Article

Implementing a Hybrid Method for Shack–Hartmann Wavefront Spots Labeling on FPGA

Ammar Abdullah [1,*], Aoife Brady [1], Daniel Heinig [1], Peter Krause [1], Matthias Goy [1], Klaus-Peter Döge [2] and Andreas Tünnermann [1,3]

[1] Fraunhofer Institute for Applied Optics and Precision Engineering IOF, 07745 Jena, Germany; aoife.brady@iof.fraunhofer.de (A.B.); daniel.heinig@iof.fraunhofer.de (D.H.); peter.krause@iof.fraunhofer.de (P.K.); matthias.goy@iof.fraunhofer.de (M.G.); andreas.tuennermann@iof.fraunhofer.de (A.T.)

[2] Department of Electrical Engineering and Information Technology, Ernst-Abbe-Hochschule, 07745 Jena, Germany; klaus-peter.doege@eah-jena.de

[3] Institute of Applied Physics, Abbe Center of Photonics, Friedrich-Schiller-University, 07745 Jena, Germany

* Correspondence: ammar.abdullah@iof.fraunhofer.de

Abstract: This paper presents a real-time implementation of a hybrid connected component labeling method for processing the Shack–Hartmann wavefront sensor's images for an adaptive optics (AO) system. The output image of a wavefront sensor is an image of spots. During the sensor's operation, it can happen that highly distorted wavefronts (WF) may cause the spots to shift outside of their sub-aperture, which may lead to the reduction of the AO system performance. This article explains the benefits of high-performance computing and parallel processing of a field programmable gate array (FPGA). The objective is to calculate the centroids of these spots. A hybrid labeling method was investigated to fulfill this purpose. First, this method was implemented using a forward and backward scan with a respective mask for each scan. Additionally, a relabeling process is applied after labeling each line, and it is carried out in both directions. After labeling, several processing units were implemented in parallel to calculate centroids. Each unit is responsible for calculating the centroid of one label. The system runs in real time with a latency of one frame, which means the output image is a fusion of a current frame and the centroids of the previous frame. Forward and backward labeling requires a large amount of memory, which is the reason for limiting the investigation to forward labeling only. The forward labeling was successfully implemented, and the centroids were detected under minimum spot distortion conditions. This forward labeling implementation also runs in real time with significant latency reduction to calculate the centroids, which leads to minimizing the overall AO system latency, enabling faster computation and correction in addition to reducing the memory usage to 1% when compared to the forward and backward labeling usage of 81% as an advantage for the hardware implementation.

Keywords: Shack–Hartmann wavefront sensor (SHWFS); adaptive optics (AO); connected components labeling (CCL); field programmable gate array (FPGA)

check for updates

Citation: Abdullah, A.; Brady, A.; Heinig, D.; Krause, P.; Goy, M.; Döge, K.-P.; Tünnermann, A. Implementing a Hybrid Method for Shack–Hartmann Wavefront Spots Labeling on FPGA. *Electronics* **2024**, *13*, 1221. https://doi.org/10.3390/electronics13071221

Academic Editor: Longqing Cong

Received: 6 February 2024
Revised: 18 March 2024
Accepted: 25 March 2024
Published: 26 March 2024

1. Introduction

The aim of adaptive optics is to reduce wavefront distortions due to, e.g., turbulence. This is accomplished by measuring the incoming wavefront with a wavefront sensor and correcting it with a wavefront corrector such as a deformable mirror, lens, or spatial light modulator. Adaptive optics is used in wide fields such as astronomy, microscopy, or laser material processing. To build an adaptive optics system, three main components are usually required: a wavefront corrector, a wavefront sensor, and a controller. A wavefront sensor sends the incoming data to the controller. The controller makes use of the received information in order to compensate for the distortion by controlling a wavefront corrector [1,2]. Several wavefront measurement techniques have been proposed during recent decades, such as the Shack–Hartmann wavefront sensor, curvature wavefront sensor, pyramid wavefront sensor, and holographic wavefront sensor, and each one has its advantages and disadvantages, with each being suitable for special usage. The most commonly used presently is the Shack–Hartmann Wavefront Sensor (SHWFS), which is one of the standards for atmospheric AO systems due to its accuracy, reliability, high resolution, and robustness. It also has the benefit of high availability since it is based on existing technologies like cameras and micro-lens arrays. This collectively makes it a very straightforward approach since the wavefront (WF) is measured and corrected. Theoretically, the holographic wavefront sensor, for example, is faster than the SHWFS, and it has different functionality than SHWFS, but it is still a subject of research with problems related to inter-modal crosstalk and strong aberrations [3]. The holographic sensing method would allow direct measurements of wavefront modes with the readout speed of photodiodes, which could be up to the MHz range [4]. The Shack–Hartmann sensor can capture data in real time, which makes it suitable for real-time systems [5]. This method has been used for the successful demonstration of adaptive optics correction of atmospheric turbulence in breadboard and free-space experiments, e.g., classical laser [6–8] and quantum communications [9,10]. In ref. [11], a Connected Components Labeling method (CCL) was used to detect the position of the spots, and the calculation of the centroid was carried out after the spot detection. The authors in ref. [12] presented a stream-based center of gravity method for centroids estimation but still leads to some centroid detection uncertainty in high wavefront distortions. Other researchers tried to increase the wavefront sensing speed using a GPU or CPU [13–15]. Others have used the artificial neural network and machine-learning-based method [16,17], and they achieved good results for wavefront sensing and system speed. To determine the centroids of the spots, we consider the spots as objects in the image, and those objects have to be detected. There are many methods for object detection [18], and in this article, we chose the connected component labeling. It basically scans the image and decides which pixels are connected to form an object. There are different methods within the CCL itself to determine whether the pixels are connected or not. The first of these methods is the one-pass method, which was previously implemented by Bailey [19] and Klaiber [20]. State-of-the-art implementations allow parallel processing up to 32 pixels per clock cycle with a clock cycle rate over 100 MHz on a Virtex 6 FPGA [20].

The second method is the two-pass method, which is first mentioned by Rosenfeld and Pfaltz [21]. Modern two-pass labeling methods are described by Lacassagne and Zavidovique [22]. One implementation of a two-pass CCL algorithm for video processing on FPGA was presented in an article from Jablonski and Gorgon [23] where a 8192×8192 pixel-sized image is labeled in under 5 s, with an implementation running at 100 MHz. It uses stream-based processing with a small local window in the first pass and a merge table lookup in the second pass, giving one frame latency. Several high-speed parallel algorithms exist for connected components labeling [24]. While such algorithms give considerable speed improvement over the classic algorithm, they still require a large number of logic elements.

The third method is the multi-pass method. An FPGA implementation was published in 1999 by Crookes and Benkrid [25]. Another FPGA implementation in ref. [26] uses very simple processing but requires multiple passes through the image to be completely labeled, and this leads to increasing the complexity of the region shapes. The intermediate image between passes has to be stored in a frame buffer, making such an algorithm unsuited for real-time processing.

In this work, a hybrid method using both two-pass and multi-pass methods is proposed. The hybrid method can be applied for two image scans only, a forward and a backward image scan, which can reduce the overall complexity and memory usage. In each image scan, the two-pass method can be applied on each line instead of the entire image to reduce the equivalence table size, which can lead to less memory usage. The paper is structured as follows: the first section reviews the image-processing operations that are necessary to use for this application as shown in Figure 1. The second section explains the hybrid labeling method. The hardware implementation, simulation, implementation results, and discussion are in sections four, five, and six. The last section is devoted to the closure.

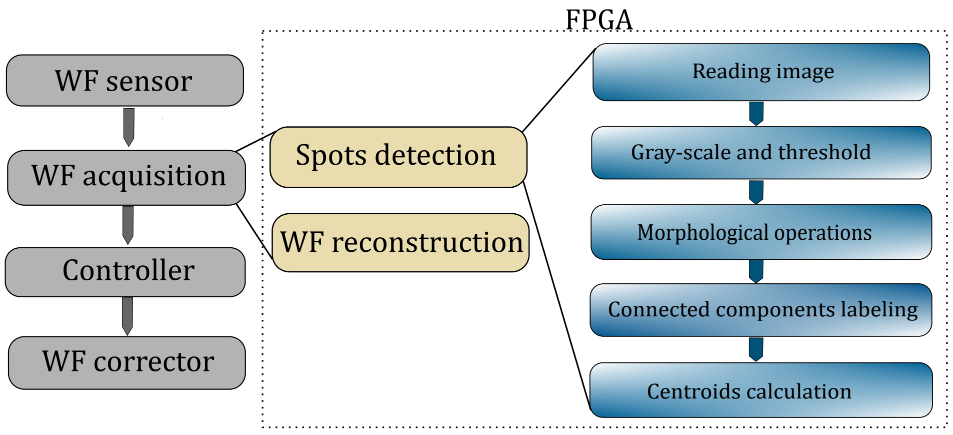


Figure 1. An AO (Adaptive Optic) system diagram and the FPGA image-processing stack.

