HARDWARE DESIGN OF SYNTHETIC DISCRIMINANT FUNCTION BANDPASS DIFFERENCE OF GAUSSIAN FILTER

By

Saima Gul
2009-NUST-MS PhD-ComE-10

Submitted to the department of Computer Engineering in fulfillment of the requirements for the degree of

Master of Science

In

Computer Engineering

Thesis Supervisor

Dr. Saad Rehman

College of Electrical and Mechanical Engineering,
National University of Sciences and Technology
2012

DECLARATION

I hereby declare that the thesis titled “Hardware Design of Synthetic Discriminant Function Bandpass Difference of Gaussian Filter” is neither whole nor as a part thereof has been copied from any source (except data). It is further declared that I developed this report entirely on the basis of my personal efforts. No portion of the work presented in this report has been submitted in support of any application for any degree and qualification of this or any other university or institute of learning. All the sources used in this thesis have been cited and the contents of this thesis have not been plagiarized.

Saima Gul

ACKNOWLEDGEMENTS

All thanks to **Almighty Allah**, who made all this possible for me. Without His help it was impossible to achieve this milestone.

I extend my deep appreciation and thank to my thesis supervisor **Dr. Saad Rehman** for all his support, guidance and kind cooperation. I am thankful to him for his precious time, valuable opinion and faith in me. A special thanks to **Dr. Shoab Ahmed Khan** for his guidance and precious time. And I am thankful to **Dr. Usman Qamar** and **Dr. Arsalan Shaukat** for their guidance.

A heartiest gratitude to my parents, brothers and sisters who always pray for me and encourage me to do whatever I want. Last but not the least, I appreciate all those who remember me in their prayers and encourage me throughout the deals of my life.

The constant advice and encouragement of my friends and the work of early scientists which blazed this study is also highly acknowledged.

Dedicated to

My loving Parents

ABSTRACT

Synthetic Discriminant Function (SDF) has been the main area of focus for many researchers. It has been applied successfully in different applications. This filter was designed to overcome the limitations of Matched Filter and Phase Only Filter. The main idea behind SDF design is to include the expected distortions in the filter design. This technique improved the immunity to the distortions. SDF filter is computationally very intensive algorithm when implemented in MATLAB. Its real-time implementation with the specialized hardware is essentially required in order to achieve high speed. It is an optical technique, which is digitized.

This research work is focused on the hardware design of **Synthetic Discriminant Function Bandpass Difference of Gaussian Filter.** The objective is to provide faster and cheaper solution to the object recognition problem. This work pays special attention to the trade-off among area, cost, performance and precision aspects. Hardware design of SDF filter is the first step towards designing fast and efficient hardware for correlation filters with distortion invariance.

Table of Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

1.1. Problem Overview

Correlation filters have been successfully applied for a number of applications, such as object tracking, automatic target recognition and recognition of biometrics e.g. face, finger print and iris [1, 2, 3]. Synthetic Discriminant Function filter is one of the well-known correlation filters. A composite image is formed by the weighted sum of training images. Input images are preprocessed using a Difference of Gaussian band pass filter which performs the edge enhancement of an input image and in this way we can get sharp correlation peaks. Preprocessed image is correlated with the composite image to produce the correlation plane which is tested for its peak value to detect the object.

SDF filter is computationally very intensive algorithm when implemented in software. Its real-time implementation with the specialized hardware is essentially required. The system's software implementation could be used but there are environments where real time implementation of SDF is required. For example, an environment in which SDF is required to be trained at run time. In this case, images are taken by the camera and these images are used to train SDF filter.

1.2. Project Objectives

The main goal is to design digital hardware for Synthetic Discriminant Function filter to provide a faster and cheaper solution to the pattern recognition problem, and to include Difference of Gaussian bandpass filter for edge enhancement of the input images.

The system design is divided into three parts. First is the hardware design for SDF synthesis process. Second, the hardware design of Difference of Gaussians and last is the Correlator design. The designed system is able to produce composite SDF image, and the Difference of two Gaussian functions, and finally to produce the correlation plane for the input image using the Correlator. All the code is written in Verilog and simulations are performed in ModelSim and the system is synthesized in Xilinx.

1.3. Thesis Outline

This report is organized in seven chapters. Chapter 1 gives a brief introduction of the research work. Chapter 2 provides an introduction to the Correlation filters and explains some of the correlation filters and SDF filter in particular. Chapter 3 briefly describes the Difference of Gaussians filter. Chapter 4 is an overview of the FPGA based digital system designs and digital signal processing on FPGAs. Chapter 5 discusses the hardware design of the SDF filter. Results and discussions are presented in Chapter 6. Conclusions and proposed future work are discussed in Chapter 7.

Chapter 2

CORRELATION FILTERS

2.1. Linear Discrimination Function

Linear Discriminant Analysis is a widely used pattern recognition method. A number of pattern recognition systems based on LDA have been developed and encouraging results have been attained.

"*Discriminant analysis finds a set of linear combinations of the variables, whose values are as close as possible within groups and as far apart as possible between groups. The linear combinations are called discriminant functions. Thus, a discriminant function is a linear combination of the discriminating variables*"[4].

We can write a linear discriminant function (LDF) of components x as:

$$s(x) = v^t x + v_0 \tag{1}$$

Where v is the weight vector and v_0 is the threshold weight. The LDF of the form of Equation (1), is a two category case [5]. This two category classifier follows the following decision rule.

- Decide v_1 if $s(x) > 0$
- Decide v_2 if $s(x) < 0$

The equation $s(x) = 0$ identifies the decision surface that can separate the points which are assigned to v_1 from the points which are assigned to v_2. When $s(x)$ is linear, the surface is called a hyperplane. If x_1 and x_2 are both on the decision surface then

$$v^t x_1 + v_0 = v^t x_2 + v_0 \tag{2}$$

By rearranging the above equation

$$v^t x_1 + v_0 - v^t x_2 - v_0 = 0 \tag{3}$$

$$v^t (x_1 - x_2) = 0 \tag{4}$$

Equation (4) concludes that v is normal to any vector present in the hyperplane. In simple words the hyperplane H divides the space into two regions. Region R_1 for v_1 and R_2 for v_2.

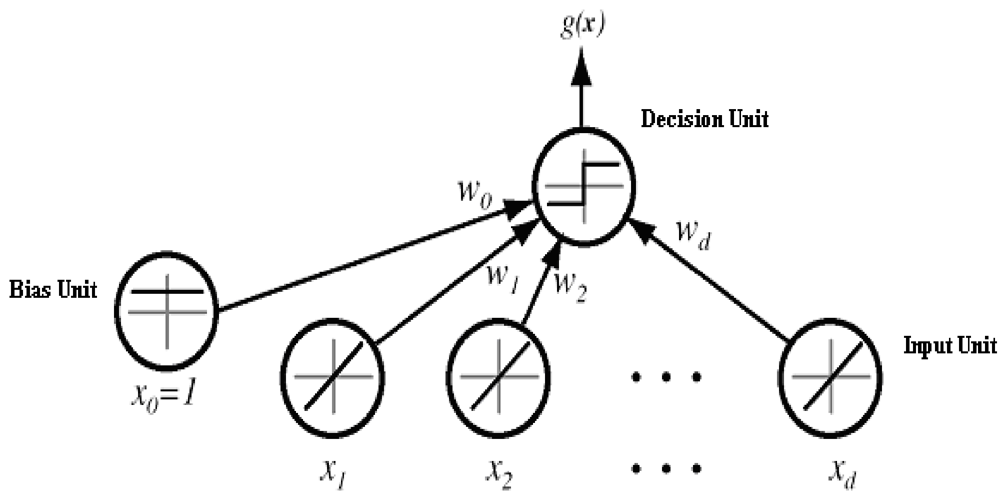


Figure 2.1: Simple Linear Classifier [5]

A simple linear classifier with d inputs is shown in fig 2.1 [5]. Each input value x_i is multiplied by the corresponding weight v_i. So sum of all the products $\sum v_i x_i$ is the final input to the Decision Unit. In a multi category case, the number of regions will increase as depicted in figure 2.2 [5].

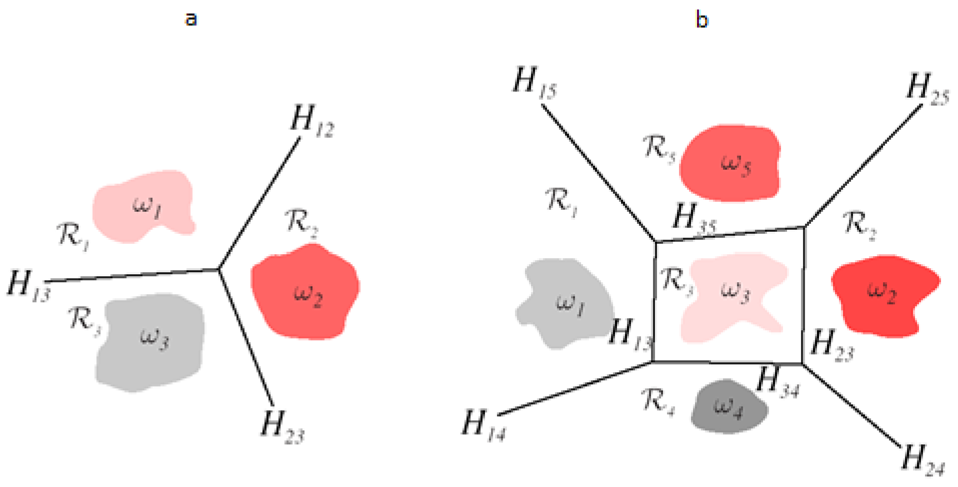


Figure 2.2: Decision boundaries (a) three class problem (b) five class problem

2.2. Correlation Filters

Correlation filters are commonly used for image processing and pattern recognition purposes. There are many applications available where these filters have been successfully applied [6, 7, 8]. A correlation filter must satisfy the following three purposes for pattern recognition:

- It is invariant to out-of- plane, in-plane and scale mismatch.
- Able to discriminate between objects in a multi class pattern recognition environment.
- Shows good tolerance to clutter and noise.

Some correlation filters are discussed in the next sections with their properties.

2.2.1. Matched Filter

The Matched Filter (MF) is optimal in detecting a target in white noise and provides a high Signal to Noise Ratio (SNR). If $x(t)$ is a waveform then a matched filter has the impulse response:

$$h(\tau) = Kx(\Delta - \tau) \qquad \textbf{(5)}$$

K and Δ are any arbitrary constants. Fourier transform of this impulse response gives the transfer function.

$$H(j2\pi f) = \int_{-\infty}^{\infty} h(\tau)e^{-j2\pi f\tau}d\tau \qquad \textbf{(6)}$$

Now replacing the value of $h(\tau)$ in equation 6.

$$H(j2\pi f) = \int_{-\infty}^{\infty} Kx(\Delta - \tau)e^{-j2\pi f\tau}d\tau \qquad \textbf{(7)}$$

Now rearranging equation (7)

$$H(j2\pi f) = K\int_{-\infty}^{\infty} x(\Delta - \tau)e^{-j2\pi f\tau}d\tau \qquad \textbf{(8)}$$

Simplifying equation (8)

$$H(j2\pi f) = Ke^{-j2\pi f\Delta}\int_{-\infty}^{\infty} x(\tau')e^{-j2\pi f\tau'}d\tau' \qquad \textbf{(9)}$$

Where $\tau' = \Delta - \tau$ is taken as the substitution. Now the Fourier transform of $x(t)$ is

$$X(j2\pi f) = \int_{-\infty}^{\infty} x(t) e^{-j2\pi ft} dt \qquad \textbf{(10)}$$

Now comparing equations (9) and (10)

$$H(j2\pi f) = KX^{*}(j2\pi f) e^{-j2\pi f\Delta} \qquad \textbf{(11)}$$

MF is also called the conjugate filter.

MF fails in the presence of distortions such as in-plane, out-of-plane rotation and scale mismatch. Matched filter cannot be used for multi-class pattern recognition applications. Also a large number of filters will be required when handling large number of distortions. Another drawback of MF is its low Horner Efficiency (Optical Efficiency) [11, 12, 13]. Horner Efficiency is given by:

$$\eta_H = \frac{C_{(0)}^2}{S^2} \qquad \textbf{(12)}$$

Where $C_{(0)}^2$ is the peak correlation energy and S^2 is the total energy of light in the input plane.

2.2.2. Phase Only Filter

The performance of the cross-correlation is improved significantly using the Phase Only Filter [11] (POF) as compared to the MF. The Phase correlation is given by equation 13.

$$r(f,g) = F^{-1}\left(\frac{F(f)^{*}.F(g)}{\left|F(f)^{*}.F(g)\right|} \right) \qquad \textbf{(13)}$$

The POF yields sharp peaks hence better object detectability. POF has 100% Horner Efficiency, as the filter passes the entire incident light. POF is slow to implement as it is difficult to perform using optical methods and requires a significant increase in operations per point on a CPU. The phase information makes it very sensitive and greatly reduces its ability to detect distorted signals [14].

2.2.3. Synthetic Discrimination Function Filter (SDF)

SDF [15, 16, 17, 18] has been the main area of focus for many researchers. It has been applied successfully in a number of applications [19, 20, 21, 22]. SDF was designed to overcome the limitations of MF and POF. The main idea behind the SDF design is to include the expected distortions in the filter design. This technique improved the immunity to the distortions. This methodology also addressed the multi class pattern recognition problem by including the non-targets in the filter design.

The conventional SDF is synthesized using a weighted linear combination of distorted reference images. A composite image is created, hence giving the name as a Composite Filter. The composite image is cross-correlated with the input images to produce equal correlation peaks for all the images of the same class. Soon after its invention it was recognized that it was not optimized for the clutter tolerance.

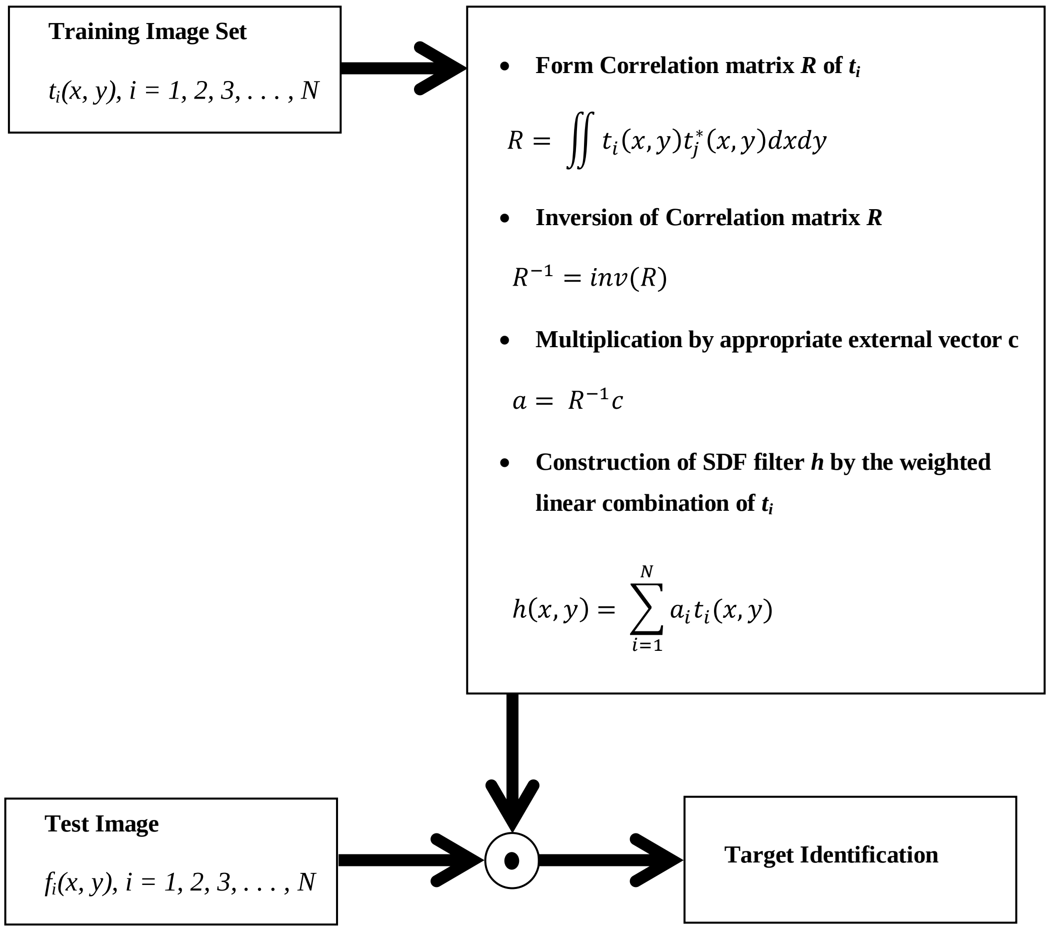
Let $h(x,y)$ denotes the composite image and $t_i(x,y)$ is the training image set where $i = 1,2,3,\dots,N$ and N is the number of the training images. Then the value at the origin of the correlation plane between the composite image and each of the training images is assumed to be equal to a constant c.

The composite image is given in terms of the training images and weights.

$$h(x,y) = \sum_{i=1}^{N} a_i t_i(x,y)$$

Where the coefficients $a_i (i = 1, 2, \ldots, N)$ are determined to satisfy the constraint c.

$$\sum_{i=1}^{N} a_i R_{ij} = c$$



Training Image Set
t$t_i(x, y), i = 1, 2, 3, \ldots, N$
Form Correlation matrix *R* of t_i
$R = \iint t_i(x,y) t_j^*(x,y) dxdy$
Inversion of Correlation matrix *R*
$R^{-1} = inv(R)$
Multiplication by appropriate external vector c
$a = R^{-1} c$
Construction of SDF filter *h* by the weighted linear combination of t_i
$h(x,y) = \sum_{i=1}^{N} a_i t_i(x,y)$
Test Image
$f_i(x, y), i = 1, 2, 3, \ldots, N$
Target Identification

igure 2.3: SDF Synthesis Process

2.3. Performance Measures for Correlation Filters

To measure the performance of the correlation filters some basic measures have to be calculated. The basic performance measure is correlation output peak intensity (COPI). It signifies the maximum intensity value of the correlation output plane. It is defined as [33]:

$$COPI = \{max\,|(C(x,y)|\}$$

C is the output Correlation Intensity. A filter with high COPI shows good performance and a high detection ability.

Another important performance measure is Peak to correlation energy (PCE). The basis of the PCE is that the COPI should be as high as possible while at the same time the overall correlation plane energy should be as low as possible. It is defined as [33]:

$$PCE = \frac{COPI}{Energy_c}$$

Where $Energy_c$ is the total correlation plane energy and is defined as:

$$Energy_c = \sum |C(x,y)|^2$$

Chapter 3

DIFFERENCE OF GAUSSIANS FILTER

Difference of Gaussians is an edge enhancement algorithm for grayscale images that performs the enhancement by subtracting one blurred version of a grayscale image from another less blurred version of the same image. The blurred versions of the images are obtained by correlating the original image with the Gaussian kernels with different standard deviations. High frequency information is suppressed because of blurring the image with the Gaussian kernel. When one blurred image is subtracted from the other blurred version of the same image, spatial information that is preserved is the one that lies between the ranges of frequencies that are preserved during the blurring of image. Thus, the difference of Gaussians is a bandpass filter that preserves only some of spatial frequencies that were present in the original grayscale image.

Difference of Gaussians is defined as:

$$f(x,y) = f_1(x,y) - f_2(x,y)$$

$$f(x,y) = \frac{1}{2\pi\sigma_1^2} e^{[-\frac{x^2+y^2}{2\pi\sigma_1^2}]} - \frac{1}{2\pi\sigma_2^2} e^{[-\frac{x^2+y^2}{2\pi\sigma_2^2}]}$$

Where $(\sigma_1,\ \sigma_2)$ are the standard deviations of the two Gaussian functions.

The value of the band pass maximum frequency response should be chosen to give the best compromise between intra-class distortion tolerance and interclass discrimination of the resulting filter. The low frequencies must be reduced to enhance the discrimination ability of the filter. The higher frequencies must be reduced enough to give adequate target distortion tolerance. The best performance of the DOG filter occurs at its closest approximation to the Mexican hat wavelet when the ratio of σ_1 to σ_2 is 1.6 [23].

