HARDWARE ACCELERATOR FOR STAR CENTROIDING

by

Nazmus Sakib

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Electrical Engineering

Approved:

Charles M. Swenson, Ph.D. Major Professor	Jonathan Phillips, Ph.D. Committee Member
Jacob Gunther, Ph.D. Committee Member	Richard S. Inouye, Ph.D. Vice Provost for Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2020

ABSTRACT

Hardware Accelerator for Star Centroiding

by

Nazmus Sakib, Master of Science

Utah State University, 2020

Major Professor: Charles M. Swenson, Ph.D.
Department: Electrical and Computer Engineering

Position calculation or, in space which is called attitude determination is a process to describe the location of a spacecraft. Over the last couple of decades, star tracker based systems are being deployed in the spacecraft for this purpose. These star tracking algorithms take star image from a CMOS sensor camera as input and uses the information on hand to determine the attitude and orientation. Speed and accuracy is critical to such a real-time system. The purpose of this research is to speed up the star tracking process without reducing the accuracy of any star pattern recognition. As FPGA accelerators are faster for extracting parallelism than traditional CPU, a hardware accelerator is developed. The goal is to use this hardware to leverage the data parallelism of the star images to achieve faster attitude determination. This hardware is tested with synthesized star images which models the property of a star in a CMOS censor and the night sky background.

(81 pages)

PUBLIC ABSTRACT

Hardware Accelerator for Star Centroiding

Nazmus Sakib

Since the dawn of civilization mankind has gazed upon the night sky and looked for directions. In this modern age, the stars have proved to be a reliable reference point for navigation in space. From small-satellites for gathering weather data to inter-planetary missions like Cassini Orbiter, figuring out the location of a spacecraft has been done successfully by taking images of star filed visible from the spacecraft, processing and analyzing the image to find the stars and then finding out from a star catalog which part of the sky the patterns of the visible stars best matches. But a lot of calculation is involved in this process which means it takes up huge amount of power and time to complete. In this research work a hardware will be developed that will reduce the power and increase the speed of this process. The accuracy of this design will be tested under different environmental conditions, modeling the actual region of operation of this deisgn which is the space.

ACKNOWLEDGMENTS

I would like to thank Dr.Charles Swenson, Dr.Jonathan Phillips and Dr. Jacob Gunther for there support throughout this work. I would also like to thank my friends and roommates Md.Abdullah al Sarfin, Ferdous Parvej Jibon, Saju Saha, Munibun Billah Nirjhar, Abrar Zahin Abir and Rakin Muhammad Shadab for there support, motivation and encouragement.

Nazmus Sakib

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

1.1 Overview

Satellite attitude determination is a complicated process that requires position calculation and orientation measurements. This attitude determination system is more less comprised of some common steps: finding an approximate position relative to some frame of reference and tracking of future position [1]. Finding the initial attitude is done by looking at different fixed directions like Sun, Earth, Magnetic field of the earth etc using respective sensors like Sun senor, Gyroscope, Earth horizon sensor etc [2]. But these sensors have relatively less accuracy than star trackers when it comes to determining the attitude of the satellite [3].

Stars, or star fields as a reference to navigate has been used by humans since around 1200 B.C by the Polynesian mariners [4]. Modern technology have taken this process in space as the star locations does not change with respect to earth. After the first generation of star trackers [5] proved the merit of this technology many other space missions have deployed star tracker based attitude determination system [6], [7] and it has become a common and effective method.

1.2 Star camera and Star tracker

A star tracker is an attitude determination system that uses a camera to observe a star field, determines the position of the stars in the image and using a star catalog determines the attitude of the star tracker using pattern recognition technique. The camera can be Charge Injected Device [8], Active pixel Sensor [9] or Charge Coupled Device [10]. The range of the price, weight and power consumption can vary vastly but it has been shown that both star location and pattern recognition systems can be used with low cost cameras

and it's optical systems [11]. Such cameras have been developed for the commercial hand held mobile consumer market.

The first step of attitude determination is finding the star positions in an image of the star field which is referred to as centroiding. A single star is imaged across a block of camera pixels in a pattern known as the Point Spread Function. The star positions on the focal plane is found in a set of Cartesian coordinates tied to the Image plane which is then transformed to an initial set of coordinate based on the star catalog [1]. The centroiding and star identification process is completed in several steps to provide the locations in the Image plane coordinates.

1.3 Background

This section will briefly discuss the centroiding method and work of previous authors. Fig 1.1 shows the summary of the centroiding process.

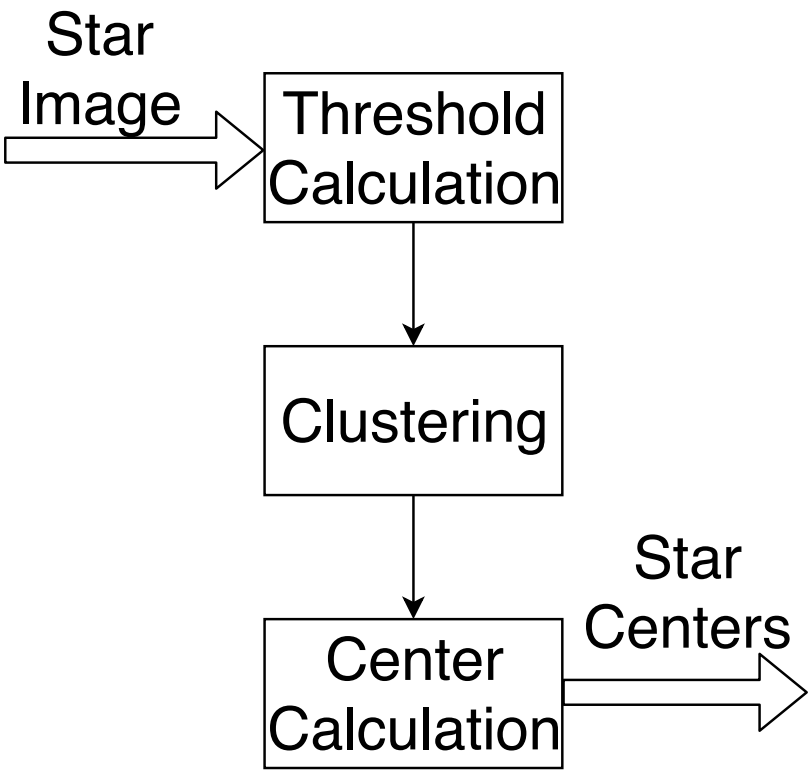

Fig. 1.1: Centroiding Flowchart

The threshold calculation and clustering are data processing steps that enables the center calculation step to perform on a small data set in the entire image.

1.3.1 Threshold

Threshold calculation is necessary as it makes the distinction between the background and the objects in an image. Several methods have been used by different authors. Liebe [1] proposed that detection threshold should be summation of average background pixel and five times the standard deviation. Zhou et al. [12] proposed a simpler method which takes the average pixel value of the bottom row of the image. Other works [13], [14] have proposed methods that are somewhat based on the same principal.

1.3.2 Clustering

Clustering is also known as Region of Interest (ROI) selection. This step is done by all authors [1], [12], [13], [14] almost similarly: when a pixel has been detected which has value greater than the detection threshold, search nearby pixels to find the brightest pixel of that region and then estimate a ROI around that pixel. Each cluster or ROI will represent an individual star in the image.

1.3.3 Centroiding

Stone et al. [20], Auer et al. [21] showed that two methods : moment calculation and functional fit are best for calculating the star centers. Among these two, Functional Fit type gives the more accurate centroiding but due to computational expense, Moment Analysis is chosen as the preferred centroid technique by [1], [13], [22] and many others.

1.4 Scope

The centroiding process is an image processing problem which involves differentiating between the star and the background or noise, selecting potential regions on the focal plane where the probability of finding a star is high and then applying some centroiding algorithm to find the star center with sub-pixel accuracy. As any image processing problem,

centroiding also takes up massive computational resource and power budget due to the fact that some steps requires brute force techniques to search or calculate certain parameters.

Microprocessor based systems have been used for star centroiding for satellites, Field Programmable Gate Array (FPGA) could also be used as it is potentially more power efficient and configurable and could be used as a dedicated computational hardware to accelerate centroiding process. Another advantage of using a FPGA is that it can handle parallel operations to speed up the process which is not suitable for general purpose CPU. Although GPU and multi-core CPU can perform parallel operations they are not typically effective for small satellites due to power and volume constraints.

1.5 Thesis Statement and Objectives

The objective of this thesis is to develop a Star Centroiding Algorithm that can be implemented on a FPGA. The objective is to both reduce the time for and overall power consumption for this part of the star imager attitude determination process.

1.5.1 Thesis Statement

The purpose of this research is to develop a Boolean solution for an inherently numerical centroiding problem and implement it as a digital hardware in a FPGA.

1.5.2 Objective

The Primary objectives of this research are:

1. Identify characteristics of a "star" in an image. This will require analyzing the distribution of light sources that builds up a "star" in the camera image.

2. Search for solutions that can be implemented in FPGA.

3. Make modifications or design choices of the solution that will be least computationally expensive.

4. Implement the developed algorithm in a development board containing a FPGA.

1.6 Outline

The thesis is organized as follows. Chapter 1 was the preliminary discussion on the star trackers, centroiding and benefits of FPGA. Chapter 2 will talk about framework and requirements of the system which will be developed. Framework is referred to the overall flow of this work, which starts from generating a star field synthetic image and ends in calculating the efficiency of the developed algorithm and implementation. Since this entire work is a sub-system of a larger system, the requirements that needs to be met to make the entire system properly function will be discussed in chapter 2. In chapter 3 the scope for an algorithm that is easy to implement in FPGA will be examined and the initial pre-processing methods will be discussed. The FPGA hardware design will be described in chapter 4, where the input and output of the developed hardware, the state machines that mirrors the logic of the algorithm developed in chapter 3 will be elaborated. Chapter 5 will discuss the results that will include the efficiency of the developed methods, the timing and power consumption. Finally chapter 6 will present a conclusive discussion of the work done in this thesis and suggestions will be made on further work.

CHAPTER 2

Framework

2.1 Overview

This chapter will present the steps of this research, framework of the design and the interfacing with FPGA. Fig 2.1 illustrates the steps of this work in the order which they were designed and tested.

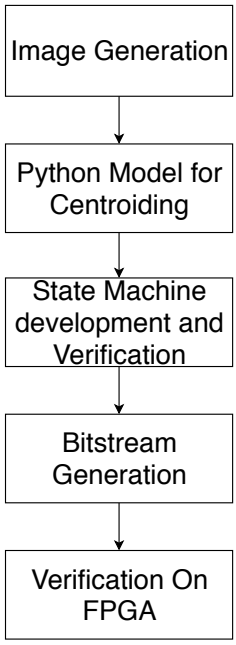

Fig. 2.1: Flowchart of the Framework

The first step of this research was to develop a synthetic image generator to generate star images with noise. The star image generation was done in Python. Randomly generated positions of stars were used to generate total 20 images with total 115 stars. Then the Python model of the different steps of centroiding algorithm was tested against the original position under different variations and noise levels. After that binary stream of

numbers were generated to be used as Verilog model. The Verilog model consists of the steps described in 1.1 and the results were compared with the Python model. At the end the generated bit stream which represents the pixels of generated images was fed into the SmartFusion-2 FPGA board to test the system in real time.

2.2 Requirements

In this section the requirements for the performance of different part of the work will be discussed. First we will discuss star images, which are input for the centroiding process. Then necessary centroiding accuracy based on previous works shall be discussed. At the end hardware module and interfacing with the hardware module will be examined.

2.2.1 Star Image

A star image, or night sky image is a combination of the dark night sky which is the background and bright stars along with different noises. To calculate the center position of stars the camera is typically de-focused slightly [1] so the photons from the star light is distributed across several pixels and stars can be described as Point Spread Function to model how the light is spread across multiple pixels to form a single star.

For the purpose of this work, night sky star images were generated using an approximation of this Point Spread Function. As in previous works [15], [16] the mathematical model of a star or the PSF was assumed to have a two-dimensional Gaussian distribution. Noises were added based on which noise sources are frequent and have greater effects on the accuracy of centroiding. The images should include multiple stars with a signal-to-noise ratio from 95.51 to 999.50. Details about the image generation will be discussed in the next chapter. Also due to the limitation of centroiding method developed in this work, the stars has to be at a distance of 15 pixels from each other. Fig 2.2 shows a sample night sky image with multiple stars.

2.2.2 Centroiding

The requirements for calculating the star centers depends on the camera parameters,

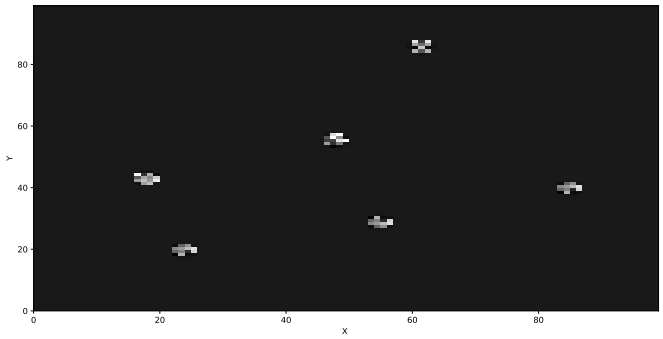

Fig. 2.2: Sample night sky

noise and algorithmic error that can cause the star centers to shift from the original value. The magnitude or radius of a star is a function of exposure time, lens aperture of the camera, number of photo-electrons exposed to camera and variations of these parameters will effect the centroid calculation. Since it has been shown that more than 3 pixel of distortion in the image focal plane can change the star to a entirely new star in the catalog [11], a goal of calculating centroid within 2 pixel of the true center was set, which is illustrated in fig 2.3. It should be noted that, this error range is independent of the PSF radius of the star. Star centroiding algorithm should be able to calculate the center position within this range for both small and large PSF radius.

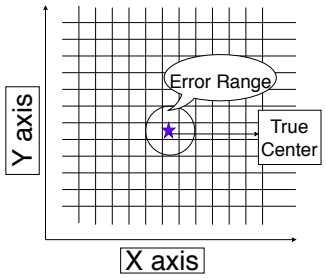

Fig. 2.3: Acceptable Position

2.2.3 Hardware and Interfacing

Hardware

The Microsemi SmartFusion2 FPGA was chosen for the hardware implementation of the centroiding method. This FPGA family has certain number of 4LUT modules, D flip-flops and RAM blocks as shown in table 2.1.

Table 2.1: Names and number of Resources in SmartFusion 2

Name of resource	Number
4LUT	27696
DFF	27696
RAM64x18	34
RAM1K18	31

The hardware design needs to use resources less than the available resources. Since SmartFusion2 has two different operating clock, minimum frequency of 1MHz was chosen for the operation. Although the maximum possible operating frequency of the developed hardware was calculated using SmartTime tool of Libero.

Interfacing

Since the generated images has to be tested in Verilog Simulator and in FPGA board, the data needed conversion to a format that is compatible with those platforms. These formats can be Hex, Binary, Intel-Hex etc. Since image generation was done in Python, a function was created to convert the data into 8 bit binary format. This was used as input for Verilog Simulator and FPGA board.

2.3 Conclusion

The overall goal was to generate realistic images with high signal-to-noise ratio, convert the data to a FPGA compatible format and perform centroiding operation on the hardware. Since this work was not part of a project with specific project, the goal was to investigate

the trade off between accuracy and speed and develop a hardware based on that.

CHAPTER 3

Methodology

In this chapter, the details and methods to meet the objectives and requirements of this thesis are shown. The image generation is described first and then the centroiding algorithm is discussed. The scope of this research will be presented and the fundamental difference of the proposed method and the tradition methods will be elaborated.

3.1 Image Generation

A good approximation of the Point Spread Function (PSF) for a star in the star field image is a bivariate Gaussian. The generalized probability function of a bivariate Gaussian is given by equation 3.1.

$$f(x,y) = \frac{1}{2\pi\sigma_x\sigma_y\sqrt{1-\rho^2}} \exp^{\frac{-1}{2(1-\rho^2)}\left[\frac{(x-\mu_x)^2}{\sigma_x^2}+\frac{(y-\mu_y)^2}{\sigma_y^2}-\frac{2\rho(x-\mu_x)(y-\mu_y)}{\sigma_x\sigma_y}\right]} \tag{3.1}$$

Where ρ is the correlation between x and y and $\sigma_x > 0$ and $\sigma_y > 0$. Generally star images have no correlation between x and y component, so $\rho = 0$. The density function describing a star can be written as [15]

$$I(x,y) = \frac{I_e}{2\pi\sigma_x\sigma_y} \exp^{-\frac{1}{2}\left[\frac{(x-\mu_x)^2}{\sigma_x^2}+\frac{(y-\mu_y)^2}{\sigma_y^2}\right]} \tag{3.2}$$

Where, $I_(x,y)$ is intensity of the pixel at coordinate (x,y), I_e is the photons captured by the sensor, μ_x and μ_y are the positions of "true" star centers, σ_x and σ_y are the radius of the PSF in x and y direction. The synthetic star image is generated in a matrix, a matrix of dimensions 100x100 was taken and erf function of Python was used to generated "stars" in the matrix. This erf function is used to find the integral of a normal distribution or in the case of discrete distribution it gives the Cumulative Distribution.

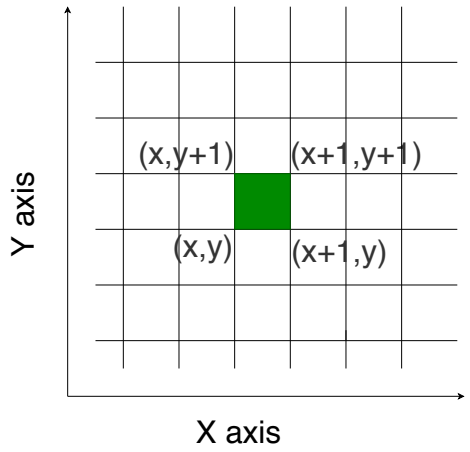

Fig. 3.1: Representation of a "Star" in a Matrix

As shown in 3.1 for a fixed mean, taking the sum of erf function at (x, y) and $(x+1, y+1)$ and subtracting it from the erf function calculated at $(x+1, y)$ and $(x, y+1)$ will give the value of pixel intensity at the index (x, y) for a star represented by that fixed mean. Fig 3.2 shows the contour plot of a sample night sky image without noise and digitization.

Note that, for the purpose of this work, in each image all stars had the same brightness but different images had different level of brightness associated to them.

3.1.1 Noise Sources

Other than the bright stars, different noise sources are present in a night sky image. In this section three noise sources will be discussed that were added to the star images. Although there are more noise sources [15], [17] which can be present in a sky image, the three sources added in this work are the dominant sources [1].

