APEIRON: composing smart TDAQ systems for high energy physics experiments

Roberto Ammendola[2], Andrea Biagioni[1], Carlotta Chiarini[3,1],
Andrea Ciardiello[3,1], Paolo Cretaro[1], Ottorino Frezza[1],
Francesca Lo Cicero[1], Alessandro Lonardo[1], Michele Martinelli[1],
Pier Stanislao Paolucci[1], Cristian Rossi[1], Francesco Simula[1],
Matteo Turisini[1], Piero Vicini[1]

[1] Istituto Nazionale di Fisica Nucleare (INFN), sezione di Roma, Rome, Italy
[2] Istituto Nazionale di Fisica Nucleare (INFN), sezione di Roma Tor Vergata, Rome, Italy
[3] Dipartimento di Fisica, Sapienza Università di Roma, Rome, Italy

E-mail: `alessandro.lonardo@roma1.infn.it`

Abstract. APEIRON is a framework encompassing the general architecture of a distributed heterogeneous processing platform and the corresponding software stack, from the low level device drivers up to the high level programming model. The framework is designed to be efficiently used for studying, prototyping and deploying smart trigger and data acquisition (TDAQ) systems for high energy physics experiments.

1. Introduction

As for the requirements imposed by applications in the class of real-time dataflow processing, FPGA devices are a good fit inasmuch as they can provide not only adequate computing, memory and I/O resources but also a smooth programming experience. High-Level Synthesis tools, after more than a decade since their appearance, are quickly reaching a technological readiness that paves the way to the adoption of these reconfigurable accelerators by a class of users much broader to that composed by skilled developers used to employ Hardware Description Language-based workflows.

The main motivation for the design and development of the APEIRON framework is that the currently available HLS tools do not natively support the deployment of applications over multiple FPGA devices, which severely chokes the scalability of problems that this approach could tackle. To overcome this limitation, we envisioned APEIRON as an extension of the Xilinx Vitis HLS framework able to support a network of FPGA devices interconnected with a low-latency direct network as the reference execution platform.

The general architecture of the APEIRON distributed processing platform includes m data sources, corresponding to the detectors or sub-detectors, feeding a sequence of n stream processing layers, making up the whole data path from readout to trigger processor (or storage server).

The processing platform features a modular and scalable low-latency network infrastructure with configurable topology. This network system represents the key element of the architecture, enabling the low-latency recombination of the data streams arriving from the different input
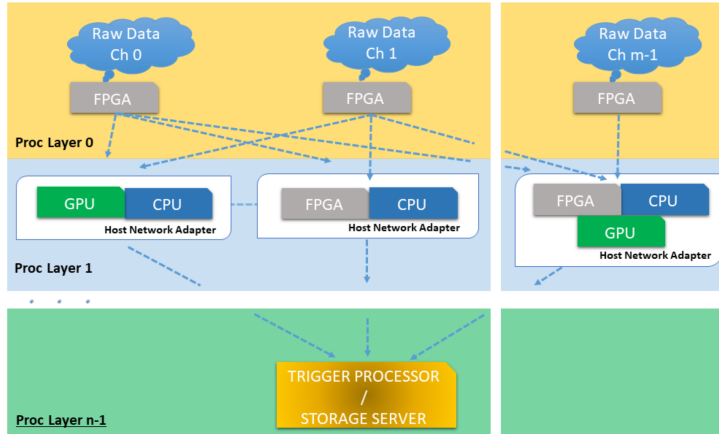
Figure 1. Recombination of data streams through specialized processing layers in an example trigger or data reduction system implemented with the APEIRON framework.

channels through the various processing layers, as shown in Figure 1.

Developers can define scalable applications using a dataflow programming model (inspired by Kahn Process Networks [1]) that can be efficiently deployed on a multi-FPGAs system: the APEIRON communication Intellectual Properties (IPs) allow low-latency communication between processing tasks deployed on FPGAs, even if hosted on different computing nodes.

Thanks to the use of High Level Synthesis (HLS) tools in the workflow, tasks are described in high level language (C/C++) while communication between tasks is expressed through a lightweight API based on non-blocking *send()* and blocking *receive()* operations.

