PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

MINISTERY OF HIGHER EDUCATION AND SCIENTIFIC RESARCH

University Mohamed El-Bachir El-Ibrahimi - Bordj Bou Arreridj

Faculty of Sciences &technology

Department of Electronics

Thesis

Presented to get

THE MASTER'S DIPLOMA

BRANCH: **ELECTRONIQUE**

Specialty: EMBEDDED ELECTRONIC SYSTEMS

By

BENMAHDI Islam

BENSAID Ayoub

Entitled

Hardware Software Co-Simulation For image filtering and blurring Using Xilinx system generator

Thesis assessed in: 12/09/2021

By the evaluation committee:

DIFFELLAH Nacira	*MCB*	*Chairman*	*Univ-BBA*
HAMADACHE Fouzia	*MAA*	*Supervisor*	*Univ-BBA*
LATOUI Abdelhakim	*MCA*	*Examiner*	*Univ-BBA*

Academic year 2020/2021

* In accordance with:

- Decree n ° 055 of January 21, 2021 Fixing exceptional authorized provisions in terms of organization and pedagogical management, evaluation and progression of students, during the COVID-19 period for the academic year 2020-2021;
- Minutes of the meeting of the Science and Technology team for the month of May 2021

Abstract

Image filtering is one of the very useful techniques in image processing and computer vision. It is used to eliminate useless details and noise from an image. Blur is a widely used effect in graphics software, the visual effect of this blurring technique is a smooth blur resembling that of viewing the image through a translucent screen.

In this thesis, a hardware-Software Co-Simulation for image filtering and blurring will be present. The filtering and blurring architectures are based on convolution module which require a significant amount of computing resources and this can be implemented using Xilinx System Generator (XSG). The Simulink simulation and hardware-in-the loop approach presents a far more cost efficient solution. The design was implemented targeting a Spartan3e device (xc3s500e). Obtained results from software and hardware are discussed and compared using the Peak Signal-to-Noise Ratio (PSNR), also the FPGA resource usage for different kernel sizes.

Résumé

Le filtrage d'images est l'une des techniques très utiles en traitement d'images et en vision par ordinateur. Il est utilisé pour éliminer les détails inutiles et le bruit d'une image. Le flou est un effet largement utilisé dans les logiciels graphiques, l'effet visuel de cette technique de flou est un flou lisse ressemblant à celui de la visualisation de l'image à travers un écran translucide.
Dans cette thèse, une Co-simulation matériel-logiciel pour le filtrage et le flou d'images sera présenté. Les architectures de filtrage et de flou sont basées sur un module de convolution qui nécessite une quantité importante de ressources et cela peut être implémenté à l'aide de Xilinx System Generator (XSG). La conception a été mise en œuvre en ciblant FPGA Spartan3e (xc3s500e). Les résultats obtenus sont discutés et comparés à l'aide du rapport signal sur bruit (PSNR), ainsi que de l'utilisation des ressources FPGA pour différentes tailles de noyau de convolution.

الملخص

تصفية الصور هي إحدى التقنيات المفيدة جدًا في معالجة الصور والرؤية الحاسوبية. يتم استخدامه لإزالة التفاصيل غير المفيدة والضوضاء من الصورة. الطمس هو تأثير مستخدَم على نطاق واسع في برامج الرسومات، والتأثير المرئي لتقنية التمويه هذه عبارة عن تمويه سلس يشبه عرض الصورة من خلال شاشة نصف شفافة.
في هذه الأطروحة، سيكون هناك محاكاة مشتركة للأجهزة والبرامج لتصفية الصور والتعتيم. تعتمد هياكل التصفية والتعتيم
على وحدة الالتفاف التي تتطلب قدرًا كبيرًا من موارد الحوسبة ويمكن تنفيذ ذلك باستخدام Xilinx System Generator.
تقدم محاكات Simulinkونهج الأجهزة في الحلقة حلاً أكثر فعالية من حيث التكلفة. تم تنفيذ التصميم باستعمال جهاز Spartan3e (xc3s500e). تتم مناقشة النتائج التي تم الحصول عليها من البرامج والأجهزة ومقارنتها باستخدام PSNR

Acknowledgements

First and foremost, praises and thanks to the God, the Almighty, for his showers of blessings throughout our research work to complete the thesis successfully.

We would like to express ourdeep and sincere gratitude to our thesis supervisor, F.HAMADACHE forgiving us the opportunity to do research and providing in valuable guidance throughout this research work. Her dynamism, vision, sincerity and motivation have deeply inspired us. She has taught us the methodology to carry out the research and to present the research work as clearly as possible. It was a great privilege and honor to work and study under her guidance. We have extremely grateful for what she has offered us. We would also like to thank her for her friendship,empathy,andgreatsenseofhumor.

We are also grateful to the board of examiners for their willingness to evaluate our work and to provide helpful comments and remarks.

We are extremely grateful to our parents for their love, prayers, caring and sacrifices for educating and preparing us for the future. Also we express our thanks to our brother, sister for their support and valuable prayers.

Finally, our thanks go to all the people who have supported us to complete the research work directly orindirectly.

Contents

List of figures

List of tables

Lists of abbreviations

PDF: *Probability Distribution Function*

MSE: *Mean Square Error*

PSNR: *Peak Signal to Noise Ratio*

FPGA: *Field Programmable Gate Array*

PLCC:*Plastic Leaded Chip Carrier*

CLBs: *Configurable Logic Blocks*

IOBs: *Input/Output Blocks*

LUTs:*Look-Up Tables*

DDR:*Double Data-Rate*

DCM: *Digital Clock Manager*

ROM: *Read Only Memory*

SRAM:*Static Random-Access Memory*

PROM:*Programmable Read-Only Memory*

SPI: *Serial Peripheral Interface*

JTAG: *Joint Test Action Group*

NI:*National Instruments*

USB:*Universal Serial Bus*

HDL:*Hardware Description Language*

VHSIC:*Very High Speed Integrated Circuits*

ASICs:*Application Specific Integrated Circuits*

MATLAB:*Matrix Laboratory*

RTL:*Register Transfer Level*

XNF:*Xilinx Netlist File*

XSG: *Xilinx System Generator*

IDE:*Integrated Design Environment*

UCF: *User ConstraintFile*

DSP:*Digital Signal Processing*

Introduction

Visual information transmitted in the form of digital images is becoming a major method of communication in the modern age, but the image obtained after transmission is often corrupted with noise. The received image needs processing before it can be used in applications. Image filtering involves the manipulation of the image data to produce a visually high quality image (Saxena & Kourav, 2014).Blur filters are designed primarily for retouching images to soften, haze, cloud, fuzz, or distort specific areas of a picture or the entire imageWidely distributed software packages such as Photoshop provide a set of ‘‘filtering’’ operations which enable the user to improve the image in some way: from image smoothing that removes noise and high frequencies, sharpening that increases high frequency content, contrast stretching, through to specialized algorithms (Behrenbruch et al.,2004).

Recently, Field Programmable Gate Array (FPGA) technology has become a viable target for the implementation of algorithms suited to image processing applications. The use of rapid prototyping tools such as MATLAB-Simulink and Xilinx System Generator becomes increasingly important because of time-to-market constraints. The Simulink simulation and hardware-in-the loop approach greatly facilitates the design process as a real-time embedded system (Saidani et al., 2009).

This thesis presents the implementing of image filtering and blurring on a reconfigurable logic platform using Xilinx System Generator (XSG). The design will be implemented targeting a Spartan3e device (xc3s500e). Hardware in the loop verification and co-simulation may also be performed. The thesis is organized as follows:

Chapter 1: provides details about image filtering and blurring algorithms. Simulation with MATLAB will be done then we compare the image quality from obtained results using the Peak Signal-to-Noise Ratio (PSNR)

Chapter 2: presents a description of Field Programmable Gate Array (FPGA),its Programming Tools and Languages and the design methodology for implementation.

Chapter 3: .provides the implementation of image filtering (Gaussian filter) and image blurring (Gaussian, motion and box) in Simulink and FPGA using FPGA-in-the-loop approach (Co-simulation), a brief description of how to acquire results. Furthermore, results and discussion of the acquired results will be discussed using peak signal to noise ratio.

Finally, a conclusion concludes our achievements, in addition to possible future work.

Chapter 1

Spatial image filtering and blurring

Chapter 1: Spatial image filtering and blurring

1.1 Introduction

When an image is acquired by a camera or other imaging system, often the vision system for which it is intended is unable to use it directly. The image may be corrupted by noise which degrades the quality of image. Image filtering is the process of remove noise from image and improve the visual quality of the image (Chandel & Gupta, 2013).

Blurring is reduced sharpness of edges and spatial details. Blurring is also a great way to make images appear a little softer and make your subject stand out from the rest of the image. In other words, you might want to create some depth of field in the image. The visual effect of the blurring technique is a smooth blur resembling that of viewing the image through a translucent screen. Gaussian smoothing is also used as a pre-processing stage in computer vision algorithms in order to enhance image structures (Patel & Dangarwala, 2014).

This chapter provides details about Gaussian noise, Gaussian kernel and Gaussian blur and also provides Matlab results and discussion of the acquired results with respect to peak signal to noise ratio).

1.2 Image degradation

Image degradation is said to occur when a certain image under goes loss of stored information either due to digitization or conversion decreasing visual quality. The detailed mechanisms by which images are degraded are, of course, dependent on the imaging modality. It is important to note that some forms of degradation are reversible while others are not. (Md.Naseem Ashraf, 2013)

An additive noise follows the rule:

$$g(x,y) = f(x,y) + n(x,y) \tag{1.1}$$

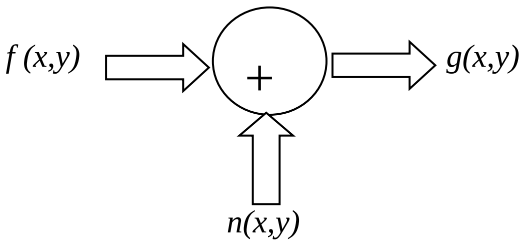


Figure 1.1: Image degradation by additive noise

Where $f(x, y)$is the original image, $n(x, y)$ denotes the noise introduced into the image to produce the corrupted image $g(x, y)$, and (x, y)represents the pixel location.

1.2.1 Noise

Noise is always presents in digital images during image acquisition, coding, transmission, and processing steps. It is very difficult to remove noise from the digital images without the prior knowledge of filtering techniques.

Image noise is random variation of brightness or color information in the images captured. It is degradation in image signal caused by external sources. Different factors may be responsible for introduction of noise in the image. The number of pixels corrupted in the image will decide the quantification of the noise. The principal sources of noise in the digital image are (Verma & Ali, 2013):

a) The imaging sensor may be affected by environmental conditions during image acquisition.

b) Insufficient light levels and sensor temperature may introduce the noise in the image.

c) Interference in the transmission channel may also corrupt the image.

d) If dust particles are present on the scanner screen, they can also introduce noise in the image.

1.2.2 Gaussian Noise

Gaussian noise is also called as electronic noise because it arises in amplifiers or detectors. Source: thermal vibration of atoms and discrete nature of radiation of warm objects. (Gonzalez et al., 2002). As the name indicates, this type of noise has a Gaussian distribution, which has a bell shaped probability distribution function PDF given by:

$$p(z) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(z-\bar{z})^2}{2\sigma^2}} \qquad (1.2)$$

Where:z represents the gray scale,$\bar{z}$ is the mean (average) value of z, and σ is its standard deviation. The standard deviation squared $\sigma^2$is called the variance.Generally Gaussian noise mathematical model represents the correct approximation of real world scenarios.

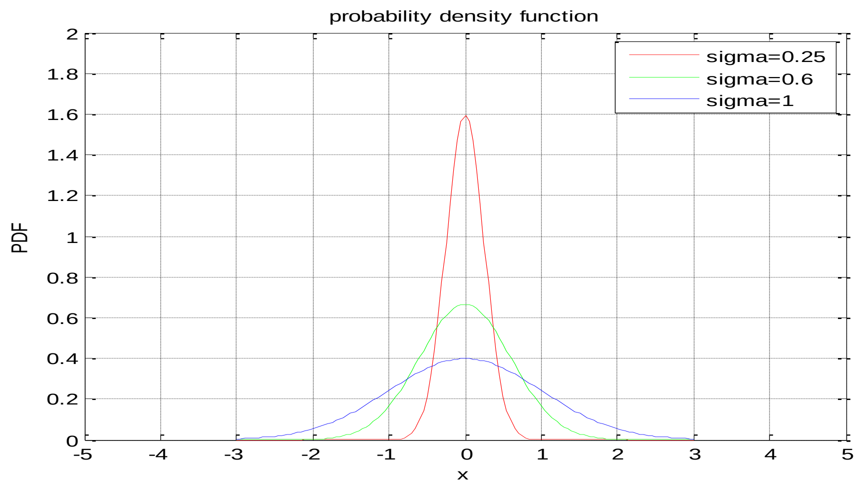


Figure 1.2: Gaussian PDFs with different standard deviation

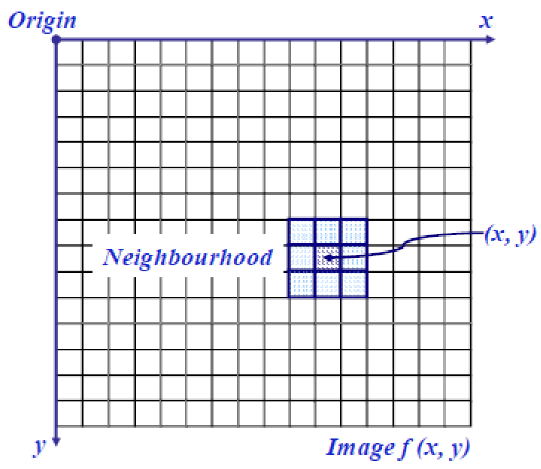
The distributions above show how the normal distribution changes as the standard deviation changes. The average is 0 and there are three different distributions with standard deviations of 0.25, 0.6, and 1. Note that the larger the standard deviation, the wider the distribution. , the variance can be thought of as controlling the width of the Gaussian PDF.

This noise model is additive in nature and follow Gaussian distribution. Meaning that each pixel in the noisy image is the sum of the true pixel value and a random, Gaussian distributed noise value (Verma & Ali, 2013).

1.3 Filtering in spatial domain

1.3.1 Neighbourhood Operations

A neighbourhood operation generates an 'output' pixel on the basis of the pixel at the corresponding position in the input image and on the basis of its neighbouring pixels. The size of the neighbourhood may vary: several techniques use (3x3, 5x5…) neighbourhoods centered at the input pixel, but many of the more advanced and useful techniques now use neighbourhoods which may be as large as 63 x63 pixels. The neighbourhood operations are often referred to as 'filtering operations'. This is particularly true if they involve the convolution of an image with a filter kernel or mask. Since convolution is such an important part of digital image processing, we will discuss it in detail before proceeding. (Vernon, 1991)



f(-1,-1)	*f*(-1,0)	*f*(-1,+1)
f(0,-1)	*f*(0.0)	*f*(0,+1)
f(+1,-1)	*f*(+1,0)	*f*(+1,+1)

Figure 1.3: Neighbourhood 3x3

1.3.2 Convolution

The convolution operation is much used in digital image processing and, though it can appear very inaccessible when presented in a formal manner for real continuous functions (signals and images), it is quite a simple operation when considered in a discrete domain. Before discussing discrete convolution, let us the two-dimensional convolution integral, which is given by the equation:

$$\tilde{f}(x,y) = (f * h)(x,y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x-i, y-j) h(i,j) \mathrm{d}i\mathrm{d}j \qquad (1.3)$$

The function h is normally referred to as the filter, since it dictates what elements of the input image f are allowed to pass through to the output image$\tilde{f}$. By choosing an appropriate filter, we can enhance certain aspects of the output and attenuate others. A particular filter h is often referred to as a 'filter kernel'. Attempts to form a conceptual model of this convolution operation in one's mind are often fruitless. However, it becomes a little easier if we transfer now to the discrete domain of digital images and replace the integral with the sigma (summation) operator. Now the convolution operation is given by:

$$\tilde{f}(x,y) = (f * h)(x,y) = \sum_i \sum_j f(x-i, y-j) h(i,j) \qquad (1.4)$$

The summation is taken only over the area where f and h overlap. This multiplication and summation is illustrated graphically in *Figure 1.4*. Here, a filter kernel h is a 3 x 3 pixel image, with the origin h (0, 0) at the center, representing a mask of nine distinct weights:

h (-1, -1)...*h* (+ 1, + 1). The kernel or mask is superimposed on the input image, the input image pixel values are multiplied by the corresponding weight, the nine results are summed, and the final value of the summation is the value of the output image pixel at a position corresponding to the position of the center element of the kernel. (Vernon, 1991)