2. Image Processing

2.1. Threshold

The first step in the image processing is to apply a threshold filter. This filter is applied to the grayscale image, and the output is a binary image. First, we detect the threshold value, and we apply this value to the image. All pixels above this value will take a value of one, and all the pixels below this value will take a value of zero. The success of the threshold depends on the selected threshold value. One of the well-known algorithms is Otsu's method [27], which automatically finds the optimal threshold intensity. The algorithm returns a single intensity threshold that separates pixels into two classes (the background and the foreground). The algorithm exhaustively searches for the threshold that minimizes the within-class variance, defined as a weighted sum of variances of the two classes. Weights ω_b and ω_f are the probabilities of the two classes separated by a threshold t, σ_b and σ_f are variances of these two classes. It searches for the threshold intensity by separating the image into two classes: foreground and background. After calculating the different probabilities of the threshold values using Equation (1), the within-class variance could be calculated using an equation for each threshold value t. The threshold value then could be chosen, which gives the maximum value of the within-class variance:

$$\sigma_W^2 = \omega_b \sigma_b^2 + \omega_f \sigma_f^2 \tag{1}$$

where:

$$\omega_b = \sum_{i=1}^{k} P(i) \quad , \quad \omega_f = \sum_{i=k+1}^{L} P(i) \tag{2}$$

and we can define the two class means as follows:

$$\mu_b = \sum_{i=1}^{k} \frac{iP(i)}{\omega_b} \quad , \quad \mu_f = \sum_{i=k+1}^{L} \frac{iP(i)}{\omega_f} \tag{3}$$

The two class variances can be computed as follows:

$$\sigma_b^2 = \sum_{i=1}^{k} (i - \mu_b)^2 \frac{iP(i)}{\omega_b} \tag{4}$$

$$\sigma_f^2 = \sum_{i=k+1}^{L} (i - \mu_f)^2 \frac{iP(i)}{\omega_f} \tag{5}$$

We can calculate what is called the total variance from the previous equations as:

$$\sigma_T^2 = \sigma_B^2 + \sigma_W^2 \tag{6}$$

with

$$\sigma_B^2 = \omega_b[\mu_b - \mu]^2 + \omega_f[\mu_f - \mu]^2 \tag{7}$$

and

$$\mu = \omega_b\mu_b + \omega_f\mu_f \tag{8}$$

Minimizing the within-class variance σ_W is equivalent to maximizing the between-class variance σ_B. The desired threshold corresponds to the minimum σ_B [27,28].

2.2. Morphological Filters

Morphological filters are a wide range of image-processing operations that process images on the basis of their shapes. For better object detection, a low-noise image is preferred. To reduce the noise in the image and to make the objects clearer, we chose to use dilation and erosion, which are two fundamental morphological operations. These operations usually use a structuring element for probing and reducing or expanding the shapes contained in the input image, which can be divided into two parts: foreground pixels and background pixels. Our aim here is to focus on the foreground pixels since they are considered to be our objects. Figure 2 shows an example of an image before and after applying these two filters.

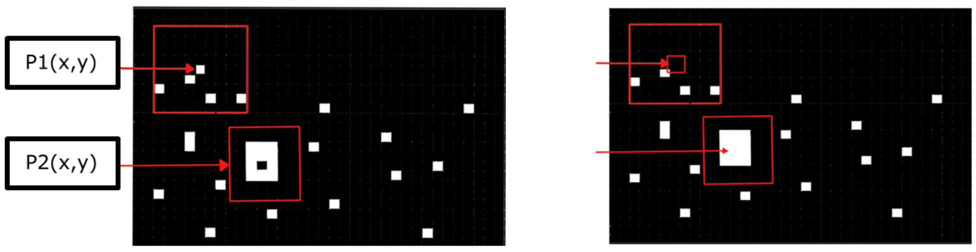


Figure 2. Decreasing the object's area by applying erosion on P1 and increasing object's area by applying dilation on P2.

Erosion is typically used to decrease the boundaries of the foreground pixels in the image. It applies an AND operation between the current pixels and the structure element (mask); thus, the area of the object will be reduced after this filter is applied. So, it can be used to remove the noise from the image. Dilation is used to increase the boundaries of the region in the image. It applies an OR operation on the image, so the area of the foreground pixels grows, and if there are holes in this area, they become smaller. The four neighbors of the pixel P1(x,y) are zeros, so it will be considered noise when applying Erosion. The pixel P2(x,y) has four neighbors with a value of one, so it will take the value of the neighbors after applying the dilation [28,29].

2.3. Connected Component Labeling Algorithms

The connected component labeling algorithm is one of the most common algorithms in image processing to detect the objects in the image. The input of this algorithm is a binary image, and the output is a labeled image. All the pixels that take the same label will be considered to be one object. It separates the objects in the image from the background and tries to give every connected object a labeled value. Connected components labeling serves a specific purpose in binary image-processing, focusing on identifying and labeling connected regions. In contrast, other methods [30–33] are more versatile and are often applied to grayscale or color images for various image segmentation and labeling tasks. In the following, common labeling methods within the CCL method are presented. We consider the input to be a binary image, which is determined after grayscaling, thresholding, and morphological filters. The output is a labeled image, and every label represents one object.

For the single-pass method, the image is first scanned once in a forward direction with a forward mask. During the scanning pass, the label of the central point of the mask is set to the smallest label of the points of the mask connected with the center point. During the scanning, a new label is created if no point of the mask is connected to a center point. This algorithm has one problem, which occurs when one of the objects has a U-shape, as shown in Figure 3. When applying the 4-connected neighborhood and when the mask reaches the position L1 (or L2), it is seen that the algorithm has labeled one connected object with three labels, 1, 2, and 3, which means that they have been considered to be three objects, because, during the checking level, there were no pixels connected to the pixel of the object on which the mask is applied. The two-pass method can be the solution to this problem. Since we are dealing with simple objects (spots), this method seems to be a good solution [34,35].

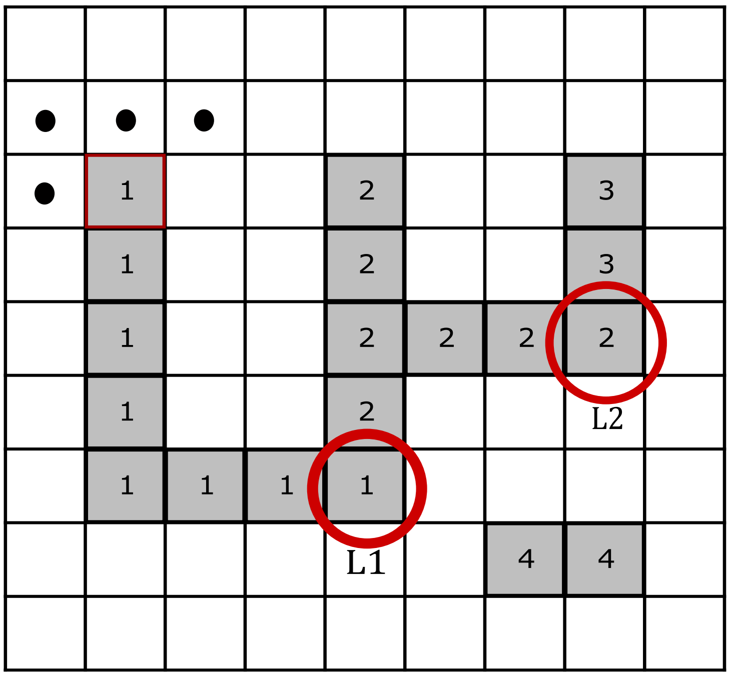


Figure 3. Labeling conflict: the image shows the labeling result for a single-pass scan, with a pattern as seen at the red square in the top left, over two objects with pixel numbering related to the detected objects. However, objects 1, 2, and 3 should have been detected as one object. The false labeling was due to the U-shape conflict originating in L1 and L2.

Another method is the two-pass method, which requires two scanning passes and an intermediate step for conflict resolution. The forward scan is identical to the forward scan of the one-pass labeling. The center point of the mask obtains the lowest label of the pixels in the mask connected with it. If no pixel is connected with the center point, a new label is assigned. A conflict occurs if two or more pixels in the scanning mask that are connected with the center point have different labels. This conflict is recorded in an equivalence table. After the first pass, a lookup table (LUT) is created, and final labels are assigned to temporary labels. This process is referred to as equivalence resolving.

In the second pass, the labels of the label mask are updated using the LUT. The critical point of the pass methods is resolving the equivalences. With many conflicts, the classical methods are not able to label an image within a reasonable time frame. To reach the required performance and due to the restricted memory bandwidth, the equivalence table has to be limited. Another solution is using the multi-pass method.

In the multi-pass method, the image is first scanned in a forward direction with a forward mask and, after that, in a backward direction with a backward mask. The forward scan is equal to the forward scan of the one-pass labeling. Forward and backward scanning is repeated until no changes occur anymore. Then, the labeling is complete. Since multi-pass labeling is purely Windows-based, it is very well suited for an FPGA implementation. However, this method is not suitable for labeling large images with complex structures. It seems that combining two methods can solve the problem that one method presents and make use of the advantages of both.

