

Fast FPGA-based identification of frequent patterns in data streams using statistical properties

Daniele Passaro

Scuola Normale Superiore & INFN Sezione di Pisa (Italy)

and

- Antonio Brau (Formerly with University of Pisa, Italy)
- Serena Castellotti (Formerly with University of Pisa, Italy)
- Federico Lazzari (University of Pisa & INFN Pisa, Italy)
- Giovanni Punzi (University of Pisa & INFN Pisa, Italy)



SCUOLA
NORMALE
SUPERIORE



Frequent data-patterns in HEP

Data-pattern = specific sequence of bits in a data streams

In the context of HEP experiments, the most obvious frequent patterns are: **hot-channels**

- at the readout level, they just appear as more frequent channel-IDs

Standard methodologies to monitor them:

- traditional methods rely on monitoring farm
 - monitored data are just a small fraction of the total available data
 - over-threshold approach: just fill histograms of the detector hitmap
 - semi-supervised ML methods: the CMS-HCAL example [[JINST 21 P10004](#)]

Curing the noisy channels often requires: automated alarms → stop run → front-end reset

- Several **minutes / hours** of data can be affected by hot-channels, even if already identified

Tracks and tracker alignment

Tracks are just “patterns” of aligned hits.

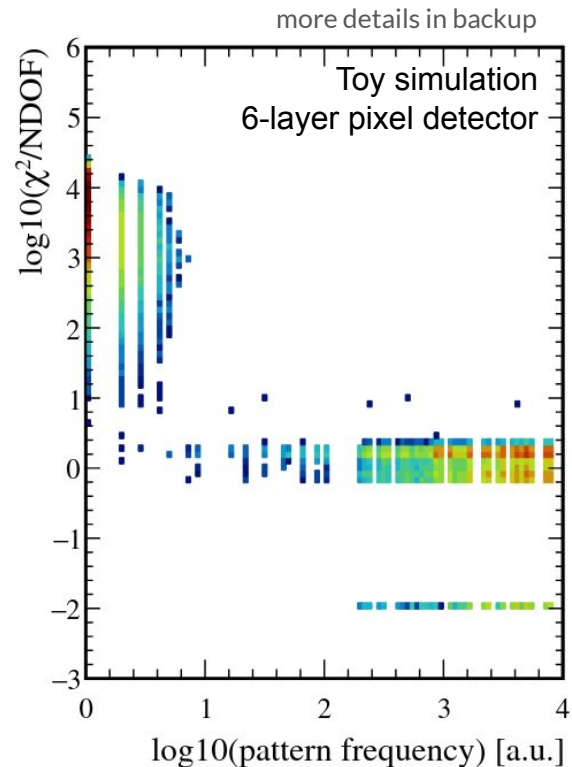
To perform the tracking, detector alignment is needed

- Computational demanding task, often takes $O(\text{days})$:
- At LHC, only LHCb performs some real-time alignment tasks with latencies of $O(10 \text{ mins})$
- data have to be buffered / re-processed before applying the alignment evaluated on the data themselves
- do we really need to run the full alignment to verify if it has changed?

Key insight [\[Del Viva et al. 2013\]](#): **combinations of on-track hits** are more likely to be observed than **random combinations of hits**

Alignment stability in terms of frequent patterns

Real and fake tracks separate neatly in the χ^2 /frequency plane

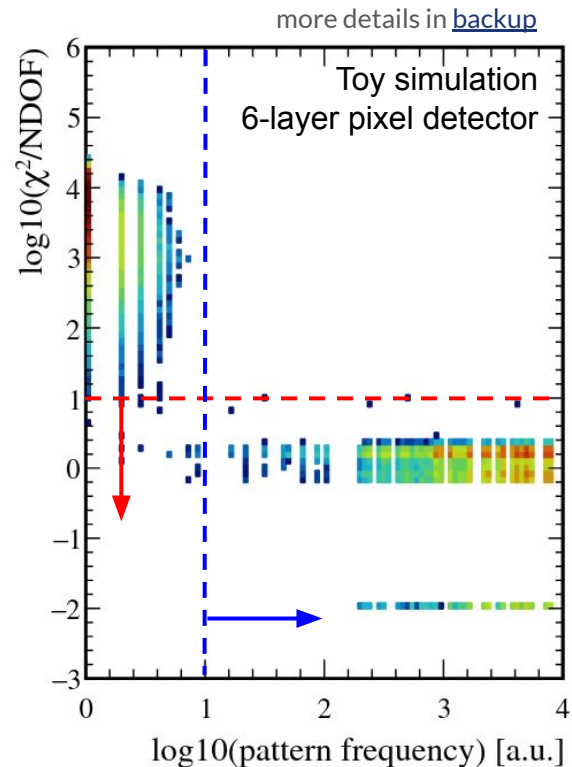


Alignment stability in terms of frequent patterns

Real and fake tracks separate neatly in the χ^2 /frequency plane

This means, in terms of monitoring only
(not for event-reconstruction):

- χ^2 -based selection is almost equivalent to selection based on the pattern frequency

