A FPGA-Based Architecture for Real-Time Cluster Finding in the LHCb Silicon Pixel Detector

G. Bassi, L. Giambastiani, K. Hennessy, F. Lazzari, M. J. Morello, T. Pajero, A. Fernandez Prieto, and G. Punzi

***Abstract*— This article describes a custom [very high speed integrated circuits (VHSIC) hardware description language (VHDL)] firmware implementation of a 2-D cluster-finder architecture for reconstructing hit positions in the new vertex pixel detector that is part of the large hadron collider beauty (LHCb) Upgrade. This firmware has been deployed to the existing field-programmable gate-array (FPGA) cards that perform the readout of the VErtex LOcator detector (VELO), as a further enhancement of the data-acquisition (DAQ) system, and will run in real-time during physics data taking, reconstructing VELO hits coordinates on-the-fly at the LHC collision rate. This preprocessing allows the first level of the software trigger to accept an 11% higher rate of events, as the ready-made hit coordinates accelerate the track reconstruction and consume significantly less electrical power. It additionally allows the raw pixel data to be dropped at the readout level, thus saving approximately 14% of the DAQ bandwidth. Detailed simulation studies have shown that the use of this real-time cluster-finding does not introduce any appreciable degradation in the tracking performance in comparison to a full-fledged software implementation. This work is part of a wider effort aimed at boosting the real-time processing capability of HEP experiments by delegating intensive tasks to dedicated computing accelerators deployed at the earliest stages of the data acquisition chain.**

***Index Terms*— Clustering, connected-component labeling (CCL), FPGA, large hadron collider beauty (LHCb), [very high speed integrated circuits (VHSIC) hardware description language (VHDL)].**

Manuscript received 7 February 2023; revised 29 March 2023; accepted 14 April 2023. Date of publication 8 May 2023; date of current version 16 June 2023.

G. Bassi and M. J. Morello are with INFN Sezione di Pisa, 56127 Pisa, Italy, and also with the Classe di Scienze, Scuola Normale Superiore, 56126 Pisa, Italy (e-mail: giovanni.bassi@cern.ch).

L. Giambastiani was with INFN Sezione di Pisa, 56127 Pisa, Italy, and also with Università di Pisa, 56126 Pisa, Italy. He is now with the Dipartimento di Fisica e Astronomia "Galileo Galilei", Università degli Studi di Padova, 35122 Padua, Italy.

K. Hennessy is with the Department of Physics, Liverpool University, L69 7ZE Liverpool, U.K.

F. Lazzari was with INFN Sezione di Pisa, 56127 Pisa, Italy, and also with the Dipartimento di Scienze Fisiche, della Terra e dell'Ambiente, Università degli Studi di Siena, 53100 Siena, Italy. He is now with INFN Sezione di Pisa, 56127 Pisa, Italy, and also with the Dipartimento di Fisica "Enrico Fermi", Università di Pisa, 56126 Pisa, Italy.

T. Pajero was with INFN Sezione di Pisa, 56127 Pisa, Italy, and also with Scuola Normale Superiore, 56126 Pisa, Italy. He is now with the Department of Physics, University of Oxford, OX1 3RH Oxford, U.K.

A. Fernandez Prieto is with the Instituto Galego de Física de Altas Enerxías (IGFAE), Universidade de Santiago de Compostela, E-15782 Santiago de Compostela, Spain.

G. Punzi is with INFN Sezione di Pisa, 56127 Pisa, Italy, and also with the Dipartimento di Fisica "Enrico Fermi", Università di Pisa, 56126 Pisa, Italy.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNS.2023.3273600

Digital Object Identifier 10.1109/TNS.2023.3273600

I. Introduction

The large hadron collider beauty (LHCb) experiment has collected data over the past decade, during Run 1 and Run 2 of the LHC, and recently underwent a major update for the current Run 3. In addition to replacing most of the subdetectors, the front-end electronics and data-acquisition system were completely renewed [1], to read out and process the complete information of the detector at the full LHC beam crossing rate of 40 MHz (30 MHz averaged over the LHC cycle). This change is motivated by the needs of the LHCb physics program, which requires the collection of low transverse momentum events that need high-level processing to be distinguished from background events [2]. This evolution puts a large computing toll on the new real-time processing system, motivating the deployment of innovative features, with a general trend of increasing customization, parallelization, and early data preprocessing. A new trigger system [1], [3] was designed to allow the experiment to collect data effectively at an instantaneous luminosity of 2×10^{33} $\mathrm{cm}^{-2}\mathrm{s}^{-1}$, five times higher than during Run 2, corresponding to a bandwidth of about 32 Tb/s. The subsequent event-building stage and software high-level-trigger (HLT) processing lead to a data storage flow of 80 Gb/s.

The triggering process is divided into two main stages, named HLT1 and HLT2. HLT1 uses an array of GPU servers to perform a faster event reconstruction, with the only purpose of reducing the event rate, while retaining as much signal as possible, to a level acceptable for HLT2. HLT2, based on an array of CPU servers, performs a complete reconstruction of events with an offline-level quality, which is permanently stored for subsequent analysis. To perform its function effectively, HLT1 needs to perform a nearly complete event reconstruction. First, it finds track segments in the VErtex LOcator detector (VELO), attaching to them hits from the further tracking stations upstream and downstream of the magnet to obtain complete tracks; then, the positions of the primary vertices of the proton–proton (pp) collisions are found, as well as those of displaced vertices that constitute the main signature of heavy-flavor particle decays.

The feasibility of implementing several parts of this sequence in a specialized architecture, using programmable digital electronics co-processors (FPGAs), has been studied with the aim of achieving a faster and cheaper reconstruction, especially in view of future runs, moving parts of it before the event-building stage [4].

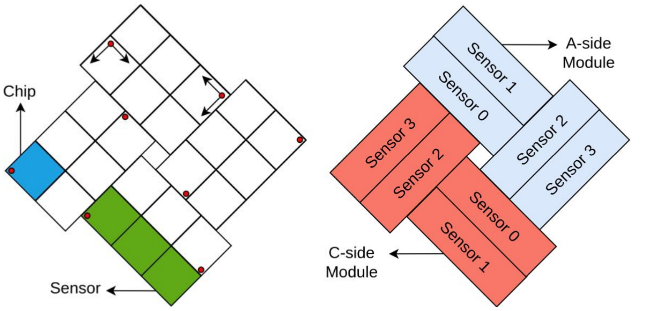

Fig. 1. Illustration of the basic constituents of a VELO layer [5]. Red dots mark the origin, pixel (0,0), of the local Cartesian coordinate system of each sensor (see Section III). As an example, the axis orientation is displayed in both sensors of the upper module.

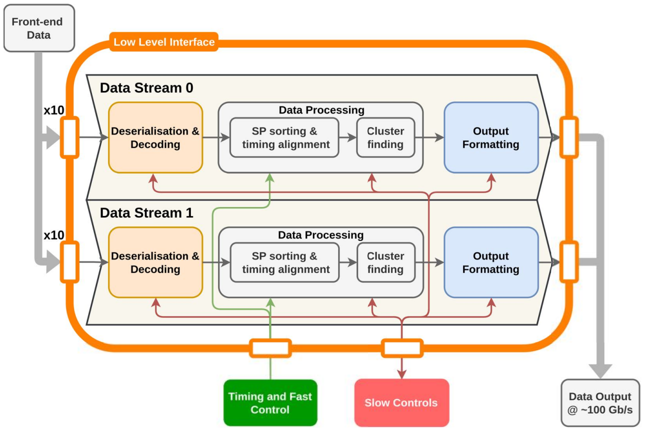

Fig. 2. Schematics of the TELL40 firmware. A detailed description can be found in [8].

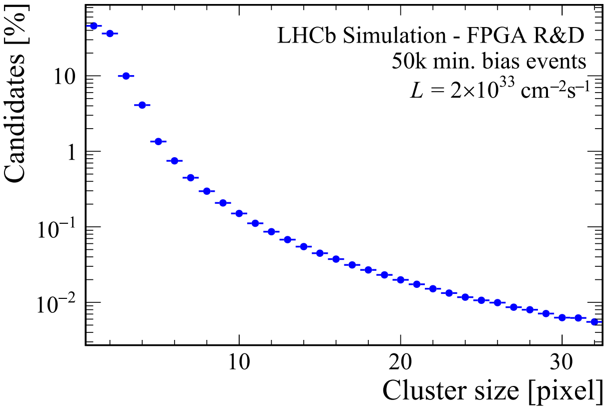

Fig. 3. Distribution of the number of pixels per cluster.

In this article, we address the very first step in LT1 event reconstruction, that is the search for clusters of active pixels in the VELO [5]. Grouping contiguous pixels in clusters is a conceptually simple but computationally demanding task, due to the 2-D geometry and the large number of pixels of the VELO detector (approximately 40 million). In the preliminary version of HLT1, designed to run entirely on CPUs, this task alone consumed 17% of the time required by the complete HLT1 reconstruction sequence. We address here this issue and describe an efficient architecture of this functionality, requiring a very modest amount of FPGA resources, while providing the throughput and the performance required for its use within the LHCb DAQ system. The core ideas underlying the design of this architecture are based on studies of an FPGA-based track-finding system, performed within the INFN-RETINA Research and Development Project [4]. The overall structure of our algorithm and its main building blocks are rather general and can be applied to any pixel detector. A baseline version is available for download from a public code repository [6]. However, the LHCb version contains specific features tailored to the VELO detector.