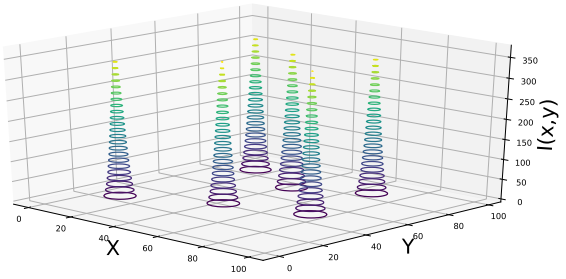

Fig. 3.2: 3-D PSF plot without noise

Dark Current

A pixel in a camera can be thought of as a well for holding electrons. The number of electrons are counted by the camera electronics during the readout after an exposure. Photons of light generate electrons in the wells of pixels across the detector. Even when no photon are applied to the imaging surface of the camera, small yet unavoidable current flows through the light sensors. Since the temperature is non zero the electrons in the sensors creates the pixel well. The amount of dark current is usually a constant for a fixed camera and environment so a constant signal dc was added to model this dark current [17].
Among all the photons arriving at the focal plane some are converted into electrons and the ratio of this converted photons to the total number of photons is called absolute Quantum Efficiency [19]. For simplicity this QE was assumed to be 100%.

Shot Noise

Light coming from the stars are photons, which are converted into electrons on the camera's detector surface. These discrete photons have random arrival time but a known mean. The arrival distribution is given by Poison Distribution [17] which measures an important noise process called Shot noise. The standard deviation of this distribution is given bye:

$$\sigma_p = \sqrt{(I(x,y))} \tag{3.3}$$

Where $I(x, y)$ is the pixel intensity at the image plane location (x, y). The noise source was modeled by adding a random normal distributed noise with σ_p as the standard deviation [18].

Read Noise

The electron charges generated by the photons has to be transferred across the surface of the sensor and to the read-out amplifier. This process results in a noise referred to as Read Noise . This noise is temporal and uncorrelated [15]. This noise was added to the image as a normal distribution with zero mean and constant standard deviation σ_{rn}.

Digitization

A digital camera has fixed number of bits representing the value of each pixel. A Python function "uint8" was used to convert the floating point image to 8 bit pixels, so pixel values in the image are between 0 to 255. Note that, the choice of 8 bit was arbitrary and the centroiding algorithm is not dependent on this.

Memory Layout

The goal of this research was to implement centroiding in FPGA and 1-D memory layout is simpler to implement in any hardware memory. Also, it reduces the computation. By row-major ordering, the image matrix (2-D) was converted to a 1-D array. From this point, the term *index* will be used as a generic reference which refers to linear index of the array. The term *rowindex* and *columnindex* are used to reference the *row*, *column* value in the 2-D image corresponding to the linear index.

Fig 3.3 shows the contour plot of the PSF functions with noise .

Signal to Noise Ration

A mathematical representation of the SNR is given by:

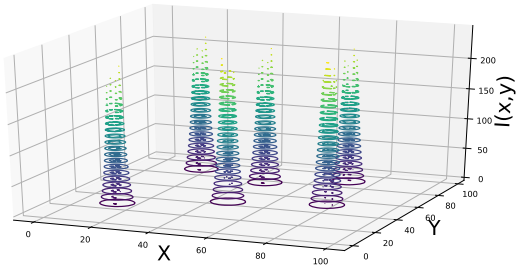

Fig. 3.3: 3-D PSF plot with noise

$$SNR = \frac{I}{\sqrt{((I + dc)^2 + R^2)}} \tag{3.4}$$

Where , I is the intensity of stars, dc is the Dark current and R is the mean of readout noise. From equation 3.4 we can see the shot noise component is the most dominating noise source. The shot noise depends on the intensity of the stars and range of the smallest to largest SNR given in the previous chapter was calculated from equation 3.4.

3.2 Threshold Calculation

The primary method for determining Regions of Interest is to use a threshold to calculate an approximate difference between dark background and bright stars. In order to locate the star center, first all the pixels of the image must be scanned to select a threshold that will be used to find a Region of Interest (ROI) which is an approximate location for a star. This ROI represents a region where pixels values are above a certain threshold. This threshold is a value that marks the difference between the background (dark night sky) and foreground (bright stars). Although as discussed before different methods have been introduced in [1], [13], [12], in this work a less computationally expensive method has been employed.

Histogram of Night Sky Image

A histogram consists of two components:

- Y axis : Frequency (number of occurrence)
- X axis : Bins(or classes)

Since the bit width of the pixels are 8 bit , the bins of our night sky image will range from 0 to 255. In a picture without noise we could have assumed that, values greater than zero are representative of a star in the image coordinate and threshold is zero. But due to the presence of noise, the threshold is above zero. The threshold calculation in this work is based on which pixel value appears maximum times. In Fig 3.4 we can see pixel value of *163* has appeared *9741* times among *10000* total pixels. Without noise this maximum appearing pixel value would have been *0*. Also, due to the digitization mentioned before, we do not see a continuous curve of frequency distribution.

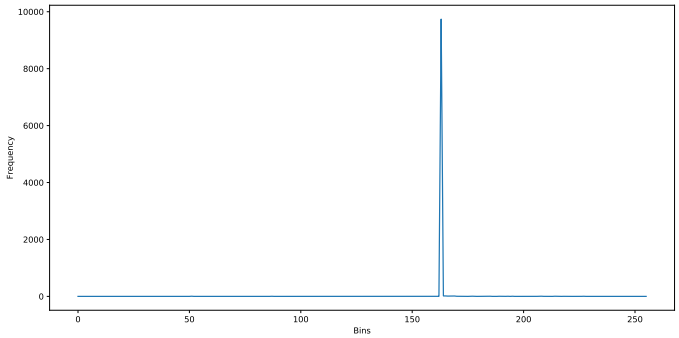

Fig. 3.4: Histogram of night sky image

This shows a clear distinction between the maximum frequency and all other values, which is expected since a night sky image must have greater number of dark pixels due to black sky. Note that, beside stars and background there can be different objects in the sky such as satellites, the Moon etc. Even in the presence of these objects which can be much brighter than the stars, this threshold calculation method will still be valid. The only time

it will give incorrect value is when the number of bright pixels becomes larger than the dark pixels, which is impossible for a night sky image.

3.3 Region of Interest Selection

A star in the image will be spread across several pixel as the camera optics is slightly de-focused on purpose [1]. So A Region of Interest is needed to limit the range of data that needs to be processed in order to calculate the center. Most, if not all proposed method [1], [13],12, [18] for the selection of ROI consists of scanning the image for values above threshold, then performing a local scan to find the approximate center considering the brightest pixel in that locality to be the center, then selection a window which can be 5x5,7x7,9x9 or 11x11 around the approximate center and finally applying a centroiding algorithm to find the sub-pixel accurate center co-ordinate of that star.

Although these methods are capable of finding centroid of a star accurately, scanning pixels near the first encountered pixel above the threshold takes up a lot of cycle and the centroiding algorithm cannot start working until it is done. In this research a faster way of selecting the ROI is proposed.

The algorithm 1 works as following:

- Initialize the image index counter *index*, buffer index counter *bi* and a counter *sc* to count the number of possible stars detected.
- Check if the pixel value is above the threshold.
- If the value of *sc* is zero then, put the index of the pixel in index buffer *indexb* and corresponding row and column value in the row buffer *rowb* and column buffer *columnb*. Take a $17x17$ window which is the ROI for that star having it's south-east corner at $column - 4$.
- If the value of *sc* is not zero then, check if the pixel is outside the ROI for all stars already detected. If it is then repeat the last step.
- Repeat the entire process for all pixels in the image.

Algorithm 1 Calculate ROI for each star

initial: $T_g \leftarrow threshold, bi \leftarrow 0, sc \leftarrow 0, N \leftarrow$ total pixels, $index \leftarrow 0$
while $index < N$ **do**
  $pixel \leftarrow Image[index]$
  if $pixel > T_g$ **then**
    if $sc == 0$ **then**
      $rowb[bi] \leftarrow row, columnb[bi] \leftarrow column - 2, index[bi] \leftarrow index, bi \leftarrow bi + 1$ take $column - 4$ as south-east corner,take a $17x17$ window which is $ROI_(sc)$ for star sc
      $sc \leftarrow sc + 1$
    else
      if $pixel \not\subset$ ROI of all stars already calculated **then**
        $rowb[bi] \leftarrow row, columnb[bi] \leftarrow column - 2, indexb[bi] \leftarrow index, bi \leftarrow bi + 1$ take $column - 4$ as south-east corner,take a $17x17$ window which is $ROI_(sc)$ for star sc
        $sc \leftarrow sc + 1$
      end if
    end if
  end if
  $index \leftarrow index + 1$
end while

As can be seen from the algorithm, the row and column value of the index is also stored, which will be used to calculate the centroid in the next section. Fig 3.5 shows the Region of Interest selection process.

The scanning process is in row first order, so bottom-left corner of the image is the starting point. When first pixel where $pixel > T_g$ is encountered, a border of 17x17 window with the south-east corner of $column-4$ was taken, which is the region where the calculation of finding the center of that particular star will be done. After this, the scanning process keeps on checking all the other pixels in the image, and it will only consider another pixel as a new potential star region only when it has made sure that pixel is outside of ROI of all star regions previously calculated.

3.3.1 Advantage

Traditionally, the step of finding ROI in the image coordinate and applying centroiding algorithm is done in two steps, in a sequential manner. But due to the mutually exclusive

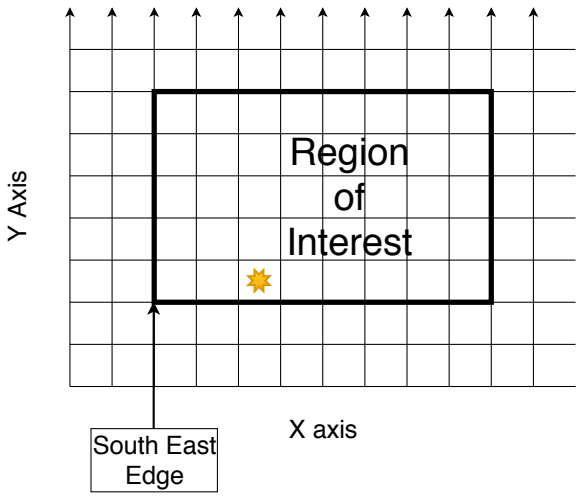

Fig. 3.5: ROI selection

nature of these two steps and the parallelism which can be leveraged in a FPGA, the centroiding process can start as soon as this scanning process selects a pixel as the south-east edge of ROI. The major advantage of the proposed method is that it does not waste clock cycles while waiting for the scan to be completed on the whole image to find all potential star locations. Rather, after locating the first ROI, the centroiding process starts working on this ROI, meanwhile the scanning process keeps on looking for more ROI in the image plane and stores them in a buffer.

3.3.2 Disadvantage

Since this method selects ROI window as a constant, stars with different radius will have the same window. This might adversely affect the centroid calculation, as selecting an ROI appropriate to the intensity distribution of that particular star in image plane is crucial to determine the centroid in the Cartesian coordinate of the image. Also two stars must be at a Euclidean distance of 24.04 pixels between each other which limits the total number of stars that can be processed. But this choice of constant value was made because no real camera images were used. In the case of a real project involving real camera, the window size can be changed considering the lens aperture area and sensitivity to star lights. In fact, to use this work for practical purposes of real time star centroid detection, calibration

based on camera parameters is highly recommended.

3.4 Centroiding

Each star in an image plane, which is a Cartesian Coordinate System, is spread across several pixels. Centroiding is finding the center of a star. This is the most important part of the image processing step of the attitude determination problem of a spacecraft. Centroid is essentially finding the row index and column index of center of the star which will translate to (x, y) co-ordinate in the Cartesian Coordinate.

Although Moment Analysis, or Center of Moment is highly sensitive to background noise [21] this is still a better choice considering the high computational value of 2-D Least Square Fit or any other function fitting technique. In this work a modified version of this Center of Moment method named Elementary is proposed. As described in Image Generation section a star has the shape of Gaussian PSF and the center can be found by equations 3.5.

$$Sum_t = \sum_{x=ROI_f}^{ROI_l} \sum_{y=ROI_f}^{ROI_l} Pixel_(x, y) \tag{3.5a}$$

$$X_t = \sum_{x=ROI_f}^{ROI_l} \sum_{y=ROI_f}^{ROI_l} \frac{x * Pixels(x, y)}{Sum_t} \tag{3.5b}$$

$$Y_t = \sum_{x=ROI_f}^{ROI_l} \sum_{y=ROI_f}^{ROI_l} \frac{y * Pixels(x, y)}{Sum_t} \tag{3.5c}$$

Where, Sum_t is the sum of all the pixels in the ROI, (X_t, Y_t) are calculated centers and ROI_f, ROI_l are the first and last index of the ROI .

3.4.1 Scope

The equations 3.5 can be divided into two parts:

- The first part where the sum of all the pixels are calculated.
- The second part where the marginal distribution in x and y direction is calculated.

- The third part where the division is taking place.

Since the goal of this research was to implement this entire process, from threshold to centroiding in an FPGA. We shall now look into the disadvantages incurred by 3.5.

The first inconvenience arises from the data dependency in the equations. The sum Sum_t and marginal distributions $x * \text{Pixels}(x, y)$, $y * \text{Pixels}(x, y)$ has to be calculated and only after that the division can take place.

Secondly, the threshold and ROI selection methods proposed in this work are computationally cheap but since no noise reduction technique has been applied, the ROI edge calculated might be far from the actual star center in case of a large PSF radius or too close to center if the radius is too small.

In order to overcome these challenges, a modified Moment analysis was proposed that will leverage the flexibility of FPGA and use information from local maxima(or minima) to find the center of the star.

3.4.2 Elementary

From the previous sections, we know that a star is represented as a Gaussian PSF. Any Bivariate Gaussian has marginal distribution in x and y direction. Also, since photons come at discrete packages,the distribution is discrete in nature. If we omit the weights we can get the marginal distributions from equations 3.5 [20] :

$$\text{Marginal}(x) = \sum_{y=ROI_f}^{ROI_l} \text{Pixels}(x, y)$$
$$\text{Marginal}(y) = \sum_{x=ROI_f}^{ROI_l} \text{Pixels}(x, y)$$

Fig 3.6 shows a marginal distribution of a star generated in the Image Generation section. As it shows, due to sensitivity to noise the marginal is far from being standard normal distribution.

What we see in the above image is index vs Intensity plot where the index can be row or column depending on which marginal it is (x or y). The objective is to find the mean

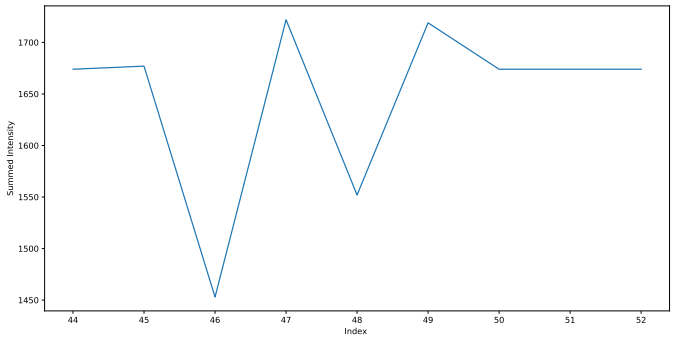

Fig. 3.6: Sample Marginal Distribution

(μ_m) of this distribution. To do that, a simple method called Elementary is implemented in this work. Although Zhou et.al [12] have proposed zero-crossing method to finding centroid in FPGA they have assumed the star center spread in four pixels in a square region, which might not be the case always. This method takes into account the characteristics or conditions of the mean of a discrete normal distribution:

1. incoming and outgoing slope of the μ_m should have same value or there difference should be close to zero (it should be a local maxima or minima).

2. μ_m should have the highest concentration of pixel intensity (the mean of a random variable is the "center location").

But the challenge was that noise and PSF radius distorts the marginal distribution so those two conditions might not indicate to the same index. Since the integer part of the centroid is nothing but the row or column index, the algorithm developed simply looks into the indexes, search for *candidates* and assign the integer and decimal part of the center based on the information gathered from the *candidates*. A *candidate* is calculated using a sliding window of 3 indexes : *center*, $left$, *right*. If both $left$ and *right* indexes have intensities greater than the *center* or both have intensities less than the *center* then the *center* index is a candidate (basically it is a local maxima or minima).
The *candidate* which meets the first characteristic of the mean of a discrete normal distribution is defined as perfect or balanced candidate and referred to as *pi*. The *candidate* which

meets the second characteristic of the mean of a discrete normal distribution is defined as the max *candidate* and referred to as mi.

Fig 2 shows the process for calculating the centroid for the x marginal. The y marginal is analogous and was calculated in parallel in FPGA. Note that the integer and decimal part was calculated separately. This is due to the fact that only 4 bits were assigned to te decimal in the Verilog implementation which will be described in the next chapter.
The basic logic behind Elementary is that there will be more than one maxima or minima in the marginal distribution and it is expected that the $perfect$ *candidate* should be less then average and is somewhere in the middle of the distribution. Summary of the process is described below:

- Choose ROI south-east edge from the buffer and assume a 9×9 window starting from this index.
- Fetch and add pixel values corresponding to that x index, which will be the Intensity.
- Repeat this up to index $x + 8$.
- Calculate which indexes are *candidates*.
- Calculate the *candidate* (pi) with minimum difference in slopes.
- Calculate the *candidate* (mi) with maximum intensity (sum of the intensity of *center*, $left$, $right$).
- Calculate the average value (avg) of all *candidates*. Note that the *candidates* are all integers and in FPGA or in digital logic design, taking an average of two values using right shift by 1 place will give the $floor$ of average of the two integers.
- Based on pi, mi, avg assign the integer and decimal of the center.

    1. If pi is less than or equal to avg we can assume that there were candidates after the perfect *candidate*. So, if the mi is equal to the pi we can conclude that both conditions of the mean of a discrete normal distribution has met and avg will

Algorithm 2 Centroid Calculation

initial: $ic \leftarrow 0, pc \leftarrow 0, counter \leftarrow 0, balanced \leftarrow 0, max \leftarrow 0, avg \leftarrow 0, mi \leftarrow 0, pi \leftarrow 0$
while $counter < bi \wedge bi \neq 0$ **do**
  $i \leftarrow index[counter], x \leftarrow column[counter], DMI[x] \leftarrow 0$
  repeat
    while $pc < 9$ **do**
      $DMI[x] \leftarrow DMI[x] + image[pc], pc \leftarrow pc + 1$
    end while
    if $ic >= 2 \wedge ic <= 8$ **then**
      if $(DMI[x-1] > DMI[x] \wedge DMI[x+1] > DMI[x]) \vee (DMI[x-1] < DMI[x] \wedge DMI[x+1] < DMI[x])$ **then**
        $iscandiate[x] \leftarrow True, dif[x] \leftarrow |DMI[x-1] - DMI[x+1]|, SM[x] \leftarrow DMI[x-1] + DMI[x+1] + DMI[x]$
      end if
      if $iscandidate[x-1] = True$ **then**
        if $SM[x-1] > max$ **then**
          $max \leftarrow SM[x-1], mi \leftarrow x-1$
        end if
        if $dif[x-1] < balanced$ **then**
          $balanced \leftarrow dif[x-1], pi \leftarrow x-1$
        end if
      end if
    end if
    $x \leftarrow x+1, ic \leftarrow ic+1$
  until $ic < 9$
  $avg \leftarrow$ Average of all candidate index
  if $pi \leq avg$ **then**
    if $mi == pi$ **then**
      $X_t(integer) \leftarrow avg, X_t(decimal) \leftarrow 0.5$
    else
      $X_t(integer) \leftarrow (avg + pi) >> 1, X_t(decimal) \leftarrow 0.93$
    end if
  end if
  if $pi > avg$ **then**
    if $mi = pi$ **then**
      $X_t(integer) \leftarrow avg, X_t(decimal) \leftarrow 0.0$
    else
      $X_t(integer) \leftarrow (avg + pi) >> 1, X_t(decimal) \leftarrow 0.93$
    end if
  end if
  $counter \leftarrow counter + 1$
end while

be chosen as the integer part of the centroid and 0.50 will be decimal part, else the average of *pi* and *avg* will be the integer part of the center and 0.93 will be decimal part.

