FPGA IMPLEMENTATION OF CIRCULAR SPATIAL FILTER UNDER HIGH NOISE VARIANCE CONDITIONS

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology
In
VLSI and Embedded systems
By

Rajulapati Bharat kumar
210EC2064

Under the Guidance of
Prof. SukadevMeher

Department of Electronics and Communication Engineering

National Institute Of Technology

Rourkela

2011-2012

FPGA IMPLEMENTATION OF CIRCULAR SPATIAL FILTER UNDER HIGH NOISE VARIANCE CONDITIONS

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology
In
VLSI and Embedded systems
By

Rajulapati Bharat kumar
210EC2064

Under the Guidance of
Prof. Sukadev Meher

Department of Electronics and Communication Engineering

National Institute Of Technology

Rourkela

NATIONAL INSTITUTE OF TECHNOLOGY
ROURKELA

CERTIFICATE

This is to certify that the thesis titled **"FPGA IMPLEMENTATION OF CIRCULAR SPATIAL FILTER UNDER HIGH NOISE VARIANCE CONDITIONS"** submitted by **Mr. RAJULAPATI BHARAT KUMAR** in partial fulfillment of the requirements for the award of Master of Technology degree in **Electronics & Communication Engineering** with specialization in "**VLSI and embedded systems** during session 2011-2012 at **National Institute Of Technology, Rourkela** (Deemed University) is an authentic work by him under my supervision and guidance.

Prof. SUKADEV MEHER
Head of the Dept.
Dept. of Electronics and Communication Engg.
National Institute of Technology.
Rourkela-769008.

Acknowledgement

I would like to express my gratitude to my thesis guide **Prof. S.Meher** for his guidance, advice and constant support throughout my thesis work. I would like to thank him for being my advisor here at National Institute of Technology, Rourkela.

Next, I want to express my respects to **Prof. K.K. Mahapatra, Prof D.P.Acharya Prof. A. K. Swain, Prof S.K.Patra,**for teaching me and also helping me how to learn. They have been great sources of inspiration to me and I thank them from the bottom of my heart.

I would like to thank all faculty members and staff of the Department of Electronics and Communication Engineering, N.I.T. Rourkela for their generous help in various ways for the completion of this thesis.

I would like to thank all my friends and especially my classmates for all the thoughtful and mind stimulating discussions we had, which prompted us to think beyond the obvious. I've enjoyed their companionship so much during my stay at NIT, Rourkela.

I am especially indebted to my parents for their love, sacrifice, and support. They are my first teachers after I came to this world and have set great examples for me about how to live, study, and work.

Rajulapati Bharat Kumar

Roll No: 210EC2064

Dept. of ECE

NIT Rourkela

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The noise in digital images is additive in nature in various cases. Such kind of noise is called to as Additive White Gaussian Noise (AWGN). This noise gets into image while transmission, reception, storage and retrieval It is difficult to suppress AWGN because it corrupts more or less all the pixels in a image. Some filters such mean filter had been proposed to suppress AWGN but in most cases it incorparates a *blurring* effect in the image. Image denoising is usually done before display or further processing like feature extraction, segmentation, object identification, texture analysis, etc. The intention of denoising is to suppress the noise efficiently and retaining the edges and other necessary features as far as possible.

Many efficient digital image filters are found that perform well under low noise conditions. But in the cases of moderate and high noise conditions their performance is limited. Thus, it is felt that there is sufficient scope to investigate and develop quite efficient. And proposed a spatial filter named as circular spatial filter which performs well under high noise conditions.

Suppose CSF has to be used for real time applications such as before displaying the video on HDTV a real time application. It is hard to implement this algorithm on a general purpose computer where high amount of concurrency is needed. So we have chosen FPGA as a target which is suitable for video and image processing. Here we chose virtex-5 Xilinx board to implement the algorithm. The performance of the designed filters is compared with the existing filters and the MATLAB simulation [1] in terms of peak-signal-to noise ratio, root-mean-squared error.

CHAPTER-1

INTRODUCTION

Image processing has got wide varieties of applications in computer vision, multimedia communication, television broadcasting, etc. That demand very good quality of images. The quality of an image degrades due to introduction of additive white Gaussian noise (AWGN) during acquisition, transmission/ reception and storage/ retrieval processes. It is very much essential to suppress the noise in an image and to preserve the edges and fine details as far as possible.

In recent times, Field Programmable Gate Array (FPGA) technology has turn out to be a feasible target for the implementation of algorithms apt for video image processing applications. The distinctive architecture of the FPGA has permitted the technology to be used in numerous such applications encircling all areas of video image processing.

Since image sizes and bit depths raise better, software has turn out to be fewer useful in the video processing dominion. Real-time applications that are the target of this project are requisite for the high speeds desired in processing video applications. In toting up, a frequent quandary is dealing with the hefty amounts of data captured by means of satellites and ground-based recognition systems. DIVP systems are being engaged to selectively diminish the quantity of data to process, ensuring that only pertinent data is conceded on to a engineer analyst. sooner or later, it is predictable that most video processing will be replaced with DIVP systems, with little human intervention. This is perceptibly beneficial, since human data analysts are luxurious and perhaps not exclusively precise.

In the present research work, efforts are made to develop efficient spatial-domain image filters that suppress noise quite effectively of FPGA Xilinx virtex-5 device.

1.1 INTRODUCTION NOISE IN DIGITAL FILTERS:

In this section, various types of noise corrupting an image signal are studied; the sources of noise are discussed, and mathematical models for the different types of noise are presented.

1.1.1 SOURCES OF NOISE

During acquisition, transmission, storage and retrieval processes an image signal gets contaminated with noise. **Acquisition noise** is usually *additive white Gaussian noise* (AWGN) with extremely stumpy variance. In many engineering applications, the acquisition noise is quite negligible. It is mainly due to very high quality sensors. In some applications like remote sensing, biomedical instrumentation, etc., the acquisition noise may be high enough. But in such a system, it is basically due to the fact that the image acquisition system itself comprises of a transmission channel. So if such noise problems are considered as transmission noise, then it may be concluded that acquisition noise is negligible. The acquisition noise is considered negligible due to another verity that the human perception system can't recognize a large dynamic range of image. That is why, an image is usually quantized at 256 levels. Thus, each pixel is represented by 8 bits (1 byte). The present-day technology offers very high quality sensors that don't have noise level greater than half of the resolution of the analog-to-digital converter (ADC), i.e., noise magnitude in time domain,

where *n(t)* is the noise amplitude at any arbitrary instant of time *t*, and *V* is the maximum output of the sensor and is also equal to the maximum allowed input voltage level for the ADC. That is, for *V = 3.3 volts*, the noise amplitude must be a lesser amount of than ~ 6.5 mV. In many practical applications, the acquisition noise level is much below this margin. Thus, the acquisition noise need not be considered. Hence, the researchers are mostly concerned with the noise in a transmission system. Usually, the transmission channel is linear, but dispersive due to a limited

1.1.2 THE MATHEMATICAL REPRESENTATION OF NOISE:

The AWGN is mathematically represented below. The Gaussian noise is given by,

$$\eta_{AWGN}(t) = \eta_G(t)$$

$$f_{AWGN} = f(x,y) + \eta_G(x,y)$$

Where h(*t*) is a random variable that has a Gaussian probability distribution. It is an additive noise that is characterized by its variance, where, s represents its standard deviation. In (1.5),

the noisy image is calculated as a sum of the original uncorrupted image and the Gaussian distributed random noise When the variance of the random noise *G* h is very low, *Gh*(*x*, *y*) is zero or very closes to zero at many pixel locations. Under such circumstances, the noisy image *AWGN f* is same or very close to the original image *f* (*x*, *y*) at many pixel locations (*x*, *y*).

The proposed filter developed in subsequent chapters is meant for suppression of AWGN. To avoid ambiguity, the noisy image is taken as *g*(*x*, *y*) in the subsequent chapters. Thus, the noisy image is expressed as

$$G(x, y) = f(x, y) + \eta\ (x, y) \quad .$$

1.2PLATFORMS USED FOR THE DESIGN

There are several different choices a designer has while implementing a DIVP system of any class. Hardware, certainly, provides much greater speed while preffering to software implementation, but one must mull over the enhancement in development time intrinsic in creating a hardware design. Most software designers are well-known with C, but with the purpose to develop a hardware system, one must learn a hardware design language such as Verilog or VHDL, or use a software-to-hardware conversion method, like MATCH which converts MATLAB image processing code to VHDL Streams-C converts the C code written in image to VHDL. While such conversion methods are admirable, they are at present in development and indisputably not appropriate to high speed applications of video or image processing.

1.2.1 FPGA

Field Programmable Gate Arrays (FPGAs) correspond to reconfigurable technology , which is in most ways supremely apt for video processing application. Reconfigurable devices are processors that can be programmed with a the desired design specification, and then can be

reprogrammed or reconfigured with almost infinite designs as the designer's needs alter. FPGAs in general consists of a system of logic blocks typically look up tables and flip-flops and a few amounts of Random Access Memory (RAM), all hardwired collectively using a enormous collection of interconnects. All of the design logic in an FPGA can be rewired reprogrammed, with a diverse designs as frequently as the designer desired. This sort of architecture allows a huge assortment of logic designs reliant on the processor's possessions, which can be interchanged for a new-fangled design as rapidly as the device can be reprogrammed.

In the present day, FPGAs can be urbanized to implement parallel design style, which is not achievable in dedicated DSP designs. ASIC design methodology can be applied for FPGA designs, prompting the designer to implement designs at gate level. However, frequently engineers use a hardware language such as VHDL or Verilog, which fovours for a design tactic comparable to software design. This software view of hardware design fovours for lowering the on the whole support cost and design abstraction. The algorithms presented in this thesis were written for virtex-5 FPGA architecture.

1.2.2 VHDL

In recent years, VHSIC (Very High Speed Integrated Circuit) Hardware Design Language (VHDL) has become a sort of industry standard for high-level hardware design. Since it is an open IEEE standard, it is supported by a large variety of design tools and is quite interchangeable (when used generically) between different vendors' tools. It also supports inclusion of technology-specific modules for most efficient synthesis to FPGAs.
The first version of VHDL, IEEE 1076-87, appeared in 1987 and has since undergone an update in 1993, appropriately titled IEEE 1076-93.

1.2.3 MATLAB

MATLAB means matrix laboratory, it is almost used in every branch of engineering. Usually in MATLAB software data is stored in the form of matrix and is processed according to our wish. It provides support to our designs with many predefined functions for every branch of engineering. Here in our project we have used it to make files that are readable by block ROM with extension of .coe and to find the metrics of image such as peak to signal noise ratio and root means square error and to plot the compared results.

1.2 DESIGN FOLW

Here in this section we present the flow of the design. This is explained with the help of the figure shown below. The design flow for this project is represented in Figure 1. This shows the the VHDL design environment on the FPGA-specific tools. First the design is created in VHDL, then the code's syntax is checked and the design is synthesized into a library. The design is next simulated with the help of test bench to check its functionality. Observing the output waveforms in the VHDL simulator allows the designer to determine proper functionality of the design. Next, the design is processed with vendor-specific place-and-route tools and mapped onto a target specific FPGA in software. This allows the engineer to view a floor plan and hierarchical view of the design, which can help verifying a proper mapping procedure. Next, the design is verified for proper functionality once again. This step is important because it assures that the design is correct in its translation from VHDL to gate level. If this is found to be correct, the design can then be programmed onto the specified FPGA

The software version of the image metrics are created in MATLAB and the processed image are cross checked with that of the results [1].