3. Hybrid Labeling Method

In this work, a hybrid method using both two-pass and multi-pass methods is proposed. The method can be applied for two image scans only, a forward and a backward image scan, which can reduce the overall complexity and memory usage. In each image scan, the two-pass method can be applied on each line instead of the entire image to reduce the equivalence table size.

3.1. Forward Scan

For a better understanding of the forward scan, pixel groups can be mathematically defined as:

$$N_f(x,y) = \{P(x-1,y-1), P(x,y-1), P(x+1,y-1), P(x-1,y)\} \tag{9}$$

which is the pixel's neighbors group $N_f(x,y)$ for the forward mask as shown in Figure 4.

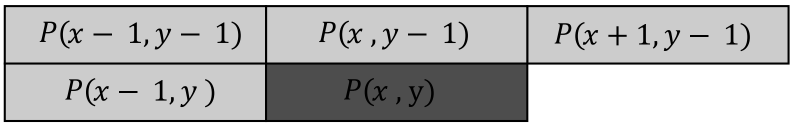


Figure 4. Shape of included pixels around the target pixel P(x,y) of the forward mask.

Neighbor groups where each pixel has the same value

$$A_f(x,y) = \{P(m,n) \mid P(m,n) \in N_f(x,y), P(m,n) = P(x,y)\} \tag{10}$$

Pixel's neighbors label value $AL_f(x,y)$

$$AL_f(x,y) = \{L(m,n) \mid P(m,n) \in A_f(x,y)\} \tag{11}$$

Label_min which represent the minimum value of the labels $AL_f(x,y)$

$$Label\_min(x,y) = min(AL_f(x,y)) \tag{12}$$

Label_max which is the maximum value of the labels $AL_f(x,y)$

$$Label\_max(x,y) = max(AL_f(x,y)) \tag{13}$$

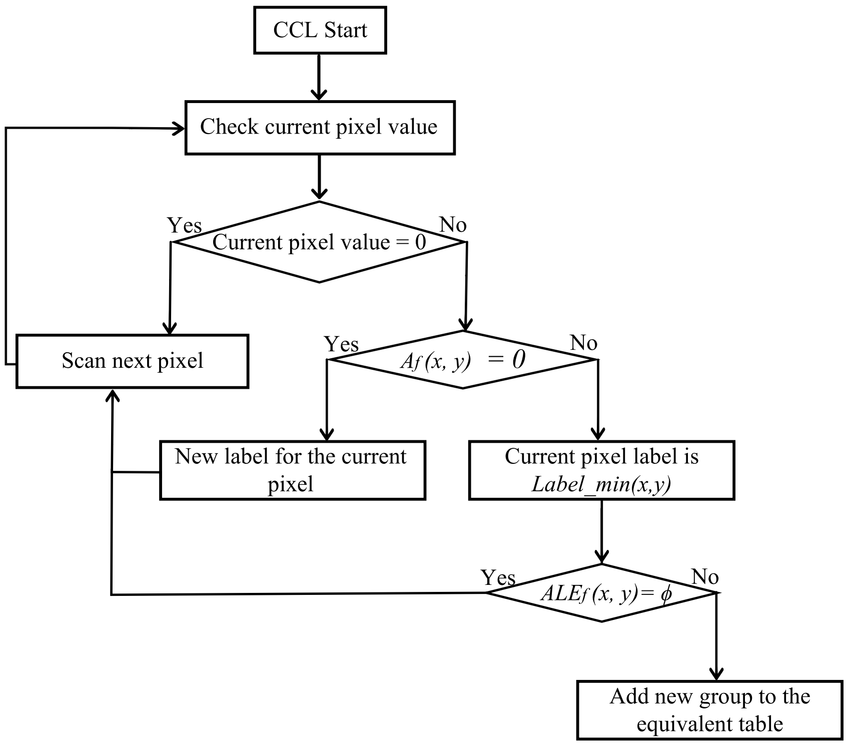
Pixel's neighbor equivalence label

$$ALE_f(x,y) = (Label\_min(x,y), Label\_max(x,y)) \tag{14}$$

Equivalence table group

$$EQ\_TABLE_f = \bigcup_{1 \le x,y \le N} ALE_f(x,y) \tag{15}$$

In the forward scan, the image is scanned in a forward direction with a forward mask. Every single line is scanned two times. The first pass is for labeling the line pixels according to the neighbor's value, and the second pass is for relabeling these pixels according to the equivalent table's values. The current pixel is given a label value according to the flowchart in Figure 5.



CCL Start
Check current pixel value
Current pixel value = 0
Yes
No
Scan next pixel
$Af(x, y) = 0$
Yes
No
New label for the current pixel
Current pixel label is *Label_min(x,y)*
$ALEf(x, y) = \phi$
Yes
No
Add new group to the equivalent table

Figure 5. Pixel's labeling flowchart.

3.2. Backward Scan

For the backward scan, it is actually the same group's definitions, but the pixel's neighbors group is different as in Figure 6. Since the scanning is in the opposite direction, the pixel's neighbors group can be defined as:

$$N_b(x,y) = \{P(x-1,y+1), P(x,y+1), P(x+1,y+1), P(x-1,y)\} \tag{16}$$

	$P(x,y)$	$P(x-1,y)$
$P(x+1,y-1)$	$P(x,y-1)$	$P(x-1,y-1)$

Figure 6. Shape of included pixels around target pixel P(x,y) of the backward mask.

The image is scanned in a backward direction with a backward mask. The purpose of the backward scan is to avoid the problem with the U shapes in the image. This scan also has two passes for every line; the first pass will label the pixels according to the pixel's neighbors label value, and the second pass will relabel the pixels according to the value in the stored equivalent table.

The input of this scan are labels between 2 and 128, and during this pass, a label value will be given to every pixel according to the flowchart in Figure 5. Also, as previously mentioned in the second pass, the input is the labels from the previous pass, which are numbers between 2 and 128. During this pass, the pixel's values are relabeled according to values stored in the equivalent table. So $Label(x,y)$ will take $Labe\_min(x,y)$ value. Then, move to the second pixel until l all the objects in the image are labeled [35,36]. Due to the restricted size of the memory, the maximum number of conflicts and the labels have to be limited. For this reason, the number of labels was limited to 128.

3.3. Centroid Calculation

Considering a digital image $I(x,y)$ of size M×N, do the 2D-Moments are defined as:

$$m_{ij} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} x^i y^j I(x,y) \tag{17}$$

The corresponding centroids are defined as:

$$\overline{x} = \frac{m_{10}}{m_{00}}; \overline{y} = \frac{m_{01}}{m_{00}} \tag{18}$$

where m_{10} and m_{01} are the first moments. The centroid's coordinates can be written as (C_x,C_y) for simplicity:

$$C_x = \frac{1}{n} \sum_{i=1}^{n} x_i; C_y = \frac{1}{n} \sum_{i=1}^{n} y_i \tag{19}$$

and n here is the area of the object, which is represented by the number of pixels that belong to the object, x_i and y_i are the coordinates of pixels that belong to the object [28,37].

After CCL, the ordering of the spots to their corresponding lenslets is required. Different methods have been presented for centroid ordering, like straight line centroid ordering [38] and spiral method for sorting spots [39]. Based on the ordered spots, the slopes and actuator values for an AO system can be calculated using the well-established methods described in refs. [5,40].

4. Hardware Implementation

This method was implemented on Zybo z20 FPGA board of type xc7z020clg40 from Digilent Pullman USA manufacturer to test and validate the architecture. A resolution of 800 × 600 (Super Video Graphics Array SVGA) is used in this system, and the output of the system is read using the UART (Universal Asynchronous Receiver-Transmitter) communication port. Figure 7 shows the general system architecture, where every process is an input for the next one. After reading the image, it should be transferred into a grayscale image and then to a binary image after applying the threshold. After that, the noises are removed, and the holes are filled out using morphological operations. The result is then fed to the unit to determine the connected objects in the image, and then the centroids of the objects can be calculated, and the result can be displayed on the monitor using HDMI protocol.

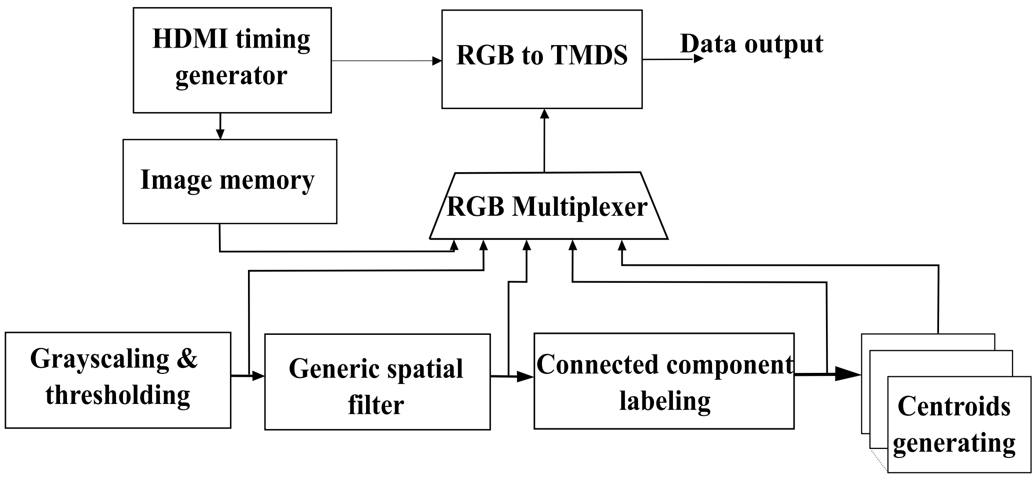


Figure 7. Image-processing structure in the FPGA.

