Jefferson Lab
Thomas Jefferson National Accelerator Facility

GEMTRD readout

Sergey Furletov
Jefferson Lab

on behalf of the eRD22 group

Joint Streaming readout VII meeting

18 Nov 2020

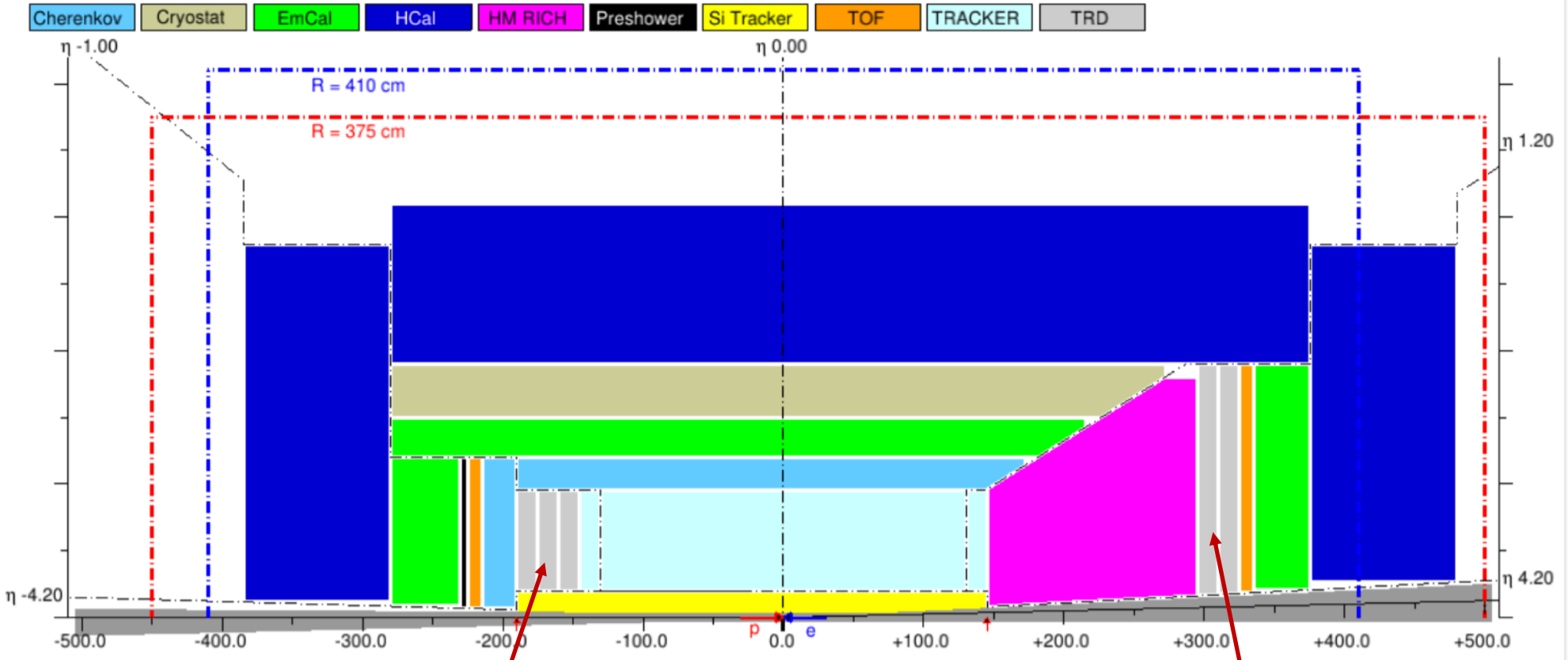
Outline

- *Experimental setup*
- *Readout electronics*
- *Offline PID analysis with ML (root / TMVA)*
- *Moving on to FPGA*
- *Outlook*

(*) Field Programmable Gate Array

GEMTRD at EIC

Jefferson Lab
Thomas Jefferson National Accelerator Facility



GEMTRD in the electron endcap

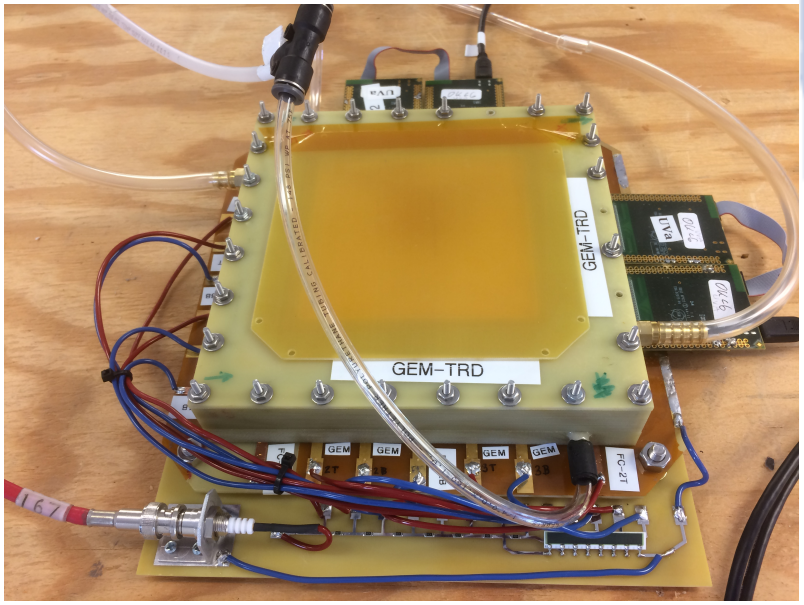
GEMTRD in the forward region

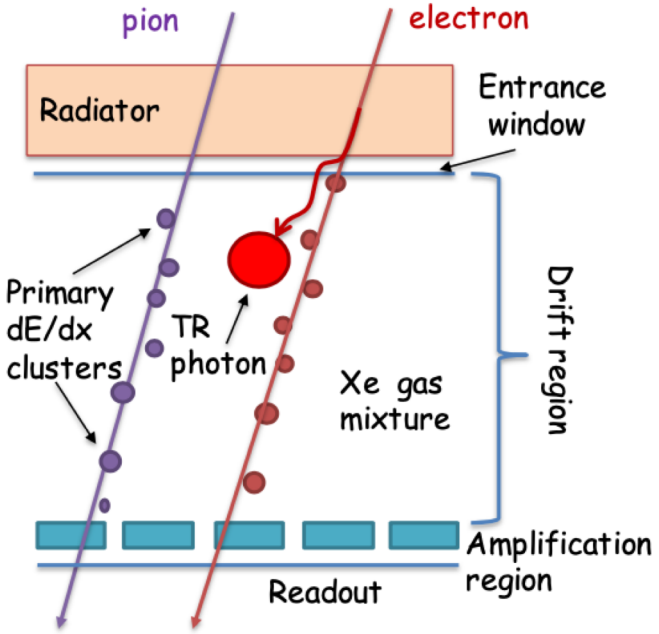
GEMTRD provides PID and also participates in track reconstruction

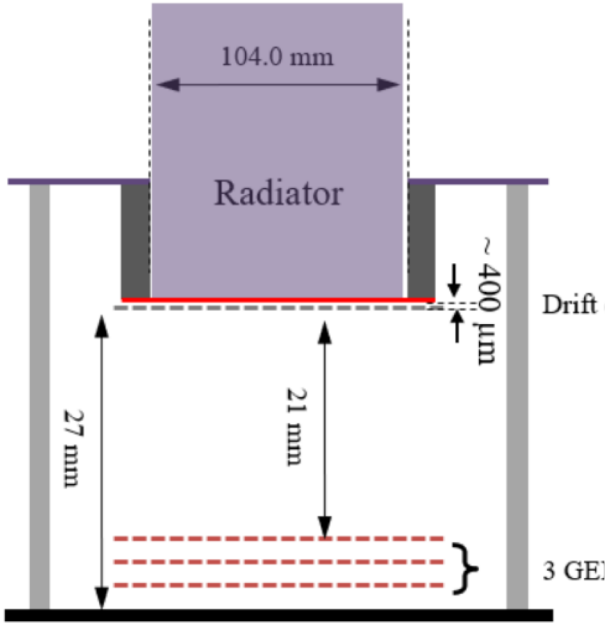
GEM-TRD prototype

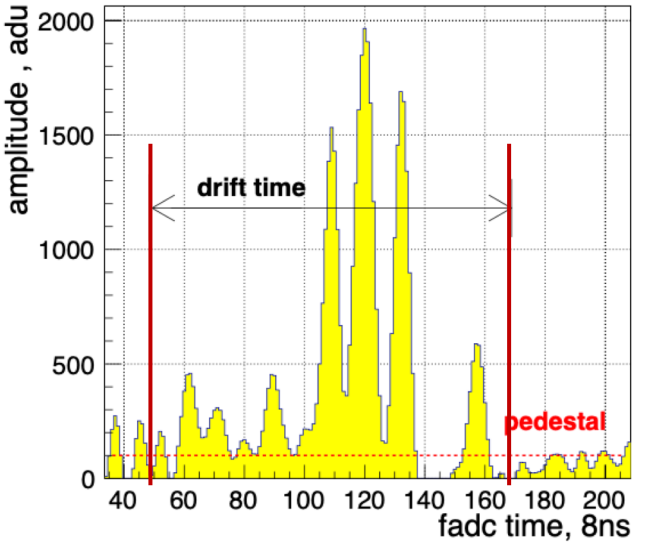
- A test module was built at the University of Virginia
- The prototype of GEMTRD/T module has a size of 10 cm × 10 cm with a corresponding to a total of 512 channels for X/Y coordinates.
- The readout is based on flash ADC system developed at JLAB (fADC125) @125 MHz sampling.

