PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

MINISTERY OF HIGHER EDUCATION AND SCIENTIFIC RESARCH

University Mohamed El-Bachir El-Ibrahimi - Bordj BouArreridj

Faculty of Sciences & technology

Department of Electronics

MasterThesis

Presented to get

THE MASTER'S DIPLOMA

BRANCH: ELECTRONICS

SPECIALTY: EMBEDDED ELECTRONIC SYSTEMS

By

BOUAZA Youcef

FEDJIRI Anouar

Entitled

Image filtering design and implementation based on Xilinx System Generator with Hardware Co-Simulation and VHDL with FPGA IP Core Generator

Thesis Submitted and defended in

Board of Examiners:

Chairman: Dr. DIFFELLAH Nacira	University of Bordj Bou Arreridj
Examiner: Dr. BOUDCHICHE Djamel	University of Bordj Bou Arreridj
Supervisor: Melle. HAMADACHE Fouzia	University of Bordj Bou Arreridj

Academic year 2022/2023

Abstract

One of the very useful techniques in Image Processing is the 2D Gaussian Filter, especially when removing gaussian noise. However, the implementation of a 2D Gaussian Filter demands significant computational resources, and when it comes down to real-time applications, efficiency in the implementation is crucial. This thesis describes the methodology for implementing gaussian image filtering using MATLAB and real-time DSP applications on FPGA using the concept of hardware software Co-simulation for digital image processing by using the Mathworks model-based design tool Simulink / Xilinx System Generator (XSG) and Very High Description Language (VHDL) using the advanced image processing cores included in the IP core lib library. Performances of efficient architectures are implemented on FPGA Spartan3e (xc3s500e). Peak Signal-to-Noise Ratio (PSNR), the Structural SIMilarity (SSIM) index, and FPGA usage of resources are used to discuss and compare the findings obtained from software and hardware.

Keywords: Gaussian image filtering, FPGA, Hardware Co-Simulation, Xilinx System Generator, VHDL, Xilinx IP CORE Generator.

Résumé

L'une des techniques très utiles en traitement d'images est le filtre gaussien 2D, notamment lorsqu'il s'agit de supprimer du bruit gaussien. Cependant, la mise en œuvre d'un filtre gaussien 2D nécessite des ressources informatiques importantes, et lorsqu'il s'agit d'applications en temps réel, l'efficacité de la mise en œuvre est cruciale. Cette thèse décrit la méthodologie pour mettre en œuvre un filtrage d'images gaussien en utilisant MATLAB et des applications DSP en temps réel sur FPGA en utilisant le concept de Co-simulation matériel-logiciel pour le traitement numérique d'images en utilisant l'outil de conception basé sur le modèle Mathworks Simulink / Xilinx System Generator (XSG) et le langage de description très haute (VHDL) en utilisant les cœurs de traitement d'images avancés inclus dans la bibliothèque IP Core lib. Les performances des architectures efficaces sont mises en œuvre sur le FPGA Spartan3e (xc3s500e). Le rapport signal sur bruit maximum (PSNR), l'indice de similarité structurale (SSIM) et l'utilisation des ressources FPGA sont utilisés pour discuter et comparer les résultats obtenus à partir du logiciel et du matériel.

Mots-clés : Filtrage d'images gaussiennes, FPGA, Co-simulation matérielle, Xilinx System Generator, VHDL, Xilinx IP CORE Generator.

الملخص

حدى التقنيات المفيدة جدًا في معالجة الصور هي تصفية الجاوس، وخاصة عند إزالة ضوضاء الجاوس. ومع ذلك، فإن تنفيذ تصفية الجاوس يتطلب موارد حوسبية كبيرة، وعندما يتعلق الأمر بتطبيقات الوقت الحقيقي، فإن الكفاءة في التنفيذ أمر بالغ الأهمية.

تصف هذه الرسالة منهجية تنفيذ تصفية الصور الجاوسية باستخدام (Matlab)وتطبيقات الوقت الحقيقي على (FPGA) باستخدام مفهوم المحاكاة المشتركة(co-simulation) بين الأجهزة والبرمجيات لمعالجة الصورالرقمية باستخدام أداة التصميم المبنية على نموذج ((XSG / Mathworks Simulink) ولغة وصف الأجهزة ذات الوصف العالي (VHDL)باستخدام نوى معالجة الصور المتقدمة المدرجة في المكتبة(IP core lib)

يتم تنفيذ أداء الهندسة الفعالة علىمتكاملة البرمجة القابلة للتجهيز((FPGA Spartan3e (xc3s500e). حيث يتم استخدام نسبة الإشارة إلى الضوضاء القصوى(PSNR)ومؤشر التشابه الهيكلي(SSIM) واستخدام الموارد (FPGA) لمناقشة ومقارنة النتائج التي تم الحصول عليها من البرامج والأجهزة .

Acknowledgements

First and before everything, we want to express our heartfelt appreciation to the Almighty Allah for His blessings, strength, and the intellectual capacity to comprehend, acquire knowledge, and document this research. Without His guidance and assistance, this thesis would not have come to fruition.

We would like to extend our sincere gratitude to our supervisor, Miss. Fouzia Hamadache, for her exceptional guidance, enthusiastic supervision, unwavering encouragement, and constant advice throughout the entire research process. Her unwavering dedication, vision, and motivation have served as a continuous source of inspiration. We are profoundly grateful for the invaluable opportunity to work and study under her tutelage.

Special recognition is due to the jury members for generously investing their valuable time in reading and analyzing our work. Their insightful comments and constructive feedback have significantly enriched the content of this thesis.

We also wish to express our appreciation to our colleagues and all those who have directly or indirectly supported us with their skills, guidance, and assistance. Your contributions have been immeasurable and have played an essential role in the successful completion of this research endeavor.

In conclusion, we are deeply grateful to our cherished family. Your love, prayers, and encouragement have served as a constant source of motivation, and we will forever remain indebted to you.

Table of contents

List of figures

Chapter I: 2D Gaussian filtering of images with MATLAB

Chapter II: Efficient gaussian image filtering Using Xilinx System Generator

Chapter III : 2D Gaussian image filtering with VHDL using IP Core Generator

List of tables

Chapter I:2D Gaussian filtering of images with MATLAB

Chapter II: Efficient gaussian image filtering Using Xilinx System Generator

Lists of abbreviations

bmp: Bitmap

BRAM: Block RAM

COE: Coefficient File

DCM: Digital Clock Manager

DSP: Digital Signal Processing

FIFO: First-In, First-Out

FPGA: Field Programmable Gate Array

FSM: Finite State Machine

FWFT: First Word Fall Through

gif: Graphics Interchange Format

GUI: Graphical User Interface

HDL: Hardware Description Language

IDE: Integrated Design Environment

IP Core: Intellectual Property Core

ISim: Integrated Simulator

jpeg: Joint Photographic Experts Group

JTAG: Joint Test Action Group

MATLAB: Matrix Laboratory

MSE: Mean Square Error

PNG: Portable Network Graphics

PSNR: Peak Signal to Noise Ratio

RAM: Random Access Memory

ROM: Read-Only Memory

RTL : Register Transfer Level

SSIM The structural similarity index measure (SSIM)

tif: Tagged Image File

USB: Universal Serial Bus

VGA: Video Graphics Array

VHDL: Very High-Speed Integrated Circuit Hardware Description Language

XSG: Xilinx System Generator

Introduction

Over the last few decades, the manipulation of digital images has aroused great interest in various fields such as medical and technological applications, yet these images are subject to a variety of noises that affect their quality. In image processing, filtering and noise reduction aim to restore the original image details as much as possible by eliminating unwanted noise. This is why filtering is one of the most widely used concepts in most image processing applications.

The need for real-time image processing means that algorithms must be implemented in hardware, and this is why Field Programmable Gate Array (FPGA) technology has recently emerged as a promising target for the implementation of algorithms suitable for image processing applications. This technology offers parallelism that significantly reduces processing time. High-level abstractions that can be automatically compiled into an FPGA are offered by Xilinx System Generator (XSG), a Simulink extension that enables hardware design. In addition, the Xilinx CORE Generator system produces and delivers parameterized cores optimized for Xilinx FPGAs, allowing for the faster development of high-density, high-performance designs.

The aim of this thesis is to design, model, simulate and synthesize Gaussian image filtering. The implementation on a Spartan3e FPGA reconfigurable logic platform (xc3s500e) will be realized using first hardware software Co-simulation through the Xilinx System Generator (XSG) tool and second the Very High-Level Description Language (VHDL) using the advanced image processing cores included in the IP Core lib library. Following is an outline of the thesis:

In Chapter 1, the process of gaussian image filtering is thoroughly explained, including how each point in the image is convolved with a Gaussian kernel. After doing a simulation with MATLAB, we will compare the image quality using the PSNR and SSIM measure.

Chapter 2 details the Gaussian image filtering model and how to read and write images using the Xilinx system generator with the MATLAB/Simulink platform. In addition, the FPGA implementation using hardware Co-simulation will be presented. The model can be co-simulated, provided that the requirements of the underlying hardware board are met. PSNR and SSIM will also be used to discuss the data and discuss the results that were obtained.

Chapter 3: provides the detailed implementation of gaussian image filtering in VHDL using Xilinx IP Core. In addition, a description of how to convert image and kernel matrix to .COE files which are loaded onto flexible Block Memory Generator core to create compact, high-performance memories, and how to design matrix convolution using finite state machine approach.

Finally, a conclusion brings to a close both our accomplishments and any potential follow-up effort.

Chapter I

2D Gaussian filtering of images with MATLAB

Chapter I: 2D Gaussian filtering of images with MATLAB

I.1.Introduction

In this chapter, we provide an overview and explanation of some concepts related to digital image processing, such as: noise in images and its sources, as well as a brief introduction to gaussian noise and gaussian filter. We focus in this chapter on specific elements such as spatial image filtering, convolution, gaussian mask and image quality assessment tools. PSNR and SSIM are two quality metrics that have been used to evaluate MATLAB results of 2D gaussian filtering.

I.2.Noise in images

I.2.1. Sources of image noise:

Image is affected by various sources of noise during the acquisition and transmission process. Typically, noise is assessed by examining the ratio of corrupted pixels in the image. Depending on the image's creation method, several factors contribute to increased noise in the original image: when scanning a photograph captured on film, noise can arise from the film grain, film damage, or introduced by the scanner or imaging system itself.The process of capturing and saving a digital image in its specific format through data collection mechanisms can introduce noise.Electronic acquisition or transmission of image data can also introduce noise.Noise can result from incorrect light penetration, caused by an improper opening of the device sensor, which affects the passage of light from the source to the device lens. Factors such as light levels and sensor temperature significantly contribute to noise creation[1].

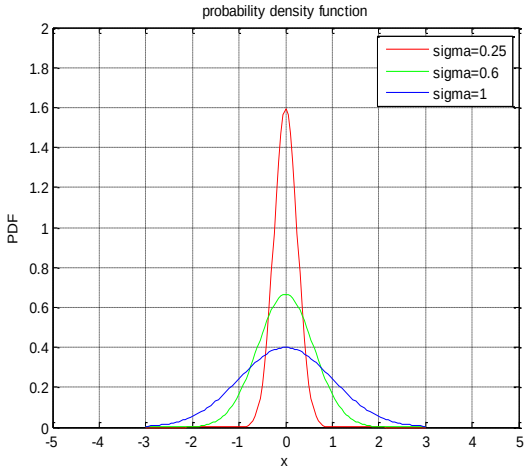
I.2.2.Gaussian noise

Gaussian noise is a type of noise that follows a gaussian probability distribution. It is characterized by a random signal value being added to each pixel of an image, which results in a slight variation of the pixel values.

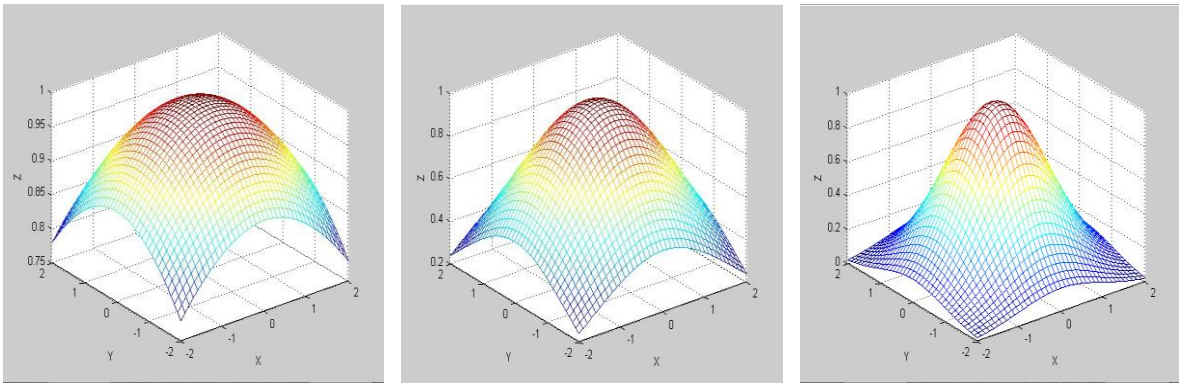
The probability density function (PDF) of gaussian noise is given by:

$$p(z) = \frac{1}{\sqrt{\sigma 2\pi}} e^{\frac{-(z-\bar{z})^2}{2\sigma^2}} \quad \text{(I.1)}$$

Where: z is the random variable, σ is the standard deviation, and $\bar{z}$ is the mean.



(a) 2D probability density function of gaussian noise *with different standard deviation*



Sigma =0.25 Sigma =0.6 Sigma =1

(b) 3D probability density function of gaussian noise *with different standard deviation*

Figure I. 1.2D and 3D probability density function of gaussian noise with different standard deviation

From the above figure It is evident that the distribution expands wider the higher the standard deviation. As a result, the variance can be viewed as a variable influencing the Gaussian probability density's width.

In the context of image processing, gaussian noise is often used to model the effect of electronic noise in image acquisition devices or random fluctuations in natural scenes. Gaussian noise is additive in nature, in other words each pixel in the noisy image is the sum of the real pixel value and a randomly generated gaussian-distributed noise value.

The noisy image is obtained by:

$$g(x,y) = f(x,y) + \eta(x,y) \tag{I.2}$$

Where $f(x,y)$is the original image, $n(x,y)$ represents the noise that was added to the image to create the noisy image $g(x,y)$, and (x,y)represents the pixel location.

It is important to remove or reduce gaussian noise from an image in order to improve its visual quality and enhance its features. Several image processing techniques, such as filtering and denoising, are used to achieve this goal[2].

I.3.Image filtering using gaussian filter

Image filtering is a fundamental technique in image processing and computer vision that is used to modify the appearance of an image by applying a filter or mask to the image. The filter is a mathematical function that is convolved with the image to produce a modified output image.A filter is characterized by a kernel, which is a small array applied to each pixel and its neighboring pixels within an image. Filters are typically categorized into two types: linear filters and non-linear filters. In this context, the spatial linear filters which are applied by convolution with a low pass filter convolution kernelare the ones being considered[3].

I.3.1. Convolution

Let's consider a monochrome image in which the function $f(i,j)$represents the light intensity of the pixel at coordinates(i,j). The digital convolution of this function with a two-dimensional impulse response $h(m,n)$results in a new image function $f(x,y)$This convolution can be written as:

$$f(i,j) = \sum_{m=-M}^{M} \sum_{n=-N}^{N} h(m,n)\, f(i-m, j-n) \qquad \text{(I.3)}$$

$f(x,y)$ is the weighted sum, using the coefficients $h(m,n)$, of the pixel intensities belonging to a neighborhood of the pixel at coordinates(i,j). This processing is referred to as localized. The impulse response is called the convolution mask.

The neighborhood of the pixel at coordinates (x,y) is represented as follows:

$(i\text{-}1, j\text{-}1)$	$(i, j\text{-}1)$	$(i\text{+}1, j\text{-}1)$
$(i\text{-}1, j)$	(i, j)	$(i\text{+}1, j)$
$(i\text{-}1, j\text{+}1)$	$(i, j\text{+}1)$	$(i\text{+}1, j\text{+}1)$

Figure I. 2.Pixel coordinates based on point (i,j).

Figure I.2 illustrates the application of a convolution mask on a monochrome digital image. The mask is moved across the entire original image to obtain a complete processed image.[4]

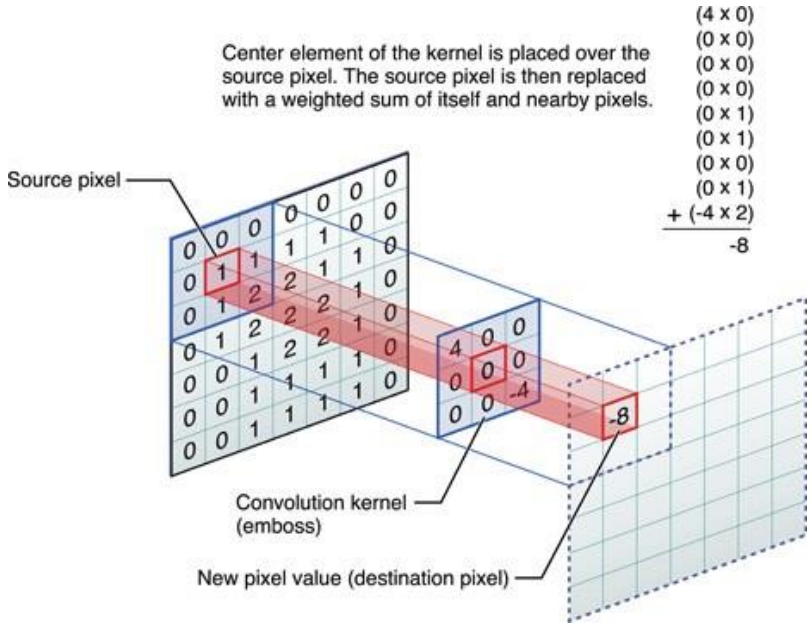


Figure I. 3.Image convolution with a filter kernel of size 3 × 3 [5]

I.3.2.Gaussian mask

A gaussian mask, also known as a gaussian filter, is a convolution kernel used in image processing to smooth or blur an image or to reduce noise. It is defined by a gaussian function, which is a bell-shaped curve that is symmetric about the center and has a standard deviation that determines the width of the curve. The values in the Gaussian mask decrease as the distance from the center pixel increases, with the highest weights at the center and lower weights toward the edges of the mask. The use of a Gaussian mask

can help to preserve edges and other high-frequency features in an image while reducing noise in the image. In digital image processing, the gaussian filter is widely used in numerous image processing applications, including edge detection, noise reduction, and feature extraction. It has good smoothing properties and can effectively remove high-frequency noise from the image while preserving the low-frequency structure of the image.[2]

According to Gonzalez and Wood (2018),the gaussian filter is defined as:

$$G(x,\ y)\ = \frac{1}{\left(2\pi\sigma^2\right)} \times\ e^{\frac{-\left(x^2+y^2\right)}{2\sigma^2}} \tag{I.4}$$

where σ is the standard deviation of the gaussian function, and x and y are the independent variables of the filter.

The Gaussian kernel matrix is the corresponding matrix structure that can be of different size and filtering can be implemented by convolving the input image matrix with the gaussian mask matrix.

1/16

1	2	1
2	4	2
1	2	1

1/273

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

1/1003

0	0	1	2	1	0	0
0	3	13	22	13	3	0
1	13	59	97	59	13	1
2	22	97	159	97	22	2
1	13	59	97	59	13	1
0	3	13	22	13	3	0
0	0	1	2	1	0	0

Mask of 3x3Mask of 5x5Mask of 7x7

Figure I. 4. Gaussian kernels $3\times3,\ 5\times5,7\times7$ *with* $\sigma=1$

I.3.3.Gaussian image filtering

Gaussian image filtering is produced by convolution between a noisy image and 2D gaussian mask which is usually odd and symmetrical $2n+1\times2n+1$with different size $3\times3,\ 5\times5,7\times7$, so the convolution become:

$$f\left(x,y\right)=\left(g*h\right)\left(x,y\right)=\sum_{i=-n}^{n}\sum_{j=-n}^{n}g\left(x+i,y+j\right)h\left(i,j\right) \tag{I.5}$$

Where $g(x,y)$ is the noisy image, $h(x,y)$ denotes the gaussian kernel , and $f(x,y)$ represents filtered image. [2]

I.4.Image quality assessment

I.4.1.PSNR

Peak Signal-to-Noise Ratio (PSNR) is a widely used objective quality metric in image and video compression, restoration, and processing. And itis the ratio of the reference signal to the distorted signal in the image, expressed in decibels. In general, we can say that the higher the value of PSNR, the closer the distorted image is to the original image thus the higher the image quality.

