People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
University M'Hamed BOUGARA – Boumerdes

Institute of Electrical and Electronic Engineering
Department of Electronics

Final Year Project Report Presented in Partial Fulfilment of
the Requirements for the Degree of

MASTER

In **Electronics**

Option: **Computer Engineering**

Title:

FPGA Implementation of Stereo Matching Algorithm for Depth Estimation

Presented by:

YAHIA Karim

Supervisor:

Pr. KHOUAS Abdelhakim

Registration Number:...../2022

Abstract

Computer vision is an artificial intelligence branch developed for machines to perceive image and videos. It interprets visual data (pictures or videos) to extract information. One of its fundamental concepts is defined as stereo vision; used to estimate 3-D information of the scene. During depth estimation, a process denoted stereo matching is considered as the complex part; it requires a considerable amount of time to execute on a processor which prevents the system to reach its minimum speed (30 frames per second). The proposed solution is moving the complex part of the system to hardware, since the latter is faster than software. To implement stereo matching, different methods and algorithms are proposed, however, only few are suitable for hardware implementation. In this project, a correlation-based on Rank Transform and Sum of Absolute Differences algorithm is implemented on a FPGA; the minimum speed reached by this module is 301 frames per second for the VGA format images.

Dedication

To my parents

To the best brother ever

To my aunt Samra and my uncle Djamel

To my grandparents

To all my aunts, my uncles and my cousins

To all my friends specially Ania and Mazigh

Acknowledgement

To begin with, I would like to express my sincere gratitude and respect to my supervisor Pr. A. KHOUAS for assisting my work and guiding me throughout the process.

Special thanks to my colleague M. BENDAHMANE, and to T. SAIDI for their great help and assistance.

I want also to thank all my friends in the institute for their support and encouragement during these six years, especially Mazigh who always helped me with my studies.

Last but not least, I want to thank my parents and brother, for their presence, their encouragement, for providing me all the necessary materials to succeed in my studies and supporting me all these years; THANK YOU.

Table of contents

List of figures

List of tables

List of abbreviations

BRAM:	Block Random Access Memory
CLBS:	Configurable Logic Blocks
CP:	Central Pixel
CT:	Census Transform
DM:	Disparity Map
DSP:	Digital Signal Processing
FF:	Flip Flop
FPGA:	Field Gate Programmable Array
HDL:	Hardware Description Language
HM:	Hamming Distance
Hsyn:	Horizontal Synchronization
IOBs:	Input Output Blocks
IP:	Intellectual Property
LIDAR:	Light Detection and Ranging
LUT:	Lock Up Table
RADAR:	Radio Detection and Ranging
ROM:	Read Only Memory
RT:	Rank Transform
SAD:	Sum of Absolute Differences
SSD:	Sum of the Squared Diffrences
VGA:	Video Graphics Array
Vsyn:	Vertical synchronization

General introduction

Stereo vision is a fundamental concept of computer vision, its principal is depth estimation of the captured scene and is driven from human vision, which is similar to all computer vision concepts. Humans have the ability to perceive depths instantly, just by starring at the surrounding objects; implementing this sense to automatic machines would be a perfection. After development of medical field, the secret on how humans perceive depth was found and it is defined as binocular vision. The first step to copy this sense is using two cameras to replace the human eyes. Using stereo vision, we can calculate 3-D information of the environment to detect obstacles and for other purposes.

The software implementation of the obstacle detection system based on stereo vision in [1] shows the inefficiency of the software design; the system does not reach the minimum speed of a stereo vision system (30 frames per second). A process denoted stereo matching, which is responsible of matching the images captured by the two cameras, was the source of the problem. Because of the limitation of software solutions (lack of parallelism), the alternative suggested was trying to overcome this difficulty using hardware, since the latter has a higher processing speed and can be combined to software easily. In order to test the proposed solution, a stereo matching part of the system is implemented and tested on an FPGA (Field Programmable Gate Array).

Stereo matching is widely known as the challenging part of a stereo vision. It matches two 2-D images to calculate another 2-D image that has 3-D information of the scene. To achieve this objective, different algorithms and methods were proposed to solve stereo matching problem. However, they are different in term of the processing speed, the result quality and even complexity. In this project, we implemented stereo matching on hardware (FPGA). This report is divided into three chapters. The first one describes the theoretical part of stereo matching. Chapter two describes the hardware design. The last one explains the implementation process and the obtained results.

Chapter I. Stereo vision principal

I.1. Introduction

Due to our vision ability, humans can perceive not only the color, the form, and the status of the object, but also the dimensions and distances of the environment and its contents, allowing us to interact and react with our environment. Camera is used to copy this sense, it takes images showing information related to the scene, however, a single camera cannot capture 3-D information. After medical research, an explanation of how human perceive depth was found and defined as Binocular vision, commonly known as stereopsis. Humans can perceive depth more accurately because the two eyes provide two different views of the world [2], otherwise, the depth perception will be less accurate. In this chapter, we present the stereo vision and its principal, we will focus more on the stereo matching and its complexities limitation and proposed algorithms.

I.2. Computer vision

Computer vision is an interdisciplinary field concerned with developing digital systems that can process, interpret, and comprehend visual input (images or videos), the challenge is to make these systems perceive data as a human visual system can do. Computer vision is predicated on teaching computers how to interpret and understand images at the pixel level. Technically, machines use sophisticated algorithms to retrieve visual input, manage it, and interpret the results [3]. The common functions of computer vision system are:

- Object classification.
- Object identification.
- Object tracking and detection.

Our work is related to the third function, how we can retrieve 3-D information for an obstacle detection system.

I.3. Binocular vision explanation

What we see is the result of signals sent from the eyes to the brain. Normally, both (bi) eyes (ocular) send messages to the brain at the same time. The information in each eye signal is slightly different, and the brain may use these differences to estimate distances and coordinate

eye movements when binocular vision is functioning properly, which allows us to perceive depth and distances between objects [2]. Each eye sees slightly different view on the scene then it transmits these details to the brain. The brain judges the distance and the depth based on the differences between the two eyes. The ability to see the third dimension and discriminate between objects relationships is the outcome.

I.4. Depth perception techniques

To perceive the third dimension of the scene or the surrounding objects, various methods can be used; they are generally classified into two categories: active and passive approaches. Active approach uses sensors, detectors, and lasers. However, passive approach uses a charge-coupled device (camera).

I.4.1. Active approach

Active approach is based on systems which are made of a transmitter and a receiver. The transmitter sends out electromagnetic waves for Radio Detection and Ranging (RADAR), sound wave for Ultrasonic sensors or pulse light wave for Light Detection and Ranging sensor (LIDAR), these signals or waves travel the surrounding then return to the sensor at their receiver. By considering the time taken between the sending and receiving process, the distances traveled can be calculated.

Radars are capable of operating in cloudy weather conditions, and have a long operating distance. However, they struggle when the objects are close to the sensors, and they cannot resolve multiple targets. Lidars are very accurate and have a consistent result, they can detect even small objects and create the exact 3-D model of the scene. Lidars are limited in dark and bad weather conditions. Ultrasonic sensors are not affected by weather conditions, they have high frequency and sensitivity, they can be easily interfaced with a microcontroller. However, they have difficulties in reading reflections from small objects or soft, curved, and thin surfaces as well as their sensitivity to temperature variation.

I.4.2. Passive approach

Passive systems are different from active ones, they don not transmit any signal to the scene. Passive systems are constituted with two or more cameras; they capture the objects present in that environment. Since this model is working with images, we can have a fast

observation of the scene with high precision. Also, these systems avoid sensor/sensor or sensor/environment deduction problems. The cons of this approach reside in the outside environment, the extreme weather (fog, rain, snow…etc.) can really affect the image quality, nights also can affect the results because of the low of brightness. The main foundation of this approach is the algorithm part that processes the images to finally get a map having the desired information.

I.5. Stereo vision systems

Stereo vision is one of the most important topics of computer vision research, its objective is the reconstruction of a scene's 3D structure using two or more pictures of the scene. The most used system is stereo vision binocular, it is inspired from the human vision. This system is constructed of two cameras that are placed close to each other; however, they have a different view on the scene because of their angles. The principal of a stereo vision system is detecting 3-D point that is present in two 2-D images, this point lines along the epipolar line as we see in figure I.1. By reading and processing these two images referred as left and right images, which are taken by these two cameras at the same instant of time, it is possible to reconstruct a 3-D model of the scene. The processing is divided into five stages [4]: image acquisition, camera calibration, image rectification, stereo correspondence and 3-D reconstruction.

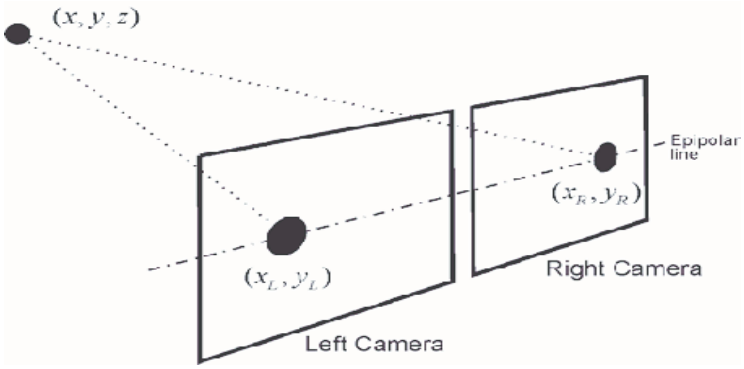


Figure I.1: Stereo vision principal, projection of 3-D point to two 2-D images [5].

I.5.1. Image acquisition

Camera is used to capture images, there are two types of cameras: analog and digital cameras. The digital ones are better; because they are less sensitive to noise, and can capture a larger resolution. To acquiree images we can use one or multiple cameras, a typical stereo vision system uses two cameras.

I.5.2. Camera calibration

Camera calibration the process of finding the geometrics transformations to project a 3-D point present in the scene into 2-D point in the image. It is dependent of two parameters: intrinsic and extrinsic. The former is related to the internal architecture of the camera (its sensor and its focal length), the latter defines the location and the orientation of the camara to the objective captured.

I.5.3. Image rectification

Image rectification the process of projecting the two stereo images (left and right images) onto a common plane in order that the resulting images are row aligned and the points in the left picture of one row to be in the same row of the right picture as shown in figure I.1. The objective of this process is simplifying the stereo matching process. Since two corresponding points lie on the same horizontal line; the matching search will be performed only along the epipolar line.

I.5.4. Stereo matching

Stereo matching denoted also as stereo correspondence is the process of matching two 2-D points present on the stereo images which are the projection of the same 3-D point as shown in figure I.1. After image rectification, two corresponding points will lie on the same epipolar line; in other words, they have the same vertical coordinates.

I.5.5. 3-D reconstruction

The shape of objects within the scene and their appearance can be reconstructed from the 2-D stereo images after the matching process. A 3-D model will be built; it is denoted as the Disparity Map (DM). This model can be used for obstacle detection purposes.

I.6. Disparity and disparity map

Disparity is defined as the horizontal difference between two corresponding pixels that lie on the same epipolar line [4]. It is inversely proportional to distance [4]; when an object is closer to a camera, its corresponding disparities will be higher, as shown in figure I.2. The sculpture and the lamp have higher intensity compared to the rest objects. The matching process

is calculating disparities of all pixels within an image. The resulting disparities will be saved in a new image known as DM. The DM is a 2-D map that has all the disparities; hence it has the 3-D information of the scene, as shown in figure I.2. The depth can be calculated based on the DM [4]. Using the DM, it is possible to detect obstacles present in that environment.

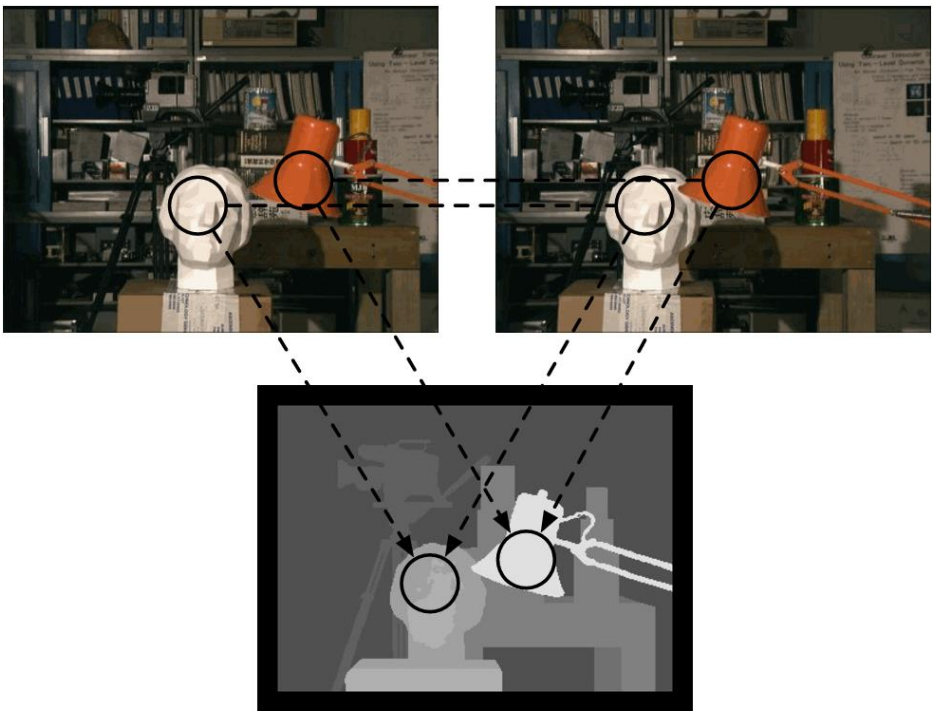


Figure I.2: Stereo images and their corresponding ideal DM [6].

I.7. Stereo matching complexity

Stereo matching is the complex task in a stereo vision system. It is even called stereo matching problem. The reason is due to the difference in the view point of the stereo images; for the lamp in the figure I.2, in the right image it is closer to the sculpture compared to the left. To overcome this difficulty, various methods and algorithm were proposed; they are all different in term of complexity, computational intensity also the resulting DM. Two methods are widely used: feature-based matching and correlation-based matching.

During the matching process, only the true matches between two pixels must be selected: they should respect a set of constraints [7]:

- **Epipolar:** Given a pixel in one image, the corresponding pixel in the second image must lie along the epipolar line, the figure I.3 shows this constraint.
- **Similarity:** The features or properties of matched pixels must be similar.
- **Ordering:** For relevant matches, the relative location between two pixels in one

image is kept in the other image.

- **Uniqueness:** Each pixel in one image should correspond to a distinct pixel in the second.
- **Disparity Range:** This constraint is used when we know the limitation of the scene; when the closet and the farthest distance from the view are available, we can set this range because disparity is inversely proportional to distance.

If two pixels are similar but they don't respect the previous constraints, they will be considered as false matches; since they do not have the same source.

