Journal of
Electronic Imaging

JElectronicImaging.org

Full image-processing pipeline in field-programmable gate array for a small endoscopic camera

Sheikh Shanawaz Mostafa
L. Natércia Sousa
Nuno Fábio Ferreira
Ricardo M. Sousa
Joao Santos
Martin Wäny
F. Morgado-Dias

Sheikh Shanawaz Mostafa, L. Natércia Sousa, Nuno Fábio Ferreira, Ricardo M. Sousa, Joao Santos, Martin Wäny, F. Morgado-Dias, "Full image-processing pipeline in field-programmable gate array for a small endoscopic camera," *J. Electron. Imaging* **26**(1), 013005 (2017), doi: 10.1117/1.JEI.26.1.013005.

Full image-processing pipeline in field-programmable gate array for a small endoscopic camera

Sheikh Shanawaz Mostafa,[a,b,*] L. Natércia Sousa,[b] Nuno Fábio Ferreira,[b,c] Ricardo M. Sousa,[d] Joao Santos,[d] Martin Wäny,[d] and F. Morgado-Dias[b,c]
[a]University of Lisbon-Instituto Superior Técnico, Av. Rovisco Pais 1, Lisboa 1049-001, Portugal
[b]Madeira Interactive Technologies Institute, Polo Científico e Tecnológico da Madeira, floor-2, Caminho da Penteada, Funchal 9020-105, Portugal
[c]Universidade da Madeira, Praca do Município, Colégio dos Jesuítas-Rua dos Ferreiros, Funchal 9000-082, Portugal
[d]AWAIBA, CMOSIS PORTUGAL, Madeira Tecnopolo, Funchal 9020-105, Portugal

Abstract. Endoscopy is an imaging procedure used for diagnosis as well as for some surgical purposes. The camera used for the endoscopy should be small and able to produce a good quality image or video, to reduce discomfort of the patients, and to increase the efficiency of the medical team. To achieve these fundamental goals, a small endoscopy camera with a footprint of $1\ \text{mm} \times 1\ \text{mm} \times 1.65\ \text{mm}$ is used. Due to the physical properties of the sensors and human vision system limitations, different image-processing algorithms, such as noise reduction, demosaicking, and gamma correction, among others, are needed to faithfully reproduce the image or video. A full image-processing pipeline is implemented using a field-programmable gate array (FPGA) to accomplish a high frame rate of 60 fps with minimum processing delay. Along with this, a viewer has also been developed to display and control the image-processing pipeline. The control and data transfer are done by a USB 3.0 end point in the computer. The full developed system achieves real-time processing of the image and fits in a Xilinx Spartan-6LX150 FPGA. *© 2017 SPIE and IS&T* [DOI: 10.1117/1.JEI.26.1.013005]

Keywords: endoscopic camera; image processing; field-programmable gate array; high frame rate.

Paper 16510P received Jun. 23, 2016; accepted for publication Dec. 13, 2016; published online Jan. 11, 2017.

1 Introduction

In the United States of America alone, 18,328,000 endoscopies were performed in 2009, taking into account the combined numbers of upper, lower, and biliary endoscopies. The estimated cost of these gastrointestinal endoscopies was of 32.4 billion US dollars.[1] Endoscopy is not only used to look at the internal organs of the body, which is otherwise impossible without a surgical procedure, but also used in some surgical procedures. Nowadays, it is also used for screening for colon and rectal cancers. Since the procedure involves placing a viewing device inside the living body, the size of the camera and the quality of images are the main factors to take into consideration when designing such a system. Considering the implementation alternatives, CMOS technology consumes less power and is simpler to control than CCD technology[2] and this is the reason the endoscopy system developed in this paper uses a CMOS image sensor that has small dimensions of $1.0\ \text{mm} \times 1.0\ \text{mm} \times 1.7\ \text{mm}$ (width × length × height). This CMOS sensor generates 10-bit pixels with a Bayer filter mosaic and it has a high frame rate. The designed system needs to demosaic the pixels to obtain proper RGB colors. In addition to that, because of the manufacturing process, there are differences between pixel size and position as well as other electronic and optical issues related to the photo transistors that cause a fixed pattern noise (FPN).[3] Therefore, noise reduction and image correction are needed to obtain the true RGB image. Because of the high frame rate of the chosen CMOS sensor, it is impossible to process all the frames in the computer. To show the video, the real-time frame is dropped which results in the loss of information and resources.

One of the obvious solutions to this problem is to use a field-programmable gate array (FPGA) because of the parallel architecture that can be created.[4] This kind of parallel and custom architecture make it a suitable candidate for solving this kind of problem. Although the main algorithms of the system are pipelined, parallelism is needed for simultaneously processing different color information as well as some internal operations of the algorithms so the system can process the information in real time. In graphics processing units (GPUs), parallelism can be achieved in the algorithms to run faster than a conventional processor. Nevertheless, GPUs have a high-energy consumption level, need to be connected to a personal computer (PC), and the solution's speed would depend on the hardware and operating system. On the other hand, some research done in this field is image compression in application-specific integrated circuits (ASICs),[5,6] which have all the facilities of FPGAs, but are costly to design (layout, masks, or other manufacturing steps). That is why FPGAs are chosen for this work. In FPGA,[7–9] gamma correction,[10] demosaicking,[11] image enhancements, and classification[12] are done. These FPGA-based systems are designed for solving one problem. They do not have set of solutions for most of the common problems of sensor image processing. Most of the systems use external memory as presented in Ref. 13. An external memory type of implementation consumes more power and resources than internal memory. The lack of a total image-processing pipeline in FPGA for an endoscopic system motivated the design of a complete image-processing pipeline without using external memory.

*Address all correspondence to: Sheikh Shanawaz Mostafa, E-mail: sheikh.mostafa@tecnico.ulisboa.pt

1017-9909/2017/$25.00 © 2017 SPIE and IS&T

For designing a full image-processing pipeline in an FPGA for a small endoscopic camera to solve the previously mentioned problems, the following criteria are applied in this work:

- A good but minimum image pipeline to get good images at high frame rates.
- A system totally customized for FPGA implementation using VHDL.
- No reliance on any external resources such as external RAM or processors.
- Fully controllable algorithms using a custom made graphics user interface (GUI) from a computer viewer over USB 3.0 for easier control.
- Hardware implementation was done and compared with two software implementation in PC.

2 System Setup

A full endoscopy system has been developed (Fig. 1) with a sensor, a standalone image-processing system on an FPGA, and a viewer to show the picture or video. The system senses the light through the sensor (Fig. 2(a)). The sensor sends the data to the FPGA board using a 2-m-flat ribbon cable, with four copper wires. After processing the raw data using image-processing algorithms on the FPGA, the system uses the USB 3.0 interface to send the picture to the computer. Finally, the received data are seen on the monitor by the viewer. A brief description of these components is given in the following sections.

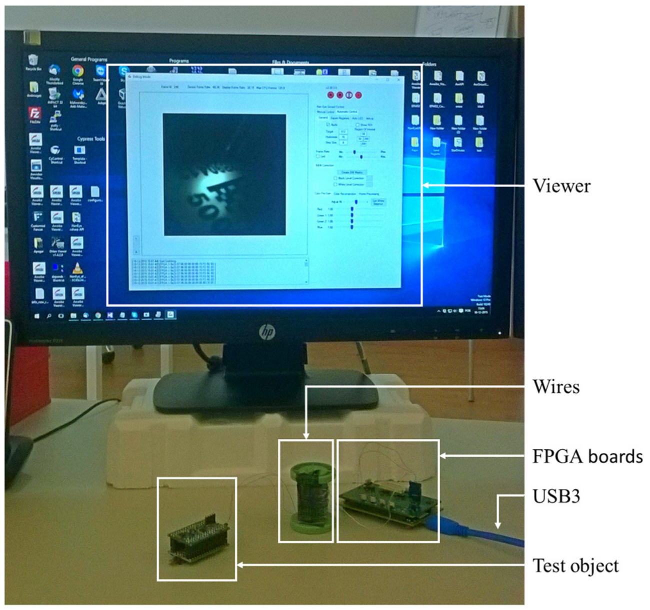


Fig. 1 Proposed endoscopy system setup.

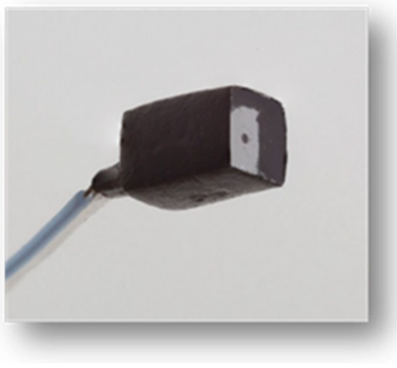
2.1 *Sensor*

In general, a digital image sensor uses a architecture suitable for scaling up to several mega pixel resolutions as well as additional electronic parts such as analog to digital converter, low-power charge pumps, exposure control, color reconstruction, and mobile processor interfaces. However, for the "2-D NanEye Module" [Fig. 2(a)] CMOS sensor, the nonmandatory features were removed to achieve a small footprint. The CMOS sensor was optimized to avoid decoupling capacitors and to transmit the raw pixels data along a cable up to 3 m in length. True silicon via is used to provide electrical contact using tiny via holes through the silicon to connect to the transistors on the opposite side with solder balls. This solution allows the final package size to be kept to proportions that are similar to the size of the image sensor.

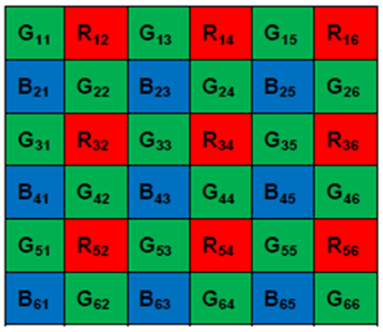
The camera lens is etched out of quartz glass and is covered by the Bayer filter [Fig. 2(b)] to achieve a color picture. In Fig. 2(b), R, G, and B indicate the red, green, and blue colors of the filter and the subscripts of R, G, B shows the row and the column numbers of the pixel. The physical dimensions are $1.0\text{ mm} \times 1.0\text{ mm} \times 1.65\text{ mm}$ with 250×250 pixels (62.5 k). It has an electronic rolling shutter that can capture 44 to 55 frames per second (fps).[14,15] The sensor vendor Awaiba is a supplier and developer of image sensors for this work. The sensor specification mentions a maximum of 55 fps to minimize the risk of damaging the sensor in the long run, however, it is possible to reach 60 fps by changing the supply voltage of the sensor operating at room temperature.

2.2 *Field-Programmable Gate Array and Extension Boards*

A CESYS EFM-02 board which has an embedded FPGA module based on a Xilinx Spartan-6LX150 FPGA is used.



(a)


G_{11} R_{12} G_{13} R_{14} G_{15} R_{16}
B_{21} G_{22} B_{23} G_{24} B_{25} G_{26}
G_{31} R_{32} G_{33} R_{34} G_{35} R_{36}
B_{41} G_{42} B_{43} G_{44} B_{45} G_{46}
G_{51} R_{52} G_{53} R_{54} G_{55} R_{56}
B_{61} G_{62} B_{63} G_{64} B_{65} G_{66}

(b)

Fig. 2 (a) CMOS "NanEye" sensor and (b) Bayer pattern for the sensor.

