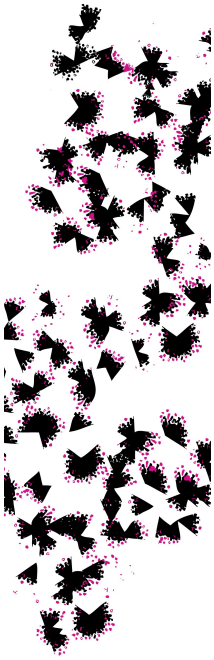


Exploiting FPGA on RaMstix for vision applications

K. (Kiavash) Mortezavi Matin

MSc Report

Committee:
Dr.ir. J.F. Broenink
Dr.ir. F. van der Heijden
Ir. E. Molenkamp

August 2017

038RAM2017
Robotics and Mechatronics
EE-Math-CS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Summary

The main goal of this assignment is to study the image processing algorithms for visual servoing applications which require fast image processing techniques. Thus, the feasibility of such algorithms is investigated for hardware implementation purposes, which can be achieved by means of an FPGA. The target FPGA for hardware implementation is the hosted FPGA on RaMstix board used in the ESL course. A camera module is interfaced with the FPGA and image processing is done after each pixel captured (i.e. streaming).

Several algorithms are studied and reformulated and thus mapped into hardware blocks, and the corresponding IPs are implemented. Moreover, each IP is tested using ModelSim and further IPs are verified on FPGA.

The results show that hardware solution for image processing algorithms of point operations class is faster and more memory efficient in comparison to the software solution. Ultimately, an alternative for the image processing part of the ESL course can be added to the design space exploration.

Contents

1. Introduction

This chapter provides a general introduction about the circumstances of this research. The research context, goals and approach are discussed. Finally, a broad outline of what is discussed in the further chapters is presented.

1.1 Context

In a negative feedback control system, the plant is controlled by the controller based on the difference between the desired state (reference point) and the current state of the plant. Information about the current state is reported by sensors. Such as microphone, ultra-sonic, infrared, light sensor, among others. In this research, a camera is the sensor in a negative feedback control system. More specifically, such system is called visual servoing, also known as vision-based robot control. Figure 1.1 shows the block diagram of this system.

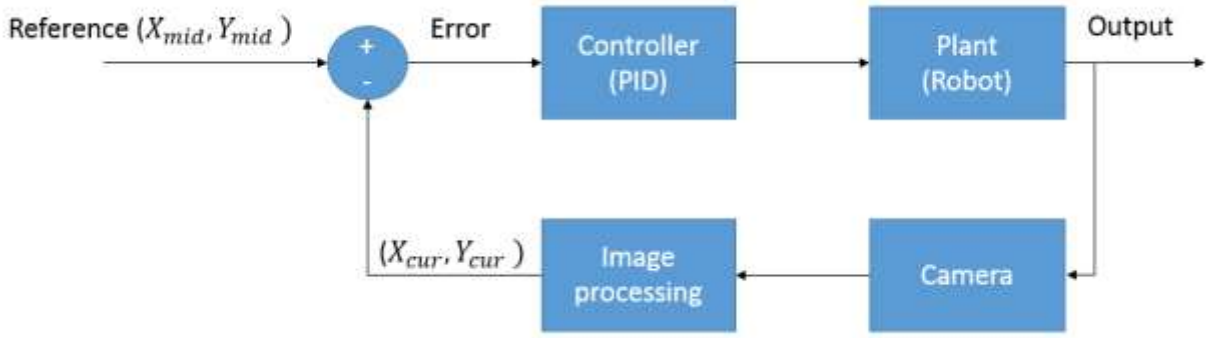


Figure 1.1: A negative feedback control system with a camera as its sensor

Using a camera as a sensor in a control loop is fairly convenient, due the amount of information and details retrieved from an image. However, fetching information from a frame requires image processing techniques, which is computationally expensive in most cases. The implementation of such algorithms can be realized in either software or hardware.

The context of this research is to evaluate the image processing part of the ESL course in particular, and evaluate the image processing in visual servoing for robotic applications in general.

One of the issues of visual servoing in software implementation is that capturing frames and applying image processing functions is considerably slow. In most of these implementations, a frame may be read and buffered and then filters can be applied to the buffered frame.

Interfacing a camera with an FPGA and implementing image processing functions on the FPGA could be a faster and optimal alternative for software solution.

One of the requirements of the ESL course is to capture frames with a camera, recognize, and track a colorful object with a white background. The image capturing and image processing is currently done

with software tools. The different possibilities of interfacing a camera with the RaMstix board has been studied in RaM group (Venema, 2016). One of the studied paths is interfacing a camera with the *Altera Cyclone III* FPGA on the RaMstix board.

The focus of this research is to implement image processing functions for object recognition on the FPGA which is hosted on RaMstix board.

1.2 Goals

The main goal of this research is to study image processing algorithms and choose the feasible ones for hardware implementation. The hardware solution should be a better alternative than the software solution. A new path for image processing may be added to the design-space-exploration phase of the ESL course. Furthermore, the implemented algorithms could be used for other setups in the RaM as a higher-level hardware blocks.

Interfacing a camera with the FPGA requires a driver module, which is able to retrieve pixels from the camera. Additionally, the location's information of the captured pixel in a frame, such as row and column addresses, can be provided by this module. After capturing the pixels, they can be processed by an image processing module and the desired information for the controller module extracted from the frame.

The focus of this research is on image processing module. The designed IPs of this module should be able to work independently of the interfaced camera with the FPGA and the controller. Figure 1.2 depicts the schematic of the modules discussed above, with the module which is the target of this project colored in green.

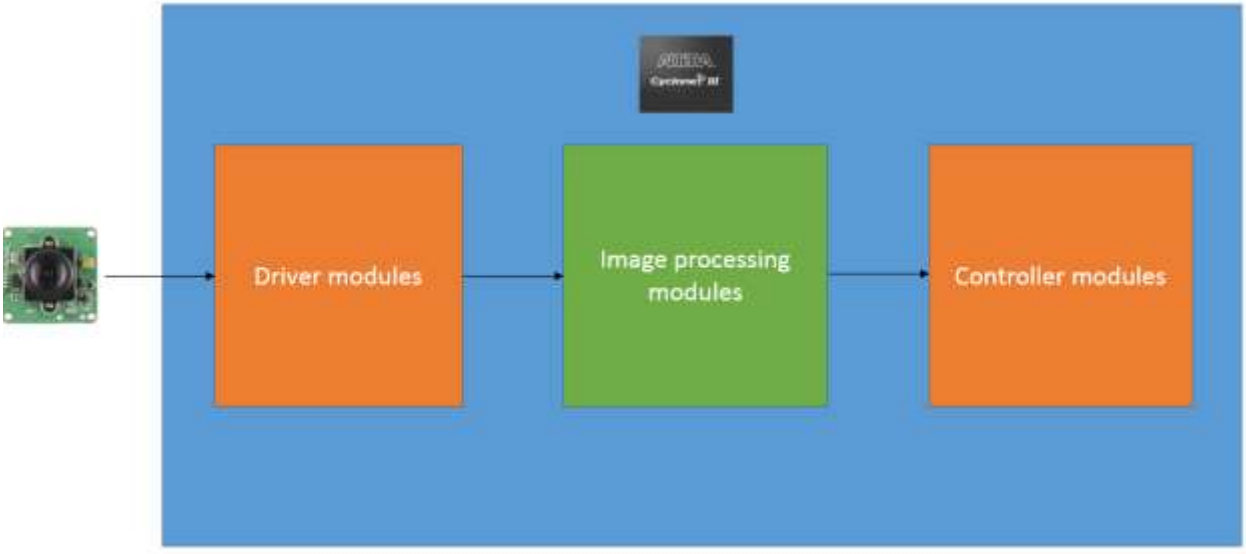


Figure 1.2: Three modules on an FPGA for visual servoing application

Although several algorithms may be fairly convenient to recognize an object from a frame, they may not be feasible for hardware implementation. Therefore, the candidate algorithms for hardware

implementation have to be satisfactorily accurate to recognize an object from a frame and feasible for hardware implementation.

1.3 Approach

To achieve these goals, relevant image processing algorithms related to object recognition which are appropriate for run-time applications are studied. Initially, algorithms are examined with MATLAB and modified accordingly for hardware implementation, with the resources and speed limitation of the FPGA on the RaMstix being taken into account. Moreover, it is investigated whether such hardware solution is capable of speeding up the image processing procedure.

The steps taken are summarized as follows:

1. Study image processing algorithms and make them feasible for hardware implementation;
2. Design and implement corresponding IPs for the algorithms;
3. Verify the functionality of the IPs using ModelSim and on RaMstix and;
4. Analyze time, accuracy and resource usage of the hardware solution.

The above mentioned steps is given as a workflow diagram depicts in Figure 1.3.

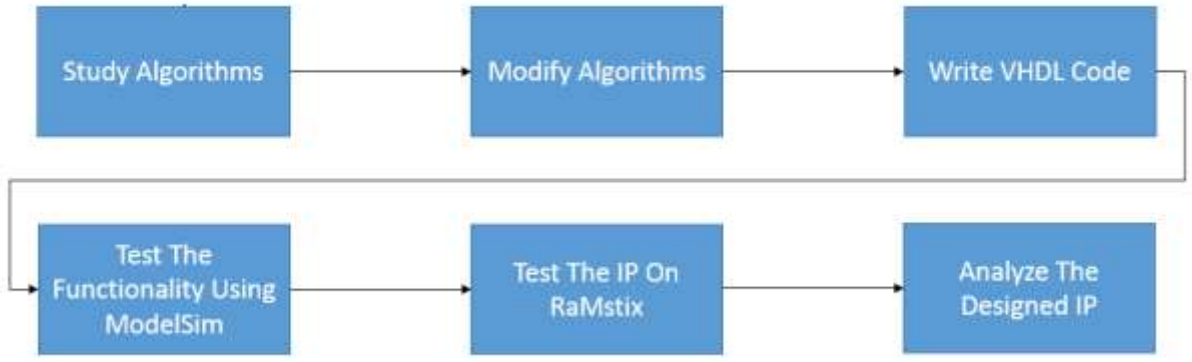


Figure 1.3: The workflow diagram of implementing an IP for an image processing algorithm

1.4 Outline

The background information of this research is given in Chapter 2. In Chapter 3, image processing classification and their respective implementation on an FPGA are described. Design and implementation of the chosen image processing algorithms for this research is documented in Chapter 4. Chapter 5 covers the result of the hardware solution and verification of the designed IPs. Finally the conclusions and recommendations are presented in Chapter 6.

2. Background

In this chapter the tools and techniques which are used in this research are briefly introduced.

2.1 Image processing

A digital image is a 2D array with discrete values, with each element being called a pixel. Each pixel is addressed by a row and a column number, and the value of a pixel is called pixel intensity.

In RGB images, a pixel intensity actually consists of three values: red, green and blue intensity. Moreover, a RGB image can be represented as a 2D array with three channels, with each channel storing the value of each color intensity. There are ways to represent RGB images as a 2D array with one channel, where RGB values are concatenated and stored in a binary vector, with each pixel intensity being represented with this binary vector. Figure 2.1 depicts two RGB standard representations, namely RGB565 and RGB888, with the former being used in this project.

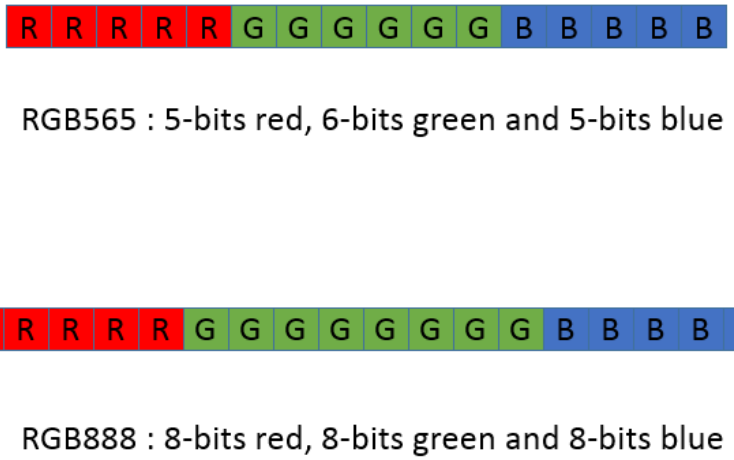


Figure 2.1: Representation of a single pixel of an RGB565 and RGB888 image

Applying mathematic operations to pixels and producing output pixels is called digital image processing. By processing an image, specific information can be retrieved from the image. For instance, the number of objects and their positions in an image can be determined. Image processing algorithms are classified into two main categories, which are discussed in detail in Chapter 3.

Image processing in visual servoing can be simply referred to retrieving desired information from a captured image for a controller.

2.2 MATLAB

MATLAB is a multi-paradigm numerical computing environment developed by *MathWorks*. It contains various built-in functions for working with images, in addition to several toolboxes for image processing.

In this research, MATLAB is used to study the results and also compare image processing algorithms. Furthermore, the proposed algorithms for hardware implementation are initially implemented and tested with MATLAB and then translated properly to VHDL code.