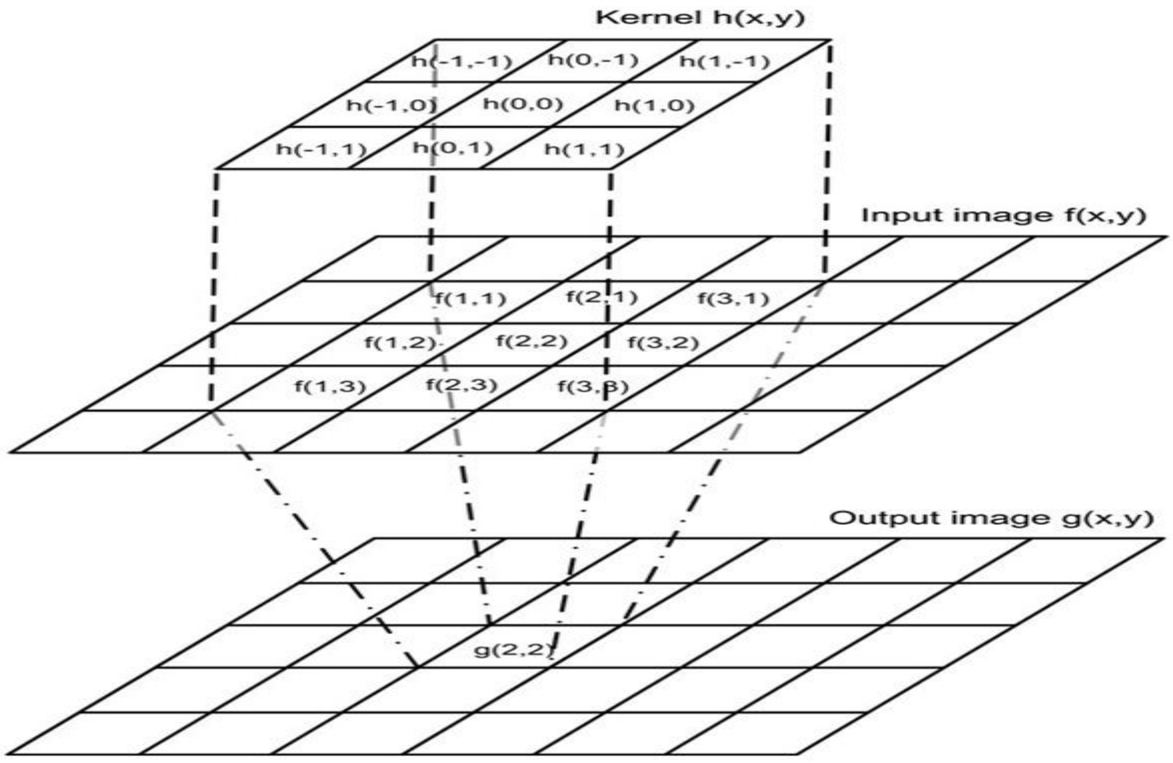


Figure 1.4: The principle of Convolution, illustrated with a 3×3 *kernel*

1.3.3 Gaussian mask

A Gaussian filter is a non-linear filter with impulse response equal to Gaussian Function. Each pixel in the given image is transformed using Gaussian function. It is often reasonable to truncate the filter window and implement the filter directly for narrow windows using a simple rectangular window function (Eswar, 2015) The Gaussian function of two variables has the basic form:

$$h(x,y) = e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{1.5}$$

Where x the distance from the origin is in the horizontal axis, y is the distance from the origin in the vertical axis, and σ is the standard deviation of the Gaussian distribution

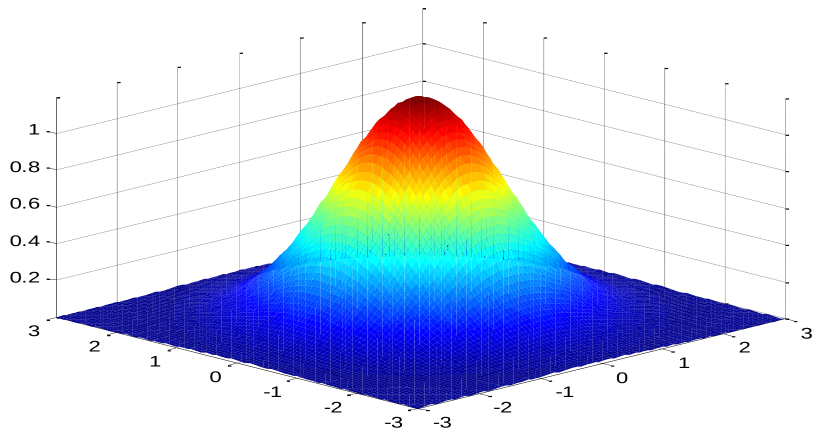


Figure 1.5: Gaussian distribution in 3D

This Gaussian kernel when needed to be convolved with a 2 dimension matrix has to be converted to an equivalent matrix structure and this matrix of the Gaussian discrete function is known as the Gaussian kernel matrix. Filtering can be implemented by convolving the input matrix with the Gaussian kernel matrix.

$$\begin{bmatrix} 0.0113 & 0.0838 & 0.0113 \\ 0.0838 & 0.6193 & 0.0838 \\ 0.0113 & 0.0838 & 0.0113 \end{bmatrix} \qquad \begin{bmatrix} 0.0000 & 0.0000 & 0.0002 & 0.0000 & 0.0000 \\ 0.0000 & 0.0113 & 0.0837 & 0.0113 & 0.0000 \\ 0.0002 & 0.0837 & 0.6187 & 0.0837 & 0.0002 \\ 0.0000 & 0.0113 & 0.0837 & 0.0113 & 0.0000 \\ 0.0000 & 0.0000 & 0.0002 & 0.0000 & 0.0000 \end{bmatrix}$$

Figure 1.6: Gaussian kernels 3×3, 5×5 *with* $\sigma = 0.5$

1.4 Image filtering and blurring

1.4.1 Image filtering

Image filtering is applied as pre-processing to eliminate useless details and noise from an image.

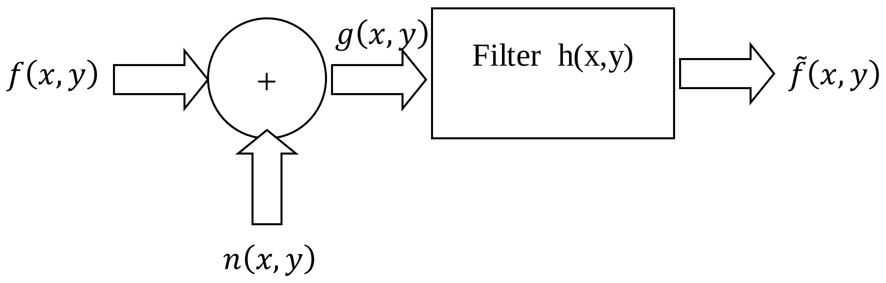


Figure 1.7: Image filtering

Image filtering is produced by convolution between a degraded image and 2D Gaussian mask which is usually odd and symmetrical $2n+1\times 2n+1$, so the convolution become:

$$f(x,y)=(g*h)(x,y)=\sum_{i=-n}^{n}\sum_{j=-n}^{n} g(x+i,y+j)h(i,j) \tag{1.6}$$

1.4.2 Image blurring

Blurring is reduced sharpness of edges and spatial details. It can be modeled as a shifting towards lower frequencies component on the frequency domain. Blurring is also a great way to make images appear a little softer and make your subject stand out from the rest of the image. In other words, you might want to create some depth of field in the image.

Widely distributed software packages such as Photoshop provide a set of ‘‘blurring’’ operations which enable the user to improve the image in some way: from image smoothing (typically local averaging) that removes noise and high frequencies, sharpening that increases high frequency content, contrast stretching, through to specialized algorithms, for example for red-eye reduction. Such image filtering is designed to improve the appearance of an image, relying on the human visual system to disregard any unwanted change of content of the image (Behrenbruch et al., 2004).

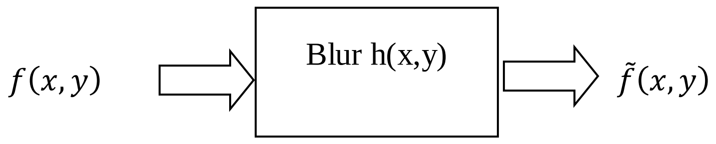


Figure 1.8: Image Blurring

1.4.2.1 Gaussian and box blur

Gaussian blur is the result of modifying the image parameters using the Gaussian function. It is mainly used to reduce the noise in the image and minimize the image details. The visual effect of this blurring effect resembles that of viewing the image through a translucent lens. This process is also used as a pre-processing stage in order to enhance image structure this blur produces a nice, smooth effect. Given its wide range of uses, Gaussian blur is definitely one of the most popular blur effects (Eswar. S, 2015). Gaussian blur is a filter whose impulse response h is a Gaussian mask

$$f(x,y) = (f*h)(x,y) = \sum_{i=x_1}^{x_2} \sum_{j=y_1}^{y2} f(x-i, y-j)h(i,j) \tag{1.7}$$

As allured to above, Box blurring is a form of low-pass filter for image processing. The kernel utilizes uniform values, meaning that every element within the kernel is equivalent. Because this uniformity is relatively simple to calculate compared to some other methods, box blurring is a fairly quick and common method for blurring images (Pulfer, 2019). Box blur is a filter whose impulse response h is a Gaussian mask

$$f(x,y) = (f*h)(x,y) = \frac{1}{2n+1 \times 2n+1} \sum_{i=-n}^{n} \sum_{j=-n}^{n} f(x+i, y+j) \tag{1.8}$$

The main parameter in box blurring is the size of kernel you would like to use.

$$\begin{bmatrix} 0.1111 & 0.1111 & 0.1111 \\ 0.1111 & 0.1111 & 0.1111 \\ 0.1111 & 0.1111 & 0.1111 \end{bmatrix} \quad \begin{bmatrix} 0.0400 & 0.0400 & 0.0400 & 0.0400 & 0.0400 \\ 0.0400 & 0.0400 & 0.0400 & 0.0400 & 0.0400 \\ 0.0400 & 0.0400 & 0.0400 & 0.0400 & 0.0400 \\ 0.0400 & 0.0400 & 0.0400 & 0.0400 & 0.0400 \\ 0.0400 & 0.0400 & 0.0400 & 0.0400 & 0.0400 \end{bmatrix}$$

Figure 1.9: Box (average) kernels 3×3, 5×5

1.4.2.2 Motion blur

Motion blur effects may deliberately be introduced to create a sense of fast movement of the object and photographers use this to produce dramatic effect to the picture taken for more image appeal. (Kalalembang, 2009), Motion Blur helps to create a sense of motion, speed and action in a picture. If you want to create a “speed trail” effect, this is the blur effect you’ll want to use. Motion Blur is perfect for those who want to make cars seem like they’re moving. You’ve definitely seen Motion Blur applied in car advertisements.

Motion blur on digital image can be modeled as a convolution between the image and the motion blur kernel having point spread factor (PSF) distribution equals to the angle of the blur:

$$\mathrm{H}(x,y) = f^{\alpha} * m = \sum_{K=0}^{K=1} m_{\mathrm{K}} f(x + \mathrm{K}\cos\alpha, y + \mathrm{K}\sin\alpha) \tag{1.9}$$

With α and K is the coefficient and the direction of shifting angle, respectively. An example of motion blur kernel with blur length of 5, 7 and angle of 45 $^{\circ}$ are shown below

$$\begin{bmatrix} 0.0000 & 0.0000 & 0.0000 & 0.0501 & 0.0304 \\ 0.0000 & 0.0000 & 0.0519 & 0.1771 & 0.0501 \\ 0.0000 & 0.0519 & 0.1771 & 0.0519 & 0.0000 \\ 0.0501 & 0.1771 & 0.0519 & 0.0000 & 0.0000 \\ 0.0304 & 0.0501 & 0.0000 & 0.0000 & 0.0000 \end{bmatrix} \quad \begin{bmatrix} 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0145 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0376 & 0.1283 & 0.0145 \\ 0.0000 & 0.0000 & 0.0000 & 0.0376 & 0.1283 & 0.0376 & 0.0000 \\ 0.0000 & 0.0000 & 0.0376 & 0.1283 & 0.0376 & 0.0000 & 0.0000 \\ 0.0000 & 0.0376 & 0.1283 & 0.0376 & 0.0000 & 0.0000 & 0.0000 \\ 0.0145 & 0.1283 & 0.0376 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0145 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \end{bmatrix}$$

Figure 1.10: Motion blur with length of 5, 7 and angle of 45°

1.5 Quality Measurement Technique

The Mean Square Error (MSE) and the PSNR are the two error measurements that were usually used to evaluate image. MSE represents the cumulative squared error between the transformed and the original images. Whereas PSNR represents a measure of the peak error. The lower the value of MSE the lower the error. Below are the equations for MSE and PSNR respectively:

$$MSE = \frac{1}{MN}\sum_{x=1}^{M}\sum_{y=1}^{N}\left(f(x,y) - f(x,y)\right)^2 \tag{1.10}$$

Where: *M and N* are the dimensions of the image.

PSNR block computes the peak signal to noise ratio in decibels between two images. This ratio is often used as a quality measurement between the original and the resulting image. The higher the PSNR, the better the quality of the resulting image (Deshpande et al., 2018).

$$PSNR = 10log_{10}(peakval^2)/MSE \tag{1.11}$$

Where ***peakval*** is the maximum value for a pixel.

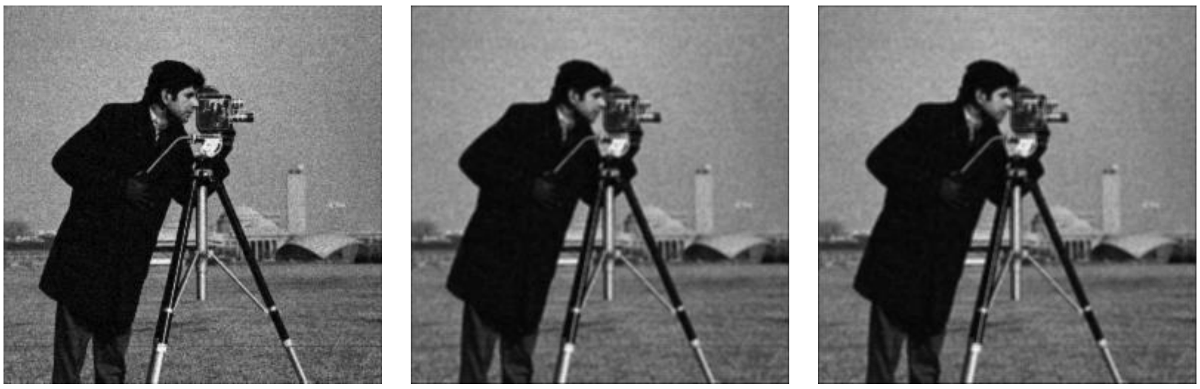
1.6 MATLAB results

Matrix laboratory software (MATLAB) has been used to analysis the performances of noise elimination filters and blurring filters.

1.6.1 Image Gaussian filtering

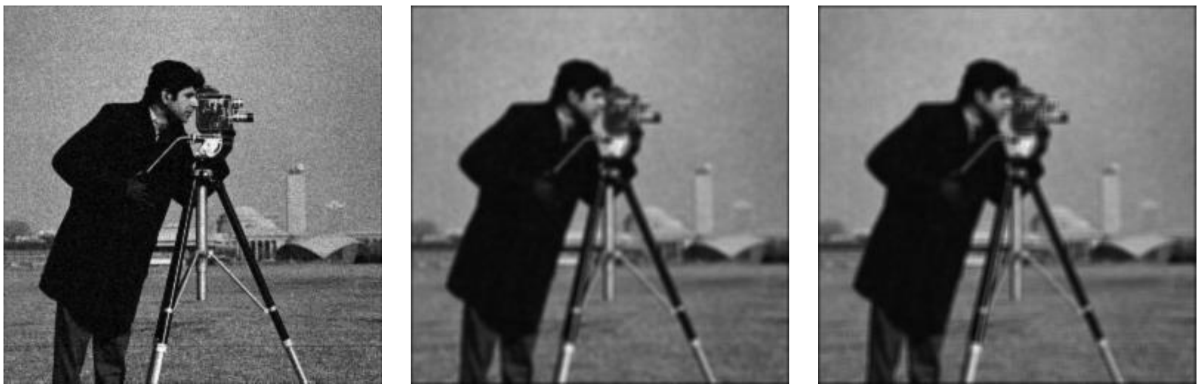
We applied a Gaussian filter with respectively (3x3) and (*5x5*) kernel with *σ=0.5, 2 and 8* on the image (cameraman.tiff) affected by Gaussian noise with *σ=0.002.*

The kernel will be conserved and sigma values will be changed in order to see their impact in the filtered image.



a) PSNR: 31.3887654 dB   b) PSNR: 23.5510394 dB   c) PSNR: 23.2932480 dB

Figure 1.11: Filtered image with 3x3 kernel and a) σ=0.5 b) σ=2 c) σ=8



a) PSNR: 31.3535081 dB   b) PSNR: 21.7628835 dB   c)PSNR: 21.1624726 dB

Figure 1.12: Filtered image with 5x5 kernel and a) σ=0.5 b) σ=2 c) σ=8

For different sigma values and kernel size, Table 1.1 resumes the corresponding PSNR of filtered images.

Table 1.1: PSNR values for different filtered images

σ	PSNR (3x3)	PSNR (5x5)
0.5	31.3887654 dB	31.3535081 dB
2	23.5510394 dB	21.7628835 dB
8	23.2932480 dB	21.1624726 dB

Normally, if PSNR value is more than 40 dB, this is an indication that the quality of the image is good. But, if the image is mean quality, the PSNR value is less than 30 dB which is the case of our selected image.

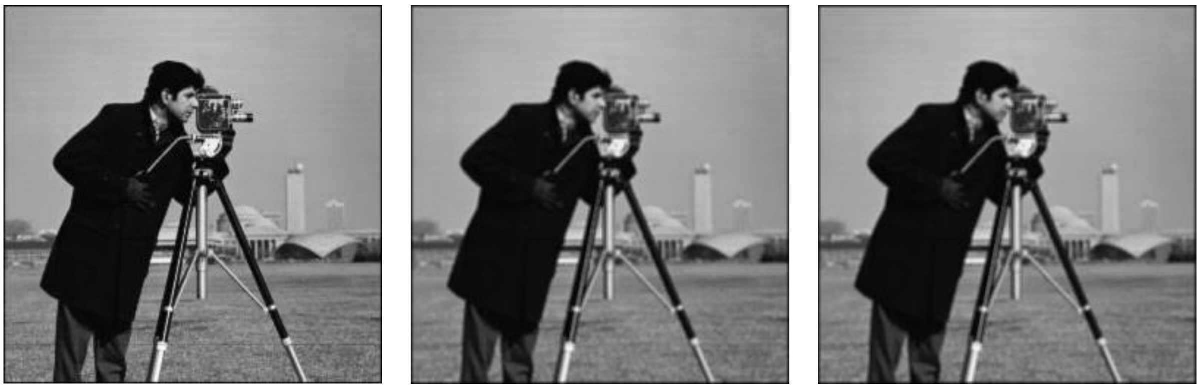
The Gaussian filtering effect depends on 2 main factors kernel matrix size and the σ (sigma). We note that when we increase the sigma value and the kernel size, the Gaussian noise is reduced, but not completely removed and the effect of smoothing increase and the PSNR decrease. So, Gaussian filtering is best suited filters for low Gaussian noise with low sigma value and kernel size

1.6.2 Image blurring

We applied a blurring filter with respectively (3x3) and (5x5) kernel on the image (cameraman.tiff).