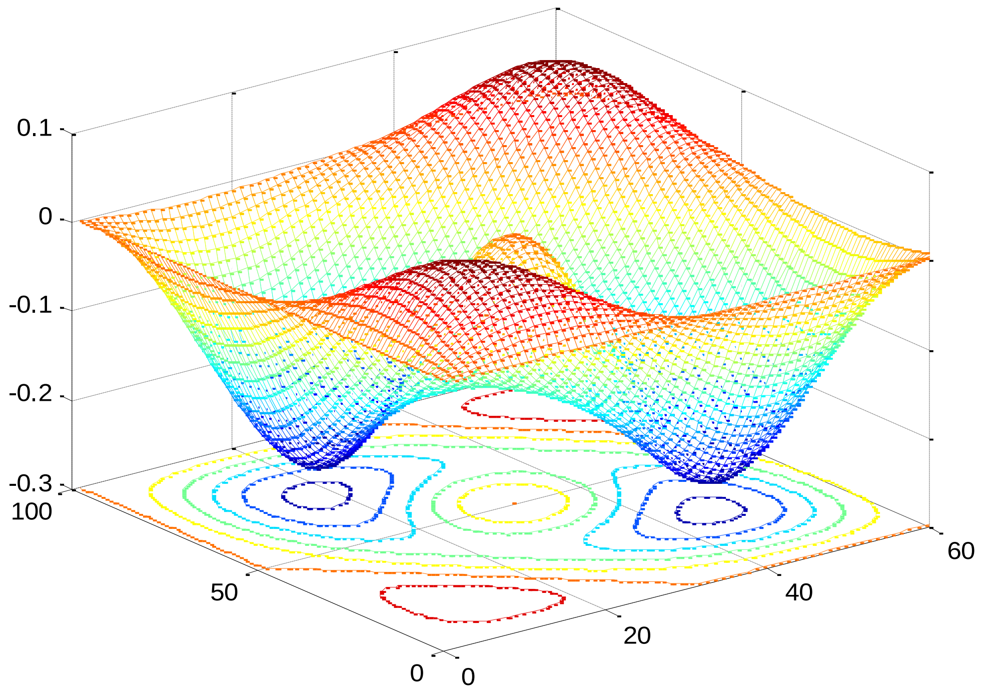


Figure 3.1: Mesh plot for Difference of Gaussian wavelet with contour

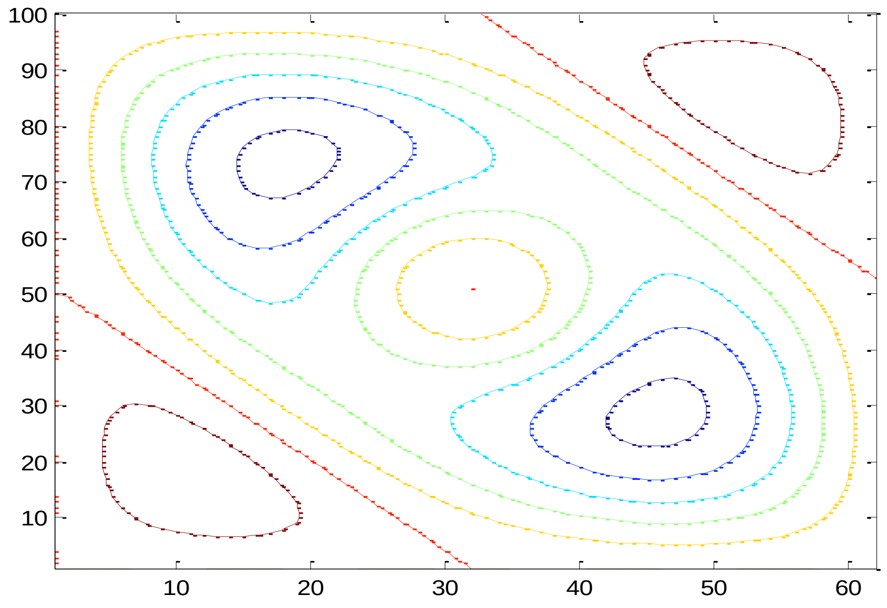


Figure 3.2: Contour plot for Difference of Gaussian wavelet

Chapter 4

DIGITAL SYSTEM DESIGN WITH FPGAS

FPGAs (Field Programmable Gate Arrays) are now widely used components that can contain very complex systems on a single chip, and in this way enable the user to obtain a very short time-to-market. FPGAs find applications in a wide range of areas including digital signal processing, image processing, aerospace and defense systems, software-defined radio, medical imaging, ASIC prototyping, pattern recognition systems, computer vision, bioinformatics, cryptography and radio astronomy.

Due to the technological advancements, FPGAs are now capable to implement more complex logic and arithmetic functions. This has drawn the attention of Digital Signal Processing (DSP) designers. And the FPGAs are being widely used for DSP applications. With the FPGAs, having digital signal processing capabilities, designers can now easily offload computationally intensive digital signal processing functions from the processor [24].

This chapter gives an introduction to the digital system design and describes the design of FPGA based hardware systems. A brief explanation of the fixed point arithmetic is given in this chapter. The use of FPGA available resources (for instance dedicated multipliers) is also described.

4.1. Components of Digital System Design Process

Three main components of the digital design process are Design, Implementation and Verification.[25]

4.1.1. Design

‘Design’ is the most important part in a digital system design process. A top level design of the system is partitioned into its components. Each of the components is then defined at the register transfer level (RTL). This is a level of abstraction where the digital designer specifies all the registers used in the system and give details of how data flows among these registers. The combinational logic between any two sets of registers is usually described using high level mathematical operations, which is drawn as a cloud.

4.1.2. Implementation

When the system components have been described at RTL level, the implementation of system is then a straightforward translation into an HDL program. HDL based designs are easier to debug than the schematics. Modern designs are programmed using hardware description languages like Verilog or VHDL, because it takes significantly less time to write an HDL code and synthesize a gate level realization of a large circuit. The saved time can be put to the other parts of the design process. The HDL program is then synthesized to map the system on an FPGA or ASIC implementation. Synthesis tools create an optimized internal representation of the system before mapping it to the target technology.

4.1.3. Verification

As the number of gates on silicon chips is increasing, so do the challenges of verification. Verification is a critical part in VLSI design as there is hardly any tolerance for bugs in the hardware. With the application specific integrated circuits, a bug may require a re-spin of fabrication, which is expensive, so it is important for an ASIC to be ‘right first time’. Even bugs found in FPGA based designs result in extended design cycles.

4.2. Design objectives of a Digital System

To achieve an effective design, a designer is needed to explore the design space for tradeoffs of competing design objectives. The following are some of the most critical design objectives the designers are needed to consider: [25]

- Area
- Critical path delays
- Testability
- Power dissipation.

The art of digital design is to find the optimal tradeoff among these objectives. These objectives are competing because, for example, if the designer tries to minimize area then the design may result in longer critical paths and may also affect the testability of the design. Similarly, if the design as synthesized for better timing means shorter critical paths, the design may result in a larger area. Better timing also means more power dissipation, which depends directly on the clock frequency.

4.3. Logic Synthesis

System is designed using a synthesis tool. A synthesis tool is a compiler that can translate the Verilog code into a gate level description of the system. Only a subset of Verilog constructs is synthesizable. The synthesizable part of Verilog is called 'RTL Verilog'. All the other constructs are 'non RTL Verilog'. These constructs are helpful in verification, testing, and simulation. It is imperative for the designer to know at the register transfer level what is being coded in the design. The RTL indicates the placement of registers in the design and the flow of data among these registers. The complete Verilog is a combination of RTL and non RTL constructs. A good hardware designer must have sound understanding of these differences and comprehensive command of RTL Verilog constructs. The programmer must also have a comprehension of the design to be coded in RTL Verilog.

The code written in RTL Verilog is synthesized for gate level implementation. The synthesis process takes the RTL Verilog and translates it into an optimized gate level net list. For logic synthesis the user specifies design constraints and the target technology in the form of a standard cell library. The library has standard basic logic gates such as AND and OR, or macro cells like adders, multipliers, flip flops, multiplexers and so on. The tool completely converts the design described in RTL hardware description language into a design that contains standard cells.

To optimally map the high level description into real HW, several steps are performed by the synthesis tool. In a typical flow of synthesis, RTL description is first converted into a non-optimized Boolean logic. Then several transformations are applied to optimize the logic subject to user constraints. This optimization is independent of the target technology. Finally, the tool maps the optimized logic to the technology specific standard cells.

4.4. Design Partitioning

The partitioning of a digital design into a number of modules is important. A module should be neither too small nor too large. Where possible, the design should be partitioned in such a way that module boundaries are residing at the register outputs. This will make it easier to synthesize the top level module or hierarchical synthesis at any level with timing constraints. The designer should also make sure that no combination cloud is present at the module boundaries. This gives the synthesis tool more leverage to generate optimized logic.

4.5. System Level Design Flow

Algorithm development is one of the most important steps in system design. Algorithms are developed using tools such as MATLAB, Simulink or C/C++/C#, or in any high level language. Functionally meeting R&S is a major consideration when the designer selects an algorithm out of several options. For example, in pattern matching the designer makes an intelligent choice out of many techniques including ‘chamfer distance transform’, ‘artificial neural network’ and ‘correlation based matching’. [25]

The developer must keep in mind the ultimate implementation of the algorithm on an embedded platform consisting of ASICs, FPGAs and DSPs. To ease design partitioning on a hybrid embedded platform, it is important for a system designer to define all the components of the design, clearly specifying the data flow among them. A component should implement a complete entity with defined functionality in the design. It is quite pertinent for the system designer to clearly define inputs and outputs and internal variables. The program flow should be defined as it will happen in the actual system. For example, with hard real time signal processing systems, the data is processed on a block by block basis. In this form, a buffer of input data is acquired and is passed to the first component in the system. The component processes this buffer of data and passes the output to the component next in execution order.

Alternatively, in many applications, especially in communication receiver design, the processing is done on a sample by sample basis. In these types of application the algorithmic implementation should process data sample by sample. Adhering to these guidelines will ease the task of HW/SW partitioning, co design and co-verification. The design is sequentially mapped from high level behavioral design to embedded system partitioning in HW mapped on ASICs or FPGAs and SW running on embedded DSPs or microcontrollers. It is important for the designers in the subsequent phases in the design cycle to stick to the same components and variable names as far as possible. This greatly facilitates going back and forth in the design cycle while the designer is making refinements and verifying its functionality and performance.

4.6. Fixed-point versus Floating-point Hardware

A digital signal processing algorithm can be implemented using fixed or floating point format. In floating point format, a numbers is stored in terms of mantissa and exponent.

Hardware that supports the floating point format, after executing each computation, automatically scales the mantissa and updates the exponent to make the result fit in the required number of bits in a defined way. Because of these operations floating point HW are more expensive in terms of area and power than fixed point HW. [25]

After executing a computation, the fixed point hardware does not specify the position of the decimal point and this responsibility is left to the programmer. The decimal point is fixed for each variable and it is predefined. By fixing the point, a variable can take only a fixed range of values. As the variable is bounded, if the result of a calculation falls outside of this range the data is lost or corrupted. This is known as overflow. There are various solutions for handling overflows in fixed point arithmetic. Handling overflows requires saturating the result to its maximum positive or minimum negative value that can be assigned to a variable defined in fixed point format. This results in reduced accuracy or performance. The programmer can fix the places of decimal points for all variables such that the arrangement prevents any overflows. This requires the designer to perform testing with all the possible data and observe the ranges of values all variables take in the simulation. Knowing the ranges of all the variables in the algorithm, programmer can determine the decimal point that avoids overflow very trivially.

The implementation of a signal processing and communication algorithm on a fixed point processor is a straight forward task. Owing to their low power consumption and relative cheapness, fixed point DSPs are very common in many embedded applications. Whereas floating point processors normally use 32 bit floating point format, 16 bit format is normally used for fixed point implementation. This results in fixed point designs using less memory. On chip memory tends to occupy the most silicon area, so this directly results in reduction in the cost of the system. Fixed point designs are widely used in multimedia and telecommunication solutions.

4.7. $Q_{n.m}$ Format for Fixed-point Arithmetic

Most signal processing applications are first implemented in double precision floating point arithmetic. An example of the tools being used is MATLAB. While implementing these algorithms, the main focus of the developer is to correctly integrate the functionality of the algorithm. This MATLAB code is then converted into floating point C/C++ code. C++ code usually runs much faster than MATLAB routines. This code conversion also gives more understanding of the algorithm as the designer might have used several functions from MATLAB toolboxes. Their understanding is critical for porting these algorithms in SW or HW for embedded devices. After getting the desired performance from the floating point algorithm, this implementation is converted to fixed point format. For this the floating point variables and constants in the simulation are converted to $Q_{n.m}$ fixed point format. $Q_{n.m}$ has a fixed position for decimal point.

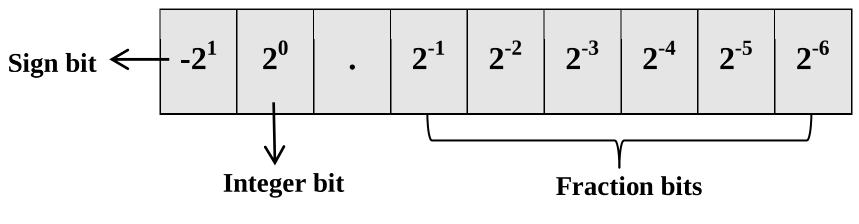


Figure 4.1: Fields of bits and their equivalent weights for the text example

A two's complement fixed point number in $Q_{n.m}$ format with equivalent floating point value:

$$-b_{n-1}.2^{n-1} + b_{n-2}.2^{n-2} + \ldots + b_1.2^1 + b_0 + b_{-1}2^{-1} + b_{-2}.2^{-2} + \ldots b_{-m}.2^{-m} \qquad (1)$$

4.7.1. Qn.m format Addition

Addition of fixed point numbers *a* and *b* of formats $Q_{n1.m1}$ and $Q_{n2.m2}$, respectively, results in a $Q_{n.m}$ format number, where n is the larger of n1 and n2 and m is the larger of m1 and m2. Although the decimal is implied and does not exist in HW, the designer needs to align the location of the implied decimal of both numbers and then appropriately sign extend the number that has the least number of integer bits. As the fractional bits are stored in least significant bits, no extension of the fractional part is required. The example below illustrates Q format addition. **Example**: Add two signed numbers a and b in $Q_{2.2}$ and $Q_{4.4}$ formats. In $Q_{2.2}$ format *a* is 1110, and in $Q_{4.4}$ format b is 0111_0110. As n1 is less than n2, the sign bit of a is extended to change its format from $Q_{2.2}$ to $Q_{4.2}$ (Figure 4.2).

$Q_{n1.m1}$	**1**	**1**	**1**	**1**	**1**	**0**			**= $Q_{4.2}$ = -8+4+2+1+0.5+0 = -0.5**
$Q_{n2.m2}$	**0**	**1**	**1**	**1**	**0**	**1**	**1**	**0**	**= $Q_{4.4}$ = 1+2+4+0.25+0.125 = 7.375**
$Q_{n.m}$	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**= $Q_{4.4}$ = 2+4+0.5+0.25+0.125 = 6.875**

Figure 4.2: Example of addition in Q format

4.7.2. Qn.m format Multiplication

If two numbers a and b in, respectively, $Q_{n1.m1}$ and $Q_{n2.m2}$ formats are multiplied, the multiplication results in a product in $Q_{(n1 + n2)(m1 + m2)}$ format. If the numbers are two's complement signed numbers, we get a redundant sign bit at the MSB position. This redundant bit could be removed by left shifting the product by 1, and the format of the product is changed to $Q_{(n1 + n2 - 1).(m1 + m2 - 1)}$.

4.8. Xilinx IP Cores

Xilinx Intellectual Property (IP) cores are the building blocks of designs targeted for Xilinx Design Platforms. A large number of IP cores are available to address the needs of the designers. Use of Xilinx IP cores save precious design time and resources that is normally spent on designing important functions, instead of focusing on the aspects of the design that differentiates the design from the competition. [26]

4.8.1. FPGA Block Memory

The Xilinx IP Block Memory Generator core is an advanced memory constructor that can generate performance and area optimized memories using embedded block RAMs available with the Xilinx FPGAs. Users can easily create optimized memories through the core generator tool to get the advantage of available memory resources in FPGAs.

The Block Memory Generator core is used to create customized memories to suit any application. Typical applications include:

- Single-port RAM: Processor scratch RAM, look-up tables
- Simple Dual-port RAM: Content addressable memories, FIFOs
- True Dual-port RAM: Multi-processor storage
- Single-port ROM: Program code storage, initialization ROM
- Dual-port ROM: Single ROM shared between two processors/systems

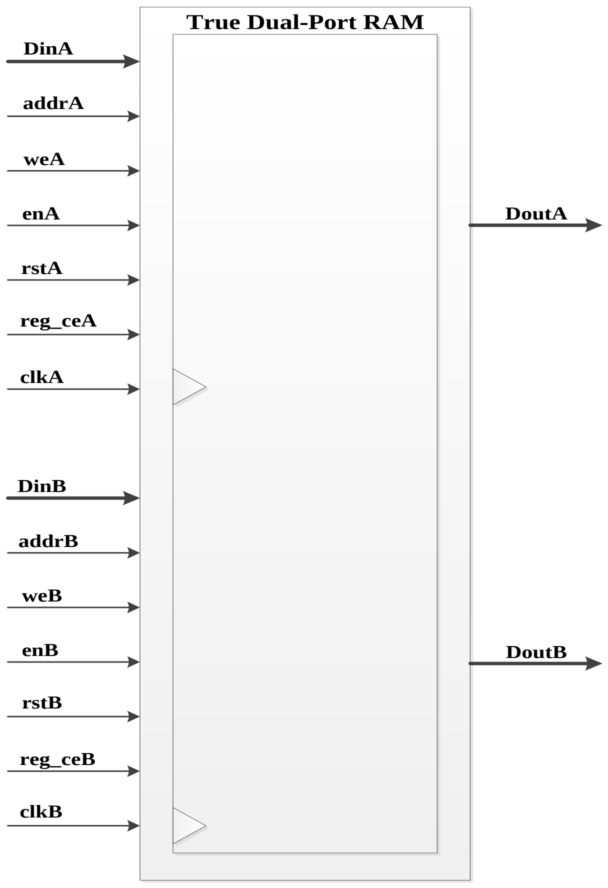


Figure 4.3: True dual port RAM

4.8.2. Multiplier IP Core

Multiplication is an essential operation in DSP Applications. Main design challenge for hardware engineers is to achieve maximum cock performance and implementation efficiency. Xilinx multiplier IP core abstracts away the device specifics of FPGAs and in this way simplify the design challenge for the designers, while maintains the required resource efficiency and the system performance. The multiplier core can generate parallel multipliers, as well as constant coefficient multipliers. Furthermore, it is possible for the designer to select the optimal solution with the help of resource estimation available with the core generator. Multiplier IP core provides the designer an option to choose between purely combinational and fully pipelined system for multiplication. Finally, fully pipelined multiplier provides maximum clock frequency of up to 450 MHz and uses DSP48 components. Available input data widths are 2 to 64 bits and could be used with Xilinx System Generator and Xilinx Core Generator.