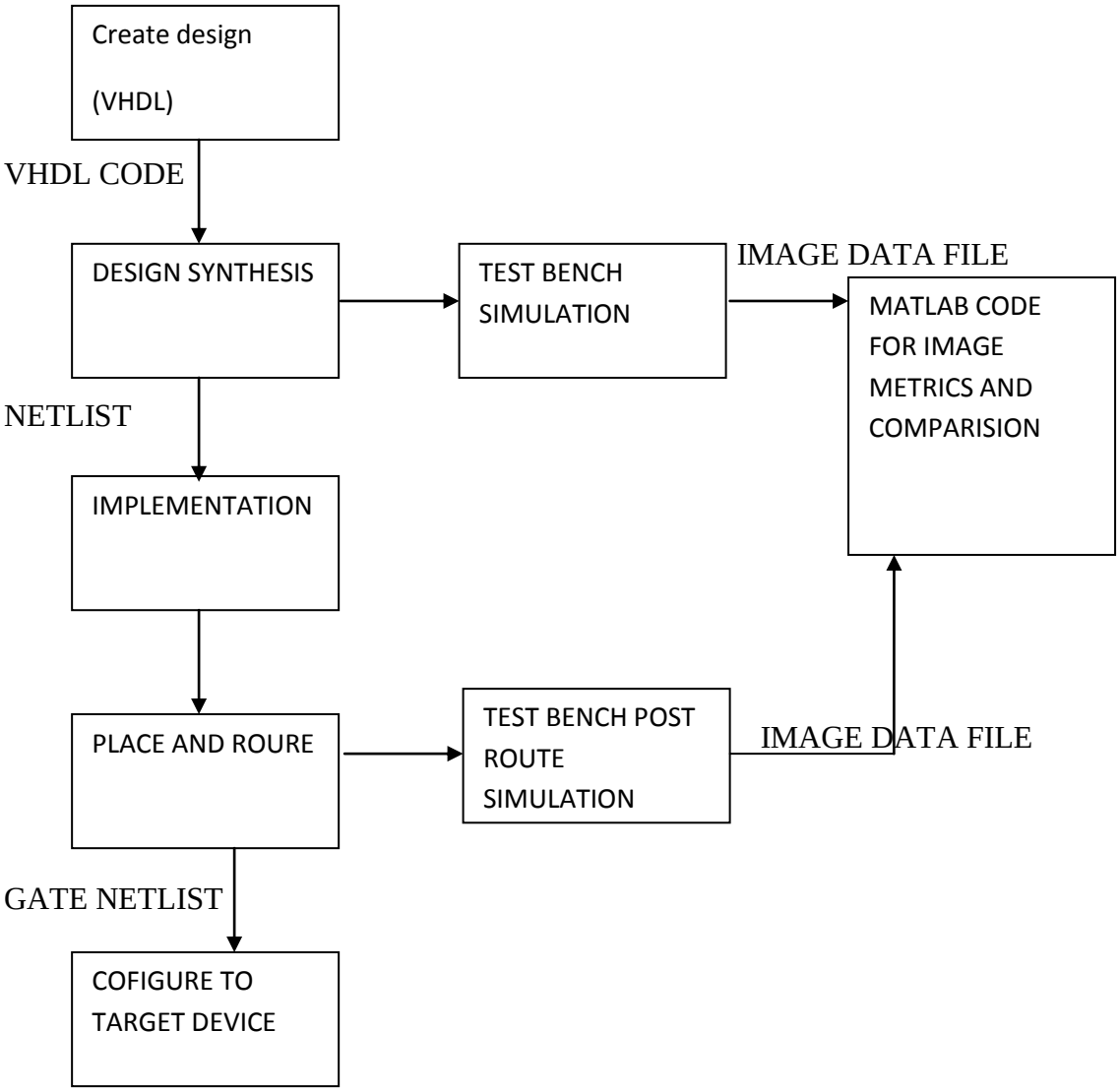

Figure1.3.1: Design flow

1.4 Aim of the project:

The foremost aim of the project is to implement circular spatial filter on FPGA under high noise variance conditions. Here the device used is virtex-5and language used is VHDL.

1.5 Thesis Layout:

CHAPTER-1 This chapter explains the Introduction of implementation of CSF on FPGA.

CHAPTER-2 Explains about the fundamentals of Image denoising filters.

CHAPTER-3 This chapter explains the fundamentals of FPGA and VHDL.

CHAPTER-4 This chapter deals with the implementation of proposed filter.

CHAPTER-5 This deals with results.

CHAPTER-6 Contains Conclusion and Future work

CHAPTER-2

STUDY OF IMAGE DENOISING FILTERS

Image denoising is a frequent course of action in digital image processing for the suppression of additive white Gaussian noise (AWGN) that might have corrupted an image during its acquisition or transmission. This method is conventionally performed in the spatial-domain or transform-domain by filtering. In spatial-domain filtering, this action is performed on image pixels directly. The main idea behind the spatial-domain filtering is to convolve a mask with the whole image. The mask is a small sub-image of any arbitrary size (e.g., 3×3, 5×5, 7×7, etc.). Other common names for mask are: window, template and kernel.

2.1 FUNDAMENTAL STEPS IN DIGITAL IMAGE PROCESSING:

The basic components in digital image processing are shown in fig 2.2.1. The first step in processing is image acquisition i.e., to get the digital image required for processing. It involves the conversion of a scene into a digital representation by sensors like microdensitometer, image dissectors, vidicon cameras and photosensitive solid-state arrays.

After digital image is obtained, the next step deals with pre-processing the image. The absolute function of pre-processing is to progress the image in areas that increases the chances for the success of the other process by which the techniques for enhancing contrast, removing noise and isolating regions. Segmentation, dividing an input image into its ingredient objects. The raw pixel data, the output of segmentation is transformed into a form suitable for computer processing and processing is done by representation block. Description, also called feature solution deals with extracting features that are is basic for differentiating one class of objects from another. Recognition is a process that assigns a label to an object based on the information provided by its recognized objects. Finally, the knowledge base controls the interaction roping and assigning pattern vectors into different pattern classes. The methods for this type of recognition are minimum distance classifiers, correlates etc.

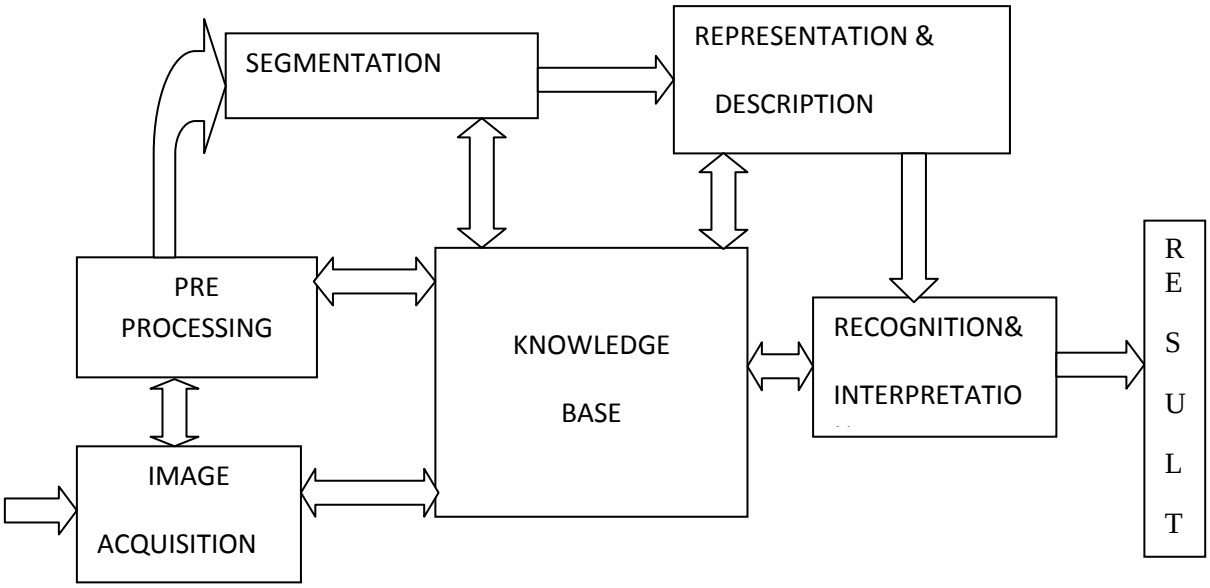

Fig.2.1.1 Fundamental steps in digital image processing

Decision theoretic approaches to recognition are based on the use of decision (or discriminate) functions. Let $X=(X_1, X_2, X_3, X_4 \ldots X_n)^T$ represent an n-dimensional pattern vector, for M pattern classesW1, W2, W3…Wm the basic problem in decision theoretic pattern recognition is to find m dimensions function $d_1(x)$, $d_2(x)$,..$D_m(x)$ with the property that, if pattern x belongs to class W_i, then

$d_i(x) > d_j(x$ …….. (2.2.1)

In other words, an unknown pattern X belongs to the i^{th} pattern class if, upon substitution of X into all decision functions, $d_i(x)$ yields the largest numerical value. Ties are found out haphazardly.

The decision boundary separating class W_i from W_j is given by valves of x for which $d_i(x) = d_j(x)$ or equivalent, by valves of x for which $d_i(x) = d_j(x) = 0$

Common practice is to identify the decision boundary between two classes by the single function $d_{ij}(x)= d_i(x)-d_j(x)$=zero. Thus $d_{ij}(x) > 0$ for patterns of class W_i and $d_{ij}(x) < 0$ for pattern of class W_j

2.2 IMAGE METRICS

The quality of an image is examined by objective evaluation as well as subjective evaluation. For subjective evaluation, the image has to be pragmatic by a human professional. The human visual system (HVS) [109] is so knotty that it is not yet modeled appropriately. Therefore, in addition to objective evaluation, the image must be pragmatic by a human expert to arbitrator its quality. There are various metrics used for objective evaluation of an image. Some of them are mean squared error (MSE), root mean squared error (RMSE), mean absolute error (MAE) and peak signal to noise ratio (PSNR) .

Let the original noise-free image, noisy image, and the filtered image be represented by $f(x, y)$, $g(x, y)$, and $\hat{f}(x, y)$, respectively. Here, x and y represent the distinct spatial coordinates of the images in digital domain. Let the images be of size $M \times N$ pixels, i.e. $x=1,2,3,\ldots,M$, and $y=1,2,3,\ldots,N$. Then, MSE and RMSE are defined as:

The PSNR is prsented in logarithmic scale, in dB. It is a ratio of peak signal power to noise power. Since the it represent the noise power and the peak signal power is unity in case of normalized image signal, the image metric PSNR is defined as:

2.3 DENOISING FILTERS

Usually, sliding window technique is employed to perform pixel-by-pixel operation in a filtering algorithm. The local statistics obtained from the neighborhood of the center pixel give a lot of information about its expected value. If the neighborhood data are ordered (sorted), then ordered statistical information is obtained. If this order statistics vector is applied to a finite impulse response (FIR) filter, then the overall scheme becomes an order statistics (OS) filter.