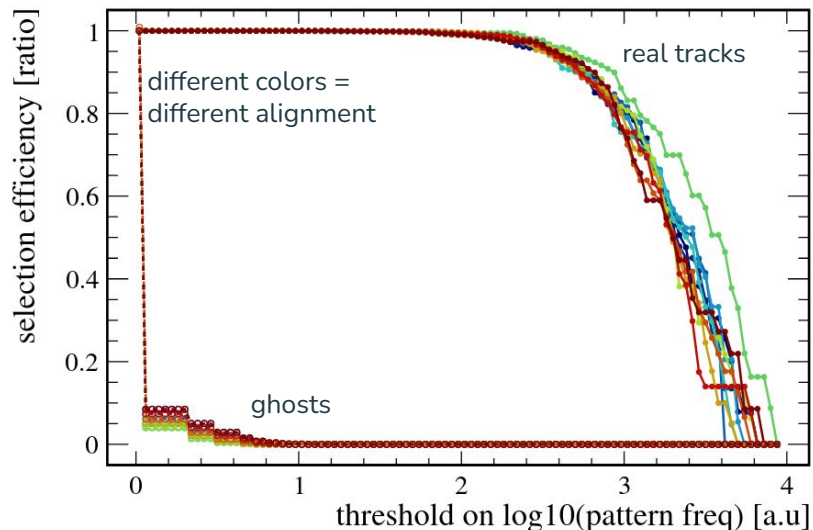


Alignment stability in terms of frequent patterns

Real and fake tracks separate neatly in the χ^2 /frequency plane

This means, in terms of monitoring only
(not for event-reconstruction):

- χ^2 -based selection is almost equivalent to selection based on the pattern frequency
- **BUT:** χ^2 value is dependent on the detector alignment, while the pattern frequency is an **emergent property of the data**
- the frequency of patterns associated to real tracks **does not change** with misalignment
- If the most frequent patterns change, it means that the alignment has changed



Core idea

In both hot-channel and alignment cases, **real-time identification of most frequent “patterns”** is a powerful proxy:

- enables **tracking of varying conditions** while taking data
- **fully data-driven approach**:
 - no need for prior knowledge of data-taking conditions, detector geometries etc.

Counting is conceptually trivial, but:

- # possible patterns = 2^{Nbits}
 - **Nbits=10** $\Rightarrow O(10^3)$ patterns, manageable
 - **Nbits=32** $\Rightarrow O(10^9)$ patterns, impossible at readout

Looking for some **flexible**, **scalable** and **efficient** algorithm, with an eye for FPGA implementation

Why ?

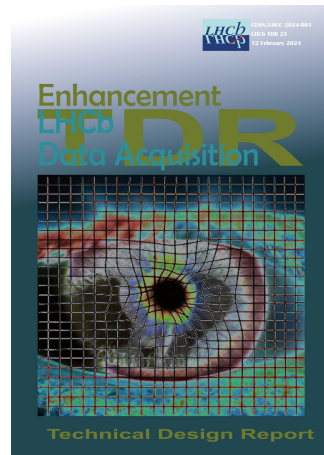
Real-life application: the LHCb DWT device

The Downstream-Tracker (DWT): an **FPGA**-based processor using the [artificial retina architecture](#)

- To be deployed in LHCb during Run 4 as part of the LS3 Enhancements
- The DWT will reconstruct track primitives downstream of the magnet
 - tracklets from the Scintillating Fiber (SciFi) sub-detector
- Track reconstruction happens at 30 MHz bunch-crossing rate, **before event-building**

Application of fast identification of frequent patterns :

- **hot-channels removal on-the-fly**
 - avoid throughput penalties and reconstruction of ghost tracks
- (speculative) **continuous alignment updates**
 - fast monitoring of alignment stability can be used to trigger alignment updates / real-time corrections



[LHCb-TDR-025](#)



**“Talk is cheap,
show me the method”**

- semicit.

Floating Top-K: theory

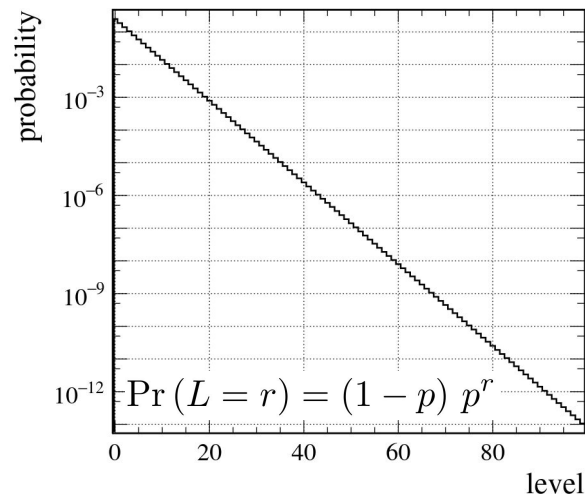
Probabilistic algorithm (FTK) introduced by [Song et al. 2019](#) to monitor social-media trends or most searched terms

Floating Top-K: theory

Probabilistic algorithm (FTK) introduced by [Song et al. 2019](#) to monitor social-media trends or most searched terms

Key features:

- 1) at each item occurrence, extract a **level** from a geometric distribution;