Mathematically, the Peak Signal-to-Noise Ratio (PSNR) for full reference image quality metrics can be expressed as follows:

$$PSNR = 10 \times log10\left((Mpp)^2 / MSE\right) \quad \text{(I.6)}$$

Where:

MPP represents the Maximum Possible Pixel value in an image. For example, in an 8-bit image, the MPP is calculated as $2^8 - 1 = 255$ pixels.

MSEdenotes the Mean Square Error between the filtered and the original images

$$MSE = \frac{1}{MN}\sum_{x=1}^{M}\sum_{y=1}^{N}\left(f(x,y) - f(x,y)\right)^2 \quad \text{(1.7)}$$

Where: *M and N* are the dimensions of the image[6]

I.4.2.SSIM

Structural Similarity Index (SSIM) is a widely used image quality assessment metric that quantifies the similarity between two images. It is based on the assumption that the human visual system is more sensitive to structural information in images such as texture, luminance, and contrast, rather than just simple pixel intensity values.

The SSIM index ranges from -1 to 1, with a value of 1 indicating perfect similarity between the two images. Higher SSIM values imply greater similarity between the two images.[7]

In a formal sense, SSIM conducts a thorough analysis of two images: a pristine reference image denoted as x, and a potentially corrupted version of the same image denoted as y. The structural similarity index can be calculated according to Eq.(I.12)[8]

$$SSIM\ (x\ ,y\ )=\frac{(2\mu_x\ \mu_y\ +C_1)(2\sigma_{x\ ,y}\ +C_2)}{(\mu_x{}^2+\mu_y{}^2+C_1)(\sigma_x{}^2+\sigma_y{}^2+C_2)} \tag{I.12}$$

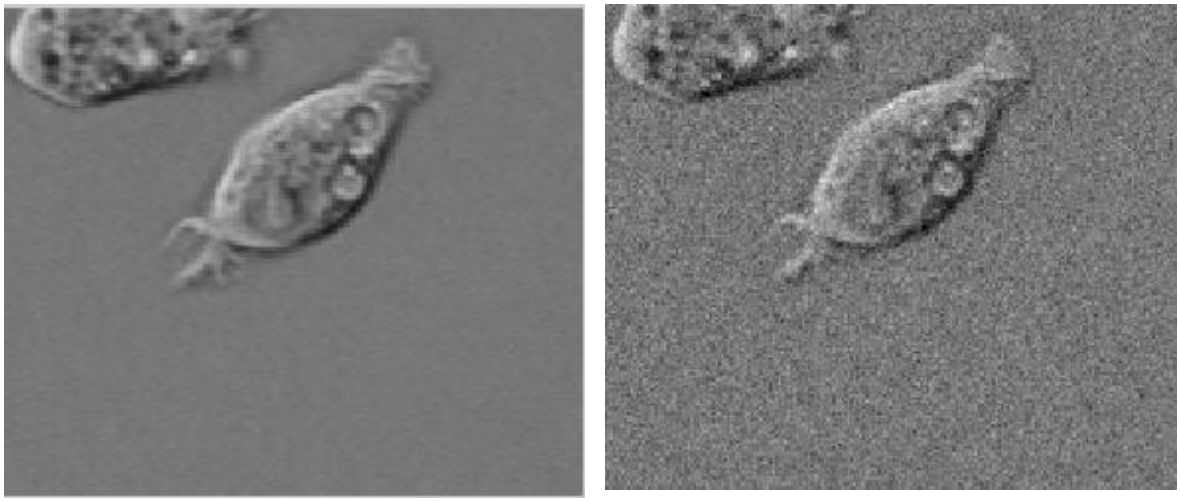
In the Eq.(I.12)The μ_x represents the average of x, μ_y represents the average of y. The variance of x is represented by $\sigma_x{}^2$, and the variance of y is represented by $\sigma_y{}^2$. C_1, C_2 are the variables to stabilize the division.

I.5.Matlab simulation results

The effectiveness of gaussian filtering for gaussian noise reduction has been examined using matrix laboratory software (MATLAB).

We applied a Gaussian filter with respectively (3x3), (5x5), and (7x7) kernel with σ=0.5, 1 and 7 on the image (cell.tiff) of size (159x191) affected by Gaussian noise with σ^2=0.003 and σ^2=0.09.

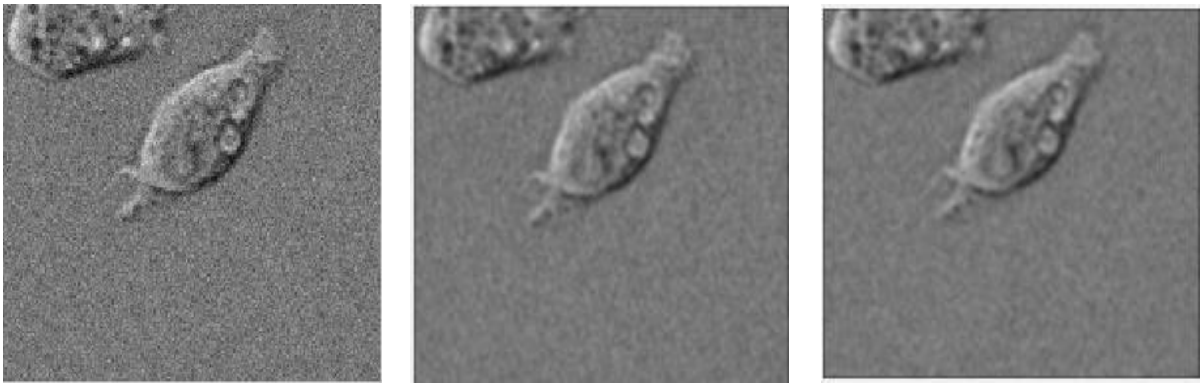
I.5.1. Gaussian filter for noise removal (Variance noise = 0.003)



a) Original image

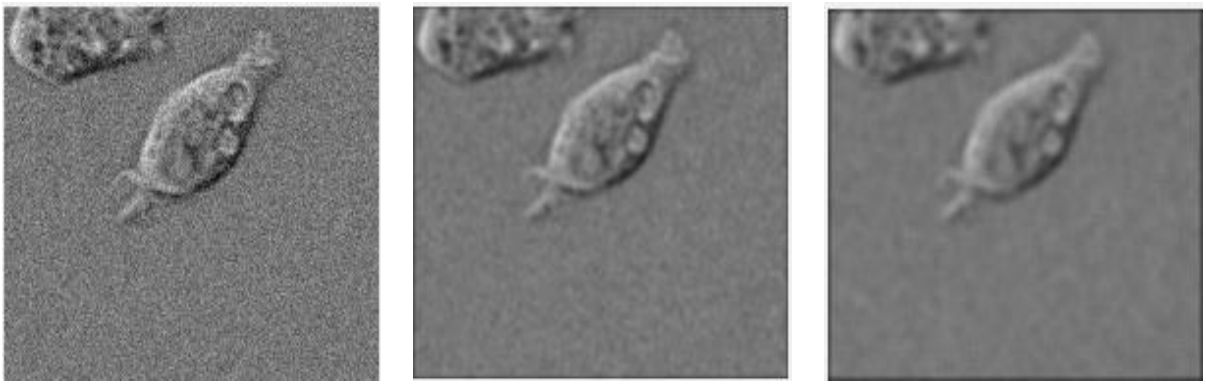
b) Noisy image PSNR=25.1638

Figure I. 5.Original and noisy image (Variance noise = 0.003)



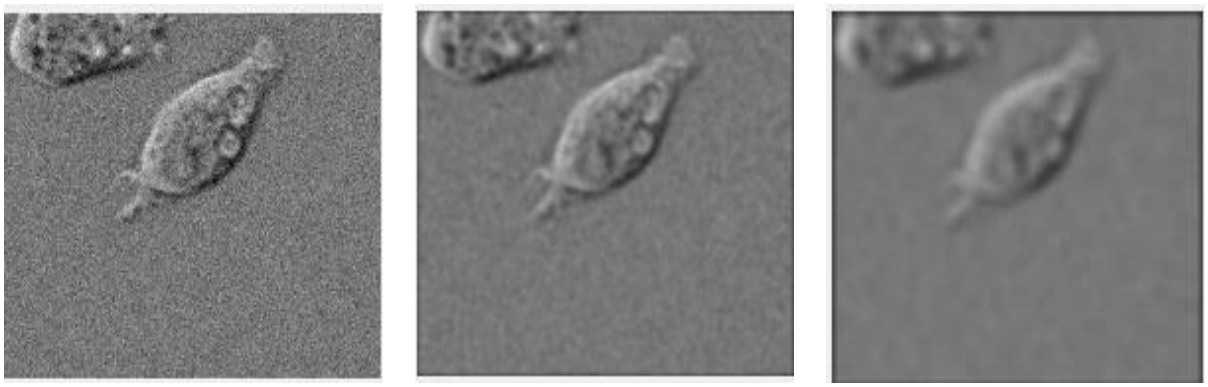
a) PSNR=21.7460 SSIM=0.2906 b) PSNR=22.7034 SSIM=0.5512 c) PSNR=21.7831 SSIM=0.6132

Figure I. 6.Filtered image with 3x3 kernel and a)σ= 0.2 b) σ=1c)σ=7



a) PSNR=18.33 SSIM=0.1848 b) PSNR=20.0511 SSIM=0.6014 c) PSNR=20.6363 SSIM=0.6706

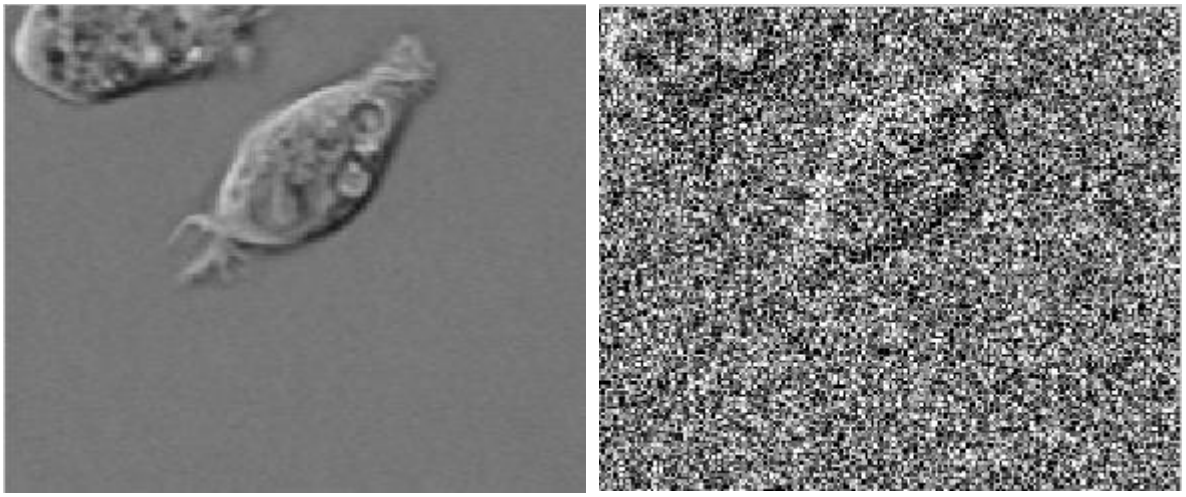
Figure I. 7.Filtered image with 5x5 kernel and a)σ= 0.2 b) σ=1 c) σ=7



a) PSNR=16.8281 SSIM=0.1789 b) PSNR=17.9386 SSIM=0.5699 c) PSNR=19.6255 SSIM=0.6725

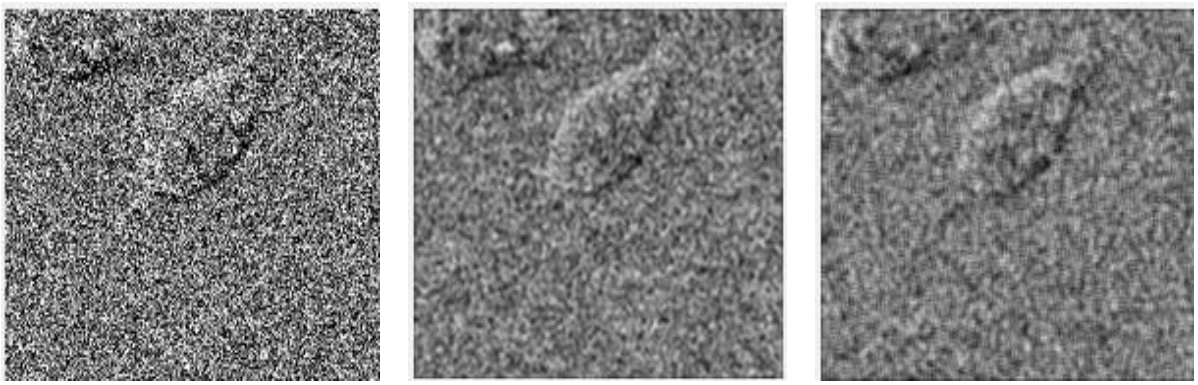
Figure I. 8.Filtered image with 7x7 kernel and a)σ= 0.2 b) σ=1c)σ=7

I.5.2. Gaussian filter for noise removal(Variance noise = 0.09)



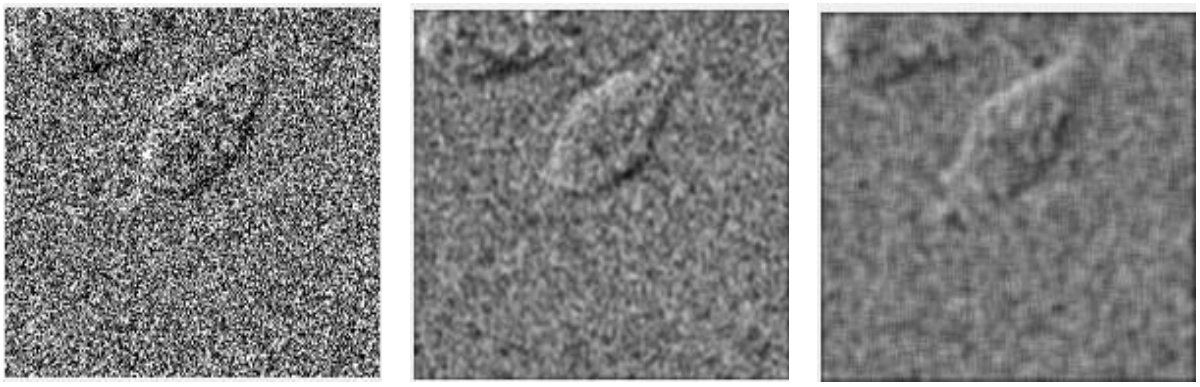
a) Original image b) Noisy image PSNR= 11.2691

Figure I. 9. .Original and noisy image (Variance noise = 0.09)



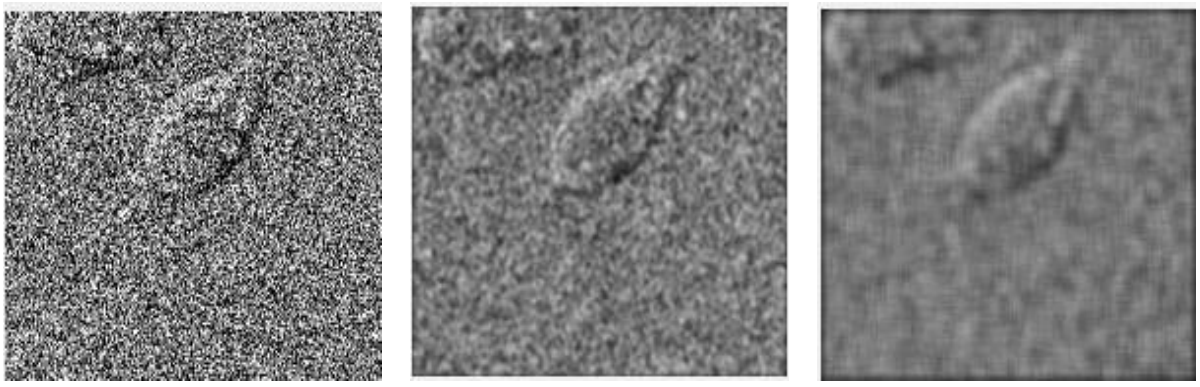
a) PSNR=11.1024 SSIM=0.0260 b) PSNR=17.0365 SSIM=0.0912 c) PSNR=18.5437 SSIM=0.1398

Figure I. 10. Filtered image with 3x3 kernel and a)σ= 0.2 b) σ=1c)σ=7



a) PSNR=10.7930 SSIM=0.088 b) PSNR=18.1547 SSIM=0.1591 c) PSNR=19.4412 SSIM=0.2998

Figure I. 11.Filtered image with 5x5 kernel and a)σ= 0.2 b) σ=1c)σ=7



a) PSNR=10.5856 SSIM=0.0104 b) PSNR=16.6997 SSIM=0.1494 c) PSNR=19.1455 SSIM=0.4472

Figure I. 12.Filtered image with 7x7 kernel and a)σ= 0.2 b) σ=1c)σ=7

I.5.3. Effect of kernel size and sigma of Gaussian filter

Table I.1 resume the PSNR and SSIM of filtered image for different sigma values and kernel size for variance noise=0.003

Table I. 1.PSNR and SSIM values for filtered images with sigma values and kernel size (Variance noise = 0.003)

Sigma kernel		0.2	1	7
Mask 3X3	PSNR	21.7460	22.7034	21.7831
	SSIM	0.2960	0.5512	0.6132
Mask 5X5	PSNR	18.33	20.0511	20.6363
	SSIM	0.1848	0.6014	0.6706
Mask 7X7	PSNR	16 .8281	17.9386	19.6255
	SSIM	0.1789	0.5699	0.6725

Table I.2 resume the PSNR and SSIM of filtered image for different sigma values and kernel size for variance noise= 0.09

Table I. 2. PSNR and SSIM values for filtered images with sigma values and kernel size (Variance noise = 0.09)

Sigma Kernel		0.2	1	7
Mask 3x3	PSNR	11.1024	17.0365	18.5437
	SSIM	0.0260	0.0912	0.1398
Mask 5x5	PSNR	10.7930	18.1547	19.4412
	SSIM	0.0088	0.1591	0.2997
Mask 7x7	PSNR	10.5856	16.6997	19.1455
	SSIM	0.0104	0.1494	0.4472

1. The effectiveness of the Gaussian filter in reducing noise can be controlled by adjusting its parameters, primarily the standard deviation (sigma) of the Gaussian kernel and the kernel matrix size.

a. We note that when we increase the sigma value for each mask size the PSNR and SSIM increase and the gaussian noise is reduced.

- for variance noise =0.003 (Mask 3x3)

for sigma kernel=0.2, PSNR = 21.7460 dB and SSIM=0.2960

for sigma kernel=7, PSNR = 21.7831dB and SSIM=0.6132

- for variance noise =0.09 (Mask 3x3)

for sigma kernel=0.2, PSNR = 11.1024 dB and SSIM=0.0260

for sigma kernel=7, PSNR = 18.5437 dB and SSIM=0.1398

Increasing the standard deviation of the mask continues to reduce and blur the intensity of the noise. A larger sigma value will result in a stronger smoothing effect, effectively reducing more noise but potentially blurring image details. Conversely, a smaller sigma value will have a weaker smoothing effect, preserving more details but reducing noise to a lesser extent.

b. When we increase the kernel matrix size, the PSNR and SSIM decrease which result in more blurring of image details.

- for variance noise =0.003 and kernel sigma =0.2

for mask 3x3, PSNR=21.74 dB and SSIM=0.2960

for mask 7x7, PSNR=16.82 dB and SSIM=0.1789

- for variance noise =0.09 and kernel sigma =0.2

for mask 3x3, PSNR=11.1024dB and SSIM=0.0260

for mask 7x7, PSNR=10.5856dB and SSIM=0.026 0

A larger kernel size implies a wider area of influence for each pixel during the filtering process. Consequently, larger kernels tend to provide stronger smoothing effects and noise reduction but may result in more blurring of image details. Conversely, a smaller kernel size limits the extent of neighboring pixels considered during filtering. This can preserve finer details in the image but may provide less effective noise reduction.

So, in our case, gaussian filtering is best suited filters for low gaussian noise with low sigma kernel and kernel size.

2. The effectiveness of the Gaussian filter in reducing Gaussian noise can be controlled also by the standard deviation (sigma) of the noise.