The mapping between the computational data flow graph and the underlying network of FPGAs, such as that shown in Figure 3, is defined by the designer with a configuration tool, by which the framework will produce all project files required for the FPGAs bitstream generation. The interconnection logic is therefore automatically built according to the application needs (in terms of input/output data channels) as shown in Figure 2, allowing the designer to focus on the processing tasks expressed in C/C++.

The aim of the APEIRON project is to develop a flexible framework that could be adopted in the design and implementation of both "traditional" low level trigger systems and of data reduction stages in trigger-less or streaming readout experimental setups characterized by high event rates. For this purpose we studied and implemented algorithms capable of boosting the efficiency of these classes of online systems based on Neural Networks (NNs), trained offline on Tensorflow/Keras and leveraging the HLS4ML [2] and QKeras [3] software packages for deployment on FPGA. We have validated the framework on the physics use case represented by the partial particle identification system for the low-level trigger of the NA62 experiment [4], working on data from its Ring Imaging Cherenkov detector to pick out electrons and number of charged particles.
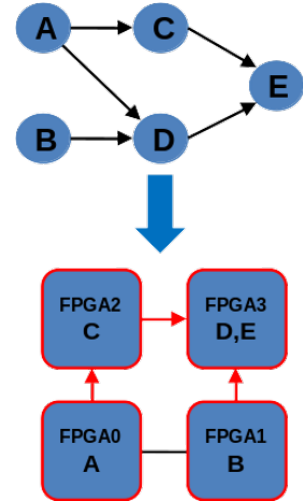
2. The APEIRON Framework

The Communication IP is the evolution of the APEnet [5] and exanet [6] designs for HPC systems and represents the main enabling component for the APEIRON framework, defined as the general architecture of an FPGA-based distributed stream processing platform and the corresponding software stack.

The Communication IP allows data transfers between processing tasks hosted in the same node (intra-node communications) or in different nodes (inter-node communications), see Figure 4.
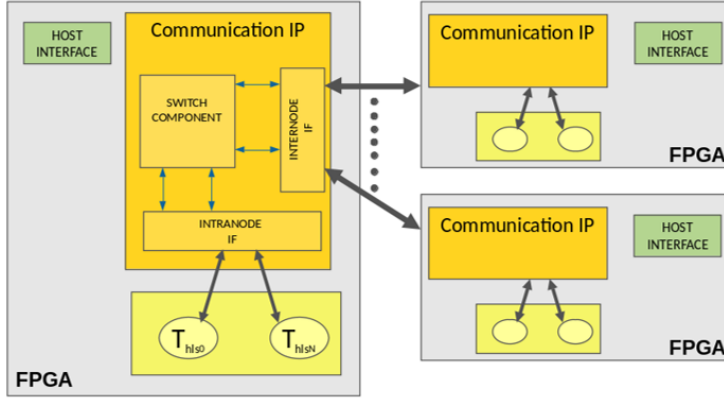
Figure 2. Communication IPs managing data streams I/O and communication between HLS computing tasks (represented as yellow ovals).

**Figure 3.** Deployment of five tasks on four interconnected FPGAs.
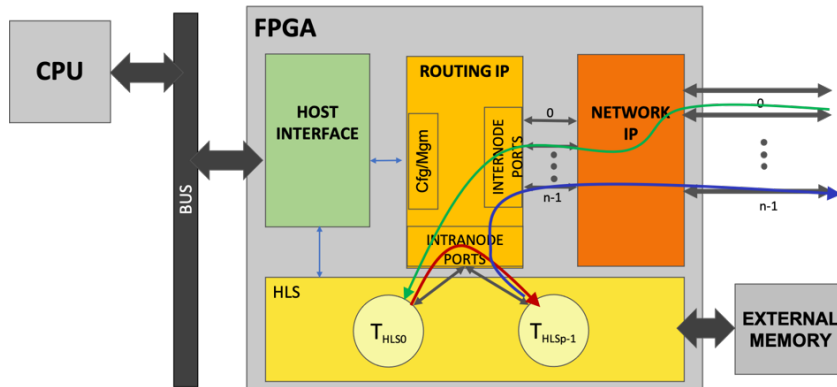
Figure 4. HLS kernels performing intra-node (red line) and inter-node (green line – receive, blue line– send) communications .