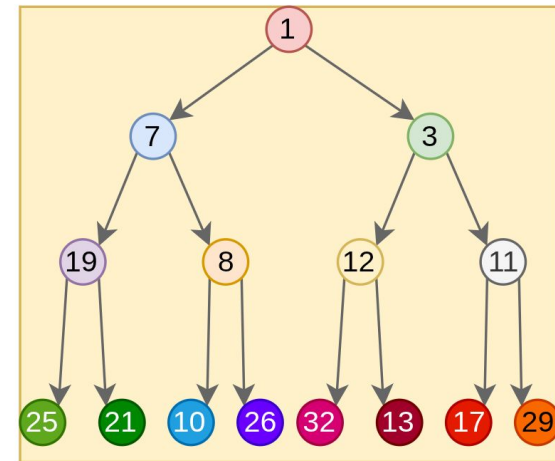


Floating Top-K: theory

Probabilistic algorithm (FTK) introduced by [Song et al. 2019](#) to monitor social-media trends or most searched terms

Key features:

- 1) at each item occurrence, extract a **level** from a geometric distribution;
- 2) store in a **priority queue** (indexed min-heap) the item and sort it based on the levels of each item;



Floating Top-K: theory

Probabilistic algorithm (FTK) introduced by [Song et al. 2019](#) to monitor social-media trends or most searched terms

Key features:

- 1) at each item occurrence, extract a **level** from a geometric distribution;
- 2) store in a **priority queue** (indexed min-heap) the item and sort it based on the levels of each item;
- 3) Update each item's level with the maximum observed level: $\ell = \max\{L_1, \dots, L_N\}$;

$$\mathbb{E}[\ell] = \sum_{r=1}^{\infty} \left(1 - (1 - p^r)^N\right)$$



Monotonically increasing in N!
Can use it as surrogate of item's frequency to sort the heap structure

Floating Top-K: theory

Probabilistic algorithm (FTK) introduced by [Song et al. 2019](#) to monitor social-media trends or most searched terms

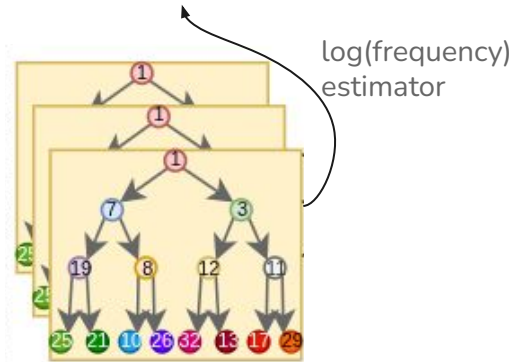
Key features:

- 1) at each item occurrence, extract a **level** from a geometric distribution;
- 2) store in a **priority queue** (indexed min-heap) the item and sort it based on the levels of each item;
- 3) Update each item's level with the maximum observed level: $\ell = \max\{L_1, \dots, L_N\}$;
- 4) Implement m statistically independent heaps;

A. Brau [Master's thesis](#)

$$\widehat{\log N} = \left(\frac{1}{m} \sum_{r=1}^m l^{(r)} + \frac{1}{2} \right) \cdot \log \frac{1}{p} - \gamma$$

$$\text{Var} \left[\widehat{\log N} \right] = \frac{\pi^2}{6m}$$

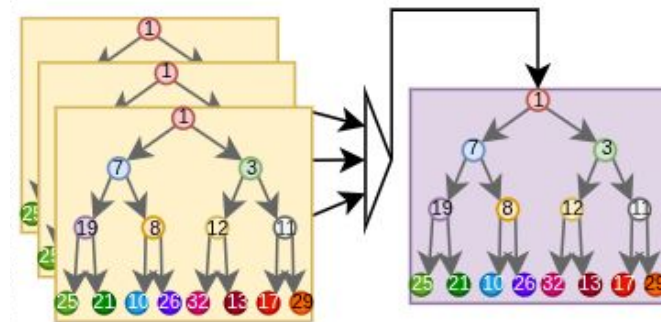


Floating Top-K: theory

Probabilistic algorithm (FTK) introduced by [Song et al. 2019](#) to monitor social-media trends or most searched terms

Key features:

- 1) at each item occurrence, extract a **level** from a geometric distribution;
- 2) store in a **priority queue** (indexed min-heap) the item and sort it based on the levels of each item;
- 3) Update each item's level with the maximum observed level: $\ell = \max\{L_1, \dots, L_N\}$;
- 4) Implement m statistically independent heaps;
- 5) At each time window t , merge them and perform time averages in any past time window



Adaption and modifications for real-time

Our requirements

- low FPGA resource usage
- high processing throughput
- stability of data-structure against random fluctuations
- availability of a snapshot of the **latest** most frequent data patterns

FTK simplifications

m parallel data-structures too resource demanding

⇒ Drop feature 4

windowed time averages not needed

⇒ Drop feature 5

Modifications

For stability & time evolution:

1. For each item, store **second** l_2 and **third** l_3 observed maxima too
2. Sort heap based on sum:
 $S = l_1 + l_2 + l_3$
3. At every time update, delete first max and shift:
 $l_2 \rightarrow l_1, l_3 \rightarrow l_2$

Adaption and modifications for real-time

Our requirements

- low FPGA resource usage
- high processing throughput
- stability of data-structure against random fluctuations
- availability of a snapshot of the **latest** most frequent data patterns

FTK simplifications

m parallel data-structures too resource demanding

⇒ **Drop feature 4**

windowed time averages not needed

⇒ **Drop feature 5**

Modifications

For stability & time evolution:

1. For each item, store **second** l_2 and **third** l_3 observed maxima too
2. Sort heap based on sum:
 $S = l_1 + l_2 + l_3$
3. At every time update, delete first max and shift:
 $l_2 \rightarrow l_1, l_3 \rightarrow l_2$

Adaption and modifications for real-time

Our requirements

- low FPGA resource usage
- high processing throughput
- stability of data-structure against random fluctuations
- availability of a snapshot of the **latest** most frequent data patterns

FTK simplifications

m parallel data-structures too resource demanding

⇒ Drop feature 4

windowed time averages not needed

⇒ Drop feature 5

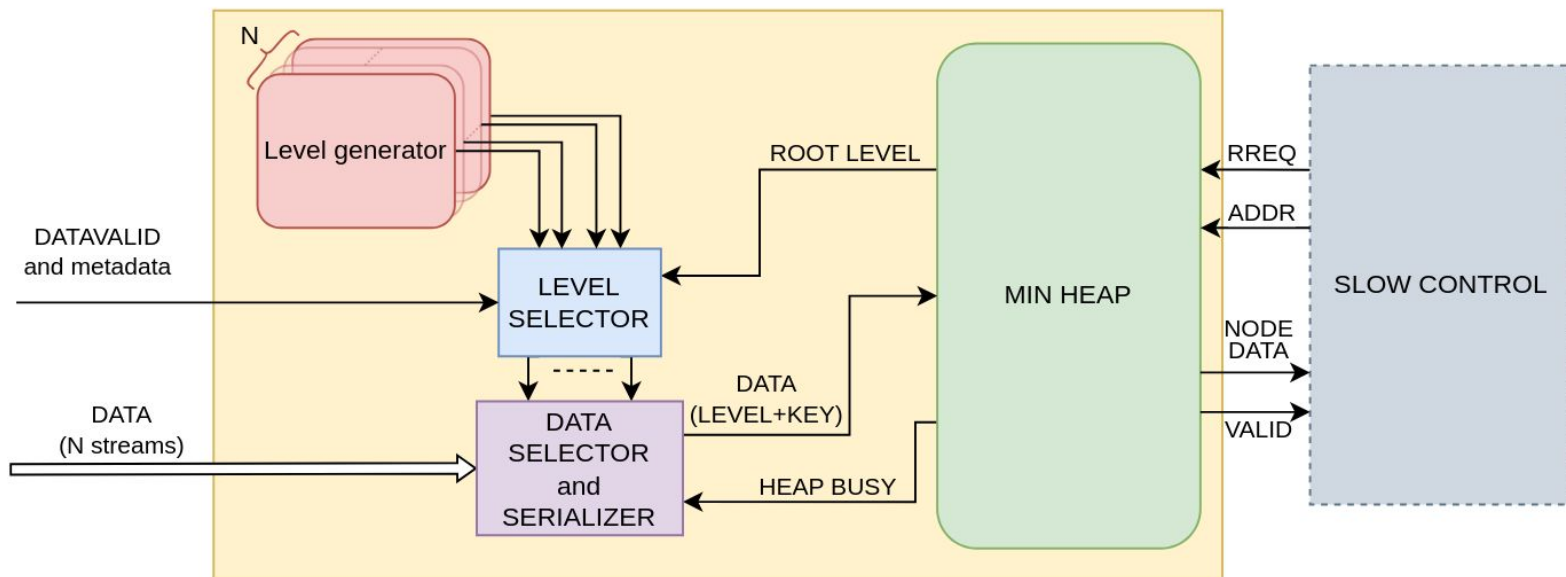
Modifications

For stability & time evolution:

1. For each item, store **second** l_2 and **third** l_3 observed maxima too
2. Sort heap based on sum:
 $S = l_1 + l_2 + l_3$
3. At every time update, delete first max and shift:
 $l_2 \rightarrow l_1, l_3 \rightarrow l_2$