4.1. Image Memory

The image memory is a ROM (Read Only Memory) that contains the image of spots. The original image is of size 800 × 600. To reduce the total size of the system memories, images of size 512 × 512 are used instead. It starts by cropping the original image around the center to a size of 512 × 512 since all the spots are located around the image center and are included in that region of interest.

4.2. Grayscaling and Threshold Unit

The input of this unit is RGB data, which is represented as 24 bits. The most significant bit (MSB) 8-bits are dedicated to red color, the least significant bit (LSB) 8-bits are dedicated to blue color, and the middle 8-bits are dedicated to green color. The output of this unit is a binary image. After reading the RGB image, each pixel of the image is converted to grayscale using the following equation:

$$I = (0.299 \times R) + (0.587 \times G) + (0.114 \times B) \tag{20}$$

where R, G and B are values of red, green, and blue channels. The coefficients in Equation (19) are floating-point numbers, and doing the mathematical operations with floating points can slow down the process because it can take several clock cycles to calculate. To avoid the usage of floating-point numbers, all the coefficients in Equation (19) are multiplied by 255 and then divided by $2^8 - 1$. This division is equivalent to a right shift by 8 bits. Equation (19) can be written as follows:

$$I \approx ((76 \times R) + (149 \times G) + (29 \times B))/(2^8 - 1) \tag{21}$$

Using this equation, the result can be obtained in one clock cycle. After that, the grayscale image is converted to a binary image using a threshold, which can be described as follows:

$$binary\_image = \begin{cases} 1, \text{if I} > \text{T} \\ 0, \text{else} \end{cases}$$

The threshold is determined after testing some samples of spot images and is set to a value of 129. Then, the threshold can be calculated dynamically after generating the histogram of the grayscale image according to the Otsu method [29]. The output of this unit is an input for the generic spatial filter unit.

4.3. Generic Spatial Filter Unit

This unit applies image erosion and dilation operators in order to remove noise, in addition to the isolation of individual elements and joining of disparate elements. The output of this unit is $filter\_out$ signal , which contains binary pixels. As a sliding window of size N × N is scanned over the image, the sum of pixels within this window is calculated and compared to a threshold value M. If the sum is superior to M, the window is considered to be fully white; otherwise, it is considered to be fully black. The $filter\_in$ signal is applied to a shift register of size 512. At the beginning of each new line in the image, the parallel output of the shift register is fed to the line1 register, which in return is fed to the line2 register and so on until the lineN register. Therefore, the line1, line2, . . . , lineN registers contain the last N lines of the scanned image. A sum unit reads these lines within a sliding window of size N, and a comparator compares the sum result to a threshold M to determine the value of the binary output $filter\_out$. The size of the sliding window is 4 × 4, and the threshold is set to 5. For this reason, the previous 4 lines are registered in an array of size 4 × 512. The current sliding window moves with the current input pixel, but it is applied on the previous 4 lines (Figure 8a,b). The filter output is a function of the total sum of input binary pixels in the sliding window, which can be described as follows:

$$filter\_out = \begin{cases} 1, \text{if sum of binary input pixels} > \text{M} \\ 0, \text{else} \end{cases}$$

The output pixel of the filter is generated with a delay of 4 lines and 2 pixels relative to the input pixel.

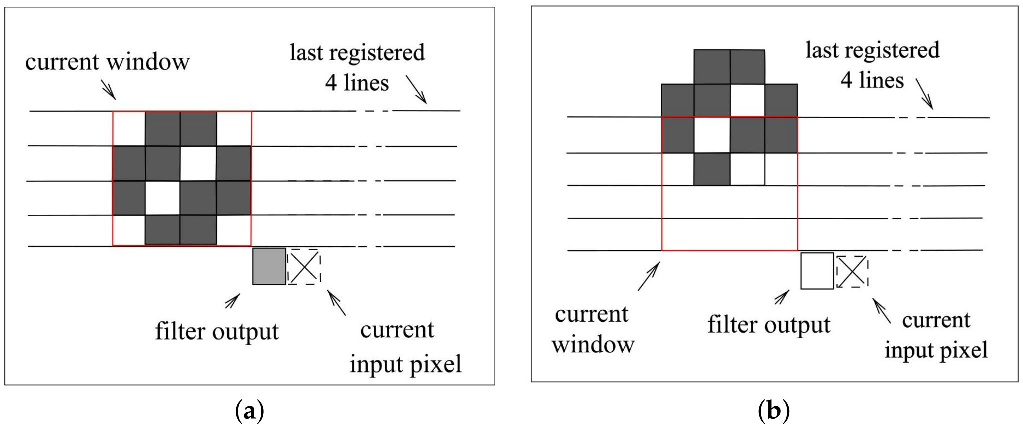


Figure 8. Applying the spatial filter in a 4 × 4 mask. In (**a**), the mask considered the pixel as an object's pixel, while as a noise in (**b**) because the number of pixels in the mask is less than 5.

4.4. Connected Components Labeling Unit

This unit applies a forward and backward-connected components labeling algorithm in order to associate each connected spot with a unique label. The image is first scanned in a forward direction with a forward mask and then in a backward direction with a backward mask, as shown in Figure 9.

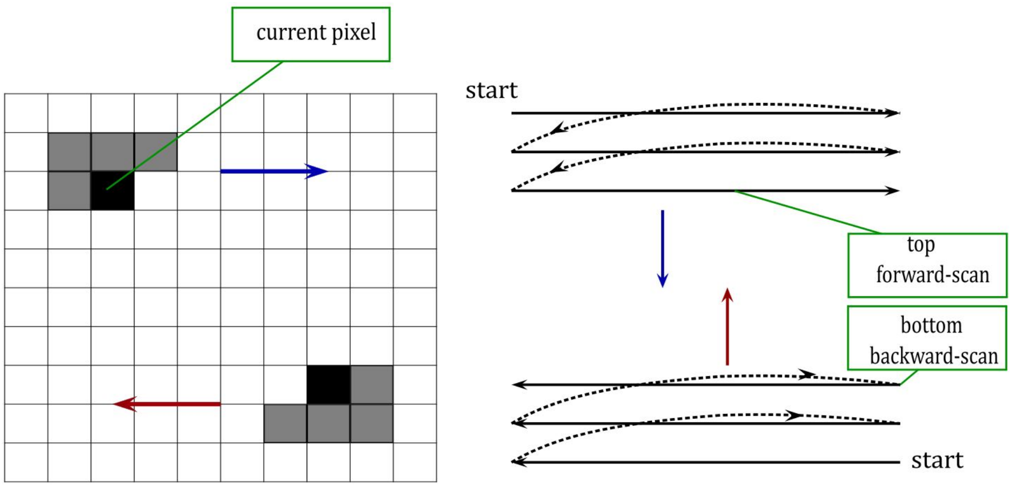


Figure 9. CCL with the forward scan (blue arrow), where the scan starts from the top left to the bottom right and the backward mask from the bottom right to the top left (red arrow).

During each scanning pass, the algorithm makes two passes over each line. The first pass merges, and the second pass relabels. In the first line pass, a new label is created if no point of the mask is connected with the center point; otherwise, the label of the central point of the mask is set to the smallest label of the points of the mask connected with the center point. In this pass, temporary labels are assigned, and equivalences are recorded. In the second line pass, temporary labels are replaced with the smallest label of its equivalence class. The relabel signal, which works with 120 MHz clock frequency, is active during the second pass, which comes directly after the first pass and after a new line.

The architecture of this unit is the first processing element (PE1) that handles the forward labeling, and a second processing element (PE2) handles the backward labeling. RAM image 1 and RAM image 2 are two random access memories (RAM) used in a flip-flop manner. When writing the result of PE1 in one of the two memories, PE2 is reading the previous result from the other memory. PE1 result is always written in a forward direction, and PE2 is reading the previous result in a backward direction. as shown in Figure 10.