2.3 Gumstix

Gumstix Overo is a computer-on-module featuring *Texas Instruments* that hosts an ARM processor, with both C and C++ code can be compiled on Gumstix Overo. In this research, such board is used to communicate with IPs implemented on an external FPGA. Among other functionalities, this board is used for retrieving the result of processed pixels, setting parameters of IPs and loading pixels to IPs.

2.4 FPGA

A field-programmable gate array (FPGA) is an integrated circuit design that can be configured by hardware designers. Unlike microprocessors, arithmetic operations can be performed in a single clock cycle on an FPGA. Moreover, FPGAs are flexible and their structure can be reconfigured according to the design requirements. High level parallelism and pipelining can also be achieved.

An FPGA may have various type of resources, with three being the main ones: logic elements, memory bits and DSP slices. The principal building of the designed system is constructed with logic elements. Logic gates, registers, multiplexers, and others are constructed with logic elements, for instance. Buffers, RAMs, ROM, lookup tables, on the other hand, can be constructed with memory bits. DSP slice contains pre-built adders and multipliers, thus instead of building adders and multipliers with logic elements, data for addition and multiplication can use such pre-built DSP slice components.

In this research the target FPGA is an *Altera Cyclone III ep3c40q240c8n* which is hosted on the RaMstix. The target FPGA used in this research is shown below.

Table 2.1 shows the available resources of such FPGA.

Total logic elements	39,600
Total memory bits	1,161,216
Embedded Multiplier 9-bit elements	252
Total pins	129
Total PLL	4

2.5 RaMstix

RaMstix is a board developed by the RaM group, and it hosts a Gumstix Overo and an FPGA, which can communicate with each other and exchange data through a General Purpose Memory Controller (GPMC). In this research, image processing operations are implemented as IPs and uploaded to the FPGA, meanwhile the parameters of those IPs are set via a software program hosted on the Gumstix. The RaMstix board is shown in Figure 2.2 below.

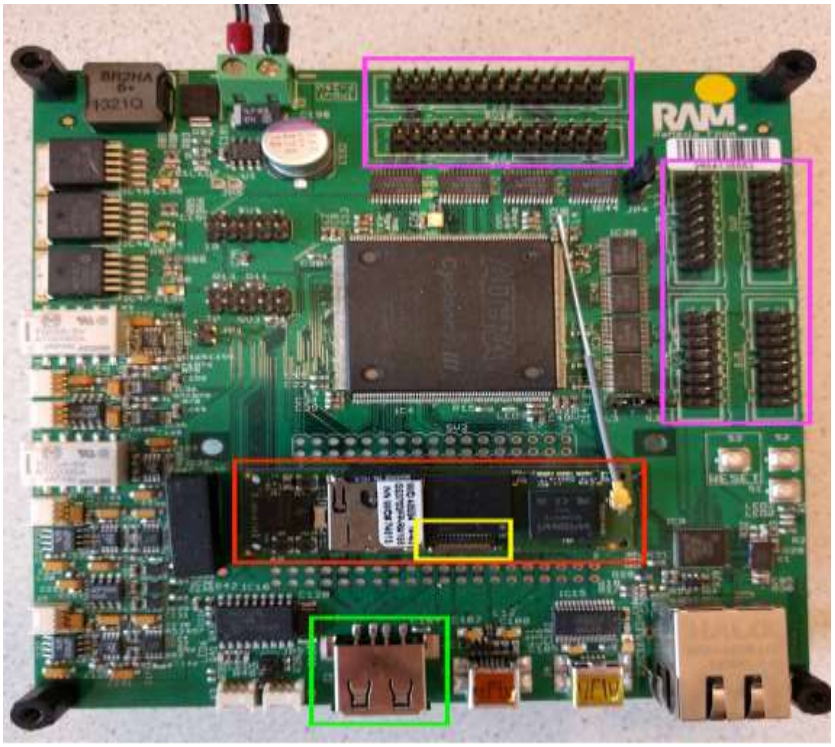


Figure 2.4: RaMstix board with colored boxes indicating the Gumstix (red) and FPGA I/O pins for camera connections (purple)

3. Image processing algorithms on FPGAs

Initially, this chapter discusses the advantages of implementing image processing algorithms on FPGAs. Further, a couple different classes of image processing algorithms are discussed and their implementation on an FPGA is investigated. This classification is according to the book Design for Embedded Image Processing on FPGAs (Bailey, 2011).

3.1 Why FPGAs for image processing?

As mentioned in Chapter 1, in most software implementations a frame is captured with a camera and stored in memory, and filters are subsequently applied to the buffered frame. This method is not time efficient due the fact the system is halted until a frame is captured and buffered.

In microcontroller systems, arithmetic operations are executed sequentially, thus applying filters to a frame also takes a long time. Real-time image processing libraries, such as OpenCV (Bradski, 2000), are proved to not be energy efficient (Venema,2016) and only available in certain (non real-time) operating systems.

FPGAs are an alternative platform for capturing and processing frames. Interfacing a camera with an FPGA makes it possible to apply filters to the captured pixels of a frame as the new pixels of the same frame are being captured - this is called pipelining. Since in an FPGA arithmetic operations like addition and multiplication can be performed in a single clock cycle, various operations can be applied to the captured pixels at the same time - this is called parallelism.

By taking advantage of pipelining and parallelism, it can be stated that using an FPGA for capturing and processing frames is faster than using a computer-on-module. Furthermore, an FPGA could be more energy efficient in many cases.

FPGAs have their own limitations: division is not a straightforward operation, external IPs may be required, proper datatype (usually fixed-point) must be proposed for intermediate signals, and memory is also limited. However, in visual servoing applications it is not necessary to buffer frames as whenever the desired information is extracted, the frame can be discarded. Memory limitation only occurs for object recognition algorithms that need to be applied to multiple frames. Therefore, it is important to choose the algorithms that can recognize an object from a single frame.

3.2 Image processing classification

According to the book Design for Embedded Image Processing on FPGAs (Bailey, 2011), image processing algorithms are classified in two main categories: point operations and neighborhood operations. In this section, such classes are briefly discussed and their main differences identified.

3.2.1 Point operations

Point operation is the simplest class of image processing operations. The output value of a pixel depends only on the corresponding pixel value of the input image. Point operation processing can be described by Equation 3.1.

$$Q[x, y] = f([I(x, y)]) \qquad (3.1)$$

Where $Q[x, y]$ is the output pixel with row address x and column address y, $I(x, y)$ is the input pixel with row address x and column address y and f is an arbitrary function.

Due the fact the output pixel value is only dependent on the input value and not on the location of the pixel in the image, point operation may be represented by a mapping or transfer function, as shown in Figure 3.1.

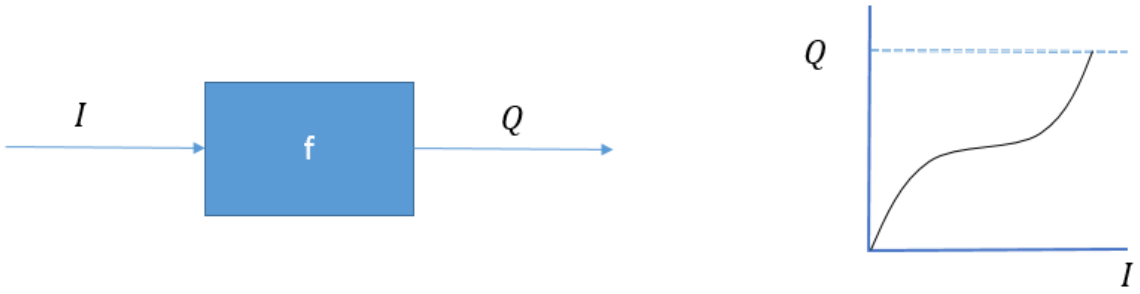


Figure 3.1: A point operation represented by a transfer function block diagram

Color thresholding is an example of point operation. In its fundamental form, color thresholding corresponds to comparing each pixel in the image with a threshold level (i.e. value) and sets the outputs to either a logical '1' or logical '0' (binary values), as shown in Equation 3.2 which is plotted in Figure 3.2.

$$Q = \begin{cases} 1, & I \geq th \\ 0, & I < th \end{cases} \qquad (3.2)$$

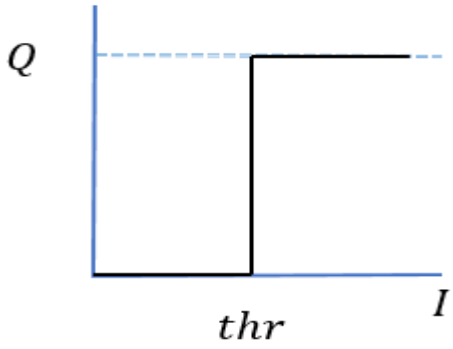


Figure 3.2: Simple thresholding plot

Contrast and brightness adjustment, image averaging and image subtracting are other examples of point operations.

3.2.2 Neighborhood operations

In neighborhood operations, in contrast to point operations, the output pixel does not only depend on the corresponding pixel value of the input image; it may also depend on the neighboring pixels of the input image.

Spatial filters are an example of this class, and is accomplished through an operation called 2D convolution. A 2D convolution is a neighborhood operation, in which each output pixel is computed by weighted summing of the neighboring input pixel (i.e. weighted averaging).

The matrix that contains the filter coefficients is called kernel matrix or window. The size of a window is represented by $n \times m$ where usually $n = m$ and it is an odd number. To apply a spatial filter to an image, the window is scanned throughout the image. Figure 3.3 illustrates applying a spatial filter to an image, with a kernel window of 3x3.

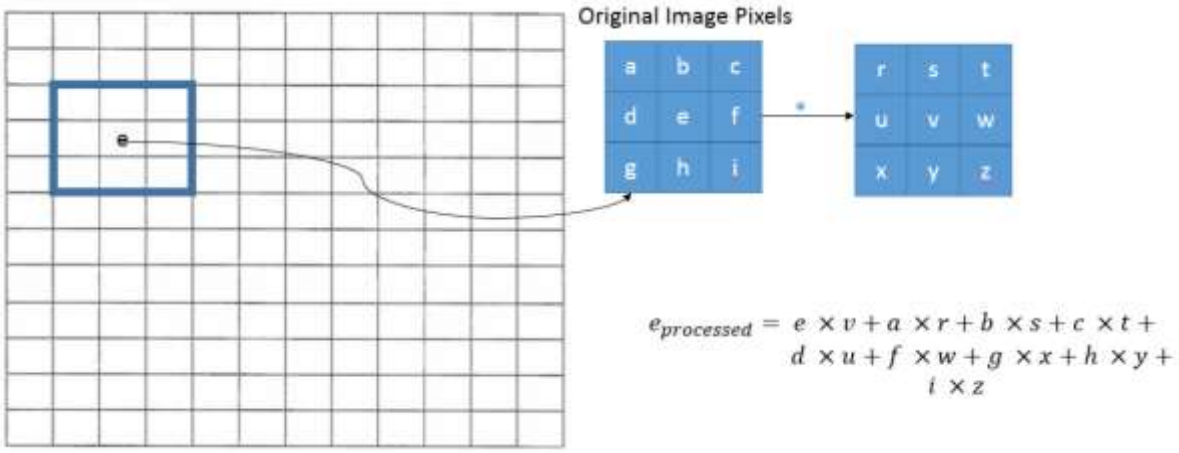


Figure 3.3: Applying a 3×3 spatial filter on pixel e

The filter type is determined by its coefficients window. Figure 3.4 depicts commonly applied windows for spatial filters.

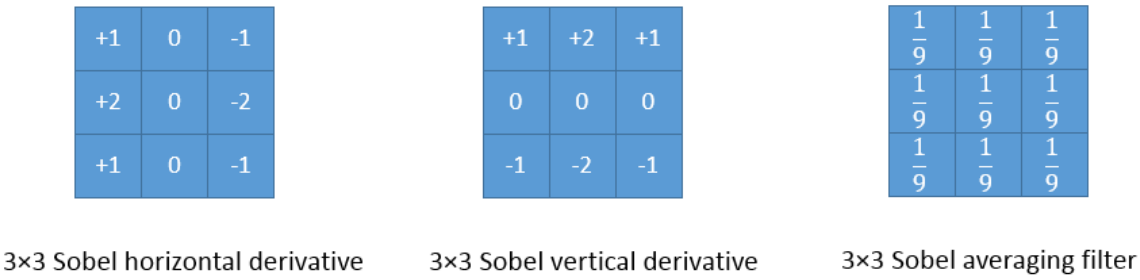


Figure 3.4: Three common spatial filter's windows

3.3 Image processing classes on FPGA

This section explores the implementation of the aforementioned image processing classes. Additionally, several time analyses for implementation of both classes is examined.

3.3.1 Point operations on FPGA

From a hardware implementation point of view, implementation of point operations may be easy, fast and memory efficient. Since each operation is applied exactly once to each pixel in the image, each captured pixel can be passed through an IP implementing the function.

Due to the fact each pixel is processed independently, point operations can also be easily implemented in parallel. After a captured pixel is processed, it can be discarded and does not need to be buffered.

