Data handling in the readout system

Roland Sipos
CERN

CDR Review
18th June 2021

Overview

- Objectives of the readout subsystem
- Functional elements
  - I/O device
  - Trigger Primitive (TP) generation
  - Data request handling
  - Data recording
- Generic readout
  - Design: concepts and models
  - Implementations
- Summary

Objectives

The DUNE readout subsystem is responsible for:

- **Receiving** raw data from a variety of front-end electronics
  - FD variants include: WIB, DAPHNE, uTCA TDE

- **Processing** the in-flight raw data stream
  - Look for errors and handle other data driven aspects (e.g.: calibration)
  - Generate and format Trigger Primitives (TPs) from recognized detector activity

- **Responding** to data requests from the trigger (with requested time windows of data)
  - Data is buffered in readout units' memory (RAM)
  - Potentially overlapping windows from order of microseconds to order of seconds
  - Including the unique Supernova Neutrino Burst (SNB) data request (~100 seconds continuous data recording)

Data reception

While internal data organization differs, all packets coming from the electronics shall have a **64b timestamp field** and another one that **fully specifies the origin.**

- **WIB** frame has a header with 1 ADC value for 256 channels. It’s with fixed arrival rate of fix sized payloads. The timestamps increment with fixed time delta and are sorted on arrival.

- **DAPHNE** frame has 375 consecutive ADC values for 1 channel. It’s with variable arrival rate of fix sized payloads. Also with variable time delta between consecutive frames’ timestamps, but they are sorted on arrival.

- Data from **uTCA TDE** boards will have N consecutive ADC values for 64 channels. It’s with fixed arrival rate of fixed sized payloads. Timestamps are sorted on arrival. (Similar to WIBs.)

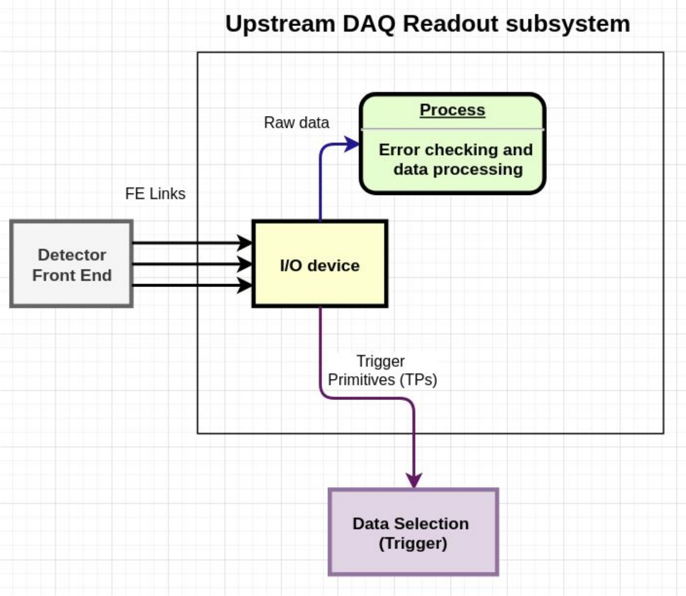
Due to the range of characteristics, different **functional element implementations** are needed!

I/O device

The main and only **input is raw data** from different front-end electronics. Front-end links are connected to an **I/O device or card** (e.g.: FELIX).

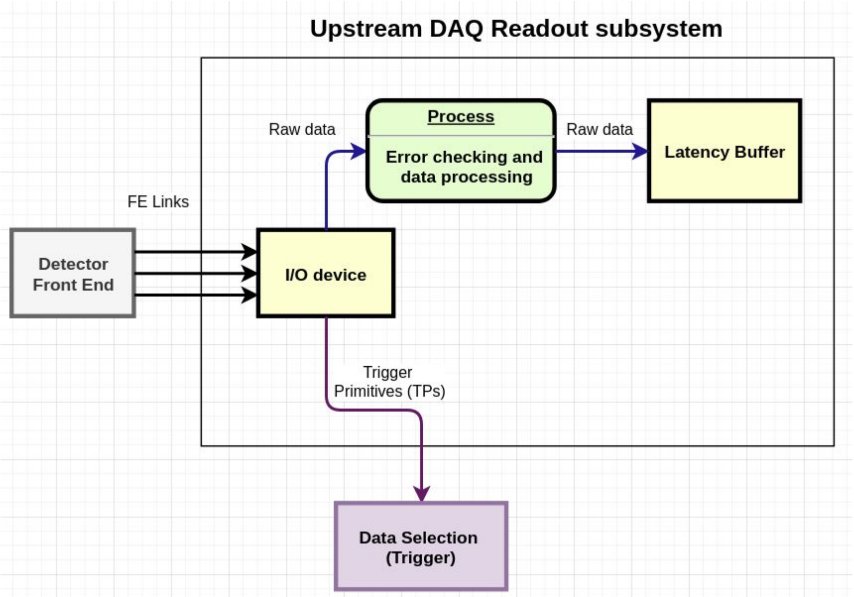
The readout needs to interpret raw data and find possible problems and errors with and within data (e.g.: data integrity). A **processing pipeline** is responsible for this.

