APEIRON: a Framework for High Level Programming of Dataflow Applications on Multi-FPGA Systems

Cristian Rossi

(INFN Roma, APE Lab)

for the APEIRON team

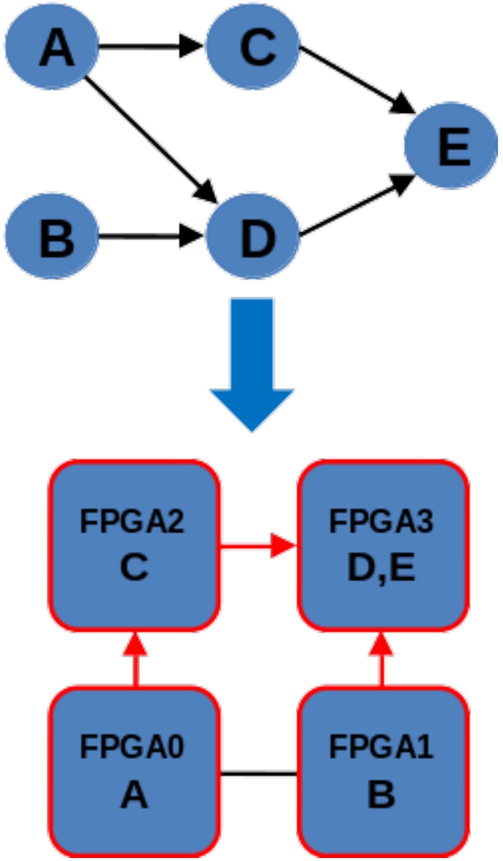
Workshop sul Calcolo nell'I.N.F.N.
Loano (Savona) 22 - 26 maggio 2023

This work is supported by the TEXTAROSSA project (G.A. n. 956831) as part of the EuroHPC-JU initiative, and by INFN National Scientific Committee 5.

APEIRON: an overview

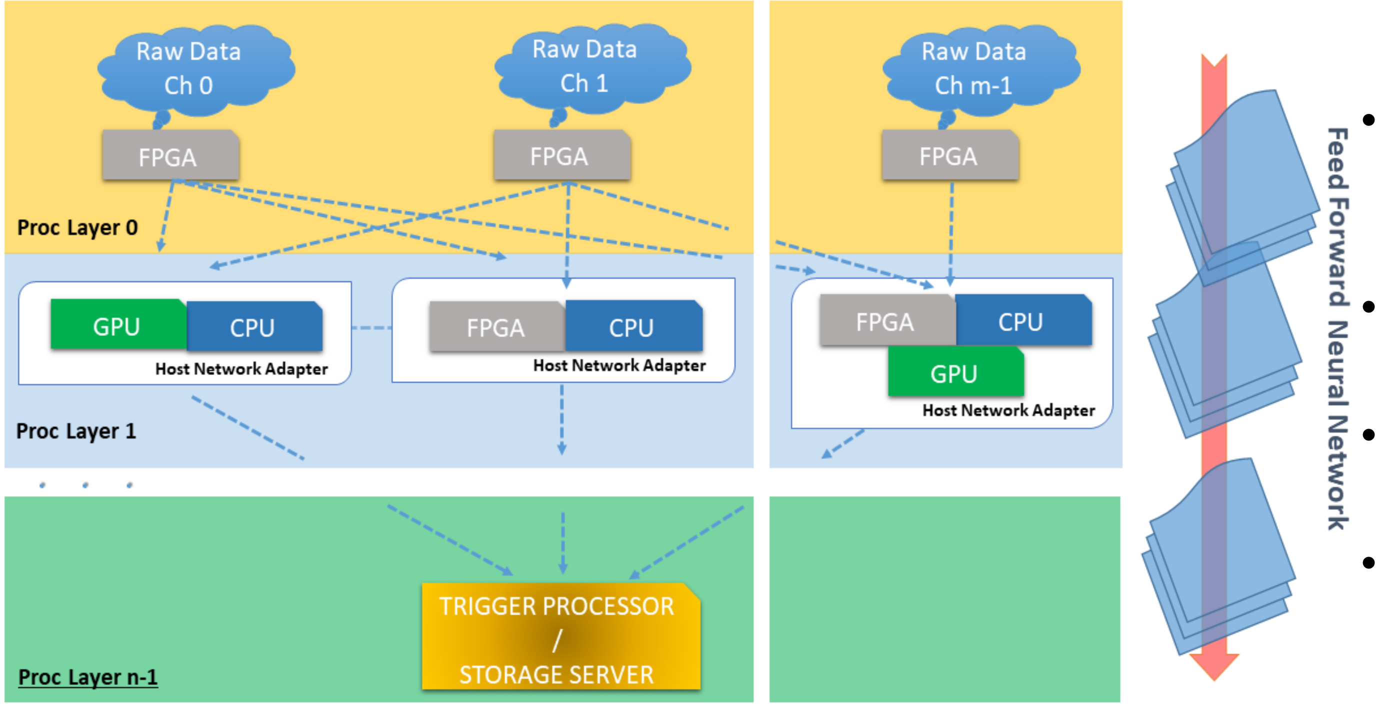
APEIRON main goal is to develop a framework offering hardware and software support for the execution of real-time dataflow applications on a system composed by interconnected FPGAs

- Enabling the mapping the dataflow graph of the application on the distributed FPGA system and offering runtime support for the execution.
- Allowing users, with no (or little) experience in hardware design tools, to develop their applications on such distributed FPGA-based platforms.
  - Tasks are implemented in C++ using High Level Synthesis tools (Xilinx® Vitis).
  - Lightweight C++ communication API (HAPECOM)
    - Non-blocking *send()*
    - Blocking *receive()*
- **APEIRON is based on Xilinx® Vitis High Level Synthesis framework and on INFN Communication IP**



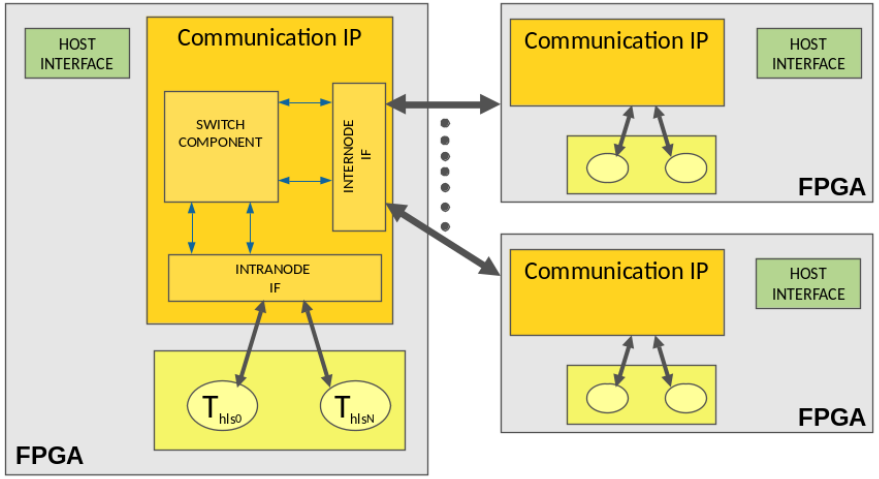
APEIRON in Trigger and Data Acquistion Systems

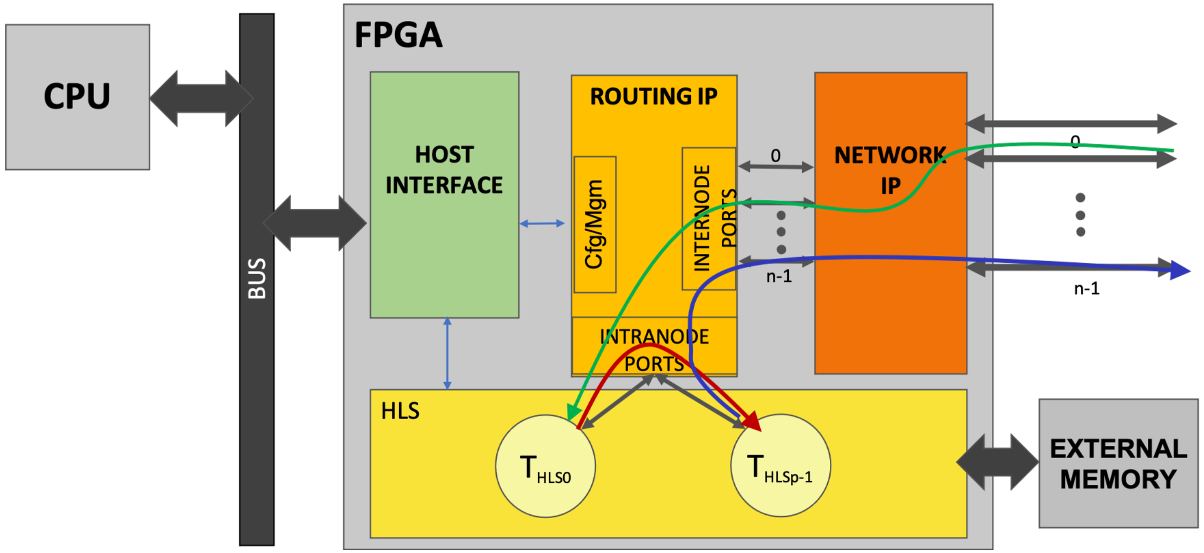
Abstract Processing Environment for Intelligent Read-Out systems based on Neural networks



- Input data from several different channels (data sources, detectors/sub-detectors).
- **Data streams** from different channels recombined through the processing layers using **a low-latency, modular and scalable network infrastructure**
- Distributed online processing on heterogeneous computing devices (FPGAs for the moment) in *n* subsequent layers.
- Typically features extraction will occur in the first NN layers on RO FPGAs.
- More resource-demanding NN layers can be implemented in subsequent processing layers.
- Classification produced by the NN in last processing layer (e.g. pid) will be input for the **trigger processor/storage online data reduction stage for triggerless systems.**