- GEM-TRD provides e/hadron separation and tracking



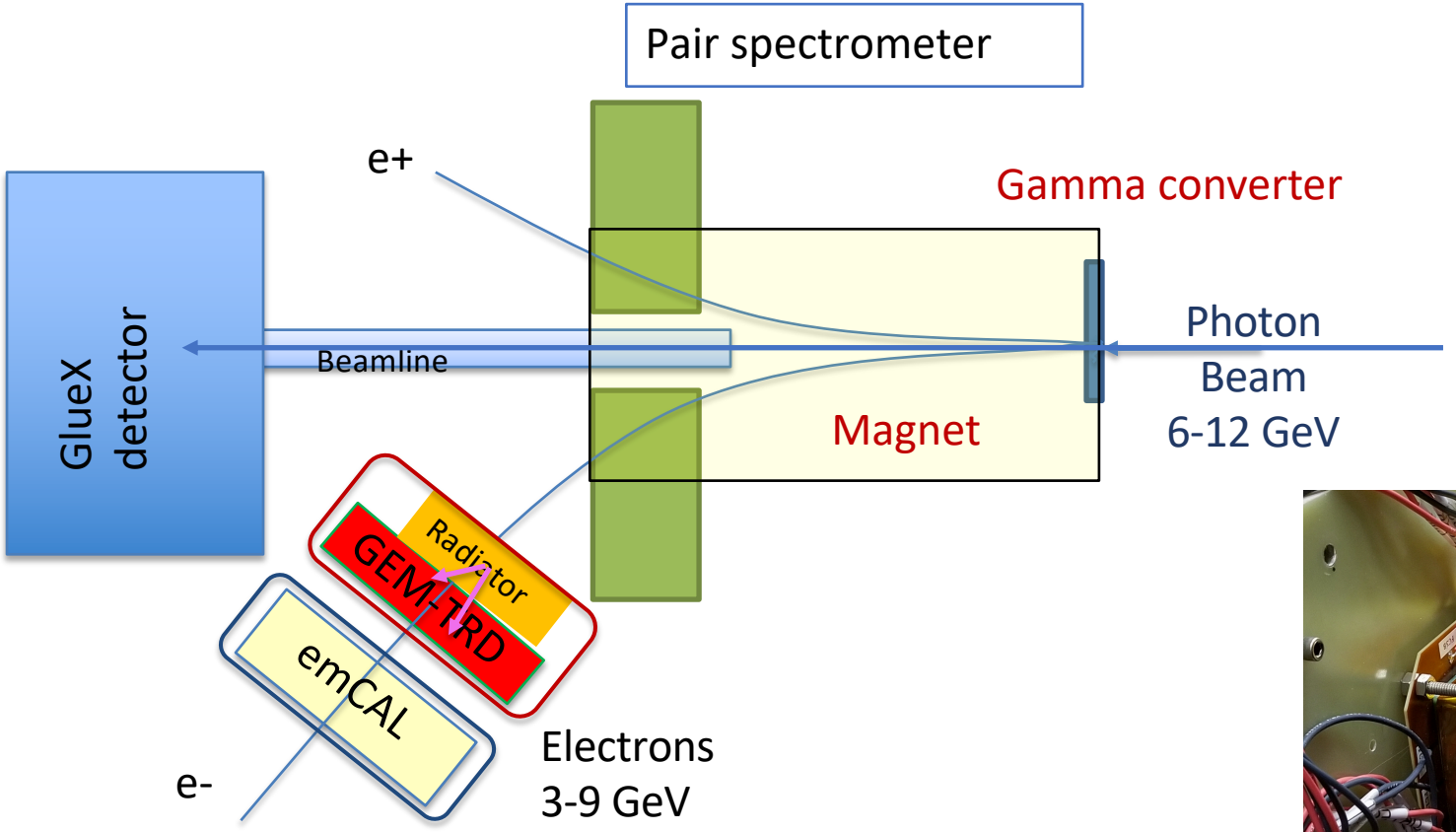


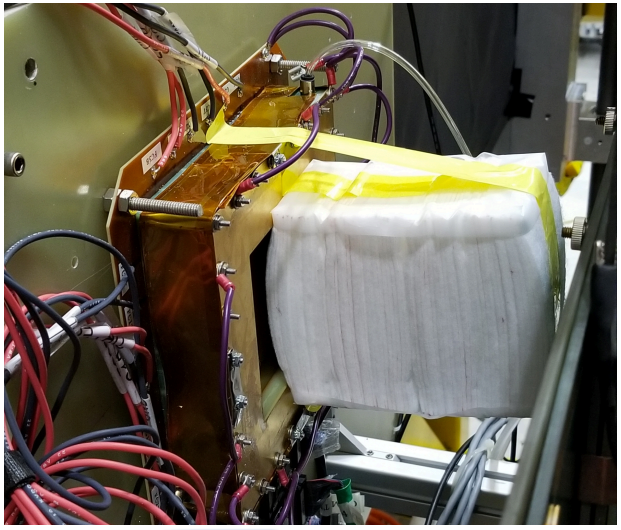




Beam setup at JLab Hall-D

- *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.*

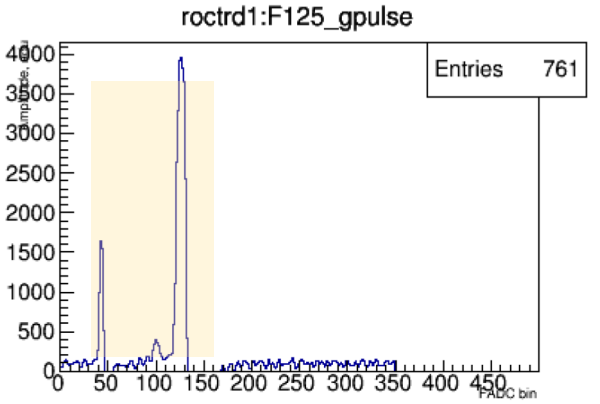


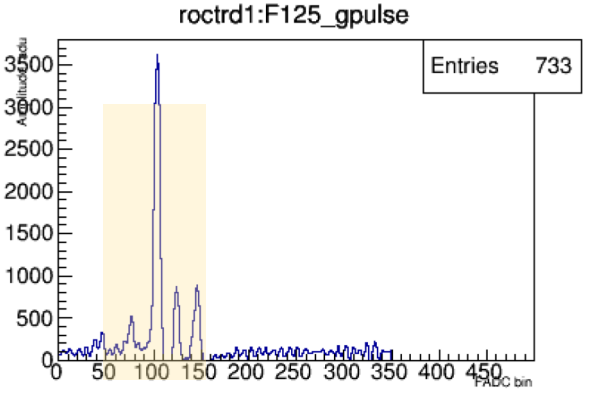


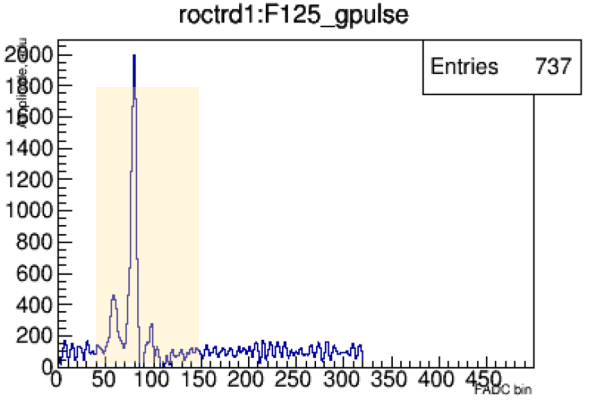
Readout electronics for GEMTRD

Jefferson Lab
Thomas Jefferson National Accelerator Facility

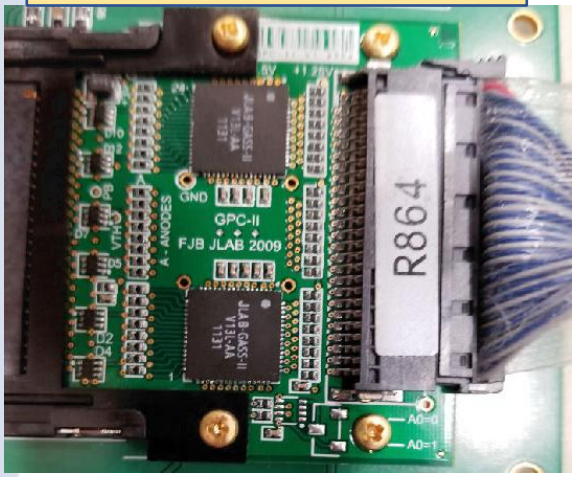
- *The standard tracking GEM readout is usually based on an APV25 chip and measures peak amplitude*
- *TRD needs information about ionization along the track, to discriminate TR photons from energy loss of the particle.*
- *For the TRD test we used a precise 125 MHz, 14 bit flash ADC, developed at JLAB (Fadc125) 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.*