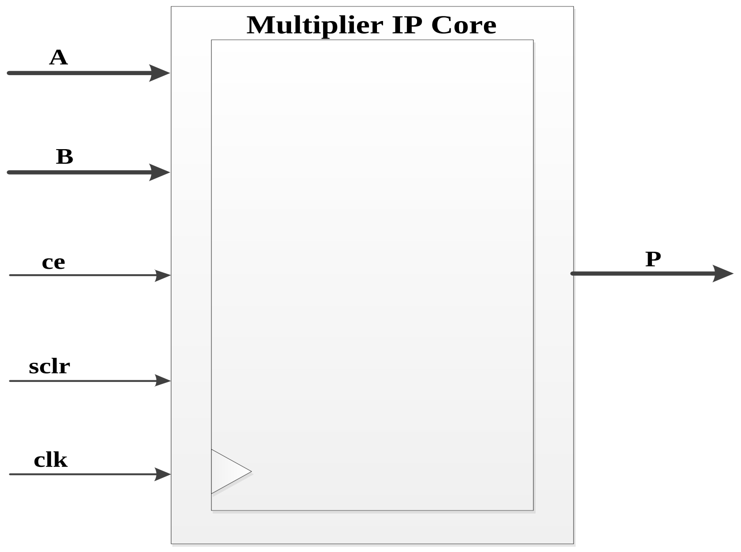


Figure 4.4: Multiplier IP core schematic symbol

Signal	Direction	Description
A[N-1:0]	*Input*	A operand input bus, N bits wide
B[M-1:0]	*Input*	B operand input bus, M bits wide (parallel multipliers only)
clk	*Input*	Rising-edge clock input
ce	*Input*	Active high Clock Enable
sclr	*Input*	Active high Synchronous Clear (sclr/ce priority is configurable)
P[X:Y]	*Output*	Product Output – bit X down to bit Y

Table 4.1: Multiplier IP core Signal Pinout

4.8.3. Divider IP Core

Division is the most complex of the four basic arithmetic operations. Available hardware solutions for division are very large and complex. The best solution is to minimize the number of divisions required for the algorithm.

Divider IP core supports two implementations for division:

- **Radix-2:** this is used for operand widths up to 16 bits.
- **High Radix**: used for operand widths greater than 16 bits. This implementation uses DSP slices and it is unavailable for the devices which do not have DSP slices.

Figure 4.5 shows the schematic symbol for high radix division.

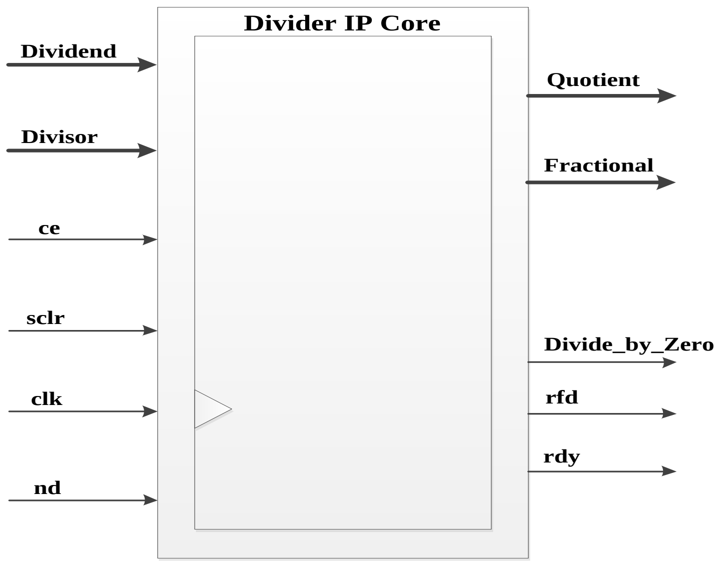


Figure 4.5: Schematic Symbol for Divider IP Core

Signal	**Direction**	**Description**
Dividend[M-1:0]	*Input*	Dividend input bus, M bits wide
Divisor[M-1:0]	*Input*	Divisor input bus, M bits wide
clk	*Input*	Rising-edge clock input
ce	*Input*	Active high Clock Enable
nd	*Input*	Signals the core that new data is present at the input
sclr	*Input*	Active high Synchronous Clear
Quotient[M-1:0]	*Output*	Signed integer part of the Quotient, M bits wide
Fractional[F-1:0]	*Output*	Unsigned fractional part of the quotient, F bits wide
Divide_by_Zero	*Output*	Detection of division by zero
rfd	*Output*	Signals that core is ready for new data
rdy	*Output*	Signals that a result is available at the output

Table 4.2: Divider IP Core Signal Pinout

4.8.4. Fast Fourier Transform IP Core

The Xilinx Fast Fourier Transform (FFT) IP core uses the Cooley-Tukey Fourier Transform algorithm, which is a computationally very efficient algorithm for the calculation of Discrete Fourier Transform (DFT).

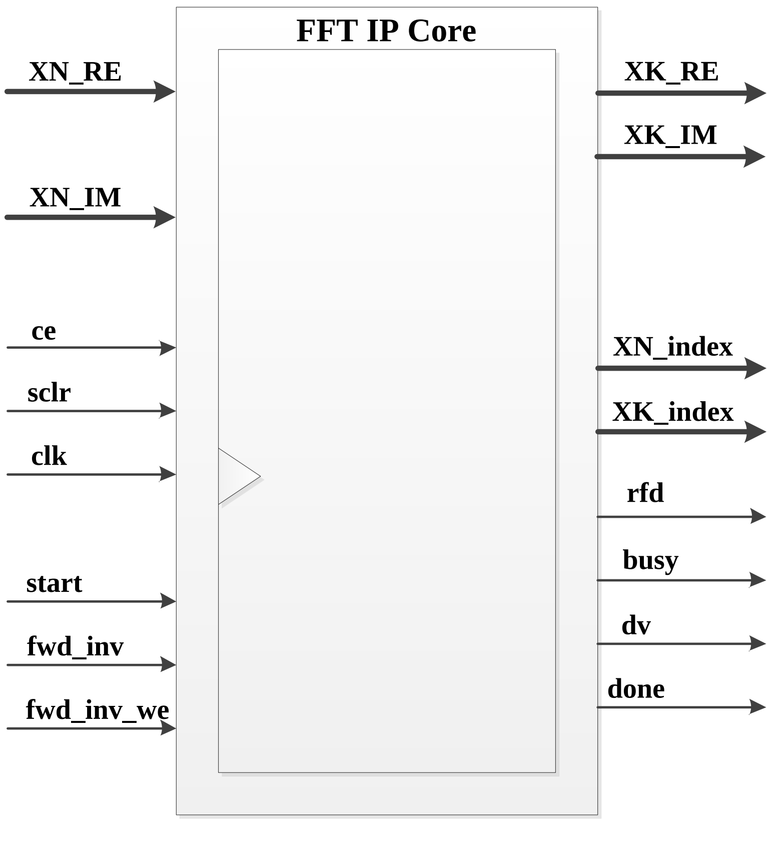


Figure 4.6: FFT IP Core schematic symbol

BUSY signal goes high while the core is busy with the calculation of transform. *DONE* is High when core has completed transform calculation.

	Direction	**Description**
XN_RE	*Input*	Input data bus: Real Component
XN_IM	*Input*	Input data bus: Imaginary Component
XK_RE	*Output*	Output data bus: Real Component
XK_IM	*Output*	Output data bus: Imaginary Component
XN_index	*Output*	Index of input data
XK_index	*Output*	Index of output data
clk	*Input*	Rising-edge clock input
ce	*Input*	Active high Clock Enable
sclr	*Input*	Master Synchronous Reset
start	*Input*	It is asserted to begin the data loading and transform calculation
fwd_inv	*Input*	It indicates if a forward FFT or an inverse FFT is performed.
fwd_inv_we	*Input*	Write enable for fwd_inv (Active High).
done	*Output*	FFT complete strobe (Active High): DONE transitions High for one clock cycle when the transform calculation has completed.
dv	*Output*	This signal is High when the valid data is presented at the output.
rfd	*Output*	Signals that core is ready for new data. rfd is High during the load operation.
busy	*Output*	Core activity indicator (Active High): This signal goes High while the core is computing the transform.

Table 4.3: FFT IP Core Signal Pinout

Chapter 5

HARDWARE DESIGN FOR SDF FILTER

This chapter describes the hardware design for **Out of Plane Invariant Synthetic Discriminant Function Bandpass Difference of Gaussian Composite Filter**. Hardware of the filter is explained using the RTL schematic diagrams for each part of the filter. Aim was to design a hardware which is fast and cheap. Hardware is parallelized where possible to make it fast and resources are reused to make it cost effective.

5.1. Design Stages

Literature survey was carried out for the SDF filter and a system model was designed using MATLAB as well as in System Verilog to identify the required building blocks for hardware design of the system and the system specifications. RTL model of the system was written using Verilog and the system simulation was carried out in ModelSim to check the result of each component of the system. System sub blocks were integrated into a top level module. A test-bench for the top level module was created to check the functionality of the system and the results of simulation were compared with the MATLAB results. Figure 5.1 shows the flow chart for the design methodology used.

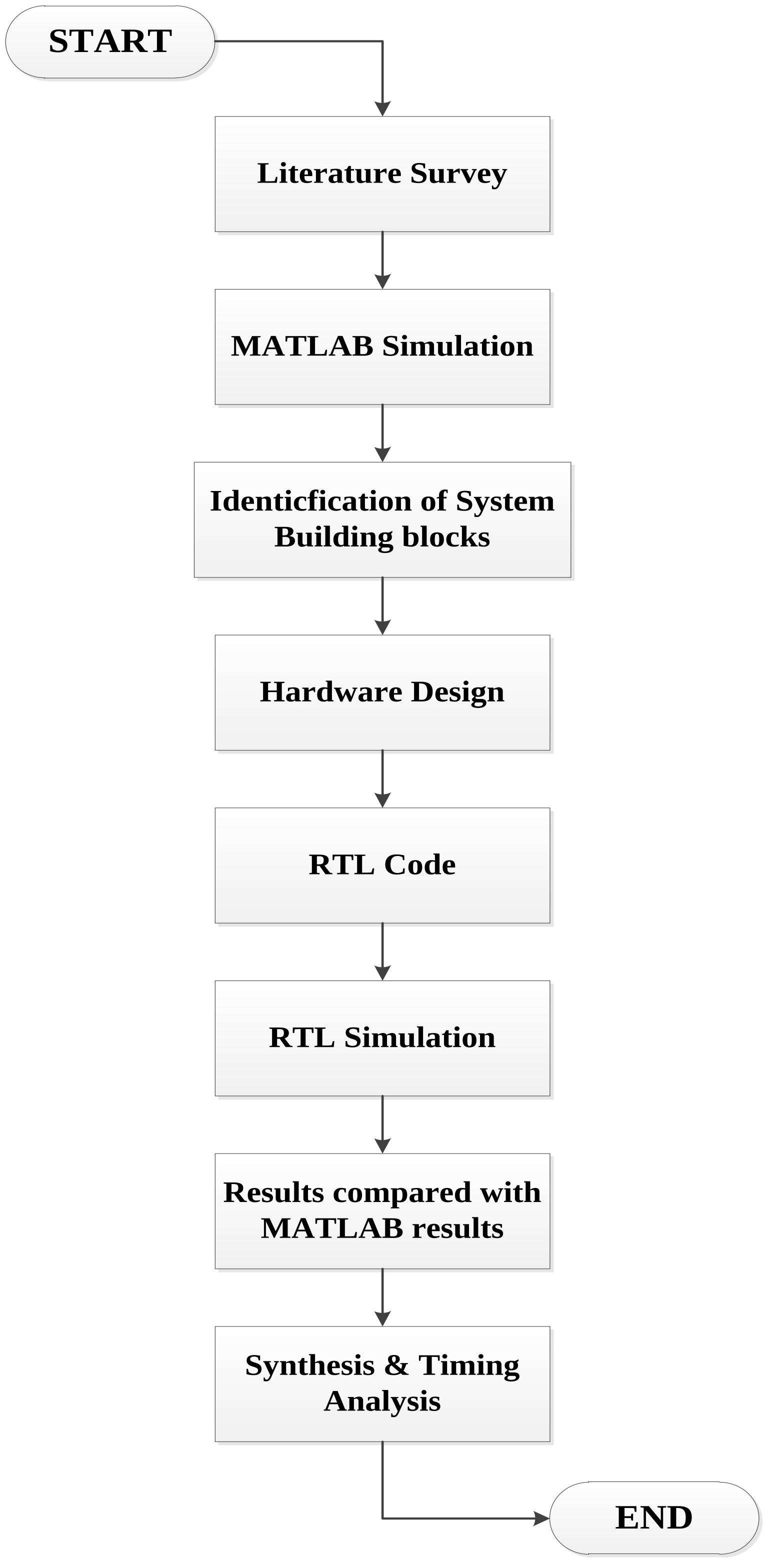

START
Literature Survey
MATLAB Simulation
Identicfication of System Building blocks
Hardware Design
RTL Code
RTL Simulation
Results compared with MATLAB results
Synthesis & Timing Analysis
END

Figure 5.1: Design Stages

Figure 5.2 is the block diagram for the out of plane invariant bandpass Difference of Gaussian SDF filter. Each part will be explained separately in the following sections.

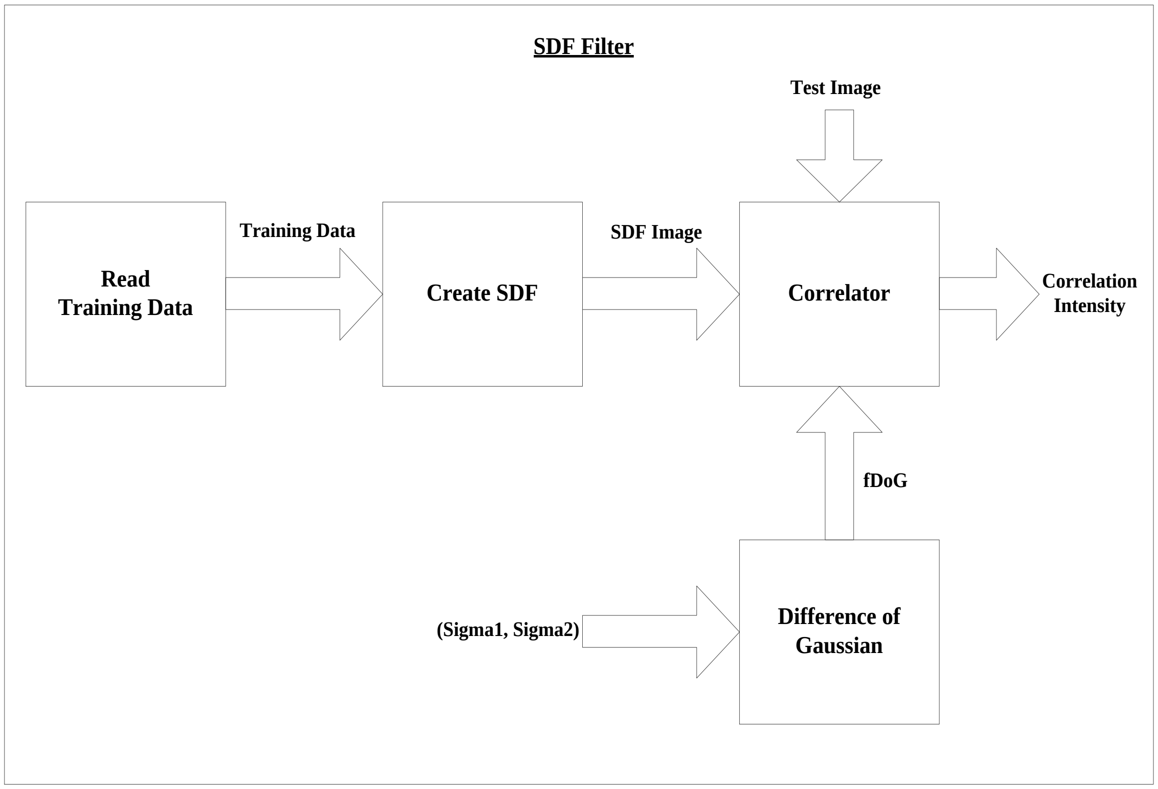


Figure 5.2: Structure of Bandpass DoG SDF Filter

Training images are taken using the installed camera or stored images are used to train SDF. BRAM is configured for the new training data. CreateSDF module reads training data from BRAM and creates the SDF image. Next the test image is read and Difference of Gaussians calculations are performed. Correlator calculates the correlation between test image and the Difference of Gaussians output and this output is correlated with the SDF image. Output of the Correlator is written to a text file and sent to MATLAB to plot the correlation intensity. Flow chart of the algorithm is shown in figure 5.3.

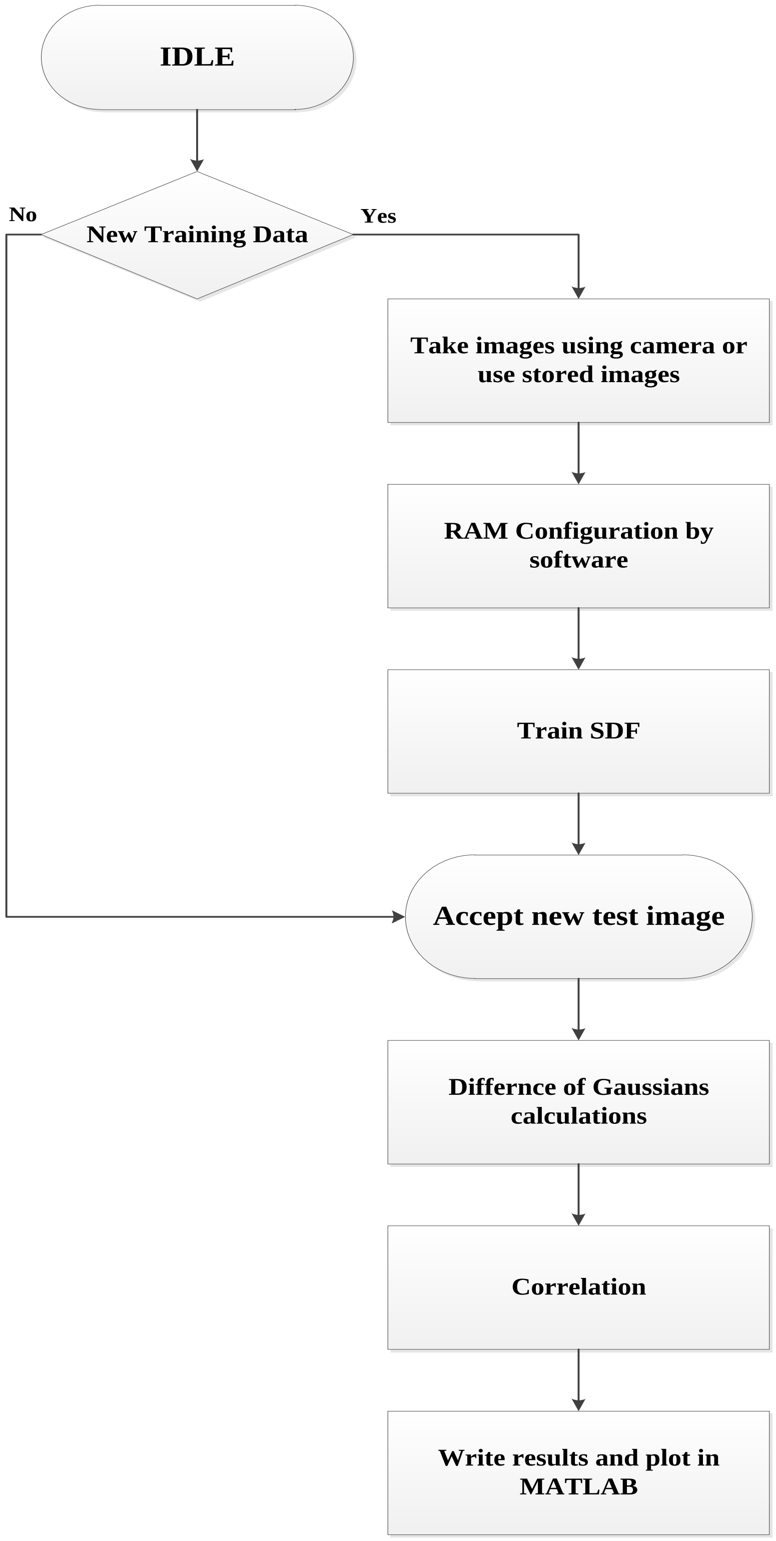

IDLE
New Training Data
No
Yes
Take images using camera or use stored images
RAM Configuration by software
Train SDF
Accept new test image
Differnce of Gaussians calculations
Correlation
Write results and plot in MATLAB