The time required to process a frame (τ) with point operation filters can be calculated with Equation 3.3.

$$\tau = (p \times n \times m) + \sum_{i=0}^{L-1} f_i \qquad (3.3)$$

Where p is the average time to capture a single pixel in term of clock cycles, n is the number of rows of the input frame, m is the number of columns the input frame, L is the number of cascaded filters and f_i is the firing time of each cascaded filter in term of clock cycles.

Figure 3.5 shows an example of calculating τ of a system with a 5×5 frame with each pixel being captured in 2 clock cycles. Clock cycles are represented as *clk*.

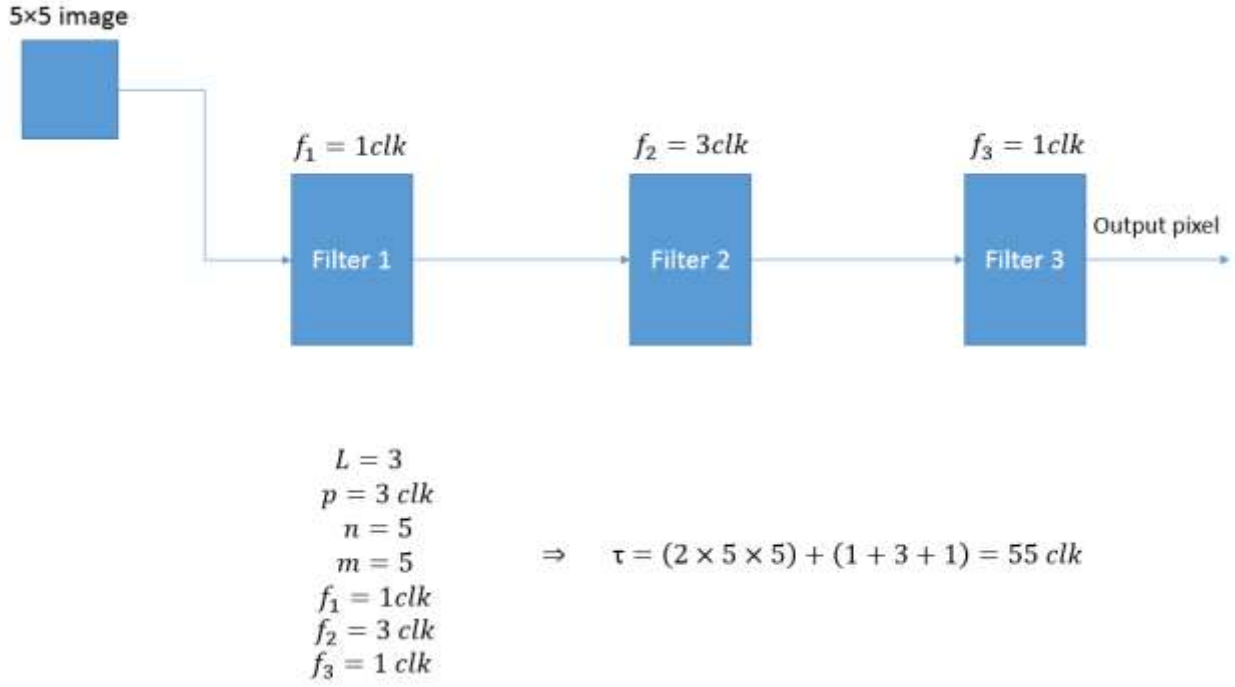


Figure 3.5: An example of calculating τ of a system with 3 cascaded filter

3.3.2 Spatial filters on FPGA

As described is section 3.2.2, in spatial filters each output pixel is the weighted sum of neighboring input pixels. In the software approach, both input and output frames are stored in buffers, and in order to apply a filter, the coefficients window is scanned throughout the buffered frame. However, in the hardware approach, the intention is to pipeline the capturing and filtering process. Therefore, instead of scanning the coefficients window, the input image is streamed through the coefficients window.

Unlike point operations, an input pixel does not get involved only in computing the value for the corresponding location in the output image: it may also take part in calculating other output pixels. For this, reason a pixel cannot be discarded immediately and it has to be buffered until it is not required anymore.

Pixels of a frame are captured row by row by an FPGA. Arriving pixels must be streamed through the coefficients window, and due to this fact, cashing for stream processing of a $w \times w$ coefficients window requires series of $w - 1$ row buffers with the size of the frame's column, as depicted in Figure 3.6. Thus,

one can conclude that implementation of spatial filters on FPGA demands additional memory to buffer input pixels.

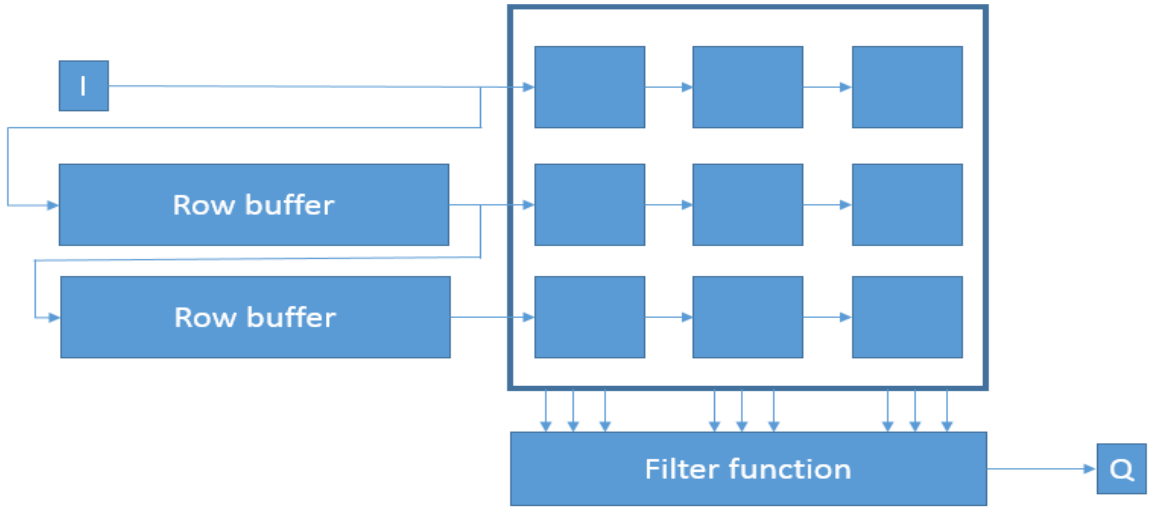


Figure 3.6: Streaming a 3×3 coefficients widow

In order to calculate the value of a specific output pixel, the system has to be halted until all the required input pixels are captured. This halt time (h_τ) is expressed in terms of the number of captured pixels and can be calculated with Equation 3.4.

$$h_\tau = \frac{w-1}{2}[m+1] + 2 \qquad (3.4)$$

Where w is the dimension of the coefficients window, and m is the number of columns of the input frame. For example, the filter computation for determining the first output pixel of a frame with 5 columns and n rows with a 3×3 coefficient window starts after capturing and buffering the first 8 pixels, as calculated below:

$$h_\tau = \frac{3-1}{2}[5+1] + 2 = 8$$

The time required to process a full frame (τ) with a spatial filter can be calculated with Equation 3.5.

$$\tau = (p \times n \times m) + h_\tau \qquad (3.5)$$

3.4 Next steps and summary

Based on the context and goal of this research, relevant image processing algorithms have been studied and their classification is considered. Their attribute are figured out and adopted to be mapped into hardware. Each candidate algorithm for hardware implementation, is implemented as a separated IP. Implemented IPs are verified using ModelSim and tested on RaMstix board.

Image processing algorithms are classified into two main categories, point operations and neighborhood operations. Point operations are simple, fast and memory efficient to be implemented as a hardware block. The FPGA on RaMstix board is small and resource limited, thus in this research the focus is to implement hardware blocks for several point operation image processing algorithms.

4. Design and implementation

The main topic of this chapter is about proposing and altering image processing algorithms for hardware implementation. With the image processing requirement of the ESL course being discussed as an example. Furthermore, the implementation of color thresholding operation is examined, and an algorithm to mark binary images which is suitable for hardware application is introduced and its implementation is discussed. Finally, the implementation of noise reduction after thresholding and before marking an image on the FPGA is explained.

4.1 ESL image processing requirements

The last part of the ESL course is about vision in loop, in which a colorful shape with a white background has to be captured by a camera and tracked by actuating the servos. The camera is interfaced with Gumstix through an USB port.

Thresholding operation can be applied to the captured frame, and after thresholding the colorful shape is represented by white pixels and its background represented by black pixels (i.e. binary image). An algorithm has to be proposed to mark a position of one of the white pixels, which could belong to the center of the shape or any other part of the shape. Then, servos move the camera until the marked pixel is positioned around the middle of the frame. One of the possible algorithm that students use to mark a pixel of the frame is described in the next section.

4.2 Erosion filter

One possible way to determine the center of a shape with white pixels in a binary image is erosion filter, which is one of the fundamental operations in morphological image processing and its task is to, as the name suggests, erode the white pixels.

A coefficients window, called Structure Element (SE), with either logical ‘1’ or logical ‘0’ as coefficient values is scanned through a binary image. The output pixel is set to logical ‘1’ if the corresponding location of the input pixel and its neighboring pixels are matched with the coefficient values of the SE, and to logical ‘0’ otherwise. Figure 4.1 shows an example of applying an erosion filter to a binary image.

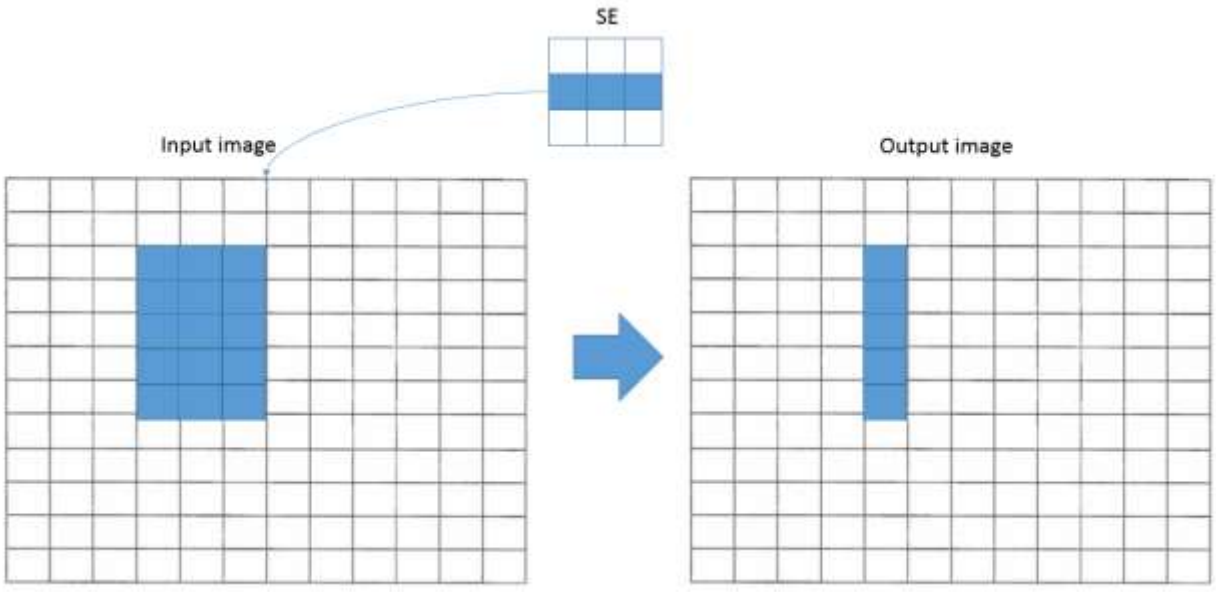


Figure 4.1: A simple erosion filter example

If the SE has the same structure as the shape of interest in a binary image, the only pixel which fulfills the erosion filter's condition is the pixel that belongs to the center of the shape, as shown in Figure 4.2 below.

