Data Processing Firmware of the Upstream Tracker for LHCb Run 3

C. Abellan Beteta[1], F. Alessio[2], G. Bassi[3], I. Bezshyiko[1], C. Hadjivasiliou[4], G. Hijano Mendizabal[1], W. Krupa[5], F. Lazzari[3,6], G. Loustau de Linares[1], A. Mathad[1], M. Peco Regales[1], G. Punzi[3,6], M. Rudolph[5], M. Tobin[2,7], L. Villardita[1], and G. Vouters[8]

***Abstract*—The LHCb experiment was significantly upgraded for Run 3 to enable data acquisition with a fully software-based trigger. A key part of it is the Upstream Tracker (UT), a four-plane silicon microstrip detector installed before the dipole magnet in early 2023. The UT consists of 968 silicon sensors (in four designs) and 4192 SALT ASICs, which perform analogue processing, digitization, common-mode subtraction, zero suppression, and serialization. This information is later sent to the TELL40 back-end readout boards.**

In simplified terms, each TELL40 board functions as a collection of optical fibre receivers connected to a high-performance Intel Agilex 10 FPGA, which processes the data and transfers it to a PCIe interface for further handling.

This publication focuses on the gateware designed to perform this task for the Upstream Tracker (UT).

It starts outlining the specific challenges posed by the UT's electronics architecture, data format and high data rates, and describes how these were addressed through solutions that reduced complexity, at the expense of certain trade-offs.

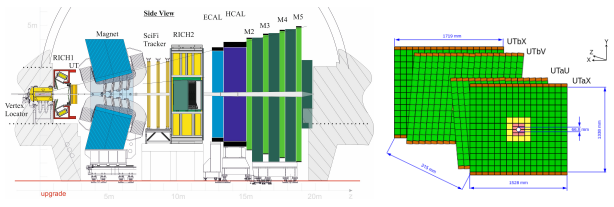
In addition to the core data acquisition functionality, several monitoring and control mechanisms were implemented as part of the Experiment Control System (ECS) and the Timing and Fast Control (TFC) system. The resulting architecture, along with the key design decisions, is presented. The development of a system with custom inputs and outputs required the usage of several design verification and debugging techniques that are also described. The task also required to write specific software that is described too.

I. Introduction

THE UT is a silicon microstrip detector forming part of the LHCb tracking system and is located upstream of the dipole magnet, allowing the measurement of particle positions before magnetic bending. It consists of four layers of silicon sensors mounted on both sides of vertical mechanical support structures, known as staves [1]. Figure 1 shows a schematic view of the LHCb detector and the position of the Upstream Tracker (UT).

The analogue signals produced by the sensors are processed and digitized by 4192 Silicon ASIC for LHCb Tracking (SALT) chips [2]. Each SALT ASIC reads out 128 detector strips at a sampling frequency of 40 MHz, matching the LHC bunch-crossing rate, and after some processing produces zero-suppressed hit information consisting of strip identifiers and ADC values. The data are transmitted as serial digital streams through up to six electrical e-links [3], although the ASICs can be configured to operate with as few as three e-links depending on the expected occupancy.

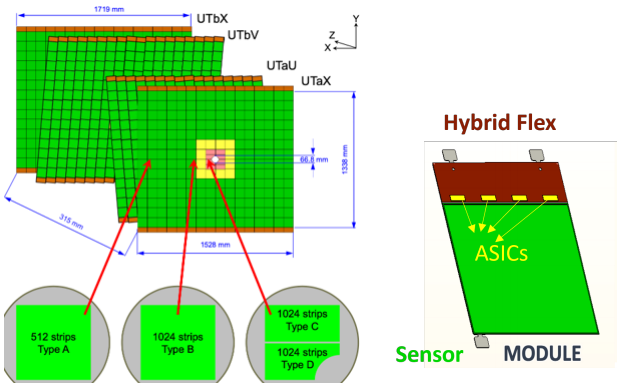
The particle flux decreases with increasing distance from the beam pipe, resulting in significantly different occupancies across the detector acceptance. To accommodate these variations, the UT employs different sensor geometries as a function of radial position: sensors with shorter strips and finer pitch are installed in the high-occupancy central region, while sensors with longer strips and coarser segmentation are used at larger radii, as shown in Figure 2. The front-end readout architecture is optimized accordingly. SALT ASICs located in the innermost region are configured to use five e-links to sustain the higher data rates, while four- and three-e-link configurations are used at intermediate and outer radii, respectively. This graded allocation of bandwidth matches the expected occupancy distribution while minimizing the amount of copper wiring required within the detector acceptance. After aggregation by the back-planes and Data and Control Boards (DCB), the SALT data are transmitted to the back-end readout system using the GBT protocol. In the UT configuration, GBT links operate at 40 MHz and transmit fixed-length 104-bit (13-byte) frames. The SALT data must therefore be packed into the payload bytes of these frames. Although the detector employs three-, four-, and five-e-link SALT configurations, only two electrical frame formats are used in practice. Three- and four-e-link ASICs are transmitted using the same GBT mapping as the five-e-link configuration, with the unused payload bytes left empty. Adopting a common frame format for all non-three-e-link-packed configurations simplified the design of the PEPI backplane—the interface board that distributes detector e-link



(a) (b)

Fig. 1: Schematic views of the LHCb Upgrade I detector (a) and the UT detector showing its four layers (b).

[1]University of Zurich, Zurich, Switzerland.
[2]CERN, Geneva, Switzerland.
[3]Università di Pisa, Pisa, Italy.
[4]University of Maryland, College Park, MD, USA.
[5]Syracuse University, Syracuse, NY, USA.
[6]INFN Sezione di Pisa, Pisa, Italy.
[7]Institute of High Energy Physics, Beijing, China.
[8]Centre National de la Recherche Scientifique (CNRS), France.
Corresponding author: C. Abellan Beteta (e-mail: [carlos.abellan.beteta@cern.ch]).



(a) (b)

Fig. 2: Location of the different UT sensor types (a) and a schematic of the UT hybrid modules and asics (b).

signals to the DCBs—by reducing the number of electrical formats that had to be supported to just two.

This simplification, however, shifted part of the complexity to the back-end firmware. Although only two electrical frame formats are present on the links, the TELL40 firmware must still distinguish between the different SALT configurations and correctly decode the corresponding payload layouts. As a result, supporting the simplified front-end architecture required additional flexibility in the firmware design. The supported frame formats are shown in Fig. 3.

GBT frame byte	13	12	11	10	9	8	7	6	5	4	3	2	1	0
4 x 3-eports			24-bit			24-bit			24-bit			24-bit		
2 x 3-eports			24-bit					24-bit						
2 x 4-eports			32-bit					32-bit						
2 x 5-eports			40-bit					40-bit						

Fig. 3: GBT frame type

The back-end readout system of LHCb is based on the PCIe40 platform. Each PCIe40 board is equipped with up to 24 optical input links and a bidirectional optical control link, an Intel Arria 10 FPGA [4], and two PCIe Gen3 $\times 8$ interfaces for data transfer to the host computer. When programmed with detector-specific readout firmware, the board is referred to as a TELL40. TELL40 boards are used throughout the LHCb experiment and provide a common hardware platform for detector readout, control, and monitoring. For the UT detector, the TELL40 receives the GBT data streams, performs the data processing described in this paper, and transfers the resulting event fragments to the Event Builder farm through the PCIe interfaces.

To streamline firmware development across the experiment, LHCb provides a common firmware framework for the PCIe40 platform [5]–[7]. The framework implements the low-level hardware functionality required by all detectors, including optical link management, clock generation and distribution, PCIe communication, configuration interfaces, and system monitoring. It also defines a common architectural structure upon which detector-specific processing firmware can be developed. A block diagram of the framework is shown in Fig. 4.

