NUCL SCI TECH (2019) 30:127
https://doi.org/10.1007/s41365-019-0643-x

Study of the retina algorithm on FPGA for fast tracking

Zi-Xuan Song[1,2] · Wen-Di Deng[1,2] · Gilles De Lentdecker[2] · Guang-Ming Huang[1] · Hua Pei[1] · Yi-Fan Yang[2] · Dong Wang[1] · Frédéric Robert[2]

Received: 2 February 2019 / Revised: 28 April 2019 / Accepted: 29 April 2019 / Published online: 15 July 2019

Abstract Real-time track reconstruction in high-energy physics experiments at colliders running at high luminosity is very challenging for trigger systems. To perform pattern recognition and track fitting, artificial retina or Hough transformation algorithms have been introduced to the field typically implemented on state-of-the-art field programmable gate array (FPGA) devices. In this paper, we report on two FPGA implementations of the retina algorithm: one using a mixed Floating-Point core and the other using Fixed-Point and Look-Up Table, and detailed measurements of the retina performance are investigated and compared. So far, the retina has mainly been used in a detector configuration comprising parallel planes, and the goal of our work is to study the hardware implementation of the retina algorithm and estimate the possibility of using such a method in a real experiment.

Keywords Fast tracking · Field programmable gate array · Trigger

This work was supported by the National Key Research and Development Program of China (No. 2016YFE0100900), Fundamental Research Funds for the central universities (No. 2018YBZZ082) and National Science Funds of China (No. 11505074) and Belgian FRS-FNRS.

✉ Gilles De Lentdecker
gdelentd@ulb.ac.be

✉ Guang-Ming Huang
gmhuang@mail.ccnu.edu.cn

[1] Central China Normal University, Wuhan 430079, China

[2] Université libre de Bruxelles, Av. F.Roosevelt 50, 1050 Brussels, Belgium

1 Introduction

For many high-energy physics experiments, such as those at the high-luminosity LHC (HL-LHC), it is now of fundamental importance to perform precise and fast tracking at the earliest stage of the trigger system. As an example, HL-LHC is expected to deliver a peak luminosity of up to 5×10^{34} Hz/cm^2, with an average of approximately 140 overlapping proton–proton (pp) collisions per 25 ns bunch crossing [1]. Such a busy environment implies unmanageable data throughput, which is an extreme challenge for the traditional trigger system. High-resolution spatial information from silicon tracks is becoming attractive to the trigger, at the cost of adding a few microseconds latency, to maintain the physics performance.

Real-time reconstruction of charged particle trajectories at high event rates is a hard task. It requires pattern recognition algorithms for massively parallel track reconstruction within a few microseconds. Over the years, several studies have been proposed for track finding with specialized processors, such as the Silicon Vertex Trigger at the CDF experiment [2] and the Fast TracKer of the ATLAS experiment [3].

A pattern recognition algorithm termed "artificial retina" has also been proposed [4], as inspired by the early stage of the mammalian vision process. Retina is a parallel process similar to the Hough transform [5]. Its advantage is to increase parallelization of pattern matching processes and decrease the number of stored patterns. Research in this field has mainly been performed by the INFN-Retina Project [6, 7], which has shown that retina is well suited for fast, highly parallel tracking.

Track reconstruction at trigger level is a road search approach. In our present work, as an independent project, we are investigating the potential of FPGA implementation of the artificial retina and compare two full FPGA-based approaches using a simple ideal detector without magnetic field: one of which uses a mixed Floating-Point core [8] and the other employing Fixed-Point and Look-Up Table (LUT) computation. With a highly configurable structure, we are capable of investigating the performances of the implementations, including latency, hardware resource usage and retina algorithm track reconstruction performance. These implementations can be considered as a demonstrator of a full retina processor system for track trigger applications.

2 Artificial retina algorithm

2.1 Inspiration from neurobiology

The artificial retina working principle for high-energy physics was first proposed in 2000 [4]. It is inspired by the vision mechanism in mammals. The first step of the human visual system is to project a given stimulus on the retina via the eye's optics, which creates chemical and electrical signals that carry the visual information to the visual cortex where it is processed by the brain. Each neuron in the visual cortex is tuned to recognize a specific shape (characteristics such as orientations and edges) on a specific region of retina, which is termed a receptive field [6]. The neuron response intensity is proportional to the degree of similarity between the shape of the specific object and the shape the neuron is tuned to [6]. In this way, different neurons react to the same stimulus with different intensities. The brain can extract precise information and reconstruct an image by interpolating and weighing neuron responses within limited latency [4]. Similarly, when applying this working principle to detectors, parameter space can be divided into a pre-calculated set of stored patterns and compared with incoming hits in real time for good matching. This process can be targeted to high parallelization of information distribution and pattern recognition systems for fast tracking in a HEP experiment.