- for variance noise =0.003 (Mask 7x7)

for sigma kernel=7, PSNR = 19.6255 dB and SSIM=0.6725

- for variance noise =0.09 (Mask 7x7)

for sigma kernel=7, PSNR = 19.1455 dB and SSIM=0.4472

By increasing the standard deviation of the noise, you introduce more pronounced variations, which can make the noise more difficult to remove completely. In such cases, the Gaussian filter may still reduce the noise but may not eliminate it entirely. On the other hand, if the standard deviation of the noise is relatively low, the Gaussian filter can effectively reduce the noise and make it less noticeable in the image.

It's important to note that the choice of kernel size should be balanced according to the characteristics of the noise, desired level of smoothing, and the specific image being processed. Experimenting with different kernel sizes and observing their effects on the noise reduction and preservation of image details can help determine the optimal size for a particular application.

I.6.Conclusion

In this chapter, we get deeper in gaussian filtering of images. Indeed, after adding gaussian noise to image, we worked on a local processing which is linear spatial filtering using gaussian filter as a means of improving the image quality and removing noise from it. The implementation of gaussian filter to gray scale image cell is realized using MATLAB software. To evaluate the quality of the filtered images, we used two image quality metrics such as SSIM and PSNR. Then, the results of the implementation were discussed based on these metrics. When applying a Gaussian image filter to Gaussian noise, the filter will smooth out the noise, blur its details, preserve edges, and provide control over the strength of noise reduction

Chapter II

Efficient gaussian image filtering Using Xilinx System Generator

Chapter II: Efficient gaussian image filtering Using Xilinx System Generator

II.1.Introduction

This chapter focuses on the model of gaussian image filtering using Xilinx System Generator blocks (XSG), which integrates itself with the MATLAB based Simulink graphics environment and relieves the user of the textual Hardware Description Language (HDL) programming, and the implementation of the design targeting a Spartan3e device (xc3s500e) using hardware co-simulation. Peak Signal-to-Noise Ratio (PSNR) and the FPGA resource are both used to assess the quality of the filtered images.

II.2.Work environment

The work environment in the context of MATLAB and Simulink refers to the integrated software tools and graphical user interface provided by MathWorks for designing, simulating, and implementing various computational and signal processing systems. MATLAB is a programming language and development environment known for its numerical computing capabilities, while Simulink is a block diagram environment that enables the modeling and simulation of dynamic systems.[9]

II.3.Xilinx system generator (XSG)

The Xilinx's Generator is a System-level modeling tool from Xilinx that facilitates FPGA hardware design. It is a system-level modeling tool in which designs are captured in the DSP friendly Simulink modeling environment using a Xilinx specific blockset. All of the downstream FPGA implementation steps including synthesis and place and route are automatically performed to generate an FPGA programming file. Over 90 DSP building blocks are provided in the Xilinx DSP blockset for Simulink. These blocks leverage the Xilinx IP core generators to deliver optimized results for the selected device. System Generator provides many features such as System Resource Estimation to take full advantage of the FPGA resources, Hardware Co-Simulation and accelerated simulation through hardware in the loop Co-simulation; which give many orders of simulation performance increase. It also provides a system integration platform for the design of FPGAs that allows the RTL, Simulink, MATLAB and C/C++ components of a DSP system to come together in a single simulation and implementation environment. Figure II.1 presents the design flow of XSG.[10]

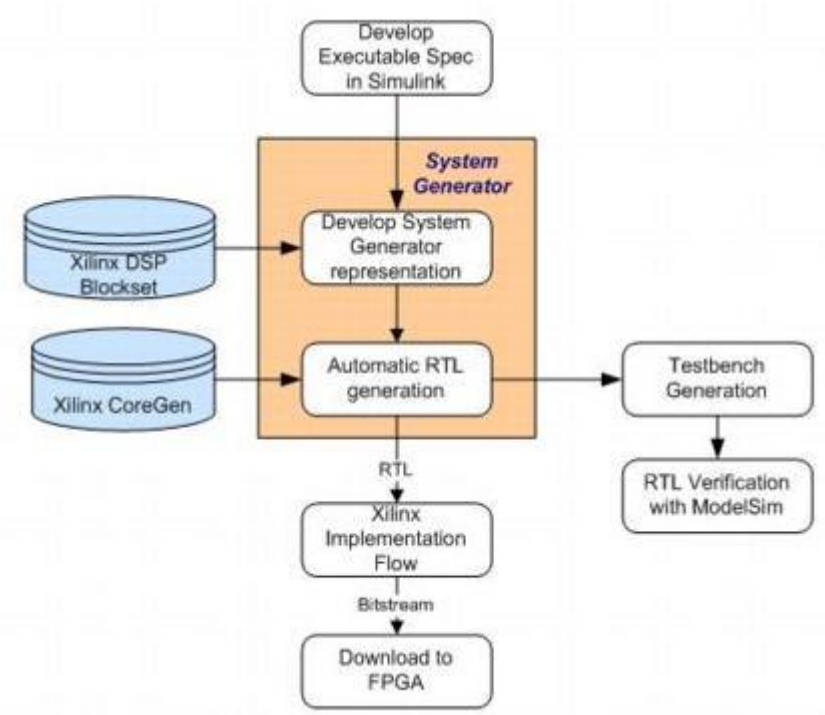


Figure II. 1.System Generator design flow

II.4.Gaussian image filtering design using Xilinx system generator

For real-time applications, the gaussian image filtering must be implemented in hardware. FPGA implementation can be carried out in a prototyping environment utilizing the Xilinx System Generator tool and MATLAB/Simulink. Fig. II.2 depicts the design flow for the hardware implementation of gaussian filtering using XSG.

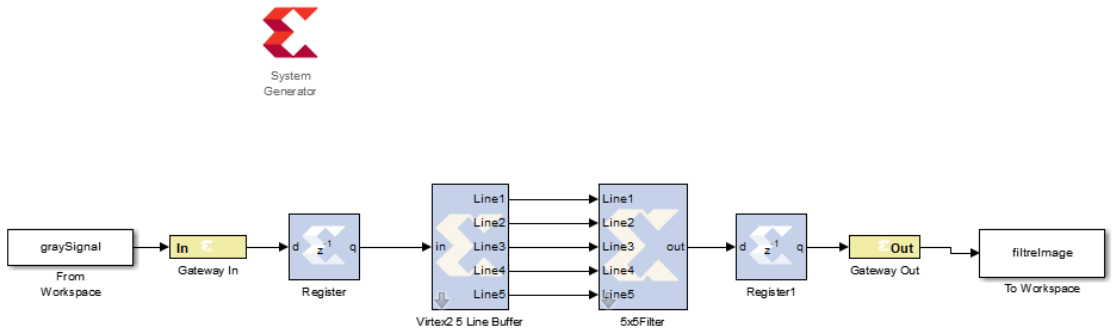


Figure II. 2.The Simulink design model

II.4.1.System Generator Token

The System Generator token functions as a control panel for managing simulation and system parameters, as well as calling upon the code generator for netlisting. In order for any

Simulink model with elements from the Xilinx Blockset to be valid, it must contain a minimum of one System Generator token. By introducing a System Generator token to a model, it becomes possible to determine how code generation and simulation are managed.[11]

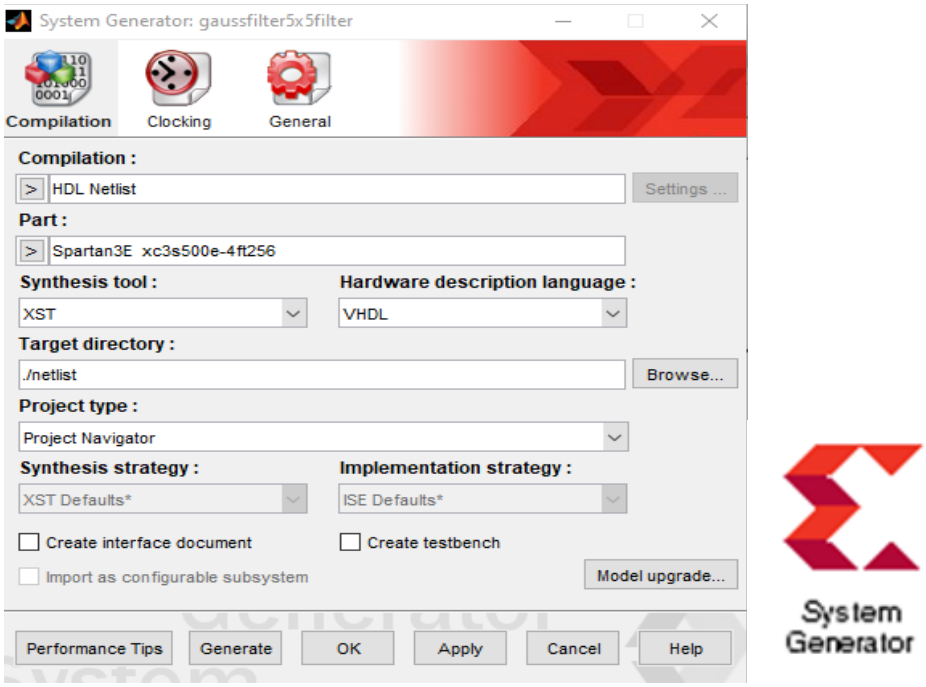
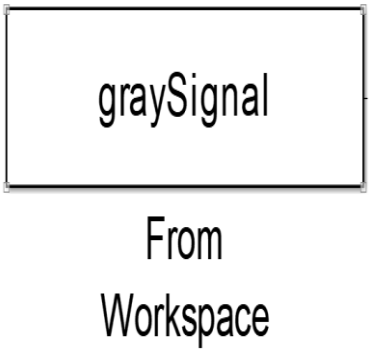


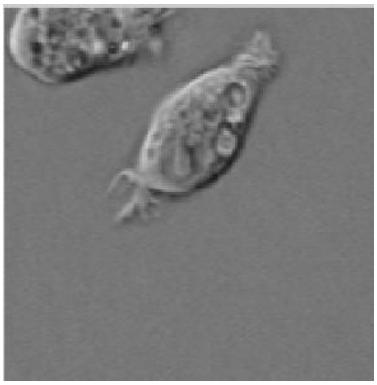
Figure II. 3.Block system Generator

II.4.2.Test image

The “From Workspace” block reads a signal from a Workspace and imports it to the Simulink workspace. The original grayscale image "Cell.tif" was noised with Gaussian noise (variance noise= 0.003, 0.09) and then converted to a grayscale signal that is a suitable data form for FPGA hardware (Figure II.4).



From Workspace block



Original image

Noisy image (Variance gaussian noise = 0.003)

Figure II. 4.Test image for filtering

II.4.3.Gateway in and Gateway out

The blocks named Xilinx Gateway In serve as inputs for the Xilinx segment in the Simulink design. They enable the conversion of Simulink's integer, double, and floating-point data types into the fixed-point type. Each block defines a high-level input port within the HDL design that is generated by System Generator.[11]

Figure II. 5.Gateway In block

The blocks labeled Xilinx Gateway Out are the outputs of the Xilinx component in the Simulink design. This block converts the fixed-point data types of System Generator into Simulink's integer, single, double, or floating-point data types.[11]

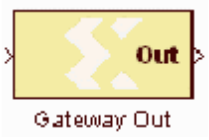


Figure II. 6.Gateway Out block

II.4.4.Register

The Xilinx Register block represents a register based on D flip-flops, which introduces a one-sample period latency. The block features a single input port for data, as well as an optional input reset port.[11]

Figure II. 7.Register block

II.4.5.Virtex 2 5 line buffer

The Xilinx Virtex2 5 Line Buffer reference block is responsible for buffering a sequence of pixels to create five lines of output, each of which is delayed by a different number of samples based on its length. Line 1 is delayed by four times the length of the line, with each subsequent line delayed by N fewer samples, until the final line (line 5) is a direct copy of the input.[11]

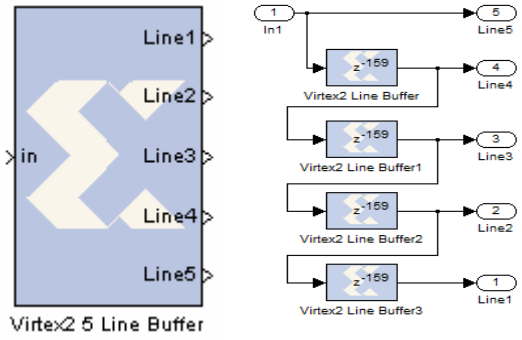
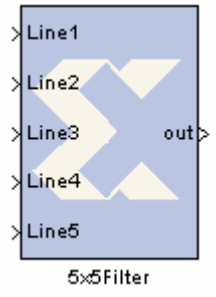


Figure II. 8.Virtex 2 5 line buffer block

II.4.6. 5x5 Filter

The Xilinx 5x5 Filter reference block is built upon 5 n-tap MAC FIR filters, which are available in the DSP library of the Xilinx Reference Blockset. The block includes nine distinct 2-D filters that can be used to filter grayscale images. We select a gaussian filter by adjusting the mask parameter on the 5x5 Filter block (Figure II. 10). The 2-D filter coefficients are stored in a block RAM, and the model doesn't make any particular optimizations for these coefficients. However, we can customize the filter by substituting our own coefficients and scale factor, which can be accomplished by modifying the mask under the initialization tab of the 5x5 filter block.[11]



gaussian = [1 1 2 1 1; ...
1 2 4 2 1; ...
2 4 8 4 2; ...
1 2 4 2 1; ...
1 1 2 1 1];
gaussianDiv = 1/52;

Figure II. 9.5x5 Filter block and the mask parameters

II.4.7. To Workspace block

The "To Workspace" block stores its input into the MATLAB workspace, and the resulting output is saved as an array or structure with a name specified by the block's "Variable name" parameter. The "Save format" parameter determines the format of the output data.[11]

Figure II. 10.To Workspace block

II.5.FPGA implementation using Hardware Co-Simulation

II.5.1. The choice of the compilation Target

Once the hardware board is installed, the starting point is to choose the new compilation target.Double click on System Generator token and in the compilation menu choose Hardware Co-Simulation and finally select the DEFB (Digital Electronics FPGA Board)platform which is Digital Electronics FPGA Target in the Hardware Co-Simulation submenu.

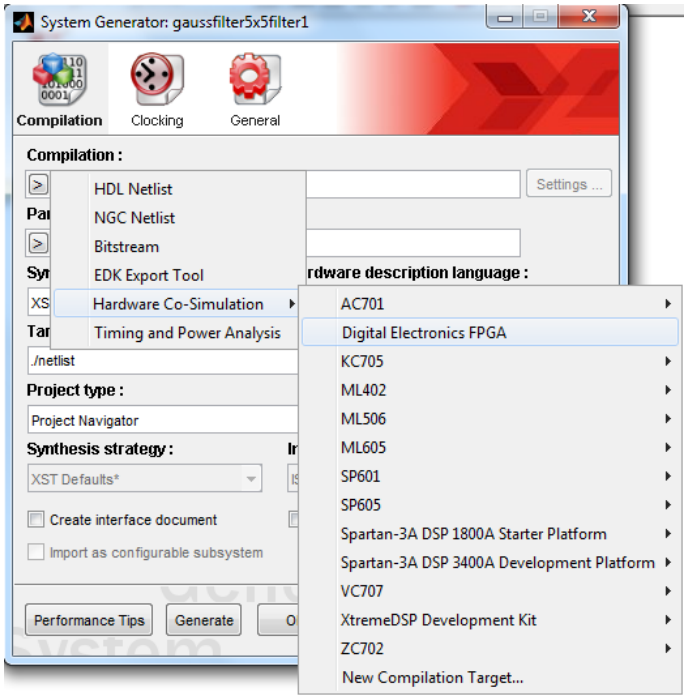


Figure II. 11. Digital Electronics FPGA compilation target

Once a compilation target is selected, the fields in the System Generator token dialog box are automatically adapted to align with the appropriate settings for the chosen target. System Generator preserves the specific dialog box configurations for each compilation target. These settings are saved when a new target is selected and restored when the target is recalled.[11]

II.5.2. Invoking the Code Generator

By clicking the **Generate** button in the System Generator block dialog box, the code generator is launched. For the design, the code generator creates an FPGA configuration bitstream appropriate for hardware Co-simulation.System Generator runs the downstream tools necessary to produce FPGA configuration files, in addition to producing HDL and netlist files for the model during compilation.The configuration bitstream includes the hardware related to the designed model as well as additional interfacing logic that enables System Generator to communicate with the design via a physical interface between the platform and the PC. System Generator may read from and write to the input and output ports on the design using the memory map interface that is part of this logic.[11]

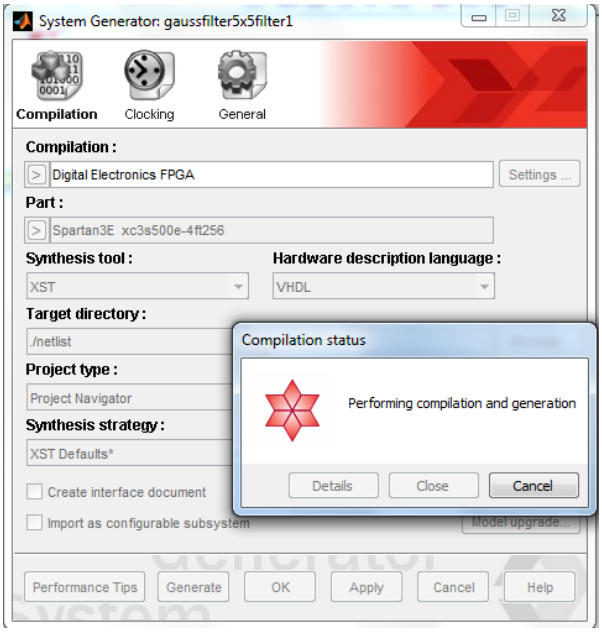


Figure II. 12.Invoking the Code Generator

II.5.3. Hardware Co-simulation block

Once System Generator has finished converting the design into an FPGA bitstream, it immediately generates a new hardware co-simulation block. A Simulink library was also created to store hardware co-simulation blocks.

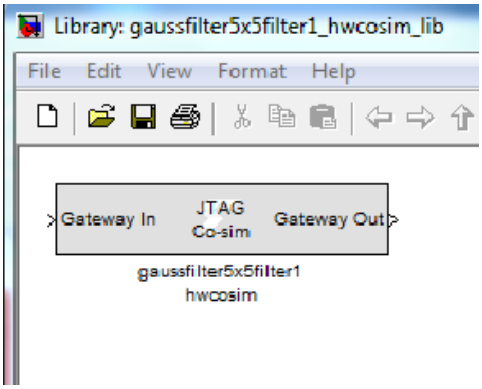


Figure II. 13.Hardware Co-simulation block

In fact, the hardware co-simulation module adopts the external interface of the model or subsystem derived from it. The same port names are used in the hardware co-simulation block as in the original subsystem. Connection types and speeds are also consistent with the original design.

The library block may now be copied and used in our System Generator design just like any other Simulink or System Generator block would be.