2. If *pi* greater than *avg*, it means perfect *candidate* comes after the average. In other words the perfect *candidate* is located at the end of the ROI which is troubling since noise will distort the edges of the ROI badly. So relative values of *pi* and *mi* will be evaluated. If *mi* equals to *pi* then the integer calculation will be same as previous step and the decimal will be set to zero, else the average of *pi* and *avg* will be the integer part of the center and 0.93 will be decimal part.

3.4.3 Advantage

Since Elementary gathers information for center calculation while doing the summation for each index, it does not have to wait for the division which is necessary for equations 3.5 of Moment Analysis, so this is easy to implement in the FPGA. Elementary also evaluates the nature of the discrete marginal distribution instead of simple weighted average like Moment Analysis which makes it more robust to different PSF radius,noise level and photon count.

3.4.4 Disadvantage

Elementary was developed keeping in mind that variations might change the intensity value and the shape of PSF. As it is shown in results section, when the marginal distributions are not distorted the Moment method works better than Elementary, although remaining within the error tolerance range described in the Framework section.

CHAPTER 4

Implemenation

4.1 Overview

As the centroiding is an image processing task, the traditional CPU based system which takes large number of clock cycles to fetch values from memory is not suitable for parallel processing for fast results. The primary objective of this research was to implement the centroiding in other computing paradigms rather than microprocessor based systems. Suitable choices were GPU which is a specialized hardware that leverages the data level parallelism inherent in any image processing problem. But interfacing with the GPU will require a complicated on-board electrical system which will not be cost effective in a Cube or Small satellite. Next suitable option was using an FPGA.

4.1.1 Advantage of FPGA

An FPGA is a programmable hardware which is used for specialized tasks, specially where parallel processing is necessary or preferred. The parallelism can be control, thread and data level parallelism. As some algorithms have specific traits that a microprocessor fails to leverage, FPGA can be used for fast prototyping and implementation. Since it is a reconfigurable architecture, it can be modified for different problem sets which makes it more suitable then ASIC based implementation.

4.1.2 Hardware acceleration

When a hardware is specifically built for the purpose of running a special class of software which can be run faster on that hardware rather than general purpose CPU then this process is called hardware acceleration. For this research FPGA was chosen as the platform and centering algorithm described in Methodology section was implemented in FPGA.

SmartFusion2 FPGA of Microsemi Corp was used in this work. One of the special features it has is the low power consumption and Radiation Hardened modules which is suitable for space applications.

4.1.3 Conclusion

In this chapter, state diagrams of the logic implemented on the FPGA will be described. First the block diagram of parallel process that was built will be described, then the state diagrams for threshold calculation, ROI edge selection and centroiding will be described.

4.2 Design

As described in Framework section, the generated images from Implementation section was converted to 8-bit, 1-D memory and Verilog command *readmemb* was used to transfer the image to the block RAM of the FPGA. The overall description of the design is shown in Fig 4.1.

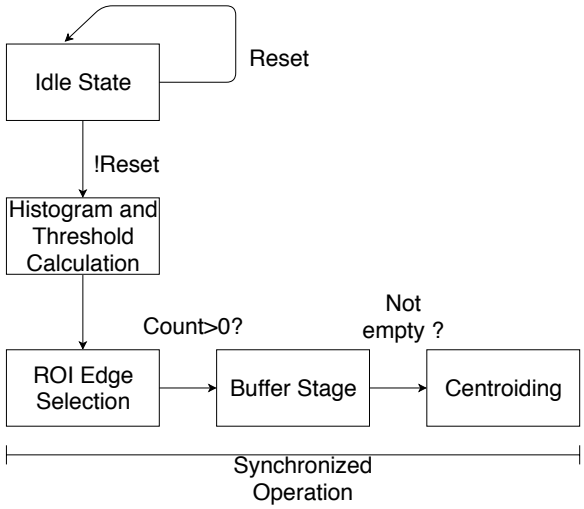

Fig. 4.1: Flowchart of Hardware Operation

As can be seen, the ROI edge selection and Centroiding modules are synchronized via the buffer and when the buffer is not empty, the both these modules work in parallel. This is the thread level parallelism that was not possible in a general purpose CPU based system.

Note that specialized multi-core CPU can achieve this parallel operation but that is not suitable for a real time system.

4.2.1 Threshold

As discussed in the Implementation chapter, the threshold selection is done by creating a histogram of the image and finding out which pixel value, out of 0 to 255, appears maximum time in the image. Fig 4.2 shows the state diagram of threshold calculation.

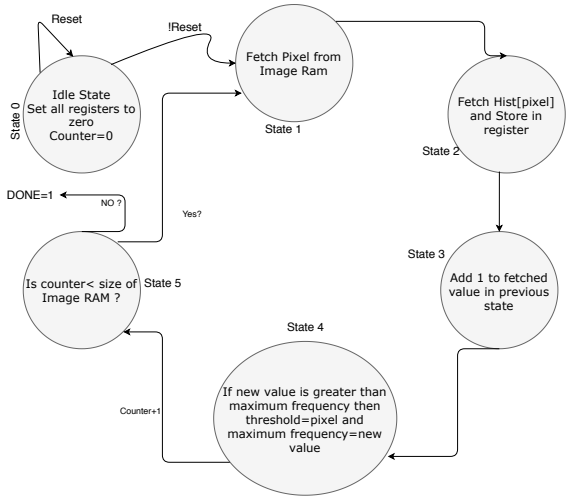

Fig. 4.2: State Diagram for Threshold Calculation

Here, the size of Image Ram refers to the number of pixels in the image which is hard coded in the Verilog module. State 0 is the Idle state. When the reset is high, a pixel value referenced by the register *counter* is fetched from the RAM. In state 2, the histogram value corresponding to that pixel is accessed from the Hist RAM block, and in the subsequent state 1 is added to this old histogram value. In state 4 the comparison between the current maximum frequency and the new value of calculated in the last state takes place. Then 1 is added to the counter. If the *counter* value is greater then the number of pixels we set the register DONE to high and the next stage, which is the ROI edge selection starts. If

counter is less than the Image RAM then we go back to state 1.

4.3 Region of Interest

This section describes the Verilog implementation of the Region of Interest Edge Selection Process. Note that, in the state machine *index*, *row* and *column* registers are mentioned which refer to the linear (1-D) index of the pixel position and the corresponding row and column value in the 2-D matrix, which was the original image. Fig 4.3 shows the state diagram for this process.

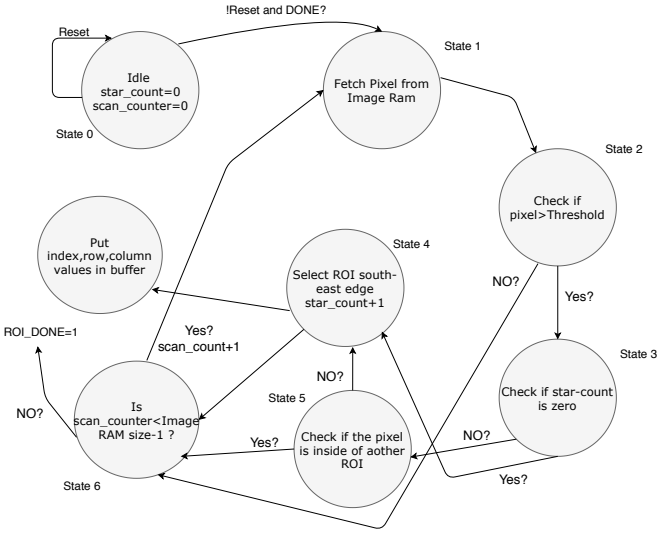

Fig. 4.3: State Diagram for Region of Interest

In the Idle state it waits until the reset is high and the the register $DONE$ is high, which indicates that the previous process of threshold calculation has completed and the hardware can start running the next module. State 1 fetches the pixel value referenced by the scan counter register and the next state checks if the value is above the threshold. If it is above threshold then it goes to state 3 to check if the value of star count register is zero or not. If the value is below threshold then it goes to state 6, checks if the value of scan count is less than the num of pixels and repeats the process by moving to state 1 again.

Meanwhile in state 3, if the number of star count is zero then it goes to state 4, select that pixel as a ROI south-east edge of a new star and put the values in buffer. While in the case of star count not being zero, the process moves to state 5 to check if that pixel belong to an ROI of another star. If not then it goes to state 4 and then goes to state 6 and repeats the process.

4.3.1 Center Calculation

Up until now what was done has been some pre-processing, just to select a specific region in the entire image where a potential star center is suspected to be located. The ROI Edge Selection has given an approximate location where the two marginal distribution might have there tails. So we can now look into the *candidates* described in previous chapter and try to find the mean of the PSF also known as the center.

Before going to the state diagram the pipelined operation will be demonstrated for the x marginal (y marginal is analogous and implemented in parallel). Fig 4.4 shows the process of calculating the center, where operations are separated in stages and only first 5 stages are shown.

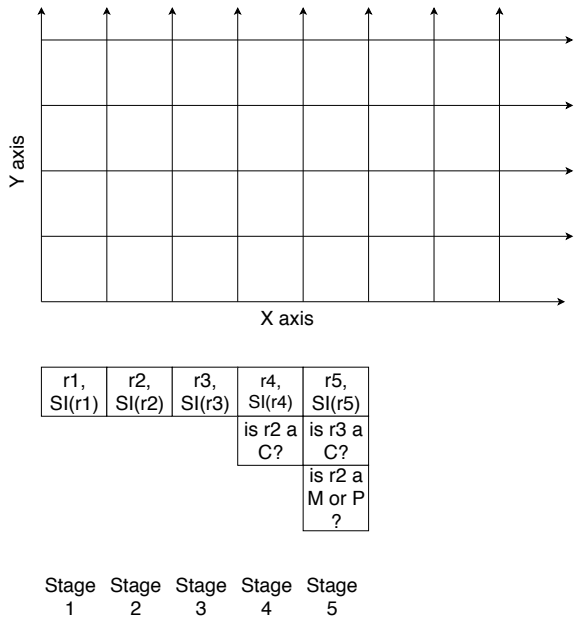

Fig. 4.4: Pipelined Operation

Where,

- r1, r2, r3, r4, r5 are column indexes.
- SI is the sum of intensities for that column.
- C is *candidate*, M is *max* and P is *balanced* indexes.

First, all the row pixels for column r1 is summed and stored in SI(r1). This process repeats itself two consecutive times and then at step 4 we can calculate if (r1) is a *candidate* or not in the fourth step and in the fifth step we determine if r1 is *max* or $perfect$. Notice that at step 5 the summing process for column r5, the *candidate* checking process for r3 and the *max* and $perfect$ checking process for r2 are taking place at the same time. Although not shown here but the same process is being run in parallel for rows. Fig 4.5 shows the state diagram for the center finding process.

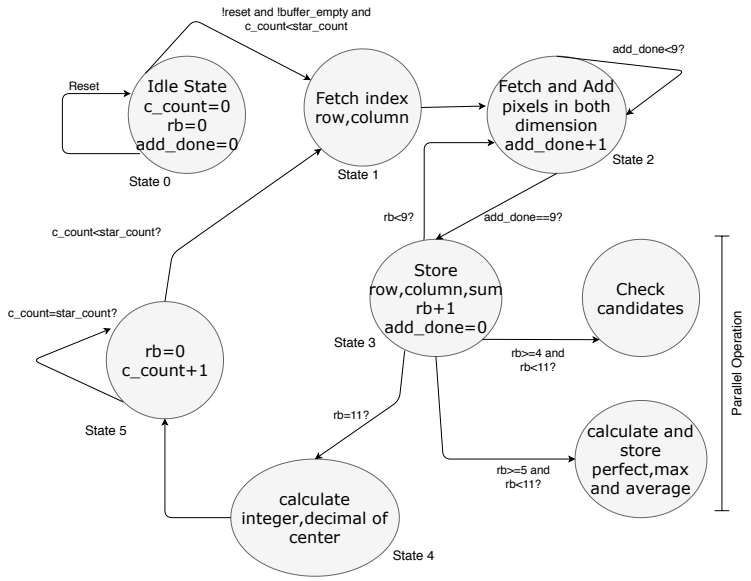

Fig. 4.5: State Diagram for center calculation

The process is as follows: if the buffer is not empty then row, column and linear index are fetched. At state 2 the process of summing the marginal intensities takes place where 9 values corresponding to a particular row and column are added. State 3 makes sure the

process is repeated 9 times, to cover the 9×9 window. Also the parallel operations takes place depending on the value of register rb. The comparison of max, $perfect$ and avg takes place in state 4. Then it goes to state 5 where the value of register c_counter is increased by 1. Then we look into whether the c_counter is less than the star_count found in the ROI EDGE Selection process previously discussed and if no new potential star regions has been detected then the process waits in state 5. It should be noted that in state 2, the fetch and add process also takes place in parallel. That is, if in n cycle pixel value P is fetched then the already fetched value in n_{-1} cycle P_{-1} will be added to the register that holds the summation.

CHAPTER 5

Results

5.1 Overview

The centroiding accuracy depends on the threshold selection, which is subjected to the level of noise present in the image. Accuracy also depends on the photo-electron count and radius of a star. This chapter will describe the effect of these parameters on the accuracy of Center of Moment and Elementary. Then the speed and power consumption will be discussed.

5.1.1 Variation Parameters

Total 20 images were generated of 100×100 dimension as described in section 3. Each image has total 10 variations and different number of stars in them. The results of COM were calculated in a PC and the results from Elementary were calculated in Modelsim and validated in the FPGA board. Names and Description of these variations are listed in 5.1. In the following section, effect of these variations on centroid calculation will be discussed.

Table 5.1: Names and number of variations

Name of Parameter	Number of Variations
PSF Radius	4
Photo-Electron count	3
Noise level	3

The effects will be discussed for both Center of Method (COM) and Elementary.

5.2 Accuracy

The accuracy of each star is determined by taking the Euclidean distance of the true centroid known from the generation of images and the calculated centroid from COM and

Elementary. The distance was calculated using equation 5.1.

$$\text{E.D} = \sqrt{(x_r - x_c)^2 + (y_r - y_c)^2} \tag{5.1}$$

Where (x_r, y_r) are the actual centroid and (x_c, y_c) are the calculated centroid. Then, the average of the E.D for all stars in one image is calculated. The process is repeated for all images to take the statistical average.

5.2.1 Effect of varying PSF radius

The Point Spread Function(psf) of a star represents the distribution of the 2-D Gaussian Distribution in generated images. The radius of this PSF effects the accuracy of the centroiding because the standard deviation of the distribution becomes larger with increasing radius and therefore makes it difficult to find the mean which is the center of the distribution. Four radius values were taken as shown in table 5.2

Table 5.2: Parameter and Value

Name of Parameter	Value
PSF1	$\sigma_x = 0.5, \sigma_y = 0.6$
PSF2	$\sigma_x = 1.1, \sigma_y = 1.0$
PSF3	$\sigma_x = 1.4, \sigma_y = 1.5$
PSF4	$\sigma_x = 2.0, \sigma_y = 1.9$

where (σ_x, σ_y) are the radius in x and y direction. These accuracy was tested in with all four PSF values, with fixed noise level and photoelectron count. The effect of different PSF radius on the accuracy of the centroid determination is shown in Fig 5.1

We can see Elementary shows more accuracy than COM, with the exception of PSF2. Since the accuracy of COM depends on the selection of ROI size and position, in the case of PSF1 which has PSF radius of 1 meaning the 2D distribution of the PSF is standard, COM has the perfect results. In other cases where the radius differs from 1 Elementary finds better results as it calculates the centroid based on the 1D segments of the 2D distribution.

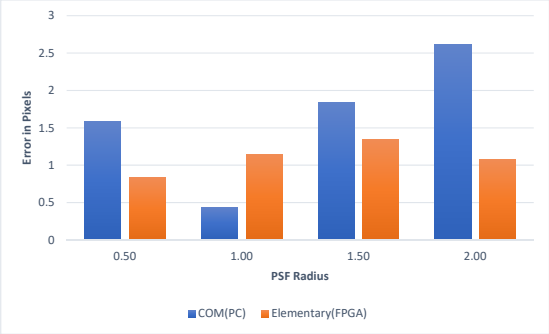

Fig. 5.1: Effect of PSF radius on Accuracy

5.2.2 Effect of varying Photon count

The number of photons that are exposed to star camera and converted to electron reveals the magnitude of the star as well as the smoothness of the distribution of PSF. The different numbers of photo-electrons chosen for this simulation were taken from [1]. Fig 5.2 shows the effect of different photo-electron on centroiding accuracy.

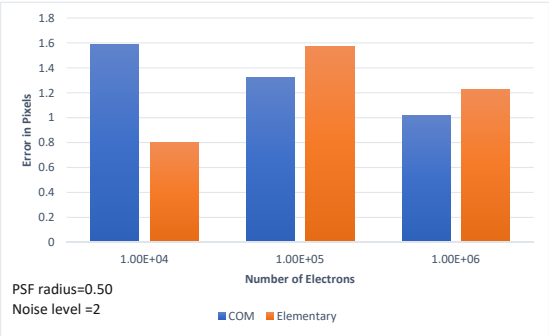

Fig. 5.2: Effect of Number of Photo-Electron on Accuracy

We can see Elementary is not performing better than COM with increasing photon count. Since the pixel depth of the generated images are 8bit, large number of electrons mean there is little decrease in illumination from the center of the star to the edge of

the star. In other words the standard deviation is very low and therefore, COM performs better since it is taking an average instead of locating the local maxima or minima of the 1-D normal distribution like Elementary.

5.2.3 Effect of varying Noise

Different level of noise degrade the image, mainly blurring the difference between dark background and bright stars. Here two algorithms will be tested under three different noise level [18]. Table 5.3 shows the three noise noise levels used to determine the effect of noise.

Table 5.3: Noise Parameters and Value

Noise Level(NL)	Parameter	value
NL1	Full Well Capacity(FWC)	100^3
	$(dc,\sigma_{\mathrm{rn}})$	2000
NL2	Full Well Capacity(FWC)	100^3
	$(dc,\sigma_{\mathrm{rn}})$	5000
NL3	Full Well Capacity(FWC)	100^3
	$(dc,\sigma_{\mathrm{rn}})$	10000

Fig 5.3 shows centroiding accuracy under different noise level. Here the effectiveness of Elementary is more visible.