II. Format and Features of VELO Data

The Run 3 VELO is a silicon pixel detector consisting of 26 layers both downstream (19) and upstream (seven) of the nominal point of pp collisions. Each layer consists of two modules, each read by a dedicated readout card. A module is made of four sensors, each of which is bump-bonded to three VeloPix [7] ASICs (chips), as shown in Fig. 1. The VELO front-end data arrive at the LHCb readout cards via optical links as aggregated groups of 4 × 2 pixels, named SuperPixels (SPs), with binary response. The data are deserialized, decoded, and sent to the data processing stage. SPs are output by the detector without a well-defined time ordering, and data from different LHC beam crossings (separated by 25 ns) may not be synchronized and mixed over time. The first step of the VELO data-processing firmware reorders the SPs, making sure that SPs coming from the same proton bunch crossing (event) are grouped together [8], before data are sent to the clustering stage. Reconstructed clusters are then formatted into LHCb event fragments and sent to the PCIe bus. Fig. 2 shows a schematic view of the firmware [8] of the custom PCIe cards (TELL40 [9]) that perform the readout of the VELO. TELL40 cards are used as readout units for each subdetector within LHCb. Each TELL40 card carries an Altera Arria-10 GX 1150 FPGA with 1150k logic elements.

The clustering firmware was designed to take as its input the list of all active SPs found in a given event and to produce a list of reconstructed clusters, each with the local (x, y) coordinates of its centroid. In addition, it provides the detailed shape of the pixel cluster, as well as some flags indicating cluster quality. These additional quantities are not required by HLT1 reconstruction, but are computed to allow HLT2 to perform a fully optimized reconstruction of tracks, despite the lack of the original raw pixel data.

The size of clusters generated by individual charged particles crossing the VELO layers is less than or equal to 4 pixels in 96% of the cases, whereas larger clusters are mostly the product of merged hits or secondary emissions (δ-rays, etc.). The distribution of cluster sizes as predicted by the LHCb Upgrade simulation (MC) [10] is shown in Fig. 3.

III. Core Architecture

The distribution in Fig. 3 implies that clusters produced by a single particle hit are often contained within a single SP word. In those cases, the reconstruction of the cluster can be performed through a lookup table (LUT), and it is therefore advantageous to perform an initial preprocessing of SPs to separate these occurrences from the others and to send them to two distinct parallel processing blocks. The separation is performed by comparing the 2-D position of each SP with that of all other SPs of the same sensor in the same event. Each SP is then flagged as “isolated” if none of its eight SP neighbors has any active pixels. The LHCb simulation predicts that isolated SPs will account for 53% of the total number of SPs at nominal Run 3 luminosity conditions.

The centroid of clusters within isolated SPs is calculated directly by means of an LUT. Each of the 2^8 possible pixel configurations within an SP is linked to the precalculated center of mass of the corresponding reconstructed cluster(s). This LUT-based reconstruction allows an extremely fast processing of isolated SPs, with a very limited amount of logic resources. It is possible for up to two distinct clusters to be present within a single SP. The firmware correctly handles this case as well, generating two independent clusters in the output.

The algorithm for nonisolated SPs requires instead the concurrent processing of multiple SPs. This part of the processing is performed at the level of individual pixels, dropping the SP-based formatting of the data. Each detector pixel is mapped to a cell within an active bit matrix, set to 1 or 0 according to the pixel status. The bit matrix has a built-in logic, capable of recognizing certain predetermined patterns, signaling the presence of a cluster corner at a certain pixel position. Since more than 96% of the reconstructed clusters are made of no more than four contiguous pixels, the most efficient choice for the patterns is an “L”-shaped sequence of inactive pixels with two different configurations of active pixels with the cluster candidate contained in a 3 × 3 pixel grid (Fig. 4). If one of the patterns is matched, the cluster candidate is recognized in the grid (green pixels in figure), as well as the anchor pixel (blue pixel in figure), positioned in one of the corners of the grid depending on the orientation of the sensor. The presence of a cluster corner is simultaneously checked on every bit of the matrix and it is completed in a single clock cycle. In the next clock cycle, the first cluster found is extracted from the matrix.

This highly parallelized mechanism is key to the successful operation of the architecture at the extremely high throughput levels required by the LHC collision rate. However, the amount of FPGA logic resources needed to implement a complete bit-matrix map of the approximately 40 millions pixels of the VELO detector would be excessive. This corresponds to approximately 780 000 pixels in input to each VELO readout card. To overcome this problem, a sparse representation of the bit matrix is adopted, breaking it down into a set of small matrices of fixed size, which get dynamically allocated only for the regions of the detector that contain active pixels. After some optimization studies, accounting for a maximum detector occupancy of around 0.125% [5], an average size of reconstructed clusters of 2 pixels, and computational requirements, the size of each small matrix has been chosen to cover the same area as 3 × 3 SPs, that is, 6 × 12 individual pixels (Fig. 5). To reconstruct cluster candidates having an anchor pixel lying near the edge, each matrix is surrounded by the edges of registers fixed at zero, as shown in Fig. 6. These edges are not used during the filling process, but are necessary to determine the 3 × 3 cluster candidate when there are active pixels at the edge of the 3 × 3 matrix. An example of such configuration is shown in Fig. 6(b). The width of the edges is determined by the VELO sensor number, the allowed patterns, and the cluster candidate topology (Fig. 4).

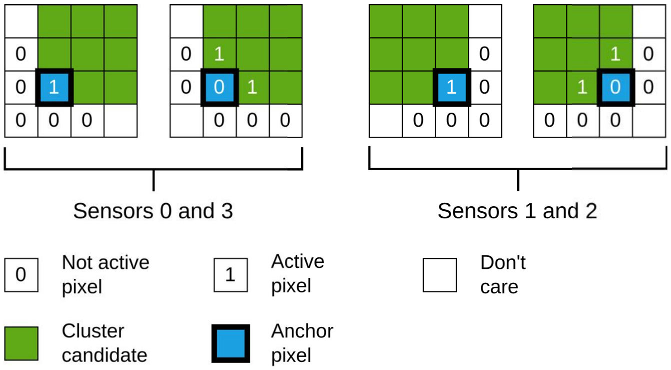

Fig. 4. Pixel patterns used to identify a cluster candidate. The patterns are optimized for the sensor mounting orientation. See Giambastiani [11] and Bassi [12] for further details.

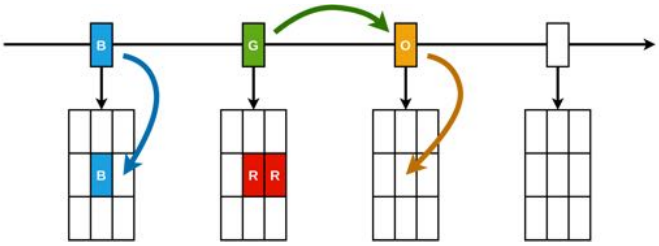

Fig. 5. Sketch of the matrix filling mechanism for nonisolated SPs. SPs with same color (label) are neighbors with active pixels. The blue SP (B) fills the first matrix in the line that is already populated with one of its neighbors. The green SP (G) does not belong to any of the already populated matrices, so it moves forward. The orange SP (O) has reached a noninitialized matrix, so it fills the center.

To allow the allocation of matrices to proceed in real-time without any delay, matrices are organized in a sequential chain for each VELO sensor, with SP data flowing continuously along the chain at the same rate as they are fed into the clustering block. All the matrices are initialized as empty. When an SP arrives at an empty matrix, it fills the center of the matrix and it defines the physical location of the matrix inside the VELO detector, as well as the set of coordinates of the other SPs that can fill it. The allocated position of the matrix is checked against the coordinates of every SP going through the chain. If an SP belongs to the region inside the matrix, it is used to fill its appropriate location; otherwise, it moves forward along the input line. Eventually, every SP gets stored in some matrix of the chain. An explanatory diagram illustrating the mechanism is shown in Fig. 5.

When the input flow of SPs has ended, data from each matrix are copied in a single clock cycle to a twin matrix (pattern recognition matrix) where cluster-finding is performed. In this way, the input matrix is ready to accept data from the next event immediately. The pattern recognition of all the potential cluster candidates in this twin matrix is then performed, and the local coordinates of the centroid, with respect to the anchor pixel position, of each found cluster are determined using an LUT. The absolute position of the cluster candidate is obtained as the sum of three vectors of coordinates: the position of the matrix with respect to the

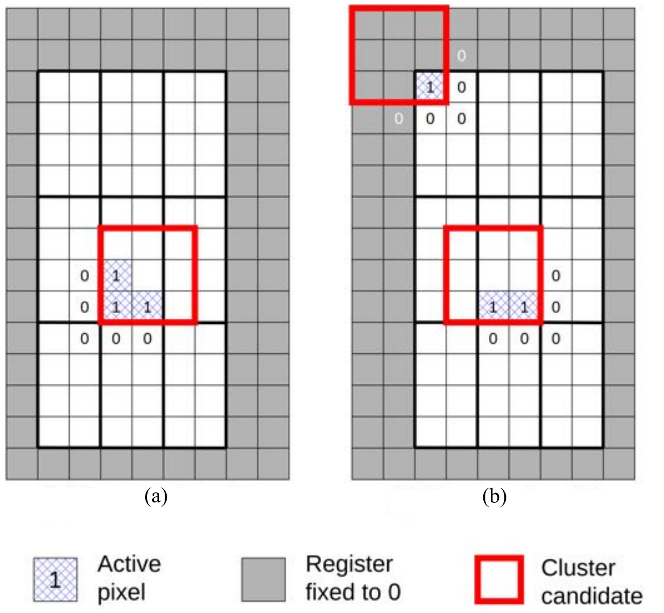

Fig. 6. Matrix edges and pattern orientations (a) for sensors 0 and 3 and (b) for sensors 1 and 2.