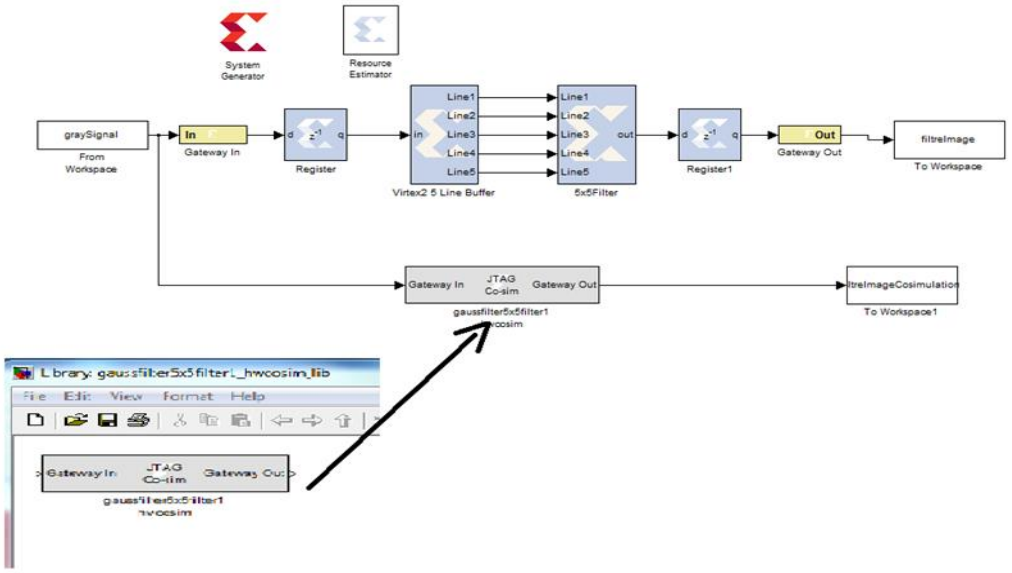


Figure II. 14.Hardware Co-Simulation block Insertion

The hardware Co-simulation block works like any other block in a Simulink design. This block allows interaction with the underlying FPGA platform during simulation and automate tasks such as device configuration, data transfer, and clocking. It consumes and generate the same types of signals used by other system generator blocks. When the input port of the hardware Co-simulation block receives a value, it transfers the appropriate data to the appropriate location in the hardware. Similarly, when an event occurs on an output port, the block retrieves data from the hardware. The system should appear as follows:

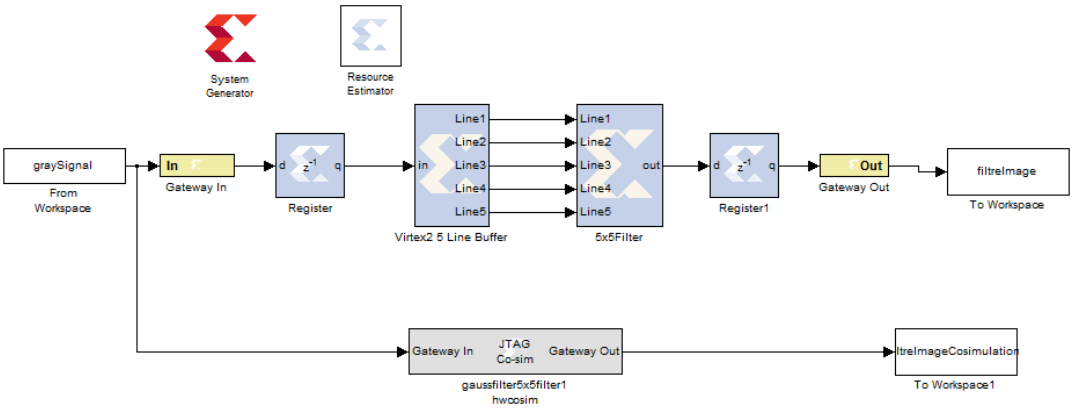


Figure II. 15.Hardware Co-Simulation model

Like other System Generator blocks, hardware co-simulation blocks offer parameter dialog boxes that allow users to configure them with various settings. The parameters available for a hardware co-simulation block depend on the specific FPGA platform it is implemented for, as each FPGA platform provides its own customized hardware co-simulation blocks.[11]

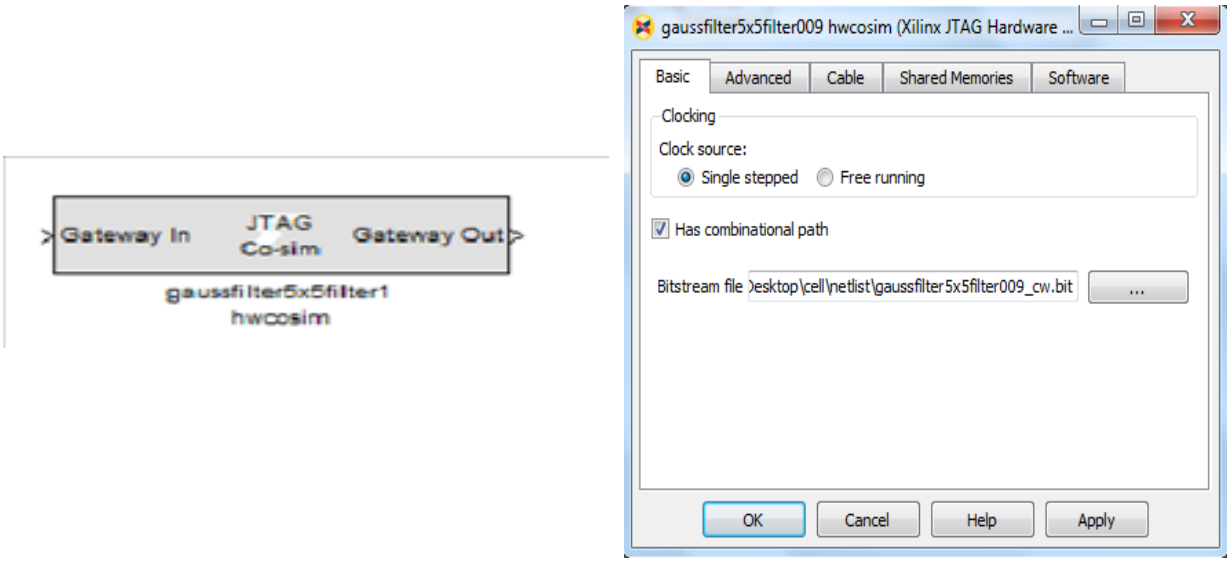


Figure II. 16.Hardware Co-Simulation block and it's dialog box

II.5.4.Co-Simulation Implementation

The implementation of the Co-simulation system can be simplified into two tasks: connecting the FPGA hardware to the host computer and running the co-simulation model. To

begin, assemble the FPGA hardware and ensure all components are set up correctly. Connect the board to the host computer using the serial to USB cable and the JTAG programming cable. This connection enables the Co-simulation to access the hardware for gaussian filtering algorithm. Once the board is properly connected, verify that the simulation time is set to the same value as in the original system design. Click the Start simulation button to initiate the simulation.The Co-simulation procedure is concluded at the end of the simulation run time, and a video viewer can be used to access the output signal.

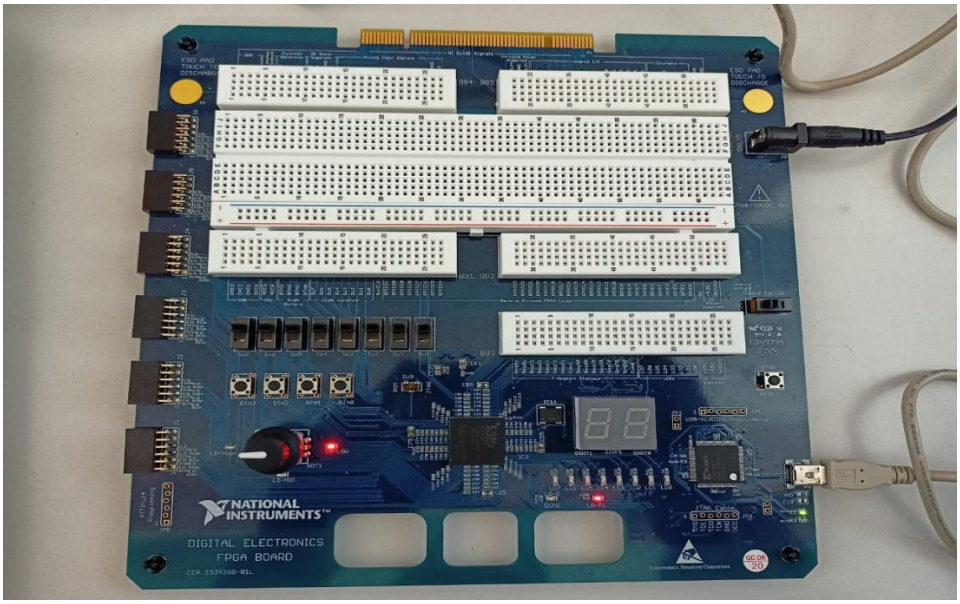


Figure II. 17.The connected Board

Figure II. 18.Co-simulation implementation

II.6.Hardware Software Co-simulation results

We applied a Gaussian filter with (5x5) kernel and σ=1 on the image (cell.tiff) affected by Gaussian noise with σ^2=0.003 and σ^2=0.09.

Figure II. 19 Hardware -Simulation filtered image with 5x5 kernel and variance noise=0.003

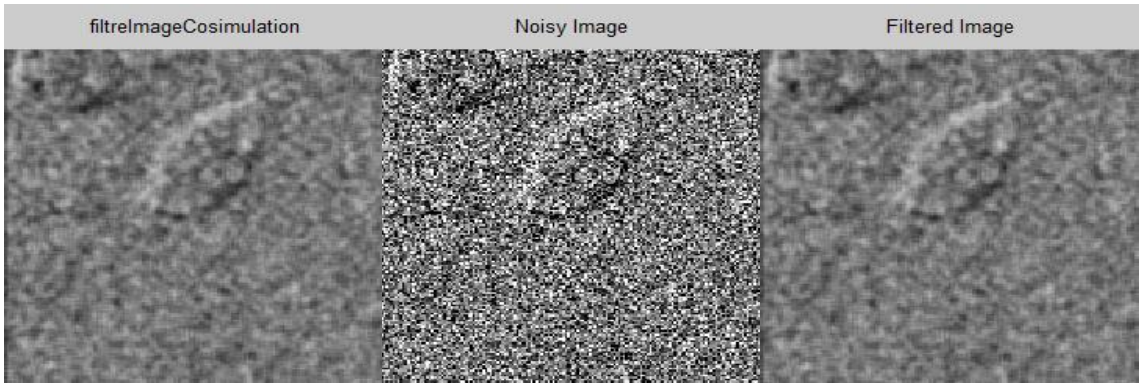


Figure II. 20 Hardware-simulation Filtered image with 5x5 kernel and variance noise=0.09

Table II.1 resumes the corresponding PSNR and SSIM of filtered images for different variance noise.

Table II. 1.Effect of gaussian noise on kernel size 5x5 with different variance noise

Variance noise		0.003	0.09
Mask 5x5	PSNR	31.0088	22.7034
	SSIM	0.8440	0.5512

The Gaussian filtering effect depends on the intensity of noise which is the standard deviation of the noise σ. The standard deviation of the noise represents the magnitude of the random variations in the noise. A higher standard deviation indicates a noisier image with larger variations in pixel values. We note that when we increase the sigma noise value, the PSNR and SSIM decrease. By increasing the noise's standard deviation, you add stronger variations, which may render the noise harder to eliminate altogether. In such instances, the Gaussian filter may still reduce noise but not entirely eliminate it. On the other hand, if the standard deviation of the noise is quite low, the Gaussian filter can successfully reduce the noise and make it less visible in the image. So, Gaussian filtering is best suited filters for low Gaussian noise. the Gaussian noise is reduced, but not completely removed.

For the target Spartan3e device (xc3s500e) FPGA device, the results are produced using XSG. The FPGA resource utilization description for the results of the gaussian filtering is shown in figure II.21.

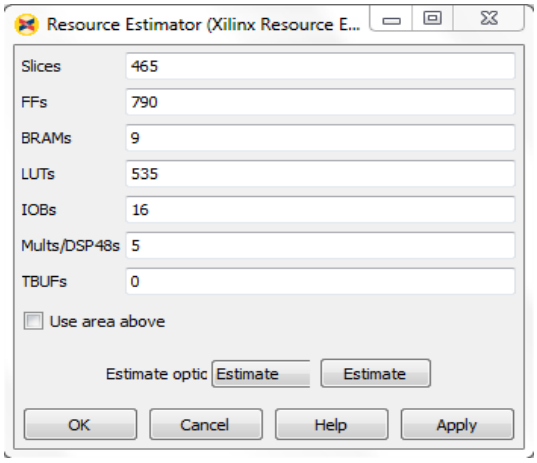


Figure II. 21.FPGA resource utilization summary

II.7. Conclusion

This chapter has focused on the efficient implementation of Gaussian image filtering using Xilinx System Generator. It's a powerful tool for designing digital systems on FPGA platforms. A design operating on an FPGA can be integrated directly into Simulink simulation thanks to System Generator's hardware co-simulation feature. When the system design is simulated in Simulink, the results for the compiled component are calculated in actual FPGA hardware, frequently yielding a substantially faster simulation time while verifying the functionality of the hardware.

.

Chapter III

2D Gaussian filtering with VHDL using IP Core Generator

Chapter III:2D Gaussian image filtering with VHDL using IP Core Generator

III.1.Introduction

In this chapter, we will explore image filtering in VHDL, focusing on the role of IP cores in enhancing the efficiency and effectiveness of image processing tasks. VHDL, or Very High-Speed Integrated Circuit Hardware Description Language, provides a powerful framework for designing and implementing digital systems, including image processing algorithms. IP cores, on the other hand, serve as pre-designed and pre-verified circuits that offer ready-made solutions for specific image processing functions. By leveraging VHDL and IP cores, engineers can accelerate the development process, optimize performance, and promote design reusability in image processing applications.

III.2.FPGA memory blocks

In Xilinx FPGAs, Block RAM (BRAM) is a dedicated two-port memory that can store up to 36Kb of data. These FPGA devices have multiple BRAM blocks available, each consisting of a configurable lookup table and a small logic block. While primarily used for logic functions, the lookup table can be reconfigured to serve as a small amount of RAM. By combining several BRAM blocks, a distributed RAM can be created, providing a larger storage capacity. BRAM operates synchronously, with read and write operations synchronized with the clock input signal and controlled by the read/write enable ports. In our specific case, BRAM2 stores the test image data generated with MATLAB using a .coe file, while BRAM1 holds the .coe file for the Gaussian mask accessed by the control module. Lastly, BRAM3 is responsible for storing the filtered data.[12]

III.2.1. Single port BRAM

The Single-Port Block Memory module is designed to meet user requirements in terms of width and depth. The read and write operations from and to the memory are based on the clock input signal's changing edge since BRAM is synchronous.During operation, the Block Memory performs all memory operations on the active edge of the clock input (CLK).The data given at the port's data input is stored in memory at the location specified by the port's address input during Write Operation(WE asserted).[13]

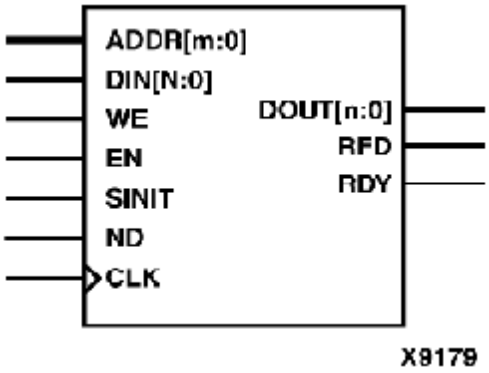


Figure.III. 1.Core Schematic Symbol (Single-Port Block Memory module)

III.2.2.Dual Port BLOCK MEMORY

The Dual Port Block RAM features two separate access ports that allow simultaneous access to a shared memory pool. Each access port can be configured independently, enabling straightforward dual-port memory functionality or the option for data formatting capabilities. Both ports offer read and write access and are functionally identical. However, it is important to avoid simultaneous operations at the same memory location, except for simultaneous reads. When reading from and writing to the same location simultaneously, the correct data is written into the memory, but invalid data is presented at the reading port.[13]

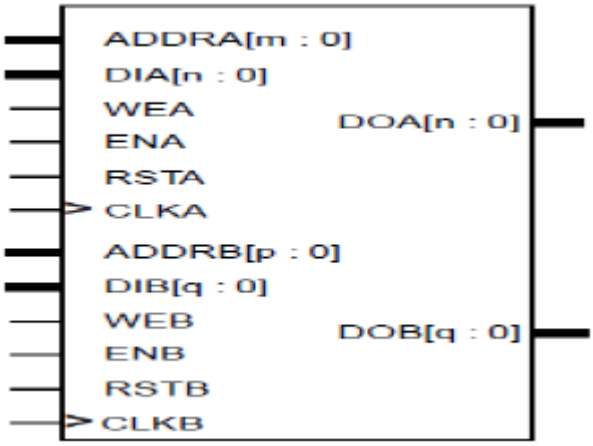
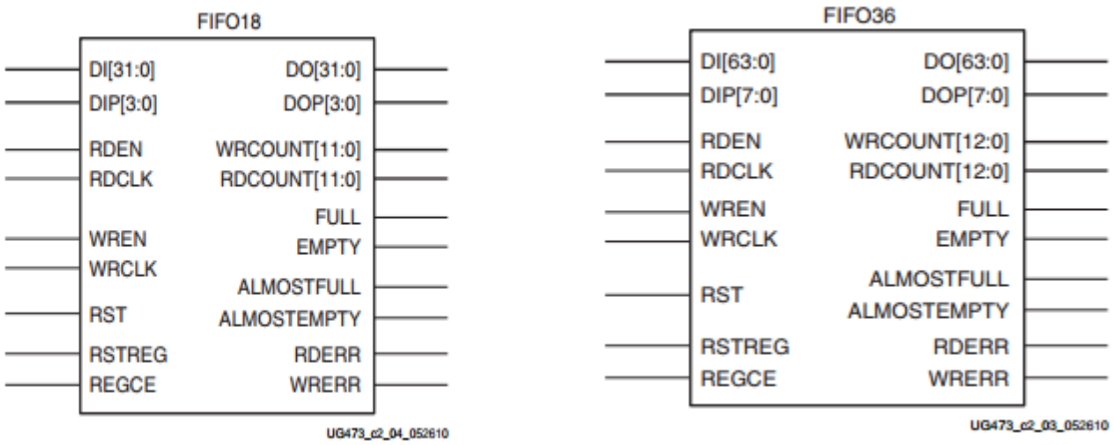


Figure.III. 2.Core Schematic Symbol (Dual Port Block RAM)

III.2.3.FIFO MEMORY

FIFO (First-In, First-Out) is an extension of Block RAM (BRAM) that operates on a first in, first out basis. It follows the principle that the data stored first is retrieved first. When a rising edge of the clock signal is detected, the available memory location is written with the data present on the data bus, and the data from the last written memory location becomes available on the output bus. FIFOs can be configured as either 18 Kbits or 36 Kbits of memory. The configuration options and operation modes are depicted below.[14]



(a) (b)

Figure.III. 3.FIFO configurations. (a) FIFO-18 (b) FIFO-36

III.3.Xilinx CORE Generator

The Xilinx CORE Generator system is a tool that allows users to access IPs (Intellectual Properties) customized for Xilinx FPGAs in order to construct FPGA chips faster and with higher densities. A full library of Xilinx LogiCORE IPs is included with the ISE Foundation software that comes with the CORE generator. These consist of memory, storage components, mathematical operations, and fundamental components.

Xilinx offers the flexible Block Memory Generator core. This core provides the option for single port and dual port block memories, which differ in their operating mode selection.

In the image filtering workflow, the MATLAB tool is utilized to convert the processed image into the .coe file format. Subsequently, the Xilinx Core Generator is employed to store the coefficient file (.coe) in a single port Block ROM. The width and depth of the image are defined during this process. Finally, the filtered image is then written to a text file and read

using MATLAB. Complex designs for image processing, storage, and display on Xilinx FPGAs can be efficiently developed by utilizing the Xilinx CORE generator, MATLAB, and FPGA technology[13][14].

