Real-Time Implementation of SURF Algorithm on FPGA Platform

by

Sichao Zhu

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

in

Electrical and Computer Engineering

by

April 2014

APPROVED:

[illegible]

Professor Xinming Huang, Major Thesis Advisor

[illegible]

Professor Lifeng Lai

[illegible]

Professor Taskin Padir

Abstract

Too many traffic accidents are caused by drivers' failure of noticing buildings, traffic sign and other objects. Video based scene or object detection which can easily enhance drivers' judgment performance by automatically detecting scene and signs. Two of the recent popular video detection algorithms are Background Differentiation and Feature based object detection. The background Differentiation is an efficient and fast way of observing a moving object in a relatively stationary background, which makes it easy to be implemented on a mobile platform and performs a swift processing speed. The Feature based scene detection such like the Speeded Up Robust Feature (SURF), is an appropriate way of detecting specific scene with accuracy and rotation and illumination invariance. By comparison, SURF computational expense is much higher, which remains the algorithm limited in real time mobile platform. In this thesis, I present two real time tracking algorithms, Differentiation based and SURF based scene detection systems on FPGA platform. The proposed hardware designs are able to process video of 800*600 resolution at 60 frames per second, the video clock rate is 40 MHz.

Acknowledgements

I would like to express my gratitude to my advisor, Professor Xinming Huang. He gave me the opportunity to do the research and guided me in my research.

Thanks to the authors of Speeded up Robust Feature Herbert Bay, Andreas Ess, Tinne Tuythlaars and Luc Van Gool for sharing the ideas of SURF with me. This thesis would not exist without the SURF algorithm.

Thanks to the author of Opensurf matlab code author Dirk-Jan Kroon. His matlab code inspired me a lot.

Last but not least, I'd like to thank all my friends and especially my project partner Jin Zhao for giving me the courage and support

Contents

1 Introduction **1**

2 Differentiation method **3**

2.1 AVNET DVI I/O board video input output signal standard and input video clock rate. 3

2.2 Video differentiation procedure 5

2.3 Resource utilization 8

3 Speeded Up Robust Feature (SURF) overview **10**

3.1 Integral image 11

3.2 Interest point localization 12

3.3 Descriptor extraction 13

3.4 Descriptor matching 14

4 FPGA implementation of SURF **16**

4.1 Overall system architecture 17

4.2 Integral image module 19

4.3 Interest points detector. 21

4.4 Interest point descriptor 23

4.4.1 Haar wavelet transform 24

4.4.2 Wavelet response reconstructor 26

4.4.3 Gaussian mask LUT and Gaussian weight module 29

4.4.4 Gaussian scaling and length of region 33

4.5 Descriptor matching module 35

4.6 Platform and results 38

5 Conclusions **40**

List of Figures

2.1 The video standard of AVNET DVI I/O board 4

2.2 The main structure of differentiation tracking 5

2.3 Ping Pong buffer structure 7

3.1 Integral image function 11

3.2 Hessian mask 12

3.3 Haar wavelet filters 14

3.4 Descriptor mathing procedure 15

4.1 Overview of SURF on FPGA 18

4.2 Integral image architecture 19

4.3 Intergal image procedure 20

4.4 Integral image prcedure 2 21

4.5 Hessian response module 22

4.6 Interest pointer descriptor main architecture 24

4.7 Haar wavelet procedure 25

4.8 Wavelet response reconstructor 27

4.9 Haar wavelet response outputs 28

4.10 Gaussian LUT 31

4.11 Multipliers multiplexing structure 33

4.12 Gaussian scale mask 35
4.13 Matching module 37
4.14 Matching result 37
4.15 KC705 platform 38
4.16 AVNET DVI FMC module 38

List of Tables

2.1 Device utilization summary 8

2.2 Device utilization summary with DDR3 MIG 8

4.1 Gaussian LUT 30

4.2 Resource utilities in the design 39

Chapter 1

Introduction

Video-based traffic detection technologies are becoming requisite to an automatic-drive car or a driver assistance system. The features of monitoring surrounding traffic conditions and detecting blind spots for drivers enhance the driving safety and driving experience for almost every possible driver. The current optional video-based driver assistance component in high-end vehicle is the revers image system which can help driver parks the car correctly and safely into parking place. The video base algorithms which can be applied on those technologies are researched and achieved marvelous results in this area. Driver assistance algorithms suck like object detection, optical flow, Kalman filter, shape based detection and Hough transform. In addition, researchers have been spent the same, if not more, attention to the real-time application on embedded platform with limited resource and power.

In this thesis, we focus on object detection using Differentiation in stationary background and feather detection algorithm Speeded up robust feature. The Differentiation in stationary background is useful in video object detection in invariant position monitoring such like surveillance camera which keeps the same position to detect movements in front of it. The surrounding environment is relatively stationary for the moving objects, which makes it a algorithm of practical use. Also this algorithm can be processed fast in real time, and less resource taken than other complex algorithm in FPGA. We also would like to discuss the Speeded up Robust Feature to be used on object detection in FPGA platform. SURF is an accurate and higher rate of correct recognition video algorithm. Even so, like other feature based algorithms (SIFT or FREAK) SURF takes long time in feature extraction, descriptor calculation and library matching in low frame rate. In order to improve the performance of SURF, we plan to apply it on FPGA platform due to the natural properties image processing – parallelism.

FPGA structure makes the video processing possible to parallelize the calculation. The hardware structure divides the input video frame into small parts and run in parallel pipelines independently and efficiently. Instead of using the expensive CPU resource in PC platform, FPGA accelerate the process massively. We choose a Xilinx Kintex 7 FPGA development board and AVNET DVI I/O FMC video board. Brief: (remember to fill them) Chapter 2 presents the video moving object tracking using differentiation method under a relatively stationary background and the algorithm implementation on FPGA platform. Chapter 3 introduces the Speeded Up Robust Feature algorithm and the function blocks in SURF. Chapter 4 explains the details of Speeded Up Robust Feature implementation on FPGA platform. Chapter 5 concludes the achievements and possible work in the future.

Chapter 2

Differentiation method

2.1 AVNET DVI I/O board video input output signal standard and input video clock rate.

Video standard is the key of controlling a video playing. There are multiple kinds of video standard; most of them are divided into control signals and data signals. According to human eyes persistence of vision, fast switching still images give us the impression of motion. Usually films playing rate is 25 frames per second, our monitor refresh rate is about 60 frames per second. Due to the signal width, we cannot transmit a whole frame to the monitor at the same time, so we transmit one whole pixel to the monitor one time (one clock cycle), which lower the expense of signal width. In the meantime, the control signals are also pushed to the monitor aside with the pixel signals. Usually the control signals are vertical synchronization signal, which is for frame switch, horizontal synchronization, which is for line switch, and data enable, which is to help select the valid pixels. There might be a little difference between video standards, but the logic is the same.

