POLITECNICO DI TORINO

Collegio di Ingegneria Elettronica, Fisica e Telecomunicazioni

Corso di Laurea Magistrale
in Ingegneria Elettronica

Tesi di Laurea Magistrale

Hardware accelerator for a rear-view generation system

Relatore

prof. Guido Masera

Candidato

Marco Farina

Luglio 2018

Abstract

Advanced Driver Assistance Systems (ADAS) are widely used by car companies. The aim of these systems is to assist the driver when parking, during the journey and engendering an overall increase in car safety. Among these systems, the *real-time* ones suffer the problem of processing a large amount of data in a very short time. This thesis addresses this problem for an application that aims to generate a wide angle rear-view, trough an image stitching algorithm.
For this purpose Scale Invariant Feature Transform (SIFT), matching and Random Sample Consensus (RANSAC) from global homography have been used. The first finds, for each image, the *keypoints* and the local descriptors around each *keypoint* to extract local properties of the image. The matching algorithm finds the correspondences between the descriptors of the first image with respect to the ones of the second. Once the set of matched points is found, the transformation matrix, *homography*, is evaluated by exploiting the RANSAC algorithm that carries out the best fitting transformation matrix.
In order to achieve the desired performances an *ad-hoc* Application Specific Integrated Circuit (ASIC) implementation has been developed. Each module has been described in Very High Speed Integrated Circuits Hardware Description Language (VHDL) and tested by using *Modelsim*. The gate netlist has been extracted by using *synopsys*, therefore the area, power and timing data were carried out.
The results obtained prove the effectiveness of the system.

Keywords: ADAS, SIFT, RANSAC, image processing, real-time systems, image stitching.

Acknowledgements

First I would like to thank my supervisor in Sweden Prof. Ahmed Hemani for giving me the possibility to work on this project and let me join his research team.

I would also like to thank my Italian supervisor Prof. Guido Masera for being always available and interested in this thesis, despite he was not completely involved in the research.

I would also like to thank the PhD students Dimitris Stathis and Yu Yang for their suggestions, tips and for the cooperation we have enjoyed.

More in general, I would like to thank all the professors met during my academic studies. I have learned a lot from your teaching in these five years.

I would like to thank all my friends, my girlfriend and my family. You really helped me during these years.

Marco Farina

List of Figures

List of Tables

List of Acronyms and Abbreviations

ACC Adaptive Cruise Control.

ADAS Advanced Driver Assistance Systems.

AEB Advanced Emergency Breaking.

ASIC Application Specific Integrated Circuit.

ASM Algorithimic State Machine.

AU Arithmetic Unit.

BRIEF Robust Independent Elementary Features.

CNN Convolutional Neural Networks.

CU Control Unit.

DLT Direct Linear Transform.

DoG Difference-of-Gaussians.

DP Data Path.

DRAM Dynamic Random Access Memory.

FLANN Fast Library for Approximate Nearest Neighbours.

FPGA Field Programmable Gate Array.

FSM Finite State Machine.

HLS High Level Synthesis.

LFSR Linear Feedback Shift Register.

LHS Left-Hand Side.

LKA Lane Keep Assist.

LUT Look-Up Table.

MAC Multiply and Accumulate.

OFMP Output Feature Map Parallelism.

ORB Oriented Fast Rotated BRIEF.

PE Processing Element.

RANSAC Random Sample Consensus.

RF Register File.

RHS Right-Hand Side.

SIFT Scale Invariant Feature Transform.

SRAM Static Random Access Memory.

SURF Speed-Up Robust Features.

SVD Singular Value Decomposition.

VHDL Very High Speed Integrated Circuits Hardware Description Language.

Contents

Chapter 1

Introduction

Nowadays, ADAS, are widely used. Automotive companies are heavily working on developing new technologies for moving toward the automated driving. Advanced Emergency Breaking (AEB), Adaptive Cruise Control (ACC), Lane Keep Assist (LKA) are already mounted and used by almost the totality of the modern cars. The purpose of these systems is to assist the driver when parking, during the journey and engendering an overall increase in car safety for everyone on the road [1].
In order to ease the driver, recent researches are focused on providing a surround view which can help to park and reduce blind spots while traveling. It offers the driver the view of the exterior of the vehicle to aid in maneuvering the vehicle to park and to alert the driver of obstacles in its path that may not be immediately visible. A display on the vehicle's interior control panel shows the surround view, which is typically composed of four wide-angle cameras that are mounted in the wing mirrors, at the front and rear of the vehicle [2].

1.1 Goal

The aim of the project is to develop a system which stitches two different images taken by separated cameras placed on the back of a car and displays the final image that covers a wider angle, hence reducing the blind spots, by helping the driver finding obstacles.
Furthermore, the system can be mounted on vehicles and sold by car companies, increasing the quality and the safety of their products. Inexpert drivers will appreciate the help provided by the surround view

while parking, as well as expert drivers who can exploit the internal display for spot and avoid obstacles and be quieter while driving.
One of the main problems to be addressed in ADAS systems is the real-time processing of data since a large amount of data is to be handled due to the several systems provided by modern cars, such as ACC, AEB, LKA. Moreover, for this application, the higher is the camera resolution, the longer is the latency of the algorithm. In order to observe a clear final image, the design has to process a significant number of samples very quickly. An ad-hoc ASIC design is implemented to achieve these goals.

1.2 Research Question

One important parameter that should be taken into account, when dealing with stream processing and real-time applications like the one studied in this paper, is the human reaction time . The latter is on the order of a quarter of second, i.e. 250 milliseconds [3].
In view of the previous considerations, the system developed for this research should be able to reach at least 30 frames per seconds, in order to be mounted on a car, allowing a flowing scene on the internal display. Is the design able to achieve these performances in order to allow the system's development?

1.3 Structure of the thesis

This thesis proposes first, in Chapter 2, the literature studies developed in this research field, by focusing mainly on the algorithms exploited for the project implementation. Section 3 describes how it has been tried to address the problem mentioned before. All the hardware blocks implemented are discussed and some details are given related to the project philosophy and the choices done. Further, Chapter 4 provides all the results extracted from the synthesized blocks and the data analysis performed. In the end, Chapter 5 reports the thesis conclusions along with author deductions and opinions that come from the results exploration. The thesis ends with the list of the future works that will be done to enhance the system.

Chapter 2

Background

Studies on image alignment and stitching started in the mid-1990s for film photography applications [4]. Pixel-to-pixel dissimilarities were minimized for these applications, but researches moved on a different class of algorithms works by extracting a sparse set of features and then matching these to each other. Feature-based approaches have the advantage of being more robust against scene movement and are potentially faster.
Many algorithms have been developed for extracting features from an image, for instance SIFT, Speed-Up Robust Features (SURF), Robust Independent Elementary Features (BRIEF), Oriented Fast Rotated BRIEF (ORB) [5], [6], [7] or based on corner and edge detectors [8].

Although SIFT is slower then SURF, it extracts a more significant number of features for different scales and rotation [9]. This been said, this chapter will provide an introduction to the different algorithm used during the thesis.
Section 2.1 describes better the SIFT, which has been chosen for the system developed among the several methods mentioned before, in order to better introduce the reader understanding the future steps.
The definition of matching the keypoints that come from the two images is given in section 2.2, whereas a deeper explanation of the global homography and RANSAC is proposed in section 2.3.

2.1 SIFT

In *Distinctive Image Features from Scale-Invariant Keypoints* Lowe describes how the SIFT algorithm finds image features that have many properties that make them suitable for matching differing images of an object or scene [7]. Those features are invariant in scale changes and rotational changes and partially invariant in illumination and camera viewpoint.

The SIFT can be divided into 4 main steps:

1. **Scale-space extrema detection**: A series of blurred images, defined *scale-space*, is obtained by convolving the image with different kernels and resizing it in order to achieve scale invariance. Then, the extrema are detected looking into the DoG which are the result of the subtraction between two adjacent scales.

2. **Keypoints localization**: For each possible keypoint a more detailed model is used in order to remove bad keypoints, hence location and scale of each keypoint are determined.

3. **Orientation assignment**: One or more orientation assignment is found for each keypoint to achieve rotation invariance.

4. **Keypoint descriptor**: Around each keypoint a local descriptor is found, by increasing the reliability of the keypoint itself when the matching algorithm is applied.

Scale-space extrema detection Detecting locations that are invariant to scale change of the image can be accomplished by searching for stable features across all possible scales, using a continuous function of scale known as *scale- space* [10].
The *scale-space* of an image is defined as a function L(x,y, σ), that is produced from the convolution of a variable-scale Gaussian, G(x,y,σ) that from now on it will be called kernel, with an input image, I(x,y):

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y),$$

where * is the convolution operator and $G = \frac{1}{2\pi\sigma^2} e^{(-(x^2+y^2)/2\sigma^2)}$.

In order to detect stable keypoints the DoG function D(x,y, σ) is defined as the difference of two next scales separated by a constant multiplicative factor *k*:

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma)$$

σ is called scale factor and represents the variance of the Gaussian kernel. The function D provides a close approximation to the scale-normalized Laplacian of Gaussian, $\sigma^2\nabla^2 G$, which contains the factor σ^2 that is required for scale invariance. Maxima and minima of the function $\sigma^2\nabla^2 G$ produce most stable image features compared to other image functions, such as Harris corner function or Hessian function [11].

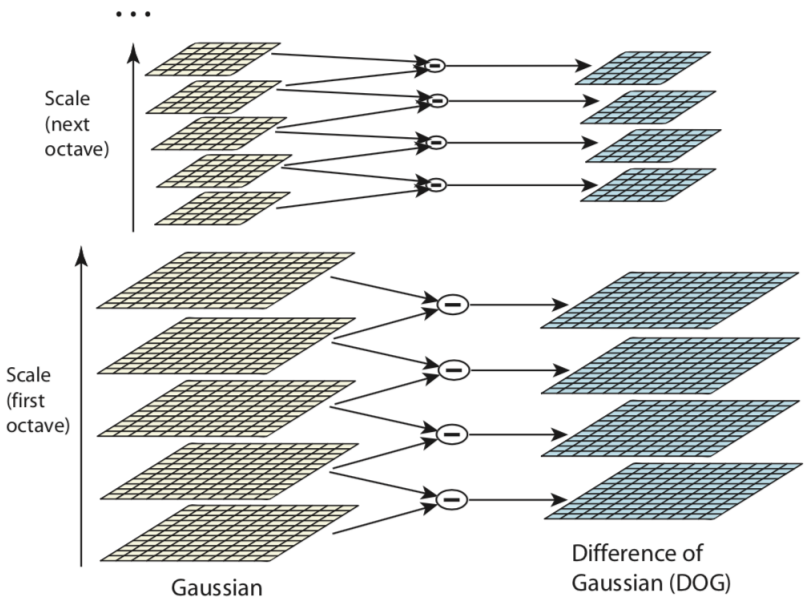


Figure 2.1: Scale space and Difference of Gaussian. *The figure has been extracted from the SIFT paper [7].*

The Gaussians function of Figure 2.1 are built by convolving the imagewith different kernels defined by a scale factor σ, obtaining an octave. The higher the scale the bigger the σ value, according to the formula:

$$\sigma = \sigma_0 2^{\frac{octave * levels + scale}{levels}}$$

with $k = 2^{1/levels}$. Moreover, *scale* and *octave* are, respectively, the index of the scale in each octave and the index of the octave itself. After an octave is completed, the image is scaled by a factor 2 and it is convolved again with different kernels, starting from the one which has the scale factor $2\sigma_0$.
For example, given 4 octaves, 4 levels and 5 scales, the sigma at the bottom of the scale-space is σ_0, while at the top of the first octave the scale factor is $\sigma = 2\sigma_0$, whereas at the top of the scale space is $16\sigma_0$.
Once the difference of Gaussian has been obtained, the *extrema* (possible keypoints), are detected by comparing each pixel in the DoG with its 8 neighbors and the 9 correspondent in the lower and upper scales, as described in Figure 2.2.

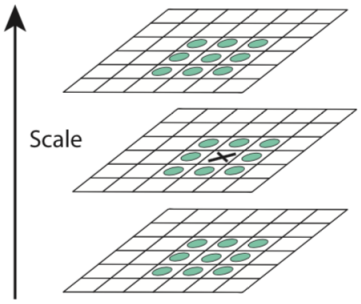


Figure 2.2: Extrema detection process. *The figure has been extracted from the SIFT paper [7].*

Keypoint localization Up till now the set of extrema has been obtained, which represents an approximation of the maxima/minimum, because in most cases this maximum value does not lie exactly on a pixel, but it is located in an intermediate position between pixels. The only way to carry out these sub-pixels' values is to create a mathematical approximation of the function D.
A Taylor expansion series is used for the DoG function and it turns out:

$$D(\mathbf{x}) = D + \frac{dD}{d\mathbf{x}^T}\mathbf{x} + \frac{1}{2}\mathbf{x}^\mathbf{T}\frac{d^2D}{d\mathbf{x}^2}\mathbf{x}$$

where D is represented by the values available that come from the subtraction of the two contiguous scales.
The extreme point of this equation can be easily calculated by differentiating it and equate to zero.

The location of the extremum is: $\hat{\mathbf{x}} = -\frac{d^2D}{d\mathbf{x}^2}^{-1}\frac{dD}{d\mathbf{x}}$.
$D(\hat{\mathbf{x}})$ is used for rejecting unstable extrema with low contrast. The values $|D(\hat{\mathbf{x}})| < threshold$ are discarded; $threshold = 0.02$ is suggested by the author.
Besides the low contrast points, one more type of extrema must be removed: the points for which the location along an edge is poorly determined are unstable to small amount of noise. When this happens the peak of the DoG will have large curvature across the edge but a small one in the perpendicular direction. The Hessian matrix's **H** eigenvalues α and β are proportional to the principal curvatures of D, where the matrix is:

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{yx} & D_{yy} \end{bmatrix}$$

By the definition of Trace and Determinant

$$Tr(\mathbf{H}) = D_{xx} + D_{yy} = \alpha + \beta,$$

$$Det(\mathbf{H}) = D_{xx}D_{yy} - (D_{xy})^2 = \alpha\beta$$

In order to check if the ratio between the principal curvatures is less then a certain threshold the value $\frac{Tr(\mathbf{H})^2}{Det(\mathbf{H})}$ is calculated.
After some computation $\frac{Tr(\mathbf{H})^2}{Det(\mathbf{H})} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(r+1)^2}{r}$, where r is the ratio between the two eigenvalues $r = \frac{\alpha}{\beta}$. The value $r_0 = 10$ is suggested by the author. If $\frac{Tr(\mathbf{H})^2}{Det(\mathbf{H})} < \frac{(r_0+1)^2}{r_0}$ the point is kept, otherwise it is discarded.

Orientation assignment For each keypoint, an orientation is assigned based on the properties of the image nearby the keypoint itself. In this way the local descriptor, which will be described later, can be represented related to the keypoint orientation, therefore achieving rotation invariance.
The idea is to collect, around each keypoint, gradients and magnitude of each sample, according to the following equations:

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$$

$$\theta(x,y) = tan^{-1}((L(x,y+1) - L(x,y-1))/(L(x+1,y) - L(x-1,y)))$$

An orientation histogram is obtained, with 36 bins covering 360 degrees range of orientations. Each sample added to the histogram is weighted by a Gaussian circular window with σ equals to 1.5 times the scale of the keypoint. The histogram peak is the orientation assigned to the keypoint. Multiple orientations could be assigned to the same keypoint if any other local peak is at least 80 % of the highest peak. Figure 2.3 shows on the LHS the orientation histogram built at this point, whereas the right-hand side shows the local keypoint descriptor which is described in the next section.

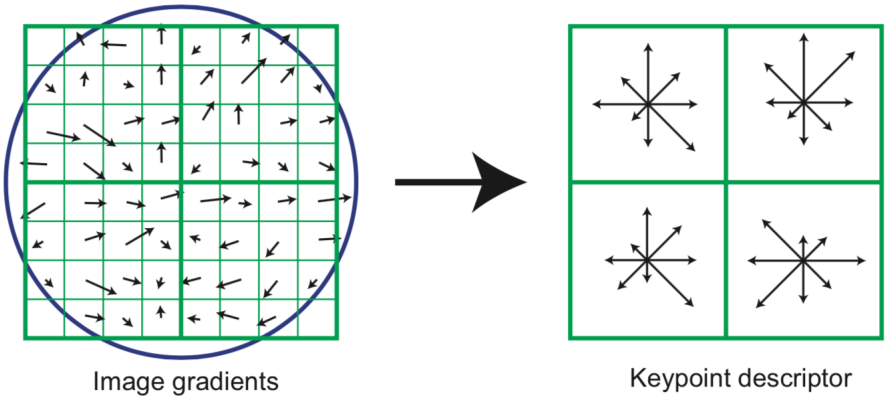


Figure 2.3: Orientation histogram (LHS) and keypoint descriptors (RHS). *The picture has been extracted from the Lowe's paper [7].*

Keypoint descriptor The last step of the SIFT algorithm consists in finding local keypoint descriptors which are highly distinctive, by removing other kinds of change, such as illumination or 3D viewpoint. The keypoint descriptor shown on the RHS of Figure 2.3 is formed by using the 16x16 values of the orientation histogram. Each sample is assigned to one of the eight possible directions defined by the keypoint descriptor, by taking the difference of the sample with respect to the keypoint orientation found at the previous step. When a 4x4 piece has been evaluated in the orientation histogram, one sample in the keypoint descriptor array has been completed, and the next one is processed until the whole 4x4 array keypoint descriptor is completed. According to these values, which are suggested in the SIFT paper, each keypoint is described, locally, by a 4x4x8=128 features vector. Finally, the feature vector is normalized to its length. This results in contrast

invariance since a constant factor would be removed by the normalization, whereas the brightness invariance is given by the magnitude evaluation, that is performed with subtractions.