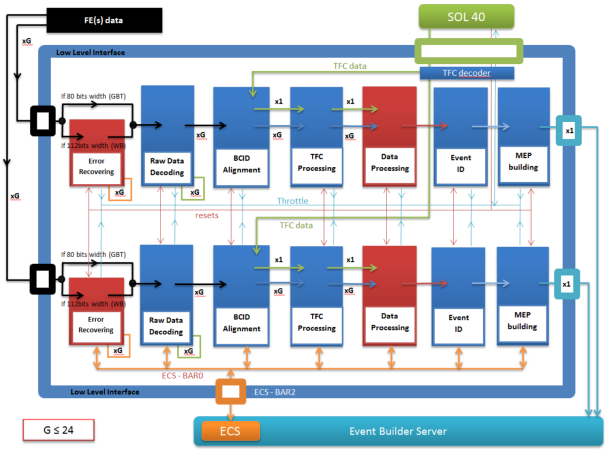


Fig. 4: LHCb common firmware block diagram

The work presented in this paper focuses on the UT-specific Data Processing block and the design challenges arising from the detector's heterogeneous front-end readout architecture.

II. Architecture Design and Preliminary Simulation

A. Input Data Format

The PCIe40 board provides two independent PCIe interfaces. Consequently, the firmware architecture is partitioned into two identical processing halves, each serving one PCIe endpoint. Apart from identifier assignments and hardware resources, both halves implement the same functionality. For clarity, the remainder of this paper describes the architecture of a single half; the complete firmware consists of two identical instances operating in parallel.

Each firmware half supports several configuration *flavours*, corresponding to the different GBT frame layouts present in the UT detector. The flavours are designated by the number of SALT ASICs carried in a frame and the number of e-links per ASIC: 4x3, 2x3, 2x4, and 2x5. In the standard configurations, a firmware half processes six optical links carrying the same frame type. The only exception is the Jx3 flavour, introduced to accommodate detector partitions containing a mixture of 4x3 and 2x3 links. In this configuration, the firmware processes two optical links carrying 4x3 frames and four optical links carrying 2x3 frames.

The common LHCb firmware framework abstracts the differences between GBT frame formats and presents the Data Processing block with a uniform interface consisting of one data stream per SALT ASIC. To accommodate the different data rates associated with the three-, four-, and five-e-link configurations, the width of these streams is adjusted accordingly, resulting in payload widths of 24, 32, and 40 bits, respectively. Since the amount of data produced by each ASIC varies from event to event, a complete event may span multiple clock cycles on a given stream.

Before reaching the Data Processing block, the incoming data are synchronized by the common framework. Each stream is associated with a 12-bit Bunch Crossing Identifier (BXID), corresponding to the LHC bunch crossing in which the data

were generated. The 12-bit BXID is obtained by extending the 4-bit identifier provided by the SALT ASICs. Following synchronization and BXID assignment, the SALT payload part of the packet is forwarded unmodified to the Data Processing block.

Figure 5 shows the packet formats delivered to the Data Processing block in the three-e-link configuration. The four- and five-e-link configurations follow the same format, differing only in the width of the payload field. Data payloads follow the structure described in [2]. A key point to remember is that SALT data packets, both headers and data payload, are always 12b wide, but because of the BXID extension made by the LHCb common framework, when data arrives to the Data Processing, the normal header is then 20b instead of 12b.

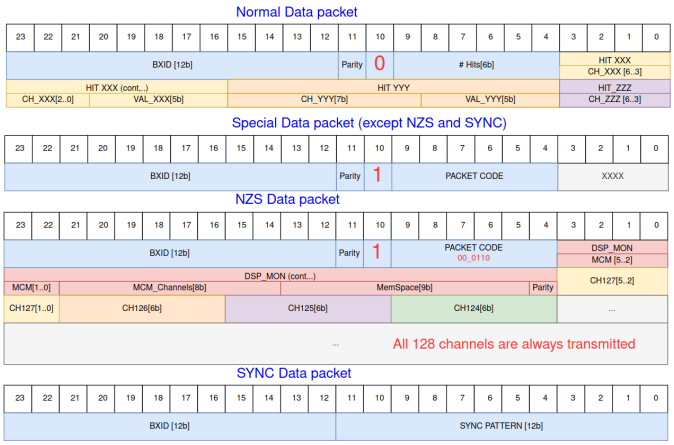


Fig. 5: Data Processing Input Data format for 3 elink ASICs

B. Output data format design

The LHCb framework [6] defines the interface between the Data Processing block and the downstream readout system. This interface consists of a 256-bit data bus at 250MHz, accompanied by a strobe signal, together with a metadata bus carrying packet descriptors, TFC information, and other control data. While the interface itself is common to all detectors, the organization of detector data within the output packet is left to the detector-specific firmware implementation.

A key aspect of the UT Data Processing design is the definition of an output data format capable of accommodating all possible combinations of SALT packet types and payload lengths. The format must simultaneously satisfy several requirements: being common to all flavours so this hardware complexity is not inherited by the next stages, efficient implementation within the FPGA resource constraints, straightforward decoding by the host software, high bandwidth utilization through minimal padding overhead, and robustness against unexpected detector behaviour or malformed input data. Reconciling these often competing requirements strongly influenced the architecture of the output data format.

Preliminary studies indicated that the most straightforward approach—concatenating the packets produced by all ASICs into a single output stream—was not practical. Each half of the firmware must process data from up to 24 independent sources, all producing packets with variable lengths. Determining the output position of every packet on an event-by-event basis would therefore require a large amount of dynamic bookkeeping and a deep hierarchy of multiplexing stages, while still maintaining deadtime-free operation.

Increasing the internal clock frequency can partially alleviate this problem by providing additional processing cycles per event. However, the high resource utilization of the design limits the achievable operating frequency of the Arria 10 FPGA to approximately 250 MHz. This increase is insufficient to compensate for the complexity introduced by simultaneously handling 24 variable-length data streams. As a result, a different output architecture was required.

To reduce the complexity associated with placing data from up to 24 independent sources into a common 256-bit output frame, the concept of *lanes* was introduced. Rather than allowing packets from any ASIC to occupy arbitrary positions within the output frame, the available bandwidth is partitioned into independent regions, each dedicated to a small group of ASICs. Specifically, a lane is restricted to carrying data from at most four ASICs.

This partitioning closely matches the detector readout structure, where a single optical link encapsulates data from either two or four SALT ASICs. By preserving this grouping throughout the processing chain, the amount of routing and arbitration logic required to assemble the output packets is significantly reduced, enabling a more scalable implementation while maintaining deadtime-free operation. Figure 6 shows an overview of the intended data format. The basic design is 6 lanes of 32b and one 64b header lane.

Hits arriving from the SALT ASICs are encoded as 12-bit words. While compact, this representation is not well suited for software processing on conventional CPUs. The Data Processing block therefore expands each hit to a 16-bit format by introducing four additional bits. As illustrated in Fig. 7, these bits are not used merely as padding; they are assigned such that the 7-bit channel identifier provided by the ASIC is transformed into an 11-bit Strip ID corresponding to the physical strip number within the sensor.

The lane organization further simplifies the event format. Since each lane is constrained to contain data from a single sensor, the ASIC identifier no longer needs to be transmitted explicitly. Hits can instead be uniquely identified by their Strip ID alone. Furthermore, all ASICs contributing to a lane are treated as a single readout group. Consequently, only the total number of hits in each lane must be recorded, rather than the contribution from every individual ASIC. This reduces the header information from up to 24 hit counters to only six, one per lane, significantly simplifying both the firmware implementation and the downstream software decoding.

Under normal operating conditions, each event begins with a single Event Header located in the header lane. This header contains the aggregated hit counts for all lanes, the event BXID, which is transmitted only once per event, and a 4-bit FLAGS field reserved for special conditions. Then all normal lanes are filled with the corresponding hits, so the packet size is determined by the busiest lane and it could be several lines long. All non-used bytes are padded with zeroes.