III.4.Hardware implementation of gaussian imagefiltering

III.4.1.Top level design flowchart

Figure III. 4 depicts the block diagram of the image filtering process. Initially, the input image and the Gaussian mask are retrieved and saved using MATLAB. These values are then converted into a vector and stored in a text file with a *.coe extension, utilizing the capabilities of the MATLAB tool. The text file containing the Gaussian mask is stored in BRAM1, while the text file containing the image is stored in BRAM2. Subsequently, the VHDL tool is employed to perform the convolution operation between the pixel values stored in BRAM1 and BRAM2. The resulting data is saved in another block called BRAM3. Finally, using a FIFO block, this convoluted output is written to a text file which is converted back into image format using the MATLAB tool to display the obtained results. In the subsequent step, each block of the diagram presented in Figure III. 1is further defined and explained.[14]

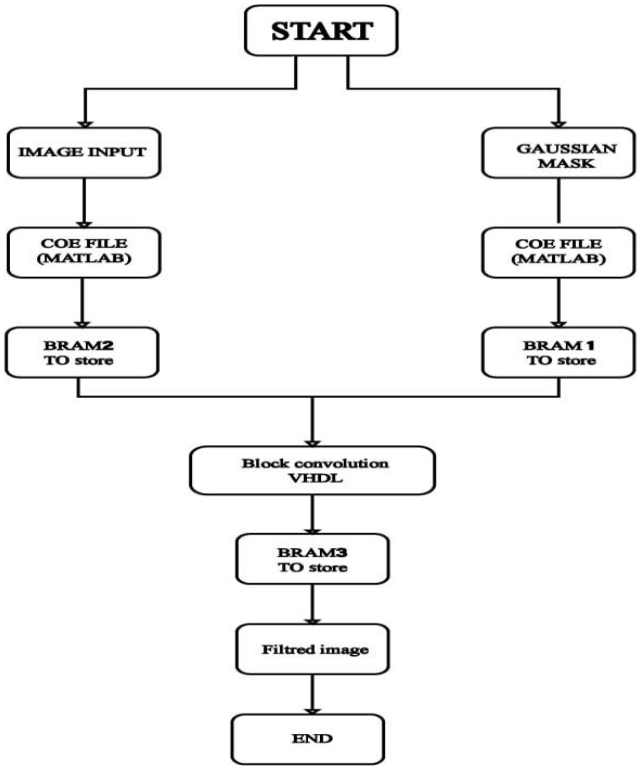


Figure.III. 4.Block diagram of image filtering [12]

III.4.2.Generating COE files

The process begins by reading the image, resizing it to the desired dimensions, in our case, a 90x90 matrix, then the image is padded on the sides with zeros, resulting in a matrix of dimensions 94x94. The modified matrix is then saved as a COE file named "image. COE" using a MATLAB function. A similar process is followed to generate the COE file for the kernel matrix. These COE files containing the image and kernel matrices are then loaded into the BRAM memory.[14]

III.4.3. Block memory generator to store image pixels

The IP Core Generator and Architecture Wizard, known as "IP(Core Generator & Architecture wizard)," can be accessed through the ISE (Integrated Software Environment) interface, as demonstrated in the figure below:

To initialize the IP core and load a COE (Coefficient) file into the BRAM (Block RAM), follow these step-by-step instructions:

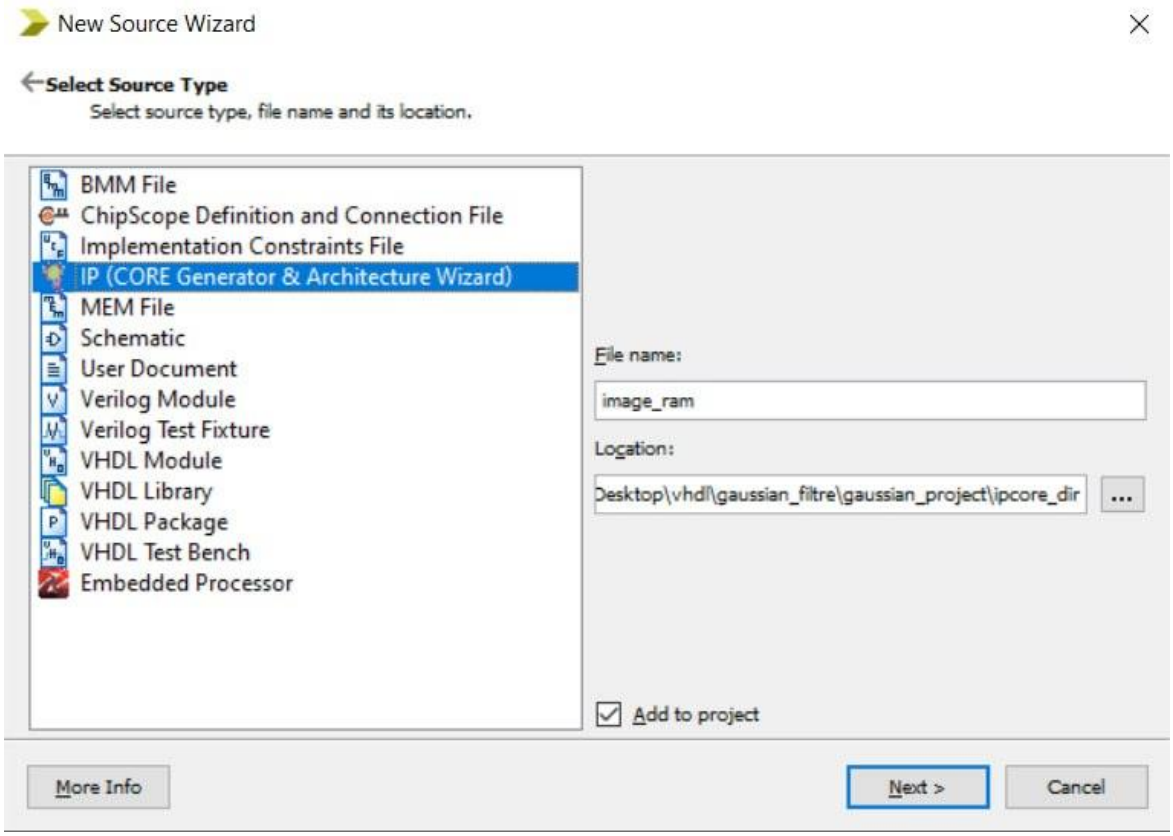


Figure.III. 5.Selecting IPCore Generator & Architecturefrom ISE with File name (image_ram)

The next step is to look for the specific option or button labeled as "Block Memory Generator" and click on it

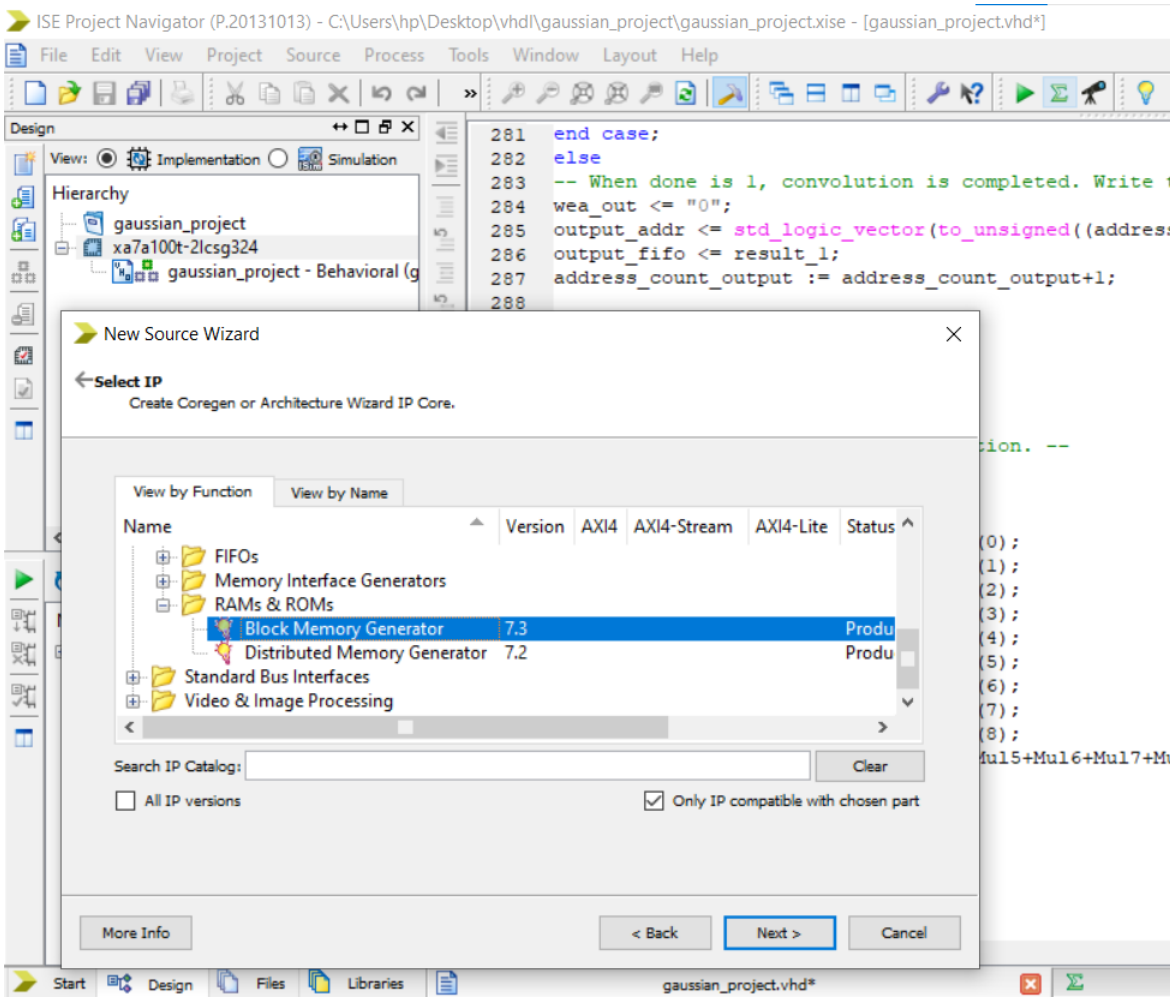


Figure.III. 6.Selection of "Block memory Generator" IP Core

Upon selecting the "Block Memory Generator," a configuration window or interface will appear and we configure the settings for the memory generator, such as the memory size, data width, and other relevant parameters

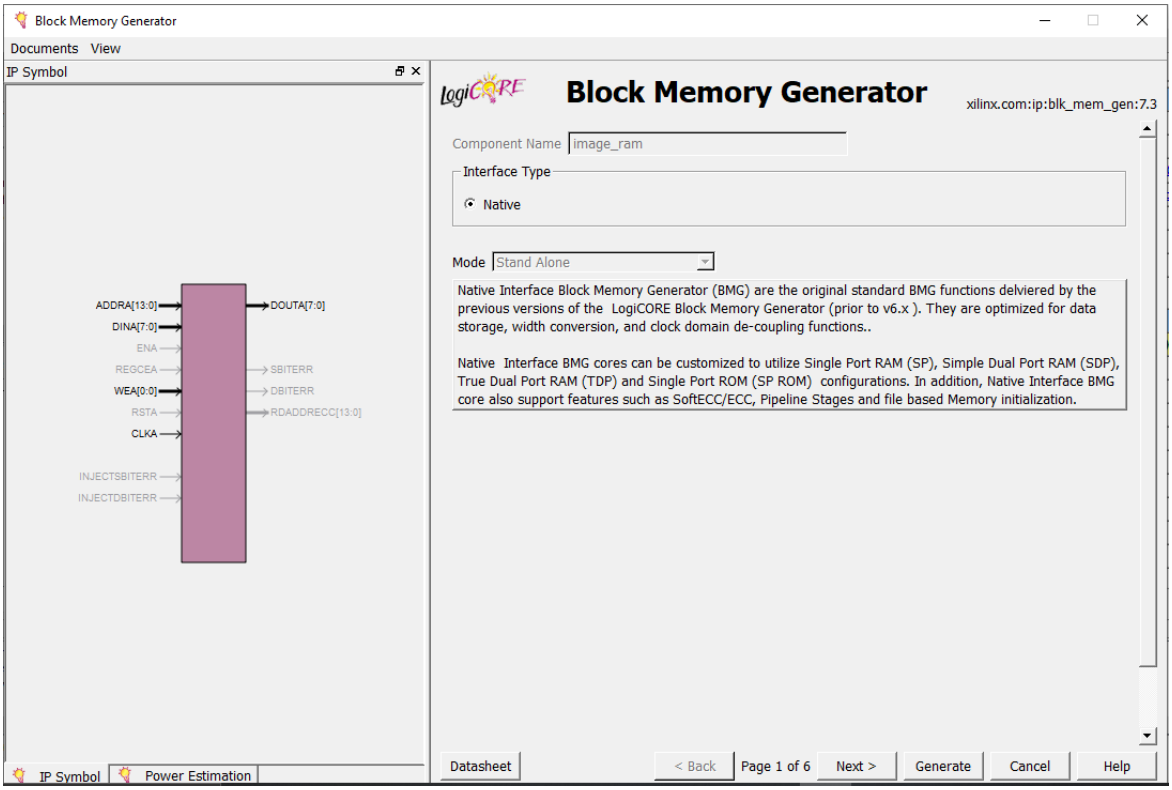


Figure.III. 7."Block memory Generator" IP core Wizard

An IP symbol and other memory kinds are displayed when the Block Memory Generator wizard is launched. Among the memory types, the "Single Port ROM" is chosen, and the values for the read width and read depth are determined based on the horizontal width and vertical length of the Image being processed.

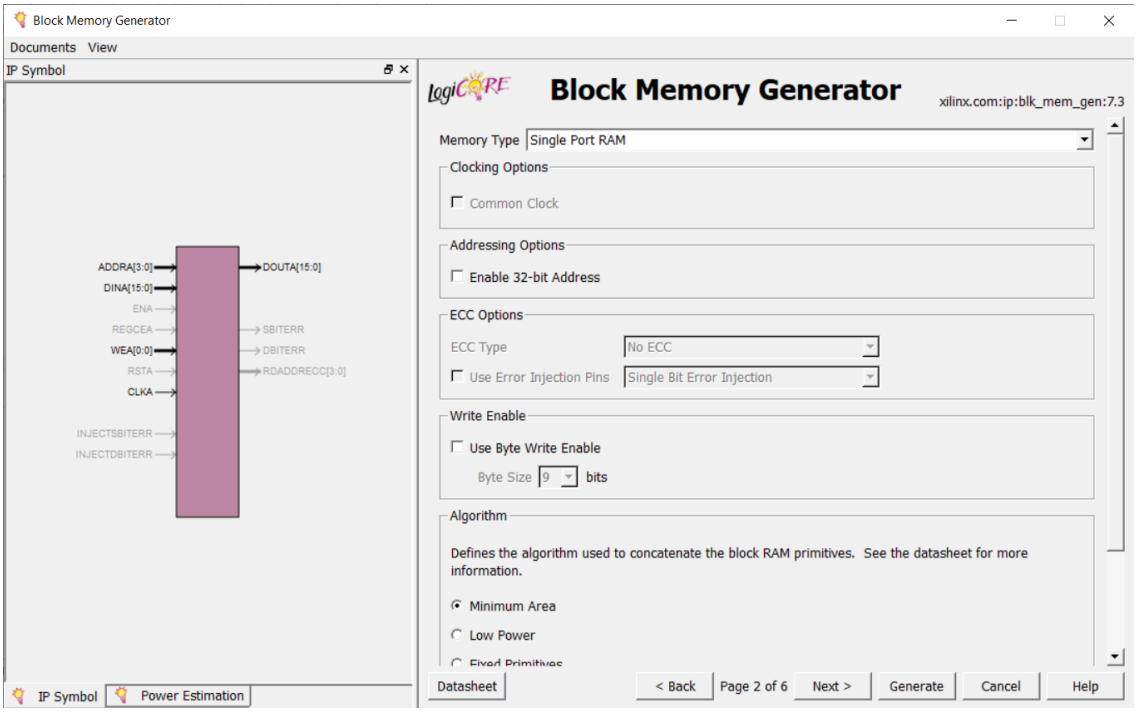


Figure.III. 8. "Block memory Generator" IP core Wizard

The read width and read depth values are established by referencing the horizontal width and vertical length of the Image being processed which is 8 bits width and (94x94=8836) depth

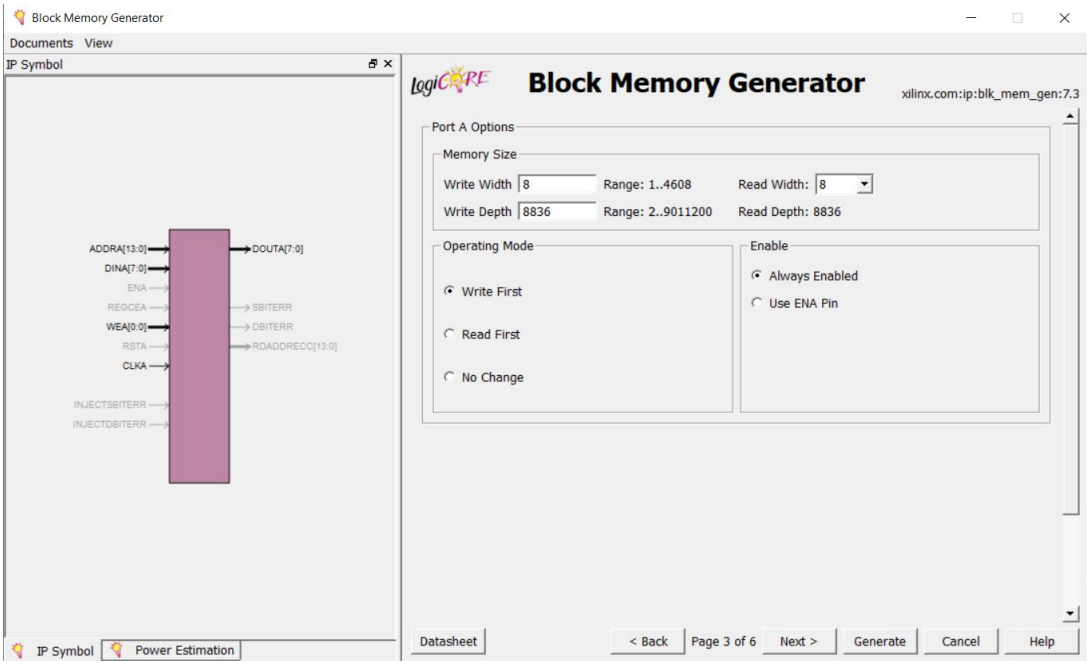


Figure.III. 9.Configure the settings of memory size (width and depth)

Image0003.coe file, short for "Coefficient file," contains the data contents of the Block Memory, specifically tailored to the specified read depth and read width values of the image. In this case, the available image size is 94x94, and the data is stored as a .coe file within a single port Block ROM using Xilinx Core Generator.

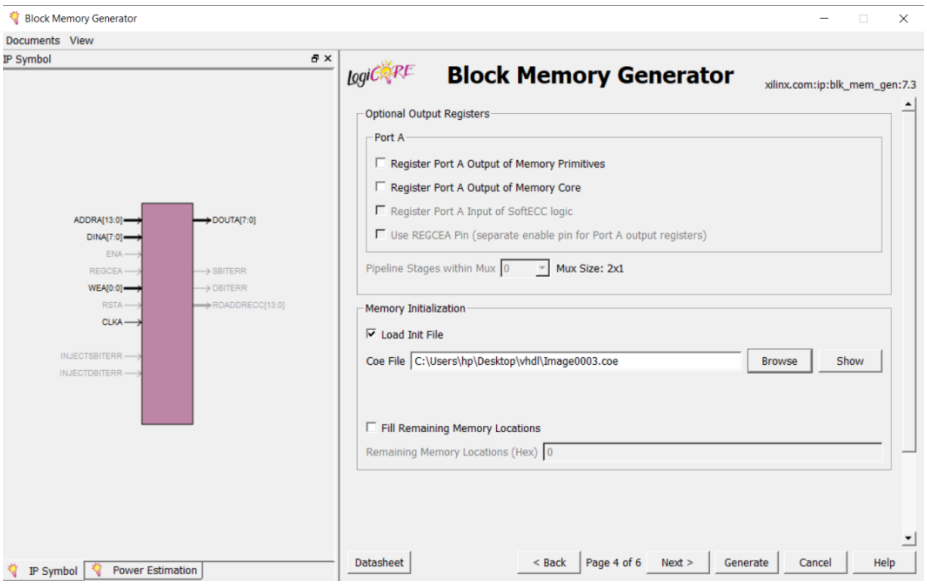


Figure.III. 10. Block memory Generator" Initializing Image0003".coe file

After proceeding to the next step, click on the "Generate" button in the Block Memory Generator wizard. This action initiates the generation process of the IP core. Once completed, a message stating "IP core successfully created" will be displayed in the Xilinx ISE environment.

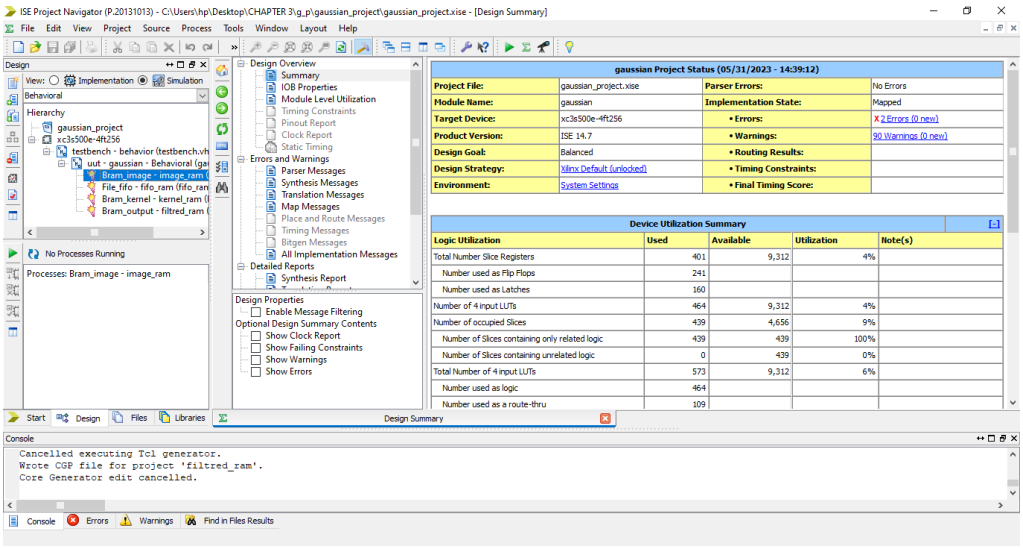


Figure.III. 11.Successful Creation of IP Core (bram_image)

III.4.3. Block memory generator to store gaussian kernel elements

To store the Gaussian kernel COE file, we follow the same steps as we did to store the image COE file with the following modifications**:**

The first step is chosen “IP(CORE Generator & Architecture Wizard) click on it and make name (Kernel_ram) for the File.