Figure 5.3: Flow chart for the Algorithm

5.2. Memory for training Data

We used Xilinx Block RAM for the training data. BRAM is initialized using COE file that was created using MATLAB. Input to this MATLAB function is the path to the directory containing the training images and output is the COE file that contains the training data.

```
function [out_file, out_directory] = create_TrainingData_coe(directory)

out_file = 'TrainingData.coe';
out_directory = directory;

initialDirectory = cd;
cd(directory);
imageNames = dir('*.bmp');
[N, dummy] = size(imageNames);
img = imread(imageNames(1).name);
H = size(img, 1);
W  = size(img, 2);
height = H * N;
width  = W;

s = fopen(out_file,'wb'); %opens the output file
fprintf(s,'%s\n','; VGA Memory Map ');
fprintf(s,'%s\n','; .COE file with hex coefficients ');
fprintf(s,'; Height: %d, Width: %d\n\n', height, width);
fprintf(s,'%s\n','memory_initialization_radix=16;');
fprintf(s,'%s\n','memory_initialization_vector=');

cf = 0;
for i = 1:N
    cnt = 0;
    im = imread( imageNames(i).name );
    for r=1:H
        for c=1:W
            cnt = cnt + 1;
            cf = cf + 1;
            Outbyte = dec2bin(im(r,c),8);
            if (Outbyte(1:4) == '0000')
                fprintf(s,'0%X',bin2dec(Outbyte));
            else
                fprintf(s,'%X',bin2dec(Outbyte));
            end

            if ( (i == N ) && (c == W) && (r == H))
                fprintf(s,'%c',';');
            else
                if (mod(cnt,32) == 0)
                    fprintf(s,'%c\n',',');
                else
                    fprintf(s,'%c',',');
                end
            end
        end
    end
end
fclose(s);  cd(initialDirectory);
```

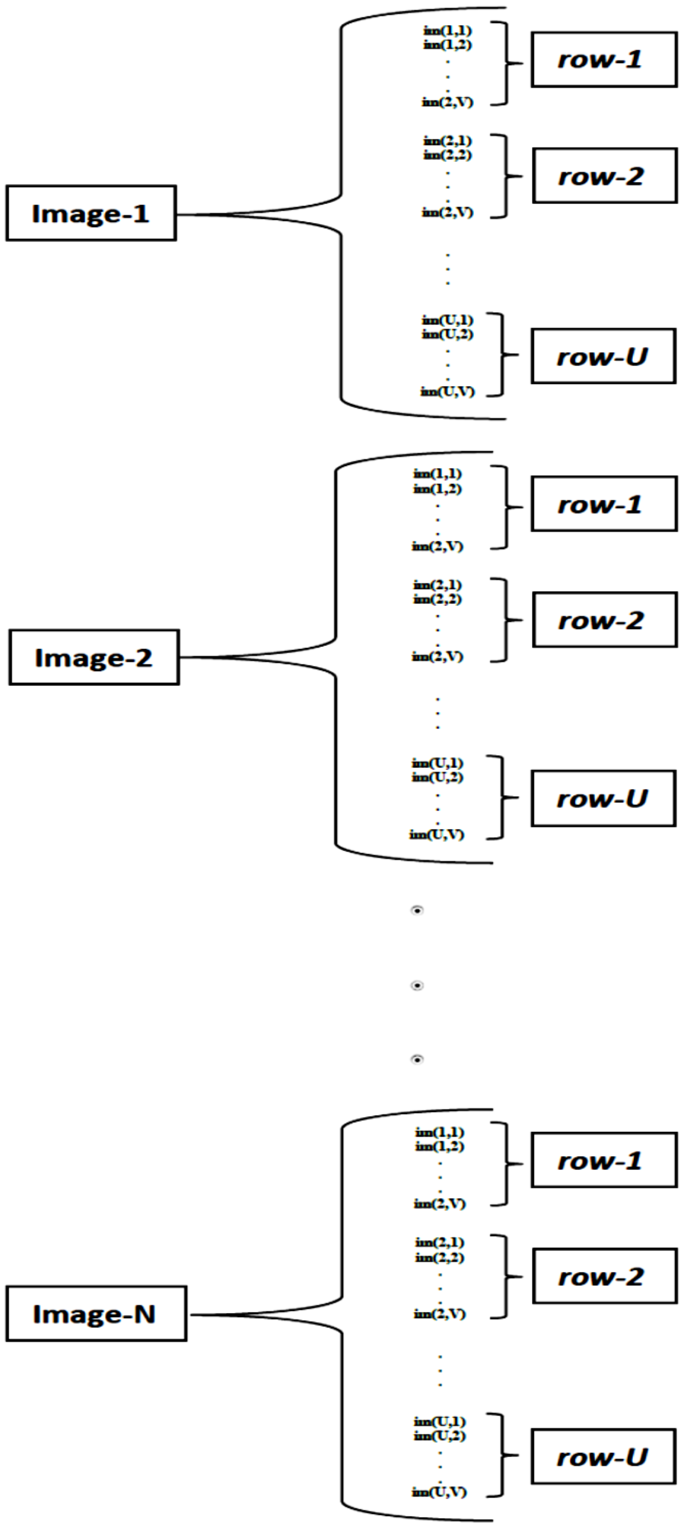
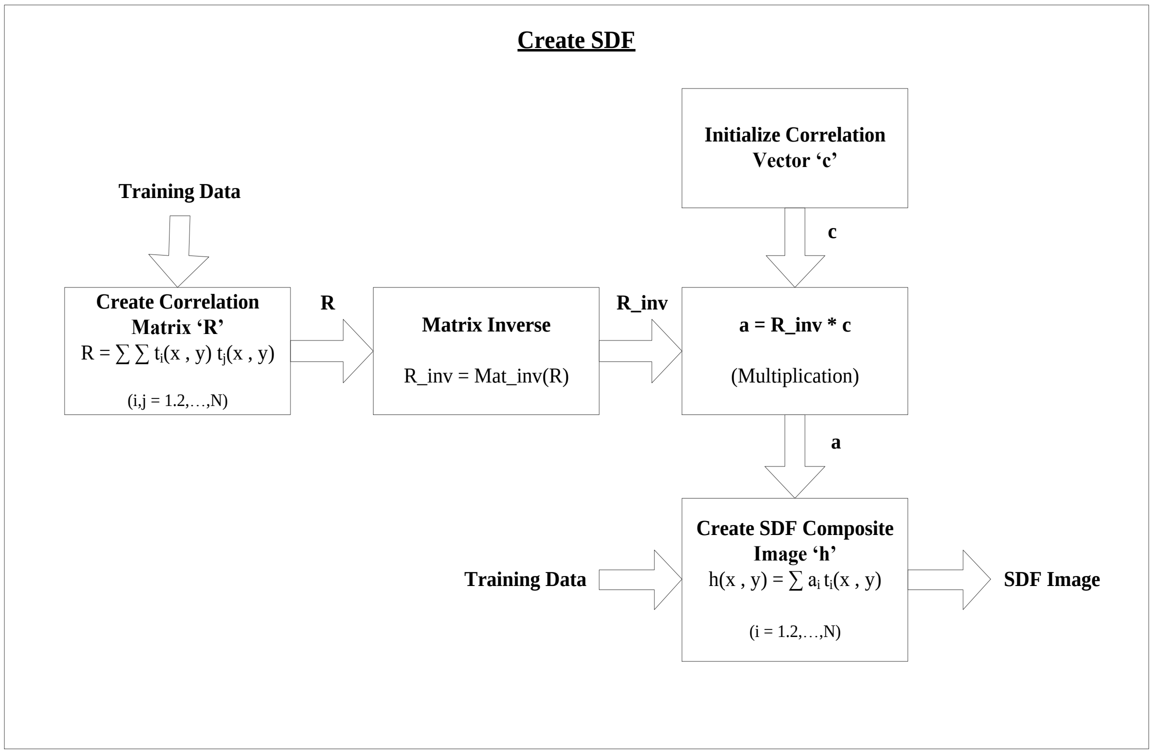


Figure 5.4: Memory Organization for Training data

Figure 5.4 shows how training data is organized in memory. ***'N'*** is the number of the training images used, ***'U'*** is the number of rows and ***'V'*** is the number of columns of an image.

5.3. Create SDF Image

Figure 5.5 shows the steps to create an SDF image.



Create SDF
Initialize Correlation Vector 'c'
Training Data
c
Create Correlation Matrix 'R'
R = ∑ ∑ t_i(x , y) t_j(x , y)
(i,j = 1.2,…,N)
R
Matrix Inverse
R_inv = Mat_inv(R)
R_inv
a = R_inv * c
(Multiplication)
a
Training Data
Create SDF Composite Image 'h'
h(x , y) = ∑ a_i t_i(x , y)
(i = 1.2,…,N)
SDF Image

Figure 5.5: CreateSDF Block Diagram

The SDF image is synthesized using weighted linear combination of distorted training or reference images. A composite image is created, hence giving the name as ***Composite filter***.

$$h(x, y) = \sum_{i=1}^{N} a_i t_i(x, y) \tag{1}$$

let h(x,y) denotes a composite image and training image is denoted by $t_i(x,y)$, where $i=1, 2, 3, \ldots, N$ and $\boldsymbol{N}$ is the number of training images used. Training images are used to create Correlation matrix $\boldsymbol{R}$, then inverse of this matrix is calculated which is used to find the coefficients vector $\boldsymbol{a}$**.** [22]

We represent the image data using a vector $\boldsymbol{x}$ of size $\boldsymbol{d} \times \boldsymbol{1}$, where $\boldsymbol{d}$ is the number of pixels in the image. This 1-dimensional image data can be obtained by lexicographic ordering of the image rows (just as we placed the images in the BRAM memory).

In equation (1) coefficients $\boldsymbol{a_i}$ are chosen to satisfy the following constraints [22].

$$\boldsymbol{h^T} x_j = c_j \ \ , \ j = 1, 2, \ \ldots \ , \boldsymbol{N} \tag{2}$$

where $\boldsymbol{T}$ denotes the transpose and c_j is a desired cross correlation output peak value. In vector form, we define the training image data matrix $\boldsymbol{X}$ as

$$\boldsymbol{X} = [\boldsymbol{x_1}, \ \boldsymbol{x_2}, \ \ldots \ \ , \boldsymbol{x_N}] \tag{3}$$

where the size of matrix $\boldsymbol{X}$ is $\boldsymbol{d} \times \boldsymbol{N}$. Then the SDF is the solution to the following optimization problem, [22]

$$min \ \boldsymbol{h^T} \ \boldsymbol{h}, \quad subject \ to \ \boldsymbol{X^T} \ \boldsymbol{h} \ = \ \boldsymbol{c} \tag{4}$$

The optimal solution is:

$$\boldsymbol{h} \ = \ \boldsymbol{X}(\boldsymbol{X^T X})^{-1} \boldsymbol{c}. \tag{5}$$

Once $\boldsymbol{h}$ is determined, we apply an appropriate threshold to the output of the cross correlation, which is the inner product of the test input image and the filter $\boldsymbol{h}$ and decide on the class of the test image. From equation ***(5)***, correlation matrix $\boldsymbol{R}$ is given by

$$\boldsymbol{R} = (\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X}) \quad \textbf{(6)}$$

Correlation matrix is of size ***N*N***. Elements of Correlation matrix corresponds to the all possible combinations of training images. Each training image is correlated with every other image in the training set and the results are stored as the elements of correlation matrix.

and from equations ***(1)*** and ***(5)***, coefficients vector $\boldsymbol{a}$ is,

$$\boldsymbol{a} = \boldsymbol{R}^{-\mathbf{1}} * \boldsymbol{c} \quad \textbf{(7)}$$

Now the hardware for each part of module Create_SDF will be explained.

5.3.1. Create Correlation Matrix

Two registers ***cnt_n1*** and ***cnt_n2*** hold the values which correspond to the image numbers which are being correlated. For example, if image 1 and image 2 are to be correlated, then ***cnt_n1*** and ***cnt_n2*** contain values 0 and 1 respectively.

BRAM is configured as dual port ROM so that we could read pixels of both the images being correlated in parallel.

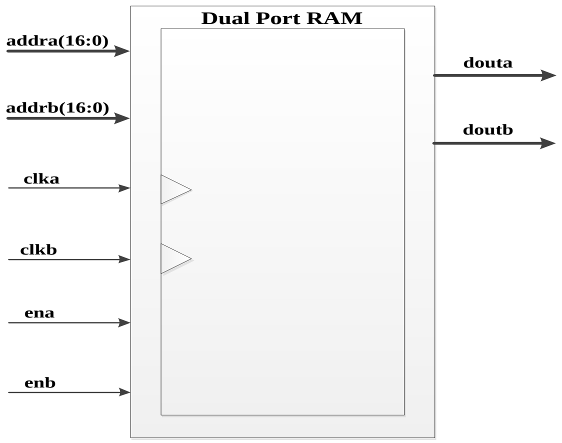


Figure 5.6: dual port RAM

Indexes to the memory locations to be read are:

$$addra = (cnt\_n1 \ * \ d) + mem\_addr$$

$$addrb = (cnt\_n2 \ * \ d) + mem\_addr$$

Here, ***d*** is the total number of pixels of an image and *mem_addr* is the output from a counter which is updated when pixels have been read and it is cleared when it reaches the value equal to the number of pixels of an image to restart counting the number of pixels for the next two images.

After reading pixel values from memory, they are multiplied and the result of multiplication is sent to accumulator. When *mem_addr* reaches the value equal to the number of pixels in an image, the output of the accumulator is sent to the port (*R_out*) and a flag (*done_Rout*) is set high. This flag tells the calling module that the result could now be written to the correlation matrix at the location specified by *cnt_n1* and *cnt_n2*. All the functionality is controlled using an FSM based control unit.

Figure 5.7 shows architecture for the hardware that creates correlation matrix R.

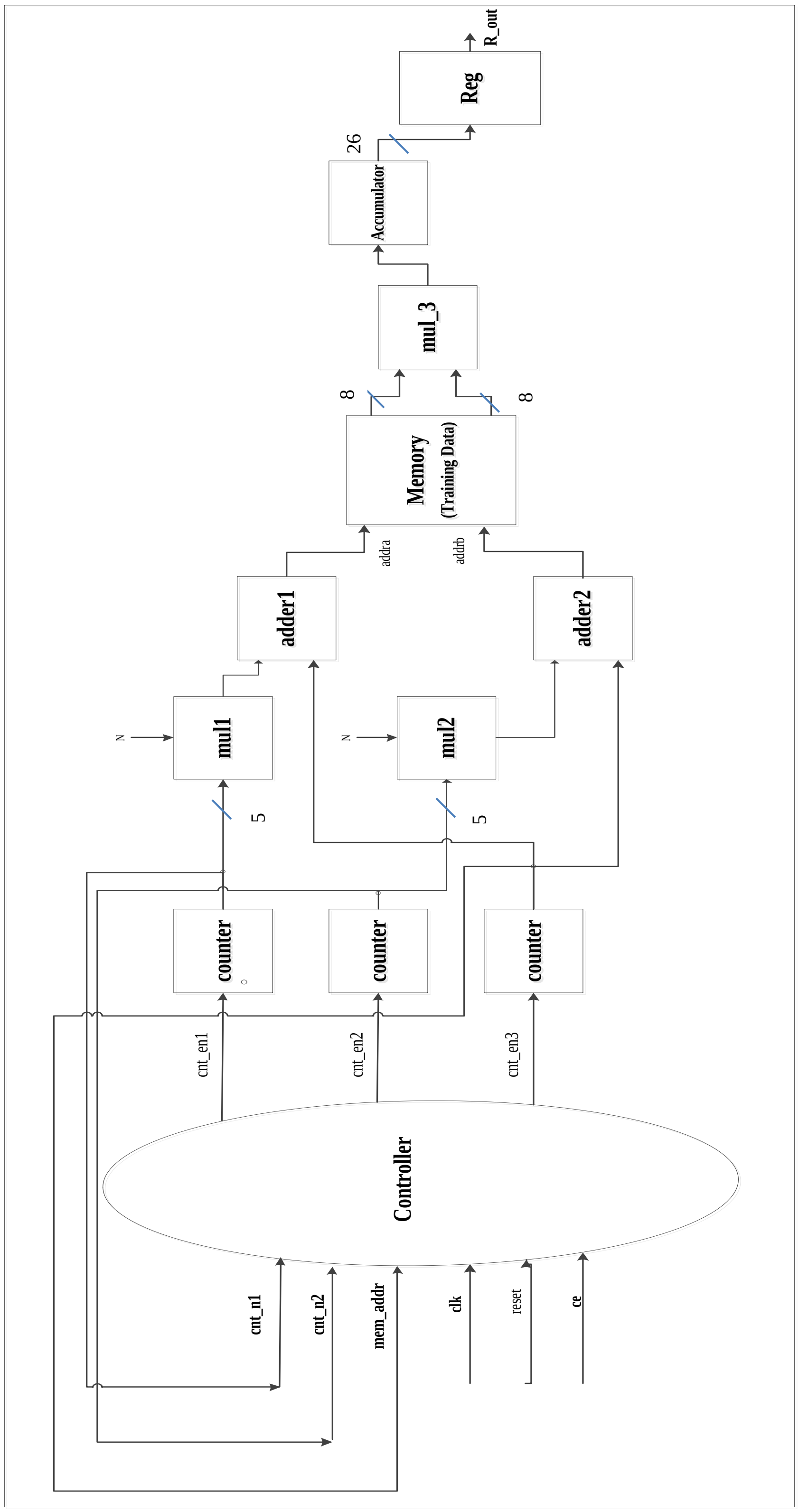


Figure 5.7: Hardware architecture for Correlation matrix creation

5.3.2. Matrix Inverse

The most computationally expensive operation in the design of SDF filter is the Matrix Inverse. Correlation matrix **R** is inverted, and the inverted matrix is multiplied with the correlation vector **c** to find the coefficient vector **a**. Matrix inversion includes the expensive division operations. It is highly required to reduce the number of divisions to improve performance and to reduce the resource usage as well. We used Xilinx ip core dividers in which DSP slices are used to improve the performance.

Some methods to perform matrix inversion in hardware are: Faddeev's algorithm [27][28][29], iterative methods[30], Given's rotations [31], and Gauss-Jordan elimination method. We are interested in a design using reconfigurable hardware with good performance and low cost, so we used Gauss Jordan elimination algorithm for matrix inversion. One important advantage of Gauss Jordan elimination method is that it requires only three different arithmetic operations, addition/subtraction, multiplication and division. All other algorithms involve more complex operations as well. For instance, the QRD Gram-Schmidt ortho-normalization method uses square root also, whereas the QRD Givens-Rotations uses sine and cosine operations [32].

The computational complexity of matrix inversion is an open question, popular sub-cubic software solutions such as the $O(n^{2.807})$ Strassen and the $O(n^{2.376})$ Coppersmith-Winograd based approaches, applied to invert $n \times n$ matrices, are of high theoretical interest, but not used in practice because the speed-up is only visible for very large matrices. Algorithmically simple methods such as Gauss-Jordan (GJ) elimination, although of higher complexity $O(n^3)$, are very important for developing practical architectural implementations.[32].

We designed a parameterizable hardware solution for the matrix inversion based on the GJ elimination algorithm, where the user can define the matrix-range. The system allows a matrix range up to 32. This range could easily be increased depending upon the user requirement. We achieved a small area hardware system for matrix inversion while maintaining the precision and performance aspects. We chose $\mathbf{Q_{26.24}}$ fixed point format for input to the matrix inversion system and $\mathbf{Q_{18.32}}$ format for the output of the system. MATLAB results were used to develop the system's precision study.

Gauss Jordan elimination method is used to invert a square matrix. This method is similar to Gaussian Elimination method but it does the elimination process above as well as below the diagonal on left hand side. In Gaussian elimination method matrix is converted to Row echelon form in which matrix contains zeros only below the diagonal. Going one step further by placing zeros above the diagonal converts the matrix to row reduced echelon form which is done in Gauss Jordan elimination method. Gauss Jordan method applies Gaussian elimination above and below the augmented $\boldsymbol{n\,X\,2n}$ matrix, where **n** is the size of the matrix.

In Gauss Jordan method, each row of the augmented matrix is divided by the diagonal element of that row, this requires $\boldsymbol{2n}$ divisions for each row. But as the divisor remains the same for an entire row, we performed only one division to get reciprocal of the divisor and multiplied all the elements of the row with that reciprocal. In this way, number of divisions greatly reduced and hence performance improved in terms of speed and area.

Figure 5.8 shows the schematic symbol for the Matrix Inversion system.