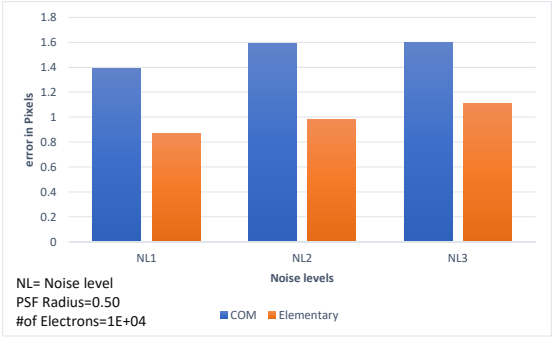

Fig. 5.3: Effect of Noise on Accuracy

As noise increases, the threshold check and ROI selection fails to capture the tail of the Gaussian PSF representing the star. Since Elementary looks for local peaks values to find the center of the star, it is more resistant to noise levels than COM.

5.3 Speed

The time it takes to complete the processing of one image and calculating all the centroid depends on the dimension (size) of the image and the number of star it contains. Let,

- Image dimension=n×m , where n and m are number of row and column respectively.
- Number of stars = s.

A fixed window of 9×9 was selected, so the maximum number of stars that can be processed with desired precision are $\frac{n \times n}{9 \times 9}$, which is much less than the size of the image, so the time and hence the speed is majorly depended on the size.

The time taken by PC was calculated using the time library of Python and run on a Intel(R) Core (TM) i5-7200U CPU with operating frequency of 2.71GHz. As FPGA does not have a fixed operating frequency, SmartTime which is the timing editor of Libero SOC for Microsemi FPGA was used to calculate the Maximum operating frequency. Time taken by FPGA was calculated by multiplying the clock ticks for the completion of one image with the time period of the Maximum operating frequency. Figure 5.4 shows the natural logarithm of time taken by the PC and the FPGA, with same image dimensions and number of stars.

We can see that FPGA is order of magnitude faster than the PC. Note that, time for the PC was calculated from a python implementation and an implementation with C/C++ language might have been faster, but still it would be order of magnitude slower than the FPGA implementation.

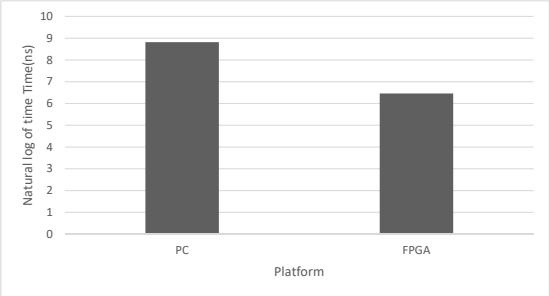

Fig. 5.4: Speed on PC and FPGA

5.4 Power

Libero SoC has a power analyzing tool called SmartPower. It was used to evaluate the power consumption of the design. Figure 5.5 shows the static power consumption of the hardware. Notice that maximum power is consumed by the core, which is the computational segment of the designed hardware. The memory banks where the image is stored consume a very little amount of power.

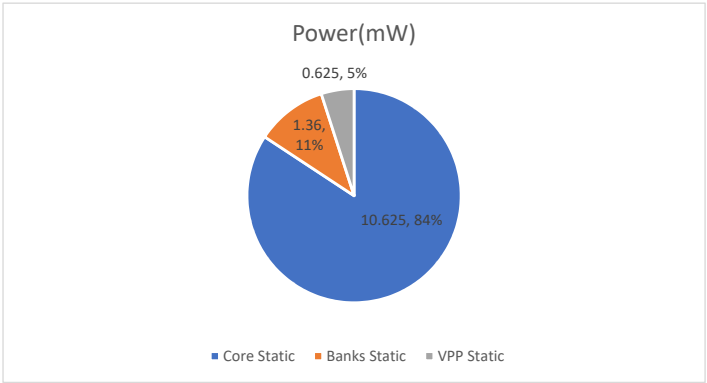

Fig. 5.5: Static Power

Note that as the image size grows, the computational blocks of the hardware remains the same, only the size of memory blocks and certain registers will increase. So the power consumption will not change dramatically for images with larger dimensions. Also, the dynamic power consumption depend on the amount of time the FPGA will run, which depend on the size (dimension) of the image, number of stars present in the image and the clock frequency. Although no analysis was done on dynamic power consumption it is safe to say that dynamic power consumption is a necessary cost that cannot be avoided in the real-time system.

Table 5.4 shows the main resource usage from the Synopsis synthesis tool in Libero SoC. The number of D flip-flops are much less than LUT combinational logic, so dynamic power consumption will also not increase rapidly with images of larger dimension.

Table 5.4: Resource Usage

Typer	Used	Total	Percentage
4LUT	1455	27695	5.25
DFF	688	27695	2.48
RAM64x18	4	34	11.76
RAM1K18	1	31	3.23

CHAPTER 6

Conclusion

6.1 Conclusion

The purpose of this thesis was met because a suitable centroiding technique for FPGA was developed and tested. The results were satisfactory as the goal was not to cross the error range of 2 pixels. The speedup over CPU and the low power consumption was demonstrated. But the method comes with one major weakness: the ROI edge selection takes a large window for each star which limits the number of stars that can be processed accurately and also two very close stars will not be detected as separate stars. But these problems can be solved easily if the hardware is calibrated with the camera parameters which is usually done in real-time systems. This calibration will tell the hardware an approximation of the maximum and average magnitude of star it can detect. The problem of two extremely close stars can also be overcome by finding a value of the possible PSF spread possible in that camera and also required for the specific mission.

There is also a fact that, since the speedup is really high and most satellites have on-board microprocessors, this design could be used as a primary centroiding process while the microprocessor takes the output of the work done in this work and calculate more accurate centroid.

6.2 Future Work

It is suggested for future work to test this system in real time, or test it with image data taken from a satellite. It will be crucial since testing the system with motion blurred images will reveal more strengths and weakness of the design developed in this work. Also, steps should be taken to figure out if the pattern recognition part of the attitude determination can be integrated in the FPGA as well.

REFERENCES

[1] C. C. Liebe, "Accuracy performance of star trackers - a tutorial," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 38, no. 2, pp. 587–599, April 2002.

[2] J. R. Wertz, *Spacecraft attitude determination and control*. Springer Science & Business Media, 2012, vol. 73.

[3] O. L. de Weck, "Attitude determination and control (adcs)," *Lecture notes from*, vol. 16, 2001.

[4] W. McGrath, "Some notes on the navigation of 1985 trans-indian ocean canoe voyage," *The Journal of Navigation*, vol. 41, no. 2, pp. 174–185, 1988.

[5] A. R. Eisenman, C. C. Liebe, and J. L. Joergensen, "New generation of autonomous star trackers," in *Sensors, Systems, and Next-Generation Satellites*, vol. 3221. International Society for Optics and Photonics, 1997, pp. 524–535.

[6] R. Lindberg, D. Lumb, R. den Hartog, P. Gondoin, N. Rando, and M. Fridlund, "Preliminary results on the internal assessment study of the esa cosmic vision mission plato," in *Instruments, Methods, and Missions for Astrobiology XI*, vol. 7097. International Society for Optics and Photonics, 2008, p. 70970F.

[7] E. F. Young, R. Mellon, J. W. Percival, K. P. Jaehnig, J. Fox, T. Lachenmeier, B. Oglevie, and M. Bingenheimer, "Sub-arcsecond performance of the st5000 star tracker on a balloon-borne platform," in *2012 IEEE Aerospace Conference*. IEEE, 2012, pp. 1–7.

[8] J. Carbone, S. Czebiniak, and J. Zarnowski, "New cid detectors/cameras for use in ionizing radiation environments," *Transactions of the American Nuclear Society*, vol. 73, p. 460, 1995.

[9] O. Yadid-Pecht, B. Pain, C. Staller, C. Clark, and E. Fossum, "Cmos active pixel sensor star tracker with regional electronic shutter," *IEEE Journal of Solid-State Circuits*, vol. 32, no. 2, pp. 285–288, 1997.

[10] G. Yang, C. Sun, C. Wrigley, O. Yadid-Pecht, and B. Pain, "A smart cmos imager with on-chip high speed windowed controiding capability," 1999.

[11] S. P. Brätt, "Analysis of star identification algorithms due to uncompensated spatial distortion," 2013.

[12] F. Zhou, J. Zhao, T. Ye, and L. Chen, "Fast star centroid extraction algorithm with sub-pixel accuracy based on fpga," *Journal of Real-Time Image Processing*, vol. 12, no. 3, pp. 613–622, 2016.

[13] M. V. Arbabmir, S. M. Mohammadi, S. Salahshour, and F. Somayehee, "Improving night sky star image processing algorithm for star sensors," *JOSA A*, vol. 31, no. 4, pp. 794–801, 2014.

[14] D. Lang, D. W. Hogg, K. Mierle, M. Blanton, and S. Roweis, "Astrometry. net: Blind astrometric calibration of arbitrary astronomical images," *The astronomical journal*, vol. 139, no. 5, p. 1782, 2010.

[15] B. R. Hancock, R. C. Stirbl, T. J. Cunningham, B. Pain, C. J. Wrigley, and P. G. Ringold, "Cmos active pixel sensor specific performance effects on star tracker/imager position accuracy," in *Functional Integration of Opto-Electro-Mechanical Devices and Systems*, vol. 4284. International Society for Optics and Photonics, 2001, pp. 43–53.

[16] K. A. Winick, "Cramér–rao lower bounds on the performance of charge-coupled-device optical position estimators," *JOSA A*, vol. 3, no. 11, pp. 1809–1815, 1986.

[17] J. Vandersteen, "Observation and estimation for space applications. hardware design and software algorithms," 2012.

[18] T. Delabie, J. De Schutter, and B. Vandenbussche, "An accurate and efficient gaussian fit centroiding algorithm for star trackers," *The Journal of the Astronautical Sciences*, vol. 61, no. 1, pp. 60–84, 2014.

[19] C. Buil, "Ccd astronomy. construction and use of an astronomical ccd camera," *Richmond: Willmann-Bell, 1991*, 1991.

[20] R. C. Stone, "A comparison of digital centering algorithms," *The Astronomical Journal*, vol. 97, pp. 1227–1237, 1989.

[21] L. Auer and W. Van Altena, "Digital image centering. ii," *The Astronomical Journal*, vol. 83, pp. 531–537, 1978.

[22] C. R. McBryde and E. G. Lightsey, "A star tracker design for cubesats," in *2012 IEEE Aerospace Conference*, March 2012, pp. 1–14.