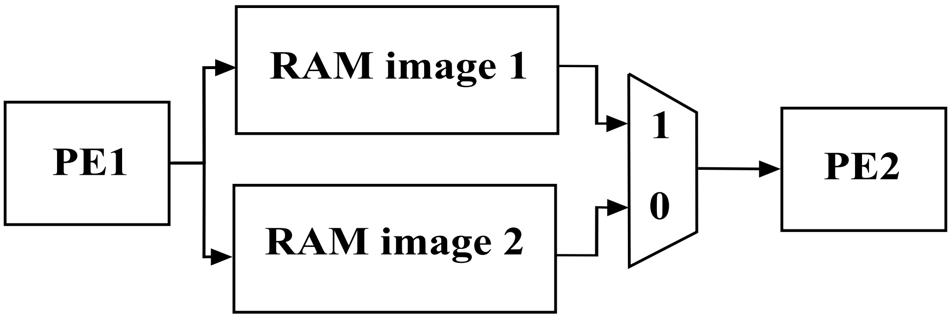


Figure 10. Connected component labeling unit. The image is held by two RAMs and a multiplexer to decide which RAM should be read when the other is full.

To read and write in a backward direction, the following address signal is generated as follows:

$$BIAddr = (2^{18} - 1) - imageAddr \tag{22}$$

where $(2^{18} - 1)$ corresponds to the last cell address in a memory of size 512×512.

PE1 and PE2 have the same architecture, which consists of line memory (RAM line), memory for the equivalent table (RAM equivalent), and memory for the line in the reversed image (RAM line 1). The line 1 memory is filled by the minimum labeling values *L_min* during the first pass (merge) and is read in the second pass (relabel) to find the equivalence classes. The RAM equivalent stores the equivalent classes. During the merge pass, each *L_min* is saved at the address of *L_max*, constructing a relation between *L_min* and *L_max*, which are connected to the same central point. In the second pass, the labels are updated using the values that have been stored in the RAM equivalent to generate new labeling values. Also, the reading and writing addresses are generated in these units.

Once the foreground pixel is detected and the neighbors pixels are zeros, then the current pixel takes a new label value (here *L_min* = *L_max* = *new_value*), and the mask continues to the next pixel and so on. When the mask values are not empty, then the values that are already stored in *L_min* and *L_max* are compared, and the smallest value *L_min* is attributed to the *L_max* (which is 2 here). The first pass is clocked by $pixel\_clk = 40$ MHz (800×600 resolution). In this pass, the *L_min* and *L_max* labels are generated for each current pixel (mask central point). The first pass is applied to each line of 512 pixels in size. The second pass is applied on the same line, but processing must end before a new line comes. For this reason, the second pass proceeds at a higher frequency, which is clocked by $relabel\_clk = 3 \times 40$ MHz $= 120$ MHz, and this guarantees that the second pass finishes before the arrival of a new line.

4.5. Generate Centroid Unit

This unit generates x and y coordinates of the spot center labeled *i*, where each connected spot has a unique label. X and Y coordinates are calculated using Equation (19), and they are updated at the end of each frame. In this system, 100 units are used to generate a centroid, and each unit has a unique parameter *i*, which corresponds to a given label *i*. Therefore, each unit works separately to generate its own centroids. Another extra unit is the draw cursor unit, which is responsible for drawing a cursor around the centroids. Since the purpose of this unit is to draw a cursor surrounding the center of every labeled object, it has to be applied for every spot in the image. Another unit is responsible for generating the timing signals for the HDMI protocol, and an extra one is responsible for the UART protocol.

The last unit is RGB2TMDS, which generates four TMDS (Transition-minimized Differential Signaling) differential pairs, which are the inputs of a standard HDMI connector.

The first pair corresponds to the TMDS clock, which is the pixel clock (40 MHz). The other three pairs transmit the red, green, and blue pixels. HDMI requires that we scramble the data and add 2 bits per color lane, so we have 10 bits instead of 8 per color, and the link ends up transporting 30 bits per pixel. The scrambling and extra bits are needed by the HDMI receiver to properly synchronize to and acquire each lane [41,42]. Ten-bit values have to be sent for every pixel clock period. Therefore, we need a serial clock of period 40 MHz $\times$ 5 = 200 MHz working on the rising and falling edge. This clock is generated by the timing clocks generator unit. The encoding method is 8 b/10 b encoding. A two-stage process converts an input of 8-bit into a 10-bit code with particular desirable properties. In the first stage, the first bit is un-transformed, and each subsequent bit is either XOR or XNOR transformed against the previous bit. The encoder chooses between XOR and XNOR by determining which will result in the fewest transitions. The ninth bit encodes the operation that was used. In the second stage, the first eight bits are optionally inverted to even out the balance of ones and zeros and, therefore, the sustained average DC level; the tenth bit encodes whether this inversion took place.

5. Simulation

To implement a particular application, one needs to define an algorithm model and an architecture model on which the algorithm will be implemented. The algorithm model is the reference model that needs to be validated after description and simulation in a software design environment. Then, the architecture model can be described and simulated in a hardware design environment. The architecture model simulation results can be compared to the reference model simulation results in order to verify the transition from algorithm to architecture. MATLAB R2014a version was used to describe and simulate the connected component labeling algorithm. VIVADO 2020.2 version was used to describe and simulate the architecture model. After reading the image in MATLAB, we were able to detect the threshold value after applying the Otsu method [29] of 0.5059, which corresponds to a 129 grayscale value.

After determining the threshold value, the binary image is also read in MATLAB to apply forward and backward labeling. Two MATLAB functions are defined to store the results of both the forward and backward scan in a labeling matrix. A simulation was made in MATLAB to emulate the hardware version of the scanning procedure with the purpose of testing the main labeling functions. This was achieved by representing the PE1 and PE2 as MATLAB functions and two matrices as RAM image 1 and RAM image 2 to store the results of the labeling. The flowchart of the labeling in the forward direction is shown in Figure 11.

The forward labeling procedure will check the pixel's value and will give the right label to the pixel according to neighbors labels. The result will be stored in the equivalence table.

The backward labeling inputs are labels read from the matrix. It checks each pixel´s label, assigns the proper label according to neighbor labels, and then stores the results in the equivalence table. The pixels are relabeled on the basis of the equivalent table values after reading each line.

Figure 12a shows the simulation result after applying the forward labeling, and Figure 12b shows the result of applying the forward labeling followed by a backward labeling. Figure 12a shows that one object is in different colors, which means that the same object has been labeled twice. The backward labeling solves this problem as shown in Figure 12b. This algorithm took 0.652 s labeling time to finish.

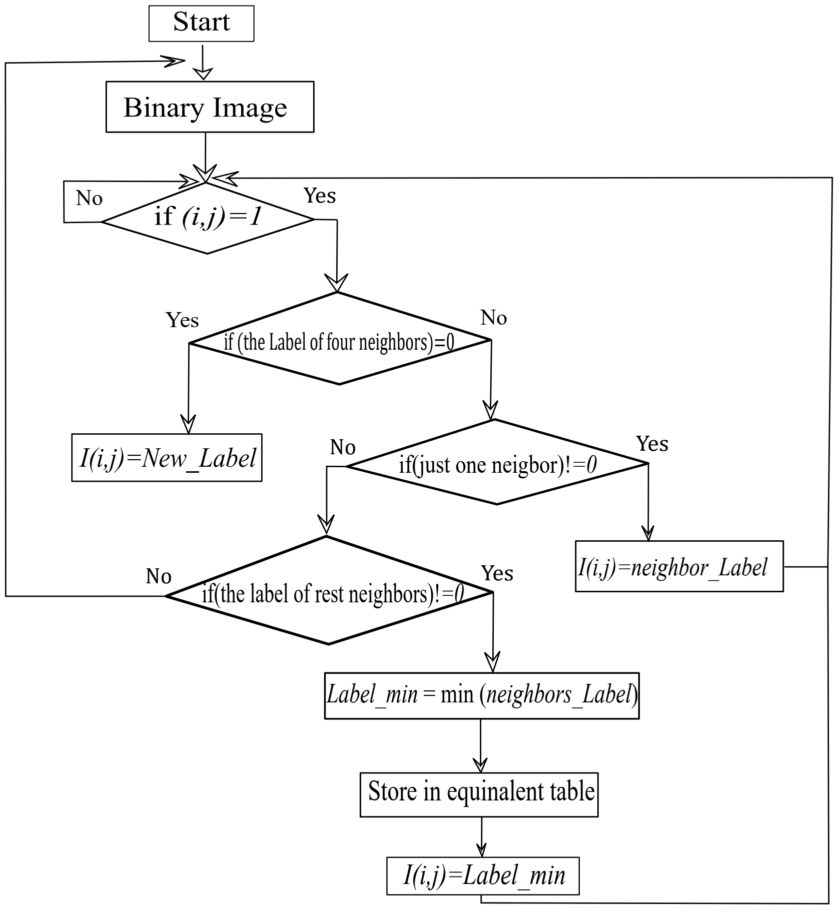

Start
Binary Image
if (i,j)=1
No
Yes
if (the Label of four neighbors)=0
Yes
No
I(i,j)=New_Label
if(just one neigbor)!=0
No
Yes
I(i,j)=neighbor_Label
if(the label of rest neighbors)!=0
No
Yes
Label_min = min (neighbors_Label)
Store in equinalent table
I(i,j)=Label_min

Figure 11. Forward labeling flowchart.