2.2 Application in 3D space without magnetic field

To apply the retina algorithm to real-time track finding, referring to the LHCb upgrade VELO detector [9], we first set up a simple ideal model of a 100% efficient position-sensitive detector made of eight parallel planes in the three-dimensional space without a magnetic field (Fig. 1). Each plane is a 7.424 cm × 7.424 cm square with a typical pixel size of 15 μm × 15 μm. The eight layers are centered on the beam line and are equally spaced along the line (the z axis) with a spacing of 2.5 cm. The z-position of the layers ranges from z_1 = 5.7 cm to z_8 = 23.2 cm. We assume that the geometric acceptance is defined such that a particle starting from initial point (0, 0, 0) crosses each layer along a straight track with a maximum opening angle of 160 mrad in each of the xz and yz planes. The half size of the eighth layer along the x- and y-axes is equal to the product of z_8 and the $\tan(\theta_{k,x,y})$. A particle passing through this detector will produce a measured hit with coordinates P (x_k, y_k, z_k) on each layer k ($k = 1, 2 \ldots 8$).

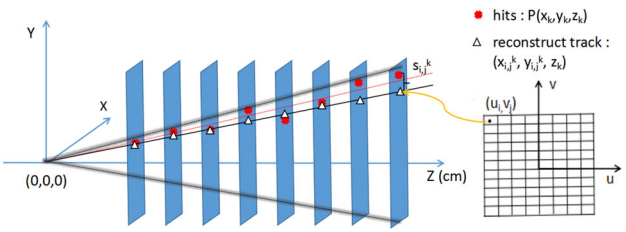

Fig. 1 A cellular unit in the (u, v) parameters space (right) identifies a specific physical track that is a set of hit coordinates P (x_k, y_k, z_k) in the detector (left)

Suppose that every trajectory of a charged particle is a straight line starting from the point (0, 0, 0) and can be described by two parameters (u, v). A two-dimensional parameter space (u, v) is made that coincides with the detector layer eighth that is positioned at $z = 23.2$ cm. It is noted that (u, v) correspond to trajectory patterns in this parameter space plane rather than hits (x, y) of charged particle trajectories in this detector layer plane. In what follows, we discretize the parameter space (u, v) into a number of cells (treated as patterns) and label each cell with (u_i,v_j). The granularity or the binning depends on the resolution we want to achieve on the track parameters. The center of each unit (u_i, v_j) identifies a track through eight planes with an array of intersections (x_k^{ij}, y_k^{ij}, z_k). This is why we create a track mapping between hit coordinates P (x_k, y_k, z_k) and the (u, v) parameter space as shown in Fig. 1. In the current simulation, no experimental effects such as detector noise, detector efficiency or multiple scattering are added.

For each incoming rth hit on layer k, the distance $(s_{ij,k}^r)^2 = (x_k^r - x_k^{ij})^2 + (y_k^r - y_k^{ij})^2$ between the measured hit P (x_k^r, y_k^r, z_k) and the intercept (x_k^{ij}, y_k^{ij}, z_k) corresponding to the cell (u_i,v_j) should be calculated. A weight function (response) is given by Eq. (1).

$$W_{ij,k}^r = \exp\left(-\frac{\left(s_{ij,k}^r\right)^2}{2\sigma^2}\right), \tag{1}$$

where σ is a parameter of the algorithm which can be adjusted to optimize the sharpness of the responses of the

cells. Finally, the (u_i, v_j) cell response $R_{i,j}$ equals the sum of the weights over all hits present in all the detector layers.

$$R_{ij} = \sum_{k,r} W^{r}_{ij,k}. \quad (2)$$

The overall response of the retina algorithm is obtained by calculating R_{ij} for all cells in the (u, v) parameter space. Therefore, for each incoming hit in an event, the algorithm computes the weighted distance of that hit corresponding to each intersection. The weighted distances are accumulated over all event hits. In general, this procedure can be processed for all the cells on all detector layers in a full parallel way. Figure 2 shows a simulation with MATLAB of just one straight track reconstruction using the retina algorithm, as applied on an eight-layer-detector toy model. The parameter space is divided into an array of 212×212 with a grid of 350 µm in the (u, v) plane. The map below represents the intensity of the responses of all the cells, with a maximum set to 8.0. The generated track with (u,v) parameter of (2.3365, 3.0126) cm is identified by the local maximum, which in the present example is approximately (2.3288, 3.0291) cm. Note that a better precision of the extracted track parameters can typically be obtained by computing the central cell of the response cluster and interpolating cells surrounding the local maximum [7]. From this perspective, the precision of the extracted track parameters can be improved.

