Optimal 2D DFT with sliding window for Image Edge Detection

A DISSERTATION

Submitted in partial fulfillment of the

requirements for the award of the degree

of

MASTER OF TECHNOLOGY

in

ELECTRICAL ENGINEERING

(With specialization in Instrumentation and Signal Processing)

By

AMIT KUMAR VISHWAKARMA

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY ROORKEE

ROORKEE - 247 667 (INDIA)

June 2019

CANDIDATE'S DECLARATION

I hereby declare that this report entitled **Optimal 2D DFT with sliding window for Image Edge Detection**, submitted to the Department of Electrical Engineering, Indian Institute of Technology, Roorkee, India, in partial fulfilment of the requirements for the award of the Degree of Master of Technology in Electrical Engineering with specialization in Instrumentation and Signal Processing is an authentic record of the work carried out by me during the period June 2018 through June 2019, under the supervision of **Dr. P. SUMATHI, Department of Electrical Engineering, Indian Institute of Technology, Roorkee.** The matter presented in this report has not been submitted by me for the award of any other degree of this institute or any other institutes.

Date:

Place: Roorkee

AMIT KUMAR VISHWAKARMA

CERTIFICATE

This is to certify that the above statement made by the candidate is true to the best of my knowledge and belief.

Dr. P. SUMATHI
Associate Professor
Department of Electrical Engineering
Indian Institute of Technology Roorkee

ABSTRACT

An edge detection technique based on two dimensional sliding window Discrete Fourier transform (2D SDFT) and thresholding is proposed in this work. The 2×2 sliding window DFT with bin indices $(k_1, k_2 = 0, 1)$ for horizontal edge detection and $(k_1, k_2 = 1, 0)$ for vertical edge detection has been proposed.

In 2D SDFT, the DFT bins at the current position of window are directly computed from the already computed bins of the previous position of window. These computed DFT bins are thresholded against a threshold value to obtain the edge map of input image. The output edge map of the proposed technique is equivalent to that of the traditional techniques. In the presence of noise and various signal to noise ratio conditions, the horizontal and vertical edges have been efficiently recovered with good Pratt figure of merit (PFOM) without any application of pre-processing and post processing techniques.

2D HDFT is also derived in this dissertation work. In 2D HDFT, the window will hope with hopping distance, L. Output of 2D HDFT and 2D SDFT with window size 4×4 and hopping distance $L = 2$ are compared. Output od both techniques are similar.

The system-on-chip implementation of the 2D SDFT/HDFT edge detector on cyclone IV FPGA is also carried out.

Acknowledgements

In addition to the efforts of myself, the accomplishment of this dissertation largely relies on many other's encouragement and collaboration. I would like to take this opportunity to express my gratitude to those people who have helped to complete this dissertation effectively.

I want to express my deep sense of gratitude and sincere thanks to my guide **Dr. P. Sumathi**, Associate Professor, Department of Electrical Engineering, Indian Institute of Technology Roorkee, for her tremendous support , help and for providing the excellent laboratory facilities. In spite of her hectic schedule, she was always available to address my doubts and reviewed constructively at the advancement of my dissertation. I am deeply indebted to my guide, who gave encouragement and constructive criticism throughout the course of this project. Without her encouragement and guidance, this dissertation would not have been conceivable.

I am thankful to all the faculty members of Electrical Engineering Department, IITR for their support. I am also thankful the technical and office staff of this department for the cooperation and help.

I am also grateful to all my friends who help me (directly or indirectly) to complete this dissertation work within a limited time frame.

I wish to express my hearty gratitude to all my family members for their support and encouragement. Finally, I am very thankful to God who gave me the blessing to carry out this dissertation work.

Amit Kumar Vishwakarma

Contents

Candidate's Declaration i

Abstract ii

Acknowledgements iii

List of Figures vi

List of Tables viii

Abbreviations ix

1 Introduction 1
- 1.1 Literature Survey . . . 1
- 1.2 Objectives of Dissertation Work . . . 4
- 1.3 Organisation of Report . . . 4

2 Proposed Edge Detector 5
- 2.1 Building Blocks of the Proposed Edge Detector . . . 6
  - 2.1.1 Input Image . . . 6
  - 2.1.2 2D DFT with Sliding Window . . . 6
    - 2.1.2.1 2D DFT Equation . . . 7
    - 2.1.2.2 2D SDFT . . . 7
    - 2.1.2.3 2D HDFT . . . 11
    - 2.1.2.4 Computational Complexity Analysis of SDFT . . . 13
    - 2.1.2.5 Computational Complexity Analysis of 2D HDFT . . . 15
  - 2.1.3 Absolute Value . . . 17
  - 2.1.4 Thresholding . . . 17
  - 2.1.5 Edge Map . . . 17

3 Simulation and Performance Evaluation 18
- 3.1 Simulation Models . . . 18
  - 3.1.1 Simulation Model for 2D - SDFT . . . 18

3.1.2 Simulation Model for 2D - HDFT 21
3.2 Simulation result 21
3.3 Performance Evaluation 25
3.3.1 Visual comparison of output with conventional techniques 25
3.3.2 PFOM comparison of output with conventional techniques 31
3.3.2.1 Pratt's figure of merit (PFOM) 31
3.3.2.2 Performance of proposed technique for image with no noise 31
3.3.2.3 Performance of proposed technique for image with noise . 32

4 Real-Time Implementation on FPGA Board 35
4.1 FPGA Board DE2-115 36
4.2 Camera Daughter Board TRDB-D5M 38
4.3 Implementation of proposed edge detector on FPGA 39
4.3.1 Camera Capture Module 40
4.3.2 Raw to RGB Data Converter Module 40
4.3.3 VGA Controller Module 42
4.3.4 SDRAM Controller Module 42
4.3.5 PLL Module 43
4.3.6 I2C sensor configuration Module 43
4.3.7 Proposed Edge Detector Module 44
4.3.8 Top Module 45
4.4 Experimental Setup 45

5 Conclusion and future work 50

Publications 52

Bibliography 53

A Verilog Files for Implementation 57
A.1 Top Module 57
A.2 Camera Capture Module 60
A.3 I2C CCD Config Module 62
A.4 Raw to RGB Module 65
A.5 SDRAM Controller Module 66
A.6 Subsystem Module 73
A.7 VGA Controller 79

List of Figures

2.1 Block Diagram of the proposed Edge Detector 5
2.2 Kernels for 2×2 window (a) for horizontal edges (bin(0,1)) (b) for vertical edges (bin(1,0)) 7
2.3 Graphical representation of window $N \times N = 2 \times 2$. 10
2.4 Single bin 2D SDFT structure. 10
2.5 Single bin 2D HDFT structure with L=2 and N=4. 13

3.1 SimuLink Model for computation single bin $(k_1, k_2 = 1, 0)$ 2D SDFT for window Size 2×2 and image size 8×8 19
3.2 2D-SDFT Model Output comparison with 2D DFT equation output . . . 20
3.3 SimuLink Model of Edge detector based on 2D SDFT for image size 640×480 22
3.4 Input and Outputs of the 2D-SDFT based Edge Detection Model 23
3.5 SimuLink Model of Edge detector based on 2D HDFT for image size 640×480 24
3.6 Input and Outputs of the 2D-SDFT and 2D-HDFT Model 25
3.7 Simulation results for some Road images - Input images are taken from ROMA database[35] 26
3.8 Simulation result and comparison with conventional techniques- Input images are taken from ROMA database [35] 27
3.9 Simulation result and comparison with conventional techniques- Input images are taken from ROMA database [35] 28
3.10 Simulation result and comparison with conventional techniques- Input images are taken from road marking dataset [36] 29
3.11 Simulation result and comparison with conventional techniques- Input images are taken from road marking dataset [36] 30
3.12 Input Parameter Selection for maximum PFOM for conventional method 31
3.13 Performance Evaluation of proposed technique without Noise 32
3.14 PFOM Vs threshold level in different SNR condition 33
3.15 Performance Evaluation of proposed technique with Noise 34

4.1 Connection Diagram of monitor and camera eith DE2-115 FPGA board . 35
4.2 FPGA Board DE2-115 36
4.3 Camera Board TRDB-D5M 38
4.4 Pixel array description 38
4.5 Block Diagram for Sequence of operations 39
4.6 Block Diagram for implementation on FPGA 41

4.7 Model for estimation of edge map based 2D-SDFT 46
4.8 Model for estimation of edge map based 2D-HDFT 47
4.9 Experimental Setup for implementation on FPGA 48
4.10 Detected Edge Image on VGA monitor 49

List of Tables

2.1 Computational Complexity of Different Algorithms 15
2.2 Comparision of Computational Complexity of 2D SDFT and 2D HDFT Algorithms 16

4.1 Key performance parameter of TRDB-D5M 39
4.2 Utilization Summary of 2D SDFT & 2D HDFT for 4×4 window size on Cyclone IV FPGA 49

Abbreviations

2D	**T**wo **D**imensional
DFT	**D**iscrete **F**ourier **T**ransform
FFT	**F**ast **F**ourier **T**ransform
FPGA	**F**ield **P**rogrammable **G**ate **A**rray
FIFO	**F**irst **I**n **F**irst **O**ut
GSFFT	**G**eneralized **SFFT**
HDFT	**H**opping **DFT**
I2C	**I**nter-**I**ntegrated **C**ircuit
JTAG	**J**oint **T**est **A**ction **G**roup
LED	**L**ight **E**mitting **D**iode
LCD	**L**iquid **C**rystal **D**isplay
PLL	**P**hase **L**ocked **L**oop
PFOM	**P**ratt **F**igure **O**f **M**erit
ROMA	**RO**ad **MA**arking
RGB	**R**ed **G**reen **B**lue
SDRAM	**S**ynchronous **D**ynamic **R**andom **A**ccess **M**emory
SRAM	**S**tatic **R**andom **A**ccess **M**emory
SFFT	**S**liding **F**ast **F**ourier **T**ransform
SDFT	**S**liding **D**iscrete **F**ourier **T**ransform
SNR	**S**ignal to **N**oise **R**atio
VGA	**V**ideo **G**raphics **A**rray
USB	**U**niversal **S**erial **B**us

Chapter 1

Introduction

Edge in an image is an outline representing the quick changes in the image intensity. In general, it is seen that the borders of objects produce quick changes in the image intensity. For example, different objects in an image are commonly with distinct colors, textures or hues and this produces the variation in the intensity of the image from one object to another. Also, different surfaces of an object do not receive same amounts of light, which again produces intensity variations.

Edge detection is an important process in image processing, machine vision and automated vehicles etc. as it can define significant variation in the intensity of the image[1],[2]. Many researchers have been worked in this field and they have made many achievements.

1.1 Literature Survey

Different type of edge detector are studied in this section.

With the time, many classical edge detectors are developed. For edge detection, Image is convolved with a directional derivative mask. Roberts, Sobel, and Prewitt edge detection

operators are popular in image processing [3],[4],[5],[6]. They are simple in computation. However, images with strong noise can not be handled by classical edge detectors to reduce the effect of noise.

Edge detectors based on Gaussian function are commonly used in the field of image processing. A Gaussian filter is used in this type of edge detector to reduce the noise effect. The Gaussian filtering of the image before edge detction in suggested by Marr [7]. This method is normally called Laplacian of Gaussian (LOG). LOG method does not perform well over edges and corners. A computational methodology to detect the edges was developed by Canny in [8]. He derived that an optimal detector can be approximated by the first derivative of a Gaussian. Canny algorithm does not perform well for blurred and shaded images.

In multi resolution method, repetition of edge detection process for different scales are used. Edges at lower resolution level are less noisy, but poorly localized. Edges at the lower resolution level will then reinforce the corresponding edges at higher resolution level. Edges at the finer scale which are not reinforced will tend to disappear. Many algorithms were proposed in this area. A new multi-resolution analysis algorithm is given in [9]. This method is based on Canny algorithm. At the higher scale, only strong edges are detected and number of edges is reduced. At the lower scale, strong and weak edges are detected with increase in number of edges. This algorithm losses the important information at higher scale. A new algorithm based on Gaussian filter is proposed in [10]. It is known as edge focussing. It has drawback of coarse edge splitting into fine edges. The problem of edge splitting while moving from high to low resolution is resolved in [11].

Wavelet type edge detectors are also available. Wavelet transform represents an image into two terms - shift and scale. Shift defines frequency domain, while scale defines spatial domain. Based on Haar wavelet transform, an edge detector is proposed in [12].

Genetic algorithm and fuzzy logic based edge detector were also proposed by many researchers. Based on Genetic algorithm as an optimization technique, edge detection was suggested in [13] and also proposed in [14] for medical imaging. In [15], the fuzzy reasoning-based algorithm of the Fuzzy Sobel method is introduced in which four

threshold values are obtained automatically, and fuzzy reasoning is applied for edge enhancement. The edges extracted by this method are very clear and provides better representation for image edges and object contours. An improvement is proposed in Sobel edge detector with wavelet technique based on soft threshold[16]. The problems encountered in FPGA implementation of Sobel algorithm has been solved and improved through gradient calculation template with increased processing speed [17]. An improvement in Canny edge detection algorithm [18] is proposed towards improving the sensitivity of noise and lose of weak edge information. The idea of gravitative field intensity was introduced to change gradient of intensity of the image. Two versatile limit determination techniques dependent on the mean standard deviation of intensity slope of image were used to enhance the noise suppression. A multilevel fuzzy edge detection for blurry images was proposed in [19]. This algorithm has two steps. First step is fast multilevel fuzzy enhancement algorithm to improve contrast and second step is the edge extraction on gradient values. A robust edge detection algorithm is proposed in [20] to cover the limitations of edge connectivity. Apart from the conventional edge detectors, the Hough transform is a much applied algorithm in lane detection and tracking systems. The computationally efficient hierarchical additive Hough transform is proposed for detecting only the straight lanes [21]. An efficient method for reliably detecting road lanes based on spatiotemporal images is proposed using Hough transform [22]. The existing edge detectors requires pre-processing like Gaussian filtering in Canny and postprocessing such as thinning in Canny, Sobel and Prewitt edge detectors

The edge detection is a linear filtering process that convolve the image pixel and kernel. In 1-D sense, DFT is a linear filter that performs the correlation between signal samples and basis function to extract the fundamental and harmonics. The 1-D sliding DFT is computationally efficient as tuned filter to extract the particular fundamental and harmonic of interest from the input signal [23], [24]. Moreover, the instantaneous values of fundamental and harmonic can be extracted from the filter operation [25]. In 2-D, similar to classical edge detectors, the 2-D sliding DFT performs correlation between image pixel and kernel, in which correlation is the convolution of image pixel and 180° rotated kernel [26]. The 2-D SDFT can be tuned to extract only edge frequencies from the image and reject all other frequencies including high noise levels. Therefore, with

this motivation, an edge detector is proposed based on 2-D sliding DFT and applied for road marking images in order to create an edge map. These edge maps can be employed in lane detection, tracking systems, and driver assistant systems.

1.2 Objectives of Dissertation Work

The objectives of this dissertation work include:

1. Propose an edge detection scheme based on 2D SDFT.
2. Simulate the edge detector with different test images.
3. Evaluate the performance of the proposed scheme.
4. Implement the proposed scheme on FPGA.

1.3 Organisation of Report

This report is organized as follows: Chapter 2 describes the proposed Edge detector based on 2D SDFT estimation. Simulation and performance evaluation investigation carried out on the proposed scheme are discussed in Chapter 3. Implementation of proposed scheme on FPGA is described in chapter 4. Conclusions and scope for future work are stated in Chapter 5.

Chapter 2

Proposed Edge Detector

The proposed Edge detector is a mechanism based on 2D SDFT estimation where one Window $N \times N$ moves horizontally row-wise over the image. Absolute value of Bin $(0, 1)$ and Bin $(1, 0)$ is computed for determining the horizontal and vertical edges respectively. To find the non-directional edges, root of sum of square of both bin values is calculated. This estimated value is compared against a threshold to define the edge map. This section explains the basic building blocks and the working of the proposed scheme.

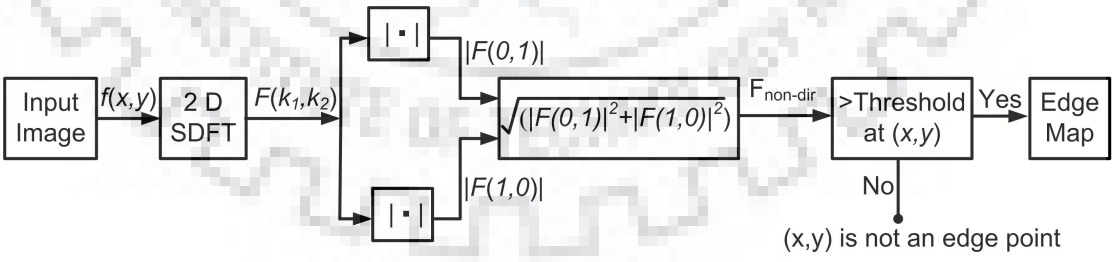


FIGURE 2.1: Block Diagram of the proposed Edge Detector

2.1 Building Blocks of the Proposed Edge Detector

The block diagram of this scheme is shown in Fig.(2.1). The basic building blocks used in the proposed edge detector are:

1. Input Image
2. 2D Sliding DFT (2D SDFT)
3. Absolute Value
4. Thresholding
5. Edge Map

The working of each of these blocks are discussed in detail in the following sections.

2.1.1 Input Image

A Input image is a 2D array of pixel data of the image to be processed. This array is used in the proposed scheme to compute the 2D SDFT. If image is in RGB form, first it is converted to gray scale to compute the 2D DFT with sliding window.

2.1.2 2D DFT with Sliding Window

For detecting the edges of an image, 2D DFT with sliding window is computed for the bin(0,1) and (1,0) for determining the horizontal and vertical edges respectively. In this project, window size is taken 2×2. Kernels for horizontal and vertical edges can be computed for bin(0,1) and bin (1,0) using window size 2×2. These kernels are given in Fig.(2.2). These kernels calculate the slope at the current pixel using three previous pixel. 2D DFT with Sliding Window based method of edge detection is basically gradient type edge detection. 2D DFT with sliding window of an image can be estimated by using 2D DFT equation, 2D SDFT or 2D HDFT and these are described in following sections.

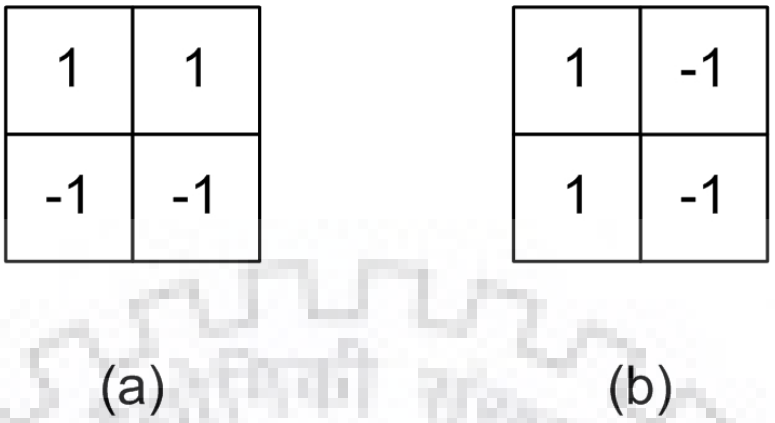


FIGURE 2.2: Kernels for 2×2 window (a) for horizontal edges (bin(0,1)) (b) for vertical edges (bin(1,0))

2.1.2.1 2D DFT Equation

In the 2D SDFT, the transform is determined on a window of pixels having fixed-size, which is normally refreshed with new pixels and the earliest ones are disposed of. For example, an window of size $N \times N$ is moved in the horizontally on the image row by row. Let us assume that $f(x, y)$ represents the pixel (x, y) of an input image and $F_{x,y}(k_1, k_2)$ represents the bin (k_1, k_2) of the $N \times N$ order DFT at pixel position (x, y) [28], which is represented by

$$F_{x,y}(k_1, k_2) = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} f_{\hat{x}+m,\hat{y}+n} W_N^{-k_1 m} W_N^{-k_2 n} \tag{2.1}$$

where $\hat{x} = x - N + 1$, $\hat{y} = y - N + 1$, $W_N = e^{j2\pi/N}$, and $k_1, k_2 = 0, 1, ..., N - 1$.

2.1.2.2 2D SDFT

We can drive the mathematical expression for the 2D SDFT as follows: For the pixel(x,y) 2D SDFT expression is given in Equation (2.1). For the pixel (x+1,y), we can write the equation as

$$F_{x+1,y}(k_1, k_2) = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} f_{\hat{x}+m+1,\hat{y}+n} W_N^{-k_1 m} W_N^{-k_2 n} \tag{2.2}$$

Letting p=m+1 in Equation (2.2), we can write

$$F_{x+1,y}(k_1,k_2) = \sum_{p=1}^{N}\sum_{n=0}^{N-1} f_{\hat{x}+p,\hat{y}+n} W_N^{-k_1(p-1)} W_N^{-k_2 n} \tag{2.3}$$

By shifting the limits of summation in Equation (2.3). We can rewrite the Equation (2.3) as

$$\begin{aligned} F_{x+1,y}(k_1,k_2) =& \sum_{p=0}^{N-1}\sum_{n=0}^{N-1} f_{\hat{x}+p,\hat{y}+n} W_N^{-k_1(p-1)} W_N^{-k_2 n} \\ &+ \sum_{n=0}^{N-1} f_{\hat{x}+N,\hat{y}+n} W_N^{-k_1(N-1)} W_N^{-k_2 n} \\ &- \sum_{n=0}^{N-1} f_{\hat{x},\hat{y}+n} W_N^{k_1} W_N^{-k_2 n} \end{aligned} \tag{2.4}$$

$$\begin{aligned} =& W_N^{k_1} \sum_{p=0}^{N-1}\sum_{n=0}^{N-1} f_{\hat{x}+p,\hat{y}+n} W_N^{-k_1 p} W_N^{-k_2 n} \\ &+ \sum_{n=0}^{N-1} f_{\hat{x}+N,\hat{y}+n} W_N^{k_1} W_N^{-k_2 n} \\ &- \sum_{n=0}^{N-1} f_{\hat{x},\hat{y}+n} W_N^{k_1} W_N^{-k_2 n} \end{aligned} \tag{2.5}$$

where $W_N^{-k_1(N-1)} = W_N^{k_1}$ by the periodicity property of the complex twiddle factor. Now from Equation (2.1) and (2.5), we can write

$$\begin{aligned} F_{x+1,y}(k_1,k_2) =& W_N^{k_1} F_{x,y}(k_1,k_2) \\ &+ W_N^{k_1} \sum_{n=0}^{N-1} f_{\hat{x}+N,\hat{y}+n} W_N^{-k_2 n} \\ &- W_N^{k_1} \sum_{n=0}^{N-1} f_{\hat{x},\hat{y}+n} W_N^{-k_2 n}. \end{aligned} \tag{2.6}$$

Without loss of generality, we can rewrite Equation (2.6) as

$$\begin{aligned} F_{x,y}(k_1,k_2) =& W_N^{k_1} F_{x-1,y}(k_1,k_2) \\ &+ W_N^{k_1} \sum_{n=0}^{N-1} f_{\hat{x}+N-1,\hat{y}+n} W_N^{-k_2 n} \\ &- W_N^{k_1} \sum_{n=0}^{N-1} f_{\hat{x}-1,\hat{y}+n} W_N^{-k_2 n} \qquad (2.7) \\ =& W_N^{k_1} F_{x-1,y}(k_1,k_2) \\ &+ W_N^{k_1} \sum_{n=0}^{N-1} [f_{\hat{x}+N-1,\hat{y}+n} - f_{\hat{x}-1,\hat{y}+n}] W_N^{-k_2 n}. \qquad (2.8) \end{aligned}$$

Let us assume $d_{x,y} = f_{\hat{x}+N-1,\hat{y}+n} - f_{\hat{x}-1,\hat{y}+n}$ and its 1D DFT is $D_{x,y}(k_2)$ which can be written as

$$D_{x,y}(k_2) = \sum_{n=0}^{N-1} d_{x,\hat{y}+n} W_N^{-k_2 n} \qquad (2.9)$$

By using Equation (2.8) and (2.9), we can write

$$\begin{aligned} F_{x,y}(k_1,k_2) =& W_N^{k_1} F_{x-1,y}(k_1,k_2) + W_N^{u} D_{x,y}(k_2) \qquad (2.10) \\ =& W_N^{k_1} [F_{x-1,y}(k_1,k_2) + D_{x,y}(k_2)]. \qquad (2.11) \end{aligned}$$

$D_{x,y}(v)$ can be calculated by following expression of 1D SDFT

$$D_{x,y}(k_2) = W_N^{k_2} [D_{x,y-1}(v) + d_{x,y} - d_{x,y-N}] \qquad (2.12)$$

By using Equation (2.12) we can rewrite the Equation (2.11) as

$$\begin{aligned} F_{x,y}(k_1,k_2) =& W_N^{k_1} [F_{x-1,y}(k_1,k_2) + W_N^{k_2} [D_{x,y-1}(k_2) \\ &+ d_{x,y} - d_{x,y-N}]]. \qquad (2.13) \end{aligned}$$

The above Equation (2.13) indicates that the 2D SDFT at any pixel can be directly calculated from that at previous pixel by using 1D DFT. Fig. (2.3) shows an example of window 2×2. The single bin 2D-SDFT implementation is shown in Fig.(2.4).