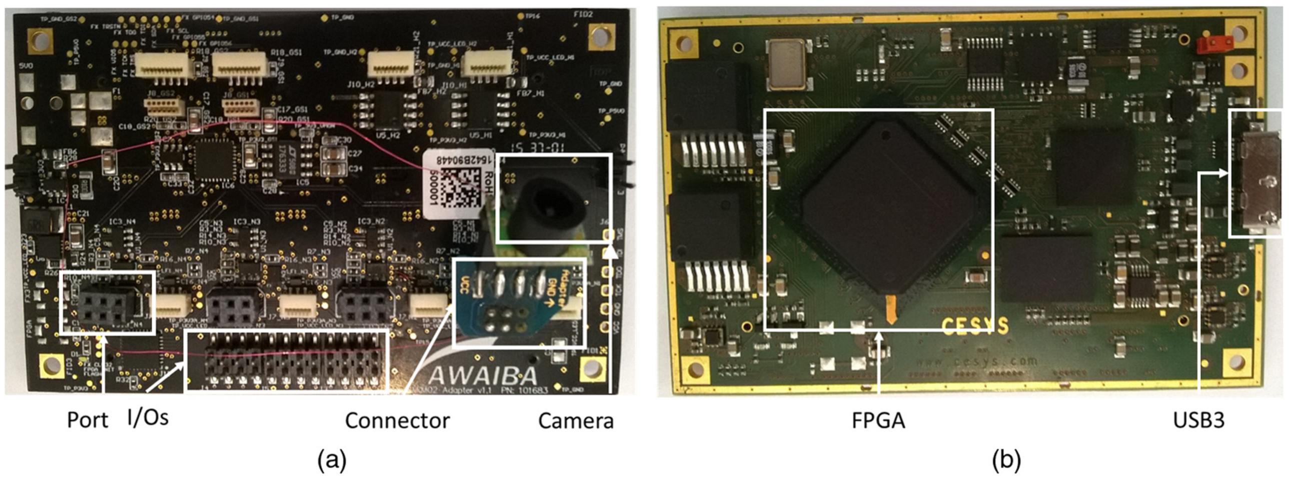


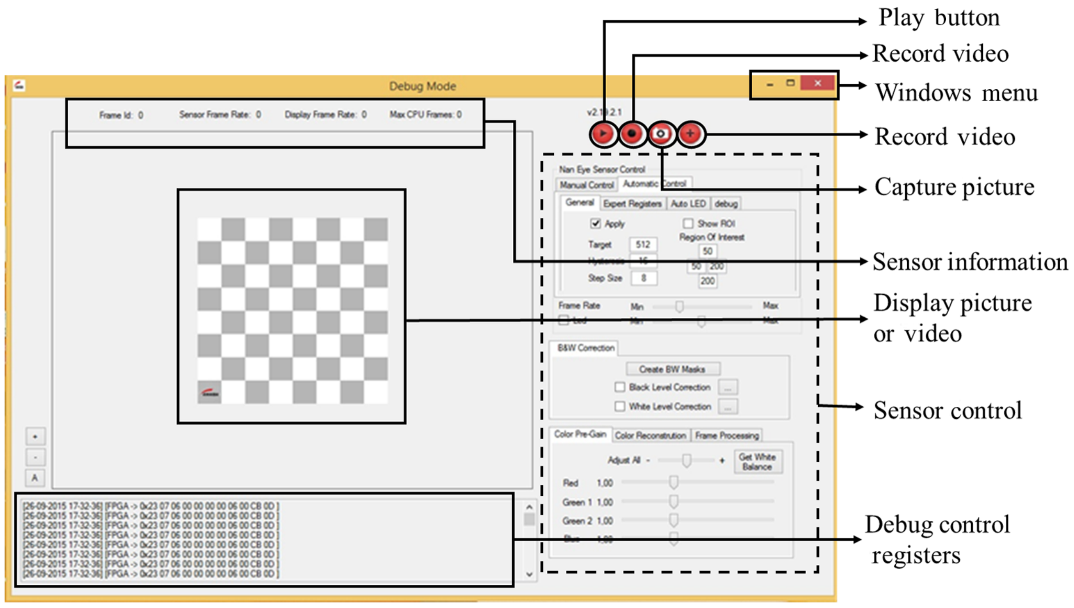
Fig. 3 (a) Top view extension board for connecting the sensor. (b) Bottom view FPGA Board.

The size of the PCB is 52.6 mm × 82.6 mm. The board has 2-Gbit DDR2 memory and a 64-Mbit dual SPI flash. The board is USB 3.0 bus-powered with a versatile Cypress EZ-FX3 controller for super speed interface [Fig. 3(b)]. Awaiba's "AWAIBA NANEYE USB 3.0 v1.0" adapter board is used for sensor interfacing. It is possible to have four sensors with light-emitting diodes connected to the board simultaneously. There is a JTAG and there are several optional I/Os pins to debug the system. It has almost the same dimensions as the CESYS EFM-02 board Fig. 3(a).[16]

2.3 *Awaiba Viewer*

The Awaiba Viewer (version number v2.19.1.2) is a software tool to prepare the image for display in a PC and it was developed in C# by Awaiba. The main purpose of the Awaiba Viewer is to show the picture to users and it is connected to the FPGA board through the Cypress EZ-FX3 controller. The wishbone bus[17] is applied on top of this to control the image-processing algorithms in the FPGA side. The visual interface is shown in Fig. 4. It has several options to take user defined values to control the image-processing pipeline and other user input components for producing and showing the picture.

However, this Awaiba Viewer was only able to capture the raw data from the sensor and all the image processing was done in the PC using in-house algorithms and open source computer vision (OpenCV)[18] algorithms. OpenCV is an open source computer vision library mostly written in C/C++. It can be used even for hardware acceleration. Its Python and Java interfaces support Windows, Linux, Mac OS, iOS, and Android. So in order to allow changing the image processing from the PC to the FPGA, the required adjustments were made on the Awaiba Viewer. A similar design template is kept in addition so that it can control the parameters for the FPGA. The new version of the viewer was developed for this research and is not commercially available by the sensor vendor. A brief description of new viewer's control, input, and display options is provided below. Later, in this



```
[26-09-2015 17-32-36] [FPGA -> 0x23 07 06 00 00 00 00 06 00 CB 0D ]
[26-09-2015 17-32-36] [FPGA -> 0x23 07 06 00 00 00 00 06 00 CB 0D ]
[26-09-2015 17-32-36] [FPGA -> 0x23 07 06 00 00 00 00 06 00 CB 0D ]
[26-09-2015 17-32-36] [FPGA -> 0x23 07 06 00 00 00 00 06 00 CB 0D ]
[26-09-2015 17-32-36] [FPGA -> 0x23 07 06 00 00 00 00 06 00 CB 0D ]
[26-09-2015 17-32-36] [FPGA -> 0x23 07 06 00 00 00 00 06 00 CB 0D ]
[26-09-2015 17-32-36] [FPGA -> 0x23 07 06 00 00 00 00 06 00 CB 0D ]
[26-09-2015 17-32-36] [FPGA -> 0x23 07 06 00 00 00 00 06 00 CB 0D ]
```

Fig. 4 Graphical user interface and the viewer for displaying the video/images capture from sensor.

paper, Awaiba's in-house algorithms will be mentioned as Awaiba's algorithms.

- Play button: This button is used to show or start the video feed from the FPGA.
- Record video: This button is for saving the video captured by the viewer.
- Capture picture: As the name suggests, this button is for saving the picture or taking a snapshot of the video.
- Sensor information: This is a display for showing the information about the sensor such as sensor frame rate, frame id sent by the FPGA, the display frame rate, and the maximum frame rate.
- Display picture or video: This window shows the picture or video captured or processed by the viewer.
- Sensor control: The sensor control panel is a collection of all the buttons required to control the sensor as well as the image-processing algorithms. All the values defined by the user are transmitted to the FPGA using the USB 3.0 end point. All the algorithms have a default enable/disable button so that it is possible to enable/disable any algorithm using the viewer. Other parameters unique to each algorithm can be controlled by the viewer and are shown in Table 1.
- Process type button: It allows the user to choose which image-processing pipeline should be applied. There are three options: the one from Awaiba (implemented by software), the one from OpenCV (implemented by software), and the one implemented on the FPGA.
- Debug control registers: This window is for debugging the sensor control and uses the USB 3.0 end point to write the registers on the FPGA. From this window, the user can check whether the register is properly written or not.
- Windows menu: This is a classic window menu. It has close, minimize, and restore buttons.

Table 1 Unique parameters controlled by the viewer.

Name of the algorithms	Unique parameters controlled by the viewer
Frame mean	Initial threshold, threshold offset
Black mask	Create black mask
White mask	Create white mask
Bad pixels removal	Threshold
Precolor gain adjustment	Gain for R, B, and two G
Color reconstruction	No unique parameters
Color adjustment matrix	$\begin{bmatrix} R_R & R_G & R_B \\ G_R & G_G & G_B \\ B_R & B_G & B_B \end{bmatrix}$
Gamma correction	Gamma value
Adjust brightness	Gain for R, G, B

3 Image-Processing Pipeline

The image-processing pipeline includes the same kind of processing and data flow for all three options (OpenCV, Awaiba, and FPGA) allowed in this system (Fig. 5). The differences are in the implementation (by software or by hardware) and in the algorithms used to do the processing. The OpenCV image-processing pipeline implements algorithms from an OpenCV library, Awaiba algorithms are written by Awaiba in C#, and the FPGA algorithms were implemented in VHDL using fixed point calculation. The algorithms used in the image-processing pipeline are explained as follows.

3.1 *Frames Mean*

The frames mean block has the main purpose of reducing the image's noise. Small changes in pixel values from the CMOS image sensor are experienced between frames. If the frames were sent to the output exactly as received, the viewer would see a flickering effect on the image, which is not desirable. Thus, the mean operation is used and allows the flickering effect to fade away. On the other hand, if the image context is changed, leading to a higher variation in pixel values for the following frame, the system should pass to the output, allowing the viewer to be aware of such context changes. For this reason, a threshold value is used to detect context changes.

In order to save the FPGA's memory resources instead of calculating the mean between the pixels of the last received frames,[19] an infinite impulse response-based method is used by applying Eq. (1), in which only one frame and one line (the current line being received) need to be stored in memory. A weight of 1/16 was used for the incoming frame to ease the FPGA implementation process. After a reset, the first received frame is stored as the current mean since it reduces the time needed for reaching the mean value. After that, for each new pixel received, the new mean [Eq. (1)] is computed by using the weight, the current mean (CurrentMean), and the current pixel value (CurrentPixVal), and giving higher weight to the current stored mean

$$\text{NewMean} = \frac{\text{CurrentPixVal} + \text{CurrentMean} \cdot 15}{16}. \tag{1}$$

On the FPGA, Eq. (1) is implemented using a dual-port RAM, an adder, an IP block that performs subtraction, and a logical shifter (Fig. 6). When a pixel arrives, its corresponding current mean address for the RAM is generated by the address generator. According to this address, the RAM delivers the value of the current mean and logically shifts left. After shifting the CurrentMean four binary positions to the left (product by 16) and subtracting the result of this operation by CurrentMean (resulting in a product by 15), the value is added to the current pixel value and then shifted right (which results in a division by 16).