INFN Communication IP

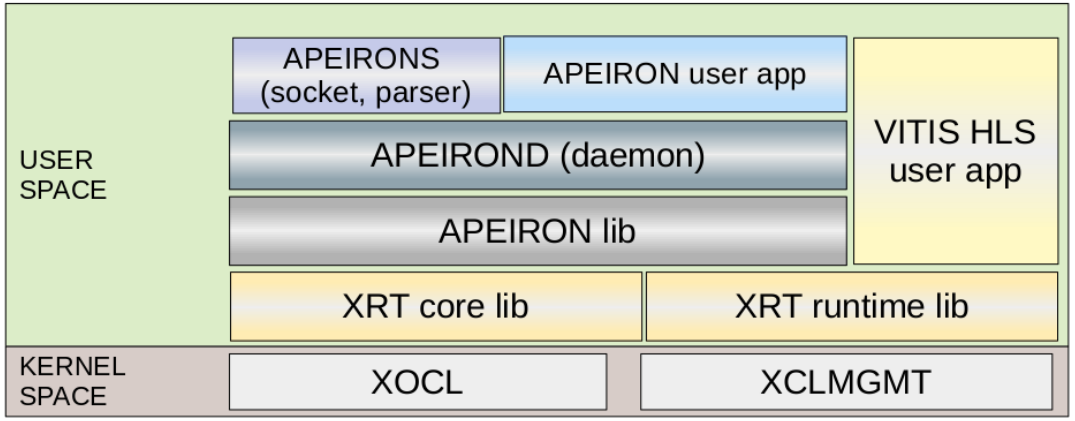


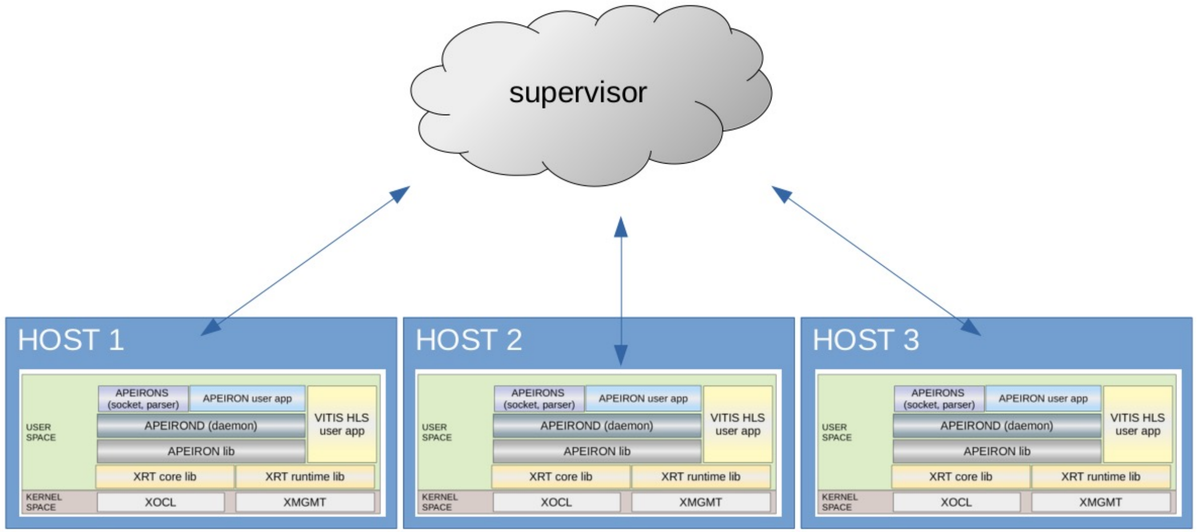


INFN is developing the **IPs** implementing a direct network that allows **low-latency data transfer** between processing tasks deployed on the same FPGA (**intra-node communication**)
and on different FPGAs (**inter-node communication**)

- **Host Interface IP**: Interface the FPGA logic with the host through the system bus.
  - Xilinx® XDMA PCIe Gen3
- **Routing IP**: Routing of intra-node and inter-node messages between processing tasks on FPGA.
- **Network IP**: Network channels and Application-dependent I/O
  - APElink 20 Gbps → 40 Gbps
  - UDP/IP over 1/10 GbE → 25/40/100 GbE
- **HLS Kernels**: user defined processing tasks

APEIRON Runtime Software Stack



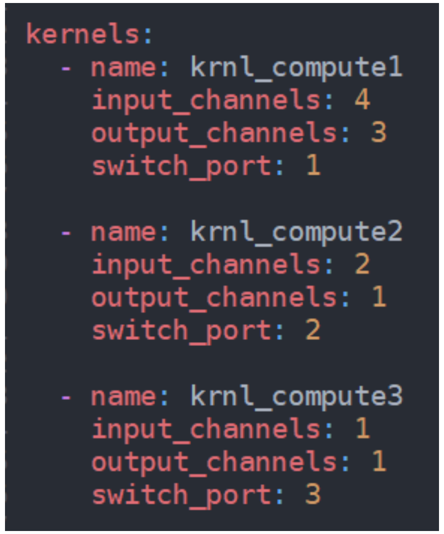


- The APEIRON runtime software stack is built on top of the Xilinx® XRT one adding three layers to:
  - ➢ add the functionalities required to manage multiple FPGA execution platforms (e.g., program the devices, configure the IPs, start/stop execution, monitor the status of IPs, ...);
  - ➢ reduce the impact of changes in XRT API introduced with any new version of Vitis on the APEIRON host-side applications;
  - ➢ decouple the APEIRON software stack from the specific platform, easing the future porting of the framework to different platforms/vendors, ideally by extending the APEIRON library layer only.
- **Apeirond** is a persistent daemon used to manage multiple access request from user apps to the board. It uses the APEIRON lib exposed functions to operate on the devices.
- Using the network socket exposed by each apeirond module, the supervisor can write commands and read answer / status of the different instances of the APEIRON framework running in each node, allowing the end user to have a complete overview of the multiple FPGA execution platform.

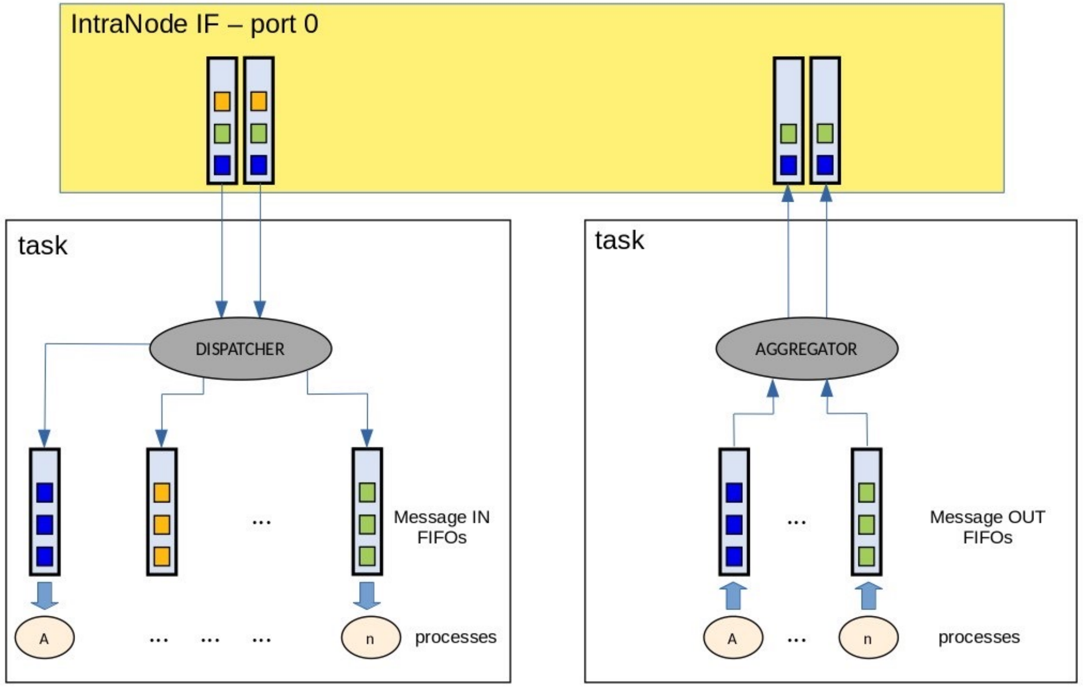
APEIRON: Workflow for FPGA bitstream generation

- The HLS task must have a generic interface, implementation is free.
- A YAML configuration file is used to describe the kernels interconnection topology, specifying how many input/output channels they have

```
void example_task(
[list of optional kernel specific parameters],
message_stream_t message_data_in[N_INPUT_CHANNELS],
message_stream_t message_data_out[N_OUTPUT_CHANNELS])
{…}
```