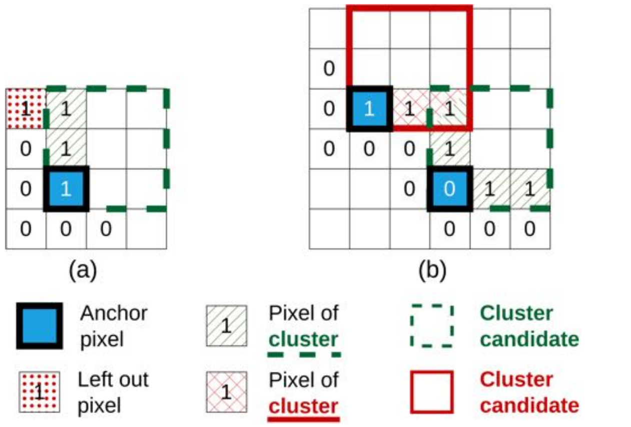

Fig. 7. Example of corner cases of the FPGA clustering algorithm: (a) partial cluster reconstruction and (b) cluster splitting.

sensor, the position of the anchor pixel with respect to the matrix, and the position of the reconstructed cluster with respect to the anchor pixel.

The clustering algorithm described has several parameters that can be tuned to optimize its performance in terms of speed, efficiency, and quality of the reconstruction. The shape and size of the matrix are determined by how nonisolated SPs are arranged together, whereas the distribution of the number of SPs with neighbors per event sets the number of matrices that need to be instantiated. The implementation of the above algorithm as FPGA firmware does not allow the number of matrices to be dynamically adjusted to cope with the variable number of nonisolated SPs per event. However, from LHCb simulations we determined that a fixed number of 20 matrices per VELO sensor is sufficient to ensure that less than 0.1% of the SPs overrun the matrix chain at the nominal Run 3 instantaneous luminosity (2×10^{33} $\mathrm{cm}^{-2}\mathrm{s}^{-1}$), and this number was adopted for the final implementation. SPs exceeding this limit are not discarded. Instead, partial information is extracted from them, by resolving them via an LUT as if they were isolated. This approach avoids inefficiencies, at the expense of a slight increase in the number of split clusters, since more than one cluster ends up being reconstructed from a single group of neighboring pixels, when they happen to be spread over multiple SPs [11], [12]. These clusters are flagged in the output as "nonisolated," to allow the reconstruction algorithms in the HLT to deal with them properly.

The LHCb experiment foresees to collect data also for heavy-ion collisions [13]. A modified version of the clustering architecture will be used to cope with the higher number of SPs (around six times larger than in pp collisions). Due to the limited amount of FPGA resources, the same matrices will be used multiple times to accommodate all the SPs. The cluster reconstruction of a heavy-ion event requires more time than the pp case, given the higher number of SPs, however, due to the much lower interaction frequency (50 kHz) the firmware can easily provide the necessary throughput also in this case.

IV. Physics Performance

The FPGA cluster-finding architecture was designed with the intent of replacing the raw pixel data with the reconstructed hit coordinates at the detector readout level. Except when running in the debug mode (that preserves the full original information together with the cluster data), the raw pixel data are discarded and cannot be recovered at any later stages. Therefore, extensive simulation studies were performed to assess the effect of using the FPGA-reconstructed clusters on the physics performance of the LHCb reconstruction, both at the HLT1 and HLT2 stages. This was compared with the alternative scenario in which the VELO hits are reconstructed by a full-fledged software reconstruction within the HLT system, free from all the constraints imposed by the FPGA architecture, and from the severe throughput requirements of operating at pre-build level (30 versus 0.17 MHz, where a farm of about 170 GPUs is assumed for HLT1).

For the sake of generality, comparisons are made to a CPU-based clustering algorithm that is free from implementation-specific constraints. However, the actual HLT1 implementation at LHCb is GPU-based, but its performance is indistinguishable from the CPU version we take as reference. The key differences between the FPGA and CPU algorithms that can potentially affect the reconstruction performances are the cluster-finding mechanism, the maximum cluster size in the FPGA algorithm (limited to a 3 × 3 pixel grid), and the constraints of the FPGA matrix filling scheme. They can potentially lead to inefficiencies, cluster splitting, or incomplete reconstruction of some clusters. An example of partial cluster reconstruction is illustrated in Fig. 7(a), where the red pixel is left out of the reconstructed cluster. The shift of the reconstructed hit position may lead to a degradation of the precision on the reconstruction of the particle trajectory, or even to a loss of efficiency if the associated track is not reconstructed at all. Fig. 7(b) shows an example of cluster splitting, where the algorithm finds two clusters, with a pixel in common, from six contiguous active pixels.

To perform the studies, a faithful software simulation of the FPGA-based clustering algorithm has been produced. This simulation was integrated within the official LHCb software simulation, and a CPU–FPGA comparison was performed on a

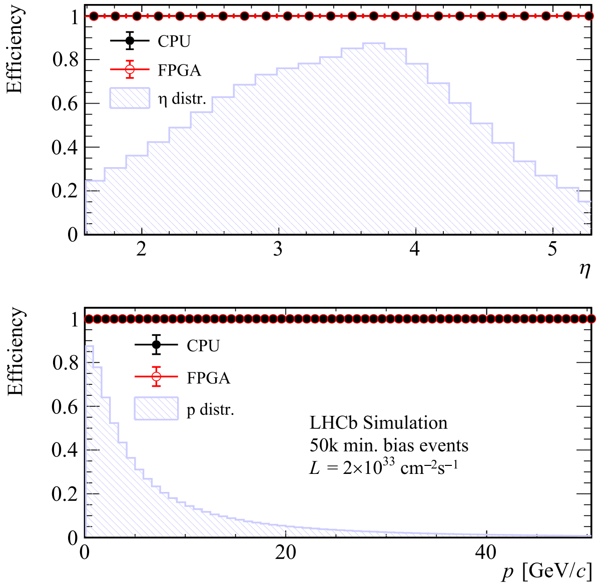

Fig. 8. Efficiency in reconstructing clusters as a function of the pseudorapidity (top) and the momentum (bottom) of the associated tracks for the CPU and FPGA-based algorithms. Clusters, from tracks that can be reconstructed using only information from VELO hits, are shown. The blue histograms show the pseudorapidity (top) and the momentum (bottom) distributions of the tracks.

sample of 50 000 bunch crossings, each containing an average number of 7.6 pp interactions, at an instantaneous luminosity of 2×10^{33} cm^{-2} s^{-1}. This corresponds to a total number of about 10^8 SPs (7×10^7 clusters), generated at the foreseen LHCb Upgrade running conditions with a center of mass energy of 14 TeV. The efficiency in reconstructing VELO clusters is defined as the ratio between the number of Monte Carlo (MC) hits matched to a cluster and the total number of MC hits. Only MC hits that produce enough charge in the detector to activate at least one pixel are considered. An MC hit is matched to a cluster if they share at least one pixel. The efficiency in reconstructing VELO clusters of the FPGA-based algorithm is about 99.8%, and almost indistinguishable from that of the CPU algorithm, as illustrated in Fig. 8. Here, the efficiency of reconstructing clusters from tracks that can be reconstructed using information from VELO hits only is shown as a function of the pseudorapidity (η) and momentum of the tracks. The overall FPGA cluster inefficiency, with respect to the CPU algorithm, is below 0.1% within the LHCb geometrical acceptance ($2 < \eta < 5$).

The quality of the reconstructed clusters is also studied by looking at the distributions of cluster residuals. The residual is defined as the distance between the position of the reconstructed cluster and the true coordinates of the hit generated by the passage of the particle on the associated detector layer. A comparison between residual distributions of reconstructed clusters, between the CPU and FPGA algorithms, is shown in Fig. 9. Distributions are plotted over the x coordinate, in the LHCb global reference frame. The two distributions are indistinguishable in the core, with very small differences in the tails. It is also checked that most of the nonreconstructed hits are of inferior quality, sitting in the tails of the resolution curve.

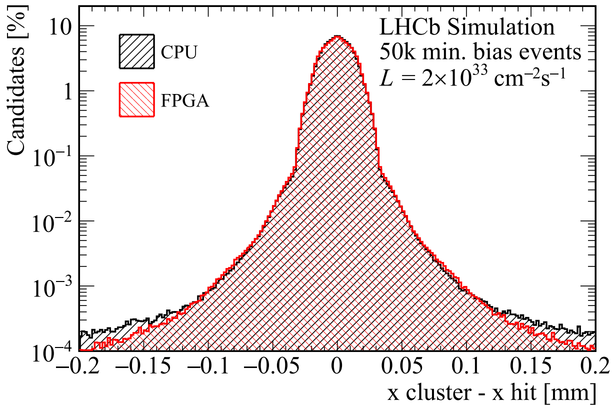

Fig. 9. Distributions of cluster residuals, along the x coordinate, for the CPU- and FPGA-based clustering algorithms. Distributions are normalized to unity. Similar results are obtained for the y coordinate.