Moreover, this averaging of frames can create a dragging effect in the video. If a fast moving object appears in the scene, a tail of the moving object can be seen. To reduce this effect, a threshold is used. However, a static threshold is not enough for pixel ranges from 0 to 1023. It should be adaptive and proportional to the current pixel value being received. In the literature, most of the adaptive threshold is used for changes in videos,[20] object detection, and

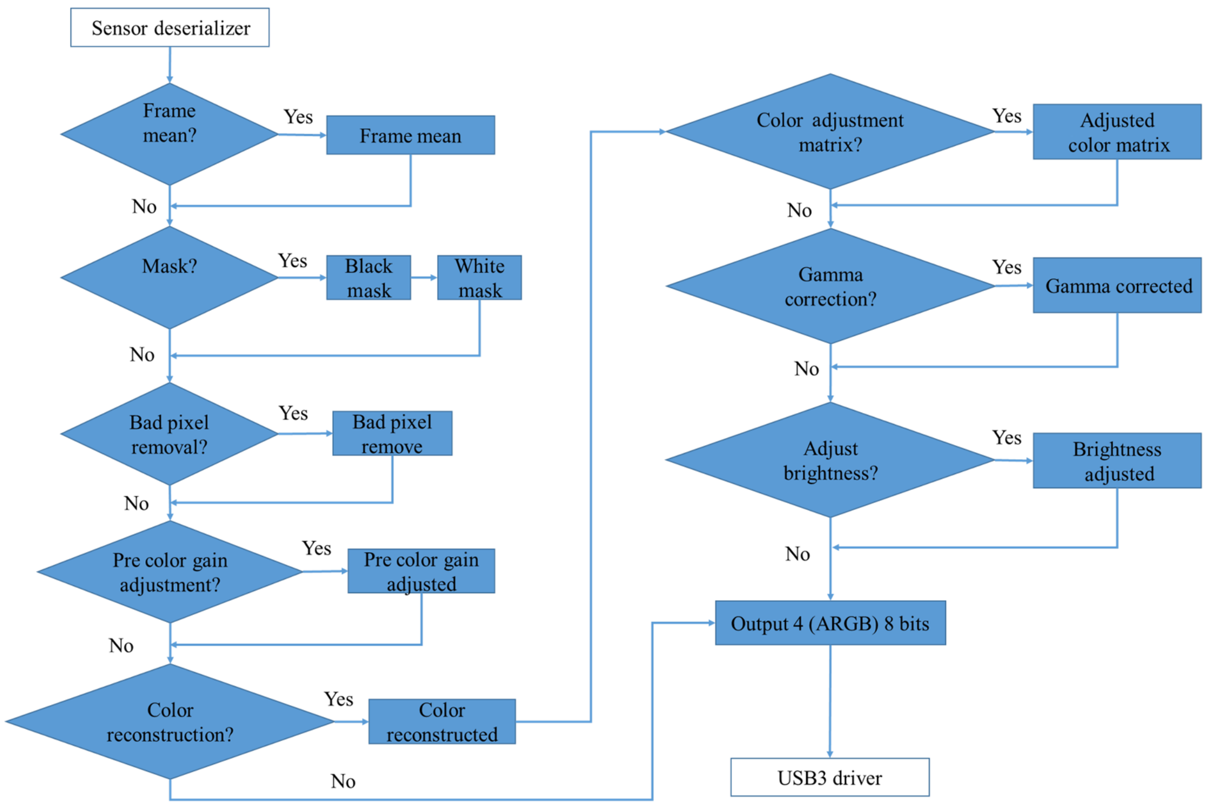


Fig. 5 Flowchart of image-processing pipeline for endoscopy.

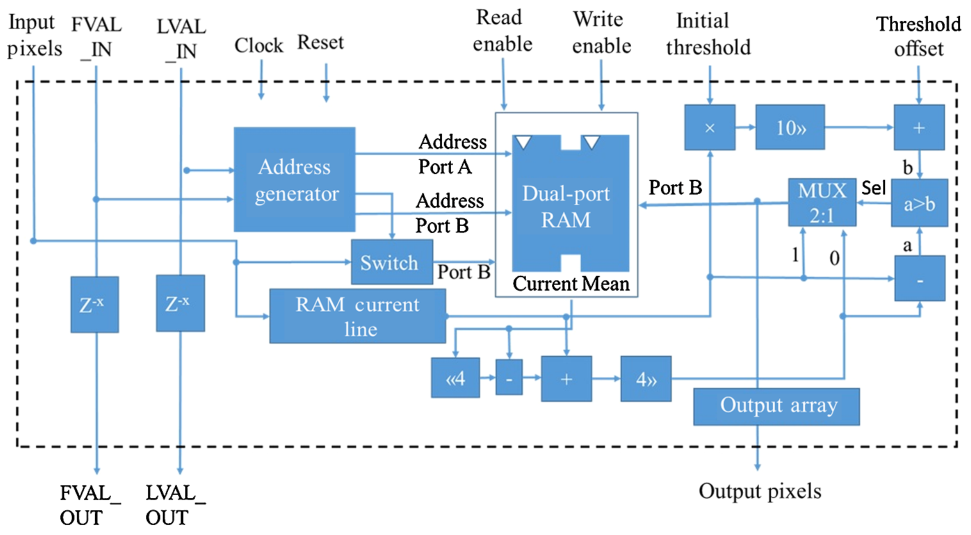


Fig. 6 Block diagram of frames mean algorithm.

segmentation.[21] Due to the simplicity of our problem compared to object detection a simpler method is chosen by experiment. It was verified that an adaptive threshold ($\mathrm{Th}_{\mathrm{ad}}$) calculated based on the linear equation leads to a significant decrease of the referred dragging effect in the video. In [Eq. (2)], an initial threshold ($\mathrm{Th}_{\mathrm{ini}}$) value is used (as input to the block), which should be a power of 2 in order to make the implementation easier, and has a value of 32 in the current implemented version. A threshold offset ($\mathrm{Th}_{\mathrm{off}}$) which is chosen by user is also summed in order to have a high enough threshold when very low pixel values are being received

$$\mathrm{Th}_{\mathrm{ad}} = \mathrm{Th}_{\mathrm{ini}} \times \left(\frac{\mathrm{CurrentPixVal}}{1024}\right) + \mathrm{Th}_{\mathrm{off}}. \tag{2}$$

The adaptive threshold is created on the FPGA by multiplying the initial threshold with the current pixel value, and then dividing it by 1024 (shifting it by 10 bits). Then the result is added to the threshold offset (Fig. 6). The choice of the

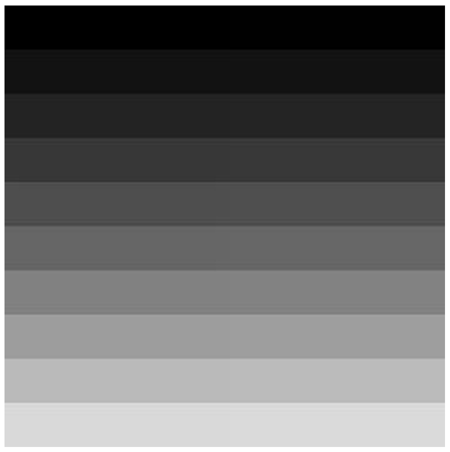


Fig. 7 Basic template of test pattern for testing frame mean algorithm.

output pixel between a new pixel and a new mean is done by using a multiplexer. The multiplexer decision is made by comparing the threshold with the difference of the current mean and current pixel.

A fixed pattern picture template is used for testing the frame mean algorithm (Fig. 7). The difference between frames is created by adding random numbers to the template. However, it is ensured that the random number is less than the adaptive threshold ($\mathrm{Th}_{\mathrm{ad}}$) [Eq. (2)] so that the effectiveness of the frame mean can be evaluated. The algorithm is run on the FPGA with the pattern and video recorded and evaluated. From the analysis, it can be seen (Fig. 8) that the frames mean algorithm has been successfully applied and is able to reduce the noise or instability from one frame to the next. If there is no difference between one frame and the next (after frame 55 in Fig. 8), the algorithm has no effect.

3.2 *Black and White Masks*

The mask blocks have the main purpose of removing FPN. There are two different kinds of masks applied here; one is a black mask and another is a white mask. The black mask has the pixel value in black and is created in black conditions (absence of light). It allows for the reduction of the dark signal nonuniformity. The white masks are used to reduce the photo response nonuniformity (PRNU) by applying an individual gain to each pixel to compensate for nonuniformity in the individual photo transistors. The implementation of these masks is described as follows.

3.2.1 *Black mask*

For each pixel received ($\mathrm{Pix}_{\mathrm{in}}$), the respective black mask value (B_{Pix}) is subtracted from the current pixel value [see Eq. (3)]. Attention is paid to underflow. (In case the subtraction yields a negative value, 0 is used.) Because of the randomness of the nature of the noise, all pixel values less than or equal to three are clamped to a zero digital number (DN) to reduce noise. This value is purely experimental and it may differ from sensor to sensor

$$\mathrm{Pix}^b = \begin{cases} \mathrm{Pix}_{\mathrm{in}} - B_{\mathrm{Pix}}, & \text{if } 3 < \mathrm{Pix}^b \leq 1024 \\ 0, & \text{other case} \end{cases}. \tag{3}$$

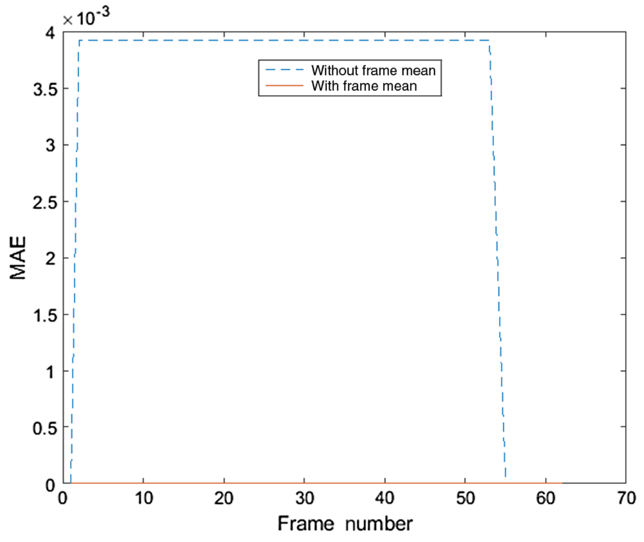


Fig. 8 MAE between frames generated using a slightly different test pattern with and without a Frames mean algorithm.

There are three main components in the black mask algorithms which are subtraction, address generator, and a dual-port RAM (Fig. 9) The address generator generates the proper address for the corresponding arriving pixels. According to the generated address, the mask is subtracted from the pixel to produce the output pixels. The address generator is also used to generate the black mask. It is done by setting the write enable signal high and at the same time covering the sensor so that it is in complete darkness. To reduce the temporal noise, frames mean is quite important, otherwise, the temporal noise is converted to the FPN which is even more visible. Due to the placement of the black mask on the image-processing pipeline (after the frames mean), no extra component is needed. The user only has to enable the frames mean algorithm before creating the black mask.

After applying the black mask, the histogram is shifted to the left and aligned with zero which indicates the successful implementation of the algorithm (Fig. 10).

3.2.2 *White mask*

PRNU is reduced by implementing the white mask, which is done by multiplying the gain of the respective pixel (W_{Pix}) with its own value ($\mathrm{Pix}_{\mathrm{in}}$) [see Eq. (4)]. In this component, attention is paid to overflow. (If the multiplication result exceeds 1024, the upper limit is used.) The masks' algorithm uses IP cores (a multiplier and a dual-port RAM)

$$\mathrm{Pix}^w = \begin{cases} \mathrm{Pix}_{\mathrm{in}} \times W_{\mathrm{Pix}}, & \text{if } 0 \leq \mathrm{Pix}^w \leq 1024 \\ 1024, & \text{if } \mathrm{Pix}^w \geq 1024 \\ 0, & \text{other cases} \end{cases}. \tag{4}$$

From Fig. 11, it is clear that the implementation of the white mask on the FPGA is almost the same as the black mask. The difference is that instead of having a subtraction at the end, it has a divider to implement Eq. (4). For the same reason as the black mask, when creating the white mask, it is advisable to enable frame mean and black mask algorithms. The implemented white mask subsystem needs a homogenous light to create the white mask. Figure 12 shows that most of the

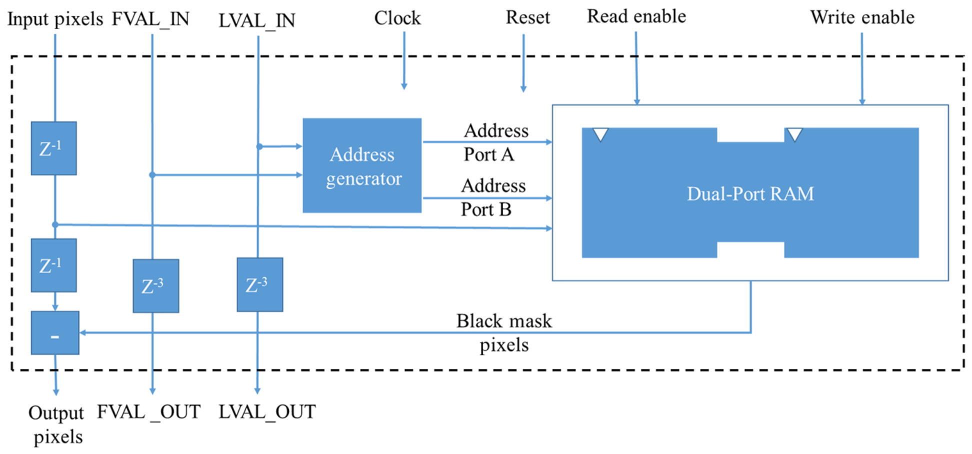


Fig. 9 Block diagram of the black mask algorithm.