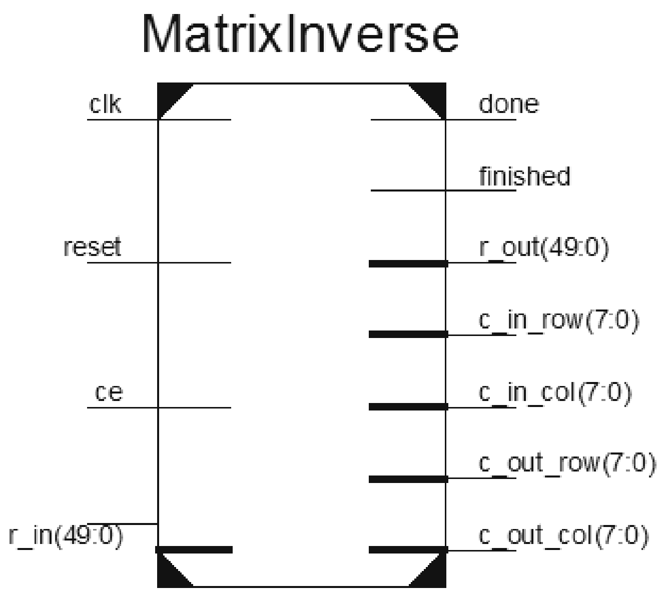


Figure 5.8: Schematic Symbol for Matrix Inversion system

System accepts the input in ***n*n-1*** clock cycles via the port ***r_in***. *Ports* ***c_in_row*** *and* ***c_in_col*** specify the row and column number of the input matrix. *Ports* ***c_out_row*** *and* ***c_out_col*** specify the row and column number of the output matrix. ***done*** signal tells the calling module that the matrix inversion has been performed, and the valid outputs are present at the port ***r_out***. ***fininshed*** signals that the system has done its work.

Flow chart in figure 5.9 shows that how matrix inversion is performed using Gauss Jordan elimination method. This chart also explains the structure of the control unit used in matrix inversion hardware.

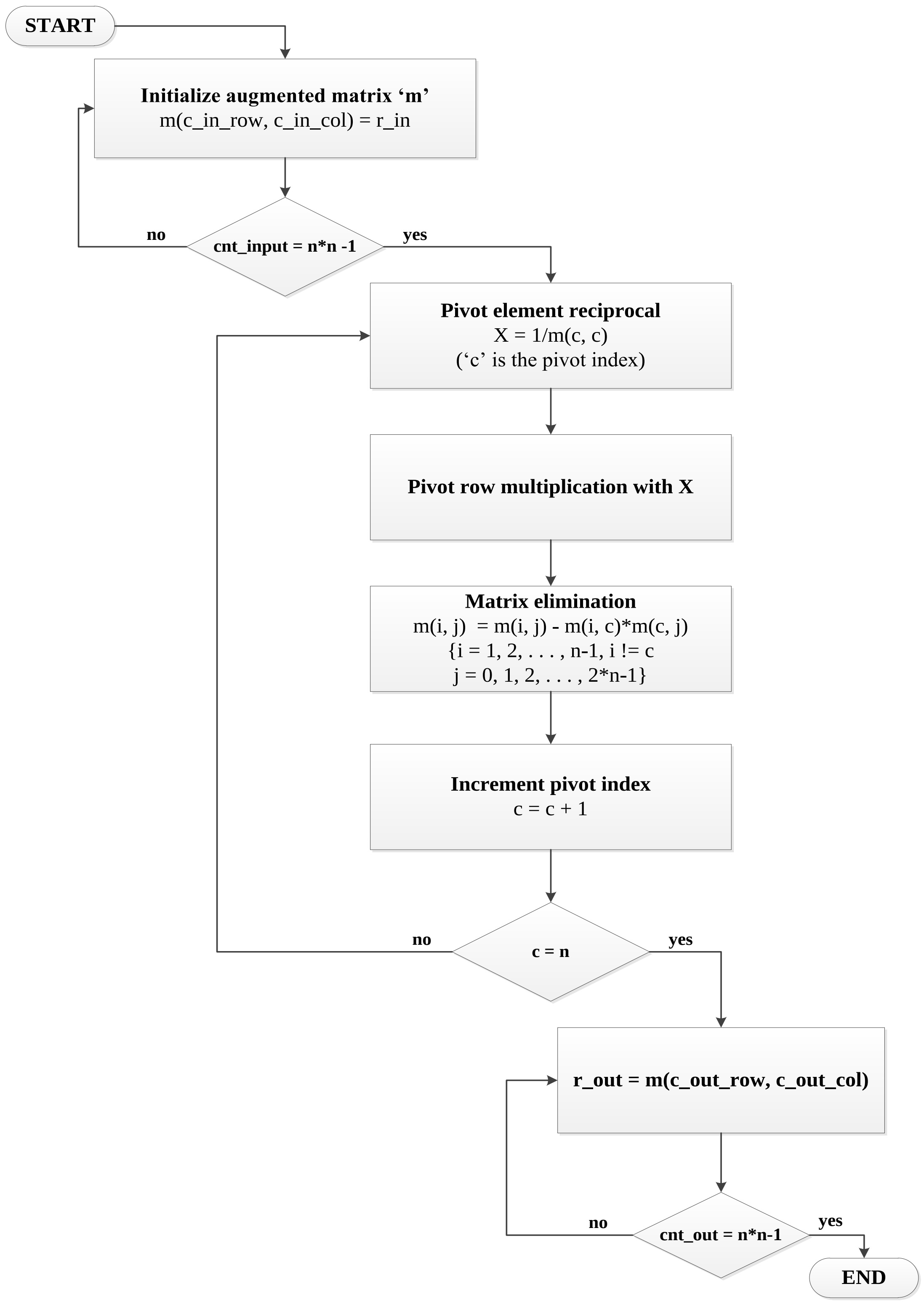

START
Initialize augmented matrix ‘m’
m(c_in_row, c_in_col) = r_in
no
cnt_input = n*n -1
yes
Pivot element reciprocal
X = 1/m(c, c)
(‘c’ is the pivot index)
Pivot row multiplication with X
Matrix elimination
m(i, j) = m(i, j) - m(i, c)*m(c, j)
{i = 1, 2, . . . , n-1, i != c
j = 0, 1, 2, . . . , 2*n-1}
Increment pivot index
c = c + 1
no
c = n
yes
r_out = m(c_out_row, c_out_col)
no
cnt_out = n*n-1
yes
END

Figure 5.9: Matrix Inversion Flow chart

m is the memory used for augmented matrix of size $\boldsymbol{n\,X\,2n}$. At the start, **m** is initialized by placing the matrix at the left side and the identity matrix of size $\boldsymbol{n\,X\,n}$ on the right side of memory **m.**

When **enable_in** is high, system starts taking input from the calling module. In our case, correlation matrix R is the input to the matrix inversion module. enable_in remains high till the system has received the complete matrix.

```
if (ce)
begin
if (enable_in)
m[c_in_row][c_in_col]  <=  R_in

end


for (i=0; i<n; i=i+1)
for (j=0; j<n; j=j+1)
if(i==j)
m[i][j+n] <= {26'd1,32'd0};   // Q26.32 format
```

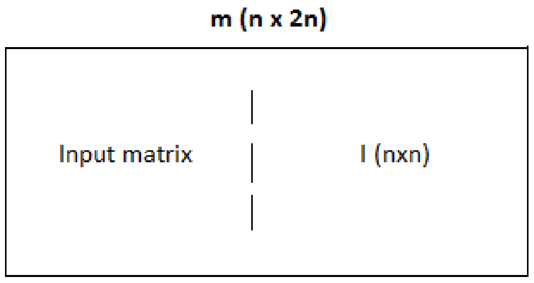


Figure 5.10: memory initialization for matrix inversion

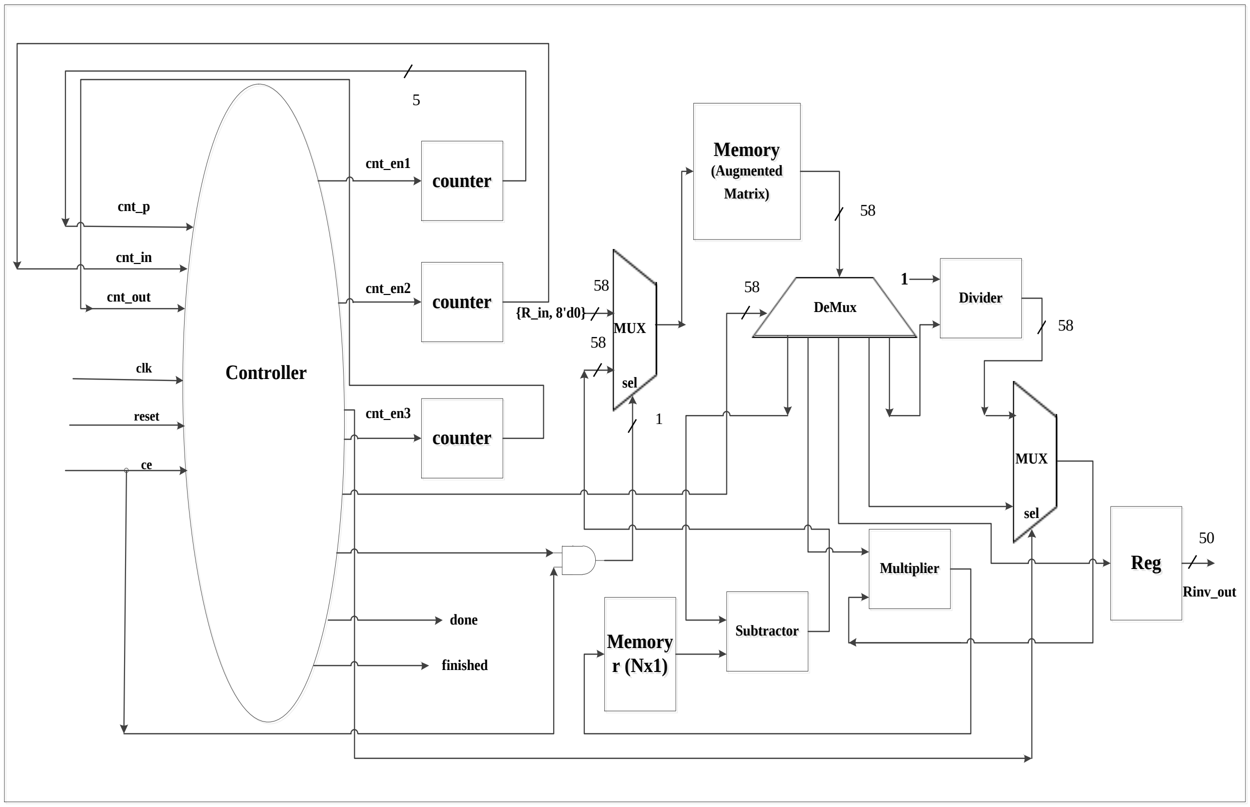


Figure 5.11: *Matrix Inversion Hardware*

5.3.3. Coefficients vector ‘a’

Coefficient vector $\boldsymbol{a}$ is given by the following equation.

$$\boldsymbol{a} = \boldsymbol{R}^{-1} * \boldsymbol{c}$$

$\boldsymbol{R}$ is the correlation matrix, $\boldsymbol{R}^{-1}$ is the output from matrix inverter and **c** is the correlation vector. Hardware for the computation of $\boldsymbol{a}$ is shown in figure 5.12.

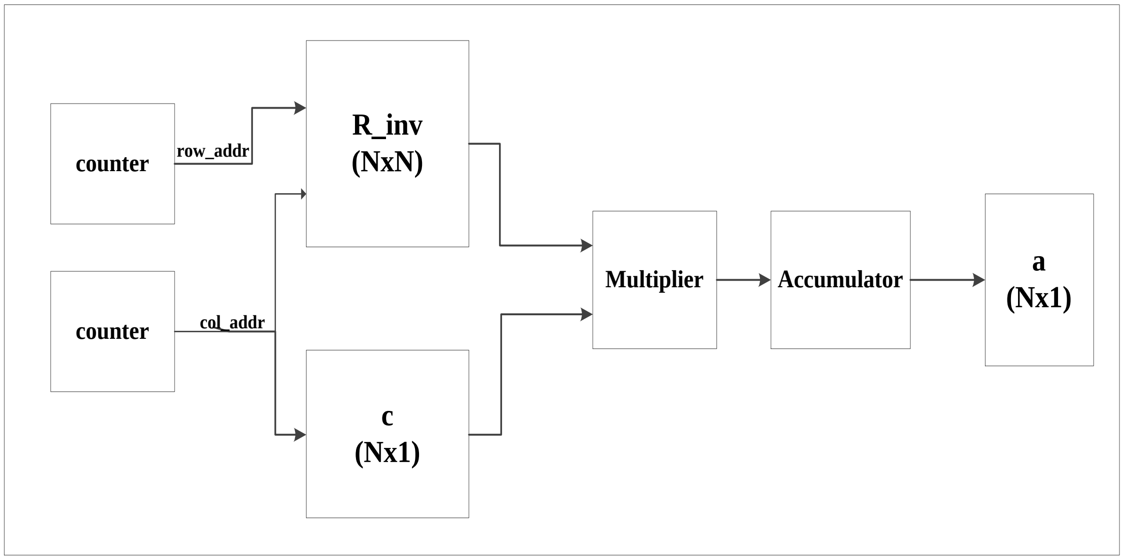


*Figure 5.12: Hardware for computation of coefficients vector **a***

5.3.4. SDF Image 'h'

SDF image is the weighted sum of training images

$$h(x, y) = \sum_{i=1}^{N} a_i t_i(x, y)$$

Each training image is multiplied with the corresponding coefficient in **a**. For a given count, coefficient from ***a*** is read and also the pixel value is read from the location indexed at:

$$addr = (cnt\_n \, * \, d) + mem\_addr\_im$$

Here, d is the total number of pixels of an image and *mem_addr_im* is the counter which is updated when pixel value has been read. Coefficient and pixel values are multiplied and result is accumulated to SDF_out and count is updated. When count reaches *N* (number of training images), SDF_out is sent to the output (calling module).

mem_addr_im is updated and the above steps are repeated for the next pixel values. And this continues until *mem_addr_im* reaches d (number of pixels of training images).

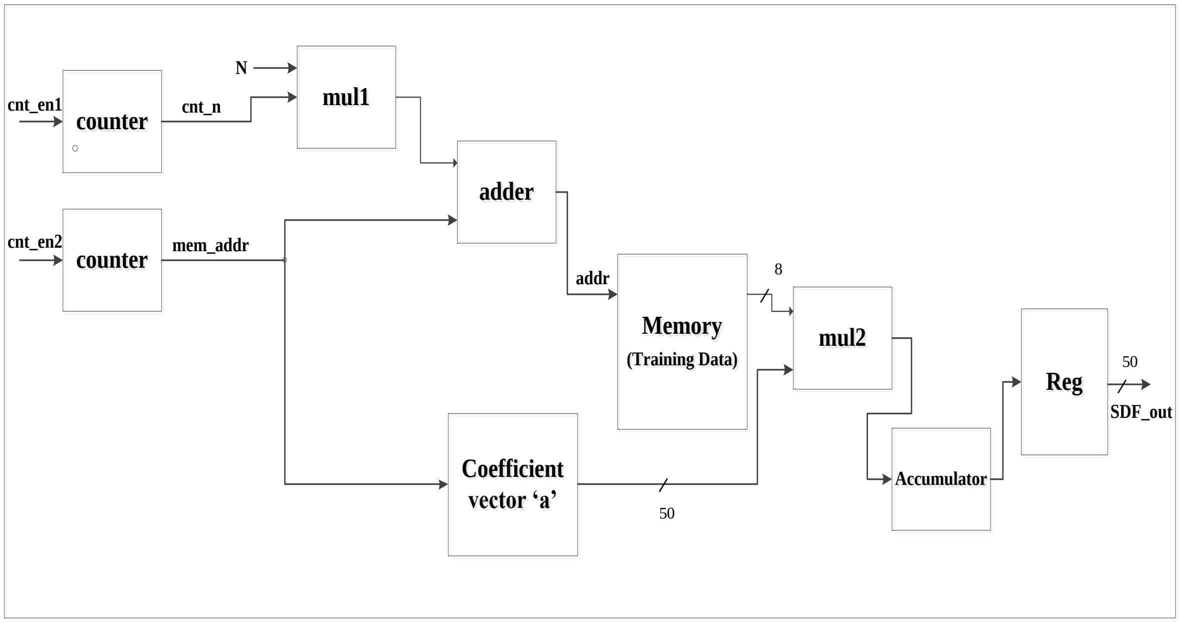


Figure 5.13:Part of CreateSDF module that generates SDF image

5.4. Difference of Gaussian

Difference of Gaussians is given by:

$$g(x,y) = \frac{1}{2\pi\sigma_1^2}\exp[-\frac{x^2+y^2}{2\pi\sigma_1^2}] - \frac{1}{2\pi\sigma_2^2}\exp[-\frac{x^2+y^2}{2\pi\sigma_2^2}]$$

In module Diff_of_Gaussian, exponentials are computed and output to the module DoG. Difference of Gaussian requires exponential function to be implemented in hardware and also a function for the integer powers of 'e'. Exponent for e i.e. $(x^2 + y^2 / 2\pi\sigma^2)$ for both σ_1 and σ_2 are calculated and these values are divided into integer and fractional parts. Fractional power of e is computed using exponential function and integer power using Integer power function.

5.4.1 Exponential Function

We implemented exponential function using its series expansion.

$$\boldsymbol{e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots}$$

Input to the system is 'x', which is a fractional number in Q0.15 format. Output of the system is e^x. Hardware for exponential function is shown is figure 5.14. One multiplier is used for calculation of numerators. Output of this multiplier is reused to save number of clock cycles. A multiplexer is used for this purpose. Inputs to this 2 to 1 mux are power (x) and the output from the multiplier.

Instead of calculating the factorial for each denominator, we are using reciprocals of factorials as constants. These constants are hardwired instead of using storage space, which were input to the multiplier through a multiplexer.

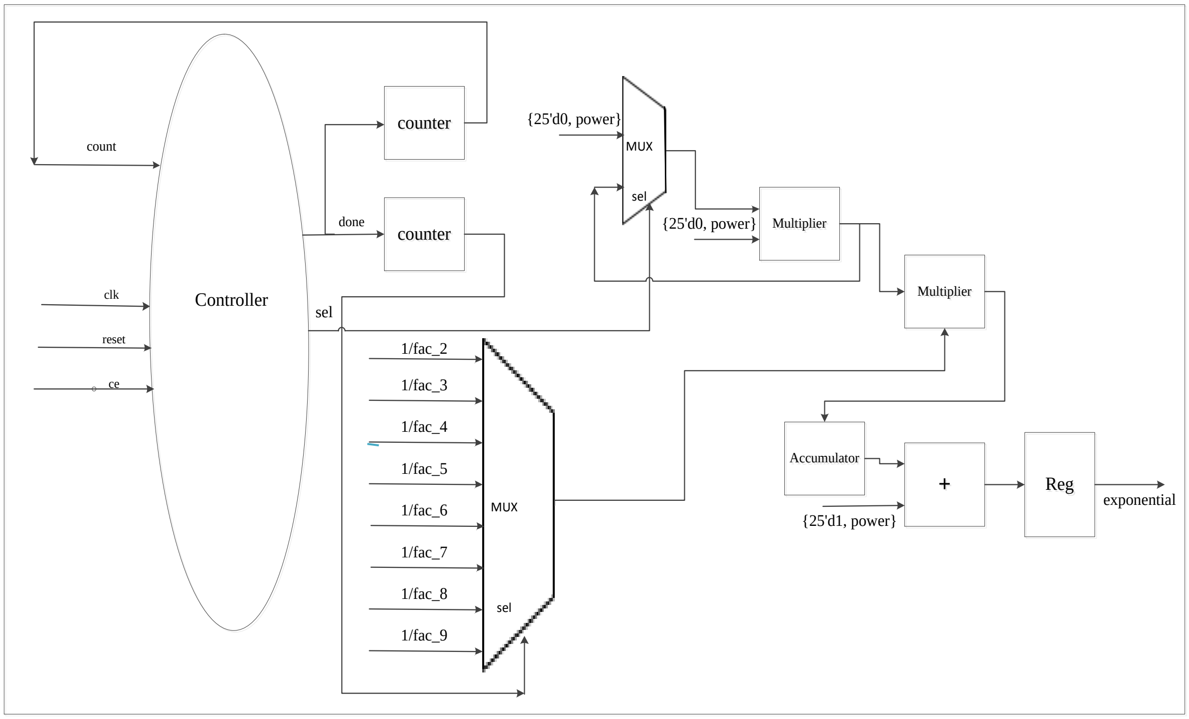


Figure 5.14: Hardware architecture for for exponential function

5.4.2 Integer Power

This function is used for the calculation of integer powers of exponential ‘e’. Inputs to the system are the numbers for which integer power is to be calculated and the power. Number should be in Q25.15 format and power in Q25.0 format.