For example, if a 3×3 window is used for spatial sampling, then 9 pixel data are available at a time. First of all, the 2-D data is converted to a 1-D data, i.e. a vector. Let this vector of 9 data be sorted. Then, if the mid value (5th position pixel value in the sorted vector of length = 9) is taken, it becomes **median** filtering with the filter weight vector [0 0 0 0 1 0 0 0 0]. If all the order statistics are given equal weight age, then it becomes a moving average or mean

filter (MF). Strictly speaking, the MF is a linear filter and it has nothing to do with the ordered statistics. Since the MF operation gives equal emphasis to each input data, it is immaterial whether the input vector is sorted or not. Thus, simply to have a generalization of OS filters, the MF is considered a member of this class. Otherwise, it is quite different from all other members of this family of filters

2.3.1 MEAN FILTER

The moving average or mean filter (MF) is a linear filter . All the input data are summed together and then the sum is divided with the number of data. It is very simple to implement in hardware and software. The computational intricacy is very squat. It works fine for low power AWGN. As the noise power increases, its filtering performance degrades. If the noise power is high, then a larger window should be employed for spatial sampling to have better local statistical information. As the window size increases, MF produces a reasonably high blurring effect and thus thin edges and fine details in an image are lost.

2.3.2 RANK ORDER FILTER

The rank order filter is median (MED) filter, on the other hand, is a nonlinear filter. The median is a very simple operation. The taxonomy (ordering) process is completed on the input vector, the job is done as the mid-value is taken as the output. Of course, if the length of the input vector is even, then the average of two mid-ordered statistical data is taken as output. Usually, such a computation is not required in most of image processing applications as the window length is normally an odd number. Thus, the MED operation can be completed in a very short time. That is, a MED filter may be used for online and real-time applications to suppress noise. If an image is corrupted with a extremely squat variance AWGN, then this filter can perform a good filtering operation.

2.4 CIRCULAR SPATIAL FILTER

Mean and Wiener filters suppress additive white Gaussian noise (AWGN) from an image very effectively under low and moderate noise conditions. But, these filters distort and blur the edges unnecessarily. Lee filter and non-local means (NL Means) filter work well under very low noise condition. The method noise [88] for these filters is low as compared to other

spatial-domain filters. The computational complexity of simple mean filter is low whereas that of NL-Means filter is very high. Mean, Wiener, Lee and NL-means filters are incapable of suppressing the Gaussian noise quite efficiently under high noise conditions.

Therefore, efficient spatial-domain filters should be designed with the following ideal characteristics.

i) Suppressing Gaussian noise very well under low, moderate and high noise conditions without distorting the edges and intricate details of an image;

ii) Having low method noise; and

iii) Having less computational complexity.

A novel circular spatial filter (CSF) is proposed [P2] for suppressing additive white Gaussian noise (AWGN). In this method, a circular spatial-domain window, whose weights are derived from two independent functions: (i) spatial distance and (ii) gray level distance is employed for filtering.. The filter is proficient of retaining the edges and intricate details of the image.

2.4. 1THE CIRCULAR SPATIAL FILTERING METHOD

In circular spatial filter (CSF), the name *circular* refers to the shape of the filtering kernel or window being circular. In this method, the filtering kernel consists of *distance kernel* and *gray level kernel*. The circular shaped kernel is moved invariably throughout the image to remove the noise. The proposed filter has got some resemblance with Bilateral filter where the filtering kernel is a combination of *domain-filtering kernel* and *range-filtering kernel*. The weighting function used in gray level kernel of circular spatial filter is similar to the weighting function used in *range-filtering kernel*. But the weighting function used in *distance kernel* of CSF and *domain-filtering kernel* of bilateral filter is different. The weighting function used in *domain-filtering kernel* is exponential whereas it is a simple nonlinear function in case of *distance kernel* of the proposed method.

Let the original image f be corrupted with AWGN . Then the corrupted image g may be expressed as:

$$g(x, y) = f(x, y) + \eta(x, y) \quad \text{..... (1)}$$

2.4.2 DISTANCE KERNEL

In an image, the spatial distance between any arbitrary pixel in a particular window at location (x1,y1) and the center pixel at location (x,y) is calculated as

Now the distance kernel is defined by

$$d_s = \sqrt{(x_1 - x_2)^2 - (y_1 - y_2)^2} \quad \text{.....2}$$

$$w_d = 1 - \frac{d_s}{d_{max}} \quad \text{.....3}$$

Where *dmax* is the maximum radial distance from center. The correlation between pixels goes on decreasing as the distance increases. Hence, when *wd* becomes very small the correlation can be taken as zero. When the small values of distance kernel are replaced by zero we get a circular shaped filtering kernel. The circular shaped kernel is denoted as *wcd* .

2.4.3 GREY LEVEL KERNEL

The gray level distance between any arbitrary pixel g(x1,y1) of a particular window at location (x1,y1) and the center pixel g(x, y) at location (x, y) is calculated as

$$d_g = \sqrt{g^2(x_1, y_1) - g^2(x_2 - y_2)} \quad \text{.....4}$$

The gray level distance *dg* can be used to find the gray level kernel which is defined

$$w_g = e^{(d_g^2 / 2\sigma_g^2)} \quad \text{.....5}$$

where, σ is the standard deviation of the distribution function *wg*.

The filtering kernel of CSF can be prepared from w_{cd} and w_g as:

$$W = W_{CD} W_G \quad \text{.....6}$$

The filtering kernel *w* is slide throughout the image corrupted with noise to get the estimated output. The estimated pixel can be expressed as:

$$f'(x, y) = \frac{\sum_{s=-a}^{a} \sum_{t=-b}^{t} w(s,t) g(x+s, y+t)}{\sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t)} \quad \text{.....7}$$

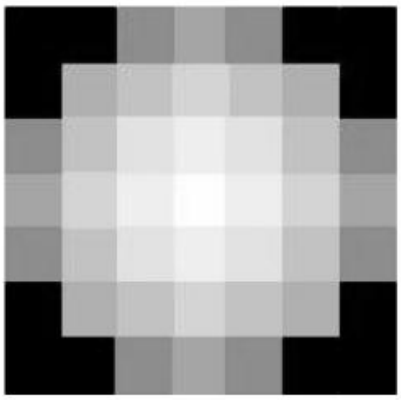

Figure2.4.1 circular spatial filter (7x7)

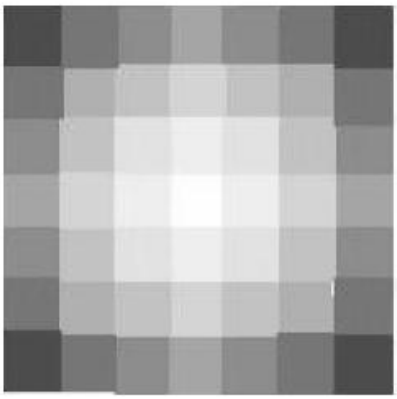

Figure2.4.2square shape filter (7x7)

In the filtering window, the center coefficient is given the highest weight. The weight goes on decreasing as distance increases from center and it is zero when correlation is insignificant. A pictorial representation of circular spatial filtering mask is shown in Figure 2.4.1 A more general filtering mask, i.e. a square mask, is also depicted in Figure 3.1 (b) to illustrate the difference. It is evident from Figure 3.1 (a) that there are necessarily some zeros in the circular spatial filter mask whereas it is not so in the case of a general square window shown in Figure 2.4.2 .

CHAPTER-3

DESIGNING ON FPGA

This chapter explains about FPGA's and designing on them and why they are preferred over other hardware devises such as DSP and microcontrollers. The type of language used to in the design, the packages, libraries, also the special functions designed by us for the project . The special features used in the design and verification of the algorithm.

3.1 BASICS OF FPGA's

Field Programmable Gate Arrays (FPGAs) are programmable digital logic chips that have reconfigurable computing technology which is suited for image processing. Reconfigurable means which can be programmed with a design and reprogrammed as desired. A computer to describe a logic function we can draw a schematic, write code file using a HDL(hardware description language) describing the function then we compile the logic function on a computer, using a software provided by the FPGA vendor. That generates a binary file that can be dumped into the FPGA by connecting a cable from your computer to the FPGA, and dumped the binary file to the FPGA and it behaves according to logic function described.

We can download FPGAs as many time as we need there is no limit as desired.. If there is a mistake in your design, just fix the logic function, recompile and download it. The designs run faster because it runs inside the hardware of FPGA, on its silicon die. FPGA loses its functionality when the power goes away (like RAM in a computer). You have to re-configure them when power is off back up to restore the functionality.

FPGAs generally consist of a system of configurable logic blocks (usually lookup tables and flip-flops) and some amount of Random Access Memory (RAM), all connected together using a vast array of interconnects. The total logic in an FPGA can be reconfigured, with a different design. This type of architecture allows a variety of logic designs to be designed

3.1.1 LOGIC CELLS

The basic unit of FPGA is a "logic-cell", duplicated to thousands time. A logic-cell constitutes a small lookup table ("LUT"), a D-flipflop and a 2-to-1 mux generally to bypass the flipflop.

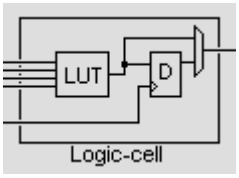

Figure 3.1.1 basic unit of FPGA

The LUT reminds of RAM that can implement any logic function. It has typically shown in the figure3.1.1.

3.1.2 Interconnect

Each logic-cell can be linked to new logic-cells through interconnect resource wires or multiplexers placed around the logic-cells. Each cell implements minute part of total design with all of them connected together, complex design can be created. This is shown in the figure below

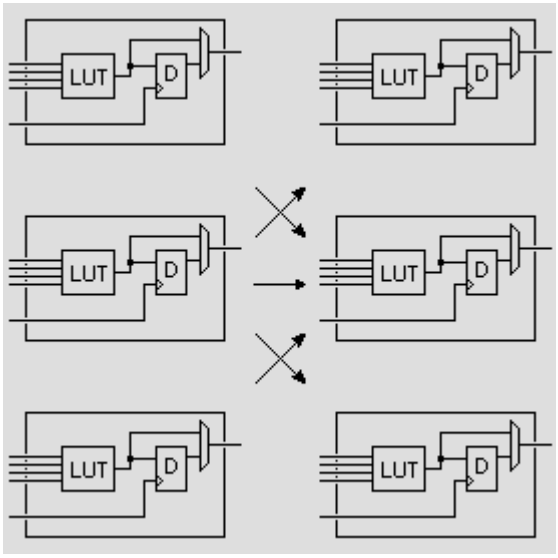

Figure3.1.2 interconnects between logic cells

3.1.3 IO-cells

The interconnects reach the boundary of the device where I/O cells are implemented and connected to the input/output pins of the FPGAs. Generally I/O cell consists a mux and a filpflop as shown in the figure.