APPENDICES

APPENDIX A

Verliog Code

```
module aa
(
input clk, input reset,
output wire  led1,output wire  led2
);
reg [6:0] hs  /* synthesis syn_noprune=1 */ ;
reg [9:0] max  /* synthesis syn_noprune=1 */ ;
reg [9:0] Max  /* synthesis syn_noprune=1 */ ;
reg [9:0] avg_max /* synthesis syn_noprune=1 */ ;
reg [9:0] south /* synthesis syn_noprune=1 */ ;
reg [9:0] east /* synthesis syn_noprune=1 */ ;
reg [9:0] west /* synthesis syn_noprune=1 */ ;
reg [9:0] temp  /* synthesis syn_noprune=1 */ ;
reg [9:0] t2 /* synthesis syn_noprune=1 */ ;
reg [31:0] count  /* synthesis syn_noprune=1 */ ;
reg [31:0] ecount  /* synthesis syn_noprune=1 */ ;
reg [31:0] wcount  /* synthesis syn_noprune=1 */ ;
reg [31:0] ncount  /* synthesis syn_noprune=1 */ ;
reg [31:0] newh  /* synthesis syn_noprune=1 */ ;
reg [31:0] sec  /* synthesis syn_noprune=1 */ ;
reg [31:0] oldh    /* synthesis syn_noprune=1 */ ;
reg [31:0] fr   /* synthesis syn_noprune=1 */ ;
reg [31:0] h[255:0]   /* synthesis syn_ramstyle=lsram */;
reg [7:0] img[9999:0]  /* synthesis syn_ramstyle="lsram" */;
localparam [7:0]Col=100; localparam [7:0]row=100;
localparam [11:0]row2=200;
localparam [11:0]row3=300;
localparam [11:0]row4=400;
localparam [11:0]row5=500;
localparam [11:0]row6=600;
localparam [11:0]row7=700;
localparam [11:0]row8=800;
localparam [15:0]row9=9900;
localparam [15:0]Col9=9900;
localparam [31:0]half_counter=5000;
reg [20:0] scan  /* synthesis syn_noprune=1 */ ;
reg [9:0] cc  /* synthesis syn_noprune=1 */ ;
reg [9:0] rc  /* synthesis syn_noprune=1 */ ;
reg [31:0] starc  /* synthesis syn_noprune=1 */ ;
reg [7:0] mstarc  /* synthesis syn_noprune=1 */ ;
reg [31:0] fifo_count  /* synthesis syn_noprune=1 */ ;
reg [5:0] fifo_state  /* synthesis syn_noprune=1 */ ;
reg [9:0] ff1  /* synthesis syn_noprune=1 */ ;
reg [9:0] ff2  /* synthesis syn_noprune=1 */ ;
reg [20:0] ff3  /* synthesis syn_noprune=1 */ ;
reg [9:0] tff1  /* synthesis syn_noprune=1 */ ;
reg [9:0] tff2  /* synthesis syn_noprune=1 */ ;
reg [20:0] tff3  /* synthesis syn_noprune=1 */ ;
reg [15:0] ss  /* synthesis syn_noprune=1 */ ;
reg [7:0] px  /* synthesis syn_noprune=1 */ ;
reg [7:0] px2  /* synthesis syn_noprune=1 */ ;
```

```
reg[2:0] done /* synthesis syn_noprune=1 */ ;
reg[9:0] xbuf[15:0] /* synthesis syn_noprune=1 */ ;
reg[9:0] ybuf[15:0] /* synthesis syn_noprune=1 */ ;
reg[20:0] ibuf[15:0] /* synthesis syn_noprune=1 */ ;
reg[9:0] r /* synthesis syn_noprune=1 */ ;
reg[9:0] c /* synthesis syn_noprune=1 */ ;
reg[20:0] i /* synthesis syn_noprune=1 */ ;
reg[9:0] tr /* synthesis syn_noprune=1 */ ;
reg[9:0] tc /* synthesis syn_noprune=1 */ ;
reg[20:0] ti /* synthesis syn_noprune=1 */ ;
reg[9:0] rx /* synthesis syn_noprune=1 */ ;
reg[9:0] ry  /* synthesis syn_noprune=1 */;
reg[20:0] ix /* synthesis syn_noprune=1 */ ;
reg[20:0] iy /* synthesis syn_noprune=1 */ ;
reg[1:0] bfd /* synthesis syn_noprune=1 */ ;
reg[7:0] buff_count /* synthesis syn_noprune=1 */ ;
reg[15:0] bs /* synthesis syn_noprune=1 */ ;
reg rolling /* synthesis syn_noprune=1 */ ;
reg [31:0]  DN /* synthesis syn_noprune=1 */ ;
 reg [31:0]  f_DN /* synthesis syn_noprune=1 */ ;
reg [31:0]  DNy /* synthesis syn_noprune=1 */ ;
reg [31:0]  tDN /* synthesis syn_noprune=1 */ ;
//reg [31:0]  thDN /* synthesis syn_noprune=1 */ ;
//reg [31:0]  fDN /* synthesis syn_noprune=1 */ ;
reg [63:0]  Smx /* synthesis syn_noprune=1 */ ;
//reg [31:0]  Smxt /* synthesis syn_noprune=1 */ ;
reg [63:0]  Smy /* synthesis syn_noprune=1 */ ;
 //reg [31:0]  Smyt /* synthesis syn_noprune=1 */ ;

reg [63:0]  Smy1 /* synthesis syn_noprune=1 */ ;
reg [63:0]  Smx1 /* synthesis syn_noprune=1 */ ;
reg [63:0]  Smy2 /* synthesis syn_noprune=1 */ ;
reg [63:0]  Smx2 /* synthesis syn_noprune=1 */ ;
reg [63:0]  Smy3 /* synthesis syn_noprune=1 */ ;
reg [63:0]  Smx3 /* synthesis syn_noprune=1 */ ;
reg [63:0]  Smy4 /* synthesis syn_noprune=1 */ ;
reg [63:0]  Smx4 /* synthesis syn_noprune=1 */ ;
reg [63:0]  Smy5 /* synthesis syn_noprune=1 */ ;
reg [63:0]  Smx5 /* synthesis syn_noprune=1 */ ;
reg [63:0]  Smy6 /* synthesis syn_noprune=1 */ ;
reg [63:0]  Smx6 /* synthesis syn_noprune=1 */ ;
reg [63:0]  Smy7 /* synthesis syn_noprune=1 */ ;
reg [63:0]  Smx7 /* synthesis syn_noprune=1 */ ;
reg [63:0]  Smy8 /* synthesis syn_noprune=1 */ ;
reg [63:0]  Smx8 /* synthesis syn_noprune=1 */ ;
reg [63:0]  Smy9 /* synthesis syn_noprune=1 */ ;
reg [63:0]  Smx9 /* synthesis syn_noprune=1 */ ;
reg [31:0]         sx /* synthesis syn_noprune=1 */ ;
 //reg [31:0]         Sx /* synthesis syn_noprune=1 */ ;
reg [31:0]         sy /* synthesis syn_noprune=1 */ ;
//reg [31:0]         Sy /* synthesis syn_noprune=1 */ ;
//reg [31:0]        delx12 /* synthesis syn_noprune=1 */ ;
//reg [31:0] delx23 /* synthesis syn_noprune=1 */ ;
//reg [31:0] delx34 /* synthesis syn_noprune=1 */ ;
//reg [31:0] delx45 /* synthesis syn_noprune=1 */ ;
//reg [31:0] delx56 /* synthesis syn_noprune=1 */ ;
//reg [31:0] delx67 /* synthesis syn_noprune=1 */ ;
//reg [31:0] delx78 /* synthesis syn_noprune=1 */ ;
//reg [31:0] delx89 /* synthesis syn_noprune=1 */ ;

//reg [31:0]        dely12 /* synthesis syn_noprune=1 */ ;
//reg [31:0] dely23 /* synthesis syn_noprune=1 */ ;
```

```
//reg [31:0] dely34 /* synthesis syn_noprune=1 */ ;
//reg [31:0] dely45 /* synthesis syn_noprune=1 */ ;
//reg [31:0] dely56 /* synthesis syn_noprune=1 */ ;
//reg [31:0] dely67 /* synthesis syn_noprune=1 */ ;
//reg [31:0] dely78 /* synthesis syn_noprune=1 */ ;
//reg [31:0] dely89 /* synthesis syn_noprune=1 */ ;

reg [31:0]       dely31 /* synthesis syn_noprune=1 */ ;
reg [31:0] dely42 /* synthesis syn_noprune=1 */ ;
reg [31:0] dely53 /* synthesis syn_noprune=1 */ ;
reg [31:0] dely64 /* synthesis syn_noprune=1 */ ;
reg [31:0] dely75 /* synthesis syn_noprune=1 */ ;
reg [31:0] dely86 /* synthesis syn_noprune=1 */ ;
reg [31:0] dely97 /* synthesis syn_noprune=1 */ ;

reg [31:0]       delx31 /* synthesis syn_noprune=1 */ ;
reg [31:0] delx42 /* synthesis syn_noprune=1 */ ;
reg [31:0] delx53 /* synthesis syn_noprune=1 */ ;
reg [31:0] delx64 /* synthesis syn_noprune=1 */ ;
reg [31:0] delx75 /* synthesis syn_noprune=1 */ ;
reg [31:0] delx86 /* synthesis syn_noprune=1 */ ;
reg [31:0] delx97 /* synthesis syn_noprune=1 */ ;

reg [31:0]        Sx1 /* synthesis syn_noprune=1 */ ;
reg [31:0]        Sx2 /* synthesis syn_noprune=1 */ ;
reg [31:0]        Sx3 /* synthesis syn_noprune=1 */ ;
reg [31:0]        Sx4 /* synthesis syn_noprune=1 */ ;
reg [31:0]        Sx5 /* synthesis syn_noprune=1 */ ;
reg [31:0]        Sx6 /* synthesis syn_noprune=1 */ ;
reg [31:0]        Sx7 /* synthesis syn_noprune=1 */ ;
reg [31:0]        Sx8 /* synthesis syn_noprune=1 */ ;
reg [31:0]        Sx9 /* synthesis syn_noprune=1 */ ;

reg [31:0]        Sy1 /* synthesis syn_noprune=1 */ ;
reg [31:0]        Sy2 /* synthesis syn_noprune=1 */ ;
reg [31:0]        Sy3 /* synthesis syn_noprune=1 */ ;
reg [31:0]        Sy4 /* synthesis syn_noprune=1 */ ;
reg [31:0]        Sy5 /* synthesis syn_noprune=1 */ ;
reg [31:0]        Sy6 /* synthesis syn_noprune=1 */ ;
reg [31:0]        Sy7 /* synthesis syn_noprune=1 */ ;
reg [31:0]        Sy8 /* synthesis syn_noprune=1 */ ;
reg [31:0]        Sy9 /* synthesis syn_noprune=1 */ ;

reg [31:0]        Sx123 /* synthesis syn_noprune=1 */ ;
reg [31:0]        Sx234 /* synthesis syn_noprune=1 */ ;
reg [31:0]        Sx345 /* synthesis syn_noprune=1 */ ;
reg [31:0]        Sx456 /* synthesis syn_noprune=1 */ ;
reg [31:0]        Sx567 /* synthesis syn_noprune=1 */ ;
reg [31:0]        Sx678 /* synthesis syn_noprune=1 */ ;
reg [31:0]        Sx789 /* synthesis syn_noprune=1 */ ;

reg [31:0]        Sy123 /* synthesis syn_noprune=1 */ ;
reg [31:0]        Sy234 /* synthesis syn_noprune=1 */ ;
reg [31:0]        Sy345 /* synthesis syn_noprune=1 */ ;
reg [31:0]        Sy456 /* synthesis syn_noprune=1 */ ;
reg [31:0]        Sy567 /* synthesis syn_noprune=1 */ ;
reg [31:0]        Sy678 /* synthesis syn_noprune=1 */ ;
reg [31:0]        Sy789 /* synthesis syn_noprune=1 */ ;



reg [31:0]        sx1 /* synthesis syn_noprune=1 */ ;
```

```
reg [31:0]        sx2 /* synthesis syn_noprune=1 */ ;
reg [31:0]        sx3 /* synthesis syn_noprune=1 */ ;
reg [31:0]        sx4 /* synthesis syn_noprune=1 */ ;
reg [31:0]        sx5 /* synthesis syn_noprune=1 */ ;
reg [31:0]        sx6 /* synthesis syn_noprune=1 */ ;
reg [31:0]        sx7 /* synthesis syn_noprune=1 */ ;
reg [31:0]        sx8 /* synthesis syn_noprune=1 */ ;
reg [31:0]        sx9 /* synthesis syn_noprune=1 */ ;

reg [31:0]        sy1 /* synthesis syn_noprune=1 */ ;
reg [31:0]        sy2 /* synthesis syn_noprune=1 */ ;
reg [31:0]        sy3 /* synthesis syn_noprune=1 */ ;
reg [31:0]        sy4 /* synthesis syn_noprune=1 */ ;
reg [31:0]        sy5 /* synthesis syn_noprune=1 */ ;
reg [31:0]        sy6 /* synthesis syn_noprune=1 */ ;
reg [31:0]        sy7 /* synthesis syn_noprune=1 */ ;
reg [31:0]        sy8 /* synthesis syn_noprune=1 */ ;
reg [31:0]        sy9 /* synthesis syn_noprune=1 */ ;

reg [31:0]        delx /* synthesis syn_noprune=1 */ ;
reg [31:0]        tdelx /* synthesis syn_noprune=1 */ ;
reg [31:0]        tdelx2 /* synthesis syn_noprune=1 */ ;
reg [31:0]        dely /* synthesis syn_noprune=1 */ ;
 reg [31:0]        tdely /* synthesis syn_noprune=1 */ ;
  reg [31:0]        tdely2 /* synthesis syn_noprune=1 */ ;
reg [1:0]        up /* synthesis syn_noprune=1 */ ;
reg [1:0]        down /* synthesis syn_noprune=1 */ ;
reg [1:0]        upx /* synthesis syn_noprune=1 */ ;
reg [1:0]        downx /* synthesis syn_noprune=1 */ ;
//reg [9:0]        xcan1 /* synthesis syn_noprune=1 */ ;
//reg [9:0]        xcan2 /* synthesis syn_noprune=1 */ ;
//reg [9:0]        xcan3 /* synthesis syn_noprune=1 */ ;
//reg [9:0]        ycan1 /* synthesis syn_noprune=1 */ ;
//reg [9:0]        ycan2 /* synthesis syn_noprune=1 */ ;
//reg [9:0]        ycan3 /* synthesis syn_noprune=1 */ ;
//reg [9:0]        xcan4 /* synthesis syn_noprune=1 */ ;
//reg [9:0]        ycan4 /* synthesis syn_noprune=1 */ ;

reg [31:0]        half  /* synthesis syn_noprune=1 */ ;
reg [31:0]        one_fourth /* synthesis syn_noprune=1 */ ;
reg [31:0]        one_eighth /* synthesis syn_noprune=1 */ ;
reg [31:0]        thr_fourth /* synthesis syn_noprune=1 */ ;
reg [31:0]        tone_fourth /* synthesis syn_noprune=1 */ ;
reg [31:0]        tpDN /* synthesis syn_noprune=1 */ ;
//reg [31:0]        tone_eighth /* synthesis syn_noprune=1 */ ;
//reg [31:0]        tthr_fourth /* synthesis syn_noprune=1 */ ;

//reg [31:0]        yhalf  /* synthesis syn_noprune=1 */ ;
//reg [31:0]        yone_fourth /* synthesis syn_noprune=1 */ ;
//reg [31:0]        yone_eighth /* synthesis syn_noprune=1 */ ;
//reg [31:0]        ythr_fourth /* synthesis syn_noprune=1 */ ;
//reg [31:0]        ytone_fourth /* synthesis syn_noprune=1 */ ;
//reg [31:0]        ytone_eighth /* synthesis syn_noprune=1 */ ;
//reg [31:0]        ytthr_fourth /* synthesis syn_noprune=1 */ ;


reg [1:0]        posx /* synthesis syn_noprune=1 */ ;
reg [1:0]        posy /* synthesis syn_noprune=1 */ ;
reg [3:0]        decx /* synthesis syn_noprune=1 */ ;
reg [3:0]        decy /* synthesis syn_noprune=1 */ ;
reg [19:0]        a /* synthesis syn_noprune=1 */ ;
reg [19:0]        b /* synthesis syn_noprune=1 */ ;
```

```
reg [31:0]        remx /* synthesis syn_noprune=1 */ ;
reg [31:0]        remy /* synthesis syn_noprune=1 */ ;
reg [3:0]        fdecx /* synthesis syn_noprune=1 */ ;
reg [3:0]        fdecy /* synthesis syn_noprune=1 */ ;
reg [19:0]        fa /* synthesis syn_noprune=1 */ ;
reg [19:0]        fb /* synthesis syn_noprune=1 */ ;
reg[7:0] t1 /* synthesis syn_noprune=1 */;
reg[7:0] T2 /* synthesis syn_noprune=1 */;
reg[7:0] t3 /* synthesis syn_noprune=1 */;
reg[7:0] t4 /* synthesis syn_noprune=1 */;
reg[7:0] t5 /* synthesis syn_noprune=1 */;
reg[7:0] t6 /* synthesis syn_noprune=1 */;
reg[7:0] t7 /* synthesis syn_noprune=1 */;
reg[7:0] t8 /* synthesis syn_noprune=1 */;
reg[7:0] t9 /* synthesis syn_noprune=1 */;
reg[7:0] s1 /* synthesis syn_noprune=1 */;
reg[7:0] s2 /* synthesis syn_noprune=1 */;
reg[7:0] s3 /* synthesis syn_noprune=1 */;
reg[7:0] s4 /* synthesis syn_noprune=1 */;
reg[7:0] s5 /* synthesis syn_noprune=1 */;
reg[7:0] s6 /* synthesis syn_noprune=1 */;
reg[7:0] s7 /* synthesis syn_noprune=1 */;
reg[7:0] s8 /* synthesis syn_noprune=1 */;
reg[7:0] s9 /* synthesis syn_noprune=1 */;

reg[7:0] ft1 /* synthesis syn_noprune=1 */;
reg[7:0] fT2 /* synthesis syn_noprune=1 */;
reg[7:0] ft3 /* synthesis syn_noprune=1 */;
reg[7:0] ft4 /* synthesis syn_noprune=1 */;
reg[7:0] ft5 /* synthesis syn_noprune=1 */;
reg[7:0] ft6 /* synthesis syn_noprune=1 */;
reg[7:0] ft7 /* synthesis syn_noprune=1 */;
reg[7:0] ft8 /* synthesis syn_noprune=1 */;
reg[7:0] ft9 /* synthesis syn_noprune=1 */;
reg[7:0] fs1 /* synthesis syn_noprune=1 */;
reg[7:0] fs2 /* synthesis syn_noprune=1 */;
reg[7:0] fs3 /* synthesis syn_noprune=1 */;
reg[7:0] fs4 /* synthesis syn_noprune=1 */;
reg[7:0] fs5 /* synthesis syn_noprune=1 */;
reg[7:0] fs6 /* synthesis syn_noprune=1 */;
reg[7:0] fs7 /* synthesis syn_noprune=1 */;
reg[7:0] fs8 /* synthesis syn_noprune=1 */;
reg[7:0] fs9 /* synthesis syn_noprune=1 */;