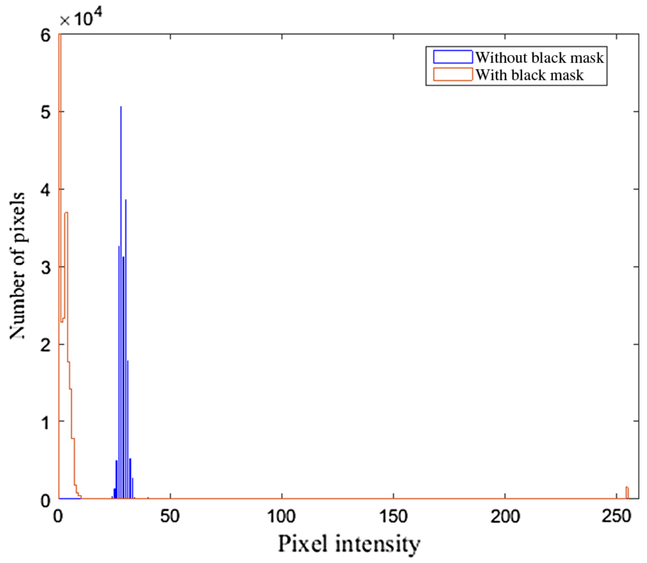


Fig. 10 Histogram with and without applying the black mask in dark conditions.

pixels are shifted to the 255 value after applying the white mask which is the maximum of the 24 bits RGB system. In this figure, the blue color represents the number of pixels before applying the white mask and the red color represents the number of pixels after applying the white mask.

3.3 *Bad Pixel Removal*

The bad pixel removal block has the main purpose of removing defective pixels that might occur due to physical faults. For instance, using this block, the pixel value can be changed. If its value is too different from its neighbor's value, it is assumed to be a defective pixel. Neighbors are not necessarily the adjacent pixels, but those immediately before and after the pixels, both horizontally and vertically, in an RGB Bayer pattern [as seen in Fig. 2(b)]. In order to decide whether a pixel value is defective, a threshold level is used.

Regarding the algorithm used for each pixel received, the computation is made using the subset composed of the available neighboring pixels along with the current pixel being analyzed. This subset can be comprised of 3 (for pixels in corners), 4 (for pixels close to the frame limits), or 5 pixels

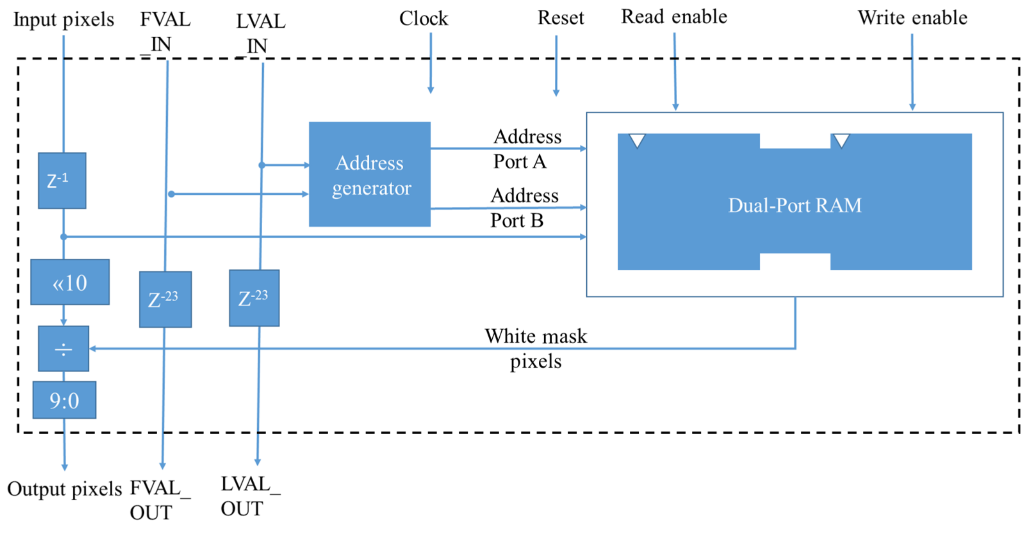


Fig. 11 Block diagram of the white mask algorithm.

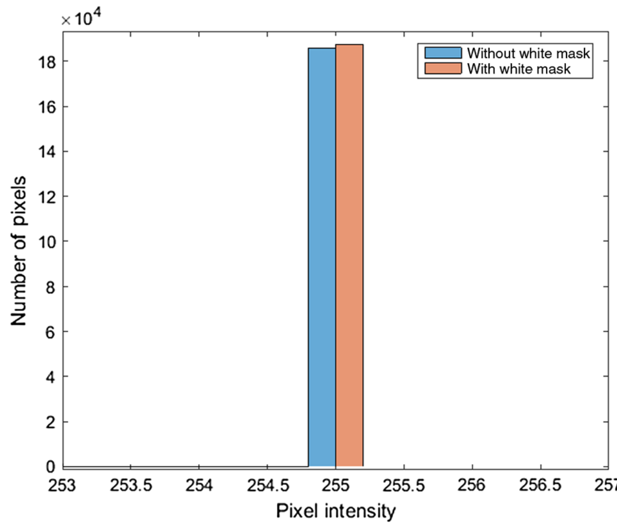


Fig. 12 Histogram with and without applying the white mask in a homogenous lighting condition.

(for pixels in the middle). For this reason, the results start being sent to the output after three lines have been received. However, in the FPGA implementation, a memory containing five complete lines is used for the cases when central pixels are being computed. The median value is obtained for each subset by doing sorting, as well as an arithmetic mean in the case of a subset with 4 pixels. The absolute value of the subtraction between the current pixel value and the median value is obtained and compared with the threshold limit. In case it is higher than the threshold, the current pixel gets the median value computed instead of holding its current value, meaning the pixel was detected as defective.

Regarding the sorting method used, and based on the research of May and Urbanek,[22] several comparators and adders are used to determine the position of each element in a subset of pixels, after which sorting can take place. In terms of resources, this algorithm uses one IP core (a dual-port RAM module with 1250 positions of 10 bits for storing five lines), several comparators, a block that provides as output the number of digital ones ("1" bit) contained in a

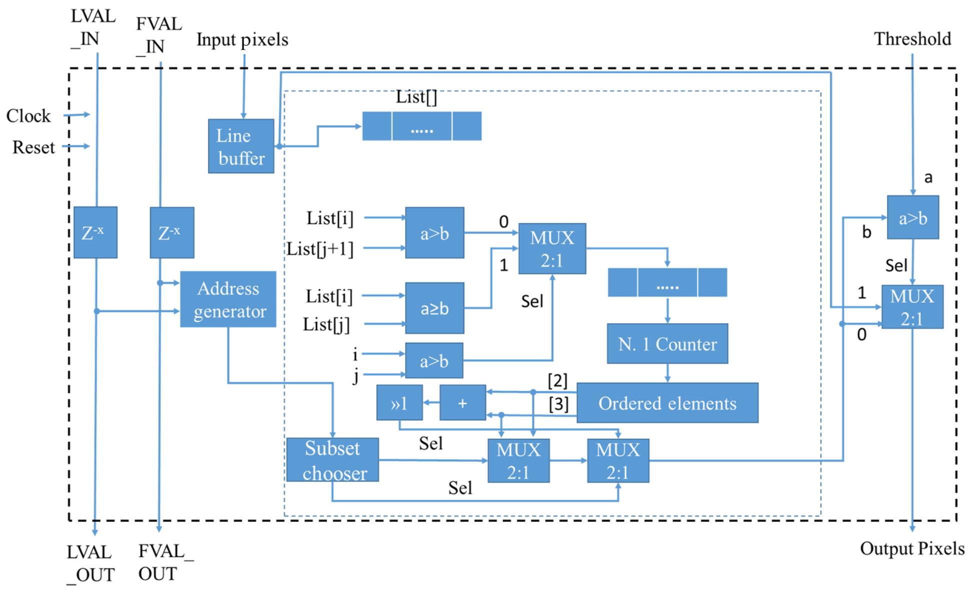


Fig. 13 Block diagram of bad pixel removal algorithm.

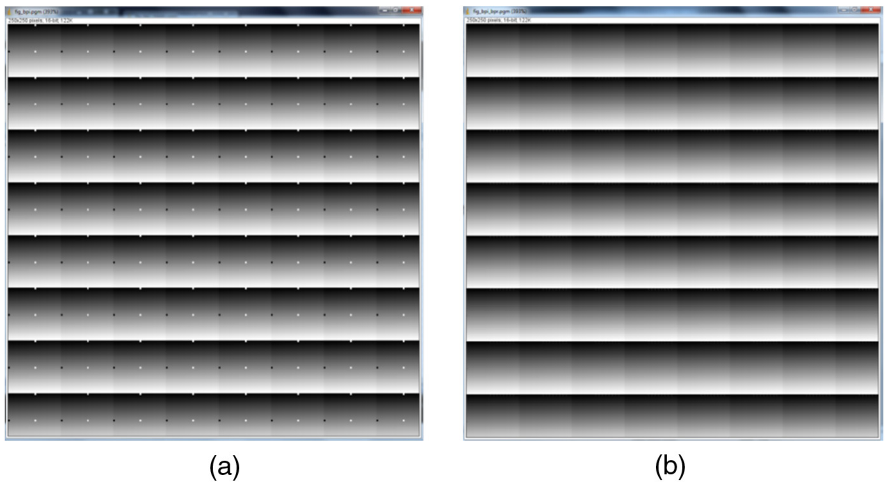


Fig. 14 Example of bad pixel removal algorithm. (a) Applied pattern. (b) Received picture.

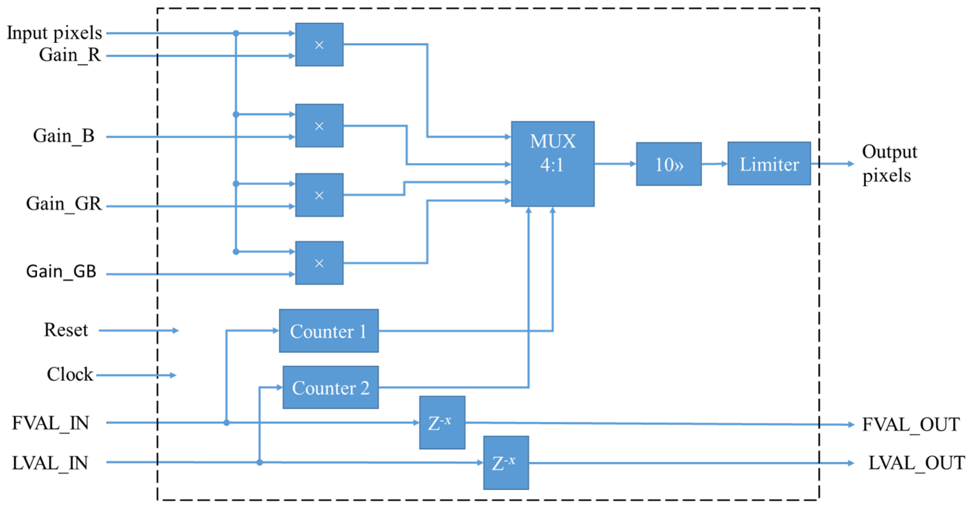


Fig. 15 Block diagram of the precolor gain adjustment algorithm.

standard logic vector input. Another block that outputs the number of pixels available for performing sorting and the median for the current pixel being analyzed. Third block that receives the current pixel availability, and returns an array of 5 pixels (filled with 0 or 1023 at the extremes when needed). Additional block that receives all the pixels to be sorted, and the corresponding calculated new position, returning the pixels already sorted. Last block that receives the sorted pixels and computes the median. The overall block diagram implemented is shown in Fig. 13.