2.2 Matching

Image matching is the process that finds the correspondences between a set of features, in general points, that are related to different images. This task is part of several computer applications, image processing and recognition. In the case proposed, the matching algorithm works on a set of 128 array features, extracted by the SIFT algorithm.
Suppose that p is a point detected in an image associated with a descriptor

$$\phi(p) = \{\phi_k(\mathbf{P}) \mid k = 1, 2, ..., K\}$$

where, for all K, the feature vector provided is:

$$\phi_k = (f^k_{1p}, f^k_{2p}, ..., f^k_{n_k p})$$

The aim is to find the best correspondence q in the set of N points in the other image $Q = q_1, q_2, ..., q_N$, by comparing the feature vector $\phi(p)$ with the points in the set Q [12]. In order to make it, the Euclidean distance $d_k(p, q)$ has been evaluated :

$$d_k(p,q) = \sqrt{(\phi_k(p) - \phi_k(q))^2} = \sqrt{(f^k_{1p} - f^k_{1q})^2 + (f^k_{2p} - f^k_{2q})^2 + ... + (f^k_{n_k p} - f^k_{n_k q})^2}$$

Considered two points (p, q), a match is accepted if:

1. p is the best match for q in relation to all the other first image's points
2. q is the best match for p in relation to all the other second image's points

The nearest neighbor algorithm is used for this purpose. Moreover, for suppressing false matching, it is evaluated both the nearest neighbor and the second-nearest neighbor (the one which has the distance d_k higher than the nearest one, but lower than the rest); if the ratio between these two distances is greater than 0.8 the point is discarded.
In any case, since finding nearest neighbors is a very hard task to be

achieved with classical algorithms like kd-tree or with "time bound" approximation search, two more efficient matching algorithms have been found, especially for high dimensional data: the randomized k-d forest and the fast library for approximate nearest neighbors (FLANN) [13].

2.3 Homography and RANSAC

The homogeneous representation of a 2D point (x, y) that lies on the projective plane P^2 can be represented by a 3D vector (a, b, c), where $x = \frac{a}{c}$, whereas $y = \frac{b}{c}$ [14].
According to Hartley and Zissermalm, an homography, also called pojectivity, can be defined as:
An invertible mapping h from P^2 to itself such that 3 points x1, x2, x3 lie on the same line if and only if h(x_1), h(x_2) and h(x_3) do.
Matematically, this can be formulated as:
A mapping h: $P^2 \rightarrow P^2$ is a projectivity if and only if there exists a non-singular 3x3 matrix H such that for any point in P^2 represented by a vector $\boldsymbol{x}$ it is true that $h(\boldsymbol{x}) = H\boldsymbol{x}$.
The transformation matrix is a 3x3 matrix represented by:

$$h = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix}$$

In the project developed, the purpose is to find this matrix h that maps the points in the second image with respect to the first one.
In order to find the homography matrix h, that best fit the set of matched points, the RANSAC algorithm is used.
Given the matched points of the two different images $(u, v, 1) \longleftrightarrow (u', v', 1)$, the system can be written:

$$\begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix}$$

The system can be manipulated and it turns out:

$$\begin{bmatrix} 0 & 0 & 0 & -u & -v & -1 & v'u & v'v & v' \\ u & v & 1 & 0 & 0 & 0 & -u'u & -u'v & -u' \\ -v'u & -v'v & -v' & u'u & u'v & u' & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \\ h_9 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \Leftrightarrow A\mathbf{h} = \mathbf{0}$$

It is possible to observe that the third row in A is a linear combination of the first and the second row $row3 = -u'row1 - v'row2$, hence each point correspondence contributes with 2 equation in the 9 unknown parameters.
Since **h** is homogeneous, the matrix A only need to have rank 8 in order to determine **h**. Assuming that there are not 3 collinear points, 4 points are sufficient to determine the vector.
The system becomes:

$$\begin{bmatrix} 0 & 0 & 0 & -u_1 & -v_1 & -1 & v'_1u_1 & v'_1v_1 & v'_1 \\ u_1 & v_1 & 1 & 0 & 0 & 0 & -u'_1u_1 & -u'_1v_1 & -u'_1 \\ 0 & 0 & 0 & -u_2 & -v_2 & -1 & v'_2u_2 & v'_2v_2 & v'_2 \\ u_2 & v_2 & 1 & 0 & 0 & 0 & -u'_2 & -u'_2v_2 & -u'_2 \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \\ h_9 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ . \\ . \\ . \end{bmatrix}$$

The problem is translated in an eigenvalues problem, and it is possible to calculate the non-trivial solution by using the Singular Value Decomposition (SVD). The last column of the **V** matrix (right singular vector without a singular value), that comes out from the SVD algorithm, is the homography evaluated.
RANSAC consists, then, in iterates for a certain number of times by picking up randomly 4 points' pairs each iteration, evaluate the homography matrix and compute the number of *Inliers* by seeing how the matrix fits all the points in the matched points set [15].

The *Inliers* are defined as the point correspondences, in the set of the matched points, that have a distance lower than a chosen threshold, with respect to the transformation matrix evaluated by the homography.
Algorithmically, the operations to be performed are represented in Algorithm 1.

Algorithm 1: Homography estimation using RANSAC

Input : Set of matched Points
Output: Global homography transformation matrix

while *iteration* < N **do**
  Pick up, randomly, 4 correspondences $C = x \longleftrightarrow x'$, from the whole set of matched points.
  SVD(C) $\longrightarrow$ carry out the matrix h, evaluated from the 4 correspondences C.
  $Inliers \longleftarrow$ dist(hx, x') < threshold.
  if $Inliers > Inliers\_max$ **then**
    $iteration \leftarrow iteration + 1$
    $Inliers\_max \leftarrow Inliers$
    $SetmaxInliers \leftarrow SetInliers$
  else
    $iteration \leftarrow iteration + 1$
  end
end
Evaluate the final matrix $h \leftarrow$ **SVD**($SetmaxInliers$)

2.3.1 SVD for eigenvalues decomposition

Given a *mxn* matrix, if σ is a non negative scalar, it is a singular value of the matrix A and u and v are respectively left and right singular vectors, if and only if:

$$A\mathbf{v} = \sigma\mathbf{u} \; and \; A\mathbf{u} = \sigma\mathbf{v}$$

The matrix product:

$$USV^*$$

is a singular value decomposition of the matrix A [16], if:

- U and V have orthonormal columns.
- D has nonnegative elements on its diagonals, 0 otherwise.
- $A = UDV^*$

There are multiple ways for solving the SVD. One is to use the Householder reduction to bidiagonal form, as described in the Algorithm 2:

Algorithm 2: SVD by exploiting Householder reduction

Input : tol, loopmax, m, n, A, where A is $m\ x\ n$.
Output: U, S, V such that the matrix have the properties mentioned before.

$U \leftarrow I_m$
$S \leftarrow A^T$
$V \leftarrow I_n$
$Error \leftarrow value$, such that $value > tol$
while $Error > tol\ \&\ loopcnt < loopmax$ **do**
    $Q, S \leftarrow QRfactorization(S^T)$
    $U \leftarrow U * Q$
    $Q, S \leftarrow QRfactorization(S^T)$
    $V \leftarrow V * Q$
    $E \leftarrow norm(triu(S))$
    $F \leftarrow diag(S)$
    if $F = 0$ **then**
        $F \leftarrow 1$
    end
    $Err \leftarrow E/F$
    $loopcnt \leftarrow loopcnt + 1$
end
$DS \leftarrow diag(S)$
$S \leftarrow 0$
for $i = 1 : length(DS)$ **do**
    $S(i, i) \leftarrow abs(DS(i))$
    if $DS(i) < 0$ **then**
        $U(all, i) \leftarrow -U(all, i)$
    end
end

The operations *triu* and *diag* extract, respectively, from the input matrix, the values of the upper triangular matrix and the diagonal.
In addition, the *abs* and the *norm* function evaluate, respectively, the absolute value of the input and the quadratic norm of the vector.

2.3.2 QR factorization for SVD

Let A be an mxn matrix with full column rank. It is defined QR factorization of A the decomposition $A = QR$, such that Q is an mxn orthogonal matrix, while R is an mxn upper triangular. Different ways are known for evaluate it, for instance using Householder matrices, using Jacobi Rotations or Gram-Schmidt orthogonalization [17]. A pseudocode that summarize the QR factorization using Householder reductions is reported in algorithm 3.

Algorithm 3: QR transofrmation of a matrix A

Input : m, n, A, where A is $m\ x\ n$.
Output: Q and R such that Q is an $m\ x\ n$ orthogonal matrix, while R is an $m\ x\ n$ upper triangular.

$R \leftarrow A$.
$Q \leftarrow I_3$
for $k = 1, ..., m-1$ **do**
    $x(i) \leftarrow 0$ for each element i of the size m array.
    $x(i) \leftarrow R(i,k)$ for $i = k, k+1, .., m$
    $g \leftarrow norm(x)$
    $v(i) \leftarrow x(i)$, for $i = 1, .., m$, with $i \neq k$.
    $v(k) \leftarrow x(k) + g$
    $s \leftarrow norm(v)$
    if $s \neq 0$ **then**
        $w \leftarrow v/s$
        $u \leftarrow 2R^T w$
        $R \leftarrow R - wu^T$
        $Q \leftarrow Q - 2Qww^T$
    end
end

2.4 Related Works

A similar algorithm was developed by Vishnu-Sharan at University of Dehli, India. The proposed design used a Xilinix Field Programmable Gate Array (FPGA), exploiting it through an ARM microprocessor. The whole algorithm has been developed in C and just the RANSAC part was accelerated trough an High Level Synthesis (HLS). At the end of the project very long latency come out, especially for the global homography estimation and the scale-space construction [18].
The results obtained are not enough to guarantee the proper functionality of the system.

Chapter 3

Methods

The system developed is designed by using VHDL and tested by using the Modelsim/Questasim's functionalities.
The project has been divided in several modules in order to handle the big structure of the design.
A 256x256 pixels camera resolution has been used and the number of scales and octaves processed are, respectively, 3 and 4. Although these values are different from the SIFT author, multiple checks have been done on MATLAB for testing the capability to detect features. Moreover more scales means more storage space and also higher latency, since the number of pixels to be processed would increase drastically, therefore this choice has been done as a trade off between robustness of keypoints and algorithm latency. Other parameters, related to different modules of the design, will be discussed during the report.
This chapter provides an in-deep presentation of the modules developed to realize the system. First, Section 3.1 will explain how the module that create the *scale-space*[1] has been developed.
Trough this section, first in Subsection 3.1.1 the details on the 2D convolution module are given. Subsection 3.1.2 focuses the attention on the DRAM controller that handle the communication between the DRAM and Static Random Access Memory (SRAM). Finally, Subsection 3.1.3 shows, shortly, how the DoG has been implemented.

[1]From now on the term *scale-space* can both be used for the set of function G and for the set of function D.

3.1 Scale-space construction

The pixels values are supposed to be stored inside the DRAM since the number of pixels' data and kernel values is too large for a SRAM due to the cell size. A memory hierarchy has been created: the $256x256$ has been divided in smaller pieces of dimension mxn, hence the local operations have been done on the pixels' values previously stored in the SRAM, by avoiding long access time, heavily speeding up the process. Since the accesses to the DRAM suffers of long latency and they are power consuming, due to the many operations that has to be done, data locality technique has been exploited to reduce them.
Each piece of image being processed at time has been kept fixed, while the kernels were changed in order to compute all the different scales, by exploiting the *clustered* OFMP technique, based on the data locality, as explained in [19] and depicted on the right hand side of Figure 3.1. Different pieces of the image are given to Processing Element (PE)s (parallelism for different pixels) as well as the multiple kernels (parallelism along scales).

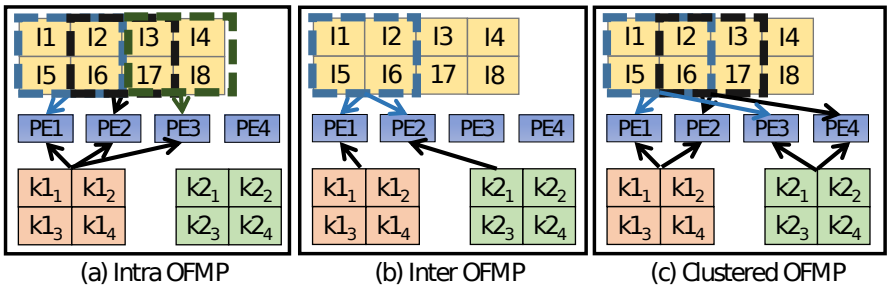


Figure 3.1: The figure shows different ways of manage the image and kernels data that feed the PE. In order to reduce as much as possible the DRAM accesses, the clustered OFMP -c)-has been implemented. The figure has been extracted been extracted from [19].

3.1.1 2D convolution

In order to create the Gaussians functions L, that form the scale space, the 2D convolution module has been implemented.
By definition of 2D convolution, the output of an matrix $I[x, y]$ is still

a matrix $L[x, y]$, according to:

$$L[x, y] = I[x, y] * k[x, y] = \sum_{u=-\infty}^{\infty} \sum_{v=-\infty}^{\infty} I[u, v]k[x - u, y - v]$$

In this way the number of operations to be performed, for a KxK kernel and an XxY image, is K^2XY.
However, the convolution style can be changed if the kernel is separable as depicted in Figure 3.2 [20].
In this way the number of operation is reduced to $2K\ XY$ and the necessary storage space for kernels is strongly reduced. Since the Gaussian kernel is also symmetric, only K values have to be stored and used, properly, both as $Kx1$ and $1xK$ filter, as shown in Figure 3.2.

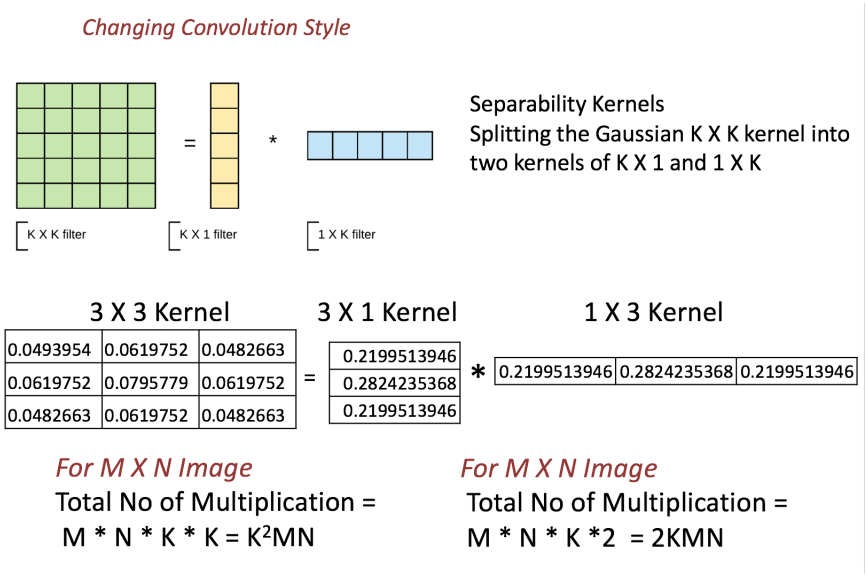
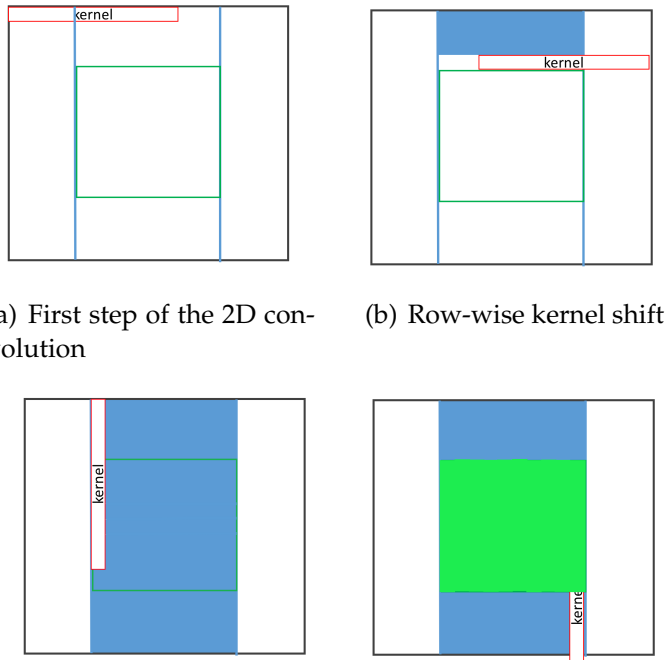


Figure 3.2: Kernel's separability and 2D convolution

The design has been created, by following the previous mentioned convolution style. The "array" kernel is shifted first row-wise, than column-wise in order to process a certain image block.
Figure 3.3 clarifies the system behavior.
A memory hierarchy has been created to speed up the process. The image is divided in pieces: the SRAM is filled by fetching from the DRAM all the pixels within the black square, as depicted in Figure 3.3 because they are necessary for the blue and, in turn, the green area. Later on the handling of images pieces will be discussed.
For the sake of clarity the 2D convolutions steps are listed below:



(a) First step of the 2D convolution

(b) Row-wise kernel shift

(c) Column-wise operations

(d) 2D convolution completed for the green area

Figure 3.3: The figure shows how the 2D convolution has been implemented by exploiting the kernel's separability. The blue area has to be processed in order to find the output correspondent to the green one.

1. The kernel is centered on the pixel to be evaluated.
2. The Multiply and Accumulate (MAC) operations are properly performed.
3. The kernel is shifted on the next pixel in the row (or shifted by n_pixels[2] if more then one pixel is processed at the same time)
4. The steps 1-3 are repeated for all the pixels in a row
5. The steps 1-4 are repeated for all the rows
6. The kernel is transposed, centered in the upper corner of the green region. If multiple pixels are processed the same transposed kernel is centered on multiple contiguous pixels.