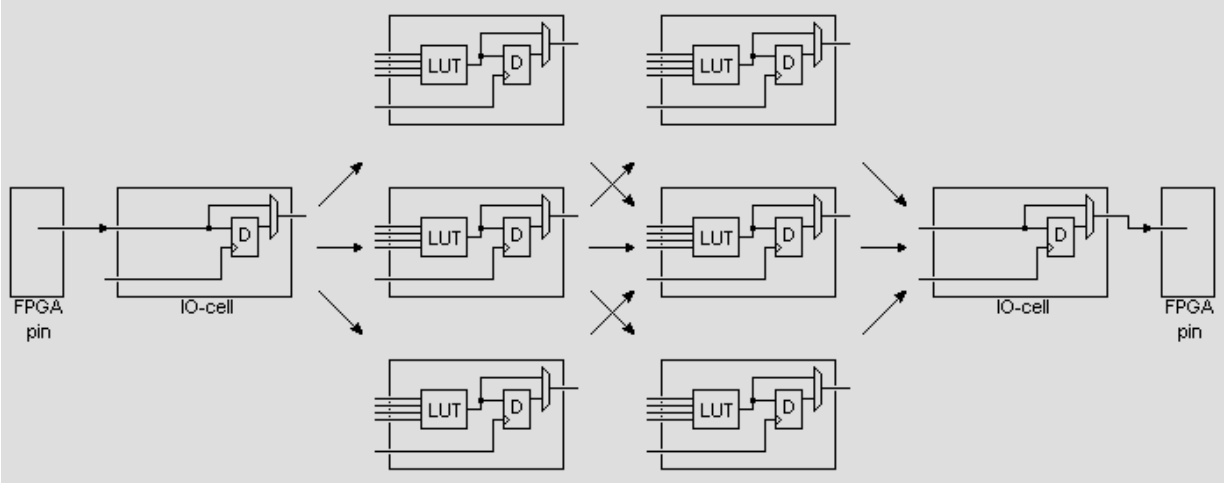

Figure3.1.3 input/output cells

3.1.4 Dedicated routing/carry chains

In addition to interconnect resources, FPGAs have very fast dedicated interconnects between neighbouring logic cells. The most usual type of these are "carry chains". Carry chains allow generating arithmetic functions like counters and adders competently low logic usage high operating speed. This is shown in the figure3.1.4 below

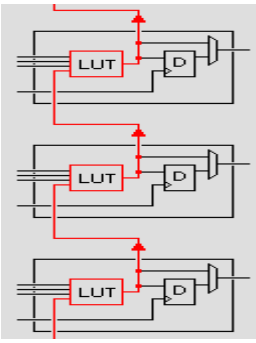

Figure3.1.4 showing carry chains

3.1.5 Internal RAM

In addition to the logic cells, all new FPGAs have dedicated blocks of static RAM distributed and controlled by the logic elements. There are many factors affecting the operation of RAM. The main factor is the number of logic elements that can access the RAM simultaneously .single-port RAM have only one agent can read/write the RAM. dual-port and quad-port RAMs have 2 and 4 logic cells that can read/write. Great to get data across clock domains each cell can use a different clock).

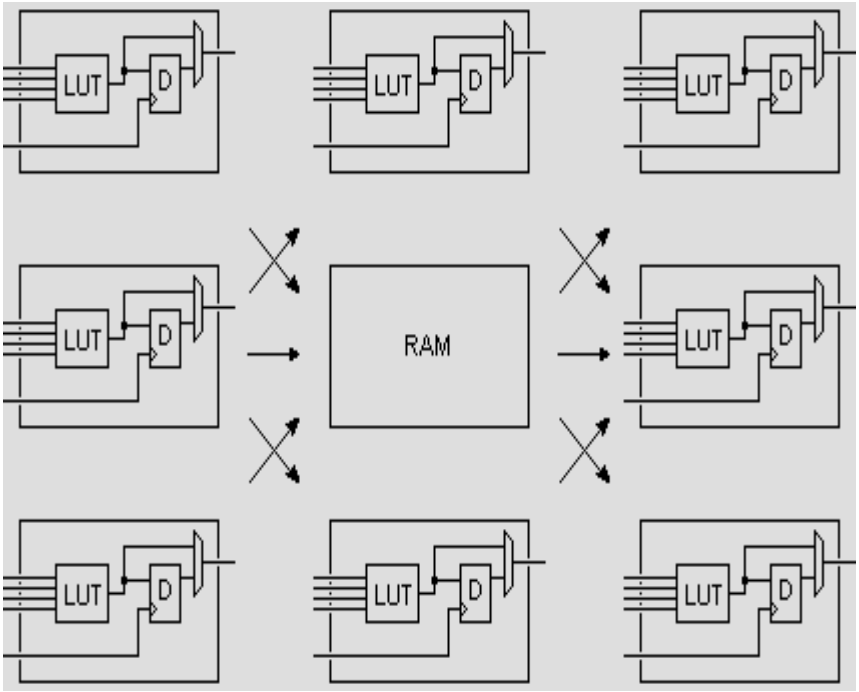

Figure3.1.5 internal RAM shared by logic cells

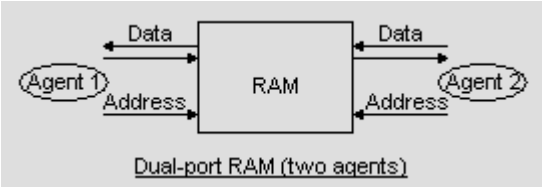

Figure3.1.6 dual port RAM

To find number of logic cells sharing , count the number of address buses going to the RAM. Each cell has a dedicated address bus. Each cell has also a read and a write data bus. Writing into the RAM is done synchronously. Reading is frequently done synchronously but can now and then be done asynchronously.

3.1.6 Blockram vs. Distributed RAM

There are two sorts of internal RAMs in an FPGA blockrams and distributed RAMs. The size of the RAM desirable habitually determines which type is used. The big RAM blocks are blockrams, which are positioned in dedicated areas in the FPGA. Each FPGA has a limited number of these, and if we don't use them, we lose them that is they cannot be used for other purposes but RAM. The small RAM blocks are either in smaller blockrams generally in Altera, or in "distributed RAM" in Xilinx. Distributed RAM permit using the FPGA logic-cells as minuscule RAMs which afford a very flexible RAM distribution in an FPGA, but isn't well-organized in term of area as the logic-cell can actually hold little RAM. Altera prefers building different size blockrams around the device more area efficient, but less flexible.

3.1.7Clocks and Global lines

An FPGA design is generally "synchronous" means the design is clock based for every clock rising edge the D-flipflops changes to new state.

In a synchronous design single clock drives lot of flipflops simultaneously which cause timing and electrical problems inside the FPGA. For proper functioning, FPGA manufacturers provide internal wires called "global routing" or "global lines". They distribute the clock signal all over the FPGA with a low skew i.e. the clock signal appears almost simultaneously to all the flipflops. When we feed a clock signal to FPGA, Use a dedicated input clock pin, shouldn't use any FPGA pin,. Usually, only such pin has the ability to drive a global line. FPGA software are aware of these dedicated clock input pins, and will automatically assign clocks .

An FPGA can use multiple clocks which in turn uses multiple global lines and dedicated input pins. Each clock forms a "clock domain" in the FPGA architecture.

For every clock domain, the FPGA software analyzes all flop-to-flop paths and gives the report with the maximum allowed frequencies. One clock domain may work at 100MHz, while another may work at 1000MHz. Given that every single clock uses a global line, and we use clock speeds that are lower than the extremely testified by the software timing issues, the design will work internally timing-wise. There may still some timing issues from the FPGA input and output pins. The software will give you a report about that.

3.2 DESIGN LANGUAGE

For creating a FPGA design, a designer has several options for algorithm implementation. Where gate-level design can result in optimized designs, the learning curve is considered prohibitory for most engineers, and the knowledge may not be portable across FPGA architectures. Here VHDL is used as high-level hardware design languages (HDLs) are discussed in which FPGA algorithms may be designed.

3.2.1 VHSIC Hardware Design Language

In recent years, VHSIC (Very High Speed Integrated Circuit) Hardware Design Language (VHDL) has become the industry standard for high-level hardware design. Since it is an IEEE standard, it is supported by a wide variety of design tools and is interchangeable between different vendors' tools. It also allows inclusion of technology-specific modules for most efficient synthesis to FPGAs. The first version of VHDL, IEEE 1076-87, appeared in 1987 and is updated in 1993, appropriately titled IEEE 1076-93. It is a high-level language, which supports the design, verification, synthesis and testing of hardware designs.

VHDL is used to write text models that describes the logic circuit. That model is synthesized, if it is part of the logic design. A simulation program is needed to test the logic design using simulation models to represent the logic circuits design. These simulation models are commonly called a testbench.

VHDL has constructs to handle the parallelism in hardware designs VHDL is strongly typed and not case sensitive. In order to represent operations which are common in hardware, there are several features of VHDL. VHDL allows arrays to be indexed in either ascending or descending order; both conventions are used in hardware VHDL has capability of handling text files input and output, but files are only used by a simulation testbench for verification data. There are some VHDL compilers which can generate executable binaries files . In this case we have to to write a VHDL testbench to verify the functionality of the design using files on the host computer to compare results with those desired.

It is easy for an inexperienced developer to produce code that simulates successfully but not be synthesized into a device, or is large to be practical. One mistake is the accidental production of latches to D-type flip-flops as storage elements. Finally when a VHDL model is translated into the "gates and wires" that are mapped onto FPGA, then it is the genuine hardware being configured, than the VHDL code being executed.

3.2.2 LIBRARIES

In VHDL we can choose the type of libraries to be used which in turn helps to choose the functions available that can be used in the design. Most of these libraries are of IEEE standard. In this project the libraries used are IEEE which contain packages which are described below

Std_logic_116: this package specifies the multi level logic system i.e the data types, text input /output to be used.

Std_numeric: this specifies the signed and unsigned data types and arithmetic's between these data types.

Std_textio: this package specifies function and procedures for text file handling

\

3.3 FLOATING POINT ARITHMETIC

IEEE floating point numbers have three basic units: the sign, the exponent, and the mantissa. The mantissa consists of the *fraction* and an implicit leading digit The exponent base (2) is implicit and is not be stored.

The following figure shows the layout for single (32-bit) and doubles (64-bit) precision floating-point values. The figure of bits for every field are represented in bit ranges are in square brackets.

Precision	**Signbit**	**Exponent length**	**Fraction bits**	**Bias value**
Single	1 [31]	8 [30-23]	23 [22-00]	127
Double	1 [63]	11 [62-52]	52 [51-00]	1023

Table 3.3.1 representation of floating point number

The Sign Bit

The sign bit denotes sign. 0 correspond to a positive number; 1 correspond to a negative number. Reversing the value of this bit flips the sign of the number.

The Exponent

The exponent field requires to represent both negative and positive exponents. For this representation , a *bias* is summed with the actual exponent. For IEEE single-precision floats, this value is 127. if exponent is zero 127 is stored in the exponent field. A stored value of 198 indicates an exponent of (198-127), or 75. For reasons discussed later, exponents of -127 (all 0s) and +128 (all 1s) are reserved for unique numbers.

The Mantissa

The *mantissa* is also called as the *significand* which represents the number of precision bits in the number. It is serene of fraction bits and the an implicit leading bit. For finding out the value of the leading bit, optimization is done, since the lone probable non-zero digit is 1. So , we can just mark as leading digit of 1, and don't need to mention it separately. As a result, the mantissa has 24 bits.

1. The sign bit is , 1 for negative ,0 for positive.
2. The exponent is of base two.
3. The exponent field contains 127 plus the true exponent for single-precision, or 1023 plus the true exponent for double precision.
4. The leading bit of the mantissa is normally supposed as 1.*f*, where *f* is the fraction bits.