5.5. Correlator

Input test image is correlated with the Difference of Gaussian function and this result is correlated with the SDF image which is output of the CreateSDF module. Correlations are performed using Fast Fourier Transform. We are using Xilinx FFT IP Core for this purpose.

Figure 5.15 shows the structure of the Correlator.

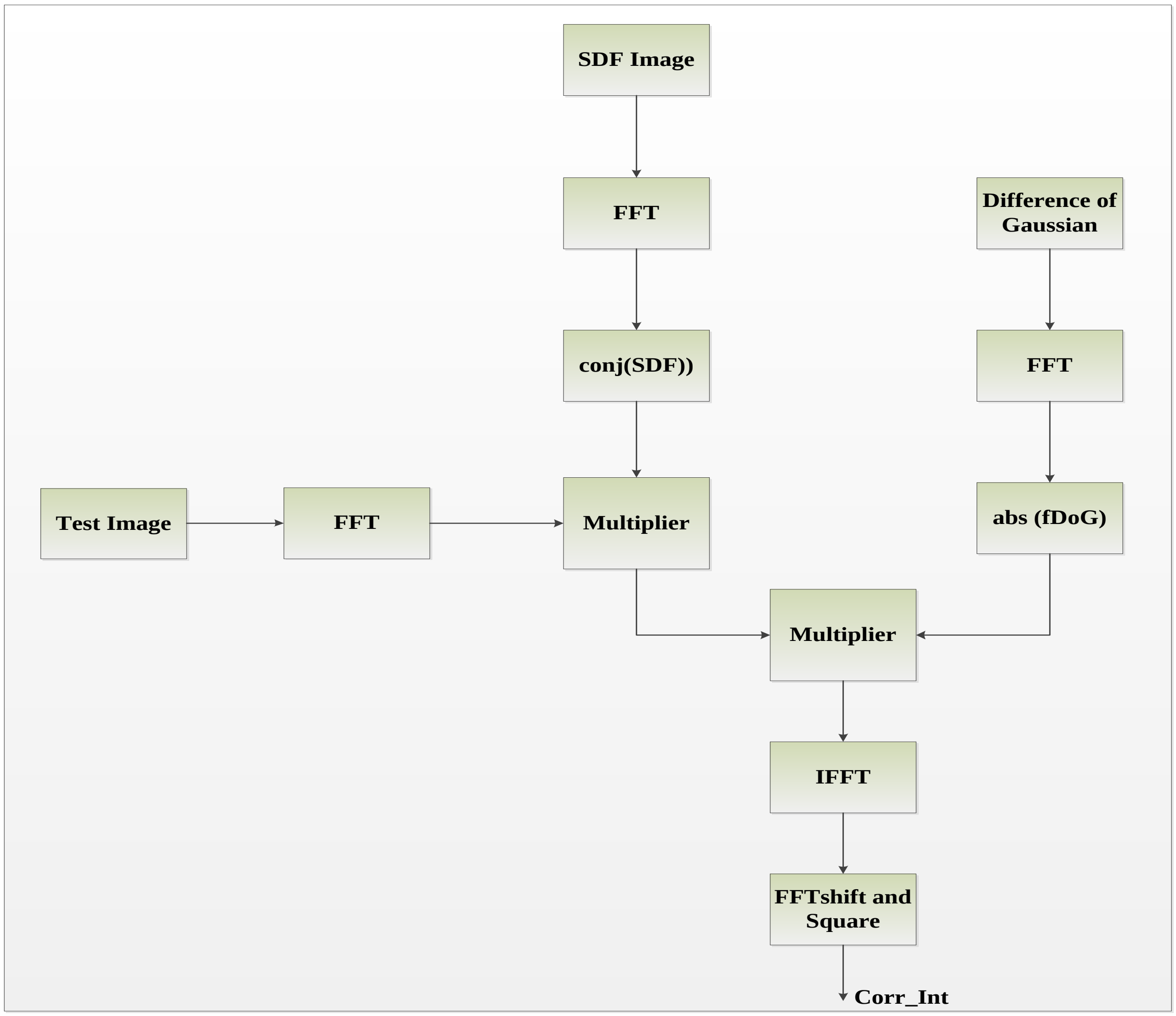


Figure 5.15: Correlator

Output of the Correlator i.e. Correlation Intensity is written to a text file and MATLAB is used to plot it. MATLAB code is given here, which reads each line of text file, converts the binary number into real number, and stores the numbers in a 2 dimensional matrix. This 2D matrix is the correlation intensity for a given test image. We plot it to show the correlation peaks.

```
fid = fopen('Corr_Int_output.txt', 'r');
i = 1;

%%%%% convert from binary Q 25.25 format to real
while 1
tline = fgetl(fid);

if ~ischar(tline)
break
end

a(i) = 0;
l = 24;
for j=1:50
x = str2num(tline(j));
a(i) = a(i) + (x*(2^l));
l = l-1;
end
i = i+1;
end
fclose(fid);
%%%%%%%%%%%%%%%% save to a 2D matrix %%%%%%%%%%%%%%%%%%
z = 1;
for n=1:128
for m=1:128
CORRint(m,n) = a(z);
z = z + 1;
end
end

%%%%%%%%%%%%%%%%%%%%%%% Plot %%%%%%%%%%%%%%%%%%%%%%%%%%
figure
surf(real(CORRint), 'EdgeColor', [0 0 0.1], 'EdgeAlpha',
0.16, ...
'FaceColor', [0.5 0.5 0.7], 'AmbientStrength', 0.75, ...
'DiffuseStrength', 0.9, 'SpecularStrength', 0.3);

grid off,lighting gouraud ;
camlight headlight;
```

Same code could be used to view the composite image i.e. SDF image in MATLAB. We write the output of CreateSDF to a text file and use the above code to plot the image.

Chapter 6

RESULTS

A database of car images was used to demonstrate the working of SDF design. Database contains total of 72 images with each image differing 5^o from each other. Some of the images are shown in figure 6.1.

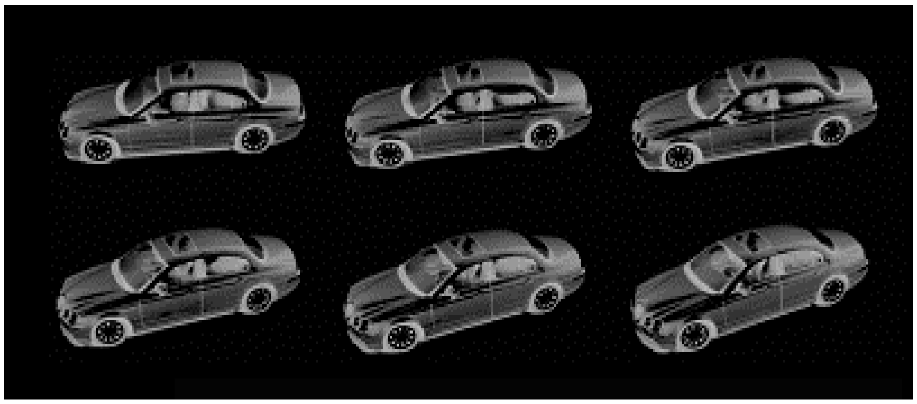


Figure 6.1: Set of Reference Images

Composite image is constructed from the reference images using the weighted linear combination technique which is shown in fig 6.2. This image is the output from the CreateSDF module. Output of CreateSDF is written to a text file and plot is shown in MATLAB.

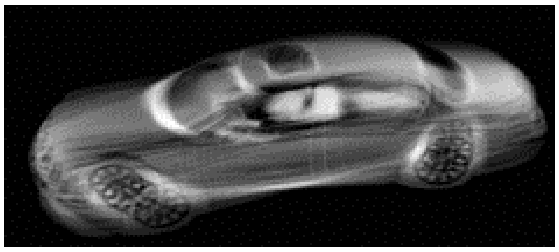


Figure 6.2: Composite Image

6.1. Simulation Results

In this section the simulation results are presented and discussed. Composite image is constructed by using the images of the car rotated at 0, 10, 15, 20, 25 and 30 degrees. We introduce a number of test images without background noise which are out of plane rotated for the detection of the car. All the simulation results are obtained using the standard deviation ratio of 1.6. Correlation Intensity is the output of the Correlator module, which is written to text file and the Correlation Intensity surfaces are shown in MATLAB. Correlation Output Peak Intensities and Peak to Correlation Energies are also recorded for all the test images. These results are compared with the results of MATLAB simulations and we have found that the results are approximately equal with very less precision errors.

Correlation plane for zero degree rotated car is shown in figure 6.4.

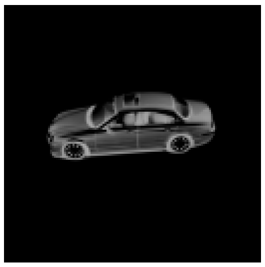


Figure 6.3: 0 degree rotated test image

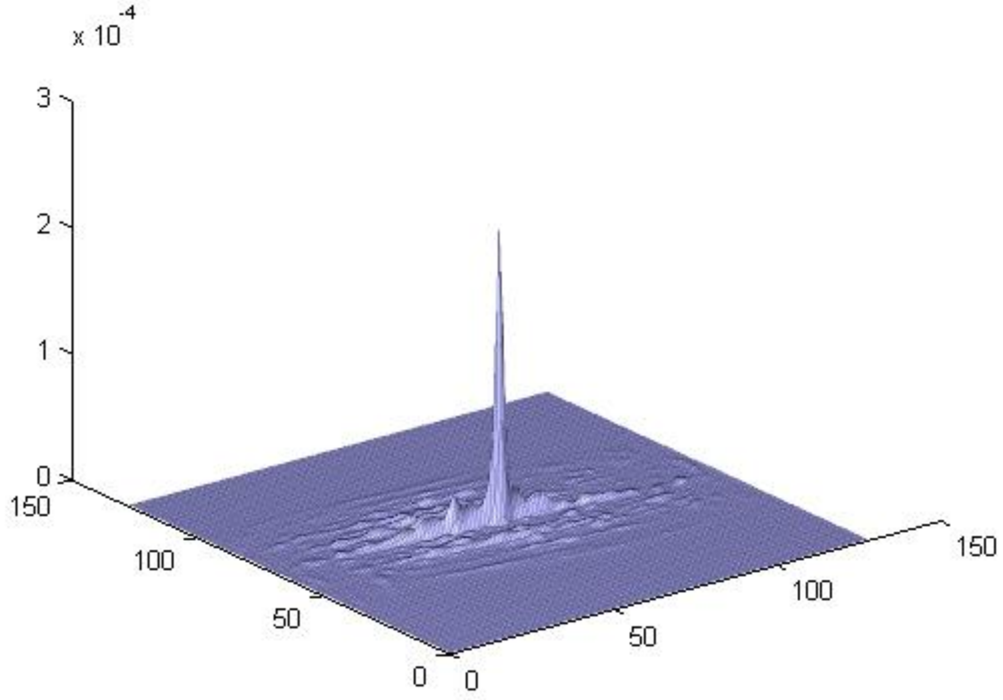


*Figure 6.4: Correlation plane for 0 degree rotated image with COPI = 2.2978*10^{-4} and PCE = 4.1160*

Correlation plane for five degree rotated car is shown in figure 6.6.

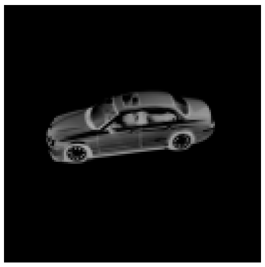


Figure 6.5: 5 degree rotated test image

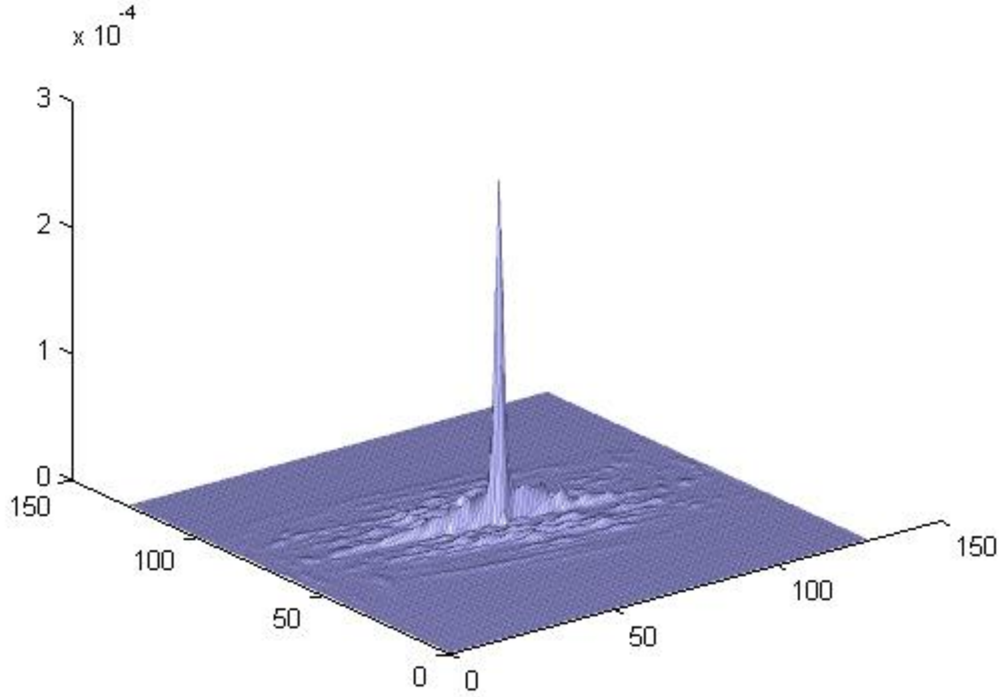


*Figure 6.6: Correlation plane for 5 degree rotated image with COPI = 2.6905*10^{-4} and PCE = 4.1943*

Correlation plane for ten degree rotated car is shown in figure 6.8.

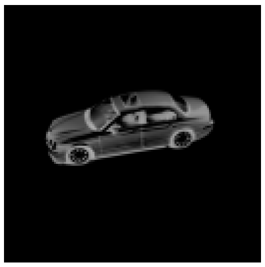


Figure 6.7: 10 degree rotated test image

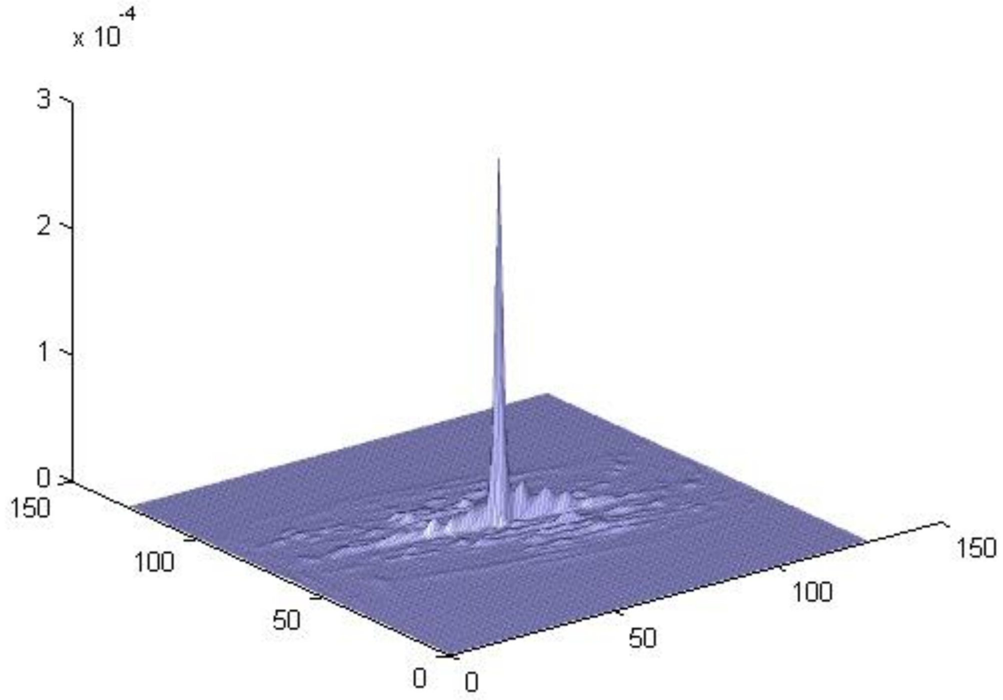


Figure 6.8: Correlation plane for 10 degree rotated image with COPI = $2.8754{*}10^{-4}$ and PCE = 5.0158

Correlation plane for fifteen degree rotated car is shown in figure 6.10.

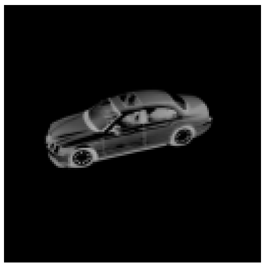


Figure 6.9: 15 degree rotated test image

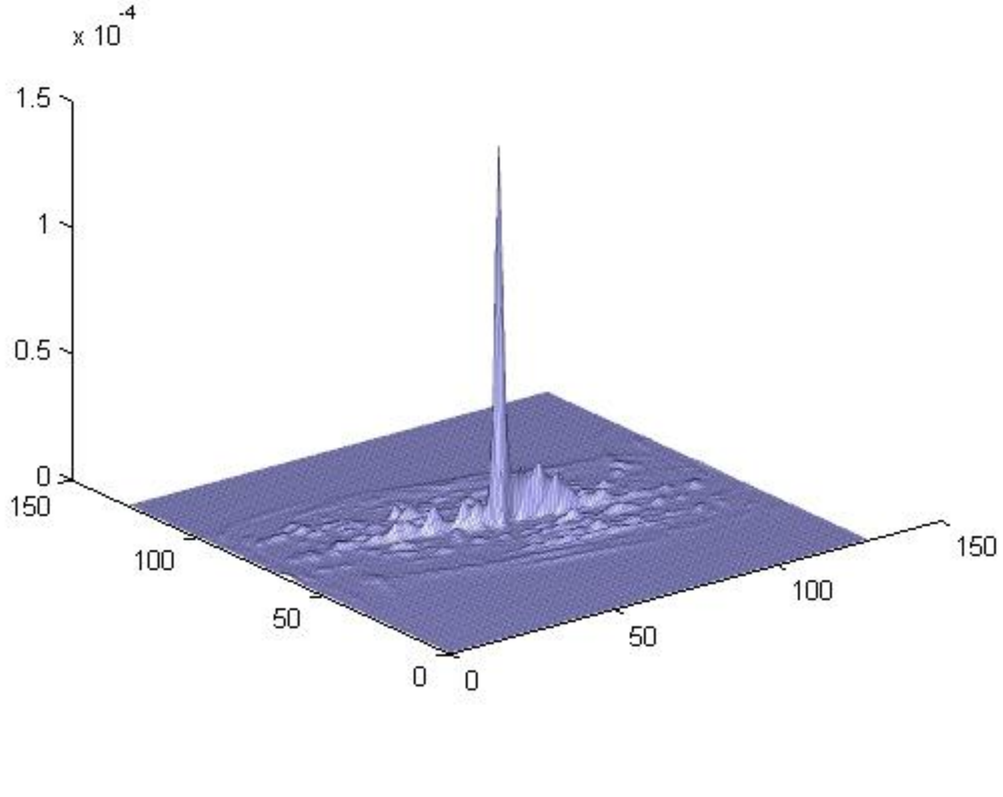


Figure 6.10: Correlation plane for 15 degree rotated image with COPI = $1.4793*10^{-4}$ and PCE = 3.5682

Correlation plane for twenty degree rotated car is shown in figure 6.12.