The usual steps of the artificial retina algorithm include the following:

1. Define the detector geometry, trajectory model of a charged particle and distance measurement and investigate the possible size of the parameter space and granularity;

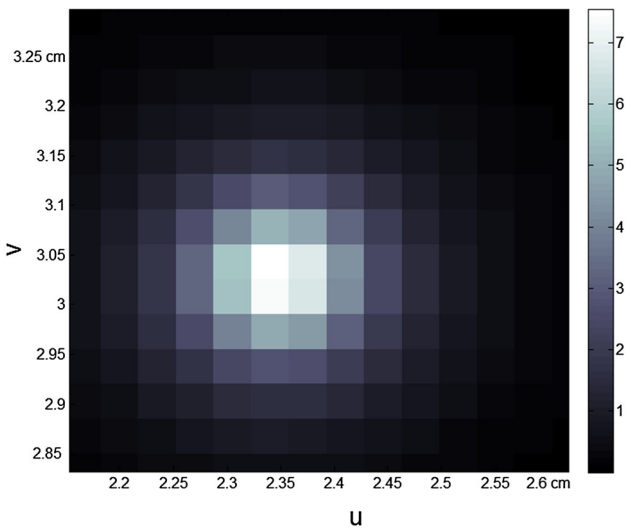

Fig. 2 Subcell excitation levels of retina to one track in the (u, v) parameter space

2. Following the retina algorithm description, use weight distances to compute the response for each cell and find the best cell candidates in the parameter space.

3 Firmware design and implementation

Due to the high parallelism feature of the artificial retina algorithm and the tight processing timing requirements of modern high-energy physics experiments, we choose to use an FPGA device to implement the retina algorithm with the target of finishing one procedure in a fixed latency < 5 µs.

According to Sect. 2.2, we can simply separate the retina algorithm into three parts: The first part is used to identify possible candidate tracks from input hit information, the second part is used to calculate the responses of all the candidate cells and the third part is used to verify the candidate with the largest response. Following this principle, the firmware design consists of five logic stages (Fig. 3):

1. Input: identifying the possible sub-sets of cells as track candidates according to the input hit coordinate information;
2. Distance: computing the distances between hits and the candidate track on each detector layer;
3. Weight: converting the distance s^2 into weight according to Eq. (1);
4. Sum: calculating the response of each candidate cell using Eq. (2);
5. Comparator: finding the best candidate with the local maximum response.

Compared to the add and multiply arithmetical operations, exponential computation is rather difficult to implement on an FPGA. Thus, our first implementation (denoted Floating-Point Design in Table 1) is mainly focused on the precision of the complex weight calculation. We skipped stages 1 and 2 but employed software-calculated distances between hit and pre-selected candidate tracks as the inputs of stage 3. For stage 3 weight, we used Floating-Point Operator cores to implement the exponential function. Since we used an eight-layer-detector toy model, each

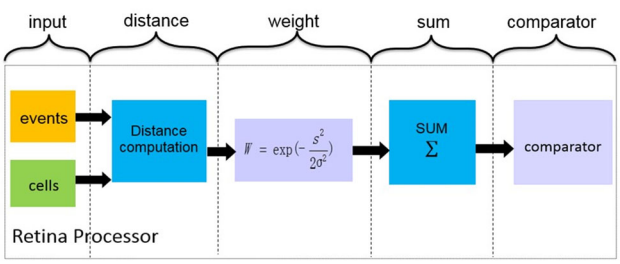

Fig. 3 Description of the function module for the retina processor

Table 1 Optimization procedure of firmware design in the retina processor

Firmware design	Floating-Point	Mixed Floating-Point	Fixed-Point & LUT
(1) Input		Fixed-point	Fixed-point
(2) Distance		Fixed-point	Fixed-point
(3) Weight	Floating-Point core	Floating-Point core	LUT
(4) Sum	Floating-Point core	Fixed-point	Fixed-point
(5) Comparator	Floating-Point core	Fixed-point	Fixed-point

weight stage involved eight exponential operations in parallel. During the initial development, our goal was to determine how many cells could be processed simultaneously and how long it would take for one whole procedure. The hardware platform we used was a KC705 [10] development board that was equipped with a 7K325T-2FFG900 FPGA. As it turned out, the Floating-Point design could host up to six cells with a usage of nearly 70% of the FPGA logical resource, and one procedure took a fixed latency of 1.97 µs [11], which has been proved by the waveform captured on an oscilloscope. We used 32-bit base-2 format Floating-Point with a precision of 0.0001% in this design, and the exponential operation results calculated by Floating-Point Operator cores are the same as those calculated by software.