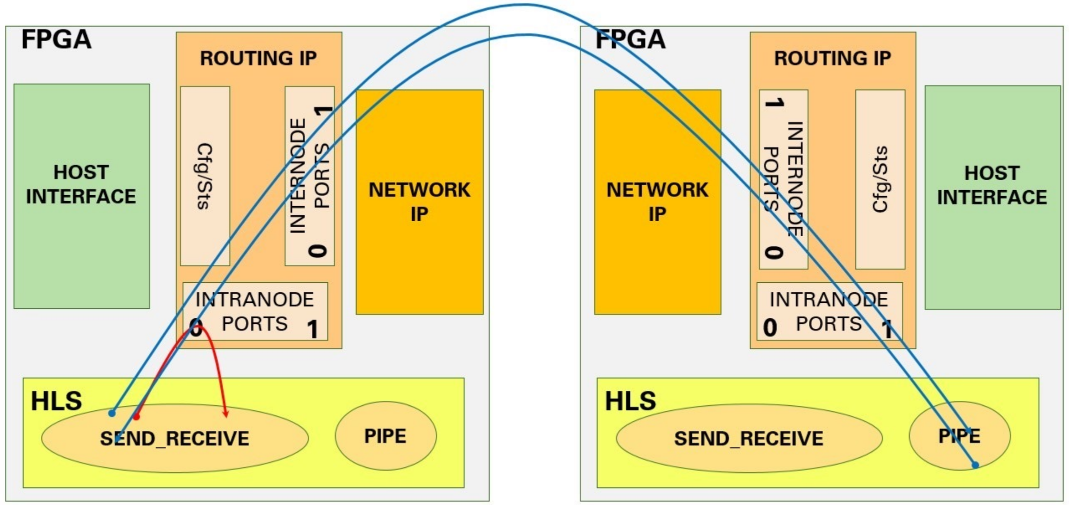


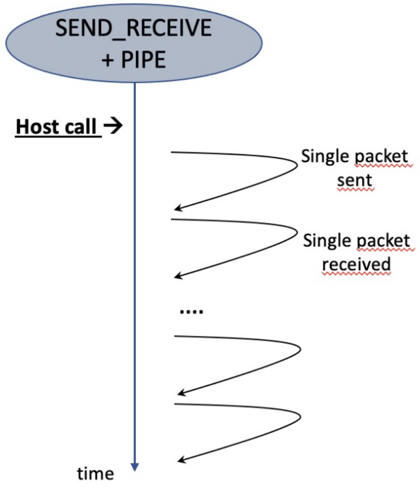
```yaml
kernels:
  - name: krnl_compute1
    input_channels: 4
    output_channels: 3
    switch_port: 1

  - name: krnl_compute2
    input_channels: 2
    output_channels: 1
    switch_port: 2

  - name: krnl_compute3
    input_channels: 1
    output_channels: 1
    switch_port: 3
```

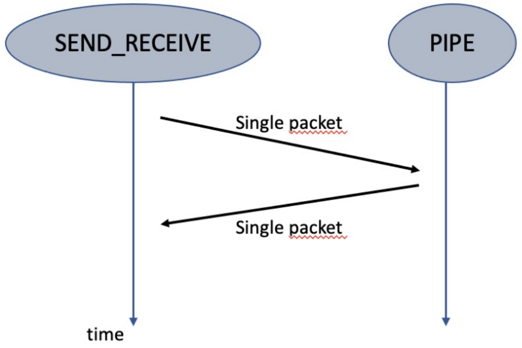


- Adaptation toward/from IntraNode ports of the Routing IP is done by the automatically generated Aggregator and Dispatcher kernel templates.

Communication Latency Test





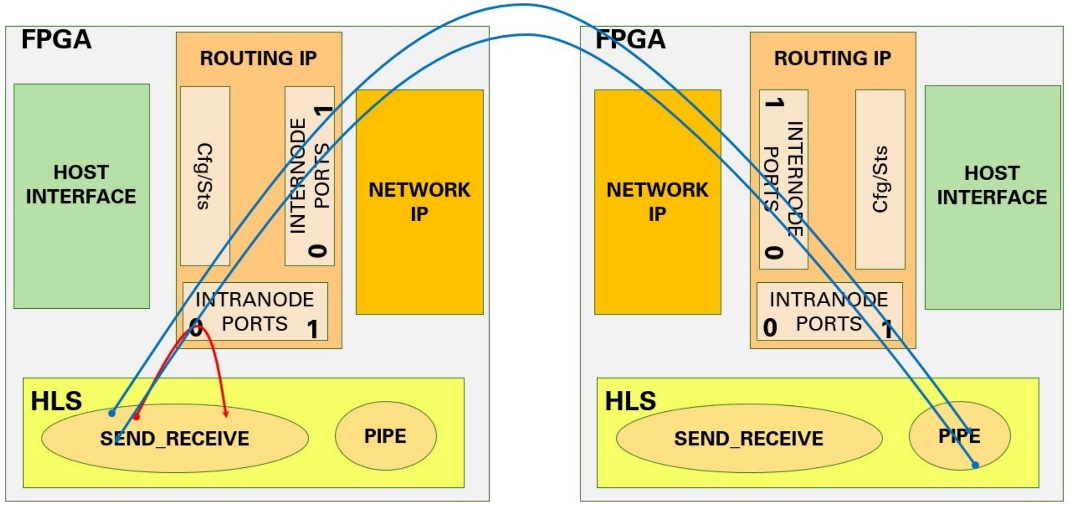


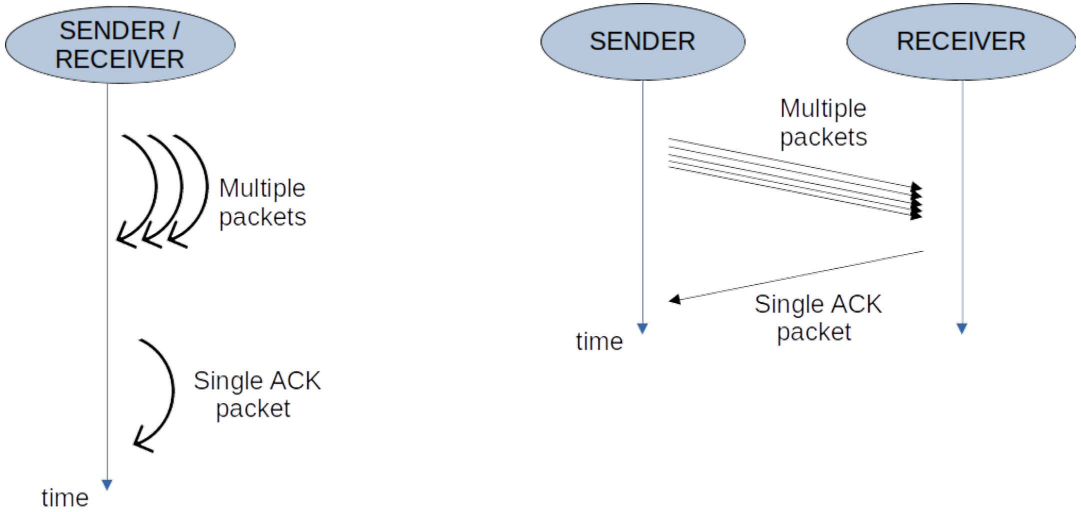
Latency test is performed using multi-task HLS kernel (krnl_sr), configurable by the host in different modes:

- **"send_receive"** mode: kernel reads a payload data item from the FPGA memory (either BRAM or DDR) and sends and receives it through/from the Communication IP to/from a second interconnected FPGA
- **"pipe"** mode has the task of receiving a single packet and bouncing it back to the initiator FPGA

Since the HLS kernel on the initiator FPGA is started via host code while the HLS kernel in "pipe" mode is free-running, the former is launched with a repetition parameter of 1 million send/receive operations before termination in order to **minimize the contribution of the host call overhead on the overall time elapsed** from the start of the first packet send to the completion of the last packet receive (measured on the host).