reg[6:0] roi_cou /* synthesis syn_noprune=1 */;
reg[4:0] rb /* synthesis syn_noprune=1 */;
reg[20:0] difx1 /* synthesis syn_noprune=1 */;
reg[20:0]  difx2 /* synthesis syn_noprune=1 */;
reg[20:0]  difx3 /* synthesis syn_noprune=1 */;
reg[20:0]  dify1 /* synthesis syn_noprune=1 */;
reg[20:0]  dify2 /* synthesis syn_noprune=1 */;
reg[20:0]  dify3 /* synthesis syn_noprune=1 */;
//reg[20:0]  difab /* synthesis syn_noprune=1 */;
//reg[20:0]  ydifab /* synthesis syn_noprune=1 */;
reg [63:0]  aDN /* synthesis syn_noprune=1 */ ;
reg [63:0]  bDN /* synthesis syn_noprune=1 */ ;

reg[4:0] addc /* synthesis syn_noprune=1 */;
//reg [31:0] fhalf_ind  /* synthesis syn_noprune=1 */ ;
reg [31:0] x_max /* synthesis syn_noprune=1 */ ;
```

```
 reg [31:0] tx_max /* synthesis syn_noprune=1 */ ;
  reg [31:0] tx_max2 /* synthesis syn_noprune=1 */ ;
//reg [31:0] shalf_ind  /* synthesis syn_noprune=1 */ ;
reg [31:0] shalf_max  /* synthesis syn_noprune=1 */ ;
//reg [31:0] yfhalf_ind  /* synthesis syn_noprune=1 */ ;
reg [31:0] y_max /* synthesis syn_noprune=1 */ ;
 reg [31:0] ty_max /* synthesis syn_noprune=1 */ ;
  reg [31:0] ty_max2 /* synthesis syn_noprune=1 */ ;
//reg [31:0] yshalf_ind  /* synthesis syn_noprune=1 */ ;
reg [31:0] yshalf_max  /* synthesis syn_noprune=1 */ ;


reg[31:0] en /* synthesis syn_noprune=1 */;
reg[9:0] beg /* synthesis syn_noprune=1 */;
reg[31:0] conflict /* synthesis syn_noprune=1 */;

reg[9:0] rowf  /* synthesis syn_noprune=1 */ ;
reg[9:0] rowl  /* synthesis syn_noprune=1 */ ;
reg[9:0] colf  /* synthesis syn_noprune=1 */ ;
reg[9:0] coll  /* synthesis syn_noprune=1 */ ;

reg[9:0] Cxbuf[15:0] /* synthesis syn_noprune=1 */ ;
reg[9:0] Cybuf[15:0] /* synthesis syn_noprune=1 */ ;
reg[31:0] Cxc  /* synthesis syn_noprune=1 */ ;
reg[31:0] Cyc  /* synthesis syn_noprune=1 */ ;

reg[1:0] isy2can /* synthesis syn_noprune=1 */ ;
reg[1:0] isy3can /* synthesis syn_noprune=1 */ ;
reg[1:0] isy4can /* synthesis syn_noprune=1 */ ;
reg[1:0] isy5can /* synthesis syn_noprune=1 */ ;
reg[1:0] isy6can /* synthesis syn_noprune=1 */ ;
reg[1:0] isy7can /* synthesis syn_noprune=1 */ ;
reg[1:0] isy8can /* synthesis syn_noprune=1 */ ;
reg[1:0] isx2can /* synthesis syn_noprune=1 */ ;
reg[1:0] isx3can /* synthesis syn_noprune=1 */ ;
reg[1:0] isx4can /* synthesis syn_noprune=1 */ ;
reg[1:0] isx5can /* synthesis syn_noprune=1 */ ;
reg[1:0] isx6can /* synthesis syn_noprune=1 */ ;
reg[1:0] isx7can /* synthesis syn_noprune=1 */ ;
reg[1:0] isx8can /* synthesis syn_noprune=1 */ ;

reg[1:0] doney /* synthesis syn_noprune=1 */ ;reg[1:0] donex /* synthesis
    syn_noprune=1 */ ;
reg[15:0] atleast1 /* synthesis syn_noprune=1 */ ;

reg[3:0] same  /* synthesis syn_noprune=1 */ ;
reg[31:0]clk_count /* synthesis syn_noprune=1 */ ;
reg[1:0] change /* synthesis syn_noprune=1 */ ;
reg[1:0] xchange /* synthesis syn_noprune=1 */ ;reg[1:0] ychange /* synthesis
    syn_noprune=1 */ ;

reg [63:0] hy1 /* synthesis syn_noprune=1 */ ;
reg [63:0] hy2 /* synthesis syn_noprune=1 */ ;
reg [63:0] hy3 /* synthesis syn_noprune=1 */ ;
reg [63:0] hy4 /* synthesis syn_noprune=1 */ ;
reg [63:0] hy5 /* synthesis syn_noprune=1 */ ;
reg [63:0] hy6 /* synthesis syn_noprune=1 */ ;
reg [63:0] hy7 /* synthesis syn_noprune=1 */ ;
reg [63:0] hy8 /* synthesis syn_noprune=1 */ ;
reg [63:0] hy9 /* synthesis syn_noprune=1 */ ;
reg [63:0] hy10 /* synthesis syn_noprune=1 */ ;

reg [63:0]  hx1 /* synthesis syn_noprune=1 */ ;
reg [63:0]  hx2 /* synthesis syn_noprune=1 */ ;
```

```
reg [63:0]  hx3 /* synthesis syn_noprune=1 */ ;
reg [63:0]  hx4 /* synthesis syn_noprune=1 */ ;
reg [63:0]  hx5 /* synthesis syn_noprune=1 */ ;
reg [63:0]  hx6 /* synthesis syn_noprune=1 */ ;
reg [63:0]  hx7 /* synthesis syn_noprune=1 */ ;
reg [63:0]  hx8 /* synthesis syn_noprune=1 */ ;
reg [63:0]  hx9 /* synthesis syn_noprune=1 */ ;
reg [63:0]  hx10 /* synthesis syn_noprune=1 */ ;

reg  [9:0]  vy1 /* synthesis syn_noprune=1 */ ;
reg [9:0]   vy2 /* synthesis syn_noprune=1 */ ;
reg  [9:0]  vy3 /* synthesis syn_noprune=1 */ ;
reg  [9:0]  vy4 /* synthesis syn_noprune=1 */ ;
reg  [9:0]  vy5 /* synthesis syn_noprune=1 */ ;
reg  [9:0]  vy6 /* synthesis syn_noprune=1 */ ;
reg  [9:0]  vy7 /* synthesis syn_noprune=1 */ ;
reg  [9:0]  vy8 /* synthesis syn_noprune=1 */ ;
reg  [9:0]  vy9 /* synthesis syn_noprune=1 */ ;
reg  [9:0]  vy10 /* synthesis syn_noprune=1 */ ;

reg  [9:0]    tvy1 /* synthesis syn_noprune=1 */ ;
reg  [9:0]    tvy2 /* synthesis syn_noprune=1 */ ;
reg [9:0]  tvy3 /* synthesis syn_noprune=1 */ ;
reg  [9:0]   tvy4 /* synthesis syn_noprune=1 */ ;
reg   [9:0]  tvy5 /* synthesis syn_noprune=1 */ ;
reg   [9:0]  tvy6 /* synthesis syn_noprune=1 */ ;
reg [9:0]  tvy7 /* synthesis syn_noprune=1 */ ;
reg [9:0] tvy8 /* synthesis syn_noprune=1 */ ;
reg  [9:0] tvy9 /* synthesis syn_noprune=1 */ ;
reg  [9:0]   tvy10 /* synthesis syn_noprune=1 */ ;

reg [9:0]   vx1 /* synthesis syn_noprune=1 */ ;
reg    [9:0] vx2 /* synthesis syn_noprune=1 */ ;
reg    [9:0] vx3 /* synthesis syn_noprune=1 */ ;
reg   [9:0] vx4 /* synthesis syn_noprune=1 */ ;
reg  [9:0]  vx5 /* synthesis syn_noprune=1 */ ;
reg   [9:0]  vx6 /* synthesis syn_noprune=1 */ ;
reg   [9:0]  vx7 /* synthesis syn_noprune=1 */ ;
reg  [9:0]  vx8 /* synthesis syn_noprune=1 */ ;
reg  [9:0]   vx9 /* synthesis syn_noprune=1 */ ;
reg  [9:0]   vx10 /* synthesis syn_noprune=1 */ ;

reg   [9:0]     tvx1 /* synthesis syn_noprune=1 */ ;
reg   [9:0]     tvx2 /* synthesis syn_noprune=1 */ ;
reg   [9:0]  tvx3 /* synthesis syn_noprune=1 */ ;
reg   [9:0]   tvx4 /* synthesis syn_noprune=1 */ ;
reg   [9:0]    tvx5 /* synthesis syn_noprune=1 */ ;
reg  [9:0]     tvx6 /* synthesis syn_noprune=1 */ ;
reg   [9:0]  tvx7 /* synthesis syn_noprune=1 */ ;
reg   [9:0]  tvx8 /* synthesis syn_noprune=1 */ ;
reg  [9:0]  tvx9 /* synthesis syn_noprune=1 */ ;
reg   [9:0]    tvx10 /* synthesis syn_noprune=1 */ ;


reg [9:0]    a1 /* synthesis syn_noprune=1 */ ;
reg    [9:0]  a2 /* synthesis syn_noprune=1 */ ;
reg    [9:0]  a3 /* synthesis syn_noprune=1 */ ;
reg   [9:0]  a4 /* synthesis syn_noprune=1 */ ;
reg  [9:0]   a5 /* synthesis syn_noprune=1 */ ;
reg   [9:0]   a6 /* synthesis syn_noprune=1 */ ;
reg   [9:0]   a7 /* synthesis syn_noprune=1 */ ;
reg  [9:0]   a8 /* synthesis syn_noprune=1 */ ;
reg  [9:0]    a9 /* synthesis syn_noprune=1 */ ;
```

```
reg  [9:0]    a10 /* synthesis syn_noprune=1 */ ;

reg   [9:0]     ta1 /* synthesis syn_noprune=1 */ ;
reg   [9:0]     ta2 /* synthesis syn_noprune=1 */ ;
reg   [9:0]  ta3 /* synthesis syn_noprune=1 */ ;
reg   [9:0]   ta4 /* synthesis syn_noprune=1 */ ;
reg   [9:0]    ta5 /* synthesis syn_noprune=1 */ ;
reg  [9:0]     ta6 /* synthesis syn_noprune=1 */ ;
reg   [9:0]  ta7 /* synthesis syn_noprune=1 */ ;
reg   [9:0]  ta8 /* synthesis syn_noprune=1 */ ;
reg  [9:0]  ta9 /* synthesis syn_noprune=1 */ ;
reg   [9:0]    ta10 /* synthesis syn_noprune=1 */ ;

reg [9:0]     b1 /* synthesis syn_noprune=1 */ ;
reg    [9:0]   b2 /* synthesis syn_noprune=1 */ ;
reg    [9:0]   b3 /* synthesis syn_noprune=1 */ ;
reg   [9:0]   b4 /* synthesis syn_noprune=1 */ ;
reg  [9:0]    b5 /* synthesis syn_noprune=1 */ ;
reg   [9:0]    b6 /* synthesis syn_noprune=1 */ ;
reg   [9:0]    b7 /* synthesis syn_noprune=1 */ ;
reg  [9:0]    b8 /* synthesis syn_noprune=1 */ ;
reg  [9:0]     b9 /* synthesis syn_noprune=1 */ ;
reg  [9:0]     b10 /* synthesis syn_noprune=1 */ ;

reg   [9:0]     tb1 /* synthesis syn_noprune=1 */ ;
reg   [9:0]     tb2 /* synthesis syn_noprune=1 */ ;
reg   [9:0]  tb3 /* synthesis syn_noprune=1 */ ;
reg   [9:0]   tb4 /* synthesis syn_noprune=1 */ ;
reg   [9:0]    tb5 /* synthesis syn_noprune=1 */ ;
reg  [9:0]     tb6 /* synthesis syn_noprune=1 */ ;
reg   [9:0]  tb7 /* synthesis syn_noprune=1 */ ;
reg   [9:0]  tb8 /* synthesis syn_noprune=1 */ ;
reg  [9:0]  tb9 /* synthesis syn_noprune=1 */ ;
reg   [9:0]    tb10 /* synthesis syn_noprune=1 */ ;

reg [63:0] ha1 /* synthesis syn_noprune=1 */ ;
reg [63:0] ha2 /* synthesis syn_noprune=1 */ ;
reg [63:0] ha3 /* synthesis syn_noprune=1 */ ;
reg [63:0] ha4 /* synthesis syn_noprune=1 */ ;
reg [63:0] ha5 /* synthesis syn_noprune=1 */ ;
reg [63:0] ha6 /* synthesis syn_noprune=1 */ ;
reg [63:0] ha7 /* synthesis syn_noprune=1 */ ;
reg [63:0] ha8 /* synthesis syn_noprune=1 */ ;
reg [63:0] ha9 /* synthesis syn_noprune=1 */ ;
reg [63:0] ha10 /* synthesis syn_noprune=1 */ ;


reg [63:0] hb1 /* synthesis syn_noprune=1 */ ;
reg [63:0] hb2 /* synthesis syn_noprune=1 */ ;
reg [63:0] hb3 /* synthesis syn_noprune=1 */ ;
reg [63:0] hb4 /* synthesis syn_noprune=1 */ ;
reg [63:0] hb5 /* synthesis syn_noprune=1 */ ;
reg [63:0] hb6 /* synthesis syn_noprune=1 */ ;
reg [63:0] hb7 /* synthesis syn_noprune=1 */ ;
reg [63:0] hb8 /* synthesis syn_noprune=1 */ ;
reg [63:0] hb9 /* synthesis syn_noprune=1 */ ;
reg [63:0] hb10 /* synthesis syn_noprune=1 */ ;


localparam [9:0]sv1=10'b0000000001;
localparam [9:0]sv2=10'b0000000010;
localparam [9:0]sv3=10'b0000000100;
localparam [9:0]sv4=10'b0000001000;
```

```
localparam [9:0]sv5=10'b0000010000;
localparam [9:0]sv6=10'b0000100000;
localparam [9:0]sv7=10'b0001000000;
localparam [9:0]sv8=10'b0010000000;
localparam [9:0]sv9=10'b0100000000;
localparam [9:0]sv10=10'b1000000000;
reg[1:0]all_over /* synthesis syn_noprune=1 */ ;
reg [3:0] stepa /* synthesis syn_noprune=1 */ ;
reg [3:0] stepb /* synthesis syn_noprune=1 */ ;

reg [15:0] round_2x /* synthesis syn_noprune=1 */ ;
reg [15:0] round_2y /* synthesis syn_noprune=1 */ ;

reg [31:0] halfy /* synthesis syn_noprune=1 */ ;
reg [31:0] one_fourthy /* synthesis syn_noprune=1 */ ;
reg [63:0]  tDNy /* synthesis syn_noprune=1 */ ;
reg [31:0] one_eighthy /* synthesis syn_noprune=1 */ ;
reg [31:0]thr_fourthy /* synthesis syn_noprune=1 */ ;
reg [31:0] tone_fourthy /* synthesis syn_noprune=1 */ ;
reg [63:0] tpDNy /* synthesis syn_noprune=1 */ ;


initial
begin

$readmemb("../fi_val.txt",img);
$readmemb("../hv.txt",h);

end


initial
begin

$readmemb("../fi_val.txt",img);
$readmemb("../hv.txt",h);

end




always@(posedge clk)

begin
if(reset==0)
begin
count<=0 ; hs<=0; max<=0; newh<=0; sec<=0; fr<=0;t2<=0;ecount<=0;wcount<=0;
    ncount<=0; same<=0;Max<=0;
end

else
begin
if(hs==0)
begin

hs<=1;
end
if(hs==1)
begin
temp=img[count];
hs<=2;
end
if(hs==2)
begin
```

```
t2<=temp;
hs<=3;
end
if(hs==3)
begin
oldh=h[t2];
hs<=4;
if(Max<t2) begin Max<=t2;end
end
if(hs==4)
begin
newh<=oldh+1;
hs<=5;
end
if(hs==5)
begin

if(fr<newh && hs==5)
begin
max<=t2;fr<=newh;hs<=6;
end
if(hs>3)
begin
hs<=6;
end
end
if(hs==6)
begin
h[t2]<=newh;
hs<=7;
end
if(hs==7)
begin

if(count<9999)
begin
count<=count+1;hs<=0;
end
else
begin
hs<=11;
$display("max %d %d" , max,Max);
end
end

if(hs==11)
begin
max<=(max+Max)>>1;hs<=10;
end

end


end

always@(posedge clk)
begin
if(reset==0)
begin
scan<=0;cc<=0;rc<=0;starc<=0;ss<=0;px=0;px2<=0;ff1<=0;ff2<=0;ff3<=0;done<=0;
    mstarc<=0;fifo_state<=0;
end
else
if(hs==10)
begin
if(ss==0)
```

```
begin
ss<=1;//$display("state %d" , ss);
end
if(ss==1)
begin
px=img[scan]; ss<=2;     //$display("state %d" , ss);
end

if(ss==2)
begin
if(cc==Col-1)
begin
rc<=rc+1;cc<=0;ss<=3;//px2<=px;//$display("cc==col pixel %d" , px);
end
else
begin
cc<=cc+1;ss<=3;//px2<=px;//$display("cc!=col  pixel %d row col %d %d " , px,
    row ,Col);
end
end

if(ss==3)
begin
if(px>max && rc>5 && cc>5 && cc<Col-5 && rc<row-5)
begin
//$display("ind %d pixel %d x %d y %d " , scan,px,rc,cc);
tff1<=rc;tff2<=cc-1;tff3<=scan;done<=1;
end
if(ss>1)
begin
ss<=4;// $display(" state %d %d   %d " , ss,rc,cc);

end
// if(scan==1483)begin
//$display("wtf %d rc cc %d %d",px,rc,cc);end

end

if(ss==4)
begin
if(starc==0 && done==1)
begin
ff1<=tff1;ff2<=tff2;ff3<=tff3;ss<=5; starc<=starc+1; fifo_state<=1;ss<=5;
//$display(" starc %d ss %d  ",starc,ss );

end
if(starc>0 && done==1)
begin

// $display("more than 1 star starc %d ss %d   ",starc, ss);
en<=starc;beg<=0; conflict<=0;ss<=11;

end

if(done==0)begin
ss<=5;end


end

if(ss==11)
begin
if(beg<en)begin
rowf= xbuf[beg]-4;rowl= xbuf[beg]+14;colf= ybuf[beg]-6;coll= ybuf[beg]+16; ss
    <=13;
//$display(" beg is less ss %d en %d beg %d",ss,en,beg);
end
```

```
else
begin
ss<=14; //$display("beg greater ss %d en %d beg %d",ss,en,beg);
end
end


if(ss==13)
begin
if( tff1<= rowl && tff1 >= rowf)begin
if(tff2<colf || tff2 > coll)begin
conflict<=conflict+1;end
end
else
begin
conflict<=conflict+1;
end
if(ss>1)
begin
beg<=beg+1; ss<=11; //$display("  row f %d rowl %d colf %d coll %d",rowf,rowl,
    colf,coll);
end

end


if(ss==14)
begin
if(conflict!=0 && conflict==starc)begin
ff1<=tff1;ff2<=tff2;ff3<=tff3;starc<=starc+1; fifo_state<=2;

end
if(ss>0)begin
ss<=5;end //$display(" conflict %d starc %d",conflict,starc);
end


if(ss==5)
begin
ss<=6;done<=0;
// $display("counter %d x %d y %d ind %d state %d #str %d" , scan,ff1,ff2,ff3,
    ss,starc);
end
if(ss==6)
begin

if(scan<9999)
begin
scan<=scan+1;ss<=0;fifo_state<=0;
end
else
begin
mstarc <= starc ;ss<=10;fifo_state<=0;
end
end
if(ss==10)
begin
ss<=9;
// $display("counter %d x %d y %d ind %d state %d #str %d" , scan,ff1,ff2,ff3,
    ss,starc);
end


end
end
```

```
always@(posedge clk)
begin

if(reset==0)
begin
bfd<=0;fifo_count <=0;
end

if(bfd==0 && ss==5 && fifo_state!=0 )