The results of using the algorithm can be seen in Fig. 14. A fixed pattern is applied for the test. The algorithm is able to remove the pattern without destroying the other content in the picture.

3.4 *Precolor Gain Adjustment*

Color correction is needed to show the true color. Though the precolor gain adjustment algorithm is not enough, it can simplify the overall color correction process by correcting each channel separately to get the correct combination between all channels.[23]

This adjustment is done by applying a different gain to each color of the Bayer pattern [Fig. 2(b)]. Since the filters can have four different colors, the user may specify four coefficients. Its block diagram is shown in Fig. 15.

The input data start being multiplied by the coefficients using multiplier IP-blocks, and the output value is chosen on a multiplexer based on the column and the row to which the current pixel belongs. The coefficients can take values between 0.5 and 2.0. Since they are real numbers, and real numbers are not permitted on the FPGA, the values at the input are those coefficients multiplied by 1024, i.e., they are shifted left by 10 bits, so they are represented by 12 bits within the block. After the multiplication, the values are then shifted right by 10 bits. The output values are truncated to the maximum value of 1024.

3.5 *Color Reconstruction*

The Bayer pattern has missing color samples in the color space. By using bilinear interpolation, the technique color reconstruction module reconstructs the full color image.

The color reconstruction module calculates the missing color values by computing the average of the neighboring pixels.[24] There are four general cases indicated by color filters. Because of the Bayer pattern, there are two different color filters for the green shown in Fig. 16.

For instance, consider the pixel R_{POS} at position POS on the red filter in Fig. 16(a). In this position, the red filter component of the pixel is already available, but it is necessary to estimate the green and blue components. The missing components can be estimated by applying the following equation:[24,25]

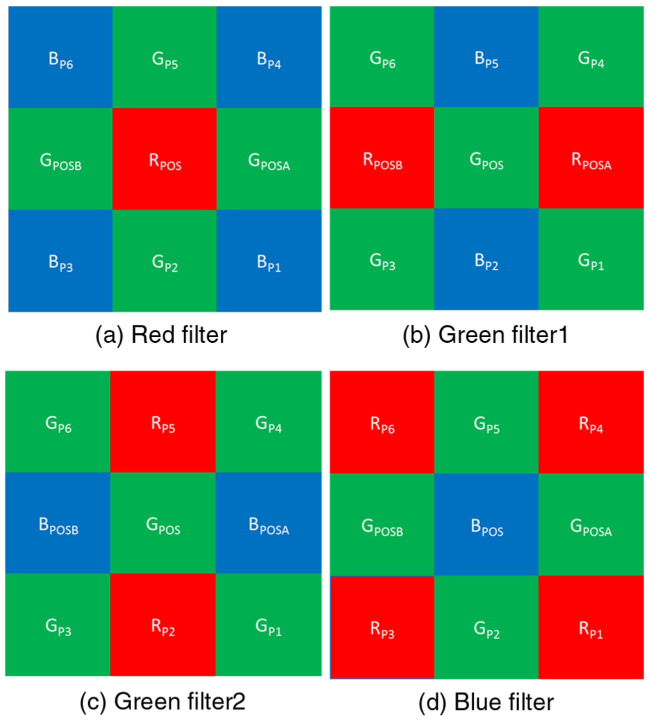


Fig. 16 Color filters and pixel positions for Bayer pattern used in color reconstruction algorithm: (a) red filter, (b) green filter 1, (c) green filter 2, and (d) blue filter.

$$(R^R_{\mathrm{POS}}, G^R_{\mathrm{POS}}, B^R_{\mathrm{POS}}) = \left(R_{\mathrm{POS}}, \frac{G_{P5}+G_{P2}+G_{\mathrm{POS}A}+G_{\mathrm{POS}B}}{4}, \frac{B_{P6}+B_{P4}+B_{P3}+B_{P1}}{4}\right), \tag{5}$$

where POS is the current pixel position to be calculated, POSA = POS − 1, POSB = POS + 1, P1 = POS + 250 + 1, P2 = POS + 250, P3 = POS + 250 − 1, P4 = POS − 250 + 1, P5 = POS − 250, P6 = POS − 250 − 1. Equations (6)–(8) should be applied when the known component is green filter 1, green filter 2, or the blue filter, respectively,

$$(R^{G1}_{\mathrm{POS}}, G^{G1}_{\mathrm{POS}}, B^{G1}_{\mathrm{POS}}) = \left(\frac{R_{\mathrm{POS}A}+R_{\mathrm{POS}B}}{2}, G_{\mathrm{POS}}, \frac{B_{P5}+B_{P2}}{2}\right), \tag{6}$$

$$(R^{G2}_{\mathrm{POS}}, G^{G2}_{\mathrm{POS}}, B^{G2}_{\mathrm{POS}}) = \left(\frac{R_{P5}+R_{P2}}{2}, G_{\mathrm{POS}}, \frac{B_{\mathrm{POS}A}+B_{\mathrm{POS}B}}{2}\right), \tag{7}$$

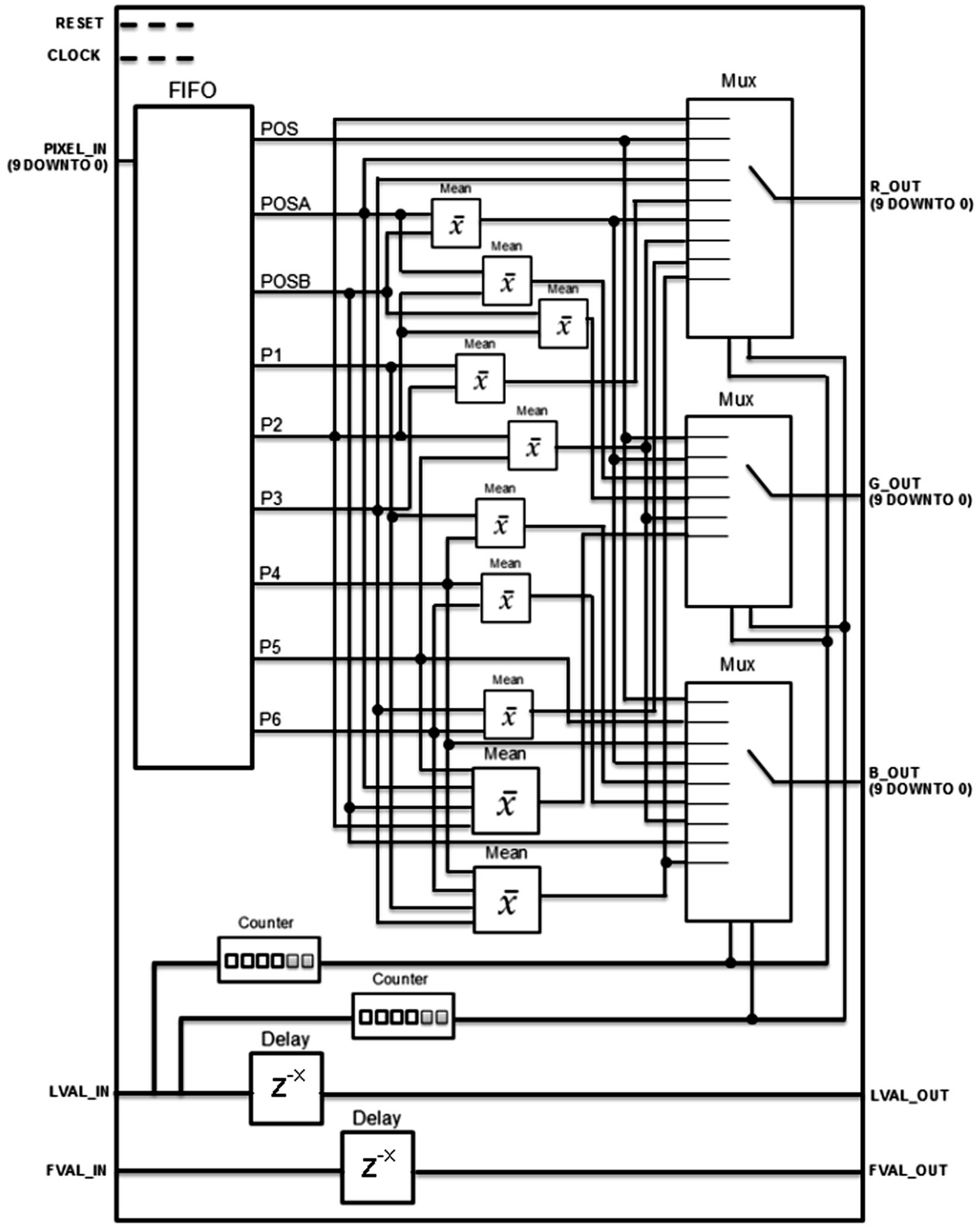


Fig. 17 Block diagram of the color reconstruction algorithm.

$$(R^B_{\mathrm{POS}}, G^B_{\mathrm{POS}}, B^B_{\mathrm{POS}}) = \left(\frac{R_{P6}+R_{P4}+R_{P3}+R_{P1}}{4}, \frac{G_{P5}+G_{P2}+G_{\mathrm{POS}A}+G_{\mathrm{POS}B}}{4}, B_{\mathrm{POS}}\right). \tag{8}$$

The implementation of this module is illustrated in Fig. 17. To compute all three components of a pixel, the values from three lines are necessary, except for the first and the last lines of a frame, which only require the values from two lines. In other words, the system uses a first in first out (FIFO) to allocate memory for three lines. The components of the frame's first pixel start being computed when the FIFO already contains two lines. Then the FIFO is shifted at each clock cycle to accommodate incoming pixels. At the border of the pixel matrix (frame) Eqs. (5)–(8) cannot be applied, as they would consider pixels outside the matrix. Therefore, for the outmost row's the equations are adjusted to use only those pixels actually present in the matrix. There are multiplexers at the end where the output is decided according to the position of the current pixel.

3.6 *Color Adjustment Matrix*

In this work, image data are acquired by a CMOS image sensor. A high color fidelity is required especially in the medical area because some medical diagnostics depend on color information. To render neutral colors correctly, a color adjustment algorithm is required. That is why the image data