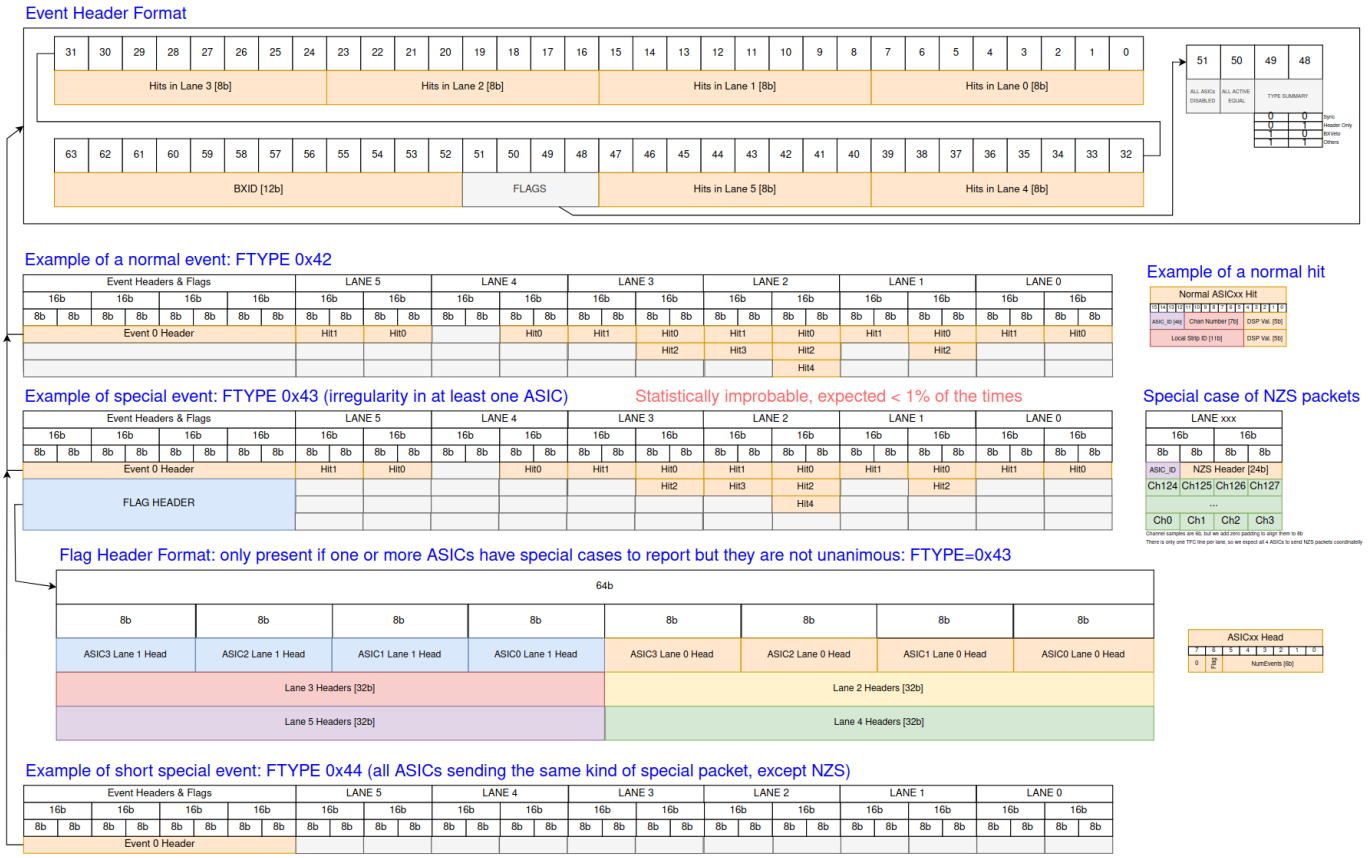


Fig. 6: Data Processing output data format

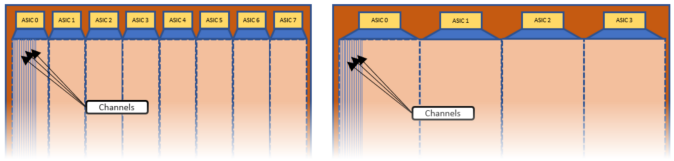


Fig. 7: ASIC positions with respect to strips

The Event Header is sufficient when all ASICs produce standard data packets. However, SALT ASICs may also generate special packet types, such as BXVeto or NZS packets, that require additional metadata. In these situations, the firmware emits a Flag Header. The Flag Header extends the Event Header with three additional 64-bit words containing a concatenation of the headers received from all ASICs participating in the event. This mechanism allows multiple ASICs to simultaneously report exceptional conditions without modifying the format of the hit data itself.

The additional information carried by the Flag Header comes at a significant bandwidth cost. Together with the Event Header, it occupies four lines, thus increasing the minimum packet size to 1kb. This overhead is acceptable only if exceptional packets occur infrequently. The architecture was designed under this assumption, and operational experience has confirmed that such conditions are indeed rare.

To further reduce the impact of common detector-wide conditions, the FLAGS field in the Event Header can directly encode selected exceptional states. These are known as ”Short Special” events. When all enabled ASICs report the same special packet type, such as a detector-wide BXVeto condition, the event can be represented using only the Event Header, avoiding the transmission of the larger Flag Header.

One packet type requires special handling: the Non-Zero-Suppressed (NZS) packet. Upon request, a SALT ASIC transmits the ADC value of every channel, regardless of whether a hit was detected. NZS packets are intended exclusively for detector monitoring, calibration, and debugging purposes, and are therefore generated only at very low rates.

Compared to standard zero-suppressed data, NZS packets are substantially larger and can temporarily consume a significant fraction of the available readout bandwidth. To avoid overflowing the front-end buffers or the data acquisition system, NZS acquisitions are followed by a configurable sequence of HeaderOnly packets. These packets contain no detector data and provide sufficient time for the readout chain to absorb the large NZS event before normal operation resumes.

Because of their unique characteristics, NZS events are represented using a dedicated output data format, also illustrated in Fig. 6.

C. Simulation-Based Validation of the Data Format

The output data format was developed and validated before the firmware implementation was available. To assess its suitability, a software model of the packet generation process was

developed in Python. The model used hits from LHCb Monte Carlo simulations as input and emulated the behaviour of the Data Processing block, producing output packets according to the proposed format.

The simulated hits from selected detector regions were mapped into lanes, assembled into events, and encoded using the same rules later implemented in firmware. By processing a large sample of simulated events, detailed statistics on packet sizes, header occupancy, and output bandwidth utilization were obtained.

These studies served two purposes. First, they verified that the proposed format could accommodate the expected detector occupancies while maintaining an acceptable overhead. Second, they provided quantitative estimates of packet-size distributions and bandwidth requirements, allowing potential bottlenecks to be identified and addressed before firmware development began.

Since the output data format is common to all firmware flavours, the software model itself does not distinguish between them. Nevertheless, the different detector regions are characterized by different occupancies and by different lane compositions, resulting in distinct output traffic patterns. In particular, the number of ASICs contributing to a lane varies between configurations, directly affecting the packet-size distributions and bandwidth utilization. To capture the most demanding operating conditions, simulations were performed using the worst-case detector regions of the central region of the detector associated with the three-, four-, and five-e-link configurations, highlighted in Fig. 8.

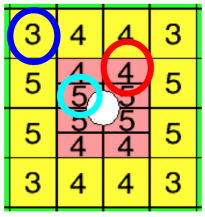


Fig. 8: Detector regions used for the simulation studies. The highlighted sensors correspond to the worst-case scenarios for the three-, four-, and five-e-link configurations.

The software model was used to characterize both the detector occupancy observed by the Data Processing block and the resulting output traffic generated by the proposed data format. Figure 9 summarizes the results for the worst-case regions associated with the three-, four-, and five-e-link configurations. The left column shows the distributions of the number of hits per lane, reflecting the occupancy patterns expected in each detector region. The right column presents the corresponding distributions of output packet size expressed in PCIe words. Together, these distributions provide a direct estimate of the bandwidth requirements imposed on the readout system and allow the overhead introduced by the event format to be quantified under realistic operating conditions.