Trigger Primitives (TPs) are "hits" in raw data that shows physics activity on the front-end channels. These are generated and sent to the Data Selection system. (*talk from Josh K.*)



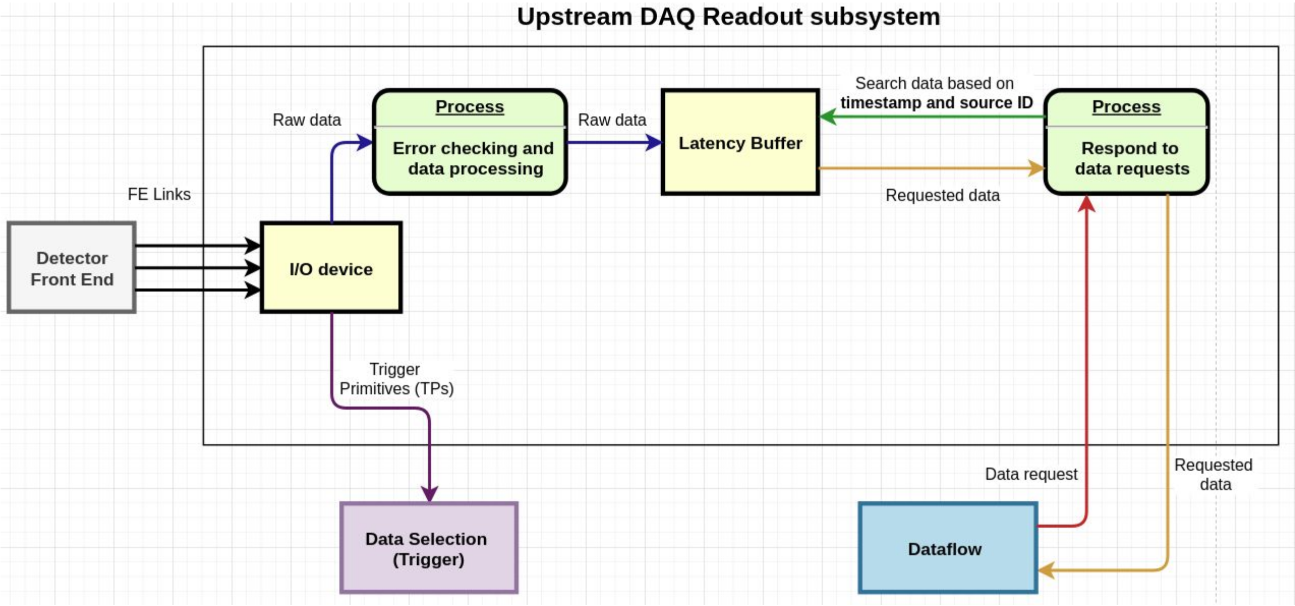
Latency buffer

Data is temporarily stored in memory buffers. These buffers has certain attributes that ensures search-ability based on a lookup criteria. A notable example for this, is the **lookup based on the timestamp**, where the timestamp can be translated to a position in the buffer.