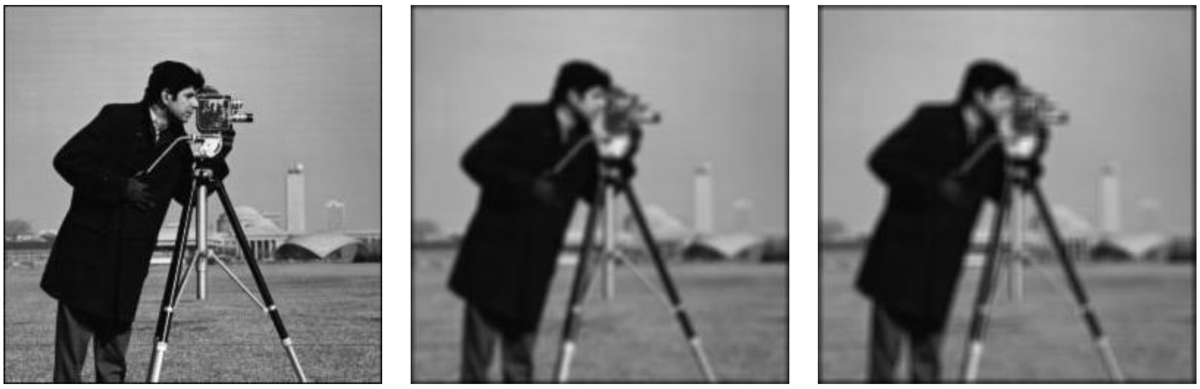
1.6.2.1 Gaussian blur

The kernel will be conserved and sigma values will be changed in order to see their impact in the blurred image.



a) PSNR: 37.5534494 dB b) PSNR: 33.9966417 dB c)PSNR: 33.8973597 dB

Figure 1.13: Gaussian blur with 3x3 kernel and a) σ=0.5 b) σ=2 c) σ=8



a) PSNR: 37.5412050 dB b) PSNR: 32.7157166 dB c)PSNR: 32.7157166 dB

Figure 1.14: Gaussian blur with 5x5 kernel and a) σ=0.5 b) σ=2 c) σ=8

In the above images we can see the amount of blurring produced by different sizes of kernel matrix and different value of sigma. Also from the images it is evident that the blurring effect increases with the increase in kernel matrix size and sigma values. Figure 1.13 shows the blurring effect associated with the kernel size is 3x3 and multiple values of sigma. It is observed that the blurring effect increases with increase in the sigma. Similar variation is seen in figure 1.14 and figure 25 along with the change in size of the matrix.

Table 1.2: PSNR values for different Gaussian blurred images

σ	PSNR (3x3)	PSNR (5x5)
0.5	37.5534494 dB	37.5412050 dB
2	33.9966417 dB	32.7157166 dB
8	33.8973597 dB	32.7157166 dB

The Gaussian blur effect also depends on 2 main factors kernel matrix size and the σ (sigma). The blurring effect is directly proportional to sigma. As the sigma value increases, more is the blurring effect on the image. This is because the deviation from the center pixel is more and will be negligible as the weighted matrix elements are reduced and making the convoluted value smaller. This will increase the blurring effect on image. Also, the other factor that affects blurring is the kernel matrix size. The blurring is more when the kernel size increases. Fig1.15 shows the plot of PSNR against various values of the Gaussian blur standard deviation and indicate that the blurring is more when the kernel size increases.

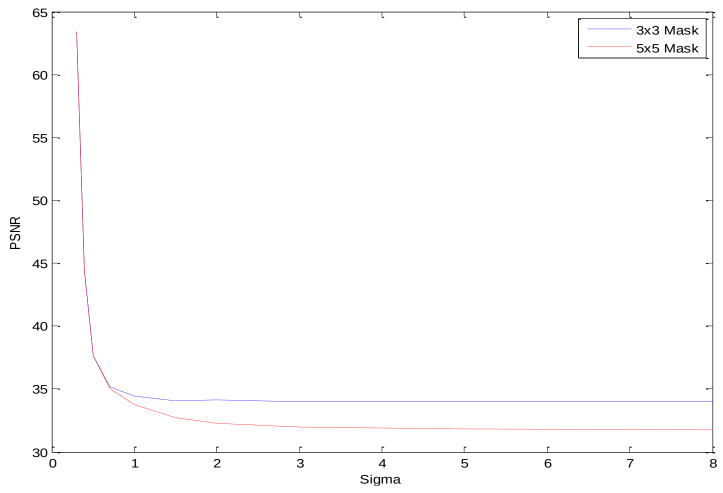
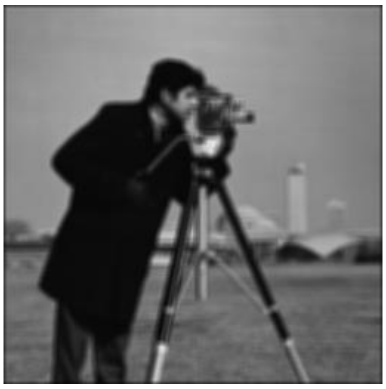


Figure 1.15: PSNR Computed against different sigma values.

1.6.2.2 Box and Motion blur

We applied a box blur with respectively (5x5) and (7x7) kernel and motion blur with length of 5, 7 and angle of 45°.


a) PSNR:32.4845860 dB

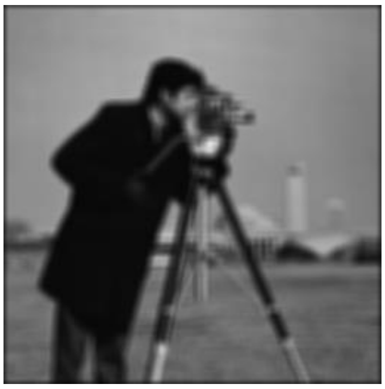
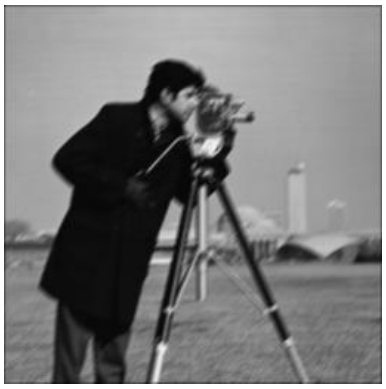

b) PSNR: 31.6905225 dB

Figure 1.16: Box blur with kernel a) 5x5, b) 7x7


a) PSNR:33.2743997 dB

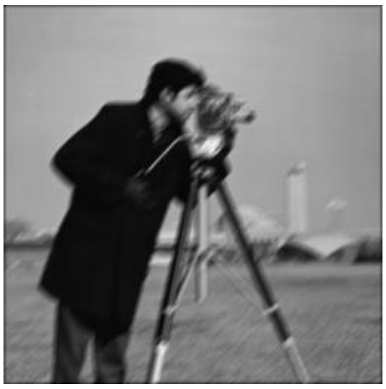

b) PSNR:32.5897816 dB

Figure 1.17: Motion blur with length of 5, 7 and angle of 45°

From the images (figure 1.16) and table 1.3, it is evident that the box blur effect increases with the increase in kernel matrix size while the motion blur is more when the blur length increases.

Table 1.3: PSNR values for box and motion blur

Box blur	5x5	32.4845860 dB	7x7	31.6905225 dB
Motion blur	$K = 5, \alpha = 45°$	33.2743997 dB	$K = 7, \alpha = 45°$	32.5897816 dB

1.7 Conclusion

In this chapter, we have introduced spatial image filtering and blurring algorithms, we started with Gaussian filter by restoring images that has been corrupted by Gaussian noise which follow Gaussian distribution then we presented box and motion blurring filters. MATLAB has been used to analysis the performances of noise elimination and blurring effects.

Conventional smoothing filters always tend to blur the images, so for noise rernovaI tasks, the filter should have the ability to preserve features in an image while reducing the noise.

Chapter 2

Xilinx FPGA

Spartan-3E (XC3S500E)

Chapter 2: Xilinx FPGA Spartan-3E (XC3S500E)

2.1 Introduction

FPGAs have become increasingly common in recent years due to their high performance compared to processors running software. FPGAs are an attractive choice due to their high performance, low energy dissipation per unit computation and re-configurability. The parallel computing power of the PGA is extremely useful in the modern world of demanding applications like signal and image processing. These flexible platforms are quickly maturing in logic capacity or programmable devices and the availability of embedded modules (Multipliers and Hard Cores) (Elamaran et al., 2012)

In this chapter, we give a brief overview of the FPGA device, a wide description of Xilinx Spartan-3E (XC3S500E) with programming tools and languages and provide FPGA design flow.

2.2 Overview of a general FPGA device

A field programmable gate array (FPGA) is a logic device that contains a two-dimensional array of generic logic cells and programmable switches. The conceptual structure of an FPGA device is shown in figure 2.1.

A logic cell can be configured to perform a simple function, and a programmable switch can be customized to provide interconnections among the logic cells. A custom design can be implemented by specifying the function of each logic cell and selectively setting the connection of each programmable switch. Once the design and synthesis is completed, we can use a simple adaptor cable to download the desired logic cell and switch configuration to the FPGA device and obtain the custom circuit. Since this process can be done "in the field" rather than "in a fabrication facility," the device is known as field programmable (Chu, 2011).

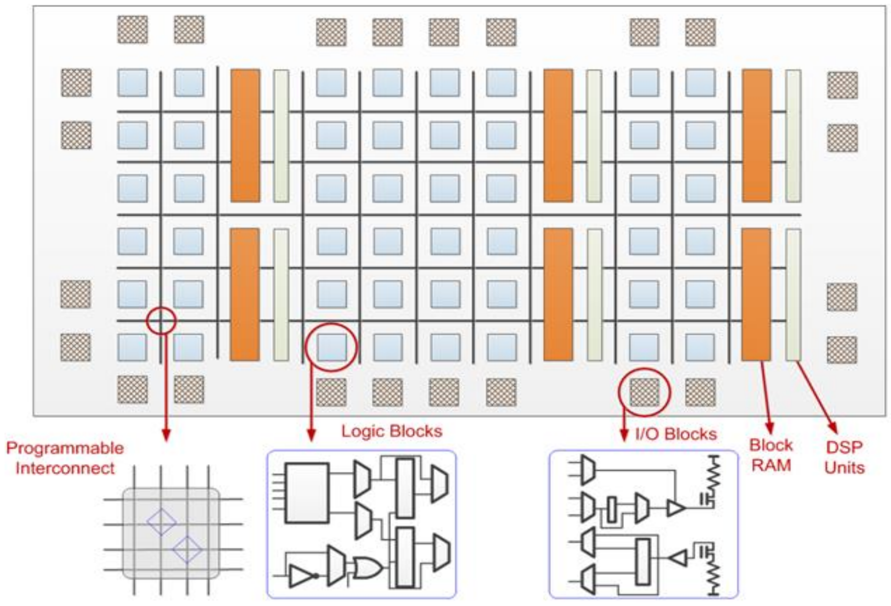


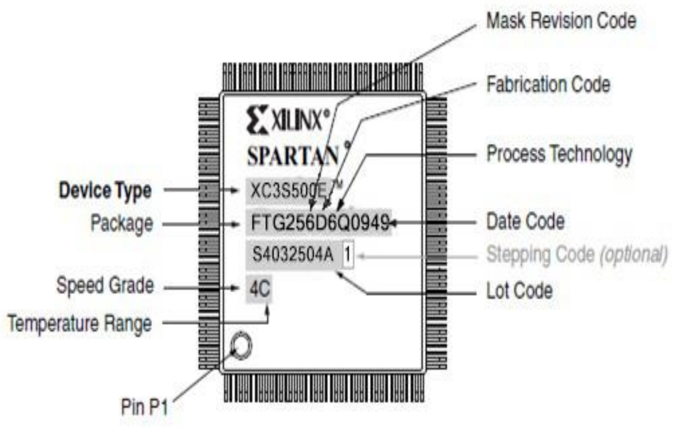
Figure 2.1: Conceptual structure of an FPGA device.

2.3 Xilinx Spartan-3E (XC3S500E)

The Spartan-3E family of Field-Programmable Gate Arrays (FPGAs) is specifically designed to meet the needs of high volume, cost-sensitive consumer electronic applications. Offers densities ranging from 100,000 to 1.6 million system gates (Xilinx, 2018).

2.3.1 Package Marking and ordering information

FPGA XC3S500E Xilinx Spartan-3E is a PLCC (Plastic Leaded Chip Carrier) circuit with 256 pins; figure 2.2 provides a top marking for XC3S500E in the quad-flat packages.



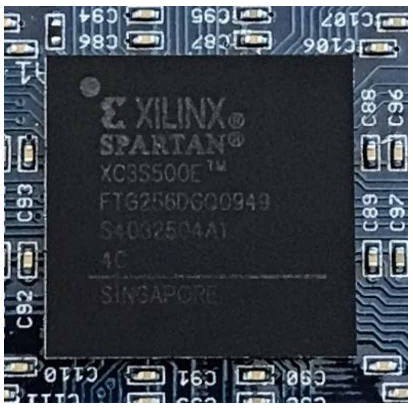


Figure 2.2: FPGA XC3S500E Package Marking

Marking and ordering information are as follows:

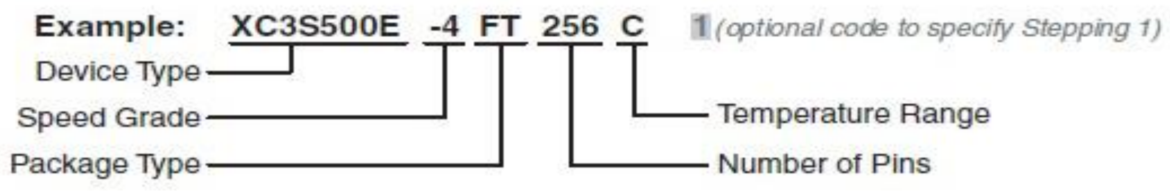

Example: XC3S500E -4 FT 256 C 1 (optional code to specify Stepping 1)
Device Type
Speed Grade
Package Type
Temperature Range
Number of Pins

Figure 2.3: Xilinx Spartan-3E (XC3S500E) Ordering Information

Valid device/package combinations are given in Table 2.1

Table 2.1**:** *Device/Package combinations*

Device	**Speed Grade**	**Package Type / Number of Pins**	**Temperature Range (TJ)**
XC3S500E	-4 : Standard Performance	FT256 :256-ball Fine-Pitch Thin Ball Grid Array (FTBGA)	C : Commercial (0°C to 85°C)

2.3.2 Architectural Overview

The Spartan-3E family architecture consists of five fundamental programmable functional elements: (Xilinx, 2018).

- **Configurable Logic Blocks (CLBs)** contain flexible Look-Up Tables (LUTs) that implement logic plus storage elements used as flip-flops or latches. CLBs perform a wide variety of logical functions as well as store data.
- **Input/Output Blocks (IOBs)** control the flow of data between the I/O pins and the internal logic of the device. Each IOB supports bidirectional data flow plus 3-state operation. Supports a variety of signal standards, including four high-performance differential standards. Double Data-Rate (DDR) registers are included.
- **Block RAM** provides data storage in the form of 18-Kbit dual-port blocks.
- **Multiplier Blocks** accept two 18-bit binary numbers as inputs and calculate the product.
- **Digital Clock Manager (DCM) Blocks** provide self-calibrating, fully digital solutions for distributing, delaying, multiplying, dividing, and phase-shifting clock signals.

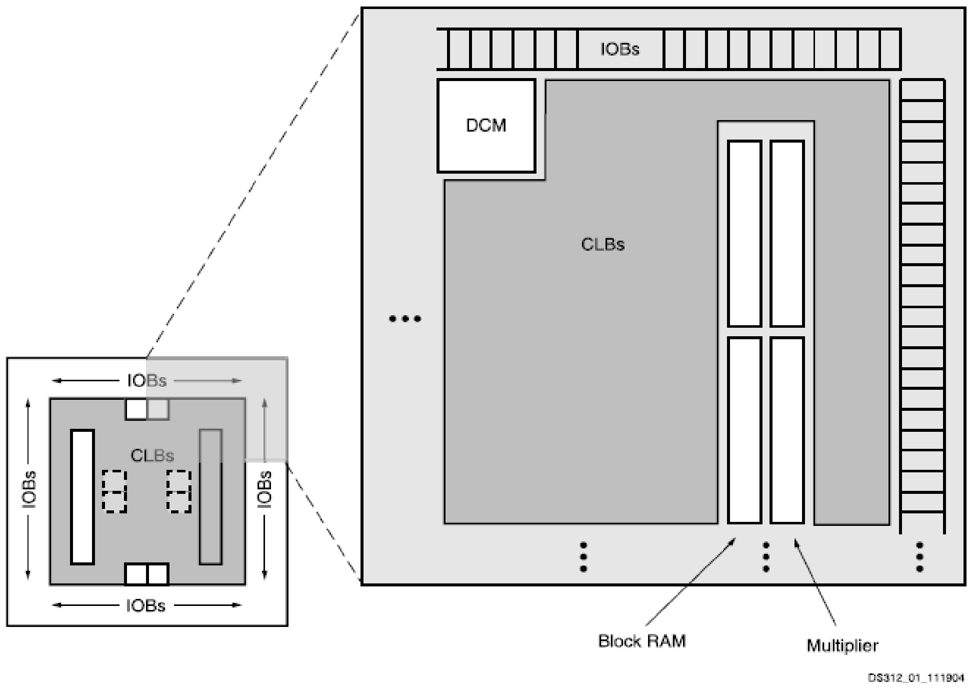


Figure 2.4: Xilinx Spartan-3E (XC3S500E) architecture

2.4. FPGA Configuration

The FPGA is made of SRAM (Volatile Memory) so the data configured inside FPGA lost at power Off state. FPGA Configuration is the process of loading the FPGA chip with Configuration data through external devices during power On state the Method of Configuring FPGA Can be divided in to:

2.4.1. Master mode

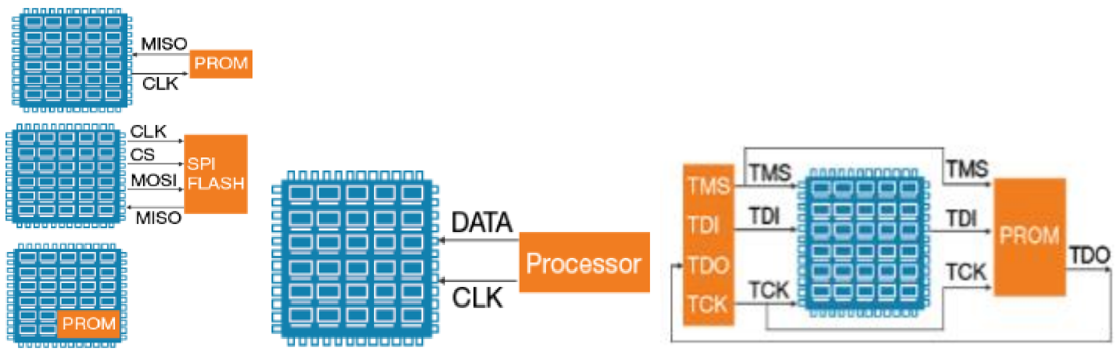
In the Master Mode the configuration data is stored in external nonvolatile memories such as SPI FLASH, Parallel FLASH, PROM and so on. During configuration process the data is loaded in the FPGA Configurable Logic Blocks to operate as a specific application. The configuration clock is provided by FPGA in Master Mode operation.