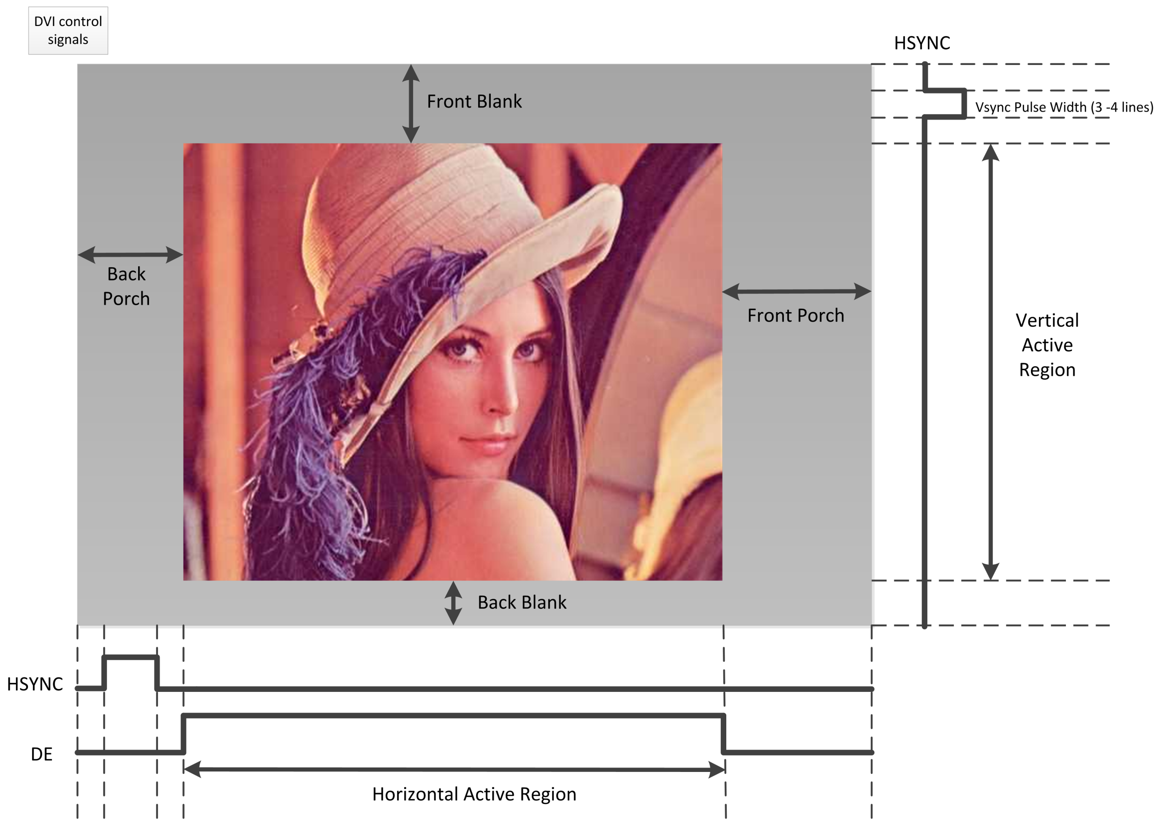

Figure 2.1: The video standard of AVNET DVI I/O board

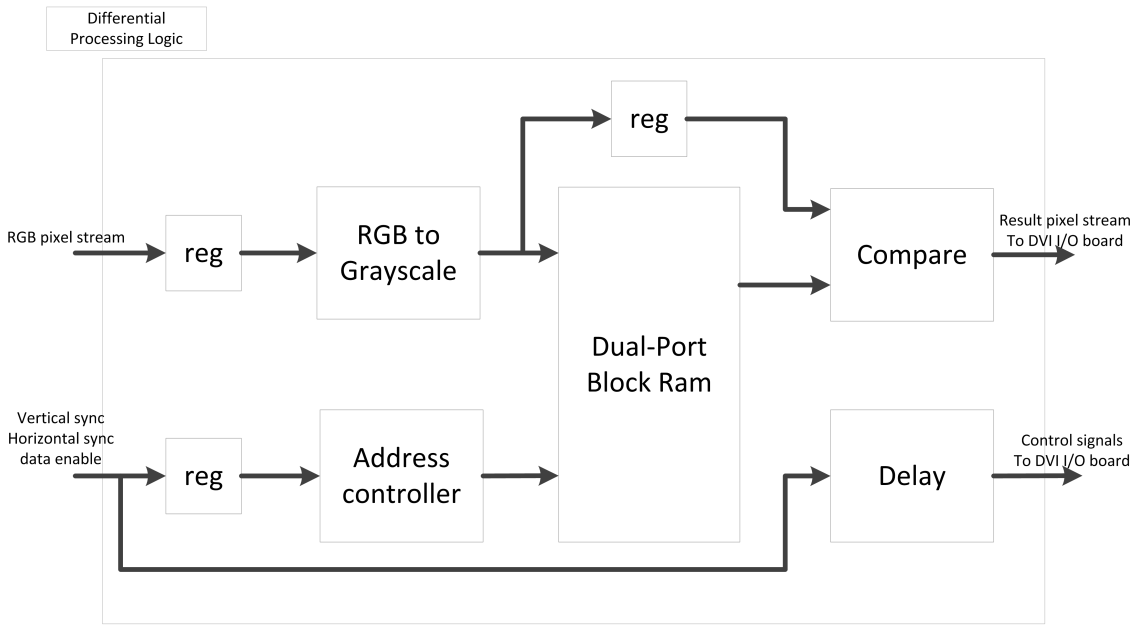
Differential Processing Logic
RGB pixel stream
reg
RGB to Grayscale
reg
Dual-Port Block Ram
Compare
Result pixel stream To DVI I/O board
Vertical sync Horizontal sync data enable
reg
Address controller
Delay
Control signals To DVI I/O board

Figure 2.2: The main structure of differentiation tracking

2.2 Video differentiation procedure

Video Differential object tracking is based on the relatively stationary background and the moving object that can change parts of background pixel value. In this method, video frames are buffered into FPGA local memory and compares with each other to find the difference. This method is founded on the background pixels value time invariance while the moving objects position in current frame is different from the position in previous frame. With the previous 800 pixels width by 600 pixels height resolution frames be buffered in Local memory like block ram or DDR, one frame size should be 800*600 pixels, each pixel is 24 bits since the pixel is RGB standard (8 bits + 8 bits + 8 bits). We can calculate the difference between the current frame and previous frame by storing one frame and comparing with the previous frame at the same time. Fig 2.1 shows how the calculation being processed.

For every clock cycle, one pixel is pushed into the hardware structure through DVI I/O board, aside with the video control signals. The first processing block is the RGB to gray scale transfer block, this block use three efficient multipliers transfer the RGB signal (24 bits) to gray scale signal (8 bits), which saves a large amount of memory consuming. The RGB to Gray scale data transfer is calculated according to:

$$Grayscale = R \times 0.2989 + G \times 0.5870 + B \times 0.114 \tag{2.1}$$

We use fixed point fraction instead of double fraction due to the calculation complexity. The solution is to use FPGA block ram resource to store the video frame. According to the video stream property, we need to store and read the pixel at the same clock cycle, a dual-port RAM is needed to read and store pixels at the same time. On the RAM address controlling block, the RAM is divided into two independent parts in order to store action and read action will not affect each other. In the first buffering period, input data is stored in part 1, in the second buffering period, input data is stored in part 2 based on the store controlling block address switching meanwhile output the data in part1. In the third buffering period, input data is stored in part 1 again meanwhile output the data in part2. That is the basic loop for the buffering procedure; two parts never affect each other. It is called Ping-Pong buffering technology. The two buffers are treated as a unit. The buffered data can be popped out without pausing. Ping Pong is often used in pipelined arithmetic operation to complete buffering and processing data seamlessly.

The block ram resource in KC705 FPGA is 16020 Kbit according to the datasheet while the whole frame is 800*600*8 bits = 3840 Kbits, since there are two parts in the buffer, the total necessary memory is 7680 Kbits, KC705 block ram size is large enough to build a ping pong buffer structure.

The compare block, is the block that detects the moving objects. Since object

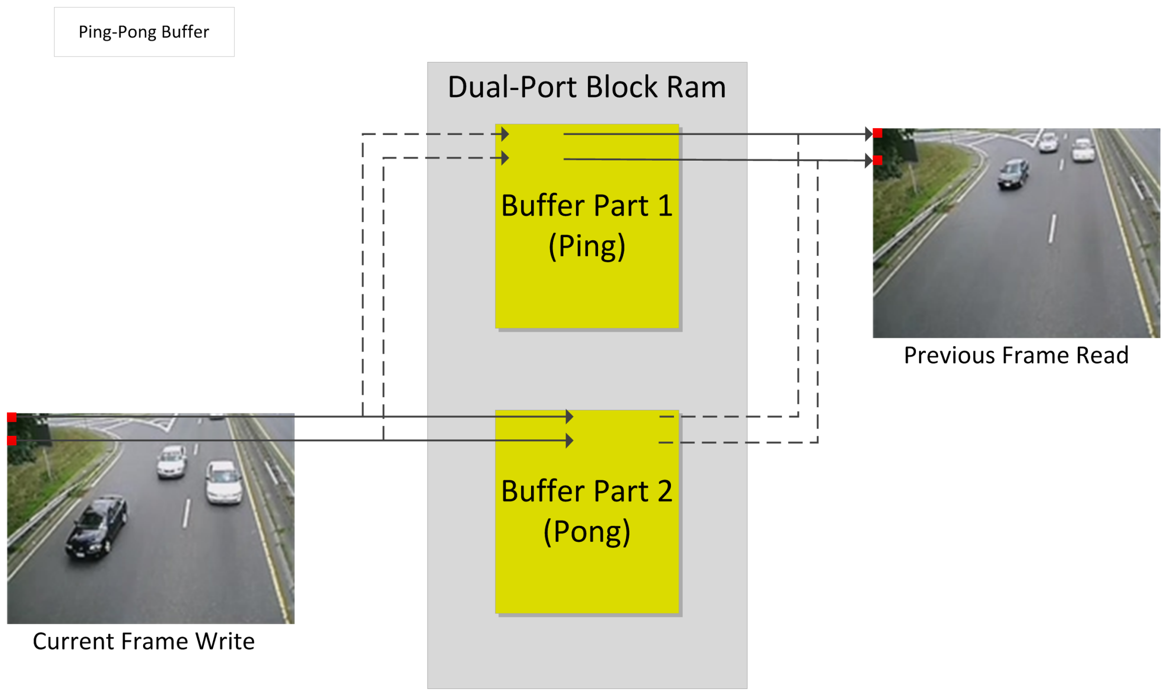

Figure 2.3: Ping Pong buffer structure

position changing result the pixel value change. Due to the address control block, the popped pixel from the buffer is exactly the same position as the current video stream pixel. The buffered frame, which plays the role as the background, we need to find the difference between the current frame and the previous frame by comparing them one pixel by one pixel. The compare block is design for this, if the difference of two pixels meets the threshold, this position will be highlighted to declare that moving object is detected and this position is part of the moving object.

Considering the limitation of FPGA block ram size, we choose to transfer the RGB standard to gray which makes the final display a gray scale video. In order to improve the display experience, we introduced the on board DDR3 as data buffer. The KC705 development board contains a DDR3 SODIMM Memory (1 GB) which is large enough for buffering the whole frame. To drive the on-board DDR3, we choose to use Xilinx IP Memory Interface Generator (MIG) to create a logic connection with DDR3. Also we use two FIFOs as the pre-MIG FIFO and post-MIG FIFO due to the unpredictable DDR Data output. The data procedure is the same only replace the Dual-Port Block RAM with DDR3.

The total design is a video processing system in three stages pipeline driven by video clock, which needs a clock global buffer, timing delay is 4 clock periods include the signal syncing registers which is about $4/(40*10\hat{}6) = 100$ ns, makes it a real time processing system.

2.3 Resource utilization

Here is the utilization summary of Kintex7 FPGA of this design:

Resources	Used	Available	Utilization
Number of Slices Registers	179	407600	≤1%
Number of Slice LUTs	283	203800	≤1%
Number of bonded IOBs	58	500	30%
Number of Block RAM/FIFO	257	445	57%
Number of BUFG/BUFGCTRLs	3	32	9%
Number of DSP48E1s	3	840	≤1%

Table 2.1: Device utilization summary

Here is the utilization summary of Kintex7 FPGA with DDR3 as a data buffer:

Resources	Used	Available	Utilization
Number of Slices Registers	6922	407600	1%
Number of Slice LUTs	10525	203800	51%
Number of bonded IOBs	171	500	34%
Number of Block RAM/FIFO	17	445	3%
Number of BUFG/BUFGCTRLs	3	32	9%
Number of DSP48E1s	6	840	≤1%

Table 2.2: Device utilization summary with DDR3 MIG

We can see from the two tables that the increased Slices Registers and Slice LUTs illustrated that MIG block used large amount of FPGA resource, while the Block RAM/FIFO utilization is lower than the previous since we remove the Block RAM and use DDR3 instead.