Extensive studies are also performed to measure the quality of the full track reconstruction, when FPGA VELO clusters are used. The trajectories of charged particles traversing the tracking system are reconstructed from hits in some of the three tracking detectors, that is the VELO, the upstream tracker (UT) placed upstream of the magnet, and the scintillating fiber (SciFi) detector placed downstream of the magnet [14]. Tracks reconstructed using only hits from the VELO detector are called VELO tracks. VELO tracks having $\eta < 2$ are used only for the primary vertex reconstruction while those with $2 < \eta < 5$ can be extended in the forward region to attach hits from the SciFi detector, and optionally from the UT. These tracks are called "long tracks." As they traverse the whole magnetic field of the LHCb detector, they have the most precise measurement of the momentum and therefore are key for physics analyses. Table I shows a comparison between the CPU- and FPGA-based reconstruction performances for VELO tracks and for the VELO segment of long tracks. It also reports the relative fraction of clone-reconstructed tracks with respect to the total number of tracks in the category they belong to and the relative fraction of ghost-reconstructed tracks with respect to the total number of tracks. A clone is defined as any additional reconstructed track matching an already truth-matched MC track, whereas a ghost is a reconstructed track not associated with any true MC track [15]. The efficiencies and clone fractions are almost indistinguishable when comparing the CPU and FPGA algorithms for VELO and long tracks, not displaying any perceptible systematic difference. The fractions of ghost tracks differ at the per-mille level. This difference is due to tracks in the pseudorapidity region below 1.5. These tracks graze VELO sensors at a very low angle and produce very large clusters. For this reason, the position of the particle hitting the detector and creating the cluster is unlikely to be accurately measured regardless of the clustering algorithm.

The quality of the reconstructed tracks is also studied in terms of momentum, primary vertex and impact parameter resolutions, as shown in Fig. 10. The robustness of the algorithm is also verified against occupancy and relative fraction of large clusters [12]. Differences in reconstruction quality

TABLE I

VELO Tracking Efficiency, Relative Fraction of Clone and Ghost Tracks, Comparing CPU- and FPGA-Based Clusters

Track type	Quantity	CPU clusters [%]	FPGA clusters [%]
All VELO tracks	efficiency	98.254 ± 0.007	98.254 ± 0.007
	clone	1.231 ± 0.006	1.234 ± 0.006
Long tracks	efficiency	99.252 ± 0.006	99.252 ± 0.006
	clone	0.806 ± 0.006	0.806 ± 0.006
	ghost	0.848 ± 0.003	0.928 ± 0.003

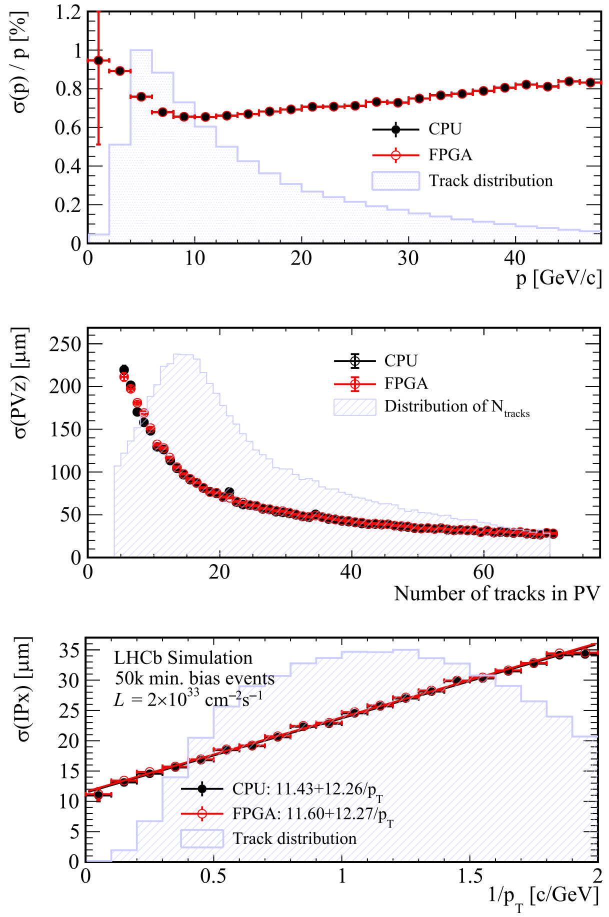

Fig. 10. Track reconstruction resolutions for the CPU- and FPGA-based clustering algorithms: momentum resolution as a function of the momentum (top), primary vertex resolution along the beam direction as a function of the number of tracks in the reconstructed primary vertex (middle), and impact parameter resolution along the horizontal direction as a function of the inverse of the transverse momentum (bottom). Impact parameter resolutions are fit with a linear function. The blue histograms show the distributions of the momentum of the reconstructed tracks (top), number of reconstructed tracks per primary vertex (middle), and inverse of the transverse momentum of the reconstructed tracks (bottom).

between the FPGA and the CPU implementations do not show any trend as a function of these probes. In conclusion, all the studies have shown that the FPGA-reconstructed clusters lead to a quality of track reconstruction that is effectively indistinguishable from the software reconstruction.

V. Implementation Details and Integration

Given the indistinguishable performance of the FPGA-based clustering algorithm with respect to the software-based one, the LHCb collaboration decided to integrate the cluster-finder architecture within the TELL40 cards that perform the readout of the VELO, exploiting spare FPGA resources not used by the readout firmware. The VELO time-ordering firmware, plus the common LHCb firmware, takes up about 44% of its logic resources and 64% of its M20K memory blocks.

Each VELO TELL40 receives data from a single VELO module. Two independent and identical parallel processing chains are implemented in the FPGAs, each of which receives and processes data from one VELO half-module [8]. The clustering architecture, with all the needed ancillary logic, is integrated as a self-contained block at the end of each chain, and it has, therefore, two identical instances running in parallel, in analogy with the readout firmware (Fig. 11). The output of the clustering is transmitted out of the readout card through its PCIe interface to the host server, which assembles the data from different subdetectors for each event.

The clustering architecture is itself composed of several units, each devoted to a specific task (Fig. 11). First, a decoding and flagging stage splits data into separate streams, while flagging isolated SPs. Second, a pair of switch blocks sends data to the cluster processing blocks. Reconstructed clusters are then finally encoded with the chosen output format. A back-pressure mechanism is implemented throughout the pipeline: each processing block sends a "ready" signal to the previous unit when it is capable of receiving data.

A detailed description of the firmware implementation, its integration and commissioning within the LHCb data acquisition, can be found in Bassi [12].

A. Clock Domains

Each unit in an instance of the clustering architecture writes its output to a First In First Out (FIFO) that is read by the subsequent unit. The purpose of the FIFOs is twofold. First, they allow buffering and flow control between the clustering units. In addition, FIFOs allow data synchronization between different clock domains. In our application, the decoder and encoder stages run on a 250-MHz clock, whereas the switch and clustering processing block use a 350-MHz clock. These values have been chosen to ensure that the system as a whole can provide a throughput in excess of 30 MHz (see Section VI), while still respecting the timing constraints due to internal signal propagation in all its parts.

B. Data Formats

Active SPs are encoded as 32-bit words. Each word contains the pixel hitmap (8 bits), the SP position inside the sensor (15 bits), and the sensor identifier within the sensor pair (1 bit). Each VELO sensor is made of 256 × 768 pixels. Each SP is composed of 4 × 2 pixels, such that 6 bits are needed to specify the SP row, whereas 9 bits are required for the column. One extra bit is needed to identify the source sensor, as each data chain receives SPs from two sensors.

Clusters are encoded in 32-bit output words, as sketched in Fig. 12. Of these, 22 bits are used to specify the position of the cluster centroid, with 18 bits specifying the position of the pixel where the cluster centroid is located (Integer column

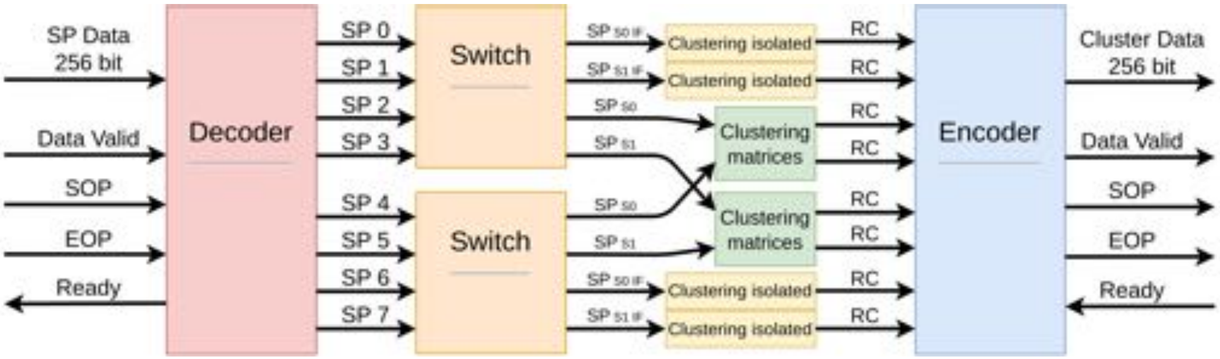

Fig. 11. Basic blocks of the clustering architecture. VELO data are received as 256-bit words, each containing eight SPs. A "Data Valid" signal states whether the incoming data are valid. SOP and EOP signals delimit the start and the end of the data corresponding to each event. The clustering block sends a ready signal to the previous architecture component when it is ready to accept data. The "decoder and isolation flagging" splits the 256-bit bus into eight 32-bit wide buses, each containing one SP. It also flags SPs that do not have any active neighbor SPs (IF). A pair of switches arrange SPs by sensor (S0/S1) and by IF. The "clustering isolated" and "clustering matrices" blocks reconstruct clusters, which are encoded back into 256-bit words by means of an encoder.