[2]n_pixels is a parameter used inside the design for specifying the parallelism of each L(σ).

7. The steps 2-5 are repeated for the columns.

As already mentioned, in the above description, some parameters have been used in order to let the design be as generic as possible.
For this module, in particular, the parameters n_scales and n_pixels have been used, respectively for determining the parallelism's grade along the different scales and along the pixels of the same scale. More precisely:

- n_pixels determines the number of the output pixels evaluated at the same time in a certain scale, e.g. $n\_pixels = 4$ means $L[j,k], L[j+1,k], L[j+2,k], L[j+3,k]$ are evaluated at the same time.

- n_scales determines the number of scales processed at the same time. E.g.: $n\_scales = 2$, $n\_pixels = 2$ means that $L[j,k,\sigma_1]$, $L[j+1,k,\sigma_1]$ and $L[j,k,\sigma_2]$, $L[j+1,k,\sigma_2]$ are carried out at the same time. Figure 3.6 shows what happens when a new row is being processed and multiple pixels, in this case 2, are evaluated at time.
  More details about this topic are given later in the paper.

Assuming, for instance, $n\_scales = 2$ and $n\_pixels = 2$, a sketch of the data path is reported in Figure 3.4:

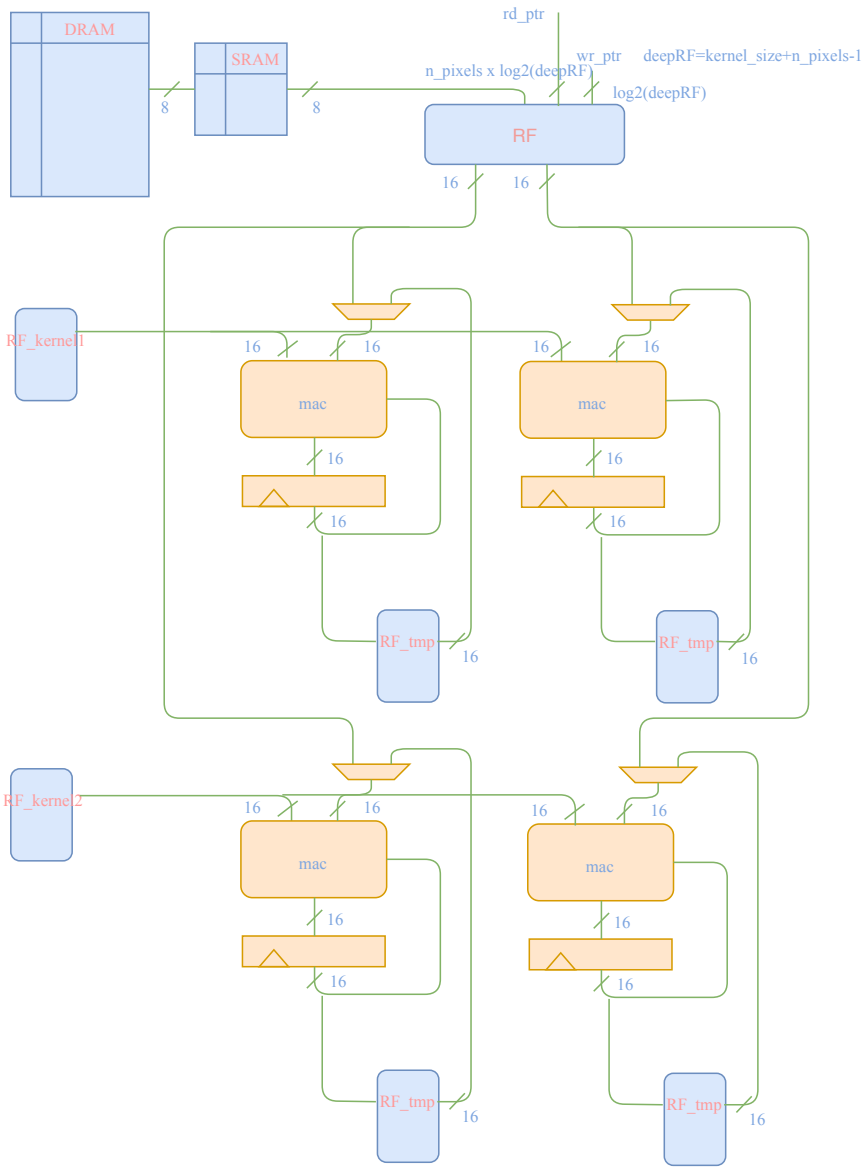


Figure 3.4: Data path of 2D convolution for $n\_scales = 2$ and $n\_pixels = 2$

Figure 3.4 shows a sketch of the data path for the 2D convolution module, with $n\_pixels = 2$ and $n\_scale = 2$. The DRAM feeds the SRAM with the original pixel's value. Once the SRAM filling is completed, the SRAM itself is read and the pixels' values are stored such that the

central pixel corresponds to the central position of the register file, RF. The register file has N slots, with $N = kernel\_size_{max} + n_{pixels} - 1$, where $kernel\_size_{max}$ is the number of kernel coefficients related to the maximum scale to be processed. Furthermore, the RF is instantiated with a number of ports that is equal to the number of pixels: the same port feeds different scales since the same pixel has to be processed with different kernels. The number of kernel register file, LHS of Figure 3.3, is equal to n_scale because each scale needs its kernel coefficients. Instead, the RFs_tmp register files are used to store, temporarily the results of row-wise convolution process, that are later needed for the column-wise step. Their size is $((kernel\_size - 1 + m) * n)/n\_pixels$, since each RF_tmp needs to keep just one pixel every n_pixels for each row.

In order to clarify the behavior of the FSM that controls the 2D convolution, a chart is reported in Figure 3.5: Every FSM that is reported, during the report, would not represent a connection between the *done* and the *start* state of each FSM. For graphical reasons it has been avoided, but when a module is in a *done* state and triggers a negated start signal, the machine goes into the *idle* state, waiting for a new event.

Crucial points of the FSM depicted in Figure 3.5 are discussed in the following. In the *UPLOAD_RF_PX_NR* state, the register file (RF_PX) is filled by using the data stored in the SRAM. NR, new row, means that a new row is being processed and all the register file $deepRF = kernel\_size + n\_pixels - 1$ slots have to be filled by the pixels values stored into the SRAM. According to Figure 3.3a) the kernel is centered on the pixel to be evaluated, hence from that pixel $kernel\_size/2$ previous and $kernel\_size/2$ later are needed for the output evaluation[3]. If $n\_pixel > 1$, for example $n\_pixel = 2$, one more pixel is needed in order to cover all the $kernel\_size/2$ subsequent data. The filled red and green boxes represent the pixels that have to be evaluated, whereas the red and green rectangles are the correspondent kernels, properly centered.

[3]It has been considered the central pixel position of an even kernel as $(kernel\_size - 1)/2$

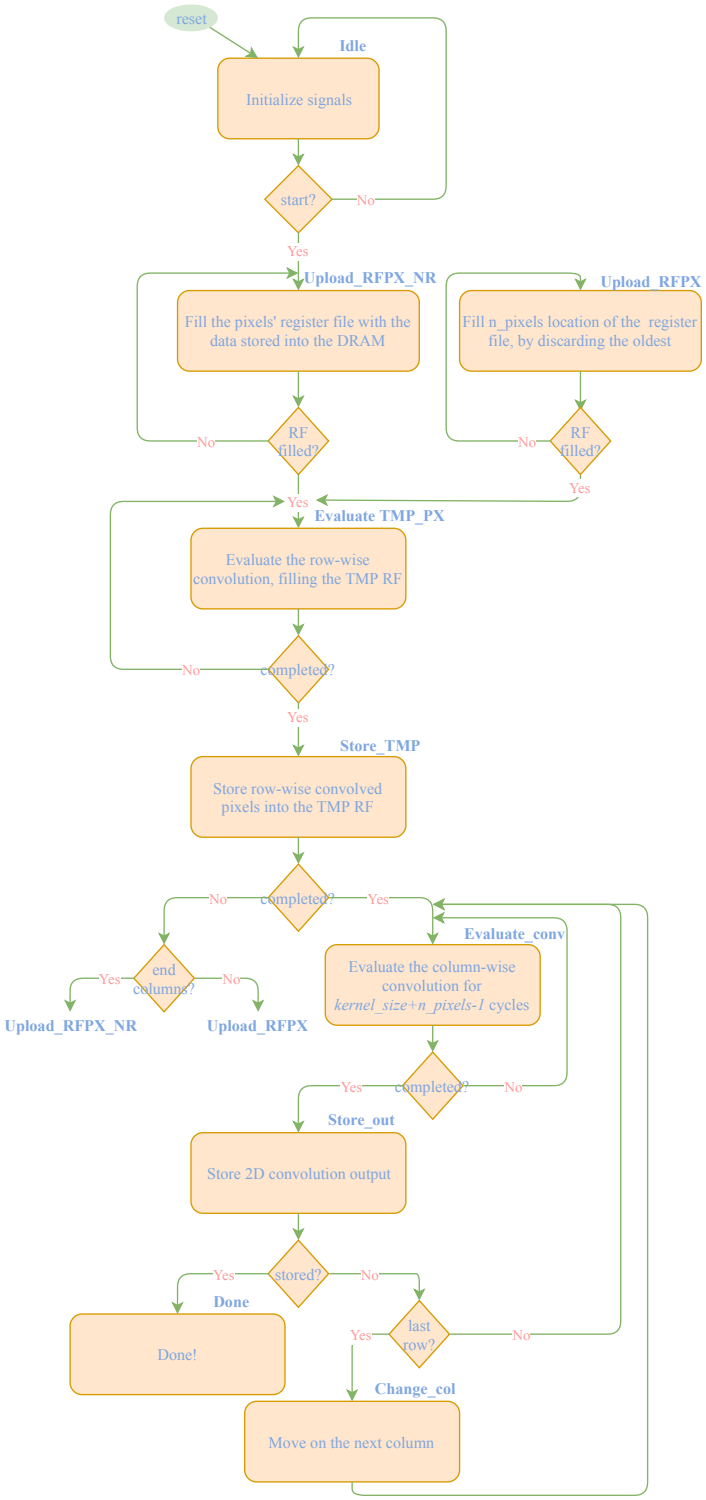


Figure 3.5: FSM that controls the 2D convolution

Once all the process is completed for all the pixels to be stored into the

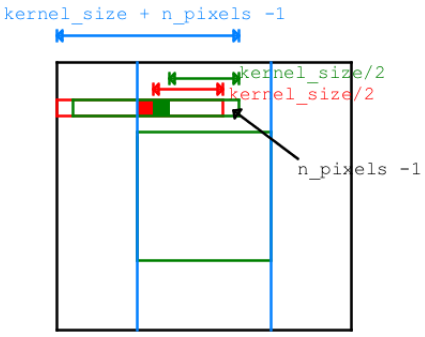


Figure 3.6: Fill pixels' RF when a new row is being processed and $n\_pixels > 1$.

TMP RF the system goes in the *EVALUATE_CONV* and the column-wise operation starts to be performed. Otherwise, if the pixels to be processed are on a new row, the next state of the system is *UPLOAD_RF_PX_NR*, whereas if the pixels to be processed are on the same row then the machine goes in the *UPLOAD_RF_PX*. In the state *UPLOAD_RF_PX* the new n_pixels, on the same row, have to be evaluated. In this case, the previous $kernel\_size - 1$ pixels are already stored, hence only the new n_pixels are brought from the SRAM to the RF, according to Figure 3.7. The notation -box/colors- follows the one used is Figure 3.6, by assum-

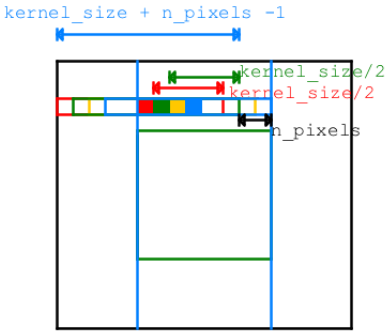


Figure 3.7: Fill pixels' RF when $n\_pixels > 1$

ing that the two new pixels -yellow and blue- have to be evaluated, therefore the correspondent yellow and blue kernels are shifted with respect to the red and green one. The final result shows that, for the

new pixels only the last n_pixels have to be stored because the others can just be kept from the previous cycle. Once the row-wise convolution has been completed, the column wise has to be developed and the result hA to be stored. If the pixels to be stored are the last in the *mxn* block the system goes in the *DONE* state and wait for a new event.

3.1.2 Handling DRAM to SRAM communication

The 2D convolution process is repeated many times, according to the mxn block size. The number of blocks of an XxY image is $n\_blocks = \frac{XY}{mn}$. This size can be arbitrarily decided, but the maximum kernel size required and the available space of the SRAM that has to store $(m + kernel\_size_{max} - 1) * (n + kernel\_size_{max} - 1)$ amount of pixels, must be taken into account.
Figure 3.8 shows a generic green block mxn that is the pixels area evaluated by the 2D convolution, whereas the black area is the image piece stored into the SRAM in order to compute the convolution's output for the green piece.

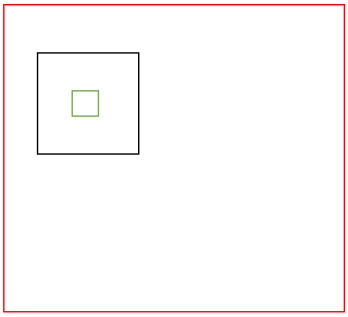


Figure 3.8: The green piece is the m x n part of the image for which the convolution's output is calculated, while the black area is the portion of image stored into the SRAM used for the 2D convolution evaluation.

Once the 2D convolution for the green block has been completed, the next one should be evaluated. According to the previous reasonings, in order to reduce the memory accesses, the part of image already stored, to be used for the next operation, is kept inside the SRAM and just the new part is fetched from the DRAM.
A better explanation is given by Figure 3.9, where two next blocks have been represented: the non overlapped area, marked by the blue color,

is the one that is, effectively, moved from the DRAM to the SRAM, hence a starting address signal is updated and used by the SRAM controller for identifying the pixels' position inside the memory itself.

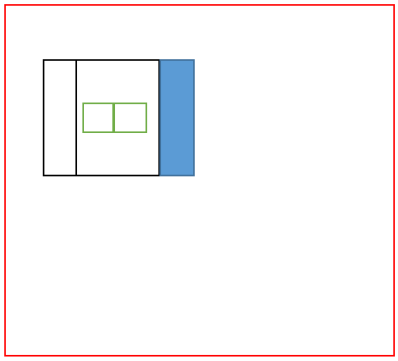


Figure 3.9: The figure shows two next greens blocks evaluated by the 2D convolution. The black areas are the pixels needed by the 2D convolution in order to evaluate the green output. The overlapped area between the two blocks is already stored into the SRAM, therefore only the blue part has to be fetched.

Furthermore, in order to avoid useless accesses to the memory for writing the L function elements, the DoG function has been computed, whereupon the D elements have been stored.
Achieving that means to compute at least 2 scales at time, therefore during the design the parallelism grade n_scale, along the scales, it has been considered $n\_scale \geqslant 2$.
The duty of this FSM that drives the DRAM control is to fill the SRAM properly, according to the block's number to be evaluated, hence address both the DRAM and the SRAM, fill the kernel's register files, according to the number of scales processed at time, wait the DoG computation and enable the storage into the DRAM itself. At the end of the process, part of the DRAM memory contains the calculated function D, where the number of pixels stored is: $3XY + 3\frac{XY}{4} + 3\frac{XY}{16}$, where X, Y are the dimensions of the image, whereas the factor 3 comes from $n\_scales - 1$, with $n\_scale = 4$ because of the project decisions.
Considering that, on the image border, part of the black area is outside the image, some zeros values were stored into the SRAM. Problem come up for this operation since, by storing the data traditionally row-wise can happens that, for several rows, some *0* need to be stored,

whereupon some image data too; after a row[4] has been completed the phenomenon turns out for the next one. This leads to a continuous open/close of the DRAM page that decreases, drastically, the performances. For each black block that has some "external image" part it has been decided to store first (or later) the *'0s*, and once the page is opened bring in all the pixels' value needed for the output evaluation.

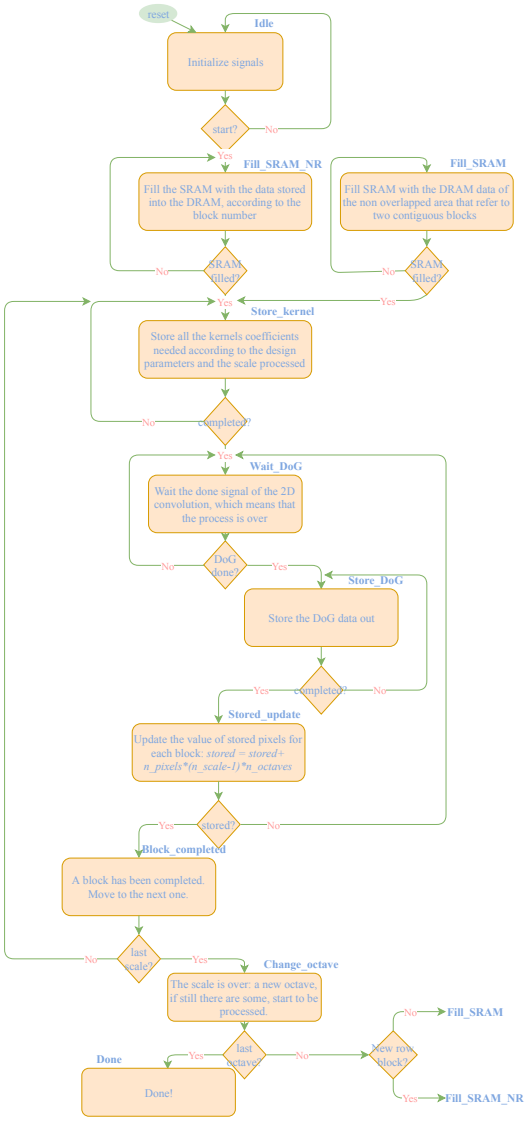


Figure 3.10: FSM for DRAM controller

[4]In this case row means $\frac{Y}{n}$ blocks to be evaluated by moving horizontally along the image.