2.4 Summary

The Differentiation Object tracking is efficient in a stationary background situation, while in real world, this is far from enough simply using only this method. More situation should be considered in to object detection such like video signal noise.

Chapter 3

Speeded Up Robust Feature (SURF) overview

Speeded Up Robust Feature (SURF) is one of the best feature based algorithms of the past few years and has applied widely in computer vision realm. It is a scale and rotation-invariant algorithm with relatively high acceleration comparing with other feature based algorithms such like SIFT (Scale-invariant feature transform) since the introduction of Integral image technology, highly boost the processing speed. Even so, SURF is still computationally expensive, often results in a very low frame rate when applying SURF in video processing. After analyzing each step of SURF, we decided to use Field Programmable Gate Array (FPGA) due to SURF's parallelism.

SURF algorithm consists of three main parts: Integral image generation, interest point localization, and interest point description generation, with object detection, one more step descriptor matching is also needed.

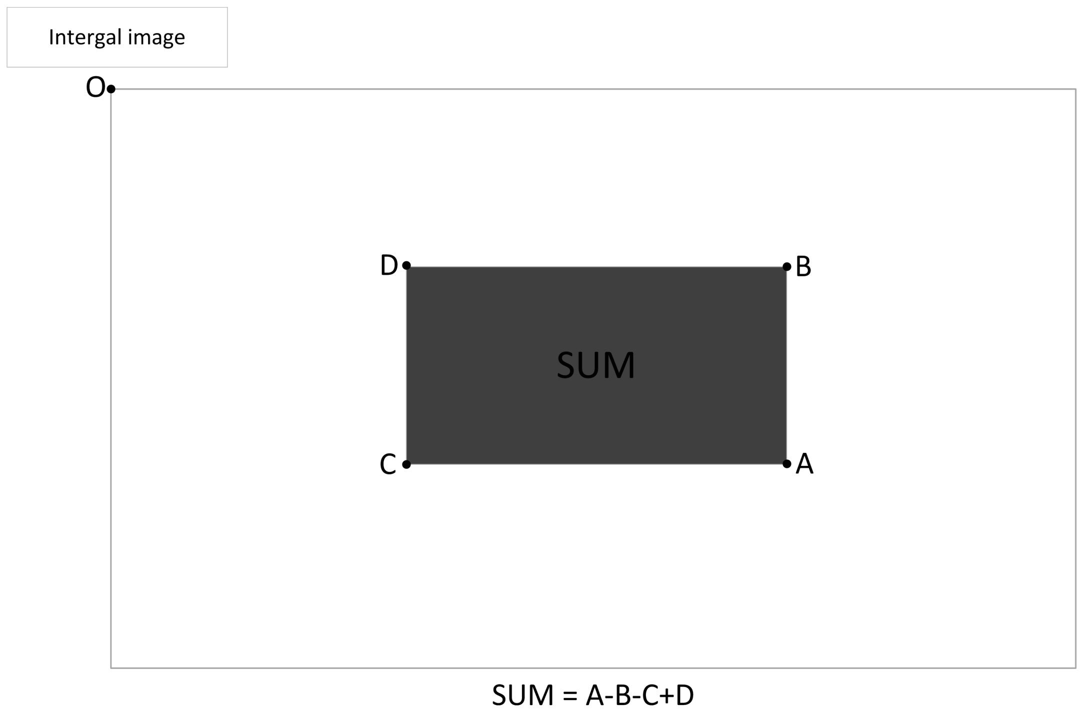

Figure 3.1: Integral image function

3.1 Integral image

Integral image is one of the main advantages of SURF, it allows for fast computation of box type 2D filters.

$$I_{\sum}(X) = \sum_{i=0}^{i\leq x} \sum_{j=0}^{j\leq y} I(i,j) \tag{3.1}$$

Integral image $I_{\sum}(X)$ in $X = (x, y)$ represents the sum of all the pixels at the left and top side of X as in (2). Integral image is used in both the subsequence interest point detection and descriptor calculation. It only takes 3 additions/subtractions to get the sum of the pixels inside the rectangle after integral image calculation (see figure 3.1) which boosts the processing speed. The sum of the rectangle is denoted simply as $SUM = A - B - C + D$. The other benefit is that the sum calculation time is independent of the box size, which is a good news for FPGA pipeline design.

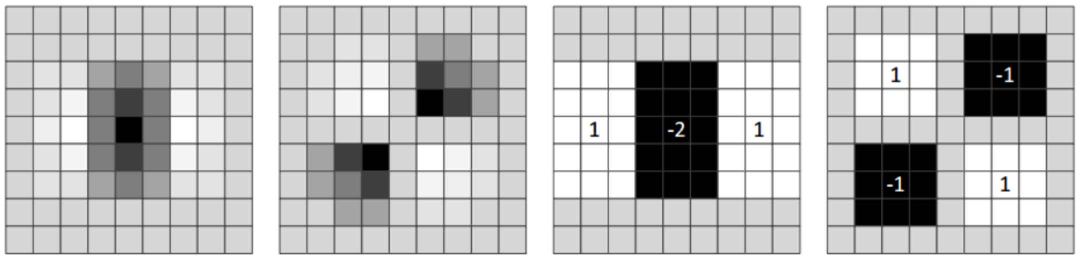

Figure 3.2: Hessian mask

3.2 Interest point localization

SURF detector locates features based on the Hessian matrix because of its good performance and accuracy. The original definition of Hessian matrix is

$$H(x,\sigma) = \begin{bmatrix} L_{xx}(x,\sigma) & L_{xy}(x,\sigma) \\ L_{xy}(x,\sigma) & L_{yy}(x,\sigma) \end{bmatrix} \tag{3.2}$$

Where x is the point location (x, y) in an image I, $L_{xx}(x,\sigma)$ denotes the convolution of Gaussian second order derivative in x direction with pixel x at scale σ, and similarly for $L_{xy}(x,\sigma)$, $L_{xy}(x,\sigma)$ and $L_{yy}(x,\sigma)$. To achieve the scale invariance, SURF used approximated box filters for the Hessian matrix, in Figure 3.2 is the box filters of different sizes:

Box filters with the integral image are used to calculate the second order Gaussian partial derivation, and avoid less computation. Denoting the blob responses by D_{xx}, D_{yy}, and D_{xy}, the determinant of the approximated Hessian matrix in SURF is approximated as following function:

$$Det(Happrox) = D_{xx} \times D_{yy} – (0.9 \times Dxy)^2$$

Where 0.9 is the relative weight of the filter response, which is to balance the Hessian determinant. Box filters of different sizes construct the scale space, which is divided into octaves. The local maxima larger than a pre-defined threshold in image and scale space are selected as the interest point candidates.

3.3 Descriptor extraction

SURF builds descriptor around the neighborhood of each interest point based on sum of Haar wavelet responses. First, a square region of $20s$ by $20s$ centered around the interest point constructed along the dominant direction. The dominant directions of interest points are set to be upright to make the calculation simpler. The region is divided into 4×4 square smaller sub regions with each window size $5s$ by $5s$. For each of the sub region, Haar wavelet responses are computed at 5×5 regularly distributed sample points. We use d_x to represent horizontal direction response and d_y to represent y vertical response, the sum of the absolute values of the response $|d_x|$ and $|d_y|$ also need to be extracted. These responses are then first weighted by a Gaussian function ($\sigma = 3.3$) centered at interest point. Each Gaussian Weighted sub-region generates a $4 - D$ descriptor vectors

$$v = (\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|) \tag{3.3}$$

All sub-regions are then concatenated into one vector of length 64, also a scale factor is needed to turn the descriptor into a unit vector, which is the final descriptor vector for this interest point.

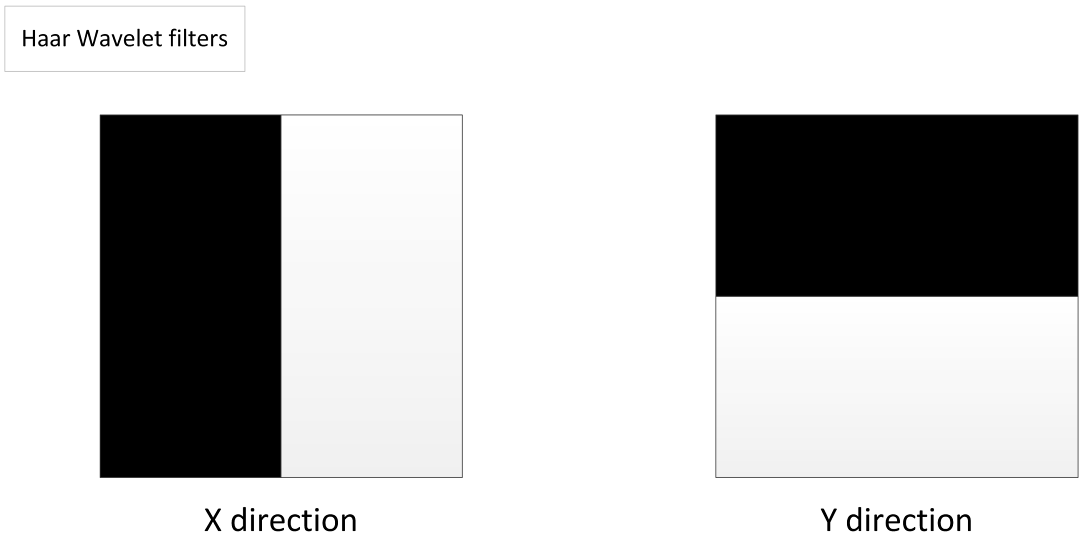

Figure 3.3: Haar wavelet filters

3.4 Descriptor matching

SURF descriptor vectors are all length of 64. By getting all descriptor in one image, next step is the application of descriptor. Detection using SURF is for detecting specific items, which means the item to tracking should be acquired before tracking. The matching step is actually processing between two images' descriptor vectors, to find the difference of each vectors. The item descriptor vectors are pre-extracted from item image using the same SURF procedure and stored as a library. This makes the video tracking calculation simpler than looping calculate the same items for each matching.

