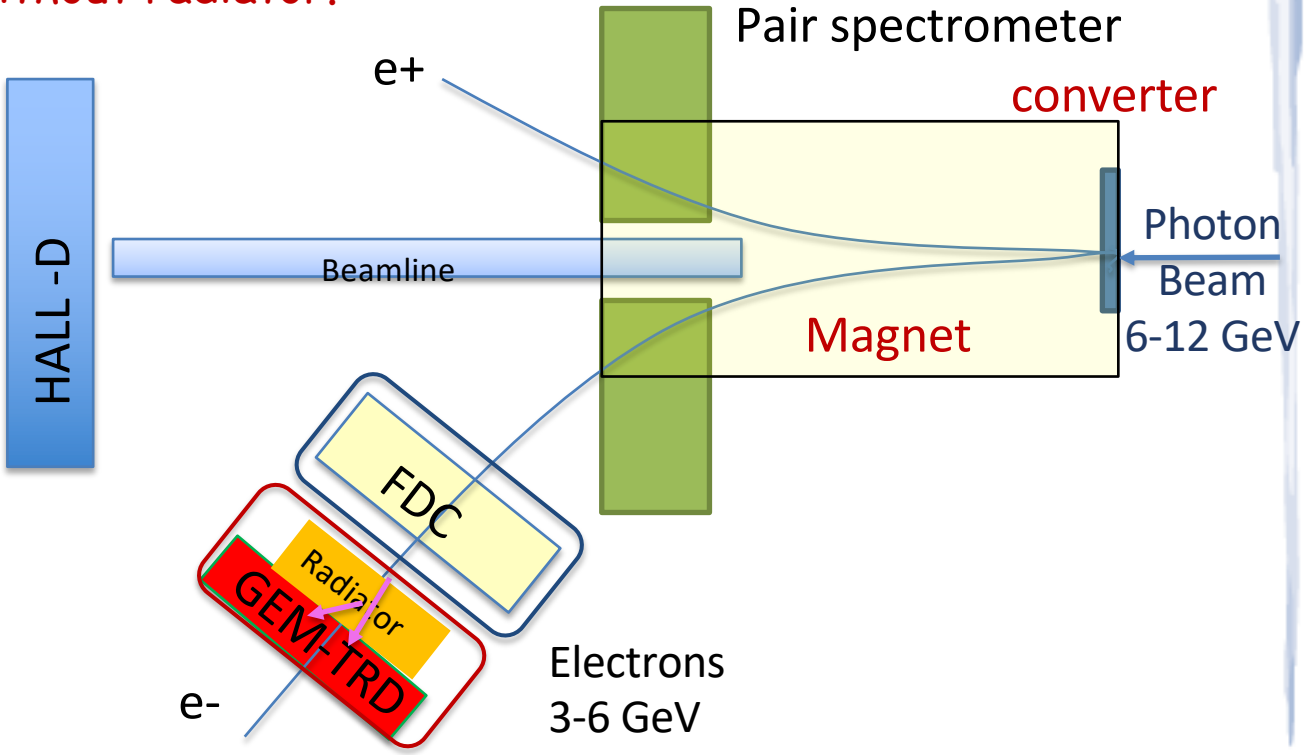




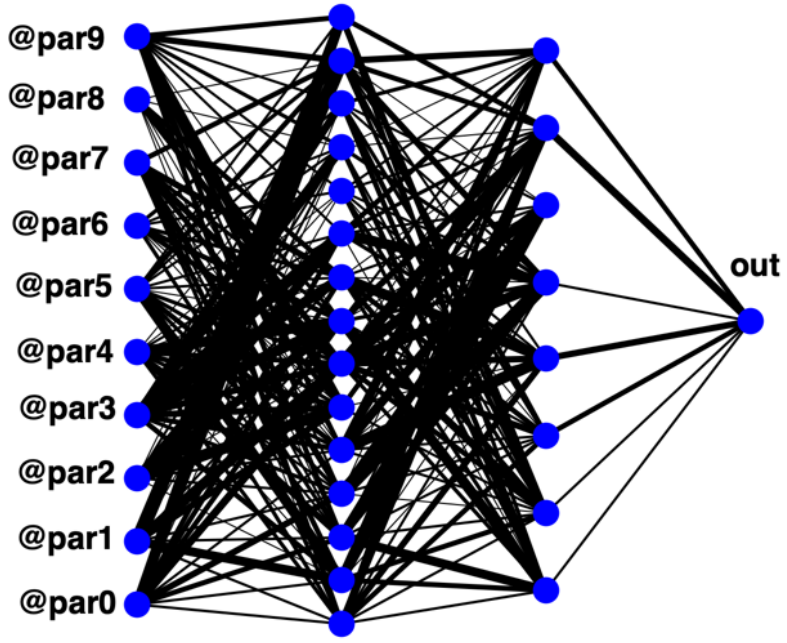
Beam setup at JLab Hall-D

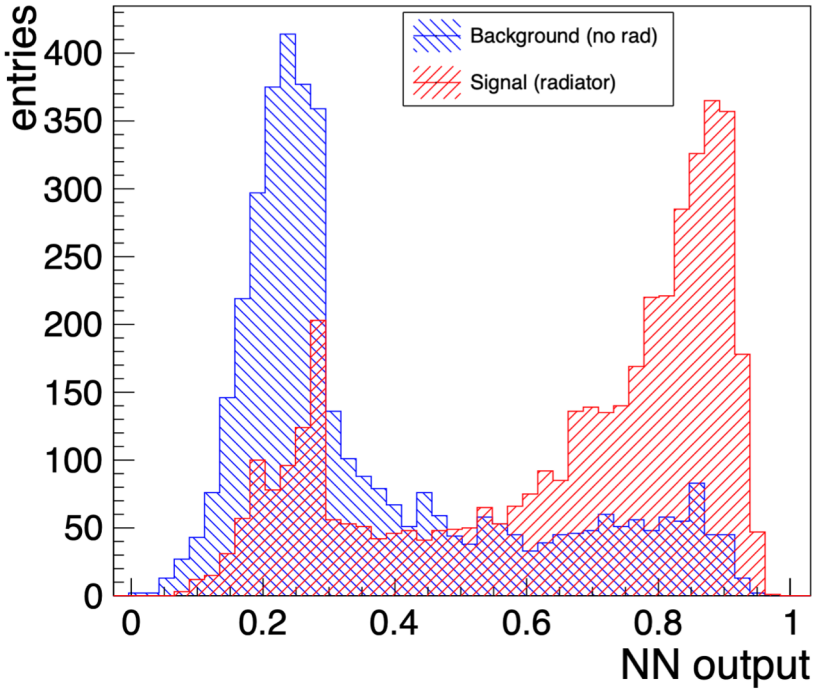
- *Tests were carried out using electrons with an energy of 3-6 GeV, produced in the converter of a pair spectrometer.*
- *The electron energy is known from the pair spectrometer.*
  - *The radiator is mounted in front of the GEM-TRD and covers about half of the sensitive area.*
  - *We do not have hadron beam in this setup:*
    - ✓ The effect of TR is evaluated by comparison of data from electrons with radiator and electrons without radiator.