In the first implementation, we have proven the correctness of the calculation. However, on the one hand, without stages 1 and 2, we cannot regard it as a complete procedure. On the other hand, it will place a large limitation on the application of the algorithm in a real case if only six cells can be supported. To answer these two questions, we then focus on the completion and optimization of the retina algorithm.

We improved our implementation in a more efficient way, and Table 1 describes our firmware completion and optimization steps in the retina processor. Because the detector has a typical pixel size of 15 µm × 15 µm, 12 bits for the fraction part in cm units can provide a precision of 2.5 µm, which can fully meet computation precision requirements. Therefore, we choose to use 32-bit fixed-point format for all the logic stages except stage 3 weight. By performing a detailed analysis of the first completed implementation (mixed Floating-Point Design in Table 1), we notice that the usages of LUT (70.72%) and FF (42.32%) are abnormally higher than for DSP (17.14%). We consider that the reason derives from the Floating-Point core application, and we therefore decide to employ the Look-Up Table (LUT) method to evaluate the weight function instead of Floating-Point core. Note that we retain the same number of cells (six) in both the mixed Floating-Point design and Fixed-Point & LUT design to compare performance. The FPGA resource cost (e.g., FF, LUT and DSP) and the processing latency are summarized in Table 2. It is shown that by using the Fixed-Point & LUT design to implement the full artificial retina algorithm, we can obtain considerable gains in resource and latency usage, and this solution has therefore further investigated.

Figure 4 provides the details of this approach. To better describe the firmware architecture, we introduce some definitions. Each cell of the parameter space is mapped into a logic module termed an engine, and each engine consists of eight collateral processors corresponding to eight layers and one accumulator. The data format is mainly based on a 32-bit fixed-point representation (1 bit for sign, 19 bits for integer and 12 bits for fraction). These trajectories of charged particles are randomly generated, the spatial coordinates P (x_k, y_k, z_k) of which are 32-bit words encoding the corresponding geometric x- and y-coordinates (12 bits for the fraction in the 16-bit fixed-point format of x or y). These hit coordinates are stored in the read-only memory block called events. The block named cells store 32-bit words of x- and y-coordinates for the pre-computed intersections (x_k^{ij}, y_k^{ij}, z_k) from parameter space (u, v). Each incoming hit selects the corresponding (x_k^{ij}, y_k^{ij}) and performs the subtraction. The outcomes of the subtractions (d_x, d_y) between hits and intersections are successively squared and summed. For a weight function in Eq. (1), we use a fast LUT method to simplify and approximate the function values. Since the value of the weight function is range limited, the input s^2 should be quantized in a well-defined range. The distributed values of the weight function can be represented using a table of 32 entries and stored as a 32 × 32 bit Look-Up Table. The LUT is implemented with a ROM, and our design remains the five least significant bits (LSB) of the distance square s^2 as the address index to this ROM. However, because of the AXI [12] particularity in the BRAM IP core, the weight function has to be mapped into a 32 × 1024 bit ROM, which is common to all the engines. The results of the LUT are accumulated for each engine and compared. One notices that each incoming hit is fed to all the engines and is computed in parallel. After all the hits have been fed to all the engines, a local maximum is identified. The cell candidates with their coordinates and maxima are output for further analysis on a PC. So far, we have implemented two complete firmware designs, mixed Floating-Point and Fixed-Point & LUT in the retina processor. Performance verification, the

Table 2 Performance comparison of FPGA resource usage and latency for different designs

Firmware design	Floating-Point	Mixed Floating-Point	Fixed-Point & LUT
FF (%)	21.71	42.32	9.8
LUTRAM (%)	13.95	15.24	8.46
LUT (%)	65.87	70.72	7.25
BRAM (%)	0.8	3.03	13.8
DSP (%)	20.71	17.14	11.43
Cycle	197	156	68
Latency (μs)	1.97	1.56	0.68

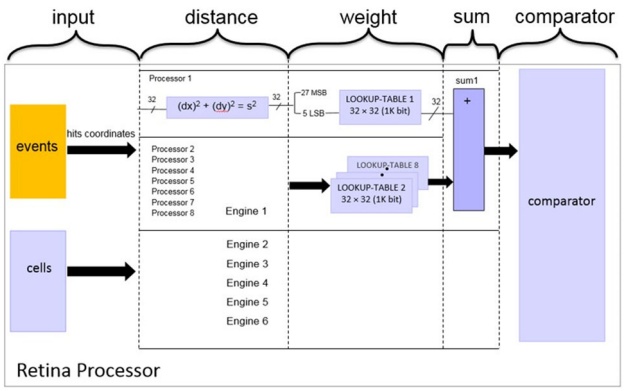

Fig. 4 Firmware architecture of the Fixed-Point & LUT design in the retina processor

simulation results and hardware limitations will be discussed in the next section.