2.4.2. Slave mode

In Slave Mode, the entire configuration process is controlled by external device. Those external device may be of processor, Microcontroller, and so on. The Configuration can performed serially or parallel method. The Clock input is supplied by the external device for Slave mode.

2.4.3. JTAG mode

The four-wire JTAG interface is common on board testers and debugging hardware.FPGA mainly uses JTAG interface for prototype download and debugging. Nearly all FPGAs are configured through JTAG, making it a good interface for tools that need to interact with a variety of devices (Gruwell, 2016).



(a) (b) (c)

Figure 2.5: FPGA configuration(a) Master mode (b) Slave mode (c) JTAG mode

2.5 Summary of FPGA XC3S500E Xilinx Spartan-3E Attributes

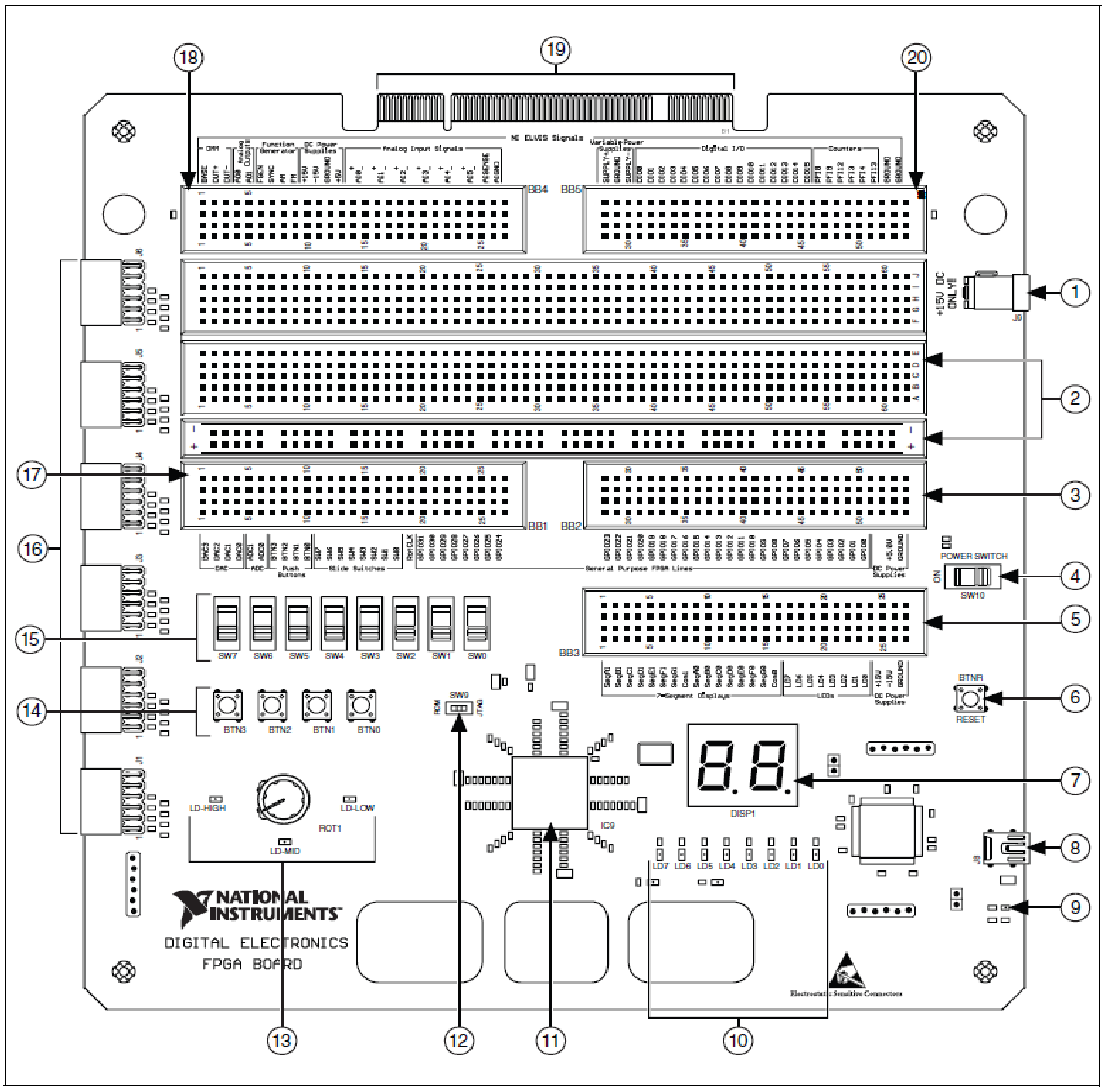
Table 2.2 gives the summary of FPGA XC3S500E Xilinx Spartan-3E attributes (Xilinx, 2018)

Table 2.2 : FPGAXC3S500 attributs

Device	**System Gates**	**Equivalent Logic Cells**	**CLB Array (One CLB = Four Slices)**				**Distributed RAM bits**	**Block RAM bits**	**Dedicated Multipliers**	**DCMs**	**Maximum UserI/O/ Maximum Differential I/O Pairs**
			Rows	**Columns**	**Total CLBs**	**Total Slices**					
XC3S500E	500K	10,476	46	34	1,164	4,656	73K	360K	20	4	232

2.6 NI Digital Electronics FPGA Board

The NI Digital Electronics FPGA Board is a circuit development platform based on the XC3S500E Xilinx Spartan-3E FPGA. Besides the FPGA, the board contains slide switches, LEDs, a two digit seven-segment display, push-buttons, a rotary push-button knob and LEDs for one external clock, Digilent Pmod terminals for external attachments, USB download interface, and large breadboard area for digital electronics circuitry experimentation.(NI,2009)



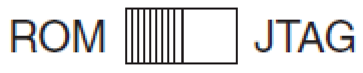
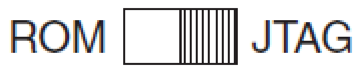
1	Power Connector	7	Seven-Segment Displays	14	Push Buttons
2	General-Purpose Breadboard Banks	8	USB Connector	15	Slide Switches
3	Signal Breadboard Bank BB2	9	LD-G LED	16	Digilent Pmod Connectors
4	Power Switch	10	LEDs	17	Signal Breadboard Bank BB1
5	Signal Breadboard Bank BB3	11	FPGA	18	Signal Breadboard Bank BB4
6	Reset Button	12	Switch SW9	19	NI ELVIS Connector
		13	Rotary Push-Button Knob/LEDs	20	Signal Breadboard Bank BB5

Figure 2.6: The NI Digital Electronics FPGA Board

2.7 FPGA Boot-Up Options

Boot-up selection on the NI Digital Electronics FPGA Board is controlled by switch SW9. The two FPGA boot-up options are described in Table 2.3.

Table2.3:FPGA Boot-Up Options

. FPGA Boot-Up Option	Switch Position
Boot-up from ROM (default)—Configures the FPGA from the image stored in the Platform Flash PROM	
Boot-up from JTAG—Does not load anything into the FPGA, waits for program download from USB-JTAG port.	

2.8 FPGA languages

Generally the languages used to program the FPGA can be divided to:

2.8.1 Hardware Description Language HDL

FPGAs have traditionally been configured by hardware engineers using a Hardware Design Language (HDL). The two principal languages used are Verilog HDL (Verilog) and Very High Speed Integrated Circuits (VHSIC) HDL (VHDL) which allows designers to design at various levels of abstraction. Verilog and VHDL are specialized design techniques that are not immediately accessible to software engineers, who have often been trained using imperative programming languages. Consequently, over the last few years there have been several attempts at translating algorithmic oriented programming languages directly into hardware descriptions (Arunmozhi & Mohan, 2012).

2.8.2 Software based languages:

Have a single source that can be compiled to both hardware and software. This results in speeding up the development process considerably (Abdelrahman, 2017). Generally software based languages can be categorized as follows:

- **Structural approach:** high level synthesis based on structure hardware language (e.g. JHDL and Quartz).
- **Augmented Languages:** mostly based on C language and can be used for modelling complete system (e.g Handel-C,Mitrion-C and System-C).

2.8.3 Graphical programming

MATLAB and Simulink for model-based design provide signal, image and video processing engineers with a development platform that spans design, modeling, simulation, code generation, and implementation.

When using MATLAB and Simulink to target FPGAs or ASICs (Application Specific Integrated Circuits), we can first design and run a simulation of the system with MATLAB, Simulink and state flow and then generate bit-true, cycle-accurate, synthesizable Verilog and VHDL code using System Generator which is provided by MATLAB(Abdelrahman, 2017).

2.9 Design flow in FPGA

Application Specific Integrated Circuit (ASIC) designs or sections of these designs that are targeted for FPGAs are often created with HDLs. The FPGA design flow comprises of several steps, namely HDL design entry, performing a Register Transfer Level (RTL) simulation of the design ,design synthesis, creating a Xilinx Netlist File (XNF) file, performing a functional simulation, floor planning the design (this step is optional) then design implementation (mapping place and route) and device programming. Figure 2.7 gives an overview of the FPGA design flow carried out by specific automation tools.

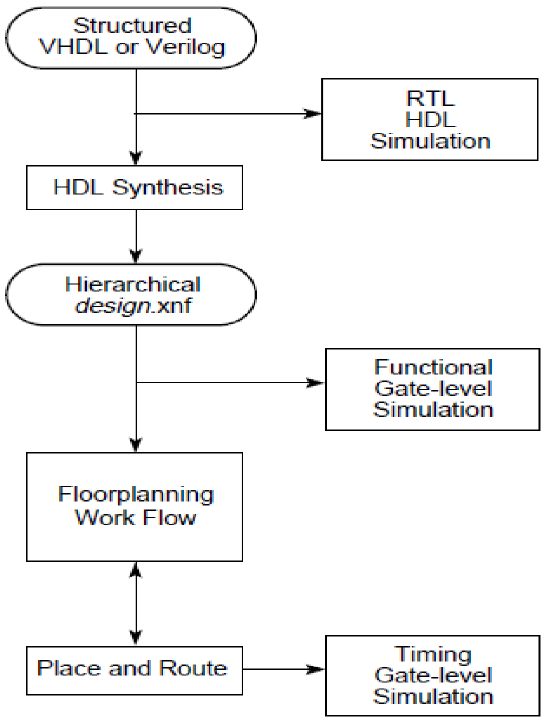

Structured VHDL or Verilog
RTL HDL Simulation
HDL Synthesis
Hierarchical *design*.xnf
Functional Gate-level Simulation
Floorplanning Work Flow
Place and Route
Timing Gate-level Simulation

Figure 2.7: HDL Flow Diagram for a New Design

2.10 Conclusion

The Spartan-3E platform builds on the success of the earlier Spartan-3 platform by adding new features that improve system performance and reduce the cost of configuration. Because of their exceptionally low cost, Spartan-3 generation FPGAs are ideally suited to a wide range of consumer electronics applications, including broadband access, home networking, display/projection, and digital television equipment.

Chapter 3

Implementation of image filtering and blurring in FPGA using Xilinx System Generator

Chapter 3: Implementation of image filtering and blurring in FPGA using Xilinx System Generator

3.1 Introduction

Usually filtering algorithms are implemented using software-based, however, it is not limited to software development but also hardware development.System Generator provides accelerated simulation through hardware Co-simulation. System Generator will automatically create a hardware simulation token for a design captured in the Xilinx DSP blockset that will run on one the supported hardware platforms. This hardware will co-simulate with the rest of the Simulink system to provide up to a 1000x simulation performance increase.

In this chapter we will discuss the Simulink model of filtering and blurring algorithms using Xilinx system generator and the implementation of the design targeting a Spartan3e device (xc3s500e) using hardware co-simulation. We compare the image qualityofhardwareCo-Simulation using the Peak Signal-to-Noise Ratio (PSNR). Also, the FPGAresource usage for different sizes of Gaussian kernel will bepresented.

.3.2 Design methodology for implementation onFPGA with Xilinx system generator

For accomplishing Image processing task using Xilinx System Generator needs two Software tools to be installed. One is MATLAB Version R2010a or higher and Xilinx ISE 14.1 or higher.

Xilinx ISE Design Suite 14.7 software which is a design and project management software for FPGA circuits and it is an FPGA circuit design tool from Xilinx. This software allows the simulation of the description and the synthesis of the equivalent logic circuit then the placement and the routing of the circuit on a prototype corresponding to a very precise FPGA technology and finally all the verifications are made comes the implantation on a real FPGA which corresponds in generating the configuration file of the chosen target circuit in order to establish the interconnections of the logic cells corresponding to the logic circuit designed with an optimization of the resources available at the level of the programmable FPGA circuit.

MATLAB R2010a with Simulink from MathWorks. Simulink is anadditional MATLAB toolbox that provides for modeling,simulating and analyzing dynamic systems within a graphicalenvironment. The software allows for both modular andhierarchical models to be developed providing the advantage ofdeveloping a complex system design that is conceptuallysimplified.

Xilinx System Generator (XSG) is an integratordesign environment (IDE) for FPGAs, which usesSimulink, as a development environment, it is presentingin the form of block set.The System Generator token available along with Xilinx has to be configured to MATLAB. This result in addition of Xilinx Block set to the Matlab Simulink environment which can be directly utilized for building algorithmic model. Xilinx System Generator (XSG) has an integrated design flow, to move directly to the configuration file (*. bit) necessary for programming the FPGA. XSG automatically generates VHDL code and a draft of the ISE model being developed. Make hierarchical VHDL synthesis, expansion and mapping hardware, in addition to generating a user constraint file (UCF), simulation and test bench and test vectors among other things (Mehra, R., and al., 2010).

3.3 System generator flow diagram

System Generator works within the Simulink model-based design methodology. Often an executable spec is created using the standard Simulink block sets. This spec can be designed using floating-point numerical precision and without hardware detail. Once the functionality and basic dataflow issues have been defined, System Generator can be used to specify the hardware implementation details for the Xilinx devices. System Generator uses the Xilinx DSP blockset for Simulink and will automatically invoke Xilinx Core Generator to generate highly-optimized netlists for the DSP building blocks. System Generator can execute all the downstream implementation tools to product a bit stream for programming the FPGA. An optional test bench can be created using test vectors extracted from the Simulink environment for use with ModelSim or the Xilinx ISE® Simulator. The System Generator of DSP(Gupta, 2015).

The hardware implementation on the FPGA requires the VHDL code and then the Bitstream file, which can be automatically generated using the design from the XSG. Figure

3.1 shows the broad flow design Xilinx SystemGenerator. As already mentioned, we can then move to theconfiguration file to program the FPGA, and the synthesis andimplementation steps are optional for the user but not for System Generator.

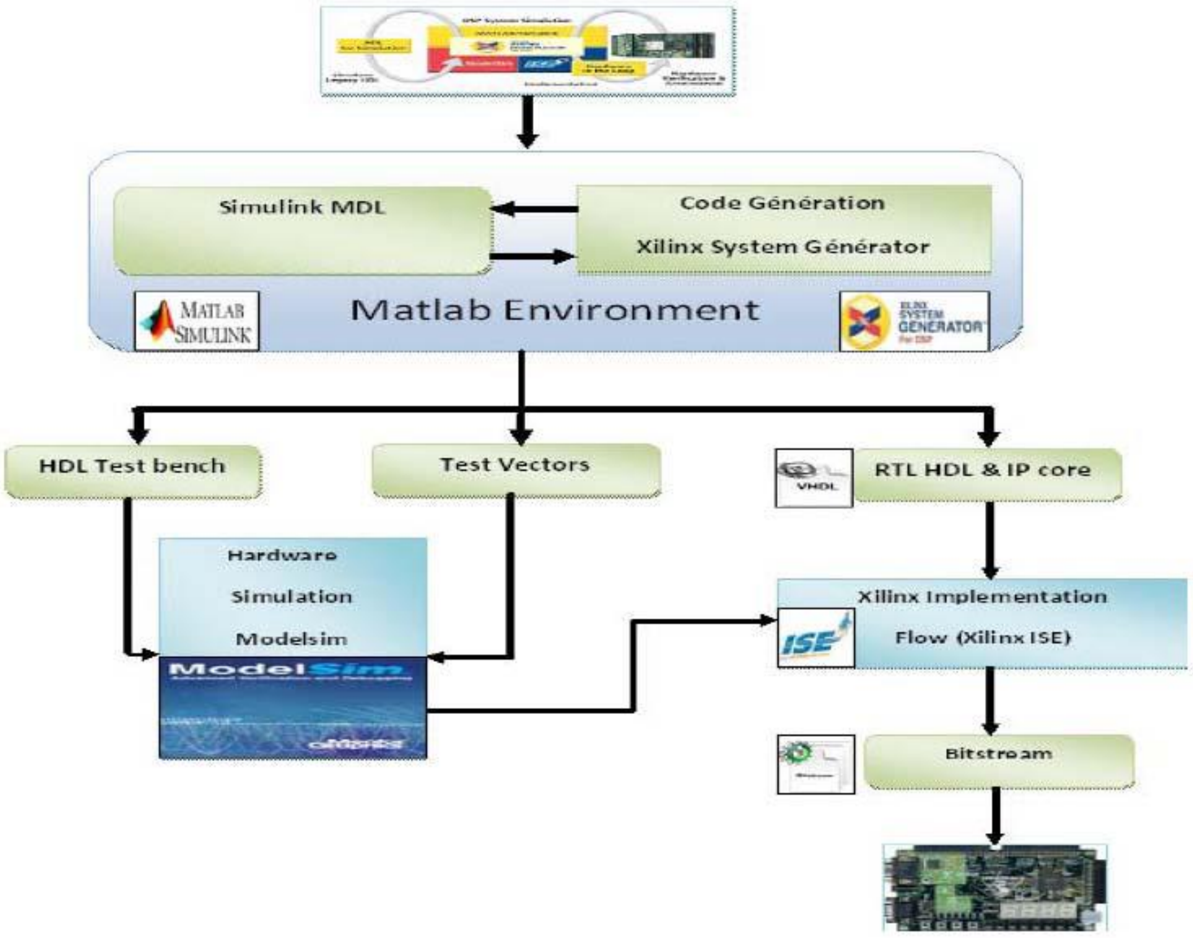


Figure 3.1: Design flow of System Generator

3.4 Image filtering and blurring design using Xilinx system generator

The entire operation for any image processing technique using Simulink and Xilinx blocks mainly goes through three phases:Image pre-processing, Image processing technique using XSG and Image postprocessing.