I.7.1. Feature-based matching

The principal of this method is searching for common features in the left and the right images. The first phase in feature-based stereo matching is termed pre-processing, in which the features that are stable when the perspective changes are retrieved. The second step is to apply matching algorithms to these features. Edges, corners, line segments, and curve segments are among the characteristics retrieved, and they are stable and do not alter with the camera's location. Edges and corners are quick to identify but suffer from occlusion, lines and curves take longer to compute, and additional features like circles, ellipses, and polygonal areas are only available in interior situations. When a prior knowledge about the scene is known, feature-based algorithms are appropriate since the best feature may be employed. Another benefit of these methods is that they are generally unaffected by the changes in lighting and highlights. The drawback of feature-based matching method is being hard to implement or debug, furthermore, the sparse maps produced by this approach may appear to be inferior to the DMs made by the correlation-based methodology [8]. This method is not examined in depth here.

I.7.2. Correlation-based matching

In this method, the matching is performed between windows of pixels. A squared window is constructed around the reference pixel; denoted as the correlation window, then it is compared to a similar window in the candidate image along the epipolar line and within the disparity range, as shown in figure I.3. At each displacement of the candidate window, a similarity measure (cost) is calculated, the index of the smallest cost will be considered as the disparity. The pros of this method it that it is easy to implement and generates dense DM. However, choosing the correlation window size is the challenging part. When the window is small, we will have some noise and the DM will not be distinctive. When the window is large,

we will have really clear DM, but a considerable resolution amount will be lost. Selecting window size is a delicate step and it takes different parameters in consideration like speed, resource usage and final results. This correlation-based matching method was chosen to implement the stereo matching module.

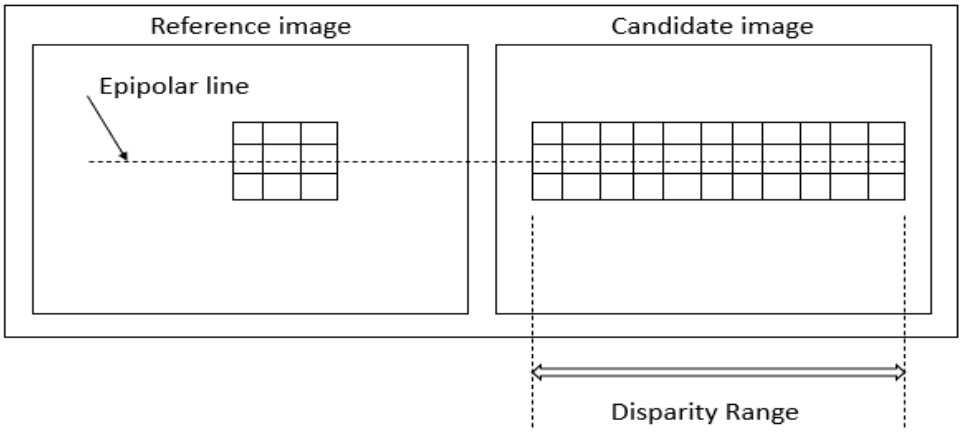


Figure I.3: Matching windows along the epipolar line.

I.8. Correlation-based stereo matching algorithms

Correlation-based stereo matching algorithms are more suitable for hardware implementation than the feature-based stereo matching algorithms [4]. They provide dense DM; this feature is important for 3-D reconstruction of the environment. The epipolar constraint reduces the search from 2-D to one dimension on the horizontal line, the disparity range constraint reduces the search into a small range, as shown in figure I.3; these two constraints increase the speed and the efficiency of the matching process.

I.8.1. Correlation metrics

Correlation metrics are used to measure the similarity (the cost) between a reference window and a candidate window; this cost is calculated for all candidate windows, then the minimum cost is selected as the match. In order to calculate similarity metric, several metrics are proposed, however, most of them are computationally intensive and does not suit hardware implementation. But there are two algorithms which are not computationally intensive: Sum of Absolute Differences (SAD) and the Sum of the Squared Differences (SSD). All the correlation metrics algorithms are sensitive to radiometric distortion (changes in illumination); because the cost is calculated using pixel intensity values.

I.8.1.1. Sum of absolute differences (SAD)

SAD is based on summation of absolute values, to calculate the similarity (cost) between a refence window and a candidate window, as shown in formula I.1, where **W** describes the window size, **u** and **v** denote the position of a pixels within a window, $\mathbf{I_1}$ and $\mathbf{I_2}$ are gray scale pixel values that represents left and right images, **(x, y)** are central pixel's coordinates (rows, columns), **(x + d, y)** stands for the correspond point in the candidate image along the epipolar line displaced by **d** [9]**.** SAD requires only subtraction and summation that does not require a huge number of resources.

$$\sum_{(u,v)\in W}|I_1(u+x,v+y)-I_2(u+x+d,v+y)| \qquad (I.1)$$

I.8.1.2. Sum of squared differences (SSD)

SSD algorithm is built on the Sum of the Squared Differences. The only difference between SSD and SAD is the way in which the similarity criterion is calculated. SAD is the sum of magnitude differences; however, SSD is the sum of squared differences values. Consequently, this difference requires more resources, since SSD uses summation and multiplication and SAD uses only summation [9].

$$\sum_{(u,v)\in W}(I_1(u+x,v+y\,)-I_2(u+x+d,v+y))^2 \qquad (I.2)$$

I.8.2. Census transform (CT)

CT is considered as non-parametric criterion [10]. CT basically is a mapping of window to a bit-vector, for a window W, its corresponding bit-vector length will be of W*W- 1. The bit-vector value is a result of comparison within a window between a central pixel (CP) and its neighbors; when the intensity of the neighbor pixel is less than the intensity of the central, its correspondent bit will be represented with a one, else it will be presented with a zero as we can see in the figure I.4; CP is presented in red color.

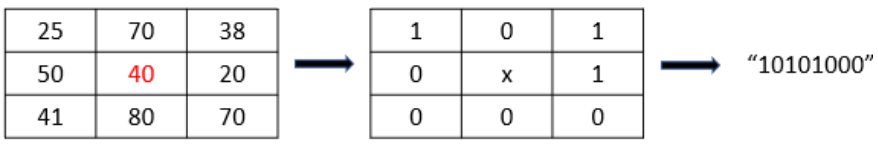


Figure I.4; Mapping a 3*3 window to a bit-vector using CT.

After performing CT to the stereo images (left and right images), the cost is calculated using a correlation metric; called Hamming Distance (HM). The HM calculates the number of bits that differ between the left and the right bit-vector as shown in figure I.5. The optimum match will be the one having the minimum HM value. Since each pixel is represented by a '0' or '1' for the cost estimation; the problem of the radiometric distortion is also solved.

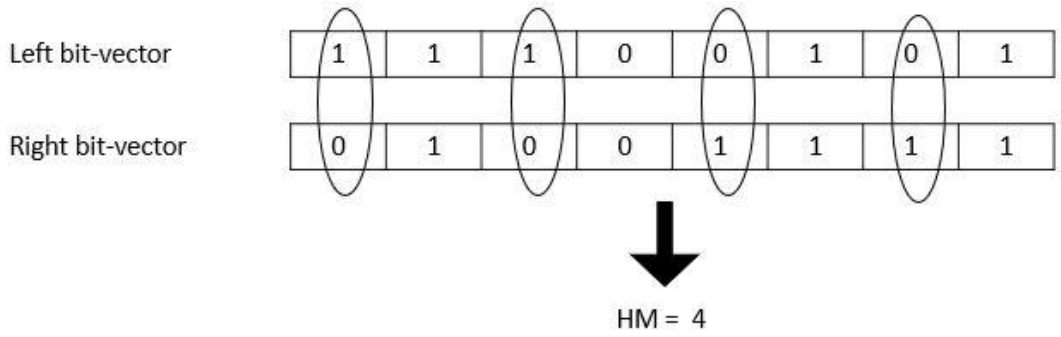


Figure I.5: HM calculation for the left and right bit-strings for a 3*3 window.

I.8.3. Rank transform (RT)

RT is also considered as non-parametric [10]. RT is similar to CT as it resists to radiometric distortion; however, it maps the window to a number, not to a bit-vector as CT. RT counts the number of the neighbor pixels that have less intensity than the CP, as shown in the figure I.6; CP is presented in red color. Afterwards, SAD is used to calculate the cost.

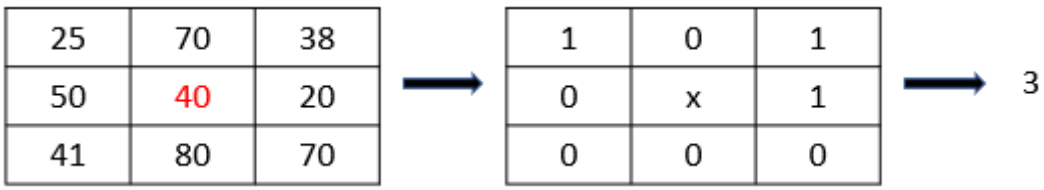


Figure I.6: Mapping a 3*3 window to a number using RT.

I.9. Stereo matching limitations

There exist some limitations when finding correspondence between two windows in stereo vision that are frequent and known, they are the sources of the mismatches. These limitations depend on different factors: object positions, lights in the scene, and the view point of the stereo camera. The first cause is occlusion, this phenomenon occurs frequently at objects boundaries, as shown in figure I.7.b; some points can be seen only from one camera. Also, a point can be captured by one camera and be covered or invisible for the second camera, as

shown in figure I.7.a, the red point can be seen only by the right camera. Also, when an object is parallel to one camera; for example, object ‘S’ for the left camera in figure I.7.a, that camera can capture only one face of that object, which is not the case for the other one, as you can see in figure I.7.a, the darkened regions are seen only from the right sensor.

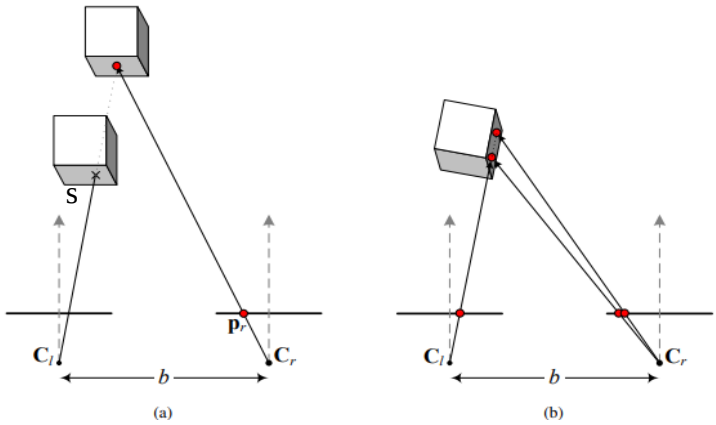


Figure I.7:Occlusion problem due to the view point [11].

Scene lighting also plays an important role in the stereo correspondence, the position of light is relative to objects, and perceiving light from different point of views is not similar. This phenomenon may cause some mismatching. Another source of mismatch is related to the point of view, some objects don not look similar from different angles, this dissimilarity increases when these objects are closer to the cameras, this phenomenon is known as object distortion.

I.10. Conclusion

In this chapter, we introduced stereo vision background, we discussed how it perceives depth or calculates 3D information of a scene, this concept is driven from human vision called binocular vision. The complex part in a stereo vision system is the stereo matching or stereo correspondence; the fundamental cause of this complexity is due to difference in viewpoints of the two cameras. Two methods were presented to perform the stereo matching: feature-based matching and correlation-based matching. The latter is more suitable for hardware implementation.

Chapter II. Hardware design of stereo matching module

II.1. Introduction

In the previous chapter, we discussed the theoretical concepts of stereo matching that matches between two 2-D images to construct a DM. Various methods were used to solve the stereo matching complexity. In this chapter, we will present the design of correlation-based algorithm for stereo matching using RT and SAD similarity criterion. We will explain the functionality of the different hardware implemented architectures, the advantage and disadvantages of each architecture.

II.2. Stereo vision system overview

In order to implement an obstacle detection system based on stereo vision, an image acquisition module that consists of two cameras is essential; it has the role of capturing stereo pictures of the scene then send them as pixels in series to the matching module. The matching module will receive those pixels and process them in real time, then send the estimated disparity values to the obstacle detection module, as shown in figure II.1. The matching module is just part of an obstacle detection system based on stereovision and it cannot be implemented alone. In this design, memories are used to emulate the cameras and to save the resulting DM.

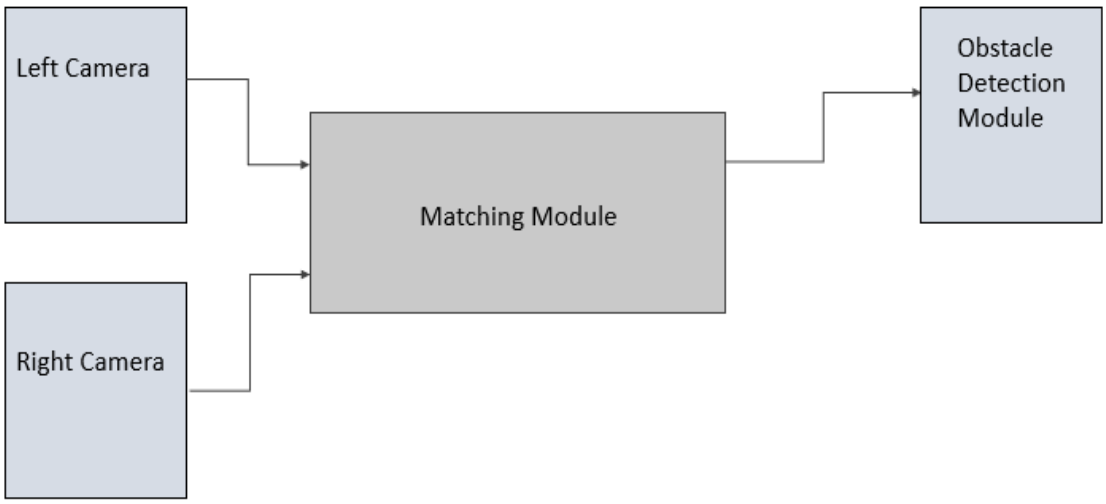


Figure II.1: Design of obstacle detection system based on stereo vision.

II.3. Matching module

The algorithm chosen to implement the matching module is the correlation-based method using RT and SAD; because it resists radiometric distortion and reduces the FPGA resource usage. Note that the RT reduces the resource usage only for 3*3 windows until 11*11 windows because the size of the generated bit-vector is less than eight bits. For 13*13 windows and more, the bit-vector generated by RT will be represented using 8-bits or more, which could be similar or greater to grayscale pixel length. The matching module design is divided into five blocks as shown in figure II.2. The first two blocks are used to calculate RT, they have similar architecture. The second block calculates the cost using SAD to calculate similarity criterions. The third block is used to find the smallest cost from all the costs generated by the second block, then it generates the disparity; which will be the index of the smallest cost. The Dsiaparity_Saver block is used for synchronization the design.