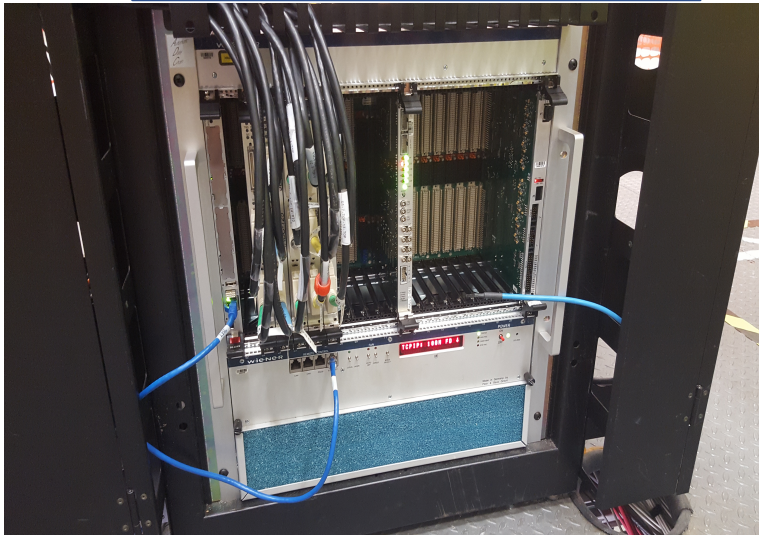




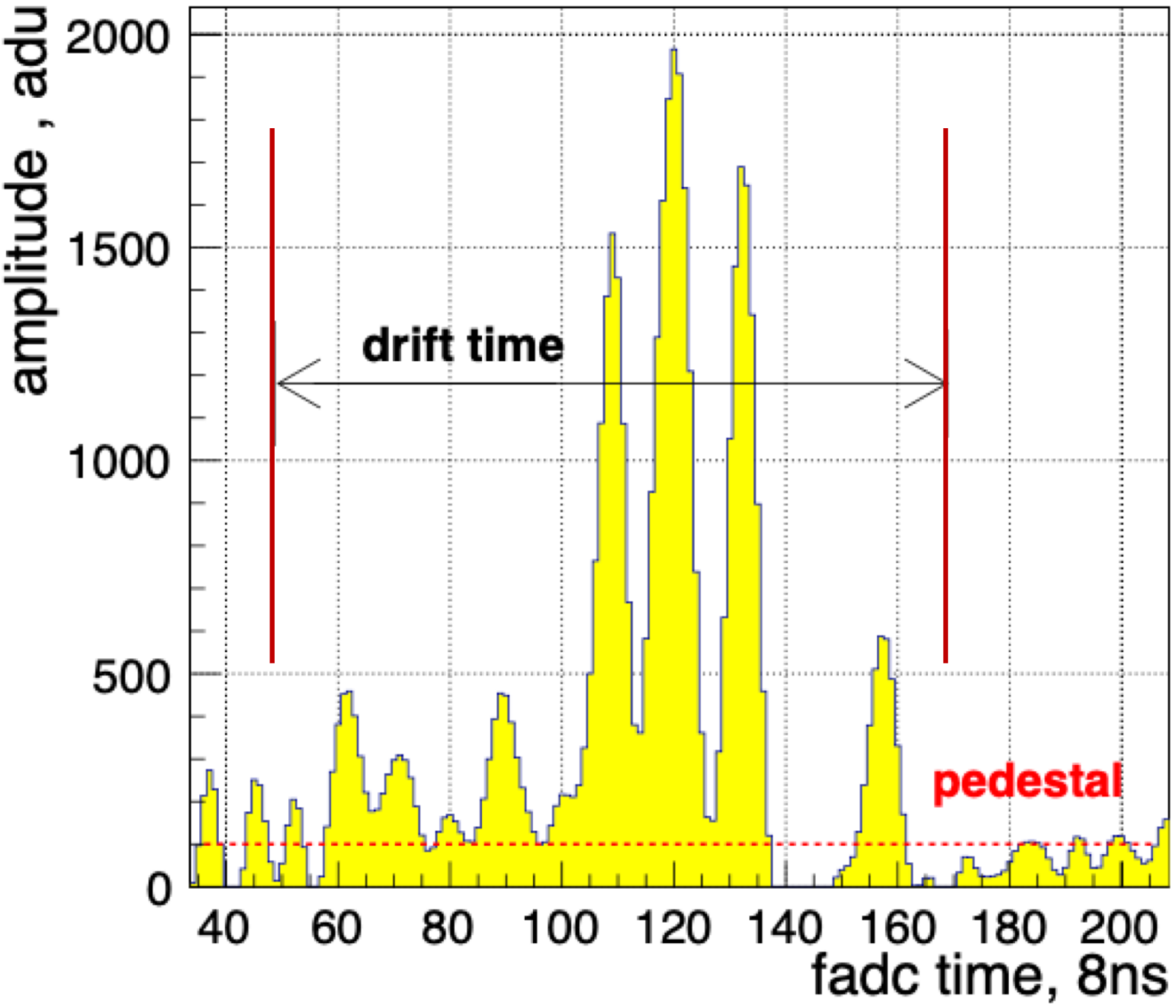
Preamplifiers board



VME crate with Fadc125

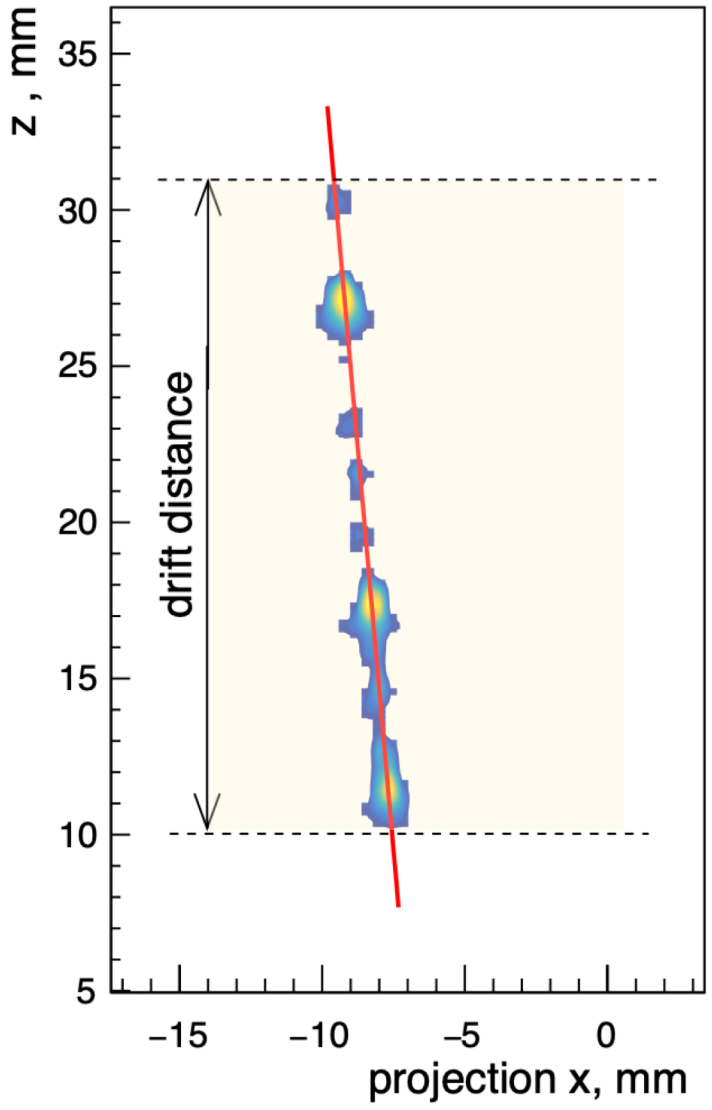


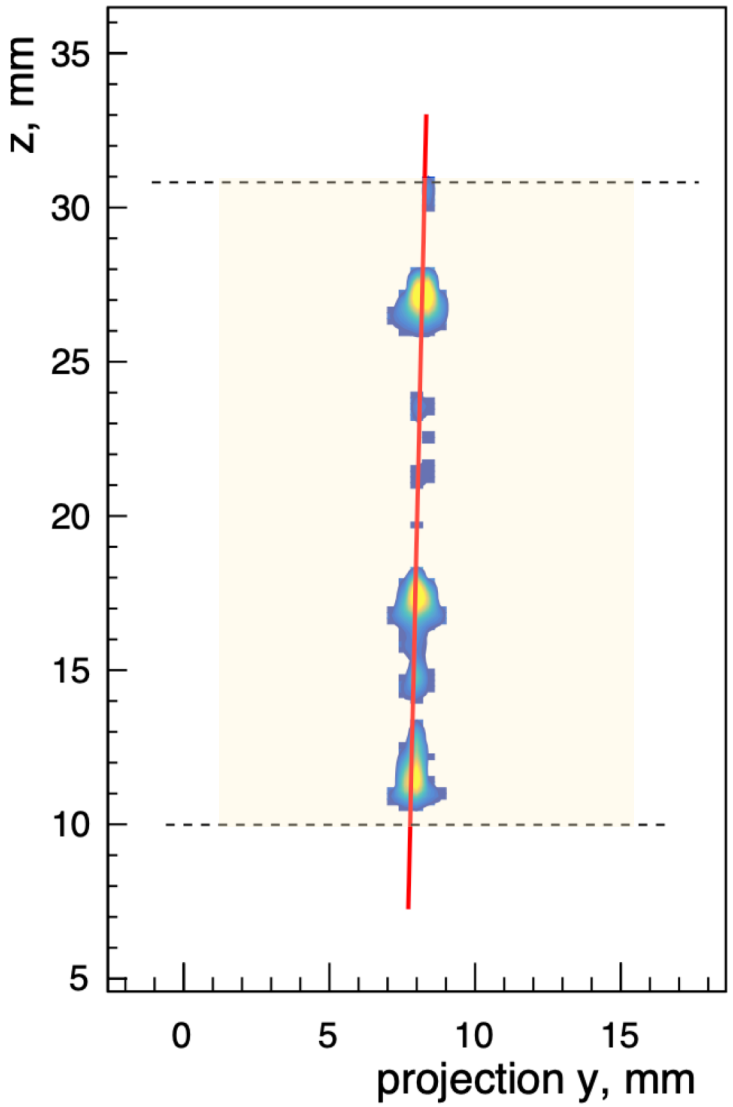
Fadc125 signal



GEMTRD clusters on the track

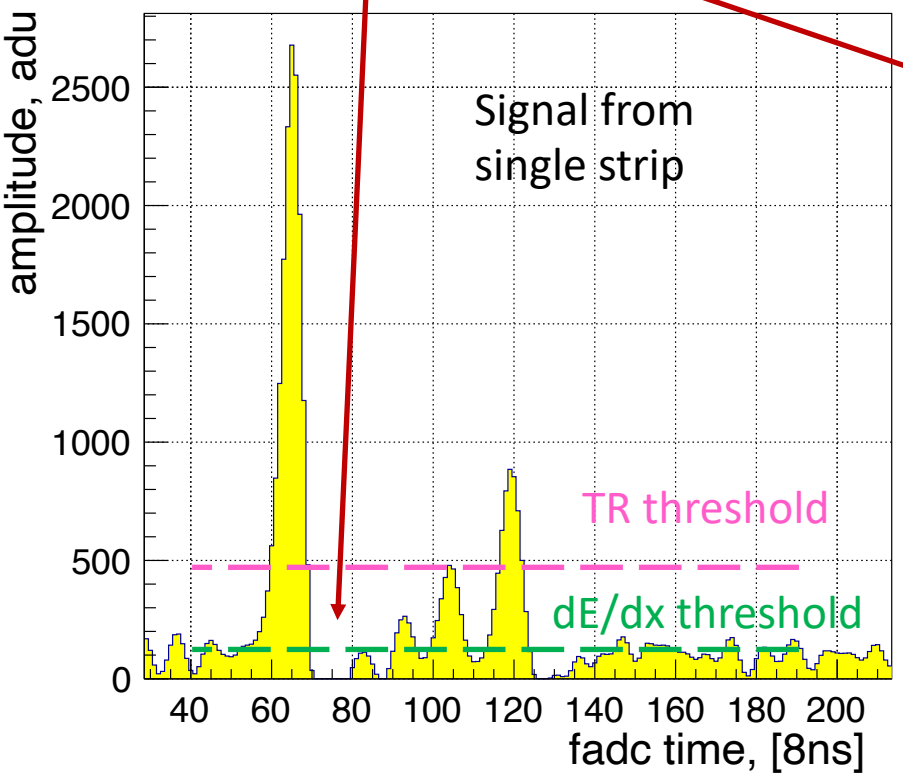
GEM-TRD can work as mini TPC, providing 3D track segments

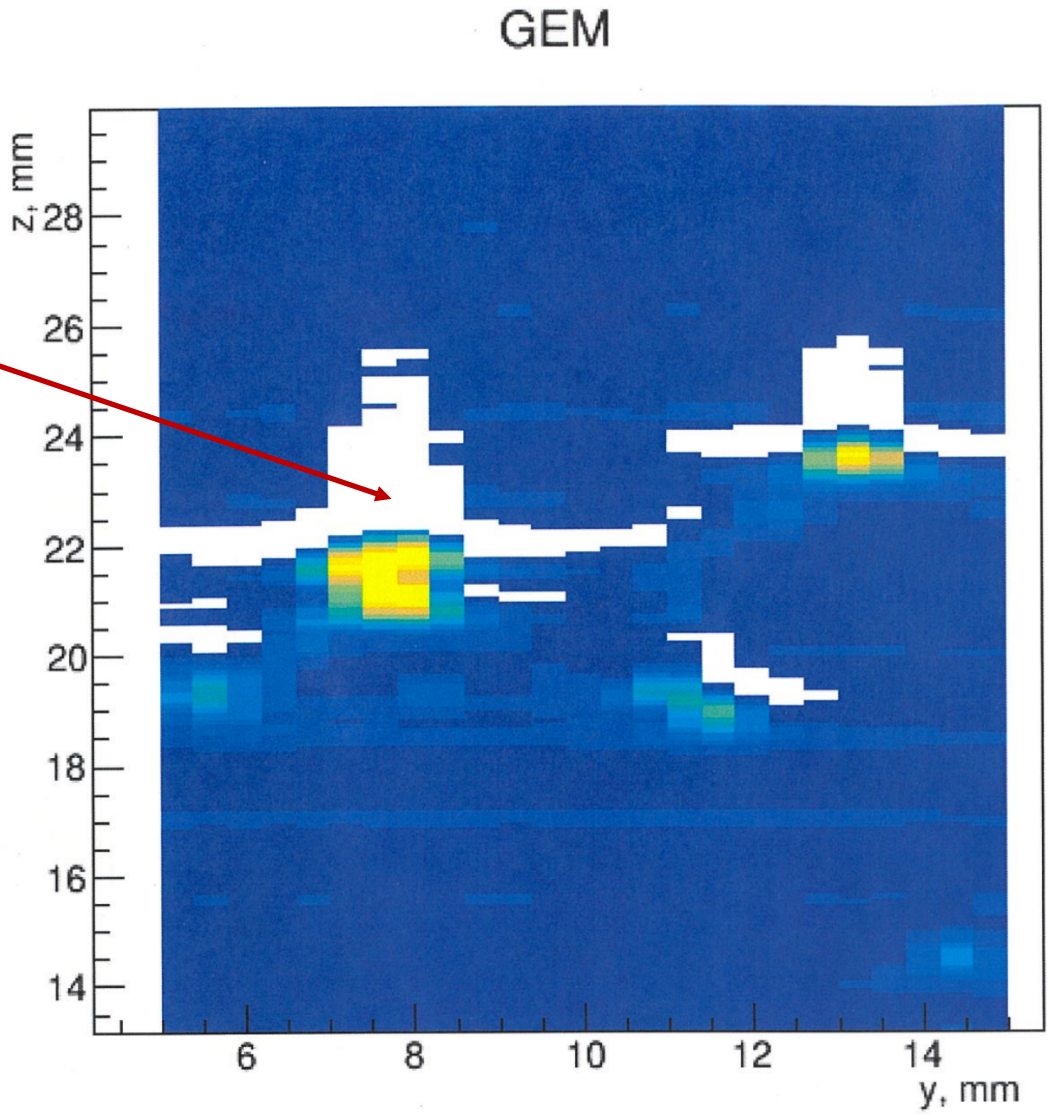




GAS-II preamp and shaper

undershoot leads to some inefficiency in counting of TR clusters.



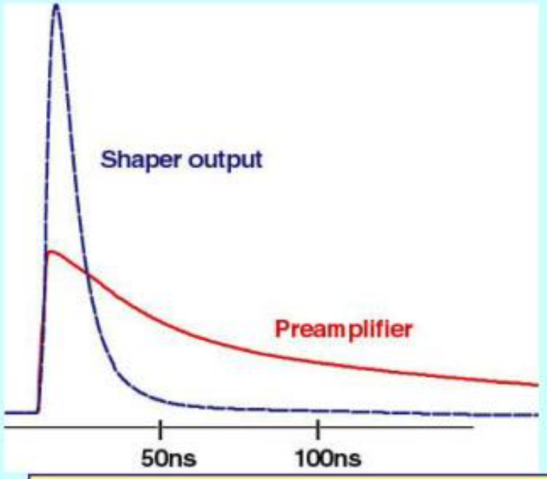


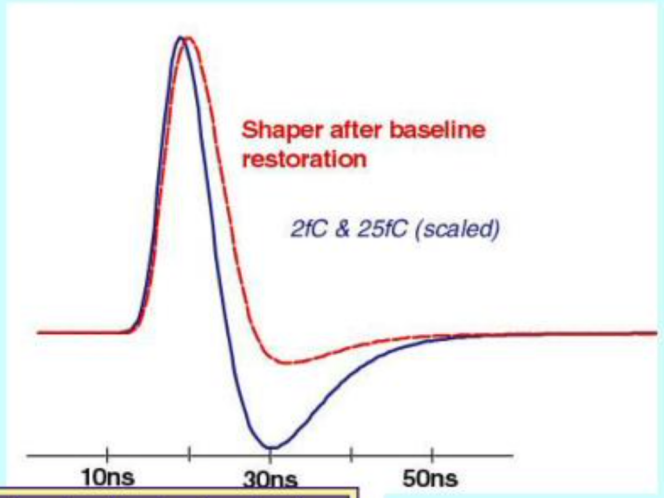
ASDBLR chip

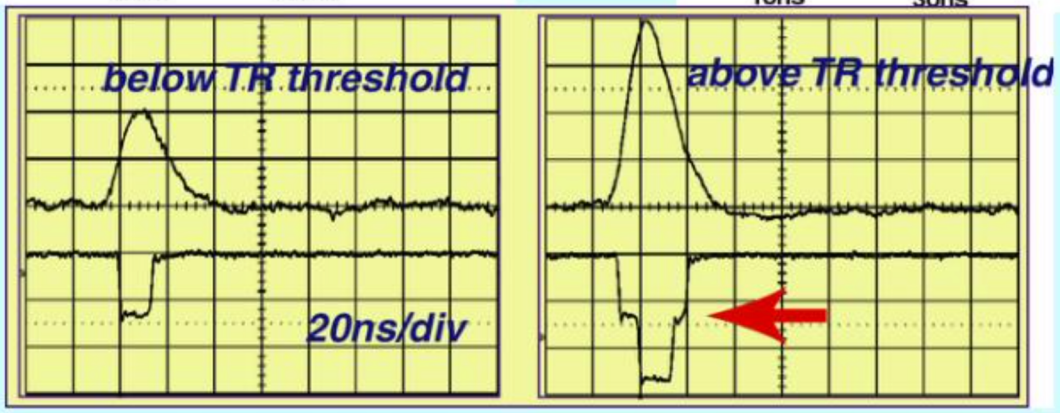
Jefferson Lab
Thomas Jefferson National Accelerator Facility

ATLAS TRT ASDBLR front end

- Amplifier =>tail cancellation and baseline restoration
  selectable for CF_4 and Xe gas mixtures







4mm straw + Xenon based gas

g.hall@ic.ac.uk www.hep.ph.ic.ac.uk/~hallg/ 9 November 2001

Readout electronics

	Sampling MHz	ns/bin	Peaking time	Pipeline / stream	Channels/chip cost	ADC bits	Remarks
FADC125 + GAS-II preamp. (JLAB)	125	8	30ns	8μs or stream	$50/channel	12bit	External preamps (GAS-II) : -Undershooting -No baseline restorer
VMM3 (ATLAS)	4	250	25-200ns		64chan/chip	10bit	L0 or continuous
SAMPA (ALICE)	10-20	100-50	80ns, 160ns	Stream 3.2Gbit/s	32chan/chip 30$/chip 1$/channel	10bit	500ns- return to baseline Baseline restorer, DSP (zero-suppression, thr)
ALPHACORE							
Minimal requirements	80		30ns	Stream		10bit	

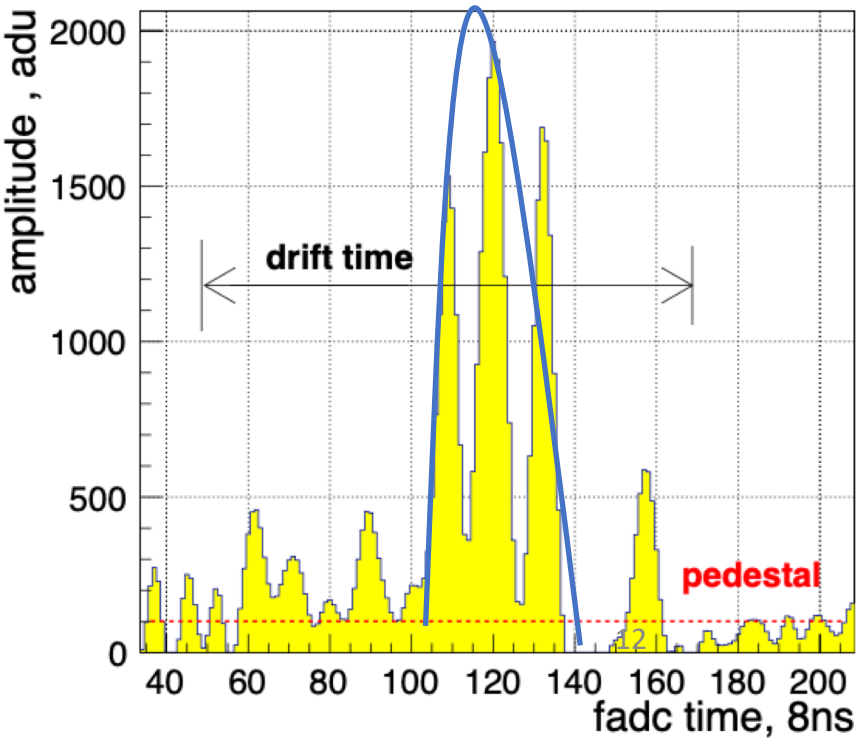
SAMPA ASIC

Jefferson Lab
Thomas Jefferson National Accelerator Facility

- SAMPA chip works great with regular GEM for tracking.
- For GEMTRD, it has too long an integration time.