Communication Bandwidth Test



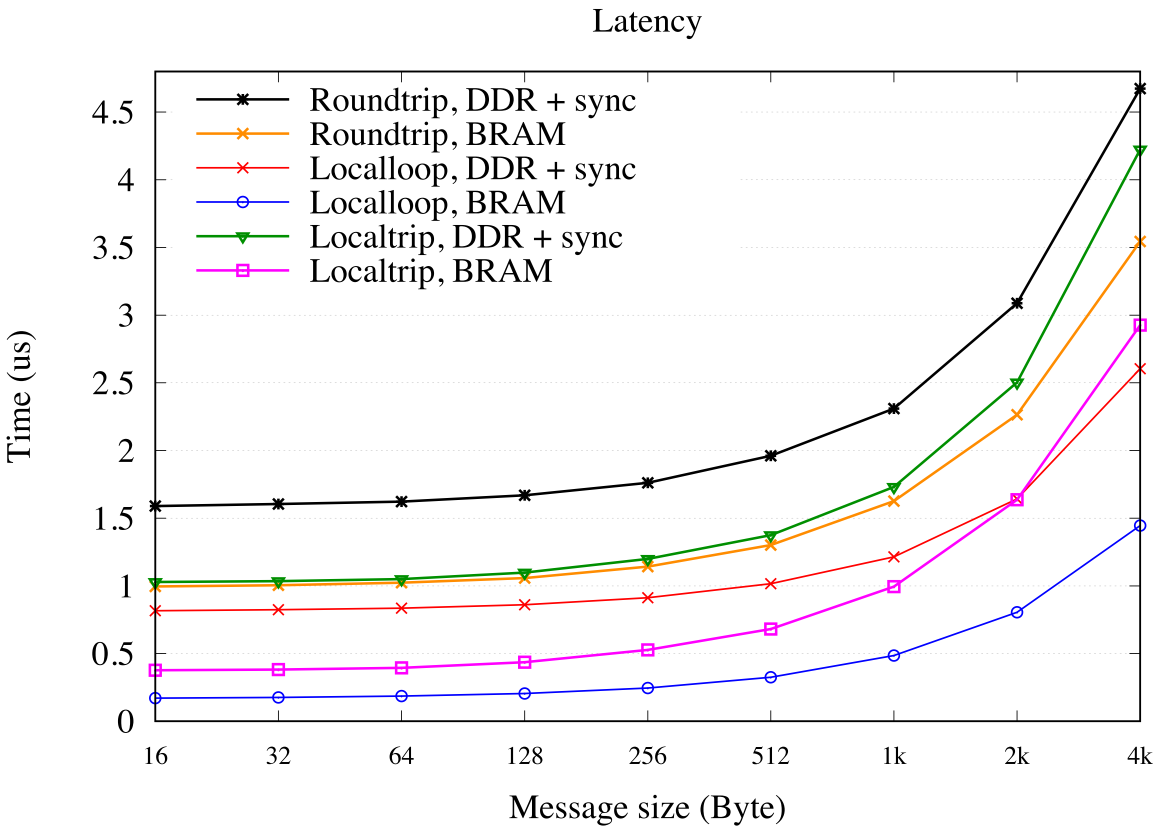


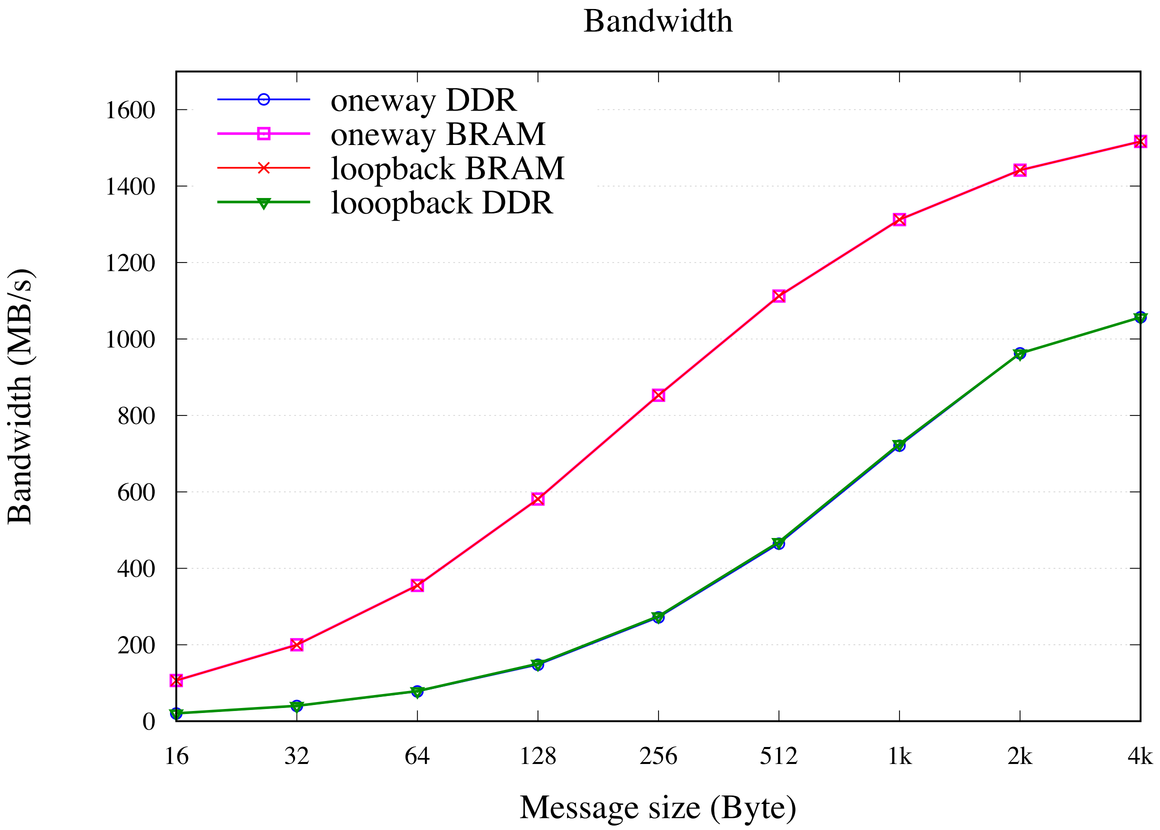
Bandwidth test is carried out by transferring multiple data packets with fixed payload size from:

- a **“sender”** HLS kernel which reads data from the source buffer in FPGA memory (either DDR or BRAM) and pushes them through the Communication IP to another FPGA
- a **“receiver”** HLS kernel: writes data into the destination buffer in memory.

After receiving the number of data packets whose integrated payload adds up to the size of the receive buffer, the second FPGA pings back a single “ACK” packet with minimal payload to confirm the reception.

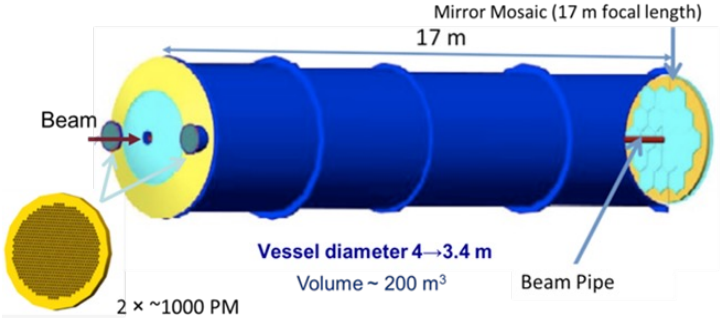
APEIRON Latency and Bandwidth

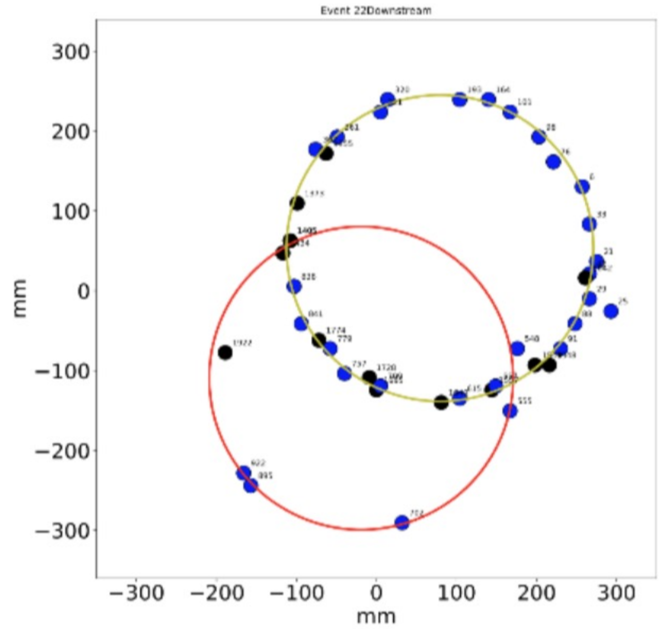




PID in NA62 RICH using NN on FPGA at L0 Trigger

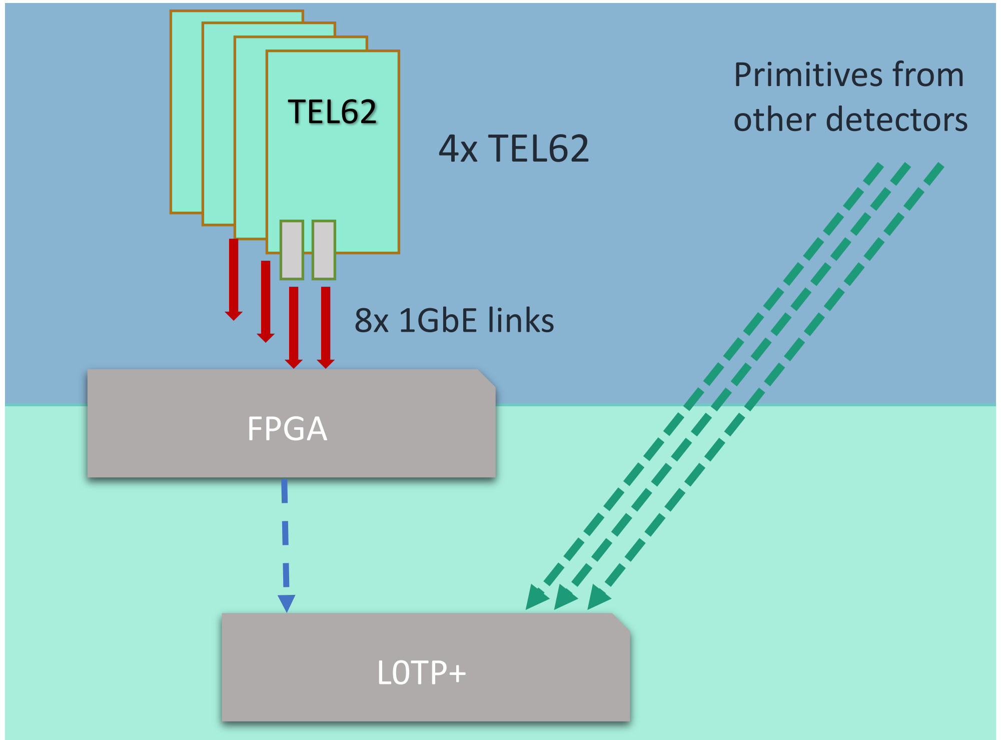
- **Goal**: for any event detected by the RICH provide an estimate for the **number charged particles** and the **number of electrons**
- Streaming readout processing on FPGA using Neural Networks for classification (10 MHz).
- Produce a new primitives stream for Level 0 Trigger Processor
- **The main challenge is the processing throughput**