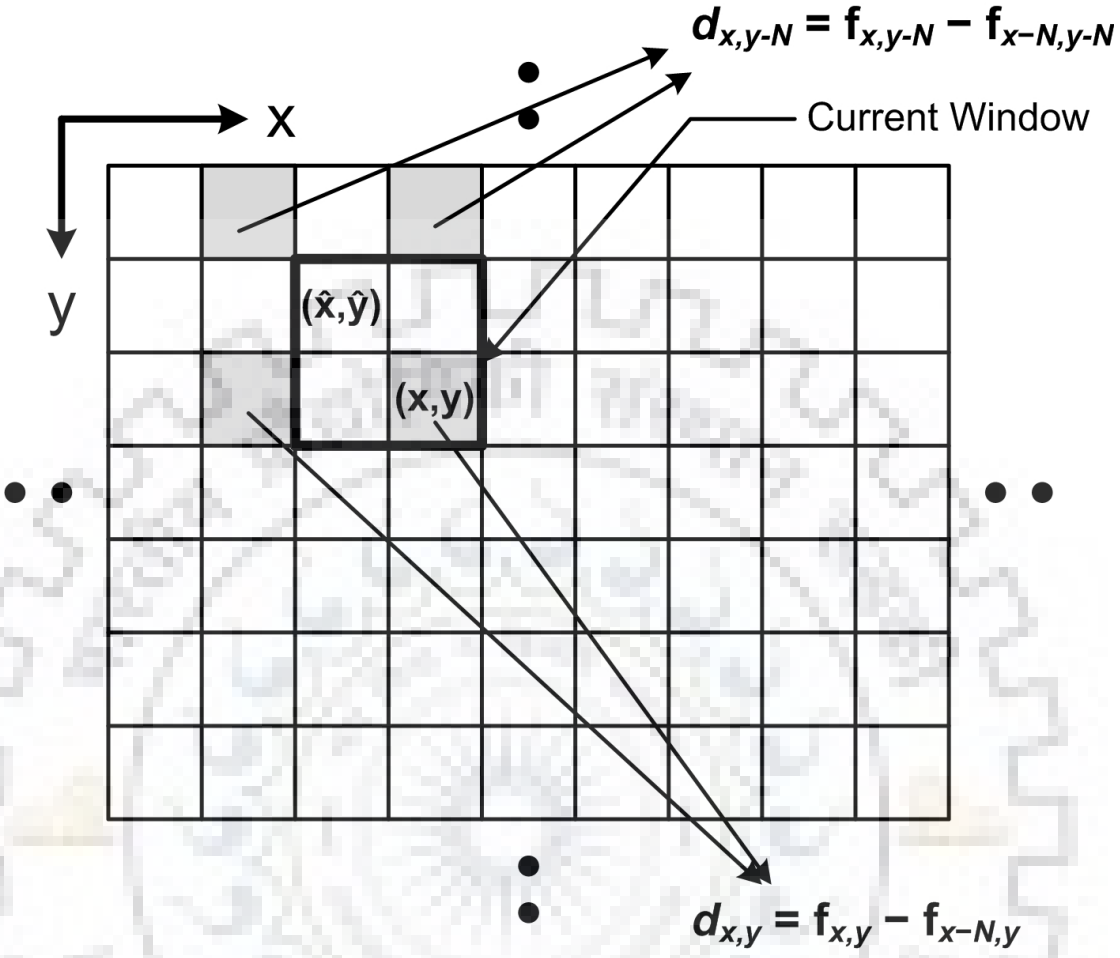


FIGURE 2.3: Graphical representation of window $N \times N = 2 \times 2$.

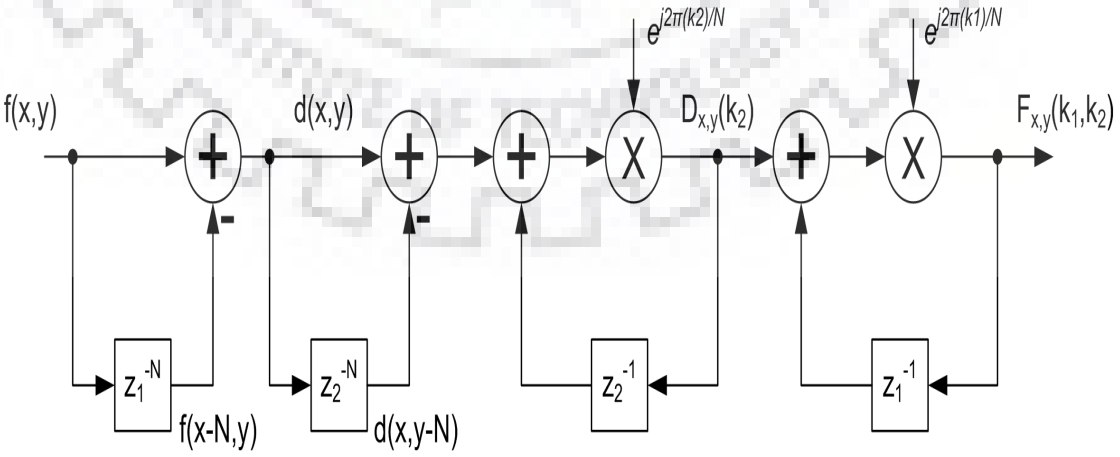


FIGURE 2.4: Single bin 2D SDFT structure.

z-Transform of $d_{x,y} = f_{x,y} - f_{x-N,y}$, Equation (2.11) and (2.12) can be written as

$$D(z_1, z_2) = F(z_1, z_2) - F(z_1, z_2)z_1^{-N} = F(z_1, z_2)(1 - z_1^{-N}) \tag{2.14}$$

$$D^{k_2}(z_1, z_2) = W_N^{k_2}[D^{k_2}(z_1, z_2)z_2^{-1} + D(z_1, z_2) - D(z_1, z_2)z_2^{-N}] \tag{2.15}$$

$$F^{k_1,k_2}(z_1, z_2) = W_N^{k_1}[F^{k_1,k_2}(z_1, z_2)z_1^{-1} + D^v(z_1, z_2)] \tag{2.16}$$

By using above Equations (2.14),(2.15) and (2.16), transfer function in z-domain for 2D SDFT can be written as

$$H_{2D-SDFT}(z_1, z_2) = \frac{W_N^{k_1} W_N^{k_2}(1 - z_1^{-N})(1 - z_2^{-N})}{(1 - W_N^{k_1} z_1^{-1})(1 - W_N^{k_2} z_2^{-1})} \tag{2.17}$$

2.1.2.3 2D HDFT

From Equation (2.9) and (2.11) can be rewritten as

$$F_{x,y}(k_1, k_2) = W_N^{k_1}[F_{x-1,y}(k_1, k_2) + \sum_{n=0}^{N-1} d_{x,\hat{y}+n} W_N^{-k_2 n}]. \tag{2.18}$$

The relationship between $F_{x,y}$ and $F_{x-L,y}$ is derived by recursively substituting $F_{x,y}$ into $F_{x-1,y}$ L times in equation(2.18)[27]. The resultant formula is given by

$$\begin{aligned} F_{x,y}(k_1, k_2) = W_N^{Lk_1}[F_{x-L,y}(k_1, k_2) &+ \sum_{n=0}^{N-1} d_{x-L+1,\hat{y}+n} W_N^{-k_2 n} \\ &+ W_N^{-k_1} \sum_{n=0}^{N-1} d_{x-L+2,\hat{y}+n} W_N^{-k_2 n} + ... \\ &+ W_N^{-(L-1)k_1} \sum_{n=0}^{N-1} d_{x,\hat{y}+n} W_N^{-k_2 n}]. \end{aligned} \tag{2.19}$$

Equation (2.19) can be written as

$$\begin{aligned} F_{x,y}(k_1, k_2) = W_N^{Lk_1}[F_{x-L,y}(k_1, k_2) &+ \\ \sum_{n=0}^{N-1} \sum_{p=0}^{L-1} d_{x-p,\hat{y}+n} W_N^{(p-L+1)k_1} W_N^{-k_2 n}]. \end{aligned} \tag{2.20}$$

Second term in above equation is 1D DFT. Let this is $D_{x,y}(v)$.

$$D_{x,y}(k_2) = \sum_{n=0}^{N-1} \sum_{p=0}^{L-1} d_{x-p,\hat{y}+n} W_N^{(p-L+1)k_1} W_N^{-k_2 n}] \tag{2.21}$$

Equation (2.21)can be written in the form of 1D SDFT as follows

$$\begin{aligned} D_{x,y}(k_2) =& W_N^{k_2}[D_{x,y-1}(k_2) + \sum_{p=0}^{L-1} d_{x-p,y} W_N^{(p-L+1)k_1} \\ & - \sum_{p=0}^{L-1} d_{x-p,y-N} W_M^{(p-L+1)k_1}] \\ =& W_N^{k_2}[D_{x,y-1}(k_2) \\ & + \sum_{p=0}^{L-1} (d_{x-p,y} - d_{x-p,y-N}) W_N^{(p-L+1)k_1}] \end{aligned} \tag{2.22}$$

Let $e_{x,y} = d_{x,y} - d_{x,y-N}$. Now above Equation (2.22) can be written as

$$D_{x,y}(k_2) = W_N^{k_2}[D_{x,y-1}(k_2) + \sum_{p=0}^{L-1} (e_{x-p,y}) W_N^{(p-L+1)k_2}] \tag{2.23}$$

The relationship between $D_{x,y}$ and $D_{x,y-L}$ is derived by recursively substituting $D_{x,y}$ into $D_{x,y-1}$ L times in equation(2.23) [27]. The resultant formula is given by

$$\begin{aligned} D_{x,y}(k_2) =& W_N^{Lk_2}[D_{x,y-L}(k_2) + \sum_{p=0}^{L-1} [e_{x-p,y-L+1} \\ & + W_N^{-k_2} e_{x-p,y-L+2} + \cdots \\ & + W_N^{-(L-1)k_2} e_{x-p,y}] W_N^{(p-L+1)k_1}] \end{aligned} \tag{2.24}$$

Equation (2.24) can be rewritten as

$$\begin{aligned} D_{x,y}(k_2) =& W_N^{Lk_2}[D_{x,y-L}(k_2) + \\ & \sum_{p=0}^{L-1} \sum_{q=0}^{L-1} e_{x-p,y-q} W_N^{(q-L+1)k_2} W_N^{(p-L+1)k_1}] \end{aligned} \tag{2.25}$$

Finally we get two equations for computing of 2D HDFT as follows

$$F_{x,y}(k_1,k_2) = W_N^{Lk_1}[F_{x-L,y}(k_1,k_2) + D_{x,y}(k_2)]. \tag{2.26}$$

and

$$D_{x,y}(k_2) = W_N^{Lk_2}[D_{\boldsymbol{x},y-L}(k_2) + \sum_{p=0}^{L-1}\sum_{q=0}^{L-1} e_{x-p,y-q} W_N^{(q-L+1)k_2} W_N^{(p-L+1)k_1}] \tag{2.27}$$

The single bin 2D-HDFT implementation for N=4 and L=2 is shown in Fig.(2.5).

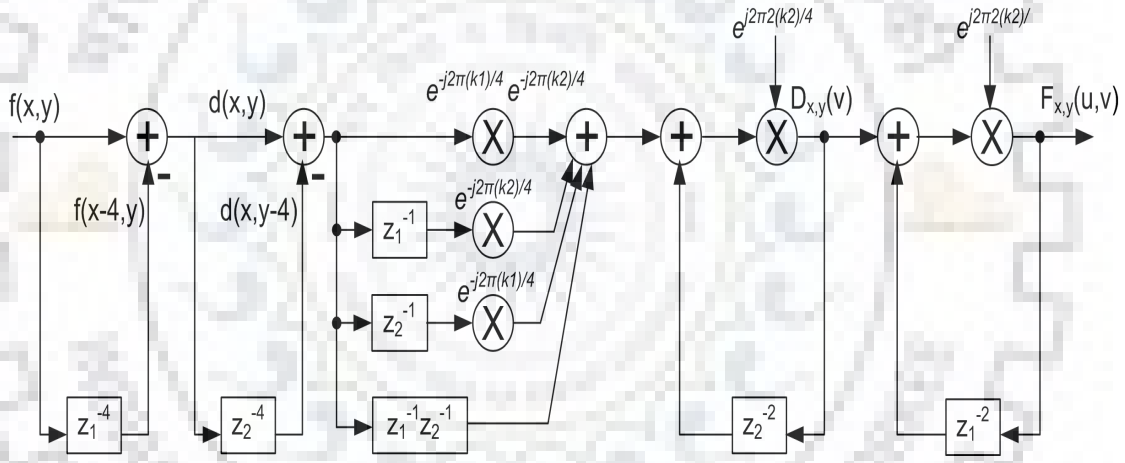


FIGURE 2.5: Single bin 2D HDFT structure with L=2 and N=4.

2.1.2.4 Computational Complexity Analysis of SDFT

For 2D SDFT algorithm with window size $N \times N$, computational requirements at the current pixel (x, y) are as shown below:

- Computation of $d_{x,y}$ require on real addition
- Computing $d_{x,y} - d_{x,y-N}$, requires one real addition because $d_{x,y-N}$ is already calculated.

- As per Equation(2.12), computation of N-point DFT $D_{x,y}(k_2)$ using $D_{x,y-1}(k_2)$ and $d_{x,y} - d_{x,y-N}$ requires N complex multiplications and N real additions.
- As per Equation(2.11), computation of the 2D DFT $F_{x,y}(k_1, k_2)$ using $F_{x-1,y}(k_1, k_2)$ and $D_{x,y}(k_2)$ requires N^2 complex multiplications and N^2 complex additions.

From above analysis, number of computations required for $N \times N$ 2D SDFT are summarized below:

$$Real\_Addition = 2 + N \tag{2.28}$$

$$Complex\_Addition = N^2 \tag{2.29}$$

$$Complex\_Multiplication = N^2 + N \tag{2.30}$$

One complex multiplication can be written in term of four real multiplication and two real addition. One complex addition can be written in terms of two real addition [29]. Hence, the number required of real additions is

$$Real\_Addition = 4N^2 + 3N + 2 \tag{2.31}$$

and real multiplication is

$$Complex\_Multiplication = 4(N^2 + N) \tag{2.32}$$

A comparison of computational complexity with different window size is presented in Table (2.1). The computations required for 2D SDFT [28], MFFT [33], vector radix FFT (VRFFT) [32], FFT [31], and DFT [30] are given. In 2D SDFT algorithm with window size 16×16, number of real multiplications is reduced by 5.56%, 64.58%, 73.44%, and 96.68% in comparison to the MFFT, VRFFT, FFT, and DFT, respectively. And, the number of real additions of the 2D SDFT is also reduced by 11.68%, 80.93%, 82.52%, and 96.62% in comparison of the MFFT, VRFFT, FFT, and DFT, respectively. As per the Table (2.1), 2D SDFT algorithm have the lowest computational complexity in the available algorithms.

TABLE 2.1: Computational Complexity of Different Algorithms

Algorithm	**Operation**	**Window size**			
		4×4	8×8	16×16	$N \times N$
DFT [30]	R_M	512	4096	32768	$8N^3$
	R_A	448	3840	31744	$8N^3 - 4N^2$
FFT [31]	R_M	128	768	4096	$4N^2 \log_2 N$
	R_A	192	1152	6144	$6N^2 \log_2 N$
VRFFT [32]	R_M	96	576	3072	$3N^2 \log_2 N$
	R_A	176	1056	5632	$(11/2)N^2 \log_2 N$
MFFT [33]	R_M	80	304	1152	$4(N^2 + (N/2) \log_2 N)$
	R_A	88	328	1216	$4N^2 + 3N \log_2 N$
SDFT [28]	R_M	80	288	1088	$4(N^2 + N)$
	R_A	78	282	1074	$4N^2 + 3N + 2$

2.1.2.5 Computational Complexity Analysis of 2D HDFT

For 2D HDFT algorithm with window size $N \times N$ and hopping distance L, computational requirements at the current pixel (x, y) are as shown below:

- Computation of $d_{x,y}$ require on real addition
- Computing $e_{x,y}$, requires one real addition because $d_{x,y-N}$ is already calculated.
- As per Equation(2.27), computation of N-point DFT $D_{x,y}(k_2)$ using $D_{x,y-L}(k_2)$ and $e_{x,y}$ requires NL^2 complex multiplications, $2N(L^2 - 1)$ real multiplication, $N(L^2 + L - 2)$ complex additions and N real additions.
- As per Equation(2.26), computation of the 2D DFT $F_{x,y}(k_1, k_2)$ using $F_{x-L,y}(k_1, k_2)$ and $D_{x,y}(k_2)$ requires N^2 complex multiplications and N^2 complex additions.

Hence, Total computation for 2D HDFT size $N \times N$ is given as:

$$R_A = 2 + N \tag{2.33}$$

$$C_A = N^2 + N(L^2 + L - 2) \tag{2.34}$$

$$R_M = 2N(L^2 - 1) \tag{2.35}$$

$$C_M = N^2 + NL^2 \tag{2.36}$$

Where, R_A, R_M, C_A and C_M are used to represent the number of real addition, complex addition, real multiplication, and complex multiplication respectively. One complex addition can can be replaced by two real addition and one complex multiplication can be replaced by four real multiplication and two real addition [29]. Hence, the number required of real additions is

$$R_A = 4N^2 + 4NL^2 + 2NL - 3N + 2 \tag{2.37}$$

and real multiplication is

$$R_M = 4N^2 + 4NL^2 + 2(L^2 - 1) \tag{2.38}$$

Computational complexity comparison of 2D SDFT and 2D HDFT algorithm is shown in Table (2.2).

TABLE 2.2: Comparision of Computational Complexity of 2D SDFT and 2D HDFT Algorithms

Window Size	L	**2D SDFT**		**2D HDFT**	
		R_A	R_M	R_A	R_M
16×16	1	1074	1088	1074	1088
	2	2148	2176	1298	1286
	4	4296	4352	2130	2078
	8	8592	8704	5330	5246

2.1.3 Absolute Value

This block is used to calculated absolute value of the complex 2D-DFT value for bin (0,1) and bin (1,0). These absolute value is used to calculate the non-directional edges by following equation.

$$F_{non-dir} = \sqrt{F(0,1)^2 + F(1,0)^2} \tag{2.39}$$

2.1.4 Thresholding

After computing the $F_{non-dir}$ using Equation (2.39) for each pixel of image, $F_{non-dir}$ is compared with a threshold to determine edge map of the image.

2.1.5 Edge Map

Output of last threshold block is plotted in 2D format to get the 2D Edge Map.

Chapter 3

Simulation and Performance Evaluation

The Model for proposed edge detector is made in MATLAB - SIMULINK ® and simulated for different images. Simulation model and simulation results are discussed in this chapter.

3.1 Simulation Models

3.1.1 Simulation Model for 2D - SDFT

A model for 2D-SDFT single bin calculation with window size 2×2 and image size 8×8 is made in Simulink and represented in Fig.(3.1). In this model, a 8×8 image is converted to a serial data in workspace and this serial data is used as simulation input. This model calculates the 2D SDFT of input image for a single bin $(k_1, k_2 = 1, 0)$ and output of model is saved in workspace. To verify this model, the out of this model is compared with the output of sliding window DFT calculated with DFT Equation(2.1). Both the outputs, on from 2D-SDFT and other from 2D-DFT are same as shown in the

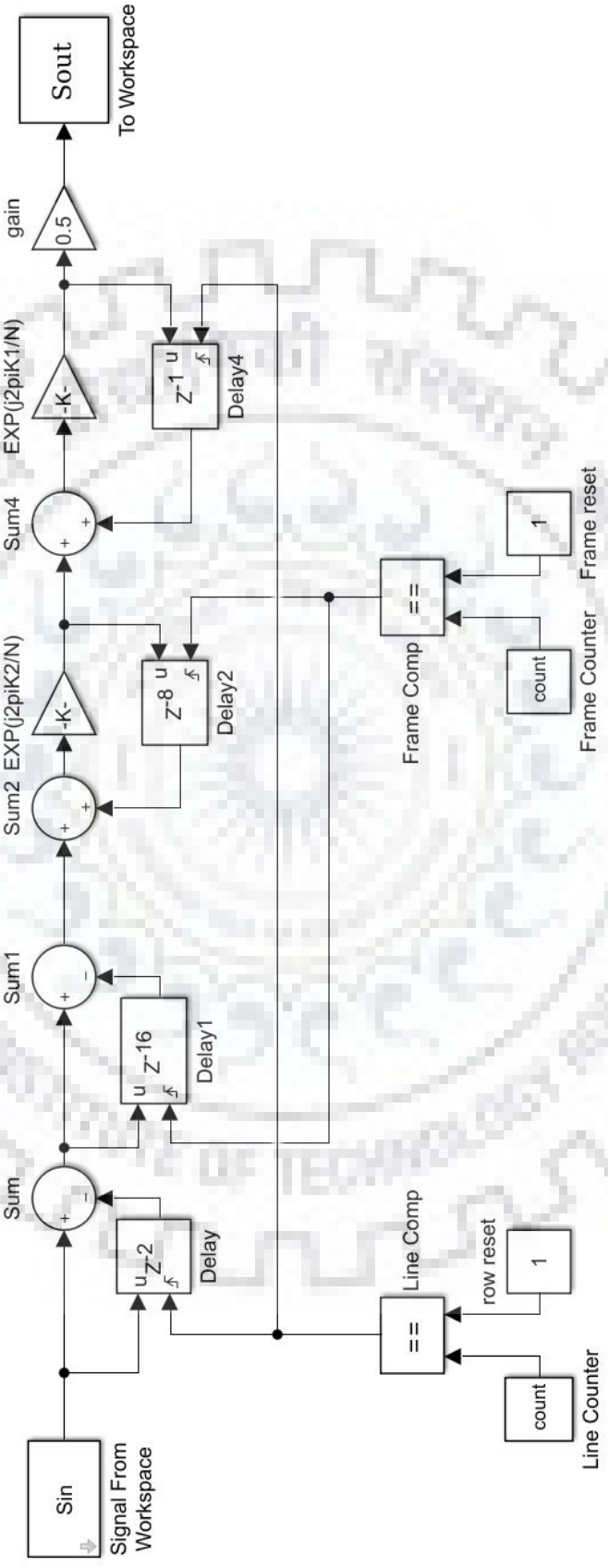


FIGURE 3.1: SimuLink Model for computation single bin $(k_1, k_2 = 1, 0)$ 2D SDFT for window Size 2×2 and image size 8×8

I

8x8 double

	1	2	3	4	5	6	7	8
1	1	2	3	4	5	6	7	8
2	2	3	4	5	6	7	8	1
3	3	4	5	6	7	8	1	2
4	4	5	6	7	8	1	2	3
5	5	6	7	8	1	2	3	4
6	6	7	8	1	2	3	4	5
7	7	8	1	2	3	4	5	6
8	8	1	2	3	4	5	6	7

(a) Input 8×8 image

Editor - DFT_8x8.m

Variables - DFT_ou

SDFT_out | DFT_out

8x8 double

	1	2	3	4	5	6	7	8
1	-0.5000	-0.5000	-0.5000	-0.5000	-0.5000	-0.5000	-0.5000	-0.5000
2	-1.5000	-1	-1	-1	-1	-1	-1	3
3	-2.5000	-1	-1	-1	-1	-1	3	3
4	-3.5000	-1	-1	-1	-1	3	3	-1
5	-4.5000	-1	-1	-1	3	3	-1	-1
6	-5.5000	-1	-1	3	3	-1	-1	-1
7	-6.5000	-1	3	3	-1	-1	-1	-1
8	-7.5000	3	3	-1	-1	-1	-1	-1

(b) Output of 2D-SDFT Model

Editor - DFT_8x8.m

Variables - SDFT_out

SDFT_out

8x8 complex double

	1	2	3	4	5	6	7	8
1	-0.5000 + 0.0000i	-0.5000 + 0.0000i	-0.5000 + 0.0000i	-0.5000 + 0.0000i	-0.5000 + 0.0000i	-0.5000 + 0.0000i	-0.5000 + 0.0000i	-0.5000 + 0.0000i
2	-1.5000 + 0.0000i	-1.0000 - 0.0000i	-1.0000 + 0.0000i	-1.0000 - 0.0000i	-1.0000 + 0.0000i	-1.0000 - 0.0000i	-1.0000 + 0.0000i	3.0000 - 0.0000i
3	-2.5000 + 0.0000i	-1.0000 - 0.0000i	-1.0000 + 0.0000i	-1.0000 - 0.0000i	-1.0000 + 0.0000i	-1.0000 - 0.0000i	3.0000 - 0.0000i	3.0000 - 0.0000i
4	-3.5000 + 0.0000i	-1.0000 - 0.0000i	-1.0000 + 0.0000i	-1.0000 - 0.0000i	-1.0000 + 0.0000i	3.0000 - 0.0000i	3.0000 + 0.0000i	-1.0000 - 0.0000i
5	-4.5000 + 0.0000i	-1.0000 - 0.0000i	-1.0000 + 0.0000i	-1.0000 - 0.0000i	3.0000 + 0.0000i	3.0000 - 0.0000i	-1.0000 + 0.0000i	-1.0000 - 0.0000i
6	-5.5000 + 0.0000i	-1.0000 - 0.0000i	-1.0000 + 0.0000i	3.0000 - 0.0000i	3.0000 + 0.0000i	-1.0000 - 0.0000i	-1.0000 + 0.0000i	-1.0000 - 0.0000i
7	-6.5000 + 0.0000i	-1.0000 - 0.0000i	3.0000 + 0.0000i	3.0000 - 0.0000i	-1.0000 + 0.0000i	-1.0000 - 0.0000i	-1.0000 + 0.0000i	-1.0000 - 0.0000i
8	-7.5000 + 0.0000i	3.0000 - 0.0000i	3.0000 + 0.0000i	-1.0000 - 0.0000i	-1.0000 + 0.0000i	-1.0000 - 0.0000i	-1.0000 + 0.0000i	-1.0000 - 0.0000i

(c) Output of DFT Equation Model

FIGURE 3.2: 2D-SDFT Model Output comparison with 2D DFT equation output

Fig.(3.2(b)) and in Fig.(3.2(c)). Based upon 2D-SDFT model shown in Fig. (3.1), a 2D-SDFT based edge detector model is made, with window size 2×2 and with input image size 640×480 depending on the camera size for real time implementation. This model is shown in Fig.(3.3). As shown, there are two blocks are used one for bin(0,1) for horizontal edges and other for bin (1,0) for vertical edges. All the components of the model are marked on the figure.