Fastest peak time of 80 ns is too slow for cluster separation and counting.

ALICE TPC upgrade and the SAMPA ASIC

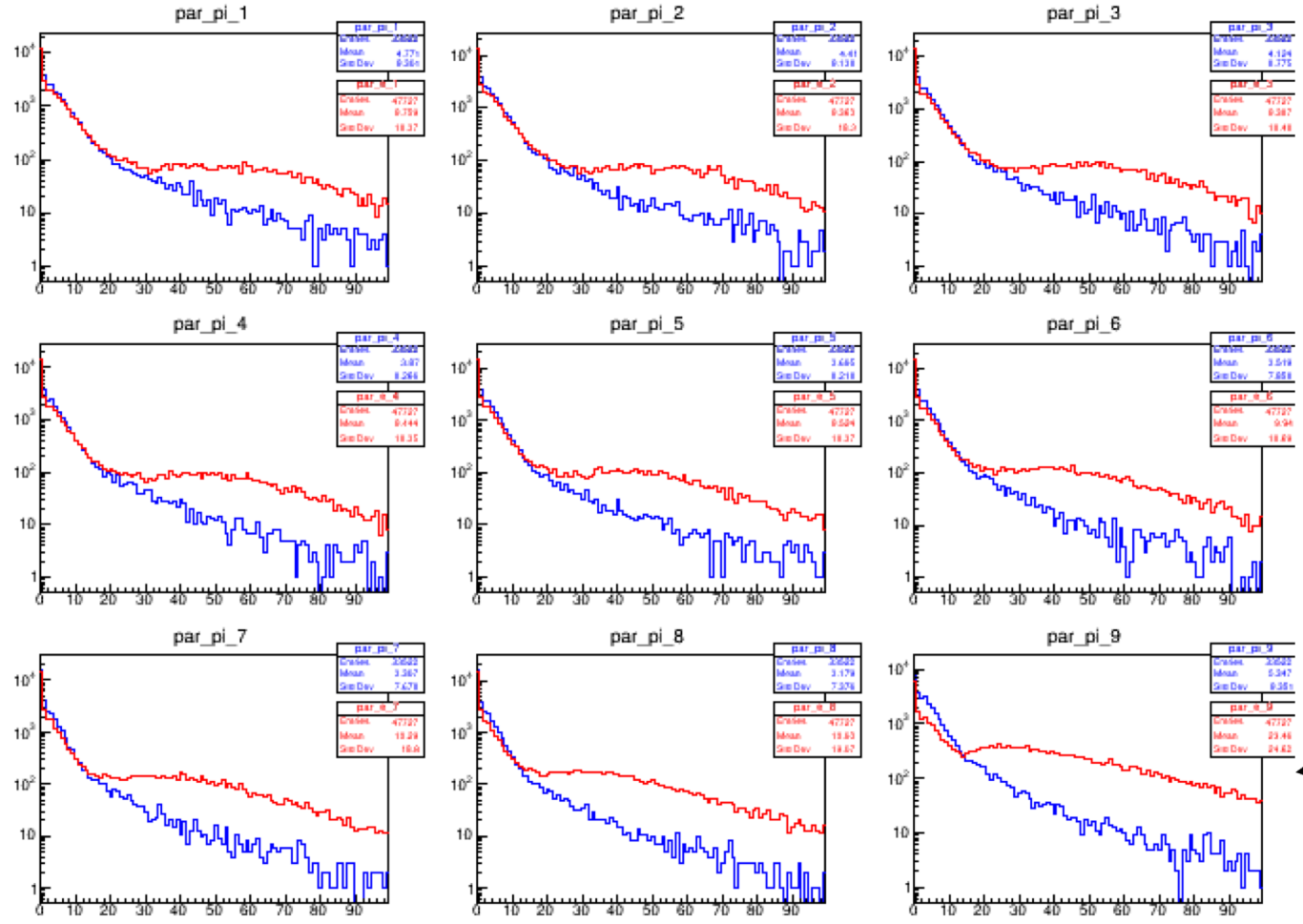


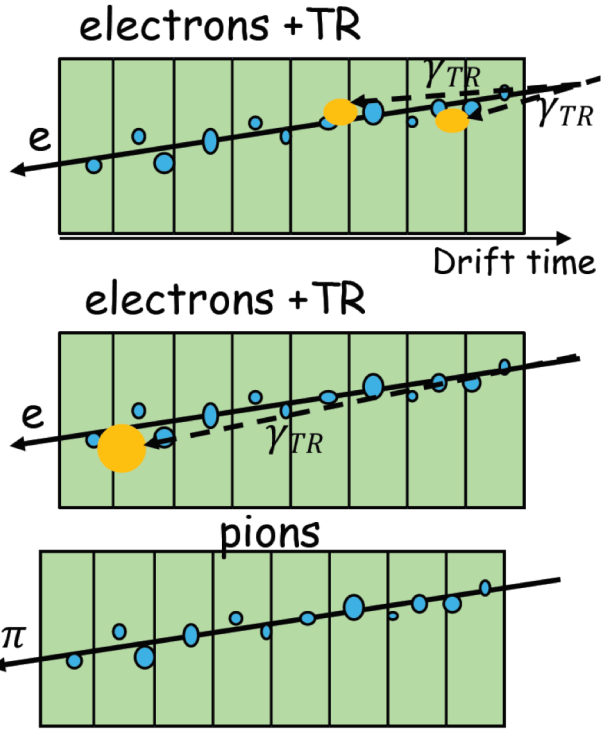
Data analysis

- TR photons move forward at a small angle within $1/\gamma$, practically along the path of the original particle, and are detected together with dE/dx from the particle..
- There are several methods that are used to discriminate TR photons and dE/dx from particle
  1. Cluster counting method
     - use one threshold on ionization amplitude (just above average dE/dx), assuming that energy deposition from TR photons is a point like and produces cluster with high amplitude. Method is widely used with straw based TRD.
  2. Total energy deposition
  3. Separation in space
     - Require high resolution detector (silicon pixels) to see natural angular distribution of TR photons, or magnetic field to deflect particle from TR photons.
  4. In case of measurements of ionization along the track, the likelihood or neural network methods can be used for separation of electrons and pions.
- For this test we used ionization along the track and Neural Network (Machine Learning)

Input parameters for ML

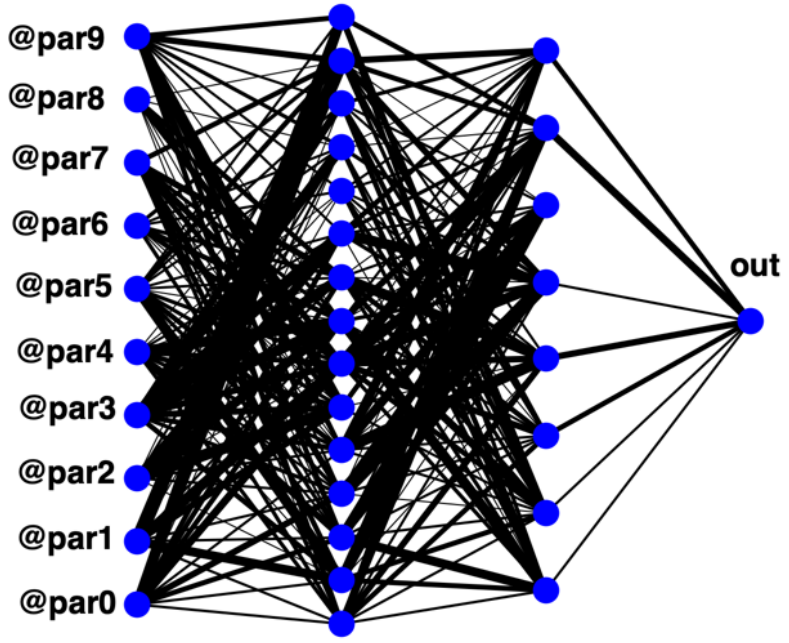
Jefferson Lab
Thomas Jefferson National Accelerator Facility

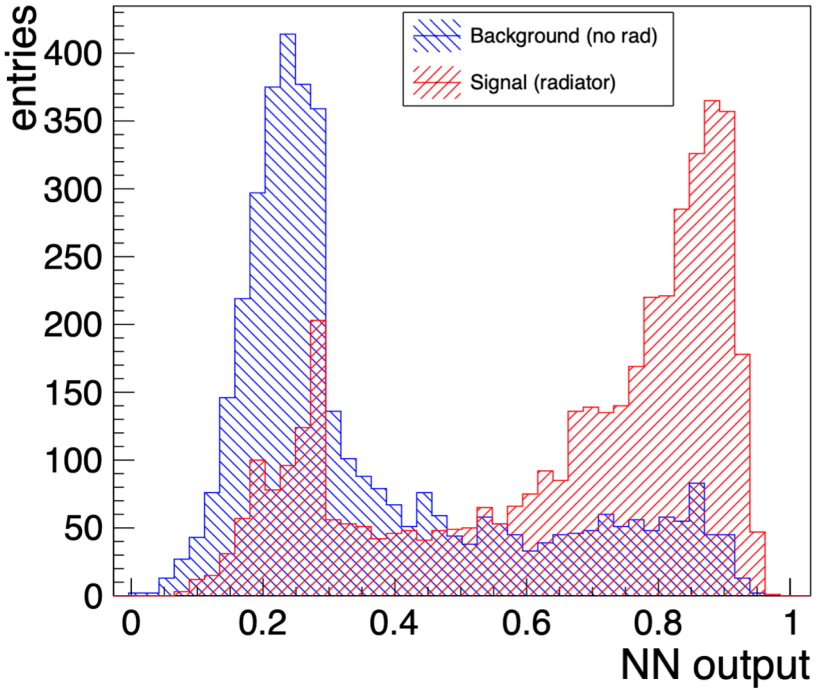




The last histogram represents the first time bin after entrance window with the most soft TR photons spectrum.