The simulation results showed that the bandwidth requirements of the proposed format remained well within the capabilities of the PCIe40 platform. For all configurations studied,

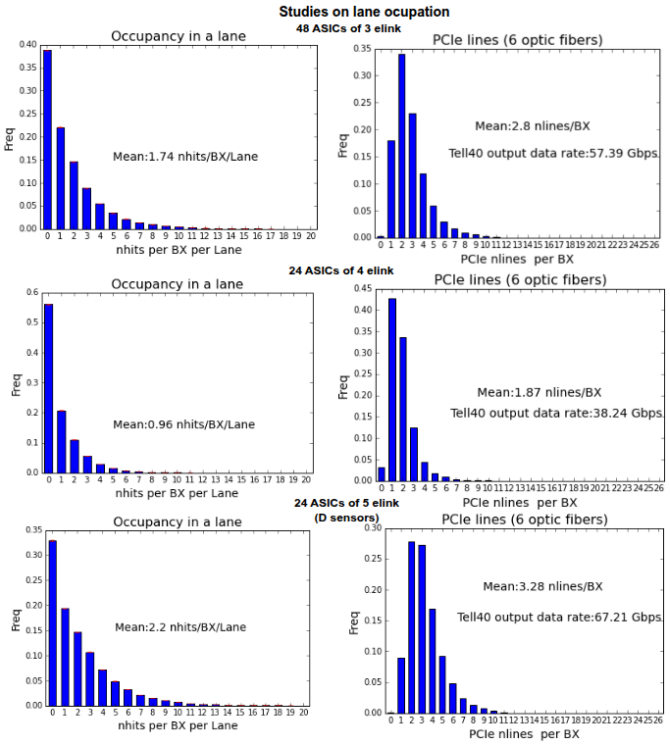


Fig. 9: Event hit rate and packet-size distributions obtained from Monte Carlo simulation for the worst-case detector regions associated with the three-, four-, and five-e-link configurations.

the worst case output bandwidth was around 50% of the 128 Gb/s bandwidth available through the PCIe interfaces. At first glance, this could suggest that additional detector channels might have been accommodated on each board. However, subsequent firmware implementation studies revealed that bandwidth was not the dominant design constraint. Instead, the limiting factor was the FPGA resource utilization associated with processing, buffering, and formatting the incoming data streams. When later implemented, the design occupied a substantial fraction of the available Arria 10 resources, leaving insufficient margin to support a significantly larger number of channels. The simulation studies therefore demonstrated that the proposed architecture provided comfortable bandwidth margins while remaining consistent with the practical resource limitations of the target hardware.

It should be noted that these results were obtained using Monte Carlo simulations performed before the detector was constructed and commissioned. At that stage, several factors influencing the final occupancy distributions and traffic patterns were still subject to uncertainty, including detector conditions, calibration parameters, and operational settings. Consequently, the simulations were regarded as a first-order estimate of the expected system behaviour rather than a precise prediction of the final bandwidth requirements. In this context, maintaining a substantial bandwidth margin was considered a desirable engineering safety factor, providing robustness against inaccuracies in the simulations and unforeseen changes in detector operation. Nevertheless, the studies proved in-

valuable for validating the architecture, identifying potential bottlenecks, and guiding the early stages of the firmware design.

III. Implementation

The Data Processing block is composed of a data path and an auxiliary Experimental Control System (ECS), as shown in Fig. 10. Depending on the firmware flavour, the data path receives either six streams corresponding to groups of four three-e-link ASICs, each carried on a 24-bit interface, or six streams corresponding to groups of two four- or five-e-link ASICs, carried on 32-bit and 40-bit interfaces, respectively. These streams are delivered through a dedicated 200 MHz interface provided by the common LHCb framework. To simplify timing closure across the FPGA, the Data Processing block operates in an independent clock domain and therefore does not directly process data at the incoming interface frequency.

The first stage of the processing chain is the Input Block. Six instances of this block are present in every firmware flavour, one per incoming stream. The Input Block receives and buffers the incoming data, performs the clock-domain crossing into the internal processing clock domain, extracts and stores event metadata in an Event FIFO, and translates the incoming stream format into a common internal representation shared by all firmware flavours. This abstraction allows the downstream processing stages to operate independently of the specific front-end configuration.

The next stage is formed by the Lane Builders, one per lane. Their function is to collect and merge the data belonging to a given lane and prepare it for output packet generation. By maintaining the lane abstraction introduced in the previous section, these blocks isolate the complexity associated with the different front-end readout configurations and occupancy patterns.

The final stage is the Output Block. This block assembles the contributions from all six lanes into the output event format, generates the appropriate event and flag headers, and buffers the resulting packets before transmission through the PCIe interface.

The ECS is distributed throughout the design, with dedicated monitoring and control logic associated with each processing stage. In addition to collecting status and error information, it provides occupancy measurements, event statistics, and diagnostic features such as register snapshots that can be triggered by exceptional operating conditions. These capabilities have proven essential during system integration, commissioning, and routine detector operation.

A. Input Blocks (IB)

The Input Block constitutes the entry point of the Data Processing chain. One instance is instantiated for each incoming stream provided by the common LHCb firmware framework, resulting in six Input Blocks per half of the design. Its primary purpose is to isolate the remainder of the Data Processing architecture from flavour-specific details providing a uniform interface to the downstream processing stages.

As described in the previous section, the framework presents detector data using different stream widths depending on the firmware flavour. Three-e-link configurations provide 24-bit payload streams corresponding to groups of four SALT ASICs, while four- and five-e-link configurations provide 32-bit and 40-bit streams corresponding to groups of two ASICs. Although they all carry the same underlying SALT information, the representation differs significantly as shown in Figure 11.

The Input Block translates the flavour-dependent input formats into a common internal representation that can be processed identically by the downstream Lane Builder and Output Block stages. As part of this translation, the hit format is expanded and aligned to 8-bit boundaries. The additional bits are not used solely as padding; they are configured through the ECS and encode the ASIC-specific offset required to transform the local 7-bit channel identifier into the corresponding sensor Strip ID, as described in the previous section.

The normalized output format of the Input Block is 64 bits wide. This width was chosen to accommodate up to four hits per clock cycle while providing sufficient bandwidth for all supported configurations. In particular, it exceeds the maximum hit throughput of the five-e-link configuration, whose 40-bit input interface can deliver at most 3.3 hits per cycle. Consequently, the common internal format introduces no bandwidth limitation while simplifying the design of the downstream processing stages.

The format conversion must be performed without introducing dead time and while sustaining the maximum input throughput of all supported firmware flavours. To achieve this, the decoder was implemented as a pipelined alignment network composed of several layers of registers. The registers are interconnected such that the incoming hit words are progressively shifted and rearranged as they propagate through the pipeline, eventually forming correctly aligned output words in the common internal format.

Figure 12 illustrates the implementation used for the three-e-link configuration. The decoder consists of three register stages arranged to produce a fully aligned output every second clock cycle. At the designated sampling points, indicated by the white arrows in the figure, the pipeline contains a complete set of correctly formatted hits that can be transferred to the next processing stage without requiring additional combinational logic or introducing stalls in the data path.

The number of alignment structures required depends on the firmware flavour. For standard zero-suppressed packets, the three-e-link configuration can be decoded using a single alignment pipeline. The four-e-link configuration requires two such structures operating in parallel, while the five-e-link configuration requires five independent alignment structures to account for the larger number of possible input alignments. Non-zero-suppressed (NZS) packets follow different data arrangements and therefore require dedicated decoding structures: one for the three-e-link flavour, two for the four-e-link flavour, and five for the five-e-link flavour.