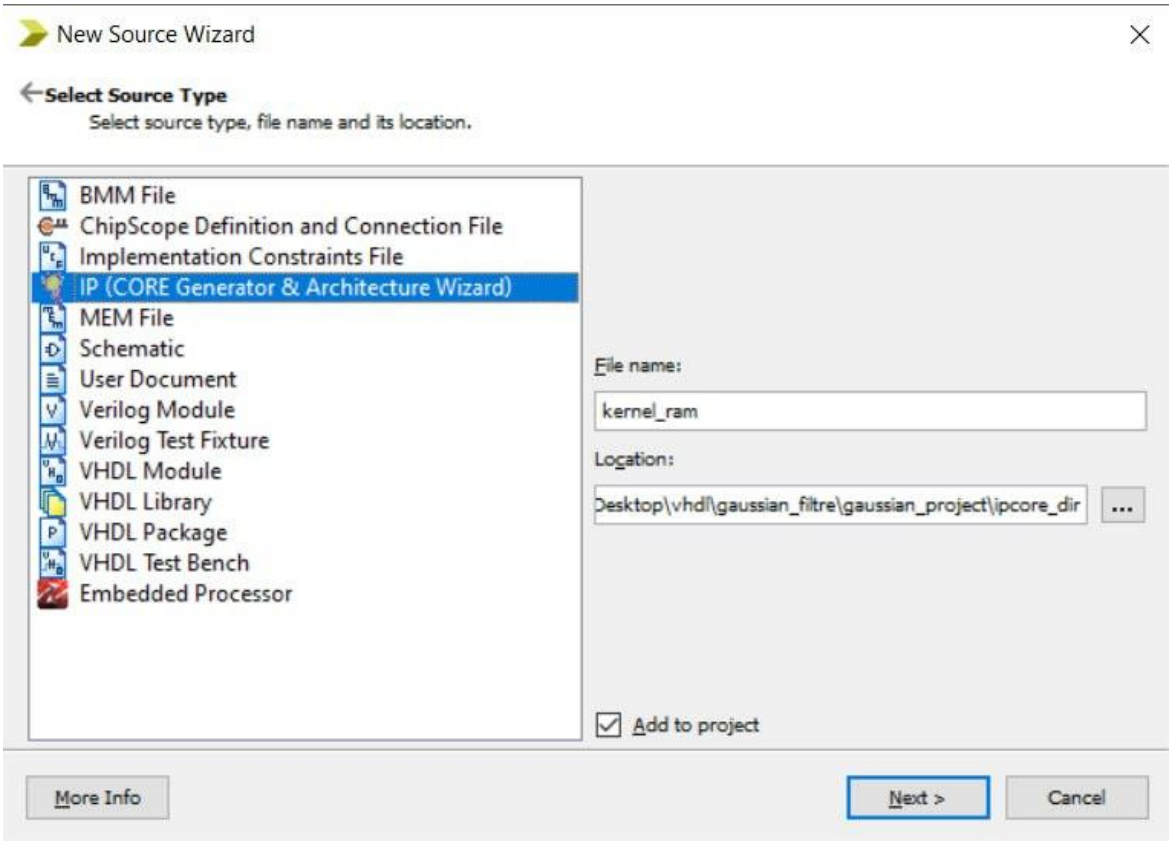


Figure.III. 12.Selecting IP Core Generator & Architecture with File name (kernel_ram)

And now we Configure the settings of Memory size (widht & depht), we choose kernel (3x3).

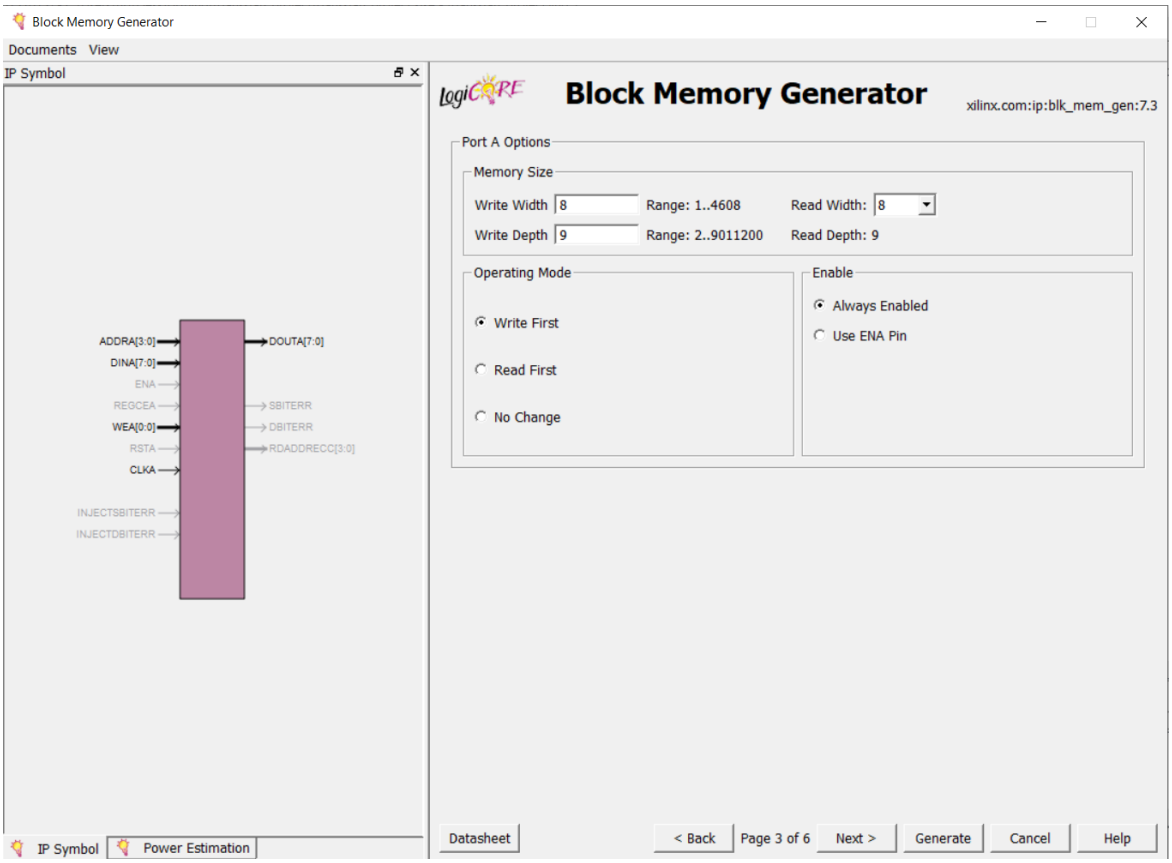


Figure.III. 13..Configure the settings of memory size (width and depth)

Kernel.COE file contains the contents ofthe Block Memory for the specified read depth and read widthvalues of the kernel (3x3).which is 8 bits width and (3x3=9) depth

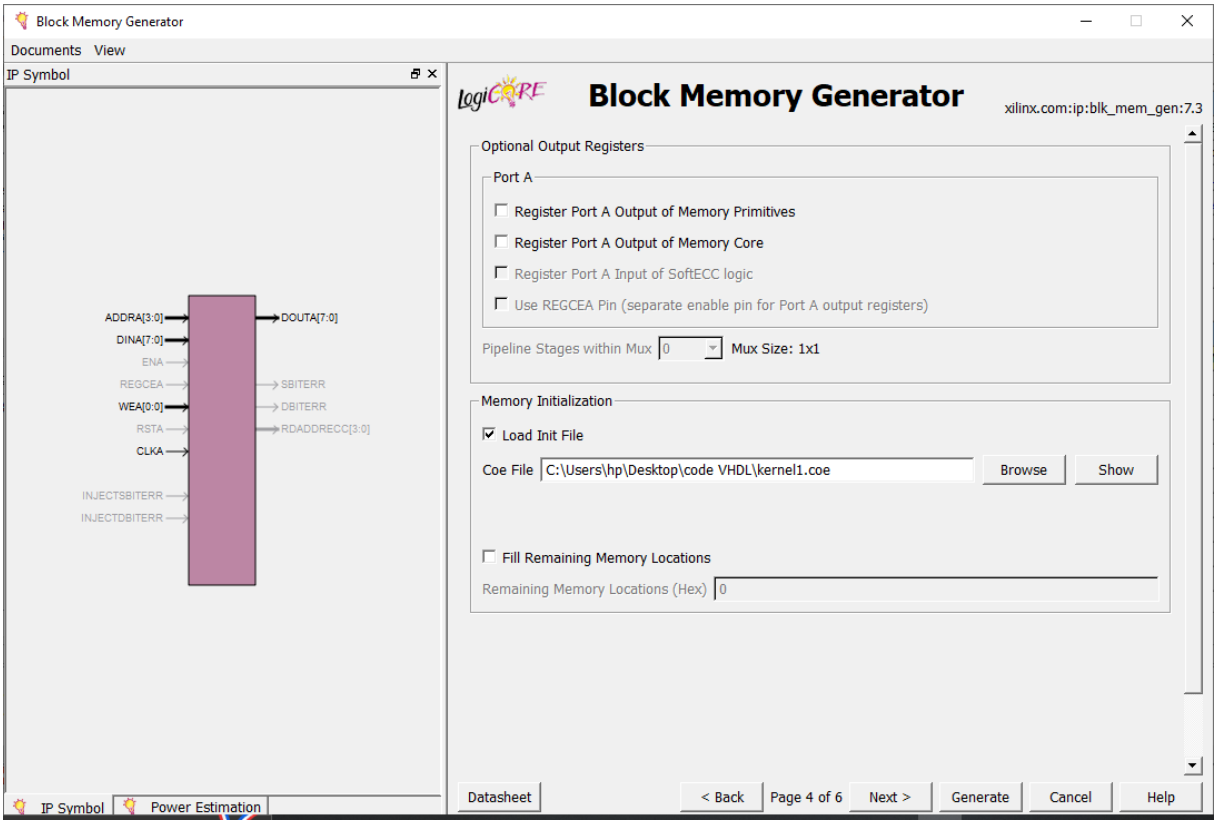


Figure.III. 14.Block memory Generator" kernel1.coe file"

After proceeding to the next step, click on the "Generate" button in the Block Memory Generator wizard. This action initiates the generation process of the IP core. Once completed, a message stating "IP core successfully created" will be displayed in the Xilinx ISE environment.

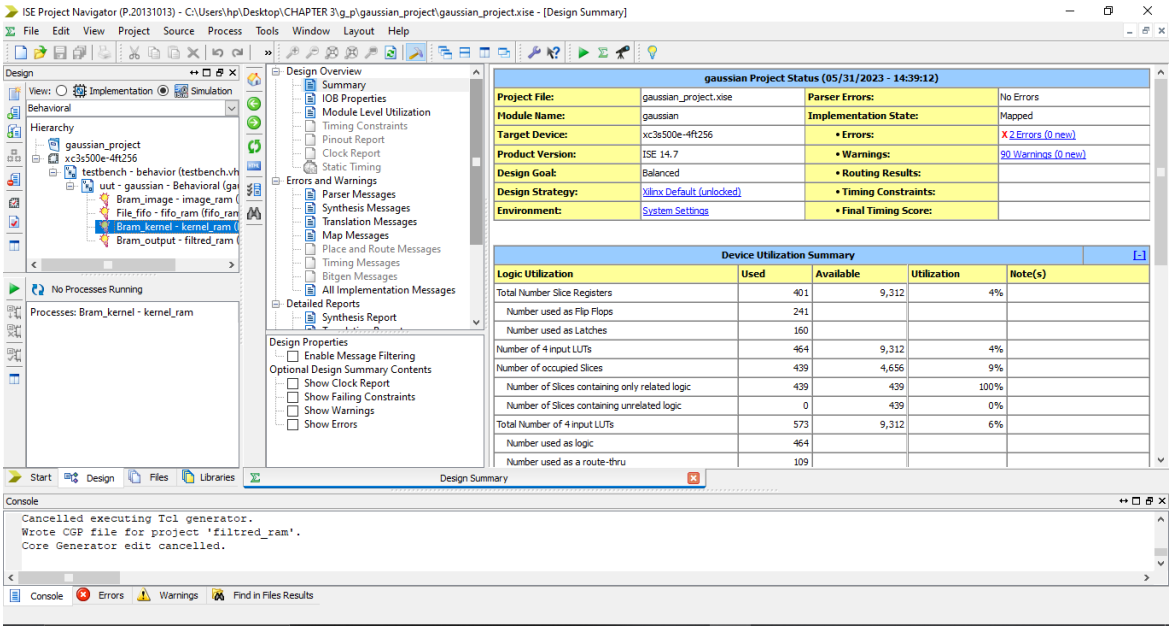


Figure.III. 15.Successful Creation of IP Core (bram_kernel)

III.4.4. Block memory generator FIFO

To store the Block memory generator FIFO, we follow the same steps as we did to store the image COE file with the following modifications**:**

The first step is chosen “IP(CORE Generator & Architecture Wizard) click on it and make name for the File name

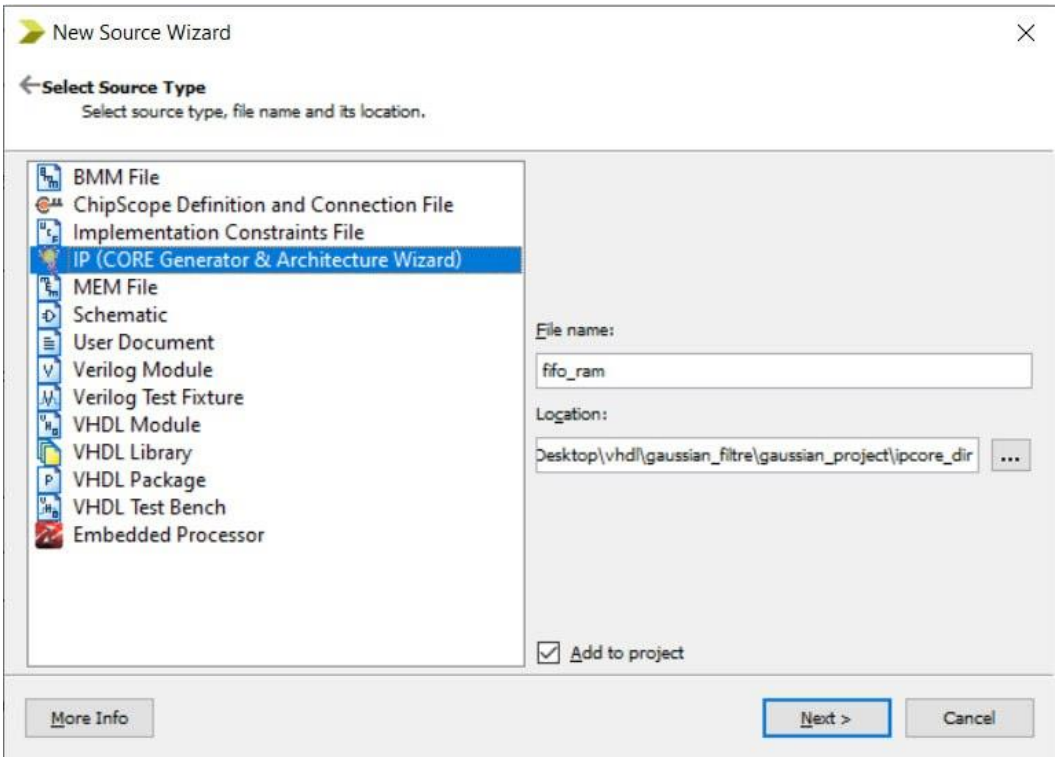


Figure.III. 16.Selecting " IP(Core Generator & Architecture)

The next step is Look for the specific option or button labeled as "FIFO generator" and click on it

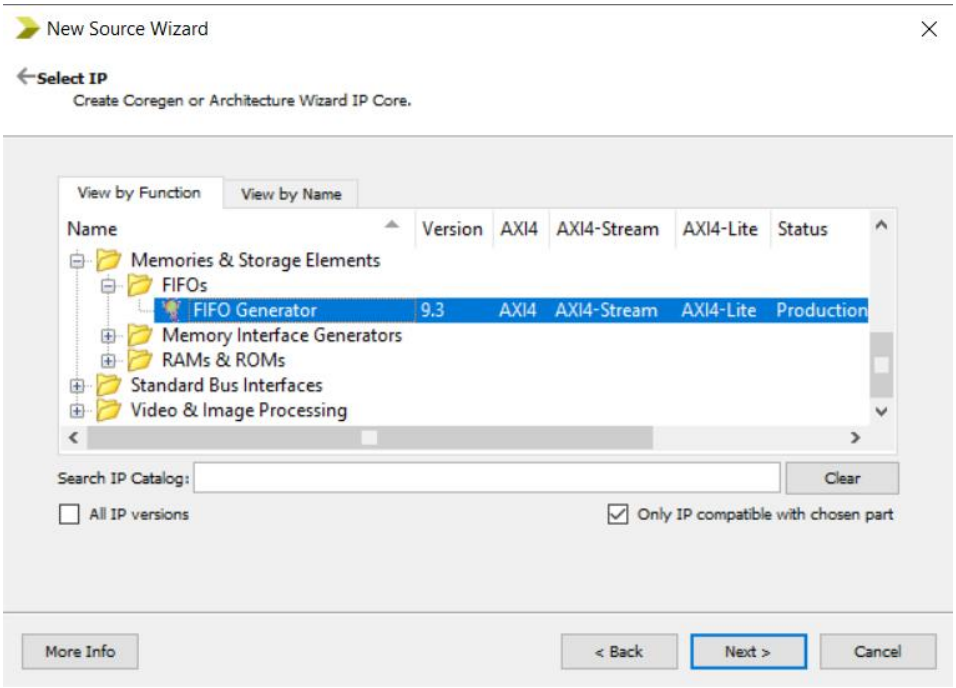


Figure.III. 17.Selection of " FIFO generator"

And now we Configure the settings of data port parametres size (widht & depht), the convolution result is 16 bits width and (92x92=8464) depth

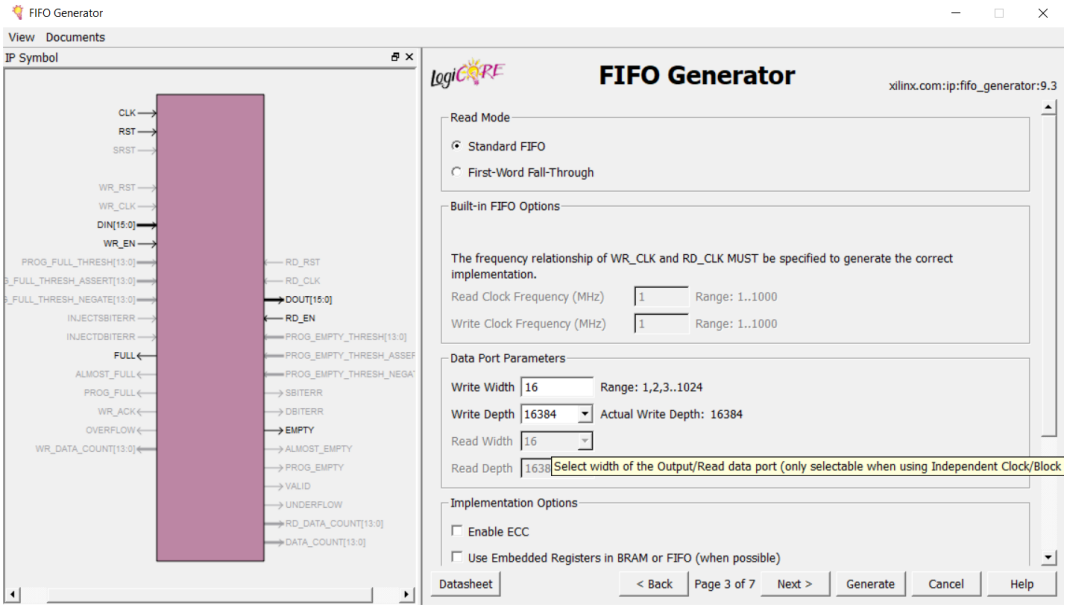


Figure.III. 18.Configure the settings of FIFO size (width and depth)

After proceeding to the next step, click on the "Generate" button in the Block Memory Generator wizard. This action initiates the generation process of the IP core. Once completed, a message stating "IP core successfully created" will be displayed in the Xilinx ISE environment.