GEMTRD offline analysis

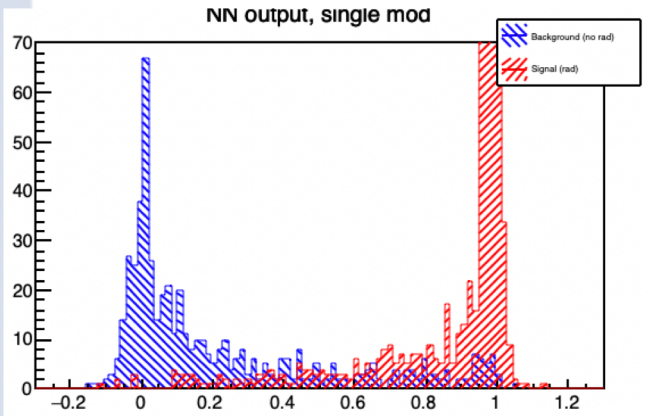


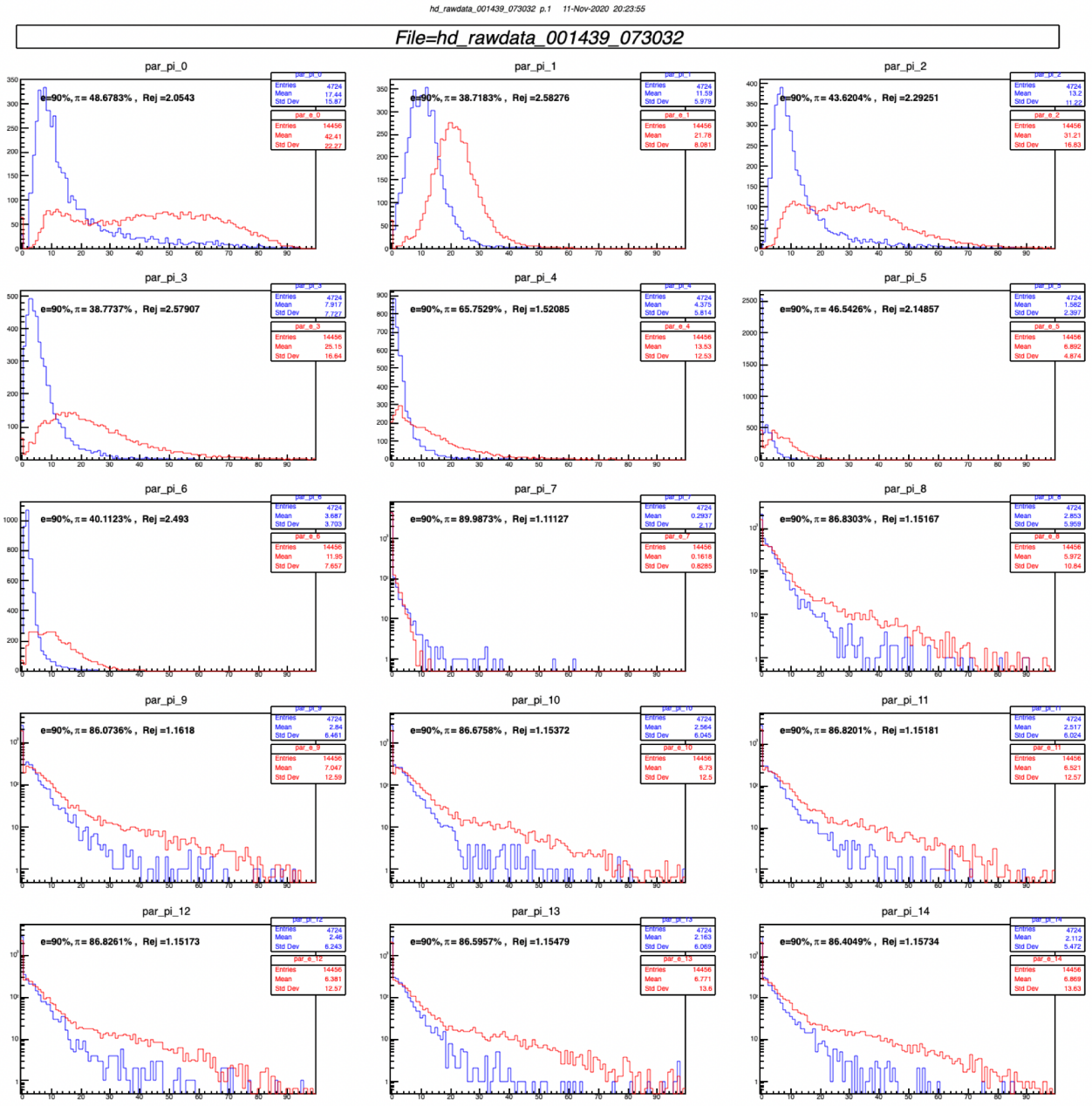


- *For data analysis we used a neural network library provided by root /TMVA package : MultiLayerPerceptron (MLP)*
- *All data was divided into 2 samples: training and test samples*
- *Top right plot shows neural network output for single module:*
  - Red - electrons with radiator
  - Blue – electrons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

NN input parameters

To improve the rejection, in addition to the dE/dx information, some integral and statistical parameters were added: the number of clusters, the number of hits, etc.





Moving forward

Jefferson Lab
Thomas Jefferson National Accelerator Facility

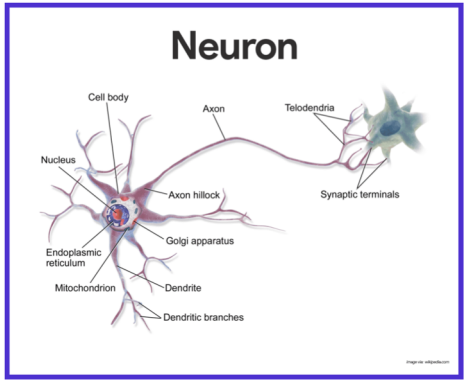
- *Offline analysis using ML looks promising.*
- *Can it be done in real time ?*
- *Here are some of the possible solutions :*
  - Computer farm.
  - CPU + GPU
  - CPU + FPGA
  - FPGA only

- *Steps for beginners 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

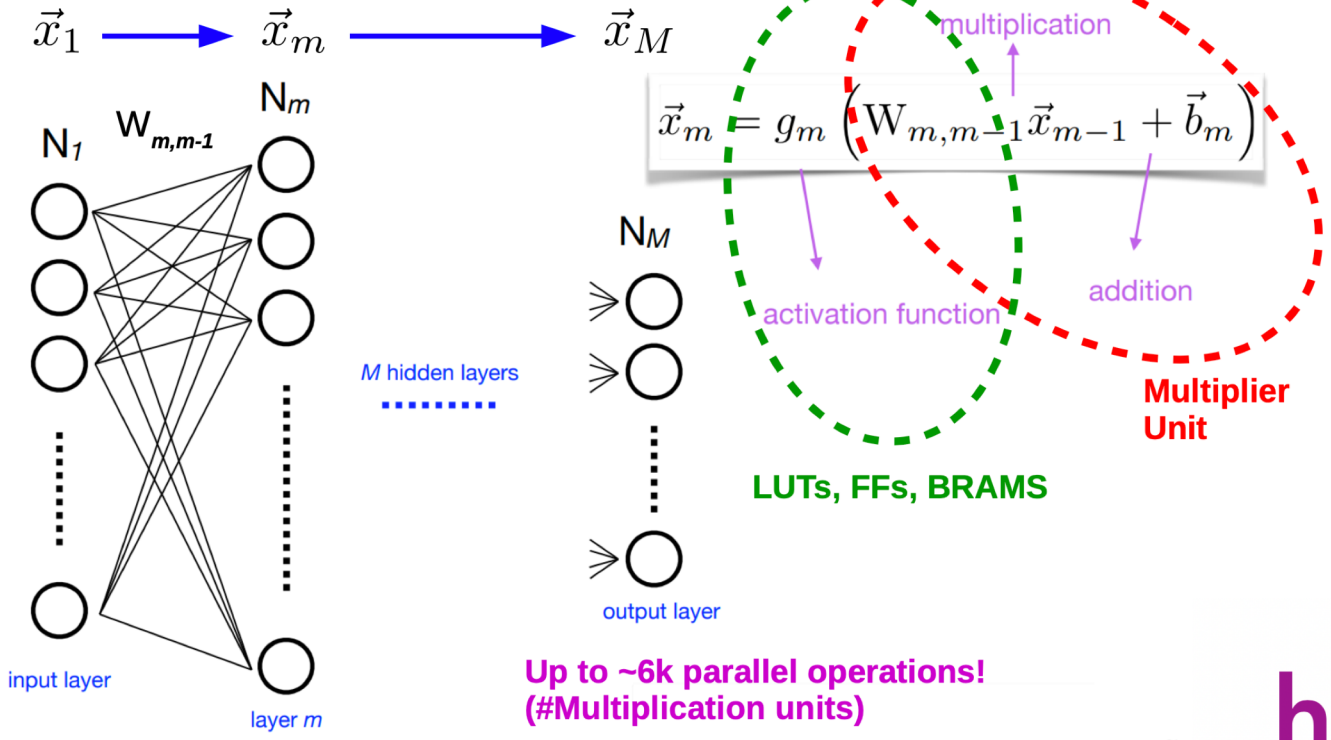
Jefferson Lab
Thomas Jefferson National Accelerator Facility

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



Inference on an FPGA

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

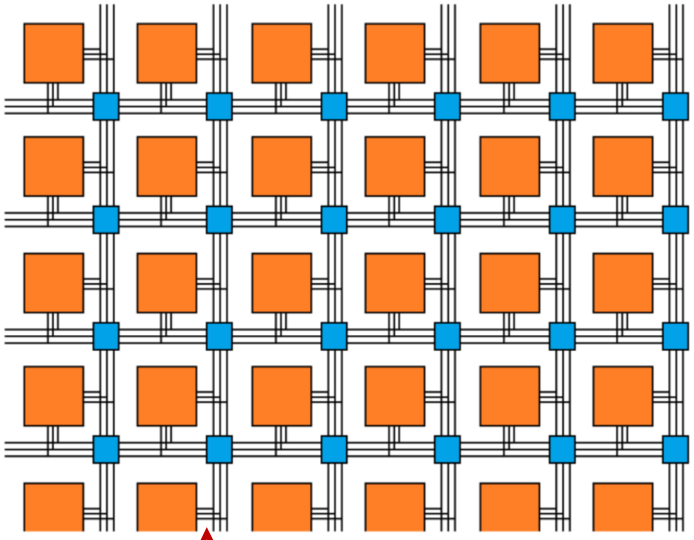


Up to ~6k parallel operations!
(#Multiplication units)

hls4ml

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

FPGA structure



- Fist FPGA have only programmable gates and routers: Field Programmable Gate Array .
- It can perform logical operation in parallel using LUTs and FFs.
- There are problems with the math operation required by the neural network.

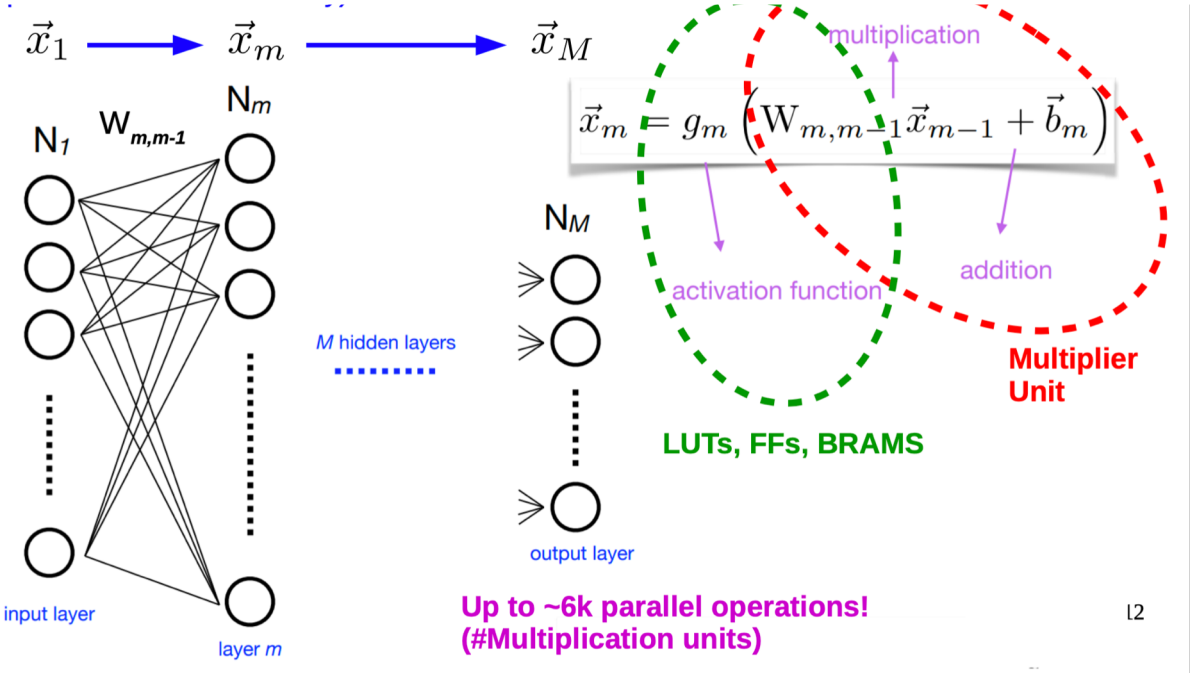
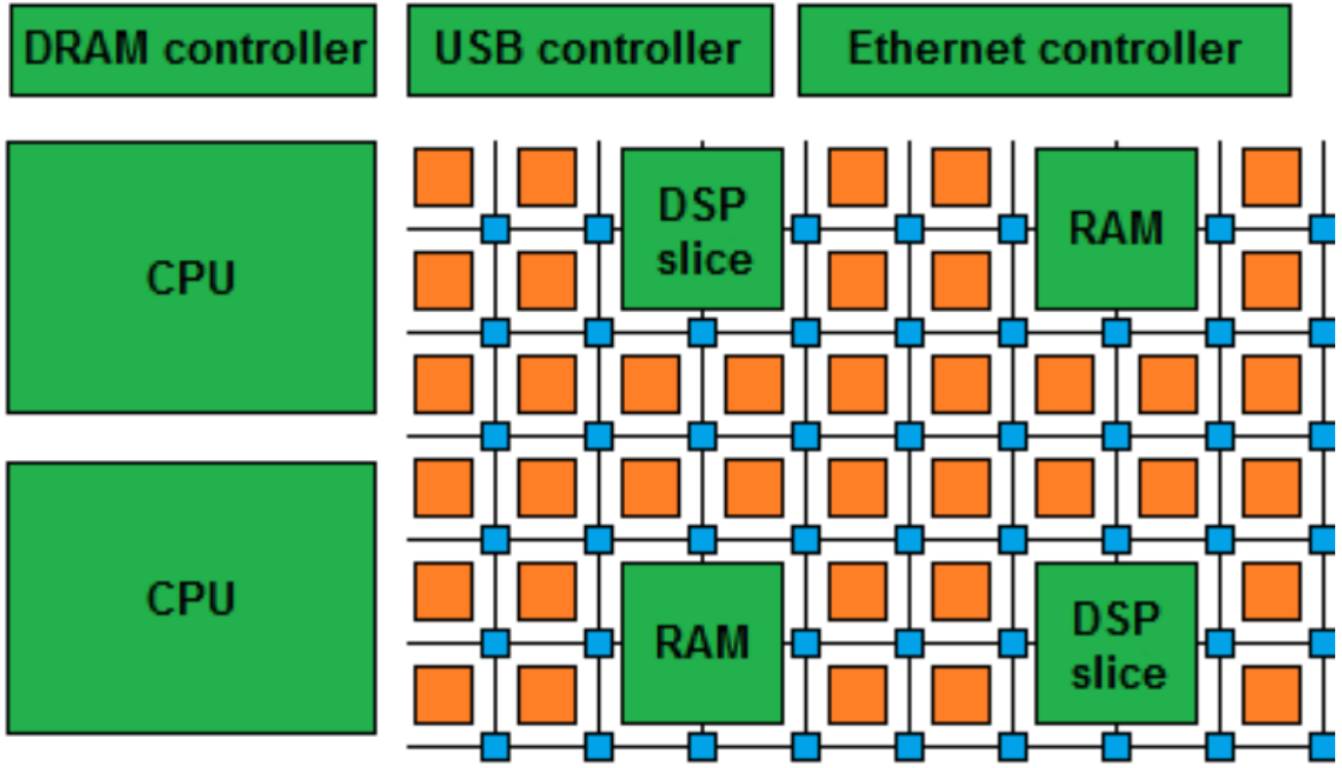