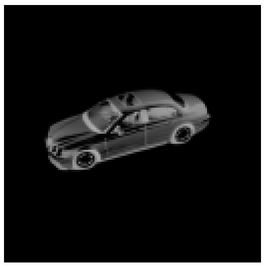


Figure 6.11: 20 degree rotated test image

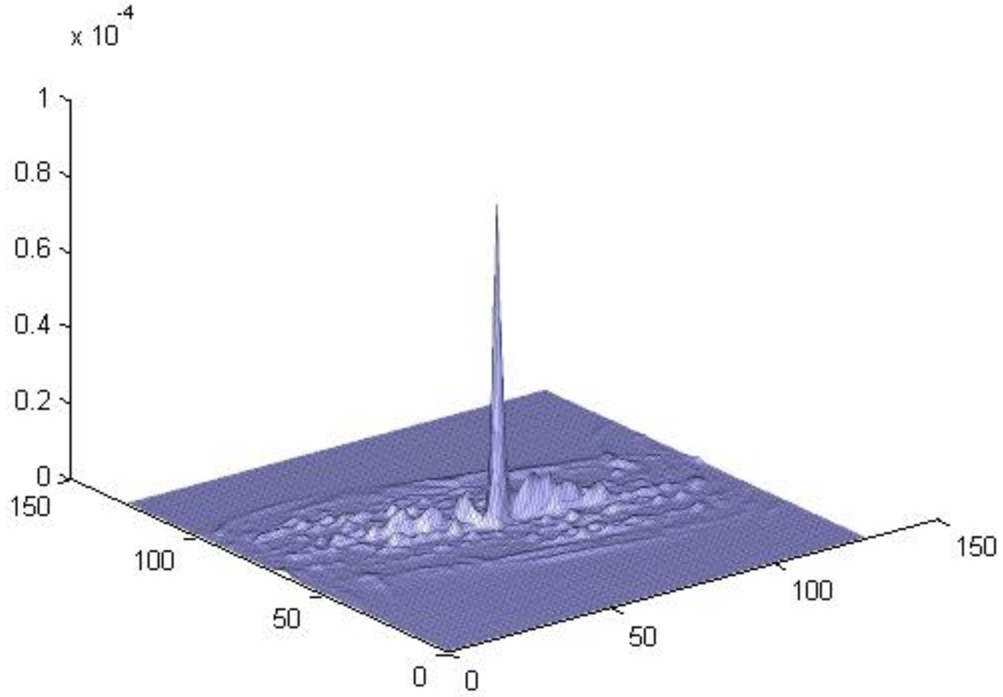


Figure 6.12: Correlation plane for 20 degree rotated image with COPI = $8.2764*10^{-5}$ and PCE = 2.4561

Correlation plane for twenty five degree rotated car is shown in figure 6.14.

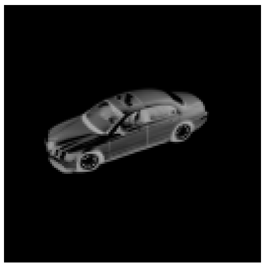


Figure 6.13: 25 degree rotated test image

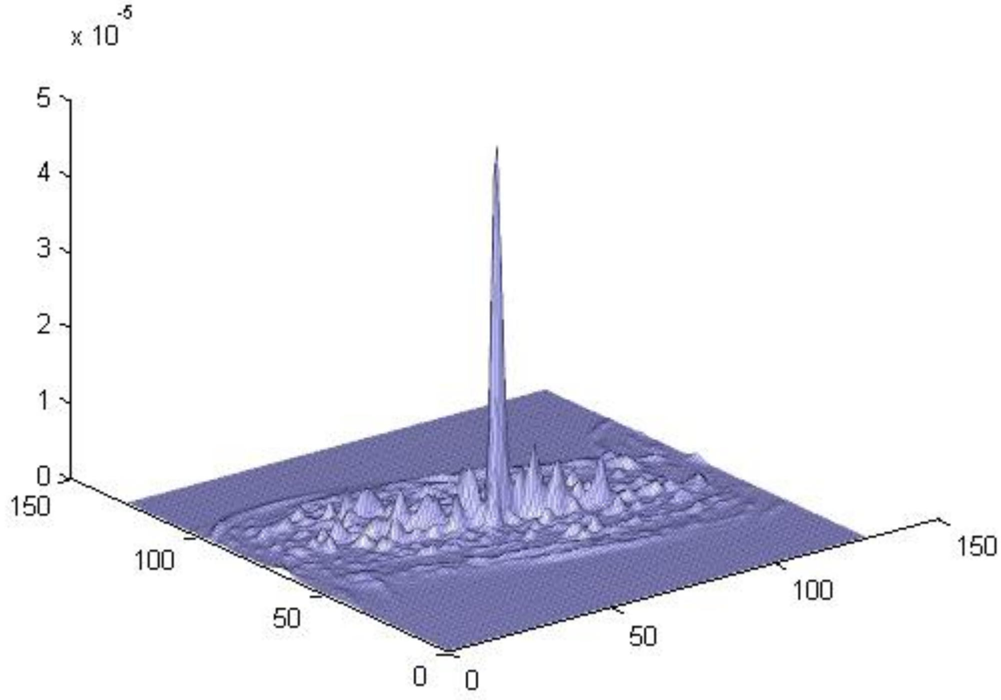


Figure 6.14: Correlation plane for 25 degree rotated image with COPI = $4.8986*10^{-5}$ and PCE = 1.5625

Correlation plane for thirty degree rotated car is shown in figure 6.16.

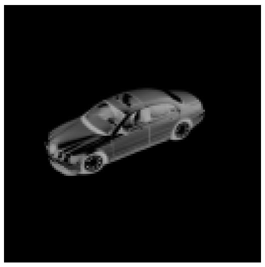


Figure 6.15: 30 degree rotated test image

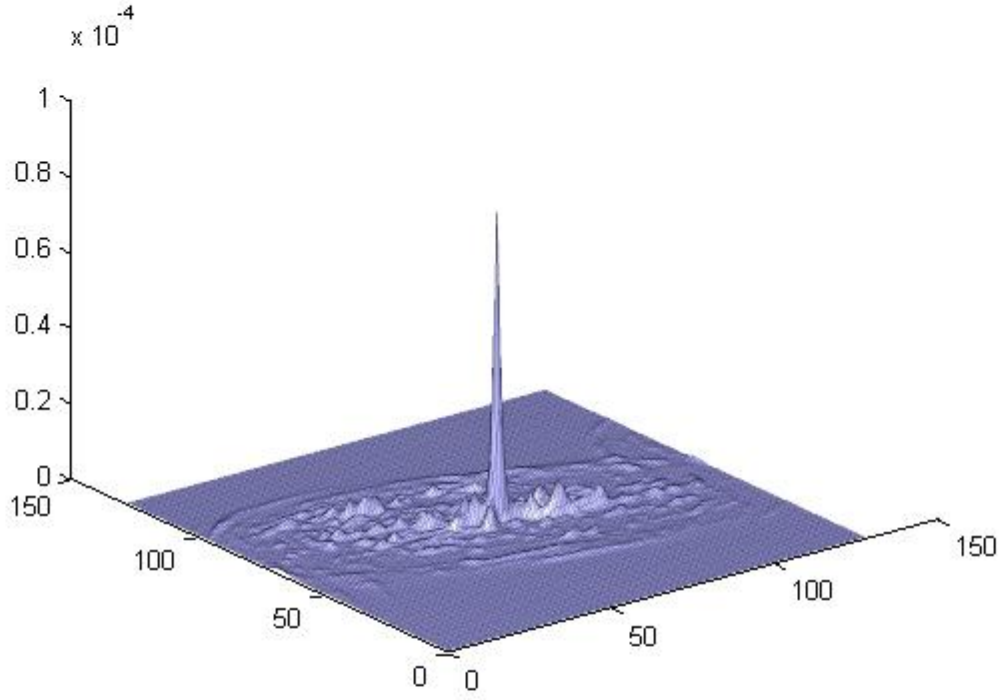


Figure 6.16: Correlation plane for 30 degree rotated image with COPI = $8.0559*10^{-5}$ and PCE = 2.6891

Correlation plane for thirty five degree rotated car is shown in figure 6.18.

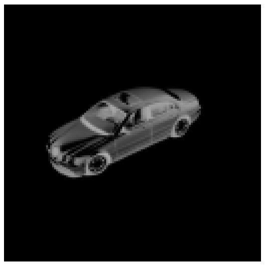


Figure 6.17: 35 degree rotated test image

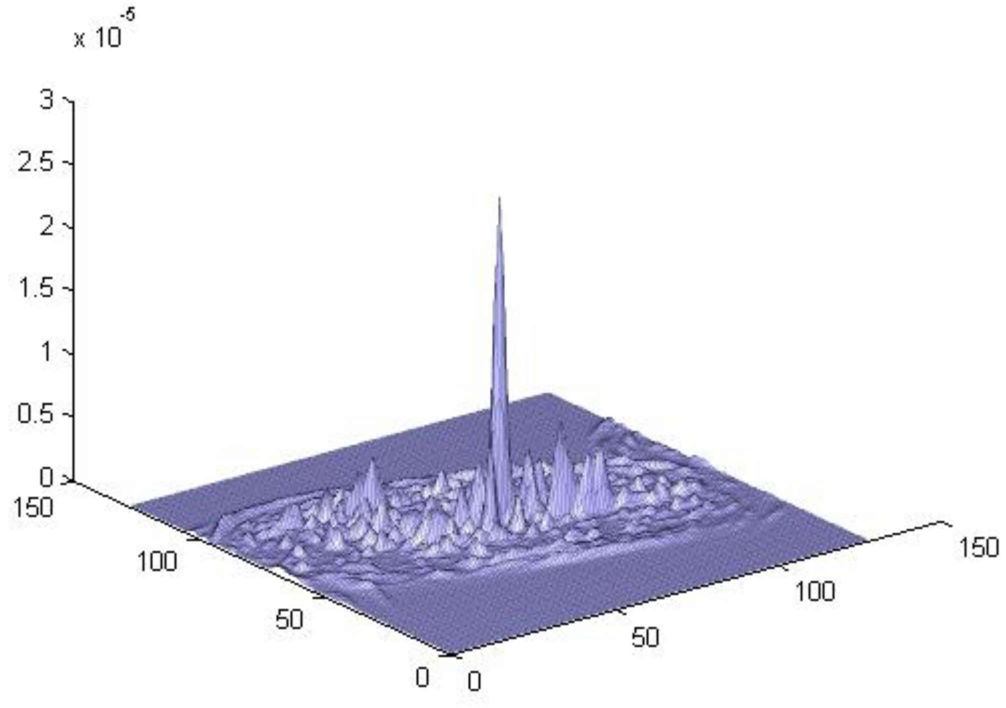


*Figure 6.18: Correlation plane for 35 degree rotated image with COPI = 2.5620*10^{-5} and PCE = 0.9531*

Correlation plane for 340 degree rotated car is shown in figure 6.20.

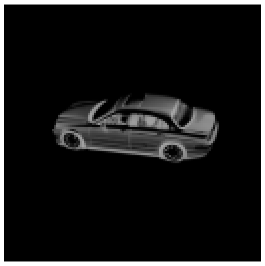


Figure 6.19: 340 degree rotated test image

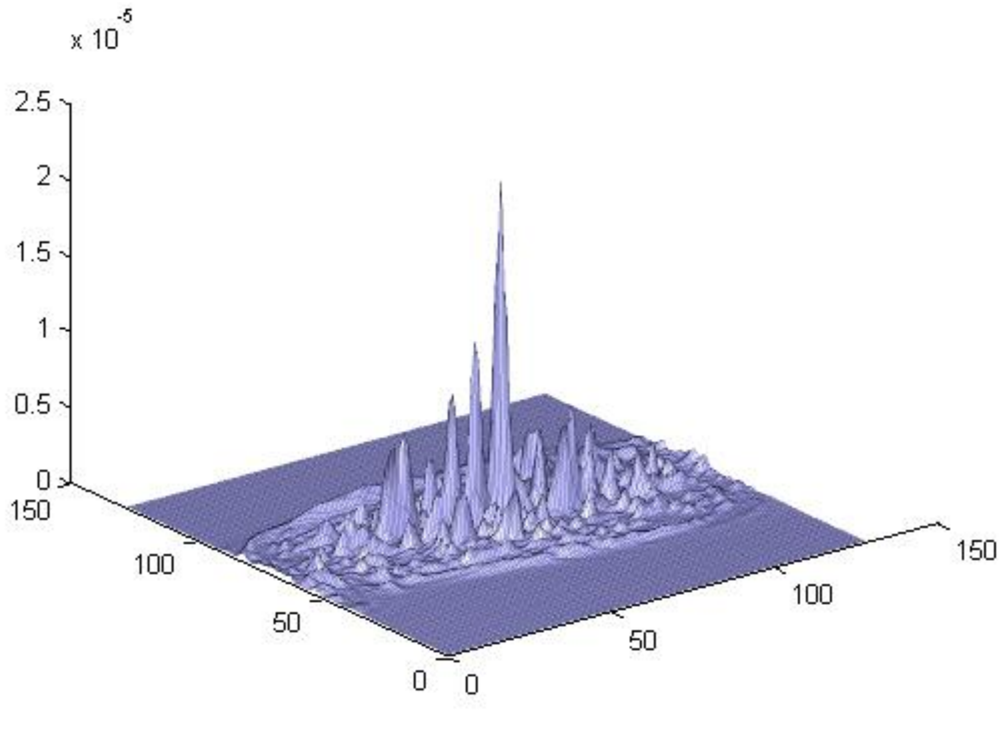


*Figure 6.20: Correlation plane for 340 degree rotated image with COPI = 2.2202*10^{-5} and PCE = 0.8288*

Correlation plane for 355 degree rotated car is shown in figure 6.21.

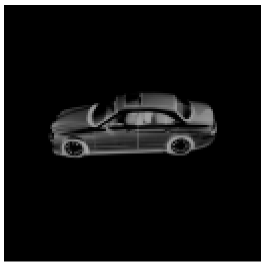


Figure 6.21: 355 degree rotated test image

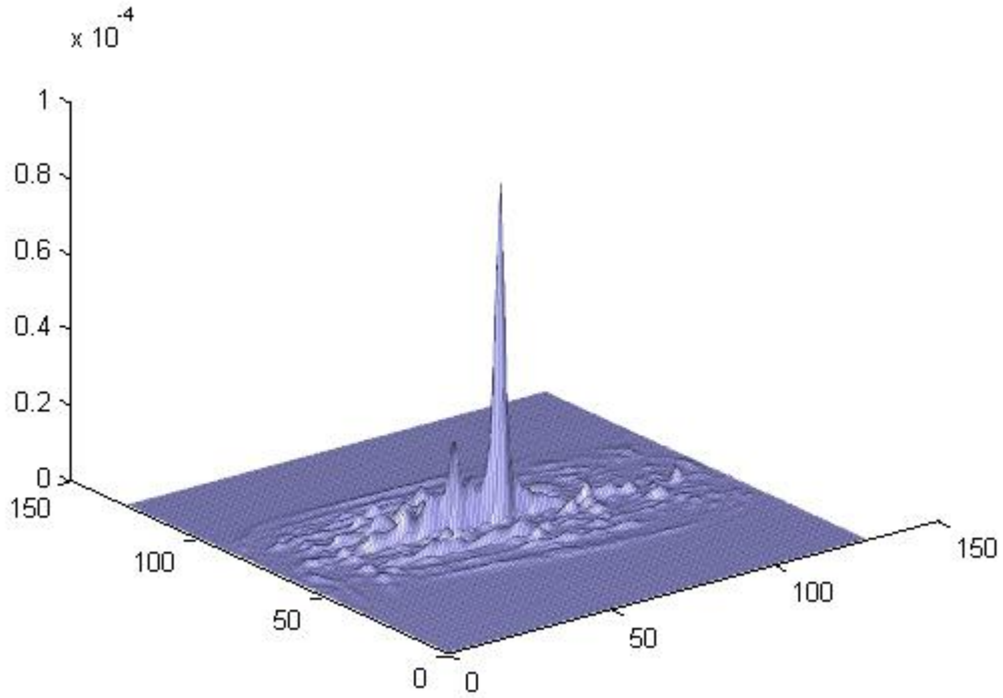


Figure 6.22: Correlation plane for 355 degree rotated image with COPI = $8.8591*10^{-5}$ and PCE = 2.2131

It can be seen from figures 6.4, 6.6, 6.8, 6.10, 6.12, 6.14 6.16, 6.18, 6.20 and 6.22 that the COPI values are greater for the training images as compared to the testing data. For the test data as the out of plane rotational angle increases, the peak height decreases. Since there is no background noise, the correlation output plane shows minimal disruption. The correlation peaks are localised due to the inclusion of the difference of Gaussian bandpass in the filter design. The composite image accommodates the out-of-plane rotation of the car allowing maintenance of the correlation peak height over the swathe of angles covered within the composite image.

The filtering operation performed by the DOG filter can be controlled by the standard deviations. The pass band of frequencies is altered by changing the standard deviation in the DoG construction whilst keeping the ratio equal to 1.6. This in turn translates to different peak widths in the correlation plane. 30 degree rotated car image is tested. We see that the peak is sharpest for standard deviations 0.8 and 0.5. [19]

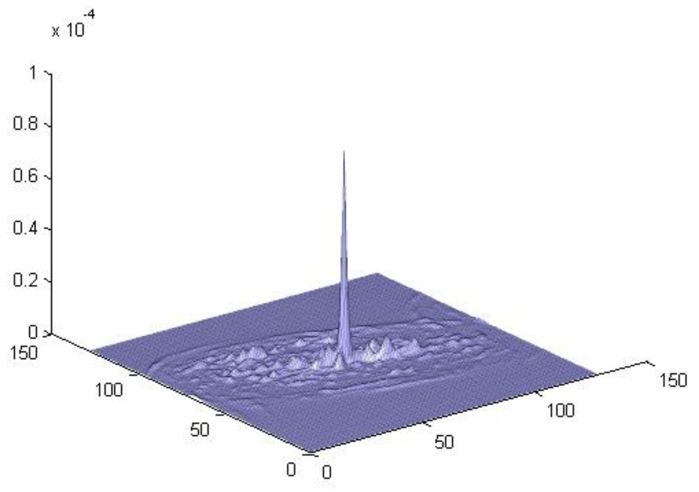


*Figure 6.23: Correlation plane for 30 degree rotated car image with σ_1= 0.8, σ_2= 0.5, COPI = 8.0559*10^{-5}, PCE = 2.6891*

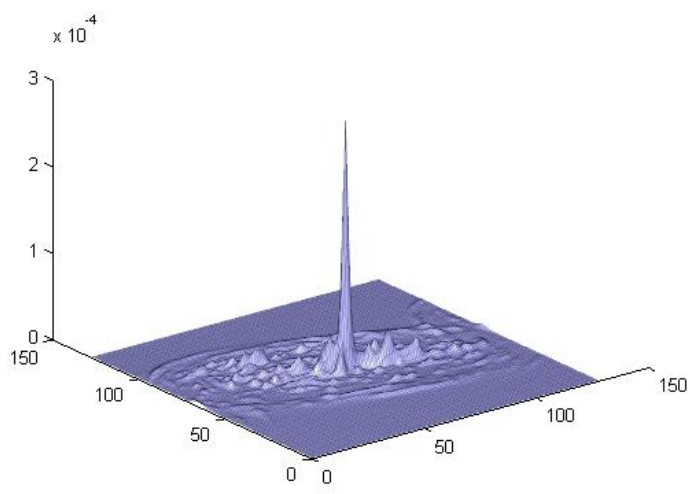


Figure 6.24: Correlation plane for 30 degree rotated car image with $\sigma_1 = 1.2$, $\sigma_2 = 0.75$, COPI = $2.8475*10^{-4}$, PCE = 1.8643