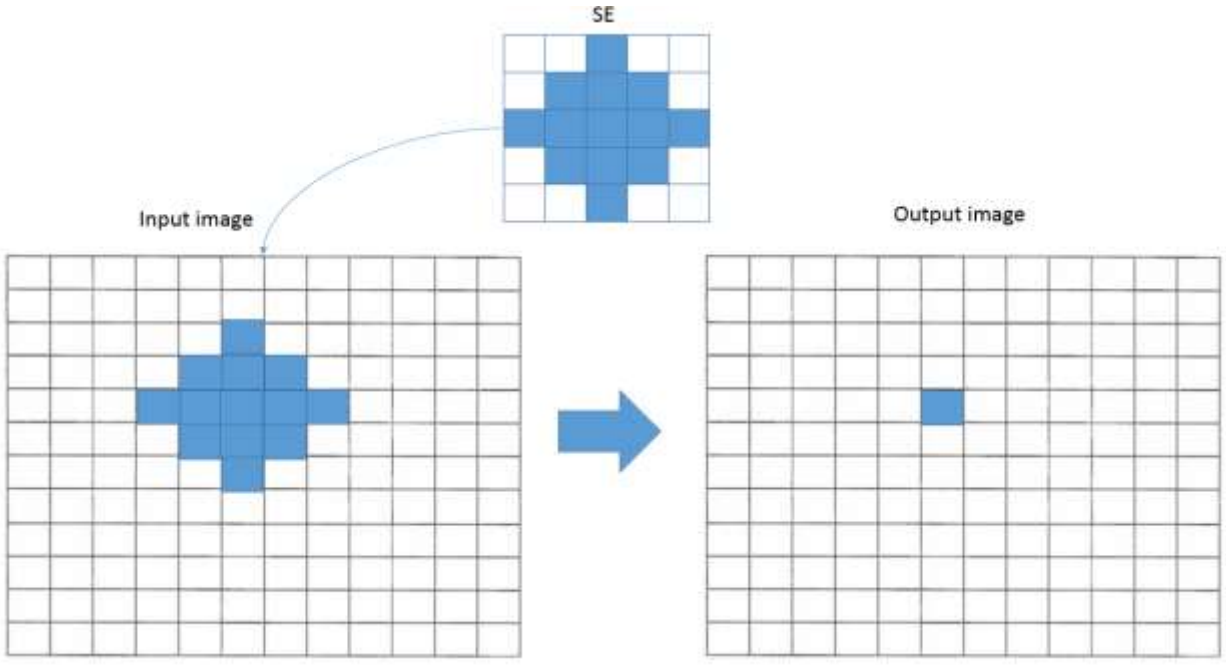


Figure 4.2: An erosion filter applied to find the center of the shape in a binary image

Since the SE is a window with constant values, if the distance between camera and object changes, the proposed SE will not correspond to the shape anymore, and the center of the shape cannot be detected.

Therefore, the distance between camera and object must be constant at all times with this approach. On the other hand, erosion filter is classified into neighborhood operations and the hardware implementation is slow and memory inefficient, thus it is not suitable for real-time applications.

4.3 Color thresholding on FPGA

Implementation of the RGB color thresholding IP is straightforward: input pixels are compared with a threshold level and the output is set to either logical '1' or '0' based on the comparison result.

Initially, a sub-IP is designed which compares color intensity and threshold value according to the given comparison operation. This IP is named *color_thresholding* and its structure are shown in Figure 4.3. Since each pixel of a RGB image is represented by three color intensities (red, green and blue), three of such IPs are required, one for each channel. The output pixel value is determined by simply AND'ing the result of all IPs, as shown in Figure 4.4.

Details of the implementation can be found in *RaM_Vision_RGB_Threshold.vhd* file, attached to this report.

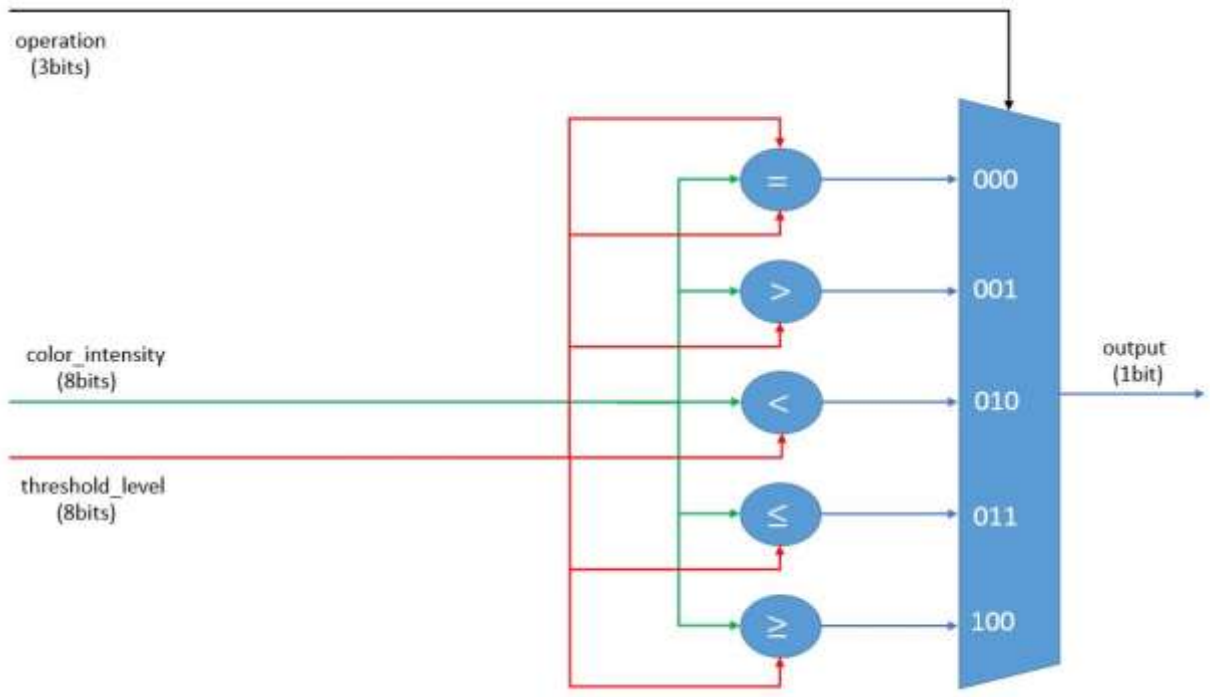


Figure 4.3: Structure of the *color_thresholding* IP

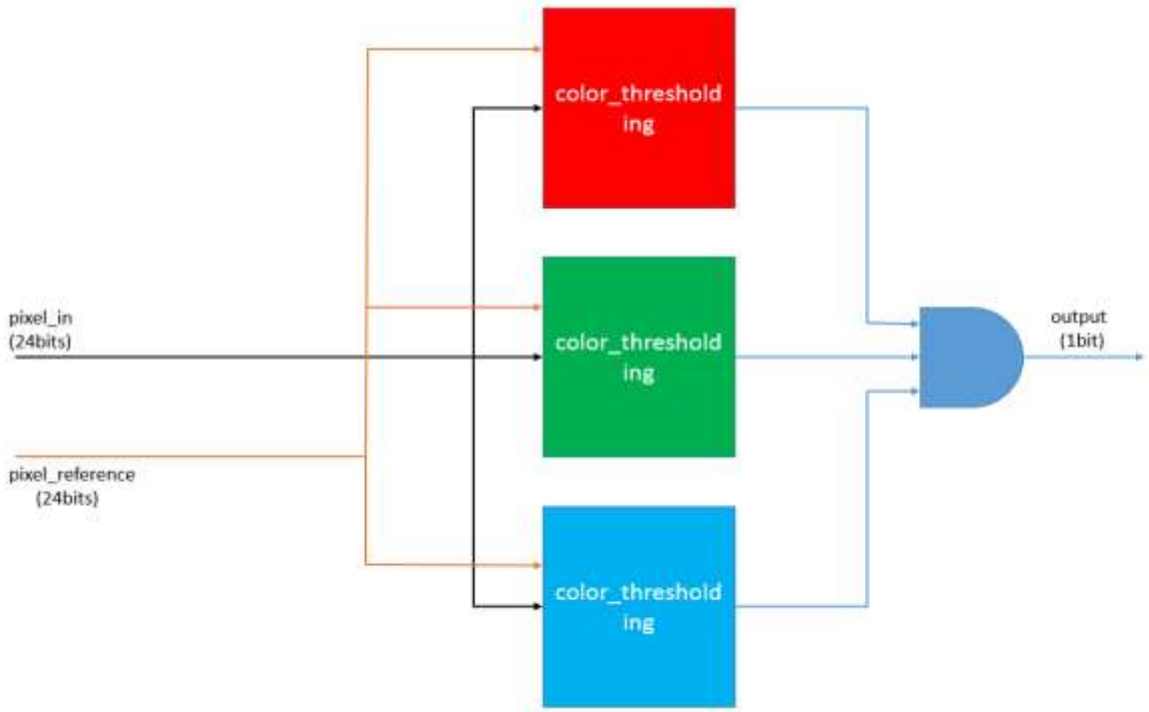


Figure 4.4: Thresholding pixels with three *color_thresholding* IPs

4.4 Introduction to the MRCC marker algorithm

Maximum Row, Center Column (MRCC) algorithm is designed to mark a pixel in a binary image. As its name suggests, it searches for a row which contains most white pixels, which can be achieved by summing the pixels values of each row. The row with the maximum sum is the row of interest.

It is possible, however, that an image has multiple rows with the same maximum sum. In this case, the row of interest is determined by averaging the first and last row indexes with the same maximum sum.

For instance, if rows number 2, 5 and 10 contain the maximum sum, the row of interest is $\left\lfloor \frac{2+10}{2} \right\rfloor = 6$. In this example, row 6 is determined by the rows with the maximum sum, although it might not be the one with the maximum sum.

The center column of each row is computed by averaging the column index of the first and last detected white pixel in each row. The center column which belongs to the row with the maximum sum is selected as the column of interest. If an image contains more than one row with maximum sum, as the example above, the center column is determined by the average of the center column of the first and last rows with the maximum sum.

The Pseudocode of MRCC algorithm is depicted in Figure 4.5 below, and Figure 4.6 illustrates an example of marking a pixel in a binary image with this algorithm.

```
%given a binary image(im) of size nxm
for i = 1:n,
    for j = 1:m,
        if(first white pixel deteted in the row )
            first_white = m;
            last_white  = m;
        elseif(the pixel is white and is not the first white detected pixel)
            last_white  = m;
        end if;
        row_sum[n]    = row_sum[n] + im[n,m];              %sum the value of the pixels
        col_center[n] = (first_white + last_white)/2;      %register the centercolumn of each row
    end for j;
end for i;

max_sum_indexes = maxIndexes(row_sum);                     %store the indexes of the maximum values of row_sum;
row_index_out = (max_sum_indexes[first] + max_sum_indexes[last])/2;
col_index_out = (col_center[[max_sum_indexes[first]]] + col_center[max_sum_indexes[first]])/2;
```

Figure 4.5: Pseudocode of MRCC algorithm

MRCC algorithm does not necessarily marks a pixel inside the shape. The marked pixel could be also located outside the shape, but somewhere close to it. Figure 4.7 presents an example of marking a pixel outside the shape with MRCC algorithm.

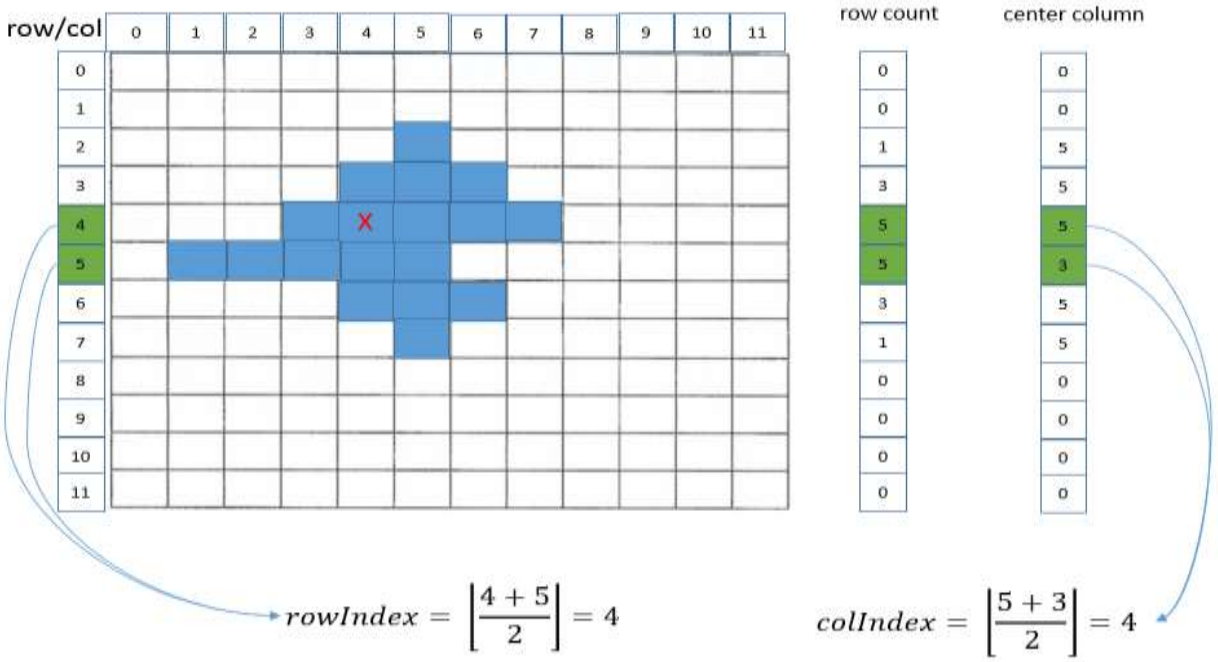


Figure 4.6: Marking a pixel of a binary image with MRCC algorithm

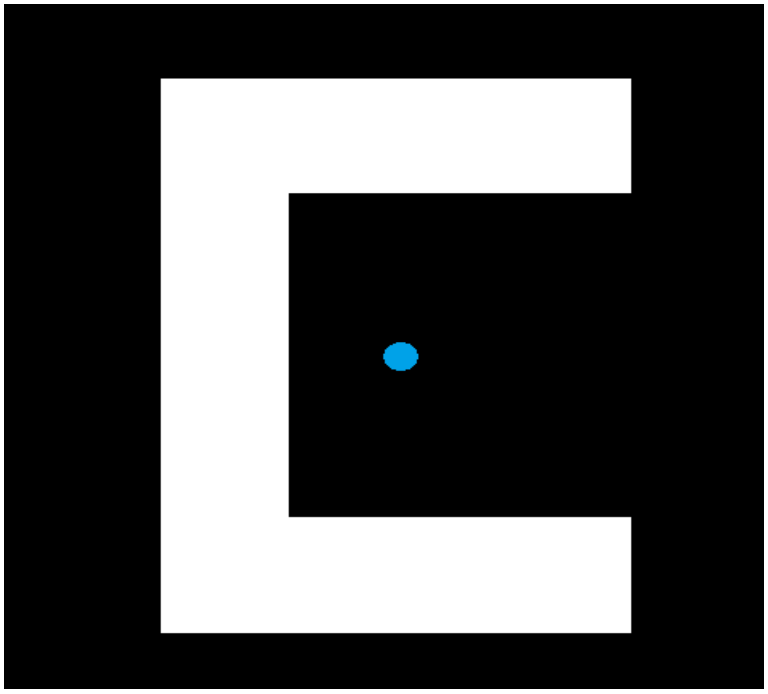


Figure 4.7: A pixel is marked outside the shape by MRCC (blue dot)

MRCC is not a rotational invariant algorithm, which means if the orientation of the shape changes, the marked pixel can be in a different position. Figure 4.8 depicts such phenomenon, by applying MRCC algorithm to the shape presented in Figure 4.7, but rotated $90°$ clockwise.