Image from: https://www.embeddedrelated.com/showarticle/195.php

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 Virtex® UltraScale+™

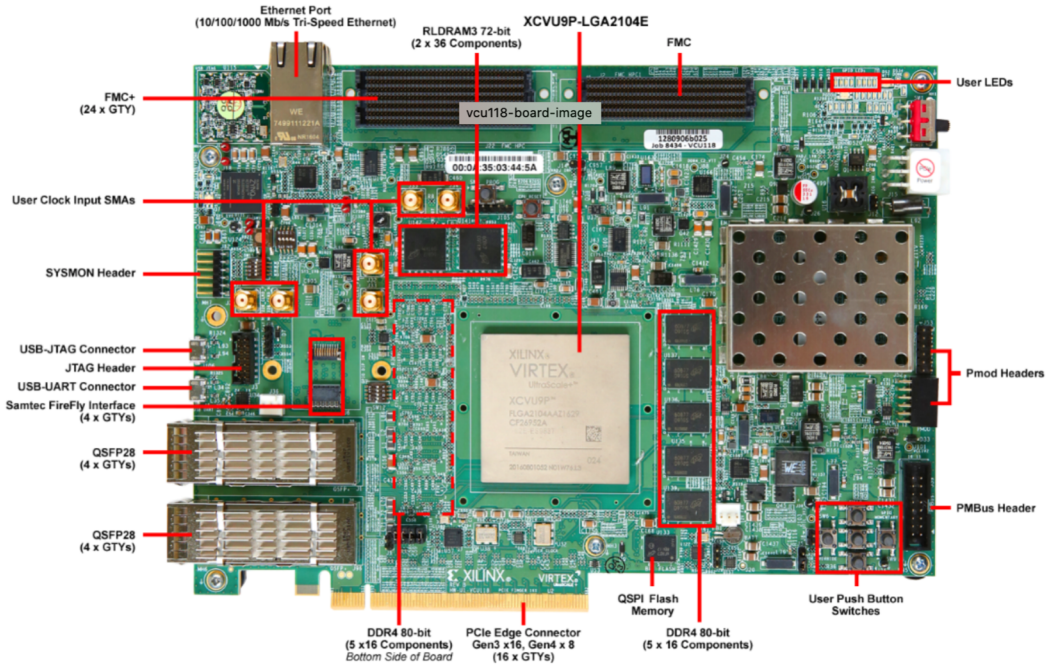
Jefferson Lab
Thomas Jefferson National Accelerator Facility

- *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 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+™

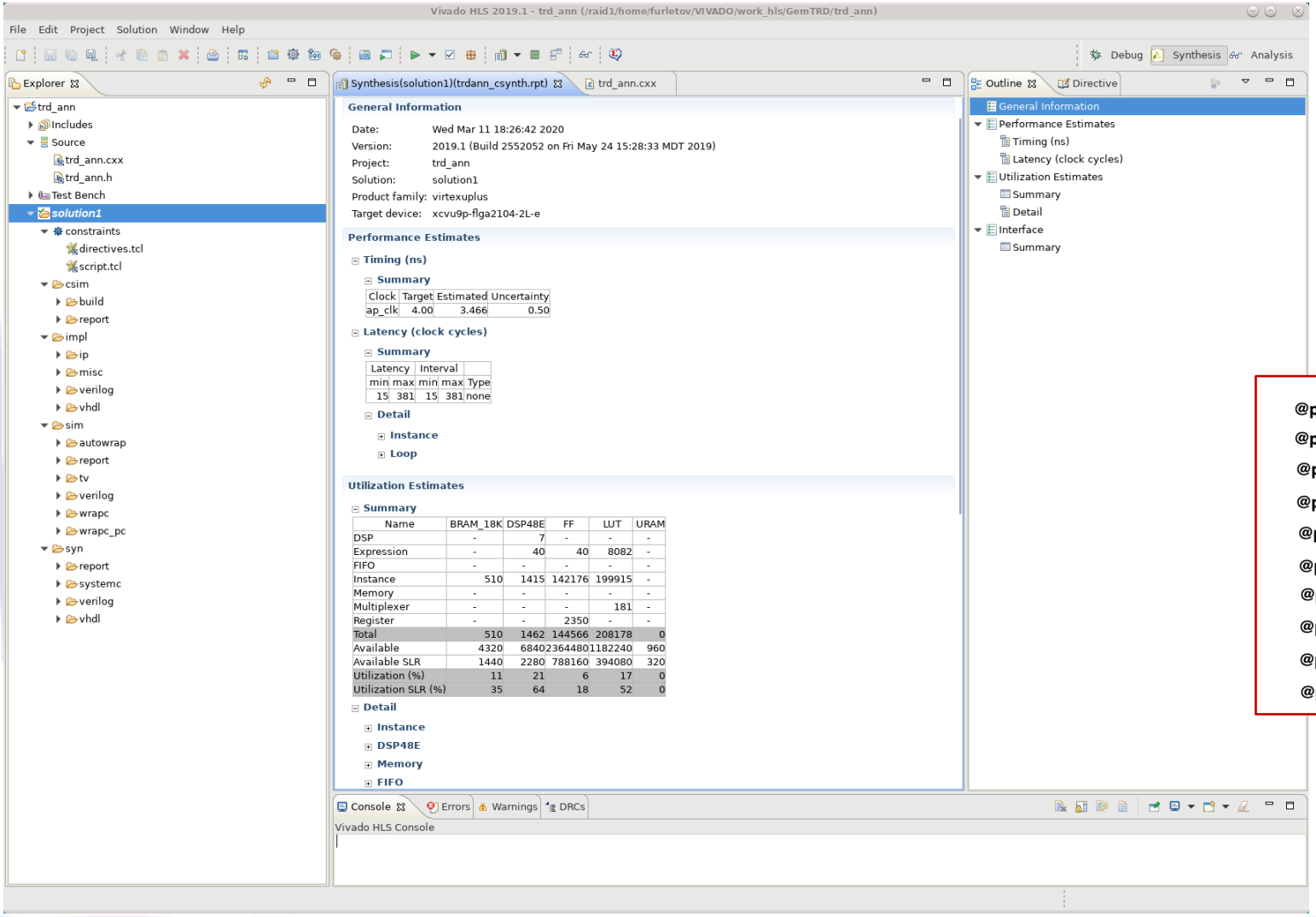
Evaluation board XCVU9P includes software license (node locked & device-locked) with 1 year of updates.

Featuring the **Virtex® UltraScale+™ XCVU9P-L2FLGA2104E FPGA**

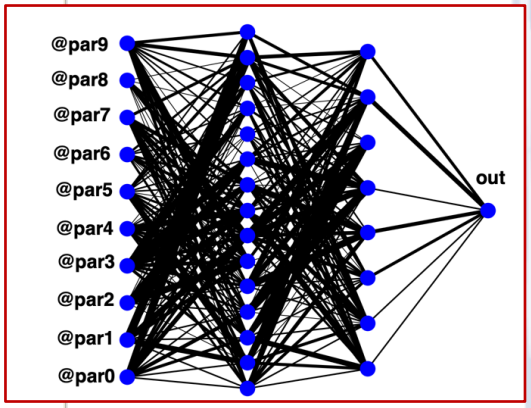


Xilinx High-Level Synthesis

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.



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



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)/(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++