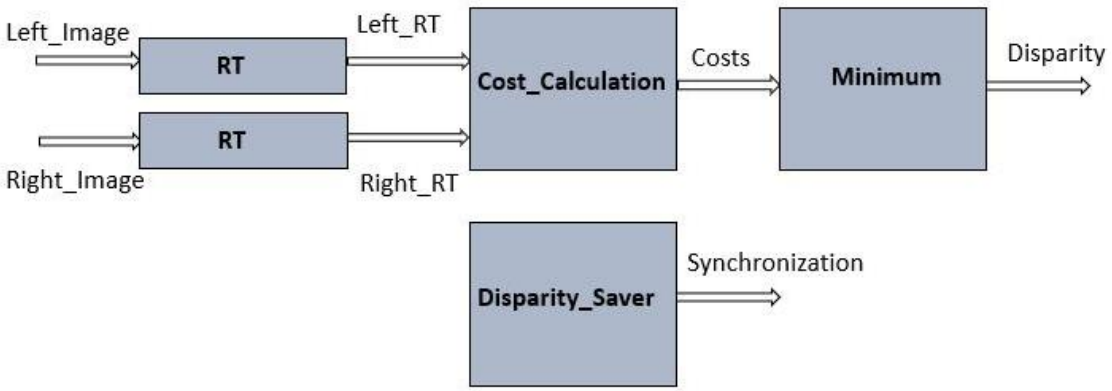


Figure II.2: Block diagram of the matching module.

II.3.1. RT block design

II.3.1.1. Shift register design

RT block receives a stereo image as an input, this image is received as pixels entering in series. As seen in the first chapter, RT is performed on window of pixels; these pixels are situated in different rows of an image, as shown in figure II.3. For 3*3 window, the first central pixel (CP) is P (2,2), the last CP is P (239.319). The pixels on boarders are avoided since they do not have enough neighbor pixels to form a window. In order to calculate RT of the first window, we need a storage space to save (320*2)+3 pixels. If it is 5*5 window, the first CP is P (3,3) and the last CP is P(238,318), the storage space should save (320*4)+5 pixels. For W*W

window, the storage space needed will be ((320*(W-1)) +W). If we replace 320 by a constant (Ncol), the size will be (Ncol*(W-1) + W); a shift register (SR) is used to save those pixels since it offers the possibility to access different data at the same time. As the pixels are entering in series row by row, P(1,1) will enter first the SR and P (3,3) will enter the last. When a next pixel in that row enters the SR all the pixels in the SR will be shifted by one position for example when P (3.4) enters SR, all the pixels will be shifted by one position; P (1.1) will leave SR, P (2,3) will be at the previous position of P (2.2). Therefore, CP and its neighbors positions are fixed in SR, because pixels are entering in series even the boarder pixels will occupy CP position at certain time; however, the results at that time will be avoided.

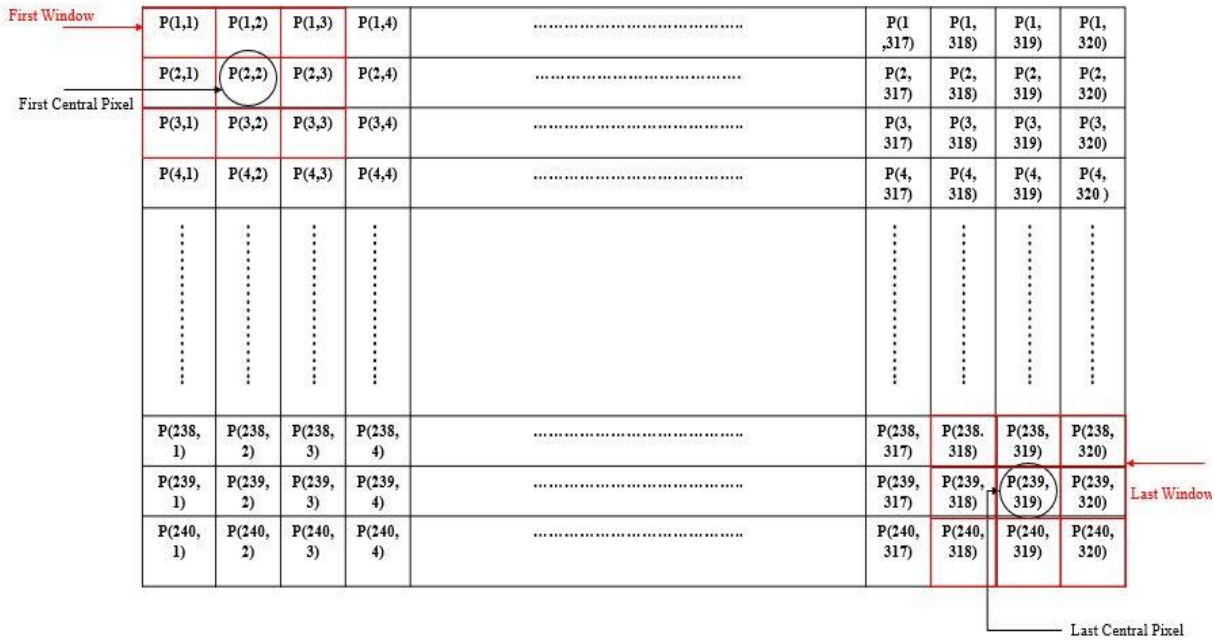


Figure II.3: Pixels arrangement of a 320*240 image.

II.3.1.2. Rank transform module

In the first chapter, we have seen how to calculate RT; basically, it counts the number of pixels that have less intensity than the CP within a window. In order to calculate RT, the hardware architecture in figure II.4 was proposed for 3*3 Window. Pix represents pixels in the window, Pix(5) is the CP. First, the comparison between Pix (5) and its neighbors is calculated in parallel; the comparator returns '1' when the neighbor is less than the CP or '0' when neighbor is greater or equal to CP. The registers are added to increase the processing speed. For a 3*3 window, the maximum RT result is 8 which can be written on 4 bits as '1000'. For this reason, a counter is initialized at zero as '0000' and three zeros are concatenated to the left of each comparison result in order to perform addition between the same number of bits. At

each stage of addition, the comparison results is added to the counter. The drawback of this architecture is noticed during the implementation process, it consumes more resources than required for such operation, also the propagation delay is high specially for larger windows.

Pix (1)	Pix (2)	Pix(3)
Pix (4)	Pix (5)	Pix (6)
Pix (7)	Pix (8)	Pix (9)

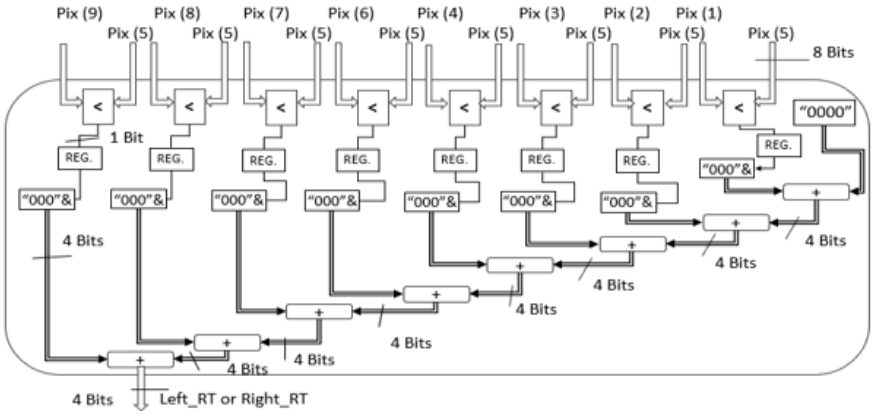


Figure II.4: First architecture proposed to calculate RT for 3*3 window.

To reduce resource usage and increase the processing speed, the hardware architecture in figure II.5 is designed for 3*3 Window to replace the one in figure II.4. First, the comparison runs in parallel as in the previous architecture. The addition runs in parallel too; the results of two comparators are summed together, then it continues summation as shown in figure II.5. The registers after the first addition are added to reduce the propagation delay, in order to increase the frequency.

Pix (1)	Pix (2)	Pix(3)
Pix (4)	Pix (5)	Pix (6)
Pix (7)	Pix (8)	Pix (9)

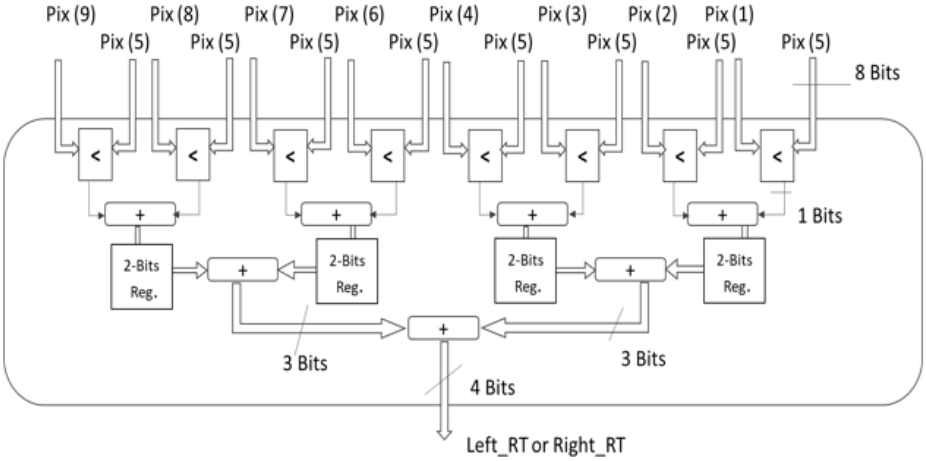


Figure II.5: The optimized architecture to calculate RT for a 3*3 window.

II.3.1.3. RT block architecture

The final architecture for RT block is given in figure II.6. The architecture is given for 3*3 window. The pixels from the image enters the SR in series; at each clock cycle, a new pixel

enters the SR, and its elements are shifted by one location. The window pixels location is fixed and its elements are presented as follow: Ncol + 2 is the location of the CP in SR, SR locations 1, 2, 3, (Ncol*2) + 1, (Ncol*2) + 2 and (Ncol*2) + 3 are the neighbor pixels and only those pixels enter the Rank Transform block. The internal architecture of the Rank Transform is given in figure II.5.-

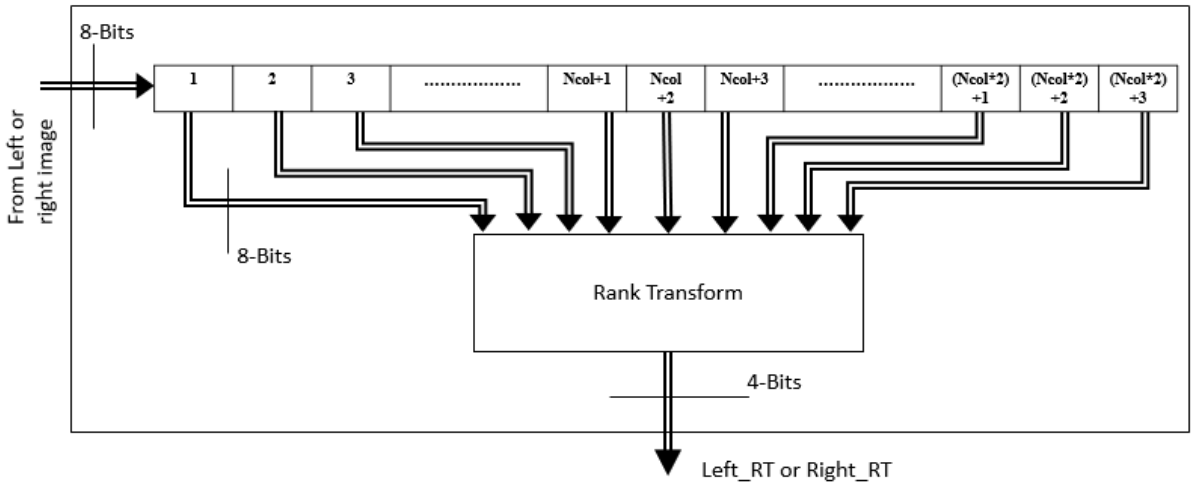


Figure II.6: Architecture of the RT block for 3*3 window.

II.3.2. Costs calculation block

This block calculates similarity criterion using SAD, in the first chapter, we have seen that the matching is performed between a reference window and its candidate windows; within the disparity range (Dmax). In this case, these windows are not constituted of pixels; the reference windows are constituted of Right_RT, and the candidate windows are constituted of Left_RT; which means the left image is used as a candidate.

II.3.2.1. Left shift register and right shift register modules

Left_RT and Right_RT values generated by the two RT blocks as shown in figure II.6 are streamed in series. To hold these data, two storage spaces (two SRs) must be constructed. Left_RT values will enter to the right shift register (RSR). Right_RT values will enter to the left shift register (LSR) as shown in figure II.7. We know that a reference window should be compared to all its candidate windows, to run this comparison in parallel, we must save all its candidate windows; in order to access them at the same time. Therefore, the size for the RSR (the one holding the candidate windows), should be equal to Ncol*(W-1) +W +Dmax; we added Dmax to have access to all the candidate windows in the disparity range simultaneously. The LSR size is Ncol*(W-1) +W +Dmax; Dmax is added because the first reference window in the

LSR should wait for its candidate windows in the RSR. Figure II.8 shows how RSR and LSR will be presented and organized for the next architectures.

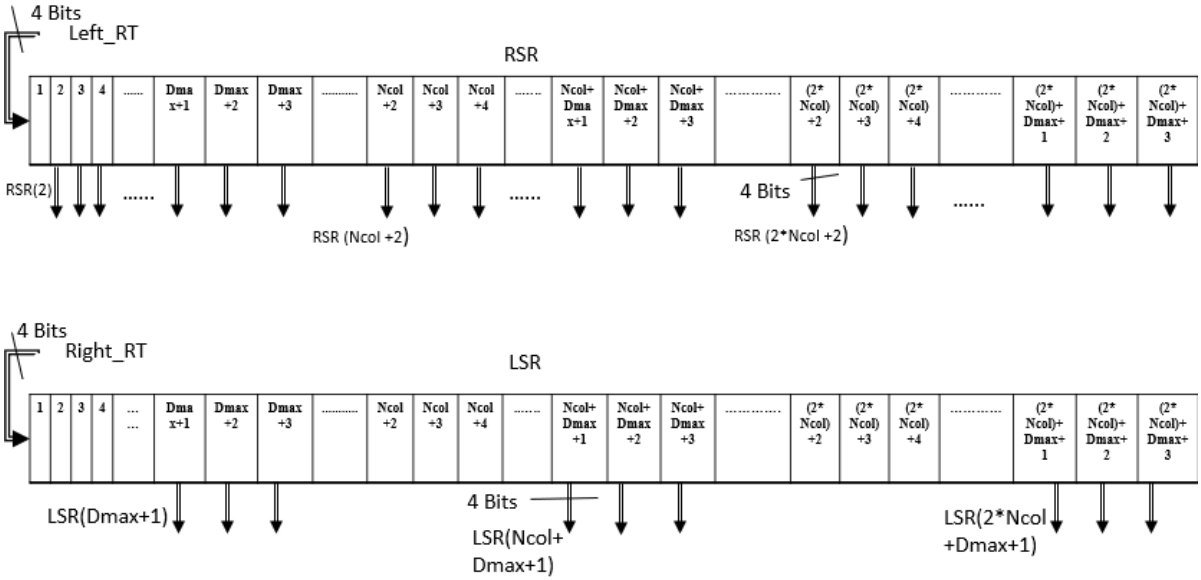


Figure II.7: Architectures of RSR and LSR for 3*3 window.