This model is used to extract the edges of the image shown in Fig.(3.4(a)) and it estimates 2D-SDFT for bin(0,1) and (1,0) as shown in Fig. (3.4(b) and 3.4(c)). 2D-SDFT values for bin(0,1) and (1,0) are combined to get the non-directional edges. This combined 2D-SDFT data is shown in Fig.(3.4(d))Edge Map of the image is extracted by comparing the combined values of 2D-SDFT values with a fixed threshold. This extracted edge map is shown in Fig.(3.4(e)).

3.1.2 Simulation Model for 2D - HDFT

To estimate the sliding window 2D-DFT, model for 2D-HDFT (2D Hopping DFT) is also made and shown in Fig(3.5). All the components of the model are marked on the figure. For estimating 2D-HDFT, a updating vector transform (UVT) is calculated for both the bins (0,1) and (1,0). Except UVT all the blocks of this model are same as 2D-SDFT model. The window size for this model is 4×4 and hopping distance (L) is 2. For detecting the edges, Bin (0,1) and (1,0) is only computed as for 2D-SDFT. The output of 2D-HDFT is compared with the output of 2D-SDFT with same window size. There is no difference in two outputs as shown in Fig.(3.6). The Input for both the models is given in Fig.(3.6(a)) and outputs of models with window size 4×4 are given Fig. (3.6(b)) and (3.6(c)).

3.2 Simulation result

By using the Matlab Simulink model as shown in Fig.(3.3), edges of road lane in the different images are extracted. These images are taken from the ROMA database[35]. Extracted edges using the proposed technique are shown in Fig. (3.7).

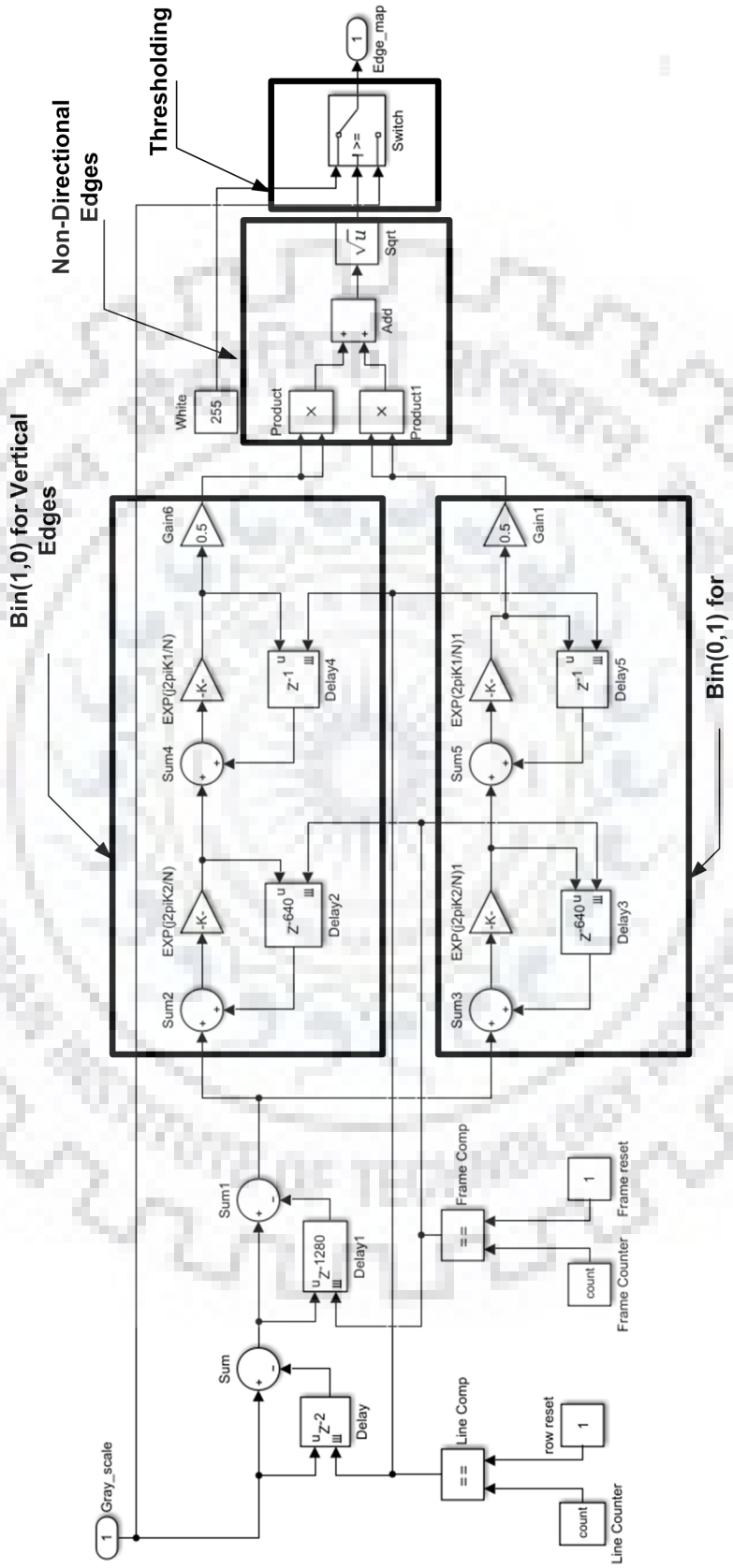
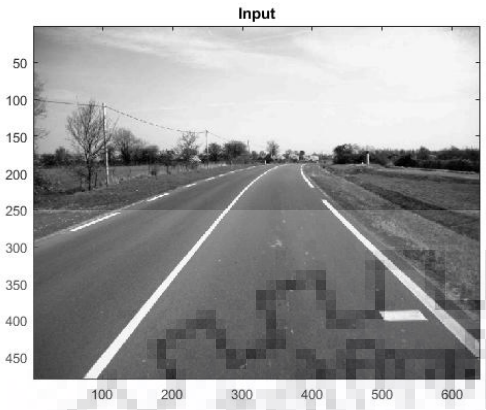
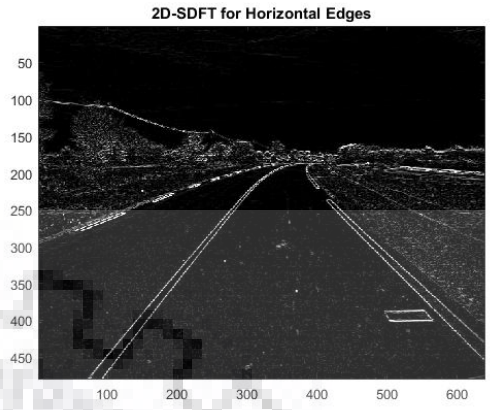


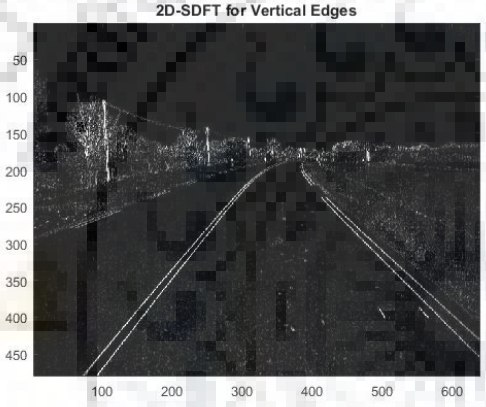
FIGURE 3.3: SimuLink Model of Edge detector based on 2D SDFT for image size 640×480



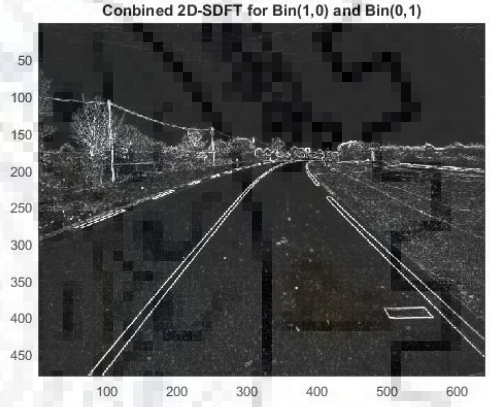
(a) Input 640×480 image



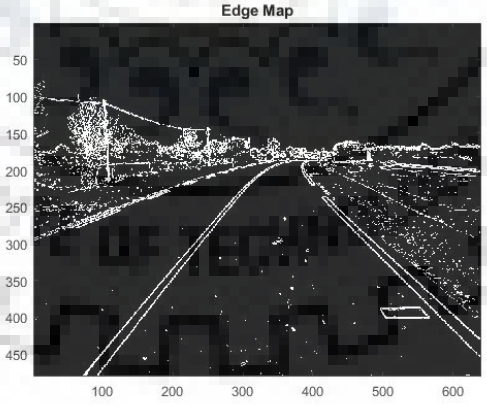
(b) 2D-SDFT for horizontal edges



(c) 2D-SDFT for vertical edges



(d) Combined 2D-SDFT for bin(0,1) and bin(1,0)



(e) Edge map of the image

FIGURE 3.4: Input and Outputs of the 2D-SDFT based Edge Detection Model

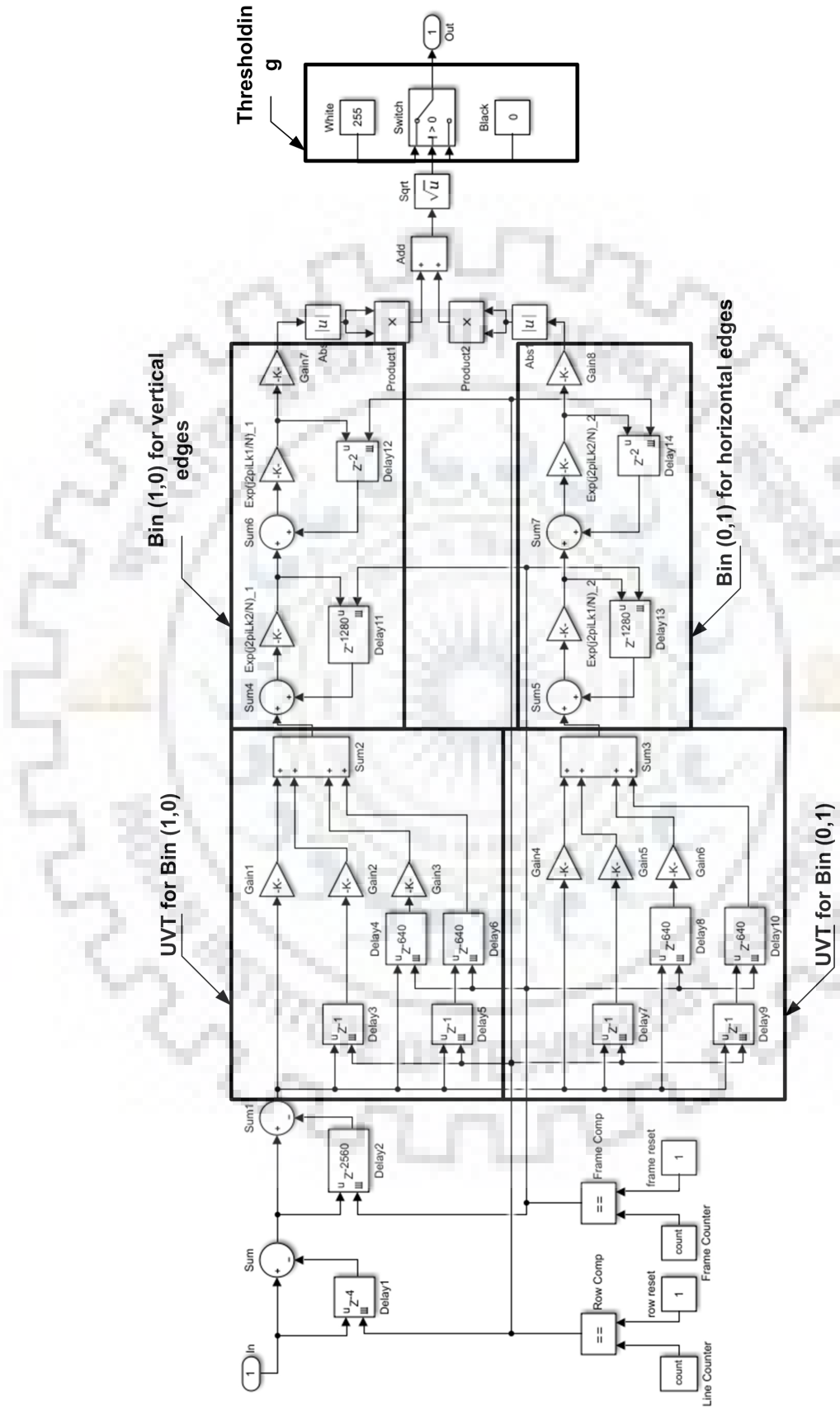
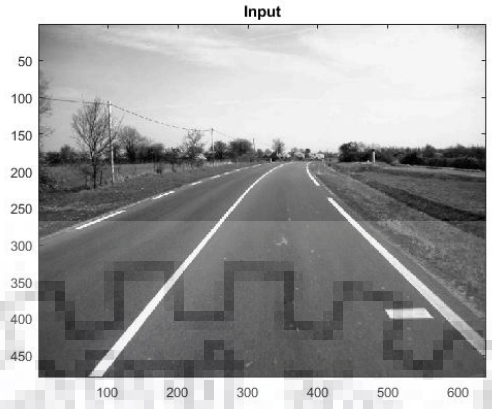
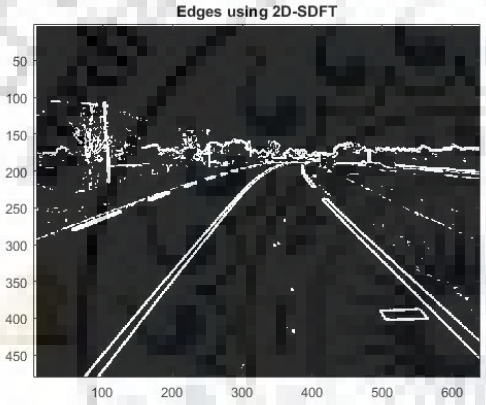


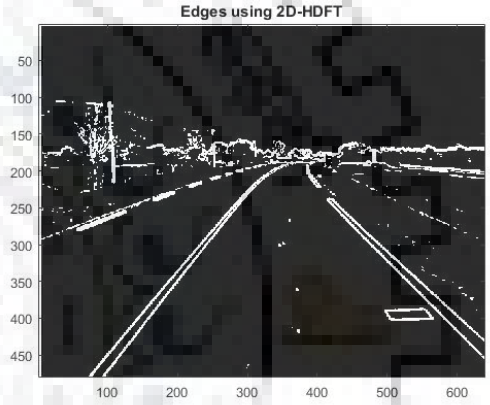
FIGURE 3.5: SimuLink Model of Edge detector based on 2D HDFT for image size 640×480



(a) Input 640 × 480 image



(b) Output of 2D-SDFT with window size 4 × 4



(c) Output of 2D-HDFT with window size 4 × 4

FIGURE 3.6: Input and Outputs of the 2D-SDFT and 2D-HDFT Model

3.3 Performance Evaluation

The performance of the proposed technique is done by visual comparison of output with that of the conventional techniques and by comparing Pratt's figure of merit (PFOM) of output with that of the conventional techniques.

3.3.1 Visual comparison of output with conventional techniques

Visual comparison of proposed technique output with conventional ones (like Canny, Sobel, Prewitt) are shown in Fig.(3.8), (3.9),(3.10), and (3.11).

FIGURE 3.7: Simulation results for some Road images - Input images are taken from ROMA database[35]

FIGURE 3.8: Simulation result and comparison with conventional techniques- Input images are taken from ROMA database [35]

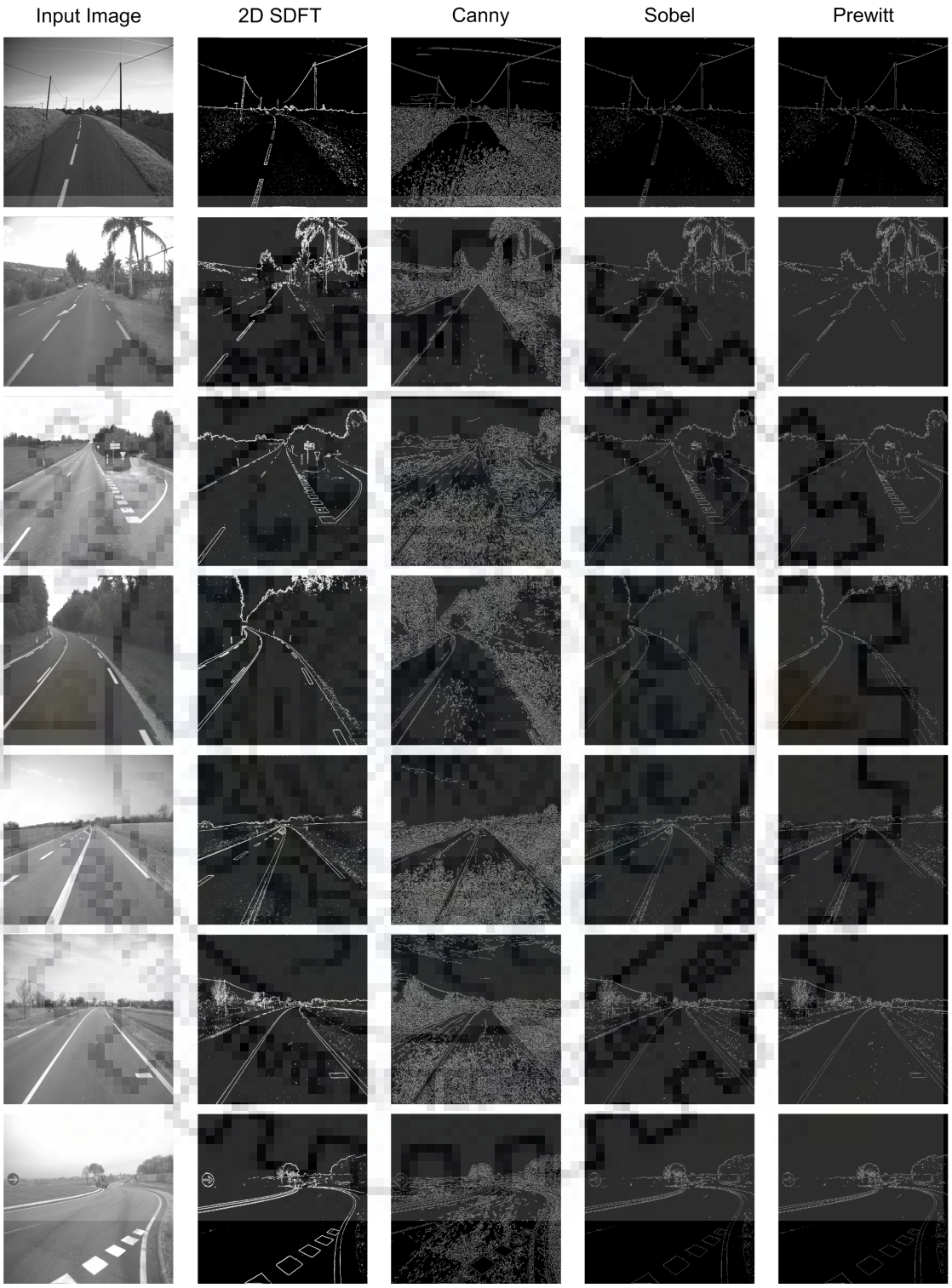


FIGURE 3.9: Simulation result and comparison with conventional techniques- Input images are taken from ROMA database [35]

FIGURE 3.10: Simulation result and comparison with conventional techniques- Input images are taken from road marking dataset [36]

FIGURE 3.11: Simulation result and comparison with conventional techniques- Input images are taken from road marking dataset [36]

3.3.2 PFOM comparison of output with conventional techniques

3.3.2.1 Pratt's figure of merit (PFOM)

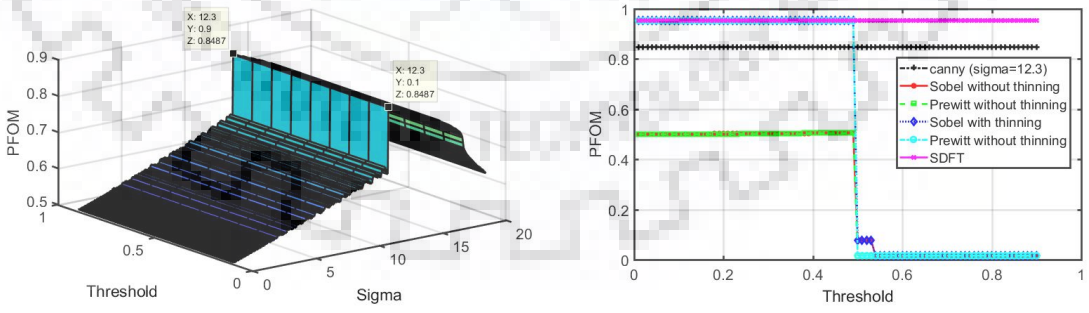
For evaluating the performance of any edge detector, Pratt's figure of merit (PFOM) is generally utilized [34]. This figure of merit was suggested by Pratt for precision evaluation of calculated edges. It speaks to deviation of a real (determined) edge point from the ideal edge and it is written as

$$R1 = \frac{1}{\max(I_I, I_A)} \sum_{i=1}^{I_A} \frac{1}{1 + \delta e^2(i)} \tag{3.1}$$

where I_A is the number of determined edge points, I_I is the number of ideal edge point, $e(i)$ is distance between determined edge point and ideal edge point and δ is constant (usually 1/9).