Machine learning in the data analysis



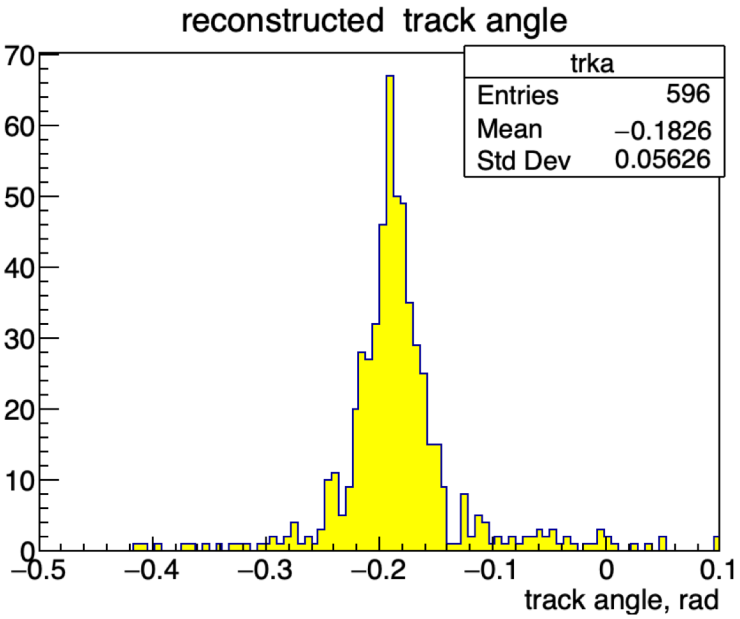


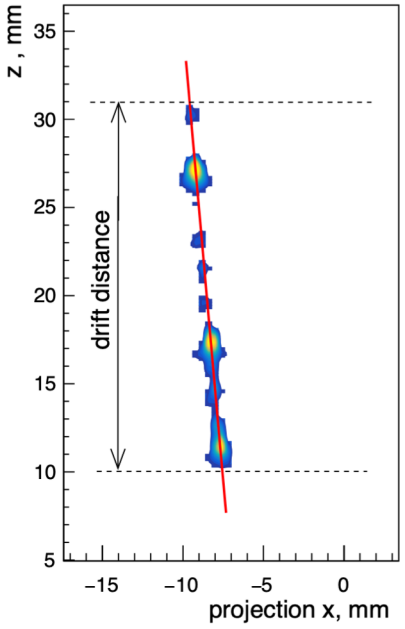
- *For data analysis we used a fast artificial neural network classifier from root: MultiLayerPerceptron (MLP)*
- *All data was divided into 2 samples: training and test samples*
- *Top right plot shows neural network output for single module:*
  - Red - electrons with radiator
  - Blue – electrons without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

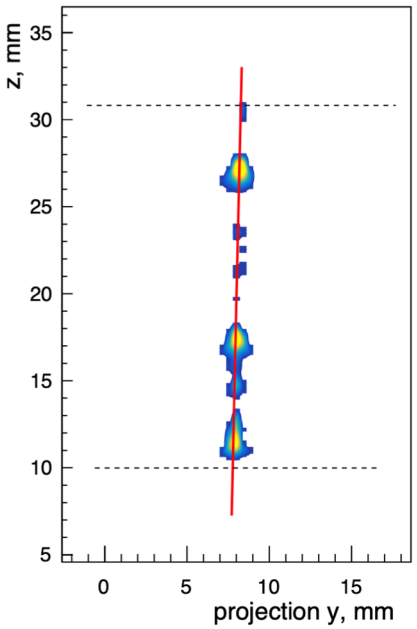
Tracking with GEM-TRD

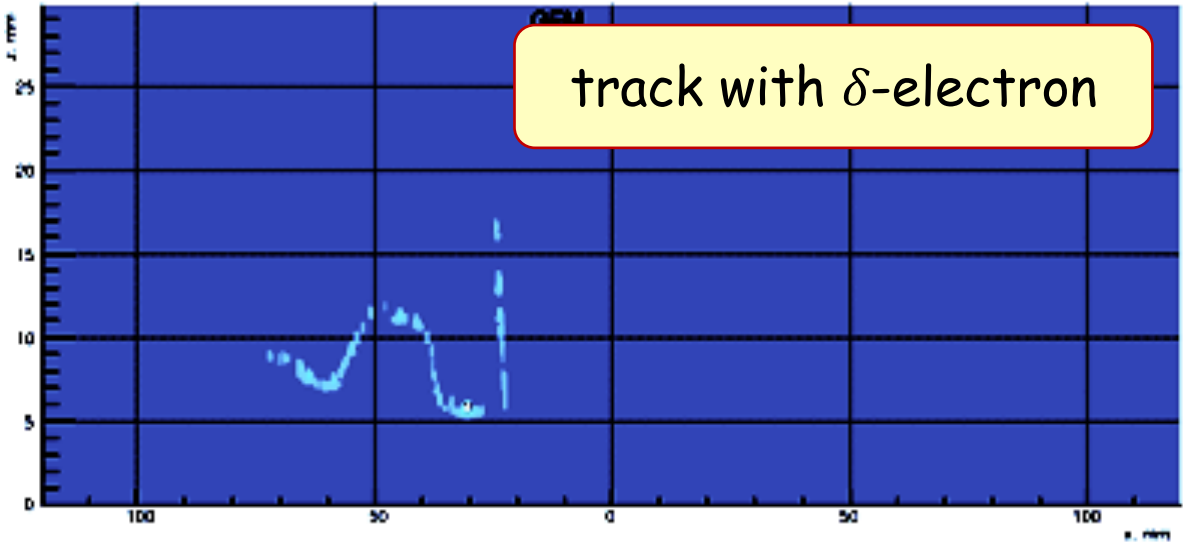
Jefferson Lab
Thomas Jefferson National Accelerator Facility