In the context of the APEIRON framework, processing tasks are implemented by HLS kernels with Xilinx Vitis. The details of the interface between HLS kernels – the endpoints of the communication – and the Communication IP are described at the end of this section. The Routing IP defines the switching technique and routing algorithm; its main components are the Switch component, the Configuration/Status Registers and the InterNode and IntraNode interfaces.

The Switch component dynamically interconnects all ports of the IP, implementing a channel between source and destination ports.

Dynamic links are managed by routing logic together with arbitration logic: the Router configures the proper path across the switch while the Arbiter is in charge of solving contentions between packets requiring the same port.

For inter-node communications, the routing policy applied is the dimension-order one: it consists in reducing the offset along one dimension to zero before considering the offset in the next dimension.
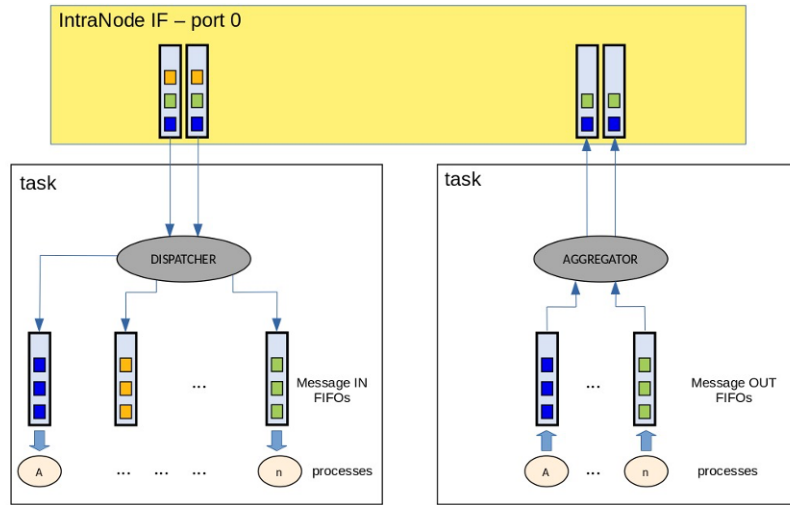
Figure 5. Interface between Intranode Port 0 and the corresponding HLS Task (task_id 0), Message IN FIFOs are identified by the ch_id API parameter.

The employed switching technique — i.e., when and how messages are transferred — is Virtual Cut-Through [7]: the router starts forwarding the packet as soon as the algorithm has picked a direction and the buffer used to store the packet has enough space. The deadlock-avoidance of dimension-order routing is guaranteed by the implementation of two virtual channels for each physical channel (with no fault-tolerance guaranteed) [8].

The transmission is packet-based, meaning that the Communication IP sends, receives and routes packets with a header, a variable size payload and a footer.

The Communication IP was co-designed with the APEIRON software stack in order to achieve very low-latency and scalable bandwidth between processing tasks defined as High-Level Synthesis Kernels.

Starting from a YAML configuration file describing the attributes of each HLS kernel, namely its number of input and output channels and the IntraNode port of the Communication IP to which it is connected, the APEIRON framework links the Communication IP and the HLS kernels that are connected to it and generates the bitstream for the overall design.

The only requisite that HLS kernels must satisfy is in the format of their prototype that must be in this form:

```
void example_apeiron_task(
    [optional kernel-specific list of parameters]
    message_stream_t message_data_in[N_INPUT_CHANNELS],
    message_stream_t message_data_out[N_OUTPUT_CHANNELS])
```

In this way, the HLS kernel implements a generic stream interface for each communication channel, based on the AXI4-Stream protocol [9]. The communication between kernels is expressed through a lightweight C++ API based on non-blocking send() and blocking receive() operations. This simple API allows the HLS developer to perform communications between kernels, either deployed on the same FPGA (intra-node communication) or on different FPGAs (inter-node communication) without knowing the details of the underlying packet communication protocol.