Floating-Point Numbers range

For instance consider single-precision floats number. i.e a 32-bit number the fields to cover a much larger range.. For instance, standard 32-bit integers, with all precision zero, can precisely pile up with integers of 32-bits wide. Single-precision floating-point, is not capable to counterpart this resolution with its 24 bits. It doesapproximate this value by effectively truncating from the LSB. For example:

11110000111111100 10101010 00001111 // 32-bit integer
= +1.11100001111111111111111 10101010 x 2^{31} // Single-Precision Float

= 11110000 1101111111110 00000000 // Corresponding Value

This approximates the 32-bit value, which do not produce an exact representation. alternatively, in addition the capability to represent fractional components (which integers lack completely), the floating-point value can characterize numbers around 2^{127}, associated to 32-bit integers maximum value around 2^{32}.

The positive floating point numbers range can be split into normalized number that safeguard the full precision of the mantissa, and *denormalized* numbers which use a portion of the fraction's precision.

	Denormalized	**Normalized**	**Approximate Decimal**
Single Precision	$\pm 2^{-149}$ to $(1-2^{-23})\times 2^{-126}$	$\pm 2^{-126}$ to $(2-2^{-23})\times 2^{127}$	$\pm \sim 10^{-44.85}$ to $\sim 10^{38.53}$
Double Precision	$\pm 2^{-1074}$ to $(1-2^{-52})\times 2^{-1022}$	$\pm 2^{-1022}$ to $(2-2^{-52})\times 2^{1023}$	$\pm \sim 10^{-323.3}$ to $\sim 10^{308.3}$

Table 3.3.2 range of floating point numbers

Since the sign of floating point numbers is represented by a unique leading bit, the range for negative numbers found out by the negation of the above values shown.

There are five distinct numerical ranges that single-precision floating-point numbers are able to represent:

1. Negative numbers less than $-(2-2^{-23}) \times 2^{127}$ (*negative overflow*)
2. Negative numbers greater than -2^{-149} (*negative underflow*)
3. Zero
4. Positive numbers less than 2^{-149} (*positive underflow*)
5. Positive numbers greater than $(2-2^{-23}) \times 2^{127}$ (*positive overflow*)

Here's a table of the effective range (excluding infinite values) of IEEE floating-point numbers:

	Binary	**Decimal**
Single	$\pm (2\text{-}2^{-23}) \times 2^{127}$	$\sim \pm 10^{38.53}$
Double	$\pm (2\text{-}2^{-52}) \times 2^{1023}$	$\sim \pm 10^{308.25}$

Table 3.3.3 equivalent decimal range

Floating point used in the project

As the image pixel varies from 0 to 255 we have used just 8 bits for fraction and 4 bits for exponent and 2 bit for sign and is shown in tne table

	sign	**exponent**	**Fraction**	**Bias**
Used in design	1[12]	4[11_8]	7[7_0]	7

3.4 IP CORE

An IP CORE (Intellectual Property) core is a block of HDL code that vendor company engineers have already written to perform a specific logic function. It is a precise piece of code designed to do a particular job. IP cores can be used in a intricate design where we need to save time. IP cores have their advantages and disadvantages. Although they may simplify the design, we have to design the interfaces to send and receive data from this "black box". Also, while an IP core may reduce design time, we have to pay for the right to use the core..

Here in the project the core used is block memory generator and in it simple ROM.Xilinx provides a Block Memory Generator core to create high-performance memories operating at frequencies up to 450 MHZ. The Block Memory Generator LogiCORE IP core automates the generation of area and performance optimized block memories for Xilinx FPGAs. This facility is provided through the ISE Design Suite CORE Generator System, the core enables users to create block memory functions to suit the design. The key features are given below

- Generates Single-Port RAM, Simple Dual-Port RAM, True Dual-Port RAM, Single-Port ROM, or Dual-Port ROM
- Performance up to 450 MHz
- Data widths from 1 to 4096 bits
- Memory depths from 2 to 9M (limited only by memory resources on target device)
- Variable Read-to-Write aspect ratios
- Optimized algorithm for minimum block RAM resource utilization
- Low Power implementation option to reduce power consumption
- Configurable memory initialization values
- Supports individual write enable per byte with or without parity.
- Selectable per-port operating mode: WRITE_FIRST, READ_FIRST or NO_CHANGE

CHAPTER-4

FPGA IMPLEMENTATION OF THE PROPOSED FILTER

The aim of this project is the implementation of the proposed algorithm on target FPGA hardware. For this we have to compose the algorithms in the VHDL language and synthesizing the algorithms for the FPGA. It convenient to develop a structural style of VHDL coding which facilitates reusability and to develop a common hierarchy. This project has been developed such that it is largely device independent means the code can be compiled for any FPGA architecture with little difficultly.

As said in earlier chapters the image size used is 512x512 and FPGA used for implementation of the proposed algorithm is virtex-5.The implementation is done according to the steps given below

1 Image data acquisition
2 Kernel extraction
3 convolution

4.1 Image data acquisition

In order to process the image the image pixels either has to be stored in FPGA internal memory or to be given as input serially according to clock. Here we have stored the image data in BROM (block ROM). From this block pixels are serially given for processing. For storing image data in internal memory there are two ways either store individually all pixel in BROM or we can make a file with extension .Coe in MATLAB, which can be directly accessed by BROM created by using IP core wizard. Here BROM is used is single port simple ROM of version 6.3 provided by logicore technologies in Xilinx ISE 13.4.

The BROM specifications used in the project of 8 bit wide and depth of 262144(512x512).since the image pixel value varies from 0 to 255 it is at most 8 bit wide and there are 262144 values of pixels to stored. For accessing every pixel is provide with a address .For every rising edge of clock and with a valid address a 8 bit data is obtained as output. While generating the core the image data file with extension .Coe is loaded by enabling option load INTI FILE shown while generating. The generated core is shown in the figure .the core synthesized automatically while it is being generated and is attached with the project files as a component and port mapping as per our requirement.

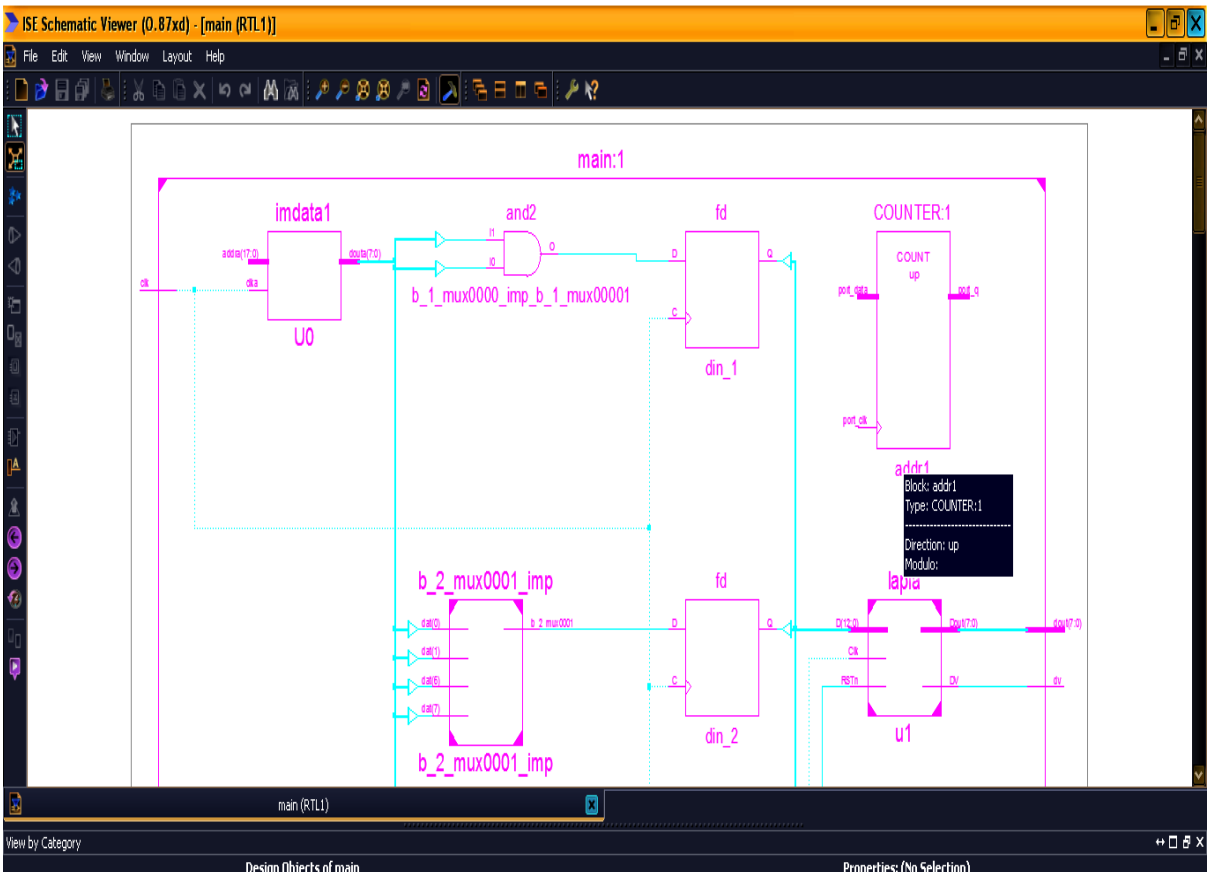

Figure 4.1.1 synthesized IP core ROM with name imdata1

4.2Kernel extraction

In the previous section as we said the output will be serially with the clock, now this data has to be processed to get the kernel for convolution. In image processing, most of the algorithms belong to a category called windowing operators. Windowing operators use a window to calculate their output. For example, it performs operations like finding the average of all pixels median, in the neighbourhood of a pixel. Generally window sizes are selected in odd number such as 3x3 or 5x5.the window size used in the project is 7x7. The centre pixel which the window is found is called the *origin*. Figure below, shows a 7 by 7 pixel window and the origin.

			origin			

Figure 4.2.1 7x7 window operator

This window with some values according to is moved along the image and both mask and window are convoluted .Actually in MATLAB mask moves all over the image but in the hardware design the pixels in the window keep changing and the kernel remains constant .for every clock cycle the data of the window is changed, convoluted with kernel or mask and centre pixel is replaced with the calculated value.

Now we shall discuss the hardware realization of the moving window. Here window moves the window not the kernel so we call moving window. In order to implement a moving window hardware in VHDL, a method was proposed that took advantage of certain features of FPGAs. FPGAs usually handle flip -flops quite easily, but instantiation of memory on chip is more complicated.. It was determined that the output of the design should be vectors for pixels in the window, along with a data -valid signal, which is used to inform an algorithm using the window generation unit as to when the data is ready for processing. Since it was necessary to achieve maximum performance in a relatively small area, FIFO units specific to the target FPGA were used. As we choose window of size 7x7 we need six FIFO's in number. The idea of the hardware is shown in the figure and the implemented and synthesized design is shown in the figure.

As we can see from the figure that as the data comes in for every clock the a pixel is sent through a series of register and flip flops and the output window is obtained .The size of the register i.e series of flip flops technically but we represent here as registers is 13 bit instead of 8 bit because the kernel coefficients are of floating point numbers. the size of FIFO used of 13 bits width and depth of 512 .

As we can see that the vectors of window are obtained using the above architecture . After the pixels are available for processing data valid signal is activated for processing i.e after the reading the 7^{th} pixel of the 3^{rd} row, so that first the centre pixel will be of 4^{th} pixel of row 4.the rows above and columns before are set to zero in output image since these pixels are not processed so the border pixels are to be set zeros. The synthesized architecture is shown in the figure below.