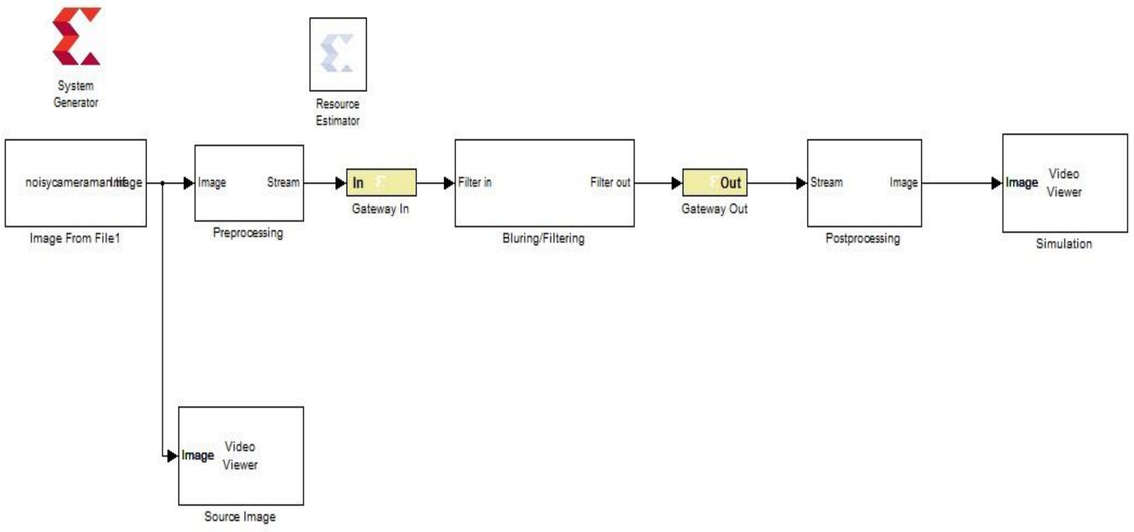
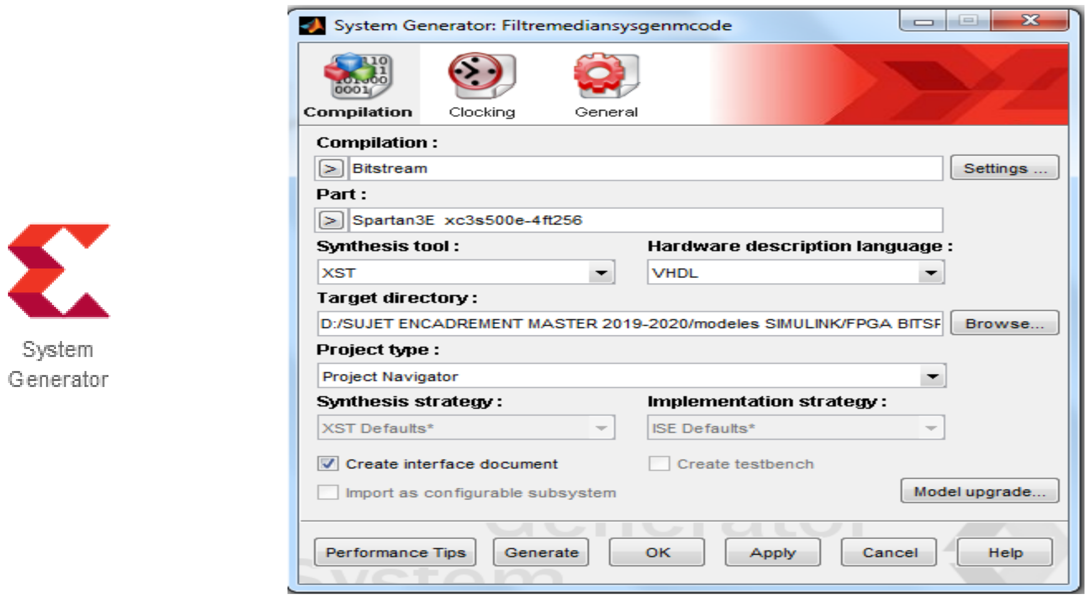


Figure 3.2:Simulink model of filtering/blurring design

3.4.1 System Generator Token

Every System Generator diagram requires that at least one System Generator token be placed on the diagram. This token is not connected to anything but serves to drive the FPGA implementation process. The property editor for this token allows to specify the target netlist, device, performance targets and system period. System Generator will issue an error if this token is absent.The symbol and the dialog box of the "System Generator" token are shown in the following figure:



System Generator: Filtremediansysgenmcode

Compilation :
Bitstream

Part :
Spartan3E xc3s500e-4ft256

Synthesis tool : XST
Hardware description language : VHDL

Target directory :
D:/SUJET ENCADREMENT MASTER 2019-2020/modeles SIMULINK/FPGA BITSF

Project type :
Project Navigator

Synthesis strategy : XST Defaults*
Implementation strategy : ISE Defaults*

[✓] Create interface document
[ ] Create testbench
[ ] Import as configurable subsystem

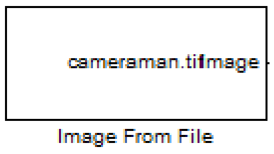
(a)

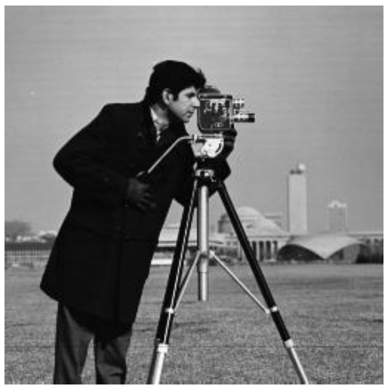
(b)

Figure 3.3:(a) Block System Generator (b) System Generator block dialog

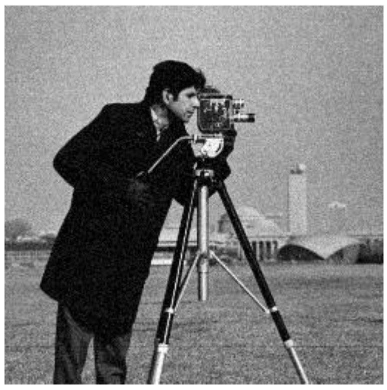
3.4.2 Test images

The Image from File block reads an image from a specified file location and imports it to the Simulinkworkspace. The input file contain a grayscale image “cameraman.tif”.





Original image



Noisy image (Gaussian noise $\sigma^2 = 0.002$

Figure 3.4 : Test images for blurring and filtering

3.4.3 Image Preprocessing Blocks

As image is two dimensional (2D) arrangement, to meet the hardware requirement the image should be preprocessed and given as one dimensional (1D) vector. The model based design used for image preprocessing is shown in Figure 3.5 , To process 2D image it is converted into 1D by using convert 2D to 1D block. Frame conversion block sets output signal to frame based data and provided to unbuffer block which converts this frame to scalar samples at a higher sampling rate.

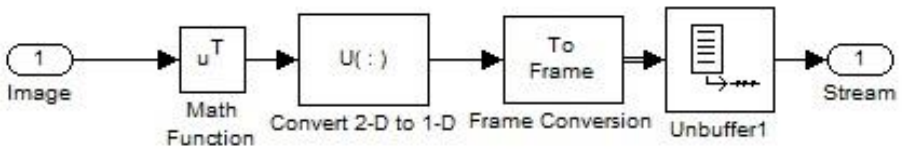


Figure 3.5: Image Pre-processing Blocks

3.4.4Gateway in and Gateway out

All Xilinx blocks should be connected between Gateway In and Gateway Out. Between those two blocks any technique can be designed. All Xilinx blocks work on fixed point but the real world signal (image, voice signal, etc.) are floating point so here the gateway in and gateway out blocks acts as translators for converting the real world signal into the desired form. The two blocks define the boundary of the FPGA from the Simulink simulation model

In
Gateway In
Out
Gateway Out

Figure 3.6:FPGA boundary from the Simulink simulation model

3.4.5 Blurring and Filteringalgorithms using XSG

The following figure illustrates the architecture of the 2D convolution block developed under XSG and which mainly consists of three major functions: line buffers, multipliers and adders. The convolution is the heart of both blurring and filtering algorithms.

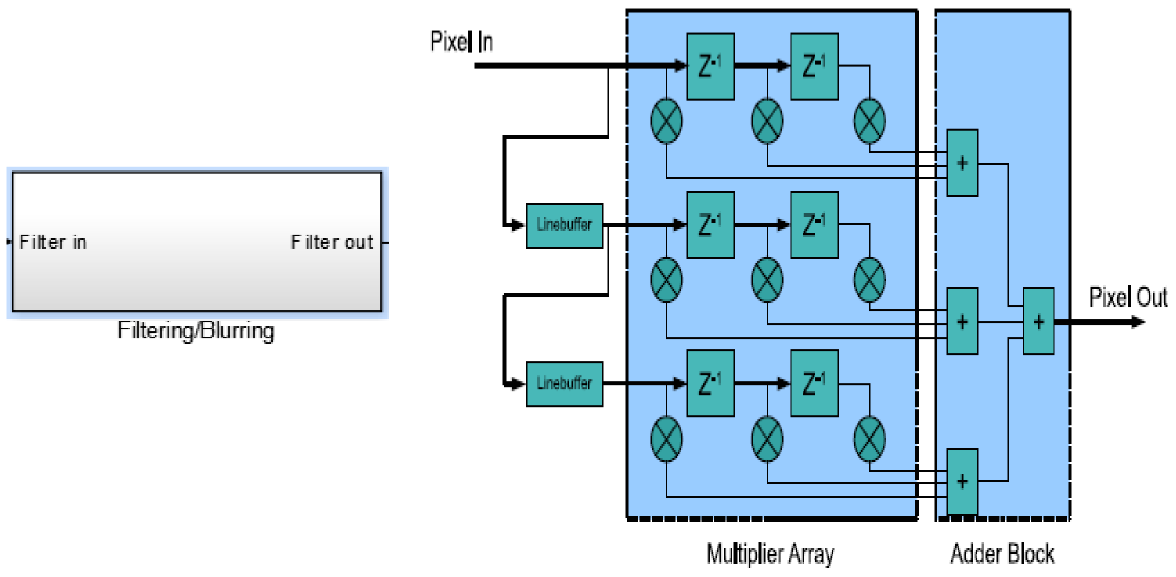


Figure 3.7: 2D convolution block developed under XSG

Figure 3.8,Figure 3.9 and Figure 3.10 show what's inside the Blurring/Filtering block using 3x3 , 5x5 and 7x7 kernel size.