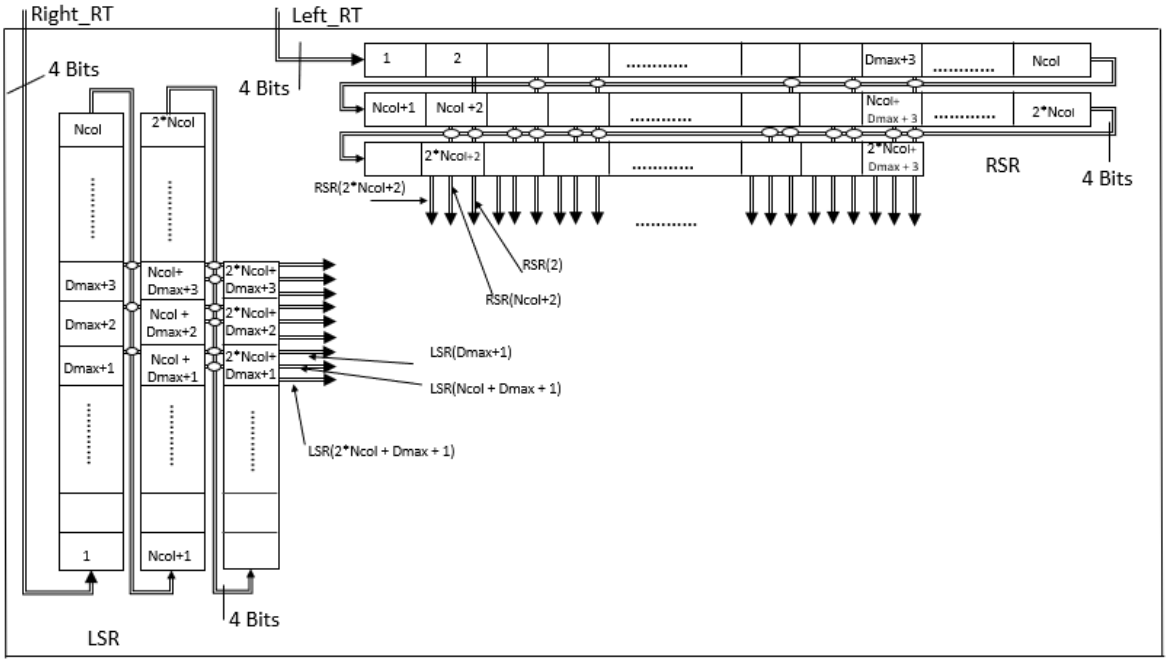


Figure II.8: Organization of LSR and RSR architectures for 3*3 window.

II.3.2.2. Sum of absolute differences module

SAD module receives its inputs in series from the LSR and RSR, these inputs may be considered as columns. For 3*3 window: RSR (2), RSR (Ncol+2) and RSR (2*Ncol+2) can be considered as the first column in RSR. LSR (Dmax +2), LSR(Ncol+Dmax+1) and LSR

(2*Ncol+Dmax+1) can be considered as the first column in the LSR, as shown in figure II.8. Three architectures were designed for SAD module. The first architecture is given in figure II.9. First, it performs absolute difference in parallel in order to calculate the cost. The absolute difference operation is performed on signed numbers; therefore, we concatenated a '0' to L and R to protect data from subtraction operation. A counter is set to zero, its length is equal to the expected length of SAD_R; at each stage, it will add a result of the absolute difference. The advantage of this design is easy to implement, however, it consumes more resources than required and its propagation delay will be high.

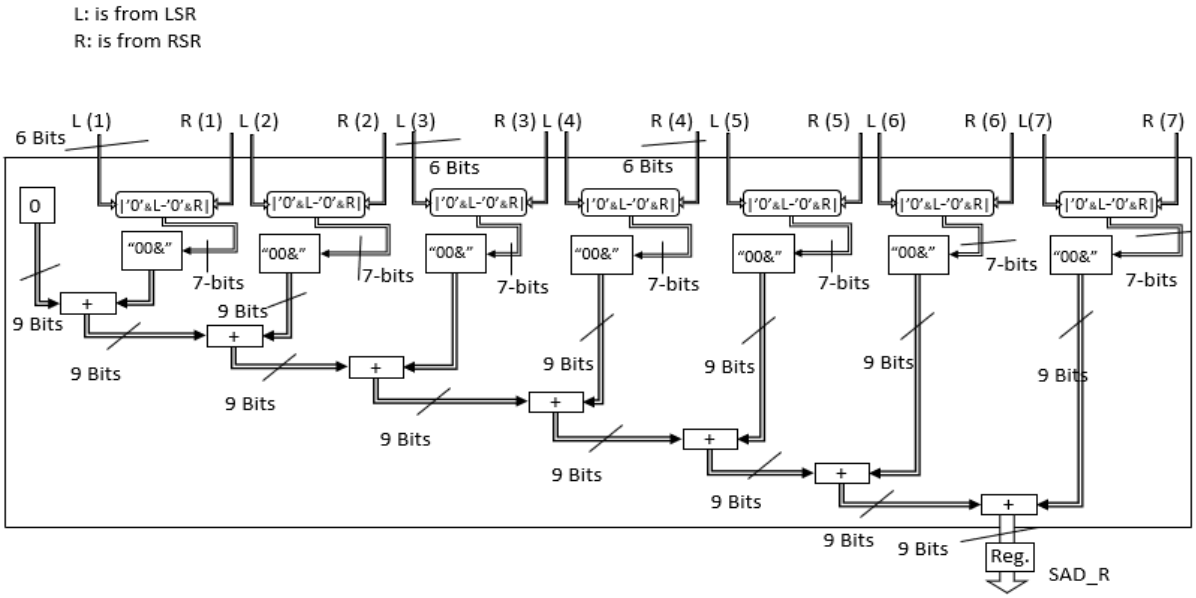


Figure II.9: First proposed architecture for SAD module.

A second architecture is proposed as shown in figure II.10; the absolute difference is calculated as the previous architecture. The first stage of addition is performed in parallel. Each two results of absolute difference are summed together. We can notice that two 7-bits vector are summed together and they result in 7-bits vector; the reason is they are considered both as unsigned vector and from absolute difference part that their MSB is a '0'. Then concatenation is performed to make the length of the absolute subtraction results equal to the SAD_R length. Afterwards summation is continued sequentially. The added registers are used to reduce the propagation delay and increase the speed. This architecture fits well 3*3, 5*5 and 7*7 windows and it is flexible. The drawback is that for 9*9 and 11*11 windows, the implementation consumes more resources, specially registers that are used to solve the delay problem.

A third architecture is considered as the optimized one, in term of the resources and the processing speed. All the processes run in parallel as shown in figure II.11. The only drawback

is the implementation, because each window should be implemented alone.

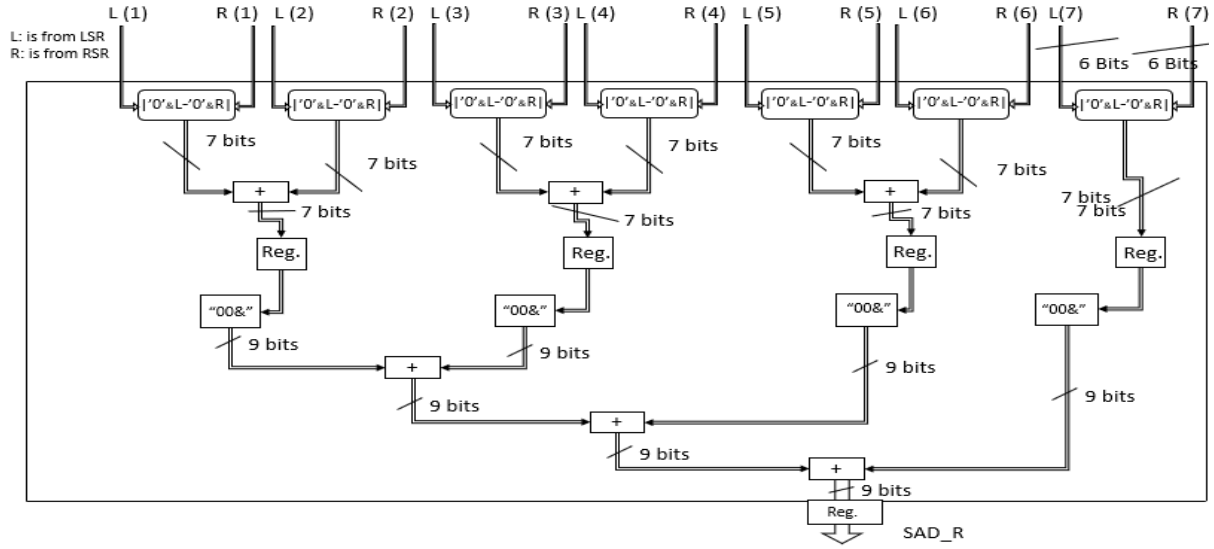


Figure II.10: The second proposed architecture for SAD module for 7*7 window.

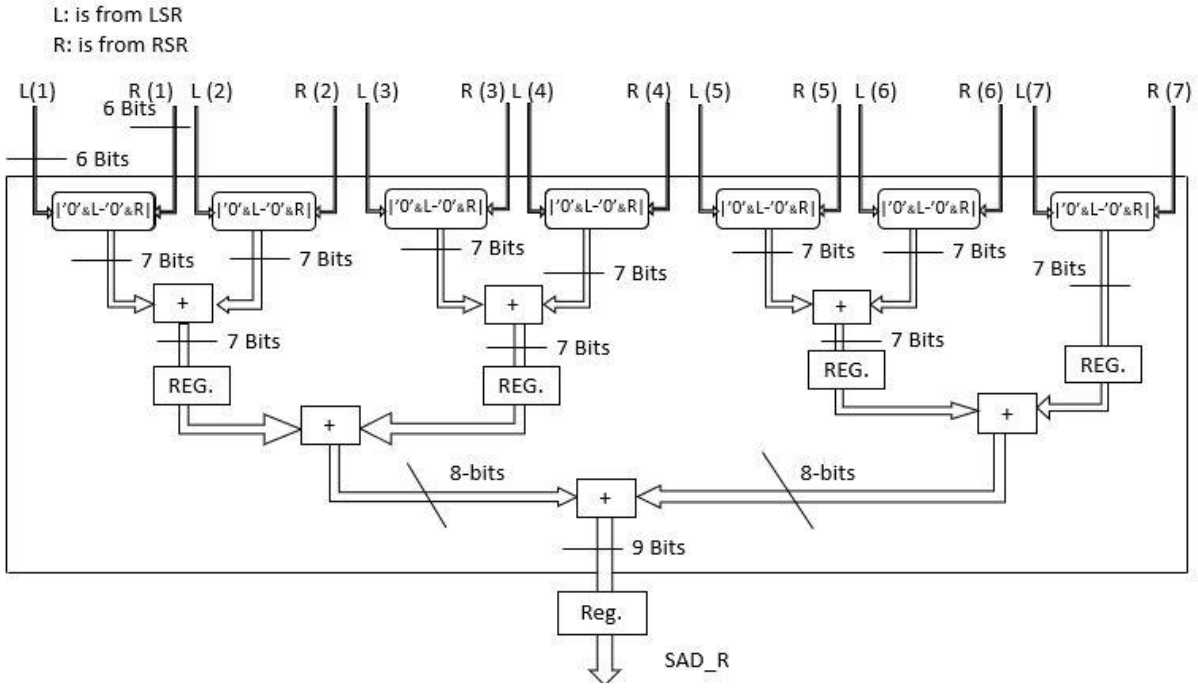


Figure II.11: The optimized architecture designed for SAD Module for 7*7 window.

II.3.2.3. Sum module

The Sum Module receives its inputs from different SAD blocks and performs summation of the inputs. Addition is performed in parallel for all stages as shown in figure II.12. Concatenation is performed to SAD_R (7), in order to perform addition between two 10-bits vectors in the second stage.

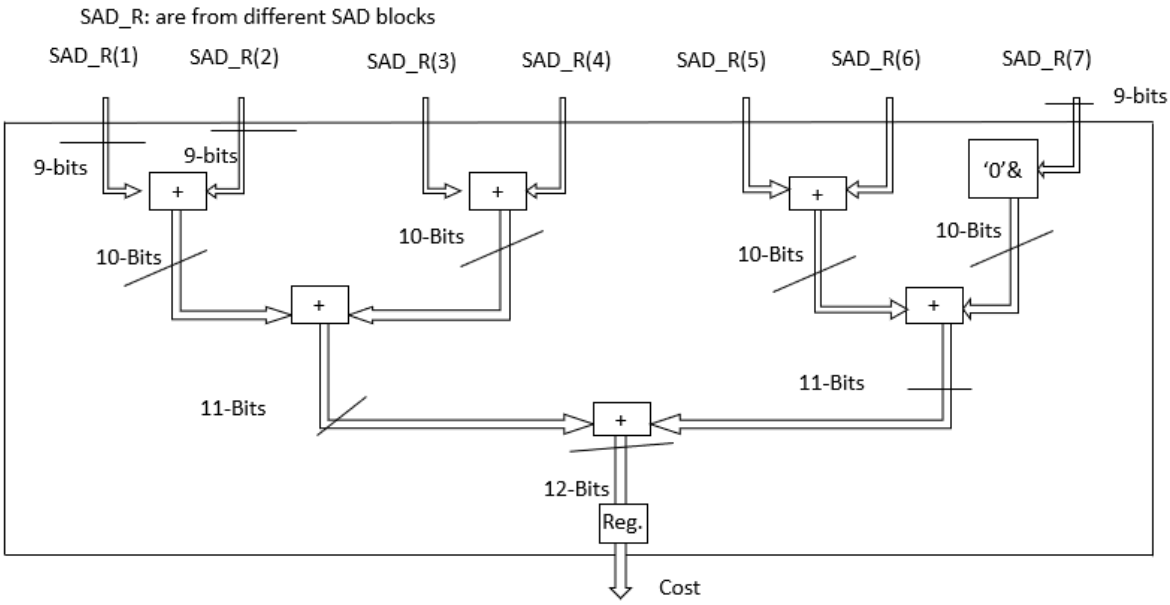


Figure II.12: Sum module architecture for 7*7 window.

II.3.2.4. Cost calculation architecture

The final architecture of the cost calculation block is shown in figure II.13. Each SAD module receives two corresponding columns, one from LSR (reference image) and the other from RSR (candidate image). The cost is calculated between a reference column and its candidate columns. The number of the candidate columns is equal to the disparity range (Dmax), therefore, the number of the SAD blocks in one horizontal line is Dmax. The total number of the SAD blocks for W*W window is W*Dmax. To find the final cost between two windows, it will be just the summation of SAD results that constitute a window. The cost between a reference window and its first candidate is the sum of the first SAD block from first line plus the first SAD block from the second line plus the first SAD block from the third line. For the reference window and its second candidate; it will be the sum of the second SAD block from first line, plus the second SAD block from the second line, plus the second SAD from the third line. The result of each SUM block is considered as the cost; the number of the costs will be equal to the Dmax. To find the best match, we should determine the minimum cost, and the disparity will be equal to its index; if cost (0) is the smallest, then its disparity will be 0.