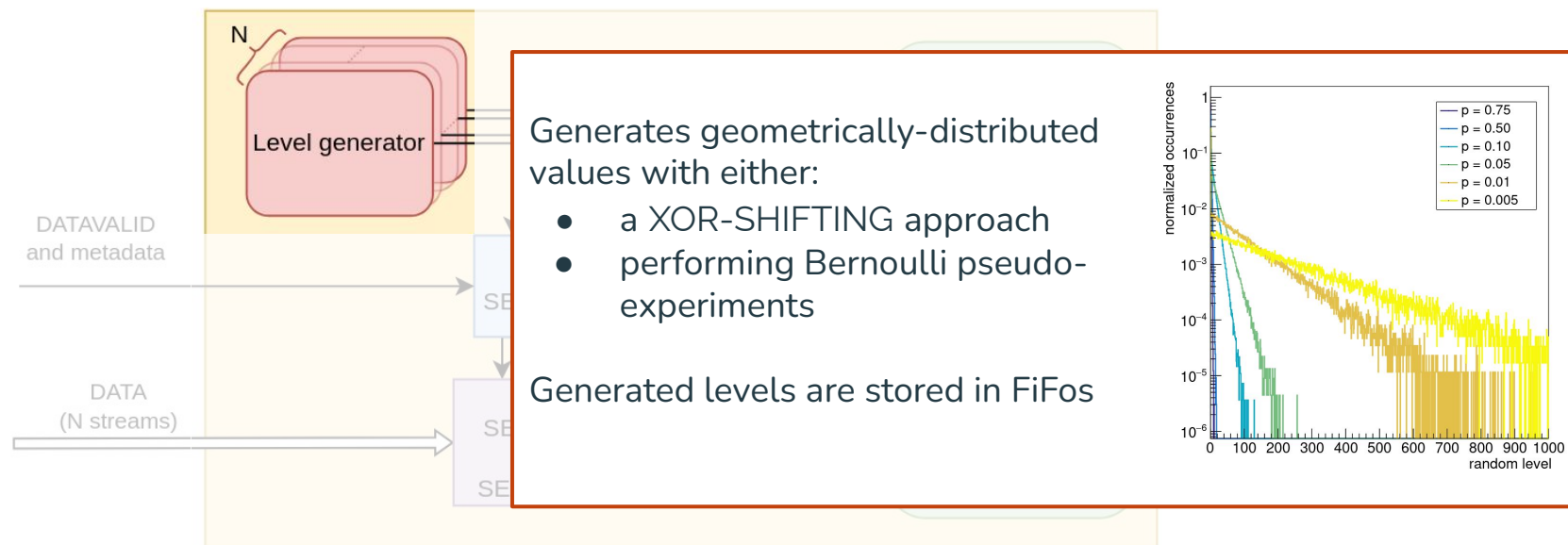
FPGA gateway design

Implemented in pure VHDL, consists of a few main blocks



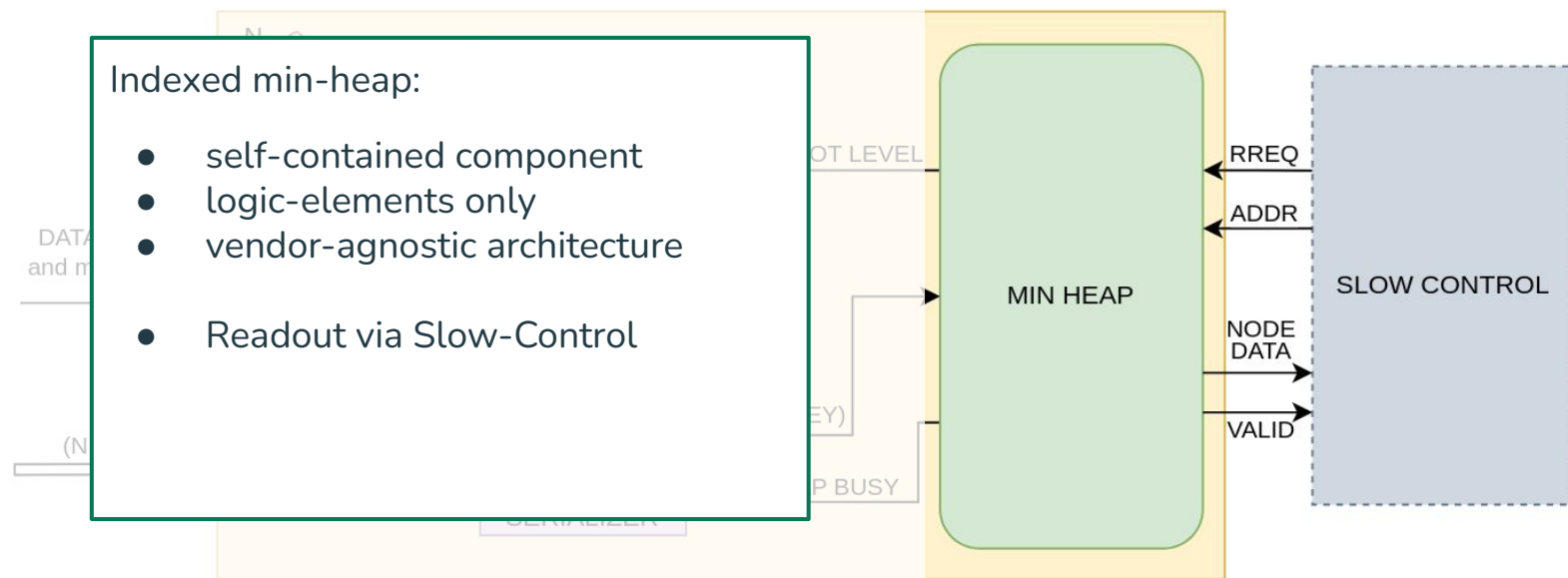
FPGA gateway design

Implemented in pure VHDL, consists of a few main blocks



FPGA gateway design

Implemented in pure VHDL, consists of a few main blocks

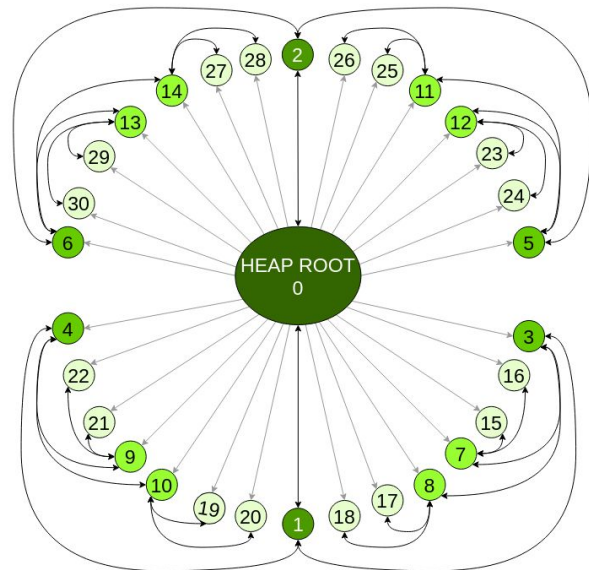


Heap priority-queue in FPGA

Leveraged full parallelism of the device:

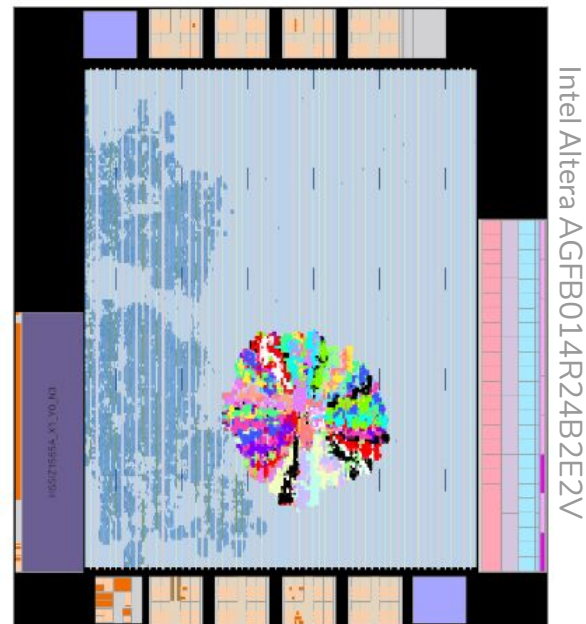
- Each node connects to its parent, two children, and the heap root
- [level-key] exchanges are managed via handshake
 - Nodes compare levels with neighbors and swap data upon mutual agreement

This enables fast, efficient heap self-evolution without the need for complex FSMs



Compilation results and design performance

- processing clock at $f_{\text{CLK}} = 500$ MHz
- ~880 LEs / head node
- (~1700 LEs + 13 M20k) / random generator
- fits hundreds of nodes (*i.e.* patterns) within one chip
 - 127 nodes $\Rightarrow$ O(10%) of LHCb and ALICE readout FPGAs



Compilation results and design performance

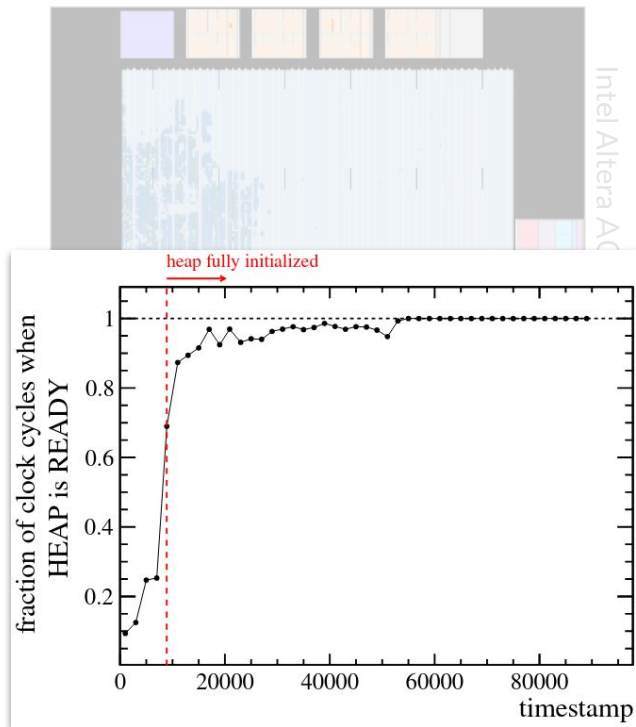
- processing clock at $f_{\text{CLK}} = 500$ MHz
- ~ 880 LEs / head node
- (~ 1700 LEs + 13 M20k) / random generator
- fits hundreds of nodes (*i.e.* patterns) within one chip
 - 127 nodes $\Rightarrow$ O(10%) of LHCb and ALICE readout FPGAs

The architecture is stall-free in stationary conditions:

- throughput limited only by the input bandwidth and processing clock:

$$\text{bandwidth} = \text{pattern}_{\text{size}} \times N_{\text{STREAMS}} \times f_{\text{CLK}}$$

O(10-100 Gbps) or O(Gpatterns/s)