3.1.3 DoG evaluator

The Difference of Gaussian evaluator is, simply, a series of subtractor that takes as input the 2D convolution's outputs. As for the 2D convolution block, according to the parameter $n\_octaves$ the DoG modules are instantiated.
Figure 3.11 shows how the blocks are bounded, by assuming $n\_ocaves = 3$, $n\_pixels = 1$, $n\_scales = 4$ which means that all the scales are processed at the same time.

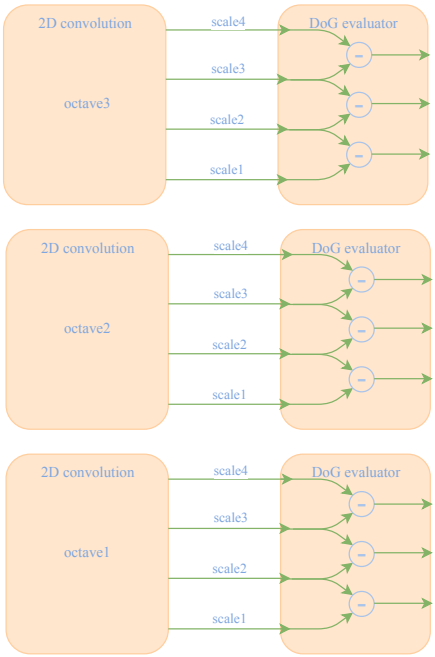


Figure 3.11: The figure shows the DoG modules' instantiation, according to $n\_octaves = 3$, $n\_pixels = 1$ and $n\_scales = 4$.

The DoG pixels are then stored back to the DRAM memory. The number of stored pixels is: $3XY + 3\frac{XY}{4} + 3\frac{XY}{16}$, where X Y are the original image dimensions.
Every data is represented on 16 bits, according to the 2D convolution's output. Since the input data are both positive, the result of the operation will never give overflow, then the 16 bits adders can be used and the D values are, in turn, represented with the same bit's width.

3.2 Accurate keypoint localization

In this section of the report is described the proposed architecture used to detect, accurately, the keypoints. Three main steps were performed in order to obtain the final correct position:

1. Localization of extrema candidates;
2. Get rid of low contrast keypoints;
3. Get rid of poorly localized on edges keypoints.

3.2.1 Extrema candidates localization

Possible extrema are detected, according to the theory, by comparing the pixel candidate with its 8 neighbors and the nine correspondent pixels in the upper and lower scale, as described in Figure 2.2. The DoG values are stored into the DRAM. For this purpose it has been thought to directly access the DoG and store the 27 pixels in the local register file, starting from the bottom to the top scale, hence the central RF data corresponds to the pixel candidate.

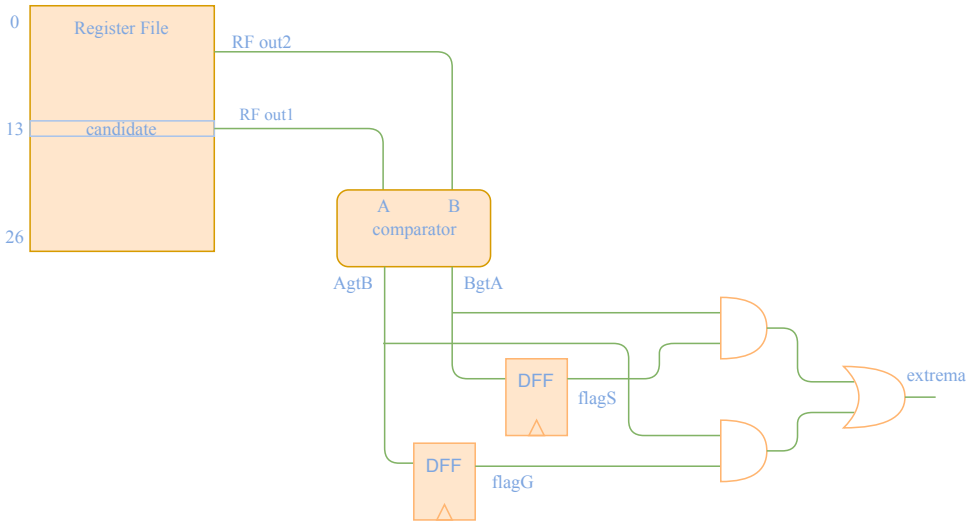


Figure 3.12: Extrema localization data path.

The RF neighbor is changed until the extrema signal becomes '0' or the whole register file has been read. The comparator gives the two

output signals: A greater then B -AgtB-, B greater then A -BgtA-. The flip flops are enabled at the first cycle, therefore if *flagG* is asserted a biggest candidate has been detected and in order to consider that pixel as an extrema for the rest of the cycles the neighbors have to be smaller than the candidate, otherwise the process is stopped and a new candidate is analyzed. If the process continue for all the register file's elements and, in the last step, the signal G or S is still asserted, then that candidate is considered an extrema and its position is stored into the DRAM.
As for the scale-space construction, it has been tried to minimize the DRAM accesses, therefore also in this case, when a new candidate has to be analyzed, instead of fetching in all the 27 pixels' data, just the next right nine more pixels are needed. Figure 3.13 shows two next candidates, the points circled by the black line, with their neighbors needed for the extrema detection. When the new candidate has to be evaluated, hence the points contained in the red area are the pixels to be stored into the RF, only the blue ones are brought from the DRAM to the register file and the others just kept from the previous cycle. An

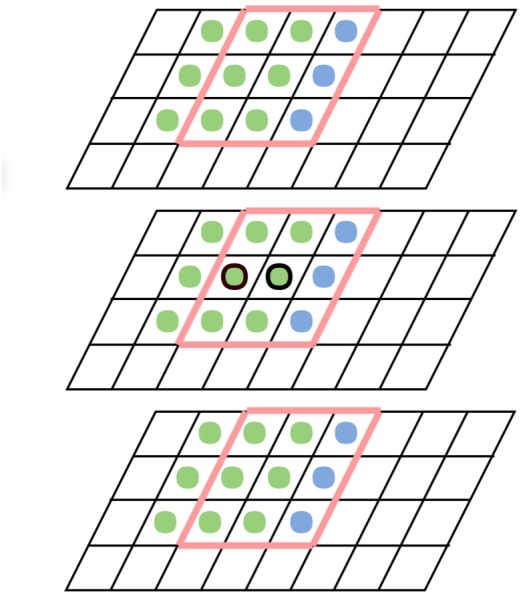


Figure 3.13: Extrema detection moving window

FSM has been designed in order to control the process and cover all the scales and the octaves.

Therefore a flow chart has been reported in order to help the reader understanding how this process has been handled and in which way all the scales have been controlled.
The FSM that controls the process is reported in Figure 3.14:

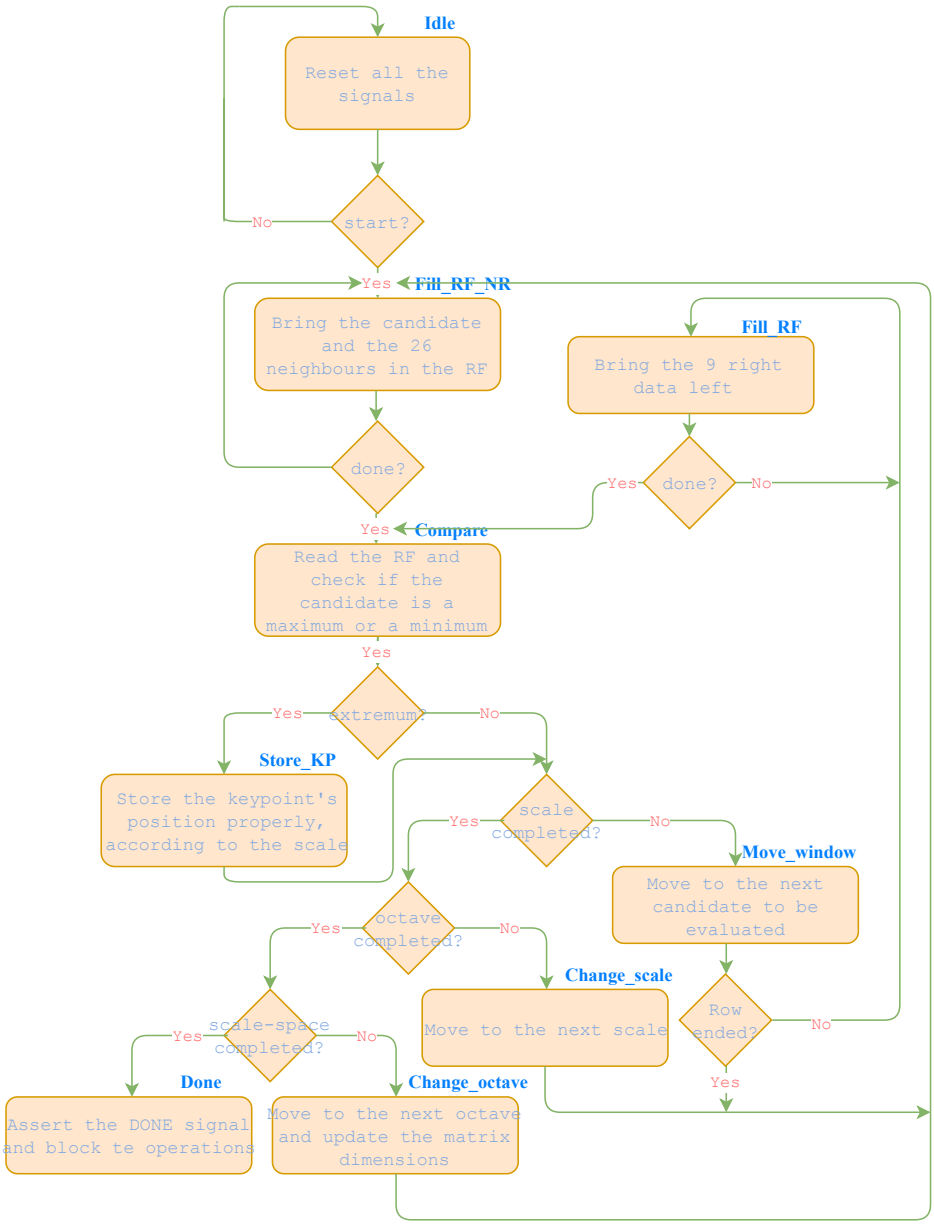


Figure 3.14: Keypoint location FSM

The FSM structure is quite similar to the one that has been described

before for the 2D convolution. The process is really straightforward: the DoG data values feed the RF that, for each candidate, is read until the point is discarder or detected as an extrema. Some states are used to identify the position, scale and octave related to the point itself. Once all the scale-space has been analyzed, the system goes in a steady-state, waiting for a new event.

3.2.2 Get rid of bad keypoints

Low-constrast keypoint

In order to remove the low contrast keypoints the maximum of the **D** function around the extrema has to be found, hence an approximation of **D**, given by the Taylor expansion has been found:

$$D(\mathbf{x}) = D + \frac{dD}{d\mathbf{x}^T}\mathbf{x} + \frac{1}{2}\mathbf{x}^{\mathbf{T}}\frac{d^2D}{d\mathbf{x}^2}\mathbf{x}$$

The process of building this expansion can be translated into a new 2D convolution with the kernels [1 2 1; -2 -4 -2; 1 2 1] and [-1 -1; 1 1], that translates the subtractions/divisions operations into 2D convolutions. The 2D convolution block can be exploited for this purpose, by using a new FSM that controls properly the fill process of the kernel register file, perform the final operations and store back the data into the DRAM. Therefore if the value of the function **D** in the position of the keypoint is lower than the threshold chosen, the point is discarded.
For this purpose it has been assumed that the the position of the keypoints as well as the Taylor expansion series **D** are stored into the DRAM, as reported in Figure 3.15, where, on the left, it is reported the way used to stored the keypoints' position, instead of using a set of two coordinates.

The position of a keypoint is a number $0 < \alpha < 254 * 254$ and from its value it is easily possible to carry out the x, y coordinates. It is clear that, due to the extrema research algorithm, that a keypoint can not lie on the border of the image. However, the LHS of Figure 3.15 is a reduced example of how the keypoint position is identified in a matrix that, in the report, is represented by the D function.

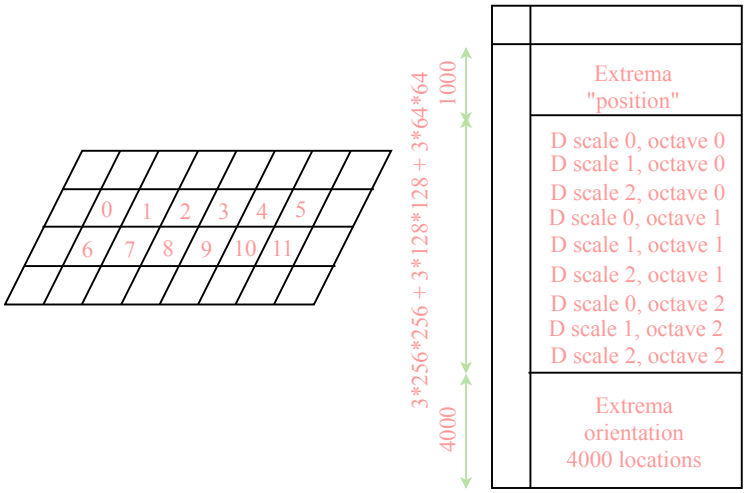


Figure 3.15: Memory structuring for extrema removing

The module that implements this process is, basically, an FSM depicted in Figure 3.16 .

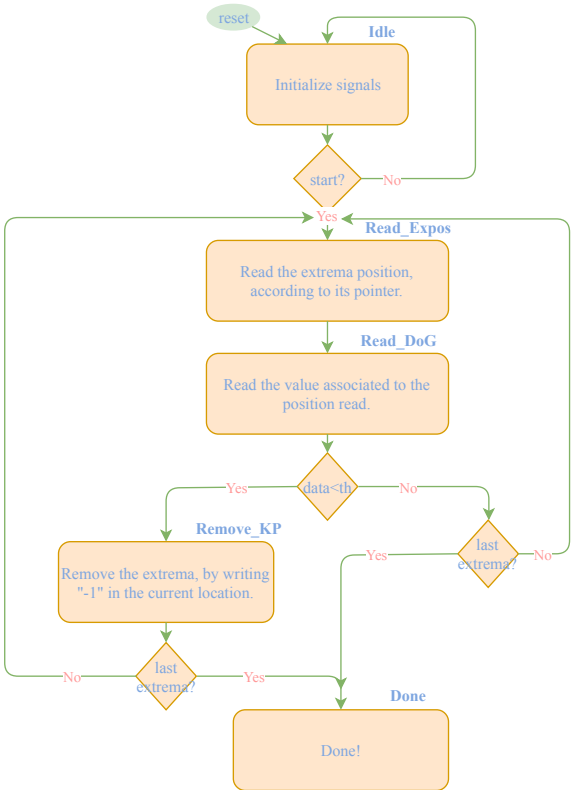


Figure 3.16: Remove low contrast keypoints' FSM

When the process shown in Figure 3.16 starts the system goes into the state *Read_Expos* where the keypoint position is read. The system, then, moves to the *Read_DoG* state in which, based on the position extracted in the previous step, the correspondent **D** value is read: if the data is lower than the contrast threshold the keypoint has to be rejected, hence the FSM moves into the *Remove_KP* state. In the *Remove_KP* state the system write "-1" inside the correspondent keypoint location. Once all the extrema are analyzed the finite state machine moves into the *Done* state and waits for a new event.

Poorly localized on edges keypoints

The matrix H elements were computed starting from the Taylor function D, already used in the previous step.
The DP is composed by an adder/sub and a multiplier, as well as a pipelined divider provided by the *synopsys* design-ware (DW) library. The arithmetic units are feed properly by a control unit reported in Figure 3.17.
When the FSM is triggered by the start signal, it goes into the *Read_Expos* state where the system check if that keypoint has already been discarded (the value "-1" is read), otherwise the machine goes into a series of states where the element of the hessian matrix **H** are evaluated. Once all the elements are evaluated the determinant and the ratio are computed, therefore if the result is greater than the threshold discussed in Section 2.1, the FSM goes in the *Remove_KP* state and "-1" is written into the location, as already described for the previous module. Once all the keypoints are analyzed the process is completed and the system goes into the *Done* state.
The threshold used in this module comes from the parameter $r = 10$, as suggested in the SIFT paper [7].