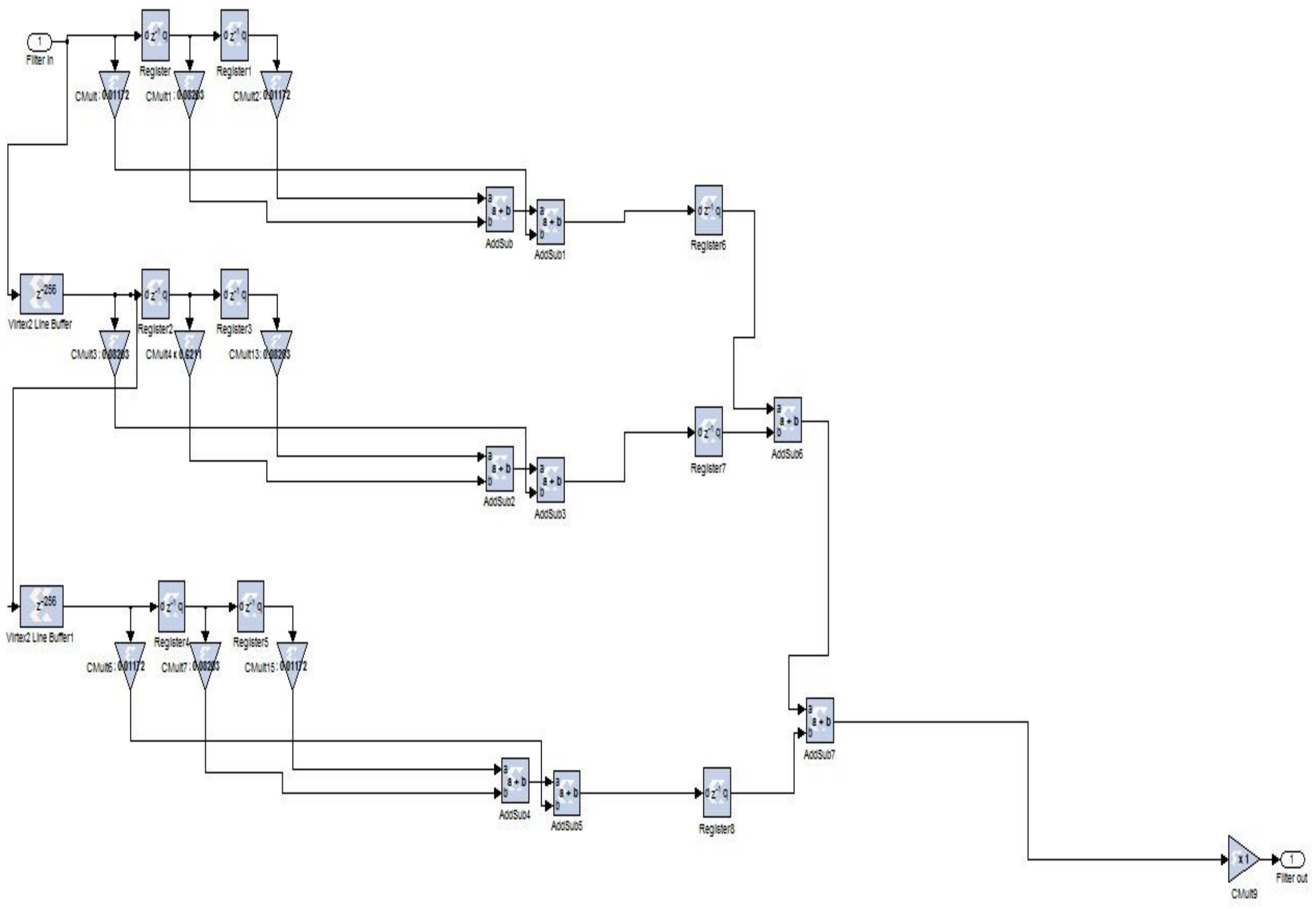


Figure 3.8: Blurring/Filtering block 3x3

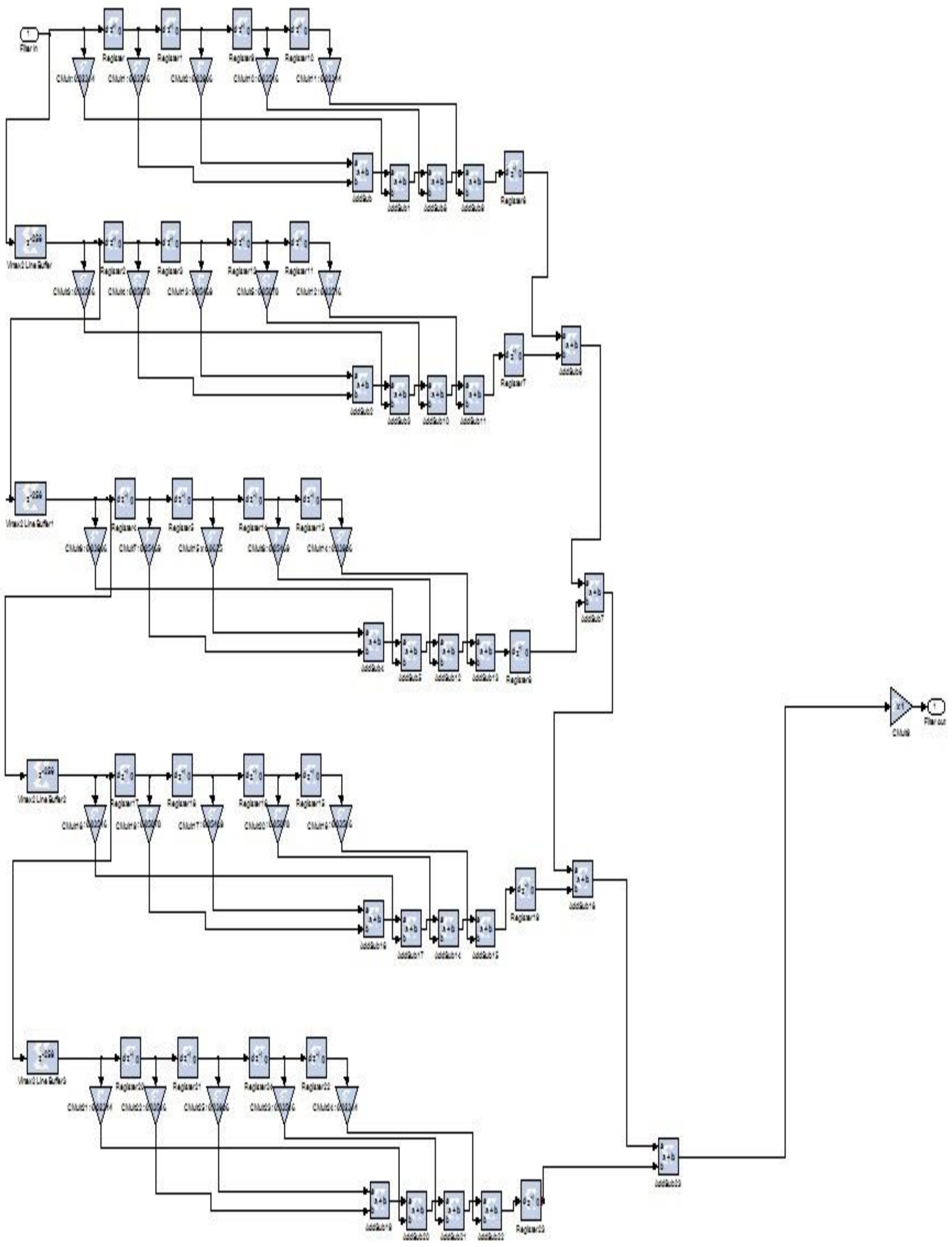


Figure 3.9: Blurring/Filtering block 5x5

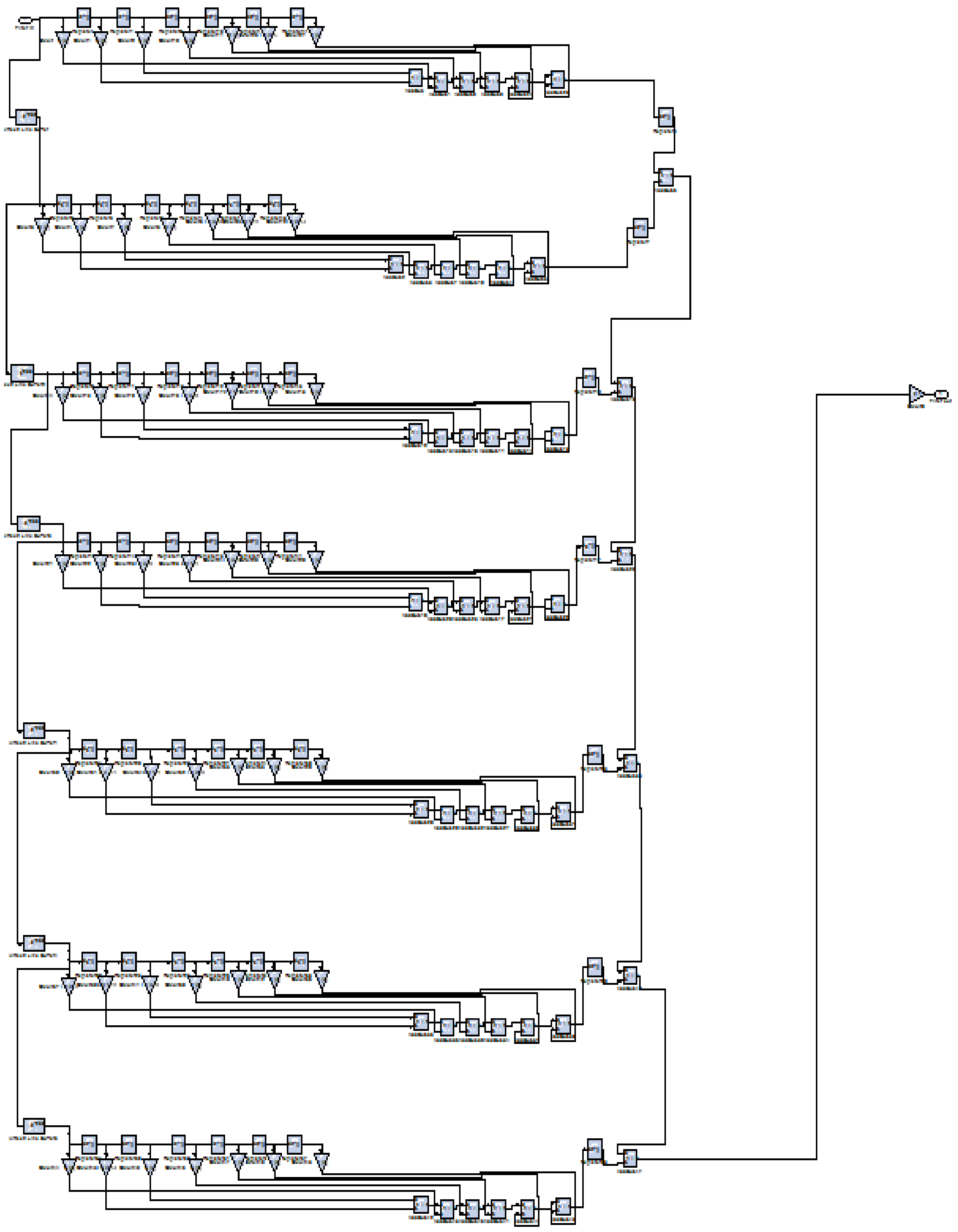


Figure 3.10: Blurring/Filtering block 7x7

The 2D filtering and blurring algorithm with 3x3 kernel maintains three line buffers. Each iteration the input pixel is pushed into the current line buffer that is being written to. The control logic rotates between these threebuffers when it reaches the column boundary. Each buffer is followed by a shiftregister and data at the current column index is pushed into the shift register. Ateach iteration a 3x3 kernel of pixels are formed from the pixel input, shift registersand line buffer outputs. The kernel is multiplied by a 3x3 filter coefficient maskand the sum of the result values is computed as the pixel output.

3.4.6 Image post processing blocks

The image post processing blocks which are used to convert the image output back to floating point type are shown in Figure 3.7. For post processing it uses a buffer block which converts scalar samples to frame output at lower sampling rate, followed by a 1D to 2D format signal block (Rao et al., 2015).

Image post- processing helps recreating image from 1D arrayshown in Fig 7. Post-processing uses(Swara& Madhumati, 2014).

Data type conversion: It converts image signal to unsignedinteger format.

Buffer: Converts scalar samples to frame output at lowersampling rate.

Convert 1D to 2D: Convert 1D image signal to 2D imagematrix.

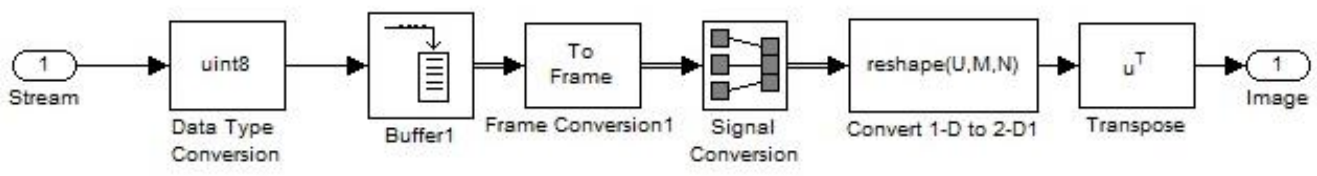


Figure 3.11:Image Post-processing Blocks

3.5 FPGA implementation

3.5.1 HardwareCo-Simulation

System Generator provides hardware co-simulation, making it possible to incorporate adesign running in an FPGA directly into a Simulink simulation. "Hardware Co-Simulation" compilation targets automatically create a bitstream and associate it to a block.When the design is simulated in Simulink, results for the compiled portion are calculatedin hardware.

This allows the compiled portion to be tested in actual hardware and canspeed up simulation dramatically.

Hardware Co-Simulation also known as Hardware in Loop (HIL) allows us to use Simulink software to test the design on real hardware for any existing HDL code. The following figure illustrates HIL

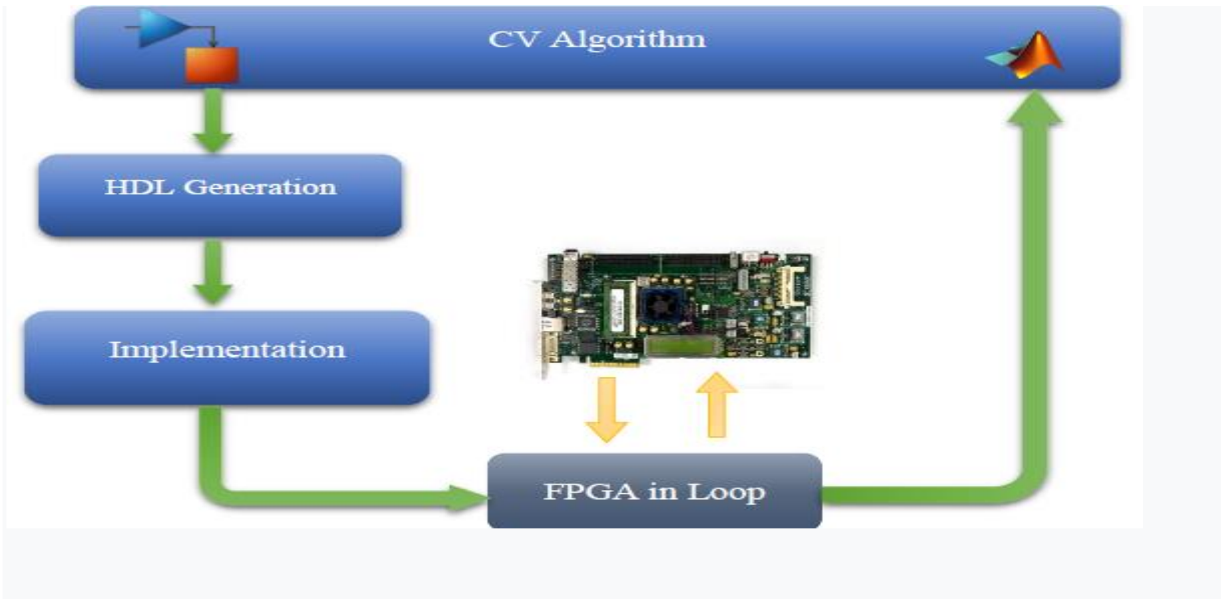


Figure 3.12: FPGA in Loop

3.5.2 Installing Hardware platform DEFB (Digital Electronics FPGA Board)

The first step in performing hardware co-simulation is to install and setup the hardware platform DEFB.

3.5.2.1 Creation of new compilation target

To create new compilation target, double click on System Generator token and in the compilation menu choose Hardware Co-Simulation and finally select New Compilation Target in the Hardware Co-Simulation submenu.

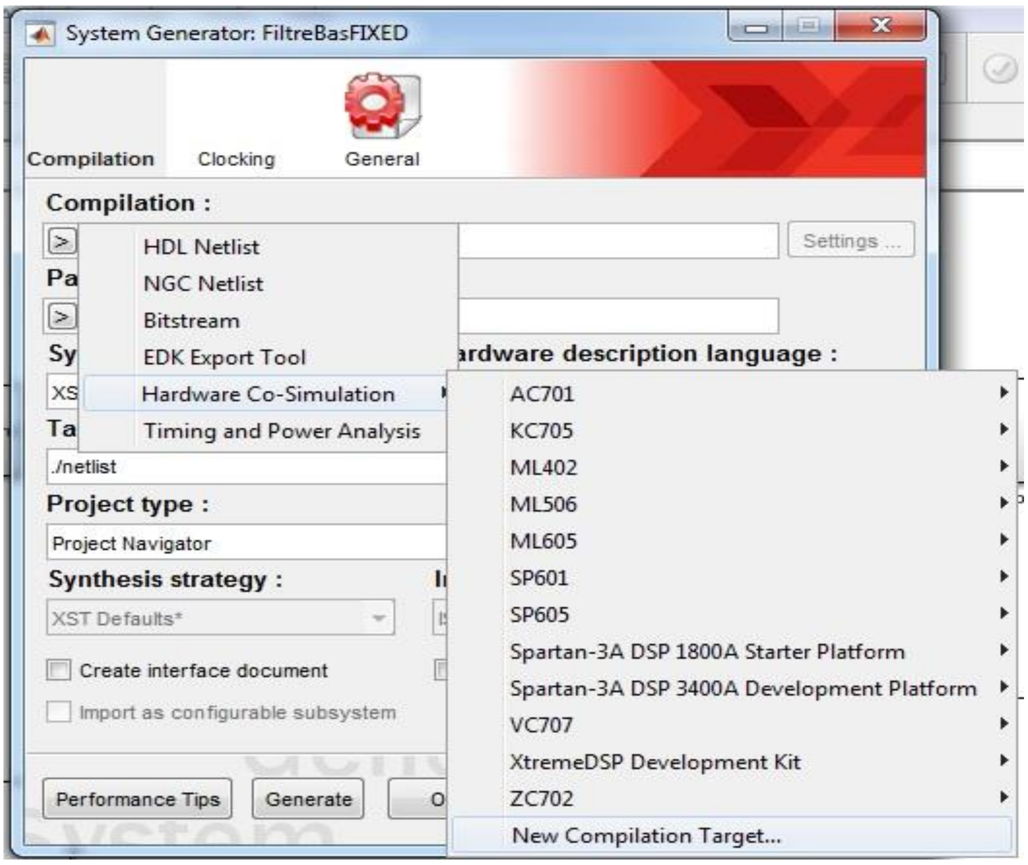
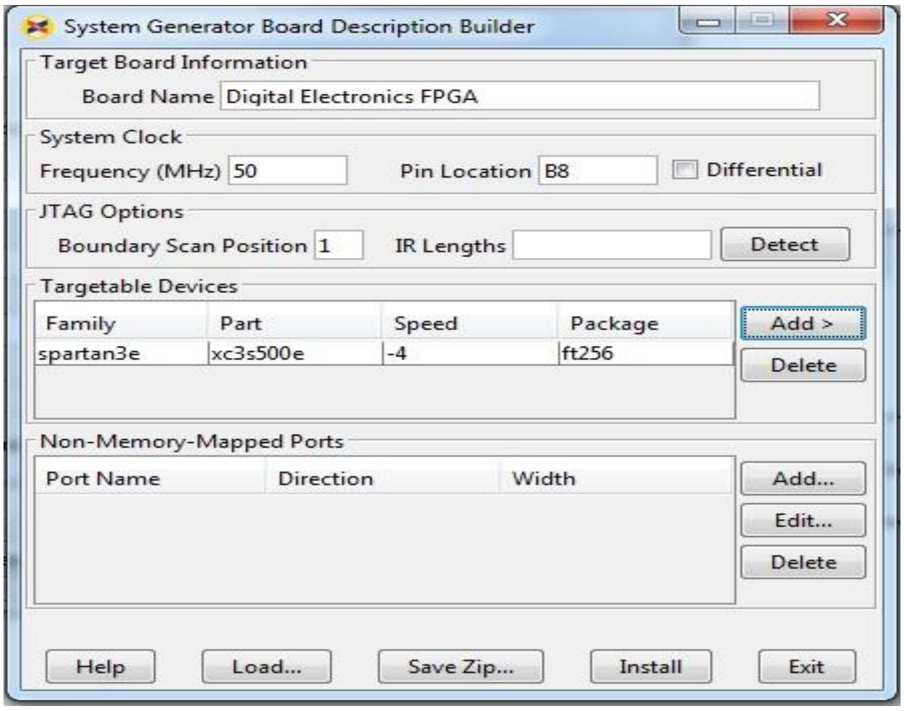

System Generator: FiltreBasFIXED
Compilation
Clocking
General
Compilation :
HDL Netlist
NGC Netlist
Bitstream
EDK Export Tool
Hardware Co-Simulation
Timing and Power Analysis
Settings ...
ardware description language :
AC701
KC705
ML402
ML506
ML605
SP601
SP605
Spartan-3A DSP 1800A Starter Platform
Spartan-3A DSP 3400A Development Platform
VC707
XtremeDSP Development Kit
ZC702
New Compilation Target...
./netlist
Project type :
Project Navigator
Synthesis strategy :
XST Defaults*
Create interface document
Import as configurable subsystem
Performance Tips
Generate

Figure 3.13: Selection of new compilation target

The following window appears to do the installation:


System Generator Board Description Builder

Target Board Information

Board Name: Digital Electronics FPGA

System Clock

Frequency (MHz): 50 Pin Location: B8 Differential

JTAG Options

Boundary Scan Position: 1 IR Lengths:

Targetable Devices

Family	Part	Speed	Package
spartan3e	xc3s500e	-4	ft256

Non-Memory-Mapped Ports

Port Name	Direction	Width

Figure 3.14: Installing Hardware platform

To do the installation, fill in all the fields.

Platform Name: Digital Electronics FPGA

Clock frequency: 50 MHz and the locator pin is B8

In order to fill in JTAoptions, start the XilinxiMPACT software with a click on Boundary scan then right click by choosing initialize JTAG chain which automatically connects to the cable and identifies boundary scan chain (devices of the DEFB platform).