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







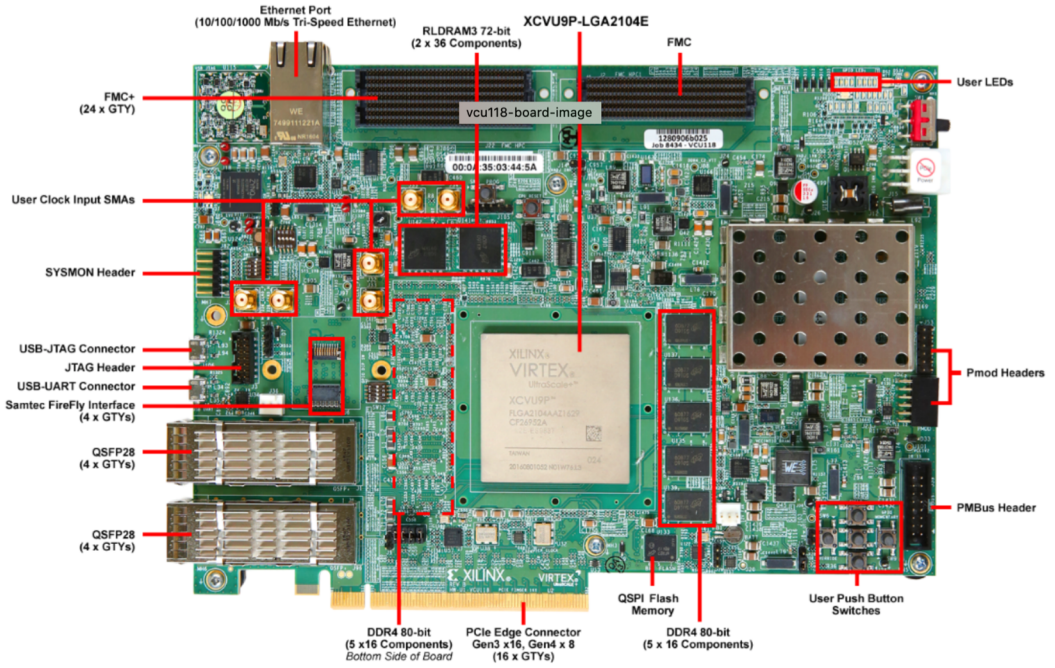


FPGA board for ML

- *At an early stage in this project, as hardware to test ML algorithms on FPGA , we use a standard Xilinx evaluation boards rather than developing a customized FPGA board. These boards have functions and interfaces sufficient for proof of principle of ML-FPGA.*
- *The proposed Xilinx evaluation board includes the Xilinx XCVU9P and 6,840 DSP slices. Each includes a hardwired optimized multiply unit and collectively offers a peak theoretical performance in excess of 1 Tera multiplications per second.*
- *Second, the internal organization can be optimized to the specific computational problem. The internal data processing architecture can support deep computational pipelines offering high throughputs.*
- *Third, the FPGA supports high speed I/O interfaces including Ethernet and 180 high speed transceivers that can operate in excess of 30 Gbps.*

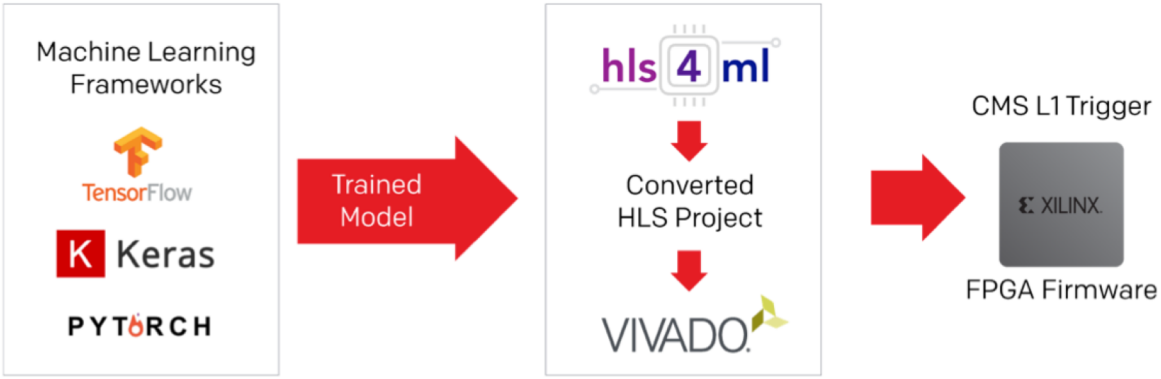
Xilinx Virtex® UltraScale+™

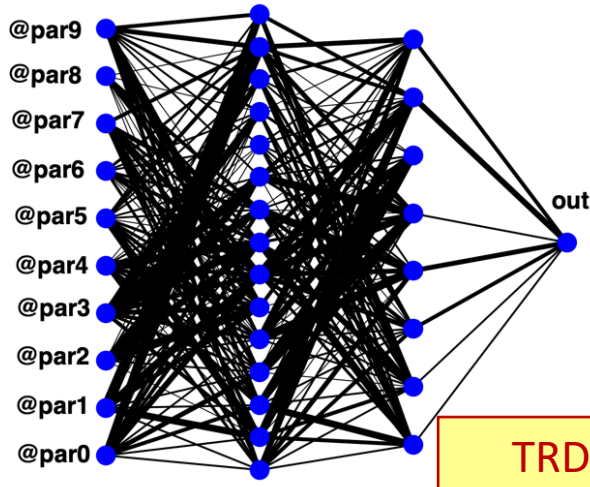
Featuring the Virtex® UltraScale+™ XCVU9P-L2FLGA2104E FPGA