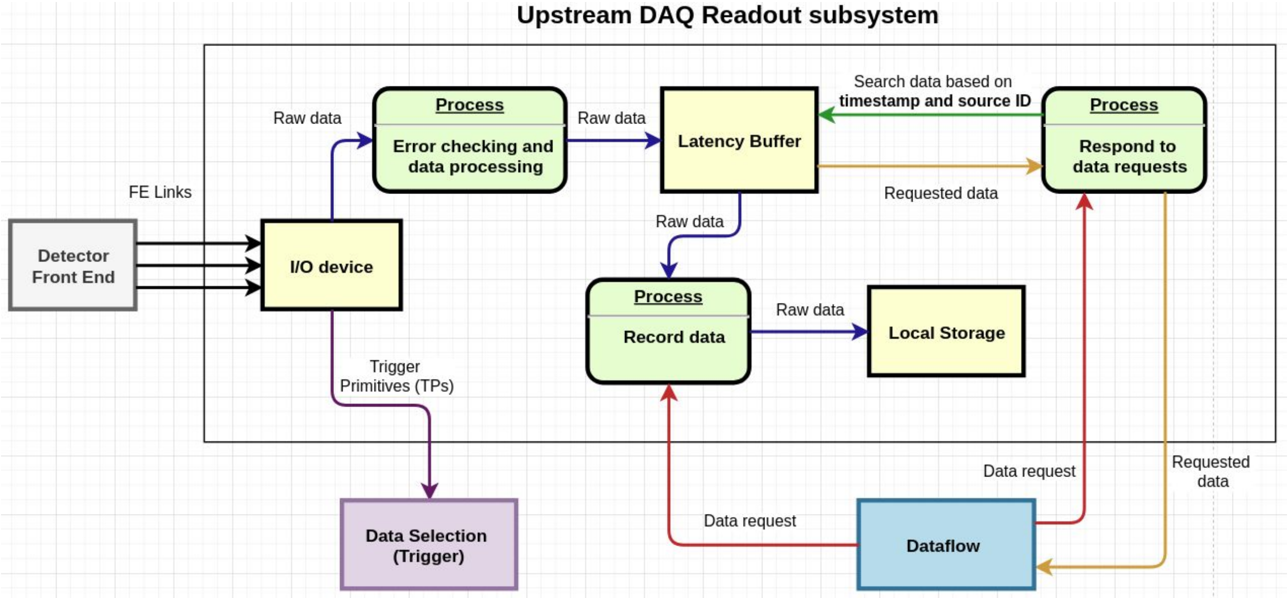
Responding to data requests

Incoming **data requests** are handled via accessing the latency buffer then match and extract the requested data with the given time window. Interfaces are provided by the application framework (*previous talk from Kurt B.*).



Recorded data

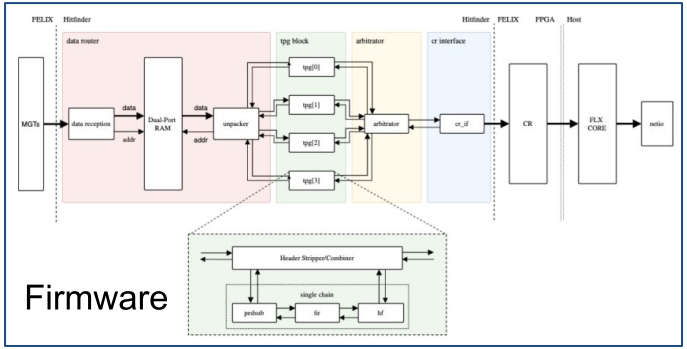
Data leaving the latency buffer can be streamed to a **transient data store**, which is local to the readout unit. The recorded data are transferred to other subsystems with additional metadata, notifications and acknowledgements.

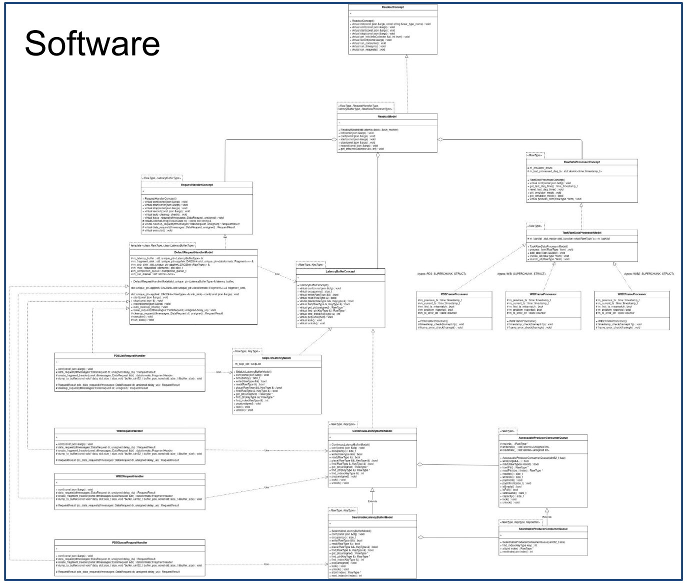


Implementation principles

Functional elements implementations’ are a mixture of firmware and software, and COTS and custom hardware:

- High rate data pre-processing in FPGA
  - Data reception and aggregation
  - Trigger primitive generation
- Buffering and post-processing on COTS hardware and custom software implementations
  - Post-formatting of TP data
  - Buffering and data request handling
  - Software driven storage





Readout design

The readout subsystem promotes **generic and reusable design principles** of the functional elements:

- Support for multiple front-end types (**transparent to data providers**)
- Exact and well defined generic interfaces (**concepts**)
- Front-end specific specializations (**models**)

With the following constraints:

- Homogeneous data path: Won’t mix FE data types and their solutions
- Concurrent access efficiency: Design needs to support highly concurrent access and corresponding rates

Implementation details

Generic readout package

- Demonstrates the handling of different FE type functionalities

Front-end receiver based on the FELIX: flxlibs

- Showcases real FE hardware handled by the FELIX and integration with the generic readout

Details on Trigger Primitive generation offloading firmware blocks

- Proven solutions to offload CPU intensive tasks to hardware accelerators

Summary on technical details and conclusions of the readout of ProtoDUNE-SP

- Technology review on readout tasks and their CPU offloading

Summary

- The readout subsystem has clear requirements and design concepts
- Most of its functional elements were validated in ProtoDUNE-SP
- Now we are in the detailed design phase based on the previous experiences
- The main target is to have a modular and extensible readout system

End

Thank you for your attention!