Selecting the appropriate decoder output is performed by a dedicated state machine. Depending on the firmware flavour and packet type, the state machine tracks the progression of the incoming data stream and samples the corresponding align-

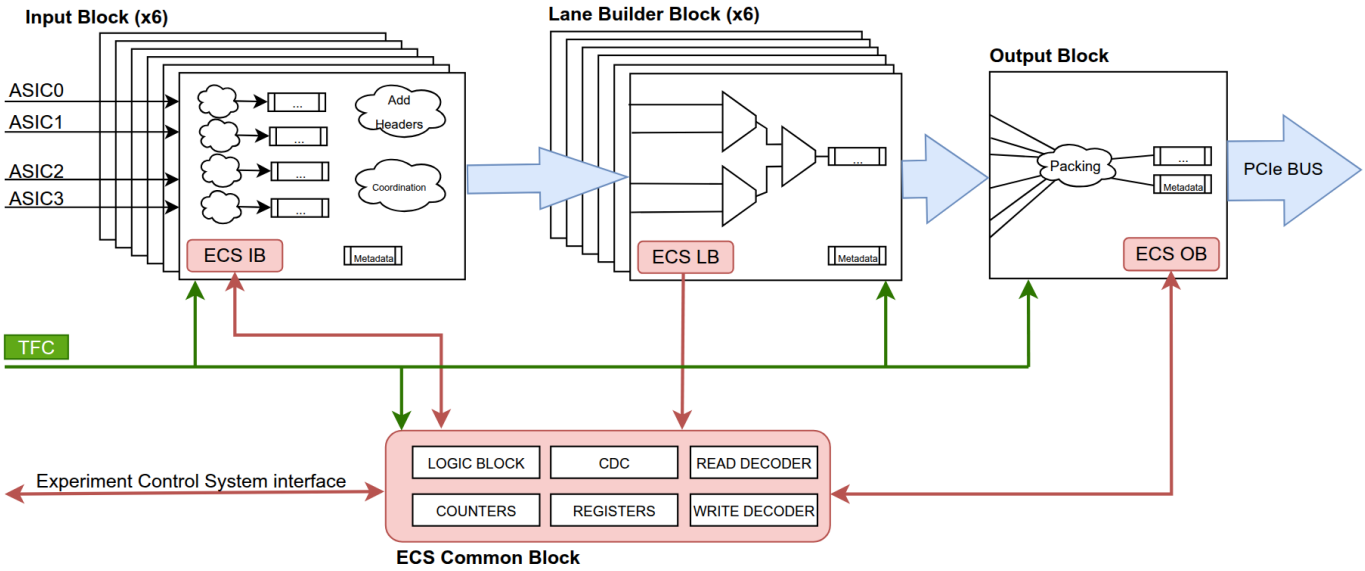


Fig. 10: Data Processing simplified diagram

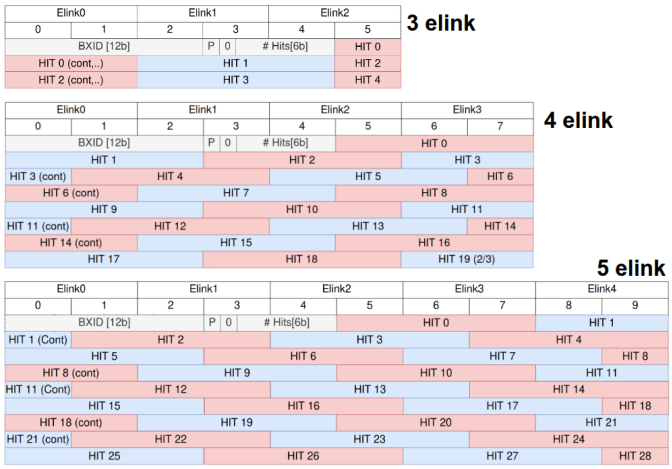


Fig. 11: Input Data formats as a function of flavour

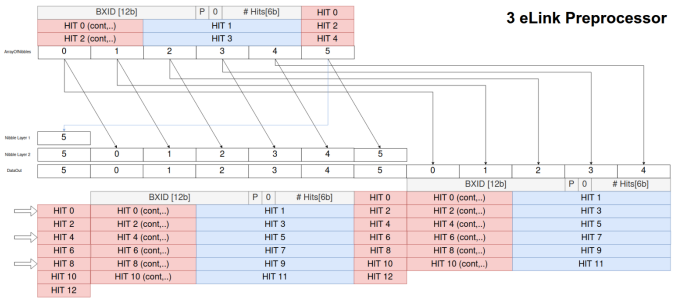


Fig. 12: 3 elink Preprocessor structure

ment structure at the appropriate clock cycles to reconstruct the correctly formatted hits. This approach allows the Input Block to sustain continuous operation without dead time while supporting the different alignment patterns imposed by the various front-end configurations.

Although all these variants are described by a single VHDL source file, the implementation is heavily parameterized. The required flavour is selected at synthesis time, and only the alignment structures, state machines, and associated logic relevant to that particular configuration are instantiated in the FPGA. As a result, the hardware implementation contains only the resources strictly necessary for the selected firmware flavour, avoiding the overhead that would result from supporting all possible configurations simultaneously.

In addition to format normalization, the Input Block performs several other essential functions. It buffers the incoming data streams, extracts and stores event metadata, and performs the clock-domain crossing between the 200 MHz framework interface and the internal Data Processing clock domain. The clock-domain crossing and buffering are implemented using dual-port FIFOs with independent read and write clocks. Incoming data are written into the FIFOs using the framework clock, while the downstream processing stages retrieve the normalized data using the internal Data Processing clock. This arrangement decouples the timing requirements of the common framework from those of the Data Processing block, allowing subsequent stages to consume data at their own pace, provided that the average processing rate is sufficient to prevent FIFO overflow.

By concentrating these functions in a dedicated front-end stage, the complexity associated with the various detector configurations is confined to a single module, considerably simplifying the design of the downstream processing chain.

B. Lane Builders (LB)

The Lane Builders constitute the intermediate stage between the flavour-dependent input processing and the final event assembly. Their primary function is to exploit the lane abstraction introduced in the output data format and merge the contributions from the different ASICs associated with a given lane into a single coherent event stream. By doing so, they isolate the Output Block from the complexity of the front-end organization and occupancy patterns.

Six Lane Builder instances are present in each half of the Data Processing chain, one per lane. Each instance receives the normalized hit streams produced by the Input Blocks together with the associated event metadata. Depending on the firmware flavour, a lane may aggregate data originating from either two or four SALT ASICs. Since the Input Blocks have already converted the local channel identifiers into Strip IDs, the Lane Builders can treat all incoming hits as belonging to a common sensor representation, independently of their ASIC of origin.

The main task of the Lane Builder is therefore event assembly. For each bunch crossing, it combines the hits associated with the ASICs contributing to the lane, determines the total number of hits to be reported for that lane, and forwards the resulting lane payload to the Output Block. This organization substantially reduces the amount of bookkeeping required by the downstream stages. Rather than handling the contributions of up to 24 independent ASICs, the Output Block only needs to process six lane-level entities, each represented by a single hit count and an associated sequence of Strip IDs and ADC values.

An important consequence of this approach is that the Lane Builders naturally absorb fluctuations in the occupancy of the individual ASICs. Since the number of hits generated by each ASIC varies from event to event, the amount of data contributing to a lane is not fixed. The Lane Builders therefore operate as event-driven merging stages, preserving the deadtime-free operation of the readout chain while presenting a uniform interface to the Output Block.

The merging operation itself is implemented recursively. Although a Lane Builder may receive contributions from up to four ASICs, it is not realized as a single four-to-one merger. Instead, the functionality is decomposed into a hierarchy of simpler building blocks referred to as *Mini Lane Builders*. Each Mini Lane Builder combines the event streams from two input sources into a single output stream while preserving event boundaries and associated metadata.

As illustrated in Fig. 10, three Mini Lane Builders arranged in two layers are sufficient to construct a complete Lane Builder. In the first layer, two Mini Lane Builders independently merge pairs of input streams. Their outputs are then forwarded to a third Mini Lane Builder, which produces the final lane-level event stream. In this way, four independent input sources are recursively reduced to a single output buffer.

Each Mini Lane Builder is connected to dedicated output buffering. Data are consumed from the output buffers of the preceding stage and, once merged, written into a local output buffer associated with the Mini Lane Builder itself. This distributed buffering strategy introduces elasticity into the processing chain, allowing temporary fluctuations in the occupancy of individual ASICs to be absorbed without immediately propagating backpressure throughout the hierarchy. As long as the average consumption rate exceeds the average production rate, the buffers accommodate variations in the instantaneous data rates while preserving deadtime-free operation.

This hierarchical approach offers several advantages. First, it decomposes the arbitration problem into a sequence of identical two-input operations, significantly reducing the combinational complexity compared to a monolithic four-input implementation. Second, the repeated use of the same Mini Lane Builder structure improves code reuse and simplifies verification, as only one merging primitive must be validated. Third, the distributed buffering naturally decouples the different stages of the hierarchy, improving robustness against occupancy fluctuations and easing timing closure. Finally, the recursive organization supports the different firmware flavours. Configurations in which a lane receives data from only two ASICs require only a single Mini Lane Builder, whereas the full two-layer hierarchy is instantiated only when four-way merging is required.