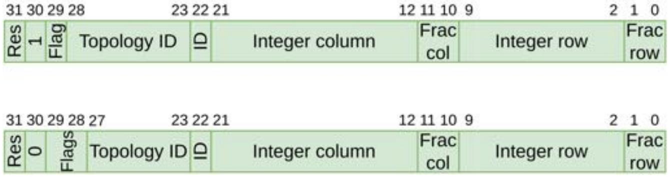

Fig. 12. Data formats for clusters reconstructed from isolated SPs (top) and clusters reconstructed from not isolated SPs (bottom). Bit 31 (Res) is reserved for internal use.

and Integer row), and additional 4 bits are used to specify the position of the centroid within the pixel, in units of 1/4 of a pixel (Frac col and Frac row). Analogously to SP data, 1 bit is used to identify the sensor (ID). Eight additional bits are used to encode a cluster-topology identifier (Topology ID) and the reconstruction-quality flags (Flags). The topology identifier is used to distinguish cluster topologies that share the same centroid position within the pixel, so that the full cluster topology can be retrieved. If the cluster is reconstructed from an isolated SP [bit 30 = 1 in Fig. 12 (top)], 6 bits are used to store the topology identifier, whereas 5 bits are needed to store the identifier for clusters reconstructed through the matrices [bit 30 = 0 in Fig. 12 (bottom)]. The cluster topology information is used both for monitoring purposes and for the ultimate optimization of tracking performance, as the uncertainty associated with the 2-D position of a cluster depends on its topology. The reconstruction-quality flags allow to distinguish between: clusters from isolated SPs, clusters reconstructed inside a matrix, and clusters built from SPs overflowing the maximum number of instantiated matrices (which are arbitrarily treated as isolated). For clusters reconstructed within matrices, the word contains two additional quality flags, which specify whether a cluster was fully contained in the 3 × 3 grid, and whether the grid touched the boundary of the host matrix (which potentially means that the reconstructed cluster is a fragment of a larger cluster).

C. Input–Output Interfaces

Our architecture block requires a "valid" signal to confirm the validity of the current input data word. Additional start of package (SOP) and end of package (EOP) signals allow to separate data coming from different events. The SOP signal is received with the first word of every event, whereas the EOP is generated together with the last input word. The "valid" SOP and EOP signals are also present on the output side, where clusters are transmitted.

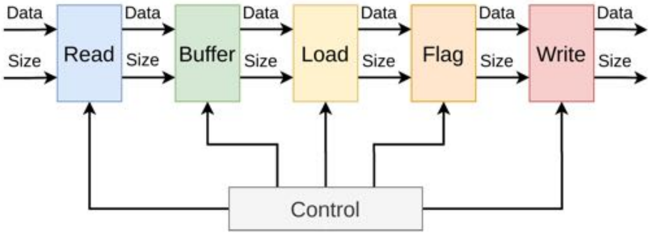

Fig. 13. Block diagram of the isolation flagging.

D. Decoder Block

The input data to the clustering firmware arrive grouped in 256-bit words, each carrying 8 SP words. The first block is a decoder, which splits the 256-bit words into eight 32-bit streams. The decoder is also responsible for converting the SOP-EOP protocol into the EndEvent (EE) protocol used within the clustering architecture: a 32-bit EE word is interposed between SPs of different events in all the eight data streams. Each EE word carries a specific flag to distinguish it from SPs, and an event identifier (5 bits), which can be used during subsequent data processing to cross-check data synchronization.

E. Isolation Flagging

Within the decoder block, SPs are flagged with an isolation bit. The flagging process includes five steps: read, buffer, load, flag, and write, arranged in a pipeline, as shown in Fig. 13.

First, all the SPs of a given event are read and stored into registers. The maximum number of SPs that can be stored in the read registers is not dynamically adjustable. It has been set to 144 based on the distribution of the expected number of SPs in the most crowded VELO module, requiring more than 98% of the LHCb simulated events to be accommodated into the read registers. In events where the number of SPs exceeds the size of the read registers, SPs are not sent to the flagging process but are instead bypassed and sent directly to the matrix chains. This causes a local slowdown of the entire chain as the matrices need to reconstruct clusters from a high number of SPs. This effect is included in the measurement of the average throughput of the entire system. In addition, as the number of input SPs increases, the fraction of SPs overflowing the number of available matrices gets higher, increasing the number of split clusters. The corresponding increase in split clusters is also taken into account in the evaluation of the reconstruction performance.

As soon as all the SPs of one event have been received, the content of the read registers is copied to the buffer. This data exchange decouples the reading and flagging operations, allowing SPs of one event to be read in while the flagging of the previous event is still ongoing. The flagging process compares the coordinates of each SP to the ones of the other SPs in the same event. A status vector is used to store the isolation flag (IF) for each SP: if two SPs are found to be neighbors, the corresponding bits in the status vector are set to 1. SP comparisons are not all performed in a single clock cycle. On each clock, the load block extracts two subsets of 16 SPs each from the buffer (Fig. 13). For each SP in the two subsets, it also computes the set of coordinates to be matched by the neighbors by one-unit additions and subtractions of the coordinates of the SP row and column. The two SP subsets, together with the coordinates of the neighbors, are passed to the flag block that performs the 16 × 16 comparisons on the two subsets. For each SP of the first subset, the flag block checks whether the SP row is equal to one of the rows of the SPs in the second subset or to the row above or below; the same check is performed on columns. If both the row and the column checks yield a positive result, the two SPs are flagged as neighbors, and the corresponding bits in the status vector are set to 1. On each clock cycle, the load block selects a different pair of SP subsets from the buffer sending them to the flag block, until all the possible combinations of 16-SP subsets have been checked. The described architecture allows reusing the same logic resources while updating the SP subsets to be flagged at each clock cycle. To perform the comparisons between n 16-SP blocks, $n(n+1)/2$ clock cycles are needed.

The number of parallel comparisons performed for every clock cycle is the result of a tradeoff between resource usage and throughput and is based on the constraints of its use within the LHCb experiment.

As soon as all the comparisons have been completed, the contents of the flagging registers and the status vector are copied to the write block, thus decoupling the flag and write processes. The write block is responsible for adding the IF to the SP words and for sending flagged data to the next component, the switch. The data exchange within the read–buffer–load–flag–write pipeline is regulated by back-pressure: if a component cannot accept the data of an event because it is still processing the previous event, the control unit keeps the previous component on hold.

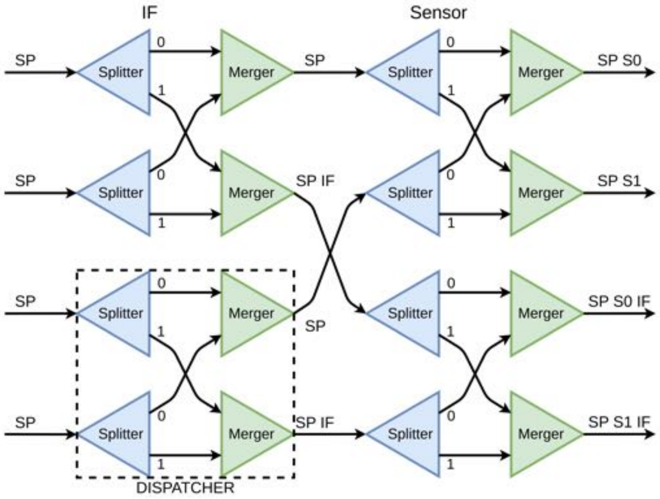

Fig. 14. Block diagram of a 4-to-4 switching unit.

The decoder block, including flagging and bypass, uses 7% of the logic and 1% of the M20K memories available in an Arria 10 FPGA.

F. Switch Block

The cluster processing chain receives SPs from both the sensors of a VELO half-module. The switch, placed after the decoder, arranges SPs by sensor and by IF, feeding them to the appropriate cluster-reconstructing blocks. Each of the two switching units shown in Fig. 11 performs a 4 → 4 switching, allowing every input data word to be directed to any of the four output streams according to its flags, regardless of the origin input stream. The basic switch constituents are the splitter and the merger (Fig. 14). The former has one input and two outputs, and it sends input data to one of the two outputs according to their IF or origin sensor. The latter has two inputs and one output, and it routes two inputs in a single output line. Two splitters and two mergers combine to form a 2 → 2 dispatcher.

The block diagram of the splitter is shown in Fig. 15. The splitter is based on a finite-state machine (FSM). The next state is determined by the R0 register state, the arrival of valid input data, and the hold state of the following processing block. On the arrival of valid input data, the FSM decides between sending it directly to the output and storing it in the R0 register, based on the input hold signal. In the latter case, a latch enable (LE) write signal is sent to the register. A multiplexer controlled by the FSM routes data to the output. If an SP is received, then one of the two valid signals is set to 1, according to the routing scheme (IF or sensor). If an EE signal arrives, it is sent to both the outputs. The input hold signal determines whether data can be sent to the output. An output hold is generated as long as the R0 register is full, since no more data can be accepted as input, given the possibility of an input hold signal assertion.