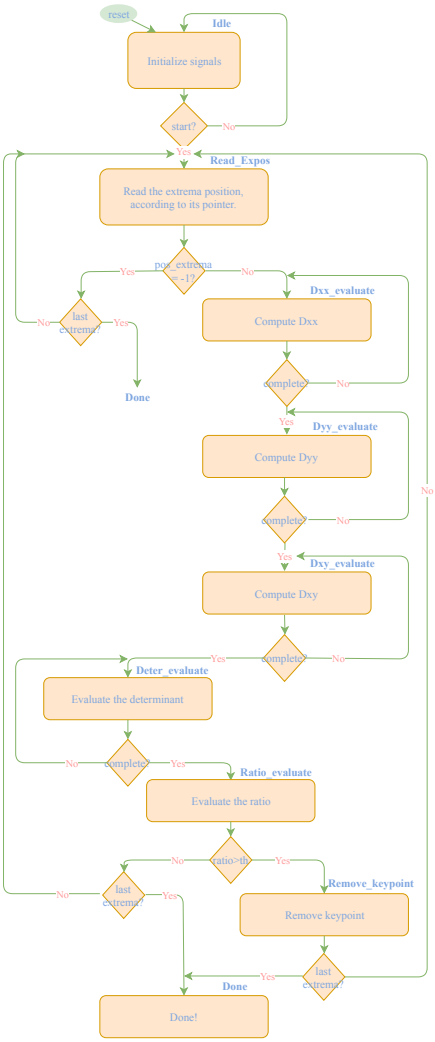


Figure 3.17: Remove edges control unit

3.3 Orientation Assignment

According to the formula mentioned in Section 2.1, an histogram has been built in the proximity of the keypoint. Each pixels' magnitude contained into the histogram is weighted by a Gaussian circular function. The higher the scale, the wider the number of pixels that are inside the histogram area itself - $width * width$- where $width = 2 * round(3 * 1.5 * \sigma)$ in order to bring in the same amount of information, due to the Gaussian shape.
According to statistics extracted from the *MATLAB code*, on $256x256$

resolution image, 1000 memory locations have to be reserved for the extrema positions. Moreover, it has been decided to have 4 maximum orientations for each keypoint, where the keypoint orientation will be a number *bin* in the range 0-35 that corresponds to the orientation $10xbin$, expressed in degrees. The memory locations reserved for this purpose are shown in Figure 3.15.
For this module it has been thought, as already done for some of the previous block, to implement and handle a memory hierarchy in order to speed up the process. Hence the control unit manage also the data communication from the DRAM to the SRAM.
The arithmetic unit architecture proposed for this purpose is reported in Figure 3.18.

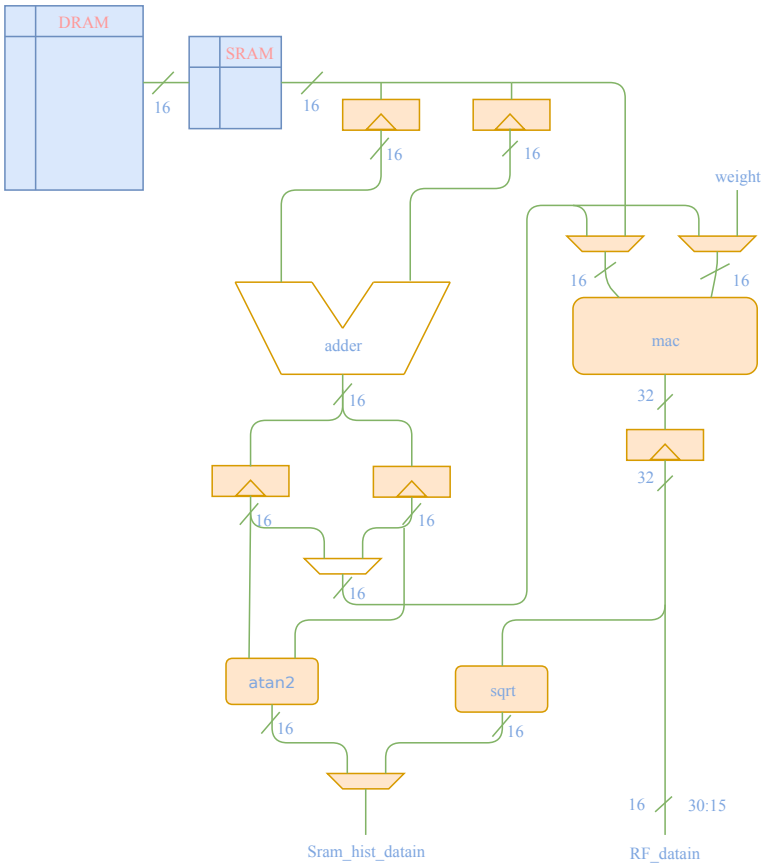


Figure 3.18: Assign Orientation data path

The *sqrt* function has been developed by exploiting the Design Ware synopsys library, using a pipelined version with 3 pipe stages. Furthermore, the *arctan* has been developed by using a small logic and a 5-inputs Look-Up Table (LUT), since the granularity needed for the purpose of the module was just 10 degrees.
It has been supposed to use two different SRAMs: one for storing the pixels' value of the area around the keypoint, the other one for storing both the magnitude and the orientation of each of them.
The flow chart which represents the control unit behavior for this module is reported in the Figure 3.19.
The FSM basically controls the following operations basically:

- The window around the keypoint needed to evaluate the pixels' "vectors" is brought from the DRAM to the SRAM.
- The computations are performed, therefore the second SRAM is filled with the magnitude and orientation values of each pixel in the window.
- The SRAM just mentioned is read, the vectors are properly weighted by the gaussian circular function and a final magnitude for each angle is obtained, therefore a 36 slots (0°, 10°,...350°) register file is filled with the magnitude, such that for each angle the correspondent magnitude is stored.
- Finally, the RF is read and the maximum orientation is found by using classical research methods.

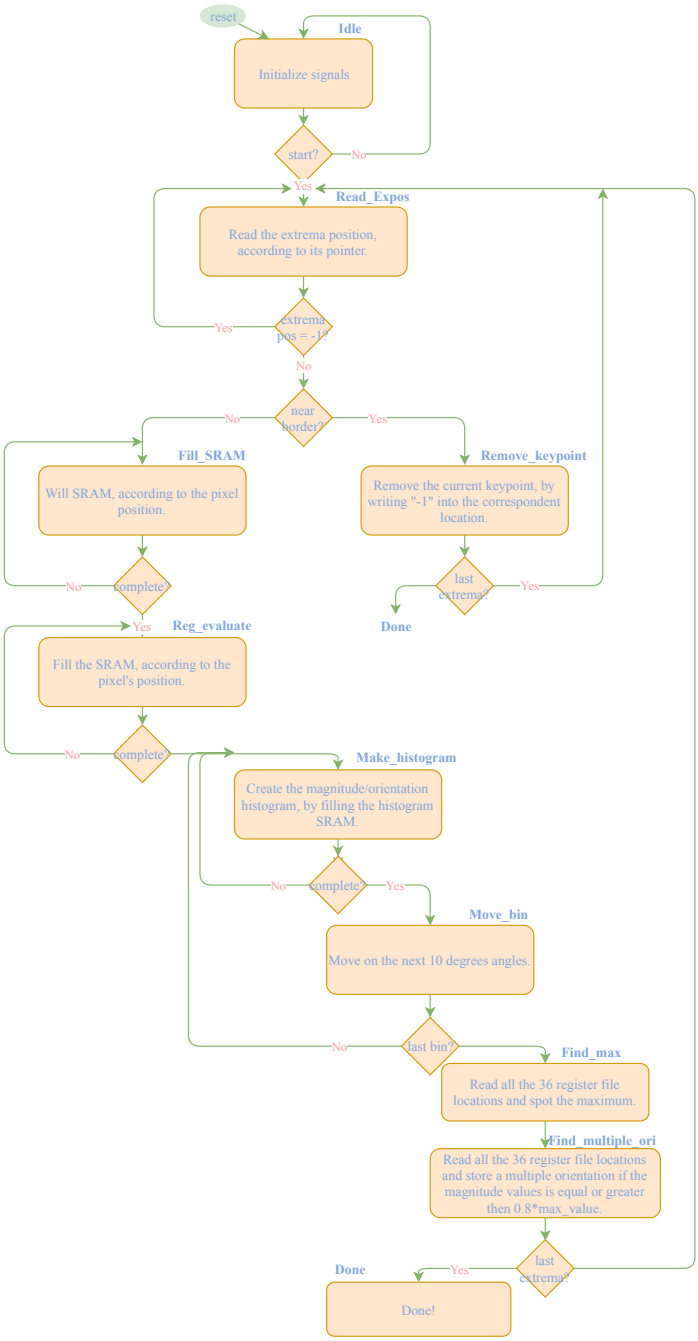


Figure 3.19: Asssign orientation FSM

3.3.1 Atan2 module

The *atan2* module is, in a variety of computer languages, a function that has two input *x,y* representing the coordinates of an arbitrary

point (x, y) and finds the arctangent of θ, where $tg\theta = \frac{y}{x}$. For the system application, the output resolution of this module needed is 10 degrees. Due to the coarse granularity, a 6- input LUT, driven by a small control logic, has been developed to carry out the $atan2(x, y)$ value. Figure 3.20 depicts the proposed architecture of this block.

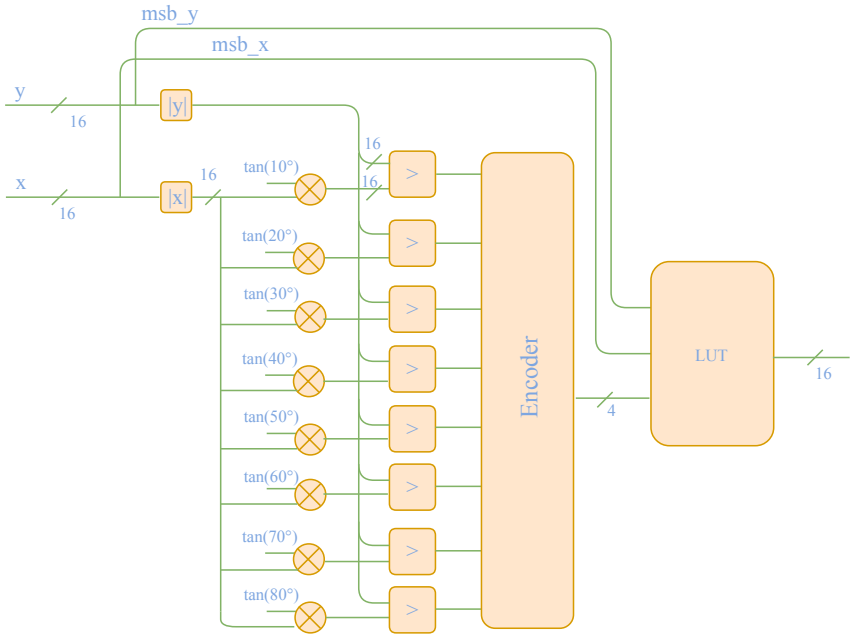


Figure 3.20: *atan2* proposed architecture

3.4 Local descriptor and matching algorithm

For this part of the algorithm it has been thought to use a software implementation in order to create the local descriptor and match the descriptors that comes up from the two different images. For the local descriptor a $MATLABR\_2016_a$ code has been implemented, as well for the matching algorithm. It has been used an *2,5 GHz Intel Core i7 processor* in order to run the program. The threshold discarding ratio between the nearest and the second nearest neighbor has been set to 0.8, as suggested in [7].

3.5 RANSAC for global homography

This section will illustrates how the global homography and RANSAC have been developed to find the transformation that allows the stitching.
According to the *MATLAB* simulations and the number of matched points that come out from the matching algorithm the iteration number has been set to 50, but can be easily modified through the parameter itself defined in the package. The results obtained were carried out by using the parameters and threshold discussed previously in the paper. Furthermore it has been decided to upper bound the number of matched points to 32 (should be increased for different threshold and parameters using in *SIFT*).
The data format used is $Q18.46$ according to the simulation results and the operations performed by the algorithm. The huge number of fractional bits come from the high resolution needed by the SVD algorithm, which iterate until an error is below a certain threshold. Many simulation have been run until this format has been chosen after verified that the convergence is always reached.
The entire process is controlled by an FSM that triggers each module and properly drives the Register File (RF)s. Duty of the same FSM is to work as an arbiter that select which module has to talk with the data path that it is shared between all the units inside the module itself.
This FSM first drive the signals for properly storing the point indexes, after while generates the matrix A discussed in Section 2.3, then gives a start signal to the SVD module and wait for the result. Iterate this process for 50 times, according to the defined parameter. Each cycle, once the matrix **h**, based on four points, is obtained (SVD is completed), triggers the module that evaluate the points' distance with respect to the homography matrix just computed. Furthermore, it evaluates the number of inliers by reading the data register file: if an inlier is spot, *1* is stored into the 5-bits RF, otherwise *0*. If the number of inliers exceed the "Inliers_max", the "Inliers_max" locations itself are updated.
Once the iterations are over, the FSM fills the A data with the inliers max and triggers the SVD module. The process is completed when the SVD gives the final homography matrix.
The FSM chart that represents the behavior is reported in Figure 3.21:

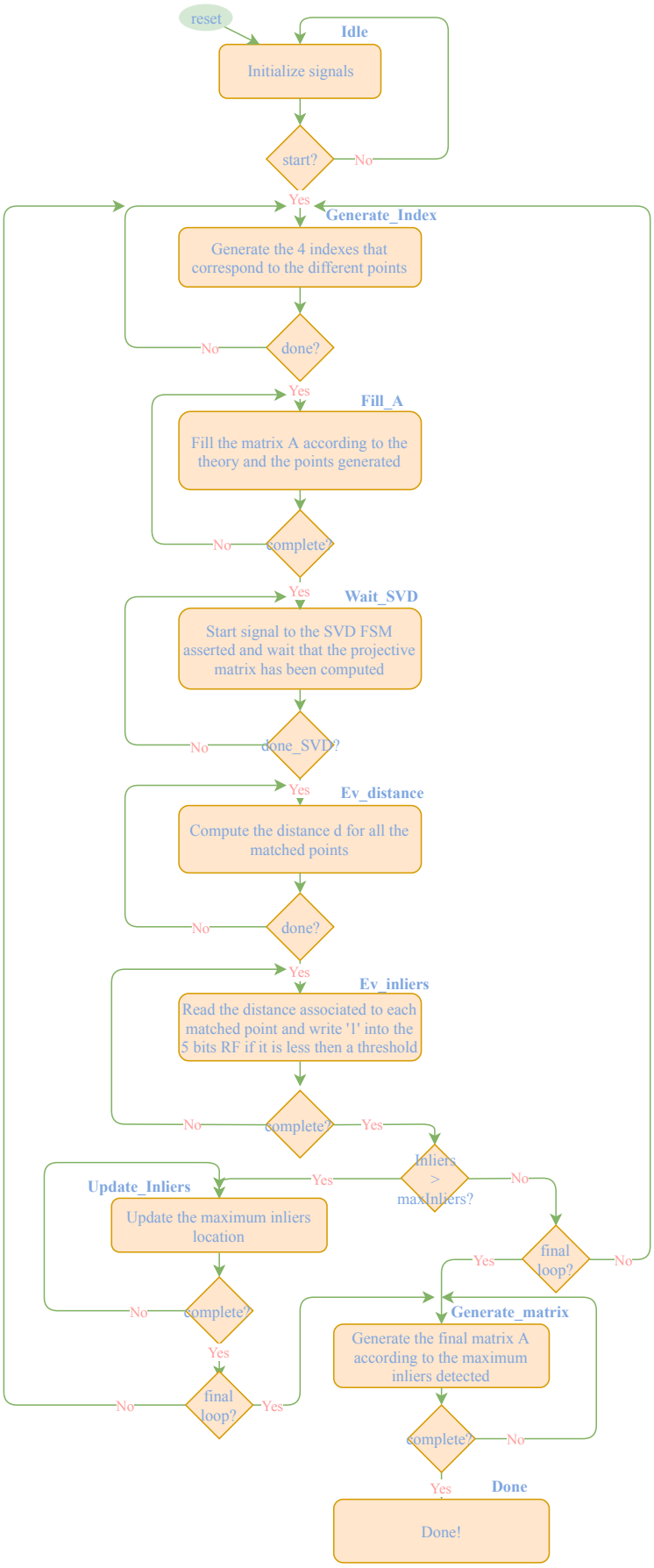


Figure 3.21: FSM RANSAC control unit

3.5.1 Random index generator

The random four points correspondences have been created by using Linear Feedback Shift Register (LFSR). The number of matched points is evaluated at run-time by the matching algorithm and it is used to select the correct range within the points have to be generated.
Therefore, multiple sizes LFSRs have been instantiated and a small logic, driven by the number of matched point signal, decides which one has to be selected, by properly controlling a 4 ways mux.
Figure 3.22 shows how the random generator block is built.

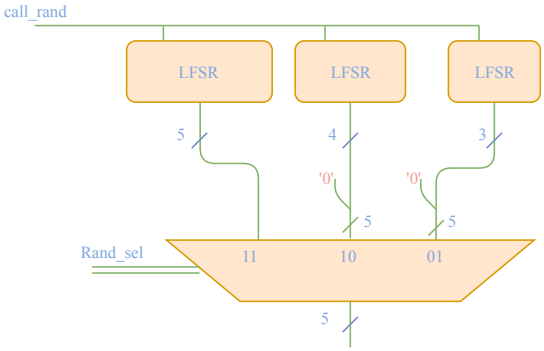


Figure 3.22: Random generator with multiple size LFSR

The *call_rand* signal enables the LFSRs to change state, then generates a new random number. The signal is asserted every time a new number is needed. Since, for the purpose of the project, four points'indexes were needed, the signal is asserted for 4 sequential cycles, hence the output is stored into a small RF.
However, the *rand_sel* signal works in a way that ensures that the index will be inside the number of matched points evaluated at run time.
For the sake of clarity the logic is described as:

$$Rand\_sel \leftarrow \begin{cases} 11 & \text{if } n\_matched = 32 \\ 10 & \text{if } 15 \leq n\_matched < 32 \\ 01 & \text{else} \end{cases}$$

The LFSR itself has been designed by using XORs feedbacks. The polynomials used are the ones that ensure the maximum count cycle: $2^n - 1$.
Different n-lenght LFSRs correspond to different polynomials.
The latters are reported in the table below:

	3 bit	4 bit	5 bit
Polynomials	$x^3 + x^2 + 1$	$x^4 + x^3 + x^1$	$x^5 + x^3 + x^1$

Table 3.1: Polynomial for maximum length LFSR

The LFSR behavior and the concept of feedbacks' polynomial is clarified by Figure 3.23, where a 16-bit LFSR is represented, with a feedback polynomium: $x^{16} + x^{14} + x^{13} + x^{11} + 1$.