4 Performance and results

4.1 Performance verification

This R&D work focuses on the study of the retina processor prototype. We choose the KC705 board as the hardware platform and validate two implementations of the retina algorithm: a mixed Floating-Point and Fixed-Point & LUT design. Every block of the retina processor developed in the VIVADO Design Suite is programmed using the Verilog language. We monitored the FPGA resource usage and latency associated with both designs and performed verification using ILA and VIO tools. Both implementations run at a clock frequency of 100 MHz with the possibility of running at a higher frequency (200 MHz tested).

We measure the latency of the Fixed-Point & LUT retina processor with an oscilloscope running at 10 Gbps. The histogram result in Fig. 5 shows the time difference between the data loading by the retina processor and the asserting of the maximum cell from the comparator module. The result in Fig. 5 shows that the latency mean value equals 679.5 ns with a standard deviation of 36 ps. Because the standard deviation is so small compared to the system clock frequency (100 MHz), we can assume that our Fixed-Point & LUT implementation has a fixed latency. In addition, it meets the typical trigger latency requirement that is to be less than a few microseconds.

4.2 Simulation results

To quantify the retina algorithm track reconstruction performance, we compare both implementation results with software simulation results. In general, the spatial resolution of the retina algorithm is mainly driven by the total number of cells in the parameter space (u, v). In the current simulation, we generate random tracks crossing our eight-layer-detector made of pixels with a size of 15 μm × 15 μm. No detector spatial resolution or efficiency effects are added to the simulation. The center position of the hit pixel is used as the hit position in the layer. With that detector geometry, different numbers of cells in the (u, v) parameter space have been studied. We investigated the impact of the parameter cell number on the RMS of the difference between the generated and reconstructed parameter positions in the eighth layer. Taking into account the hardware resource usage with the growth of cell number and the degradation of RMS when lowering the number of cells, we have finally chosen to divide the (u, v) parameter space into 44,944 cells with a granularity of 350 μm. The results are shown in Figs. 6 and 7, in which we histogram the differences between the generated and the computed track parameters. Figure 6 shows, respectively, the difference histograms between the generated and reconstructed parameters positions along the u coordinate (left) and v coordinate (right) for the software simulation (top) and Fixed-Point & LUT implementation in the retina processor (bottom). Figure 7 shows the same histograms between the mixed Floating-Point implementation (bottom) and software simulation (top) results. In Table 3, the Q12 RMS column lists the spatial resolutions (cm) in the (u,v) plane for both implementations against the software simulation. The simulation result shows that the retina algorithm has a good performance because it can always identify the best parameter location with our ideal detector. Under this condition, the uniform distribution of the