Each Mini Lane Builder operates on complete events and exploits the metadata generated by the Input Blocks to determine the amount of data that must be consumed from each input stream. Event fragments corresponding to the same bunch crossing are merged and written into the output buffer, which is subsequently consumed by the next stage of the hierarchy or, in the case of the final Mini Lane Builder, by the Output Block.

C. Output Block (OB)

The Output Block constitutes the final stage of the Data Processing chain. Its primary responsibility is to assemble the lane-level event fragments into complete PCIe frames following the output data format described in the previous sections. In addition to event formatting, it provides the interface between the internal Data Processing logic and the downstream LHCb common framework, isolating their respective timing domains and buffering the generated events until they are consumed by the PCIe infrastructure.

The Output Block receives one event stream from each of the six Lane Builders. These streams contain the fully assembled lane payloads together with the corresponding metadata, including the number of hits associated with each lane. The task of the Output Block is therefore no longer to interpret detector data, but rather to orchestrate their placement into the final event structure while respecting the constraints imposed by the PCIe interface and the LHCb framework.

The implementation is organized around six *DumpFIFO* blocks, one associated with each lane. Their function is to sequentially extract the lane payloads from the Lane Builder output buffers and present them to the event assembly logic. Since the occupancy of each lane varies from event to event, the amount of data contributed by each DumpFIFO is inherently variable. Complementing these blocks, six *LanePadding* units generate the padding required by the output data format whenever the payload associated with a lane does not naturally align with the boundaries imposed by the PCIe frame organization.

A central control unit orchestrates the operation of these building blocks. Based on the lane metadata and the event characteristics, it determines the sequence in which data are transferred into the output frame, computes the amount of padding required for each lane, and identifies the appropriate header format to be generated. In particular, this logic decides whether a standard Event Header is sufficient or whether a Flag Header must be emitted to accommodate exceptional

packet types originating from the front-end ASICs. It is also responsible for generating the auxiliary information required by the LHCb common framework, including packet sizes, frame type identifiers, and other metadata associated with the PCIe transmission.

Once assembled, the event information is written into two dedicated output buffers implemented using dual-port FIFOs with independent clock domains. The first buffer stores the 256-bit PCIe frame payload, while the second stores the corresponding metadata required by the common framework. These buffers decouple the timing requirements of the Data Processing block from those of the downstream PCIe infrastructure, allowing the common framework to retrieve complete events at its own pace while preserving deadtime-free operation, provided that the average consumption rate exceeds the average event production rate.

By concentrating the complexity associated with header generation, padding insertion, framework signaling, and PCIe interfacing into a single module, the Output Block presents the remainder of the system with a simple abstraction: six lane-level event streams are transformed into fully compliant PCIe events ready for transmission to the Event Builder farm.

D. Experimental Control System (ECS)

The Experimental Control System (ECS) provides the interface between the LHCb control software, based on WinCC-OA[1], and the hardware components of the experiment. It is responsible for configuring firmware parameters, collecting monitoring information, and providing a centralized view of the operational status of the different processing blocks. Through the ECS, detector operators can monitor the health of the system, identify abnormal conditions, and access detailed diagnostic information during operation.

At the firmware level, the ECS associated with the Data Processing (DP) system is organized into a set of dedicated blocks distributed throughout the processing chain. A common ECS block gathers status information from the Input Buffer (IB), Lane Builder(LB), and Output Buffer (OB) modules and makes it available to the control system. This information includes error flags, status registers, and monitoring counters that allow the state of the different modules to be tracked in real time. The common ECS block also implements counters for event categories that are shared across all processing stages, providing a coherent view of the overall system activity.

In addition to status monitoring, the ECS provides access to a number of configuration and diagnostic signals. One example is the *snapshot* feature, which captures the contents of a predefined set of registers whenever a buffer reaches full occupancy, an error condition is detected, or a Timing and Fast Control (TFC) trigger is received. This mechanism provides a snapshot of the internal state of the system at the time of the event and has proven particularly useful for investigating unexpected behaviours and validating firmware functionality. Furthermore, a number of additional monitoring signals can be selectively enabled to provide increased visibility into the operation of specific processing stages. These observables have been extensively used during system integration and testing to characterize the behaviour of the DP chain under different operating conditions.

Each IB, LB, and OB module contains a dedicated ECS block that monitors the occupancy of the internal FIFOs and reports both average and peak values to the control system. This functionality requires dedicated counters and accumulation logic to continuously collect and process occupancy information. For each lane, the implementation includes 12 48-bit counters, 63 32-bit counters, and 14 accumulation blocks. Given the significant amount of arithmetic required, the accumulation functions were implemented using FPGA DSP resources in order to reduce Adaptive Logic Module (ALM) utilization. This optimization proved essential for meeting the resource constraints of the design. Without the use of DSP blocks, ALM utilization can approach 97 per cent, making it unfeasible to meet timing closure requirements. The counter values reported by the ECS can be cross-checked against registers available in the ASICs and SOL40 boards, which independently store the number of transmitted and received TFC commands. This provides an additional consistency check for the monitoring infrastructure.

Since the ECS operates at 40 MHz while the DP logic runs at its own clock, dedicated clock-domain crossing (CDC) blocks are put in place.

An existing LHCb common software framework is used to import the monitored quantities into the experiment control system. The information is then displayed through dedicated WinCC-OA panels specifically developed for the DP system. These panels provide both a high-level overview of the system status and access to detailed monitoring information, enabling efficient operation and rapid diagnosis of potential issues.

The ECS monitoring panel implemented in WinCC-OA is shown in Fig 13.

E. Back-pressure mechanism

A fundamental design principle of the Data Processing chain is the extensive use of distributed buffering to decouple the different processing stages. Rather than implementing the firmware as a tightly synchronized pipeline in which all blocks are required to operate in lockstep, the architecture adopts a data-driven approach in which each stage behaves independently as both a data consumer and a data producer. Data are exchanged exclusively through queues, allowing each block to request new events only when sufficient processing and buffering resources are available.

This organization provides several advantages. First, it isolates the internal implementation details of the different stages. The Input Blocks, Mini Lane Builders, Lane Builders, and Output Block can each process events at their own instantaneous rate without requiring precise coordination with the rest of the system. Second, the queue-based interfaces naturally absorb short-term fluctuations in occupancy arising from the stochastic nature of the detector data. Finally, the architecture promotes modularity, since each processing stage only depends on a well-defined interface consisting of input

[1]SIMATIC WinCC Open Architecture, a commercial supervisory control and data acquisition (SCADA) software developed by Siemens