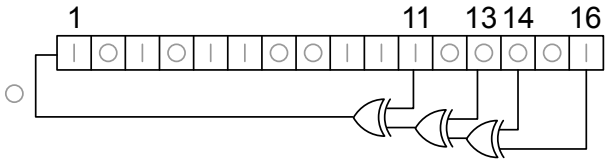


Figure 3.23: Fibonacci series LFSR

The indexes are stored into the first 4 location of a small RF that containts that are then used to point to the correct location of the RF that cointains the coordinates of the matched points.

3.5.2 Data path

As already said, the RANSAC data path is completely reused for all the algorithm involved in the global homography process. An *arbiter* signal controls in which way the AUs are fed.
Basically, four arithmetic units have been used: adder/subtractor, square root unit, divider and a MAC. The synopsy design ware library -DW- has been exploited for the sqrt and the divider, both in a pipelined version, with 5 pipelined stages each.
The internal units are controlled by the FSM that is currently using the arithmetic units, by properly driving the severals control signals such as muxes selectors and different units enables.

The proposed architecture is reported in Figure 3.24.

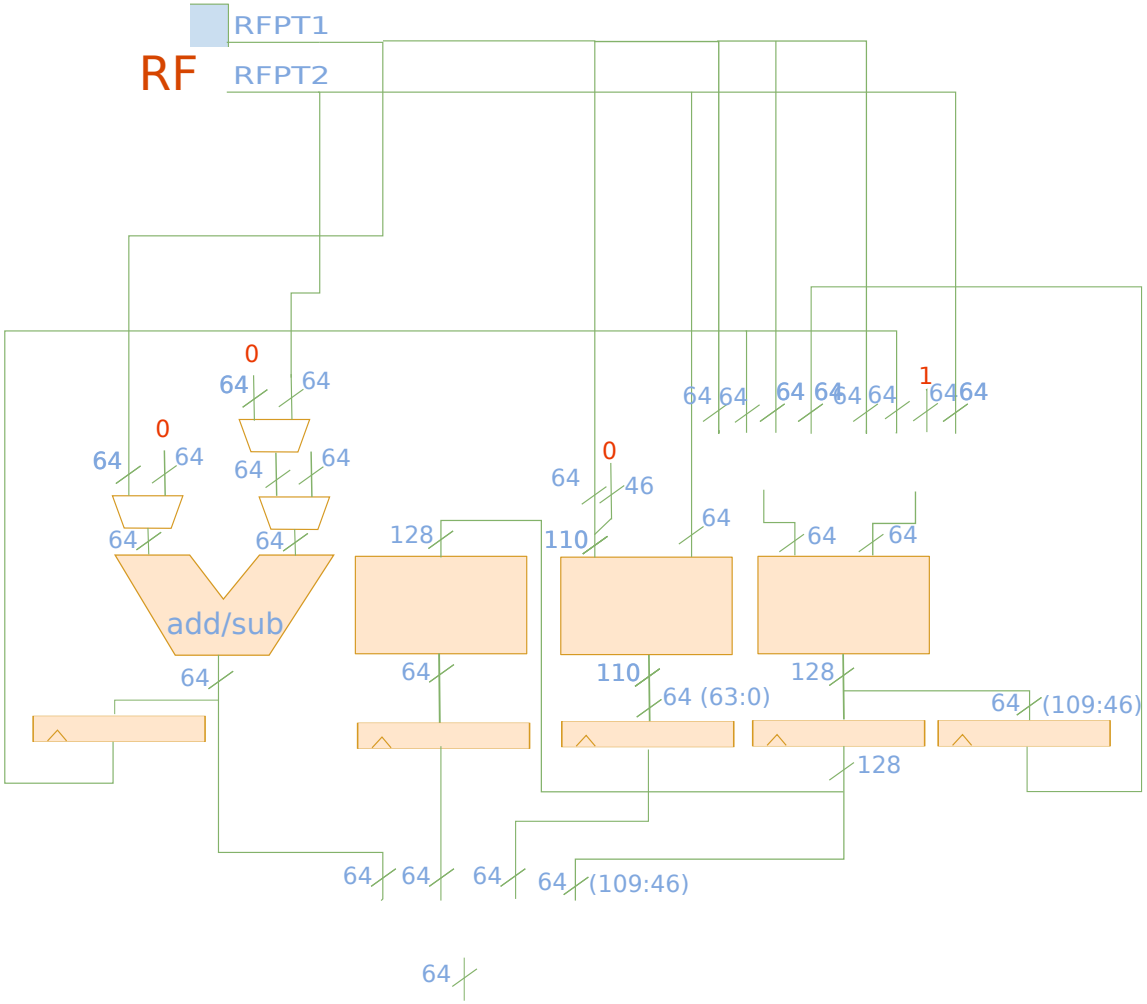


Figure 3.24: Proposed architecture for global homography datapath

According to the operations, the divisions' result is always a number lower than 1. To avoid precision's losses it has been shifted the dividend by 46 bits (the number of fractional bits used) and then the first 64 bits have been extracted from the output.
However, for the MAC unit, it has been chosen to doubled the register output size in order to avoid overflow in each multiplier operation. Because of the data format and the *sum+multiplication* operations the output data has the format Q36.92, therefore the bits *109:46* have been picked up, going back to the Q18.46 representation. Figure 3.25 shows how the fixed-point operations are performed.

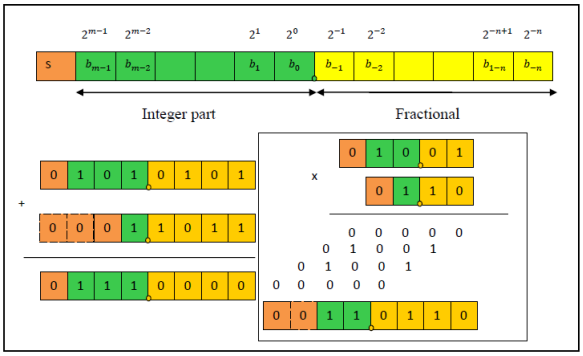


Figure 3.25: Basic operations fixed point format. The figure has been extracted from [21].

3.5.3 SVD implementation

The module that performs the operation which allow the decomposition of the A matrix by exploiting the SVD is basically a pair of hierarchically FSMs that properly drive the controls and signals to the Data Path (DP) and the storage units.
Among the different ways for the decomposition evaluation, the QR factorization trough the householders' method has been chosen, due to its simple implementation. However, since for the project's purpose U and S, defined in Section 2.3.1 are useless, the hardware design has been projected *ad hoc* for it, by skipping the operations that carry out the all the output parameters of the SVD function.
Basically, finding the matrix V (as well as the other parameters of the factorization) is an iterative algorithm that "calls" twice the QR factorization method and iterates until the error is lower of a desired tolerance. According to statistics the tolerance $tol = 0.1$ has been chosen as a trade off between data resolution and algorithm accuracy.
One of the two FSM performs the QR factorization, whereas the second one triggers the QR module itself, evaluates the matrix v and controls the entire iterative process.
Furthermore, whether the FSM does not reach the desired tolerance, it will be bounded by a maximum number of iterations, set to 5.
Statistics, previously done on MATLAB shows that the desired tolerance is reached 95% of cases in 2-4 iterative cycles, otherwise a wrong

result comes out.
The FSM that handles the entire loop and drives the QR module is reported in Figure 3.26.
The CU represented in Figure 3.26 is straightforward related to the algorithm reported in Algorithm 2. Once the process start, the FSM goes into the state that allow the computation of the first QR factorization. Once the process is completed, the data are properly stored into the correct RF location such that the next module's call has its input ready to be processed. It is not possible to reuse the same RF locations of the output data as input since some data would be overwritten and the result would not be correct. Once the second QR factorization and the data are correctly stored, matrix V is evaluated, therefore the error is computed. The *Check_err* state verifies if the error is below the desired tolerance or the number of loops reached the maximum decided and goes to *Done* if Yes, otherwise goes back to the beginning of the loop.

3.5.4 QR factorization implementation

The QR module is, as already said before, an FSM that properly controls the DP in order to perform the operations reported in algorithm 3.
The FSM is in the *Idle_QRfact* until the SVD FSM assert the start signal. The system starts and the matrix R is filled, by copying the values from A to R. The system moves, then, into the *Build_X* state, where the variable X is built. Algorithm 3 is followed in order to perform the operations. In the states *Prod_R* and *Prod_Q* some temporary variables are used to store intermediate values that are then used for the final result. Once the loop is completed, the system goes into the *Done* state by giving R and Q as result.

Due to the operations of this module, a very high resolution is needed to obtain good results, that is the reason why 46 bits were used for the fractional bits. The input and output range of the module is very wide - MATLAB simulations in floating point notation shows that numbers goes from very large number that need 18 bits to be represented, as well as small number that are of the order $1e-30$. A good compromise has been achieved by using the notation Q18.46.
The FSM behavior is reported in Figure 3.27.

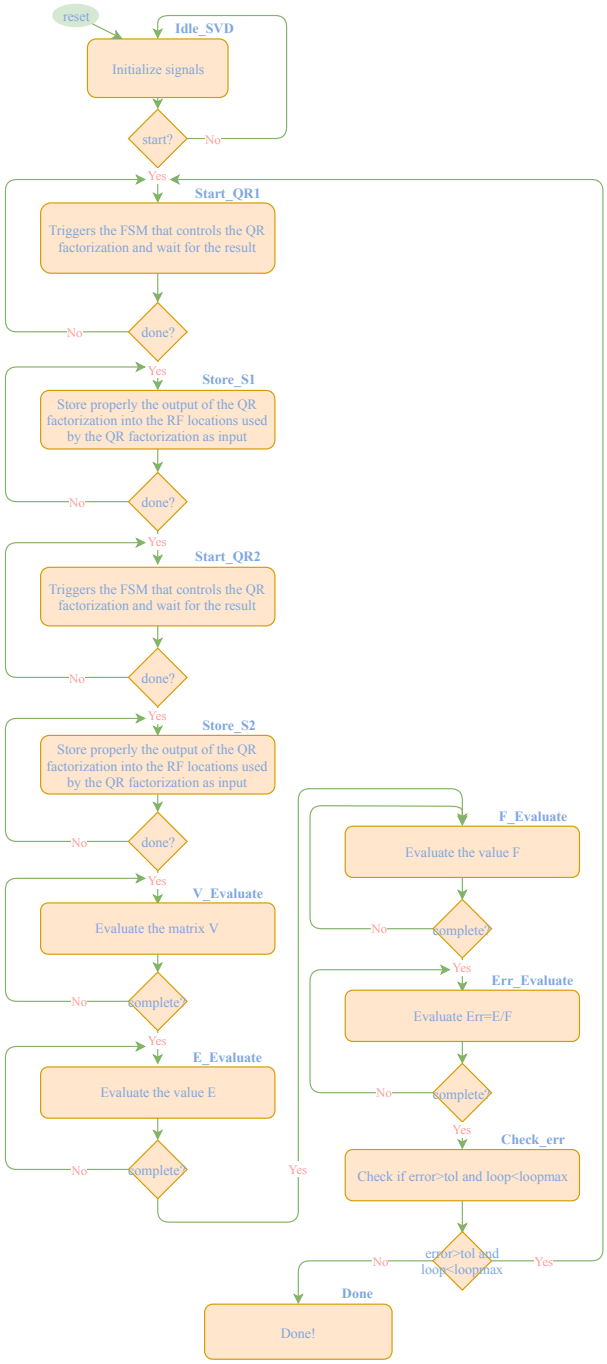


Figure 3.26: FSM that controls SVD operations

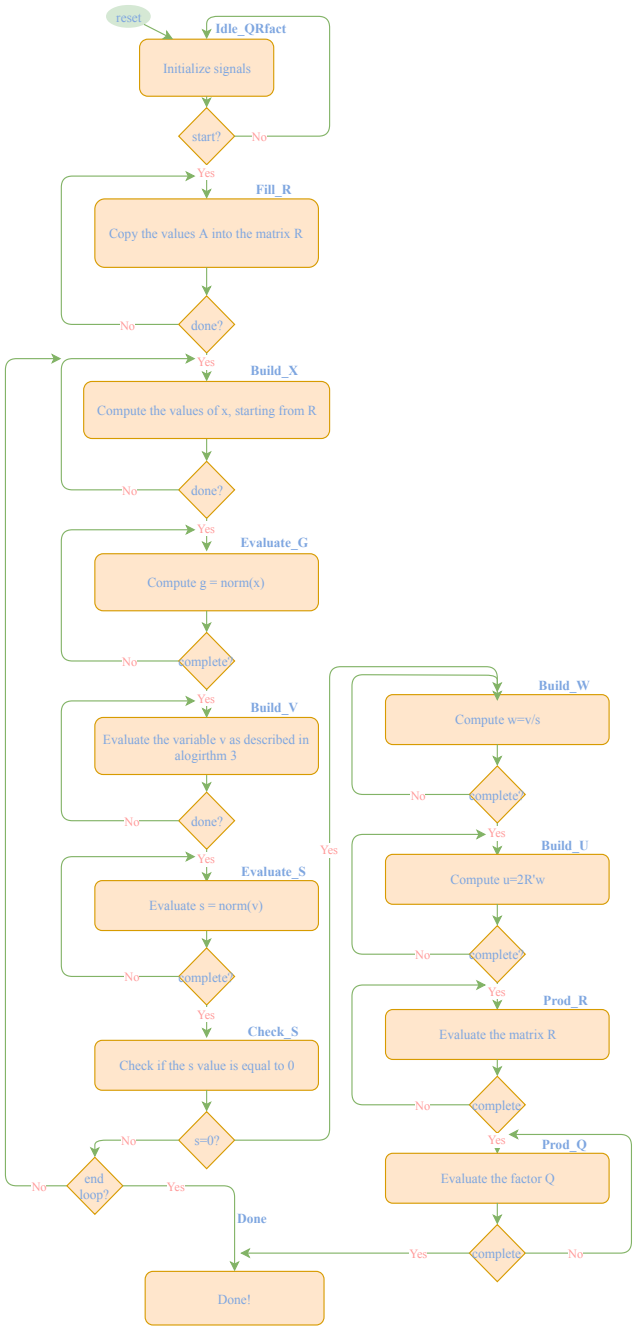


Figure 3.27: FSM that controls the QR fact operations.

3.5.5 Distance evaluator

The last module implemented for the global homography design is the distance evaluator. This block has the duty to evaluate, for each

point in the set of matched points, the distance with respect to the homography matrix previously evaluated. To do that, first the points of the second matrix are transformed by multiplying them with the transformation matrix. Once the transformed points, called *PTS3* in the chart of Figure 3.28, are obtained, the distance is computed with the squared differences between the matched point of the fixed image and the transformed points of the second one.

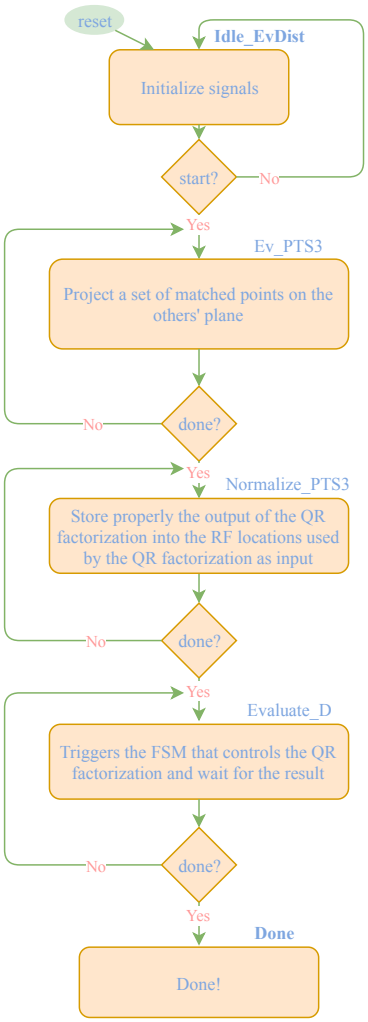


Figure 3.28: FSM that controls the system for the distance evaluation.

3.5.6 Global homography overview

A global vision of the interconnected block is reported in Figure 3.29.

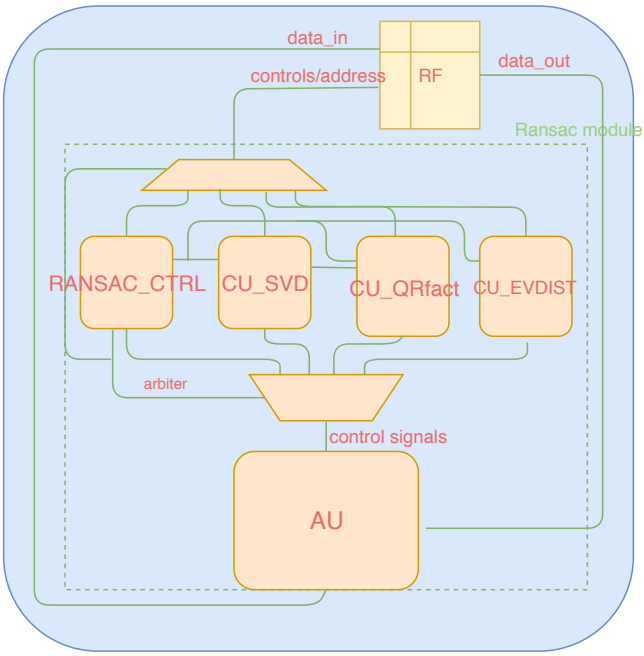


Figure 3.29: Global homography by using RANSAC overview

Figure 3.29 summarize the previous discussion. The *Ransac_ctrl* block feeds the others FSM and triggers, properly, the different modules. The arbiter signal, that comes from the *Ransac_ctrl* itself, selects the correct group of signals for feeding the AU and the control/addresses to the RF. The output data of the RF is bounded to the input port of the RF, whereas the two output ports of the RF feed the arithmetic unit that are then used for the internal operations.