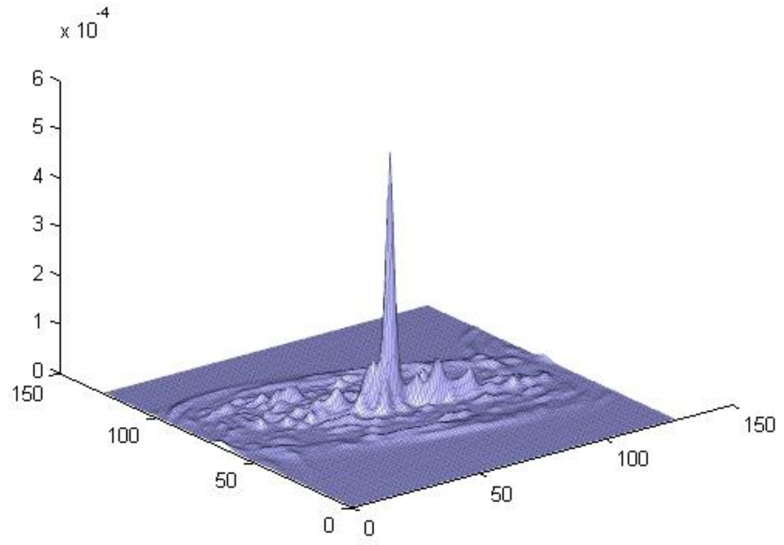
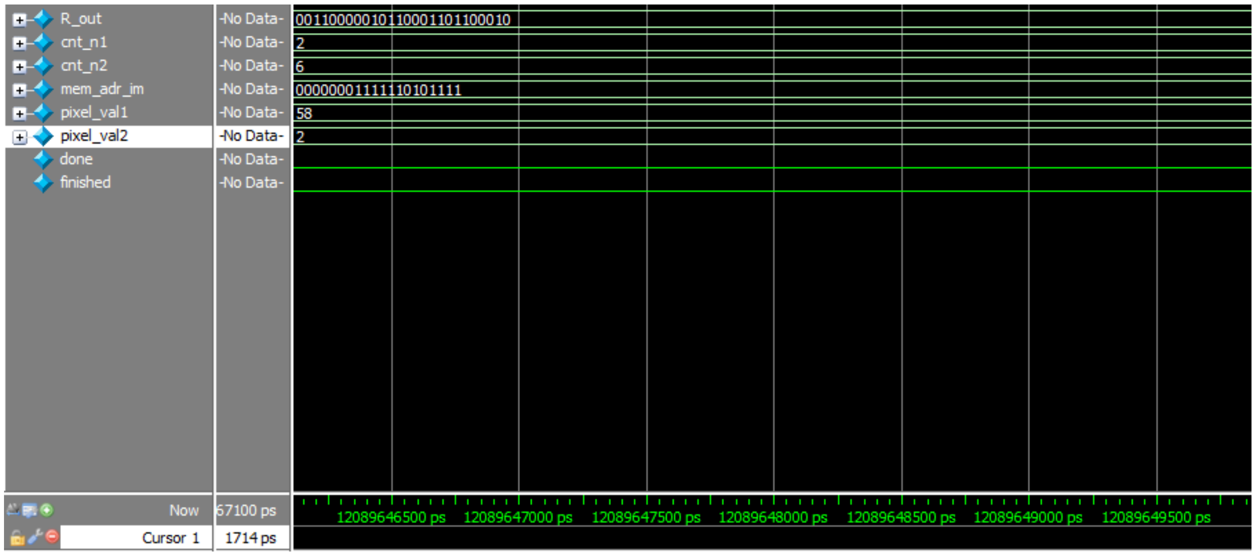


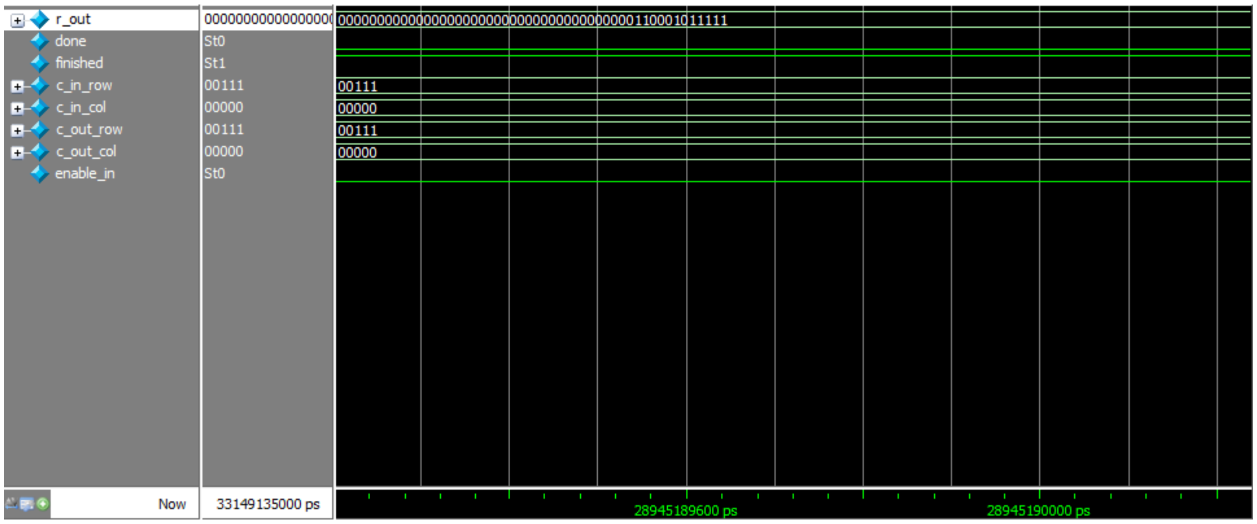
Figure 6.25: Correlation plane for 30 degree rotated car image with σ_1 = 1.6, σ_2 = 1, COPI = $5.1740*10^{-4}$, PCE = 1.2828

6.2. ModelSim Simulations



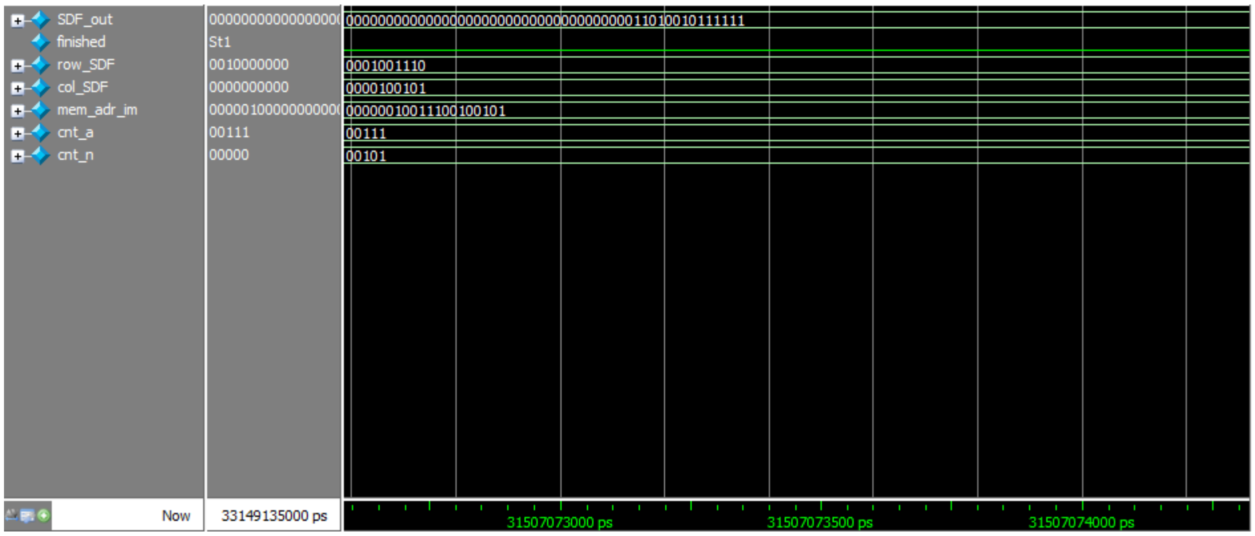
```
R_out        -No Data-  0011000001011000110110001 0
cnt_n1       -No Data-  2
cnt_n2       -No Data-  6
mem_adr_im   -No Data-  00000001111110101111
pixel_val1   -No Data-  58
pixel_val2   -No Data-  2
done         -No Data-
finished     -No Data-
Now          67100 ps
Cursor 1     1714 ps
```

Figure 6.26: Create_Corr_Matrix Simulation



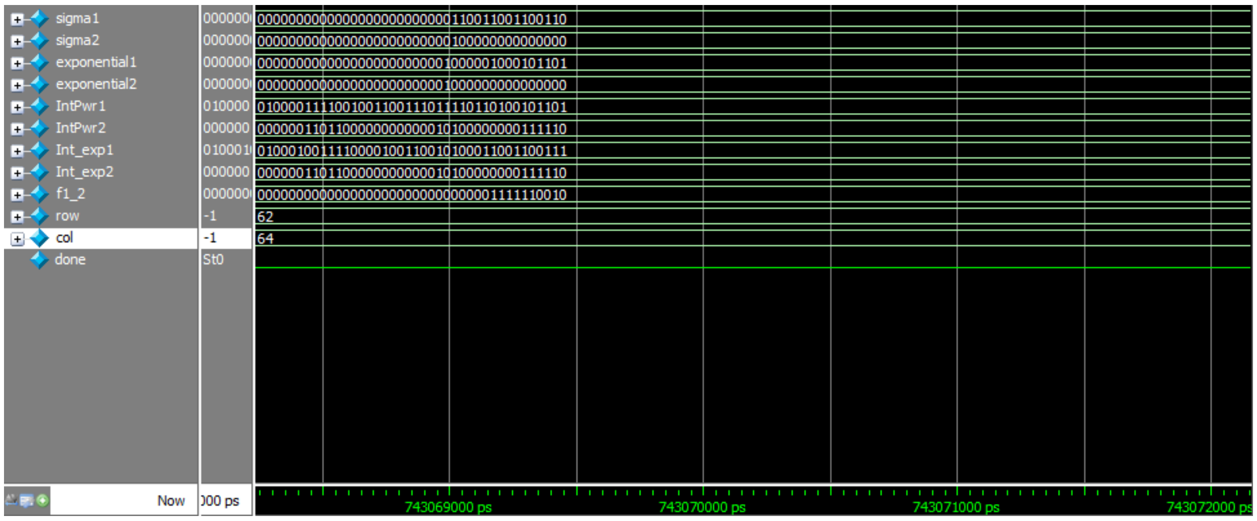
```
r_out        000000000000000000  0000000000000000000000000000000000000000000000011000101111 1
done         St0
finished     St1
c_in_row     00111              00111
c_in_col     00000              00000
c_out_row    00111              00111
c_out_col    00000              00000
enable_in    St0
Now          33149135000 ps
```

Figure 6.27: Matrix_Inverse Simulation



```
SDF_out       0000000000000000(  000000000000000000000000000000000000000001101001011111 1
finished      St1
row_SDF       0010000000         0001001110
col_SDF       0000000000         0000100101
mem_adr_im    00000100000000000( 00000010011100100101
cnt_a         00111              00111
cnt_n         00000              00101
Now           33149135000 ps     31507073000 ps   31507073500 ps   31507074000 ps
```

Figure 6.28: CreateSDF Simulation



```
sigma1         000000(  000000000000000000000000001100110011001 10
sigma2         000000(  000000000000000000000000001000000000000 000
exponential1   000000(  00000000000000000000000001000001000101101
exponential2   000000(  00000000000000000000000001000000000000000
IntPwr1        010000   010000111100100110011101111101101001 01101
IntPwr2        000000   000000110110000000000000101000000000111110
Int_exp1       010001(  010001001111000010011001010001100110011 1
Int_exp2       000000   000000110110000000000000101000000000111110
f1_2           000000(  0000000000000000000000000000000011111100 10
row            -1       62
col            -1       64
done           St0
Now            )00 ps   743069000 ps   743070000 ps   743071000 ps   743072000 ps
```

Figure 6.29: Difference of Gaussian Simulation

6.3. Synthesis Results

Timing Summary:

Speed Grade: -2

Minimum period: 4.766ns (Maximum Frequency: 209.813MHz)

Minimum input arrival time before clock: 2.205ns

Maximum output required time after clock: 1.317ns

Device utilization:

Device Utilization Summary (estimated values)

Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	50718	2443200	2%
Number of Slice LUTs	66371	1221600	5%
Number of fully used LUT-FF pairs	48658	78964	62%
Number of bonded IOBs	220	1200	18%
Number of Block RAM or FIFO	60	1292	5%
Number of BUFG or BUFGCTRLs	4	128	3%
Number of DSP48E1s	115	2160	5%

Table 6.1: Device Utilization

Chapter 7

CONCLUSIONS AND FUTURE WORK

7.1. Conclusions

The design was first modeled using MATLAB to get the specifications and requirements for the hardware system. The RTL code of the design was successfully simulated in ModelSim. The working of the designed hardware is verified using ModelSim simulations and compared with the model designed in MATLAB. Fixed point implementation of the design was carried out by obtaining the floating-point values from MATLAB because the fixed- point representation is more efficient.

The designed system was synthesized using Xilinx. The operating frequency of the design is 209.813MHz and minimum period of 4.766ns on Xilinx FPGA. The top design takes about 115 DSP slices out of 2160 slices on a Virtex 7 FPGA. The device selected is xc7v2000t at speed grade of 2.

7.2. Future Work

To increase the speed of the system, parallelism and pipelining concepts can be employed in the implementation of building blocks of SDF filter. The hardware design for SDF filter provides a base for the design of other correlation filters. This work could be extended to the hardware designs for the advanced correlation filters. Hardware design of log polar mapping could be added to the current design for the in-plane rotation invariance.

REFERENCES

[1] A. VanderLugt, “Signal detection by complex spatial filtering” *IEEE Trans. Informatiom Theory*, no.10, pp.139-145,1964.

[2] B.V.K. Vijay Kumar, “Tutorial survey of composite filter design for optical correlation” *Applied Optics*, vol. 31, pp. 4773-4801, 1992.

[3] B.V.K. Vijay Kumar, M. Savvides, C. Xie, K. Venkataramani, J. Thornton, and A. Mahalanobis, “Biometric Verification with Correlation Filters,” *Applied Optics*, Vol 43, No.2, pp. 391-402, 2004.

[4]United Nations Educational, Scientific, and Cultural Organization (UNESCO) http://www.unesco.org/webworld/idams/advguide/Chapt9_2.htm : Visited April 09, 2011.

[5] Richard O.Duda, Peter E. Hart, David G. Stork, “*Pattern Classification*”, John Wiley & Sons, 2nd Edition

[6] S. R. F. Sims and A. Mahalanobis. Performance evaluation of quadratic correlation filters for target detection and discrimination in infrared imagery. Optical Engineering, 43:1705–1711, 2004.

[7] J. Thornton, M. Savvides, and B. V. K. Vijaya Kumar. A Bayesian approach to deformed pattern matching of iris images. IEEE Trans. Pattern Anal. Machine Intell., 29(4):596–606, 2007.

[8] K. Venkataramani and B. V. K. Vijaya Kumar. Performance of composite correlation filters for fingerprint verification. Opt. Eng., 43:1820–1827, 2004.

[9] B.V.K. Vijaya Kumar, “Minimum variance synthetic discriminant functions,” J.Opt.Soc.Am.A, vol. 3, no. 10, pp. 1579–1584, 1986.

[10] A. Mahalanobis, B.V.K. Vijaya Kumar, and D. Casasent, "Minimum average correlation energy filters," Appl. Opt, vol. 26, no. 17, pp. 3633–3640, 1987.

[11] Joseph L. Horner and Peter D. Gianino, "Phase-only matched filtering", Applied Optics, Vol 23, No 6, (1984).

[12] Jeffery A. Davis, John J. Kane, and Don M. Cottrell, "Horner efficiency of phase-only and binary phase-only filters", 10 September 1993 / Vol. 32, No. 26 / APPLIED OPTICS, 5095-5099

[13] Ph. Refregier, "Optimal trade-off filters for noise robustness, sharpness of the correlation peak, and Horner efficiency", June 1, 1991 / Vol. 16, No. 11 / OPTICS LETTERS.

[14] A. A. S. Awwal, H. E. Michel, "Enhancing the Discrimination Capability of Phase Only Filter", Asian Journal of Physics, Vol. 8, pp. 381-384, (1999).

[15] H. J. Caulfield and W. Maloney, "Improved discrimination in optical character recognition", Applied Optics, Vol. 8, No. 11, pp. 2354-2356, (1969).

[16] C. F. Hester and D. Casasent, "Multivariant technique for multiclass pattern recognition", Applied Optics, Vol. 19, pp. 1758-1761, (1980).

[17] Z. Bahri and B. V. K. Kumar, "Generalized synthetic discriminant functions", Journal of Optical Society of America, Vol. 5, No. 4, pp. 562-571, (1988).

[18] B. Javidi and J. Wang, "Optimum distortion invariant filters for detecting a noisy distorted target in background noise", Journal of Optical Society of America, Vol. 12, 2604-2614, (1984).

[19] Saad Rehman, Peter Bone, Rupert Young, Chris Chatwin, “Object Detection and Recognition in Cluttered Scenes Using Fully Scale and In-Plane Invariant Synthetic Discriminant Function Filters”, Journal of Computer and Information Sciences, Vol 1, No 1, 2007.

[20] Peter Bone, Rupert Young, Chris Chatwin, "Position, rotation, scale and orientation invariant multiple object recognition from cluttered scenes", Optical Engineering, Volume 45, pp. 077203-1 to 8, No. 7, 2006

[21] K. Raghunath Rao and Jezekiel Ben-Arie,, “Multiple Template Matching Using the Expansion Filter”, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, VOL. 4, NO. 5, OCTOBER 1994.

[22] Kyu-Hwa Jeong, Puskal P. Pokharel, Jian-Wu Xu, Seungju Han, Jose C. Principe,” Kernel Based Synthetic Discriminant Function for Object Recognition”, ICASSP 2006.

[23] D. Marr, E. Hildreth, “Theory of edge detection”, Proc. R. Soc. Lon. B 20, pp.187-217, (1980).

[24] Mariusz Rawski, Bogdan J. Falkowski, and Tadeusz Łuba “Digital Signal Processing Designing for FPGA Architectures” ELEC. ENERG. vol. 20, no. 3, December 2007, 437-459

[25]. Digital Design of Signal Processing Systems: A Practical Approach, First Edition. Shoab Ahmed Khan._ 2011 John Wiley & Sons, Ltd. Published 2011 by John Wiley & Sons, Ltd.

[26] http://www.xilinx.com

[27] Hen-Geul Yeh, "Kalman filtering and systolic processors," Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP '86. , vol.11, no., pp. 2139-2142, Apr 1986

[28] Bigdeli, A., Biglari-Abhari, M., Salcic, Z., and Lai, Y. T. 2006. A new pipelined systolic array-based architecture for matrix inversion in FPGAS with Kalman filter case study. EURASIP J. Appl. Signal Process. 2006, 1 (Jan. 2006), 75-75. DOI= http://dx.doi.org/10.1155/ASP/2006/89186

[29] Rao, P.; Bayoumi, M.A., "An efficient VLSI implementation of real-time Kalman filter," Circuits and Systems, 1990., IEEE International Symposium on , vol., no., pp.2353-2356 vol.3, 1-3 May 1990

[30] Bonato, Vanderlei; Marques, Eduardo; Constantinides, George A., "A Floating-Point Extended Kalman Filter Implementation for Autonomous Mobile Robots," Field Programmable Logic

[31] Baheti, R.S.; O'Hallaron, D.R.; Itzkowitz, H.R., "Mapping extended Kalman filters onto linear arrays," Automatic Control, IEEE Transactions on , vol.35, no.12, pp.1310-1319, Dec 1990

[32] Janier Arias-Garc´ıa1; Ricardo Pezzuol Jacobi2; Carlos H. Llanos1; Mauricio Ayala-Rinc´on21, A Suitable Fpga Implementation of Floating-Point Matrix Inversion based on Gauss-Jordan Elimination 978-1-4244-8848-3/11 ©2011 IEEE

[33] B.V.K. Kumar and L. Hassebrook, “Performance measures for correlation filters”, Applied Optics, Vol. 29, No. 20, pp. 2997-3006, (1990).