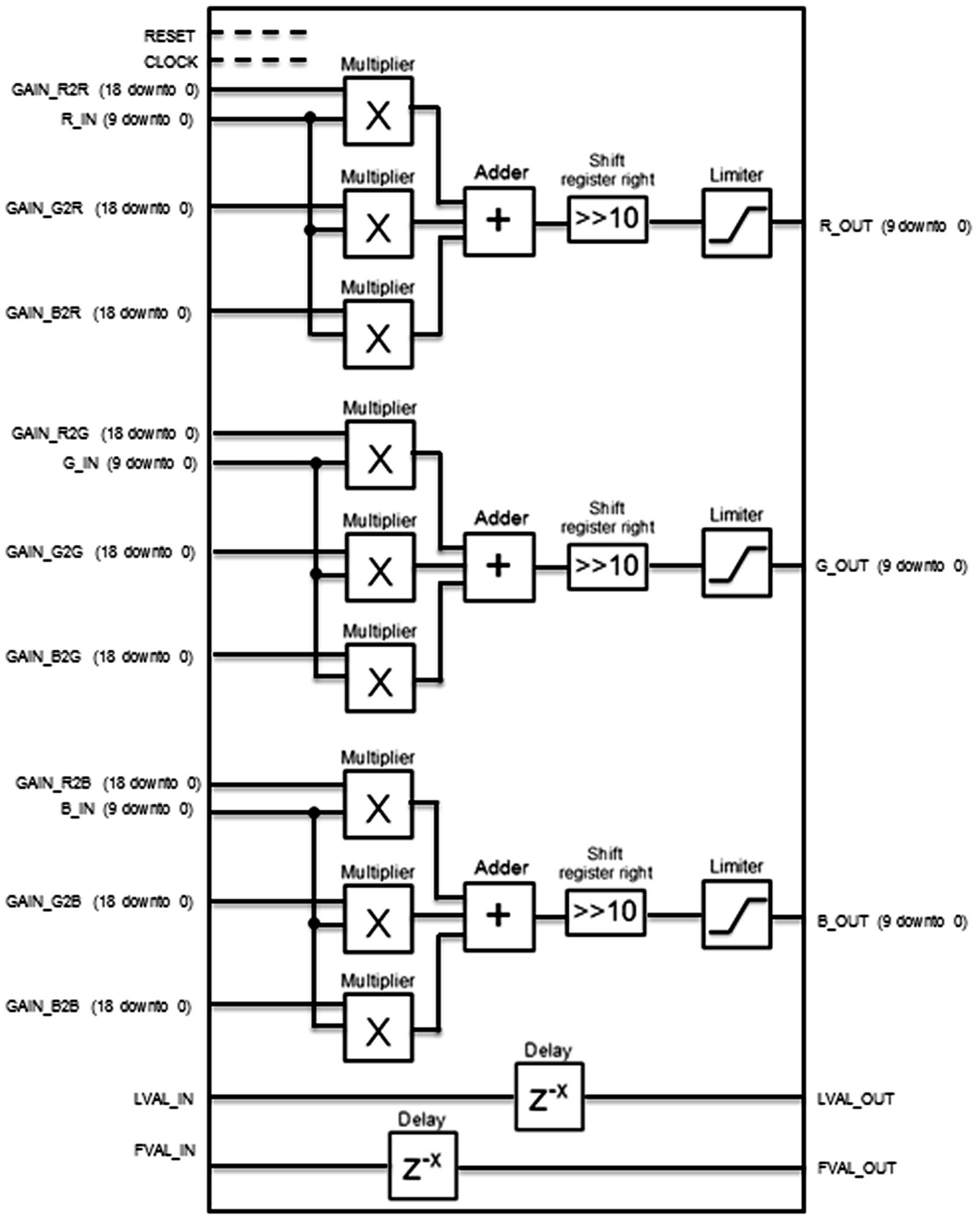


Fig. 18 Color adjustment matrix block diagram.

are transformed to new values so that it is appropriate for color reproduction or display, matching the natural color. In this module, the RGB components of each pixel are multiplied by a 3×3 matrix[26] as shown in the following equation:

$$[R_{\text{new}} \quad G_{\text{new}} \quad B_{\text{new}}] = [R_{\text{old}} \quad G_{\text{old}} \quad B_{\text{old}}] \times \begin{bmatrix} R_R & R_G & R_B \\ G_R & G_G & G_B \\ B_R & B_G & B_B \end{bmatrix}. \tag{9}$$

The goal of this matrix is to saturate the images, changing the pixel's RGB values to obtain better color images. To allow the brightness to stay the same, the condition

$$\begin{cases} R_R + G_R + B_R = 1 \\ R_G + G_G + B_G = 1 \\ R_B + G_B + B_B = 1 \end{cases}$$

should be respected. The gains have a range from -255 to $+255$. The implementation's block diagram is shown in Fig. 18. The matrix coefficients are received after a multiplication by 1024, since real numbers are not allowed on the FPGA. These coefficients are signed and have 19 bits. After computing the multiplications and the sums, the resulting values are divided by 1024 in order to obtain the actual values, and then they are limited to a range from 0 to 1023. By choosing the value of the matrix coefficient, users can control the color information according to their preference.

3.7 *Gamma Correction*

The digital sensor and display are almost linear in nature, however, because of human nonlinear vision brightness, correction is needed to faithfully reproduce brightness.[27] The solution to this problem is gamma-corrected pixels for every kind of display device.[28] This nonlinear function has an output which is compatible with human vision. Because of the use of the exponential form, it is impossible to directly implement the algorithm on the FPGA,[29] so the main method of reproducing the function faithfully is to use data points stored in the memory. The output of the function is saved in read-only memory (ROM) and according to the input pixels, the resulting pixels are output. However, if the system has a wide range of gamma values such as this implementation, the memory use is large. The alternative solution to this problem is a piecewise linear approach[30] (see Fig. 19). In this method, the nonlinear function is divided into 10 linear functions and then it is computed.

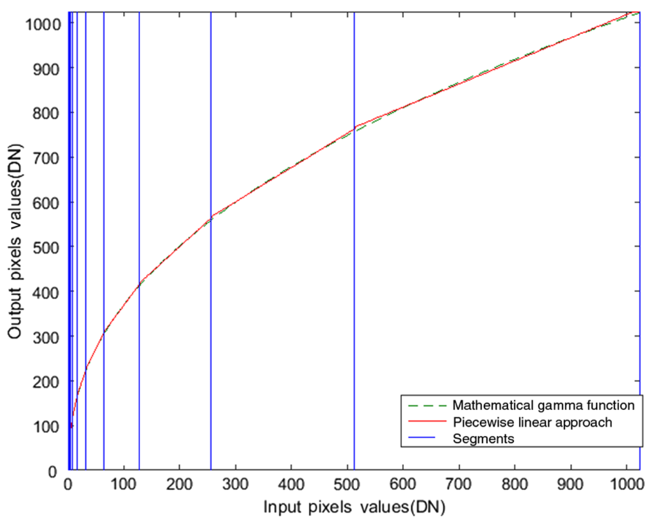


Fig. 19 The comparison between mathematical gamma and piecewise linear gamma in FPGA for the gamma value of 4.

The design presented in Fig. 20 has a gamma range from 0 to 4 with 0.1 resolutions, thus a total of 32 values of gamma can be chosen. The input and output of the gamma block is composed of 10 bits. The coefficients (C^s_{1x}, C^s_{2x}) for all

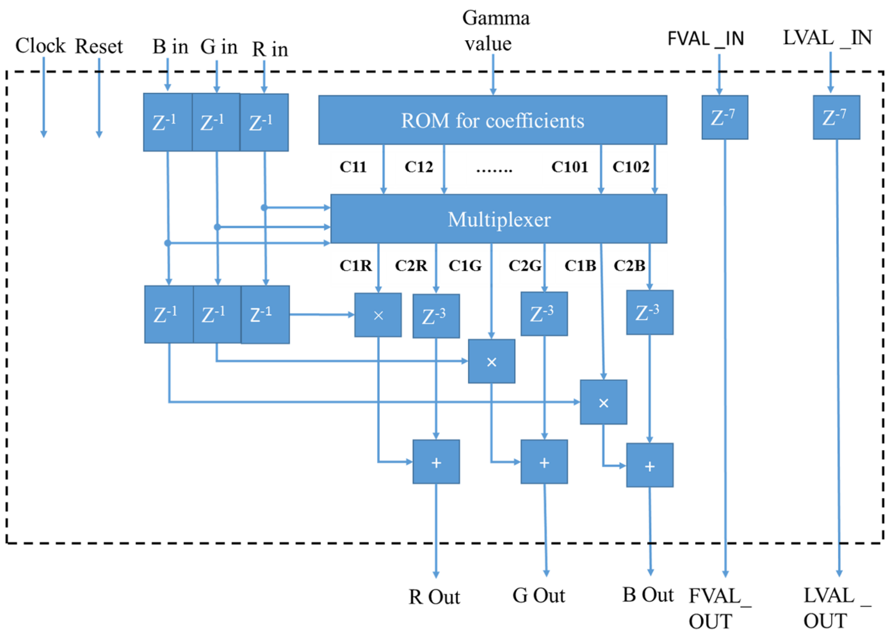


Fig. 20 Gamma block diagram. Reconstructed diagram from Ref. 27.

gamma values are saved in an ROM. When a gamma value [gamma in Eq (10)] is chosen by the user, the block receives the value using the wishbone bus. According to the gamma value, all the coefficients are passed to the multiplexer. The main purpose of the multiplexer is to select proper coefficients for each pixel (pixel_in) according to each slice(s). In the next step, the output pixel is calculated using Eq. (11)

$$\text{pixel\_output} = \text{input\_pixel}^{\frac{1}{\text{gamma}}}, \tag{10}$$

$$\text{pixel\_out} = C_{1x}^{s} \times \text{pixel\_in} + C_{2x}^{s}. \tag{11}$$

This piecewise linear implementation has the average mean absolute error (MAE) of 2.1747.[30]

3.8 *Adjust Brightness*

In this module, the brightness of the image can be increased by the application of a gain, which is multiplied by all the image pixels.[23] Since real numbers are not allowed on the FPGA, the gain value is received after a multiplication by 1024, and it is represented using 12 bits. The block diagram can be seen in Fig. 21.

After multiplication, the values are truncated to 10 bit maximum values (1024), and 2 bit right shifts, to convert to an 8-bit value. The output values are constituted by the 8 LSB.

4 Quality Evaluation and Comparison

After the implementation of the algorithms in hardware, the quality of the image was evaluated using both numerical approaches and user evaluation, as will be shown in the following sections. Four different scenes were chosen as well as four different conditions (i.e., combination between the parameters of algorithms is used) for both evaluations. The four scenes have four different images such as: Fig. 22(a) shows a 3-D shape, Fig. 22(b) shows a color pattern, Fig. 22(c) shows color bars, and Fig. 22(d) shows a black and white picture. The main purpose of the developed algorithms is to remove noise and bad pixels, and reconstruct and correct the colors. It makes the colorful images suitable for the evaluation. The gray scale image has the same importance since the gray tones must also be properly reconstructed. Each scene was captured using each one of four different conditions, and using each one of the three possible image-processing pipelines, which means that 64 images were captured. Each scene, processed by the FPGA algorithms with default conditions, can be seen in Fig. 22.

The conditions applied are described in Table 2, in which the fields in "parameters" column are described in the previous sections (with respect to each algorithm). Instead of just one setting of these user controllable parameters for precolor gain adjustment, color adjustment matrix, gamma, and adjust brightness, four different settings were chosen to increase the versatility of the experiment. The default condition is saved in the viewer so every time the user starts the camera, it is in that condition. The other three conditions are varied for test purposes.

4.1 *Numerical Comparison of Images*

An error is calculated using MAE using Eq. (12), in which Pix_F indicates the FPGA generated pixels, Pix_C indicates the compared pixels, i, j are the pixel positions, and Ch is the color channel. The result is shown in Fig. 23.

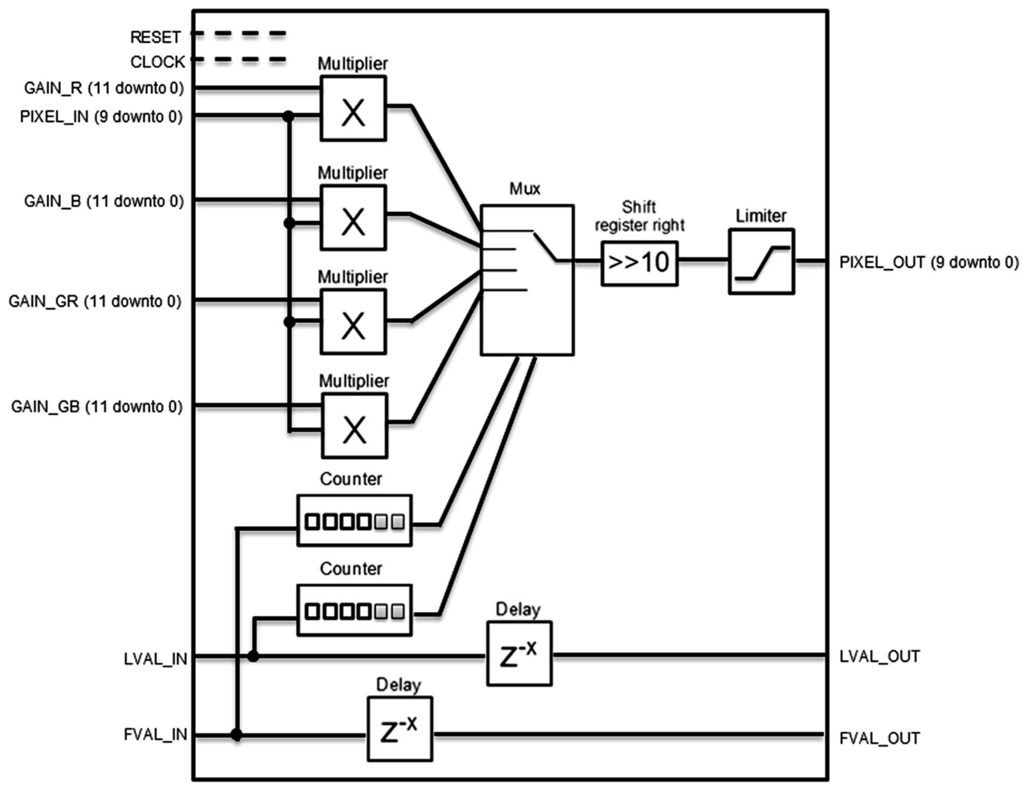


Fig. 21 Block diagram of the adjust brightness algorithm.

$$\text{MAE} = \frac{1}{3*250*250} \sum_{\text{Ch}=\text{R,G,B}} \sum_{i=1,j=1}^{i=250,j=250} (\text{Pix}\_F_{i,j}^{\text{Ch}} - \text{Pix}\_C_{i,j}^{\text{Ch}}). \tag{12}$$

It can be stated that in general, the images are quite different. For condition default, condition 1 and condition 2 FPGA versus Awaiba have fewer differences compared to OpenCV with the exception of scene 1 condition 1, and scene 4 condition 1. In the case of condition 4 in all scenes, FPGA versus OpenCV has fewer errors than FPGA versus Awaiba. More information can be seen in Fig. 23.