3.3.2.2 Performance of proposed technique for image with no noise

For Canny technique, PFOM is calculated for different values of threshold and sigma. This is plotted in a 3D graph as shown in Fig.(3.12(a)). From there, values of sigma and threshold are selected for maximum PFOM. For Sobel and Prewitt, PFOM is calculated for different values of the threshold and plotted in Fig.(3.12(b)). The values of threshold



(a) Canny Parameters Selection

(b) PFOM Vs Threshold of different techniques

FIGURE 3.12: Input Parameter Selection for maximum PFOM for conventional method

are selected form this graph and by using values for different techniques, PFOM and

edges are calculated. Edge map by different techniques with input parameter and with PFOM are shown in Fig.(3.13). In the figure, S = sigma and T = threshold.

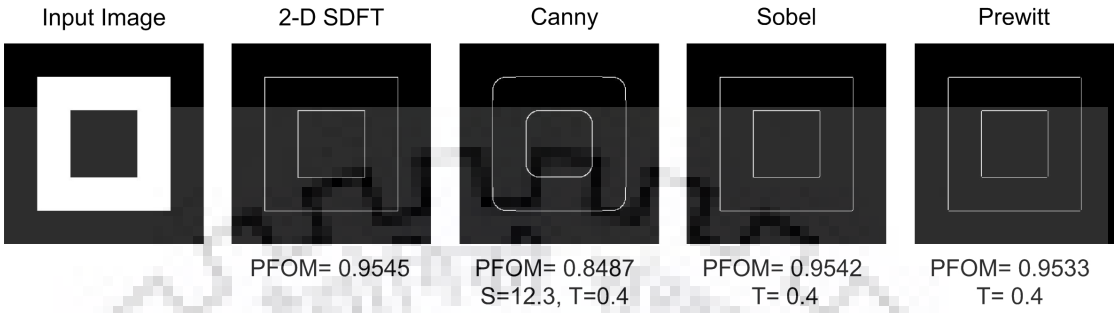
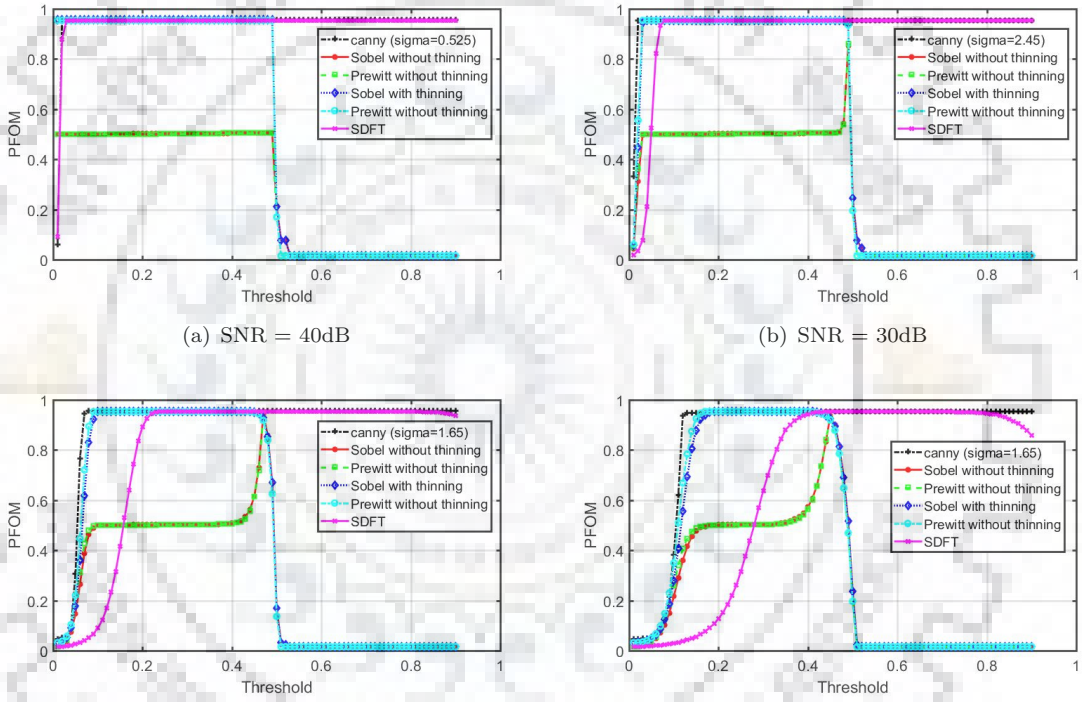


FIGURE 3.13: Performance Evaluation of proposed technique without Noise

3.3.2.3 Performance of proposed technique for image with noise

For this purpose, the Gaussian noise is added to the image. For maximum PFOM, threshold and sigma (only for Canny) parameter of different techniques are calculated as described in section (3.3.2.2). The graphs for threshold with different noise level are shown in FIG.(3.14). By using extracted parameters for different techniques, PFOM and edges are calculated. Edge map by different techniques with input parameter and with PFOM are shown in Fig.(3.15). In the figure, S = sigma and T = threshold.



(a) SNR = 40dB

(b) SNR = 30dB

(c) SNR = 20dB

(d) SNR = 15dB

FIGURE 3.14: PFOM Vs threshold level in different SNR condition

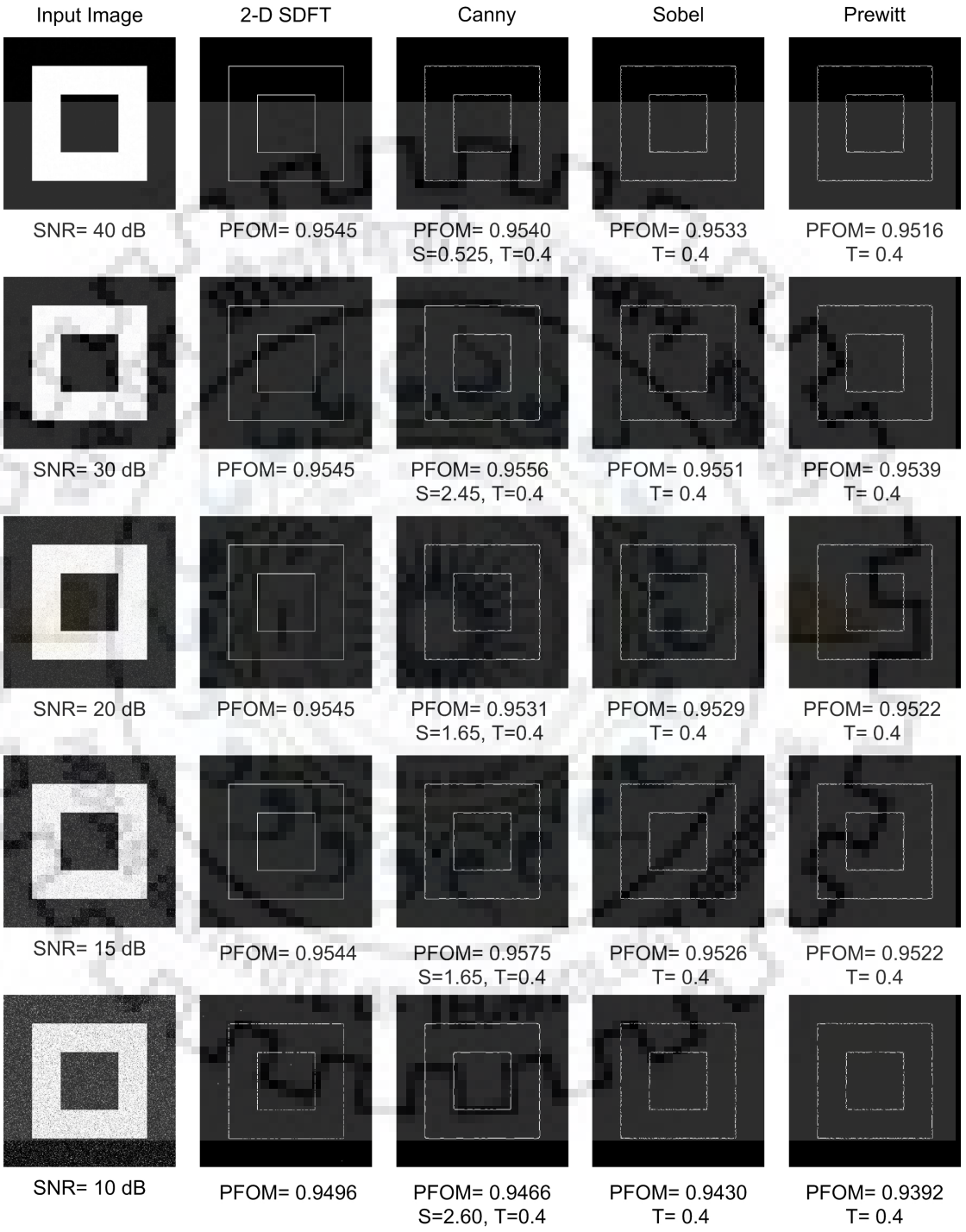


FIGURE 3.15: Performance Evaluation of proposed technique with Noise

Chapter 4

Real-Time Implementation on FPGA Board

The real-time implementation of proposed edge detector on FPGA Board **DE2-115** is done using camera board **TRDB-D5M**. For implementation, a real-time video is captured as an input by the camera and corresponding processed frame is displayed as an output on the monitor through VGA.

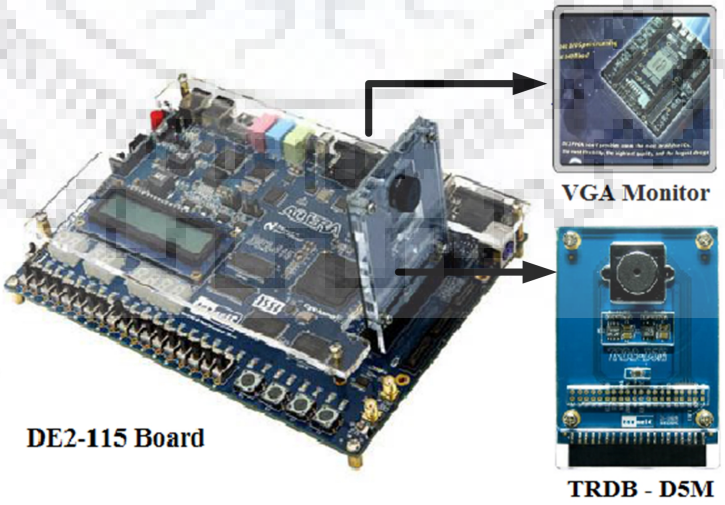


FIGURE 4.1: Connection Diagram of monitor and camera eith DE2-115 FPGA board

The connection diagram is shown in Fig.(4.1). The main components used for implementation are DE2-115 FPGA board, TRDB-D5M camera daughter board, and VGA monitor. Details of FPGA board, camera daughter board and block diagram for implementation are described in next sections.

4.1 FPGA Board DE2-115

FPGA Board DE2-115 is used for implementation for the proposed edge detector. Many features are incorporated in this FPGA board. These features make this board suitable to implement wide range of projects from simple to multimedia. A picture of the FPGA board DE2-115 is represented in Fig.(4.2). The design of FPGA board and area of various connectors and key segments are given in this figure.

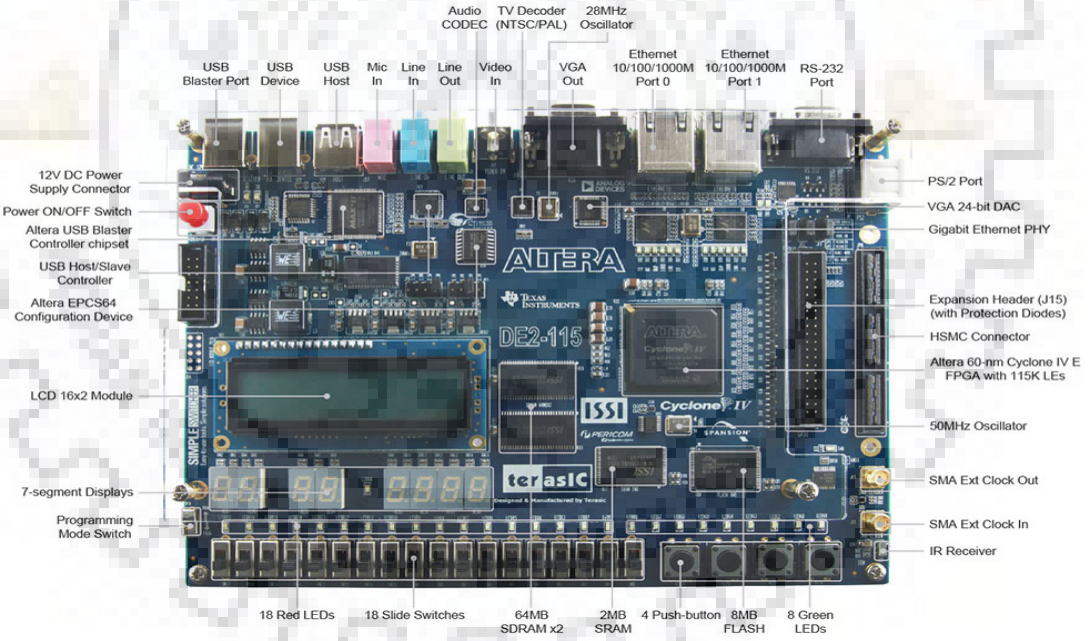


FIGURE 4.2: FPGA Board DE2-115

The following parts are available on the FPGA board DE2-115:

- FPGA - Altera Cyclone IV
- Serial programming device - EPCS64

- USB Blaster for JTAG and Active Serial programming
- SRAM - 2MB
- 2 Nos. SDRAM - 64MB
- Flash memory - 8MB
- Socket for SD Card - 1 No.
- Push-buttons - 4 Nos.
- Slide switches - 18 Nos.
- User LEDs (Red) - 18 Nos.
- User LEDs (Green) - 9 Nos.
- Clock sources - Oscillator, 50MHz
- microphone-in, line-out, and line-in jacks with 24-bit CD-quality audio CODEC
- VGA-out connector with 8 bit triple DACs - 1 No.
- TV in connector and TV Decoder
- RJ45 connectors with 2 Gigabit Ethernet PHY
- USB type A and type B connectors with USB Host/Slave Controller
- RS-232 connector 9-pin with transceiver IC - 1 No.
- PS/2 connector for mouse or keyboard - 1 No.
- IR Receiver - 1 No.
- SMA connectors for using external clock input and output - 2 Nos.
- Diode protected 40-pin Header for expansion - 1 No.
- High Speed Mezzanine Card (HSMC) connector - 1 No.
- LCD module - 16x2

4.2 Camera Daughter Board TRDB-D5M

For implementation, a camera daughter board TRDB-D5M is used which has a CMOS sensor. A photograph of the camera daughter board TRDB-D5M is shown in Fig.(4.3). The CMOS sensor of camera daughter board has $2752(columns) \times 2004(rows)$ pixels. The upper right corner of the camera array holds the location of column 0 and row 0 as shown in Fig.(4.4). The active region of the CMOS sensor array has 2592 columns and 1944 rows. On the periphery of this active region, a boundary of active pixels is situated, which is encased by dark pixels.

FIGURE 4.3: Camera Board TRDB-D5M

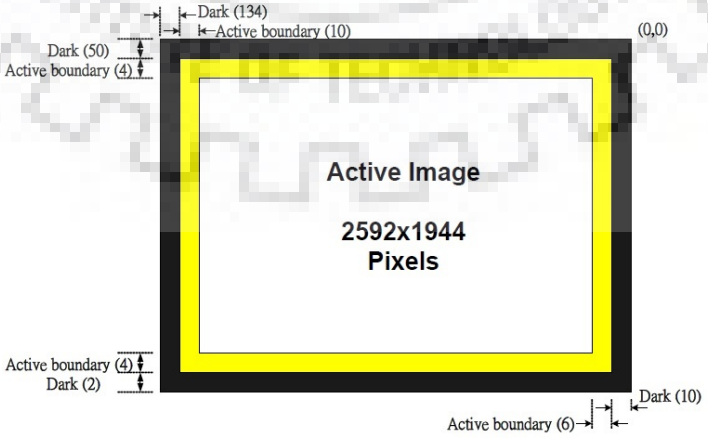


FIGURE 4.4: Pixel array description

The key performance parameter of camera daughter board TRDB-D5M is given in Table(4.1):

TABLE 4.1: Key performance parameter of TRDB-D5M

Parameter	Values
Pixel Size	$2.2\mu m \times 2.2\mu m$
Active Pixel	$2,592H \times 1,944V$
Color Filter Array	RGB Bayer Pattern
Frame rate (VGA $640X480$)	Programmable up to 70fps
Frame rate (Full resolution)	Programmable up to 15fps
Maximum Data rate	96 Mp/s at 96 MHz
ADC Resolution	12-bit
Pixel Dynamic Range	$70.1dB$
Responsivity	1.4 V/lux-sec (550 nm)
SNRMAX	$38.1dB$
Shutter Type	Global Reset Release
Input-Output Voltage	$1.7 - 3.1V$
Power Supply Voltage	$3.3V$

4.3 Implementation of proposed edge detector on FPGA

In implementation, a real time video frame is used as an input and corresponding processed frame is displayed as the output on the VGA monitor. Block Diagram of sequence of operations for real-time implementation of proposed edge detector is shown in fig.(4.5).

FIGURE 4.5: Block Diagram for Sequence of operations

As shown in block diagram, first step is to capture CMOS camera raw data. This captured data is raw data and it is converted to RGB data in second step. This RGB data is saved into SDRAM in third step. Now this RGB data is read from SDRAM and used

for edge detection in fourth step. This edge data is sent to VGA controller to display the detected edge in last step. For that purpose following modules are used and they are connected as shown in Fig (4.6).

1. Camera Capture Module,
2. Raw to RGB data Converter Module,
3. VGA Controller Module,
4. SDRAM Controller Module,
5. PLL Module,
6. I2C sensor configuration Module,
7. Proposed Edge Detector Module.
8. Top Module.

4.3.1 Camera Capture Module

This module is used to read the data of valid pixels of the camera which are indicated by the level of the LVAL (Line Valid) and FVAL (Frame Valid) signals. For the operation of this module, PIXLCLK is the clock frequency. Whether output data are from valid pixel or not is decided by the signal named as oDVAL. The number of frames is also counted by this module by the use of FVAL signal and displayed on FPGA board's seven segment display. The location of each valid pixel in terms of X and Y coordinates of image are The X and Y position of each valid pixel of an image are passed down to next module i.e. RAW2RGB.v. The verilog code for this image capturing module is stored in the file having the name CCD_Capture.v.

4.3.2 Raw to RGB Data Converter Module

The raw (Bayer pattern) data received from camera capture module is converted to RGB data by the use of this module. The RGB value is calculated from the four near

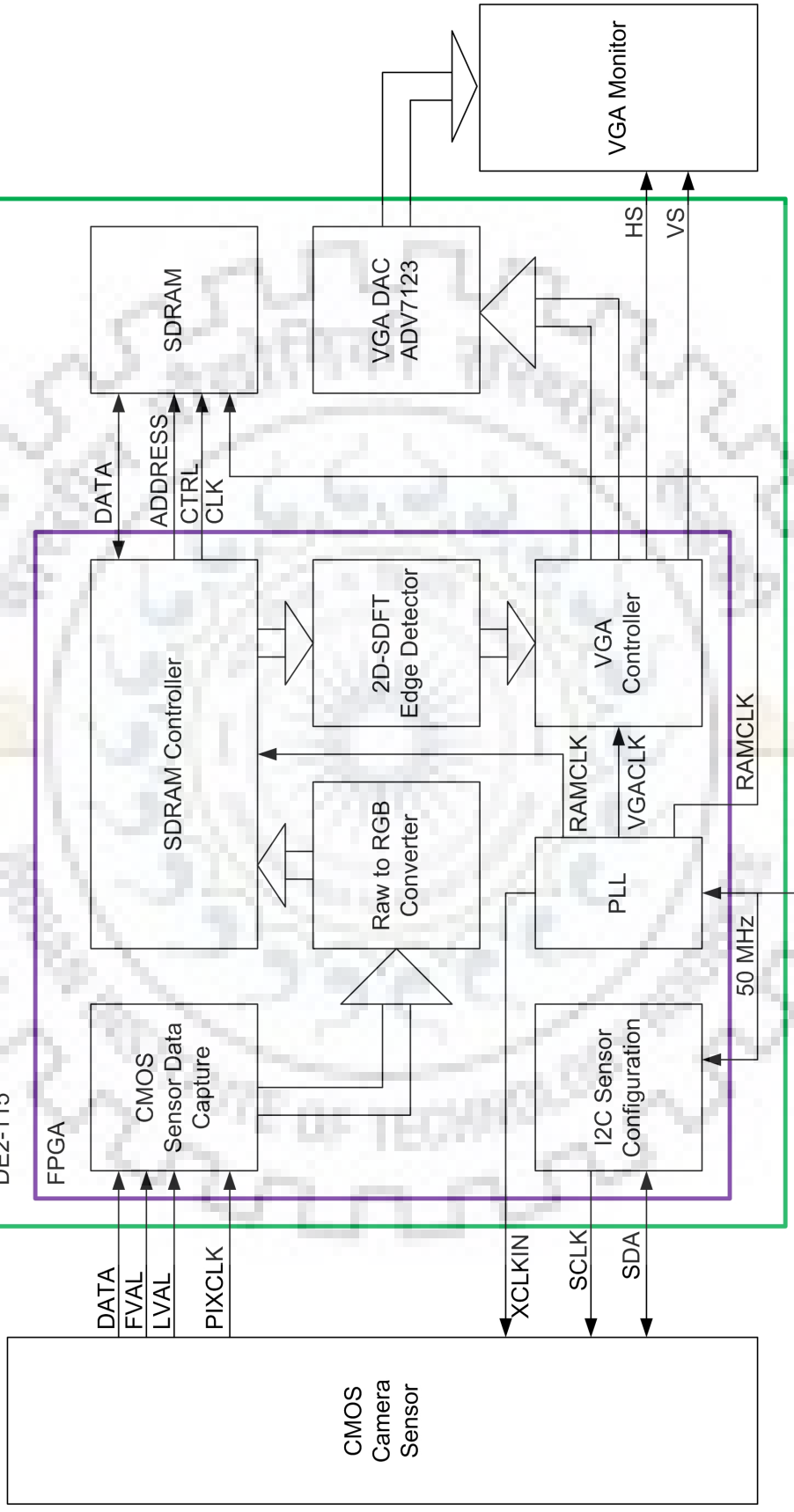


FIGURE 4.6: Block Diagram for implementation on FPGA

by pixels situated in a square (Blue, Green2, Green1, Red).

oRed = Red,

oGreen = (Green1 + Green2)/2, and

oBlue = Blue.

The verilog code for this is stored in the file having the name RAW2RGB.v.

4.3.3 VGA Controller Module

Correspond to input RGB signal, the VGA controller module generates the signals to be sent to VGA monitor and also generates synchronisation signal. The verilog code for this is stored in the file having the name VGA_Controller.v.

4.3.4 SDRAM Controller Module

SDRAM controller module is used to generate proper control signals for communicating with the SDRAM. SDRAM addresses, commands and control signals are generated by the use of the data path and logic control which are defined in this module.