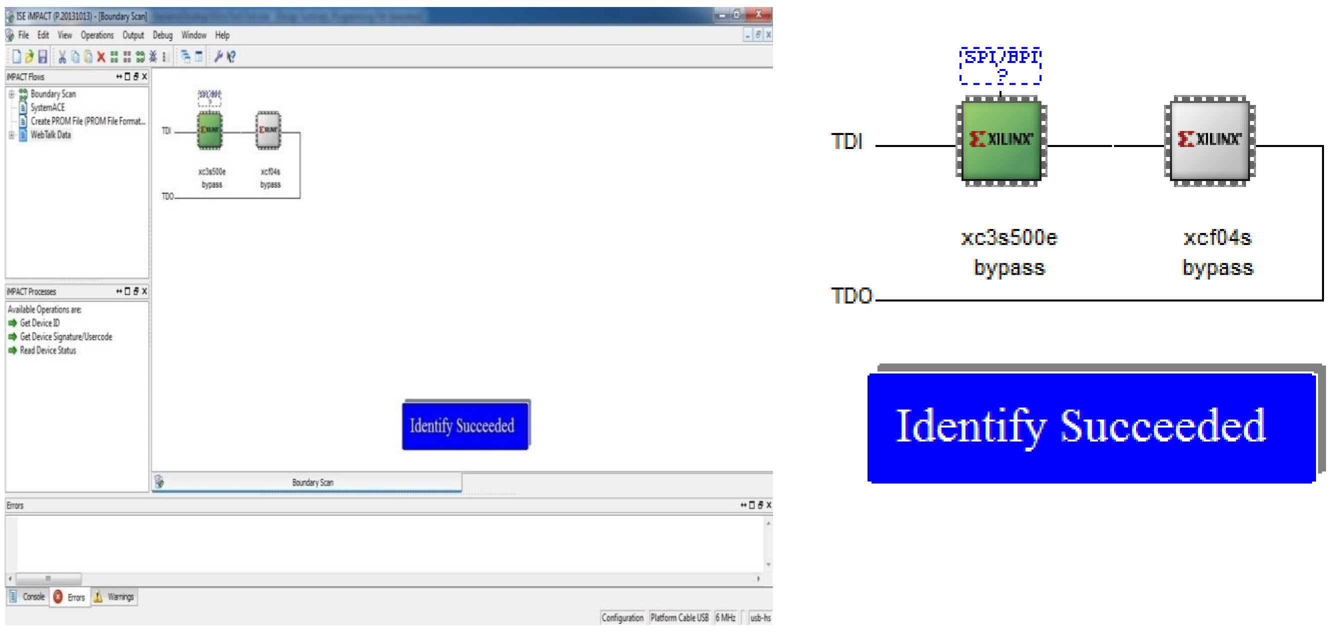


Figure 3.15: XilinxiMPACT APP

The existing circuits in the platform:

Device 1: (FPGA, xc3s500e) and device2: xcf04s

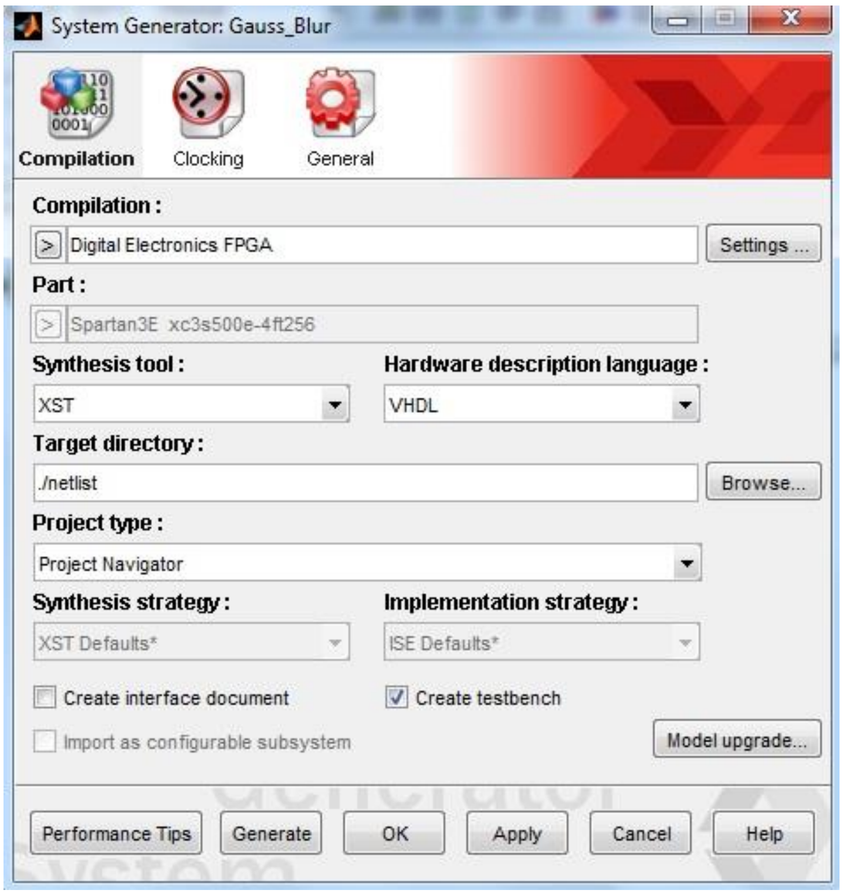
We choose Boundary Scan Position: 1 to program device 1 and we launch the detection of the length of the instruction register (IR length) and that gives 6.8 and we add the target device to program family: Spartan 3e, part: xc3s500e, speed: -4, package: FT256

Then we start the installation

3.5.3 Compiling thefiltering/ blurring Model for Hardware Co-Simulation

3.5.3.1 Choice of the installed platform

Once the installation is complete, the DEFB platform is now included in the list of Co-Simulation platforms



Compilation : Digital Electronics FPGA
Settings ...
Part : Spartan3E xc3s500e-4ft256
Synthesis tool : XST
Hardware description language : VHDL
Target directory : ./netlist
Browse...
Project type : Project Navigator
Synthesis strategy : XST Defaults*
Implementation strategy : ISE Defaults*
☐ Create interface document
☑ Create testbench
☐ Import as configurable subsystem
Model upgrade...
Performance Tips | Generate | OK | Apply | Cancel | Help

Figure 3.16:DEFB compilation target

When a compilation target is selected, the fields on the System Generator block dialog boxare automatically configured with settings appropriate for the selected compilation target.System Generator remembers the dialog box settings for each compilation target. Thesesettings are saved when a new target is selected, and restored when the target is recalled.

3.5.3.2 Invoking the Code Generator

The code generator is invoked by pressing the **Generate** button in the System Generatorblock dialog box.The code generator produces a FPGA configuration bitstream for thedesign that issuitable for hardware co-simulation. System Generator not only generates the HDL and
netlist files for the model during the compilation process, but it also runs the downstreamtools necessary to produce an FPGA configuration file.

The configuration bitstream contains the hardware associated with your model, and alsocontains additional interfacing logic that allows System Generator to communicate withyour design using a physical interface between the platform and the PC. This logicincludes a memory map interface over which System Generator can read and write valuesto the input and output ports on your design. It also includes any platform-specificcircuitry (e.g., DCMs, external component wiring) that is required for the target FPGAplatform to function correctly.

3.5.3.3 Hardware Co-simulation block

System Generator automatically creates a new hardware co-simulation block once it hasfinished compiling your design into an FPGA bitstream. A Simulink library is also createdin order to store the hardware co-simulation block. At this point, you can copy the blockout of the library and use it in your System Generator design as you would other Simulinkand System Generator blocks.

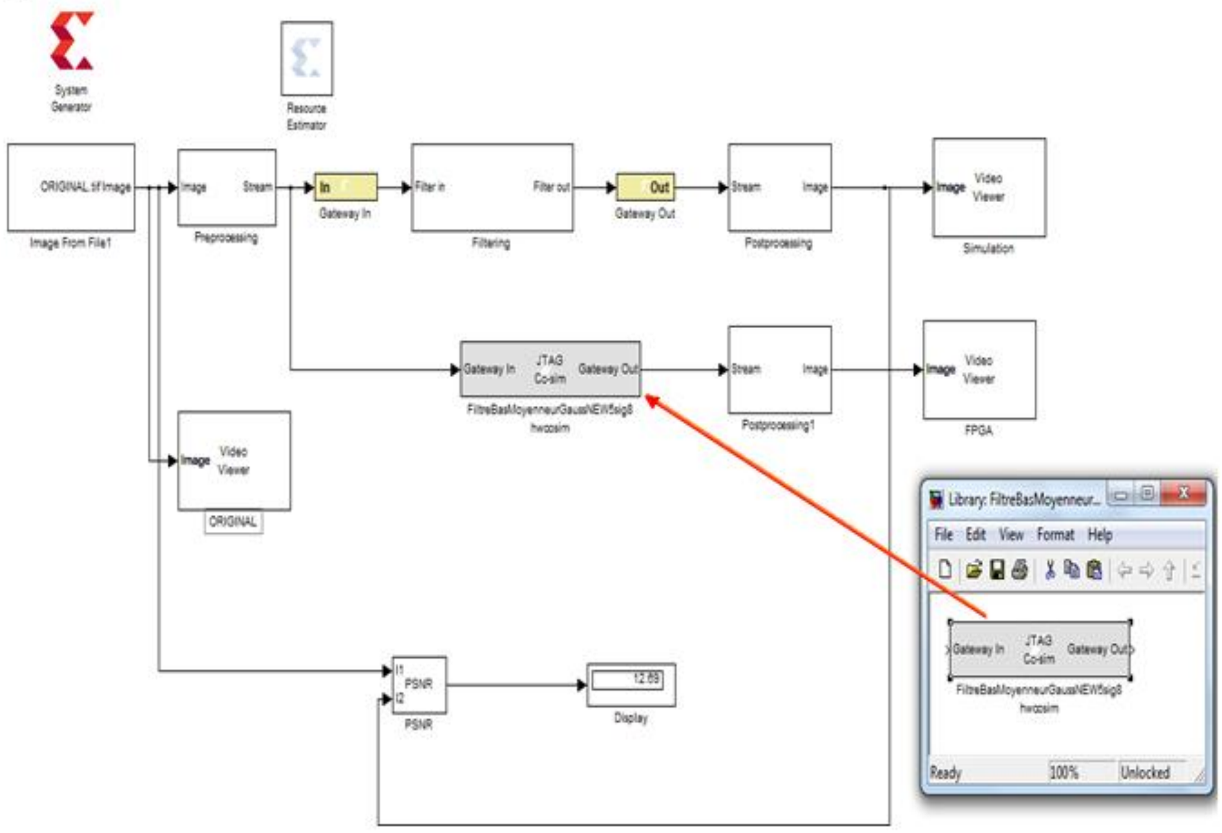


Figure 3.17:Simulation-Hardware Co-Simulation block Insertion

The hardware co-simulation block assumes the external interface of the model orsubsystem from which it is derived. The port names on the hardware co-simulation blockmatch the ports names on the original subsystem. The port types and rates also match theoriginal design.

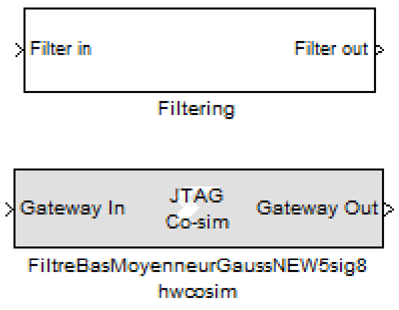


Figure 3.18: Hardware co-simulation block

Hardware co-simulation blocks are used in a Simulink design the same way other blocksare used. During simulation, a hardware co-simulation block interacts with the underlyingFPGA platform, automating tasks such as device configuration, data transfers, andclocking. A hardware co-simulation block consumes and produces the same types ofsignals that other System Generator blocks use. When a value is written to one of theblock's input ports, the block sends the corresponding data to the appropriate location inhardware. Similarly, the block retrieves data from hardware when there is an event on anoutput port.

Hardware co-simulation blocks may be driven by Xilinx fixed-point signal types, Simulinkfixed-point signal types, or Simulink doubles. Output ports assume a signal type that isappropriate for the block they drive. If an output port connects to a System Generatorblock, the output port produces a Xilinx fixed-point signal. Alternatively, the portproduces a Simulink data type when the port drives a Simulink block directly.

Like other System Generator blocks, hardware co-simulation blocks provide parameterdialog boxes that allow them to be configured with different settings. The parameters thata hardware co-simulation block provides depend on the FPGA platform the block isimplemented for (i.e., different FPGA platforms provide their own customized hardwareco-simulation blocks).

The system should appear as follows:

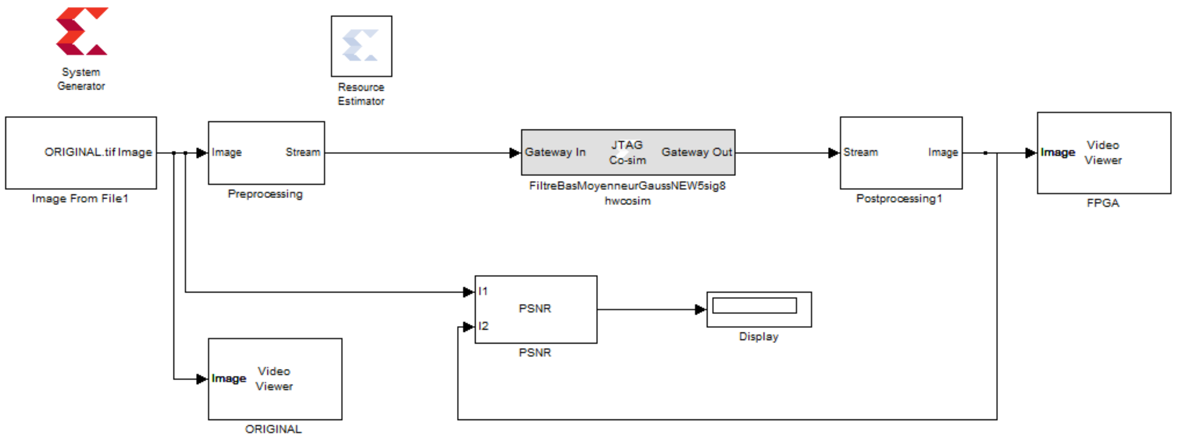


Figure 3.19: Hardware Co-Simulation model

3.5.3.4 Co-SimulationImplementation

Implementation of the co-simulation system consists of only two simple tasks. The first is to connect the FPGA hardware to the host computer, and the second is to run the co-simulation model. Begin by assembling the FPGA hardware. Set up all hardwareand connect the board to the host computer using the serial to USB cable and the JTAG programming cable. This will allow the co-simulation to access the hardware for the filtering/blurring algorithms. Once the board is connected properly, check that the simulation time is still set to the same value as in the original system design and click the Start simulation button: Upon completion of the simulation run time, this co-simulation process is complete, and the output signal can be accessed through video viewer.

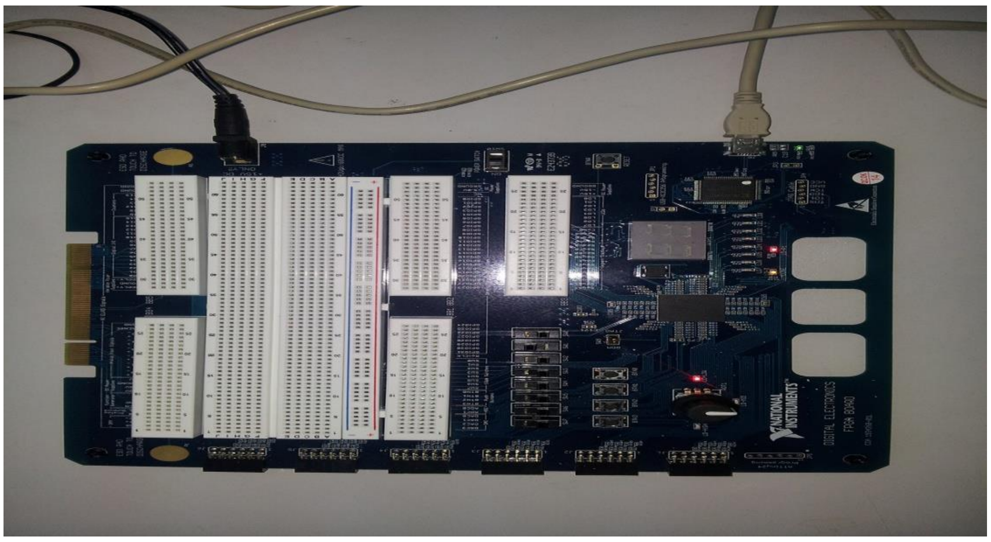


Figure 3.20: The connected Board

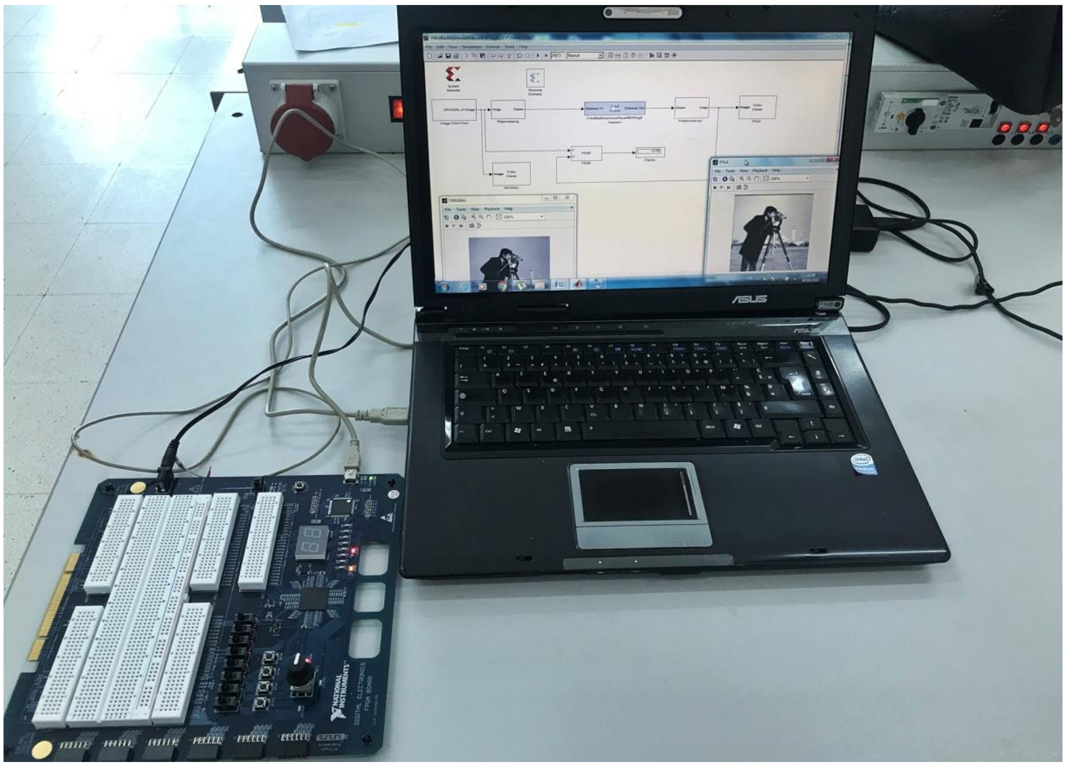
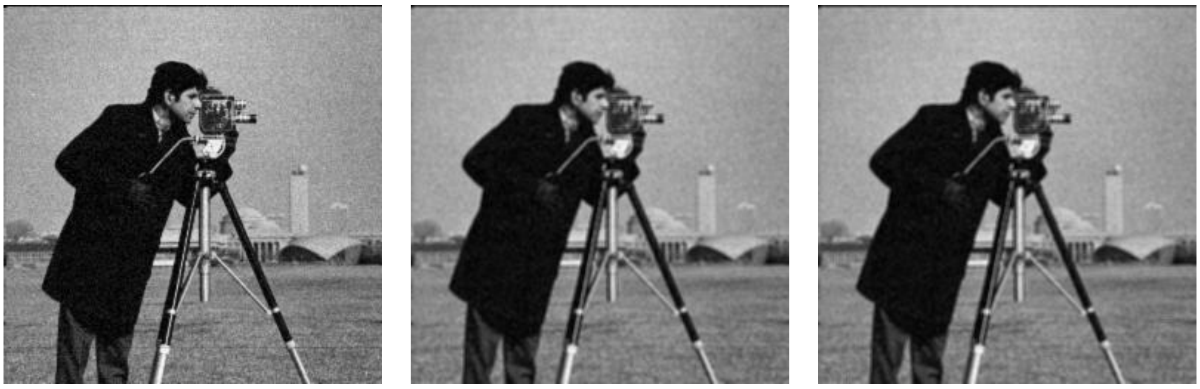


Figure 3.21: Co-simulation implementation

3.6 Hardware Co-simulation results

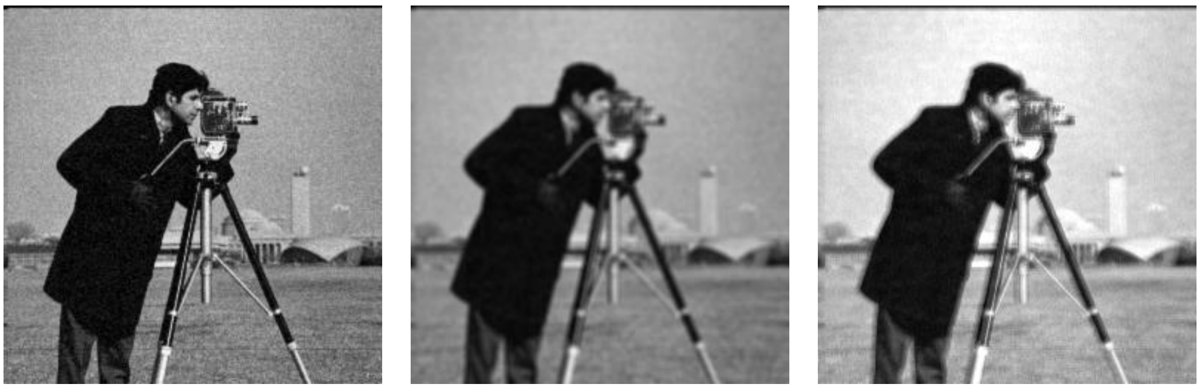
3.6.1 Image Gaussian filtering

We applied a Gaussian filter with respectively (3x3) and (5x5) kernel with σ=0.5, 2 and 8 on the image (cameraman.tiff) affected by Gaussian noise with σ=0.002.The kernel will be conserved and sigma values will be changed in order to see their impact in the filtered image.



a) PSNR: 17.16 dB b) PSNR: 18.14 dB c) PSNR: 18.17 dB

Figure 3.22:Hardware Filtered image with 3x3 kernel and a) σ=0.5 b) σ=2 c) σ=8



a) PSNR: 15.65 dB b) PSNR: 16.93 dB c) PSNR: 12.52 dB

Figure 3.23:Hardware Filtered image with 5x5 kernel and a)σ=0.5 b) σ=2 c) σ=8

For different sigma values and kernel size, Table 3.1 resumes the correspondingPSNR of filtered images.