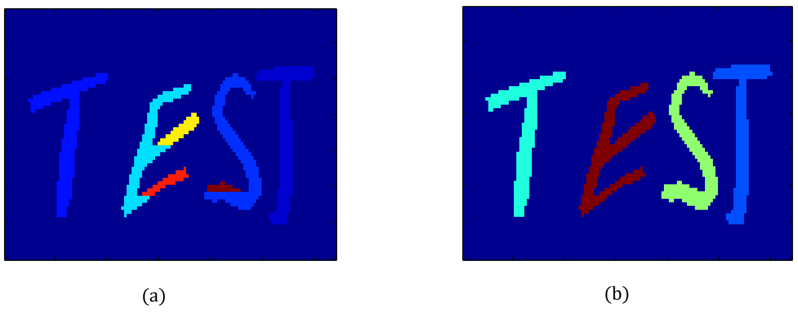


Figure 12. Forward labeling where the letter "E" has been detected as three objects with the letter "S" as two objects as in (**a**). Using forward and backward labeling, the letters have been detected as one object each in (**b**).

6. Implementation Results and Discussion

Figure 13 illustrates the system timing diagram. During the frame reading, the frame is converted to a grayscale image and then to a binary image after applying a threshold. A spatial filter is then applied using a sliding window of size 4 $\times$ 4, which requires saving the last 4 lines, and therefore, a delay of 4 lines is introduced in this phase.

The output of the filter is subject to the forward labeling scan, and the result is written in a memory, which is scanned in the next frame to apply backward labeling. Therefore, a latency of one frame is required since the backward scan cannot start until the forward scan is finished.

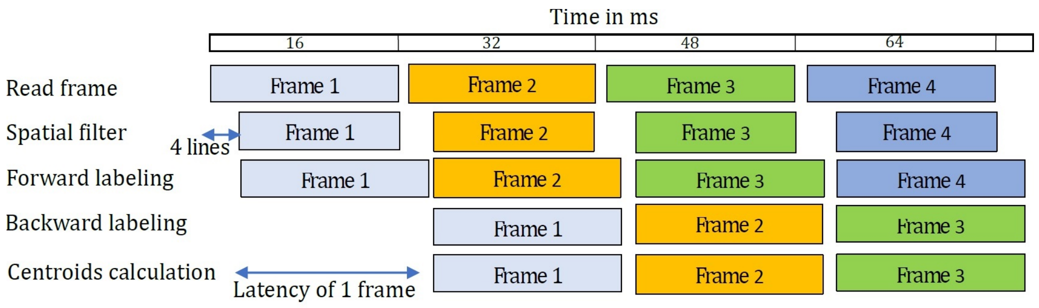


Figure 13. System timing diagram with a 4-line time delay and one frame delay for backward scan.

The output of the backward labeling is applied to the next unit, which generates centroids. Each unit calculates the x and y coordinates of a given connected spot. Later in the next frame, cursors are drawn around the centroids using draw cursor units. A latency of one frame is required to generate centroids, and a latency of two frames is required to draw cursors. Figure 14a shows the result of the forward labeling after applying the selected threshold, which was fixed to 129, and Figure 14b shows the result of the backward labeling after applying the spatial filter which looks like an inverted image due to the backward scan.

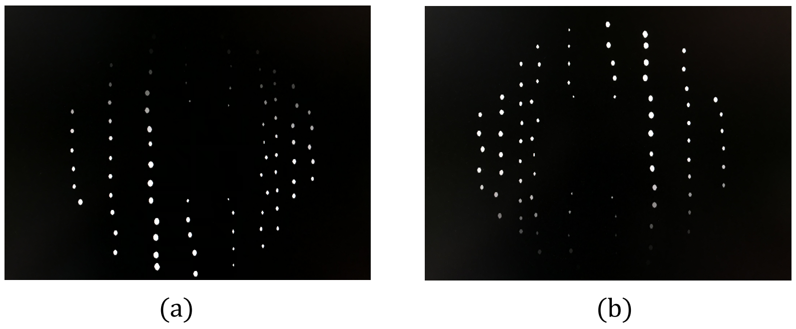


Figure 14. Forward labeling (**a**) and backward labeling (**b**) of the processed image and (**b**) is inverted due to the backward scan where the last processed pixel from (**a**) is the first processing pixel for (**b**).

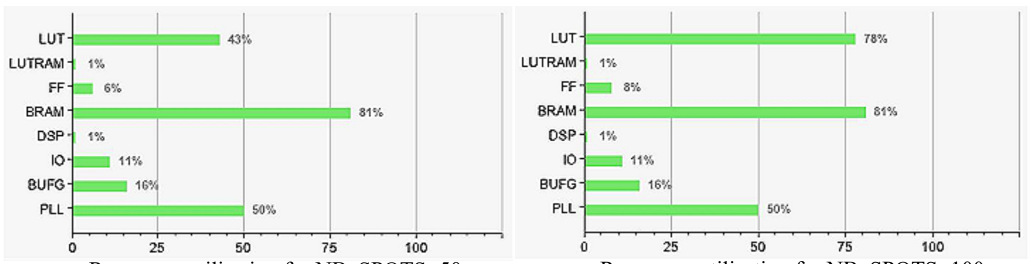
The architecture is generically described in VHDL and Verilog by defining some parameters that allow us to implement changes easily without committing errors. The defined parameters are:

- NB_SPOTS: The maximum number of detected spots. This parameter is used to generate centroid and cursor units several times equal to NB_SPOTS.
- LABEL_WIDTH: The number of bits considered to present the labels. Therefore, the number of labels is between 2 and 2^{NB_SPOTS} -1 This parameter is used to define the labeling memories, such as the equivalence table. The case here is LABEL_WIDTH = 7, which means that the number of labels is between 2 and 128.
- AREA_WIDTH: The number of bits considered to present the area of a spot. The parameter is used in the centroid unit to define adder widths. AREA_WIDTH = 8, which means that a spot can contain 256 pixels.

The system is implemented on an FPGA of type xc7z020clg400 (Zybo board). The system works in real time with a latency of 2 frames, and the centroids are generated on the fly without any delay. Table 1 and Figure 15 shows the resources utilization for NB_SPOTS = 50 and NB_SPOTS = 100. The system consumes a great number of block RAM (BRAM), lookup tables (LUTs), and flip-flops (FF).

Table 1. Resources Utilization.

FPGA Zybo	LUT	FF	BRAM
NB_SPOTS = 100	41,908	8925	114
NB_SPOTS = 50	22,806	6034	114



Resources utilisation for NB_SPOTS=50 Resources utilisation for NB_SPOTS=100

Figure 15. Resources utilization decreased when dealing with 50 spots and 100 spots; LUT are lookup tables, FF is the flip flop, and BRAM is the block RAM.

The memory is essentially consumed for buffering purposes between the PE1 unit (forward labeling) and the PE2 unit (backward labeling). The number of LUTs is related to NB_SPOTS, in other words, to the centroid unit, which is repeated NB_SPOTS times. The centroid unit contains two large dividers used for calculating centroid coordinates.

The number of LUTs can be reduced by saving $\sum_{i=1}^{n} x_i$ and $\sum_{i=1}^{n} y_i$ for each label in 3 memories and re-using one centroid unit for calculating all centroids coordinates. In return, this solution requires additional memories and can introduce a delay in centroids calculation.

Another weakness in this architecture can be noticed in the use of a second clock in the relabeling process of each line. A second faster clock has been used to re-read the line for relabeling reasons before the arrival of a new line. Using 2 line-memories allows relabeling to be done on a previous line using the same clock.

The memory usage can be reduced by only using forward labeling, which means frame buffering is not necessary. However, this solution requires that each spot has a circular shape with no missing parts, which in turn requires input images with good conditions and a good threshold value. Table 2 and Figure 16 show the resource utilization for 100 spots (NB_SPOTS = 100) with a noticeable reduction in memory blocks. Figure 17 shows the final result (left) and FPGA utilization when only using forward labeling.

Table 2. Resources utilization of forward labeling only.

FPGA Zybo	LUT	FF	BRAM
NB_SPOTS = 100	48,909	9261	1

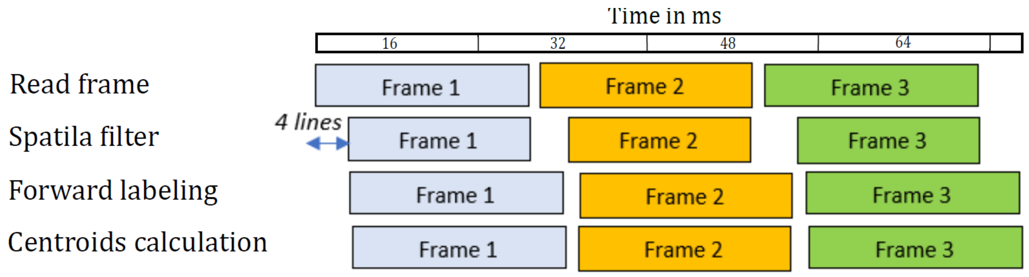
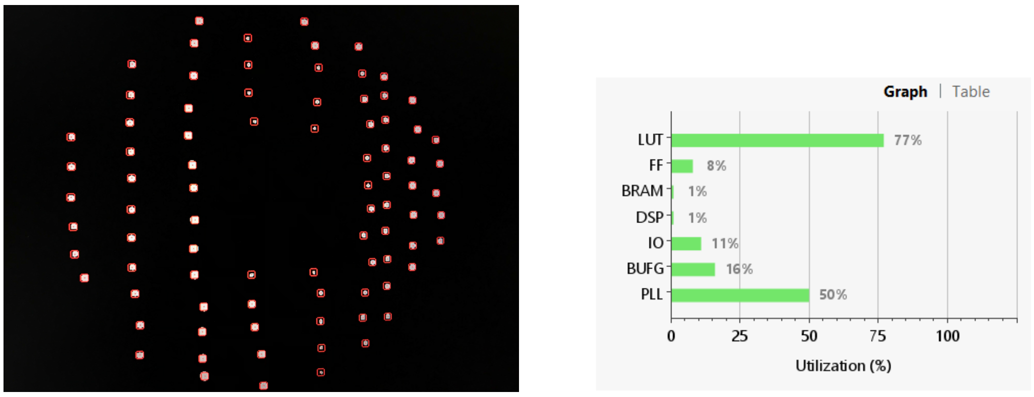


Figure 16. Forward labeling timing diagram and new centroids are generated for each new frame.