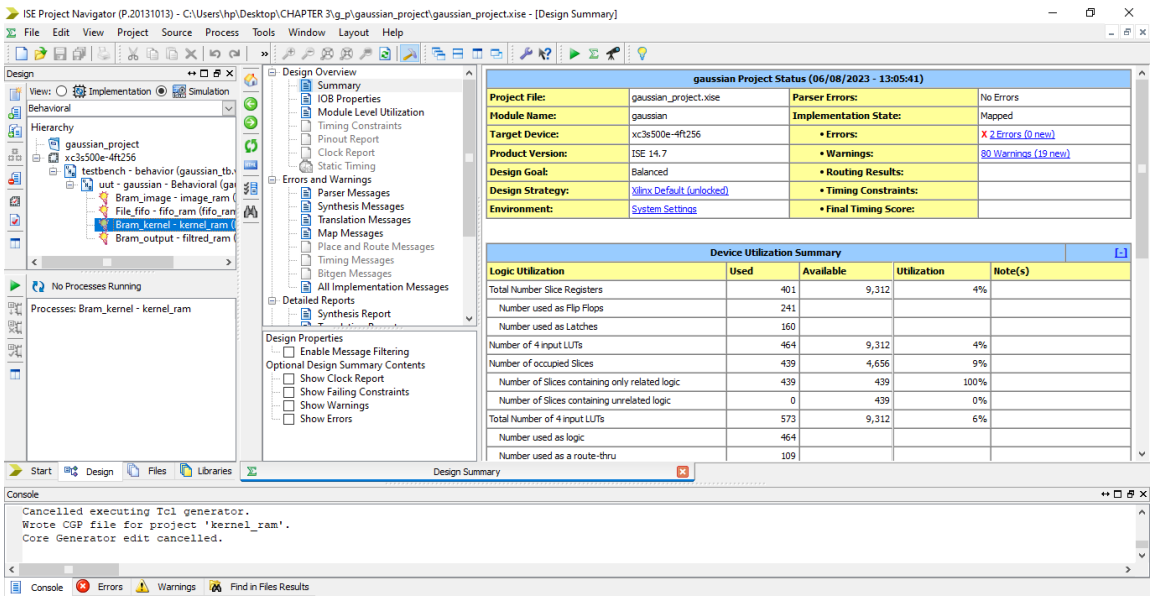


Figure.III. 19.Successful Creation of IP Core(FIFO RAM)

III.4.5. Block memory generator BRAM_OUT

To store the Block memory generator BRAM_OUT, we follow the same steps as we did to store the image COE file with the following modifications**:**

Upon selecting the "Block Memory Generator," a configuration window or interface will appear and we Configure the settings for the memory generator, such as the memory size, data width, and other relevant parameters

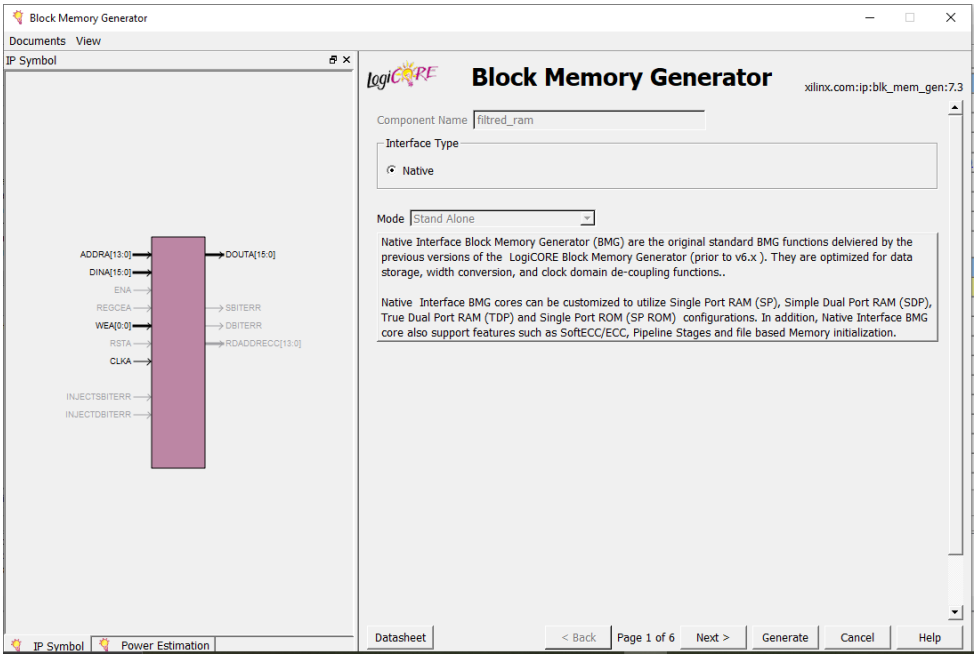


Figure.III. 20."Block memory Generator" IP core Wizard

At this moment, we proceed toConfigure the settings of Memory size (widht & depht), the filtrede image which is the convolution result is 16 bits width and (92x92=8464) depth

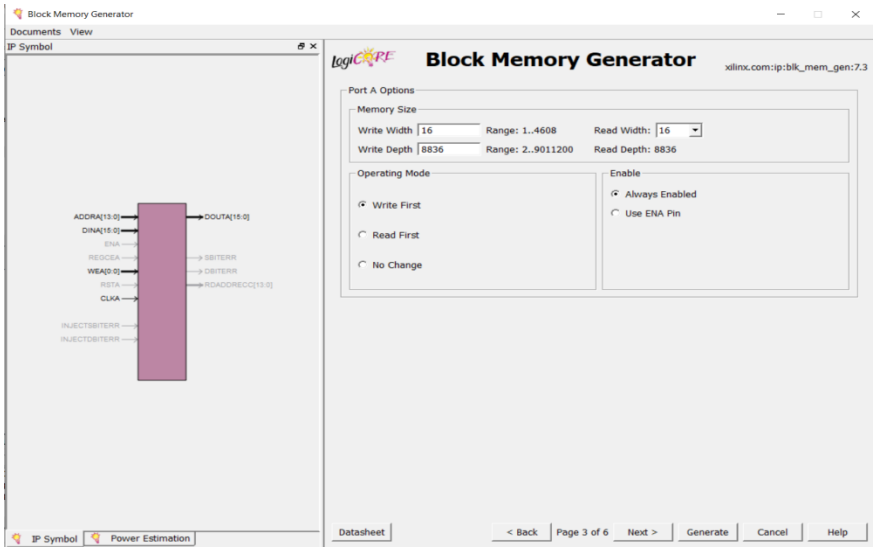


Figure.III. 21.Configure the settings of memory size (widht and depth)

After proceeding to the next step, click on the "Generate" button in the Block Memory Generator wizard. This action initiates the generation process of the IP core. Once completed, a message stating "IP core successfully created" will be displayed in the Xilinx ISE environment.

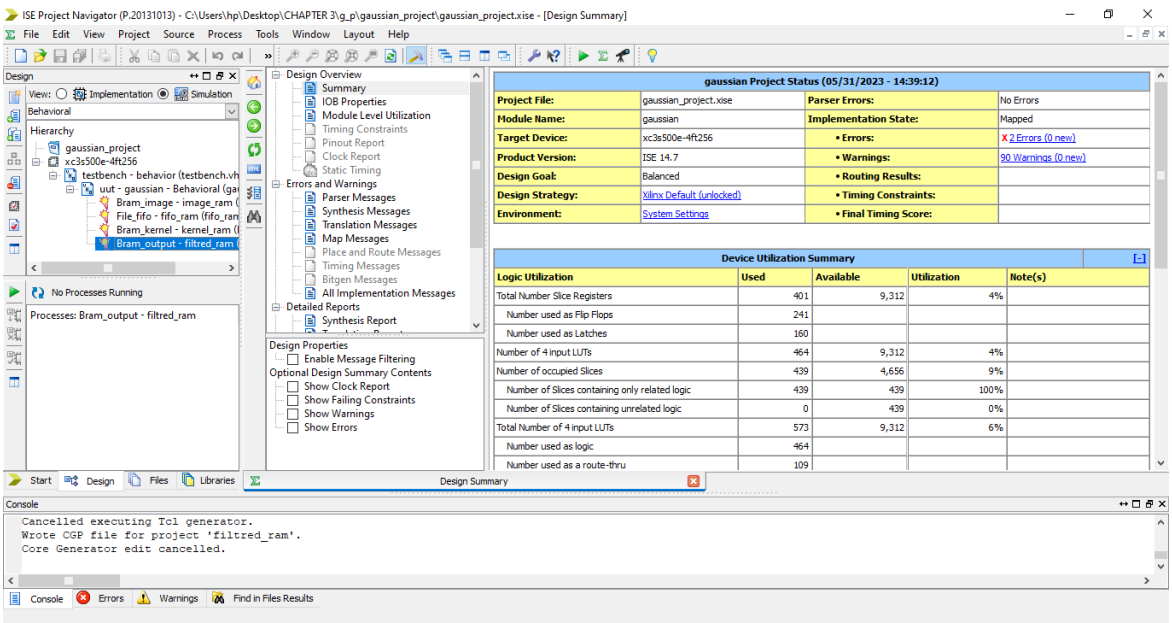


Figure.III. 22.Successful Creation of IP Core

III.4.6. Convolution module using finite state machine:

The process involves multiplying a 3x3 image by a 3x3 mask through signal multiplication. Input values for the multiplier are stored in the address port registers of the RAM blocks. The outputs of the RAM blocks (BRAM1 and BRAM2) serve as inputs to the multiplier. Each multiplication operation is completed within a single clock cycle, meaning the remaining 9 multiplications will require 9 clock cycles. The partial products from each multiplication are summed together to obtain the final result. The final results from the multipliers are also stored in RAM blocks (BRAM3).

The convolution module is designed as a Finite State Machine (FSM) simulated in VHDL. Figure III.23 illustrates the Finite State Machine (FSM) of the matrix convolution. The design incorporates five states: Set address, Read input, Computation, Store output, and Complete. Each state serves a specific purpose, and their names within the FSM convey their respective functions clearly.

The "Set_address" state sets the address of the memory elements for image pixels and kernel elements. One clock cycle is required to write the address to the address bus. When the address is set, the next state "Read_input" is called, which reads the data stored in the memory locations indicated by the address bus. The next address is then set in the set_address state, and the values of this memory location are read in the next clock cycle. The loop of set_address and read_input states continues and repeats until all elements required for convolution have been successfully read. After successfully reading the elements, the state transitions to the computation state. In this state, a flag called 'compute' is activated, triggering a combinational process block responsible for performing the convolution. In this process, the pixel values of the image are multiplied by the corresponding elements of the kernel. The resulting products are then summed together and stored. After the computation of the convoluted result, the state transitions to 'store_output'. In this state, the value present on the signal bus is stored or written to the memory location in the BRAM3 (Block RAM) indicated by the address bus of the output memory block. Furthermore, when the last element of the input matrix is reached, the state machine terminates, and the data stored in the output BRAM3 memory block is written to a text file.[14]

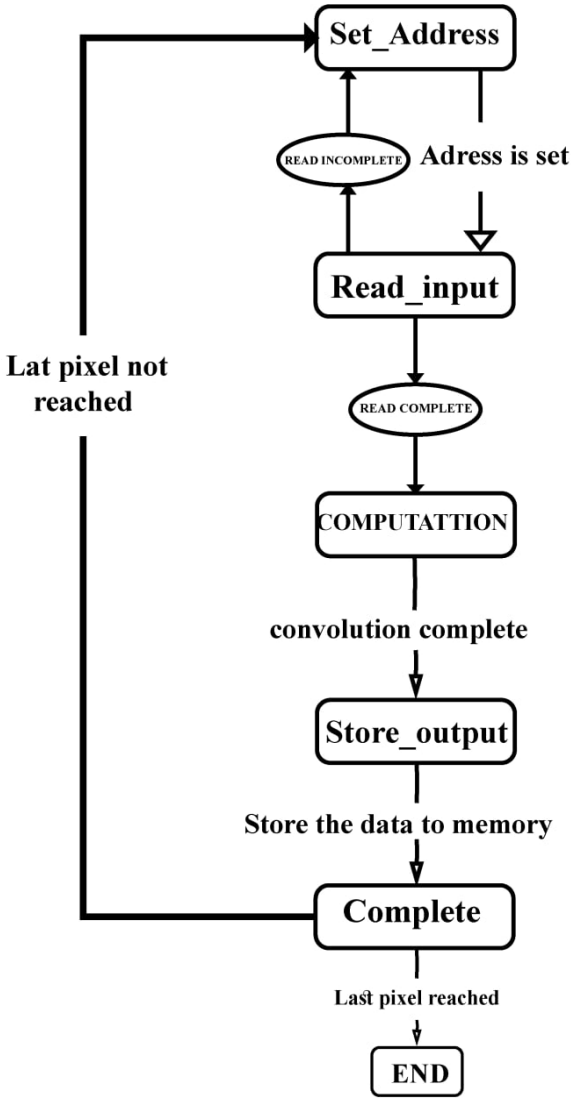


Figure.III. 23.Finite state machine of matrix convolution.[14]

III.4.7. Displaying the output image using MATLAB

The output matrix values have been saved in a text file containing a total of 8464 data values. To retrieve these values, MATLAB is utilized, which reads the file and stores the data as a 2-D matrix with dimensions of 92x92. The resulting matrix is then visualized as an image in MATLAB.

Original image Noisy image Filtered image

Figure.III. 24.Displaying the output image using MATLAB

III.5. Synthesis and simulation

For the purposes of synthesis and simulation in this study, two different platforms are used. Utilizing Xilinx Synthesis Technology (XST), a feature of the ISE software, the synthesis stage is carried out. The ISIM (I Simulator), a different Xilinx package, is used for simulation, on the other hand. The Xilinx software bundle comes with both XST and ISIM.

III.5.1. XST synthesis

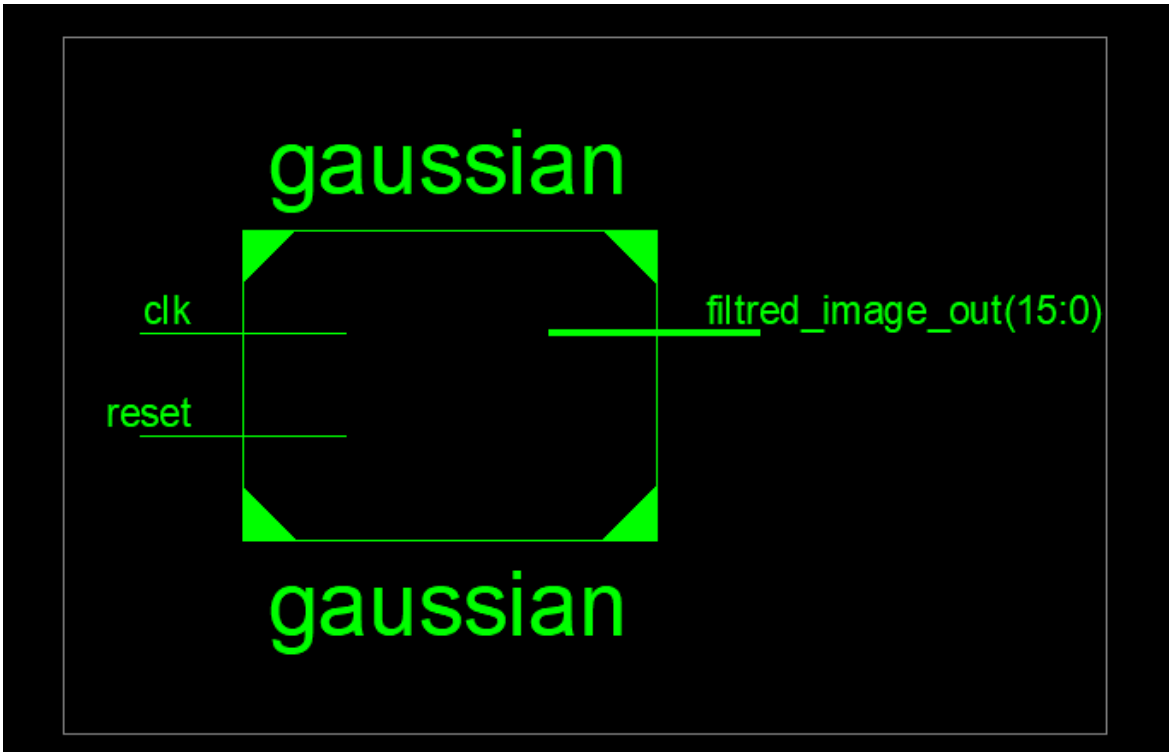


Figure.III. 25.Top level design

The top-level design depicted in the given figure illustrates the system clock, reset input, and the output port named "filtred_image_out'.

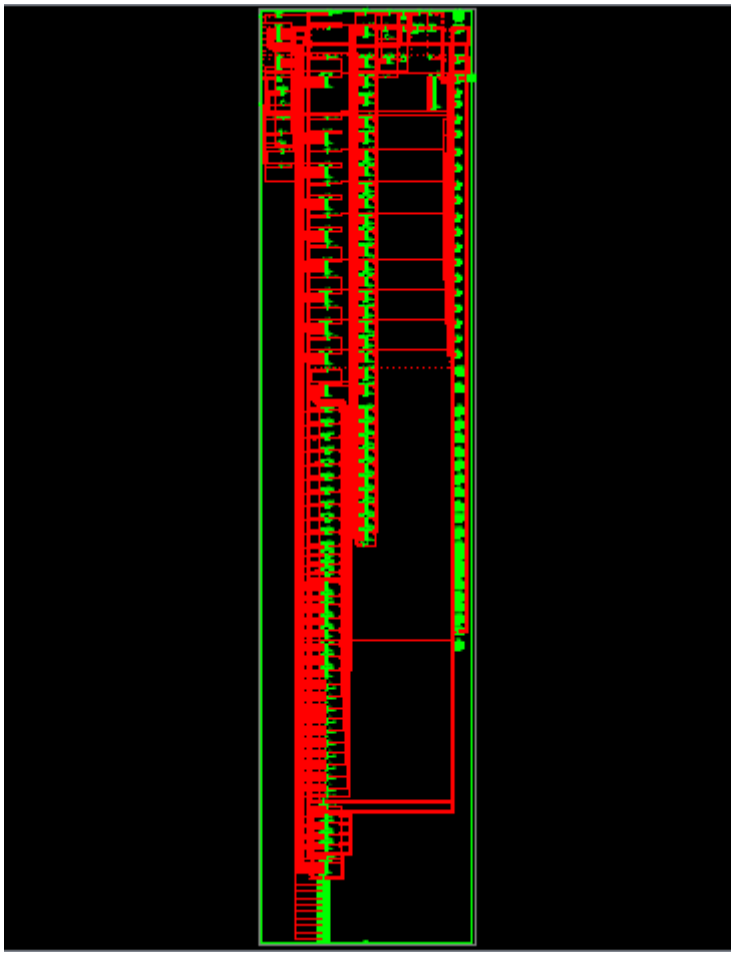


Figure.III. 26.Complete design schematic

The schematic provided above presents a comprehensive view of the design, showcasing all the necessary blocks. The key blocks are magnified and displayed in detail below.