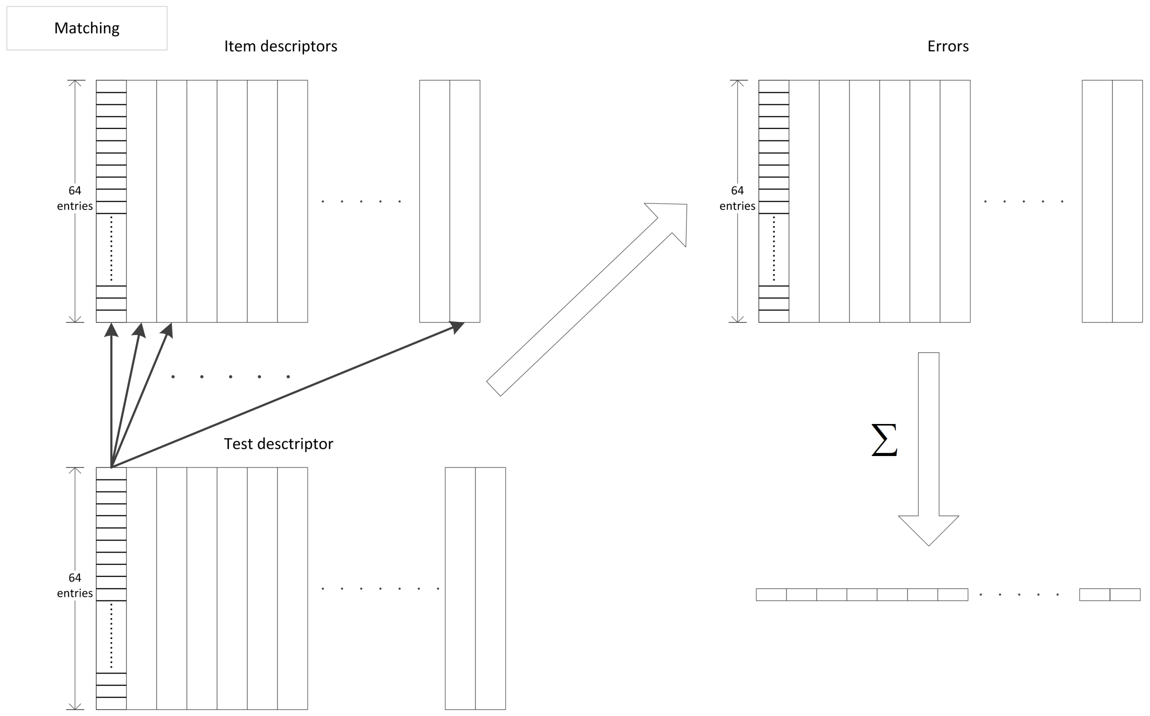

Figure 3.4: Descriptor mathing procedure

We call the descriptor vectors from video the test descriptors, and the descriptor vectors from the pre-stored library the item descriptors. Each of the test descriptors compares with item descriptor to find the differences of each entry of the vector. We still need a result to represent the difference between two descriptors. The sum of total 64 entries is the best way to illustrate the result. We use distance to describe the result, the smaller the distance, the slighter difference between two descriptors. Usually we select the smallest difference 30∼50 points to determine if the item is successfully tracked in the current frame.

Chapter 4

FPGA implementation of SURF

The surf introduced in chapter 3 is a robust, accurate and boosted object identification algorithm to extract and generate descriptors of an image or a video frame. Descriptors can be applied not only in object tracking but in image correction and video stabilization. However, according to its computation complexity and high data access frequency, SURF is difficult to be implemented as a real-time architecture. On a quad-core 3.8 GHz processor, SURF simulation in Matlab usually takes 4.38 seconds to detect all 284 interest points, extract their descriptors from an image of 800*600, and finish the matching step to track the object. An FPGA architecture was proposed and its processing rate of 10 fps on a Xilinx XC5VFX70 FPGA with PowerPC-440 CPU. This is a solution of hardware/software cooperation. The Integral image generator and fast Hessian generator implemented by FPGA hardware while the rest of interest points localization is implemented in PowerPC software. However, the frame rate is still unsatisfactory for real-time systems like traffic object detection.

This chapter describes a hardware-only architecture to implement the real-time SURF algorithm for 800*600 video at frame rate 60 fps. Each function part will be described in detail.

4.1 Overall system architecture

As we introduced before, SURF is an effective and accurate algorithm that can be implemented in hardware architecture, while we still have to do several approximations for the hardware features because of the hardware pipeline structure and resource limitation. Primarily, all the floating point calculations are replaced with fixed point calculation not only because of the processing speed but also about the floating point unit resource taken. An FPGA-only architecture does not allow too many parallel floating point units exits. Floating point to fixed point data conversion is carefully conducted with tiny loss of precision. Secondary, Objects usually occupies small part of a frame, makes the first two octaves as the interest points frequently being detected octaves. Based on this and the limitation of hardware resource, Fast Hessian responses and the interest point detection are computes only at the first two octaves. Thirdly, in order to achieve a higher identification rate and to compensate the accuracy loss, we built a modified descriptor extractor using larger neighbors from 20×20 size matrix to 24×24 size matrix. And overlapping sub-region strategy from 5×5 size matrix to 9×9 size matrix to extract more details from the interest point neighbor area. Finally, the dominant direction of each interest point is set to upright to simplify the calculation, which makes the rotation angle less than 15.

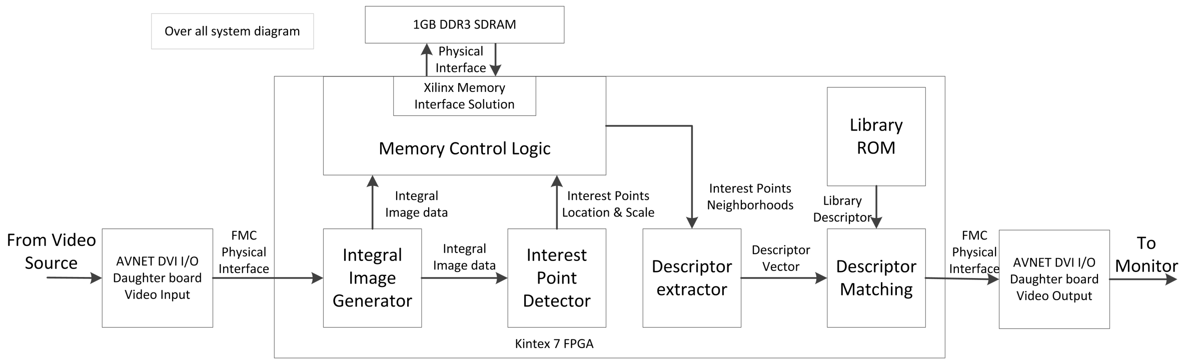

Figure 4.1: Overview of SURF on FPGA

The Overview of FPGA-based SURF architecture is demonstrated in Figure 4.1, consists of 5 main modules, Integral Image generator, Interest Point Detector, Descriptor and Memory Control Logic and Descriptor Matching Module (with Library ROM). Integral Image Generator module accepts RGB pixel signal from AVNET DVI I/O Module and convert RGB to gray scale data first. The pixel pushing clock rate is 40MHz with the video control signals.

Integral image switches the gray scale data into integral image pixel stream and transfers them to Interest Point Detector module and Memory Control Logic. The Interest Point Detector module is used to search all the Fast Hessian Maxima then exclude the pseudo extremes. The Memory Control Logic is designed for communicate with SDRAM and enhance the performance of Xilinx Memory interface Solution. DDR3 SDRAM stores the integral images and transfer Interest points Neighbors to Descriptor Extractor module according to the location and scales from Interest Point Detector module. The Descriptor Matching Module accepts Descriptors from both Descriptor Extractor module and Library ROM to determine the searching results and output the results to AVNET DVI I/O board video output. The details of each module are demonstrated below.

4.2 Integral image module

Integral image generation module is the entry gate of the whole system. The modules are shown in Figure 4.2. The input video stream data type is RGB format, SURF algorithm requires gray scale instead. The first step is to switch RGB pixel into gray scale pixel from 24 bit width to 8 bit width. The RGB to Gray Scale module is responsible for the translation. The row and column location is generated from video control signals vsync, hsync and de. The location information is also treated as the integral pixel's DDR input address.