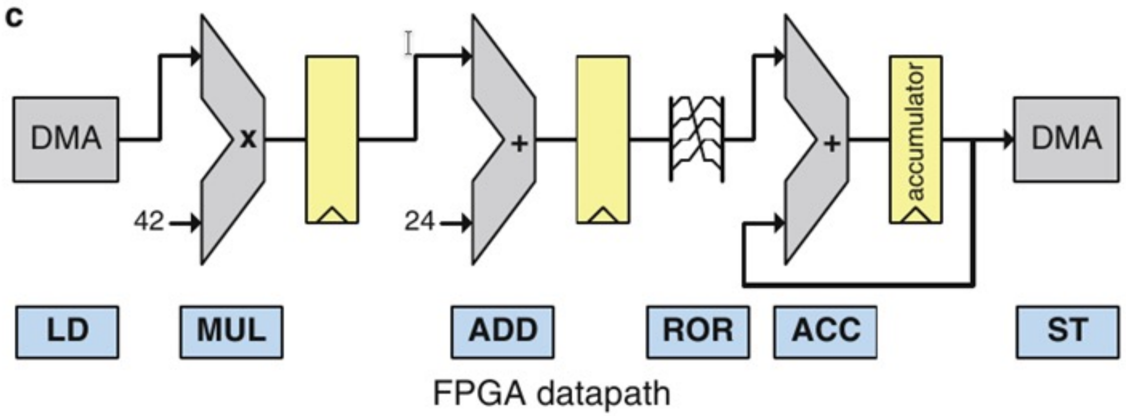
2048 Readout channels
➔ INPUT

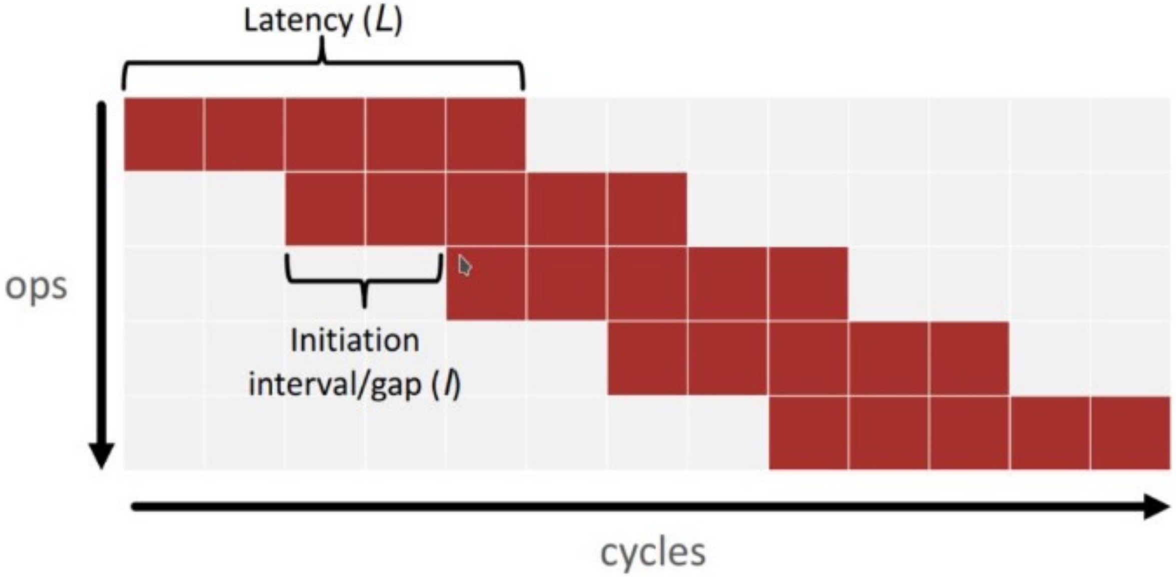
Can we produce rings information online for the L0TP+ level 0 trigger?



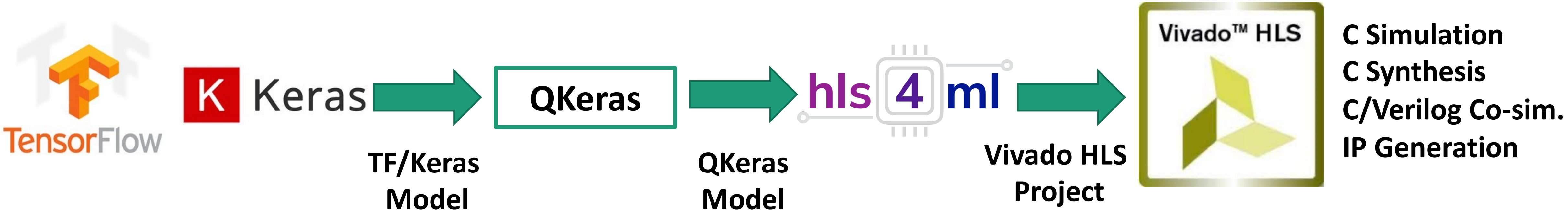
Why FPGA are good for real-time inference?

- Customizable I/O and deterministic latency make them well suited for TDAQ systems.
- Improvements to silicon manufacturing process made them very interesting for heavy computation as well.
- In our case, the challenge is **the processing throughput ➔** a pipelined design can potentially produce a new output at each clock cycle.
- ***Initiation interval (II)****:* Number of clock cycles before the function can accept new input data. **The lower the II, the higher the throughput**
- The greater the number of pipeline stages, the greater the latency.
- High level synthesis tools allows to describe datapaths in FPGA using high level software languages (C/C++, OpenCL, SYCL,...).



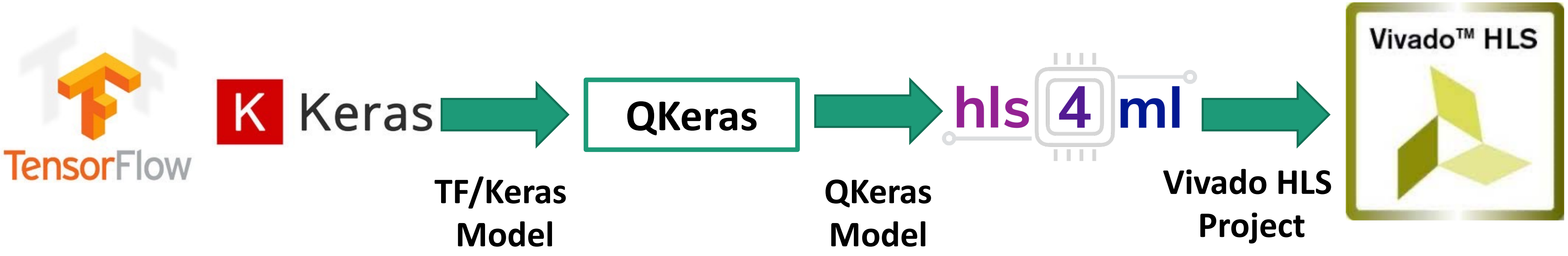


How? → Design and Implementation Workflow



Design targets (efficiency, purity, throughput, latency) and **hardware constraints** (mainly FPGA resource usage) must be taken into account and verified at any stage:

How? ➔ Design and Implementation Workflow



QKeras

TF/Keras Model

QKeras Model

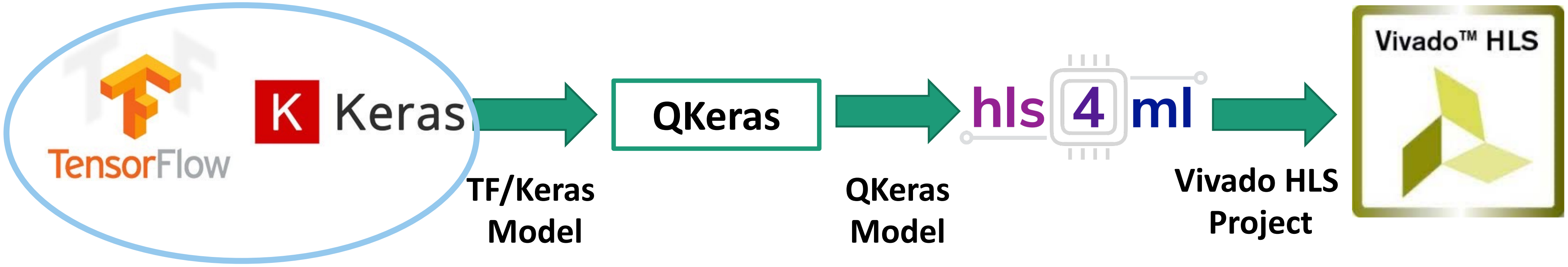
Vivado HLS Project

C Simulation
C Synthesis
C/Verilog Co-sim.
IP Generation

Design targets (efficiency, purity, throughput, latency) and **hardware constraints** (mainly FPGA resource usage) must be taken into account and verified at any stage:

- **Generation strategy of training and validation data sets**.

How? ➔ Design and Implementation Workflow



QKeras

TF/Keras Model

QKeras Model

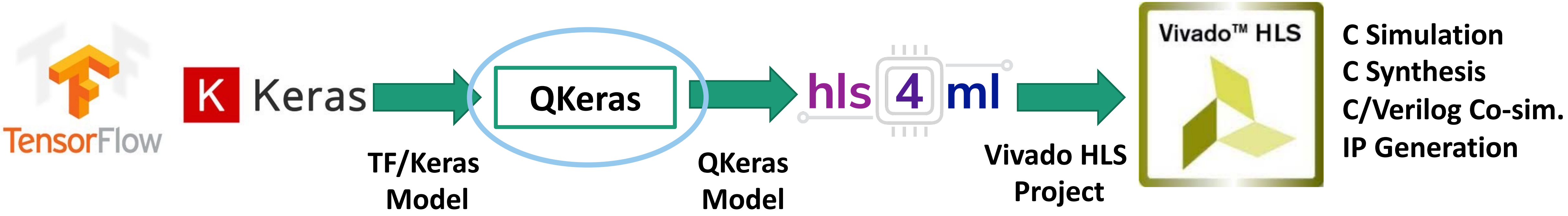
Vivado HLS Project

C Simulation
C Synthesis
C/Verilog Co-sim.
IP Generation

Design targets (efficiency, purity, throughput, latency) and **hardware constraints** (mainly FPGA resource usage) must be taken into account and verified at any stage:

- **TensorFlow/Keras**
  ➔ NN architecture (number and kind of layers) and **representation of the input**
  ➔Training strategy (class balancing, batch sizes, optimizer choice, learning rate,...).