In this project, RAW2RGB module gives the RGB output at frequency equal to PIXCLK and SDRAM and controller for that operates at a frequency of 100 MHz. This mismatch of clock frequencies is conquered by the deployment of FIFOs as buffering. The 4-port SDRAM controller having 4 FIFOs is used in this project. The output of RAW2RGB module has 30bits per pixel (10 bits per color) for RGB data. In this project, only two of the four available banks in the SDRAM are used. Two FIFOs are used for each one of the banks one for write the data and other for read the data. The information of a pixel in RGB format is separated into two parts using the two banks and FIFOs. The complete 10 red bits and half 5 green bits are stored in one bank, and the remaining half 5 green bits and complete 10 blue bits are stored in the another bank. Basically, the controller for SDRAM is made of main three things:

1. 4 banks - By the use of an array of 8192 rows, 512 column and 16bits, each bank is created as an array of rows and columns (8192 rows x 512 columns x 16 bits).

Number of row, column and bank convey complete information about each memory location.

2. A decoder for command - Commands received are decoded this decoder to tell the controller what function is to be done.
3. A mode register - It is used to define the current mode of operation of the SDRAM.

This module is defined in sdram_control.v.

4.3.5 PLL Module

This module is implemented to generate the clocks for CMOS camera sensor, SDRAM, and VGA as follows:

1. 25 MHz for CMOS Camera sensor
2. 100 MHz for SDRAM
3. 25 MHz for VGA

This module is defined in sdram_pll.v.

4.3.6 I2C sensor configuration Module

The CMOS camera sensor uses I2C bus protocol. The I2C bus have a two-wired serial interface as follows:

1. SDA (Serial Data) bidirectional line
2. SCL (Serial Clock)

A different address is allotted to each device connected on the I2C bus. All the devices are connected in parallel. Each connected device can perform as a transmitter or a receiver. In this communication at least two devices are required - one act as a master

and another as a slave. The clock signal on SCL is produced by the master device and the transfer is also initiated/terminated by the master device. The data is transferred using the SDA. Here, CMOS camera as a slave and FPGA chip as a master are connected by this I2C bus.

Beside transmitting the slave address and a directional bit, FPGA chip(master) also sends camera setting to the internal registers i.e. the register address and data having 16 bits to CMOS sensor (slave). So, the writing of camera setting into internal registers is done using following step:

1. Start pulse
2. Address of Slave device + bit describing the information about direction
3. Address of Register to be modified
4. First data byte to be sent
5. Next data byte to br sent
6. Stop pulse

A finite state machine (FSM) is used to implement this module and to generate proper SCL & SDA signals. The verilog code for this module is stored in the file having the name I2C_CCD_Config.v.

4.3.7 Proposed Edge Detector Module

The main function of this module is to get the data from SDRAM and estimate the 2D-sdft for extracting the Edge map of the image. This edge map data is sent to VGA controller to display the edge map on VGA monitor. To implement this module, model of Proposed Edge detector is made in SimuLink. In these model, all the complex twiddle factor are converted in COS and SIN terms to calculate real and imaginary parts of DFT. All the terms equal to zero is removed in the model to reduce complexity. These models are converted to verilog code using HDL coder. For estimation of sliding window

DFT, SDFT and HDFT is used and model for both are given in Fig.(4.7) and Fig.(4.8) respectively. These modules perform the following tasks:

1. Receive RGB data as input.
2. Calculate the Gray data from RGB data.
3. Calculate the sliding window SDFT/HDFT using Gray data.
4. Compare the SDFT/HDFT value with a threshold to extract the edge map.
5. Combining of edge map and RGB data
6. Send the combined data as output

This module is defined in a verilog file Subsystem.v.

4.3.8 Top Module

All the above modules are instantiated in the top module. The main function of this top module is provide proper communication between all modules. This module is defined in DE2_115_CAMERA.v.

All the verilog files for different modules are attached in appendix 'A'.

4.4 Experimental Setup

The photograph of experimental set-up is shown in the fig.(4.9). In this setup, VGA monitor and CMOS Camera TRDB-D5M are connected to FPGA board. Verilog files of the real time system are compiled and converted into a Bit file using the Quartus Prime. This bit file is downloaded to FPGA using the same software from PC to FPGA board.

The proposed edge detection algorithm using 2×2 window size is implemented in FPGA. The Altera DE2-115 FPGA board of clock frequency 50 MHz with Cyclone

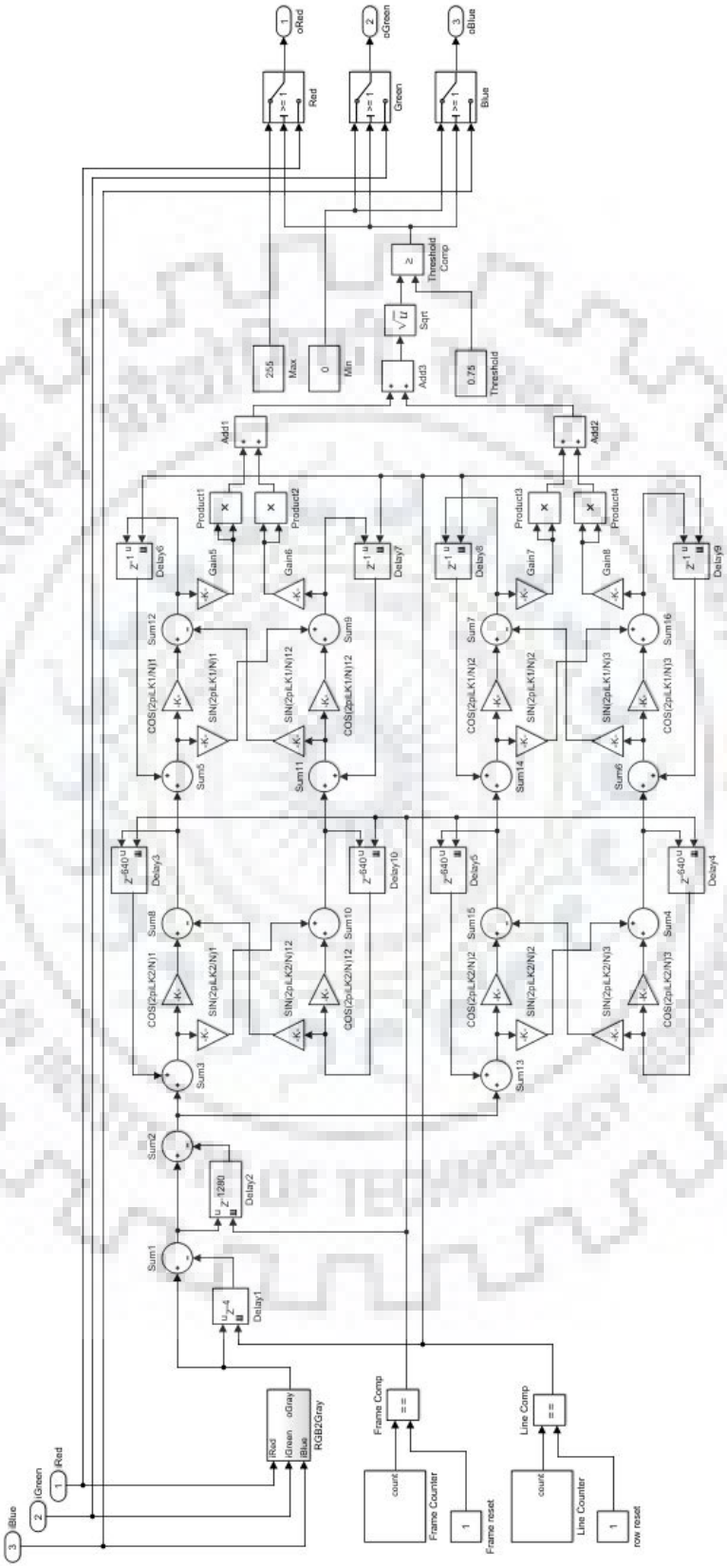


FIGURE 4.7: Model for estimation of edge map based 2D-SDFT

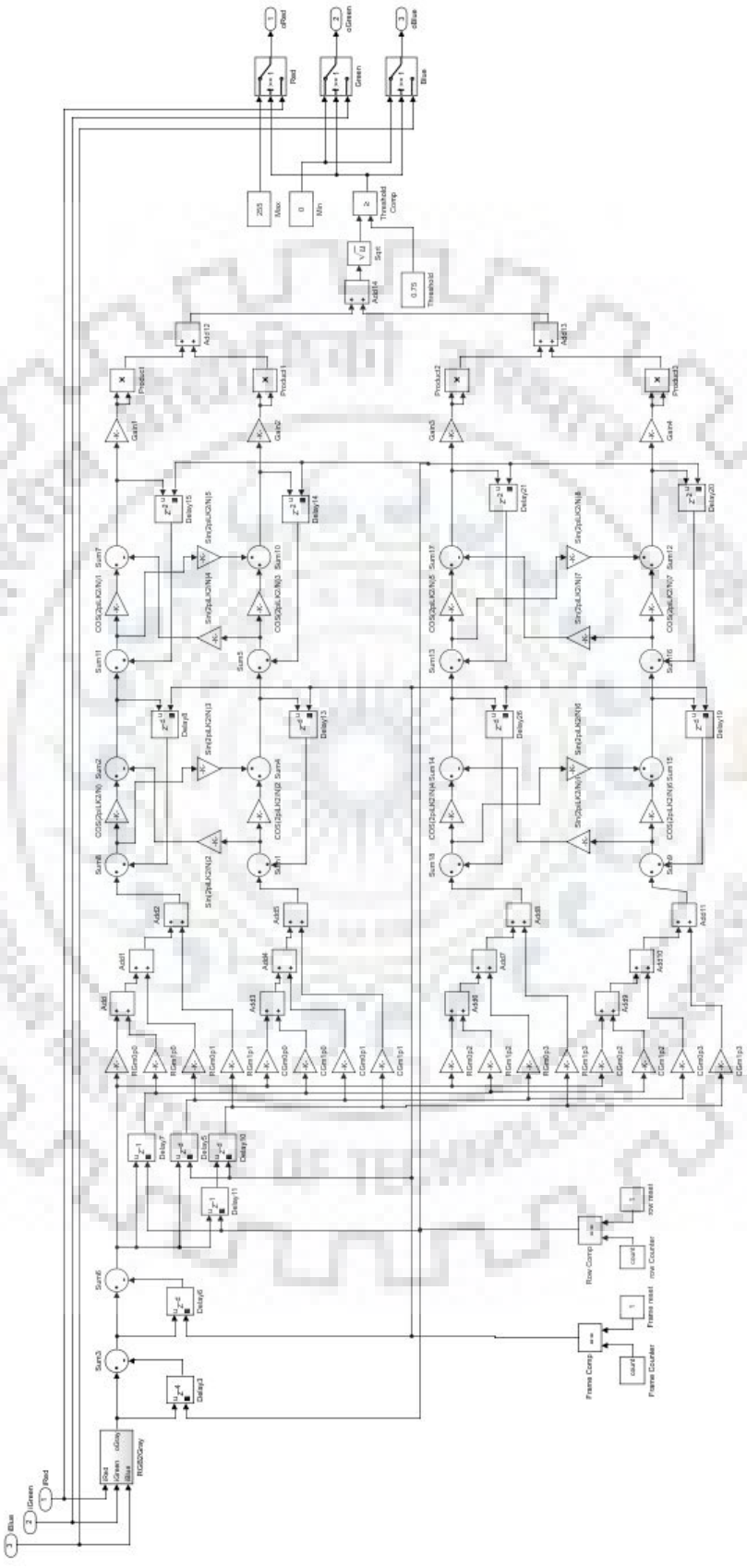


FIGURE 4.8: Model for estimation of edge map based 2D-HDFT

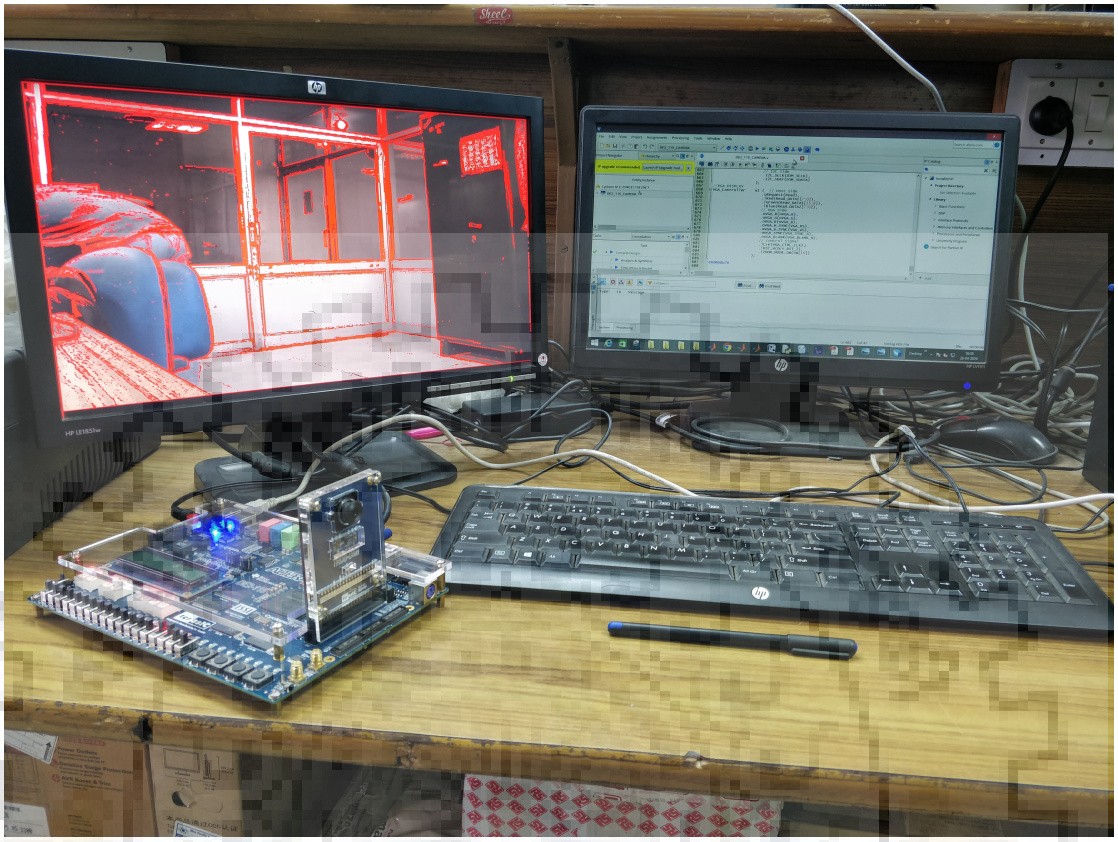


FIGURE 4.9: Experimental Setup for implementation on FPGA

IV 4CE115F29 FPGA device is used for implementing the algorithm in real-time. The proposed edge detector blocks are developed in simulink environment and through HDL-coder 3.1, the equivalent verilog codes are generated and downloaded into FPGA. The PC and FPGA board communication is established by USB-blaster. The camera TRDB-D5M is connected to FPGA board using GPIO connector. Pixel information is read from the camera and stored into SDRAM. That information is read by FPGA and after edge detection, the edge detected image is brought out to the real-world through DAC-VGA connector and finally the edge detected image is displayed in the TFT monitor. The proposed edge detector algorithm is made available as system-on-chip device by programming the FPGA through active serial programming, so that the algorithm is available permanently on the chip, the output of the edge map could be seen from Fig.(4.10).

The utilization summary of the proposed method on Cyclone IV FPGA is as follows: Total logic elements 39,697 / 114,480 (35%), total combination functions 39,435 / 114,480

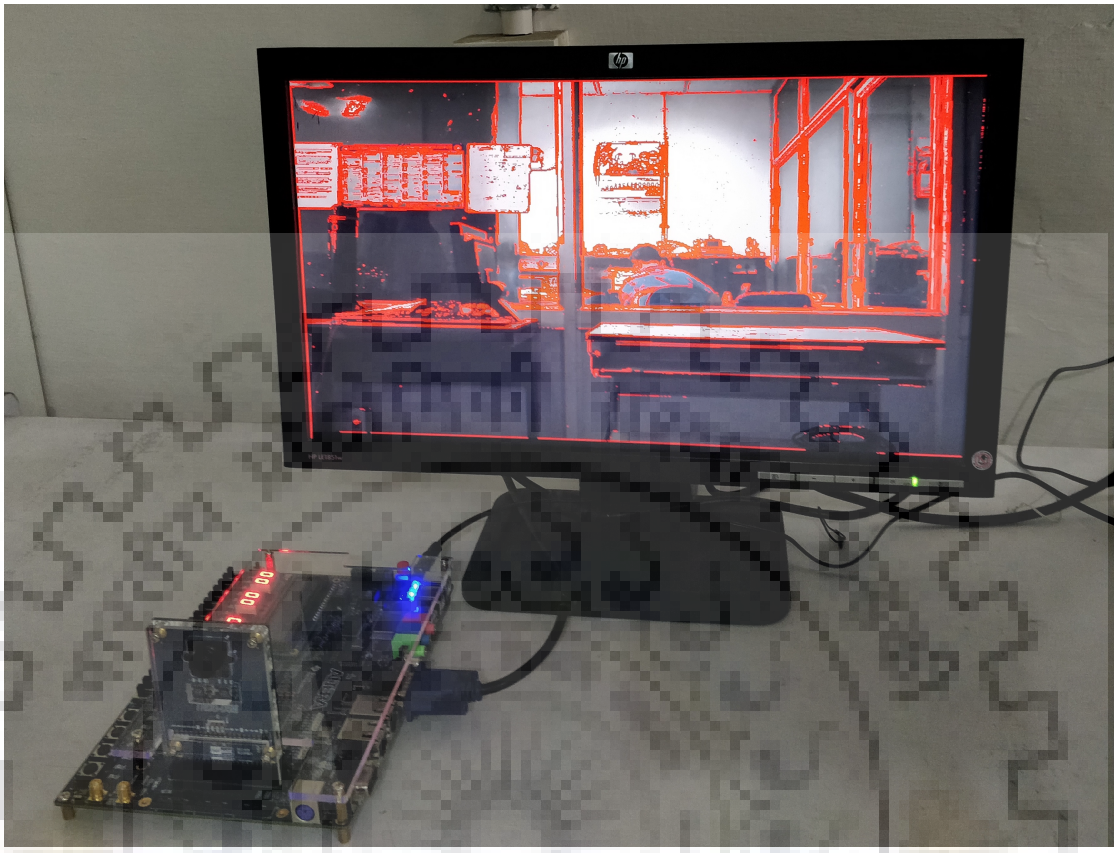


FIGURE 4.10: Detected Edge Image on VGA monitor

(34%), dedicated logic registers 38,282 / 114,480 (33%), total memory bits 59,344 / 3,981,312 (1%), embedded multiplier 9-bit elements 3 / 532 (< 1%).

Using 4×4 window, 2D-SDFT and 2D-HDFT based edge detector is also implemented on DE2i-150 FPGA board. the comparison of utilization summary of these methods on Cyclone IV FPGA is given in following Table (4.2):

TABLE 4.2: Utilization Summary of 2D SDFT & 2D HDFT for 4×4 window size on Cyclone IV FPGA

Parameter	2D SDFT	2D HDFT (L=2)
Total Logic Elements	48,804/149,760(33%)	100,288/149,760(67%)
Total Memory Bits	51,152/6,635,520(< 1%)	51,152/6,635,520(< 1%)
Embedded Multiplier 9-Bit Elements	8/720(1%)	8/720(1%)

L = Hope Distance.

Chapter 5

Conclusion and future work

The 2-D sliding DFT based edge detector is proposed for lane and road marking detection. The suitability of this edge detector is explored for lane, road markings and object detection in road images. The proposed edge detector performs better in comparison with Canny, Sobel, and Prewitt edge detectors in the presence of noise and at various noise levels. The edge detector is made available as system-on-chip device through FPGA implementation. It could be concluded that the proposed method detects almost all the edges of image and video frames successfully and hence it is very much suitable for lane detection systems.

Simulation result of the proposed technique is visually as good as existing techniques. As per performance evaluation, PFOM of the proposed technique with no noise is 0.9545, which is comparable with the PFOM of canny, sobel and prewitt technique as shown in fig. (3.13). With 40dB dB, 30dB, 20dB, 15dB and 10 dB signal to noise ratio (SNR), PFOM of proposed technique is 0.95 which is comparable with canny, sobel and prewitt techniques as shown in fig. (3.15). To get this PFOM, Gaussian filter is used to reduce the noise effect in canny and edge thinning is used in sobel and prewitt. While this type of processing is not used in proposed method.

Proposed method is also implemented with using 2D HDFT. Output of both the methods 2D SDFT and 2D HDFT is absolutely same. But utilization of total logic elements is more than 2D SDFT based method as shown in Table (4.2).

At present, proposed technique is implemented by using fixed threshold. In future, this technique can be implemented using auto threshold and can be used in lane detection, tracking system and driver assistant systems. At present, suitability of this detector is explored only for road images. In future, suitability can be explored for other type of images.

Publications

1. Amit Kr. Vishwakarma and P Sumathi, "2D SDFT based Edge Detection for Lane and Road Marking Detection ", in *IEEE Transactions on Image Processing* (submitted).

Bibliography

[1] V. Torre, T.A. Poggio, "On Edge Detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-8, no. 2, pp. 147-163, Mar. 1986.

[2] T. Aydin, Y. Yemez, E. Anarim, B. Sankur, "Multidirectional and multiscale edge detection via M-band wavelet transform," *IEEE Trans. Image Process.*, vol. 5, no. 9, pp. 1370-1377, Sep. 1996.

[3] L.G. Roberts, "Machine perception of three-dimensional solids," in *Optical and Electro-optical Information Processing*, J.T. Tippett, et al., Ed. MIT Press, Cambridge, MA, 1965.

[4] I. Sobel, G. Feldman, "A 3x3 Isotropic Gradient Operator for Image Processing," presented at a talk at the Stanford Artificial Project in 1968, unpublished but often cited, orig. in Pattern Classification and Scene Analysis, Duda, R. and Hart, P., John Wiley and Sons, pp. 271-272, 1973.

[5] I.Sobel, "Neighborhood coding of binary images for fast contour following and general array binary processing," *Comput. Graph. Image Process.*, vol. 8, no. 1, pp. 127-135, Aug. 1978.

[6] J.M.S. Prewitt, "Object enhancement and extraction," in *Picture Processing and Psychopictorics*, B. Lipkin and A. Rosenfeld, Ed. New York, New York: Academic, 1970, pp. 75-149.

[7] D. Marr, E. C. Hildreth, "Theory of edge detection," *Proc. R. Soc.*, vol. B207, issue. 1167, pp. 187-217, Feb. 1980.

[8] J. Canny, "A computational approach to edge detection," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-8, no. 6, pp. 679-698, Nov. 1986.

[9] B. G. Schunck, "Edge detection with Gaussian filters at multiple scales," *Proc. IEEE Comp. Soc. Work. Comp. Vis.*, pp. 208-210, 1987.

[10] F. Bergholm, "Edge Focusing," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-9, no. 6, pp. 726-741, Nov. 1987.

[11] V. Lacroix, "The primary raster: a multiresolution image description," *Proc. 10th Int. Conf. Pattern Recognit.*, vol. 1, pp. 903-907, Jun. 1990.

[12] D. Heric and D. Zazula, "Combined edge detection using wavelet transform and signal registration," *Image and Vision Computing*, vol. 25, pp. 652-662, May 2007.

[13] S.M. Bhandarkar, Y. Zhang and W.D. Potter, "An edge detection technique using genetic algorithm-based optimization," *Pattern Recognition*, vol. 27, no. 9, pp. 1159-1180, Sep. 1994.

[14] M. Gudmundsson, E. A. El-Kwae and M. R. Kabuka, "Edge detection in medical images using a genetic algorithm," *IEEE Trans. Med. Imag.*, vol. 17, no. 3, pp. 469-474, Jun. 1998.

[15] S. E. El-Khamy, M. Lotfy, and N. El-Yamany, "A modified fuzzy Sobeledge detector," in *Proc. IEEE 17th Nat. Radio Sci. Conf.*, Feb. 2000, pp.1-9.