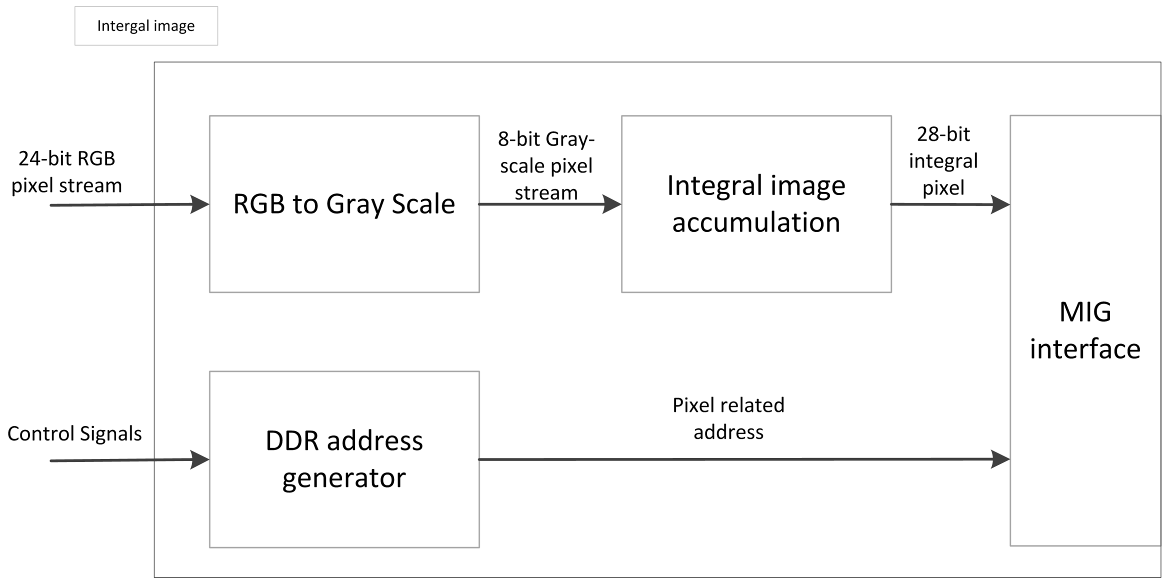

Figure 4.2: Integral image architecture

From the definition of integral image, any location (x, y), the value itself is the sum of all the pixel values above and to the very left coordinate including the original pixel value. As show in the picture, the sum of Integral image pixel A equals to the sum of all pixels at the gray area. the accumulation process is illustrated in the following figure.

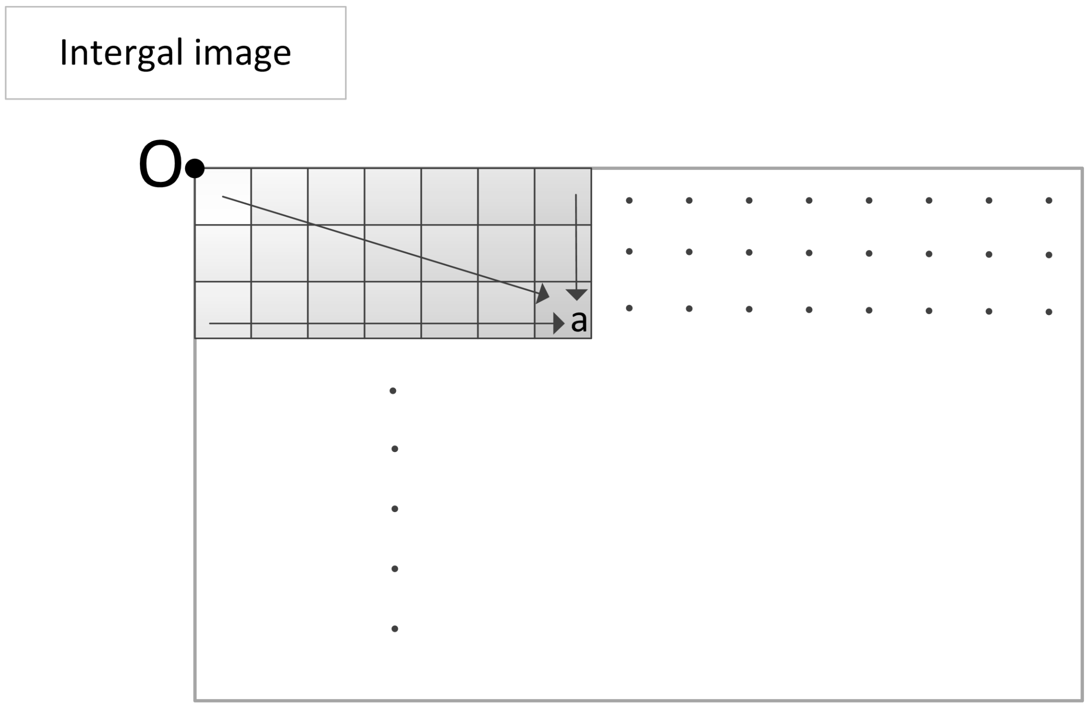

Figure 4.3: Intergal image procedure

Based on the video stream input method, we can simplify the calculation procedure. Suppose the current location is a in Figure 4.4, integral pixel b, c, d are pre-calculated integral pixels at a's upper left, upper and left. A's value can be quickly acquired in fixed steps from $I_a = I_b – I_d – I_c + Pixel_a$. According to this, two lines integral pixels, upper line and current line, need to be buffered in FPGA to help further calculation. The integral pixel DDR storing operation is processing at the same time of accumulation to reduce register usage.

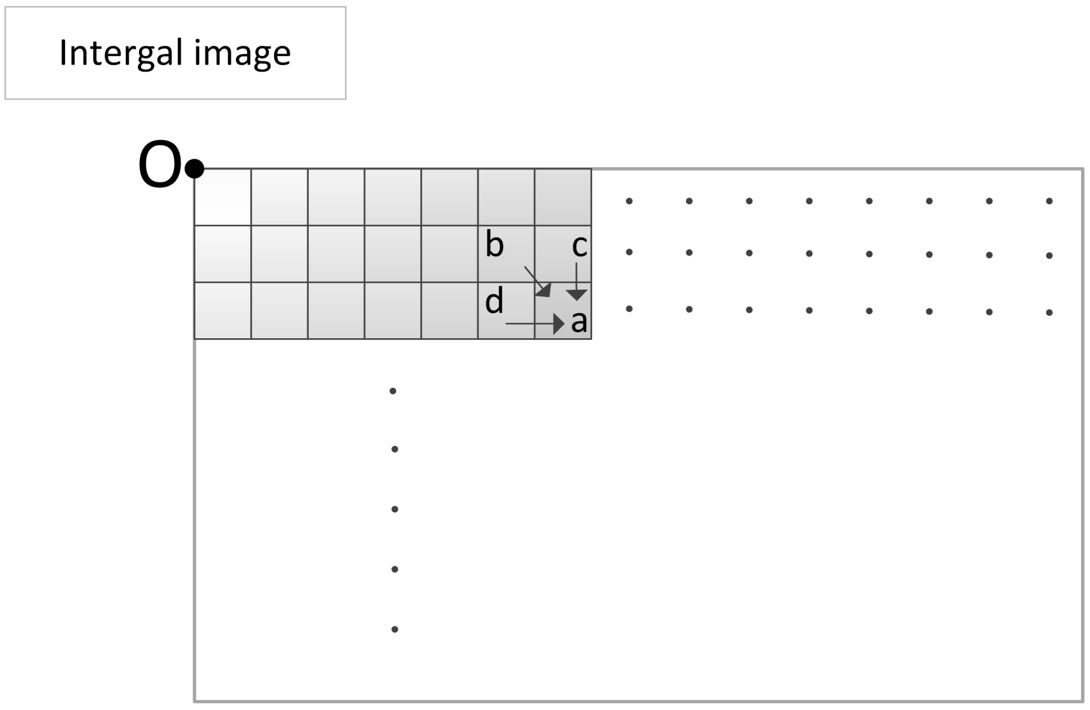

Figure 4.4: Integral image prcedure 2

The video of integral image has the same layout as the input video except the RGB pixels are replaced with integral pixels from gray scale value. The whole frame buffered in DDR to be used in later interest points detection and descriptor generation.

4.3 Interest points detector.

This module is designed for calculate fast-Hessian responses of all sample points at multi-scale and find the local maxima's location and scale space as interest point candidates. This part is done by my partner Jin Zhao. I will briefly introduce the architectures.

The interest points detector is divided into 3 blocks: (1) Build_Hessian_Response, (2) Is_Extreme, (3) Interpolate_Extremum. The diagram is shown in Figure 4.5.

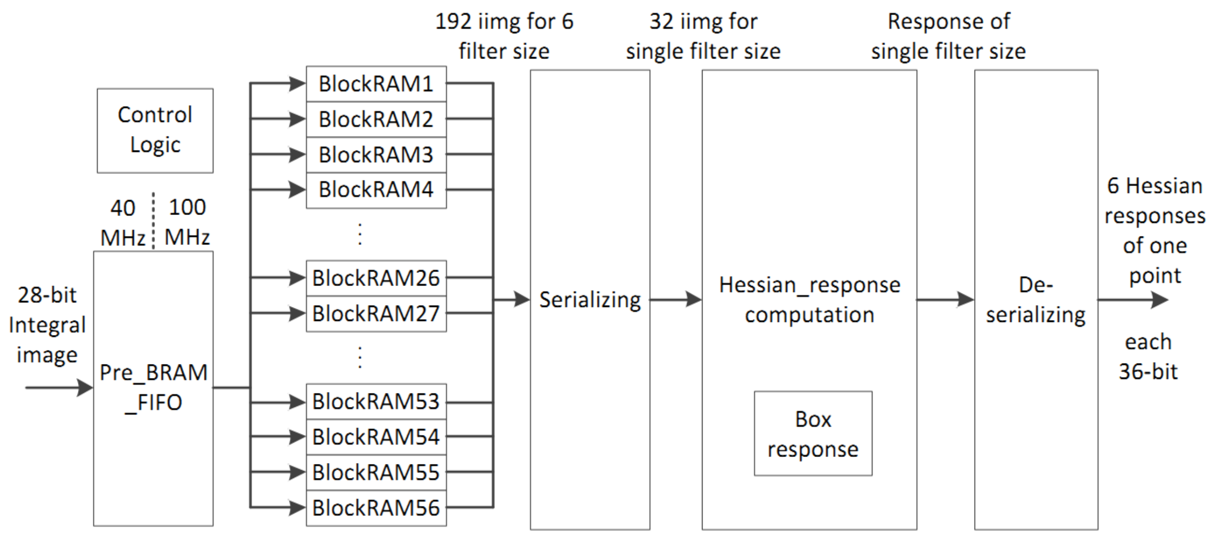

Figure 4.5: Hessian response module

Build_Hessian_Response module aims at calculating Hessian matrix response. The definition of Hessian matrix is introduced in 3.1.2. Also the Hessian matrix response is defined in 3.1.2. Due to the resource and processing speed, 2 octaves rather than 5 octaves are used in this module. This block uses independent accessed 56 block RAMs, depth of 800 to buffer 56 integral image lines. Totally 600 lines of integral image are stored into the Block RAMs for the concurrent access of incoming pixels and Hessian response. A FIFO at 40 MHz incoming data and 100MHz output data is used to buffer the integral image. After calculation, 6 Hessian determinants are deserialzied and sent to Is_Extreme module in parallel.