The block diagram of the merger is shown in Fig. 16. As for the splitter, an FSM determines whether input data can be sent directly to the output or must be stored in appropriate registers

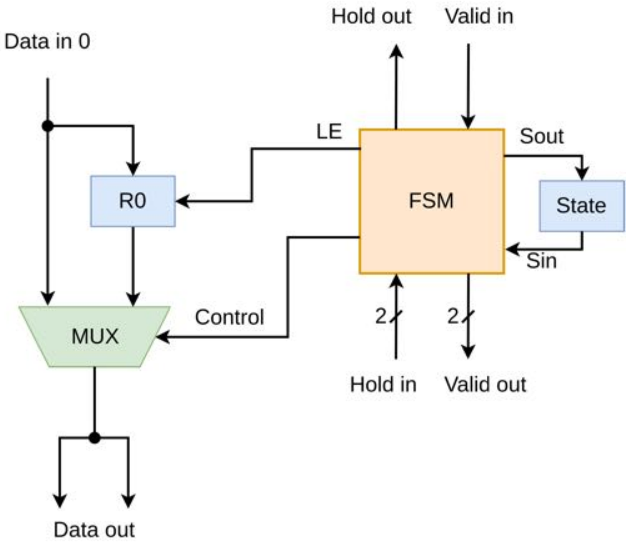

Fig. 15. Splitter block diagram. R0 and State are registers, MUX is a multiplexer, and FSM is a finite state machine that manages hold, valid, control, and LE write signals.

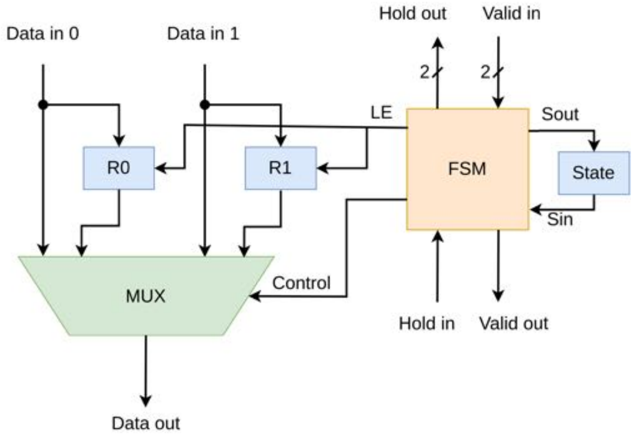

Fig. 16. Merger block diagram. R0, R1, and State are registers, MUX is a multiplexer, and FSM is a finite state machine that manages hold, valid, control, and LE write signals.

(R0 and R1). If an EE word arrives on one of the inputs, it is stored until a second EE word arrives at the other input. The two EE words are then compared and, if their event IDs match, a single EE word is output; otherwise, a sync error signal is set to 1.

G. Cluster Reconstruction

All the isolated SPs, identified by the switch, are sent to the corresponding clustering block and are resolved by means of an LUT, as shown in Fig. 17.

The LUT reconstructs the cluster centroid from the active-pixel hitmap extracted from the SP word. The cluster word is built by combining the LUT output with the original SP row and column. If two different clusters are reconstructed inside an isolated SP, a bit is set to 1, and the two outputs are combined by a merger block (Fig. 16). Reconstructed clusters are then sent to the output FIFO.

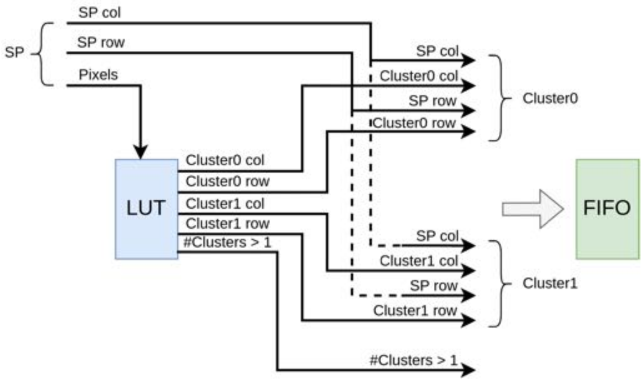

Fig. 17. Cluster reconstruction of isolated SPs by means of an LUT.

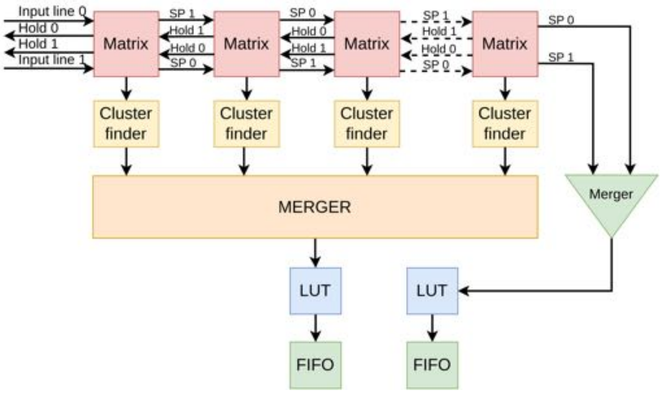

Fig. 18. SP distribution in a matrix chain. Clusters are reconstructed through the cluster-finder block and merged into a FIFO.

The reconstruction of clusters from nonisolated SPs requires two different processing steps. Input data are first sent and distributed in the matrix chain, and then, when matrices have been filled with SPs, the actual reconstruction of the clusters takes place, as shown in Fig. 18. To ensure a high throughput, each matrix receives data from two parallel input lines. Each input line is combined with a hold signal that is propagated backward through the whole chain to control the data flow by back-pressure to avoid data loss. As the first SP populates a matrix, a set of coordinates is calculated and stored, to be matched with all the further SPs arriving at the same matrix. The initialization of an empty matrix is done using only one of the two input lines, since only a single SP can enter the center of the matrix at a time. A second SP coming simultaneously from the other parallel line would need the coordinates of free slots to fill the matrix that cannot be immediately available for timing constraints. For this reason, input line 0 (Fig. 18) has priority over input line 1, which is put on hold as the matrix is initialized. To keep a good load balancing, input lines are swapped when going from one matrix to the next: line 0 of a matrix feeds line 1 of the next matrix and vice versa. When EE words have arrived on both the input lines, the content of the matrix is moved to the cluster-finder block. An error is raised if two different EE signals are detected.

During the second step, the cluster-finder block processes the content of the corresponding filled matrix. Fig. 19 shows the logic of how clusters are reconstructed, starting from the

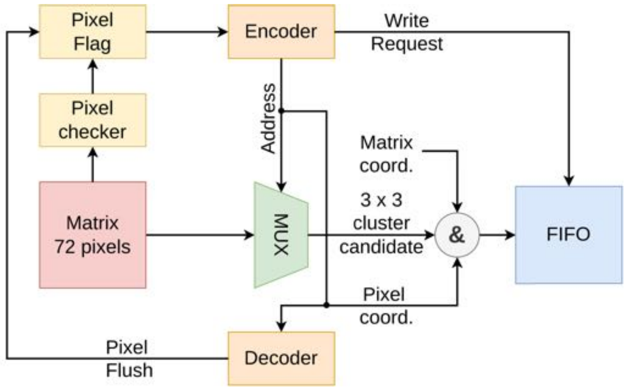

Fig. 19. Cluster-finder block diagram and its data flow.

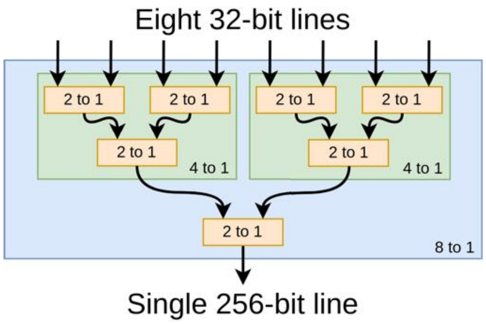

Fig. 20. Structure of an 8-to-1 encoder built from 2-to-1 encoders.

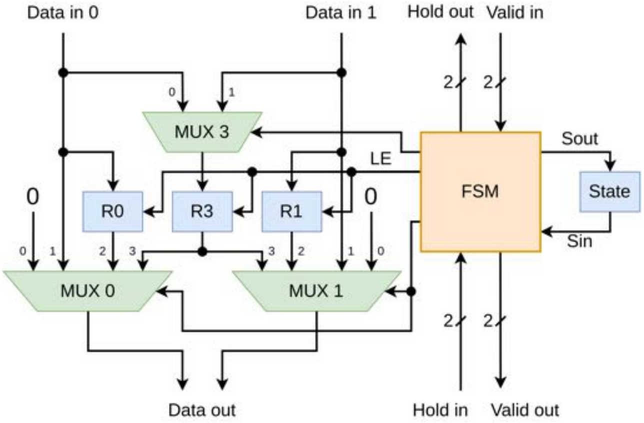

Fig. 21. 2-to-1 encoder block diagram. R0, R1, R3, and State are registers, MUX0, MUX1, and MUX3 are multiplexers, and FSM is a finite state machine that manages hold, valid, and LE write signals.

matrix pixel content. Each pixel in a matrix checks whether it belongs to one of the L-shaped patterns of the algorithm through the pixel checker block. This process is performed in parallel at full speed for each pixel in each matrix. When a pattern match is found, an anchor pixel in the matrix is identified. As a consequence, the bit in the pixel flag vector corresponding to the position of the anchor pixel in the matrix is set to 1. An encoder reads the pixel flag vector content and passes the addresses of all the found anchor pixels to a multiplexer, one at a time. The multiplexer extracts the 3 × 3 cluster candidate corresponding to the address received from the encoder. As soon as an anchor pixel has been processed and corresponding cluster candidate found, the decoder block receives the pixel address from the encoder and resets the corresponding bit to zero in the pixel flag vector. The reset operation is performed by means of the pixel flush signal. For each cluster, a word containing the matrix coordinates, the anchor pixel position, and the 3 × 3 cluster candidate is written in the matrix FIFO. A merger reads the cluster candidates from all the matrix FIFOs and sends them to an LUT, which computes the centroid of each cluster (Fig. 18). The cluster position is obtained by combining the matrix position in the detector, the anchor-pixel position in the matrix, and the LUT output. The cluster words are then saved into a FIFO that contains all the clusters from nonisolated SPs of a VELO sensor that do not overflow the matrix chain. The two data lines at the end of the matrix chain which carry overflow SPs are merged into a single line. Overflow SPs are reconstructed as if they were isolated by means of an LUT, and the reconstructed clusters are stored into a FIFO.