Forward Labeling Result

Resource Utilization for NB_SPOTS=100

Figure 17. Forward labeling result, the figure on the left shows the cursor surrounded by the spots centroids while the figure on the right shows the FPGA utilization.

The required time for executing the complete series of steps in the experimental implementation is listed in Table 3. An image of size 512 $\times$ 512 is processed by the implemented architecture (forward labeling) at the clock of 40 MHz. The case here is that the centroid calculations are done in parallel, which requires more LUTs but no additional delays. The timing diagram for the implementation (forward labeling) is also shown in Figure 18 that the centroids calculation of the detected objects (black rectangles, for example) is parallel with the CCL.

By implementing the hybrid method, one can ensure better accuracy for spot detection by adding one frame latency for the image reading and simultaneous centroid calculation. When the turbulence is not so strong, the forward labeling could be enough by calculating the centroids while reading the image. When turbulence conditions are strong (i.e., when the Fried Parameter value is smaller than the lenslet size of SHWFS), the centroids may split [12]. In this particular case, forward and backward labeling may be needed. As an advantage, the method was implemented in a way that is easily configured for different FPGAs.

Table 3. Required executing time.

Sequence	Cycles	clk (MHz)
read image	512 $\times$ 512 = 262,144	40
CCL	262,144	40
x and y for centroid of label i	512	40
division of centroid of label i	1	40

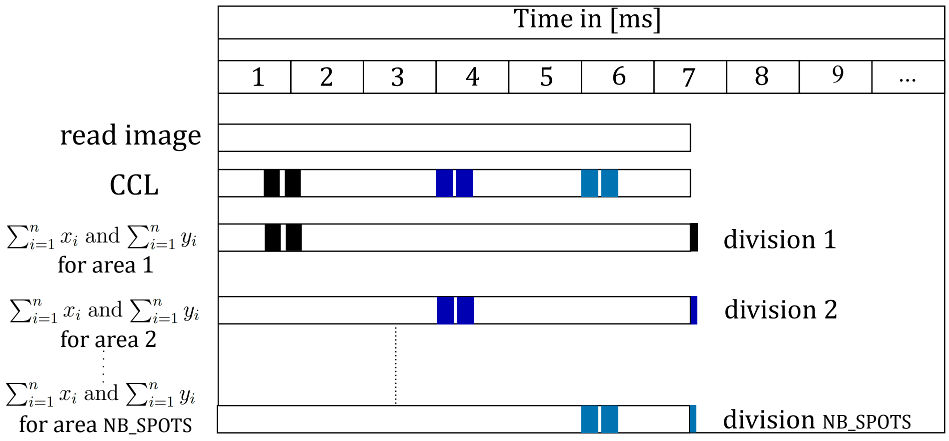

Time in [ms]
1
2
3
4
5
6
7
8
9
...
read image
CCL
$\sum_{i=1}^{n} x_i$ and $\sum_{i=1}^{n} y_i$ for area 1
division 1
$\sum_{i=1}^{n} x_i$ and $\sum_{i=1}^{n} y_i$ for area 2
division 2
$\sum_{i=1}^{n} x_i$ and $\sum_{i=1}^{n} y_i$ for area NB_SPOTS
division NB_SPOTS

Figure 18. Implementation timing diagram where all the centroids are on the fly calculated after detecting the spot.

7. Conclusions

In this paper, we have implemented a real-time spot-detection system for the Shack–Hartmann Wavefront Sensor on FPGA using a hybrid connected components labeling architecture. Hybrid labeling combines forward and backward image labeling methods with a two-pass method for line relabeling.

The proposed architecture runs in real time, and the calculation of spot centroids is made on the fly. The centroids calculation in our method has been done in one clock cycle. After the object detection while scanning the image, the centroid calculation unit will be triggered in parallel and run with a 40 MHz FPGA clock signal, which means that after the object is detected, it takes only 25 ns to calculate its centroid. Other methods, like in ref. [43], run with a faster FPGA clock signal (100 MHz) and still take more than one clock cycle to calculate the centroids. In contrast, this architecture consumed a great number of memory blocks and LUTs. The number of LUTs is related to the maximum number of spots that could be detected.

For the backward-forward labeling, a latency of one frame is required, consuming 81% of memory blocks. The memory consumption can be reduced to 1% when only using forward labeling. This method is new because of the combination of two methods to overcome the weakness of each one, the pixels conflict of the single-pass method, and the memory usage of the two-pass method. On the other hand, the speed of the single-pass method and the accuracy of object detection of the two-pass method should be used. Its advantage is that when the objects are complicated and with sharp edges connected, it will be more effective to use this method for object detection, and it will add a delay of one frame as a disadvantage, but still, the calculation of the centroid is on the fly and more robust when dealing with large spots displacement. The AO system provides real-time data processing and parallel computing, which improves the overall measurement speed by providing extra time to apply the correction. Future prospects will implement the architecture on a bigger FPGA development kit so that memory usage will not be a problem. Two extra units will be added as well. One is the data acquisition unit, which is connected directly to the sensor, and the other is an automatic image threshold unit.

Author Contributions: Formal analysis, K.-P.D.; funding acquisition, A.T.; investigation, A.A., A.B. and K.-P.D.; methodology, A.A.; resources, A.A., A.B., D.H., P.K. and M.G.; software, A.A.; supervision, A.B. and K.-P.D.; validation, A.A.; visualization, A.B.; writing—original draft, A.A.; writing—review & editing, A.A., A.B., D.H., P.K., M.G., K.-P.D. and A.T. All authors have read and agreed to the published version of the manuscript.

Funding: Bundesministerium für Wirtschaft und Klimaschutz.

Data Availability Statement: All data supporting this publication classified as confidential and stored according to Fraunhofer IOF Handling Research Material and Data Procedure.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Tokunaga, A.T.; Jedicke, R. Chapter 39—New Generation Ground-Based Optical/Infrared Telescopes. In *Encyclopedia of the Solar System*, 2nd ed.; McFadden, L.A., Weissman, P.R., Johnson, T.V., Eds.; Academic Press: San Diego, CA, USA, 2007; pp. 719–734. [CrossRef]
2. Rigaut, F.; Van Dam, M. Simulating astronomical adaptive optics systems using yao. In Proceedings of the 3rd O4ELT Conference Adaptive Optics for Extremely Large Telescopes, Florence, Italy, 26–31 May 2013.
3. Liu, M.; Dong, B. Efficient wavefront sensorless adaptive optics based on large dynamic crosstalk-free holographic modal wavefront sensing. *Opt. Express* **2022**, *30*, 9088–9102. [CrossRef] [PubMed]
4. Zepp, A.; Gladysz, S.; Stein, K.; Osten, W. Simulation-based design optimization of the holographic wavefront sensor in closed-loop adaptive optics. *Light Adv. Manuf.* **2022**, *3*, 384–399. [CrossRef]
5. Mauch, S.; Reger, J.; Reinlein, C.; Appelfelder, M.; Goy, M.; Beckert, E.; Tünnermann, A. FPGA-accelerated adaptive optics wavefront control. In Proceedings of the MEMS Adaptive Optics VIII, San Francisco, CA, USA, 1–6 February 2014; Volume 8978, p. 897802.