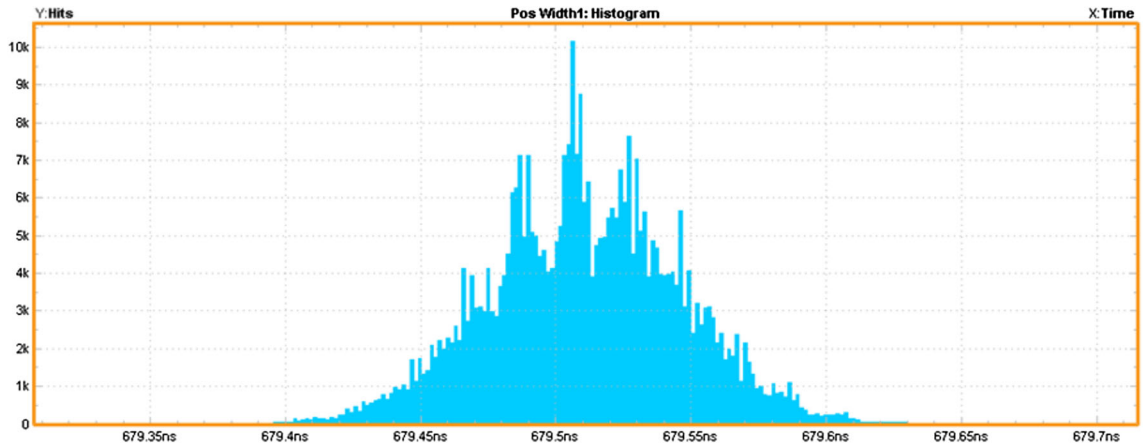

Fig. 5 Latency histogram result

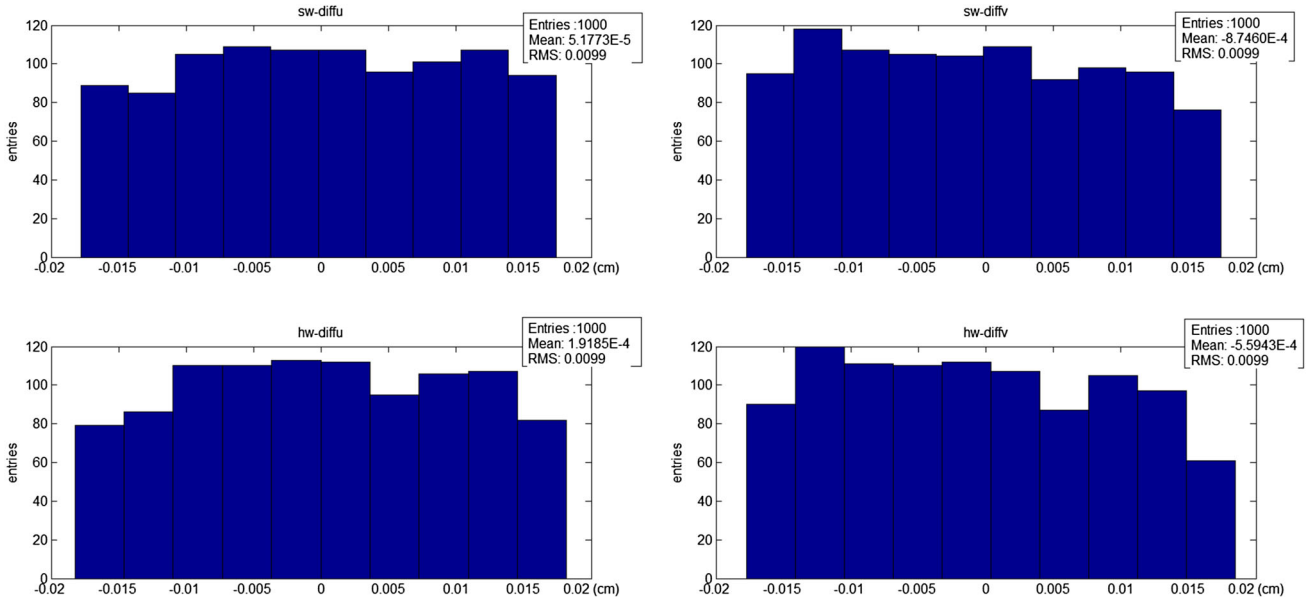

Fig. 6 Comparison of the spatial resolution (cm) in the (u, v) plane for the Fixed-Point & LUT implementation against the software simulation (sw)

differences between the generated and reconstructed track parameters shown in Figs. 6 and 7 follows the distribution of the track parameters that we generated, which is a uniform distribution. In addition, our retina processor is capable of providing a track resolution of 99 μm in both implementations and their performance is close to the level of the software results. Considering the FPGA resource usage and latency, we investigate in Sect. 3 the Fixed-Point & LUT design offers a better choice for retina algorithm implementation. Note that both implementations use a 12-bit fixed-point for the fraction, and the Floating-Point processing block uses 8 bits for exponent and 23 bits for fraction.

In the Fixed-Point & LUT design, we also investigate the fixed-point spatial resolution for 12 bits and 10 bits for the fraction. The performance comparisons between Fixed-Point & LUT implementation against the software simulation are summarized in Table 3. It is indicated that when we change the number of fixed-point fractional bits from 12 bits (Q12) to 10 bits (Q10), the spatial resolution (cm) in the (u,v) plane of our retina processor degrades by 25% but remains much better than the granularity of the (u,v) parameter space.

4.3 Hardware limitation

Because the Fixed-Point & LUT design provides high potential for the growth of parameter space cells that can be processed at one time, our goal is speed, and we want to find the limitation on the KC705 hardware platform.