begin
if( starc>0) begin
xbuf[starc-1]=ff1;ybuf[starc-1]=ff2;ibuf[starc-1]=ff3; fifo_count<=fifo_count
    +1;
//$display(" starc  %d %d %d",starc,mstarc,fifo_count);
end
end

if(ss==10 && starc== mstarc)
begin
bfd<=1;//fifo_count<=fifo_count+1;
//$display("state 10 starc %d mstarc %d ",starc,mstarc);
end


end


always@(posedge clk)begin
if(reset==0)begin
y_max<=0; ty_max<=0; yshalf_max <=0; Cyc <=0; dely<=2147483648; tdely
    <=2147483648; tdely2<=255;end
else
begin
if(rb>=4 && bs==4)
begin

if(rb==4)begin
//isy2can==1 && Sx123>Cyc && dely > dely31
if( isy2can==1 && Sx123>Cyc    )begin

Cyc<=Sx123; y_max<=rx-3;//dely<=dely31; //$display("rb for y %d dely31 %d dely
    %d,rx %d ",rb,dely31,dely,rx );

end
if(isy2can==1 && tdely > dely31) begin  ty_max<=rx-3; tdely<=dely31;  end
//if(isy2can==1 && tdely2 > dely31) begin  ty_max2<=rx-3; tdely2<=dely31;  end
end

if(rb==5)begin
//isy3can==1   && Sx234>Cyc && dely > dely42
if(isy3can==1   && Sx234>Cyc   )begin

Cyc<=Sx234;    y_max<=rx-3;//dely<=dely42; //$display("rb for y %d dely31 %d
    dely %d  ,rx %d",rb,dely42,dely,rx);

end
if(isy3can==1   && tdely > dely42) begin  ty_max<=rx-3;tdely<=dely42;end
// if(isy3can==1   && tdely2 > dely42) begin  ty_max2<=rx-3;tdely2<=dely42;end
end
```

```
if(rb==6)begin
//isy4can==1   && Sx345>Cyc && dely > dely53
if(isy4can==1   && Sx345>Cyc  )begin

Cyc<=Sx345; y_max<=rx-3;//dely<=dely53;    //$display("rb %d for y dely53 %d
    dely %d,rx %d ",rb,dely53,dely,rx);

end
if(isy4can==1   && tdely > dely53) begin  ty_max<=rx-3;tdely<=dely53;end
//   if(isy4can==1   && tdely2 > dely53) begin  ty_max2<=rx-3;tdely2<=dely53;
    end
end

if(rb==7)begin
//isy5can==1  && Sx456>Cyc && dely > dely64
if(isy5can==1  && Sx456>Cyc )begin

Cyc<=Sx456;y_max<=rx-3;//dely<=dely64;  //$display("rb %d for y dely64 %d dely
    %d,rx %d ",rb,dely64,dely,rx);

end
if(isy5can==1  && tdely > dely64) begin  ty_max<=rx-3;tdely<=dely64;end
//if(isy5can==1  && tdely2 > dely64) begin  ty_max2<=rx-3;tdely2<=dely64;end
end

if(rb==8)begin
//isy6can==1   && Sx567>Cyc && dely > dely75
if(isy6can==1   && Sx567>Cyc  )begin

Cyc<=Sx567;y_max<=rx-3;//dely<=dely75;  //$display("rb %d for y dely75 %d dely
    %d ,rx %d",rb,dely75,dely,rx);

end
if(isy6can==1   && tdely > dely75) begin  ty_max<=rx-3;tdely<=dely75;end
// if(isy6can==1   && tdely2 > dely75) begin  ty_max2<=rx-3;tdely2<=dely75;end
end


if(rb==9)begin
//isy7can==1  && Sx678>Cyc && dely > dely86
if(isy7can==1  && Sx678>Cyc )begin

Cyc<=Sx678;y_max<=rx-3;//dely<=dely86;    //$display("rb %d for y dely86 check
   %d dely %d ,rx %d ",rb,dely86,dely,rx );

end
if(isy7can==1  && tdely > dely86) begin  ty_max<=rx-3;tdely<=dely86;end
//if(isy7can==1  && tdely2 > dely86) begin  ty_max2<=rx-3;tdely2<=dely86;end
end

if(rb==10)begin
//isy8can==1  && Sx789>Cyc &&  dely > dely97
if(isy8can==1  && Sx789>Cyc )begin

Cyc<=Sx789;y_max<=rx-3;//dely<=dely97;  //$display("rb %d for y dely97 %d dely
    %d ,rx %d ",rb,dely97,dely,rx );

end

if(isy8can==1  && tdely > dely97) begin  ty_max<=rx-3; tdely<=dely86;end
// if(isy8can==1  && tdely2 > dely97) begin  ty_max2<=rx-3;tdely2<=dely86;end
end
```

```
//$display("rb b %d  rx %d (ymax) %d dely %d  Sx %d %d %d %d %d %d %d   ",rb,
    rx,y_max,dely,Sx123,Sx234,Sx345,Sx456,Sx567,Sx678,Sx789);
//$display("rb rx %d %d",rb,rx);
//$display("BBB and the dels %d %d %d %d %d %d %d  ",dely31,dely42,dely53,
    dely64,dely75,dely86,dely97);
//$display("BBB the cans %d %d %d %d %d %d %d",isy2can,isy3can ,isy4can ,
    isy5can ,isy6can,isy7can ,isy8can );
end

if(bs==15)begin
y_max<=0; ty_max<=0; yshalf_max <=0;Cyc <=0;dely<=2147483648;tdely
    <=2147483648;tdely2<=255;
end

end
end


always@(posedge clk)begin
if(reset==0)begin
x_max<=0; tx_max<=0;  shalf_max <=0;Cxc <=0; delx<= 2147483648; tdelx<=
    2147483648; tdelx2<= 255;
end

else
begin
if(rb>=4 && bs==4)
begin

if(rb==4)begin
//isx2can==1  &&  Sy123>Cxc && delx > delx31
if( isx2can==1  &&  Sy123>Cxc    )begin

Cxc<=Sy123; x_max<=ry-3;//delx<=delx31;//$display("rb %d delx31 %d delx %d ,ry
    %d",rb,delx31,delx,ry);

end
if(isx2can==1  && tdelx > delx31) begin  tx_max<=ry-3;tdelx<=delx31; end
// if(isx2can==1  && tdelx2 > delx31) begin  tx_max2<=ry-3;tdelx2<=delx31; end
end

if(rb==5)begin
//isx3can==1  &&     Sy234>Cxc && delx > delx42
if(isx3can==1  &&   Sy234>Cxc  )begin

Cxc<=Sy234;   x_max<=ry-3;//delx<=delx42;//$display("rb %d delx42 check %d
    delx %d ,ry %d",rb,delx42,delx,ry);

end
if(isx3can==1  && tdelx > delx42) begin  tx_max<=ry-3;tdelx<=delx42;  end
// if(isx3can==1  && tdelx2 > delx42) begin  tx_max2<=ry-3;tdelx2<=delx42;
    end
end


if(rb==6)begin
//isx4can==1  && Sy345>Cxc  && delx > delx53
if( isx4can==1  &&    Sy345>Cxc   )begin

Cxc<=Sy345;  x_max<=ry-3;//delx<=delx53;//$display("rb %d delx53 %d delx %d ,
    ry %d",rb,delx53,delx,ry);

end
if(isx4can==1  &&  tdelx > delx53) begin  tx_max<=ry-3;tdelx<=delx53;  end
//if(isx4can==1  &&  tdelx2 > delx53) begin  tx_max2<=ry-3;tdelx2<=delx53;
    end
end
```

```
if(rb==7)begin
//isx5can==1  &&  Sy456>Cxc && delx > delx64
if( isx5can==1  && Sy456>Cxc   )begin

Cxc<=Sy456;     x_max<=ry-3;//delx<=delx64;//$display("rb %d delx64 %d delx %d
    ,ry %d",rb,delx64,delx,ry);

end
if(isx5can==1  && tdelx > delx64) begin  tx_max<=ry-3; tdelx<=delx64;  end
//if(isx5can==1  && tdelx2 > delx64) begin  tx_max2<=ry-3; tdelx2<=delx64;
    end
end

if(rb==8)begin
//isx6can==1  &&  Sy567>Cxc &&  delx > delx75
if(isx6can==1  &&  Sy567>Cxc   )begin

Cxc<=Sy567;    x_max<=ry-3;//delx<=delx75;//$display("rb %d delx75 %d delx %d,
    ry %d ",rb,delx75,delx,ry);

end
if(isx6can==1  && tdelx > delx75) begin  tx_max<=ry-3; tdelx<=delx75;  end
// if(isx6can==1  && tdelx2 > delx75) begin  tx_max2<=ry-3; tdelx2<=delx75;
    end
end


if(rb==9)begin
//isx7can==1   &&   Sy678>Cxc  &&  delx > delx86
if(isx7can==1   &&      Sy678>Cxc  )begin

Cxc<=Sy678;    x_max<=ry-3;//delx<=delx86;//$display("rb %d delx86 %d delx %d ,
    ry %d",rb,delx86,delx,ry);

end
if(isx7can==1   &&   tdelx > delx86) begin  tx_max<=ry-3;tdelx<=delx86;   end
//if(isx7can==1   &&   tdelx2 > delx86) begin  tx_max2<=ry-3;tdelx2<=delx86;
      end
end

if(rb==10)begin
//isx8can==1  &&   Sy789>Cxc && delx > delx97
if(isx8can==1  &&      Sy789>Cxc )begin

Cxc<=Sy789;  x_max<=ry-3;//delx<=delx97;//$display("rb %d delx97 %d delx %d,ry
     %d ",rb,delx97,delx,ry );

end
if(isx8can==1  &&   tdelx > delx97) begin  tx_max<=ry-3;tdelx<=delx86; end
//if(isx8can==1  &&   tdelx2 > delx97) begin  tx_max2<=ry-3;tdelx2<=delx86;end
end


//$display("rb b %d  ry %d (xmax) %d delx %d  Sy %d %d %d %d %d %d %d   ",rb,
    ry,x_max,delx,Sy123,Sy234,Sy345,Sy456,Sy567,Sy678,Sy789);
//$display("rb ry %d %d",rb,ry);
//$display("AAA and the dels %d %d %d %d %d %d %d  ",delx31,delx42,delx53,
    delx64,delx75,delx86,delx97);
//$display("AAA the cans %d %d %d %d %d %d %d",isx2can,isx3can ,isx4can ,
    isx5can ,isx6can,isx7can ,isx8can );
end



if(bs==15)begin
```

```
x_max<=0;tx_max<=0  ;shalf_max <=0;Cxc<=0;delx<= 2147483648; tdelx<=
    2147483648;tdelx2<= 255;end

end

end



always@(posedge clk)
begin
if(reset==0)
begin

buff_count<=0;bs<=42;rolling<=1;addc<=0;atleast1<=0;
fs1<=0;fs2<=0;fs3<=0;fs4<=0;fs5<=0;fs6<=0;fs7<=0;fs8<=0;fs9<=0;
ft1<=0;fT2<=0;ft3<=0;ft4<=0;ft5<=0;ft6<=0;ft7<=0;ft8<=0;ft9<=0;
difx1 <=0;difx2<=0; difx3<=0;dify1<=0; dify2<=0;  dify3<=0;     a<=0;b<=0;
r<=0;c<=0;i<=0;sx<=0;sy<=0;DN<=0;Smx<=0;Smy<=0;
up<=0; down<=0; upx<=0;downx<=0; roi_cou<=0; rb<=0; DN<=0;
posx<=0;posy<=0;remx<=0;remy<=0;aDN<=0;bDN<=0; change<=0;
isy2can<=0   ;isy3can <=0 ;isy4can <=0 ;isy5can <=0 ;isy6can <=0 ;isy7can <=0
    ;isy8can <=0 ;
isx2can <=0 ;isx3can <=0 ;isx4can <=0 ;isx5can <=0 ;isx6can <=0 ;isx7can <=0 ;
     isx8can <=0 ;
Sx123 <=0; Sx234 <=0; Sx345 <=0; Sx456 <=0; Sx567 <=0; Sx678 <=0; Sx789 <=0;
Sy123 <=0; Sy234 <=0; Sy345 <=0; Sy456 <=0; Sy567 <=0; Sy678 <=0; Sy789 <=0;
hx1<=0;  hx2<=0;  hx3<=0;  hx4<=0;  hx5<=0;  hx6<=0;  hx7<=0;  hx8<=0;  hx9
    <=0;  hx10<=0;
hy1<=0;  hy2<=0;  hy3<=0;  hy4<=0;  hy5<=0;  hy6<=0;  hy7<=0;  hy8<=0;  hy9
    <=0;  hy10<=0;
ha1<=0;  ha2<=0;  ha3<=0;  ha4<=0;  ha5<=0;  ha6<=0;  ha7<=0;  ha8<=0;  ha9
    <=0;  ha10<=0;
hb1<=0;  hb2<=0;  hb3<=0;  hb4<=0;  hb5<=0;  hb6<=0;  hb7<=0;  hb8<=0;  hb9
    <=0;  hb10<=0;
stepa<=0;stepb<=0;  tx_max2<=0;ty_max2<=0; f_DN<=0; all_over<=0;
end

else
if( fifo_count>0 )
begin
if(bs==42)begin
//$display("%d %d , %d ",buff_count,fifo_count,bs);
if( buff_count<fifo_count) begin bs<=0;end
else if (mstarc>0 && buff_count==mstarc ) begin bs<=49;all_over<=1; end
else   begin bs<=42;end


end

if(bs==0)
begin
//if(buff_count<fifo_count)
//  begin
//$display("%d %d %d %d ",atleast1,buff_count,fifo_count,bs);
tr=xbuf[atleast1];tc=ybuf[atleast1];ti=ibuf[atleast1];bs<=21;
//end
end
if(bs==21)
begin
r<=tr;c<=tc;i<=ti;bs<=1;//$display("row col in %d %d %d",tr-1,tc-2,ti);
end

if(bs==1)
begin
// $display("row col in %d %d %d",r,c-2,i-row-1);
rx<=r; ry<=c-2; ix<=i-2; iy<=i-2; bs<=2;
```

```
end

if(bs==2)
begin
if(roi_cou==0)
begin
s1=img[ix]; t1=img[iy]; roi_cou<=roi_cou+1;
end
if(roi_cou==1)
begin
s2=img[ix+row]; T2=img[iy+1]; roi_cou<=roi_cou+1; fs1<=s1;ft1<=t1;
tvx1=ry&1; tvx2=ry&2; tvx3=ry&4; tvx4=ry&8;tvx5=ry&16; tvx6=ry&32;tvx7=ry&64;
    tvx8=ry&128;tvx9=ry&256; tvx10=ry&512;
tvy1=rx&1; tvy2=rx&2; tvy3=rx&4; tvy4=rx&8;tvy5=rx&16; tvy6=rx&32;tvy7=rx&64;
    tvy8=rx&128;tvy9=rx&256; tvy10=rx&512;
end
if(roi_cou==2)
begin
s3=img[ix+row2]; t3=img[iy+2]; roi_cou<=roi_cou+1; fs2<=s2;fT2<=T2;

end
if(roi_cou==3)
begin
s4=img[ix+row3]; t4=img[iy+3]; roi_cou<=roi_cou+1; fs3<=s3;ft3<=t3;sy<=fs1+fs2
    ;sx<=ft1+fT2;
end
if(roi_cou==4)
begin
s5=img[ix+row4]; t5=img[iy+4]; roi_cou<=roi_cou+1;fs4<=s4;ft4<=t4;sy<=sy+
    fs3;sx<=sx+ft3;
end

if(roi_cou==5)
begin
s6=img[ix+row5]; t6=img[iy+5]; roi_cou<=roi_cou+1;fs5<=s5; ft5<=t5; sy<=sy
    + fs4;sx<=sx+ ft4;
end
if(roi_cou==6)
begin
s7=img[ix+row6]; t7=img[iy+6]; roi_cou<=roi_cou+1; fs6<=s6;ft6<=t6; sy<=sy+
    fs5;sx<=sx+ ft5;
end
if(roi_cou==7)
begin
s8=img[ix+row7]; t8=img[iy+7]; roi_cou<=roi_cou+1; fs7<=s7;ft7<=t7; sy<=sy
    + fs6;sx<=sx+ ft6;
end
if(roi_cou==8)
begin
s9=img[ix+row8]; t9=img[iy+8]; roi_cou<=roi_cou+1;fs8<=s8;ft8<=t8; sy<=sy+
     fs7;sx<=sx+ ft7;
end
if(roi_cou==9)
begin
roi_cou<=roi_cou+1; fs9<=s9;ft9<=t9;sy<=sy+ fs8;sx<=sx+ ft8;
end
if(roi_cou==10)
begin
roi_cou<=roi_cou+1; sy<=sy+ fs9;sx<=sx+ ft9;
end
if(roi_cou==11)
begin
roi_cou<=roi_cou+1; bs<=26; DN<=DN+sy;
end
```

```
//$display("toi cou %d",roi_cou);
end

if(bs==26)begin
bs<=27;
vx1 <=tvx1 ; vx2 <= tvx2; vx3 <=tvx3; vx4 <=tvx4;vx5<=tvx5; vx6 <=tvx6;vx7 <=
    tvx7; vx8 <=tvx8;vx9 <=tvx9; vx10 <=tvx10;
vy1<=tvy1 ; vy2<= tvy2; vy3<=tvy3; vy4<=tvy4;vy5<=tvy5; vy6<=tvy6;vy7<=tvy7;
    vy8<=tvy8;vy9<=tvy9; vy10<=tvy10;
//$display("bs 26 %d",vx1);
end
if(bs==27)begin
if(bs>0) begin bs<=4;end

if( vx1==1)begin  hy1<= sy;end
if( vx2==2)begin  hy2<= sy<<1;end
if( vx3==4)begin  hy3<= sy<<2;end
if( vx4==8)begin  hy4<= sy<<3;end
if( vx5==16)begin  hy5<= sy<<4;end
if( vx6==32)begin  hy6<= sy<<5;end
if( vx7==64)begin  hy7<= sy<<6;end
if( vx8==128)begin  hy8<= sy<<7;end
if( vx9==256)begin  hy9<= sy<<8;end

if( vy1==1)begin  hx1<= sx;end
if( vy2==2)begin  hx2<= sx<<1;end
if( vy3==4)begin  hx3<= sx<<2;end
if( vy4==8)begin  hx4<= sx<<3;end
if( vy5==16)begin  hx5<= sx<<4;end
if( vy6==32)begin  hx6<= sx<<5;end
if( vy7==64)begin  hx7<= sx<<6;end
if( vy8==128)begin  hx8<= sx<<7;end
if( vy9==256)begin  hx9<= sx<<8;end
//if( vx10==512)begin  hy10<= sy<<9;end
//if( vy10==512)begin  hx10<= sx<<9;end
//$display("bs 27 %d %d",sx,sy);
end
if(bs==4)
begin

if(rb==0)begin

bs<=6;rb<=rb+1;

Sy1<=sy;Sx1<=sx;

Smy<=Smy+ hy1+ hy2+ hy3+ hy4+ hy5+ hy6+ hy7+ hy8+ hy9+ hy10 ;
Smx<=Smx+hx1+hx2+hx3+hx4+hx5+hx6+hx7+hx8+hx9+hx10    ;



end
else if(rb==1)begin

bs<=6; rb<=rb+1;
Sy2<=sy;Sx2<=sx;
Smy<=Smy+ hy1+ hy2+ hy3+ hy4+ hy5+ hy6+ hy7+ hy8+ hy9+ hy10 ;
Smx<=Smx+hx1+hx2+hx3+hx4+hx5+hx6+hx7+hx8+hx9+hx10    ;


end
else  if(rb==2)begin

bs<=6; rb<=rb+1;
Smy<=Smy+ hy1+ hy2+ hy3+ hy4+ hy5+ hy6+ hy7+ hy8+ hy9+ hy10 ;
```

```
Smx<=Smx+hx1+hx2+hx3+hx4+hx5+hx6+hx7+hx8+hx9+hx10    ;
Sy3<=sy;Sx3<=sx;

Sy123<=Sy1+Sy2;Sx123<=Sx1+Sx2;

end

else if(rb==3)begin

bs<=6; rb<=rb+1;
Smy<=Smy+ hy1+ hy2+ hy3+ hy4+ hy5+ hy6+ hy7+ hy8+ hy9+ hy10 ;
Smx<=Smx+hx1+hx2+hx3+hx4+hx5+hx6+hx7+hx8+hx9+hx10    ;
Sy4<=sy;Sx4<=sx;

Sy123<=Sy123+Sy3; Sx123<=Sx123+Sx3;     Sy234<=Sy2+Sy3;Sx234<=Sx2+Sx3;
if(Sy1>Sy3)begin delx31<=Sy1-Sy3;end
else begin delx31<=Sy3-Sy1;end
if(Sx1>Sx3)begin dely31<=Sx1-Sx3;end
else begin dely31<=Sx3-Sx1;end

if((Sy1>Sy2 && Sy2<Sy3) || (Sy1<Sy2 && Sy2>Sy3))begin isx2can<=1;end
if((Sx1>Sx2 && Sx2<Sx3) || (Sx1<Sx2 && Sx2>Sx3))begin isy2can<=1;end
//$display("Sy1 Sy2 Sy3 Sx1 Sx2 Sx3 %d %d %d %d %d %d ",Sy1, Sy2 ,Sy3 ,Sx1,
    Sx2 ,Sx3);

end
else if(rb==4)begin

bs<=6; rb<=rb+1;
Smy<=Smy+ hy1+ hy2+ hy3+ hy4+ hy5+ hy6+ hy7+ hy8+ hy9+ hy10 ;
Smx<=Smx+hx1+hx2+hx3+hx4+hx5+hx6+hx7+hx8+hx9+hx10    ;
Sy5<=sy;Sx5<=sx;

Sy234<=Sy234+Sy4; Sx234<=Sx234+Sx4; Sy345<=Sy3+Sy4;Sx345<=Sx3+Sx4;

if(Sy2>Sy4)begin delx42<=Sy2-Sy4;end
else begin delx42<=Sy4-Sy2;end