6. Leonhard, N.; Berlich, R.; Minardi, S.; Barth, A.; Mauch, S.; Mocci, J.; Goy, M.; Appelfelder, M.; Beckert, E.; Reinlein, C. Real-time adaptive optics testbed to investigate point-ahead angle in pre-compensation of Earth-to-GEO optical communication. *Opt. Express* **2016**, *24*, 13157–13172. [CrossRef] [PubMed]
7. Brady, A.; Berlich, R.; Leonhard, N.; Kopf, T.; Böttner, P.; Eberhardt, R.; Reinlein, C. Experimental validation of phase-only pre-compensation over 494 m free-space propagation. *Opt. Lett.* **2017**, *42*, 2679–2682. [CrossRef] [PubMed]
8. Brady, A.; Rössler, C.; Leonhard, N.; Gier, M.; Böttner, P.; Eberhardt, R.; Tünnermann, A.; Reinlein, C. Validation of pre-compensation under point-ahead-angle in a 1 km free-space propagation experiment. *Opt. Express* **2019**, *27*, 17840–17850. [CrossRef]
9. Goy, M.; Berlich, R.; Kržič, A.; Rieländer, D.; Kopf, T.; Sharma, S.; Steinlechner, F.O. High performance optical free-space links for quantum communications. In Proceedings of the International Conference on Space Optics—ICSO, Online, 30 March–2 April 2021; Volume 11852, pp. 213–221.
10. Kržič, A.; Sharma, S.; Spiess, C.; Chandrashekara, U.; Töpfer, S.; Sauer, G.; del Campo, L.; Kopf, T.; Petscharnig, S.; Grafenauer, T.; et al. Metropolitan free-space quantum networks. *arXiv* **2022**, arXiv:2205.12862.
11. Mauch, S.; Barth, A.; Reger, J.; Reinlein, C.; Appelfelder, M.; Beckert, E. FPGA-accelerated adaptive optics wavefront control part II. In Proceedings of the Laser Resonators, Microresonators, and Beam Control XVII, San Francisco, CA, USA, 7–12 February 2015; Volume 9343, p. 93430Y.
12. Kong, F.; Cegarra Polo, M.; Lambert, A. FPGA Implementation of Shack–Hartmann Wavefront Sensing Using Stream-Based Center of Gravity Method for Centroid Estimation. *Electronics* **2023**, *12*, 1714. [CrossRef]
13. Mocci, J.; Busato, F.; Bombieri, N.; Bonora, S.; Muradore, R. Efficient implementation of the Shack–Hartmann centroid extraction for edge computing. *JOSA A* **2020**, *37*, 1548–1556. [CrossRef] [PubMed]
14. Mompeán, J.; Aragón, J.L.; Prieto, P.M.; Artal, P. GPU-based processing of Hartmann–Shack images for accurate and high-speed ocular wavefront sensing. *Future Gener. Comput. Syst.* **2019**, *91*, 177–190. [CrossRef]
15. Mocci, J.; Quintavalla, M.; Trestino, C.; Bonora, S.; Muradore, R. A multiplatform CPU-based architecture for cost-effective adaptive optics systems. *IEEE Trans. Ind. Inform.* **2018**, *14*, 4431–4439. [CrossRef]
16. Li, Z.; Li, X. Centroid computation for Shack-Hartmann wavefront sensor in extreme situations based on artificial neural networks. *Opt. Express* **2018**, *26*, 31675–31692. [CrossRef]
17. Hu, L.; Hu, S.; Gong, W.; Si, K. Learning-based Shack-Hartmann wavefront sensor for high-order aberration detection. *Opt. Express* **2019**, *27*, 33504–33517. [CrossRef]
18. Bovik, A.C. *Handbook of Image and Video Processing*; Academic Press: Cambridge, MA, USA, 2010.
19. Johnston, C.T.; Bailey, D.G. FPGA implementation of a single pass connected components algorithm. In Proceedings of the 4th IEEE International Symposium on Electronic Design, Test and Applications (delta 2008), Hong Kong, China, 23–25 January 2008; IEEE: Piscataway, NJ, USA, 2008; pp. 228–231.
20. Manohar, M.; Ramapriyan, H.K. Connected component labeling of binary images on a mesh connected massively parallel processor. *Comput. Vis. Graph. Image Process.* **1989**, *45*, 133–149. [CrossRef]
21. Rosenfeld, A.; Pfaltz, J.L. Sequential operations in digital picture processing. *J. ACM* **1966**, *13*, 471–494. [CrossRef]
22. Lacassagne, L.; Zavidovique, B. Light speed labeling: Efficient connected component labeling on RISC architectures. *J. Real-Time Image Process.* **2011**, *6*, 117–135. [CrossRef]
23. Jablonski, M.; Gorgon, M. Handel-C implementation of classical component labelling algorithm. In Proceedings of the Euromicro Symposium on Digital System Design, Rennes, France, 31 August–3 September 2004; IEEE: Piscataway, NJ, USA, 2004; pp. 387–393.
24. Alnuweiri, H.M.; Prasanna, V.K. Parallel architectures and algorithms for image component labeling. *IEEE Comput. Archit. Lett.* **1992**, *14*, 1014–1034.
25. Crookes, D.; Benkrid, K. FPGA implementation of image component labeling. In Proceedings of the Reconfigurable Technology: FPGAs for Computing and Applications, International Society for Optics and Photonics, Boston, MA, USA, 19–22 September 1999; Volume 3844, pp. 17–23.
26. Crookes, K.; Benkrid, A. An FPGA-Based Image Connected Component Labeller. In Proceedings of the 2003 International Conference on Field Programmable Logic and Applications, Lisbon, Portugal, 1–3 September 2003; Volume 2778, pp. 1012–1015.
27. Otsu, N. A Threshold Selection Method from Gray-Level Histograms. *IEEE Trans. Syst. Man, Cybern.* **1979**, *9*, 62–66. [CrossRef]
28. Woods, R.; González, R. Algoritmos de Procesamiento de Imagen Satelitales con Transformada Hough. *Rev. Vis. Electron.* **2009**, *5*, 26–41.
29. Birchfield, S.T. Pixel-Based Image Processing Chapter 2. 2011. Available online: https://cecas.clemson.edu/~stb/ece847/internal/cvbook/ch02_pixproc.pdf (accessed on 15 January 2024).
30. Vincent, L.; Soille, P. Watersheds in digital spaces: An efficient algorithm based on immersion simulations. *IEEE Trans. Pattern Anal. Mach. Intell.* **1991**, *13*, 583–598. [CrossRef]
31. Comaniciu, D.; Meer, P. Mean shift: A robust approach toward feature space analysis. *IEEE Trans. Pattern Anal. Mach. Intell.* **2002**, *24*, 603–619. [CrossRef]
32. Achanta, R.; Shaji, A.; Smith, K.; Lucchi, A.; Fua, P.; Süsstrunk, S. SLIC superpixels compared to state-of-the-art superpixel methods. *IEEE Trans. Pattern Anal. Mach. Intell.* **2012**, *34*, 2274–2282. [CrossRef]

33. Rother, C.; Kolmogorov, V.; Blake, A. "GrabCut" interactive foreground extraction using iterated graph cuts. *ACM Trans. Graph.* **2004**, *23*, 309–314. [CrossRef]
34. AbuBaker, A.; Qahwaji, R.; Ipson, S.; Saleh, M. One scan connected component labeling technique. In Proceedings of the 2007 IEEE International Conference on Signal Processing and Communications, Dubai, United Arab Emirates, 24–27 November 2007; IEEE: Piscataway, NJ, USA, 2007; pp. 1283–1286.
35. Schwenk, K.; Huber, F. Connected Component Labeling algorithm for very complex and high-resolution images on an FPGA platform. In Proceedings of the High-Performance Computing in Remote Sensing V and International Society for Optics and Photonics, Toulouse, France, 21–24 September 2015; Volume 9646, p. 964603.
36. Wu, K.; Otoo, E.; Suzuki, K. Optimizing two-pass connected-component labeling algorithms. *Pattern Anal. Appl.* **2009**, *12*, 117–135. [CrossRef]
37. Döge, K.P. *Videodetektion im Straßenverkehr: Signalmodelle und Analyseverfahren*; Walter de Gruyter: Berlin, Germany, 2013.
38. Mauch, S.; Reger, J. Real-time spot detection and ordering for a Shack–Hartmann wavefront sensor with a low-cost FPGA. *IEEE Trans. Instrum. Meas.* **2014**, *63*, 2379–2386. [CrossRef]
39. Mauch, S.; Reger, J. Real-time implementation of the spiral algorithm for Shack-Hartmann wavefront sensor pattern sorting on an FPGA. *Measurement* **2016**, *92*, 63–69. [CrossRef]
40. Tyson, R.K.; Frazier, B.W. *Principles of Adaptive Optics*; CRC Press: Boca Raton, FL, USA, 2022.
41. Du, X.; Zhang, H.; Feng, J.; Xie, Q. A method of converting cameralink into hdmi based on fpga. In Proceedings of the 6th International Conference on Optical, Photonic Engineering (icOPEN 2018) and International Society for Optics and Photonics, Shanghai, China, 8–11 May 2018; Volume 10827, p. 1082716.
42. Manufacturer, H. HDMI (High-Definition Multimedia Interface). Available online: https://www.immagic.com/eLibrary/ARCHIVES/GENERAL/WIKIPEDI/W120621H.pdf (accessed on 20 September 2023).
43. Thier, M.; Paris, R.; Thurner, T.; Schitter, G. Low-latency Shack–Hartmann wavefront sensor based on an industrial smart camera. *IEEE Trans. Instrum. Meas.* **2012**, *62*, 1241–1249. [CrossRef]