[16] W. Gao, X. Zhang, L. Yang, and H. Liu,"An improved Sobel edge detection," in *Proc. 3rd Int. Conf. Comput. Sci. Inf. Technol. (ICCSIT)*, Jul. 2010, pp. 67-71.

[17] K. Zhang, Y. Zhang, P. Wang, Y. Tian, and J. Yang, "An improved Sobel edge algorithm and FPGA implementation," in *Proc. 8th Int. Congr. Inf.Commun. Technol.*, 2018, pp. 243-248.

[18] W. Rong, Z. Li, W. Zhang, and L. Sun, "An improved CANNY edge detection algorithm," in *Proc. IEEE Int. Conf. Mechatron. Automat.*, 2014, vol. 2, no. 2, pp. 577-582.

[19] X. J. Wu, Z. Yin and Y. Xiong, "The Fast Multilevel Fuzzy Edge Detection of Blurry Images," *IEEE Signal Process. Lett.*, vol. 14, no. 5, pp. 344-347, May 2007.

[20] Mamta Mitta, Amit Verma, Iqbaldeep Kaur,Bhavneet Kaur, Meenakshi Sharma, Lalit Mohan Goyal, Sudipta Roy, and Tai-hoon Kim, "An Efficient Edge Detection Approach to Provide Better Edge Connectivity for Image Analysis," *IEEE Access*, vol. 7, pp. 33240-33255, Mar. 2019.

[21] R. K. Satzoda, S. Sathyanarayana, T. Srikanthan, and S. Sathyanarayana, "Hierarchical Additive Hough Transform for Lane Detection," *IEEE Embedded Systems Letters*, vol. 2, no. 2, pp. 23-26, June 2010.

[22] S. Jung, J. Youn, and S. Sull, "Efficient Lane Detection Based on Spatiotemporal Images," *IEEE Trans. Intelligent Transportation systems*, vol. 17, no. 1, pp. 289-295, Jan. 2016

[23] R. Lyons, *"Understanding Digital Signal Processing,"* New Jersey: Prentice Hall PTR, 2001.

[24] E. Jacobsen and R. Lyons, "An update to the sliding DFT," *IEEE Signal Process. Mag.*, vol. 21, no. 1, pp. 110-111, Jan. 2004.

[25] P. Sumathi, P. A. Janakiraman, "Integrated Phase-Locking Scheme for SDFT-Based Harmonic Analysis of Periodic Signals," *IEEE Transactions on Circuits and Systems-II: Express Briefs*, vol. 55, no.1, pp. 51 - 55, Jan. 2008.

[26] Rafael C. Gonzalez and Richard E. Woods, *"Digital Image Processing,"* Pearson Education, 2011.

[27] C. Park and S. Ko, "The hopping discrete Fourier transform," *IEEE Signal Process. Mag.*, vol. 31, no. 2, pp. 135-139, Mar. 2014.

[28] C.S. Park, "2D Discrete Fourier Transform on Sliding Windows," *IEEE Trans. Image Process.*, vol. 24, no. 3, pp. 901-907, Mar. 2015.

[29] Li, H. ,Jiang, T. and Zhou, Y. ,"A novel subblock linear combination scheme for peak-to-average power ratio reduction in OFDM systems," *IEEE Trans. Broadcast.*, vol. 58, no. 3, pp. 360-369, Sep. (2012).

[30] K. R. Rao, D. N. Kim, J. J. Hwang, "Two-dimensional discrete Fourier transform," in *Fast Fourier TransformAlgorithms and Applications*. Amsterdam, The Netherlands:Springer-Verlag, pp. 127-184, 2010.

[31] A. V. Oppenheim, R. W. Schafer, J. R. Buck, *Discrete-Time Signal Processing*. Englewood Cliffs, NJ, USA:Prentice Hall, pp. 635, 1999.

[32] K. R. Rao, D. N. Kim, J. J. Hwang, "Vector-radix 2D-FFT algorithms," in *Fast Fourier TransformAlgorithms and Applications*. Amsterdam, The Netherlands:Springer-Verlag, pp. 185-193, 2010.

[33] B. G. Sherlock, D. M. Monro, "Moving discrete Fourier transform," *IEE Proc. F Radar Signal Process.*, vol. 139, no. 4, pp. 279-282, Aug. 1992.

[34] W. K. Pratt, *Digital Image Processing*, New York: John Wiley & Sons ,1991.

[35] T. Veit, J. Tarel, P. Nicolle, P. Charbonnier, "Evaluation of Road Marking Feature Extraction," *Proc. IEEE Conf. Intelligent Transportation Systems*, 2008.

[36] T. Wu, A. Ranganathan, "A Practical System for Road Marking Detection and Recognition," *Proc. IEEE Intell. Veh. Symp.*, pp- 25-30, 2012.

Appendix A

Verilog Files for Implementation

A.1 Top Module

```
module DE2_115_CAMERA(
//////////// CLOCK //////////
input                                   CLOCK_50,
input                                   CLOCK2_50,
input                                   CLOCK3_50,

//////////// LED //////////
output                  [8:0]           LEDG,
output                 [17:0]           LEDR,

//////////// KEY //////////
input                   [3:0]           KEY,

//////////// SW //////////
input                  [17:0]           SW,

//////////// SEG7 //////////
output                  [6:0]           HEX0,
output                  [6:0]           HEX1,
output                  [6:0]           HEX2,
output                  [6:0]           HEX3,
output                  [6:0]           HEX4,
output                  [6:0]           HEX5,
output                  [6:0]           HEX6,
output                  [6:0]           HEX7,

//////////// LCD //////////
output                                  LCD_BLON,
inout                   [7:0]           LCD_DATA,
output                                  LCD_EN,
output                                  LCD_ON,
output                                  LCD_RS,
output                                  LCD_RW,

//////////// VGA //////////
output                  [7:0]           VGA_B,
output                                  VGA_BLANK_N,
output                                  VGA_CLK,
output                  [7:0]           VGA_G,
output                                  VGA_HS,
output                  [7:0]           VGA_R,
output                                  VGA_SYNC_N,
output                                  VGA_VS,

//////////// SDRAM //////////
```

```
output              [12:0]          DRAM_ADDR,
output               [1:0]          DRAM_BA,
output                              DRAM_CAS_N,
output                              DRAM_CKE,
output                              DRAM_CLK,
output                              DRAM_CS_N,
inout               [31:0]          DRAM_DQ,
output               [3:0]          DRAM_DQM,
output                              DRAM_RAS_N,
output                              DRAM_WE_N,

////////////// GPIO, GPIO connect to D5M - 5M Pixel Camera //////////
input               [11:0]          D5M_D,
input                               D5M_FVAL,
input                               D5M_LVAL,
input                               D5M_PIXLCLK,
output                              D5M_RESET_N,
output                              D5M_SCLK,
inout                               D5M_SDATA,
input                               D5M_STROBE,
output                              D5M_TRIGGER,
output                              D5M_XCLKIN
);
//=============================================================
//  REG/WIRE declarations
//=============================================================
wire    [15:0]  Read_DATA1, Read_DATA2;
wire    [11:0]  mCCD_DATA;
wire            mCCD_DVAL, mCCD_DVAL_d;
wire    [15:0]  X_Cont, Y_Cont;
wire    [9:0]   X_ADDR;
wire    [31:0]  Frame_Cont;
wire            DLY_RST_0, DLY_RST_1, DLY_RST_2;
wire            DLY_RST_3, DLY_RST_4;
wire            Read;
reg     [11:0]  rCCD_DATA;
reg             rCCD_LVAL, rCCD_FVAL;
wire    [11:0]  sCCD_R, sCCD_G, sCCD_B;
wire    [11:0]  gray;
wire    [7:0]   sdft;
wire            sCCD_DVAL;
wire            sdft_DVAL;
wire            sdram_ctrl_clk;
wire    [9:0]   oVGA_R;                 //      VGA Red[9:0]
wire    [9:0]   oVGA_G;                 //      VGA Green[9:0]
wire    [9:0]   oVGA_B;                 //      VGA Blue[9:0]
//power on start
wire            auto_start;
//=============================================================
//  Structural coding
//=============================================================
// D5M
assign  D5M_TRIGGER     =       1'b1;  // tRIGGER
assign  D5M_RESET_N     =       DLY_RST_1;
assign  VGA_CTRL_CLK = ~VGA_CLK;

assign  LEDR            =       SW;
assign  LEDG            =       Y_Cont;
assign  UART_TXD = UART_RXD;

//fetch the high 8 bits
assign  VGA_R = oVGA_R[7:0];
assign  VGA_G = oVGA_G[7:0];
assign  VGA_B = oVGA_B[7:0];
assign  gray = (3*sCCD_R + 6*sCCD_G + 1*sCCD_B)/10;

//D5M read
always@(posedge D5M_PIXLCLK)
begin
        rCCD_DATA       <=      D5M_D;
        rCCD_LVAL       <=      D5M_LVAL;
        rCCD_FVAL       <=      D5M_FVAL;
end
//auto start when power on
assign auto_start = ((KEY[0])&&(DLY_RST_3)&&(!DLY_RST_4))?
```

```
1'b1:1'b0;
//Reset module
Reset_Delay      u2      (.iCLK(CLOCK2_50),.iRST(KEY[0]),
                          .oRST_0(DLY_RST_0),.oRST_1(DLY_RST_1),
                          .oRST_2(DLY_RST_2),.oRST_3(DLY_RST_3),
                          .oRST_4(DLY_RST_4)
                         );
//D5M image capture
CCD_Capture      u3      (.oDATA(mCCD_DATA),
                          .oDVAL(mCCD_DVAL),
                          .oX_Cont(X_Cont),
                          .oY_Cont(Y_Cont),
                          .oFrame_Cont(Frame_Cont),
                          .iDATA(rCCD_DATA),
                          .iFVAL(rCCD_FVAL),
                          .iLVAL(rCCD_LVAL),
                          .iSTART(!KEY[3]|auto_start),
                          .iEND(!KEY[2]),
                          .iCLK(~D5M_PIXLCLK),
                          .iRST(DLY_RST_2)
                         );
//D5M raw date convert to RGB data
RAW2RGB          u4      (.iCLK(D5M_PIXLCLK),
                          .iRST(DLY_RST_1),
                          .iDATA(mCCD_DATA),
                          .iDVAL(mCCD_DVAL),
                          .oRed(sCCD_R),
                          .oGreen(sCCD_G),
                          .oBlue(sCCD_B),
                          .oSDFT(sdft),
                          .oDVAL(sCCD_DVAL),
                          .iX_Cont(X_Cont),
                          .iY_Cont(Y_Cont)
                         );
//Frame count display
SEG7_LUT_8       u5      (.oSEG0(HEX0),.oSEG1(HEX1),
                          .oSEG2(HEX2),.oSEG3(HEX3),
                          .oSEG4(HEX4),.oSEG5(HEX5),
                          .oSEG6(HEX6),.oSEG7(HEX7),
                          .iDIG(Frame_Cont[31:0])
                         );

sdram_pll        u6      (.inclk0(CLOCK2_50),
                          .c0(sdram_ctrl_clk),.c1(DRAM_CLK),
                          .c2(D5M_XCLKIN),.c3(VGA_CLK)      //25M
                         );

//SDRam Read and Write as Frame Buffer
Sdram_Control    u7      (//      HOST Side
                          .RESET_N(KEY[0]),
                          .CLK(sdram_ctrl_clk),
                          //      FIFO Write Side 1
                          .WR1_DATA({sCCD_G[7:0],sCCD_R[7:0]}),
                          .WR1(sCCD_DVAL),
                          .WR1_ADDR(0),
                          .WR1_MAX_ADDR(640*480/2),
                          .WR1_LENGTH(8'h50),
                          .WR1_LOAD(!DLY_RST_0),
                          .WR1_CLK(~D5M_PIXLCLK),
                          //      FIFO Write Side 2
                          .WR2_DATA({8'b0,sCCD_B[7:0]}),
                          .WR2(sCCD_DVAL),
                          .WR2_ADDR(23'h100000),
                          .WR2_MAX_ADDR(23'h100000+640*480/2),
                          .WR2_LENGTH(8'h50),
                          .WR2_LOAD(!DLY_RST_0),
                          .WR2_CLK(~D5M_PIXLCLK),
                          //      FIFO Read Side 1
                          .RD1_DATA(Read_DATA1),
                          .RD1(Read),
                          .RD1_ADDR(0),
                          .RD1_MAX_ADDR(640*480/2),
                          .RD1_LENGTH(8'h50),
```

```
                        .RD1_LOAD(!DLY_RST_0),
                        .RD1_CLK(~VGA_CTRL_CLK),
                       //      FIFO Read Side 2
                        .RD2_DATA(Read_DATA2),
                        .RD2(Read),
                        .RD2_ADDR(23'h100000),
                        .RD2_MAX_ADDR(23'h100000+640*480/2),
                        .RD2_LENGTH(8'h50),
                        .RD2_LOAD(!DLY_RST_0),
                        .RD2_CLK(~VGA_CTRL_CLK),
                       //      SDRAM Side
                        .SA(DRAM_ADDR),
                        .BA(DRAM_BA),
                        .CS_N(DRAM_CS_N),
                        .CKE(DRAM_CKE),
                        .RAS_N(DRAM_RAS_N),
                        .CAS_N(DRAM_CAS_N),
                        .WE_N(DRAM_WE_N),
                        .DQ(DRAM_DQ),
                        .DQM(DRAM_DQM)
                       );
//D5M I2C control
I2C_CCD_Config  u8     (//      Host Side
                        .iCLK(CLOCK2_50),
                        .iRST_N(DLY_RST_2),
                        .iEXPOSURE_ADJ(KEY[1]),
                        .iEXPOSURE_DEC_p(SW[0]),
                        .iZOOM_MODE_SW(SW[16]),
                       //      I2C Side
                        .I2C_SCLK(D5M_SCLK),
                        .I2C_SDAT(D5M_SDATA)
                       );

//VGA DISPLAY
VGA_Controller  u1     (//      Host Side
                        .oRequest(Read),
                        .iRed(Read_DATA1[7:0]),
                        .iGreen(Read_DATA1[15:8]),
                        .iBlue(Read_DATA2[7:0]),
                       //      VGA Side
                        .oVGA_R(oVGA_R),
                        .oVGA_G(oVGA_G),
                        .oVGA_B(oVGA_B),
                        .oVGA_H_SYNC(VGA_HS),
                        .oVGA_V_SYNC(VGA_VS),
                        .oVGA_SYNC(VGA_SYNC_N),
                        .oVGA_BLANK(VGA_BLANK_N),
                       //      Control Signal
                        .iCLK(VGA_CTRL_CLK),
                        .iRST_N(DLY_RST_2)
                       );
endmodule
```

A.2 Camera Capture Module

```
module  CCD_Capture(
input    [11:0]   iDATA,
input             iFVAL,
input             iLVAL,
input             iSTART,
input             iEND,
input             iCLK,
input             iRST,
output   [11:0]   oDATA,
output   [15:0]   oX_Cont,
output   [15:0]   oY_Cont,
output   [31:0]   oFrame_Cont,
output            oDVAL
);
```

```
reg             Pre_FVAL;
reg             mCCD_FVAL;
reg             mCCD_LVAL;
reg     [11:0]  mCCD_DATA;
reg     [15:0]  X_Cont;
reg     [15:0]  Y_Cont;
reg     [31:0]  Frame_Cont;
reg             mSTART;

parameter COLUMN_WIDTH = 1280;

assign  oX_Cont         =       X_Cont;
assign  oY_Cont         =       Y_Cont;
assign  oFrame_Cont     =       Frame_Cont;
assign  oDATA           =       mCCD_DATA;
assign  oDVAL           =       mCCD_FVAL&mCCD_LVAL;

always@(posedge iCLK or negedge iRST)
begin
        if(!iRST)
        mSTART  <=      0;
        else
        begin
                if(iSTART)
                mSTART  <=      1;
                if(iEND)
                mSTART  <=      0;
        end
end

always@(posedge iCLK or negedge iRST)
begin
        if(!iRST)
        begin
                Pre_FVAL        <=      0;
                mCCD_FVAL       <=      0;
                mCCD_LVAL       <=      0;

                X_Cont          <=      0;
                Y_Cont          <=      0;
        end
        else
        begin
                Pre_FVAL        <=      iFVAL;
                if( ({Pre_FVAL,iFVAL}==2'b01) && mSTART )
                mCCD_FVAL       <=      1;
                else if({Pre_FVAL,iFVAL}==2'b10)
                mCCD_FVAL       <=      0;
                mCCD_LVAL       <=      iLVAL;
                if(mCCD_FVAL)
                begin
                        if(mCCD_LVAL)
                        begin
                                if(X_Cont<(COLUMN_WIDTH-1))
                                X_Cont  <= X_Cont+1;
                                else
                                begin
                                        X_Cont  <= 0;
                                        Y_Cont  <= Y_Cont+1;
                                end
                        end
                end
                else
                begin
                        X_Cont  <=      0;
                        Y_Cont  <=      0;
                end
        end
end

always@(posedge iCLK or negedge iRST)
begin
        if(!iRST)
        Frame_Cont      <=      0;
        else
        begin
                if( ({Pre_FVAL,iFVAL}==2'b01) && mSTART )
                Frame_Cont      <=      Frame_Cont+1;
```

```
        end
end

always@(posedge iCLK or negedge iRST)
begin
        if(!iRST)
                mCCD_DATA       <=      0;
        else if (iLVAL)
                mCCD_DATA       <=      iDATA;
        else
                mCCD_DATA       <=      0;
end

reg     ifval_dealy;

wire ifval_fedge;
reg     [15:0]  y_cnt_d;

always@(posedge iCLK or negedge iRST)
begin
        if(!iRST)
                y_cnt_d <=      0;
        else
                y_cnt_d <=      Y_Cont;
end

always@(posedge iCLK or negedge iRST)
begin
        if(!iRST)
                ifval_dealy     <=      0;
        else
                ifval_dealy     <=      iFVAL;
end

assign ifval_fedge = ({ifval_dealy,iFVAL}==2'b10)?1:0;
endmodule
```

A.3 I2C CCD Config Module

```
module I2C_CCD_Config (
//      Host Side
input           iCLK;
input           iRST_N;
input           iZOOM_MODE_SW;
input           iEXPOSURE_ADJ;
input           iEXPOSURE_DEC_p;
//      I2C Side
output          I2C_SCLK;
inout           I2C_SDAT;
};
//      Internal Registers/Wires
reg     [15:0]  mI2C_CLK_DIV;
reg     [31:0]  mI2C_DATA;
reg             mI2C_CTRL_CLK;
reg             mI2C_GO;
wire            mI2C_END;
wire            mI2C_ACK;
reg     [23:0]  LUT_DATA;
reg     [5:0]   LUT_INDEX;
reg     [3:0]   mSetup_ST;
//////////////    CMOS sensor registers setting ////////////////////
parameter       default_exposure                = 16'h07c0;
parameter       exposure_change_value           = 16'd200;

reg     [24:0]  combo_cnt;
wire            combo_pulse;

reg     [1:0]   izoom_mode_sw_delay;

reg     [3:0]   iexposure_adj_delay;
wire            exposure_adj_set;
```

```
wire            exposure_adj_reset;
reg     [15:0]  senosr_exposure;
wire    [17:0]  senosr_exposure_temp;

wire [23:0] sensor_start_row;
wire [23:0] sensor_start_column;
wire [23:0] sensor_row_size;
wire [23:0] sensor_column_size;
wire [23:0] sensor_row_mode;
wire [23:0] sensor_column_mode;

assign sensor_start_row    = iZOOM_MODE_SW ?  24'h010036 : 24'h010000;
assign sensor_start_column = iZOOM_MODE_SW ?  24'h020010 : 24'h020000;
assign sensor_row_size     = iZOOM_MODE_SW ?  24'h0303BF : 24'h03077F;
assign sensor_column_size  = iZOOM_MODE_SW ?  24'h0404FF : 24'h0409FF;
assign sensor_row_mode     = iZOOM_MODE_SW ?  24'h220000 : 24'h220011;
assign sensor_column_mode  = iZOOM_MODE_SW ?  24'h230000 : 24'h230011;

always@(posedge iCLK or negedge iRST_N)
        begin
            if (!iRST_N)
                    begin
                    iexposure_adj_delay <= 0;
                    end
            else
                    begin
                    iexposure_adj_delay <=
                    {iexposure_adj_delay[2:0],iEXPOSURE_ADJ};
                    end
        end
assign  exposure_adj_set =
        ({iexposure_adj_delay[0],iEXPOSURE_ADJ}==2'b10) ? 1 : 0 ;

assign  exposure_adj_reset =
        ({iexposure_adj_delay[3:2]}==2'b10) ? 1 : 0 ;

assign  senosr_exposure_temp = iEXPOSURE_DEC_p ? (senosr_exposure-
        exposure_change_value):(senosr_exposure+exposure_change_value);

always@(posedge iCLK or negedge iRST_N)
        begin
                if (!iRST_N)
                        senosr_exposure <= default_exposure;
                else if (exposure_adj_set|combo_pulse)
                        if (senosr_exposure_temp[17])
                                senosr_exposure <= 0;
                        else if (senosr_exposure_temp[16])
                                senosr_exposure <= 16'hffff;
                        else
                                senosr_exposure <=
                                senosr_exposure_temp[15:0];
        end


always@(posedge iCLK or negedge iRST_N)
        begin
                if (!iRST_N)
                        combo_cnt <= 0;
                else if (!iexposure_adj_delay[3])
                        combo_cnt <= combo_cnt + 1;
                else
                        combo_cnt <= 0;
        end

assign combo_pulse = (combo_cnt == 25'h1fffff) ? 1 : 0;

wire    i2c_reset;
assign i2c_reset = iRST_N & ~exposure_adj_reset & ~combo_pulse ;

////////////////////////////////////////////////////////////////////

//      Clock Setting
parameter       CLK_Freq        =       50000000; //    50 MHz
parameter       I2C_Freq        =       20000;    //    20 KHz
//      LUT Data Number
parameter       LUT_SIZE        =       25;
```

```
//////////////////////    I2C Control Clock        ////////////////////////
always@(posedge iCLK or negedge i2c_reset)
begin
        if(!i2c_reset)
        begin
                mI2C_CTRL_CLK    <=      0;
                mI2C_CLK_DIV     <=      0;
        end
        else
        begin
                if( mI2C_CLK_DIV         < (CLK_Freq/I2C_Freq) )
                mI2C_CLK_DIV     <=      mI2C_CLK_DIV+1;
                else
                begin
                        mI2C_CLK_DIV     <=      0;
                        mI2C_CTRL_CLK    <=      ~mI2C_CTRL_CLK;
                end
        end
end
////////////////////////////////////////////////////////////////////
I2C_Controller  u0 (.CLOCK(mI2C_CTRL_CLK), // Controller Work Clock
                    .I2C_SCLK(I2C_SCLK),   // I2C CLOCK
                    .I2C_SDAT(I2C_SDAT),   // I2C DATA
                    .I2C_DATA(mI2C_DATA),  // DATA
                    .GO(mI2C_GO),          // GO transfor
                    .END(mI2C_END),        // END transfor
                    .ACK(mI2C_ACK),        // ACK
                    .RESET(i2c_reset)
                );
////////////////////////////////////////////////////////////////////
//////////////////////  Config Control  ////////////////////////////
//always@(posedge mI2C_CTRL_CLK or negedge iRST_N)
always@(posedge mI2C_CTRL_CLK or negedge i2c_reset)
begin
        if(!i2c_reset)
        begin
                LUT_INDEX        <=      0;
                mSetup_ST        <=      0;
                mI2C_GO          <=      0;

        end

        else if(LUT_INDEX<LUT_SIZE)
                begin
                        case(mSetup_ST)
                        0:      begin
                                    mI2C_DATA <= {8'hBA,LUT_DATA};
                                    mI2C_GO       <= 1;
                                    mSetup_ST <= 1;
                                end
                        1:      begin
                                    if(mI2C_END)
                                    begin
                                        if(!mI2C_ACK)
                                        mSetup_ST    <= 2;
                                        else
                                        mSetup_ST    <= 0;
                                        mI2C_GO      <= 0;
                                    end
                                end
                        2:      begin
                                    LUT_INDEX <= LUT_INDEX+1;
                                    mSetup_ST <= 0;
                                end
                        endcase
                end
end
////////////////////////////////////////////////////////////////////
//////////////////////    Config Data LUT    ////////////////////////
always
begin
        case(LUT_INDEX)
        0   : LUT_DATA <= 24'h000000;
        1   : LUT_DATA <= 24'h20c000; // Mirror Row and Columns
        2   : LUT_DATA <= {8'h09,senosr_exposure}; // Exposure
        3   : LUT_DATA <= 24'h050000; // H_Blanking
```

```
        4   : LUT_DATA <= 24'h060019; // V_Blanking
        5   : LUT_DATA <= 24'h0A8000; // change latch
        6   : LUT_DATA <= 24'h2B0013; // Green 1 Gain
        7   : LUT_DATA <= 24'h2C009A; // Blue Gain
        8   : LUT_DATA <= 24'h2D019C; // Red Gain
        9   : LUT_DATA <= 24'h2E0013; // Green 2 Gain
        10  : LUT_DATA <= 24'h100051; // set up PLL power on
        11  : LUT_DATA <= 24'h111f04; // PLL_m_Factor<<8+PLL_n_Divider
        12  : LUT_DATA <= 24'h120001; // PLL_p1_Divider
        13  : LUT_DATA <= 24'h100053; // set USE PLL
        14  : LUT_DATA <= 24'h980000; // disble calibration
        15  : LUT_DATA <= 24'hA00000; // Test pattern control
        16  : LUT_DATA <= 24'hA10000; // Test green pattern value
        17  : LUT_DATA <= 24'hA20FFF;           // Test red pattern value
        18  : LUT_DATA <= sensor_start_row;     // set start row
        19  : LUT_DATA <= sensor_start_column;// set start column
        20  : LUT_DATA <= sensor_row_size;      // set row size
        21  : LUT_DATA <= sensor_column_size; // set column size
        22  : LUT_DATA <= sensor_row_mode; // set row mode in bin mode
        23  : LUT_DATA <= sensor_column_mode; // set column bin mode
        24  : LUT_DATA <= 24'h4901A8;           // row black target
        default:LUT_DATA <= 24'h000000;
        endcase
end
endmodule
```