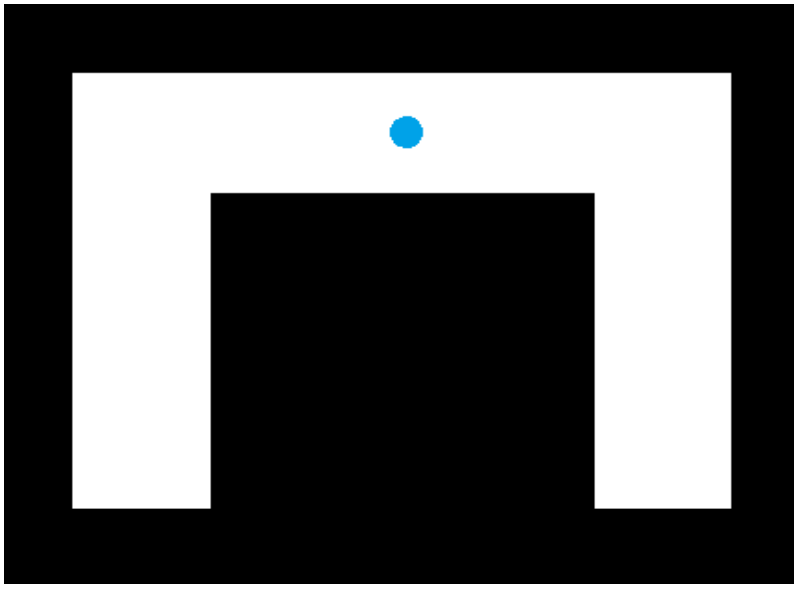


Figure 4.8: Shape in Figure 4.7 rotated $90°$ clockwise and marked with MRCC (blue dot)

It is noticeable that MRCC algorithm cannot be applied to binary images that contain multiple shapes or lots of noise. If noise is present in a binary image, the MRCC algorithm might mark the wrong pixel which may be located far away from the shape. This issue is further discussed in section 4.6.

4.5 MRCC marker algorithm implementation on FPGA

MRCC is classified as a point operation algorithm and its hardware implementation is simple, easy and memory efficient. After each captured pixel goes thorough the thresholding IP, it is forwarded to the MRCC IP.

One of the tasks of this IP is to sum up the pixels values corresponding to each row and compare the result with the maximum sum which was determined in previous rows. Two registers are used for this purpose, namely *cnt* and *cnt_reg*. The former is an accumulator register and its output is fed back to one input of the adder. The other input of the adder is connected to the arriving pixel value. On the other hand, *cnt_reg* stores the maximum sum of the previous rows. Two additional registers are used to hold the index of the rows with maximum sum, namely *row_idx_reg_0* and *row_idx_reg_1*.

When a new row is detected, the value of *cnt* and *cnt_reg* are compared. If *cnt* is smaller than *cnt_reg*, *cnt* is set to zero. Otherwise, it means that a new row with maximum sum is detected, therefore, *cnt_reg* is set to the value of *cnt* and *cnt* is set to zero. Furthermore, registers *row_idx_reg_0* and *row_idx_reg_1* are set to the value of the current row index. If *cnt* is equal to *cnt_reg*, it means that the current row has the same sum as the last registered row with maximum sum. In this case, only *row_idx_1* is updated to the index of the current row. Finally, the index of the row of interest is calculated by adding *row_idx_reg_0* and *row_idx_reg_1* and the result is shifted to the right by one bit (i.e. divided by 2) – essentially the middle value. Figure 4.9 shows the corresponding circuit of the process explained above, with flip-flops not being depicted for simplicity.

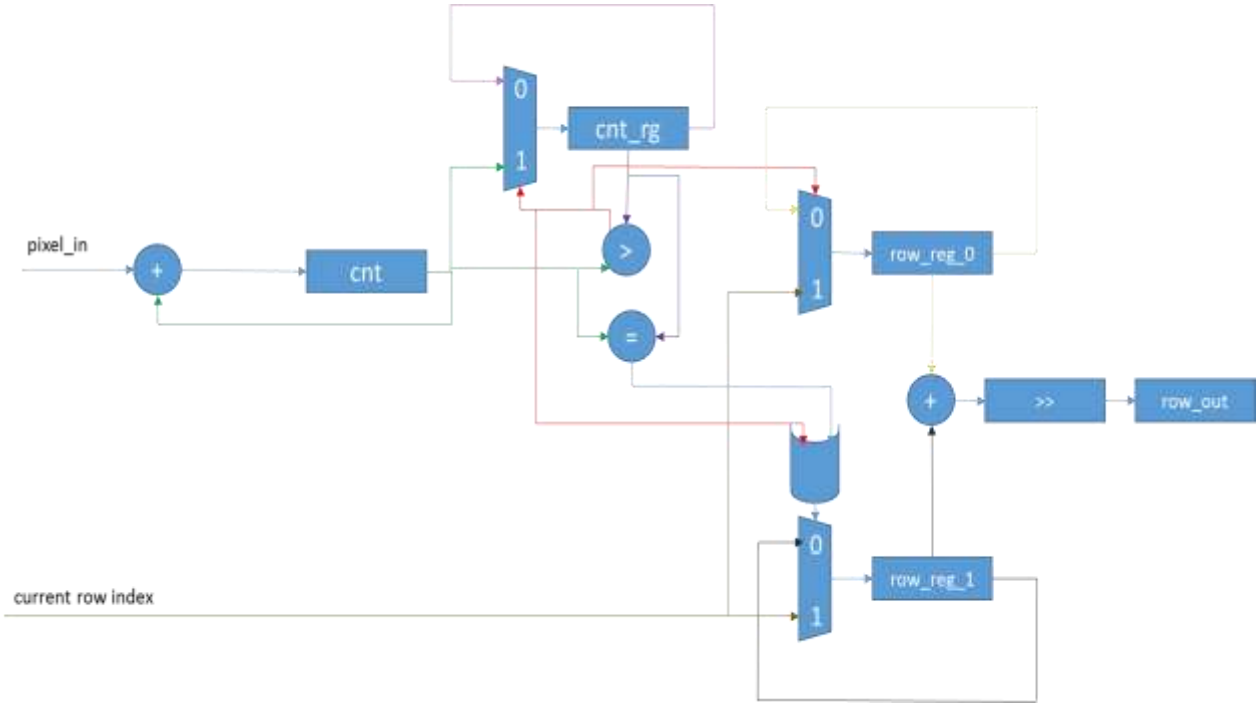


Figure 4.9: Digital circuit for determining the row with maximum sum

Another task of MRCC IP is to detect the center of white pixels at each row. Two registers are used for this purpose, namely *col_idx_reg_0* and *col_idx_reg_1*. The arriving signals corresponding to each row

are checked, and when a white pixel (a bit with value one) is detected for the first time in a row, *col_idx_reg_0* and *col_idx_reg_1* are set to the index of the current column. Furthermore, *col_idx_reg_1* is set to the index of the column of the latest detected white pixel in a row. The index of the column of interest can be calculated by adding *col_idx_reg_0* and *col_idx_reg_1* and shifting the result to the right by one bit (i.e. diving by 2). Figure 4.10 illustrates the corresponding circuit of this process, with flip-flops not being depicted for simplicity.

The complete VHDL code of MRCC can be found in *RaM_Vision_MRCC.vhd* in the attached zip file.

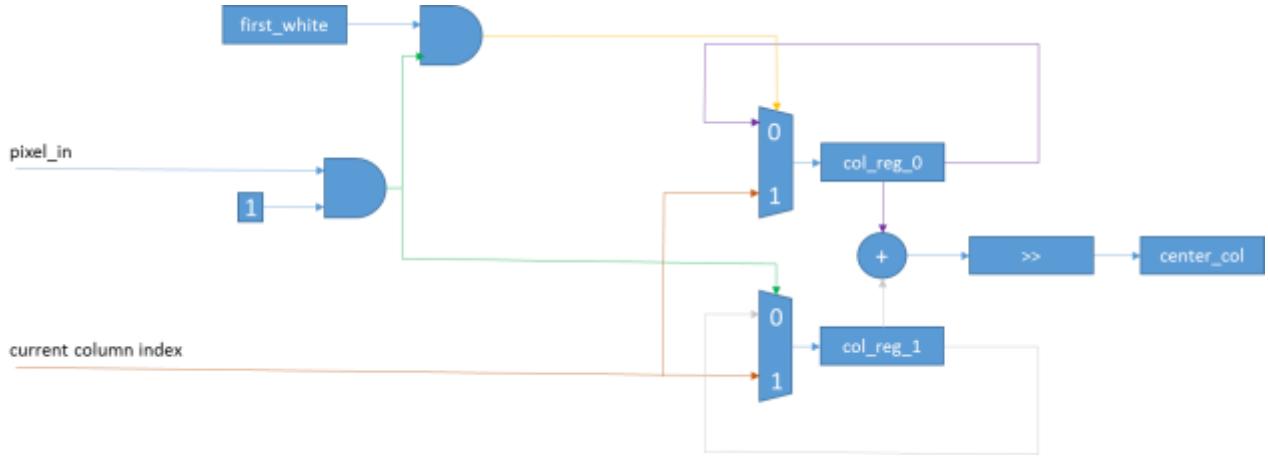


Figure 4.10: Digital circuit for determining the center column of each row

4.6 Erosion filter for noise reduction

As it discussed in section 4.4, by applying the MRCC algorithm to a noisy binary image, a pixel located far away from the shape of interest might be marked, especially if the noise is distributed around the rows with maximum sum, as illustrated in Figure 4.11 below.

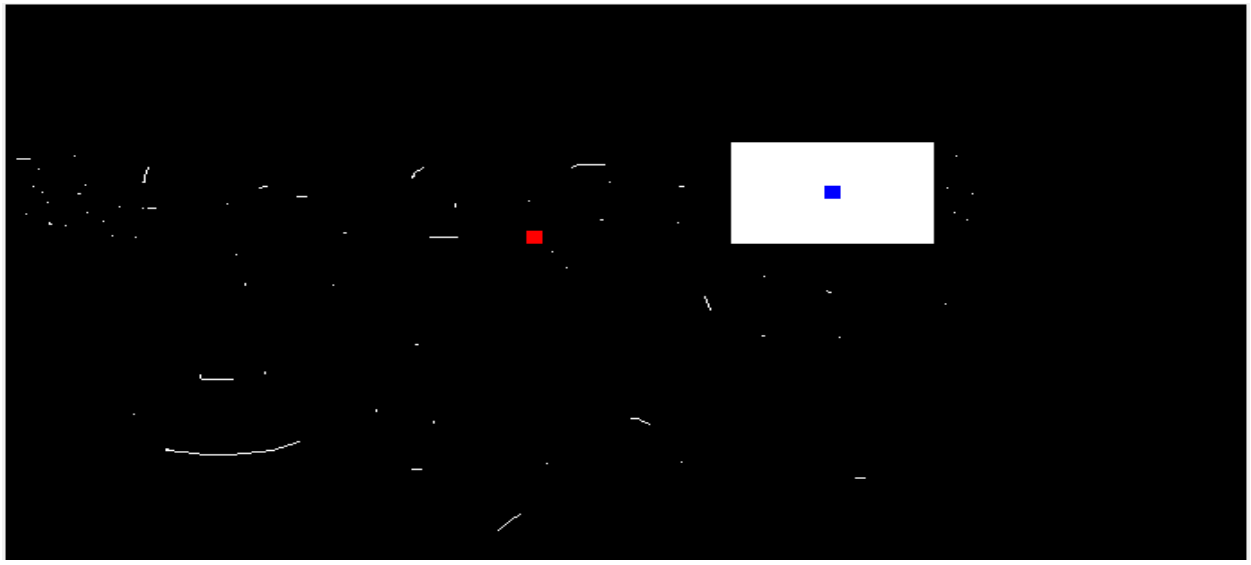


Figure 4.11: MRCC algorithm marked a wrong pixel in the noisy binary image (red rectangle)

In Figure 4.11, the blue rectangle depicts the location of the pixel which should be marked by MRCC algorithm without presence of any noise in the binary image. The red rectangle, on the other hand, shows the location of the pixel which is actually marked by MRCC algorithm in the noisy image. Thus, before applying the MRCC algorithm to a binary image, noise reduction is essential.