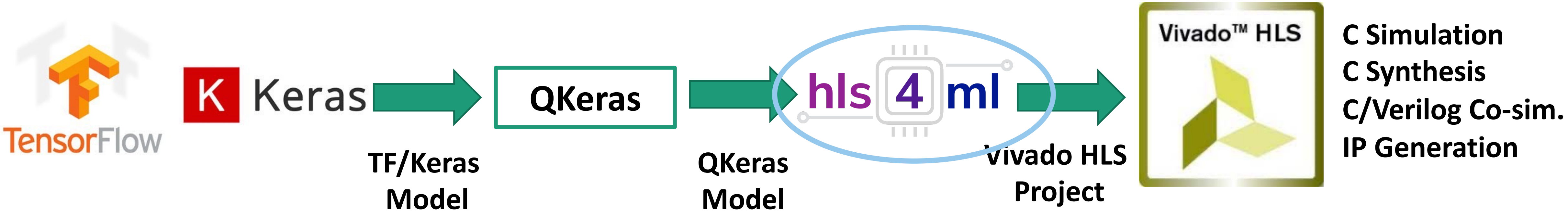
How? ➔ Design and Implementation Workflow



Design targets (efficiency, purity, throughput, latency) and **hardware constraints** (mainly FPGA resource usage) must be taken into account and verified at any stage:

- **Qkeras ➔** Search iteratively the minimal representation size in bits of weights, biases and activations.

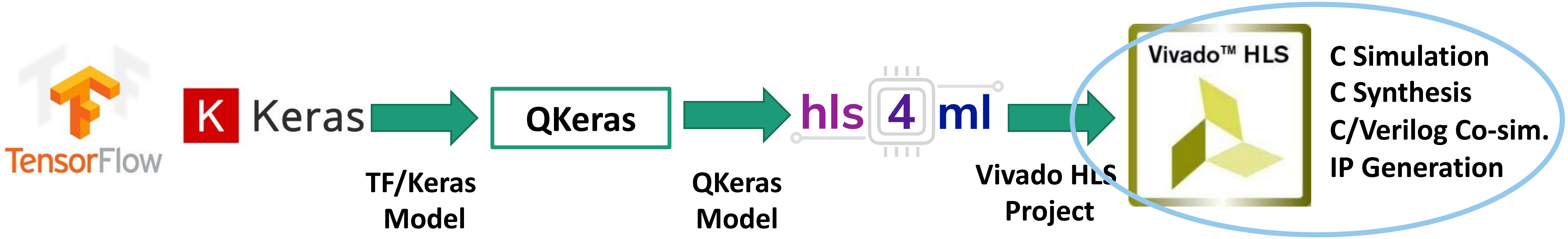
How? ➔ Design and Implementation Workflow



Design targets (efficiency, purity, throughput, latency) and **hardware constraints** (mainly FPGA resource usage) must be taken into account and verified at any stage:

- **hls4ml ➔** Tuning of REUSE FACTOR config param (low values -> low latency, high throughput, high resource usage), clock frequency.

How? ➔ Design and Implementation Workflow

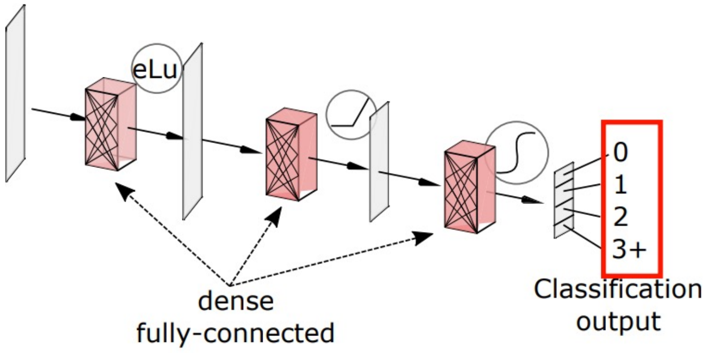


Design targets (efficiency, purity, throughput, latency) and **hardware constraints** (mainly FPGA resource usage) must be taken into account and verified at any stage:

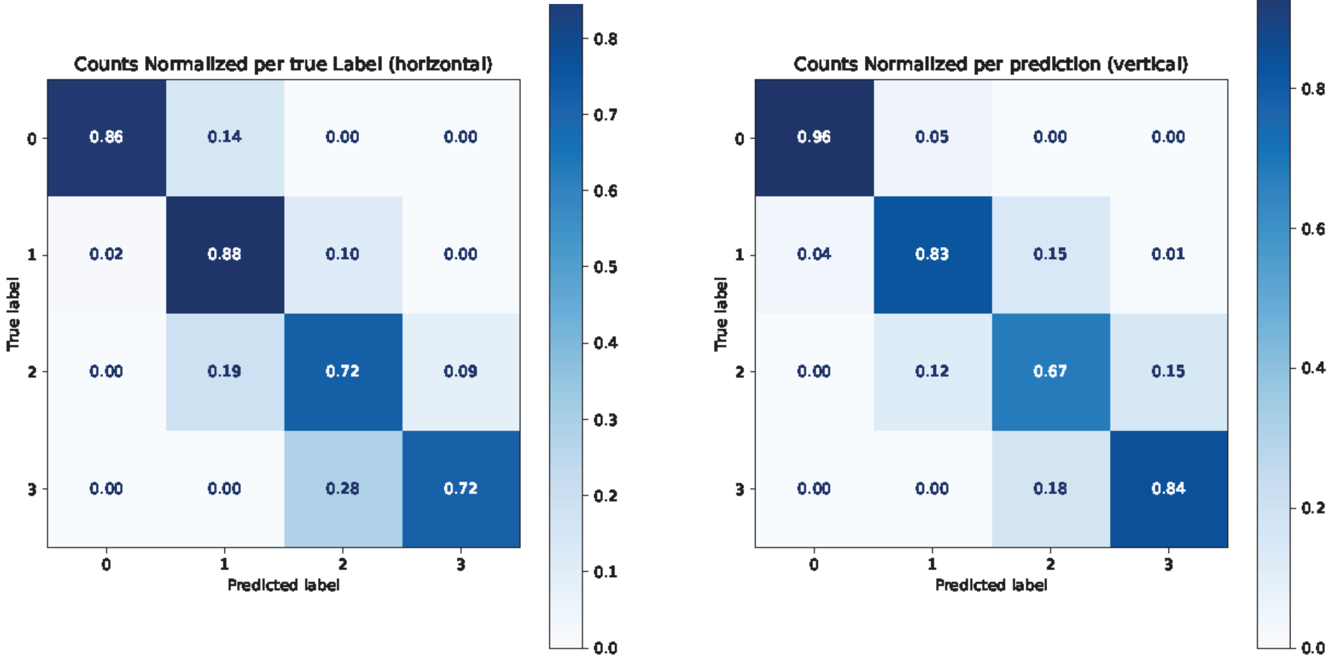
- **Vivado HLS ➔** co-simulation for verification of performance (experimented very good agreement with QKeras Model)

NN Architectures: Dense Model

- Input representation: normalized hitlist (max 64 hits per event)
- Output: 4 classes (0, 1, 2, 3+ rings)
- Quantization (fixed point)
  - Weights and biases: 8 bits <8, 1>
  - Activations: 16 bits <16, 6>
- FPGA resource usage (VCU118)
  LUT 14%, DSP 2%, BRAM 0%
- Latency: 22 cycles @ 150MHz
- Initiation Interval (II): 8 cycles
- Throughput: 18.75 MHz

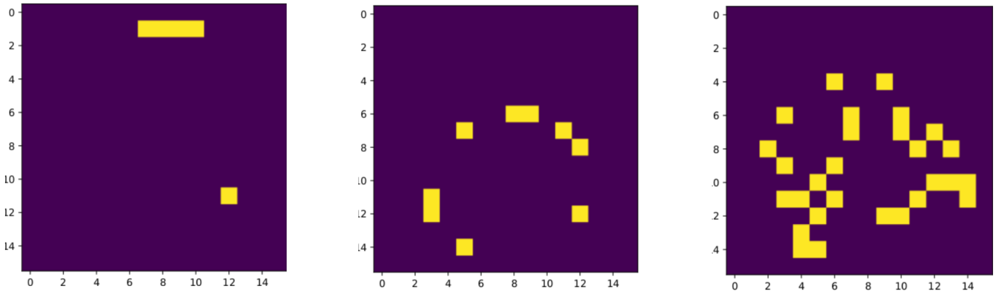


Class 0 (0 rings) Efficiency 85.7 Purity 95.6
Class 1 (1 rings) Efficiency 87.7 Purity 82.9
Class 2 (2 rings) Efficiency 72.3 Purity 67.4
Class 3 (3+ rings) Efficiency 71.9 Purity 84.3