Verilog

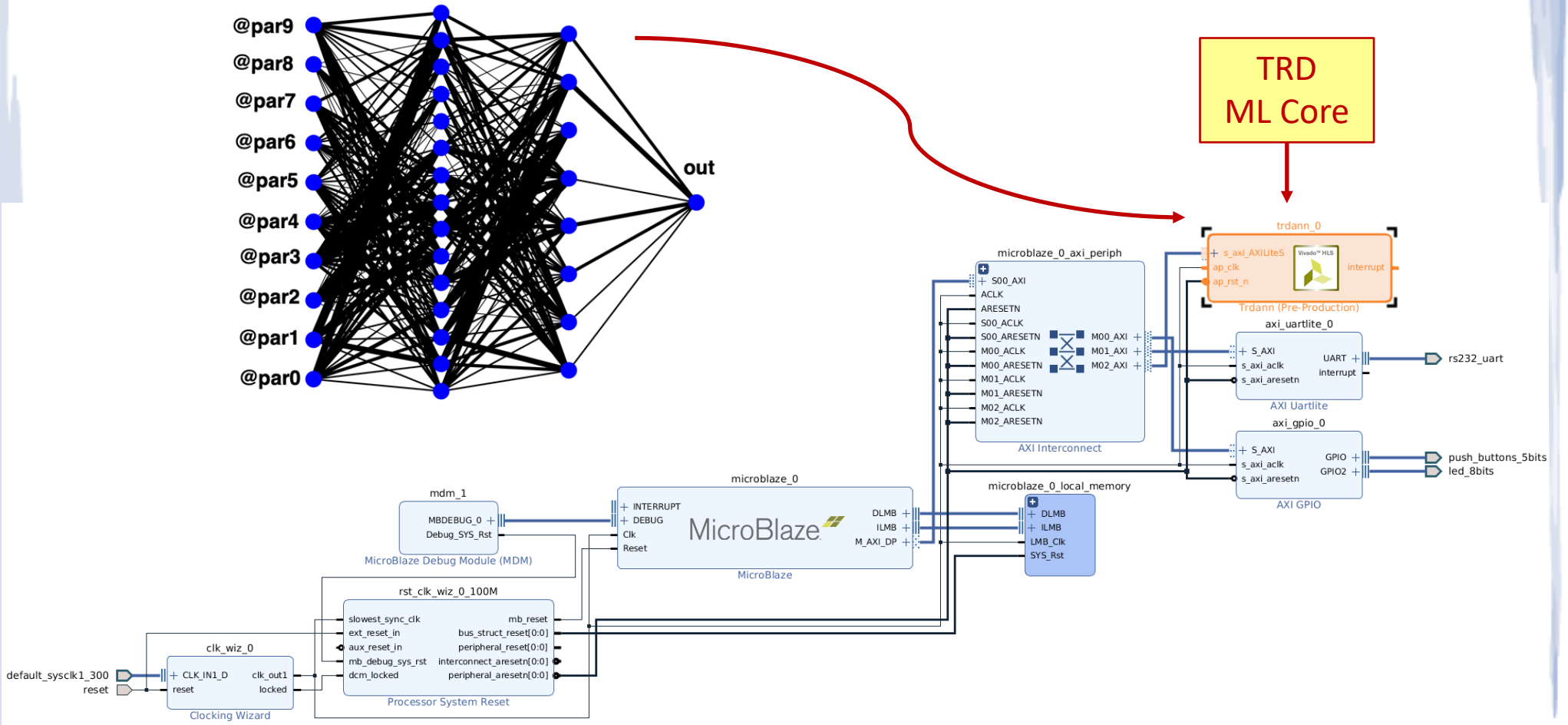
Note: fixed point calculation

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;
```

ML FPGA Core for TRD

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



Vivado implementation report

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

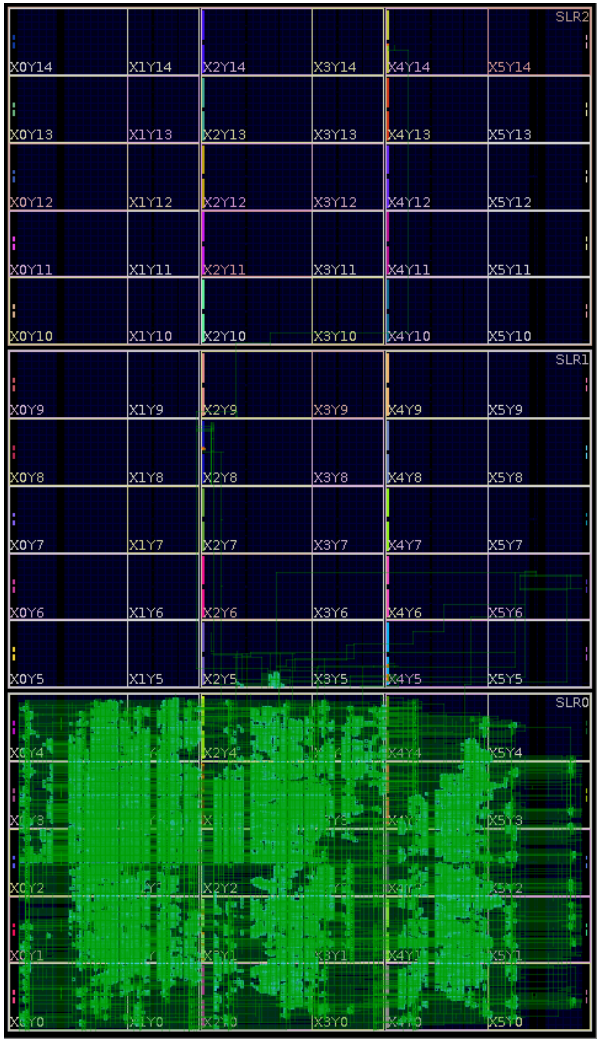
Latency		Interval		
min	max	min	max	Type
15	381	15	381	none

Initial latency estimation:
From 60 ns to 1.5 μs.

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



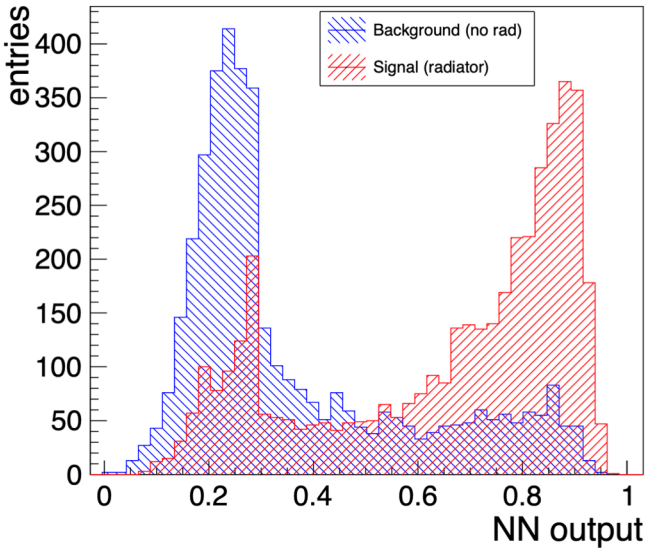
Test ML FPGA

Jefferson Lab
Thomas Jefferson National Accelerator Facility

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 :

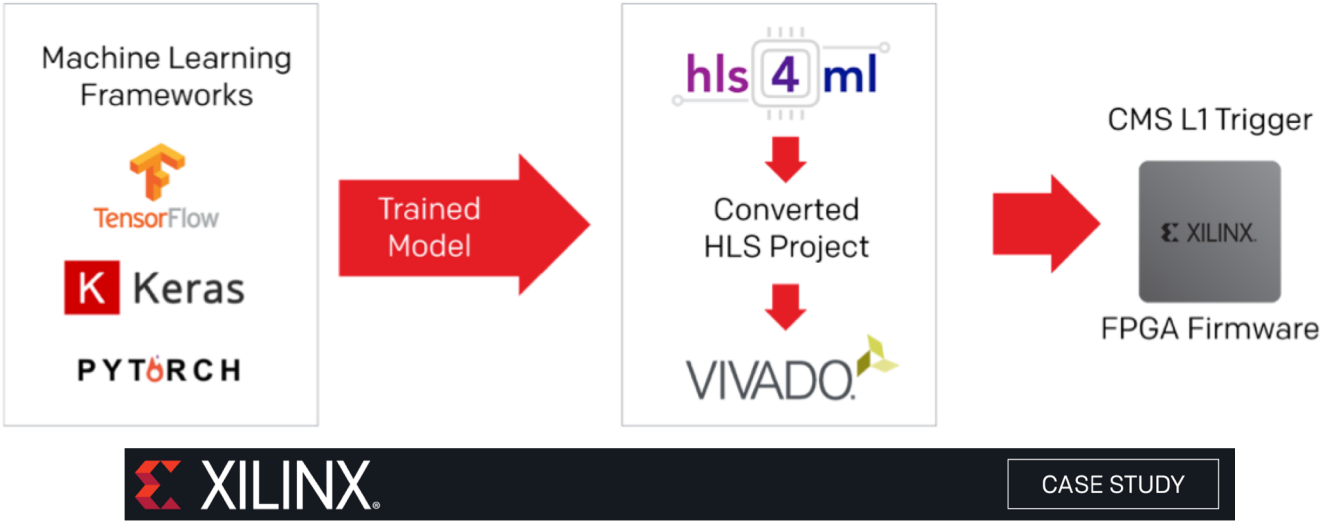
XTrdann ann; // create an instance of ML core.