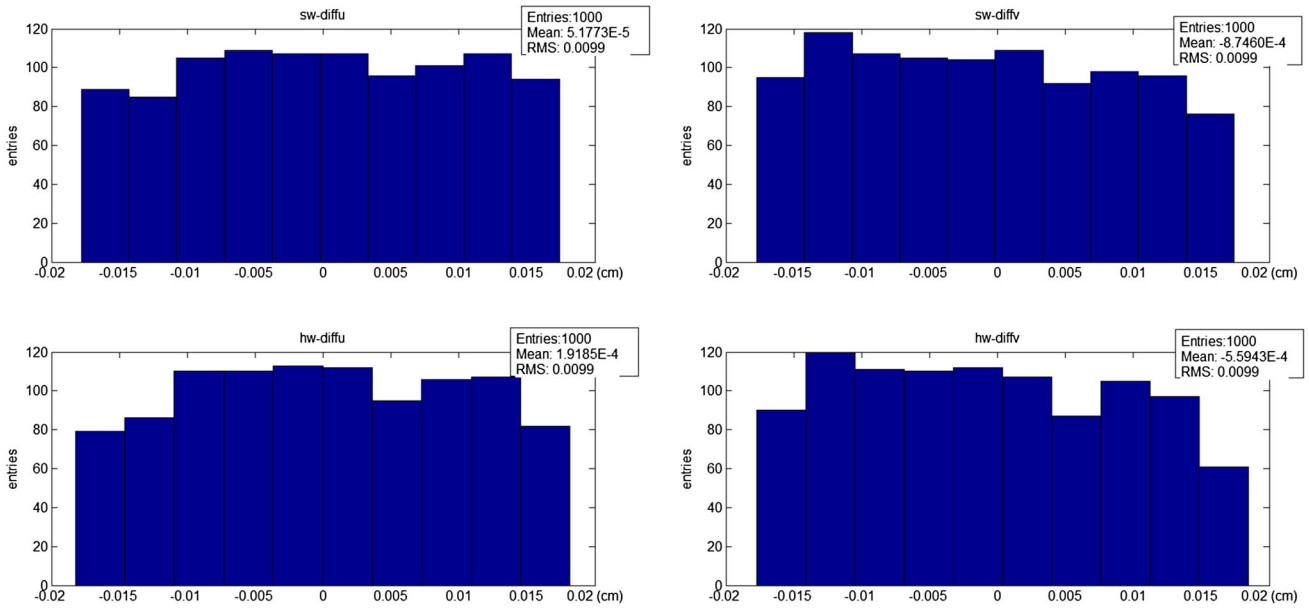

Fig. 7 Comparison of the spatial resolution (cm) in the (u, v) plane for the mixed Floating-Point implementation against the software simulation (sw)

Table 3 Performance comparison of the spatial resolution

Firmware design	Q12 RMS (cm)	Q10 RMS (cm)
Sw_diffu	0.0099	0.0099
Fixed-Point & LUT_diffu	0.0099	0.0132
Mixed Floating-Point_diffu	0.0099	
Sw_diffv	0.0099	0.0099
Fixed-Point & LUT_diffv	0.0099	0.0141
Mixed Floating-Point_diffv	0.0099	

Table 4 FPGA resource usage and latency for different numbers of cells

Cell number	6	25	49
FF (%)	7.12	25.48	49.38
LUTRAM (%)	0.36	1.4	2.82
LUT (%)	4.83	15.7	30.28
BRAM (%)	3.7	6.4	8.2
DSP (%)	11.43	47.62	93.33
Cycle	64	71	74
Latency (μs)	0.64	0.71	0.74

The plane size of the detector model is only a few square centimeters, and a 32-bit fixed-point coordination representation is somewhat of an overkill; therefore, we choose to use 20-bit fixed-point with 12 bits in the fraction part in the following Fixed-Point & LUT design. In addition, BRAMs are mainly used for eight 32×32 bits Look-Up Tables per cell. To save memory resources, we use distributed RAM to replace BRAM, which is sufficient for our design requirement. These optimizations further economically improve the Fixed-Point & LUT design performance. We then expand the system size from 6 cells to 25 cells. As given in Table 4, the main logic resource costs (LUT, FF, and DSP) of the FPGA almost linearly increase (Fig. 8), and the limitation will mainly depend on the DSP. Therefore, we try to use all the DSP and manage to implement 49 cells, with the comparison as follows.

We observe that the resource usage is not very balanced, and the DSPs are mainly used for the 16 multiplication operations inside each cell; therefore, we can continue to increase the number of cells by using LUT and FF to implement the multiplication operation. Finally, we fit 61 cells with 49 DSP-based and 12 LUT- and FF-based multiplications into one KC705.

5 Conclusion

In this note, we study high parallel FPGA-based implementations of the artificial retina algorithm for fast track reconstruction in a trigger system. We apply retina to a simple ideal detector ignoring magnetic field effects and present the results for a KC705 platform using both mixed Floating-Point cores and Fixed-Point & LUT solutions. The performance of the implementations, including latency, hardware resource budget and retina track reconstruction performance, has been investigated. The