H. Encoder

The last processing block of the clustering architecture is devoted to encoding the eight separate 32-bit data streams into a single 256-bit bus, to comply with the required output format. The encoder architecture has been designed as a tradeoff between speed and bandwidth optimization. The encoder is required to output a 256-bit word at each clock cycle, to maintain a throughput larger than 30 MHz. Given the speed constraint, the SP packing performed by the encoder is not optimal in each event, interleaving zero-padded words in between 256-bit words to match the output width. To build the complete 8-to-1 encoder, seven 2-to-1 encoders are instantiated, as shown in Fig. 20. The 2-to-1 encoder block puts together two input data lines ($N+N$ bits) into a single output ($2N$ bits) by means of buffer registers (R0, R1, and R3) and a control FSM (Fig. 21). If two cluster words are received and no hold signal is asserted by the subsequent block, the two words are packed together and sent out. If a single cluster is received, it is stored in the R3 register and matched with the next input cluster. If a hold signal is received, the incoming cluster is stored in the R0 or R1 register depending on its input line. In case an odd number of words is received within an event, a zero-padded word is added to match the $2N$ output width. When two EE signals are received, they are compared and, if they match, sent out. Otherwise, an error signal is generated.

I. Monitoring and Error Handling

The clustering architecture has several blocks whose behavior affects the functioning of the entire data processing chain. Therefore, a monitoring procedure is implemented to probe each block throughout the whole reconstruction process to ensure a correct data handling. Between each block of the diagram illustrated in Fig. 11, a FIFO is inserted as a buffering element. It decouples the data writing process of the previous input block from the data reading of the subsequent output block, absorbing local processing rate fluctuations.

The occupancy levels of all the interposed FIFOs, as well as their maxima over a certain time interval, are periodically read to check for, and diagnose, possible slowdowns of any processing blocks. The fraction of SPs overflowing the matrix chain is also monitored. Each processing block is also equipped with an error-checking logic, which monitors two types of errors. The first type corresponds to a data loss, occurring when a block receives valid data in input and the register in which data should be written are already full. The second type occurs when mismatching EE signals are received, indicating a loss of synchronization in the input data. In both the cases, a signal is generated and an error word is output, containing a code to trace back the origin of the error for debugging purposes. A reset signal needs to be sent to the clustering logic and memories to recover from both the error types.

VI. FPGA Resource Usage and Throughput

The clustering architecture was initially compiled and tested standalone on a Stratix V-based prototyping board [16]. The FPGA device mounted on the prototyping board has a similar amount of logic, memory resources, and clock speed to the Arria 10 carried by the TELL40 readout boards. During the test, the firmware was fed with simulated SP data from RAM memories that are read in a loop. The clusters reconstructed in hardware were compared with the output of the high-level C++ simulation of the algorithm, run on the same set of input SPs. The quality of the reconstruction and the reliability of the measurements were verified.

The firmware can process events with up to an average of 32 SPs per VELO half-module, using a 350-MHz clock rate. This condition is met for the whole VELO detector, where the average occupancy is 26 SPs per event, near the nominal interaction point. An average event processing rate of 38.9 MHz is measured on minimum-bias LHC collision events, in the VELO module with the highest occupancy. The measurement is also performed on pp collisions with higher than average track multiplicity, containing reconstructible $B_s^0 \to \phi\phi$ decays, as a sample of typical data that the LHCb DAQ would select and save on permanent storage. The measured throughput of 30.9 MHz is still higher than the average LHC bunch crossing rate and ensures that even a random fluctuation leading to the occurrence of several high-occupancy events in a row poses no risk of clogging the pipeline. The clustering firmware is therefore expected to run safely throughout the entire Run 3 physics data taking.

Compiling the entire VELO firmware within the Arria 10 allows for the measurement of the amount of resources needed to perform clustering in real-time. The clustering firmware requires roughly 31% of the logic and 11% of the memory of an Arria 10 chip to process an entire VELO module. After standalone validation, the clustering firmware was combined and fully integrated with the readout firmware to build the complete VELO readout firmware. Additional features were added in the integration process, like the handling of global LHCb control signals, response to errors, and an optional bypass that allows both the SP and cluster data to be output for debugging purposes. SPs are then fed to the LHCb simulation that outputs software-based clusters which are then compared with the firmware-based ones. The optional bypass option will be enabled periodically, or in case of a need to debug, during data taking to check that the firmware is reconstructing clusters correctly. The overall final chip occupancy turns out to be about 75%. Some tuning was required to fix timing violations occurring due to the large fraction of resource usage and to the complex connectivity of the design. The complete firmware was then compiled and loaded on the LHCb readout boards and successfully tested within the DAQ system by means of signal injection in the detector front-end. The firmware has also been tested on the detector readout boards using an internal front-end generator within the firmware, capable of generating input data at the nominal data rate (64 Gb/s). At the time of this writing, the firmware is fully commissioned and has started to take physics data in LHC Run 3.

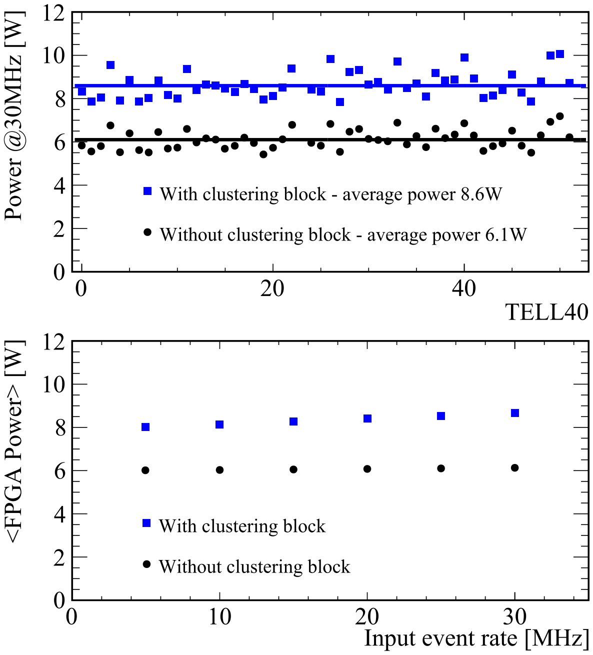

Fig. 22. Power consumption of individual VELO TELL40 FPGAs processing data at an event input rate of 30 MHz (top). The average value over the 52 FPGAs is also superimposed with a horizontal line. Average power consumption over all the 52 FPGAs as a function of the input event rate (bottom). Measurements using the firmware without the cluster-finding block (outputting SPs instead of clusters) are also reported for comparison.

The FPGA power consumption of all VELO TELL40s is measured at the nominal event rate of 30 MHz (Fig. 22). For comparison, the same measurements are performed using the firmware without the cluster-finding block, outputting SPs instead of clusters. The average power consumption of an FPGA within a TELL40 card when processing an event rate of 30 MHz with the readout firmware only is 6.1 W; this increases to a total of 8.6 W for the full firmware, including the clustering block. The same measurements are repeated for different values of the input event rate, showing a very slow increase in power consumption with the input rate.

VII. Summary and Conclusion

A novel 2-D clustering architecture was developed, implemented in the [very high speed integrated circuits (VHSIC) hardware description language (VHDL)], and integrated in the LHCb readout FPGA cards. The architecture exploits the principles developed within the INFN-RETINA Research and Development Project [4] for real-time track reconstruction and effectively represents its first processing stage.

This firmware proved capable of directly processing every event at the 30- MHz LHC crossing rate (a total flow of 5 Tb/s) without time-multiplexing or buffering of any sort, in a manner that serves the needs of an actual high-energy physics experiment. The physics performances of the algorithm were extensively studied and showed to be effectively indistinguishable from software clustering algorithms. The sparse-matrix technique adopted in its implementation proved successful in handling large detectors (order of 40 million pixels) with a modest amount of logic and memory resources. This allowed its insertion into the existing LHCb readout hardware, for use in the Run 3 physics data taking. This is a significant advancement over the previous state-of-the-art in HEP. The previous best performing cluster-finding system implemented in FPGAs has a throughput of about 100 kHz and requires the deployment of about four parallel firmware copies, processing about 15 MPixel/s each [17].