Nevertheless, the images do not need to be the same. The only requirement is that the FPGA processed image has good quality from the user's point of view.

4.2 *User Evaluation of Images*

Although the MAE shows the difference between the images, it does not directly reflect the quality of the image. So a qualitative approach was followed by asking users to evaluate the images. A subjective image quality evaluation was done through the user's feedback to understand how the image quality was changed by the image-processing pipeline implemented on the FPGA, in comparison to Awaiba and OpenCV[18] (which is considered a reference for open software) image-processing pipelines that are implemented in software and benefit from virtual infinite resources on their PC implementation. The method used to conduct this experiment is called Forced Choice[31] and the Psychopy[32] software was used to make the presentation available to the users. In this presentation, two kinds of comparison were done: the images adjusted on the FPGA versus the ones adjusted with Awaiba's image-processing pipeline; and the images adjusted on the FPGA versus the ones adjusted with OpenCV's image-processing pipeline. A pair of images was presented each time, and the observer

Table 2 Conditions description.

Algorithm	Parameters	Default condition	Conditions 1	Conditions 2	Conditions 3
Precolor gain adjustment	Red	1	1.33	1	1
	Green 1	1	1	1	1
	Green 2	1	1	1.1	1
	Blue	1	1	1	1
Color adjustment matrix	RR	1.2	0.7	1.15	1
	GR	−0.1	0.15	−0.07	0
	BR	−0.1	0.15	−0.07	0
	RG	−0.1	−0.05	−0.17	0
	GG	1.2	1.1	1.35	1
	BG	−0.1	−0.05	−0.17	0
	RB	−0.1	−0.05	−0.05	0
	GB	−0.1	−0.05	−0.05	0
	BB	1.2	1.1	1.1	1
Gamma	Value	1.2	1.5	1.0	1.2
Adjust brightness	Gain	1.2	1.68	1.48	1.2

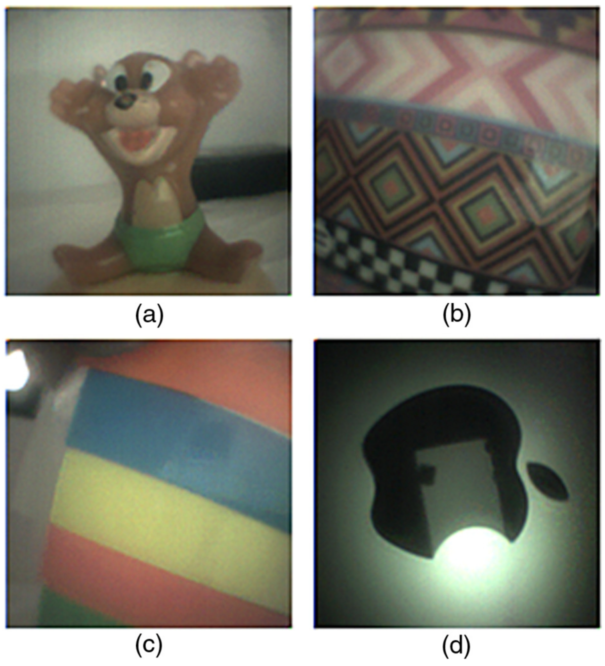


Fig. 22 Some images used for the user evaluation.

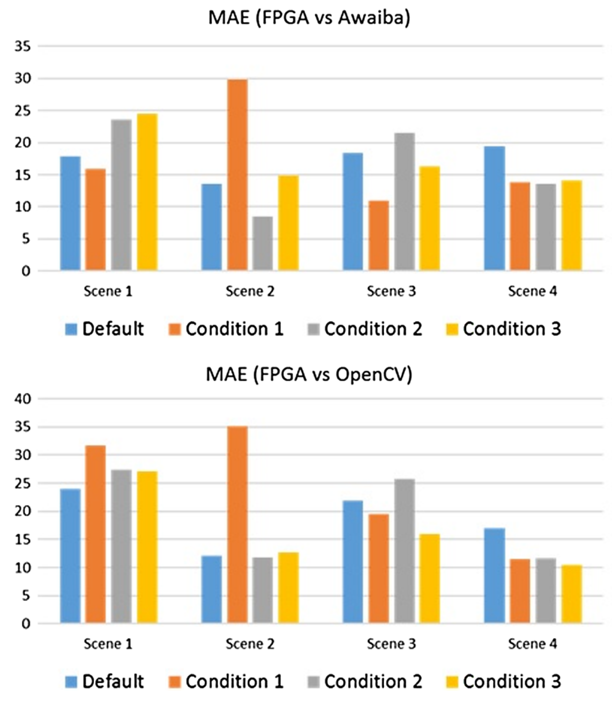


Fig. 23 MAE for four different scenes with four different conditions.

was asked to choose which one had the higher quality (see Fig. 24).

This experiment was done with 73 observers (45 males and 28 females), with an age range of 18 to 59. Each observer compared 32 pairs of images twice. The results of this experiment are summarized in Fig. 25.

Considering the user decision (the algorithm with more votes), in 81.25% of the comparisons between FPGA's image-processing pipeline and Awaiba's image-processing pipeline, the image processed with FPGA has more votes. When compared with the OpenCV image-processing pipeline, this percentage is 87.50% (Fig. 25). Considering the total number of votes in each option, for comparison between FPGA and Awaiba, the FPGA obtained 64.64% of the votes; for the comparison between FPGA and OpenCV, the FPGA obtained 74.83% of the votes (Fig. 25).

Altogether, this means that according to the users, the quality of the images processed by the FPGA image-processing pipeline is better or equal to those which are processed by Awaiba's and OpenCV's image-processing pipelines.

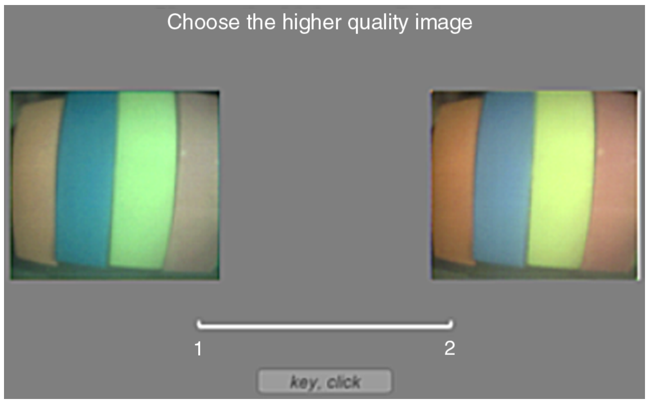


Fig. 24 Experiment user interface.

4.3 *Comparison between Different Implementations*

The pictures were captured on a computer and shown as a live video with "Max CPU frames" setting in the Awaiba viewer. The computer used has an AMDFX6300 six core processor, 16GB RAM and AMD Radeon HD 5450 Graphic Card. For the processing on the computer, OpenCV took 32 ms and Awaiba algorithms took 37 ms to process in the debug mode. The viewer first grabs the frame and then starts the process so the total delay will be one frame initial delay plus the processing delay. At the end, with computer process algorithms 31 fps for OpenCV and 27 fps for Awaiba algorithms can be achieved.

On the other hand, the FPGA processed algorithms, which are the same as Awaiba algorithms processed on FPGA, can reach 60 fps, almost twice the speed when compared to both computer-based algorithms.

Considering the resource use, Table 3 shows that the FPGA implemented algorithms use 5% FFs and 13% LUTs in the Spartan 6 FPGA. The most significant resource used by the algorithms is 108 BRAM16, which is 40% of the total in this FPGA. The next most significant resource use is 16% of DSP48A1s. So from the resources used by the system, it is clear that two cameras can be used

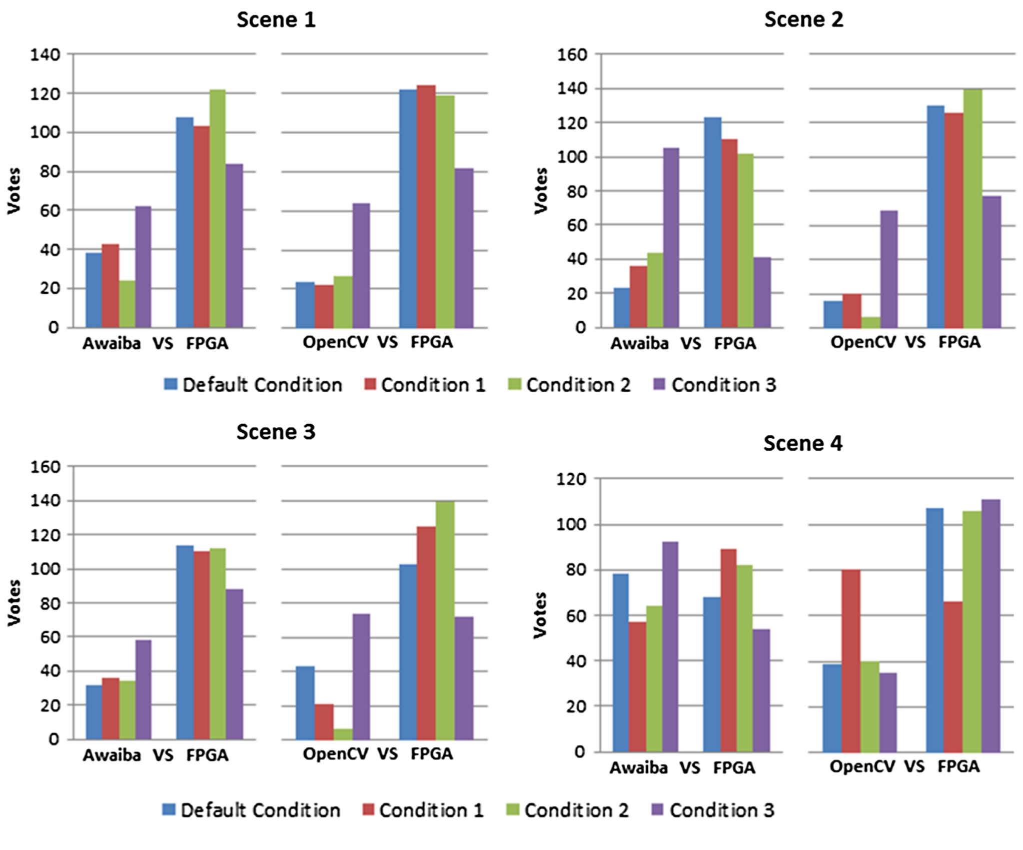


Fig. 25 The votes of user evaluation of images.