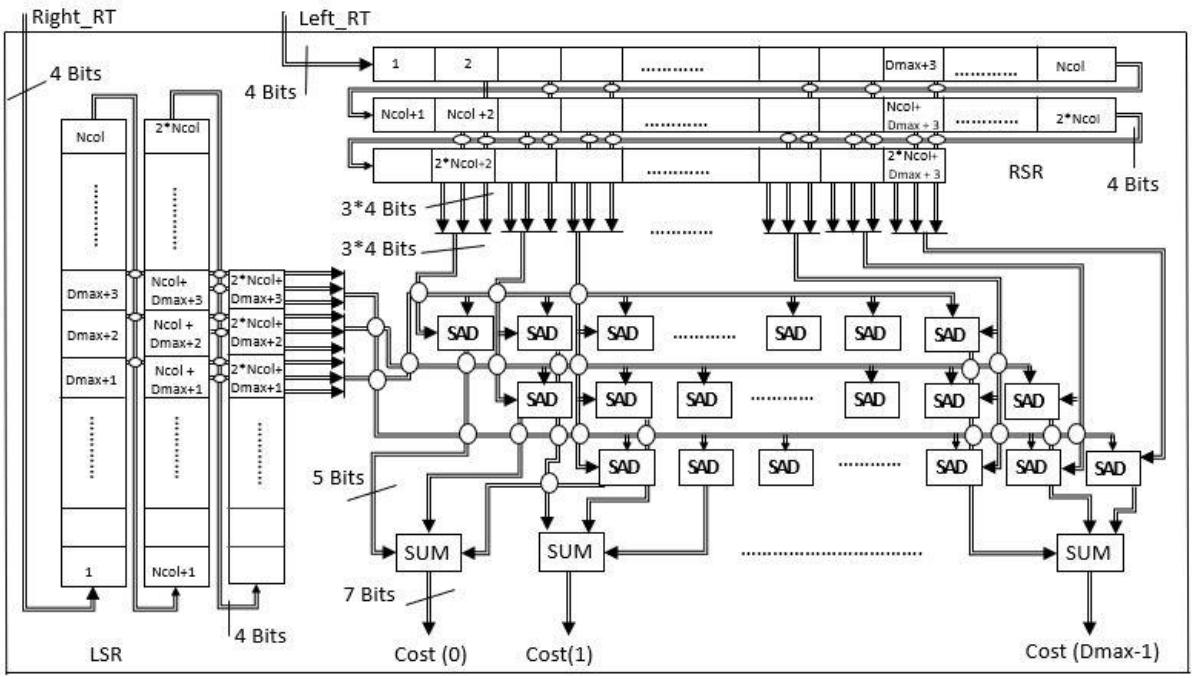


Figure II.13: Cost calculation block architecture for 3*3 window.

II.3.3. Minimum block design

After the calculation of all the costs between a reference window and all its corresponding candidates' windows, we should select the best match; which will be the minimum cost. To find the smallest cost, we have minimum search blocks and final disparity block.

II.3.3.1. Minimum search module

The minimum search module finds the smallest between 10 costs from cost calculation block. It gives as an output the smallest cost and its index. The index ranges from 0 to 9. The number of the minimum search modules that will be used depend on Dmax; if Dmax = 10 there will be one module, if Dmax = 20 there will be two modules, and if Dmax = 30 there will be three modules. However, this design causes limitation on Dmax, because its forces Dmax to be a multiple of 10 for the whole matching module. This design must be modified.

Two architectures were proposed to implement minimum search. The first architecture is shown in figure II.14, the comparisons are performed sequentially. The min block compares two costs, then returns the smallest cost (transparent buses) and its index (black buses). The minimum cost is given at the output as the "smallest" and its corresponding index is given at the output as "Disp". The registers are used to reduce the propagation delay to increase the

frequency. This architecture is easy to implement, however, it gives erroneous results; it does not give the smallest value in some situations, the error occurs only when this block is connected to other blocks.

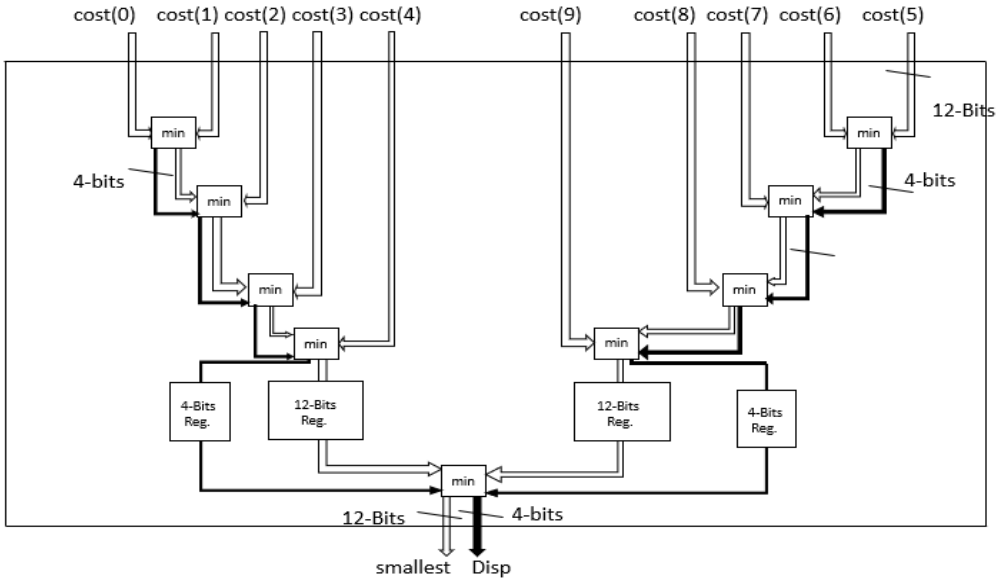


Figure II.14: First implemented architecture for minimum search module.

The second architecture is shown in figure II.15, this one is used for the implementation part. The comparisons are performed in parallel. The min block compares two costs then returns the smallest (transparent buses) and its corresponding index (the buses in black); if the smallest is cost (0) then the index will be 0. The registers are used to increase the processing speed.

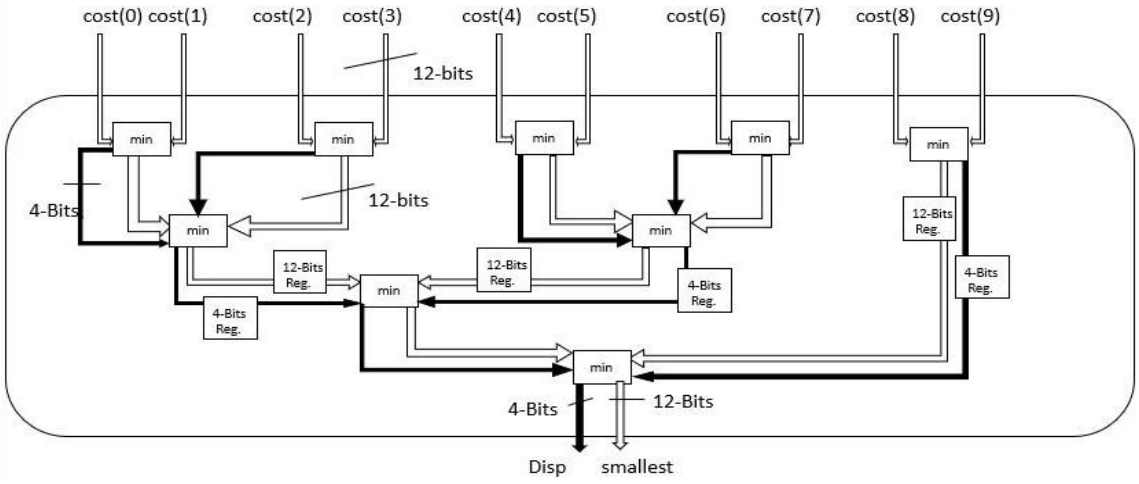


Figure II.15: Second implemented architecture for minimum search module.

II.3.3.2. Final disparity module

We have seen that the number of the minimum search modules is dependent on Dmax. Let's suppose that Dmax = 30; therefore, we will have three minimum search modules. We know that each module has two outputs (smallest and Disp), hence, we will have (smallest (1) and Disp (1)) from minimum search 1, (smallest (2) and Disp (2)) from minimum search 2, and smallest (3) and Disp (3) from minimum search 3. For Dmax = 30, Cost (0) to Cost (9) from cost calculation block are connected to minimum search 1, Cost (10) to Cost (19) from cost calculation block are connected to minimum search 2, and Cost (20) to Cost (29) from cost calculation block are connected to minimum search 3. To determine the final cost, we should first find the minimum between smallest (1), smallest (2) and smallest (3). Figure II.16 shows how the minimum is found, it compares smallest (1) and smallest (2); if smallest (1) has the minimum value hence its value will be saved for the next comparison and the index will be set to 1, and if smallest (2) has the minimum value hence its value will be saved for the next comparison and the index will be set to 2. Then minimum value from the previous comparison will be compared to smallest (3); if smallest (3) has the minimum value the index will be set to 3, if smallest (3) is larger the index will not be changed. Since, the Disp range is from 0 to 9 for all minimum search Blocks; if the smallest (2) is the minimum we should add 10 to Disp (2), if the smallest (3) is the minimum we should add 20 to Disp (3), that's why we added (Index-1 /10) to Disp (index).

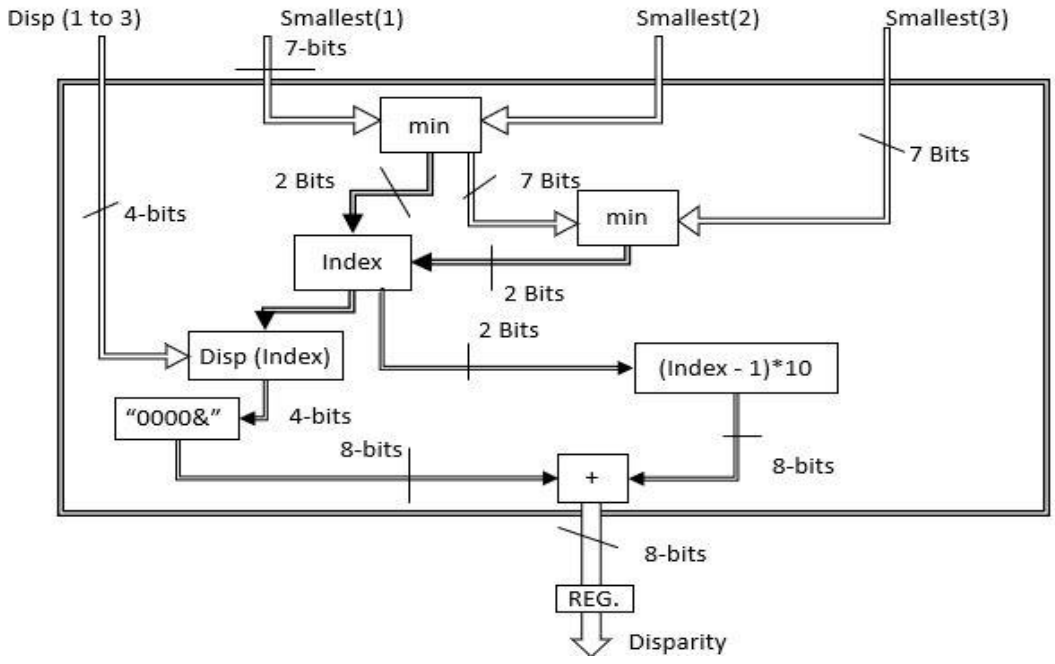


Figure II.16: Architecture of the final disparity module.

II.3.4. Disparity saver block design

The role of this block is generating a write signal and the saving address, this signal is used to say if the disparity at the output would be correct or not. When stereo images are entering in series to matching module, in order to have the first disparity we should wait for latency of the design. The latency of the RT is given by the size of the SR +1; 1 is the for the register in the Rank transform Block, hence it will be (Ncol*(W-1)) +W +1; Ncol is the number of columns of the stereo image. The latency of the cost calculation block is the size of the LSR or RSR + 3; we have 2 registers in the SAD Block and 1 register in SUM Block, the latency of the cost calculation is given by Ncol*(W-1) + W + Dmax+3. The latency of the minimum block is 2; 1 for the register in the minimum search and 1 for the register in the final Disparity. The total latency of the matching module is given in formula II.1.

In RT block we have said that pixels on the borders Cannot be CP of the window because they don't have neighbors, their size is (W-1). Because we have another window of similar size in cost calculation block will lose another W-1, therefore we will lose 2*(W-1). Also, in cost calculation we will lose Dmax at each column; Because the reference windows at the end of the columns they do not have enough candidates' windows along the disparity range, therefore, the windows of the new row will be considered as candidate which may cause mismatching. Therefore, 2*(W-1) + Dmax will be removed from each column, hence we should stop saving after we reach Ncol-2*(W-1)-Dmax as shown in formula II.2. To move to next row, we just wait for 2*(W-1) +Dmax as shown in formula II.3. We can say that number of columns of the DM is Ncol-Dmax-2*(W-1); Ncol is the number of columns of the stereo images. The number of rows of the DM is Nrow-(2*(W-1)), Nrow represents the number of rows of the Stereo image. The DM size is given in formula II.4.

$$Start\ Saving = \big(Ncol * 2(W-1)\big) + (2 * W) + Dmax\ +\ 7. \quad \text{(II.1)}$$

$$Stop\ Saving = Start\ Saving + Ncol - (2 * (W-1)) - Dmax\ . \quad \text{(II.2)}$$

$$New\ row = Stop\ \text{Saving} + 2(W-1) + Dmax\ . \quad \text{(II.3)}$$

$$DM\ size = (Ncol - 2 * (W-1) - Dmax\ ) * (Nrow - (2 * (w-1))). \quad \text{(II.4)}$$

To synchronize all the modules, we will need two counters, the first counter (Count_1) will start counting when receiving the first pixel, it will count from 1 to new row, when it reaches new row, the counter will be set to start saving. When the counter is between start

saving and stop saving, the write signal should be set to 1. A second counter (count_2) will count only when the write signal is set to one, it counts from 1 to DM size. When count_2 reaches DM size, it means the first frame is finished, therefore, we should receive new one. To receive the next frame Count_1 will be set to (2*(W-1) + Dmax + 7); 7 is the number of registers in the design, and (2*(W-1) +Dmax) for count_1 reaches new row. Count_2 is also used to drive the saving address.

II.4. Conclusion

In this chapter we described the global design of the stereo matching module, we explained how it is divided into four blocks: RT block that calculates the RT of the stereo images, Cost calculation block used to find the costs between two corresponding windows, Minimum block responsible of finding the minimum cost and selecting the disparity value, and Disparity saver block used to synchronize the design based on counters. Furthermore, we described the architectures for each module, along with the architectures optimization of some modules to achieve better performances.

Chapter III. Implementation and results

III.1. Introduction

In the previous chapter, we described the design and the architecture of the matching module. In this chapter, we will explain the hardware implementation of each block on FPGA using VHDL, we will describe briefly the architecture of the used FPGA, we will give the results, along with a discussion of the resource usage and the processing speed of the implemented matching module.

III.2. Used boards

The design has been implemented on two Xilinx boards, Nexys 4DRR and Zed -Board. First, the design was implemented and tested on the Nexys and then migrated to Zed-Board which offers the possibility to implement SoC FPGA designs that combines an ARM processor and a 7-series FPGA. In our design, only the FPGA and the Video Graphics Array (VGA) option are used.