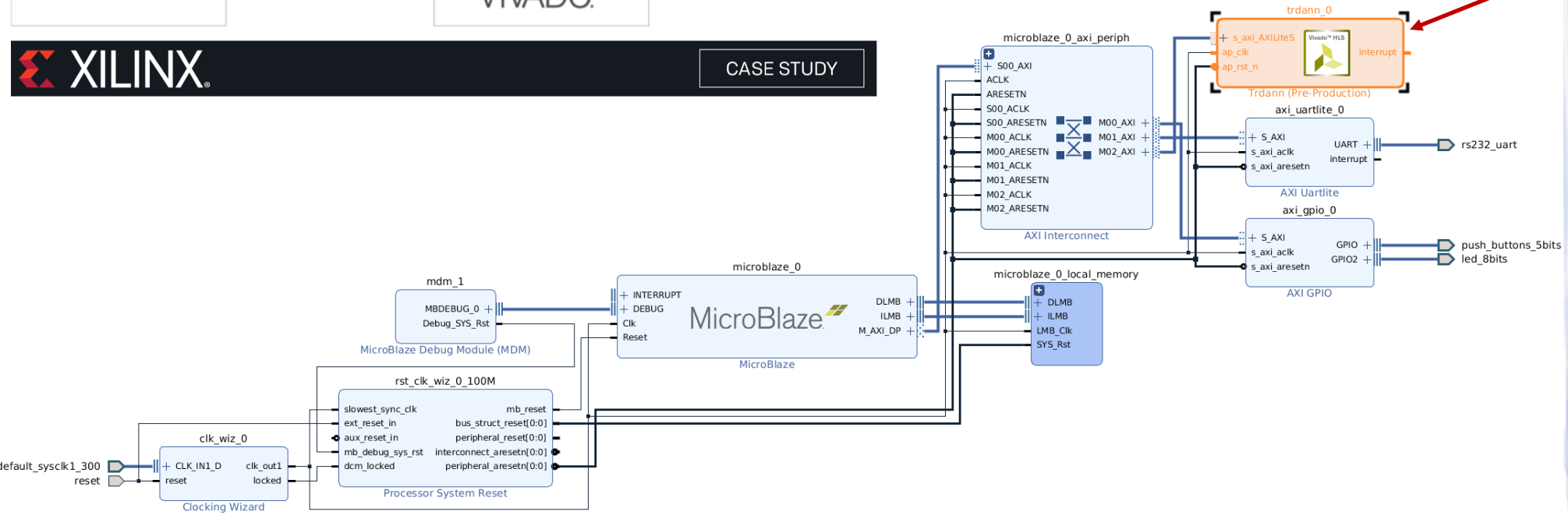
Creating ML FPGA Core

- **Vivado** High-Level Synthesis (**HLS)** transforms a C, C++, or SystemC design specification into Register Transfer Level (RTL) code for synthesis and implementation by the **Vivado** tools.
- *Using HLS significantly decreases development time. (at the cost of lower efficiency of use of FPGA resources).*





TRD ML Core



Xilinx HLS: C++ to Verilog

Jefferson Lab
Thomas Jefferson National Accelerator Facility

```
//------------------------------
// 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)/(
  input1 = (fx_t(input[1]) - (fx_t)4.75766)/(fx_t)
  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++

Verilog

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;
```

Xilinx vivado implementation

Jefferson Lab
Thomas Jefferson National Accelerator Facility

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	4.00	3.466	0.50

Latency (clock cycles)

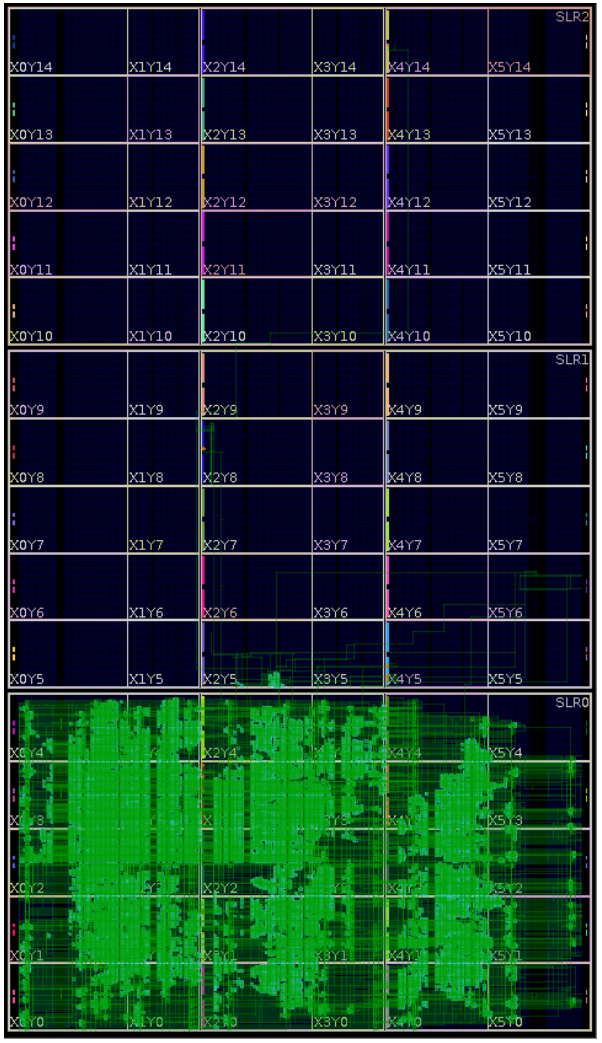
Summary

Latency		Interval		
min	max	min	max	Type
15	381	15	381	none

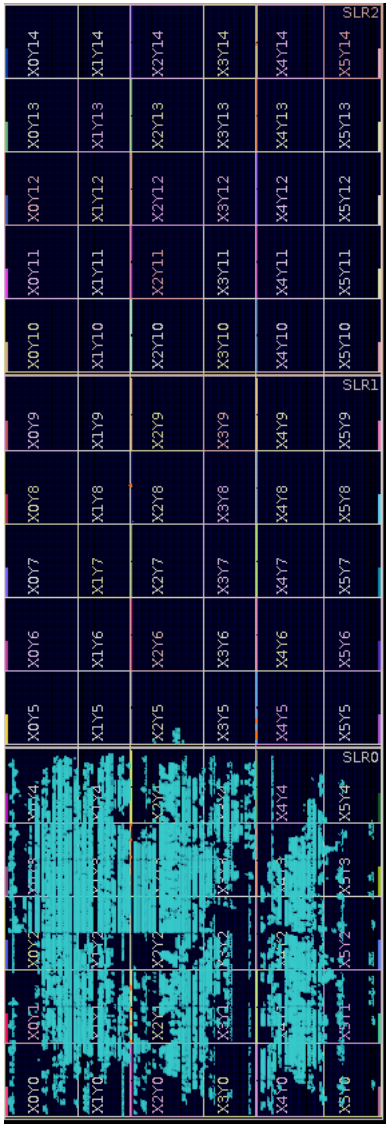
Utilization Estimates

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



Test ML FPGA



C++ code for test :