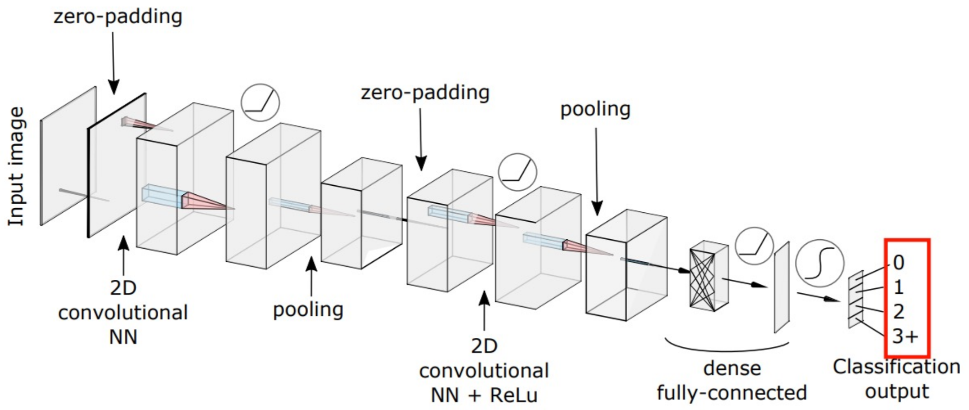


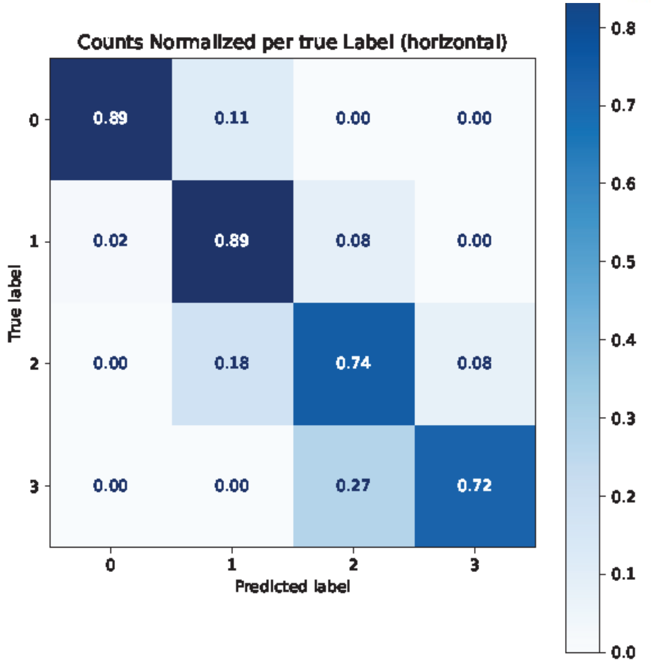
NN Architectures: Convolutional Model

- **Input representation: 16x16 images**



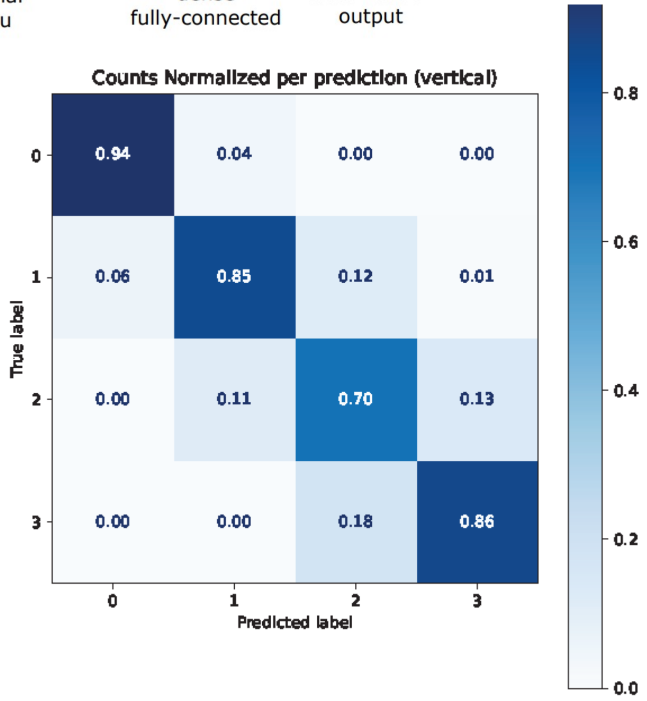
- **Output: 4 classes (0, 1, 2, 3+ rings)**
- **Quantization (fixed point):**
  - Weights and biases: 8 bits <8, 1>
  - Activations:16 bits <16, 6>
- **FPGA resource usage (Alveo U200)**
  - LUT 5.2%, FF 1.5%, DSP 4.8%,
  - BRAM 0.05%
- **Latency: 388 cycles @ 220MHz**
- **Initiation Interval (II): 369 cycles**
- **Throughput: 0.6 MHz**





Counts Normalized per true Label (horizontal)

True label \ Predicted label	0	1	2	3
0	0.89	0.11	0.00	0.00
1	0.02	0.89	0.08	0.00
2	0.00	0.18	0.74	0.08
3	0.00	0.00	0.27	0.72

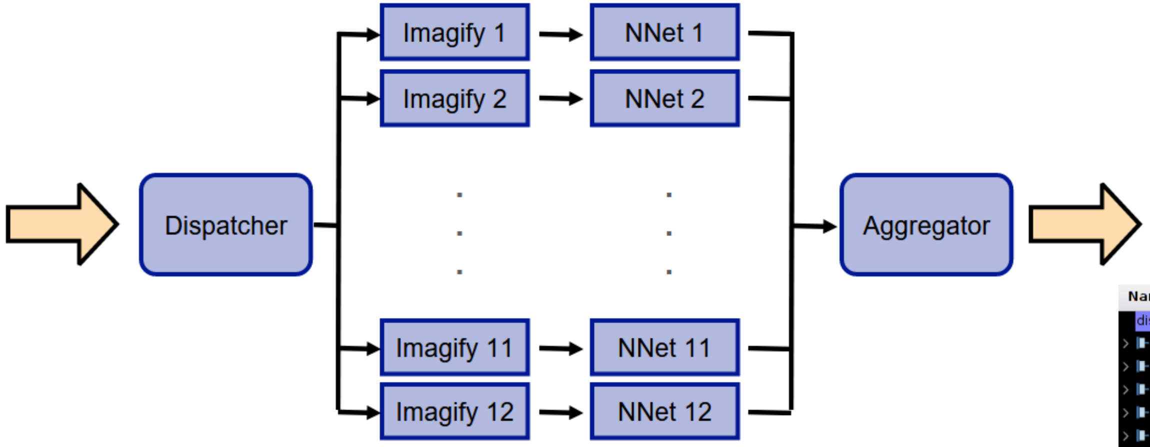


Counts Normalized per prediction (vertical)

True label \ Predicted label	0	1	2	3
0	0.94	0.04	0.00	0.00
1	0.06	0.85	0.12	0.01
2	0.00	0.11	0.70	0.13
3	0.00	0.00	0.18	0.86

Convolutional model issue ➔ Kernel replication

Throughput is not enough to sustain L0 rate, but we can replicate the network multiple times, also on multiple devices if necessary (APEIRON).

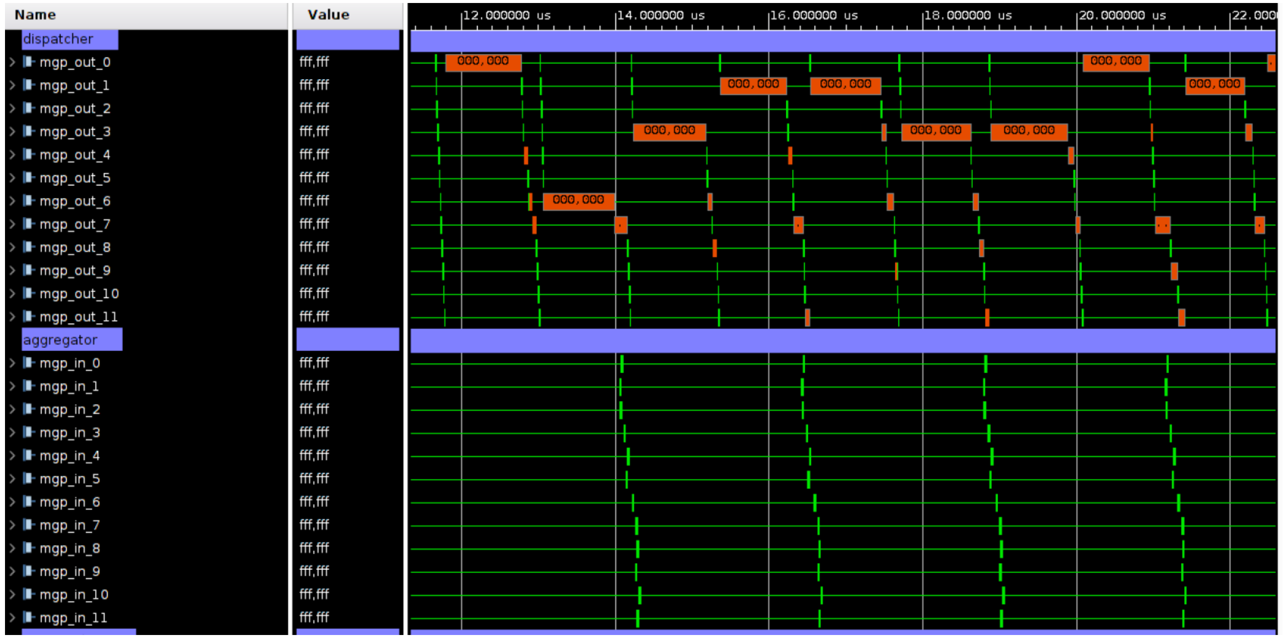


Resources usage for 12 replicas:

- LUT 74%
- FF 17%
- DSP 61%
- BRAM 1.4%

Processing time @220MHz: 137 ns per event

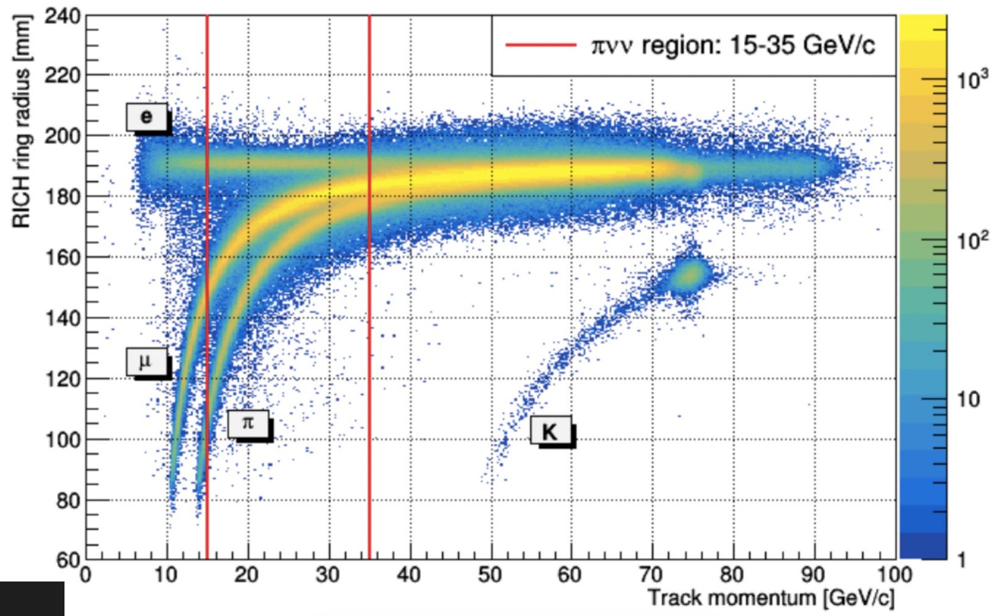
Processing throughput: 7.2 MHz

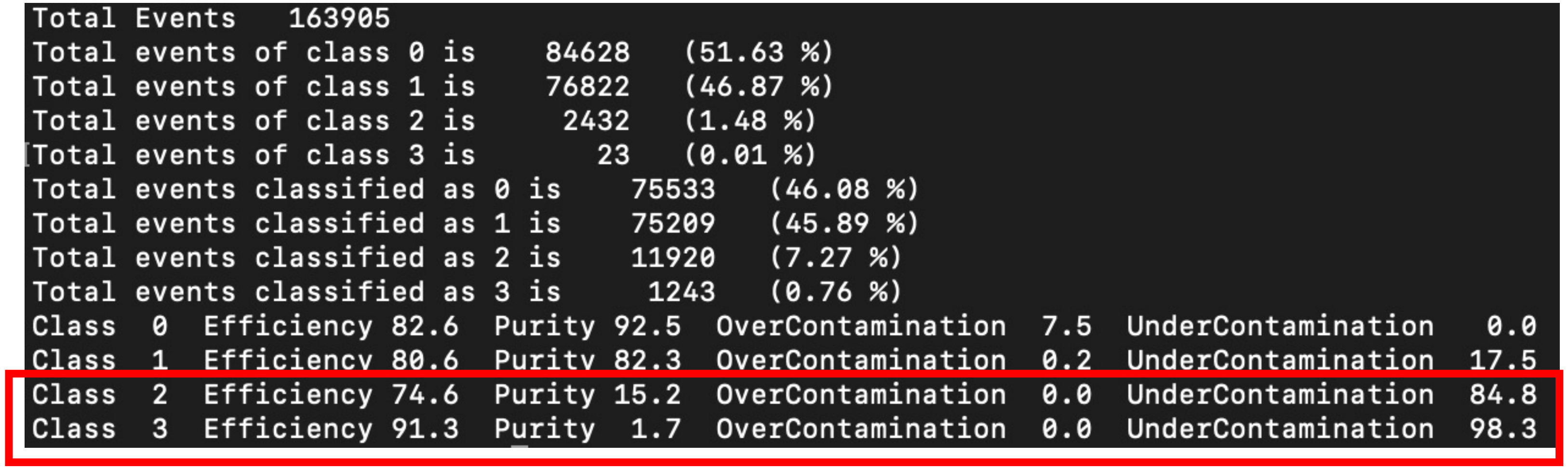


Name	Value
dispatcher	
mgp_out_0	fff,fff
mgp_out_1	fff,fff
mgp_out_2	fff,fff
mgp_out_3	fff,fff
mgp_out_4	fff,fff
mgp_out_5	fff,fff
mgp_out_6	fff,fff
mgp_out_7	fff,fff
mgp_out_8	fff,fff
mgp_out_9	fff,fff
mgp_out_10	fff,fff
mgp_out_11	fff,fff
aggregator	
mgp_in_0	fff,fff
mgp_in_1	fff,fff
mgp_in_2	fff,fff
mgp_in_3	fff,fff
mgp_in_4	fff,fff
mgp_in_5	fff,fff
mgp_in_6	fff,fff
mgp_in_7	fff,fff
mgp_in_8	fff,fff
mgp_in_9	fff,fff
mgp_in_10	fff,fff
mgp_in_11	fff,fff

Results for classification of number of electrons

- Preliminary results for online classification of the **number of "electrons"** show that even the very simple NN architectures that we tested are capable, below 35 GeV/c momentum, of reaching a non-negligible performance (see terminal picture below).
- It can be improved for the online unfiltered event stream using a **dedicated NN receiving in input data from other detectors** (e.g. L0CALO).





```
Total Events    163905
Total events of class 0 is     84628    (51.63 %)
Total events of class 1 is     76822    (46.87 %)
Total events of class 2 is      2432    (1.48 %)
Total events of class 3 is        23    (0.01 %)
Total events classified as 0 is     75533    (46.08 %)
Total events classified as 1 is     75209    (45.89 %)
Total events classified as 2 is     11920    (7.27 %)
Total events classified as 3 is      1243    (0.76 %)
Class  0  Efficiency 82.6  Purity 92.5  OverContamination  7.5  UnderContamination   0.0
Class  1  Efficiency 80.6  Purity 82.3  OverContamination  0.2  UnderContamination  17.5
Class  2  Efficiency 74.6  Purity 15.2  OverContamination  0.0  UnderContamination  84.8
Class  3  Efficiency 91.3  Purity  1.7  OverContamination  0.0  UnderContamination  98.3
```

Conclusions and Future Work

- The APEIRON framework enables the development and deployment of Vitis HLS dataflow applications distributed on multiple-FPGA systems.
- The co-design of its software stack and of the Communication IP allowed to reach very **low and deterministic latency and** a **high fraction of the channel's raw bandwidth** for communications between FPGAs, addressing fundamental bottlenecks for real-time distributed dataflow applications.
- We are working to improve the framework and the Communication IP
  - to **increase the internal datapath of the IP to 256 bits** and to use the transceiver with 4 lanes to support applications requiring an increased communication bandwidth
  - To implement a **new channel interface based on the Xilinx® 10G/25G High Speed Ethernet Subsystem** in order to enable interoperability with standard switched networks, either to support (e.g. UDP over IP) input and output streams or to implement a switched network topology.
- We control the workflow for the implementation of real-time/high throughput classifiers on FPGA using limited resources, this hints for applying the methodology also to:
  - less capable (i.e. front-end) FPGAs
  - complex design making use of a large fraction of FPGA resources (e.g. L0TP+)

The APEIRON Team

@INFN Roma – APE Lab

A. Lonardo

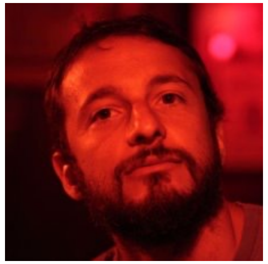
P. Vicini

F. Lo Cicero

F. Simula

M. Martinelli

P. S. Paolucci


R. Ammendola

A. Biagioni

P. Cretaro

O. Frezza

C. Rossi

M. Turisini
(now @CINECA)

@Università
Sapienza Roma

A. Ciardiello

THANK YOU!

BACKUP SLIDES

Datasets' generation

- Dataset for training and validation obtained using the NA62 analysis framework
- Analyser called RingDumperAPE
- Single run or in batch (run list) from CTRL trigger sample
- Output: Histograms + Events dumped on plain text files →

RICH Hit list (TDCEvent)
RICH trackless reconstruction (TRecoRICHEvent)
Downstreamtrack reconstruction (Downstreamtrack)
L0TP (TNA62L0Data)
Event Labels

- Different labels are dumped to be used as ground truth
  1. Number of rings from RichReco
  2. Number of rings from Downstreamtrack
  3. Number of electrons from RichReco (based on ring radius only)
  4. Number of electrons from Downstreamtrack (based on MostLikelyHypothesis)
  5. Number of electrons as 4 + check on the radius + check on Energy over momentum ratio (EOP)

Electron radius = [185,195] mm
Eop = [0.90,1.10]

- Event rejection criteria can be optionally activated
  - Formal check on the reconstructed tracks and rings (e.g. chi2)
  - Event characteristics e.g. NHit, Momentum, etc

Most likely hypothesys = multiple are rejected

Communication IP: APENET header

0	VirtualChannel	32	🟥	64	Destination Virtual Address	96	🟧
1		33		65		97	
2		34		66		98	
3		35		67		99	
4		36		68		100	
5	PID/Ch ID	37	Intratile Port	69		101	
6		38		70		102	
7		39		71		103	
8		40		72		104	
9		41		73		105	
10		42	OutOfLattice pkt	74		106	
11		43	Packet Type	75		107	
12		44		76		108	
13		45		77		109	
14		46		78		110	Num of Hops
15		47		79		111	
16		48	Length	80		112	
17		49		81		113	
18		50		82		114	
19		51		83		115	
20		52		84		116	
21	COORDINATE	53		85		117	
22		54		86		118	
23		55		87		119	
24		56		88		120	ECC_CR
25		57		89		121	
26		58		90		122	
27		59		91		123	
28		60		92		124	
29		61		93		125	
30		62	🟧	94		126	
31		63		95		127	