Is_Extreme module is aimed at finding local maxima of Hessian-matrix determinants as interest points candidates. Local maxima are detected on layers from different filters. For each scale, Is_Extreme uses 4 block RAMs of depth 400 or 200 to store current Hessian-matrix determinant line and line buffering. Is_Extreme module also calculate the first and second order derivatives of Hessian-matrix responses. The derivatives will stored into a FIFO with their coordinates and scales if it passes the threshold check.

Interpolate_Extremum module is to performa the matrix operation shown in the following equation:

$$O = -H\backslash D = -\begin{bmatrix} H_{xx} & H_{xy} & H_{xs} \\ H_{xy} & H_{yy} & H_{ys} \\ H_{xs} & H_{ys} & H_{ss} \end{bmatrix} \backslash \begin{bmatrix} D_x \\ D_y \\ D_s \end{bmatrix} = -\frac{1}{det(H)} \begin{bmatrix} H^*_{xx} & H^*_{xy} & H^*_{xs} \\ H^*_{xy} & H^*_{yy} & H^*_{ys} \\ H^*_{xs} & H^*_{ys} & H^*_{ss} \end{bmatrix} \bullet \begin{bmatrix} D_x \\ D_y \\ D_s \end{bmatrix} \tag{4.1}$$

The faction $\frac{1}{det(H)}$ needs a divider which is a resource consumption component. The maxima checking procedure is that if absolute value of O is less than 0.5, the local maxima is considered as an interest point, also coming with the coordinate of the interest point and the scale to be sent to the interest point descriptor module.

4.4 Interest point descriptor

SURF is originally a PC based algorithm. FPGA structure is different from CPU. The main steps of SURF in FPGA are the same as in PC while there are still some differences. Unlike the original SURF implementation, the FPGA based SURF generates descriptor from the neighbor region centered at interest point of size $24s \times 24s$, where s is the scale rather than SURF's 20s by 20s, which contains more details around. Like original SURF, FPGA SURF descriptor neighbor region also be split into 16 sub-regions. Each sub-region edge size is 9*9 rather than original 5*5 with overlaps region between each sub-region. The main structure is demonstrated in Figure 4.5.

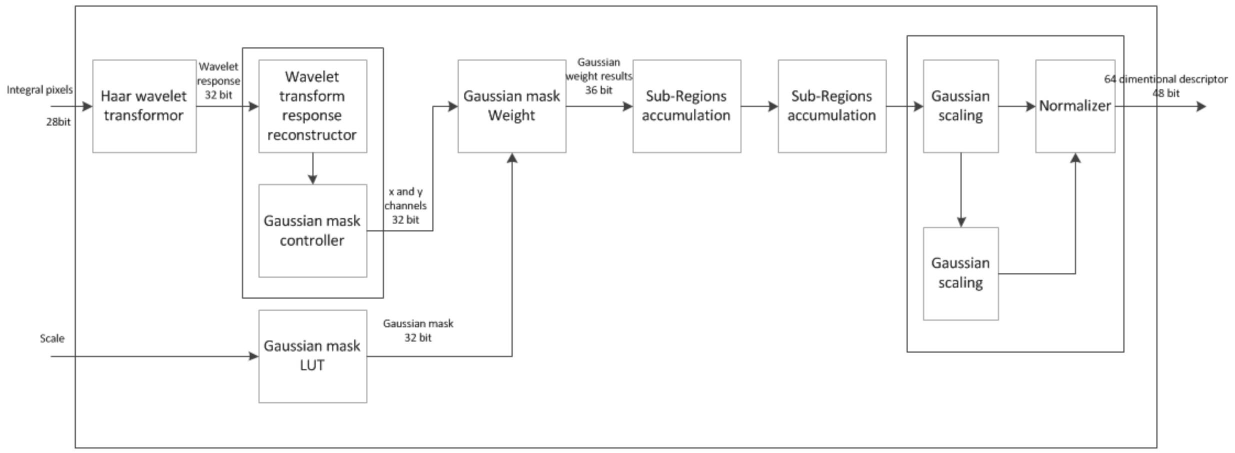

Figure 4.6: Interest pointer descriptor main architecture

4.4.1 Haar wavelet transform

The entry point of the interest point descriptor module is the Haar wavelet transformer. Integral pixels from DDR popped out according to the interest point. Instead of using a 20×20 matrix, we use 24×24 matrix centered at the interest point in order to gain more details. Since we simplify the computation to make the orientation as upright, the direction information is not important in current calculation. As we show in Figure 3.3, Haar wavelet filters are used to compute the responses in horizontal direction and vertical direction. The dark parts are for weight -1 and the white parts are for weight 1. In order to follow the pixel input method and construct a pipeline structure, we need to buffer three lines of serial input integral pixels to help calculate the Haar wavelet response.

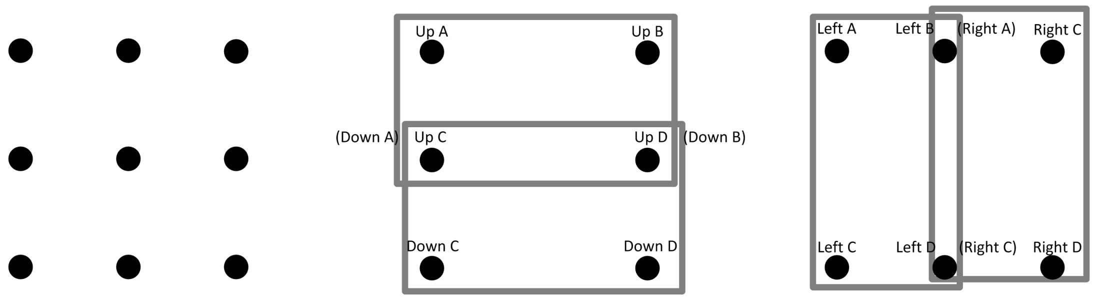

Figure 4.7: Haar wavelet procedure

Figure 4.7 demonstrates the Haar wavelet transform calculation procedure. The 9 points in left picture denotes the 9 pixels of the front three pixels of first line, second line and third line in the integral buffer. The eight pixels on the edges are divided into four groups which are upper group, lower group, left group, and right group for the Haar wavelet response calculation. Based on the definition, suppose the center pixel is the current location(x, y), it is easy to deduce the Up A location is $(x-1, y-1)$, Up B is $(x+1, y-1)$, Up C or Down A is $(x-1, y)$ and so on. Each group area sum defines as the following equations according to the integral image's feature mentioned in chapter 3.1.1 that the any area inside an image can be quickly calculated using the related integral image in fixed step just add the current location integral pixel and the upper left integral pixel then subtract the left and up integral pixels.

$$S_b = I_{(x-1,y)} + I_{(x+1,y+1)} - I_{(x-1,y+1)} - I_{(x+1,y)} \tag{4.2}$$

$$S_t = I_{(x-1,y-1)} + I_{(x+1,y)} - I_{(x+1,y-1)} - I_{(x-1,y)} \tag{4.3}$$

$$S_l = I_{(x-1,y-1)} + I_{(x,y+1)} - I_{(x,y-1)} - I_{(x-1,y+1)} \tag{4.4}$$

$$S_r = I_{(x,y-1)} + I_{(x+1,y+1)} - I_{(x,y+1)} - I_{(x+1,y-1)} \tag{4.5}$$

Where S_b, S_t, S_l, S_r are for sum of bottom, sum of top, sum of left and sum of right respectively and I represents the integral pixel in certain position. In Figure 3.3 we introduced the Haar wavelet filter masks, the black side represent -1 while white side represent 1. The Haar wavelet response result in two directions, horizontal and vertical. We can simply acquire d_x from subtract sum left from sum right.

$$d_x = S_l + S_r \tag{4.6}$$

$$d_y = S_d + S_u \tag{4.7}$$

Where d_y and d_x are for Haar response in Vertical direction and Horizontal position respectively. The response popping out operation is also processed at the same time with the integral pixels pushed into the buffer in order to save register resource. A 24×24 size response matrix is generated after all neighbor integral pixels are pushed into Haar wavelet module. The time delay in this module is two lines plus 2 pixels for the buffer and two adders pipelines, totally is 26*2+2+2 = 56 clock periods.

4.4.2 Wavelet response reconstructor

The size of wavelet response matrix is $24s \times 24s$ for both horizontal and vertical directions, which is about to be split into smaller 16 sub-regions as mentioned before. For reasons of accuracy, each sub-region is a 9×9 matrix which makes overlaps between each sub-region. Based on the system processing speed and the pipeline structure, the Haar wavelet response matrix needs to be re-ordered to ensure the pipeline structure to be well maintained. The constructor procedure is demonstrated in Figure 4.8.