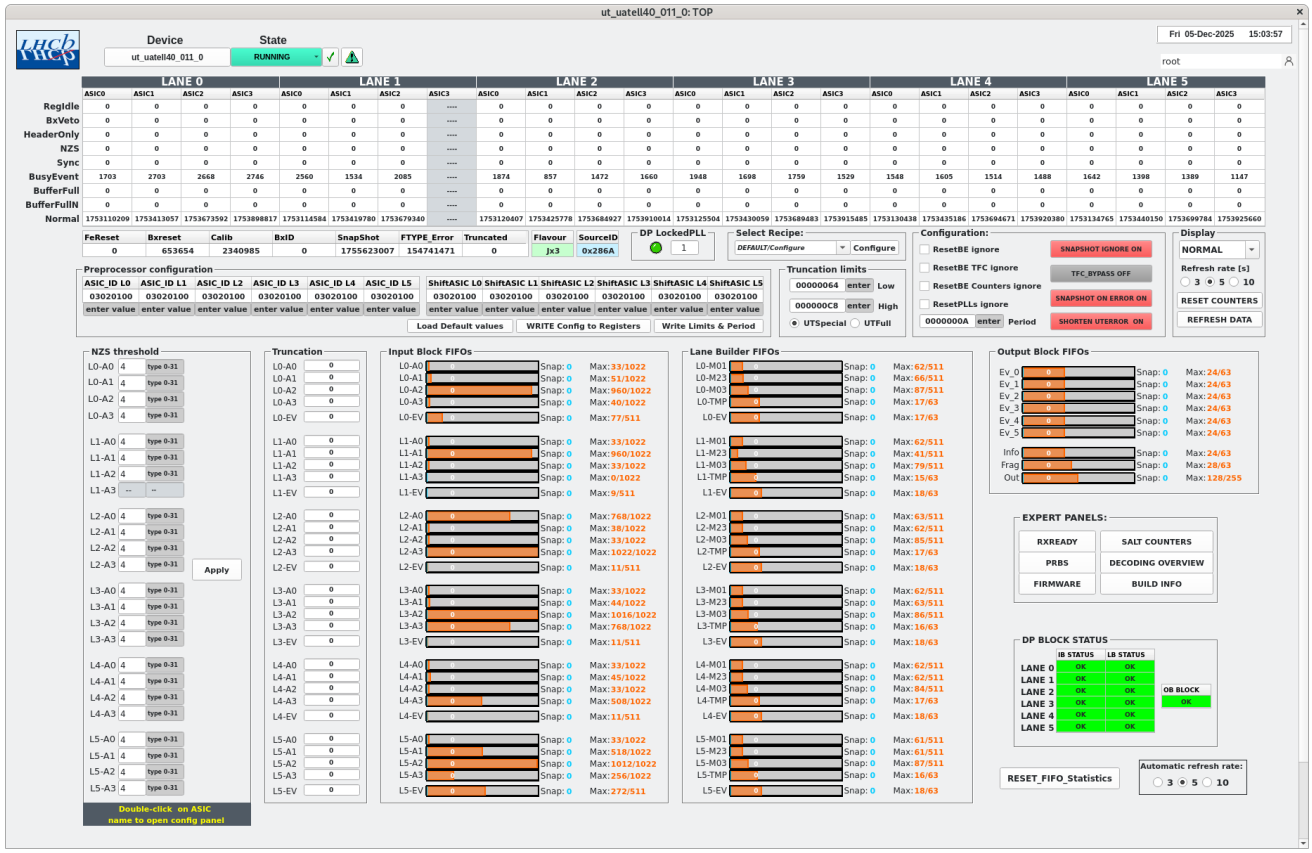


Fig. 13: Data processing overview showing the status, event counters and FIFO occupancy in the different data processing blocks in a WinCC-OA panel as displayed in the control system.

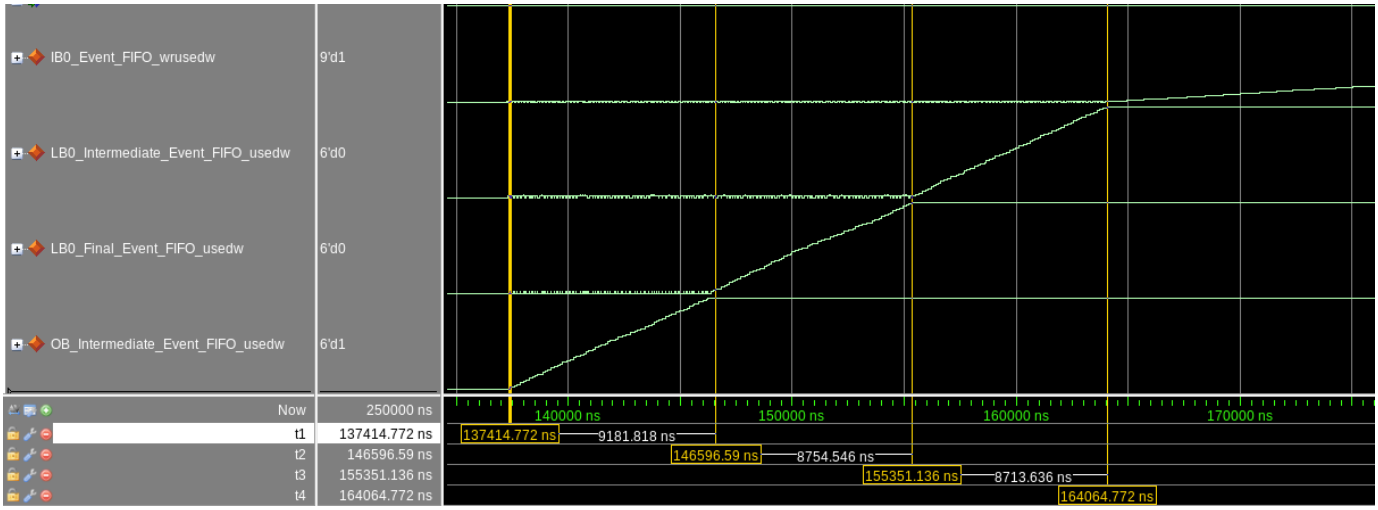


Fig. 14: Simulation of the back-pressure mechanism under sustained overload conditions. As events are generated faster than they can be processed, the buffers of the downstream stages progressively fill. Once a buffer reaches capacity, the corresponding stage temporarily ceases requesting additional events from the preceding stage. This causes occupancy to build up recursively in the upstream buffers, eventually propagating back to the Input Block FIFOs. The figure illustrates the elastic behaviour of the Data Processing architecture and the controlled propagation of back-pressure through its hierarchical structure.

queues, output queues, and a request mechanism controlling the transfer of complete events.

The flow of events through the Data Processing chain is therefore governed by demand rather than by a global schedule. A processing stage consumes an event from its upstream queue only when it has sufficient resources to process it and sufficient buffer space to store the result. Once processing is complete, the generated event becomes available in its output queue for downstream consumption. This producer–consumer model propagates naturally through the hierarchy, from the Input Block FIFOs to the recursive Mini Lane Builders and ultimately to the Output Block and PCIe interface.

An inevitable consequence of this approach is the emergence of a backpressure mechanism under sustained overload conditions. When the output queue of a processing stage reaches its capacity, the stage temporarily suspends requests for new events from the preceding stage. The upstream stage then continues to accumulate events in its own buffers until sufficient capacity becomes available downstream. In this way, occupancy increases propagate upstream in a controlled manner while preserving event integrity. Figure 14 illustrates this behaviour for an artificially overloaded system.

As a final protection mechanism against prolonged overload conditions, the occupancy of the Input Block FIFOs is monitored. If an upper configurable threshold, ε_H, is exceeded, the system temporarily enters a truncation mode in which hit information is discarded and the affected events are explicitly flagged. A hysteresis mechanism is implemented by requiring the occupancy to fall below a lower threshold, ε_L, before normal operation resumes.

IV. Simulation, debugging and verification

The validation of the UT firmware, encompassing both compilation/simulation and hardware verification stages, is carried out through a comprehensive workflow involving the use of Quartus and Questa for firmware synthesis and simulation, and the SignalTap Logic Analyzer for on-board diagnostics. Dedicated decoder modules have been implemented at each stage of the firmware processing chain to facilitate detailed inspection and verification of internal data structures and signal integrity. These decoders are complemented by a suite of analysis and debugging tools, as well as the preparation of specifically tailored data injection files.

To further support the validation process, a dedicated emulator has been developed to reproduce the passage of data through the successive processing layers of the firmware. This emulator enables precise comparison between the decoded outputs and the corresponding expected values obtained from reference models. Automated verification scripts are employed to systematically perform these comparisons, flagging any discrepancies that may arise. Identified inconsistencies are logged and analyzed to trace their origin, thereby contributing to the iterative refinement of the firmware design and its compliance with the functional and performance requirements.

In parallel, a detailed emulator for the UT SALT and GBT data links has been implemented, based on the Boole framework—the official digitization software of the LHCb experiment. This emulator generates realistic input stimuli for firmware testing by processing Monte Carlo (MC) hit data through the full UT digitization chain to produce emulated GBT output packets. However, as standard MC simulations only include nominal (i.e., correctly formatted) data packets, an additional tool was developed to extend the range of test conditions. This auxiliary software allows the modification or creation of custom injection files using empirical or parameterized models, as well as the direct use of real data extracted from MDF (Multi Data Format) files recorded by the experiment. This capability enables the inclusion of atypical or corrupted data frames—such as special or error packets—derived from both simulated and real detector conditions. These extended datasets are instrumental in testing the robustness of the firmware under non-ideal operating scenarios, including high-occupancy environments and data rate saturation, thereby ensuring reliability and resilience during final detector operation.