Chapter 4

Results

In this chapter the results obtained will be reported. First an overview of the image processing is provided by showing intermediate results of the algorithm up to the final stitched image.
Then, more in details, Section 4.2 will provide all the area, power and latency results related to the modules implemented.

4.1 Overview

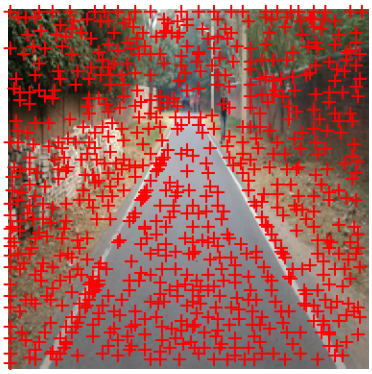
The system has been tested by using the two $256x256$ pixels images shown in Figure 4.1.

(a) Left image (b) Right image

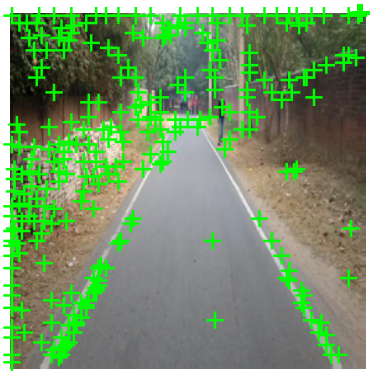
Figure 4.1: Image used for testing the system.

The data input have been given to the *modelsim* testbench through a file, by exploting the *textio* library. The output has been then written on a file, hence MATLAB has been used for plotting the results that are

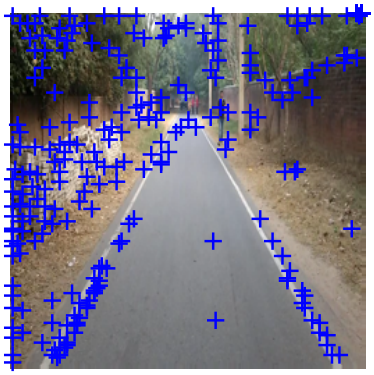
then reported in the paper.
Partial results of the SIFT algorithm are shown in Figure 4.2, where only the image depicted in Figure 4.1a) has been used for the report.
Then, the result of the matched algorithm is reported in the next Figure 4.3, by using 0.85 as a threshold on the distance's ratio.



(a) Candidate



(b) Extrema after low contrast correction



(c) Extrema after poorly localized correction

Figure 4.2: SIFT results: Red points mark the candidate after the first localization step. Green and blue are respectively after the rejections of low contrast and poorly localized on edges keypoints.

Finally, the stitched image is reported after the transformation of the second image with respect to the $3x3$ matrix evaluated by the homography hardware module.
The final plot of the two stitched image is reported in Figure 4.4.

Figure 4.3: Matched point of the two images.

Figure 4.4: Final result of the image stitching algorithm, after the second matrix transformation.

4.2 Synthesis and data analysis

Each module has been synthesized by using *synopsys* through *design_vision* in order to have some results to be discussed and to verify the performance of the system.
A logic syntesys has been done, by using a *40 nm* technology library, and the clock gating technique has been used to reduce the power consumption. Furthermore, during the synthesis, the delay and area introduced by interconnections have been neglected, by using the *zero delay* model. It has been also assumed to work at 25 °C and *NCCOM*

operating conditions. An example script is reported in the Appendix A if more informations are needed on the *synopsys* commands used.

4.2.1 Scale-space construction results

For the construction of the scale space, different parameters have been changed in order to compare the different designs. Three modules are needed for the scale-space construction: *2D convolution evaluator*, *DRAM to SRAM controller*, *DoG evaluator*. Before going ahead with the discussion, it is worth to remind that the kernels coefficient used for each scale are fixed. The kernels' size, related to each scale, are, in order from the bottom to the top of the *scale-space*:

$$9, 12, 15, 19, 19, 24, 30, 38, 38, 48, 60, 76$$

The green block represented in Figure 3.8 has been chosen $mxn = 4x4$ for the following results (it is not necessary to choose a square: the only mandatory condition is that the n dimension is sub-multiple of the number of columns of the whole image).
The values $n\_scale = 2$ and $n\_scale = 4$ have been used: for each $n\_pixels = 1$, $n\_pixels = 2$, $n\_pixels = 4$.The area and power results reported in Table 4.1

	2D convolution	
	Area	Power $[mW]$
$n\_pixels = 1$	52082.45	8.32
$n\_pixels = 2$	54811.36	8.37
$n\_pixels = 4$	60199.85	8.4529

Table 4.1: Result 2D convolution for $n\_scale = 2$. The area is expressed in terms of gate count.

The latency of the 2D convolution module, depends on the kernel size evaluated, therefore after having verified that the constraint on the $5ns$ clock period, arbitrarily decided, were satisfied, multiple simulations, by changing the kernel size parameters, were done in order to extract the performances that are reported in Table 4.2.

	Latency 2D convolution[μs]					
$kernel\_size$	12	19	24	38	48	76
$n\_pixels = 1$	6.4	12.9	19.2	43.6	68.8	160
$n\_pixels = 2$	3.7	7.7	11.5	26.0	40.0	95.2
$n\_pixels = 4$	2.4	5.1	7.6	17.2	26.5	63.2

Table 4.2: Latency result 2D convolution for $n\_scale = 2$.

As well as for the 2D convolution module, the synthesis has been performed for the DRAM controller and for the DoG evaluator. The results are reported in table 4.3, for $n\_scales = 2$.

	DRAM controller		DoG evaluator	
	Area	Power[mW]	Area	Power[mW]
$n\_pixels = 1$	10827.79	0.74	76.91	0.03
$n\_pixels = 2$	10905.93	0.73	153.29	0.06
$n\_pixels = 4$	10802.03	0.73	406.05	0.12

Table 4.3: Result DRAM controller and DoG evaluator for $n\_scale = 2$. The area is expressed in terms of gate count.

Since at the current project status the modules have been developed separately, a rough estimation was done for the scale-space construction's perfomances evaluation.
The following formulas, for $n\_scales = 2$ have been used in order to carry out significant values:

$$Latency\ 1st\ octave = \underbrace{\frac{X}{m}(m + kernel\_size\_4 - 1)(n + kernel\_size\_4 - 1)\ T}_{Storage\ of pixels\ value\ from\ DRAM\ to\ SRAM\ each\ "new\ row"} +$$

$$+ \underbrace{(\frac{XY}{mn} - \frac{X}{m})(n)(m + kernel\_size\_4 - 1)T}_{Storage\ of pixels\ value\ from\ DRAM\ to\ SRAM\ rest\ of\ the\ blocks} + \frac{XY}{mn}(L_{2D_{12}} + L_{2D_{19}})$$

$$Latency\ 2nd\ octave = \underbrace{\frac{X}{2m}(m + kernel\_size\_8 - 1)(n + kernel\_size\_8 - 1)\ T}_{Storage\ of pixels\ value\ from\ DRAM\ to\ SRAM\ each\ "new\ row"} +$$

$$+ \underbrace{(\frac{XY}{4mn} - \frac{X}{2m})(n)(m + kernel\_size\_8 - 1)T}_{Storage\ of pixels\ value\ from\ DRAM\ to\ SRAM\ rest\ of\ the\ blocks} + \frac{XY}{4mn}(L_{2D_{24}} + L_{2D_{38}})$$

$$Latency\ 3rd\ octave = \underbrace{\frac{X}{4m}(m + kernel\_size\_12 - 1)(n + kernel\_size\_12 - 1)\ T}_{Storage\ of pixels\ value\ from\ DRAM\ to\ SRAM\ each\ "new\ row"} +$$

$$+ \underbrace{(\frac{XY}{16mn} - \frac{X}{4m})(n)(m + kernel\_size\_12 - 1)T}_{Storage\ of pixels\ value\ from\ DRAM\ to\ SRAM\ rest\ of\ the\ blocks} + \frac{XY}{16mn}(L_{2D_{48}} + L_{2D_{76}})$$

The final latency of the scale-space construction will be:

$$L_{SS} = Latency\ 1st\ octave + Latency\ 2nd\ octave + Latency\ 3rd\ octave$$

In order to clarify the meaning of each term in the equations Table 4.4 is reported, by showing a legend of the variables mentioned in the equation.

X , Y	image dimensions
m, n	2D conv dimensions
$kernel\_size\_i$	kernel's length of scale i
T	clock period
L_{2D_i}	2D convolution latency for $kernel\_size = i$

Table 4.4: Legend of variables mentioned in the previous equation.

By using $T = 5\ ns$, after having verified that constraints on setup time are met by using the logical synthesis, the results reported in Table 4.5 were carried out, hence the latency of the scale-space construction is reported for all the n_pixels values evaluated before.

	Latency scale-space [ms]
$n\_pixels = 1$	206
$n\_pixels = 2$	124
$n\_pixels = 4$	83

Table 4.5: Scale-space construction latency for $n\_scales = 2$.

The same operations were repeated by setting $n\_scales = 4$ and the results are reported in Table 4.6.

	2D convolution	
	Area	Power [mW]
$n\_pixels = 1$	151883.22	24.09
$n\_pixels = 2$	154783.24	24.20
$n\_pixels = 4$	162169.28	24.4

Table 4.6: Results of the 2D convolution for $n\_scales = 4$. The area is expressed in terms of gate count.

The results reported in Table 4.6 are what is expected from the experiments. The AUs' number increased, hence the area and the power increase as well. It is worth to note that the area evaluated for $n\_scale = 4$, $n\_pixels = 1$ (4 MAC units) is much bigger than the one evaluated for $n\_scale = 2$, $n\_pixels = 2$ (4 MAC) units as well, because a larger register file was needed, which must be able to store more kernels at the same time. The same reasons are valid for the power values since part of the energy is used by the register (clock distribution, higher switching activity).
The different designs were synthesized again to check if the clock constraints were met.
As in the previous case, the 2D convolution results for latency as well as the area and power of the DRAM controller and DoG evaluator are shown, respectively, in table 4.7 and 4.8.

	Latency 2D conv[μs]		
$kernel\_size$	19	38	76
$n\_pixels = 1$	12.9	43.6	160
$n\_pixels = 2$	7.7	26.0	95.2
$n\_pixels = 4$	5.1	17.2	63.2

Table 4.7: Latency result 2D convolution for $n\_scale = 4$.

	DRAM controller		DoG evaluator	
	Area	Power[mW]	Area	Power[mW]
$n\_pixels = 1$	10951.79	0.74	229.67	0.09
$n\_pixels = 2$	10951.79	0.74	153.29	0.18
$n\_pixels = 4$	10951.79	0.74	406.05	0.36

Table 4.8: Result DRAM controller and DoG evaluator for $n\_scale = 4$. The area is expressed in terms of gates counted.

As the parallelism increases the latency continues to decrease. The overall area and the power increases as expected, expecially in modules that require multiple AUs.
Moreover, the latency that is reported in Table 4.7 refers just to the kernel with size $19,\ 38,\ 76$ since all the scales in each octaves are evaluated concurrently and the latency for those values is "hidden" inside the one of the biggest kernel size, which determines the performance of the design.
The overall latency has been estimated as described before with one latency 2D convolution contribution to each block. The results obtained are reported in table 4.9.

	Latency scale-space [ms]
$n\_pixels = 1$	139.5
$n\_pixels = 2$	86.4
$n\_pixels = 4$	59.0

Table 4.9: Scale-space construction latency for $n\_scales = 4$.

The current results of the scale-space construction do not allow a very fast process, therefore a final performance estimation has been done, increasing the parallelism grade, for a $mxn = 16x16$ block, by using $n\_scales = 4$ and $n\_pixels = 16$.
According to the usual formulas it has been obtained:

	Latency scale-space [ms]
$n\_pixels = 16$	7.2

Table 4.10: Scale-space construction latency for $n\_scales = 16$ and $mxn = 16x16$.

The higher the parallelism, the faster the scale-space construction with the drawback determined by the number of pixels needed to be stored in the SRAM in order to compute the 2D convolution.
The area, power and latency results are summarized in Figures 4.5, 4.6 and 4.7. It has been considered to evaluate those parameters for the whole *scale-space* construction process by considering $Area = Area_{2Dconv} + Area_{DRAM-SRAMctrl} + Area_{DoG}$ and $P = P_{2Dconv} + P_{DRAM-SRAMctrl} + P_{DoG}$. Each bar in the figures is identified by $sNpxMscL$. The letter s stands for size and can be either $size1 = 4x4$ block or $size2 = 16x16$ block, pxM means $n\_pixels = M$, whereas scL means $n\_scales = L$. Increasing the parallelism grade the area increases and, how expected, the power increases as well. A bigger gap is visible between values of different scales because both the AUs and the storage units increases. Figure 4.7 has to be discussed a little bit more due to the *red*, *green* and *violet* results that seems to be different for what expected. However, $n\_scales = 2$ and $n\_pixels = 4$ means $Parallelism = 8$, while *green* and *violet* have, respectively, $Parallelism = 4$ and $Parallelism = 8$. Hence, it is clear that the *red* latency has to be lower than the green one. What happens between the *red* and the *violet* is that with the same number of arithmetic units, due to the algorithm, parallelize more pixels has more effect than more scales because the 2D convolution module's latency decreases drastically. There is no optimal solution after this result exploration. The designer has to find trade-off between latency and area/power.

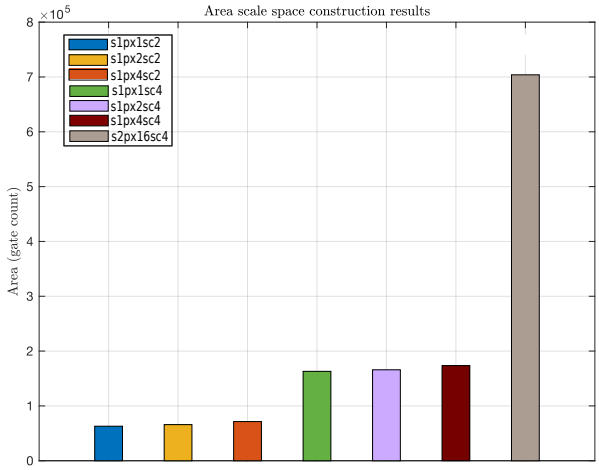


Figure 4.5: Area overall results

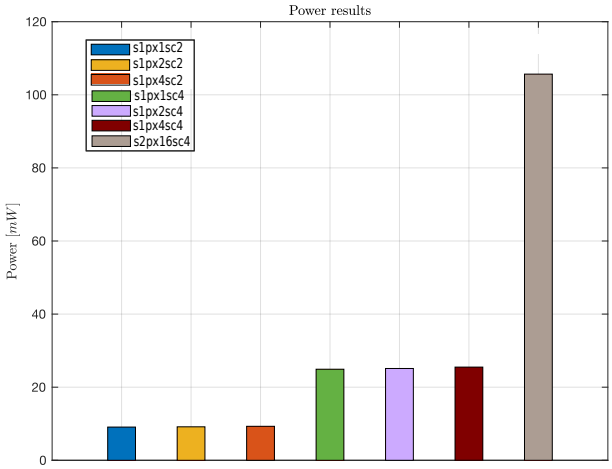


Figure 4.6: Power overall results

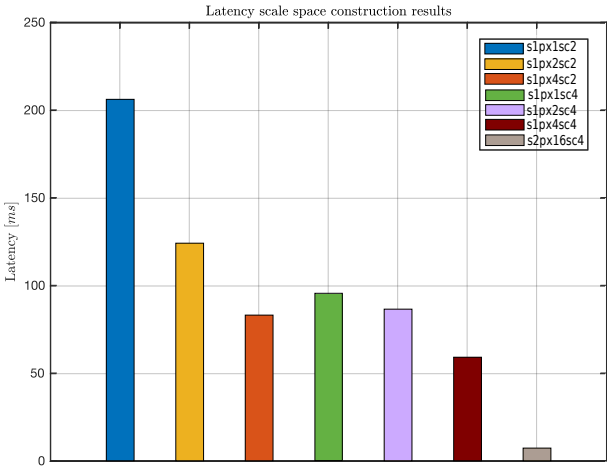


Figure 4.7: Latency overall results

4.2.2 Results accurate keypoints localization

In this section are reported the results for each module involved in the keypoint localization: first the extrema detection module and then the results of the less robust keypoints rejection.

Extrema detection

The VHDL description has been given to *synopsys* for the logic synthesis as it has already been done for the previous blocks. The parameters chosen are the same used for the scale-space construction.
The area and power results obtained that come from the synthesis are reported in Table 4.11 as well as the module latency with a clock period[1] $T = 5\ ns$.

Extrema detection		
Area	Power $[mW]$	Latency $[ms]$
10244.96	0.49	0.12

Table 4.11: Area and power of the extrema detection module.

[1]From now each clock period that has been used for the extracted values has to be considered verified for the module itself in terms of timing constaints.

Results of low contrast rejection module

In the same conditions of the previous blocks the results were extracted out from the synthesis and the simulations. Table 4.12 shows the values obtained.

Low contrast rejection		
Area	Power [mW]	Latency [ms]
2619.01	0.13	0.0076

Table 4.12: Area, power and latency of low contrast keypoints rejection module.

Results of poorly localized on edges keypoints rejection

The results of the poorly localized on edges keypoint rejection are reported in Table 4.13.

Poorly localized on edges rejection		
Area	Power [mW]	Latency [ms]
7232.58	0.34	0.027

Table 4.13: Area, power and latency of the poorly localized on edges rejection module.