An erosion filter, which is also mentioned in section 4.2 for determining the center of a shape in a binary image, is a convenient tool for noise reduction. In general, this filter is classified into neighborhood operations class. It requires a structural element (SE) to determine the output pixels, which is described as a $n \times m$ window, and $n - 1$ row buffers are required for implementing it as previously discussed (check section 3.3.2). However, if the SE is defined as a horizontal line, n is equal to one, and therefore no row buffer is required. In this scenario, the latency of the filter is only proportional to the length of the horizontal line (m). Additionally, the value of each output pixel is determined by AND'ing the current input pixel with its horizontal neighborhood pixels, indicated by the SE. The length of the SE (m) is determined according to the noise distribution in the rows.

Figure 4.12 and Figure 4.13 show the same binary image as in Figure 4.11 but after applying an erosion filter with an SE of size 1×10 and 1×20, respectively. Furthermore, the red rectangle indicates the location of the marked pixel according to the MRCC algorithm in each figure.

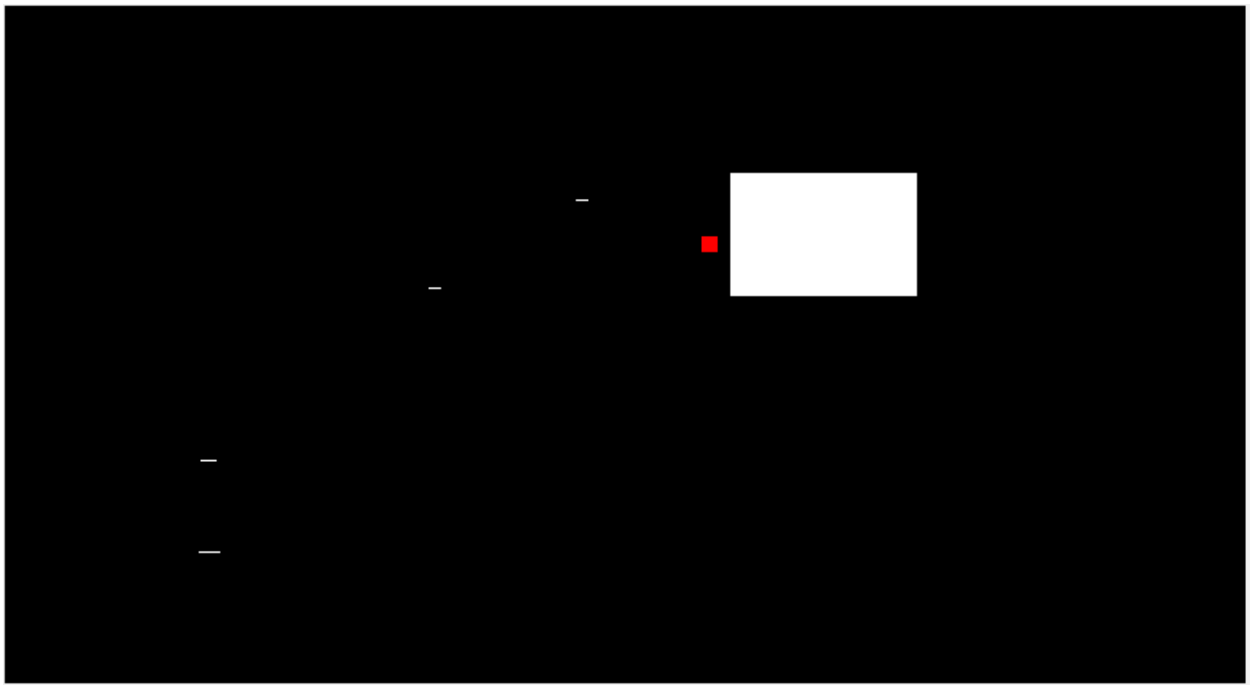


Figure 4.12: Marking the noisy binary image according to MRCC algorithm after noise reduction with an SE of size 1×10

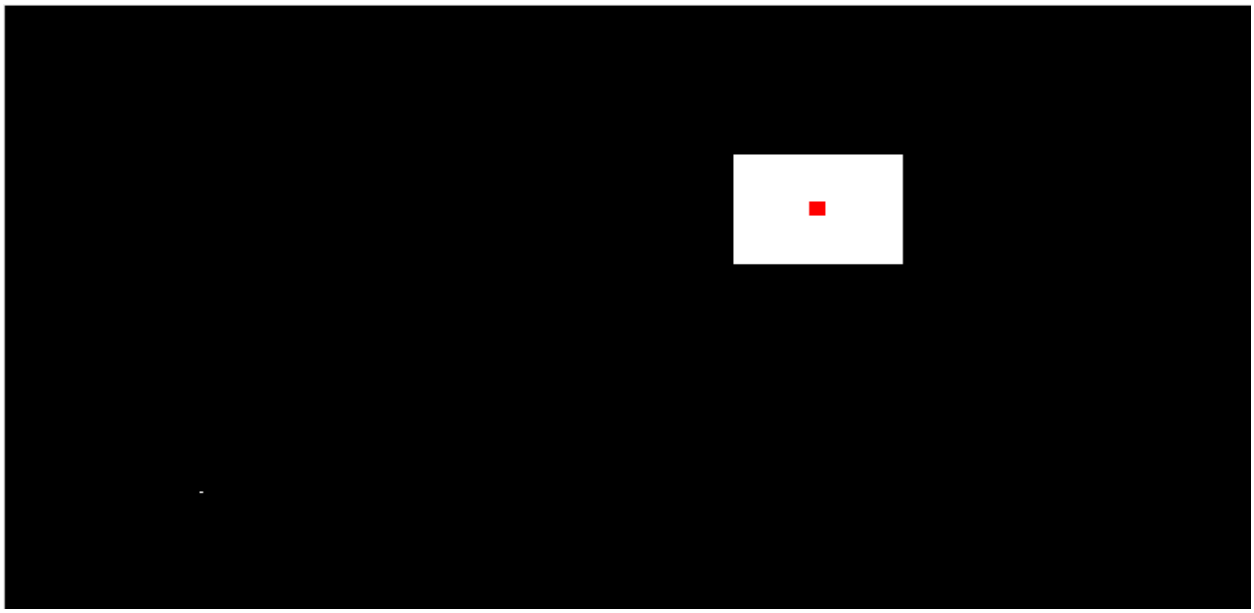


Figure 4.12: Marking the noisy binary image according to MRCC algorithm after noise reduction with an SE of size 1×20

It is simple to notice that the erosion filter directly affects the size of the shape of interest, as it erodes the shape in one orientation only: horizontally. Thus, the location of the marked pixel by MRCC algorithm may be slightly different than the location of the marked pixel in the original binary image without presence of any noise.

4.7 Erosion filter implementation on FPGA

Implementation of an IP for the erosion filter discussed in the previous section is straight forward. Initially, a register file with word size of 1-bit for each register block is considered. The depth of this register file is equal to the size of the SE (m) itself, and the initial value of each register block is set to zero. After a pixel is received by the IP, it is stored in the corresponding register block. Once more, the output is determined by AND'ing the value of all register blocks. The latency of producing an output pixel after receiving the corresponding input pixel is a single clock cycle. Figure 4.13 shows the architecture of the filter explained above, with a size three SE ($m = 3$).

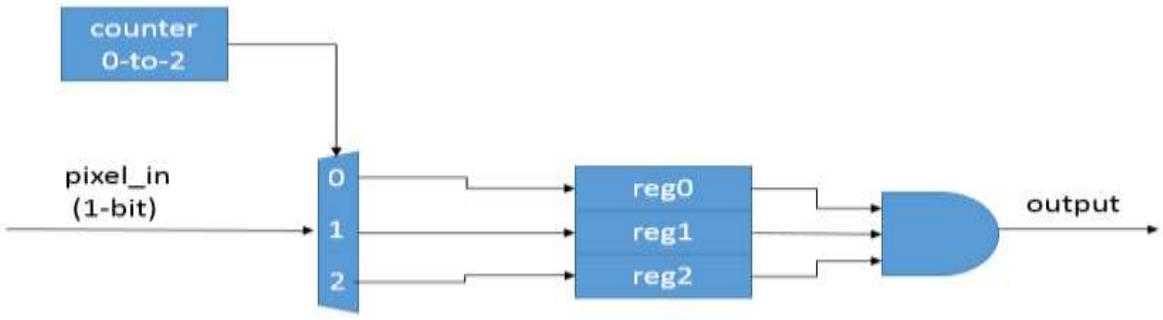


Figure 4.13: Architecture of an erosion filter for noise reduction with $m = 3$

4.8 Threshold-Marker parallel chains

Since color thresholding is classified as a point operation, the captured pixels can go through various *color_thresholding* IPs with different threshold levels at the same time. With this capability, multiple objects with different colors or an object with multiple colors as it shown in Figure 4.15, can be recognized.

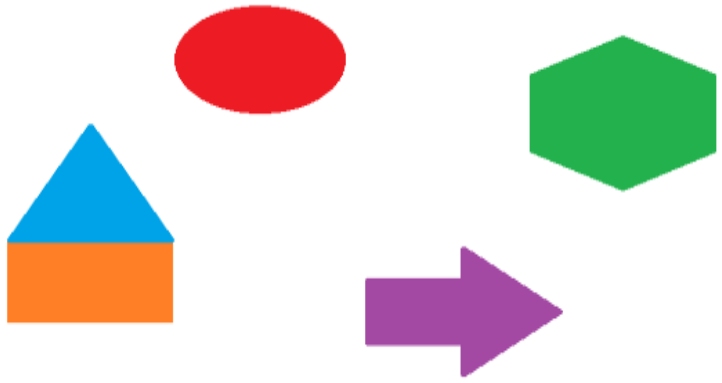


Figure 4.15: An image with objects with different colors

Furthermore, the output pixel of each *thresholding_IP* goes through the erosion filter for noise reduction. Finally, the MRCC IP marks a pixel on or around each shape. To accomplish this, a threshold-marker node is configured, which corresponds to a *color_thresholding* IP followed by an *erosion* and *MRCC* IP. Due to the fact a node can process each captured pixel independently, each captured pixel can go through each node at the same time to be checked for different threshold levels.

Multiple threshold-marker nodes in a system compose threshold-marker parallel chains. Figure 4.16 shows the structure of a *threshold-marker* node and a *threshold-marker* parallel chain.

The threshold levels, threshold operations and size of the SE of the erosion filter of each node can be set via a software program which uses the GPMC bus. For this purpose, the C++ class *RaMVision* is implemented, and an explanation on how to handle it and its methods, including configuring the parallel threshold-marker chain are described in Appendix B.

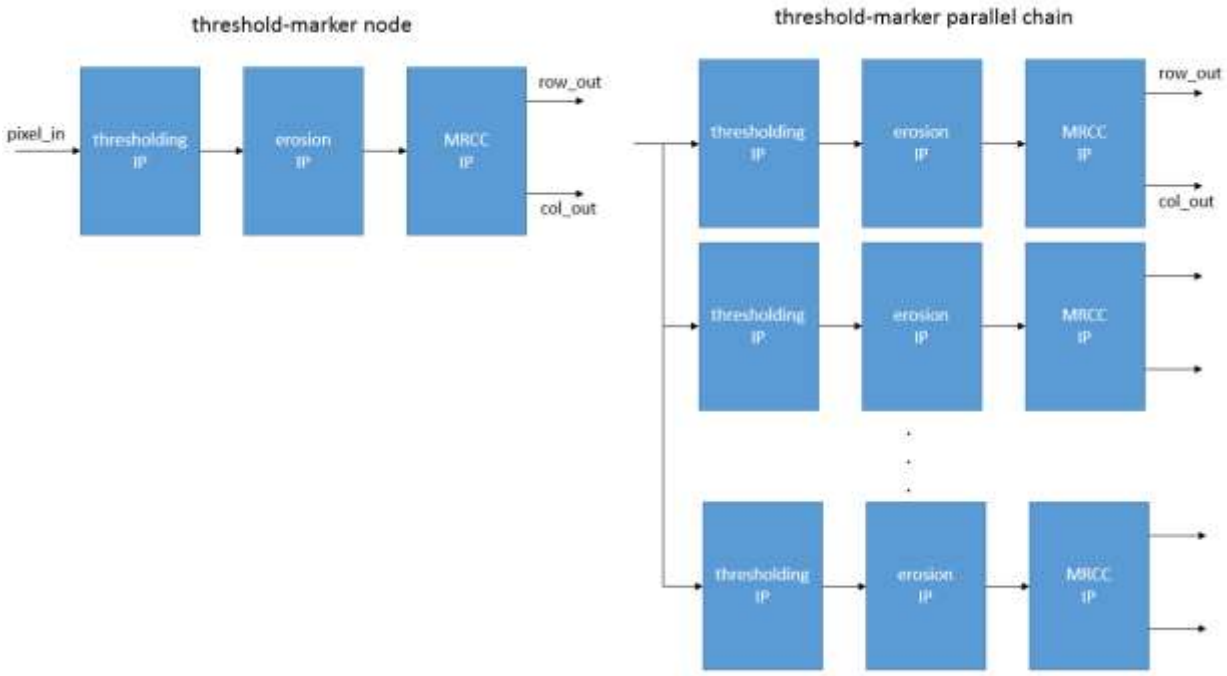


Figure 4.15: A threshold-marker node and a threshold-marker parallel chain

5. Results and verifications

In this chapter, the method to verify the functionality of the designed IPs with ModelSim is initially explored. Further, the procedure to test the designed IPs on RaMstix board without interfacing any camera with the board is discussed. Which is possible due the functionality of the IPs are independent of the interfaced camera. Finally, latency of the implemented IPs and the limitations of RaMstix board are examined.