A.4 Raw to RGB Module

```
module  RAW2RGB(
input   [10:0]  iX_Cont;
input   [10:0]  iY_Cont;
input   [11:0]  iDATA;
input           iDVAL;
input           iCLK;
input           iRST;
output  [11:0]  oRed;
output  [11:0]  oGreen;
output  [11:0]  oBlue;
output  [7:0]   oSDFT;
output          oDVAL;
);
wire    [11:0]  mDATA_0;
wire    [11:0]  mDATA_1;
wire    [11:0]  gray;
reg     [11:0]  mDATAd_0;
reg     [11:0]  mDATAd_1;
reg     [11:0]  mCCD_R;
reg     [12:0]  mCCD_G;
reg     [11:0]  mCCD_B;
reg                     mDVAL;

//assign        oRed    =       mCCD_R[11:0];
//assign        oGreen  =       mCCD_G[12:1];
//assign        oBlue   =       mCCD_B[11:0];
assign  oDVAL   =       mDVAL;

Subsystem       u1      (
                         .clk(iCLK),
                         .reset(iRST),
                         .clk_enable(mDVAL),
                         .iRed(mCCD_R[11:4]),
                         .iGreen(mCCD_G[12:5]),
                         .iBlue(mCCD_B[11:4]),
                         .oRed(oRed),
                         .oGreen(oGreen),
                         .oBlue(oBlue)
                        );
//
```

```
Line_Buffer1     u0       (
                           .clken(iDVAL),
                           .clock(iCLK),
                           .shiftin(iDATA),
                           .taps0x(mDATA_1),
                           .taps1x(mDATA_0));

always@(posedge iCLK or negedge iRST)
begin
        if(!iRST)
        begin
                mCCD_R   <= 0;
                mCCD_G   <= 0;
                mCCD_B   <= 0;
                mDATAd_0 <= 0;
                mDATAd_1 <= 0;
                mDVAL    <= 0;
        end
        else
        begin
                mDATAd_0 <= mDATA_0;
                mDATAd_1 <= mDATA_1;
                mDVAL <= {iY_Cont[0]|iX_Cont[0]} ? 1'b0 : iDVAL;
                if({iY_Cont[0],iX_Cont[0]}==2'b10)
                begin
                        mCCD_R  <=      mDATA_0;
                        mCCD_G  <=      mDATAd_0+mDATA_1;
                        mCCD_B  <=      mDATAd_1;
                end
                else if({iY_Cont[0],iX_Cont[0]}==2'b11)
                begin
                        mCCD_R  <=      mDATAd_0;
                        mCCD_G  <=      mDATA_0+mDATAd_1;
                        mCCD_B  <=      mDATA_1;
                end
                else if({iY_Cont[0],iX_Cont[0]}==2'b00)
                begin
                        mCCD_R  <=      mDATA_1;
                        mCCD_G  <=      mDATA_0+mDATAd_1;
                        mCCD_B  <=      mDATAd_0;
                end
                else if({iY_Cont[0],iX_Cont[0]}==2'b01)
                begin
                        mCCD_R  <=      mDATAd_1;
                        mCCD_G  <=      mDATAd_0+mDATA_1;
                        mCCD_B  <=      mDATA_0;
                end
        end
end

endmodule
```

A.5 SDRAM Controller Module

```
module Sdram_Control (
                //      HOST Side
                RESET_N,
                CLK,
                //      FIFO Write Side 1
                WR1_DATA,
                WR1,
                WR1_ADDR,
                WR1_MAX_ADDR,
                WR1_LENGTH,
                WR1_LOAD,
                WR1_CLK,
                //      FIFO Write Side 2
                WR2_DATA,
                WR2,
                WR2_ADDR,
                WR2_MAX_ADDR,
                WR2_LENGTH,
                WR2_LOAD,
```

```
                WR2_CLK,
                //      FIFO Read Side 1
                RD1_DATA,
                RD1,
                RD1_ADDR,
                RD1_MAX_ADDR,
                RD1_LENGTH,
                RD1_LOAD,
                RD1_CLK,
                //      FIFO Read Side 2
                RD2_DATA,
                RD2,
                RD2_ADDR,
                RD2_MAX_ADDR,
                RD2_LENGTH,
                RD2_LOAD,
                RD2_CLK,
                //      SDRAM Side
                SA,
                BA,
                CS_N,
                CKE,
                RAS_N,
                CAS_N,
                WE_N,
                DQ,
                DQM
);
//=============================================================
//  PARAMETER declarations
//=============================================================
`include  "Sdram_Params.h"
//=============================================================
//  PORT declarations
//=============================================================
//      HOST Side
input                       RESET_N;        //System Reset
input                       CLK;
//      FIFO Write Side 1
input   [`DIOSIZE-1:0]      WR1_DATA;       //Data Input
input                       WR1;            //Write Request
input     [`ASIZE-1:0]      WR1_ADDR;       //Write Start Address
input     [`ASIZE-1:0]      WR1_MAX_ADDR;   //Write Max Address
input           [7:0]       WR1_LENGTH;     //Write Length
input                       WR1_LOAD;       //Write FIFO Clear
input                       WR1_CLK;        //Write FIFO Clock
//      FIFO Write Side 2
input   [`DIOSIZE-1:0]      WR2_DATA;       //Data Input
input                       WR2;            //Write Request
input     [`ASIZE-1:0]      WR2_ADDR;       //Write Start Address
input     [`ASIZE-1:0]      WR2_MAX_ADDR;   //Write Max Address
input           [7:0]       WR2_LENGTH;     //Write Length
input                       WR2_LOAD;       //Write FIFO Clear
input                       WR2_CLK;        //Write FIFO Clock
//      FIFO Read Side 1
output  [`DIOSIZE-1:0]      RD1_DATA;       //Data Output
input                       RD1;            //Read Request
input     [`ASIZE-1:0]      RD1_ADDR;       //Read Start Address
input     [`ASIZE-1:0]      RD1_MAX_ADDR;   //Read Max Address
input           [7:0]       RD1_LENGTH;     //Read Length
input                       RD1_LOAD;       //Read FIFO Clear
input                       RD1_CLK;        //Read FIFO Clock
//      FIFO Read Side 2
output  [`DIOSIZE-1:0]      RD2_DATA;       //Data Output
input                       RD2;            //Read Request
input     [`ASIZE-1:0]      RD2_ADDR;       //Read Start Address
input     [`ASIZE-1:0]      RD2_MAX_ADDR;   //Read Max Address
input           [7:0]       RD2_LENGTH;     //Read Length
input                       RD2_LOAD;     //Read FIFO Clear
input                       RD2_CLK;      //Read FIFO Clock
//      SDRAM Side
output         [11:0]       SA;           //SDRAM address output
output          [1:0]       BA;           //SDRAM bank address
```

```
output          [1:0]     CS_N;          //SDRAM Chip Selects
output                    CKE;           //SDRAM clock enable
output                    RAS_N;         //SDRAM Row address Strobe
output                    CAS_N;         //SDRAM Column address Strobe
output                    WE_N;          //SDRAM write enable
inout    [`DSIZE-1:0]     DQ;            //SDRAM data bus
output [`DSIZE/8-1:0]     DQM;           //SDRAM data mask lines

//=============================================================
//  Signal Declarations
//=============================================================
//      Controller
reg      [`ASIZE-1:0]     mADDR;         //Internal address
reg              [7:0]     mLENGTH;       //Internal length
reg      [`ASIZE-1:0]     rWR1_ADDR;     //Reg write address
reg      [`ASIZE-1:0]     rWR1_MAX_ADDR; //Reg max write address
reg              [7:0]     rWR1_LENGTH;   //Reg write length
reg      [`ASIZE-1:0]     rWR2_ADDR;     //Reg write address
reg      [`ASIZE-1:0]     rWR2_MAX_ADDR; //Reg max write address
reg              [7:0]     rWR2_LENGTH;   //Reg write length
reg      [`ASIZE-1:0]     rRD1_ADDR;     //Reg read address
reg      [`ASIZE-1:0]     rRD1_MAX_ADDR; //Reg max read address
reg              [7:0]     rRD1_LENGTH;   //Reg read length
reg      [`ASIZE-1:0]     rRD2_ADDR;     //Reg read address
reg      [`ASIZE-1:0]     rRD2_MAX_ADDR; //Reg max read address
reg              [7:0]     rRD2_LENGTH;   //Reg read length
reg              [1:0]     WR_MASK;       //Write port active mask
reg              [1:0]     RD_MASK;    //Read port active mask
reg                        mWR_DONE;   //Flag wr done, 1 pulse SDR_CLK
reg                        mRD_DONE;   //Flag rd done, 1 pulse SDR_CLK
reg                        mWR,Pre_WR;//Internal WR edge capture
reg                        mRD,Pre_RD;//Internal RD edge capture
reg              [9:0]     ST;         //Controller status
reg              [1:0]     CMD;        //Controller command
reg                        PM_STOP;    //Flag page mode stop
reg                        PM_DONE;    //Flag page mode done
reg                        Read;       //Flag read active
reg                        Write;      //Flag write active
reg      [`DSIZE-1:0]     DATAOUT;    //Controller Data output
wire     [`DSIZE-1:0]     mDATAIN;    //Controller Data input
wire     [`DSIZE-1:0]     mDATAIN1;   //Controller Data input 1
wire     [`DSIZE-1:0]     mDATAIN2;   //Controller Data input 2
wire                       CMDACK;     //Controller command ackledg
//      DRAM Control
reg    [`DSIZE/8-1:0]     DQM;        //SDRAM data mask lines
reg             [11:0]     SA;         //SDRAM address output
reg              [1:0]     BA;         //SDRAM bank address
reg              [1:0]     CS_N;       //SDRAM Chip Selects
reg                        CKE;        //SDRAM clock enable
reg                        RAS_N;      //SDRAM Row address Strobe
reg                        CAS_N;      //SDRAM Column address Strobe
reg                        WE_N;       //SDRAM write enable
wire     [`DSIZE-1:0]     DQOUT;      //SDRAM data out link
wire   [`DSIZE/8-1:0]     IDQM;       //SDRAM data mask lines
wire            [11:0]     ISA;        //SDRAM address output
wire             [1:0]     IBA;        //SDRAM bank address
wire             [1:0]     ICS_N;      //SDRAM Chip Selects
wire                       ICKE;       //SDRAM clock enable
wire                       IRAS_N;     //SDRAM Row address Strobe
wire                       ICAS_N;     //SDRAM Column address Strobe
wire                       IWE_N;      //SDRAM write enable
//      FIFO Control
reg                        OUT_VALID; //Output data rqst to read fifo
reg                        IN_REQ;    //Input data rqst to write fifo
wire             [7:0]     write_side_fifo_rusedw1;
wire             [7:0]     write_side_fifo_rusedw2;
wire             [7:0]     read_side_fifo_wusedw1;
wire             [7:0]     read_side_fifo_wusedw2;
//      DRAM Internal Control
```

```
wire     [`ASIZE-1:0]     saddr;
wire                      load_mode;
wire                      nop;
wire                      reada;
wire                      writea;
wire                      refresh;
wire                      precharge;
wire                      oe;
wire                      ref_ack;
wire                      ref_req;
wire                      init_req;
wire                      cm_ack;
wire                      active;


//=============================================================
//  Sub-module
//=============================================================
control_interface  u_control_interface (
        .CLK(CLK),
        .RESET_N(RESET_N),
        .CMD(CMD),
        .ADDR(mADDR),
        .REF_ACK(ref_ack),
        .CM_ACK(cm_ack),
        .NOP(nop),
        .READA(reada),
        .WRITEA(writea),
        .REFRESH(refresh),
        .PRECHARGE(precharge),
        .LOAD_MODE(load_mode),
        .SADDR(saddr),
        .REF_REQ(ref_req),
        .INIT_REQ(init_req),
        .CMD_ACK(CMDACK) );

command   u_command (
         .CLK(CLK),
         .RESET_N(RESET_N),
         .SADDR(saddr),
         .NOP(nop),
         .READA(reada),
         .WRITEA(writea),
         .REFRESH(refresh),
        .LOAD_MODE(load_mode),
         .PRECHARGE(precharge),
         .REF_REQ(ref_req),
        .INIT_REQ(init_req),
         .REF_ACK(ref_ack),
         .CM_ACK(cm_ack),
         .OE(oe),
        .PM_STOP(PM_STOP),
        .PM_DONE(PM_DONE),
         .SA(ISA),
         .BA(IBA),
         .CS_N(ICS_N),
         .CKE(ICKE),
         .RAS_N(IRAS_N),
         .CAS_N(ICAS_N),
         .WE_N(IWE_N) );

sdr_data_path  u_sdr_data_path (
         .CLK(CLK),
         .RESET_N(RESET_N),
         .DATAIN(mDATAIN),
         .DM(2'b00),
         .DQOUT(DQOUT),
         .DQM(IDQM) );

Sdram_WR_FIFO  u_write1_fifo (
                                  .data(WR1_DATA),
                                  .wrreq(WR1),
                                  .wrclk(WR1_CLK),
                                  .aclr(WR1_LOAD),
```

```
                                .rdreq(IN_REQ&&WR_MASK[0]),
                                .rdclk(CLK),
                                .q(mDATAIN1),
                                .rdusedw(write_side_fifo_rusedw1));
Sdram_WR_FIFO   u_write2_fifo (
                                .data(WR2_DATA),
                                .wrreq(WR2),
                                .wrclk(WR2_CLK),
                                .aclr(WR2_LOAD),
                                .rdreq(IN_REQ&&WR_MASK[1]),
                                .rdclk(CLK),
                                .q(mDATAIN2),
                                .rdusedw(write_side_fifo_rusedw2));
Sdram_RD_FIFO   u_read1_fifo (
                                .data(mDATAOUT),
                                .wrreq(OUT_VALID&&RD_MASK[0]),
                                .wrclk(CLK),
                                .aclr(RD1_LOAD),
                                .rdreq(RD1),
                                .rdclk(RD1_CLK),
                                .q(RD1_DATA),
                                .wrusedw(read_side_fifo_wusedw1));
Sdram_RD_FIFO   u_read2_fifo (
                                .data(mDATAOUT),
                                .wrreq(OUT_VALID&&RD_MASK[1]),
                                .wrclk(CLK),
                                .aclr(RD2_LOAD),
                                .rdreq(RD2),
                                .rdclk(RD2_CLK),
                                .q(RD2_DATA),
                                .wrusedw(read_side_fifo_wusedw2));

//=============================================================
//  Structural coding
//=============================================================
assign mDATAIN  = (WR_MASK[0])  ? mDATAIN1 : mDATAIN2;
assign DQ       = oe    ? DQOUT : 'DSIZE'hzzzz;
assign active   =       Read | Write;


always @ (posedge CLK)
begin
    SA      <= (ST==SC_CL+mLENGTH) ? 12'h200 : ISA;
    BA      <= IBA;
    CS_N    <= ICS_N;
    CKE     <= ICKE;
    RAS_N   <= (ST==SC_CL+mLENGTH) ? 1'b0 : IRAS_N;
    CAS_N   <= (ST==SC_CL+mLENGTH) ? 1'b1 : ICAS_N;
    WE_N    <= (ST==SC_CL+mLENGTH) ? 1'b0 : IWE_N;
    PM_STOP     <= (ST==SC_CL+mLENGTH) ? 1'b1 : 1'b0;
    PM_DONE     <= (ST==SC_CL+SC_RCD+mLENGTH+2) ? 1'b1 : 1'b0;
    DQM         <= (active && (ST>=SC_CL)) ? (((ST==SC_CL+mLENGTH)
    && Write)? 4'b1111 : 4'b0) : 4'b1111;
    mDATAOUT    <= DQ;
end

always@(posedge CLK or negedge RESET_N)
begin
        if(!RESET_N)
        begin
                CMD         <= 0;
                ST          <= 0;
                Pre_RD      <= 0;
                Pre_WR      <= 0;
                Read        <= 0;
                Write       <= 0;
                OUT_VALID   <= 0;
                IN_REQ      <= 0;
                mWR_DONE    <= 0;
                mRD_DONE    <= 0;
        end
```

```
        else
        begin
                Pre_RD      <= mRD;
                Pre_WR      <= mWR;
                case (ST)
                0:      begin
                                if (!Pre_RD && mRD)
                                begin
                                        Read    <= 1;
                                        Write   <= 0;
                                        CMD     <= 2'b01;
                                        ST      <= 1;
                                end
                                else if (!Pre_WR && mWR)
                                begin
                                        Read    <= 0;
                                        Write   <= 1;
                                        CMD     <= 2'b10;
                                        ST      <= 1;
                                end
                        end
                1:      begin
                                if (CMDACK)
                                begin
                                        CMD     <=  2'b00;
                                        ST      <=  2;
                                end
                        end
                default:  begin
                                if (ST!=SC_CL+SC_RCD+mLENGTH+1)
                                  ST    <=  ST+1;
                                else
                                  ST    <=  0;
                        end
                endcase

                if (Read)
                begin
                        if (ST==SC_CL+SC_RCD+1)
                          OUT_VALID     <=      1;
                        else if (ST==SC_CL+SC_RCD+mLENGTH+1)
                        begin
                                OUT_VALID       <=      0;
                                Read            <=      0;
                                mRD_DONE        <=      1;
                        end
                end
                else
                  mRD_DONE          <=    0;

                if (Write)
                begin
                        if (ST==SC_CL-1)
                          IN_REQ        <=      1;
                        else if (ST==SC_CL+mLENGTH-1)
                          IN_REQ        <=      0;
                        else if (ST==SC_CL+SC_RCD+mLENGTH)
                        begin
                                Write   <=      0;
                                mWR_DONE<=      1;
                        end
                end
                else
                  mWR_DONE  <=  0;
        end
end

//      Internal Address & Length Control
always@(posedge CLK or negedge RESET_N)
        if (!RESET_N)
        begin
                rWR1_ADDR           <=    WR1_ADDR;
                rWR2_ADDR           <=    WR2_ADDR;
                rRD1_ADDR           <=    RD1_ADDR;
                rRD2_ADDR           <=    RD2_ADDR;
                rWR1_MAX_ADDR       <=    WR1_MAX_ADDR;
                rWR2_MAX_ADDR       <=    WR2_MAX_ADDR;
                rRD1_MAX_ADDR       <=    RD1_MAX_ADDR;
                rRD2_MAX_ADDR       <=    RD2_MAX_ADDR;
```

```
                rWR1_LENGTH         <=    WR1_LENGTH;
                rWR2_LENGTH         <=    WR2_LENGTH;
                rRD1_LENGTH         <=    RD1_LENGTH;
                rRD2_LENGTH         <=    RD2_LENGTH;
        end
        else
        begin
                //      Write Side 1
    if (mWR_DONE&&WR_MASK[0])
            begin
                if(rWR1_ADDR<rWR1_MAX_ADDR-rWR1_LENGTH)
                        rWR1_ADDR <= rWR1_ADDR+rWR1_LENGTH;
                else
                        rWR1_ADDR <= WR1_ADDR;
            end
                //      Write Side 2
            if (mWR_DONE&&WR_MASK[1])
            begin
                if(rWR2_ADDR<rWR2_MAX_ADDR-rWR2_LENGTH)
                        rWR2_ADDR <= rWR2_ADDR+rWR2_LENGTH;
                else
                        rWR2_ADDR <= WR2_ADDR;
            end
                //      Read Side 1
            if (mRD_DONE&&RD_MASK[0])
            begin
                if(rRD1_ADDR<rRD1_MAX_ADDR-rRD1_LENGTH)
                        rRD1_ADDR <= rRD1_ADDR+rRD1_LENGTH;
                else
                        rRD1_ADDR <= RD1_ADDR;
            end
                //      Read Side 2
            if (mRD_DONE&&RD_MASK[1])
            begin
                if(rRD2_ADDR<rRD2_MAX_ADDR-rRD2_LENGTH)
                        rRD2_ADDR <= rRD2_ADDR+rRD2_LENGTH;
                else
                        rRD2_ADDR <= RD2_ADDR;
            end
    end
//      Auto Read/Write Control
always@(posedge CLK or negedge RESET_N)
        if (!RESET_N)
        begin
                mWR     <=      0;
                mRD     <=      0;
                mADDR   <=      0;
                mLENGTH <=      0;
                RD_MASK <=      0; //Peli
                WR_MASK <=      0;
        end
        else
        begin
            if ((mWR==0) && (mRD==0) && (ST==0) &&
                (WR_MASK==0) &&     (RD_MASK==0) &&
                (WR1_LOAD==0) &&    (RD1_LOAD==0) &&
                (WR2_LOAD==0) &&    (RD2_LOAD==0))
            begin
                //  Write Side 1
                if ((write_side_fifo_rusedw1 >=
                    rWR1_LENGTH) && (rWR1_LENGTH!=0))
                begin
                        mADDR       <=      rWR1_ADDR;
                        mLENGTH     <=      rWR1_LENGTH;
                        WR_MASK     <=      2'b01;
                        RD_MASK     <=      2'b00;
                        mWR         <=      1;
                        mRD         <=      0;
                end
                    //      Write Side 2
                else if ((write_side_fifo_rusedw2 >=
                    rWR2_LENGTH) && (rWR2_LENGTH!=0))
                begin
                        mADDR       <=      rWR2_ADDR;
                        mLENGTH     <=      rWR2_LENGTH;
                        WR_MASK     <=      2'b10;
```

```
                        RD_MASK     <=      2'b00;
                        mWR <=      1;
                        mRD <=      0;
                end
                    //      Read Side 1
                else if ((read_side_fifo_wusedw1 < rRD1_LENGTH))
                begin
                        mADDR       <=      rRD1_ADDR;
                        mLENGTH     <=      rRD1_LENGTH;
                        WR_MASK     <=      2'b00;
                        RD_MASK     <=      2'b01;
                        mWR <=      0;
                        mRD <=      1;
                end
                    //      Read Side 2
                else if ((read_side_fifo_wusedw2 < rRD2_LENGTH))
                begin
                        mADDR       <=      rRD2_ADDR;
                        mLENGTH     <=      rRD2_LENGTH;
                        WR_MASK     <=      2'b00;
                        RD_MASK     <=      2'b10;
                        mWR <=      0;
                        mRD <=      1;
                end
        end

        if (mWR_DONE)
        begin
                WR_MASK     <=      0;
                mWR <=      0;
        end

        if (mRD_DONE)
        begin
                RD_MASK     <=      0;
                mRD <=      0;
        end
    end
endmodule
```