V. Software

The development of software for a fully installed detector in the LHCb cavern necessitated the creation of several custom systems capable of efficiently decoding and analysing data from the entire detector. The principal objective of this software is to provide a comprehensive suite of tools for front-end calibration, spatial and temporal detector alignment, the generation of simulation parameters, online data monitoring, and more. In particular, calibration tools proved invaluable during the hardware and firmware commissioning phases of the Upstream Tracker (UT).

The core software was developed within the Gaudi framework, the standard event-processing architecture employed by the LHCb experiment at CERN. Gaudi provides a modular and extensible environment, robust data management, and seamless integration with the existing LHCb analysis tools and workflows—features that are essential for scalable detector operations and collaborative software development. Notably, this project builds upon code originally developed in the TbUT package [8], which was designed for test beam studies of the LHCb Upstream Tracker hybrids and utilised the Kepler project. Kepler is itself a Gaudi-based application created for LHCb test beam and test stand data analysis, facilitating the analysis of telescope data and providing functionalities such as track fitting, alignment, and charge calibration.

All algorithms from TbUT have been adapted to the Vetra project to meet the evolving software requirements for LHCb Run 3 and Run 4. The Vetra project encompasses the following aims:

- Analogue calibration and monitoring: precise baseline calibration using an 8-bit trim DAC register.
- Digital calibration and monitoring: complementary baseline calibration employing a 6-bit pedestal register.
- Calculation and monitoring of common mode, a noise component coherent for a group of channels belonging to a particular SALT chip and related to low voltage oscillations effects and others (Fig.15).
- Hit threshold determination based on common mode subtracted (CMS, incoherent) noise levels.

- Identification of noisy and faulty channels.

Data analysed by Vetra are taken during the special calibration run (outside physics data taking during no-beam time at LHC).

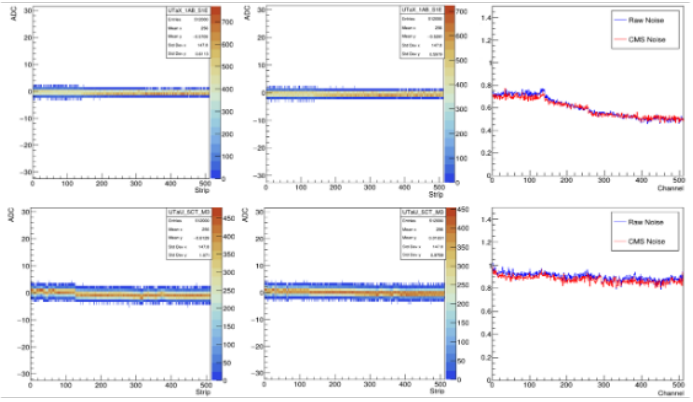


Fig. 15: (a) Distribution of ADC vs. strip number for data taken before pedestal subtraction and CM suppression. (b) Distribution of ADC vs. strip number for data taken after pedestal subtraction and CM suppression. (c) Total- and common-mode–subtracted noise vs. strip number (example for A-type sensor). (d) Corresponding plots for a D-type sensor with beam-pipe cutout, following the same processing steps.

VI. Conclusions

The firmware presented in this work implements the Data Processing block of the LHCb Upstream Tracker back-end readout system. Its role is to transform the heterogeneous data streams produced by the front-end electronics into a common output format suitable for transmission through the PCIe40 infrastructure and subsequent event building. The coexistence of multiple SALT configurations, the variable-length nature of the detector data, and the requirement for deadtime-free operation imposed a number of architectural challenges that shaped the final design.

To address these constraints, the firmware adopts a hierarchical and data-driven architecture. flavour-dependent processing is confined to the Input Blocks, while the recursive Lane Builder structure progressively aggregates detector data into lane-level entities before final event assembly in the Output Block. The extensive use of distributed buffering and producer–consumer interfaces decouples the different processing stages, allowing them to absorb occupancy fluctuations and operate independently while maintaining a controlled backpressure mechanism.

The proposed output data format and processing architecture were validated through software emulation using LHCb Monte Carlo simulations prior to firmware development. These studies demonstrated that the expected bandwidth requirements remained comfortably within the capabilities of the PCIe40 platform, while also guiding several architectural decisions. Subsequent firmware implementation confirmed that FPGA resource utilization, rather than PCIe bandwidth, constituted the dominant design constraint. The available bandwidth margin therefore provided a valuable engineering safety factor against uncertainties in the detector simulations and future operational conditions.

An additional lesson emerging from this work is the importance of considering firmware complexity during the early stages of detector and front-end design. The use of sophisticated data formats featuring multiple operating modes, variable-length packets, exceptional packet types, and detector-specific optimizations can significantly increase the complexity of the downstream processing logic. Although such mechanisms may improve front-end efficiency, reduce material within the detector acceptance, or optimize bandwidth utilization, they often translate into additional buffering requirements, control logic, arbitration mechanisms, and verification effort in the back-end firmware. In the present case, implementation studies showed that FPGA resource availability, rather than external bandwidth, ultimately limited the number of detector channels that could be accommodated on a single PCIe40 board. Future systems would therefore benefit from evaluating these trade-offs from an end-to-end perspective, considering the impact of front-end protocol choices on the scalability, complexity, and maintainability of the readout architecture.

The firmware has been successfully implemented on the PCIe40 platform and deployed as part of the LHCb Upstream Tracker readout system. Operational experience has confirmed the robustness of the proposed architecture and validated the design choices adopted throughout the development process. The concepts presented in this work, including recursive event aggregation, distributed buffering, and queue-based flow control, may also prove valuable in the design of future high-throughput readout systems facing similar constraints.

References

[1] L. Collaboration, "LHCb Tracker Upgrade Technical Design Report," CERN, Tech. Rep., 2014. [Online]. Available: https://cds.cern.ch/record/1647400

[2] C. Abellan Beteta, D. Andreou, M. Artuso, A. Beiter, S. Blusk, R. Bugiel, S. Bugiel, A. Carbone, I. Carli, B. Chen *et al.*, "The salt—readout asic for silicon strip sensors of upstream tracker in the upgraded lhcb experiment," *Sensors*, vol. 22, no. 1, 2022. [Online]. Available: https://doi.org/10.3390/s22010107

[3] S. Bonacini, K. Kloukinas, and P. Moreira, "E-link: A radiation-hard low-power electrical link for chip-to-chip communication," *Semantic Scholar*, accessed: 2025-09-26. [Online]. Available: https://api.semanticscholar.org/CorpusID:6354380

[4] I. Corporation, "Arria 10 fpgas," accessed: 2025-09-26. [Online]. Available: https://www.intel.com/content/www/us/en/products/details/fpga/arria/10.html

[5] M. Bellato, G. Collazuol, I. D'Antone, P. Durante, D. Galli, B. Jost, I. Lax, G. Liu, U. Marconi, N. Neufeld, R. Schwemmer, and V. Vagnoni, "A pcie gen3 based readout for the lhcb upgrade," *Journal of Physics: Conference Series*, vol. 513, no. 1, p. 012023, jun 2014. [Online]. Available: https://dx.doi.org/10.1088/1742-6596/513/1/012023

[6] G. Vouters *et al.*, "Front-end and back-end data format of the lhcb upgrade," CERN, LHCb Technical Note EDMS 2114571, 2019. [Online]. Available: https://edms.cern.ch/document/2114571/1

[7] G. Vouters, "Tell40 error handling," CERN, Tech. Rep. EDMS 2114573, 2022. [Online]. Available: https://edms.cern.ch/document/2114573/2.1

[8] A. M. Dendek, "Machine learning based long-lived particle reconstruction algorithm for run 2 and upgrade lhcb trigger and a flexible software platform for the ut detector read-out chip emulation," Jun. 2021.