if(Sx2>Sx4)begin dely42<=Sx2-Sx4;end
else begin dely42<=Sx4-Sx2;end


if((Sy2>Sy3 && Sy3<Sy4) || (Sy4<Sy3 && Sy3>Sy4))begin  isx3can<=1;end
if((Sx2>Sx3 && Sx3<Sx4) || (Sx4<Sx3 && Sx3>Sx4))begin isy3can<=1;end
//$display("Sy2 Sy3 Sy4 Sx2 Sx3 Sx4 %d %d %d %d %d %d ",Sy2, Sy3, Sy4, Sx2,
    Sx3, Sx4);
if(stepa==0)begin
if(isx2can==1) begin tx_max2<=ry-3; stepa<=stepa+1;end
end
else if(stepa<7) begin
if(isx2can==1) begin tx_max2<=((tx_max2+ry-3)>>1); stepa<=stepa+1; end
end

if(stepb==0)begin
if(isy2can==1) begin ty_max2<=rx-3; stepb<=stepb+1;end
end
else if(stepb<7) begin
if(isy2can==1) begin ty_max2<=((ty_max2+rx-3)>>1); stepb<=stepb+1; end
end



end
else if(rb==5)begin

bs<=6;rb<=rb+1;
Smy<=Smy+ hy1+ hy2+ hy3+ hy4+ hy5+ hy6+ hy7+ hy8+ hy9+ hy10 ;
Smx<=Smx+hx1+hx2+hx3+hx4+hx5+hx6+hx7+hx8+hx9+hx10    ;
Sy6<=sy;Sx6<=sx;
```

```
Sy345<=Sy345+Sy5;Sx345<=Sx345+Sx5; Sy456<=Sy4+Sy5;Sx456<=Sx4+Sx5;

if(Sy3>Sy5)begin delx53<=Sy3-Sy5;end
else begin delx53<=Sy5-Sy3;end
if(Sx3>Sx5)begin dely53<=Sx3-Sx5;end
else begin dely53<=Sx5-Sx3;end


if((Sy3>Sy4 && Sy4<Sy5) || (Sy3<Sy4 && Sy4>Sy5))begin isx4can<=1;end
if((Sx3>Sx4 && Sx4<Sx5) || (Sx3<Sx4 && Sx4>Sx5))begin isy4can<=1;end

//$display("Sy3 Sy4 Sy5 Sx3 Sx4 Sx5 %d %d %d %d %d %d ",Sy3, Sy4 ,Sy5, Sx3 ,
    Sx4 ,Sx5);

if(stepa==0)begin
if(isx3can==1) begin tx_max2<=ry-3; stepa<=stepa+1;end
end
else if(stepa<7) begin
if(isx3can==1) begin tx_max2<=((tx_max2+ry-3)>>1); stepa<=stepa+1; end
end

if(stepb==0)begin
if(isy3can==1) begin ty_max2<=rx-3; stepb<=stepb+1;end
end
else if(stepb<7) begin
if(isy3can==1) begin ty_max2<=((ty_max2+rx-3)>>1); stepb<=stepb+1; end
end


end
else if(rb==6)begin

bs<=6; rb<=rb+1;
Smy<=Smy+ hy1+ hy2+ hy3+ hy4+ hy5+ hy6+ hy7+ hy8+ hy9+ hy10 ;
Smx<=Smx+hx1+hx2+hx3+hx4+hx5+hx6+hx7+hx8+hx9+hx10    ;
Sy7<=sy;Sx7<=sx;


Sy456<= Sy456+Sy6 ;Sx456<=Sx456+Sx6; Sy567<=Sy5+Sy6; Sx567<=Sx5+Sx6;


if(Sy4>Sy6)begin delx64<=Sy4-Sy6;end
else begin delx64<=Sy6-Sy4;end
if(Sx4>Sx6)begin dely64<=Sx4-Sx6;end
else begin dely64<=Sx6-Sx4;end


if((Sy4>Sy5 && Sy5<Sy6) || (Sy4<Sy5 && Sy5>Sy6))begin   isx5can<=1;end
if((Sx4>Sx5 && Sx5<Sx6) || (Sx4<Sx5 && Sx5>Sx6))begin   isy5can<=1;end
// $display("Sy4 Sy5 Sy6 Sx4 Sx5 Sx6 %d %d %d %d %d %d ",Sy4, Sy5, Sy6, Sx4,
    Sx5, Sx6 );
if(stepa==0)begin
if(isx4can==1) begin tx_max2<=ry-3; stepa<=stepa+1;end
end
else if(stepa<7) begin
if(isx4can==1) begin tx_max2<=((tx_max2+ry-3)>>1); stepa<=stepa+1; end
end

if(stepb==0)begin
if(isy4can==1) begin ty_max2<=rx-3; stepb<=stepb+1;end
end
else if(stepb<7) begin
if(isy4can==1) begin ty_max2<=((ty_max2+rx-3)>>1); stepb<=stepb+1; end
end


end
```

```
else if(rb==7)begin

bs<=6; rb<=rb+1;
Smy<=Smy+ hy1+ hy2+ hy3+ hy4+ hy5+ hy6+ hy7+ hy8+ hy9+ hy10 ;
Smx<=Smx+hx1+hx2+hx3+hx4+hx5+hx6+hx7+hx8+hx9+hx10    ;
Sy8<=sy;Sx8<=sx;

Sy567<=Sy567+Sy7;Sx567<=Sx567+Sx7; Sy678<=Sy6+Sy7;Sx678<=Sx6+Sx7;

if(Sy5>Sy7)begin delx75<=Sy5-Sy7;end
else begin delx75<=Sy7-Sy5;end
if(Sx5>Sx7)begin dely75<=Sx5-Sx7;end
else begin dely75<=Sx7-Sx5;end


if((Sy5>Sy6 && Sy6<Sy7) || (Sy5<Sy6 && Sy6>Sy7))begin    isx6can<=1;end
if((Sx5>Sx6 && Sx6<Sx7) || (Sx5<Sx6 && Sx6>Sx7))begin      isy6can<=1;end
//$display("Sy5 Sy6 Sy7 Sx5 Sx6 Sx7 %d %d %d %d %d %d ",Sy5, Sy6, Sy7, Sx5,
    Sx6, Sx7 );
if(stepa==0)begin
if(isx5can==1) begin tx_max2<=ry-3; stepa<=stepa+1;end
end
else if(stepa<7) begin
if(isx5can==1) begin tx_max2<=(tx_max2+ry-3)>>1; stepa<=stepa+1; end
end

if(stepb==0)begin
if(isy5can==1) begin ty_max2<=rx-3; stepb<=stepb+1;end
end
else if(stepb<7) begin
if(isy5can==1) begin ty_max2<=(ty_max2+rx-3)>>1; stepb<=stepb+1; end
end


end
else if(rb==8)  begin

bs<=6;rb<=rb+1;
Smy<=Smy+ hy1+ hy2+ hy3+ hy4+ hy5+ hy6+ hy7+ hy8+ hy9+ hy10 ;
Smx<=Smx+hx1+hx2+hx3+hx4+hx5+hx6+hx7+hx8+hx9+hx10    ;
Sy9<=sy;Sx9<=sx;
Sy678<=Sy8+ Sy678;Sx678<=Sx8+Sx678; Sy789<=Sy7+Sy8;Sx789<=Sx7+Sx8;

if(Sy6>Sy8)begin delx86<=Sy6-Sy8;end
else begin delx86<=Sy8-Sy6;end
if(Sx6>Sx8)begin dely86<=Sx6-Sx8;end
else begin dely86<=Sx8-Sx6;end


if((Sy6>Sy7 && Sy7<Sy8) || (Sy6<Sy7 && Sy7>Sy8))begin  isx7can<=1;end
if((Sx6>Sx7 && Sx7<Sx8) || (Sx6<Sx7 && Sx7>Sx8))begin    isy7can<=1;end
//$display("Sy6 Sy7 Sy8 Sx6 Sx7 Sx8 %d %d %d %d %d %d ",Sy6, Sy7, Sy8, Sx6,
    Sx7, Sx8);
if(stepa==0)begin
if(isx6can==1) begin tx_max2<=ry-3; stepa<=stepa+1;end
end
else if(stepa<7) begin
if(isx6can==1) begin tx_max2<=(tx_max2+ry-3)>>1; stepa<=stepa+1; end
end

if(stepb==0)begin
if(isy6can==1) begin ty_max2<=rx-3; stepb<=stepb+1;end
end
else if(stepb<7) begin
if(isy6can==1) begin ty_max2<=(ty_max2+rx-3)>>1; stepb<=stepb+1; end
end
```

```
end
else if(rb==9)begin


rb<=rb+1;bs<=6;
Sy789<=Sy9+Sy789;Sx789<=Sx789+Sx9;
if(Sy7>Sy9)begin delx97<=Sy7-Sy9;end
else begin delx97<=Sy9-Sy7;end
if(Sx7>Sx9)begin dely97<=Sx7-Sx9;end
else begin dely97<=Sx9-Sx7;end


if((Sy7>Sy8 && Sy8<Sy9) || (Sy7<Sy8 && Sy8>Sy9))begin  isx8can<=1;end
if((Sx7>Sx8 && Sx8<Sx9) || (Sx7<Sx8 && Sx8>Sx9))begin    isy8can<=1;end
//$display("Sy7 Sy8 Sy9 Sx7 Sx8 Sx9 %d %d %d %d %d %d ",Sy7, Sy8, Sy9, Sx7,
    Sx8, Sx9);
if(stepa==0)begin
if(isx7can==1) begin tx_max2<=ry-3; stepa<=stepa+1;end
end
else if(stepa<7) begin
if(isx7can==1) begin tx_max2<=(tx_max2+ry-3)>>1; stepa<=stepa+1; end
end

if(stepb==0)begin
if(isy7can==1) begin ty_max2<=rx-3; stepb<=stepb+1;end
end
else if(stepb<7) begin
if(isy7can==1) begin ty_max2<=(ty_max2+rx-3)>>1; stepb<=stepb+1; end
end


end

else if(rb==10)begin

rb<=rb+1;bs<=6;end
//$display("rb 10 %d %d %d %d %d %d %d ",Sy2,Sy3,Sy4,Sy5,Sy6,Sy7,Sy8);

if(stepa==0)begin
if(isx8can==1) begin tx_max2<=ry-3; stepa<=stepa+1;end
end
else if(stepa<7) begin
if(isx8can==1) begin tx_max2<=(tx_max2+ry-3)>>1; stepa<=stepa+1; end
end

if(stepb==0)begin
if(isy8can==1) begin ty_max2<=rx-3; stepb<=stepb+1;end
end
else if(stepb<7) begin
if(isy8can==1) begin ty_max2<=(ty_max2+rx-3)>>1; stepb<=stepb+1; end
end


//$display("bs 4  %d",Smy);

end



if(bs==6)
begin
//$display("bs 6  %d %d ",rx,ry);
rx<=rx+1; ry<=ry+1; ix<=ix+1; iy<=iy+row;sy<=0;sx<=0;bs<=7;roi_cou<=0;
hx1<=0;  hx2<=0;  hx3<=0;  hx4<=0;  hx5<=0;  hx6<=0;  hx7<=0;  hx8<=0;  hx9
    <=0;  hx10<=0;
```

```
hy1<=0;  hy2<=0;  hy3<=0;  hy4<=0;  hy5<=0;  hy6<=0;  hy7<=0;  hy8<=0;  hy9
    <=0;  hy10<=0;
end

if(bs==7)
begin

if(rb<9)
begin bs<=2; end
else if(rb==10 || rb==9 )
begin bs<=4;end

else if(rb==11) begin  bs<=43;end
//$display("bs 7 %d",rb);

end


if(bs==43)
begin

if(tx_max>x_max && tx_max>tx_max2) begin a<=x_max ;decx<=12; end
else if(tx_max<x_max && tx_max<tx_max2) begin a<=(x_max+tx_max)>>1 ;decx<=12;
    end
else begin a<=(x_max+tx_max2)>>1 ;decx<=8; end


if(ty_max>y_max && ty_max>ty_max2) begin b<=y_max ;decy<=12; end
else if(ty_max<y_max && ty_max<ty_max2) begin b<=(y_max+ty_max)>>1 ;decy<=4;
    end
else begin b<=(y_max+ty_max2)>>1 ;decy<=8; end

if(bs>0) begin bs<=14;end


end

if(bs==14) begin

//a<=((a+tx_max2)>>1) ;
//b<=((b+ty_max2)>>1) ;

$display( "beforeeeeeeeeeeeeeee [%d  %d ,%d  %d],%d %d" ,a, decx,b, decy,a-(c
    -2),b-r);
if(bs>0)  begin bs<=8;end


//$display("at 14 ,a %d remx %d posx %d b %d remy %d posy %d,dn %d ,%d",a,remx
    ,posx,b,remy,posy,DN,change);

end



if(bs==8)
begin
// $display(" ##################  bs8 %d  %d are a b   ",a,b);
//a<=tx_max2;b<=ty_max2;
round_2x<=a-(c-2);round_2y<=b-r;

rb<=0; bs<=50;

end

if(bs==50)
begin
if(bs>0) begin bs<=51;end
```

```
//$display( "beforeeeeeeeeeeeeeeee [%d  %d ,%d  %d],%d %d" ,a, decx,b, decy,a-(
    c-2),b-r);
if(round_2x ==1) begin Smy<= Smy1+ Smy2+ Smy3; DN<=Sy1+Sy2+Sy3   ; end

else if(round_2x ==2) begin Smy<= Smy1+ Smy5 +  Smy2+ Smy3 +Smy4 ; DN<=Sy1+
    Sy5 +   Sy2+ Sy3 +Sy4 ; end

else if(round_2x ==3) begin Smy<=Smy2+ Smy6 +   Smy3+ Smy4 +Smy5 ; DN<=Sy2+
    Sy6 +   Sy3+ Sy4 +Sy5 ;end

else if(round_2x ==4)  begin Smy<=Smy3+ Smy7 +   Smy4+ Smy5 +Smy6 ; DN<=Sy3+
    Sy7 +    Sy4 +Sy5+ Sy6; end

else if(round_2x ==5)  begin Smy<=Smy4+ Smy8 +  Smy5+ Smy6+Smy7 ; DN<=Sy4+ Sy8
     +   Sy7+  Sy5+ Sy6; end

else if(round_2x ==6)  begin Smy<=Smy5+ Smy9 +  Smy6+ Smy7+Smy8 ; DN<=Sy5+ Sy9
     + Sy7+ Sy8+   Sy6;
 end

else if(round_2x >=7)  begin Smy<=  Smy7+Smy8+ Smy9; DN<= Sy7+ Sy8+ Sy9 ; end


if(round_2y ==1) begin Smx<= Smx1+ Smx2+ Smx3; DNy<=Sx1+Sx2+Sx3   ; end

else if(round_2y ==2) begin Smx<= Smx1+ Smx5 +  Smx2+ Smx3 +Smx4 ; DNy<=Sx1+
    Sx5 +   Sx2+ Sx3 +Sx4 ; end

else if(round_2y ==3) begin Smx<=Smx2+ Smx6 +   Smx3+ Smx4 +Smx5 ; DNy<=Sx2+
    Sx6 +   Sx3+ Sx4 +Sx5 ;end

else if(round_2y ==4)  begin Smx<=Smx3+ Smx7 +   Smx4+ Smx5 +Smx6 ; DNy<=Sx3+
    Sx7 +    Sx4 +Sx5+ Sx6; end

else if(round_2y ==5)  begin Smx<=Smx4+ Smx8 +  Smx5+ Smx6+Smx7 ; DNy<=Sx4+
    Sx8 +   Sx7+  Sx5+ Sx6; end

else if(round_2y ==6)  begin Smx<=Smx5+ Smx9 +  Smx6+ Smx7+Smx8 ; DNy<=Sy5+
    Sx9 + Sx7+ Sx8+   Sx6; end

else if(round_2y >=7)  begin Smx<=  Smx7+Smx8+ Smx9; DNy<= Sx7+ Sx8+ Sx9 ; end


end

if(bs==51)
begin
if(bs>0) begin bs<=52;end
half<= (DN>>1);
one_fourth<=      (DN>>2);
tDN<=DN<<1; one_eighth<=DN>>3;
halfy<= (DNy>>1);
one_fourthy<=      (DNy>>2);
tDNy<=DNy<<1; one_eighthy<=DNy>>3;
end


if(bs==52)
begin
if(bs>0) begin bs<=12;end
thr_fourth<=DN+half;tone_fourth<=  half+one_fourth;tpDN<=tDN+half;
thr_fourthy<=DNy+halfy;tone_fourthy<=  halfy+one_fourthy;tpDNy<=tDNy+halfy;
end


if(bs==12)
begin
//aDN<=a*DN;bDN<=b*DN;
```

```
bs<=29;
ta1=a&1; ta2=a&2; ta3=a&4; ta4=a&8;ta5=a&16; ta6=a&32;ta7=a&64;
 ta8=a&128;ta9=a&256; ta10=a&512;
tb1=b&1; tb2=b&2; tb3=b&4; tb4=b&8;tb5=b&16; tb6=b&32;tb7=b&64;
 tb8=b&128;tb9=b&256; tb10=b&512;
// $display("bs 12  %d %d %d b %d %d %d ",
 x_max,tx_max,tx_max2,y_max,ty_max,ty_max2);
end



if(bs==29)begin
if(bs>0)begin bs<=30;end
a1 <=ta1 ; a2 <= ta2; a3 <=ta3; a4 <=ta4;a5<=ta5;
a6 <=ta6;a7 <=ta7; a8 <=ta8;a9 <=ta9; a10 <=ta10;
b1 <=tb1 ; b2 <= tb2; b3 <=tb3; b4 <=tb4;b5<=tb5;
 b6 <=tb6;b7 <=tb7; b8 <=tb8;b9 <=tb9; b10 <=tb10;

end


if(bs==30)begin
if(bs>0)begin bs<=31;end
if( a1==1)begin   ha1<= DN;end
if( a2==2)begin   ha2<= DN<<1;end
if( a3==4)begin   ha3<= DN<<2;end
if( a4==8)begin   ha4<= DN<<3;end
if( a5==16)begin   ha5<= DN<<4;end
if( a6==32)begin   ha6<= DN<<5;end
if( a7==64)begin   ha7<= DN<<6;end
if( a8==128)begin   ha8<= DN<<7;end
if( a9==256)begin   ha9<= DN<<8;end

if( b1==1)begin   hb1<= DNy;end
if( b2==2)begin   hb2<= DNy<<1;end
if( b3==4)begin   hb3<= DNy<<2;end
if( b4==8)begin   hb4<= DNy<<3;end
if( b5==16)begin   hb5<= DNy<<4;end
if( b6==32)begin   hb6<= DNy<<5;end
if( b7==64)begin   hb7<= DNy<<6;end
if( b8==128)begin   hb8<= DNy<<7;end
if( b9==256)begin   hb9<= DNy<<8;end
//$display("bs 30 %d %d", DN,hb9);
end

if(bs==31)begin
if(bs>0)begin bs<=13;end
aDN<= ha1+ ha2+ ha3+ ha4+ ha5+ ha6+ ha7+ ha8+ ha9+ ha10 ;
bDN<= hb1+ hb2+ hb3+ hb4+ hb5+ hb6+ hb7+ hb8+ hb9+ hb10 ;
//$display("bs 31 %d %d", ha2,hb9);
end

if(bs==13)
begin
//$display("adn %d  also %d bdn %d also %d  anndd DN %d",aDN,a*DN,bDN,b*DN,DN)
    ;


if(Smy>(aDN))
begin
remx<=Smy-(aDN);
posx<=1;
end
else
begin
```

```
remx<=(aDN)-Smy;
posx<=2;
end

if(Smx>(bDN))
begin
remy<=Smx-(bDN);
posy<=1;
end
else
begin
remy<=(bDN)-Smx;
posy<=2;
end

if(bs>0) begin bs<=16;end

end

if(bs==16)begin

if(remx<=one_eighth) begin
decx<=1; end

else if( remx<half)begin
decx<=8; end

else if(remx >= half && remx<thr_fourth)begin
decx<=15; end
else begin
decx<=0;

end

if(remy<=one_eighth) begin
decy<=1; end

else if(remy<half)begin
decy<=8; end

else if(remy >= half && remy<thr_fourthy)begin
decy<=15; end

else begin

decy<=0;

end

if(bs>0)begin bs<=15; end

end

if(bs==15)begin
// $display(" %d %d are the decs , %d %d are int, %d %d are +-",decx,decy,a,b,
    posx,posy);
//$display("at 14 ,a %d remx %d posx %d decx %d ",a, remx,posx,decx);
//$display ("b %d remy %d posy %d decy %d ",b, remy,posy,decy);
//$display("[%d %d %d %d ],",Smy,Smx,DN,DNy);
//$display( "after [%d %d %d ,%d %d %d],%d %d" ,a,posx,decx,b,posy,decy,a-(c
    -2),b-r);

sx<=0;sy<=0;Smx<=0;Smy<=0;DN<=0; DNy<=0;roi_cou<=0; addc<=0; bDN<=0;aDN<=0;
isy2can<=0 ;isy3can <=0 ;isy4can <=0 ;isy5can <=0 ;isy6can <=0 ;isy7can <=0
    ;isy8can <=0 ;
isx2can <=0 ;isx3can <=0 ;isx4can <=0 ;isx5can <=0 ;isx6can <=0 ;isx7can <=0 ;
     isx8can <=0 ;
Sx123 <=0; Sx234 <=0; Sx345 <=0; Sx456 <=0; Sx567 <=0; Sx678 <=0; Sx789 <=0;
```

```
Sy123 <=0; Sy234 <=0; Sy345 <=0; Sy456 <=0; Sy567 <=0; Sy678 <=0; Sy789 <=0;
ha1<=0;  ha2<=0;  ha3<=0;  ha4<=0;  ha5<=0;  ha6<=0;  ha7<=0;  ha8<=0;  ha9
    <=0;  ha10<=0;
hb1<=0;  hb2<=0;  hb3<=0;  hb4<=0;  hb5<=0;  hb6<=0;  hb7<=0;  hb8<=0;  hb9
    <=0;  hb10<=0;
stepa<=0;stepb<=0;   tx_max2<=0;ty_max2<=0; decx<=0;decy<=0;
bs<=17;
end


if(bs==17)begin
// $display(" DONEEEEEE %d %d %d %d %d %d",atleast1,mstarc,buff_count,
    fifo_count,starc,bs);

if ( buff_count<fifo_count) begin   atleast1<=atleast1+1;buff_count<=
    buff_count+1;bs<=42;end
else begin bs<=49;all_over<=1;  end
end
end
end
always@(posedge clk)begin
if(reset==0)begin clk_count <=0;end
else begin
if(all_over==0) begin clk_count<=clk_count+1;end
if(all_over==1) begin $display("count %d",clk_count); clk_count<=0;$finish ;
    end

end
end
assign led1=   max[3] || max[2] || all_over[0]  ;
assign led2=   max[1] || max[5]  || all_over[1]    ;

endmodule
```