III.3. FPGA

Field Programmable Gate Array (FPGA) is a programmable logic device, with internal hardware that can be configured depending on the need of the user, using Hardware Description Language (HDL). Using FPGAs, we can design circuits to perform a specific task. The main advantage of these integrated circuits: they are reprogrammable. The “XC7A100T-1CSG324C” and “XC7Z020-CLG484” are used for this project, both are 7-series from Artix-7 family, the only difference is the number of resources as shown in table III.1. The internal architecture of these FPGAs is similar and divided into six important parts: Configurable Logic Blocks (CLBs), Block Random Access Memory (BRAM), Digital Signal Processing (DSPs), Clock sources, Programmable interconnects and Input Output Blocks (IOBs). These components are arranged as shown in figure III.1.

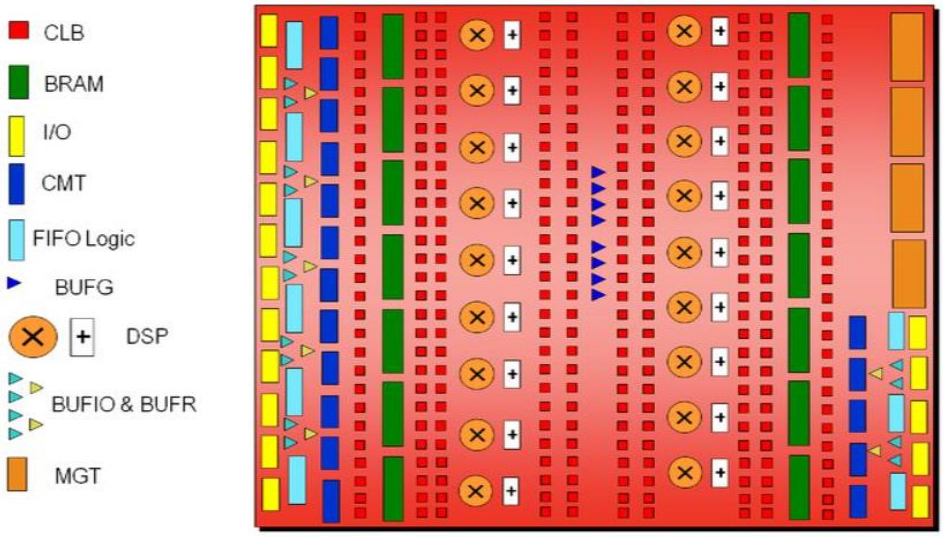


Figure III. 1: 7-series FPGA architecture [12].

III.3.1. CLBs

The CLBs implement logic function and also used as a memory. Each CLB is divided into two Slices: they can be SliceM (M: Memory) and SliceL (L: Logic) for CLBLM, or Both SliceL for CLBLL. Each Slice has four Lock Up Tables (LUTs) of 6-bits, three of 2-to-1 Multiplexers (MUX), Carry- 4 Logic; used for fast arithmetic addition and subtraction, and eight Flip-Flops (FFs).

- SliceL allows implementation of logic and arithmetic operation, and used for construction of Read Only Memory (ROM).
- SLiceM implements SliceL's Functions, plus construction of distributed Random Access Memory and shift Registers.

III.3.2. BRAM

The BRAMs in these FPGA are 36-Kbits: the data bus is 32-bits and the address bus is 16-bits. Each BRAM is constituted of two 18-Kbits RAM: each RAM has a data bus of 16-bits and an address bus of 15-bits. It can be programmed as a simple port, dual port or true dual port. BRAMs are used to create different memories RAMs, ROMs and First In First Out (FIFO) registers.

III.3.3. Clocking sources

The clocking sources manage simple and complex clock requirement with global clock, regional I/O and Clock Management Titles (CMTs). Each CMT is constituted of one Mixed-Mode Clock manager (MMCM) and Phase-Locked Loop (PLL).

III.3.4. DSP blocks

The DSPs are used to accelerate digital signal processing operations such as fast Fourier transform, and they include multipliers (25 bits * 18 bits) for complex mathematical operations.

III.3.5. Programmable interconnects and IOBs

Programmable Interconnects are used to implement routing paths between CLBs, they are constituted of wires and switch boxes. The Input Output Banks (IOBs) allow the FPGA to connect with external components, there are multiple (IOBs) and they are characterized by a voltage value.

Table III. 1: Resources of the used FPGAs.

Resources	XC7A100T-1CSG324C	XC7Z020-CLG484
CLBs	7925	6650
CMTs	6	4
BRAM (36Kbits)	135	140
IO	210	200
DSP Blocks	6	4

III.4. Used IP cores.

To implement the matching module, two IP cores are used: Clock Wizard and Block Memory Generator. The former is used to generate the system clock. The latter is used to generate two ROMs to store the stereo images, and one RAM to save the final disparity map.

III.4.1. Clock wizard

This IP core simplifies creation of clock circuits; these circuits can be configured at specific clock. It offers the selection between two primitives a Phase-locked Loop (PLL) and Mixed-Mode Clock Manager (MMCM). This block offers a stable and a valid clock at the output. Also, it can generate up to seven output clocks when PLL is selected, and six output clocks when MMCM is selected. At the input it accepts two clocks. Furthermore, this IP core offers different features, two have been used: Frequency Synthesis allows to have different frequencies at the output, Frequency Alignment is an option to make the output frequency phase locked to the input clock. In this system we selected a PLL primitive, with one input clock connected to the board clock (100MHz), and two output clocks: the first is used to control the stereo vision system and the second to control the display part of the system.

III.4.2. Block memory generator

Block RAMs IP core can create or generate five types of memories. They are listed as:

- Single port RAM.
- Simple dual port RAM
- True dual port RAM
- Single port ROM
- Dual port ROM

Single port means only one port is used to access the memory. For the RAM, the port can be used either to read or to write to memory at the same time. Dual port means two ports are used A and B to access the memory at the same time; for the RAM, Port A is used to write into the memory and Port B is used to read from the memory, operating mode must be selected; read first, write first or no change can be chosen. Hence collision may appear in some situations; when writing and reading into the same address. For the ROM both ports are used for reading. True Dual port RAM has also two ports; both ports can read and write to the memory, collision may appear in this type also. Moreover, optional output registers may be added for each port for all types. During the implementation, two single port ROMs used to store the initial stereo images, they do not have any register at the output. It has a dual port RAM used to save to final disparity, without any register at the output.

III.5. Matching module implementation.

The matching modules is a combination of four blocks as shown in figure III.2: RT block, cost calculation block, minimum block, and disparity saver block. The blocks are implemented using RTL descriptions VHDL. The matching module has as inputs: pixel_left, pixel_right and clk. It has as outputs: Disparity, write and saving_address. The matching module has four generic values: W, Ncol, Nrow and Dmax, where W represent the window size, Ncol represents the columns number of the Stereo image, Nrow represents the rows number of the Stereo image, and Dmax represent the disparity range.

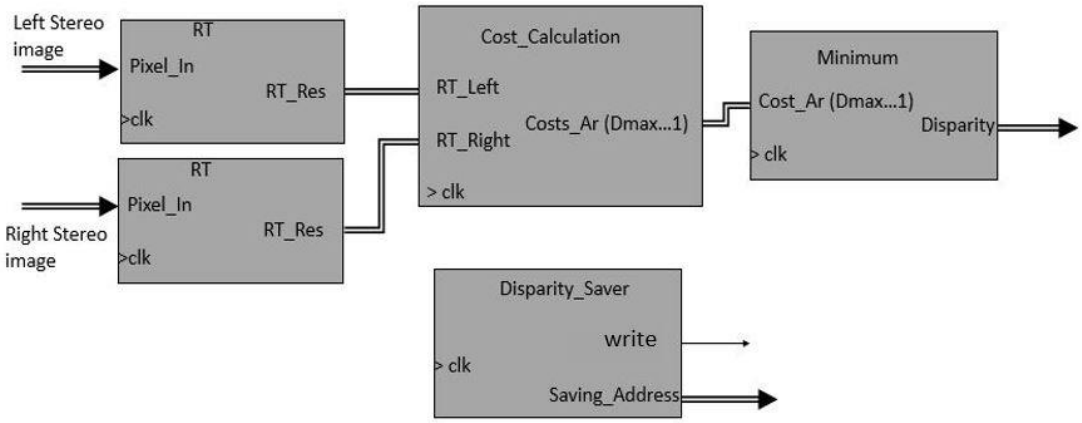


Figure III. 2: Block diagram of the matching module.

Figure III. 3: Matching module as a block.

III.5.1. RT block implementation

The RT block was divided into three modules: SR, comparator, and summator modules. RT block has two inputs: Pixel_in which is an 8-bits vector, and clk. It has one output RT_Res which is an RTL(W)-bits vector, where RTL is a constant array of integers declared in separate library, since the size of the RT_Res changes when changing the generic value W. Also, RT module generates the needed number of comparator modules using for generate…statement based on the window size W.

SR module has two inputs: clk and the Pixel_In. It has one output PIX which 8-bits vector array having W*W elements. Since the input pixel is 8-bits, eight SRs are created using vector slicing method; by inserting the input at one end. The size of the eight SRs depends on the two generic values: W and Ncol, the SR size is given by Ncol*(W-1) + W, as we have seen in the previous chapter. The output (elements of the window) of the SR are selected from shift registers using two for…generate statements as nested loops; one will count the number of rows of the window and the second will count the number of columns. The elements of the window are assigned to the output PIX.

The comparator module has two inputs neighbour_pixel and central_pixel, and one output S. The comparison is performed using if-statement; S is set to one when neighbour_pixel is less than the central_pixel.

The summator module has five internal modules, depending on the generic value W the module that will perform addition operation is selected using if…generate statement. These modules have similar inputs and output, which are named similarly: clock and C_results as inputs and RT_Res as an output. C_result is a (W*W-1)-bits vector holding all the comparison results. RT_Res is an RTL (W)-bits vector. The addition operations are performed using std_logic_unsigned library for all five internal modules.

III.5.2. Cost calculation block implementation.

Cost calculation block is divided into four modules: LSR, RSR, SAD, and SUM. Cost calculation block has three inputs: clk, RT_Left and RT_Right. It has one output: Costs_Ar which is an array of Dmax elements that identifies the result of each SUM module. Also, this module rearranges the windows that it receives from LSR and RSR to convert rows of the windows to columns and columns of the windows to rows, in order to access the initial columns

easily; because the similarity is calculated between columns as said in the second chapter. The number of the SAD modules that will be used is W*Dmax, as it was mentioned in the previous chapter, to generate the required SAD modules we used the nested for…generate concept. The number of the SUM modules is equal to Dmax, hence for…generate statement is used to generate these modules.

LSR module has two inputs clk and RT_Right, and one output L_window. The input RT_Right is RTL(W)-bits vector; therefore, the length of the input is not fixed, we know in hardware each bit should have its shift register. Hence, the number of the shift registers varies according to RTL(W) value. We created a submodule SR which uses vector slicing method to create the shift register. The shift register size depends on three generic values: W, Ncol and Dmax, the size is given by Ncol*(W-1) + W + Dmax. Afterwards, we used for…generate statement to generate the required SR modules. The output L_window is an RTL(W)-bits vector having W*W elements. L_window elements are selected from SRs using for…generate statement as nested loops. RSR module is similar to LSR module, with only small difference at the output. RSR has R_window as an output which is an RTL(W)-bits vector having W*((W-1) + Dmax) elements.

The SAD module has three inputs: clk, Left, and right. It has one output SAD_R. Left and right are arrays of RTL(W)-bits vector, having W elements. SAD_R is a Res_S_Length(W)-bits vector, where Res_S_Length is a constant array of integers holding the size of SAD_R, Res_S_Length is declared in the same library as RTL. This module calculates absolute subtraction operation using numeric_std library. The SAD module has 5 submodules to perform the addition operations shown in figure II.11. Based on W, the needed submodule will be selected using if…generate statement depending on W.

Sum module has two inputs: clk, SAD_R_A. It has one output Cost. SAD_R_Ar is a Res_S_Length(W)-bits vector array having W elements. Cost is a Cost_Length(W)-bits vector; Cost_Length is a constant array of integers holding the size of Cost depending on W. The SUM module has 5 submodules that performs summation, shown in figure II.12. Based on W, the needed submodule is selected using if…generate statement. The submodules have the same inputs and outputs as SUM module and they perform addition operations using std_logic_unsigned library.

III.5.3. Minimum block implementation

The minimum block has two inputs: clk and Cost_Ar, where Cost_Ar is Cost_Length(W)-bits vector array of Dmax elements. It has one output Disparity, which is 8-bits vector. This block has two internal modules: minimum search module and final disparity module. For…generate statement is used to generate the required number of the minimum search modules. The outputs from the minimum search modules are saved in two arrays declared as signals: smallest_Ar, Disp_Ar.

The minimum search module has two inputs: clk and Cost_Ar which is a Cost_Length(W)-bits vector array of ten elements as shown in figure II.15. It has two outputs: smallest and disp, where smallest is Cost_Length(W)-bits vector and disp is 4-bits vector. We divided the comparison into four stages or submodules denoted as: Stage_1, Stage_2, Stage_3, and Stage_4. The comparisons are performed using if-statement in all submodules except the second stage where the comparison is performed at the clock edge. Stage_1 has five comparators, Stage_2 has two comparators, Stage_3 and Stage_4 have one comparator.The indexes which will be considered as Disp are created in Stage_1, then passed to next modules.

The final disparity module has three inputs: clk, smallest_Ar and Disp_Ar, where smallest_Ar is a Cost_Length(W)-bits array of Dmax/10 elements, and Disp_Ar is a 4-bits vector array of Dmax/10 elements. It has one output Disparity which is 8-bits vector. The comparisons shown in figure II.16 are performed using if-statement. Index shown in figure II.16 is a variable that depends on the comparison result. The multiplication operation shown in figure II.16 is performed using NUMERIC_STD Library.

III.5.4. Disparity saver implementation

The disparity implementation requires two counters, as explained in chapter two, the counters are controlled using if-statement. The equations in section II.3.4 are fixed as constant and they depend on generic values. The generation of the write signal was controlled using when/else assignment.

III.6. VGA controller module

VGA is video standard display, is used to interface a system to a monitor for image and video displaying. The VGA displaying format is 640 columns by 480 rows of pixels. To control a VGA monitor, five signals are needed: Red, Green and Blue, H_syn (Horizontal synchronization) and (Vertical synchronizaation). For synchronization purpose, Horizontal and vertical counters are required Hcount and Vcount: Hcount is counting from 1 to 800, Vcount is counting from 1 to 525. H_syn = '1' when Hcount ≤ 704, and Vcount = '1' when Vcount ≤ 515. Also, these two counters are used to generate a video signal (vid_Sig) in order to control the address counter, which is used to read data from the RAM, and to enable its port. vid_sig = '1' when (33 ≤ Vcount < 10 + Nb_Row) and (48 ≤ Vcount < 16 + Nb_col); Nb_col represents columns number of the DM and Nb_row represents the rows number of DM. The address counter will count from 1 to Nb_row * Nb_col; when the vid_sig = '1', if the counter reaches Nrow * Ncol it will set to 1 again. This all configured in one block named vga_controller using RTL VHDL.