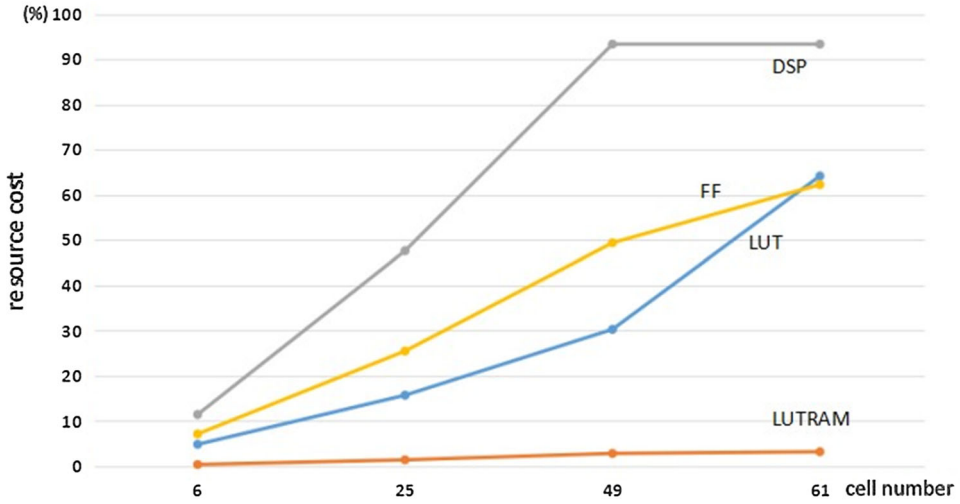

Fig. 8 FPGA resource usage for different numbers of cells

limitation of the hardware platform is also under discussion. These results are quite promising. Moreover, this approach implies that we have to traverse the whole parameter space in order to find the best track candidate. Considering a more realistic case, in our example, in order to obtain a spatial resolution of 350 µm with a detector layer size of 7.42 cm by 7.42 cm, the whole parameter space needs to be 210 by 210, which yields a general cell number of approximately 40000. To traverse such a huge parameter space with a fixed latency below 5 µs, we have to work on a way to enable our algorithm repetitively.

In addition, another interesting geometry is a tracking detector with a barrel shape made of multiple concentric cylindrical layers surrounded by a strong (several Tesla) magnetic field. In this case, the reconstruction of bent charged particle tracks is expected to be more complex. We will target the application of the retina algorithm under more complex cases in the near future.

References

1. F. Palla, M. Pesaresi, A. Ryd, Track finding in CMS for the level-1 trigger at the HL-LHC. J. Instrum. **11**, C03011 (2016). https://doi.org/10.1088/1748-0221/11/03/c03011
2. W. Ashmanskas, A. Bardi, M. Bari et al., Silicon vertex tracker: a fast precise tracking trigger for CDF. Nucl. Instrum. Methods A **447**, 218 (2000). https://doi.org/10.1016/s0168-9002(00)00190-x
3. J. Anderson, A. Andreani, A. Andreazza et al., FTK: a fast track trigger for ATLAS. J. Instrum. **7**, C10002 (2012). https://doi.org/10.1088/1748-0221/7/10/C10002
4. L. Ristori, An artificial retina for fast track finding. Nucl. Instrum. Methods A **453**, 425 (2000). https://doi.org/10.1016/s0168-9002(00)00676-8
5. R.O. Duda, P.E. Hart, Use of the Hough transformation to detect lines and curves in pictures. Commun. ACM **15**, 11 (1972). https://doi.org/10.1145/361237.361242
6. A. Abba, F. Bedeschi, M. Citterio et al., The artificial retina processor for track reconstruction at the LHC crossing rate. J. Instrum. **10**, C03018 (2015). https://doi.org/10.1088/1748-0221/10/03/c03018
7. R. Cenci, F. Bedeschi, P. Marino et al., First results of an "artificial retina" processor prototype, in *EPJ Web of Conferences*, vol. 217, p. 00005 (2016). https://doi.org/10.1051/epjconf/201612700005
8. Xilinx PG060, Floating-Point Operator V7.1 LogicCore IP Product Guide. https://china.xilinx.com/pg060-floating-point.pdf. Accessed 4 Oct 2017
9. A. Piucci, Reconstruction of tracks in real time in the high luminosity environment at LHC. Dissertation, Universit'a degli Studi di Pisa (2013)
10. Xilinx UG810, KC705 Evaluation board for the Kintex-7 FPGA user guide. https://china.xilinx.com/ug810_KC705_Eval_Bd.pdf. Accessed 4 Feb 2019
11. Z. Song, G. De Lentdecker, J. Dong et al., Study of hardware implementation of fast tracking algorithms. J. Instrum. **12**, C02068 (2017). https://doi.org/10.1088/1748-0221/12/02/C02068
12. AMBA®AXI™ and ACE™ Protocol Specification AXI3™, AXI4™, and AXI4-Lite™ ACE and ACE-Lite™. http://www.gstitt.ece.ufl.edu/AXI4_specification.pdf. Accessed 28 Oct 2011