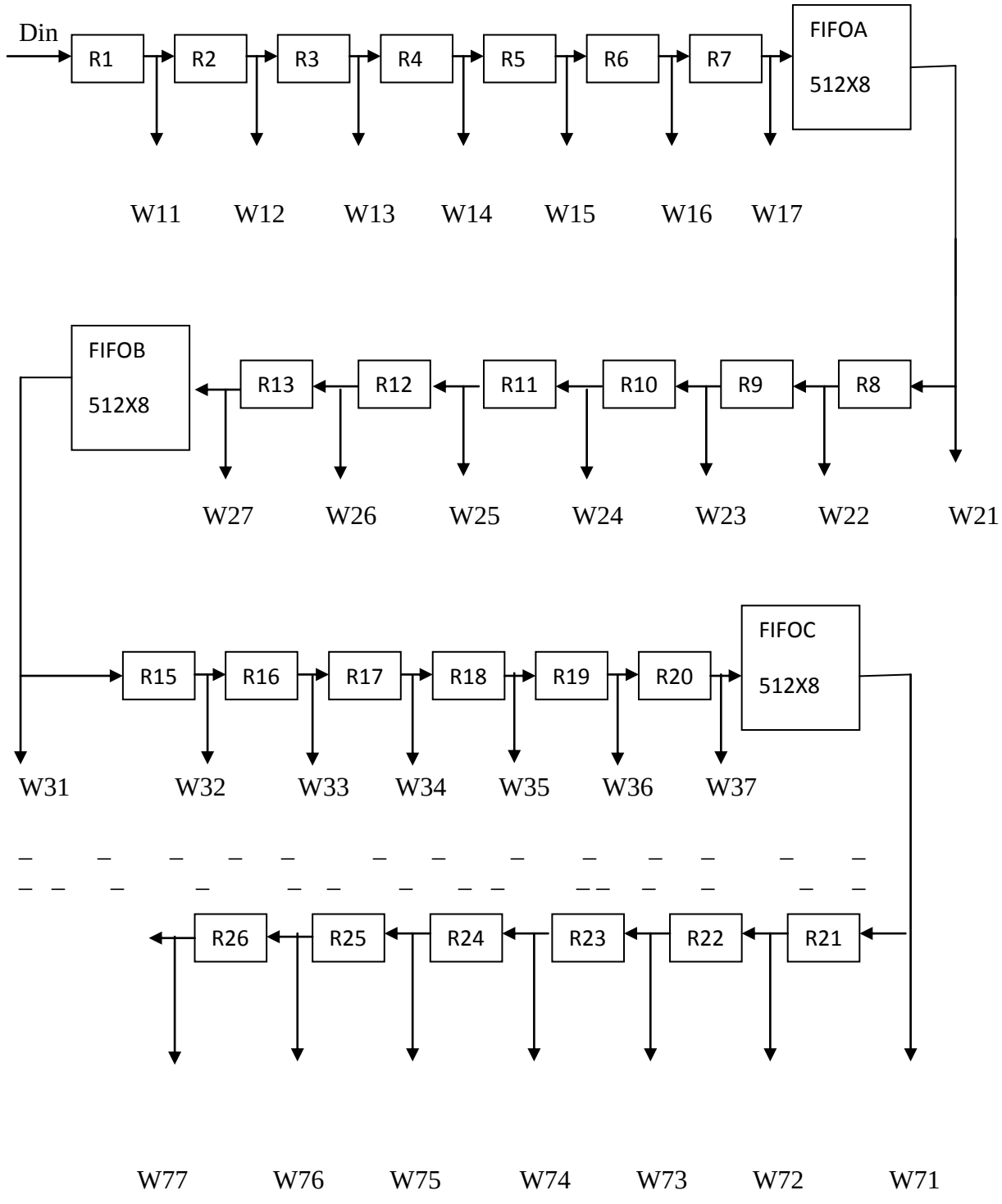

Figure4.2.2 Moving window architecture

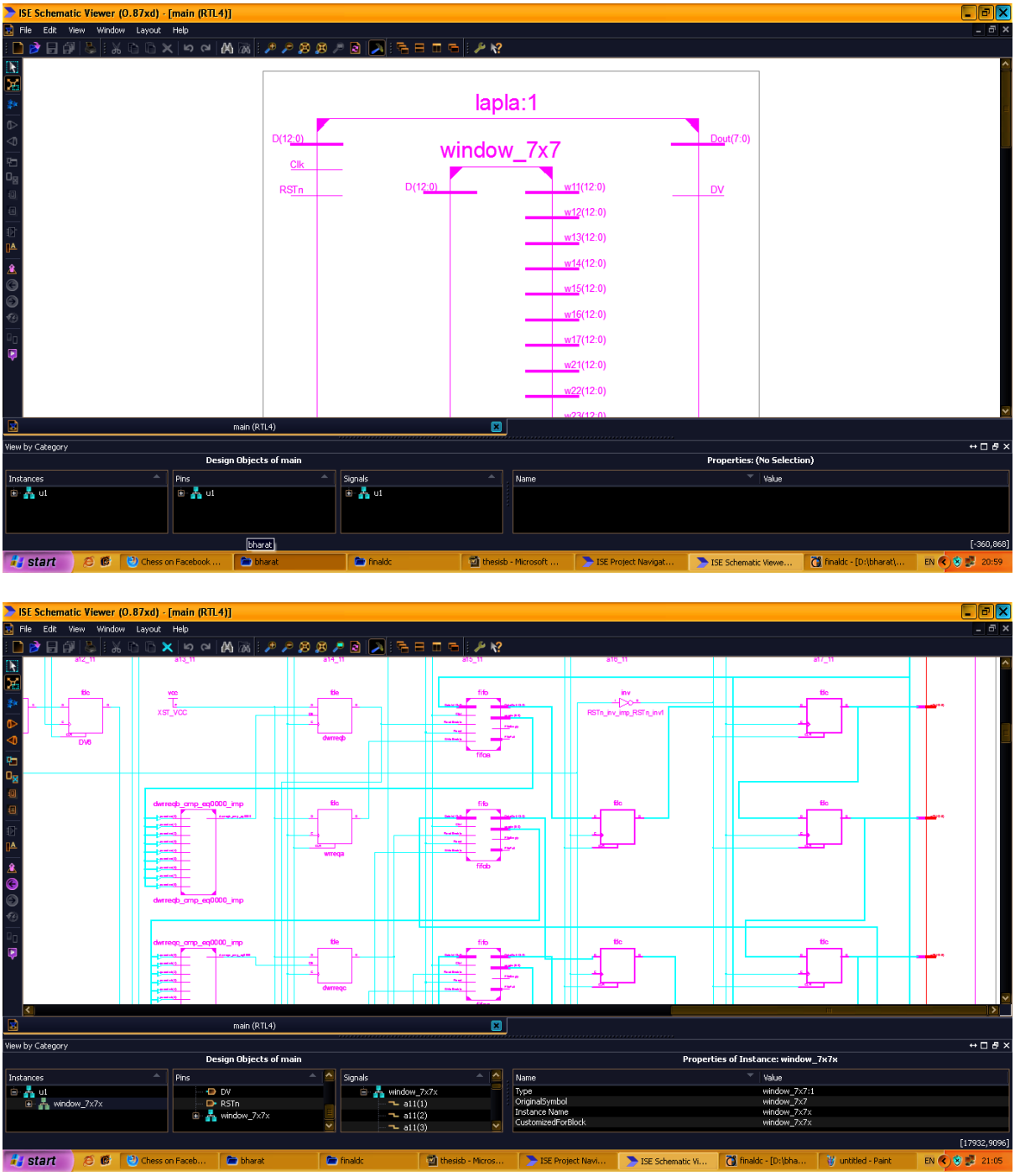

Figure4.2.3 synthesized moving window architecture

Now the window obtained has to be processed for this filter to get the kernel the equations used are given earlier and shown here. Let the original image f be corrupted with AWGN . Then the corrupted image g may be expressed as:

$$g(x,y) = f(x,y) + \eta(x,y) \quad \text{..... (1)}$$

In an image, the spatial distance between any arbitrary pixel in a particular window at location (x1,y1) and the center pixel at location (x,y) is calculated as

Now the distance kernel is defined by

$$d_s = \sqrt{(x_1 - x_2)^2 - (y_1 - y_2)^2} \quad \text{.....2}$$

$$w_d = 1 - \frac{d_s}{d_{max}} \quad \text{.....3}$$

where, *dmax* is the maximum radial distance from center. The correlation between pixels goes on decreasing as the distance increases. Hence, when *wd* becomes very small the correlation can be taken as zero. When the small values of distance kernel are replaced by zero we get a circular shaped filtering kernel. The circular shaped kernel is denoted as *wcd* .

The gray level distance between any arbitrary pixel g(x1,y1) of a particular window at location (x1,y1) and the center pixel g(x,y) at location (x,y) is calculated as

$$d_g = \sqrt{g^2(x_1, y_1) - g^2(x_2 - y_2)} \quad \text{.....4}$$

The gray level distance *dg* can be used to find the gray level kernel which is defined

$$w_g = e^{(d_g^2 / 2\sigma_g^2)} \quad \text{.....5}$$

where, σ is the standard deviation of the distribution function *wg*.

The filtering kernel of CSF can be prepared from w_{cd} and w_g as:

$$w = w_{cd}\ w_g \quad \text{.....6}$$

The distance kernel is calculated prior according to the and stored as constants in a package .The constants are shown in the figure below and this is distance kernel and denoted by wcd

0	0.15	0.25	0.29	0.25	0.15	0
0.15	0.3	0.47	0.52	0.47	0.3	0.15
0.25	0.47	0.6	0.76	0.6	0.47	0.25
0.29	0.52	0.76	1	0.76	0.52	0.29
0.25	0.47	0.6	0.76	0.6	0.47	0.25
0.15	0.3	0.47	0.52	0.47	0.3	0.15
0	0.15	0.25	0.29	0.25	0.15	0

Figure 4.2.4 distance kernel

As these are floating point numbers for these numbers special packages are written for their arithmetic and included in the project. In the package functions written are described here .they are

(1) Std_float13
(2) Float13_std
(3) “+” overloading operator for addition
(4) “*“ overloading operator for multiplication
(5) “-“ overloading operator for subtraction

The function Std_float13 is used to convert 8 bit image pixel to 13 bit floating point as explained in the chapter 3, this is done when we are reading the data from IP core. The function Float13_std is used to convert 13 bit float point to std_logic_vector. Finally it is converted integer while writing text file. and the other functions are for float point arithmetic.

Now the grey kernel is calculate by the window obtained from the moving window architecture using over loading operators, denoted by wg. Now final kernel is obtained by multiplying both distance kernel and grey level kernel.

Finally the kernel is obtained and has to be convoluted by the window at the same time so kernel extraction and convolution has to be done in single clock cycle.