The eight_to_four_bits module is used to map an 8-bit pixel to 4-bits, it has two inputs video_sig and pixel; the pixel is 8-bits and connected to output of the RAM, where DM is saved. Since the VGA in the board is 4-bits, this module selects 4-bits from the pixel, and assigns them to Red, Green and Blue; this process is done when Vid_sig = '1'. Otherwise, it assigns "0000" to the outputs, in order to display black screen in the rest part of the display; because the DM size is less than the VGA format.

III.7. Final design

The connection between the ROMs, matching module, RAM and the VGA controller is achieved using block diagram, as shown in figure III.4. The address generator is used to count the addresses of the ROMs.

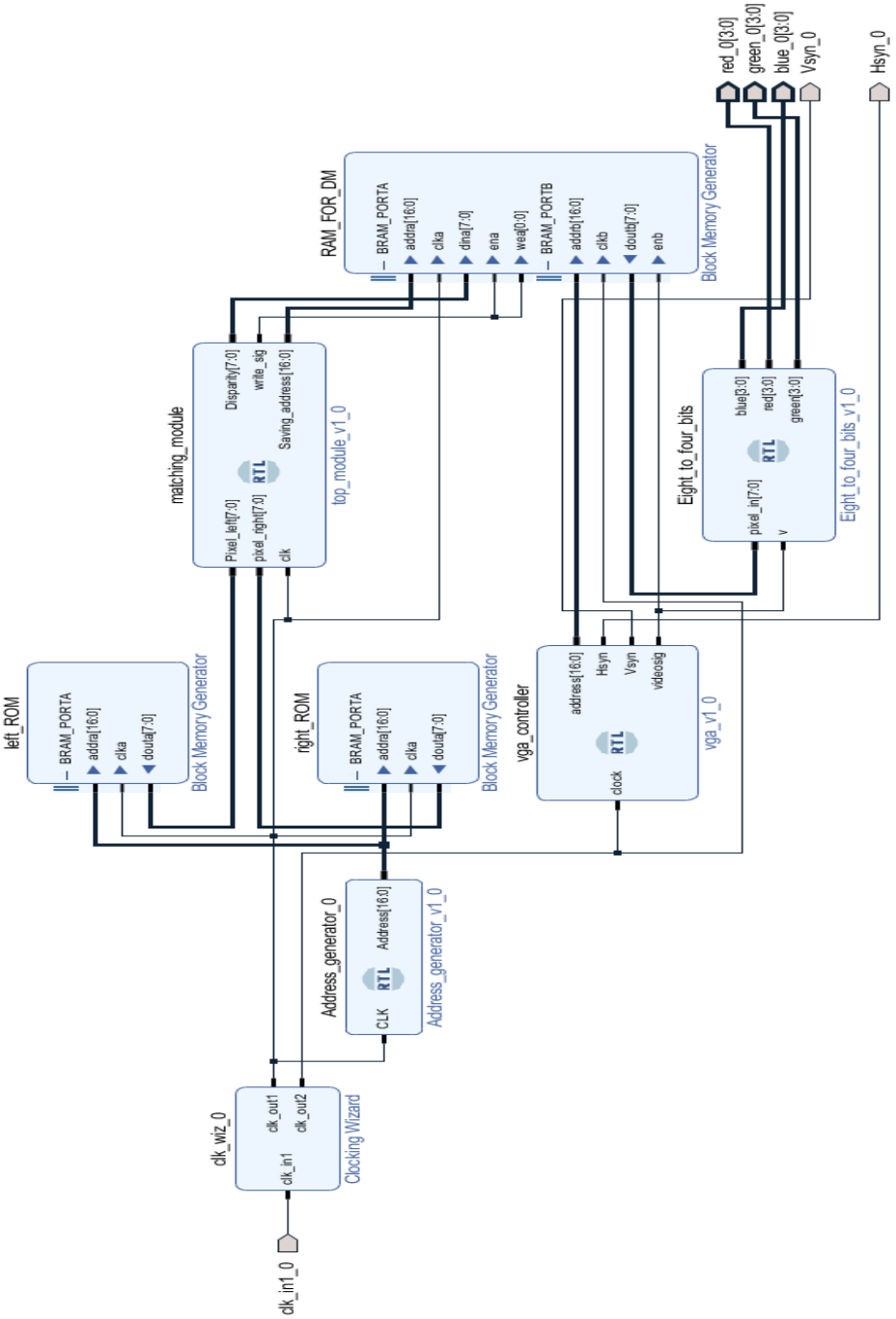


Figure III. 4: Block diagram showing connection between matching module, VGA controller and the used IP cores.

III.8. Design validation

To validate the design, the different blocks were verified, first step was verifying the RT module. The second was is verifying the minimum module. The cost calculation module is verified during the verification of the matching module.

III.8.1. RT verification

At the beginning, each module in RT block was verified alone. Afterwards, the whole RT block was verified; MATLAB was used to generate the correct results which are saved as a Coefficient file, then load that file into the ROM. The disparity saver block is modified in order to save the results of RT in RAM. A comparator module is created to calculate the inverse of the difference between the ROM data and the RAM data as shown in figure III.5.

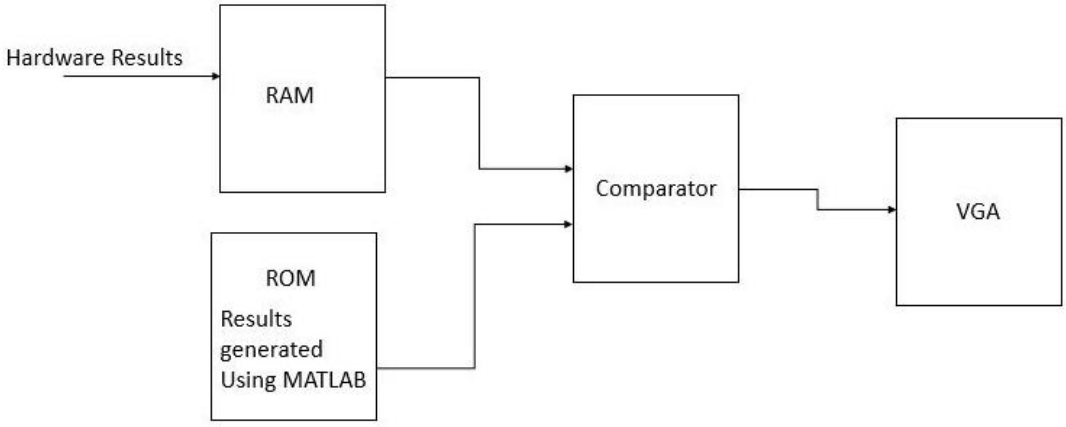


Figure III.5: Connection between the comparator and the memories.

Figure III.6 shows simulation results of the RT block during the validation process. We compared the content of the RAM and the ROM address by address. RAM_data corresponds for the results generated by the hardware design. ROM_data holds the correct results generated by MATLAB. The signal 'result' in figure III.6 is used to compare RAM_data and ROM_data, if they are equal, the result value is FF.

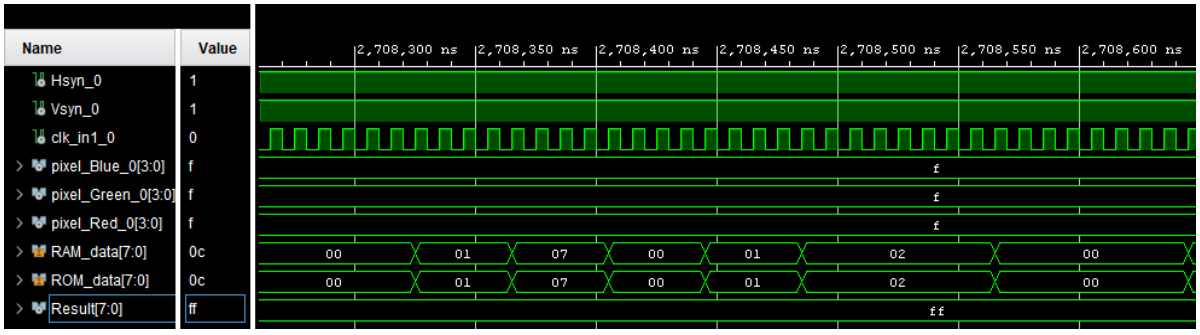


Figure III. 6: Verification of RT block using simulation.

Figure III.7 shows the RT block validation results, the image on the top left shows the right stereo image, the image on the top right shows the right image after RT. The white screen is the result of the comparison, since FF_H in the gray scale represent the white color.

Figure III. 7: Validation of the RT block on the board.

III.8.2. Minimum block validation

Minimum search and final disparity modules were verified separately. We created a test bench for each module, then set values to the inputs and verified the resulting outputs during the simulation. We run this process for different inputs, at each run, the smallest value location is changed within the array. In figure III.8, the minimum values in the Cost_Ar is 4 and its index 9. Figure III.9 shows the results of the final disparity module, three cases were shown. The first case, we have the minimum in the smallest is 42 and its index 3, Disp (3) = 4, hence, disparity will be 20 +4. The second case, the minimum in the smallest is 42 and its index 1, Disp (1) = 2, hence, disparity will be 2. The last case, we have the minimum in the smallest is 42 and its index 2, Disp (2) = 2, hence, disparity will be 10+2.

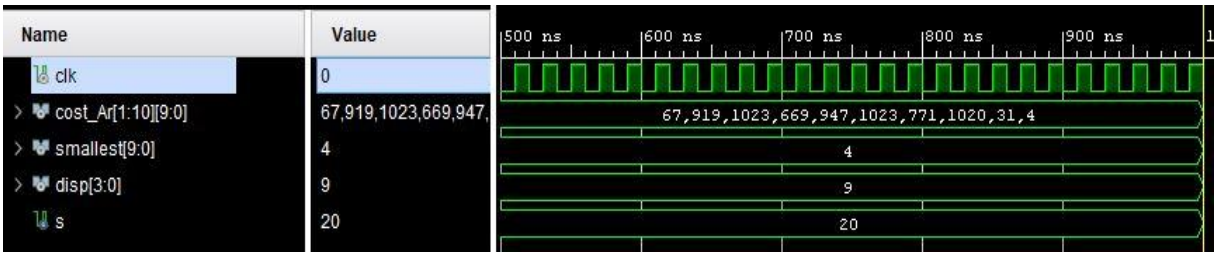


Figure III. 8: Minimum_seacrh module verification.

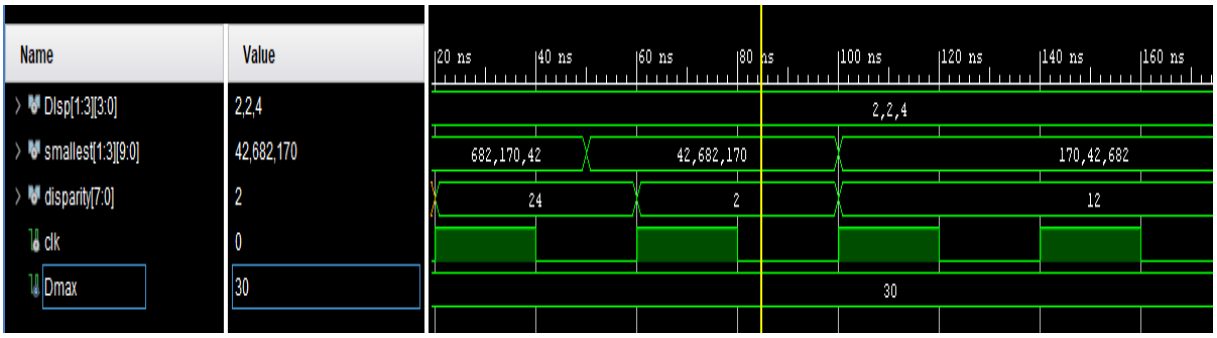


Figure III. 9: Final disparity module verification.

III.8.3. Matching module validation

After validating RT block and minimum block, the cost calculation block is verified during the verification of the matching module; obviously SRs, SAD and SUM blocks are verified separately, however, testing all these modules together is done during the verification of the final matching module. Similar concept to the verification of RT; the correct DM generated by MATLAB is saved in ROM and the DM map generated by the hardware is saved in RAM. A comparator is used as in the figure III.5.

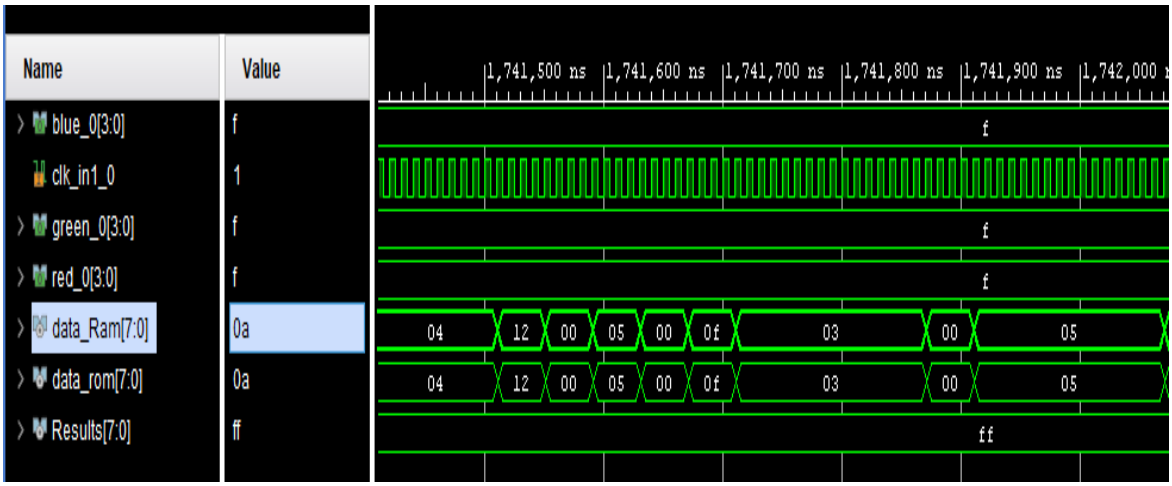


Figure III. 10: Simulation validation of the matching module.

Figure III.11 shows the validation of the matching module and the DM results. The picture on the top left is DM for window 5*5 calculated in hardware, and the one on the top right is the DM found using MATLAB. The white picture is the inverse of the difference between the two DMs. This verification was performed for all windows and the following disparity ranges 10, 20 and 30.

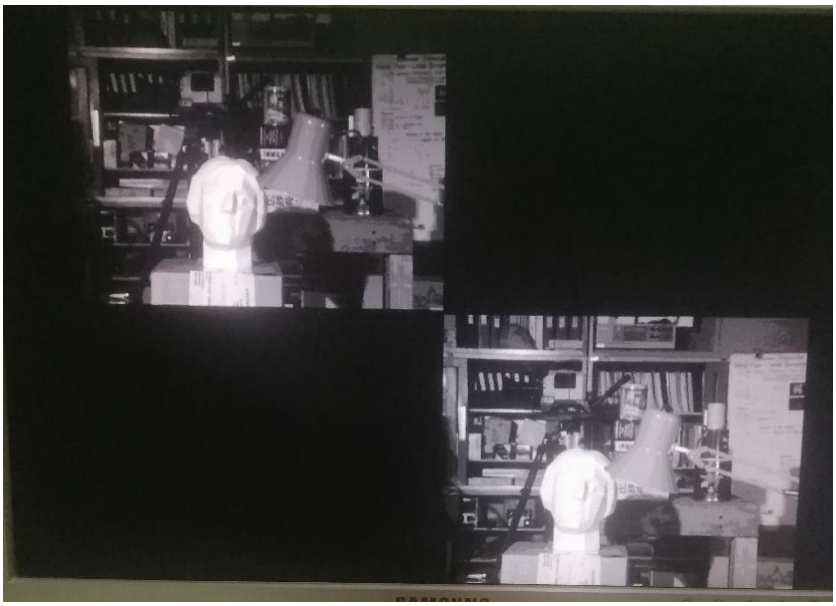
Figure III. 11: Implementation validation of the matching module.