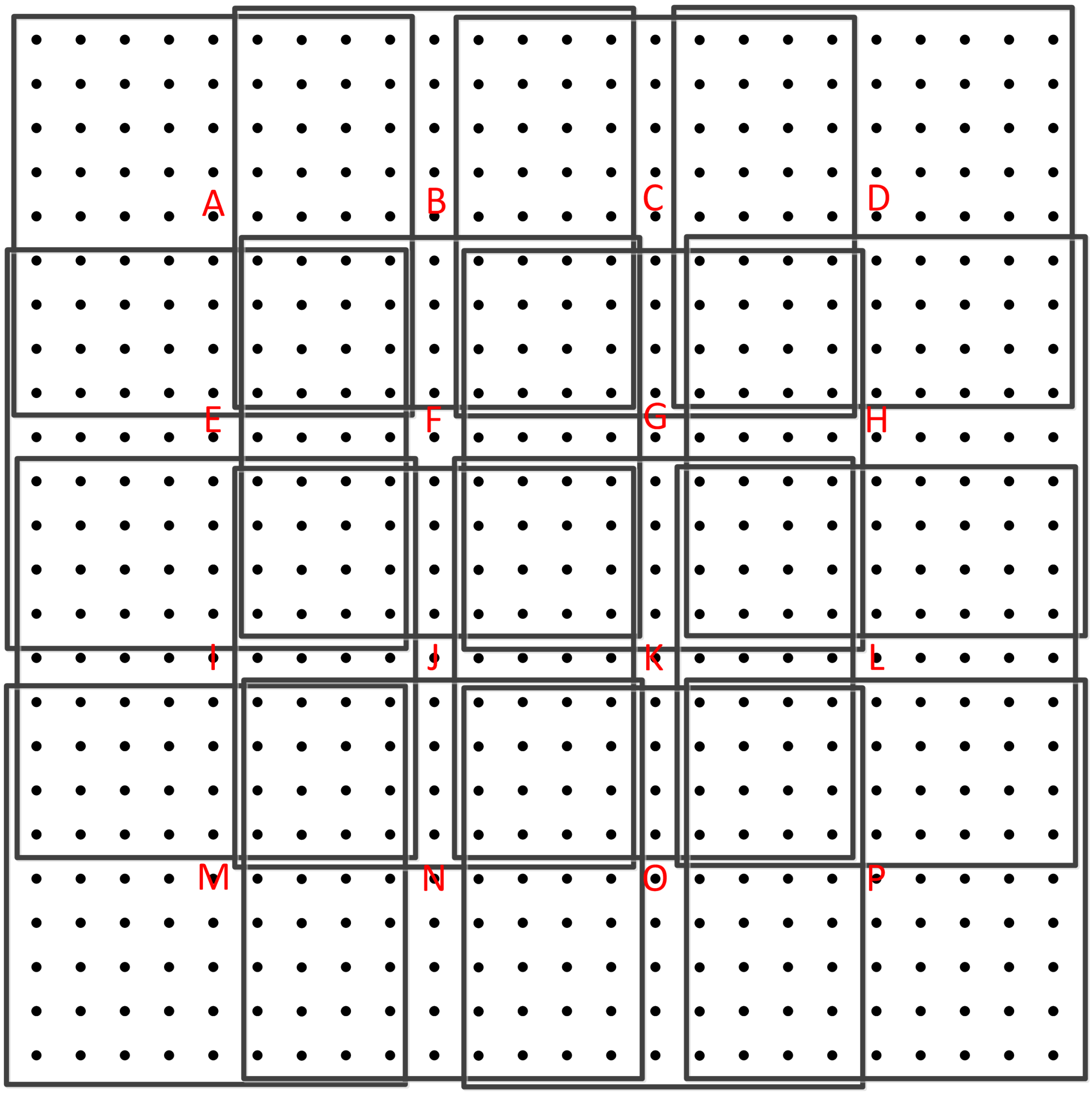

Figure 4.8: Wavelet response reconstructor

The Haar response stream serially enters module wavelet response reconstructor. Two counters act as the coordinates for the location of the response, to represent the relative distance and position of each response. In order to construct 16 sub-regions, we set 16 output ports to stand for the 16 sub-regions; also a state machine is built to help determine the right direction for each Haar wavelet response. In figure 4.8, A to P represent 16 sub-regions, each sub-region is a 9*9 matrix. There are several overlap regions between adjacent sub-regions, we take advantage of the overlap characteristics to design a high efficient state machine, as show in Figure 4.8, which help determine the right port for certain Haar wavelet response. For illustrative purpose, we show 16 outputs ports of horizontal direction instead of 32 ports in both horizontal and vertical directions. Some response may be duplicated and sent to different posts if they are in the overlap regions. One response can be sent up to 4 ports such like the responses in the overlap area of sub-region A, sub-region B, sub-region E and sub-region F. The advantage of the overlap is to make the best effort to extract the details around the interest point, and be more precise for the estimation of descriptor calculation. The reordered response dx and dy then be transmitted to Gaussian weight module.

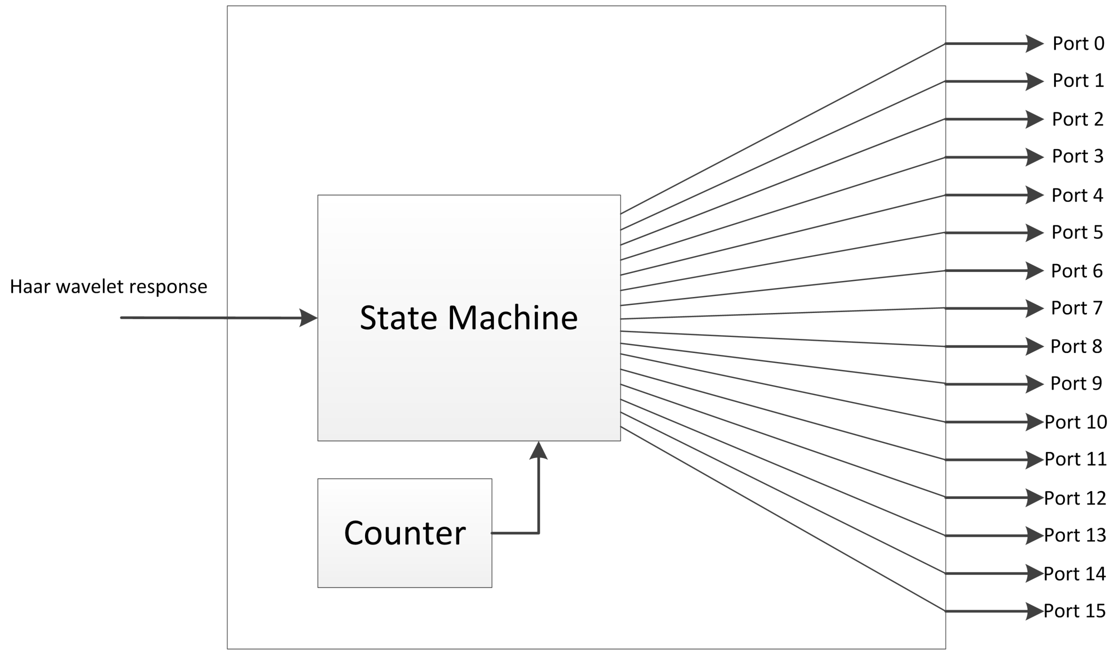

Figure 4.9: Haar wavelet response outputs

4.4.3 Gaussian mask LUT and Gaussian weight module

The responses dx and dy are first weighted with a 2D Gaussian mask centered at the interest point. The task for this module is to generate a 2D Gaussian mask of 9*9 based on the given scale parameter. The function of $2D$ Gaussian is:

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Where σ is the standard deviation of normal distribution; and $\sqrt{x^2+y^2}$ is the blur radius. In $2D$ space, the surface contours generated from the formula are concentric circles from the center. This is also called Gaussian blur, which is used to enhance the performance of image in different scales. Also Gaussian can be considered as a low pass filter for image since the Fourier transformation of Gaussian function is another Gaussian function. The image pre-processing with Gaussian function can reduce the fake high frequency information. Fortunately, according to the former analysis and calculation, the minim value of scale (S) is 2 and the maximum value of scale (S) is 5 since the octave setting in FPGA SURF is simplified with 2 layers. Based on that fact, we build a Gaussian mask look up table for a quick Gaussian value pick up compatibly with the re-ordered Haar wavelet response output sequence. The Gaussian mask LUT consists of 4 units from $\sigma = 2$ to $\sigma = 5$, each unit contains a 9×9 pre-calculated 2D Gaussian blur matrix, ready for output according to the input scale value. The Gaussian is stored in FPGA local Block RAM, with an address controller module to determine the correct output for the next module.

The advantage of building a Gaussian blur matrix LUT is the acceleration of processing and simplification of whole system design. In PC platform, we can calculate Gaussian distribution using floating point or library functions, but it is difficult to implement the square root and exponential operations in FPGA, which will occupy large scale FPGA resources. In the same time, use square root module, exponential module is oppose to the original design intention that processing SURF FPGA in real time and is hard to consistent in pipeline structure. To follow the pipeline structure's requirements and compatible with other module's time delay, a LUT is necessary to be built. It only takes one clock cycle to determine the correct Gaussian mask based on the input scale value. All the 2D Gaussian mask values are represented in fixed point format with 32 bit data width.