5.1 Testing the designed IPs with ModelSim

ModelSim is a convenient tool to simulate and verify the functionality of the designed IPs. The IPs of this research must be fed with a complete frame in order to correctly compute the final result. For this reason, the behavior of OV7670 camera is described with a VHDL module and the IPs are tested with this module. This module is not synthesizable and is used only for testing purpose, however it generates signals according to the OV7670 datasheet [https://www.voti.nl/docs/OV7670.pdf]. Using this module in ModelSim environment also provides the possibility to test and verify the driver's IP for such camera.

In order to make this module able to generate pixels of a frame, a colorful image is initially loaded in MATLAB. Each color intensity of each pixel belonging to the image is then written in separate files, which are read with functions of VHDL's library *textio* in the OV7670 camera module, and the value of each pixel is formed and generated by the OV7670 camera module. Figure 5.1 shows the output signals that are generated by OV7670 module.

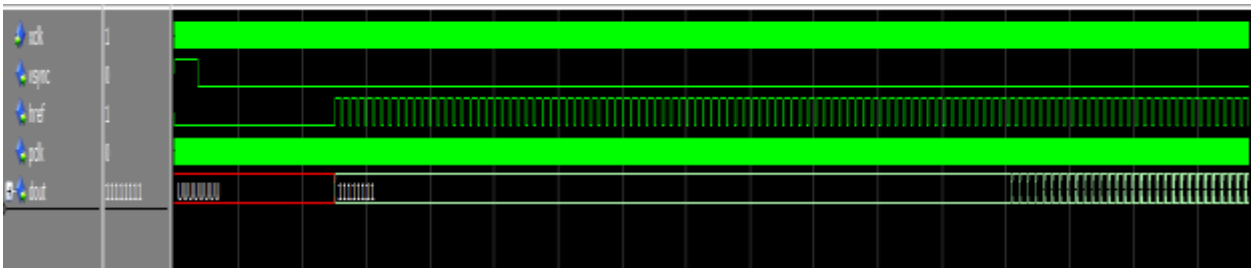


Figure 5.1: Generated signal by OV7670 camera module

In the test bench, the OV7670 camera module is connected to the *frame_capture* IP, that captures data from the module and composes the value of each pixel as a 16-bits output (RGB565 format). This output goes through the designed IPs to verify their functionality.

The test bench contains a *thresholding* and *MRCC* IP cascaded. In order to test the functionality of the former, *textio* library can be helpful. The output of *thresholding* IP can be written in a file through functions of the *textio* library. Such file can be read in MATLAB, displayed as a figure and compared with the thresholding result of the same image. The *MRCC* IP outputs two scalar numbers: row and column addresses. These values can be compared with the outputs of the MRCC function implemented in MATLAB which is applied to the same image. In Figure 5.2 the structure of the test bench and the test strategy is illustrated.

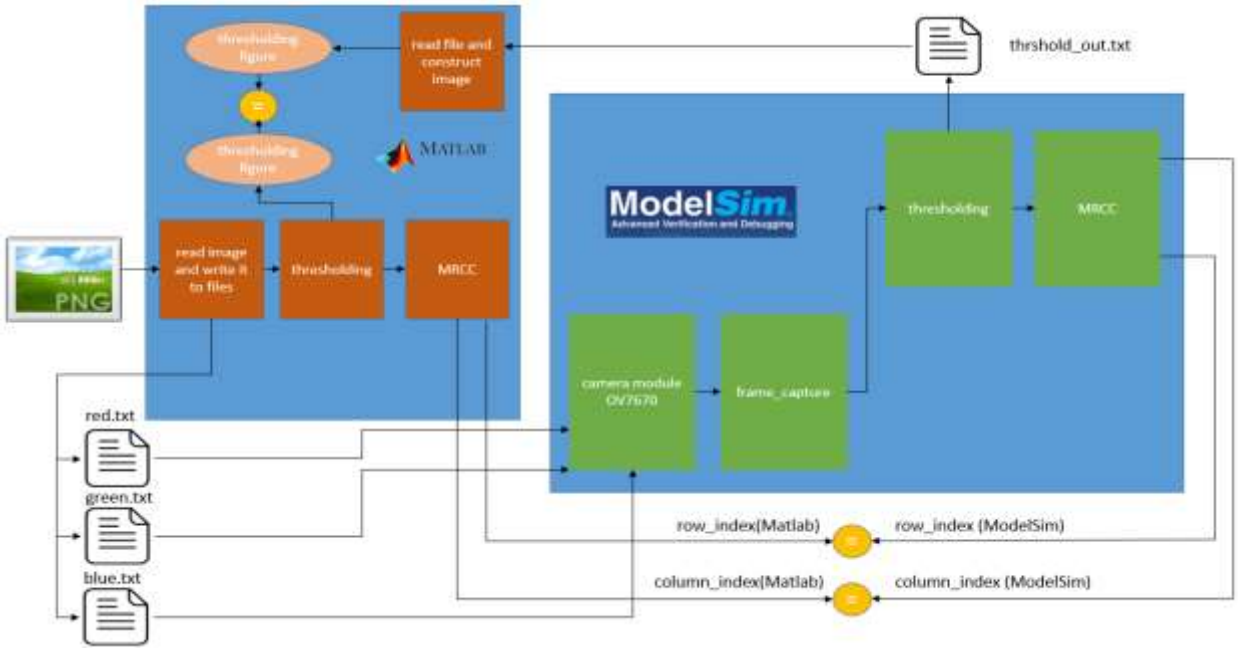


Figure 5.2: Test strategy with MATLAB and ModelSim

5.2 Testing the IPs without a camera on RaMstix

As discussed in section 1.2, this research focuses on the image processing part of visual servoing applications, independent of the camera module interfaced with the FPGA. Therefore, all designed IPs are able to work with any camera module as far as the required signals, such as row and column addresses, pixel value and pixel ready are provided by the camera's driver.

Due to this fact, the designed IPs can be tested even without the presence of a camera. A colorful image is read in MATLAB and the color intensities of each pixel are written in a separate file as described in section 5.1. These files are read with a C++ program which runs on the Gumstix, hosted on RaMstix board. Each pixel is then formed with RGB565 format and transferred to the FPGA through the GPMC bus. A memory block is intended in the FPGA to store the transferred pixels from the Gumstix, which contains 19200 slots, equal to the number of pixels of a frame with dimensions 120×160, and each slot has a size of 16-bits.

When the memory is fully loaded with pixels values of a frame, each pixel is read from the memory and forwarded to the IPs. Meanwhile, corresponding row and column addresses of each pixel is provided by an auxiliary IP called *address_generator*.

To test the functionality of the *thresholding* IP, its output is stored in a memory block with a size of 19200 slots, with each slot corresponding to 1-bit. The memory is read by a C++ program, which runs on the Gumstix, through the GPMC bus. Furthermore, the retrieved values are stored in a text file by the C++ program. Such file can then be loaded in MATLAB in order to be verified.

The outputs of "MRCC" IP are written to the GPMC registers which can be read by the C++ program and are shown in the terminal. Figure 5.3 depicts the schematic of the discussed test environment above.

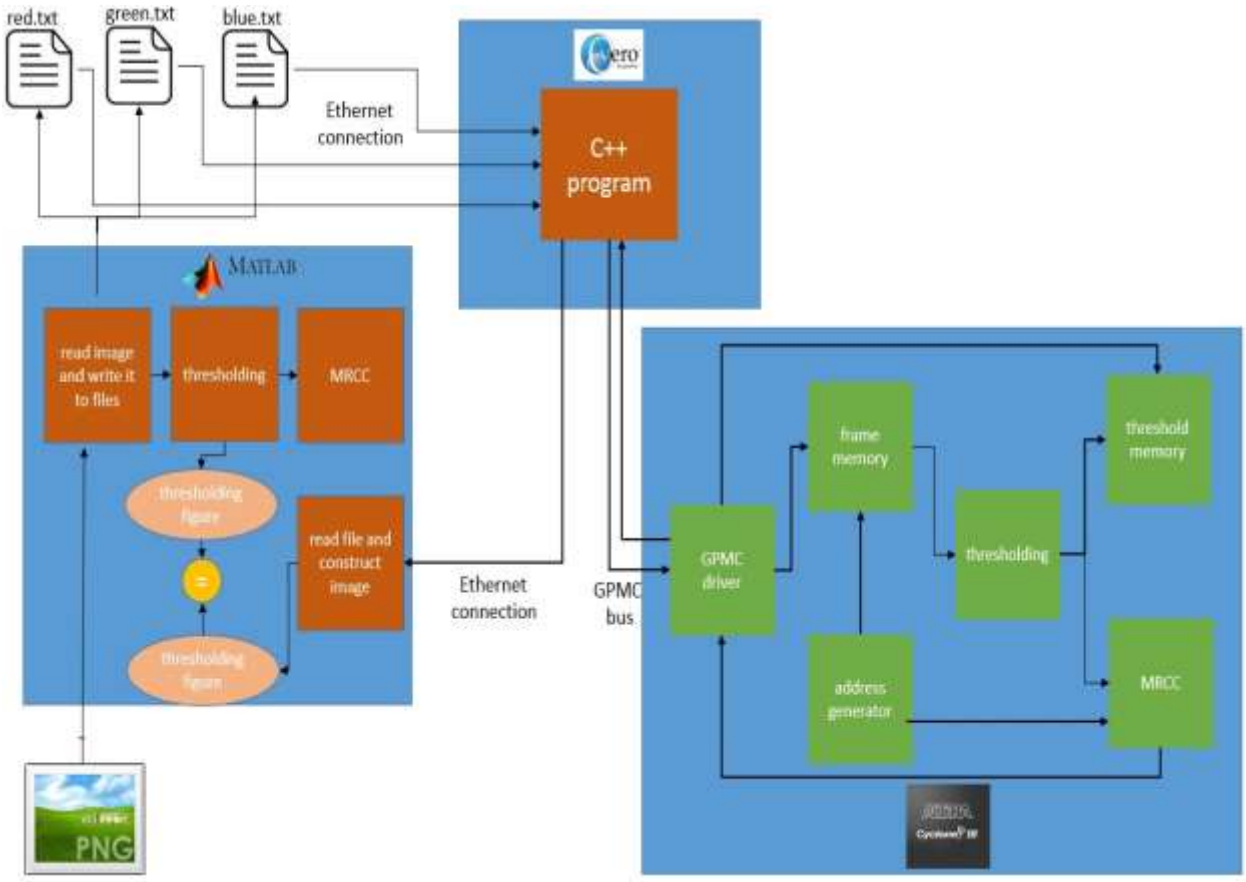


Figure 5.3: Testing IPs on RaMstix without the presence of a camera module

5.3 Latency and limitations of RaMstix

The time required for determining the position of a colorful shape in a frame by a threshold-marker node (τ) can be computed according to Equation 3.3, discussed in section 3.3.1. The firing time of both *thersholding* and *MRCC* IPs are one clock cycle. Assuming the value of each pixel is retrieved in eight clock cycles on average and considering a frame with dimensions 120×160, τ is calculated as follow:

$$\tau = (8 \times 120 \times 160) + 2 = \ 153602 \ clock \ cycles$$

In the calculation above it is assumed that the interfaced camera is an OV7670 camera module. This camera by default outputs frames in VGA format (480×640). Considering this format, a pixel with RGB565 format is retrieved in two clock cycles. However, in order to extract a pixel value in QQVGA format (120×160), eight clock cycles are needed on average. For full specifications, one should refer to the OV7670 datasheet [https://www.voti.nl/docs/OV7670.pdf].

RaMstix can provide a maximum clock of 50 MHz for the hosted FPGA. However, the filters operations are limited to the input clock frequency of the camera module. Assuming the input clock frequency of the camera is 25 MHz, the calculated τ above can be expressed as: $\tau = 153602 \times \frac{1}{25Mhz} = 6.1 \ ms$.