4.3 convolution

Convolution is commonly used algorithm in DSP systems. It is from a class of algorithms called spatial filters. Spatial filters use a wide variety of *masks*, also known as *kernels*, to calculate different results, depending on the function required. For example certain masks results smoothing,while others results edge detection. The convolution algorithm is shown here. For each input pixel of the window, the values in that window are multiplied by the convolution mask. They are summed together and then divided by the total of pixels in the window. This value is replaced for the origin pixel in the output image for that position. Mathematically, this is represented using the following equation

$$y(n_1,n_2) = \sum_{k1=-\infty}^{\infty} \sum_{k2=-\infty}^{\infty} A(k_1,k_2)k(n_1-k_1,n_2-k_2), \quad \text{.....7}$$

This is explained by using a 3x3 window in the figure . Here the window pixela are multiplied and the added and divided by and rounded off and replaces the original pixel.

10	20	30
30	40	25
32	45	55

4.3.1 Input window

2	1	1
1	1	1
1	1	2

4.3.2 mask

Output pixel=10x2+20+30+30+40+25+32+45+55x2/9=39

The hardware design of the convolution algorithm in VHDL is difficult problem . This is due to its use of more complex mathematics. The convolution algorithm uses adders, multipliers,

and dividers to compute its output. On FPGAs, use of mathematics reduces performance. Many hardware designers favor techniques that reduce the algorithm's dependency on complex mathematics. Since we have wrote some packages supporting floating point arithmetic it is achievable. the architecture is simple kernel and window are multiplied and added and replace the original pixel in the output image .the synthesized architecture is according to the equation sown below

The filtering kernel w is slide throughout the image corrupted with noise to get the estimated output. The estimated pixel can be expressed as

$$f'(x,y)=\frac{\sum_{s=-a}^{a}\sum_{t=-b}^{t} w(s,t)g(x+s,y+t)}{\sum_{s=-a}^{a}\sum_{t=-b}^{b} w(s,t)} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots 8$$

4.3.1 ROW COLUMN COUNTER

A row column counter must be used to set data valid signal .The output image must be the exact dimension as the input image. A VHDL counter was written to count pixels as the data enters into the entity. Since images are two dimensional , two counters were needed one to count

rows and other to count columns in the image. The VHDL entity that implements this functionality is called rc_counter. Since it is a separate VHDL entity, this counter was usable to later algorithms, where this functionality was also needed. This is shown in the synthesized figure 4.3.3.

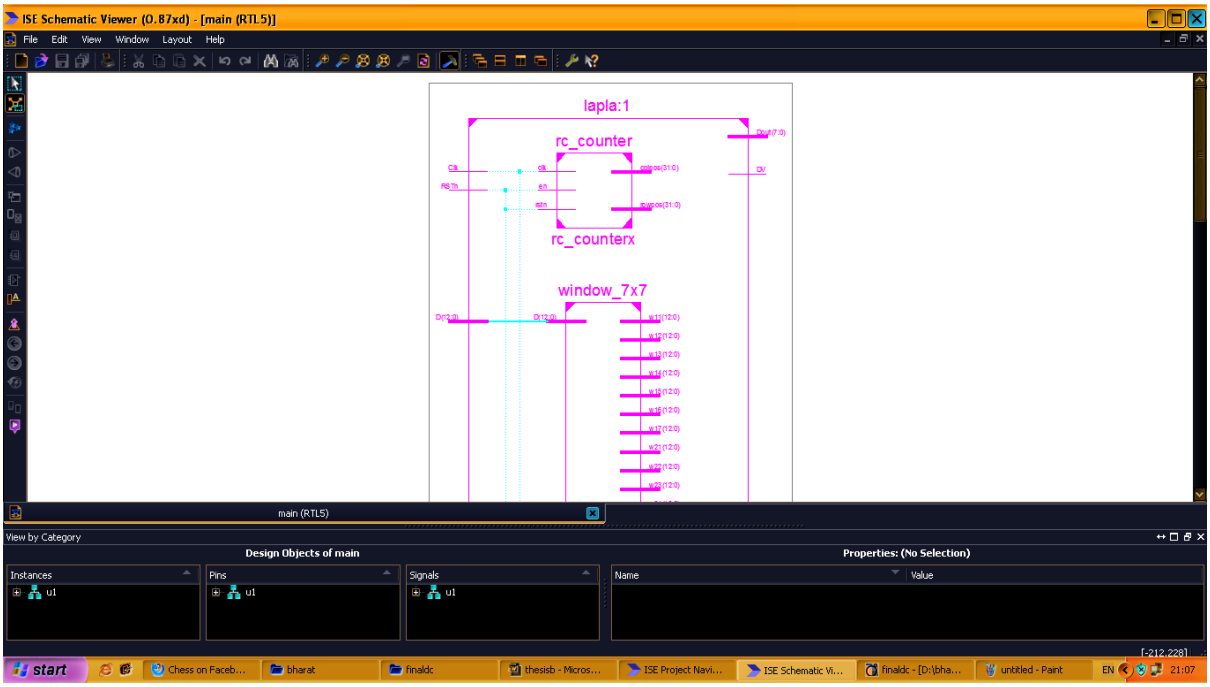

Figure 4.3.3 synthesized convolution architecture named as lapla

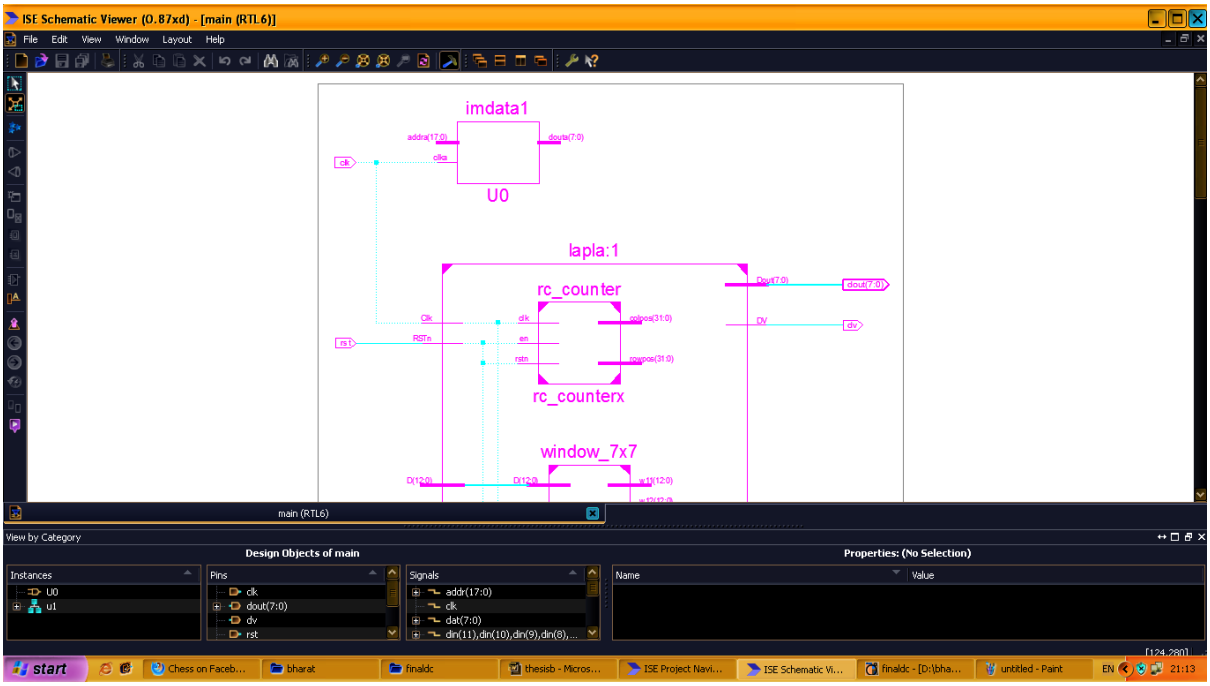

Figure 4.3.4 Synthesized architecture for circular spatial filter

CHAPTER-5

RESULTS

In this chapter results of synthesis report, simulation of xilinx ise tool are presented. The output images obtained from FPGA are presented.

5.1 Synthesis report

Synthesis report presents the amount and type of the hardware used in the fpga

5.1.1 Report for simulation code

Device utilization summary:

Selected Device : 5vlx110tff1136-3

- Slice Logic Utilization:

  Number of Slice Registers: 1968 out of 69120 2%

  Number of Slice LUTs: 16166 out of 69120 23%

  Number used as Logic: 15523 out of 69120 22%

  Number used as Memory: 643 out of 17920 3%

  Number used as RAM: 624

  Number used as SRL: 19

- Slice Logic Distribution:

  Number of LUT Flip Flop pairs used: 17332

  Number with an unused Flip Flop: 15364 out of 17332 88%

  Number with an unused LUT: 1166 out of 17332 6%

  Number of fully used LUT-FF pairs: 802 out of 17332 4%

  Number of unique control sets: 38

- IO Utilization:

  Number of IOs: 24

  Number of bonded IOBs: 24 out of 640 3%

- Specific Feature Utilization:

  Number of BUFG/BUFGCTRLs: 1 out of 32 3%

  Number of DSP48Es: 46 out of 64 71%

- Timing summary

  Minimum period: 218.650ns (Maximum Frequency: 4.574MHz)

  Minimum input arrival time before clock: 1.947ns

  Maximum output required time after clock: 2.775ns

5.1.2 Report for dump code

- Device utilization summary:

  Selected Device : 5vlx110tff1136-3

- Slice Logic Utilization:

  Number of Slice Registers: 1967 out of 69120 2%

  Number of Slice LUTs: 16026 out of 69120 23%

  Number used as Logic: 15399 out of 69120 22%

  Number used as Memory: 627 out of 17920 3%

  Number used as RAM: 608

  Number used as SRL: 19

- Slice Logic Distribution:

  Number of LUT Flip Flop pairs used: 17123

  Number with an unused Flip Flop: 15156 out of 17123 88%

  Number with an unused LUT: 1097 out of 17123 6%

  Number of fully used LUT-FF pairs: 870 out of 17123 5%

  Number of unique control sets: 44

- IO Utilization:

  Number of IOs: 11

  Number of bonded IOBs: 11 out of 640 1%

- Specific Feature Utilization:

  Number of Block RAM/FIFO: 64 out of 148 43%

  Number using Block RAM only: 64

  Number of BUFG/BUFGCTRLs: 1 out of 32 3%

  Number of DSP48Es: 46 out of 64 71%

- Timing summary

  Minimum period: 220.985ns (Maximum Frequency: 4.525MHz)

  Minimum input arrival time before clock: 1.947ns

  Maximum output required time after clock: 2.775ns

5.2 Design summary

5.2.1 Report for simulation code

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	1968	69120	2%
Number of Slice LUTs	16166	69120	23%
Number of fully used LUT-FF pairs	802	17332	4%
Number of bonded IOBs	24	640	3%
Number of BUFG/BUFGCTRLs	1	32	3%
Number of DSP48Es	46	64	71%