III.9. DM estimation for different window size

Figure III.12 shows the left and the right stereo images; with image resolution of 320*240, and their corresponding disparity maps for different window size. When increasing the window size, the mismatches are decreasing with the cost of losing data on the boarders; because the final resolution of DM is affected by two parameters: window size and Dmax. Table III.2 shows the DM resolution for Dmax = 20, Ncol = 320 and Nrow =240.

Table III. 2: DM resolution of stereo images of 320*240 pixels for different window size

Window size	DM resolution
3*3 pixels	296*236
5*5 pixels	292*232
7*7 pixels	288*228
9*9 pixels	284*224
11*11 pixels	280*220



a) Right and left images.

b) Window 3*3

c) Window 5*5

e) Window 7*7

f) Window 9*9

d) Window 11*11

Figure III. 12: DM for different window sizes and Dmax = 20.

III.10. Resource usage and processing speed

III.10.1. Resource usage and processing speed for each configuration

Table III.3 shows the resource consumption for different window size and different disparity range. The maximum frequency for each configuration and its equivalent speed, the given parameters in the tables concern only the matching module. However, we could not have more details for the last configuration where window size is 11*11 and Dmax is 30 due to the limitation of resources on the board. Table III.4 shows the resources usage for each block.

Table III. 3: Resource usage and processing speed for the matching module for images size of 320*240 pixels.

	Dmax	LUTs	Pct (%)	FFs	Pct (%)	Slices	Pct (%)	Frequency Max (MHz)	Speed (fps)
window 3*3	10	1727	2.72	894	0.70	552	3.48	163.437	2092
	20	2867	4.52	1535	1.21	895	5.64	105.411	1349
	30	4002	6.31	2188	1.72	1233	7.77	79.494	1017
window 5*5	10	5024	7.92	2277	1.79	1526	9.62	155.150	1954
	20	8589	13.56	3873	3.02	2556	16.12	101.981	1284
	30	12225	19.28	5496	4.33	3667	23.13	72.746	916
window 7*7	10	10404	16.41	4524	3.56	3276	20.66	143.040	1773
	20	18324	28.90	7625	6.01	5884	37.12	102.160	1266
	30	26900	42.42	10733	8.46	8540	53.88	71.611	886
window 9*9	10	18931	29.86	7683	6.05	5694	35.92	141.503	1726
	20	34361	54.19	12898	10.17	10269	64.78	100.705	1228
	30	49820	78.58	18114	14.28	14486	91.39	69.898	852
window 11*11	10	28353	44.72	11052	8.71	8370	52.80	125.28	1505
	20	52231	82.38	18390	14.50	15042	94.90	93.411	1122
	30	75675	119.36	25730	20.29	/	/	/	/

Table III. 4: Resource usage for each block for images size of 320*240 pixels and disparity range Dmax=20.

Blocks		Cost_Calculation	RT	Minimum_search	Disparity_Saver
Window 3*3	LUTs	2359	199	71	39
	FFs	1288	78	57	34
	Slice	704	75	62	25
Window 5*5	LUTs	7569	444	99	42
	FFs	3340	212	75	34
	Slice	2223	145	92	24
Window 7*7	LUTs	16674	737	115	41
	FFs	6672	416	87	34
	Slice	5329	256	92	24
Window 9*9	LUTs	32027	1085	123	41
	FFs	11425	673	93	34
	Slice	9431	398	91	24
Win dow 11*11	LUTs	48812	1467	122	40
	FFs	16211	1023	99	34
	Slice	13944	528	122	26

III.10.2. Resource usage analysis

The bar graphs in figure III.13 show the slices usage variation with the window size; the resources are increasing with the window size. The first bar graph shows how the slices are changing with Dmax; the number of the slices increases significantly when increasing Dmax. Dmax affects directly the number of the SAD blocks, the number of the sum blocks and the the number of minimum search blocks. The second bar graph shows how the image resolution influences the number of resources, the difference between the two image resolutions (320*240

and 640*480) is not huge because only the SRs size is affected when changing the resolution.

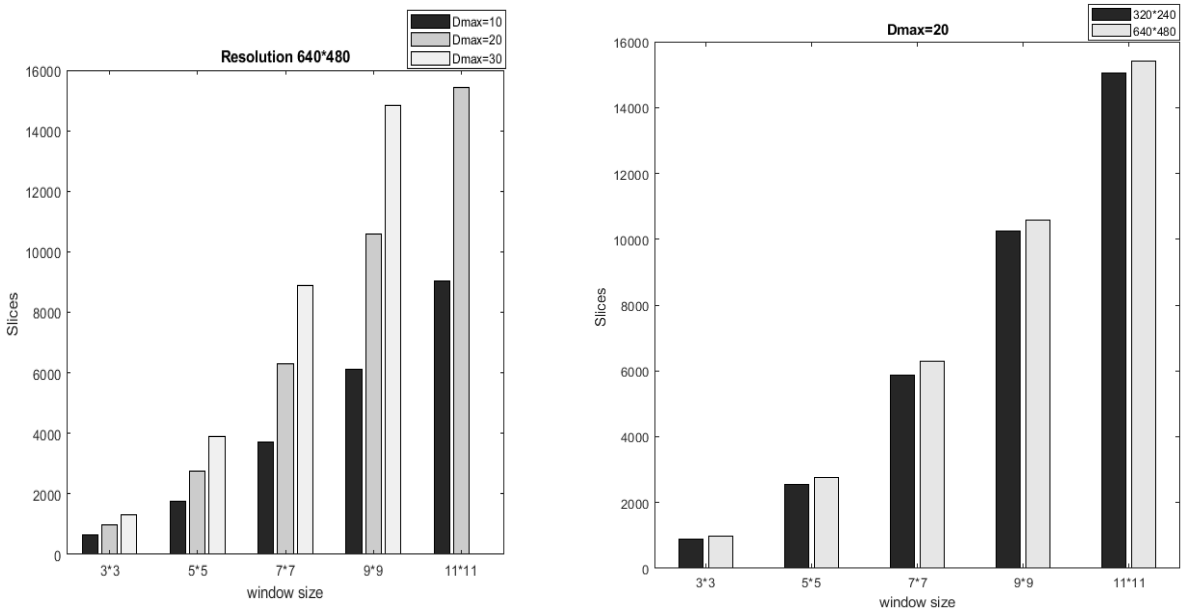


Figure III. 13: Slices usage in term of window size, disparity range and stereo image resolution.

III.10.3. Processing speed analysis

The bar graphs in figure III.12 show the processing speed change with respect to window size. When increasing the window size, the processing speed is slightly decreasing; the propagation delay increases because more addition operations are required in the SAD modules and the SUM modules, this delay reduces the frequency which reduces the speed. The first bar graph figure III.12 shows how the speed is changing with the resolution; the speed is reduced approximately by the quarter, because the image format is also increased by the quarter; from 320*240 to 640*480. The second bar graph in figure III.12 shows how the disparity range influences the processing speed, this is due to the propagation delay between minimum search and final disparity blocks; the architecture of the final disparity must be modified.

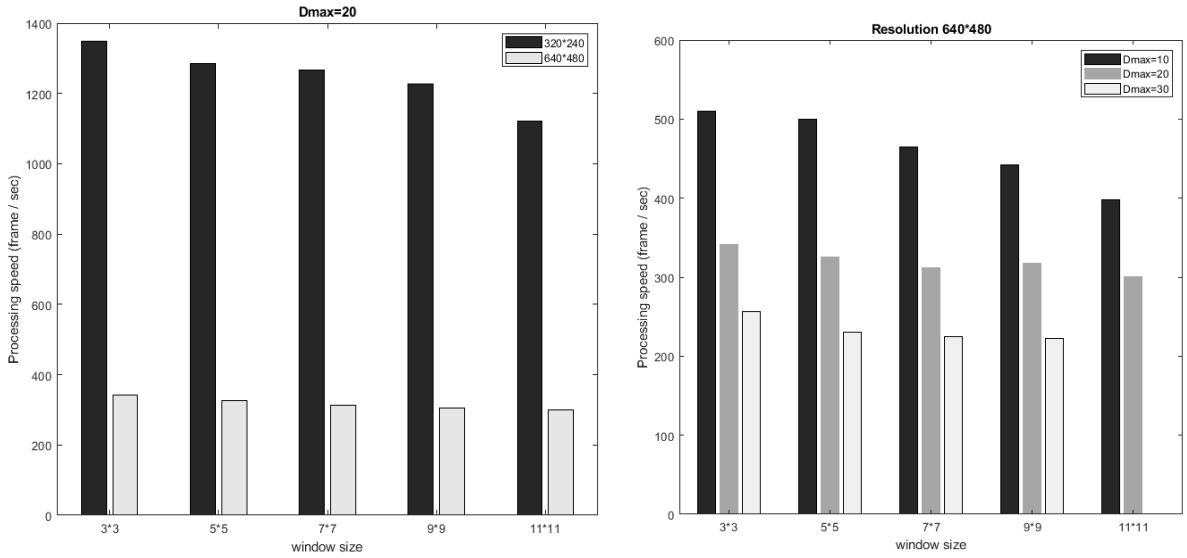


Figure III. 14: Processing speed in term of the window size, resolution and disparity range.

III.10.4. BRAM usage

To run and test the matching module, we used BRAM to store the stereo images and to save the DM generated. However, the number of BRAMs in the FPGA is limited. For stereo images of resolution 320*240, the number used of BRAMs is 76, which is 56.29 % of available BRAMs. Hence, to run this module for stereo images of VGA format, we can use the stereo cameras, external memory or a dynamic RAM available on the Zed-Board.

III.11. Migration

At the beginning, the design was implemented on the Nexys 4ddr board then migrated to the Zed-board. The objective of the migration process is to interface the matching module with the final hardware/software system of obstacle detection; because the zed-board is a zynq device that contains both ARM processor and a 7-syries FPGA. The design was only verified using the simulation, however, the FPGAs on both boards have similar architecture; they are both 7-series from Artix family, hence, the implementation works successfully, however, for a configuration of a window 9*9 and Dmax 30 cannot be implemented due to resource limitation.

III.12. Conclusion

In this chapter, we described how the matching module is implemented, along with the different blocks, and how they are connected. We explained how each block is verified and how the matching module is validated. Also, a visualization of the resulting DMs for the different window size is shown. Finally, we discussed the resource usage and the processing speed for different window sizes, and we gave the resource usage for each block. We found that the cost calculation block is the one consuming more resources.

General conclusion

In this project, we implemented a hardware stereo matching module on a FPGA using correlation-based algorithm using a non-parametric transformation RT to solve the radiometric distortion, and the SAD correlation metric to measure similarity between the stereo images. We explained how we divided the matching modules into four blocks: RT block, cost calculation block, minimum block, and disparity saver block. Then we described the designed architecture of each block. RT block basically calculates the RT of the stereo image, since we have two images left and right hence, we need two RT blocks. Cost calculation block uses SAD correlation metric to measure the cost between a reference window and its candidate windows. Minimum block finds the smallest cost and returns its corresponding disparity. Disparity saver block is used to synchronize the design and to generate the write signal and the saving address to write disparities into a RAM. Afterwards, we tackled the implementation process, where we declared W, Dmax, Ncol and Nrow as generic values. All the matching modules were implemented using RTL description VHDL. The RT block is implemented with three internal modules: SR module, comparator module and summator module. The cost calculation block is implemented with four internal modules: LSR module, RSR module, SAD module and the SUM module. Minimum block is implemented with two internal modules: minimum search and final disparity. Also, when changing generic values, the size of some inputs and outputs changes, hence we used constant array integers to save the sizes of each window size.

The implementation results show that we reached a high processing speed, the minimum speed recorded is greater than 200 frames per second for the VGA format. For a quarter VGA format, the processing speed is multiplied approximately by four. The processing speed decreases when increasing the window size or the search range Dmax. The resource usage is more affected by the window size and the search range Dmax, when increasing these two parameters the resource usage increases. The stereo images format can slightly affect the resource usage of the matching module. During the implementation, we were not able to test the resulting DMs for the VGA format due to the limitation in the memory size. Also, we were obliged to fix the search range Dmax to be a multiple of 10. We deduced that Dmax affects directly the number of the SAD modules and SUM modules which affects significantly the resource usage. As perspective, the challenge will be fixing the number of the SAD module and SUM module when changing the search range Dmax and remove the constraint where Dmax should be a multiple of 10. Also, use an external memory to test the DM for VGA format.

Bibliography

[1] M. Bendahmene, "Hardeware/Software Codesign of Real Time Obstacle Detection System using Stereovision," Master Thesis, Boumerdes, 2022.

[2] S. George, "Binocular Vision: A Physical and a Neural Theory," *The American Journal of Psychology*, vol. 83, no. 4, pp. 461-534, 1970.

[3] B. Nick, "What Is Computer Vision & How Does it Work? An Introduction," 20 July 2020. [Online]. Available: https://xd.adobe.com/ideas/. [Accessed 22 06 2022].

[4] I. Bannacer, "Conception et implémentation sur FPGA d'une architecture pour l'extraction temps-réel du plan de sol," Magister Thesis, El Harrach, 2014.

[5] B. Theodor and A. Dumitrache, Advances in Robot Manipulators, Bucharest: Master Thesis, 2010.

[6] P. Dufour, "Real-Time Stereo-Matching on Embedded Hardware for MAVs," 2010.

[7] S. Lea, R. Daniel, W. Jakob, K. Jacques, R. Arne and D. Rüdiger, "Neuromorphic Stereo Vision: A Survey of Bio-Inspired Sensors and Algorithms," p. 20, 28 May 2019.

[8] I. Zohir, B. Hamza, D. Michel and K. Abdelhakim, "FPGA implementation of the V-disparity based obstacles detection approach," in *21st Mediterranean Conference on Control & Automation (MED)*, Platanias-Chania, 2013.

[9] B. Hamza and K. Abdelhakim, "Stereo vision IP design for FPGA implementation of obstacle detection system," in *8th International Workshop on Systems, Signal Processing and their Applications (WoSSPA)*, 2013.

[10] R. Zabih and J. Woodfill, "Non-parametric local transforms for computing visual," in *in ECCV'94: Proceedings of the Third European Conference on*, Secaucus, NJ, USA, 1994.

[11] F. Wade S., "Improved Stereo Vision Methods for FPGA-Based Computing Platforms," Thesis, Brigham Young University, 2011.

[12] T. Gray, "Zynq Architecture 7-Series FPGA Architecture," *presentation*, 2018.