The FPGA resource usage for a module with 1 threshold-marker node and 20 threshold-marker nodes with maximum SE size of 100 for erosion filter in each node is given in table 5.1.

threshold-marker nodes	Total combinational functions	Dedicated logic registers	Total logic elements	Total memory bits
1	3620 (9%)	4603 (12%)	5773 (15 %)	0 (0%)
20	18300 (46%)	12750 (32%)	38650 (97%)	0 (0%)

Table 5.1: Resource usage of the FPGA for a module with 1 and 50 threshold-marker nodes

A high speed camera requires high input clock frequency to operate correctly, with the OV10625 camera module, for instance, being able to output 60 frame per seconds. Such device can be interfaced with RaMstix board, and image processing functions can be applied to the retrieved frames. Higher speed cameras which require more than 50 MHz clock as their input frequency cannot be interfaced with RaMstix. Although, if an input frequency greater than 50 MHz can be provided for a high speed camera (for instance 160 frame per seconds), the frames could be retrieved, but filters' arithmetic operations may not be completed within a single clock cycle. Due to this fact, frame buffering may be required, thus extra delay is introduced in the system.

6. Conclusions and recommendations

This chapter summarize the goals, approach and results this research context. The recommendations of follow-up of this research are likewise presented in this last chapter.

6.1 Conclusions

The goal of this project is to investigate the implementation of the image processing part of the visual servoing applications on FPGA with the ESL course requirements as an example. To this end, relevant image processing algorithms are studied and their feasibilities for hardware implementation examined. Algorithms are restructured in such a way to be suitable for hardware design. Higher level of pipelining and parallelism can be achieved than the software approach by implementing image processing algorithms on an FPGA.

Image processing algorithms are classified into two classes: point operations and neighborhood operations. In point operations, output pixel value only depends on the corresponding pixel value of the input image. Hardware implementation of the algorithms belonging to this class is not difficult, fast and memory efficient. In neighborhood operations, on the other hand, the output pixel value depends on the corresponding pixel value and its neighbors' pixels values of the input image. Hardware implementation in this case is arduous, slower and less memory efficient. However, more detailed information could be retrieved from a frame by applying these algorithms.

In this research several algorithms of point operations class are designed as IPs. The implemented IPs can work independently of the camera module interfaced with the FPGA as long as the required signals are provided by the camera's driver.

Implementing image processing algorithms on an FPGA can speed up the processing time, but it has its own limitations. In this application, the hosted FPGA on RaMstix board has limited resources (e.g. memory). Therefore, algorithms which requires storing multiple frames to derive relevant information is not suitable for this specific platform.

The hosted FPGA on RaMstix board is capable of running on a clock up to 50 MHz, but in order to execute image processing algorithms, the computation frequency is limited by the camera's input frequency: 25 MHz. High-speed cameras with greater input frequencies can be interfaced with an FPGA fed with a higher clock frequency, and in this case, frames can be captured at a higher rate, but the computation frequency is limited to the minimum period that the filter operations are performed on the FPGA.

A *color_thresholding* IP is designed to threshold the image by comparing each pixel's color intensity with the certain threshold level. After thresholding, the output pixels go through the *erosion* IP for noise reduction. Furthermore, the filtered pixels are forwarded to the *MRCC* IP. This IP marks a pixel on or around the shape of interest. The center pixel of the row which contains most white pixels, is marked by *MRCC* IP.

A *color_thresholding* IP followed by an *erosion* IP and *MRCC* IP is called threshold-marker node. Various threshold-marker nodes is a system is called threshold-marker parallel chain. The threshold level of each node of the chain can be set with different value and a captured pixel can be examined with different threshold level in parallel.

6.2 Recommendations

In this research, several image processing algorithms are studied and implemented, but more advance algorithms can be investigated for hardware implementation.

With respect to the ESL course, the recommendations are to provide a 60 fps camera and install it on top of servos. This enables students to have an extra option for image processing during the design exploration part of the course. Image processing can take place either on the Gumstix, the Gumstix hosted on RaMstix or the FPGA hosted on RaMstix.

The ESL manual should be updated with new information about the designed IPs for image processing in addition to a guide to configure them on the RaMstix board.

For research purposes, the recommendations are to study more advance image processing algorithms and reconstruct them for hardware implementation. For instance, reformulating the algorithm for finding the connected components in a binary image for hardware implementation is an interesting idea. As this algorithm is iterative and not convenient for hardware implementation, by slightly sacrificing the accuracy, it could be reformulated to be feasible for hardware implementation.

Several advanced methods such as either classic image processing algorithms or neural network approach can be investigated and implemented on an FPGA with more resources rather than the hosted one on RaMstix board.

A. Creating RGB files from MATLAB

A RGB image can be read in MATLAB with imread function. This function returns a three dimension matrix. Each dimension of the matrix contains one of the red, blue and green color intensity. Each dimension of the matrix is converted to RGB565 format and reshaped as a row array with reshape function. Furthermore, each reshaped dimension of the matrix is saved in a separate file. Figure A.1 shows the code list of the procedure discussed above.

```
im = imread('image.png');                    %read a RGB image
im565(:,:,1) = round(im(:,:,1) * (31/255)); %convert each dimension of
im565(:,:,2) = round(im(:,:,2) * (63/255)); %the matrix to RGB565 format
im565(:,:,3) = round(im(:,:,3) * (31/255));
red = reshape(im565(:,:,1)',120*160,1);     %reshape each dimension to
green = reshape(im565(:,:,2)',120*160,1);   % the row array
blue = reshape(im565(:,:,3)',120*160,1);
fileID1 = fopen('redChannel','w');          %create a file to store
fileID2 = fopen('blueChannel','w');         %the values of each dimension
fileID3 = fopen('greenChannel','w');        %of the mtrix
fprintf(fileID1,'%d\n',red);
fprintf(fileID2,'%d\n',blue);
fprintf(fileID3,'%d\n',green);
fclose(fileID1);
fclose(fileID2);
fclose(fileID3);
```

Figure A.1: MATLAB code to store the RGB values of an image in separate files with RGB565 format

B. IPs catalogue

RaM_Vision_Computer_Vision_pkg

This package contains constants, types and components that are used in IPs.

Name	Description
IS_GREATER	Color thresholding operation
IS_GREATER_EQUAL	Color thresholding operation
IS_LESS	Color thresholding operation
IS_LESS_EQUAL	Color thresholding operation
IS_EQUAL	Color thresholding operation

Table C.1: Constants of the RaM_Vision_Computer_Vision_pkg

Name	Description
COLOR	Contains of 8-bits red, green and blue color intensity

COLOR_OPERATION	Contains of 3-bit comparative operation for each color intensity which can be matched with one of the constants in table C.1

Table C.2: Types of the RaM_Vision_Computer_Vision_pkg

Name	Description
RaM_Vision_Color_Threshold	An IP for thresholding a color
RaM_Vision_RBG_Threshold	An IP for thresholding a RGB color
RaM_Vision_MRCC	An IP for marking a binary image
RaM_Vision_Erosion	An IP for reducing noise in a binary image
RaM_Vision_Threshold_Marker_Node	An IP which contains a thresholding IP followed by an erosion and a MRCC IPs

Table C.3: Components of the RaM_Vision_Computer_Vision_pkg

RaM_Vision_Color_Threshold

This IP receives pixels at its output and compare the value of the received pixel with the given threshold.

Name	I/O	Width	Description
ce	Input	1	Chip enable
clk	Input	1	Clock
reset	Input	1	Reset
reference_color	Input	8	Reference color for comparison
input_color	Input	8	Input pixel
comando	Input	3	Comparison operation
ready	Output	1	Output ready
threshold_out	Output	1	Threshold result

Table C.4: Ports of the RaM_Vision_Color_Threshold

RaM_Vision_RBG_Threshold

This IP consists of three RaM_Vision_Color_Threshold for thresholding each red, blue and green color intensities.

Name	I/O	Width	Description
ce	Input	1	Chip enable
clk	Input	1	Clock
reset	Input	1	Reset
reference_color	Input	16	Reference color for comparison
input_color	Input	16	Input pixel
comando	Input	9	Comparison operation
ready	Output	1	Output ready
pixel_out	Output	1	Threshold result

Table C.5: Ports of the RaM_Vision_RBG_Threshold

RaM_Vision_MRCC

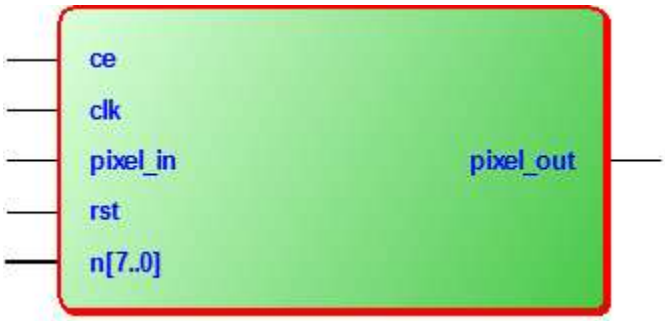
This IP marks a binary image according to the MRCC algorithm.

Name	I/O	Width	Description
ce	Input	1	Chip enable
clk	Input	1	Clock
rst	Input	1	Reset
din	Input	1	Input pixel
new_row	Input	1	New row detected
row_idx	Input	15	Row address of the input pixel
col_idx		15	Column address of the input pixel
row_idx_out	Output	16	Calculated row
col_idx_out	Output	16	Calculated column

Table C.4: Ports of the RaM_Vision_MRCC

RaM_Vision_Erosion

This IP reduce the rows noise in a binary image.

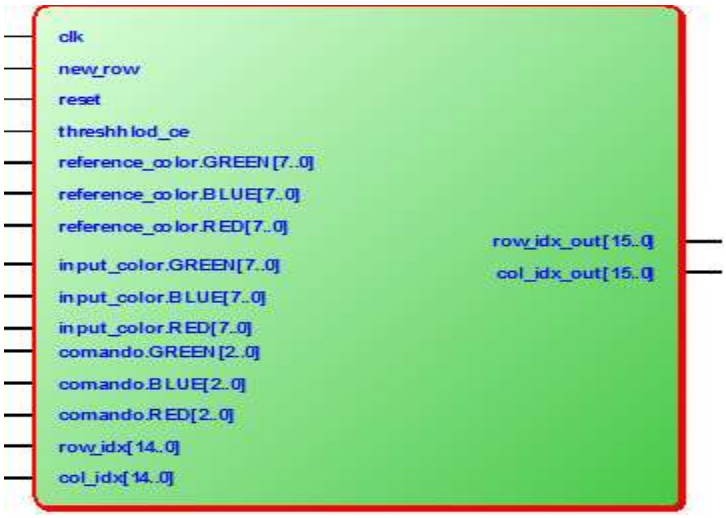


Name	I/O	Width	Description
ce	Input	1	Chip enable
clk	Input	1	Clock
rst	Input	1	Reset
pixel_in	Input	1	Input pixel
n	Input	8	Size of SE
pixel_out	Output	1	Pixel out

Table C.7: Ports of the RaM_Vision_erosion

RaM_Vision_Threshold_Marker_Node

This IP consists a thresholding IP followed by an erosion and MRCC IPs.



Name	I/O	Width	Description
threshold_ce	Input	1	Chip enable
clk	Input	1	Clock
reset	Input	1	Reset
reference_color	Input	16	Reference color for comparison
input_color	Input	16	Input pixel
comando	Input	9	Comparison operation
row_idx	Input	14	Row address of the input pixel
col_idx	Input	14	Column address of the input pixel
row_idx_out	Output	16	Calculated row
col_idx_out	Output	16	Calculated column

Table C.3: Ports of the RaM_Vision_Threshold_Marker_Node

Bibliography

Altera Corporation (2013), Quartus II version 13.1.
https://dl.altera.com/13.1/?edition=web

Gumstix Inc. (2015), Overo Firestorm-P COM.
https://store.gumstix.com/coms/overo-coms/overo-firestorm-p-com.html

OmniVision (2006), OV7670/OV7171 CMOS VGA (640x480) CameraChip with OmniPixel Technology.
http://www.eleparts.co.kr/data/design/product_file/Board/OV7670_CMOS.pdf

Schwirtz, M. (2014), RaMstix FPGA board.
https://www.ram.ewi.utwente.nl/ECSSoftware/RaMstix/docs/index.html

MATLAB computer vision system toolbox.
https://nl.mathworks.com/products/computer-vision.html

IPOL Journal. Image processing online
http://www.ipol.im/

Bailey, Donald Graeme (2011), Design for embedded image processing on FPGAs, New Zealand: John Wiley & Sons (Asia) Pte Ltd

Gonzalez, Rafael (2008), Richard Digital image processing (3rd edition), USA: Pearson Prentice Hall

Rastislav Lukac (2007), Color Image Processing Methods and Applications, Canada: Taylor & Francis Group

Sonka, Milan (2008), Image processing analysis, and machine vision, USA: Thomson learning, Thomson Corporation

Hani, Nicolai and Isler Volkan(2016), Visual servoing in Orchard setting

Rosadi, Aqwam and Arif Sunny(2016), Efficient implementation of mean formula for image processing using FPGA device

Zhang Shuyan and Zhu Juan(2016) Application of TEXTIO in the simulation of FPGA image processing algorithm