Figure 5.2.1 design summary for simulation code

5.2.2 Report for dump code

Device Utilization Summary				
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	1,973	69,120	2%	
Number used as Flip Flops	1,967			
Number used as Latch-thrus	6			
Number of Slice LUTs	15,124	69,120	21%	
Number used as logic	14,417	69,120	20%	
Number using O6 output only	13,289			
Number using O5 output only	160			
Number using O5 and O6	968			
Number used as Memory	627	17,920	3%	
Number used as Single Port RAM	608			
Number using O6 output only	608			
Number used as Shift Register	19			
Number using O6 output only	10			
Number using O5 output only	9			
Number used as exclusive route-thru	80			
Number of route-thrus	254			
Number using O6 output only	183			
Number using O5 output only	14			
Number using O5 and O6	57			
Number of occupied Slices	5,892	17,280	34%	
Number of LUT Flip Flop pairs used	15,833			
Number with an unused Flip Flop	13,860	15,833	87%	
Number with an unused LUT	709	15,833	4%	
Number of fully used LUT-FF pairs	1,264	15,833	7%	
Number of unique control sets	54			
Number of slice register sites lost to control set restrictions	106	69,120	1%	
Number of bonded IOBs	11	640	1%	
Number of BlockRAM/FIFO	64	148	43%	
Number using BlockRAM only	64			
Number of 36k BlockRAM used	64			
Total Memory used (KB)	2,304	5,328	43%	
Number of BUFG/BUFGCTRLs	1	32	3%	
Number used as BUFGs	1			
Number of DSP48Es	46	64	71%	
Average Fanout of Non-Clock Nets	5.42			

Figure 5.2.2 design summary for dump code

5.3 simulation result in ISE simulator

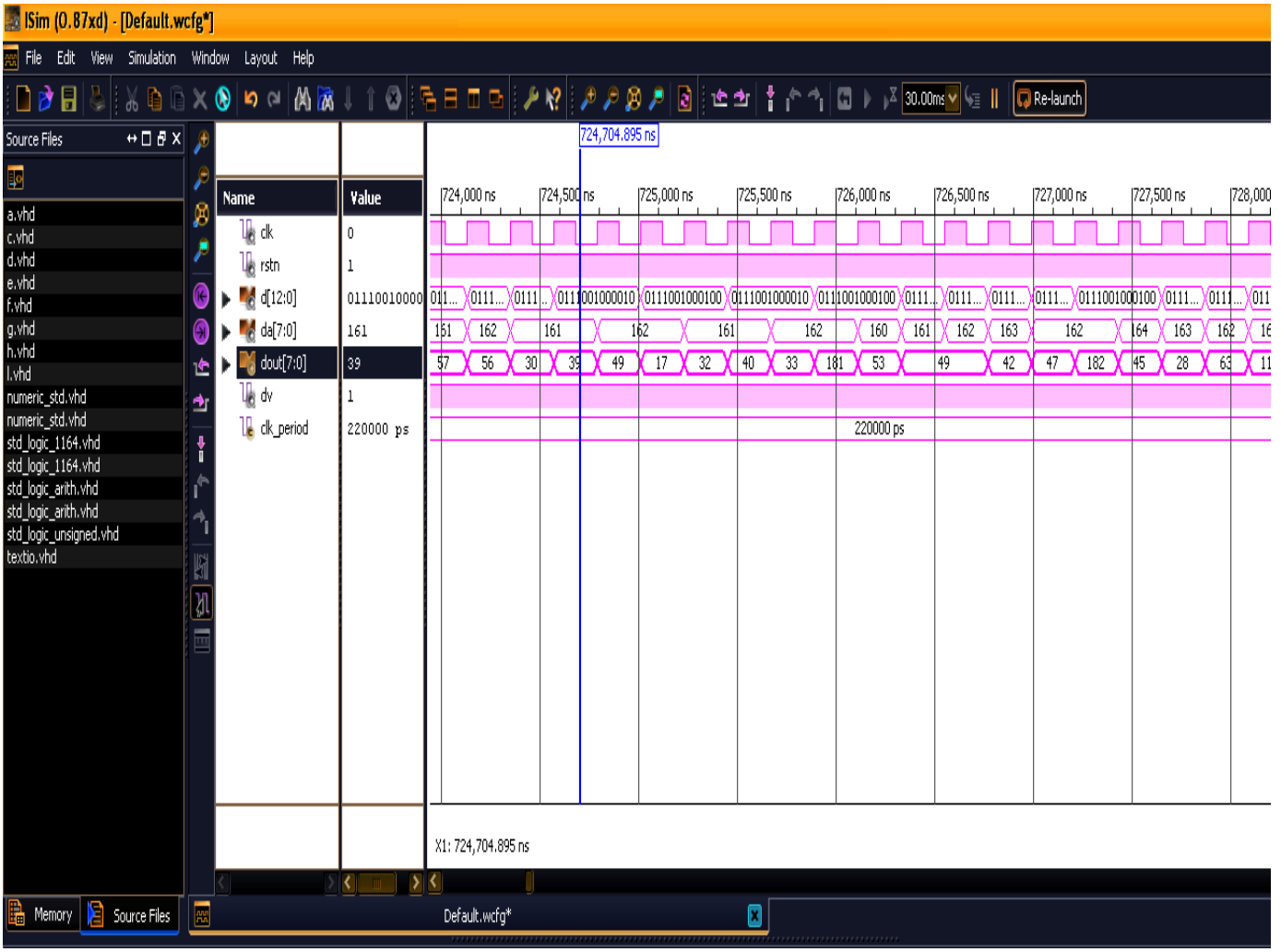

Figure 5.3.1 simulation in xilinx ise 13.4 simulator

5.4 Lena image

For noisy Lena image corrupted with AWGN of standard deviation =15

Figure 5.4.1 (a) original image (b) noisy image (c) processed image

For noisy Lena image corrupted with AWGN of standard deviation =40

Figure 5.4.2 (a) original image (b) noisy image (c) processed image

5.5 Pepper image

For pepper image corrupted with AWGN of standard deviation =15

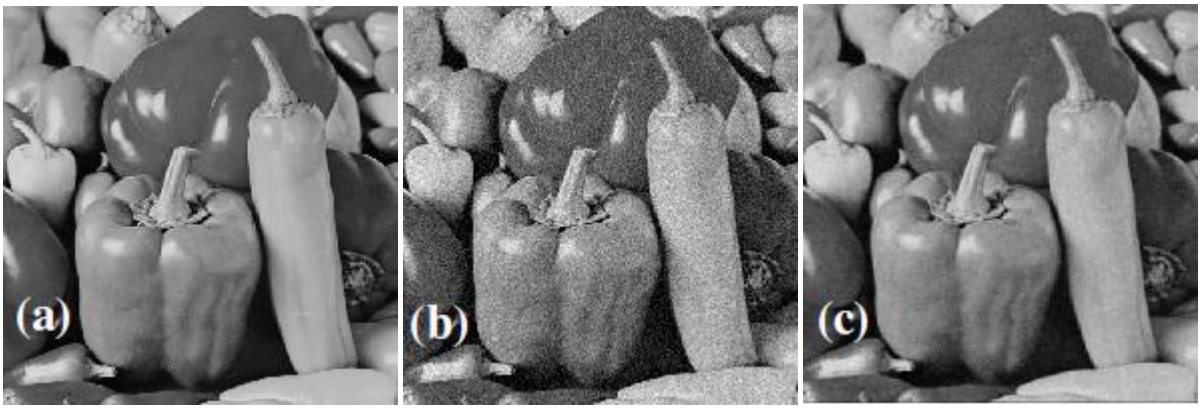

Figure 5.4.3 (a) original image (b) noisy image (c) processed image

For pepper image corrupted with AWGN of standard deviation =40

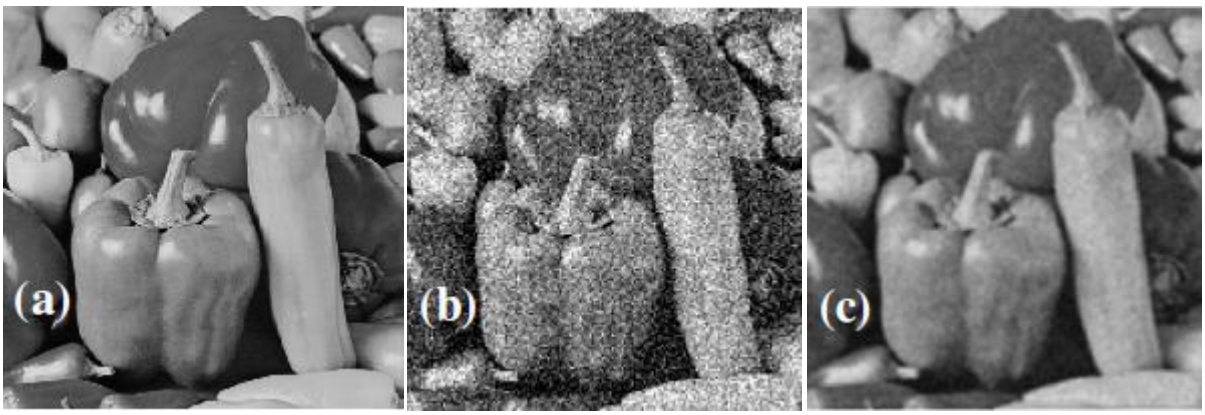

Figure 5.4.4 (a) original image (b) noisy image (c) processed image

CHAPTER-6

CONCLUSION AND FUTURE WORK

6.1Comparative Analysis:

6.1.1 Comparative analysis for PSNR

pepper	Standard deviation=15	Standard deviation =40
CSF 7X7 FPGA	35.54	31.68
CSF 7X7 MATLAB	31.26	27.92

TABLE 6.1.1 COMPARATIVE ANALYSIS FOR PSNR

Lena	Standard deviation =15	Standard deviation =40
CSF 7X7 FPGA	32.43	28.71
CSF 7X7 MATLAB	29.62	26.99

TABLE 6.1.2 COMPARATIVE ANALYSIS FOR PSNR

6.1.2 Timing comparison

	FPGA	PENTIUM IV CORE 2DUO PROCESSOR (64 BIT OS)	PENTIUM IV DUO PROCESSOR (32BIT OS)	PENTIUM IV PROCESSOR (32 BIT OS)
TIME TAKEN FOR A FRAME	57.9ms	3.52s	7.29s	19.3s
Clock frequency	4.525MHZ	2.4 GHZ	2.8 GHZ	1.7 GHZ

TABLE 6.1.2 TIMING COMPARISON

6.2CONCLUSION:

6.3 SCOPE FOR FUTURE WORK:

References

[1] sukadev meher ,nilmani bhoi Circular "spatial filtering under high-noise-variance conditions **"** elesiver Computers & Graphics 32 (2008) 568–580.
[2] Hussian z"practical appilication of prallelprocssing techniques ellis harword west sussex uk 1991
[3] Nelson a "further study of image processing on fpga"independent study paper may2005
[4] Mattahis jung "real time implementation of miscellanious edge detection algorithm"independent study paper 2010
[5] pedroni "system design using vhdl'
[6] mathwork.com
[7] Gonzalez RC, Woods RE. Digital image processing. 2nd ed. Englewood Cliffs,NJ: Prentice-Hall; 2002.
[8] Jain AK. Fundamentals of digital image processing. Englewood Cliffs, NJ:1532–46. Prentice-Hall; 1989.
[9] Blum RS, Liu Z. Multi-sensor image fusion and its applications. Boca Raton:Taylor & Francis; 2006.
[10] Liu J, Moulin P. Information-theoretic analysis of interscale and intrascale dependencies between image wavelet coefficients. IEEE Transactions on Image Processing 2001;10(11):1647–58.
[11] Kingsbury NG. Image processing with complex wavelets. Philosophical Transactions of the Royal Society of London Series A 1999;357(1760):2543–60..
[12]www.cosmiac.com
[13] www.fun4fpga.com
[14]www.xilinx.com
[15]www.edaboard.com
[16]www.velocityreviews.com