A.6 Subsystem Module

```
module Subsystem(
input   clk;
input   reset;
input   clk_enable;
input   [7:0] iRed;  // uint8
input   [7:0] iGreen;  // uint8
input   [7:0] iBlue;  // uint8
output  ce_out;
output  [7:0] oRed;  // uint8
output  [7:0] oGreen;  // uint8
output  [7:0] oBlue;  // uint8
);

  wire enb;
  wire [7:0] R_Const_out1;  // ufix8_En7
  wire [15:0] Rx_mul_temp;  // ufix16_En7
  wire signed [15:0] Rx_out1;  // sfix16_En4
  wire [7:0] G_Const_out1;  // ufix8_En7
  wire [15:0] Gx_mul_temp;  // ufix16_En7
  wire signed [15:0] Gx_out1;  // sfix16_En4
  wire [7:0] B_Const_out1;  // ufix8_En7
  wire [15:0] Bx_mul_temp;  // ufix16_En7
  wire signed [15:0] Bx_out1;  // sfix16_En4
  wire signed [31:0] Add_add_cast;  // sfix32_En4
  wire signed [31:0] Add_add_cast_1;  // sfix32_En4
  wire signed [31:0] Add_add_temp;  // sfix32_En4
  wire signed [31:0] Add_add_cast_2;  // sfix32_En4
  wire signed [31:0] Add_add_temp_1;  // sfix32_En4
  wire signed [15:0] Add_out1;  // sfix16_En4
```

```
reg [9:0] HDL_Row_Counter_out1;  // ufix10
reg [9:0] HDL_Row_Counter_stepreg;  // ufix10
wire [9:0] mean_const1_out1;  // ufix10
wire Relational_Operator3_relop1;
reg signed [15:0] Delay_reg [0:1];  // sfix16 [2]
wire signed [15:0] Delay_reg_next [0:1];  // sfix16_En4 [2]
wire signed [15:0] Delay_out1;  // sfix16_En4
wire signed [31:0] Sum_sub_cast;  // sfix32_En4
wire signed [31:0] Sum_sub_cast_1;  // sfix32_En4
wire signed [31:0] Sum_sub_temp;  // sfix32_En4
wire signed [15:0] Sum_out1;  // sfix16_En4
reg [18:0] HDL_Frame_Counter_out1;  // ufix19
reg [18:0] HDL_Frame_Counter_stepreg;  // ufix19
wire [18:0] mean_const_out1;  // ufix19
wire Relational_Operator1_relop1;
wire signed [15:0] Delay1280_out1;  // sfix16_En4
wire signed [31:0] Sum1_sub_cast;  // sfix32_En4
wire signed [31:0] Sum1_sub_cast_1;  // sfix32_En4
wire signed [31:0] Sum1_sub_temp;  // sfix32_En4
wire signed [15:0] Sum1_out1;  // sfix16_En4
wire signed [15:0] Sum2_out1;  // sfix16_En4
wire signed [15:0] Delay640_out1;  // sfix16_En4
wire signed [31:0] Sum2_add_cast;  // sfix32_En4
wire signed [31:0] Sum2_add_cast_1;  // sfix32_En4
wire signed [31:0] Sum2_add_temp;  // sfix32_En4
wire signed [15:0] COS_2piK1_N_1_out1;  // sfix16_En8
reg signed [15:0] Delay4_switch_delay;  // sfix16
wire signed [15:0] Delay4_out1;  // sfix16_En8
wire signed [31:0] Sum4_add_cast;  // sfix32_En8
wire signed [31:0] Sum4_add_cast_1;  // sfix32_En8
wire signed [31:0] Sum4_add_temp;  // sfix32_En8
wire signed [15:0] Sum4_out1;  // sfix16_En4
wire signed [16:0] COS_2piK1_N_1_cast;  // sfix17_En4
wire signed [16:0] COS_2piK1_N_1_cast_1;  // sfix17_En4
wire signed [31:0] COS_2piK1_N_1_cast_2;  // sfix32_En19
wire signed [31:0] Gain6_mul_temp;  // sfix32_En27
wire signed [15:0] Gain6_out1;  // sfix16_En4
wire signed [31:0] Product6_mul_temp;  // sfix32_En8
wire signed [15:0] Product6_out1;  // sfix16_En4
wire signed [15:0] COS_2piK2_N_1_out1;  // sfix16_En4
wire signed [15:0] Delay640_1_out1;  // sfix16_En4
wire signed [31:0] Sum9_add_cast;  // sfix32_En4
wire signed [31:0] Sum9_add_cast_1;  // sfix32_En4
wire signed [31:0] Sum9_add_temp;  // sfix32_En4
wire signed [15:0] Sum9_out1;  // sfix16_En4
wire signed [16:0] COS_2piK2_N_1_cast;  // sfix17_En4
wire signed [16:0] COS_2piK2_N_1_cast_1;  // sfix17_En4
wire signed [31:0] COS_2piK2_N_1_cast_2;  // sfix32_En19
wire signed [15:0] Sum11_out1;  // sfix16_En4
reg signed [15:0] Delay11_switch_delay;  // sfix16
wire signed [15:0] Delay11_out1;  // sfix16_En4
wire signed [31:0] Sum11_add_cast;  // sfix32_En4
wire signed [31:0] Sum11_add_cast_1;  // sfix32_En4
wire signed [31:0] Sum11_add_temp;  // sfix32_En4
wire signed [31:0] Gain2_mul_temp;  // sfix32_En23
wire signed [15:0] Gain2_out1;  // sfix16_En4
wire signed [31:0] Product1_mul_temp;  // sfix32_En8
wire signed [15:0] Product1_out1;  // sfix16_En4
wire signed [31:0] Add6_add_cast;  // sfix32_En4
wire signed [31:0] Add6_add_cast_1;  // sfix32_En4
wire signed [31:0] Add6_add_temp;  // sfix32_En4
wire signed [15:0] Add6_out1;  // sfix16_En4
wire [15:0] Data_Type_Conversion_out1;  // ufix16_En4
wire [15:0] Sqrt_out1;  // ufix16_En4
wire [31:0] Gain3_mul_temp;  // ufix32_En16
wire [15:0] Gain3_out1;  // ufix16_En4
wire [15:0] Threshold_out1;  // ufix16_En4
wire Relational_Operator2_relop1;
wire switch_compare_1;
wire [7:0] Max_intensity_out1;  // uint8
```

```
wire [7:0] Red_out1;  // uint8
wire switch_compare_1_1;
wire [7:0] min_intensity_out1;  // uint8
wire [7:0] Green_out1;  // uint8
wire switch_compare_1_2;
wire [7:0] Blue_out1;  // uint8

// Gray = 0.299 * R + 0.587 * G + 0.114 * B

assign R_Const_out1 = 8'b00100110;
assign Rx_mul_temp = iRed * R_Const_out1;
assign Rx_out1 = {3'b0, Rx_mul_temp[15:3]};

assign G_Const_out1 = 8'b01001011;
assign Gx_mul_temp = iGreen * G_Const_out1;
assign Gx_out1 = {3'b0, Gx_mul_temp[15:3]};

assign B_Const_out1 = 8'b00001111;
assign Bx_mul_temp = iBlue * B_Const_out1;
assign Bx_out1 = {3'b0, Bx_mul_temp[15:3]};

assign Add_add_cast = {{16{Rx_out1[15]}}, Rx_out1};
assign Add_add_cast_1 = {{16{Gx_out1[15]}}, Gx_out1};
assign Add_add_temp = Add_add_cast + Add_add_cast_1;
assign Add_add_cast_2 = {{16{Bx_out1[15]}}, Bx_out1};
assign Add_add_temp_1 = Add_add_temp + Add_add_cast_2;
assign Add_out1 = Add_add_temp_1[15:0];

assign enb = clk_enable;

// Count limited, Unsigned Counter
//  initial value   = 1
//  step value      = 1
//  count to value  = 640
always @(posedge clk or negedge reset)
  begin : HDL_Row_Counter_step_process
    if (reset == 1'b0) begin
      HDL_Row_Counter_stepreg <= 10'b0000000001;
    end
    else begin
      if (enb) begin
        if (HDL_Row_Counter_out1 == 10'b1001111111) begin
          HDL_Row_Counter_stepreg <= 10'b0110000001;
        end
        else begin
          HDL_Row_Counter_stepreg <= 10'b0000000001;
        end
      end
    end
  end

always @(posedge clk or negedge reset)
  begin : HDL_Row_Counter_process
  if (reset == 1'b0) begin
    HDL_Row_Counter_out1 <= 10'b0000000001;
  end
  else begin
    if (enb) begin
    HDL_Row_Counter_out1 <= HDL_Row_Counter_out1 +
    HDL_Row_Counter_stepreg;
    end
  end
  end

assign mean_const1_out1 = 10'b0000000001;

assign Relational_Operator3_relop1 = HDL_Row_Counter_out1 ==
mean_const1_out1;

always @(posedge clk or negedge reset)
  begin : Delay_process
    if (reset == 1'b0) begin
      Delay_reg[0] <= 16'sb0000000000000000;
      Delay_reg[1] <= 16'sb0000000000000000;
    end
    else begin
      if (enb) begin
```

```
        if (Relational_Operator3_relop1 == 1'b1) begin
          Delay_reg[0] <= 16'sb0000000000000000;
          Delay_reg[1] <= 16'sb0000000000000000;
        end
        else begin
          Delay_reg[0] <= Delay_reg_next[0];
          Delay_reg[1] <= Delay_reg_next[1];
        end
      end
    end
  end

assign Delay_out1 = (Relational_Operator3_relop1 == 1'b1 ?
16'sb0000000000000000 :
            Delay_reg[1]);
assign Delay_reg_next[0] = Add_out1;
assign Delay_reg_next[1] = Delay_reg[0];

assign Sum_sub_cast = {{16{Add_out1[15]}}, Add_out1};
assign Sum_sub_cast_1 = {{16{Delay_out1[15]}}, Delay_out1};
assign Sum_sub_temp = Sum_sub_cast - Sum_sub_cast_1;
assign Sum_out1 = Sum_sub_temp[15:0];

// Count limited, Unsigned Counter
//  initial value   = 1
//  step value      = 1
//  count to value  = 307200
always @(posedge clk or negedge reset)
  begin : HDL_Frame_Counter_step_process
    if (reset == 1'b0) begin
      HDL_Frame_Counter_stepreg <= 19'b0000000000000000001;
    end
    else begin
      if (enb) begin
        if (HDL_Frame_Counter_out1 == 19'b1001010111111111111)
        begin
          HDL_Frame_Counter_stepreg <= 19'b0110101000000000001;
        end
        else begin
          HDL_Frame_Counter_stepreg <= 19'b0000000000000000001;
        end
      end
    end
  end

always @(posedge clk or negedge reset)
  begin : HDL_Frame_Counter_process
    if (reset == 1'b0) begin
      HDL_Frame_Counter_out1 <= 19'b0000000000000000001;
    end
    else begin
      if (enb) begin
        HDL_Frame_Counter_out1 <= HDL_Frame_Counter_out1 +
        HDL_Frame_Counter_stepreg;
      end
    end
  end

assign mean_const_out1 = 19'b0000000000000000001;

assign Relational_Operator1_relop1 = HDL_Frame_Counter_out1 ==
mean_const_out1;

Delay1280 u_Delay1280 (.clk(clk),
                       .reset(reset),
                       .enb(clk_enable),
                       .In1(Sum_out1),  // sfix16_En4
                       .In2(Relational_Operator1_relop1),
                       .Out1(Delay1280_out1)  // sfix16_En4
                       );

assign Sum1_sub_cast = {{16{Sum_out1[15]}}, Sum_out1};
assign Sum1_sub_cast_1 = {{16{Delay1280_out1[15]}},
Delay1280_out1};
assign Sum1_sub_temp = Sum1_sub_cast - Sum1_sub_cast_1;
assign Sum1_out1 = Sum1_sub_temp[15:0];
```

```
Delay640 u_Delay640 (.clk(clk),
                     .reset(reset),
                     .enb(clk_enable),
                     .In1(Sum2_out1),  // sfix16_En4
                     .In2(Relational_Operator1_relop1),
                     .Out1(Delay640_out1)  // sfix16_En4
                     );

assign Sum2_add_cast = {{16{Sum1_out1[15]}}, Sum1_out1};
assign Sum2_add_cast_1 = {{16{Delay640_out1[15]}},
Delay640_out1};
assign Sum2_add_temp = Sum2_add_cast + Sum2_add_cast_1;
assign Sum2_out1 = Sum2_add_temp[15:0];


always @(posedge clk or negedge reset)
  begin : Delay4_process
    if (reset == 1'b0) begin
      Delay4_switch_delay <= 16'sb0000000000000000;
    end
    else begin
      if (enb) begin
        if (Relational_Operator3_relop1 == 1'b1) begin
          Delay4_switch_delay <= 16'sb0000000000000000;
        end
        else begin
          Delay4_switch_delay <= COS_2piK1_N_1_out1;
        end
      end
    end
  end

assign Delay4_out1 = (Relational_Operator3_relop1 == 1'b1 ?
16'sb0000000000000000 :
            Delay4_switch_delay);

assign Sum4_add_cast = {{12{Sum2_out1[15]}}, {Sum2_out1,
4'b0000}};
assign Sum4_add_cast_1 = {{16{Delay4_out1[15]}}, Delay4_out1};
assign Sum4_add_temp = Sum4_add_cast + Sum4_add_cast_1;
assign Sum4_out1 = Sum4_add_temp[19:4];

assign COS_2piK1_N_1_cast = {Sum4_out1[15], Sum4_out1};
assign COS_2piK1_N_1_cast_1 =  - (COS_2piK1_N_1_cast);
assign COS_2piK1_N_1_cast_2 = {COS_2piK1_N_1_cast_1,
15'b000000000000000};
assign COS_2piK1_N_1_out1 = COS_2piK1_N_1_cast_2[26:11];

assign Gain6_mul_temp = 16'sb0110011001100110 *
COS_2piK1_N_1_out1;
assign Gain6_out1 = {{7{Gain6_mul_temp[31]}},
Gain6_mul_temp[31:23]};

assign Product6_mul_temp = Gain6_out1 * Gain6_out1;
assign Product6_out1 = Product6_mul_temp[19:4];

Delay640_1 u_Delay640_1 (.clk(clk),
                         .reset(reset),
                         .enb(clk_enable),
                         .In1(COS_2piK2_N_1_out1),  // sfix16_En4
                         .In2(Relational_Operator1_relop1),
                         .Out1(Delay640_1_out1)  // sfix16_En4
                         );

assign Sum9_add_cast = {{16{Sum1_out1[15]}}, Sum1_out1};
assign Sum9_add_cast_1 = {{16{Delay640_1_out1[15]}},
Delay640_1_out1};
assign Sum9_add_temp = Sum9_add_cast + Sum9_add_cast_1;
assign Sum9_out1 = Sum9_add_temp[15:0];

assign COS_2piK2_N_1_cast = {Sum9_out1[15], Sum9_out1};
assign COS_2piK2_N_1_cast_1 =  - (COS_2piK2_N_1_cast);
assign COS_2piK2_N_1_cast_2 = {COS_2piK2_N_1_cast_1,
```

```
15'b0000000000000000};
assign COS_2piK2_N_1_out1 = COS_2piK2_N_1_cast_2[30:15];

always @(posedge clk or negedge reset)
  begin : Delay11_process
    if (reset == 1'b0) begin
      Delay11_switch_delay <= 16'sb0000000000000000;
    end
    else begin
      if (enb) begin
        if (Relational_Operator3_relop1 == 1'b1) begin
          Delay11_switch_delay <= 16'sb0000000000000000;
        end
        else begin
          Delay11_switch_delay <= Sum11_out1;
        end
      end
    end
  end

assign Delay11_out1 = (Relational_Operator3_relop1 == 1'b1 ?
16'sb0000000000000000 :
          Delay11_switch_delay);


assign Sum11_add_cast = {{16{COS_2piK2_N_1_out1[15]}},
COS_2piK2_N_1_out1};
assign Sum11_add_cast_1 = {{16{Delay11_out1[15]}}, Delay11_out1};
assign Sum11_add_temp = Sum11_add_cast + Sum11_add_cast_1;
assign Sum11_out1 = Sum11_add_temp[15:0];

assign Gain2_mul_temp = 16'sb0110011001100110 * Sum11_out1;
assign Gain2_out1 = {{3{Gain2_mul_temp[31]}},
Gain2_mul_temp[31:19]};

assign Product1_mul_temp = Gain2_out1 * Gain2_out1;
assign Product1_out1 = Product1_mul_temp[19:4];

assign Add6_add_cast = {{16{Product6_out1[15]}}, Product6_out1};
assign Add6_add_cast_1 = {{16{Product1_out1[15]}},
Product1_out1};
assign Add6_add_temp = Add6_add_cast + Add6_add_cast_1;
assign Add6_out1 = Add6_add_temp[15:0];

assign Data_Type_Conversion_out1 = Add6_out1;

Sqrt u_Sqrt (.din(Data_Type_Conversion_out1),  // ufix16_En4
             .dout(Sqrt_out1)  // ufix16_En4
             );

assign Gain3_mul_temp = 16'b1010000000000000 * Sqrt_out1;
assign Gain3_out1 = Gain3_mul_temp[27:12];

assign Threshold_out1 = 16'b0000000001110000;

assign Relational_Operator2_relop1 = Gain3_out1 >=
Threshold_out1;

assign switch_compare_1 = Relational_Operator2_relop1 == 1'b1;

assign Max_intensity_out1 = 8'b11111111;

assign Red_out1 = (switch_compare_1 == 1'b0 ? iRed :
          Max_intensity_out1);

assign oRed = Red_out1;

assign switch_compare_1_1 = Relational_Operator2_relop1 == 1'b1;

assign min_intensity_out1 = 8'b00000000;

assign Green_out1 = (switch_compare_1_1 == 1'b0 ? iGreen :
          min_intensity_out1);

assign oGreen = Green_out1;

assign switch_compare_1_2 = Relational_Operator2_relop1 == 1'b1;

assign Blue_out1 = (switch_compare_1_2 == 1'b0 ? iBlue :
```

```
              min_intensity_out1);
  assign oBlue = Blue_out1;
  assign ce_out = clk_enable;
endmodule  // Subsystem
```

A.7 VGA Controller

```
module  VGA_Controller( //      Host Side
                        iRed,
                        iGreen,
                        iBlue,
                        oRequest,
                        //      VGA Side
                        oVGA_R,
                        oVGA_G,
                        oVGA_B,
                        oVGA_H_SYNC,
                        oVGA_V_SYNC,
                        oVGA_SYNC,
                        oVGA_BLANK,
                        //      Control Signal
                        iCLK,
                        iRST_N
                        );
//      Horizontal Parameter    ( Pixel )
parameter       H_SYNC_CYC      =       96;
parameter       H_SYNC_BACK     =       48;
parameter       H_SYNC_ACT      =       640;
parameter       H_SYNC_FRONT=   16;
parameter       H_SYNC_TOTAL=   800;

//      Virtical Parameter              ( Line )
parameter       V_SYNC_CYC      =       2;
parameter       V_SYNC_BACK     =       33;
parameter       V_SYNC_ACT      =       480;
parameter       V_SYNC_FRONT=   10;
parameter       V_SYNC_TOTAL=   525;
//      Start Offset
parameter       X_START         =       H_SYNC_CYC+H_SYNC_BACK;
parameter       Y_START         =       V_SYNC_CYC+V_SYNC_BACK;
//      Host Side
input           [9:0]   iRed;
input           [9:0]   iGreen;
input           [9:0]   iBlue;
output  reg             oRequest;
//      VGA Side
output  reg     [9:0]   oVGA_R;
output  reg     [9:0]   oVGA_G;
output  reg     [9:0]   oVGA_B;
output  reg             oVGA_H_SYNC;
output  reg             oVGA_V_SYNC;
output  reg             oVGA_SYNC;
output  reg             oVGA_BLANK;

wire            [9:0]   mVGA_R;
wire            [9:0]   mVGA_G;
wire            [9:0]   mVGA_B;
reg                     mVGA_H_SYNC;
reg                     mVGA_V_SYNC;
wire                    mVGA_SYNC;
wire                    mVGA_BLANK;

//      Control Signal
input                   iCLK;
input                   iRST_N;

//      Internal Registers and Wires
reg             [12:0]  H_Cont;
reg             [12:0]  V_Cont;
```

```
wire            [12:0]  v_mask;

assign v_mask = 13'd0 ;//iZOOM_MODE_SW ? 13'd0 : 13'd26;

////////////////////////////////////////////////////
assign  mVGA_BLANK      =       mVGA_H_SYNC & mVGA_V_SYNC;
assign  mVGA_SYNC       =       1'b0;

assign  mVGA_R  = (H_Cont>=X_START && H_Cont<X_START+H_SYNC_ACT &&
                V_Cont>=Y_START+v_mask && V_Cont<Y_START+V_SYNC_ACT )
                  ?       iRed    :       0;
assign  mVGA_G  = (H_Cont>=X_START && H_Cont<X_START+H_SYNC_ACT &&
                V_Cont>=Y_START+v_mask && V_Cont<Y_START+V_SYNC_ACT )
                  ?       iGreen  :       0;
assign  mVGA_B  = (H_Cont>=X_START && H_Cont<X_START+H_SYNC_ACT &&
                V_Cont>=Y_START+v_mask && V_Cont<Y_START+V_SYNC_ACT )
                  ?       iBlue   :       0;

always@(posedge iCLK or negedge iRST_N)
        begin
                if (!iRST_N)
                        begin
                                oVGA_R          <= 0;
                                oVGA_G          <= 0;
                                oVGA_B          <= 0;
                                oVGA_BLANK      <= 0;
                                oVGA_SYNC       <= 0;
                                oVGA_H_SYNC     <= 0;
                                oVGA_V_SYNC     <= 0;
                        end
                else
                        begin
                                oVGA_R          <= mVGA_R;
                                oVGA_G          <= mVGA_G;
                                oVGA_B          <= mVGA_B;
                                oVGA_BLANK      <= mVGA_BLANK;
                                oVGA_SYNC       <= mVGA_SYNC;
                                oVGA_H_SYNC     <= mVGA_H_SYNC;
                                oVGA_V_SYNC     <= mVGA_V_SYNC;
                        end
        end
//      Pixel LUT Address Generator
always@(posedge iCLK or negedge iRST_N)
begin
        if(!iRST_N)
        oRequest        <=      0;
        else
        begin
            if( H_Cont>=X_START-2 && H_Cont<X_START+H_SYNC_ACT-2 &&
                        V_Cont>=Y_START && V_Cont<Y_START+V_SYNC_ACT )
            oRequest    <=      1;
            else
            oRequest    <=      0;
        end
end
//      H_Sync Generator, Ref. 40 MHz Clock
always@(posedge iCLK or negedge iRST_N)
begin
        if(!iRST_N)
        begin
                H_Cont          <=      0;
                mVGA_H_SYNC     <=      0;
        end
        else
        begin
                //      H_Sync Counter
                if( H_Cont < H_SYNC_TOTAL )
                H_Cont  <=      H_Cont+1;
                else
                H_Cont  <=      0;
                //      H_Sync Generator
                if( H_Cont < H_SYNC_CYC )
                mVGA_H_SYNC     <=      0;
                else
                mVGA_H_SYNC     <=      1;
        end
end
//      V_Sync Generator, Ref. H_Sync
```

```
always@(posedge iCLK or negedge iRST_N)
begin
        if(!iRST_N)
        begin
                V_Cont          <=      0;
                mVGA_V_SYNC     <=      0;
        end
        else
        begin
                //      When H_Sync Re-start
                if(H_Cont==0)
                begin
                        //      V_Sync Counter
                        if( V_Cont < V_SYNC_TOTAL )
                        V_Cont  <=      V_Cont+1;
                        else
                        V_Cont  <=      0;
                        //      V_Sync Generator
                        if(     V_Cont < V_SYNC_CYC )
                        mVGA_V_SYNC     <=      0;
                        else
                        mVGA_V_SYNC     <=      1;
                end
        end
end
endmodule
```