The Communication API can be represented with the following pseudo-code:

```
    size_t send (msg, size, dest_node, task_id, ch_id);
    size_t receive (ch_id);
```

where:

```
dest_node is the n-Dim coordinate of the destination node (FPGA);
task_id is the local-to-node receiving task (kernel) identifier (0-3);
ch_id is the local-to-task receiving FIFO (channel) identifier (0-127).
```

The Communication Library leverages AXI4-Stream Side-Channels to encode all the information needed to forge the packet header.

Adaptation toward/from IntraNode ports of the Routing IP is done by two APEIRON IPs: Aggregator and Dispatcher, shown in Figure 5. The Dispatcher receives incoming packets from the Routing IP and forwards them to the right input channel, according to the relevant fields of the header. The Aggregator receives outgoing packets from the task and forges the packet header, filling then the header/data FIFOs of the Routing IP.

3. Physics Use Case

NA62 is a fixed-target experiment at the CERN SPS North Area, dedicated to measurement of rare kaon decays. We have designed FPGA-RICH, a Particle Identification (PID) system, based on the APEIRON framework and implemented on a single FPGA device, capable of providing results to the online trigger.

This systems represents the evolution of the GPURICH one that provided the same capabilities but on a more complex architecture, with a GPU performing a geometry based PID algorithm and a FPGA hosting the the NaNet design [10, 11, 12] implementing the low-latency direct data transfer between the detector and the GPU memory.

FPGA-RICH receives RICH detector events in a streaming fashion and performs the PID task using a NN, supporting a throughput greater than 10 MHz as per experiment specifications. According to the APEIRON workflow the NN is implemented as a HLS Kernel and receives input data from the RICH detector only (*seedless* model). The resulting model, depicted in Fig. 6, is a three layer Dense network (64x16x4) having in input up to 64 normalized IDs of the photomultipliers hit by the Cherenkov photons in a single event. To limit the FPGA resources footprint we performed a quantization step on the model using QKeras, resulting in two different fixed point representations: <8, 1> for weights and biases and <16, 6> for activations.

Two different features can be inferred for each event: the number of charged particles (N_r) and the number of $e^\pm$ (N_e). In order to prepare the training and validation data for the NN, we prepared different data sets composed by events extracted from NA62 physics runs using the experiment analysis framework. The ground truth for training was provided by the seedless *RichReco* offline reconstruction method.

Since the NN result would be used to enforce a trigger decision the inference performance of the NN is of utmost importance: to get a training set as much as possible similar to online data we trained the network with 3 Mevents extracted from run 8011. Validation has been done on 3.5 Mevents from run 8893 with satisfying results, as shown by ROC curves for N_r in Fig. 7.

Since the NA62 RICH detector is able to discriminate the kind of charged particles only in the $15 - 35$ GeV/c energy range, results for N_e are not equally satisfying.

The model was synthesized on a Xilinx VCU118 FPGA platform at a 150 MHz clock frequency, and used a very limited amount of resources (14% LUT, 2% DSP), being able to sustain a 18.75 MHz throughput with a latency of 146.66 ns.
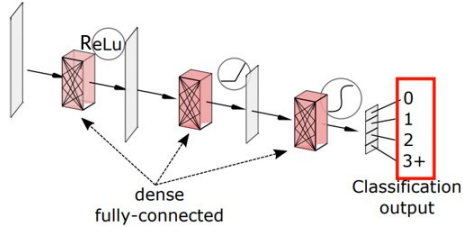
Figure 6. Dense Model (64x16x4) schematics.
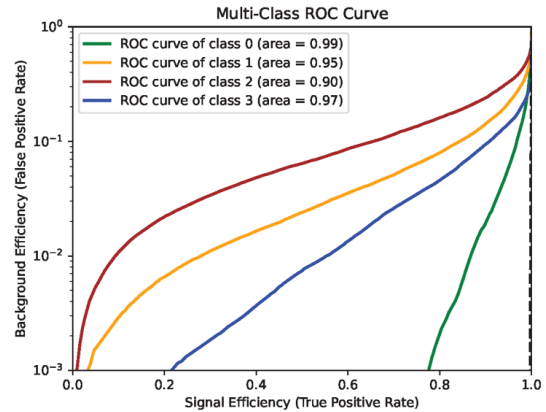
Figure 7. ROC curves for N_r.