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

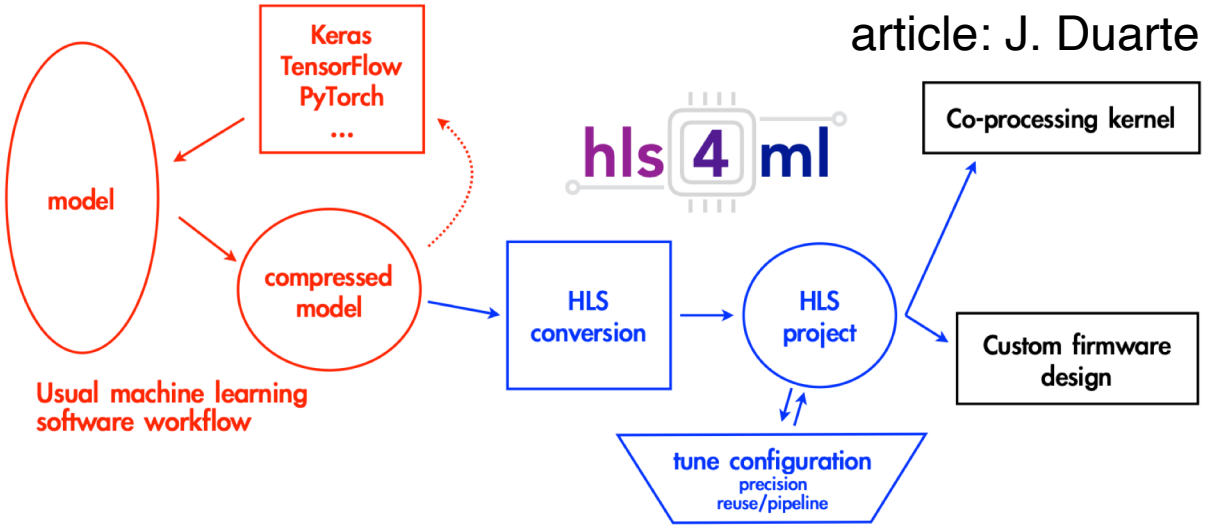
```
XTrdann ann;
int ret = XTrdann_Initialize(&ann, 0);

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);
}
```

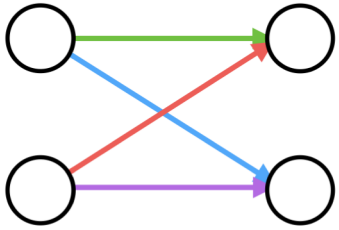
Optimization with hls4ml package

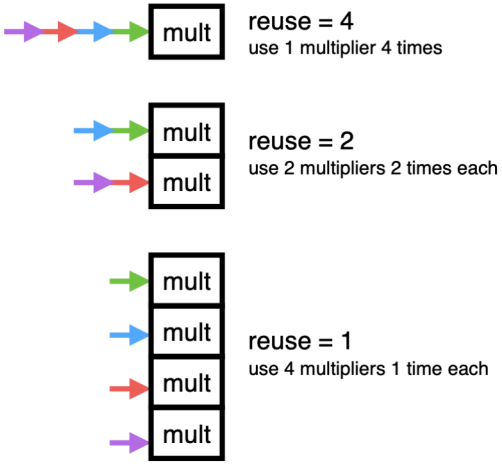
Jefferson Lab
Thomas Jefferson National Accelerator Facility

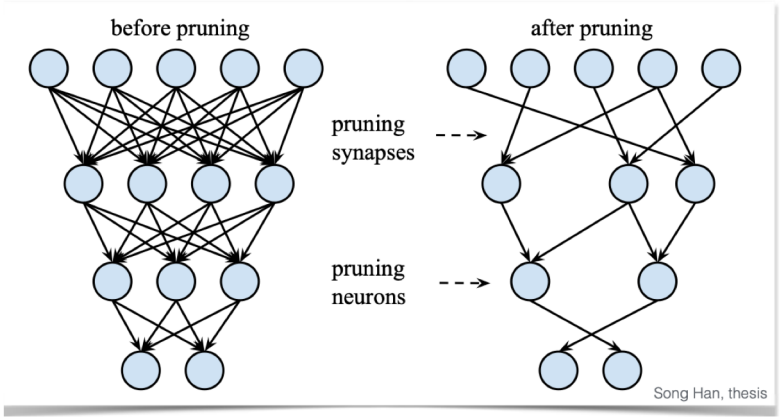
- A package hls4ml is developed based on High-Level Synthesis (HLS) to build machine learning models in FPGAs.

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



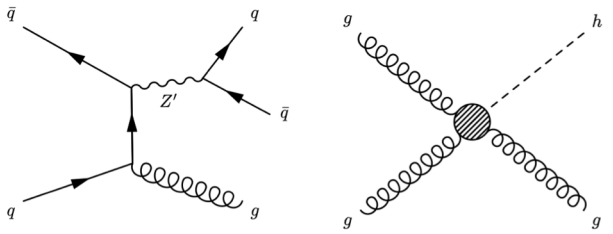






Conclusion

- ML-FPGA project just recently started.
- The initial goal is make a test setup
  - The setup will be used to identify and optimize artificial neural network algorithms and topologies, suitable for real time FPGA applications.
  - It will also be used to perform beam tests in Hall-D with GEM-TRD and calorimeter prototypes as PID detectors to estimate performance of ML on FPGA in a real time environment.
  - Test results could be used to calculate resource scaling for planned large scale experiments (EIC, SOLID, etc).
  - Results on performance and price could also serve as a feasibility study on building a full scale ML-FPGA filter for current experiments such as CLAS12 and/or GlueX.
- The ultimate goal is to build real-time event filter based on physics signatures.