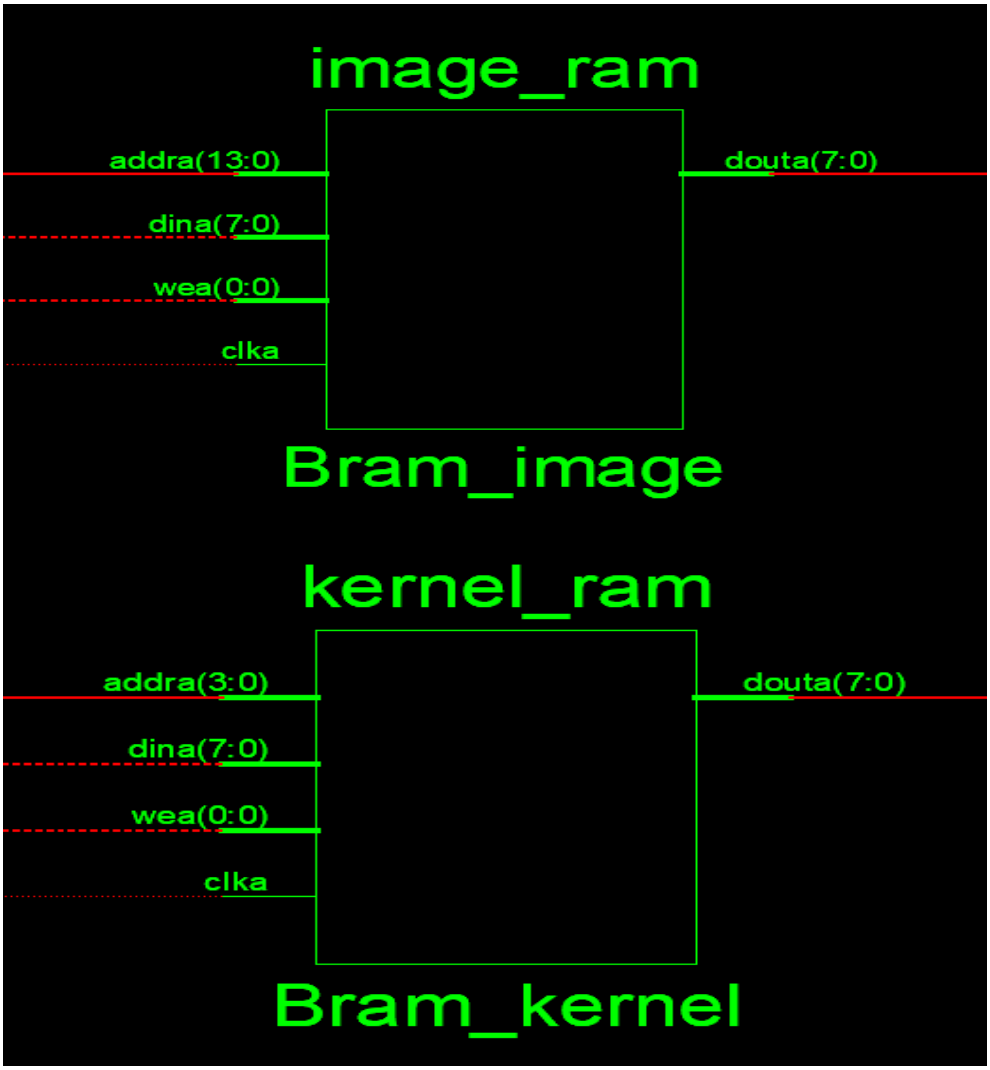


Figure.III. 27.Schematic showing BRAM

On the other hand, inputs such as the address bus and clock are shown as connected since their values are continuously supplied throughout the execution of the design. The range specified within parentheses indicates the size of the respective bus. For instance, "douta (7:0)" signifies that an 8-bit data can be transmitted via this data bus, addra (13,0) signifies that a 14-bit data for image and addra (3,0) signifies that a 4-bit data for kernel can be transmitted via the address bus.

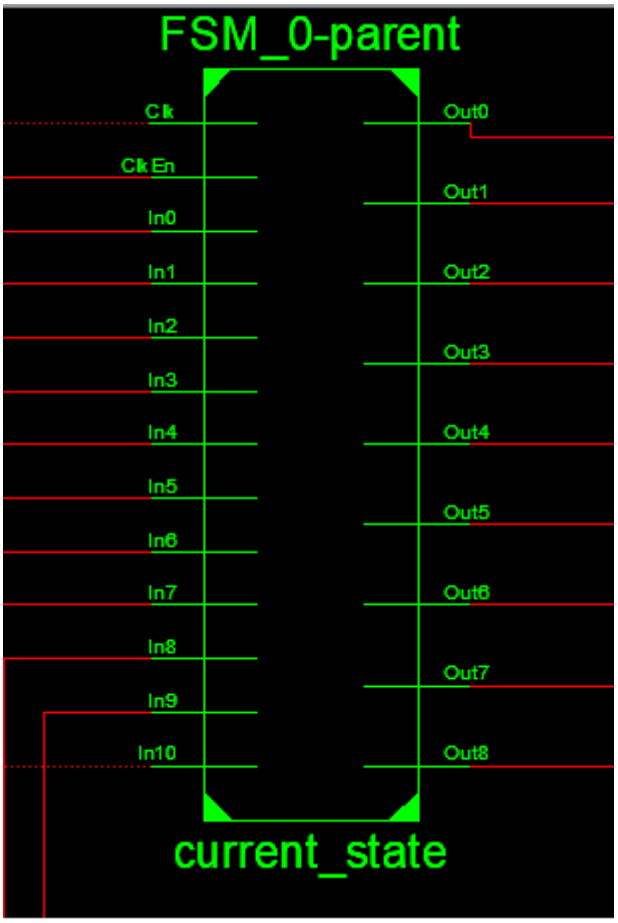


Figure.III. 28.Schematic showing FSM

The provided figure illustrates the Finite State Machine (FSM) employed in the design project. The design employs a total of 5 states within the state machine. To represent these 5 states, a combination of 3 bits is utilized. the current state is set to one of the following states: set_address, read_input, computation, store_output, and complete.

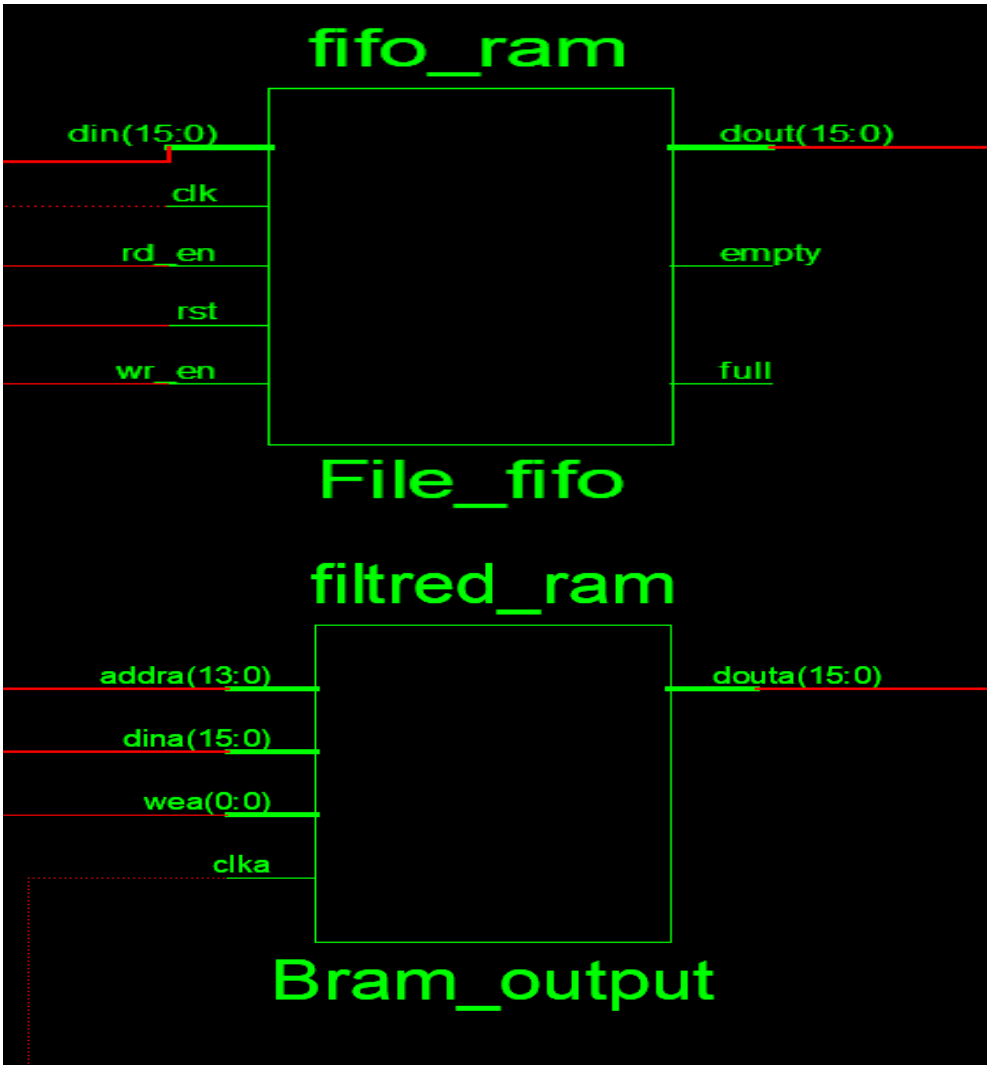


Figure.III. 29.Schematic showing the output storage phase

The output storage step is depicted in the offered schematic. The convoluted result is present on the output BRAM's input data bus When all of the convoluted results have been saved in BRAM, they are registered and written to the FIFO in succeeding clock cycles. After that, the output is taken from the FIFO and written to a text file.

III.5.2 ISim simulation

The simulation was conducted using ISim, which is an integrated HDL simulator package within ISE (Integrated Software Environment). ISim offers two operation modes: Graphical User Interface (GUI) and Command Line mode. The GUI mode was chosen due to its ability to visually display data through graphs and waveforms, facilitating analysis and debugging.[14]

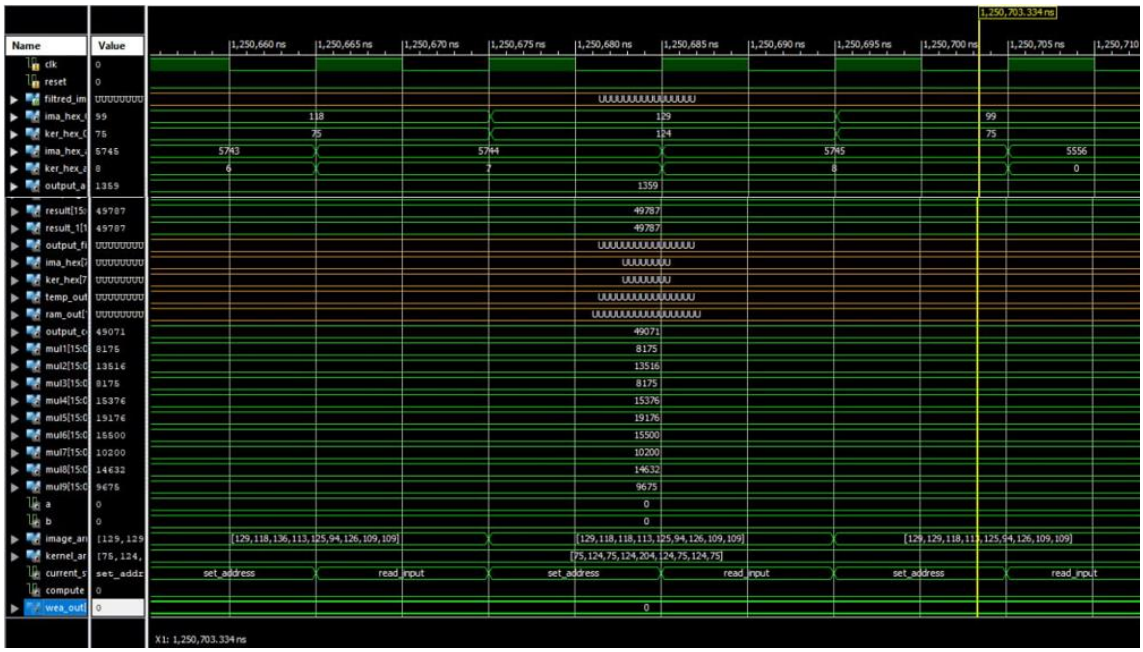


Figure.III. 30.Simulation showing set_address and read_input states

The process of setting the address and reading the input from the memory location is repeated until all of the elements required for convolution are read entirely. We can observe that it takes one clock cycle to establish the address and one clock cycle to read the data. As a result, it takes 18 clock cycles to read one set of inputs required for convolution. The values retrieved from memory are saved in a temporary array called Image_array and kernel_array, which are used in the combinational block to conduct convolution (Figure III.31).

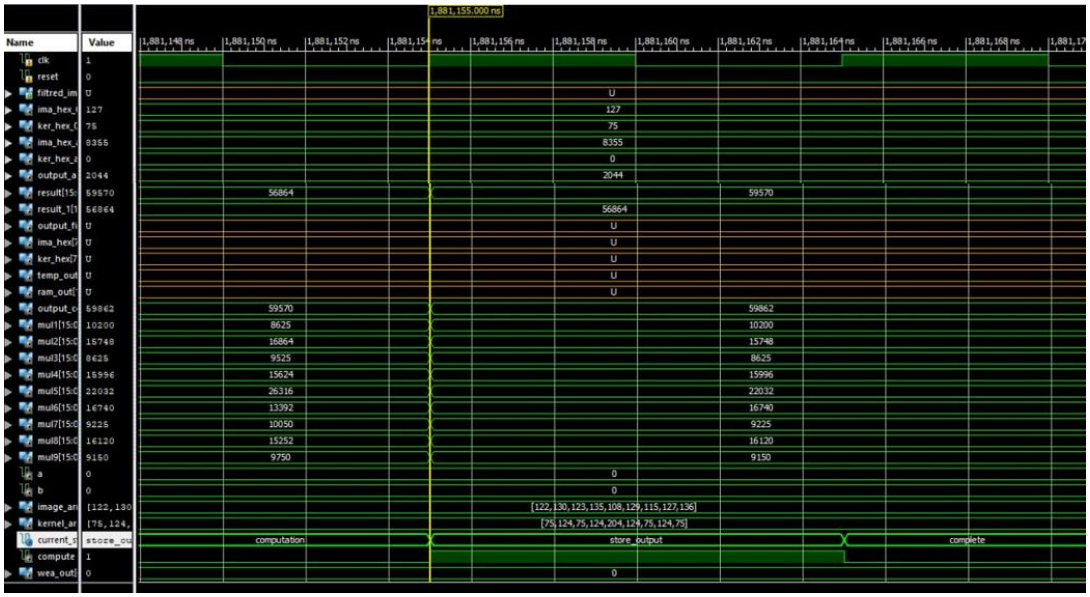


Figure.III. 31.Simulation showing computation, storing and complete state

The provided diagram illustrates the simulation details of the FSM (Finite State Machine) for the computation, store_output, and complete states. The computation is performed in a combinational process block, the result is computed within a clock cycle, and the data is available on the input bus of the BRAM that is used to store the output. The store_output state is reached in the following clock cycle, and the data is written to the memory location available on the address bus. Following this, the FSM enters the complete state, where it verifies whether the end of a row or the end of the entire image has been reached. Once the last pixel has been reached, the state machine is maintained in the full state until the end of the simulation.[14]

The resource utilization description for the results of the gaussian filtering is shown in figure III.32.

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Total Number Slice Registers	401	9,312	4%	
Number used as Flip Flops	241			
Number used as Latches	160			
Number of 4 input LUTs	464	9,312	4%	
Number of occupied Slices	439	4,656	9%	
Number of Slices containing only related logic	439	439	100%	
Number of Slices containing unrelated logic	0	439	0%	
Total Number of 4 input LUTs	573	9,312	6%	
Number used as logic	464			
Number used as a route-thru	109			
Number of bonded IOBs	18	190	9%	
Number of RAMB16s	30	20	150%	OVERMAPPED
Number of BUFGMUXs	2	24	8%	
Number of MULT18X18SIOs	9	20	45%	
Average Fanout of Non-Clock Nets	2.92			

Figure III. 32.FPGA resource utilization summary

III.6. Conclusion

In conclusion, this section offered a comprehensive overview of the hardware implementation of image Gaussian filtering on an FPGA with VHDL. It began with an introduction and proceeded to explore various FPGA memory blocks, such as single-port BRAM, dual-port block memory, and FIFO memory. The discussion also encompassed the Xilinx CORE Generator, which serves as a valuable tool for generating IP cores. Furthermore, the hardware implementation process of image Gaussian filtering was detailed, covering

essential elements like the top-level design flowchart, COE file generation, and block memory generators for storing image pixels and Gaussian kernel elements. The section also examined the utilization of block memory generators for FIFO, BRAM_OUT, and the convolution module, incorporating a finite state machine. Finally, the section concluded by outlining the steps involved in displaying the output image using MATLAB and conducting synthesis and simulation, which included utilizing ISim for simulation purposes.

CONCLUSION

The primary purpose of this thesis was to highlight the importance of gaussian filter for reducing gaussian noise, as well as to design, simulate and implement this filter on Spartan3e device (xc3s500e) FPGA device using Xilinx system generator and VHDL

First, from MATLAB, a widely-used software platform for image processing, it was noticed that the Gaussian filter is effective in reducing gaussian noise by averaging the pixel values in the neighborhood of each pixel, giving more weight to pixels closer to the center., it applies a convolution operation that takes into account the spatial relationships between pixels. The gaussian filter can be controlled by adjusting the kernel size and its standard deviation parameter. A lower kernel size and standard deviation, have limited noise reduction but preserve more details, in other hand, a higher standard deviation leads to more aggressive smoothing, effectively reducing more noise but potentially blurring fine details

Second, Xilinx system generator offers a friendly environment design for image filtering because filtering units are designed by blocks. This tool support software simulation, but the most important is that can synthesize in FPGAs hardware, it is an easy and efficient tool for implementing filtering algorithms into FPGA which is an efficient real-time filtering. A hardware in loop verification and hardware software co-simulation were performed with the gaussian image filter, the simulated and synthesized results shows that the design can work at an estimated frequency of 50 MHz by using Spartan 3e FPGA device.

Third, we focused on VHDL with an IP Core generator for designing and implementing customizable Gaussian filters on the FPGA. This approach highlighted the advantages of VHDL-based design and the flexibility provided by the IP Core generator. Gaussian image filtering was performed in VHDL using Block Memory Generator which is one of the IP core that is provided by Xilinx Core Generator that allows to store larger images, as well as the design of the convolution is in the heart of filtering with finite state machine (FSM).

In summary, this thesis provides valuable insights into different methodologies and tools for Gaussian image filtering. MATLAB offered a versatile software platform, XSG with the FPGA card demonstrated the potential for efficient real-time filtering, and VHDL with the IP Core generator provided customization capabilities. The choice of methodology depends on specific requirements and constraints, such as performance needs, customization options,

Researchers and practitioners can consider these approaches based on their specific application demands and resource availability. Ultimately, this thesis serves as a valuable source for researchers and practitioners interested in image filtering and its implementation using various methodologies.

References

[1] T. Issam, R. Salah, and M. Brahim, "Filtering Techniques To Reduce Speckle Noise And Image Quality Enhancement Methods On Porous Silicon Images Layers," Majlesi Journal of Electrical Engineering, 2022.

[10] S. Mittal, S. Gupta, and S. Dasgupta, "System generator: The state-of-art FPGA design tool for dsp applications," in Third International Innovative Conference on Embedded Systems, Mobile Communication and Computing (ICEMC2 2008), pp. 187-190, Aug. 2008.

[11] "System Generator for DSP Reference Guide UG638 (v14.5)," March. 20, 2013.

[12] A. Sghaier, A. Douik, and M. Machhout, "FPGA implementation of filtered image using 2D Gaussian filter," International Journal of Advanced Computer Science and Applications, vol. 7, no. 7, 2016.

[13] B. Muralikrishna, K. G. Deepika, B. R. Kanth, and V. S. Vemana, "Image processing using IP core generator through FPGA," International Journal of Computer Applications, vol. 46, no. 23, pp. 48-52, 2012.

[14] S. Eswar, "Noise reduction and image smoothing using gaussian blur," Doctoral dissertation, California State University, Northridge, 2015.

[2] R. C. Gonzalez and R. E. Woods, "Digital image processing 4th edition, global edition," 2018.

[3] Z. Afrose, "A comparative study on noise removal of compound images using different types of filters," International Journal of Computer Applications, vol. 47, no. 14, 2012.

[4] E. Tisserand, J. Pautex, and P. Schweitzer, "Analyse et traitement des signaux Méthodes et applications au son et à l'image, Algeria-Educ. com. DUNOD edition, Paris, pp287-288," 2008.

[5] S. Kim and R. Casper, "Applications of convolution in image processing with MATLAB," University of Washington, pp. 1-20, 2013.

[6] M. Goyal, Y. Lather, and V. Lather, "Analytical relation & comparison of PSNR and SSIM on baboon image and human eye perception using MATLAB," International Journal of Advanced Research in Engineering and Applied Sciences, vol. 4, no. 5, pp. 108-119, 2015.

[7] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," IEEE Transactions on Image Processing, vol. 13, no. 4, pp. 600-612, 2004.

[8] I. Bakurov, M. Buzzelli, R. Schettini, M. Castelli, and L. Vanneschi, "Structural similarity index (SSIM) revisited: A data-driven approach," Expert Systems with Applications, vol. 189, p. 116087, 2022.

[9] A. K. Tyagi, "Matlab and Simulink for Engineers by Agam Kumar Tyagi," Oxford University Press, 2012.