Moving the VELO clustering reconstruction from the HLT1 sequence to the FPGA readout cards leads to a measurable throughput improvement. Without accounting for isolation flagging, for which no software implementation is available for comparison, the present cluster-finder firmware allows a savings of about 11% of the computing power of the LHCb HLT1 full reconstruction sequence, allowing a corresponding increase in the LHCb data-taking rate. As a further advantage, a reduction of the VELO data size of approximately 14% was obtained, which allows to save resources both in the DAQ chain and in permanent data storage. Details on the GPU-based VELO clustering reconstruction can be found in Cámpora Pérez [18]. In addition, the FPGA implementation consumes significantly less electrical power than its GPU analog. From the data in Fig. 22, it follows that the set of 52 VELO TELL40s requires about 130 W of power to perform cluster reconstruction of the entire VELO, while the GPU implementation would require about 6 kW (again not including isolation flagging). The power needed to perform cluster reconstruction on GPUs is estimated by multiplying the GPU power usage (230 W) by the number of GPUs (236) required to process a 30-MHz input event rate and by the fraction of time spent in cluster reconstruction (11%). This is also in agreement with the measurements presented in Aaij et al. [19].

VIII. Further Considerations

In a broader perspective, this work can be seen as a special case of connected-component labeling (CCL) with center of gravity calculation (COG)—a computation that often occurs in image processing systems with the purpose of identifying connected sets of pixels belonging to the same visual feature. The main difference is the modest size of the features of our interest, which we could contain within a 3 × 3 matrix, and their sparseness, which makes our problem somewhat simpler. However, this greater simplicity comes with a "frame rate" requirement (30 MHz) that is orders of magnitude larger than typical image processing rates (<1 kHz). In fact, a CPU implementation exists of the same VELO clustering task discussed in this article that was inspired by some algorithms in use in image processing problems, appropriately revisited to exploit the smallness of the size of the components and their sparseness [20].

In recent years, also this type of image processing tasks is increasingly being moved from CPUs to dedicated FPGA firmware to achieve greater speed and efficiency, and it may be interesting to compare those solutions to the present work. As an illustrative example, we take the FPGA implementation described in Spagnolo et al. [21]. There, frames of 640 × 480 pixels are processed at a rate of 730 Hz, by a Zynq AP-SOC 7045 FPGA, running a 225-MHz clock, without COG. This system compares well with our case, where each of the 104 instances of our firmware processes a matrix of 512 × 768 pixels, and our clock frequency and resource usage are also quite similar. The Arria 10 FPGA mounted on TELL40 cards has a capacity of 1150k logic elements, whereas the Zynq 7045 FPGA has 350k logic cells. A single instance of the clustering firmware requires about 15% of the available logic on the chip, while for the studies reported in Spagnolo et al. [21] we assume a typical usage of about 50% of the total resources. However, our frame rate is larger by a huge factor, of nearly 10^5. This difference is likely due to the sequential structure of the image processing firmware, which proceeds by a raster scan rather than by a massively parallel calculation; but is definitely also a consequence of the greater simplicity of our problem in terms of cluster size and occupancy. In fact, cases of FPGA-based CCL implementations that reach a throughput comparable to that of our architecture are based on breaking down the image in smaller parts that are analyzed in parallel, and later coalesced [22]; an approach that bears some resemblance to our use of sparse matrices.

However, all the above examples assume that the image data arrive as an ordered sequence of pixels and do not provide detailed topology analysis of the found clusters, so they could not be straightforwardly applied to our problem. Conversely, the smallness of the components addressed by our system may not be of interest in general image processing applications; nevertheless, it cannot be excluded that some of the ideas described in this article could find some use in image processing tasks, at least in some specific instances.

Acknowledgment

The authors are thankful for the support of the LHCb Real-Time Analysis Group, within which this project was developed. A special thank goes to the LHCb VELO Group for the tight collaboration, support, and integration coordination, without which this work would not have been possible. The authors would also like to thank the LHCb computing and simulation teams for providing the simulated samples used in the article.

References

[1] LHCb Collaboration, "LHCb trigger and online upgrade technical design report," CERN, Geneva, Switzerland, Tech. Rep. CERN-LHCC-2014-016, 2014. [Online]. Available: https://cds.cern.ch/record/1701361

[2] LHCb Collaboration, "Expression of interest for a phase-II LHCb upgrade: Opportunities in flavour physics, and beyond, in the HL-LHC era," CERN, Geneva, Switzerland, Tech. Rep. CERN-LHCC-2017-003, Feb. 2017. [Online]. Available: https://cds.cern.ch/record/2244311

[3] LHCb Collaboration, "LHCb upgrade GPU high level trigger technical design report," CERN, Geneva, Switzerland, Tech. Rep. CERN-LHCC-2020-006, May 2020. [Online]. Available: https://cds.cern.ch/record/2717938

[4] R. Cenci et al., "Development of a high-throughput tracking processor on FPGA boards," in *Proc. Topical Workshop Electron. Part. Phys. (TWEPP)*, Santa Cruz, CA, USA, 2017, p. 136. [Online]. Available: https://pos.sissa.it/313/136/

[5] LHCb Collaboration, "LHCb VELO upgrade technical design report," CERN, Geneva, Switzerland, Tech. Rep. CERN-LHCC-2013-021, LHCB-TDR-013, 2013. [Online]. Available: https://cds.cern.ch/record/1624070

[6] G. Bassi et al., "FPGA implemention of a fast 2D clustering algorithm (VHDL language)," INFN Sezione di Pisa, Pisa, Italy, Tech. Rep. 23524, 2019, doi: 10.15161/oar.it/23524.

[7] T. Poikela et al., "VeloPix: The pixel ASIC for the LHCb upgrade," *J. Instrum.*, vol. 10, no. 1, Jan. 2015, Art. no. C01057, doi: 10.1088/1748-0221/10/01/C01057.

[8] K. Hennessy et al., "Readout firmware of the vertex locator for LHCb run 3 and beyond," *IEEE Trans. Nucl. Sci.*, vol. 68, no. 10, pp. 2472–2479, Oct. 2021. [Online]. Available: https://cds.cern.ch/record/2789034

[9] J. P. Cachemiche, P. Y. Duval, F. Hachon, R. L. Gac, and F. Réthoré, "The PCIe-based readout system for the LHCb experiment," *J. Instrum.*, vol. 11, no. 2, Feb. 2016, Art. no. P02013. [Online]. Available: http://cds.cern.ch/record/2262859

[10] S. Miglioranzi et al., "The LHCb simulation application, Gauss: Design, evolution and experience," CERN, Geneva, Switzerland, Tech. Rep. LHCb-PROC-2011-006, CERN-LHCb-PROC-2011-006, Jan. 2011. [Online]. Available: https://cds.cern.ch/record/1322402

[11] L. Giambastiani, "A 2D FPGA-based clustering algorithm for the LHCb silicon pixel detector running at 30 MHz," M.S. thesis, Dipartimento di Fisica Enrico Fermi, Univ. di Pisa, Pisa, Italy, 2020. Accessed: Jul. 16, 2020. [Online]. Available: https://cds.cern.ch/record/2725831

[12] G. Bassi, "A FPGA-based architecture for real-time cluster finding in the LHCb silicon pixel detector," Ph.D. dissertation, Scuola Normale Superiore, Pisa, Italy, 2023. [Online]. Available: https://cds.cern.ch/record/2845901

[13] R. Litvinov, "LHCb: Heavy-ion physics results and prospects," *Int. J. Mod. Phys. E*, vol. 30, no. 11, Nov. 2021, Art. no. 2141004. [Online]. Available: https://cds.cern.ch/record/2804032

[14] LHCb Collaboration, "LHCb tracker upgrade technical design report," CERN, Geneva, Switzerland, Tech. Rep. CERN-LHCC-2014-001, 2014. [Online]. Available: http://cds.cern.ch/record/1647400/

[15] LHCb Collaboration, "Tracking definitions and conventions for run 3 and beyond," CERN, Geneva, Switzerland, Tech. Rep. LHCb-PUB-2021-005 ; CERN-LHCb-PUB-2021-005, Feb. 2021. [Online]. Available: https://cds.cern.ch/record/2752971

[16] Dini Group. *Board Model: DNS5GX F2*. [Online]. Available: https://www.synopsys.com/verification/prototyping/dini-products.html

[17] C.-L. Sotiropoulou et al., "A multi-core FPGA-based 2D-clustering implementation for real-time image processing," *IEEE Trans. Nucl. Sci.*, vol. 61, no. 6, pp. 3599–3606, Dec. 2014, doi: 10.1109/TNS.2014.2364183.

[18] D. H. C. Pérez, "Optimization of high-throughput real-time processes in physics reconstruction," Ph.D. dissertation, Univ. de Sevilla, Seville, Spain, 2019, ch. 3.1. [Online]. Available: http://cds.cern.ch/record/2718278

[19] R. Aaij et al., "Evolution of the energy efficiency of LHCb's real-time processing," in *Proc. EPJ Web Conf.*, vol. 251, 2021, Art. no. 04009. [Online]. Available: https://cds.cern.ch/record/2773126

[20] A. Hennequin, B. Couturier, V. V. Gligorov, and L. Lacassagne, "SparseCCL: Connected components labeling and analysis for sparse images," in *Proc. Conf. Design Archit. Signal Image Process. (DASIP)*, Montréal, QC, Canada, 2019, pp. 65–70. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02343598

[21] F. Spagnolo, F. Frustaci, S. Perri, and P. Corsonello, "An efficient connected component labeling architecture for embedded systems," *J. Low Power Electron. Appl.*, vol. 8, no. 1, p. 7, Mar. 2018. [Online]. Available: https://www.mdpi.com/2079-9268/8/1/7

[22] M. J. Klaiber, D. G. Bailey, S. Ahmed, Y. Baroud, and S. Simon, "A high-throughput FPGA architecture for parallel connected components analysis based on label reuse," in *Proc. Int. Conf. Field-Program. Technol. (FPT)*, Kyoto, Japan, Dec. 2013, pp. 302–305.