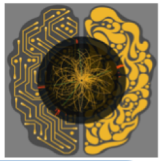


Backup

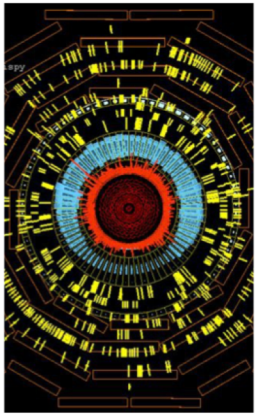
Event processing

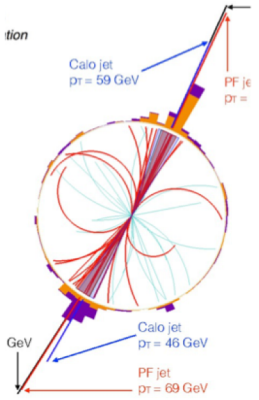
Jefferson Lab
Thomas Jefferson National Accelerator Facility

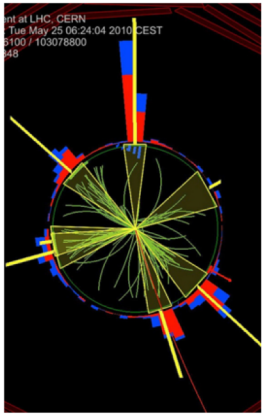
From RAW to High Level data

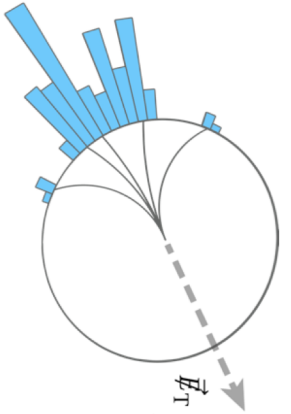


Detector Data	Local reconstruction	Particle representation	Jet Clustering	High level features









Event Processing

Dimensionality reduction

Globalization of information

The reconstruction of an event goes from the digital signal of the individual sub-detector to a sequence of particles, jets, and high-level features

A **Kalman Filter Muon Track-Finder** has been written in **HLS firmware** for the barrel region **already for Run II**

Algorithm does **track propagation** and **parameter updating**

A **large amount of matrix math**

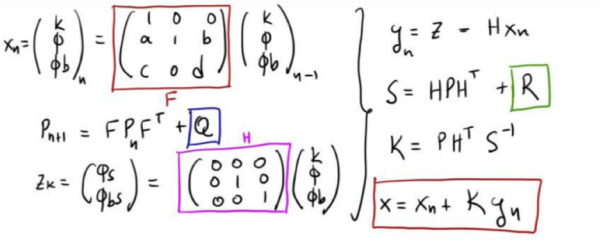
Solution: use DSP cores to reduce FPGA resource utilization

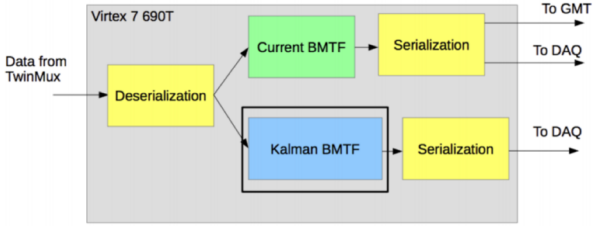
- Programmable using HLS

Latency ~200 ns

Data and emulator agreement is 99.7%

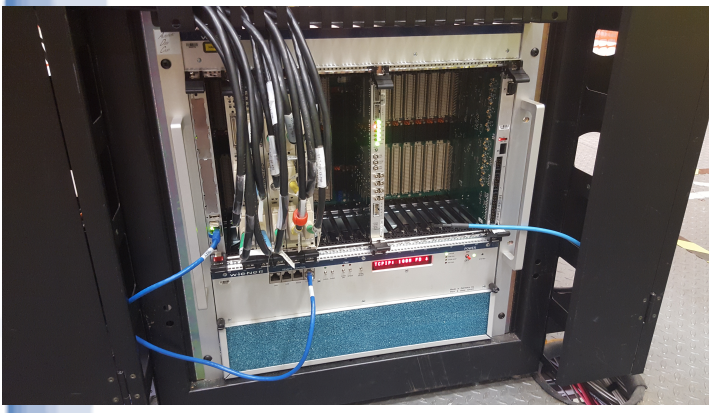
Parallel implementation in current Phase-1 BMTF firmware