The overall latency of the accurate keypoints localization process is:

$$Latency_{AccurateLocalization} = (0.12 + 0.076 + 0.027)\ ms = 0.22\ ms$$

4.2.3 Orientation Assignment results

The Orientation Assignment results are reported in Table 4.14, by using a clock period $T = 5ns$, verified with the synthesis.

Assign Orientation		
Area	Power [mW]	Latency [ms]
24988.12	0.99	31.14

Table 4.14: Area, power and latency of the Assign Orientation module.

The high latency is determined by both the communication between the storage modules and the iterations among all the keypoints detected.

4.2.4 RANSAC and global homography results

Since for this block a final defined design has been developed, only a global synthesis and performance estimation has been done (area, power and latency will not reported for each independent module).
From the synthesis results, performed with 5 pipelined stages, both for the *sqrt* and the *divider* it has been reached a satisfied clock constraint $T = 8\ ns$, but still the critical path is forced by one of this two module, hence could be reduced more by adding pipelined stages.
The results extracted are reported in Table 4.15.

RANSAC for Global Homography		
Area	Power [mW]	Latency [ms]
247482.50	10.83	14.0

Table 4.15: Results of global homography and RANSAC

4.3 Overall algorithm latency

The overall latency at the current status of the project is

$$Latency_{stitching} = 1.6\ s$$

The flow along the values step by step is reported in Figure 4.8. The SW values are obtained by using *tic* and *toc* command on *MATLAB*

2016a, on intel i7 processor 2.5 GHz.
The SW results are reported just for completeness, but they are totally distant from the system target and their meaning is definitely not significant.

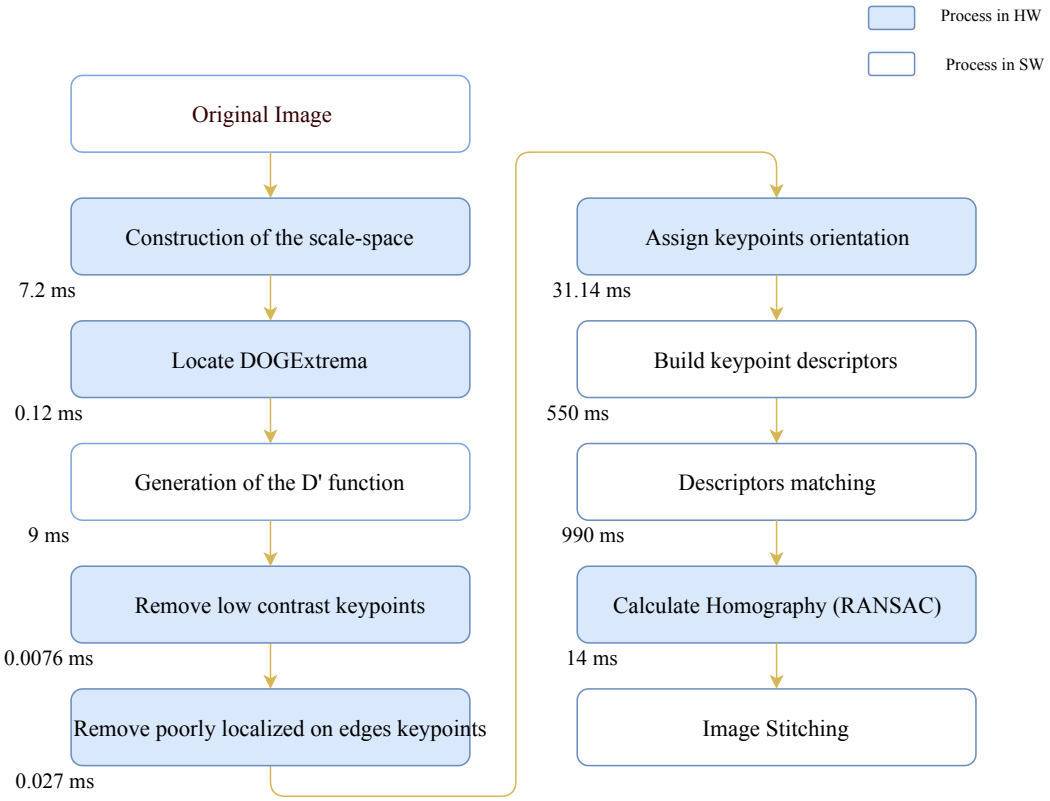


Figure 4.8: Image processing flow with latency.

Chapter 5

Conclusions

This chapter provides some conclusions on the research. The limitation of the design will be discussed in Section 5.1, whereas 5.2 discusses the future work that will be done to improve the quality of the research.

5.1 Conclusions and Limitations

By looking at the results obtained it is clear that, in terms of algorithm latency, at the current project status it is not possible to use this system for a real-time application. The result obtained $Latency_{overall} = 1.6s$ is far away from the desirable latency that a real time system should have $\frac{1}{FR_{desired}} = \frac{1}{30FPS}$, hence $Latency_{desired} \approx 33ms$, where FR stands for frame rate.
Figure 4.8 depicts the flow as well as the latency associated to each module.
It comes out that the SW implemented blocks are not allowed in this design since their latency drastically decreases the system's performances.
However, by focusing the attention on the HW processed still a long latency comes out, in particular for the *Assign Orientation* module, the *scale-space construction* module and the *homography evaluator*.
In the future, a pipelined version of the whole algorithm has to be implemented, therefore the system's throughput will address the problems related to the real-time processing for the system target, but keeping a short latency as well to let the design be useful for the rear-

view generation purpose. Problems grow up when a shorter latency is needed in critical applications, such as aerospace. In that case more improvements are needed to adopt the algorithm proposed for those applications.
In the car case, despite the latency can not allow steady stream, it is enough short to avoid accidents and safety problems.

5.2 Future Work

First, the work will be completed by implementing the last blocks missed in the algorithm: *D' generation*, which should be just an FSM that controls 2D convolution modules, as described in Section 3.2.2 of chapter 3, the *matching* module and the *build descriptors* module
More parallelism will be added for the scale-space construction to further speed-up the performances. Besides parallelizing for multiple pixels and multiple scales, each pixel will be parallelized: *kernel_size* operations will be performed in less then *kernel_size* clock cycles. In this case, a pipelining overhead is expected when the parallelism grade is bigger then *kernel_size* that is being evaluate.
A ping-pong buffer for the sckratchpad SRAM will be implemented in order to by-pass the DRAM memory accesses [22].
This will lead to performances improvement both for the scale-space construction and for the *Assign Orientation* module that spend a lot of time bringing data from a memory to the other one. Further the *atan* module could be exploited to avoid post processing on orientation evaluation, due to its granularity.
Moreover, since the RANSAC algorithm has still a long critical path, it will be speeded-up by heavily pipelining the units inside the module. It will be also parallelized by reducing the number of iterations, such that $N_{new} = \frac{N}{p}$, where N_{new} is the number of iterations after the parallelization and p is the parallelism grade.
Further, before the scale-space construction, the image will be interpolated, by increasing the image size by a factor 2 before the filtering process, i.e. the robustness will be increased and in turn the final result will enhance [7].
Moving Direct Linear Transform (DLT) and alpha blending algorithms will be developed for removing local problems of the stitched image, such as parallax [18].

A physical synthesis will be done to figure out the interconnections contribute and extract more accurate results.
The final ASIC result will be benchmarked against a Convolutional Neural Network (CNN) version of the same algorithm.
At the end of the research the system will be then mapped on coarse grain reconfigurable building blocks called SiLago blocks [23].

Appendix A

Unnecessary Appended Material

Listing A.1: Example of a script used for synthesis

```
set designer "Farina Marco"
set company "KTH"
set SynopsysHome "/afs/kth.se/pkg/vol/contrib/ict/nveg/
   pkg/vol2/synopsys/syn_M-2016.12"
set outfname "RANSAC"
set version "3.31"
####### Set Directory #########
set SYNDIR ../EXE
set OUTDIR ../OUTPUT
set SYNDB  ../EXE
set RPTDIR ../REPORTS
######Enviroment############
define_design_lib work -path $SYNDIR/work

set write_name_nets_same_as_ports "true"

#set sdc_write_unambiguous_names false
######Libary#################
# Minimal .synopsys_dc.setup file
set search_path "$search_path  /mnt/storage/stathis/
   Documents/silago/Libs/tcbn40lpbwp_120b/FrontEnd/
   tcbn40lpbwp_120b /afs/kth.se/pkg/vol/contrib/ict/nveg
   /pkg/vol2/synopsys/syn_M-2016.12/libraries/syn"
set synlib "/afs/kth.se/pkg/vol/contrib/ict/nveg/pkg/
   vol2/synopsys/syn_M-2016.12/libraries/syn/
   dw_foundation.sldb"
```

```
set target_library tcbn40lpbwptc.db
set link_path "$search_path"
set synthetic_library "$synthetic_library $synlib /afs/
   kth.se/pkg/vol/contrib/ict/nveg/pkg/vol2/synopsys/
   syn_M-2016.12/libraries/syn/standard.sldb"
set link_library "* $target_library $synthetic_library"

define_design_lib WORK -path ./SYN/WORK

set_clock_gating_style -sequential_cell latch

##
analyze -library WORK -format vhdl {/afs/kth.se/home/m/f
   /mfarina/rear_view_system_new/synth/RANSAC/SOURCE/
   misc.vhd}
analyze -library WORK -format vhdl {/afs/kth.se/home/m/f
   /mfarina/rear_view_system_new/synth/RANSAC/SOURCE/
   myPackage.vhd}
analyze -library WORK -format vhdl {/afs/kth.se/home/m/f
   /mfarina/rear_view_system_new/synth/RANSAC/SOURCE/
   lfsr.vhd}
analyze -library WORK -format vhdl {/afs/kth.se/home/m/f
   /mfarina/rear_view_system_new/synth/RANSAC/SOURCE/
   Rand_gen.vhd}
analyze -library WORK -format vhdl {/afs/kth.se/home/m/f
   /mfarina/rear_view_system_new/synth/RANSAC/SOURCE/
   CU_EvDist.vhd}
analyze -library WORK -format vhdl {/afs/kth.se/home/m/f
   /mfarina/rear_view_system_new/synth/RANSAC/SOURCE/
   CU_QRfact.vhd}
analyze -library WORK -format vhdl {/afs/kth.se/home/m/f
   /mfarina/rear_view_system_new/synth/RANSAC/SOURCE/
   CU_SVD.vhd}
###analyze -library WORK -format verilog {/afs/kth.se/
   home/m/f/mfarina/rear_view_system_new/synth/RANSAC/
   SOURCE/DW02_prod_sum1.v}
###analyze -library WORK -format verilog {/afs/kth.se/
   home/m/f/mfarina/rear_view_system_new/synth/RANSAC/
   SOURCE/DW_div.v}
#analyze -library WORK -format verilog {/afs/kth.se/home
   /m/f/mfarina/rear_view_system_new/synth/RANSAC/SOURCE
   /DW_div_pipe.v}
```

```
#analyze -library WORK -format verilog {/afs/kth.se/home
   /m/f/mfarina/rear_view_system_new/synth/RANSAC/SOURCE
   /DW_sqrt.v}
#analyze -library WORK -format verilog {/afs/kth.se/home
   /m/f/mfarina/rear_view_system_new/synth/RANSAC/SOURCE
   /DW_sqrt_pipe.v}
analyze -library WORK -format vhdl {/afs/kth.se/home/m/f
   /mfarina/rear_view_system_new/synth/RANSAC/SOURCE/
   DP_SVD.vhd}
analyze -library WORK -format vhdl {/afs/kth.se/home/m/f
   /mfarina/rear_view_system_new/synth/RANSAC/SOURCE/
   RANSAC_ctrl.vhd}
analyze -library WORK -format vhdl {/afs/kth.se/home/m/f
   /mfarina/rear_view_system_new/synth/RANSAC/SOURCE/
   RANSAC.vhd}
elaborate RANSAC -architecture BEHAVIOUR -library WORK
set_wire_load_mode top
set_wire_load_model -name ZeroWireload
set_operating_conditions NCCOM
create_clock -name "clock" -period 5 -waveform {0 2.5} {
   clock}
set_false_path -from [get_port rst_n]
compile -map_effort high
write -hierarchy -format ddc -output /home/m/f/mfarina/
   rear_view_system_new/synth/RANSAC/RANSAC.dcc
write -hierarchy -format verilog -output ./RANSAC.v
report_constrains > ./REPORTS/constraint_RANSAC.rep
report_cell > ./REPORTS/cell_RANSAC.rep
report_area > ./REPORTS/area_RANSAC.rep
report_timing > ./REPORTS/timing_RANSAC.rep
report_power -analysis_effort low > ./REPORTS/
   power_RANSAC.rep
```

Bibliography

[1] Jeremy Laukkonen, *Advanced Driver Assistance Systems*. March 23, 2018.
Available: https://www.lifewire.com/advanced-driver-assistance-systems-534859

[2] Renesas, *Surround view*.
Available: https://www.renesas.com/en-us/solutions/automotive/adas/surround.html

[3] Thomas Burger, *How Fast is Realtime? Human Perception and Technology*. February 9, 2015. PubNub.
Available: https://www.pubnub.com/blog/2015-02-09-how-fast-is-realtime-human-perception-and-technology/

[4] Richard Szeliski, *Image Alignment and Stitching: A Tutorial*. January 26, 2005. Microsoft Research.
Available: https://courses.cs.washington.edu/courses/cse576/05sp/papers/MSR-TR-2004-92.pdf

[5] Ebrahim Karami, Siva Prasad, Mohamed Shehata, *Image Matching Using SIFT, SURF, BRIEF and ORB: Performance Comparison for Distorted Images. Faculty of Engineering and Applied Sciences, Memorial University, Canada*.
Available: https://arxiv.org/pdf/1710.02726.pdf

[6] Herbert Bay, Andreas Ess, Tinne Tuytelaars, Luc Van Gool, *Speeded-Up Robust Features (SURF)*. 31 October 2006, ETH Zurich, K.U. Lewen.
Available: https://www.sciencedirect.com/science/article/pii/S1077314207001555

[7] David G. Lowe, *Distinctive Image Features from Scale-Invariant Keypoints*. January 5, 2004, Univeristy of British Columbia Vancouver.
Available: https://people.eecs.berkeley.edu/~malik/cs294/lowe-ijcv04.pdf

[8] Chris Harris, Mike Stephens, *A combined Corner and Edge detector*. 1988, Plessey Research Roke Manor.
Available: http://www.bmva.org/bmvc/1988/avc-88-023.pdf

[9] P M Panchal, S R Panchal, S K Shah, *A Comparison of SIFT and SURF*. April 2 2013, International Journal of Innovative Research in Computer and Communication Engineering.
Available: http://www.ijircce.com/upload/2013/april/21_V1204057_A%20Comparison_H.pdf

[10] Scale space.
Available: http://www.cs.uu.nl/docs/vakken/ibv/reader/chapter9.pdf

[11] Krystian Mikolajczyk, Cordelia Schmid, *Scale & Affine Invariant Interest Point Detectors*. Accepted January 22, 2004, INRIA Rhne-Alpes GRAVIR-CNRS.
Available: https://www.robots.ox.ac.uk/~vgg/research/affine/det_eval_files/mikolajczyk_ijcv2004.pdf

[12] M. Hassaballah, Aly Amin Abdelmgeid and Hammam A. Alshazly, *Image Features Detection, Description and Matching*. Springer.
Available: https://pdfs.semanticscholar.org/322f/f387087134ac776a4270cd55e7f3334edeb2.pdf

[13] Marius Muja, Member, IEEE and David G. Lowe, Member, IEEE, *Scalable Nearest Neighbor Algorithms for High Dimensional Data*. November 2014.
Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6809191

[14] Elan Dubrofsky, *Homography Estimation*. Carleton University, 2007.
Available: https://www.cs.ubc.ca/grads/resources/thesis/May09/Dubrofsky_Elan.pdf

[15] Thomas Opsahl, *Estimating homographies from feature correspondences*. UNIK.
Available: Estimating homographies from feature correspondences

[16] Alan Kaylor Cline, Inderijt S. Dhillon. Computation of the Singular Value Decomposition. Univeristy of Texas at Austin.
Available: http://www.cs.utexas.edu/users/inderjit/public_papers/HLA_SVD.pdf

[17] Jim Lambers, *The QR Factorization*. Summer Session 2009-10. The University of Southern Mississippi. Available: http://www.math.usm.edu/lambers/mat610/sum10/lecture9.pdf

[18] Vishnu Sharan Chandrakar, *Design of a rear-view generation for the driver assistance system*. Department Of Integrated Electronics and Circuits Engineering Indian Institute of Technology Delhi, June 2017.

[19] Syed M. A. H. Jafri, Ahmed Hemani, Kolin Paul, and Naeem Abbas, *MOCHA: Morphable locality and compression aware architecture for convolutional neural networks*. 2017 IEEE International Parallel and Distributed Processing Symposium.

[20] Song Ho Ahn, *Proof of Separable Convolution 2D*.
Available: http://www.songho.ca/dsp/convolution/convolution2d_separable.html

[21] Khanh Van Mai. *High-Performance Fixed-Point Architecture for Financial Application.* Jenuary 2006.
Available: file:///Users/marcofarina/Downloads/IntershipReport.pdf

[22] C.Zinner, W.Kubinger. *ROS-DMA: A DMA Double Buffering Method for Embedded Image Processing with Resource Optimized Slicing*. 24 April 2006, IEEE.
Available: https://ieeexplore.ieee.org/document/1613350/

[23] Ahmed Hemani, Syed Mohammed Asad Hassan Jafri, Shayesteh Masoumian. *Synchoricity and NOCs could make Billion Gate Custom Hardware Centric SOCs Affordable*.
Available: https://dl.acm.org/citation.cfm?id=3132339