4. Conclusions and Future Work

We are continuing the development of the APEIRON framework in order improve its performance and usability. We are finalizing the development of the FPGA-RICH system, integrating the NN kernel in the framework encouraged by the good performance on the identification of charged particles. We envisioned a solution to improve results in identification of $e^{\pm}$, using the LKr calorimeter online primitives that provide information related to the energy of the event.

References

[1] Gilles K 1974 *Information processing* **74** 471–475
[2] Duarte J, Han S, Harris P, Jindariani S, Kreinar E, Kreis B, Ngadiuba J, Pierini M, Rivera R, Tran N and Wu Z 2018 *Journal of Instrumentation* **13** P07027–P07027
[3] Coelho C N, Kuusela A, Li S, Zhuang H, Aarrestad T, Loncar V, Ngadiuba J, Pierini M, Pol A A and Summers S 2021 *Nature Mach. Intell.* **3** 675–686 (*Preprint* 2006.10159) URL https://cds.cern.ch/record/2724942
[4] Ammendola R, Angelucci B, Barbanera M, Biagioni A, Cerny V, Checcucci B, Fantechi R, Gonnella F, Koval M, Krivda M, Lamanna G, Lupi M, Lonardo A, Papi A, Parkinson C, Pedreschi E, Petrov P, Piandani R, Pinzino J, Pontisso L, Raggi M, Soldi D, Sozzi M, Spinella F, Venditti S and Vicini P 2019 *Nucl. Instrum. Methods Phys. Res., A* **929** 1–22. 32 p 32 pages, 23 figures URL https://cds.cern.ch/record/2670907
[5] Ammendola R, Biagioni A, Frezza O, Lonardo A, Lo Cicero F, Paolucci P, Rossetti D, Simula F, Tosoratto L and Vicini P 2013 *Journal of Instrumentation* **8** C12022 URL http://stacks.iop.org/1748-0221/8/i=12/a=C12022
[6] Ammendola R *et al.* 2017 The next generation of exascale-class systems: The exanest project *2017 Euromicro Conference on Digital System Design (DSD)* pp 510–515
[7] Kermani P and Kleinrock L 1979 *Computer Networks* **3** 267–286
[8] Duato J 1995 *IEEE Transactions on Parallel and Distributed Systems* **6** 1055–1067
[9] ARM 2021 Amba axi-stream protocol specification Technical report URL https://developer.arm.com/documentation/ihi0051/latest
[10] Ammendola R, Biagioni A, Frezza O, Lamanna G, Lonardo A, Lo Cicero F, Paolucci P S, Pantaleo F, Rossetti D, Simula F, Sozzi M, Tosoratto L and Vicini P 2014 *Journal of Instrumentation* **9** C02023 URL http://stacks.iop.org/1748-0221/9/i=02/a=C02023
[11] Lonardo A, Ameli F, Ammendola R, Biagioni A, Ramusino A C, Fiorini M, Frezza O, Lamanna G, Cicero F L, Martinelli M, Neri I, Paolucci P, Pastorelli E, Pontisso L, Rossetti D, Simeone F, Simula F, Sozzi M, Tosoratto L and Vicini P 2015 *Journal of Instrumentation* **10** C04011 URL http://stacks.iop.org/1748-0221/10/i=04/a=C04011
[12] Ammendola R, Biagioni A, Fiorini M, Frezza O, Lonardo A, Lamanna G, Lo Cicero F, Martinelli M, Neri I, Paolucci P, Pastorelli E, Pontisso L, Rossetti D, Simula F, Sozzi M, Tosoratto L and Vicini P 2016 *Journal of Instrumentation* **11** C03030 URL http://stacks.iop.org/1748-0221/11/i=03/a=C03030