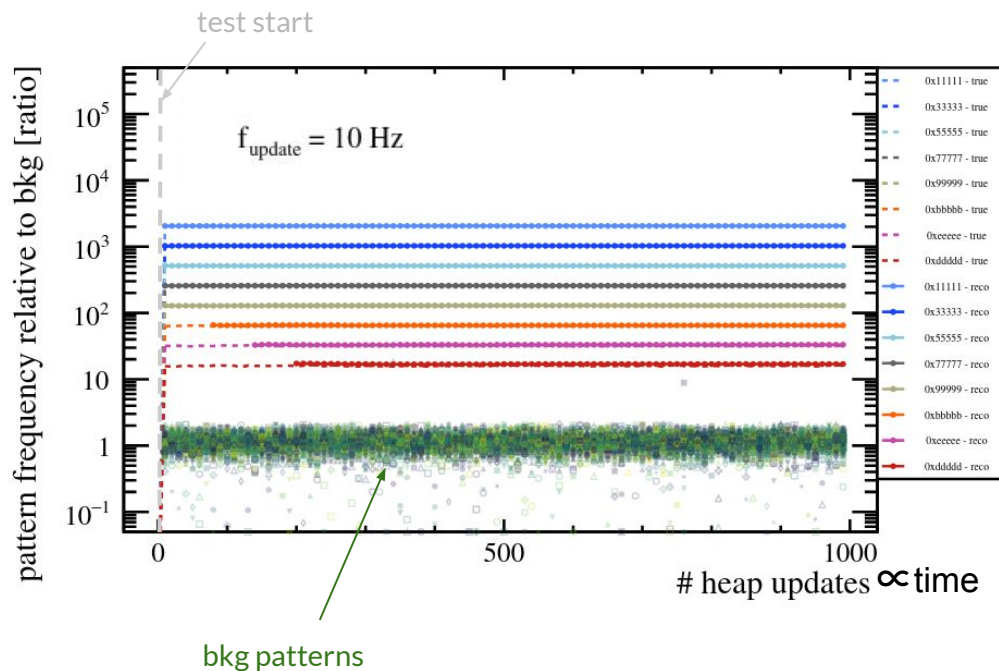


Results on real hardware (1)

Setup:

- heap structure with 127 nodes
- input random patterns generated internally @ $5 \cdot 10^8$ patterns/s
- pattern size: 20 bits
 - encoding $O(1M)$ channels
- injected 8 “hot” patterns with higher frequencies than “bkg”

Sensitivity test: patterns in a broad range of frequency are identified by the algorithm



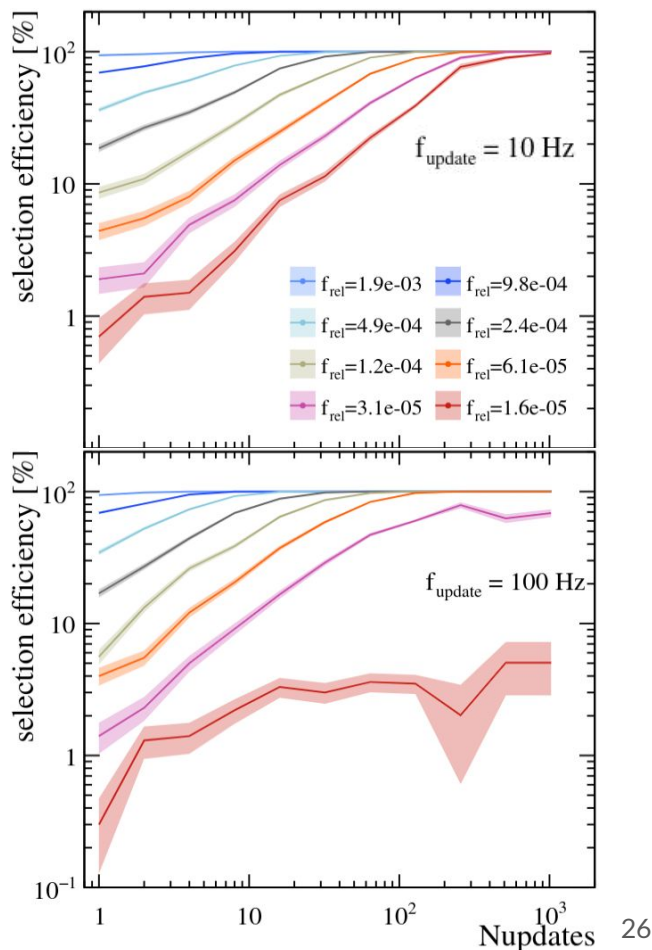
Results on real hardware (2)

Consistency check: the probability of observing a pattern should

- **increases** with relative pattern probability f_{rel}
- **increases** with the number of heap updates $N_{updates}$ before the readout
- **decreases** with increasing the frequency of the heap updates f_{update}
 - means decreasing the statistics accumulated within each time window

Performance:

- Patterns with $f_{rel} \sim 10^{-3}$ identified in $\sim 10\text{ms}$
- More rare patterns may require seconds or longer



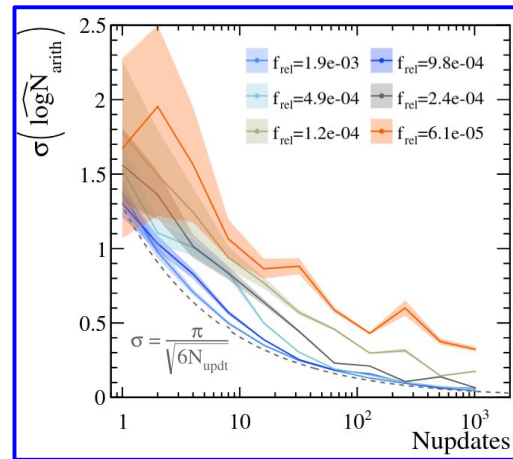
Resolution of the frequency estimate

The FTK algorithm implements m priority queues, so that

$$\text{Var} \left[\widehat{\log N} \right] = \frac{\pi^2}{6m}$$

Under stationary conditions, “spatial” (m) averaging is the same as “time” (Nupdates) averaging:

- with an **arithmetic mean**, resolution $\sim 10\%$ in 100 updates (1s at 100 Hz of update frequency)
- items with lower frequencies have larger σ due to the lower selection efficiency $\Rightarrow$ larger fluctuations