Table 3.1: PSNR values for different Hardware filtered images

σ	PSNR (3x3)	PSNR (5x5)
0.5	17.16 dB	15.65 dB
2	18.14 dB	16.93 dB
8	18.17dB	12.52dB

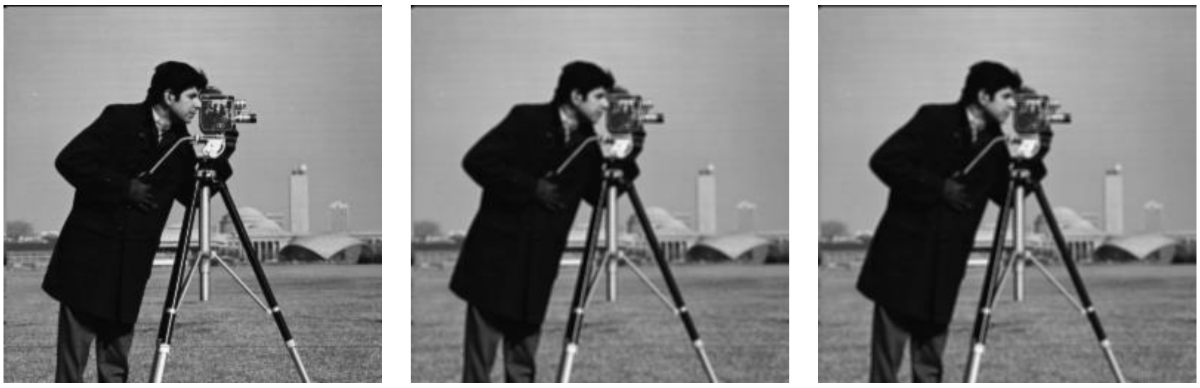
The Gaussian filtering effect depends on 2 main factors kernel matrix size and the σ (sigma). We note that when we increase the sigma value and the kernel size, the Gaussian noise is reduced, but not completely removed and the effect of smoothing increase and the PSNR decrease. So, Gaussian filtering is best suited filters for low Gaussian noise with low sigma value and kernel size. The blurring effect increase with kernel size but we find a conflict result when increasing sigma value (PSNR increase) which is not conform with MATLAB results.

3.6.2 Image blurring

We applied a blurring filter with respectively (3x3) and (5x5) kernel on the image (cameraman.tiff).

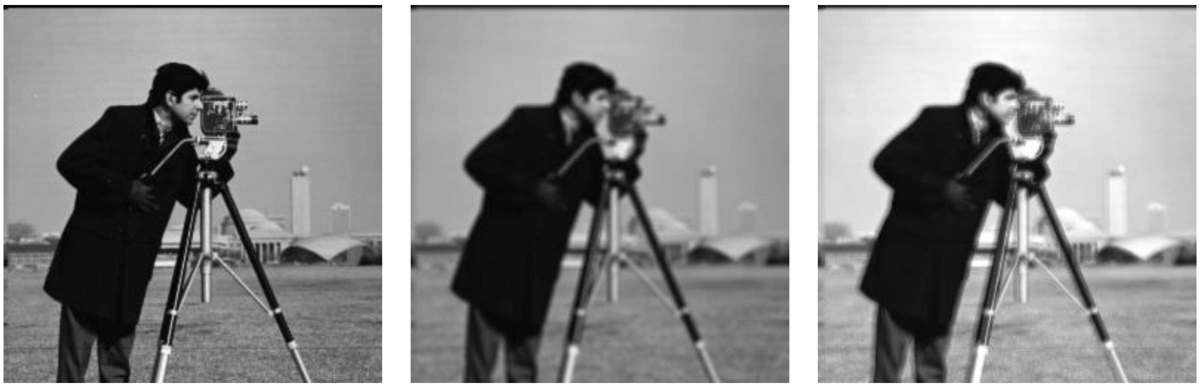
3.6.2.1 Gaussian blur

The kernel will be conserved and sigma values will be changed in order to see their impact in the blurred image.



a) PSNR: 17.8 dB b) PSNR: 18.77 dB c) PSNR: 18,81 dB

Figure 3.24:Hardware Gaussian blur with 3x3 kernel and a)σ=0.5 b) σ=2 c) σ=8



a) PSNR: 16.09 dB b) PSNR: 17.36 dB c) PSNR: 12.69dB

Figure 3.25:Hardware Gaussian blur with 5x5 kernel and *a)σ=0.5 b) σ=2 c) σ=8*

For different sigma values and kernel size, Table 3.2 resumes the correspondingPSNR of blurredimages.

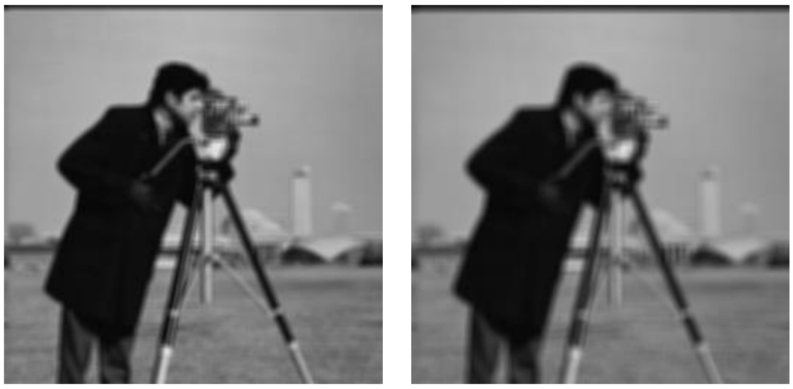
Table 3.2: PSNR values for different Gaussian blurred images

σ	PSNR (3x3)	PSNR (5x5)
0.5	: 17.8 dB	16.09 dB
2	18.77 dB	17.36 dB
8	18,81 dB	12.69dB

The Gaussian blur effect also depends on 2 main factors kernel matrix size and the σ (sigma). The blurring effect is directly proportional to kernel size. As the kernel matrix size increases, more is the blurring effect on the image. Tableindicate that the blurring is more when the kernel size increases. This will increase the blurring effect on image. Also, the other factor that affects blurring is the sigma value. PSNR values indicate that the blurring effect is less when sigma increases which is contradictory with MATLAB results.

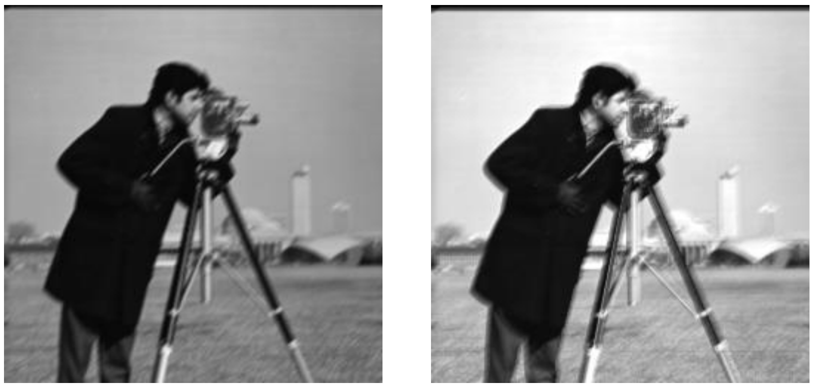
3.6.2.2 Box and Motion blur

We applied a box blur with respectively (5x5) and (7x7) kernel and motion blur with length of 5, 7 and angle of 45°.



a) PSNR:17.6 dB b) PSNR: 16.68 dB

Figure 3.26:Hardware Box blur with kernel a) 5x5, b) 7x7



a) PSNR:16.92 dB b) PSNR: 12.02 dB

Figure 3.27:Hardware Motion blur with length of 5, 7 and angle of 45°

From the images (figure 2.25, figure 2.26) and table 3.3, it is evident that the box blur effect increases with the increase in kernel matrix size while the motion blur is more when the blur length increases. The results are consistent with MATLAB results.

Table 3.3: *PSNR values for Hardware box and motion blur*

Box blur	5x5	17.6 dB	7x7	16.68 dB
Motion blur	$K = 5, \alpha = 45°$	16.92 dB	$K = 7, \alpha = 45°$	12.02 dB

3.7 FPGA HardwareResource

System Generator supplies tools that estimate the FPGA hardware resources needed to implement a design. Estimates include numbers of slices, lookup tables, flip-flops, block memories, embedded multipliers, I/O blocks and tristate buffers. These estimates make it easy to determine how design choices affect hardware requirements. To estimate the resources needed for a subsystem, drag a Resource Estimator block into the subsystem, double-click on the estimator, and press the Estimate button.

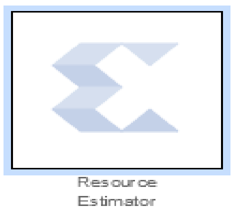


Figure 3.28:Xilinx Resource Estimator block

The results are produced using Simulink and Xilinx DSP Tools, synthesized with ISE 14.7 for the target Spartan3e device (xc3s500e) FPGA device. Table3.4 shows the resource utilization summary for image filtering and blurring results with different kernels.

Table 3.4: Resource Utilization Summary with different kernel

Resource	Kernel 3x3	Kernel 5x5	Kernel 7x7
Slices	139	389	711
Flip Flops	118	282	510
BRAMs	2	4	6
LUTs	158	480	853
IOBs	36	36	36

Above table 3.4, shows the comparison of image filtering and blurring with different kernel size. We conclude that FPGA Hardware Resource increases with the increase in kernel matrix size.

3.8 Conclusion

The Xilinx System Generator tool is a new application in image processing, and offers a friendly environment design for the processing, because the filters are designed by blocks. This tool support software simulation, but the most important is that can synthesize in FPGAs hardware, with the parallelism, robust and speed, this features are essentials in image processing.

Image filtering and blurring design is implemented in the device Spartan 3(xc3s500-4ft256) using Hardware Co-simulation. The cost-efficient hardware co-simulation is adopted to evaluate the real time performance of the desired algorithms.

Conclusion and future work

In this thesis, we focus on real-time image processing, software and hardware implementation of Gaussian filter and motion, Gaussian box blur were achieved. The effect of the kernel size onthe result images qualities has been studied and analyzed.

Convolution of a smoothing kernel with the desired noisy images produces a filtered or blurred image. The convolution is in the heart of filtering and blurring operations, so the convolution module has been developed using advanced design tool Xilinx System Generator (XSG) and the design was implemented on a reconfigurable logic platform targeting a Spartan3device (xc3s500e).The cost-efficient hardware co-simulation is adopted to evaluate the real time performance of the desired algorithms.

Objectives of this thesis were achieved Results using filtering and blurring image processing were obtained and it was noticed that the Gaussian filter has a better performance mainly as it uses the smallest kernel size, the Gaussian noise may be reduced, but not completely removed, by Gaussian filter typically yields the smoothest image, so Gaussian filter is more effective at smoothing images. By controlling the value of sigma and the size of kernel, the degree of smoothing can also be controlled. Also, larger kernel will blur the image more than a smaller kernel.

Furthermore, the results indicate that Xilinx System Generator tool offers an easy and efficient method for implementing image algorithms into FPGA. It allows the design of hardware system starting from a graphical high level Simulink environment, extends the traditional Hardware Description Language (HDL) design providing graphical modules, and thus does not require a detailed knowledge of this complex language.

The FPGA, being a reprogrammable piece of hardware, is able to perform complex calculations in a timely manner compared to that of a microprocessor due to its ability to perform multiple tasks and calculations at a single time.

Future works include the use of the Xilinx System Generator development tools for the implementation of wiener filtering which lead to better performance in removing Gaussian noise with high kernel size and also use Xilinx System Generator for removing blurring artifacts using deblurring techniques.

References

Abdelrahman, H. M. W. (2017). *Image Processing and Computer Vision: A Comparison between CPU and FPGA* (Doctoral dissertation, University of Khartoum).

Arunmozhi, R., & Mohan, G. (2012). Implementation of digital image morphological algorithm on FPGA using hardware description languages. *International Journal of Computer Applications*, *57*(5).

Bailey, D. G. (2019). Image processing using FPGAs.

Behrenbruch, C. P., Petroudi, S., Bond, S., Declerck, J. D., Leong, F. J., & Brady, J. M. (2004). Image filtering techniques for medical image post-processing: an overview. The British journal of radiology, 77(suppl_2), S126-S132.

Chandel, R., & Gupta, G. (2013). Image filtering algorithms and techniques: A review. International Journal of Advanced Research in Computer Science and Software Engineering, 3(10).

Chu, P. P. (2011). FPGA prototyping by VHDL examples: Xilinx Spartan-3 version. John Wiley & Sons.

Deshpande, R.G., Ragha, L.L. and Sharma, S.K. (2018) Video Quality Assessment through PSNR Estimation for Different Compression Standards. Indonesian Journal of Electrical Engineering and Computer Science, 11, 918-924.

Elamaran, V., Praveen, A., Reddy, M. S., Aditya, L. V., &Suman, K. (2012). FPGA implementation of spatial image filters using Xilinx system generator. Procedia Engineering, 38, 2244-2249.

Eswar, S. (2015). Noise reduction and image smoothing using gaussian blur (Doctoral dissertation, California State University, Northridge).

Gonzalez, R. C., & Woods, R. E. (2002). Digital image processing.

Gruwell, A., Zabriskie, P., &Wirthlin, M. (2016, September). High-speed FPGA configuration and testing through JTAG. In 2016 IEEE AUTOTESTCON (pp. 1-8). IEEE.

Kalalembang, E., Usman, K., & Gunawan, I. P. (2009, November). DCT-based local motion blur detection. In International Conference on Instrumentation, Communication, Information Technology, and Biomedical Engineering 2009 (pp. 1-6). IEEE.

Md.Naseem Ashraf, (2013) Image degradation and noiseMehra, R., & Devi, S. (2010). Efficient Hardware Co-Simulation Of Down Convertor for Wireless Communication Systems. International journal of VLSI design & Communication Systems (VLSICS), 1(2), 13-21.

Nasri .S Design and Implementation of FPGA-Based Systems, 2009

NI 2009 NI Digital Electronics FPGA Board User Manual Circuit Development Platform

Patel, N., Shah, A., Mistry, M., &Dangarwala, K. (2014). A study of digital image filtering techniques in spatial image processing. In Proceedings of the 2014 International Conference on Convergence of Technology (I2CT) (pp. 1-6).

Rao, R. S., & Nakkeeran, R. (2015). High level abstraction method for implementing Image Processing Techniques on FPGA. In International Conference on Knowledge Collaboration in Engineering March (pp. 27-28).

Robert K. Dueck. Digital design with CPLD applications and VHDL, 2005.

R. S. Boyer, and J. Moore, a ComputationalLogic Handbook, Academic Press, Boston, 1988.

Saidani, T., Dia, D., Elhamzi, W., Atri, M., & Tourki, R. (2009, July). Hardware co-simulation for video processing using xilinx system generator. In Proceedings of the World Congress on Engineering (Vol. 1, pp. 3-7).

Saxena, C., & Kourav, D. (2014). Noises and image denoising techniques: a brief survey. International journal of Emerging Technology and advanced Engineering, 4(3), 878-885.

Søgaard, J., Krasula, L., Shahid, M., Temel, D., Brunnstrom, K. and Razaak, M. (2016) Applicability of Existing Objective Metrics of Perceptual Quality for Adaptive Video Streaming. Society for Imaging Science and Technology IS&T International Symposium on Electronic Imaging.

Swaraj, D., & Madhumati, G. L. (2014). FPGA Implementation of SIFT Algorithm Using Xilinx System Generator. *Int. J. Emerging Trends in Electrical and Electronics*, *10*(10), 80-85.

Verma, R., & Ali, J. (2013). A comparative study of various types of image noise and efficient noise removal techniques. International Journal of advanced research in computer science and software engineering, 3(10).

Vernon, D. (1991). Machine vision-Automated visual inspection and robot vision. NASA STI/Recon Technical Report A, 92, 40499.

Warkari, K. D. S., &Kshirsagar, U. A. (2015). Design of Point Processing Algorithm using Hardware Co-simulation for Digital Image Processing Application. International Journal of Electronics, Communication and Soft Computing Science & Engineering (IJECSCSE), 4, 70.

Xilinx 2018, " Spartan-3E FPGA Family Data Sheet".