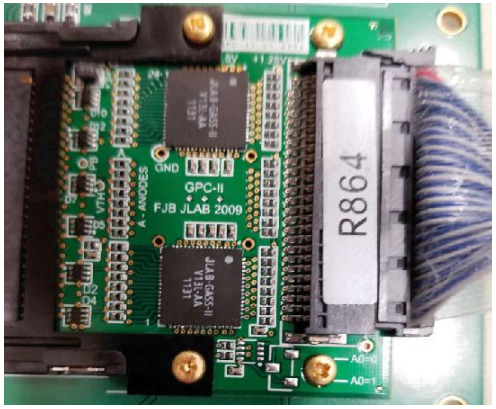


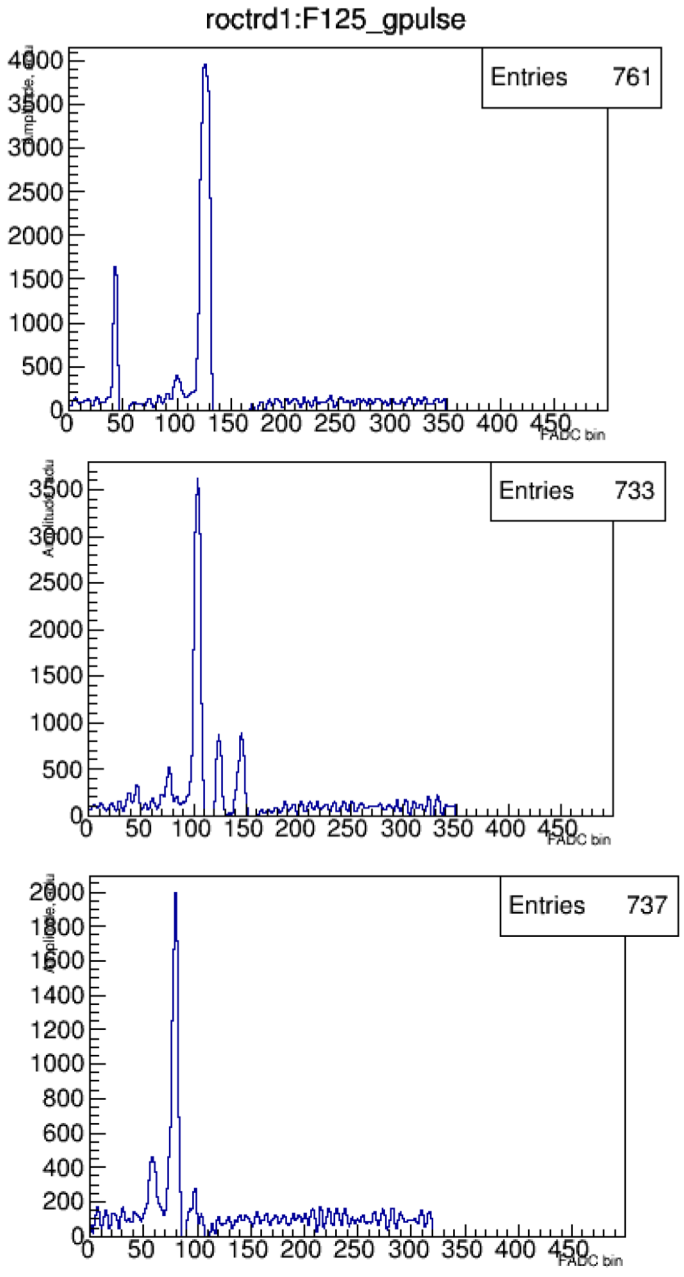


Readout electronics for GEM–TRD

- *The standard tracking GEM readout is usually based on an APV25 chip and measures peak amplitude*
- *TRD needs information about ionization along the track, to discriminate TR photons from energy loss of the particle.*
- *For the TRD test we used a precise 125 MHz, 14 bit flash ADC, developed at JLAB with VME readout.*
  - FADC readout window (pipeline) up to 8 μs
- *Pre-amplifier has GAS-II ASIC chips, provides 2.6 mV/fC amplification and has a peaking time of 10 ns.*



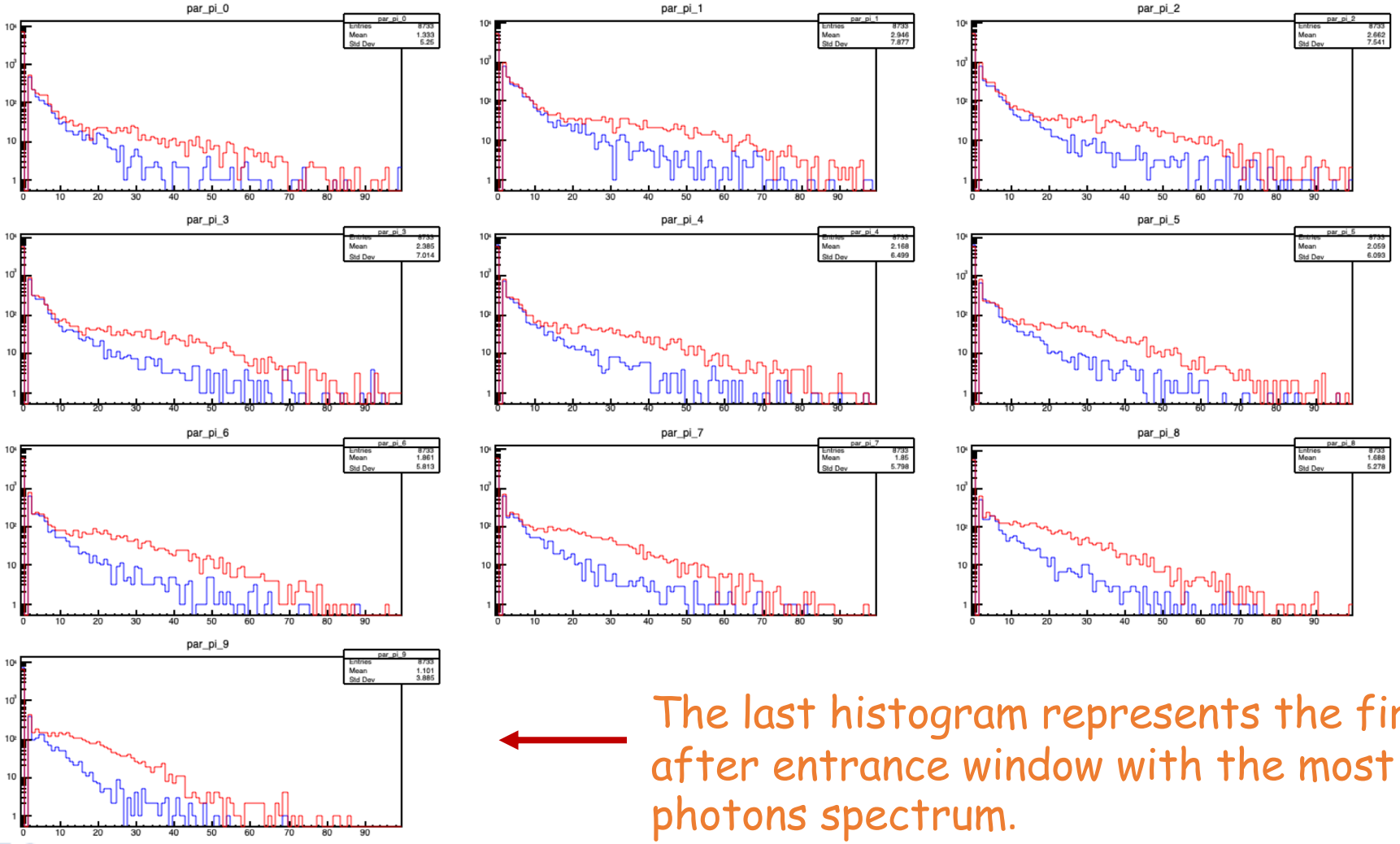




NN input parameters distribution

Jefferson Lab
Thomas Jefferson National Accelerator Facility

Distribution of energy deposition in each of 10 time slices.



The last histogram represents the first time bin after entrance window with the most soft TR photons spectrum.