```
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);
}
```

hls4ml software

- ***hls4ml*** is a software package for creating HLS implementations of neural networks
- Supports common layer architectures and model software
- Highly customizable output for different latency and size needs
- Simple workflow to allow quick translation to HLS

https://fastmachinelearning.org/hls4ml/

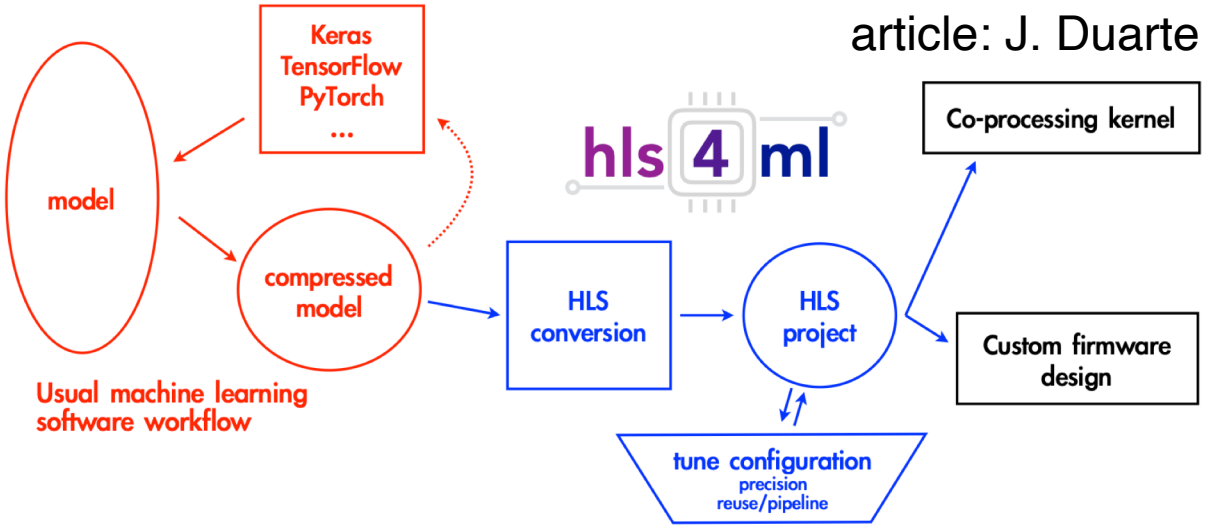


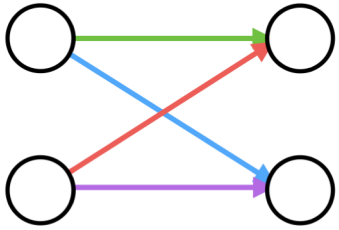
Optimization with hls4ml package

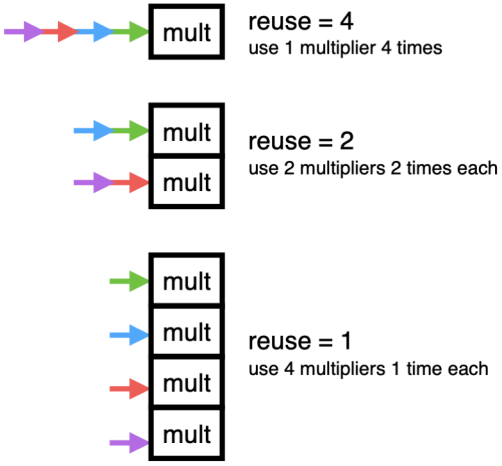
Jefferson Lab
Thomas Jefferson National Accelerator Facility

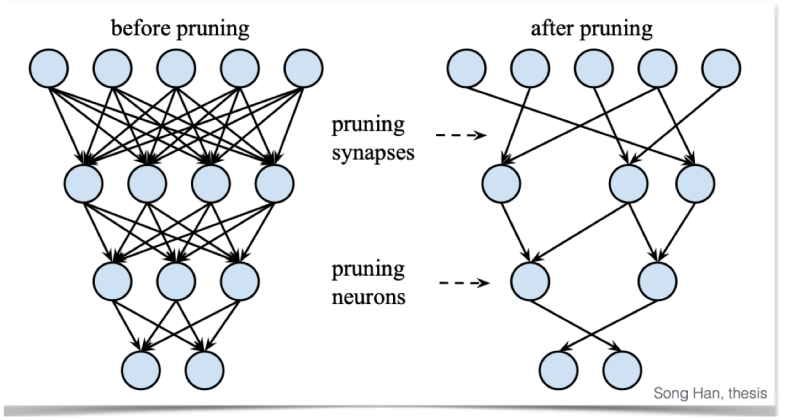
- A package hls4ml is developed based on High-Level Synthesis (HLS) to build machine learning models in FPGAs.

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



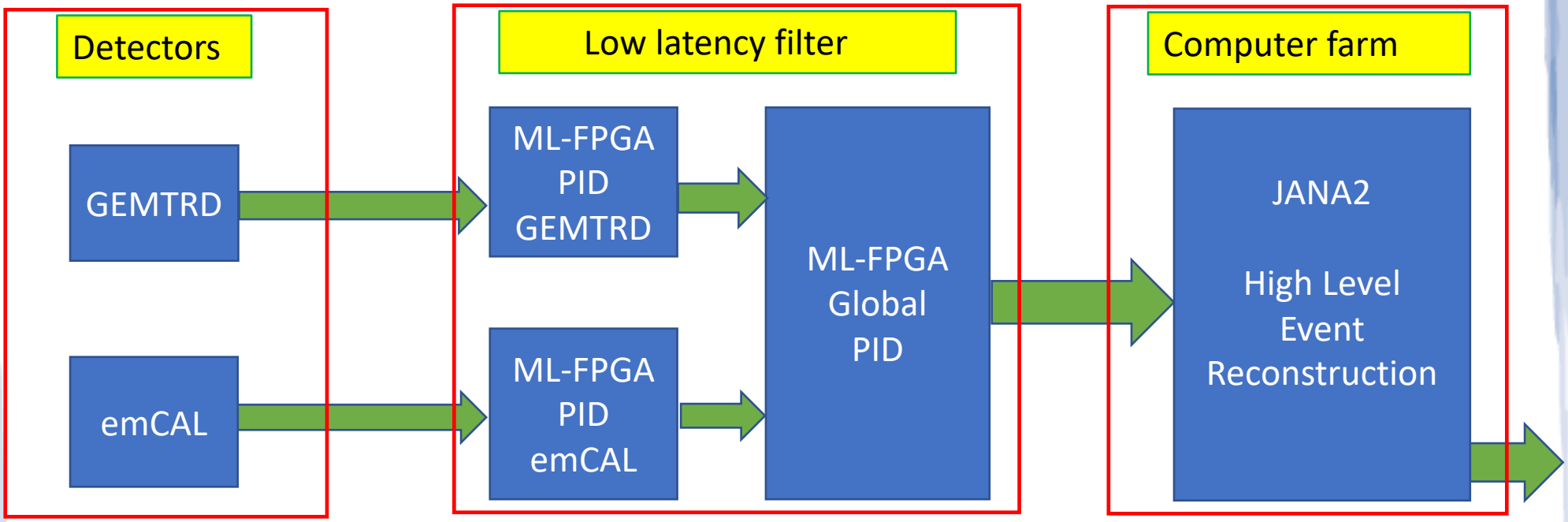


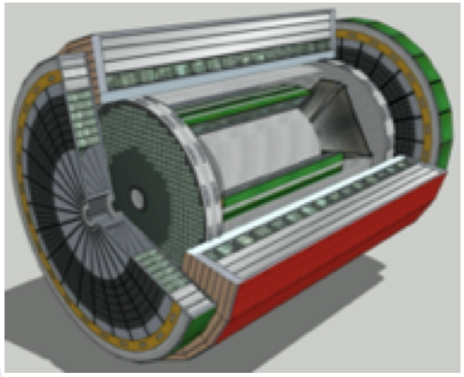


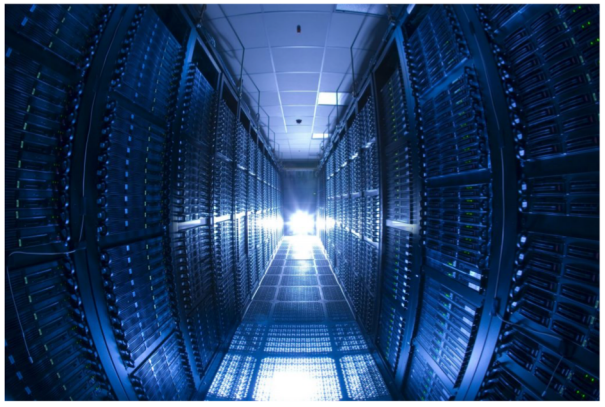


Combined PID / Filter

Jefferson Lab
Thomas Jefferson National Accelerator Facility







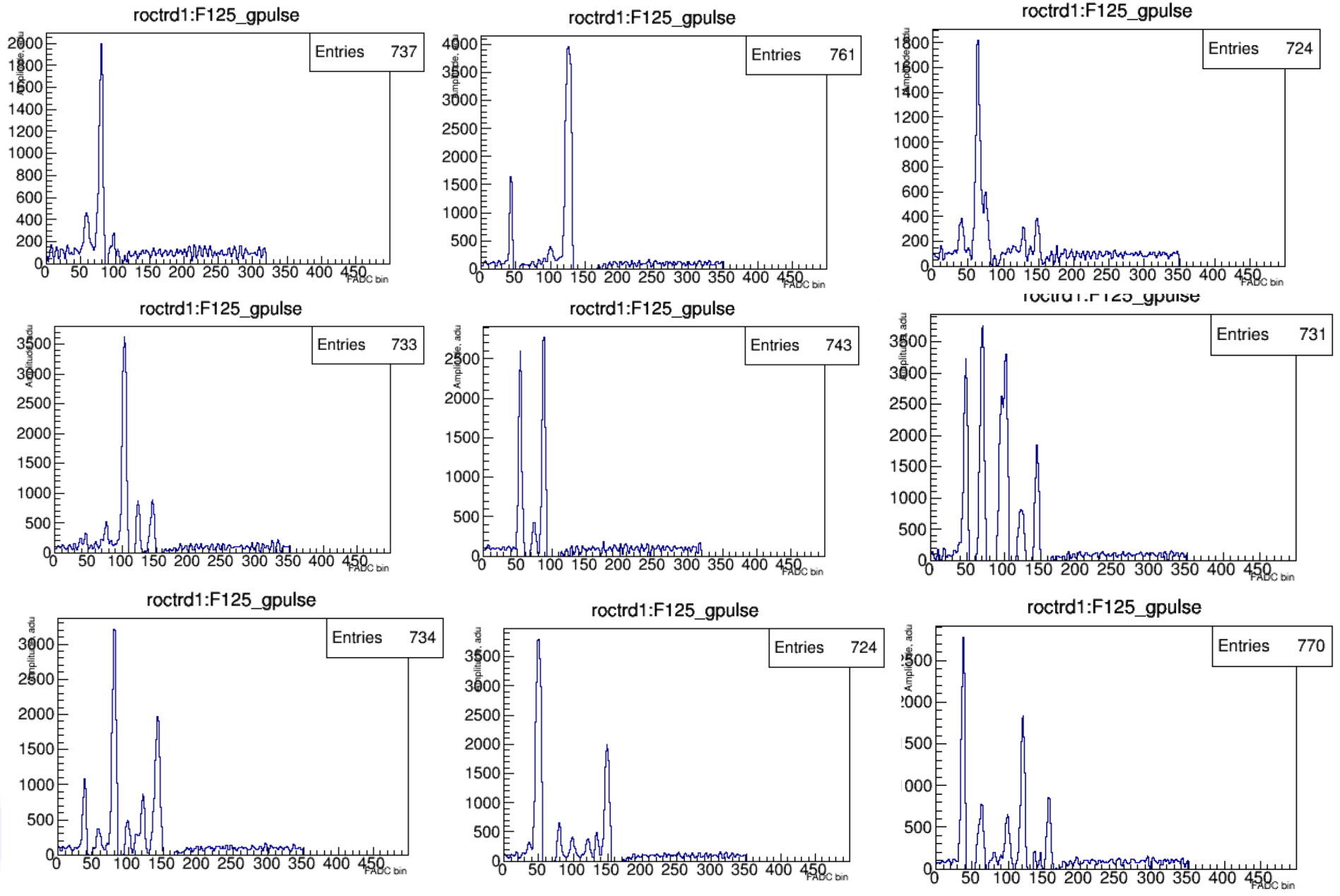
Images from the Internet are for illustration of scale only.

Outlook

- *GEMTRD readout based on Fadc125 shows good results.*
- *Work is in progress on a streaming version of Fadc125.*
- *Preamplifiers perform well in cluster separation.*
- *However, the $ 50 per channel seems too high, so we are looking for other solutions as well.*
- *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.*
- *Open-source hls4ml software tool with Xilinx® Vivado® High Level Synthesis (HLS) accelerates machine learning neural network algorithm development.*

Backup

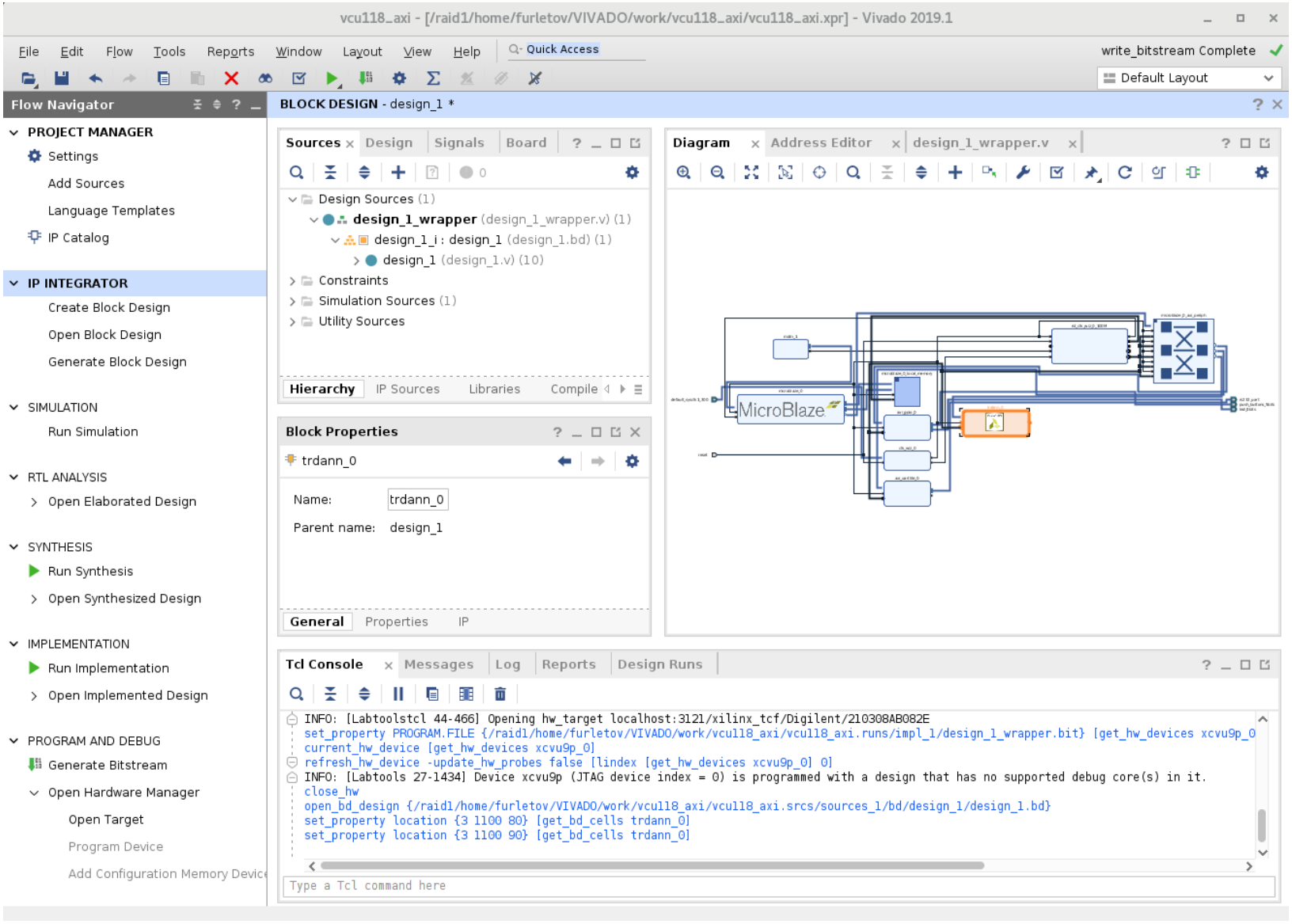
GEMTRD signals with Fadc125



Xilinx Vivado

Jefferson Lab
Thomas Jefferson National Accelerator Facility

Vivado Design Suite is a software suite produced by Xilinx for synthesis and analysis of HDL designs.



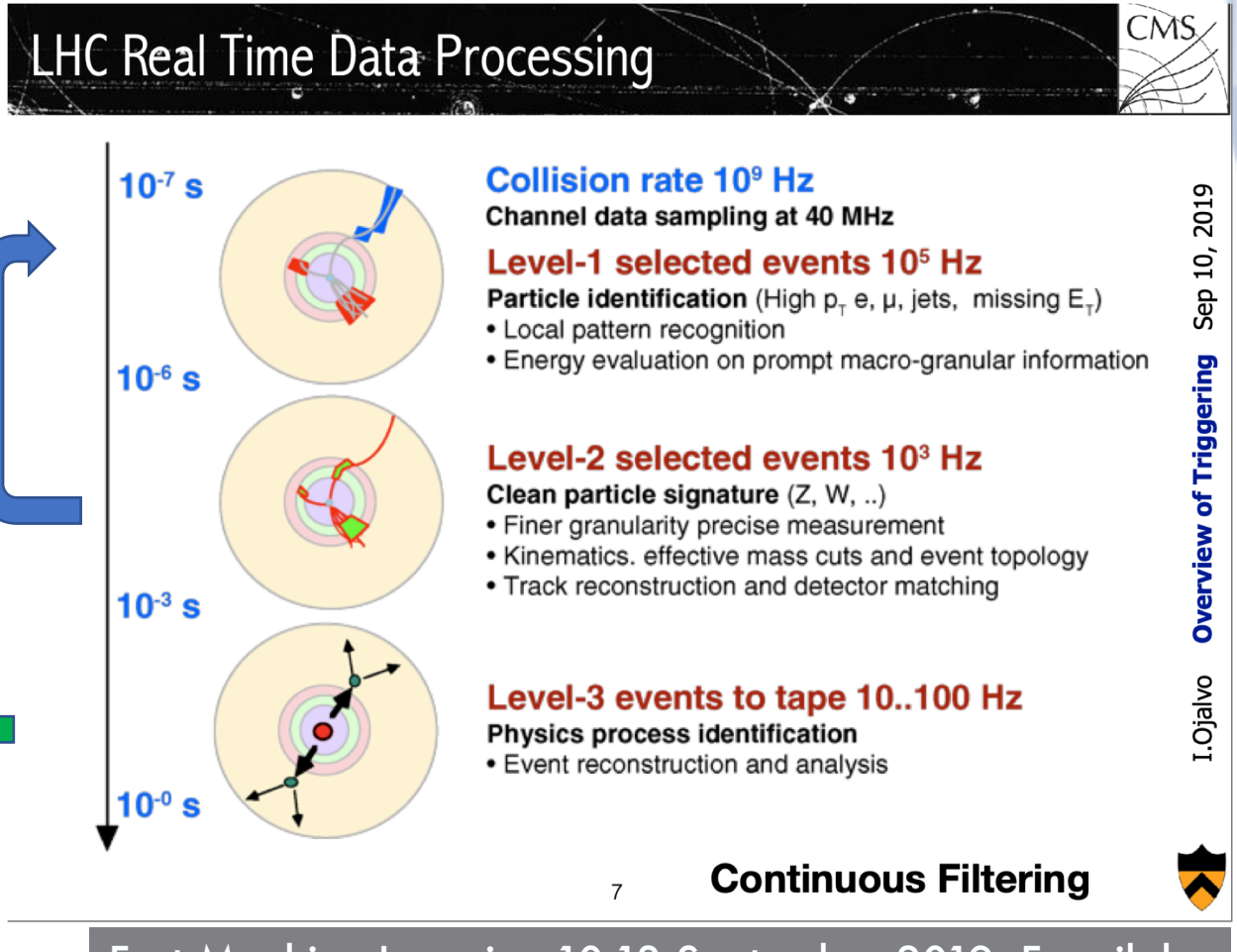
Motivation ML on FPGA

Jefferson Lab
Thomas Jefferson National Accelerator Facility

- *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



Fast Machine Learning,10-13 September 2019, Fermilab

GEMTRD in front of DIRC