Table 3 Resources used by the FPGA implementation.

Resource	Amount used	Percentage of total resources (%)
FFs	10,781	5
LUTs	12,396	13
BRAM16	108	40
BRAM8	2	1
BUFGs	2	12
DSP48A1s	29	16

on a Spartan 6 FPGA. Using two cameras in the system, the total delay of each camera process will not increase because of the parallel processing nature of the FPGA implementation, which is not true for computers. However, the use of two cameras is in the scope of future research for 3-D image processing.

4.4 *Comparison between Different Design*

In the proposed implementation, a whole image-processing pipeline is done in an FPGA. In the other implementations presented in Table 4, collected from the literature, only compression and transmission are done. Due to FPGA implementation, a higher frame rate is achievable. The achievement in frame rate is doubled (60fps) compared to its nearest competitor which has 30fps [6,8] and thirty time faster than Ref. [9] which has only 2fps. In addition, it can be seen that the physical size of the sensor is the smallest among the ones used, but with a modest pixel number which gives an extra advantage in endoscopic methods where size does matters.

Considering the technology of these implementations, it can be seen that the recent trend is ASIC 180 nm,[5,6] FPGA 90 nm,[8,9] and 65 nm,[7] which are power efficient compared to processor-based solutions.

5 Conclusion

The work presented here is capable of implementing a full processing pipeline on an FPGA. From the user experience point of view, it produces better pictures than the software counterpart and the OpenCV, which is a reference for image processing. However, in the case of numerical comparisons of images, there are some differences between Awaiba and FPGA generated pictures. These differences were mainly generated because of the floating point to the fixed point conversion and some of the assumptions made for implementing algorithms such as gamma and frame mean.

The actual maximum speed of the algorithms could not be calculated because of the frequency range of the sensors. The system was capable of reaching the highest possible speed of the particular CMOS sensor. In this way, FPGA implemented algorithms can achieve twice the frame rate compared to the computer implemented algorithms.

The resources used by the algorithms indicate that in Spartan-6 FPGA, it is possible to use two sensors. Stereo type cameras would allow implementation of 3-D view, a project that will be covered in future research.

Acknowledgments

Acknowledgments to the Portuguese Foundation for Science and Technology for their support through Projeto Estratégico LA 9—UID/EEA/50009/2013. Also acknowledged is the Funding Program +Conhecimento II: Incentive System to Research and Technological Development and Innovation of Madeira Region II, through the project Vision 3D-MADFDR-01-0190-FEDER-000014. Acknowledgment to ARDITI—Agência Regional para o Desenvolvimento e Tecnologia under the scope of the Project M1420-09-5369-000001—PhD Studentship.

Table 4 Comparison between different design implementations.

System	Frame rate (fps)	Sensor size	Pixels	Clock (MHz)	Technology
Ref. 5	8	3.0 mm × 4.2 mm	320 × 288	40	ASIC 180 nm
Ref. 6	30	Not mentioned	VGA	20 to 24	ASIC 180 nm
Ref. 7	24	2.5 mm × 3.5 mm [29,33]	320 × 240	24	FPGA 65 nm
Ref. 7	12	2.5 mm × 3.5 mm [29,33]	320 × 240	12	FPGA 65 nm
Ref. 8	30	1.1 mm × 1.4 mm	640 × 480	24	FPGA 90 nm
Ref. 9	2	Not mentioned	480 × 480	12	FPGA90 nm
Awaiba	27	1.0 mm × 1.0 mm	250 × 250	50	Computer (Awaiba algorithms)
OpenCV	31	1.0 mm × 1.0 mm	250 × 250	50	Computer (OpenCV)
Proposed	60	1.0 mm × 1.0 mm	250 × 250	50	FPGA 45 nm

References

1. A. F. Peery et al., "Burden of gastrointestinal disease in the United States: 2012 update," *Gastroenterology* **143**, 1179–1187 (2012).
2. D.R. Cave et al., "A multicenter randomized comparison of the endocapsule and the Pillcam SB," *Gastrointest. Endoscopy* **68**(3), 487–494 (2008).
3. M. Schoberl et al., "Non-linear dark current fixed pattern noise compensation for variable frame rate moving picture cameras," in *17th European Signal Processing Conf.*, pp. 268–272 (2009).
4. I. Kuon, R. Tessier, and J. Rose, "FPGA architecture: survey and challenges," *Found. Trends Electron. Design Automation* **2**, 135–253 (2008).
5. X. Xie et al., "A low-power digital IC design inside the wireless endoscopic capsule," *IEEE J. Solid-State Circuits* **41**(11), 2390–2400 (2006).
6. X. Chen et al., "A wireless capsule endoscope system with low-power controlling and processing ASIC," *IEEE Trans. Biomed. Circuits Syst.* **3**(1), 11–22 (2009).
7. P. Turcza and M. Duplaga, "Low power FPGA-based image processing core for wireless capsule endoscopy," *Sensors Actuators A* **172**, 552–560 (2011).
8. D. Covi et. al, "Miniaturized digital camera system for disposable endoscopic applications," *Sens. Actuators A* **162**, 291–296 (2010).
9. Y. Gu et. al., "A new system design of the multi-view micro-ball endoscopy system," in *Annual Int. Conf. of the IEEE Engineering in Medicine and Biology*, Buenos Aires, pp. 6409–6412 (2010).
10. Y.-K. Lai and S.-M. Lee, "Dynamic gamma-correction algorithm for improving color LCD systems," in *IEEE Int. Conf. on Consumer Electronics*, pp. 811–812 (2011).
11. H. S. Malvar, L.-W. He, and R. Cutler, "High-quality linear interpolation for demosaicing of Bayer-patterned color images," in *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP '04)*, Vol. **3**, p. iii–485 (2004).
12. M. Qasaimeh, A. Sagahyroon, and T. Shanableh, "FPGA-based parallel hardware architecture for real-time image classification," *IEEE Trans. Comput. Imaging* **1**, 56–70 (2015).
13. D. Seidner, "Improved low-cost FPGA image processor architecture with external line memory," in *IEEE Int. Conf. on Industrial Technology (ICIT '13)*, pp. 1128–1133 (2013).
14. "Specification NanEye 2D Web v4.4," 2015, http://www.cmosis.com/products/product_detail/naneye (5 Nov 2015).
15. "Product sheet NanEye area scan sensors," 2015, http://www.cmosis.com/products/product_detail/naneye (5 Oct 2015).
16. "EFM-02 hardware reference (Hardware Revision 1.2)," 2014, www.cesys.com (15 July 2015).
17. "OpenCores, Wishbone B4," 2010, https://opencores.org/cdn/downloads/wbspec_b4.pdf (3 Dec 2016).
18. G. Bradski, "The opencv library," *Doct. Dobbs J.* **25**, 120–126 (2000).
19. C. Huddleston, *Intelligent Sensor Design Using the Microchip dsPIC*, Newnes, USA (2006).
20. C. Su and A. Amer, "A real-time adaptive thresholding for video change detection," in *Int. Conf. on Image Processing*, Atlanta, Georgia, pp. 157–160 (2006).
21. D. Samanta and G. Sanyal, "Novel approach of adaptive thresholding technique for edge detection in videos," *Proc. Eng.* **30**, 283–288 (2012).
22. S. May and P. Urbanek, "Sorting data in two clock cycles," 2005, EE Times-India, http://m.eetindia.co.in/ARTICLES/2005FEB/EEIOL_2005FEB01_EMS_TA.pdf?SOURCES=DOWNLOAD (3 December 2016).
23. S. Wright, *Compositing Visual Effects: ESSENTIALS for the Aspiring Artist*, Taylor & Francis, United Kingdom (2011).
24. K. S. Rani and W. J. Hans, "FPGA implementation of bilinear interpolation algorithm for CFA demosaicing," in *Int. Conf. on Communications and Signal Processing*, pp. 857–863 (2013).
25. R. Ramanath et al., "Demosaicking methods for Bayer color arrays," *J. Electron. Imaging* **11**, 306–315 (2002).
26. "Motorola semiconductor," Application Note AN1904/D, http://www.nxp.com/assets/documents/data/en/application-notes/AN1904.pdf (14 Feb 2016).
27. B. Walker, "Video gamma correction," U.S. Patents 8743234 B1 (2014).
28. P.-M. Lee and H.-Y. Chen, "Adjustable gamma correction circuit for TFT LCD," in *IEEE Int. Symp. on Circuits and Systems (ISCAS '05)*, pp. 780–783 (2005).
29. W. Jang et al., "Implementation of the gamma (γ) line system similar to non-linear gamma curve with 2bit error (LSB)," in *IEEE Asia Pacific Conf. on Circuits and Systems (APCCAS '06)*, pp. 283–286 (2006).
30. S. S. Mostafa et. al., "FPGA implementation of gamma correction using a piecewise linear approach for a small size endoscopic camera," in *Proc. Image Sensors and Imaging Systems*, Vol. **12**, pp. 1–6 (2016).
31. R. K. Mantiuk, A. Tomaszewska, and R. Mantiuk, "Comparison of four subjective methods for image quality assessment," *Comput. Graphics Forum* **31**(8), 2478–2491 (2012).
32. J. W. Peirce, "PsychoPy—psychophysics software in Python," *J. Neurosci. Methods* **162**, 8–13 (2007).
33. M. Vatteroni et. al, "Smart optical CMOS sensor for endoluminal applications," *Sensors Actuators A* **162**, 297–303 (2010).

Sheikh Shanawaz Mostafa received his BSc Engg degree in electronics and communication engineering from Khulna University in 2010 and his MSc degree in biomedical engineering from Khulna University of Engineering and Technology (KUET), in 2012. He is currently pursuing his PhD in networked interactive cyber physical systems at the Instituto Superior Técnico in a partnership with Carnegie Mellon University, Miti, and LARSyS. His research interest includes the development of hardware for biological/medical treatment/diagnosis techniques.

L. Natércia Sousa received her master's degree in telecommunications and energy networks engineering from the University of Madeira in 2015. For a year, she worked as a research assistant on a project at Madeira Interactive Technologies Institute (MITI). Then she joined Awaiba Lda Company, where she is currently working as a digital engineer focused on FPGA firmware development and image sensor test and characterization.

Nuno Fábio Ferreira received his diploma degree in networks and telecommunications engineering from the University of Madeira (UMa) in 2005, and his doctoral degree in electrotechnics engineering from the University of Aveiro. He is teaching at the UMa. He has integrated a team in an image processing R&D project, in collaboration with Awaiba Company. He currently works as a research fellow in numerical simulation of gas discharges with the Physics Group at the UMa.

Ricardo M. Sousa received his master's degree in telecommunications and energy networks engineering from the University of Madeira in 2014. In 2015, he won an IS&T/SPIE Electronic Imaging 2015 best student paper award. He currently works at Awaiba Lda, where he is an electronic engineer specialized in FPGA firmware development and image sensor test and characterization. He is a member of SPIE.

Joao Santos received his degree in software engineering in 2009 and his master's degree in software engineering in 2010 from the University of Madeira. He is working as a computer programmer in Awaiba, Funchal, Portugal. His work is focused on drivers development, design GUI, and image processing algorithms.

Martin Wäny graduated in physical electronics from IMT Neuchâtel in 1997. He worked on CMOS image sensors at IMEC Belgium in 1998. In 1999 he joined CSEM to work on high-dynamic range pixels and invented the LINLOG™ Technology, which he spun off as cofounder to Photonfocus AG in 2001. In 2004 he founded AWAIBA Lda a design company for specialized image sensors, which in 2015 became part of AMS. He currently is member of the AMS technology office and directs the marketing of micro camera modules.

F. Morgado-Dias received his MSc degree from the University of Joseph Fourier, Grenoble, in 1995, and his PhD from the University of Aveiro in 2005. He was a lecturer at the Technical University of Setúbal and is currently an assistant professor at the University of Madeira and a researcher at Madeira Interactive Technologies Institute. He is currently the vice president of the Portuguese Control Association. His research interests include renewable energy, artificial neural networks, and FPGA implementations.