Comparison Data with MC

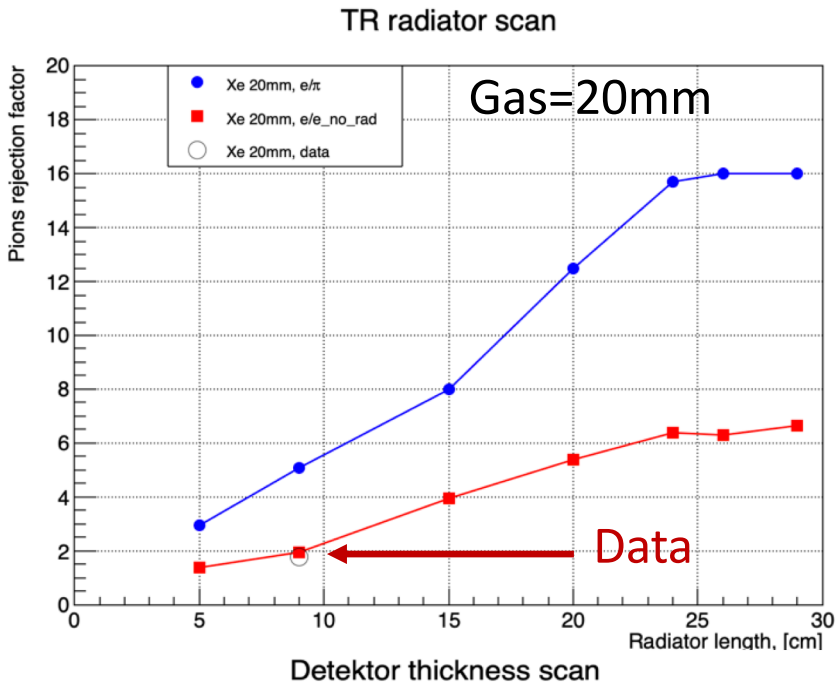
Jefferson Lab
Thomas Jefferson National Accelerator Facility

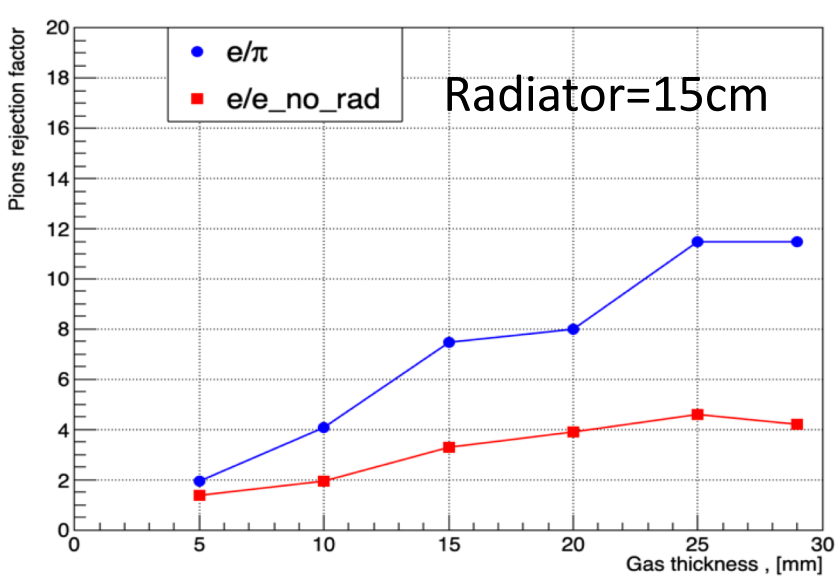
- *GEM-TRD was tested with ~9cm radiator, and has ~21mm drift gap*
- *To understand how far the detector parameters are from the optimal, two Monte Carlo scan were performed:*

1. Fixed gas thickness at 20mm and radiator length varied from 5cm to 30cm
2. Fixed radiator length at 15cm and gas thickness varied from 5mm to 30mm

- *The data point was found in good agreement with Monte Carlo*
- *From MC scans one can predict:*

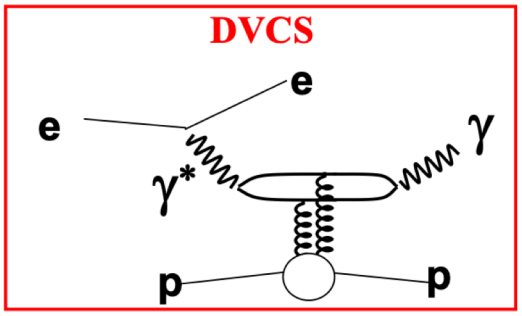
1. The current setup is able to separate e/π with pion rejection factor of ~5.5
2. The detector gas thickness is optimal
3. With radiator length of 25cm e/π rejection will be 16 for a single module.

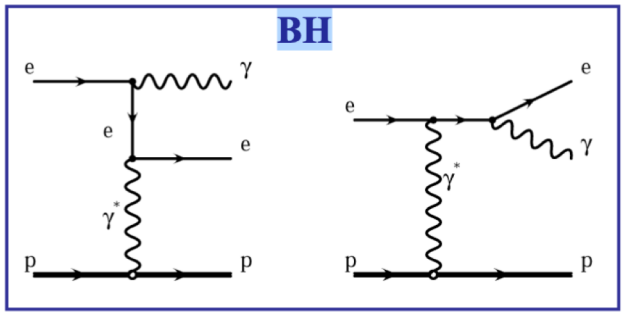




Physics signature examples

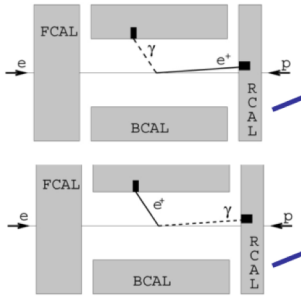
DVCS @ ZEUS - Strategy





two electromagnetic candidates (ordered in energy) and up to one track

BH must be removed **[uncertainty on BH xsec ~ 3%]**



γ sample: no tracks matching to the second candidate — **Signal sample (DVCS+BH)**

e sample: a track match to the second candidate — **Control sample (BH+ dilepton + J/ψ)**

Wrong-sign sample: a negative track match to the second candidate — **Control sample (dilepton + J/ψ)**

Kinematic region:
$1.5 < Q^2 < 100$ GeV^2
$40 < W < 170$ GeV

XILINX

CASE STUDY

https://www.xilinx.com/publications/powered-by-xilinx/cerncasestudy-final.pdf

RESULTS:

Achieving 100ns Inference Latency on 150 Terabytes/Second Data Rates

The data rate of the CMS detector is staggering and what makes the trigger filtering problem such a unique challenge. To overcome these challenges, extremely low-latency inference times are produced by the team's machine learning algorithms running on the Xilinx FPGAs. The data rates coming into the CMS are measured in hundreds of terabytes/second. The FPGAs receive and align sensor data, perform tracking and clustering, machine learning object identification, and trigger functions, before formatting and delivery of event data.

"Whether it's low-level aggregation of hits in some calorimeter all the way up to taking the full event and optimizing for a particular topology. It allows the spread and adoption of machine learning more quickly across the experiment."