Resolution of the frequency estimate

The FTK algorithm implements m priority queues, so that

$$\text{Var} \left[\widehat{\log N} \right] = \frac{\pi^2}{6m}$$

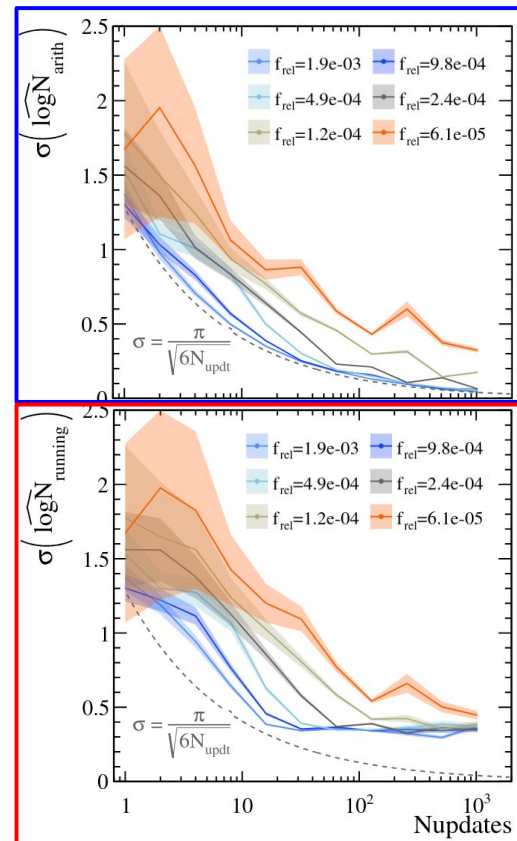
Under stationary conditions, “spatial” (m) averaging is the same as “time” (N_{updates}) averaging:

- with an **arithmetic mean**, resolution $\sim 10\%$ in 100 updates (1s at 100 Hz of update frequency)
- items with lower frequencies have larger σ due to the lower selection efficiency $\Rightarrow$ larger fluctuations

Running mean approach implemented:

$$\hat{x}_t = \lambda \cdot \hat{x}_{t-1} + (1 - \lambda) \cdot x_t^{\text{obs}}$$

- worse resolution, but allows to continuously track dynamically changing conditions



Conclusions and future prospects

Presented **novel idea for FPGA-based (real-time) pattern monitoring**

Ideal applications:

- **automatic** identification of **hot-channels** during readout
- **tracker alignment monitor pre-reconstruction**

Preliminary results:

- architecture is compact enough to fit in readout boards across HEP experiments
 - throughput limited only by input bandwidth $\Rightarrow$ can process $O(10-100 \text{ Gbps})$
- hardware processing is consistent with the theory expectations
 - can achieve $O(10\%)$ resolution on frequency estimate in $O(1s)$

For further information:

- Application to natural vision processing of images: [link](#)
- A. Brau [Master's thesis](#)

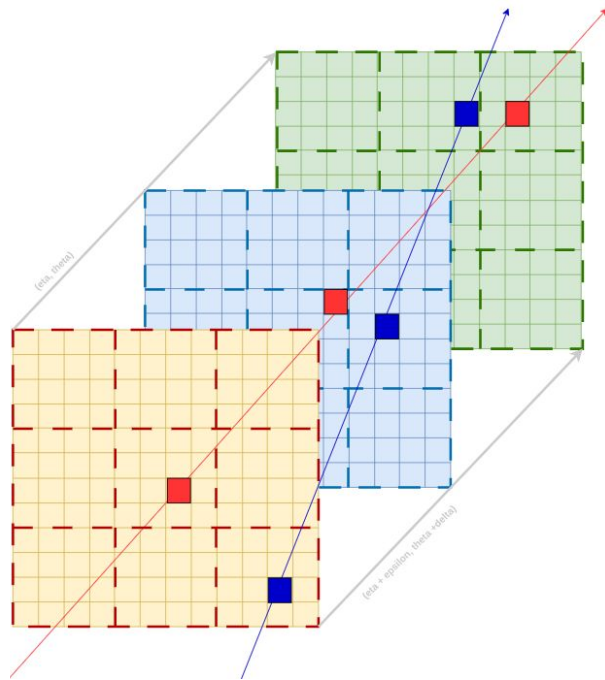


BACKUP

Toy simulation for alignment studies

Developed **toy simulation** to visualize the problem:

- 6-layer pixel detector, focus into a small (η, θ) region
- pitch size $55 \mu\text{m}$
- uniform hit-resolution distribution:
 $\sigma = 55/\text{sqrt}(12) \mu\text{m} = 16 \mu\text{m}$
- digitization discretization applied
- hit-patterns encoded as the concatenation of hit-position difference on consecutive layers
- random noisy pixels injected into the events
- patterns are merged based on their similarity



PATTERN is: $(1 < 4 \cdot 3) \mid (2 < 4 \cdot 2) \mid (2 < 4 \cdot 1) \mid (2 < 4 \cdot 0) = 0x1222$

PATTERN is: $(-1 < 4 \cdot 3) \mid (5 < 4 \cdot 2) \mid (-2 < 4 \cdot 1) \mid (3 < 4 \cdot 0) = 0xFFE3$