06071530	091CA6A3	0C90D860	0FCEF973	12249FF2	12FED916	12249FF2	0FCEF973	0C90D860
091CA6A3	0DC63A25	12FED916	17E5BD57	1B6D3275	1CB714CD	1B6D3275	17E5BD57	12FED916
0C90D860	12FED916	1A321DDF	20F4AD33	25D2973C	27998497	25D2973C	20F4AD33	1A321DDF
0FCEF973	17E5BD57	20F4AD33	2975D1C5	2F95401B	31D1931F	2F95401B	2975D1C5	20F4AD33
12249FF2	1B6D3275	25D2973C	2F95401B	369C282F	392D004A	369C282F	2F95401B	25D2973C
12FED916	1CB714CD	27998497	31D1931F	392D004A	3BDCB4DC	392D004A	31D1931F	27998497
12249FF2	1B6D3275	25D2973C	2F95401B	369C282F	392D004A	369C282F	2F95401B	25D2973C
0FCEF973	17E5BD57	20F4AD33	2975D1C5	2F95401B	31D1931F	2F95401B	2975D1C5	20F4AD33
0C90D860	12FED916	1A321DDF	20F4AD33	25D2973C	27998497	25D2973C	20F4AD33	1A321DDF

Table 4.1: Gaussian LUT

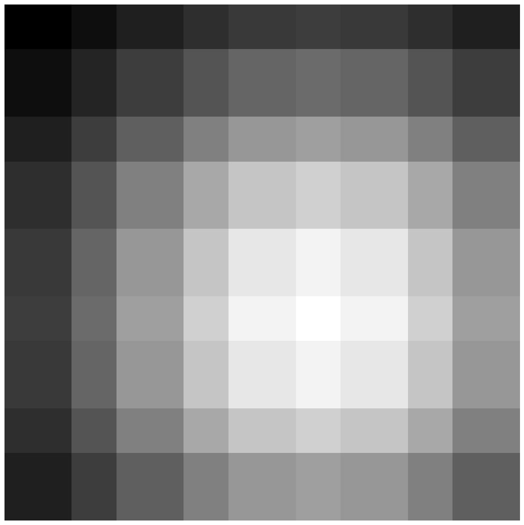

Figure 4.10: Gaussian LUT

Table 4.1 is one of the four Gaussian LUTs represented in Hex format with scale value of 2. Figure 4.10 is the image generated from Table 4.1.

The following module is the Gaussian weight module. We use Gaussian blur mask from Gaussian LUT to weight Haar wavelet response from Wavelet module in each sub-region simultaneously. 16 ports of Haar wavelet response stand for 16 sub-regions in interest point neighbor matrix. Each of the matrix needs to be weighted by Gaussian blur mask. In order to accelerate the processing, we build a parallel weight structure for the 16 matrix. Since the sequential input method of Haar wavelet response, different sub-region matrix response may be pushed into the Gaussian weight module at the same time especially at the overlap regions. To ensure a parallel processing structure, independent multiplier is allocated in Gaussian weight module for each Haar wavelet response matrix. After the analysis of Haar wavelet response matrix entering order, a multipliers multiplexing method is introduced for reducing the use of multipliers from DSP48 slice. The main structure of multiplier multiplexing structure is shown in Figure 4.10.

The weighted matrix's sum is the weighted response result as pre-descriptor for the whole system. To get the sum of weighted response, an accumulator of gathering all valid data is placed for each output port. There are two structures like the multiplier multiplexing structure in Gaussian weight module since not only horizontal response $\sum d_x$but also vertical$\sum d_y$

We also extract the sum of the absolute values of responses, $|d_x|$ and $|d_y|$, which is accumulated through the Absolute -¿ Accumulator path shown in Figure 4.10, for the information of the polarity of the intensity changes. Since the Absolute module is combination logic, which does not affect the pipeline structure, there is no delay between d_x, d_y and $|d_x|$, $|d_y|$. Each of d_x, d_y $|d_x|$ and $|d_y|$ is a vector of length 16; The bit width of d_x, d_y $|d_x|$ and $|d_y|$ are 48 bits for a better accuracy in fixed point format. $[\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|]$ consists of a vector of length 64, which is the prototype of final interest point descriptor but needs several more processing steps. The bit width output from this module is 48 bits.

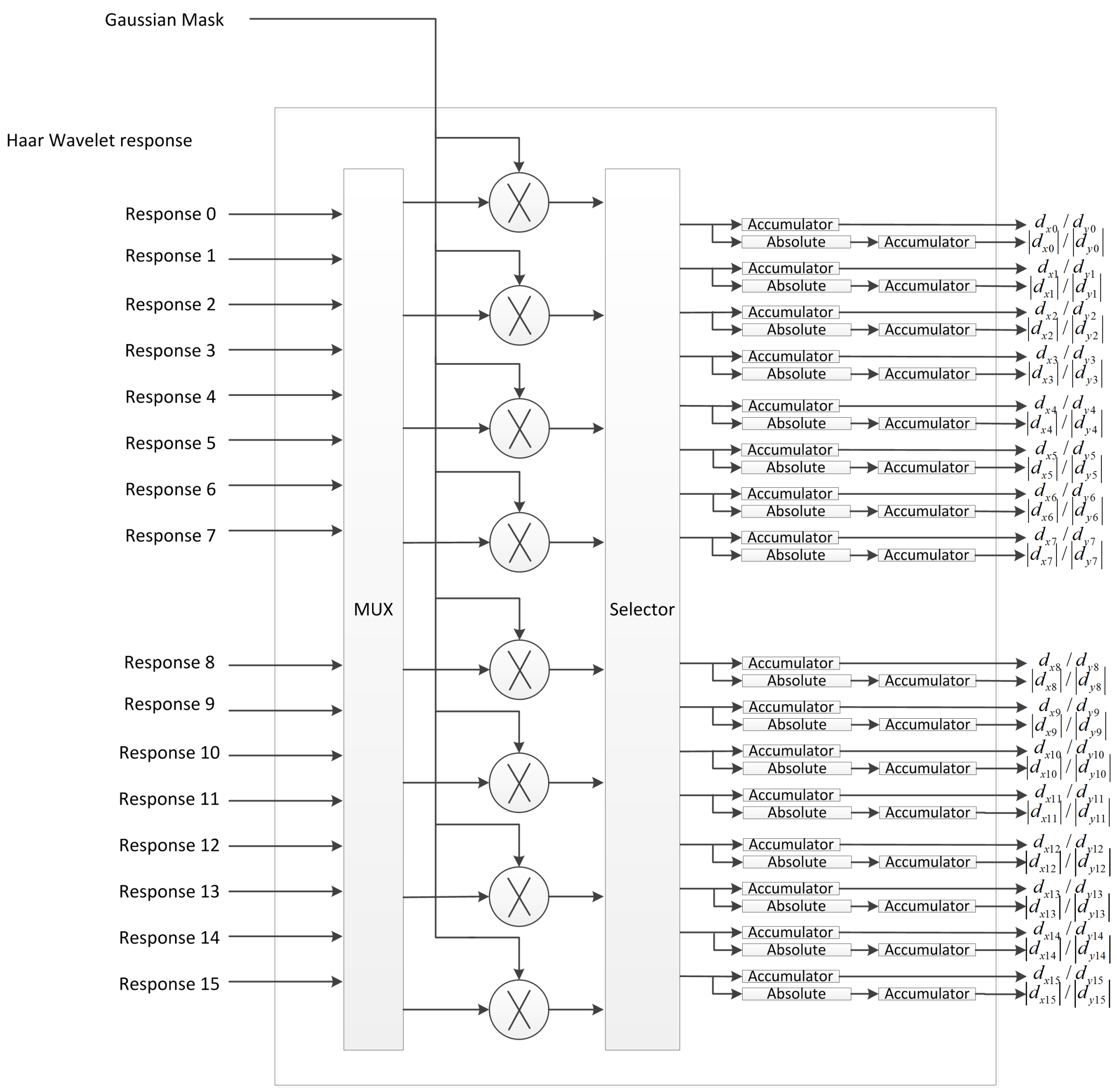

Figure 4.11: Multipliers multiplexing structure

4.4.4 Gaussian scaling and length of region

Gaussian blur function is used widely in image processing, not only in previous mentioned Haar wavelet sub-regions weight, but the sub-regions accumulation result vector's blurring. The result of Haar wavelet sub-regions weight operation successfully deduces 4 vectors of length 16 in the Gaussian weight module and accumulation module, but we still need to weight the 4 vectors for the further accurate result. A Gaussian blur function with $\sigma = 1.5$ and size of 4×4 is introduced to weight the four vectors of length 16. The basic calculation procedure is illustrated in Figure 4.12. We call this Gaussian blur mask a Gaussian scaling mask. Each of the elements in Gaussian scaling mask weights the corresponding element in pre-descriptor $pred = [\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|]$. The wavelet responses are invariant to scale factor, the weighted pre-descriptor needs to be turned into a unit vector. The procedure of acquiring unit vector is show below:

Step 1: Calculate normal number of pre-descriptor by square rooting the sum of all elements' square in pred.

Step2: Divided pred by $\sum pred$'s reciprocal.

Since the square root is difficult to implement in FPGA structure, we simply the design by change the square operation into absolute operation. Step 1 can be described in details:

Part1: Setup an absoluter for the absolute value of $[\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|]$

Part2: Setup an accumulator for $\sum pred$.

Also withdraw the square root operation in order to follow the pipeline structure.

In this module, a divider is needed to calculate $\sum pred$'s reciprocal. We choose to use Xilinx divider IP to fulfill this task. This IP divider is a pipeline structure with over 50 clock cycles calculation period. After all the steps, we obtain the final descriptor for one interest point. The descriptor can be represented as

$$D = [D_{x,} D_y, |D_x|, |D_y|]$$

with a length of 64, each of the elements bit widths is 48 bits, which stands for the interest point and its neighbor matrix' feature preciously. We will use the descriptors

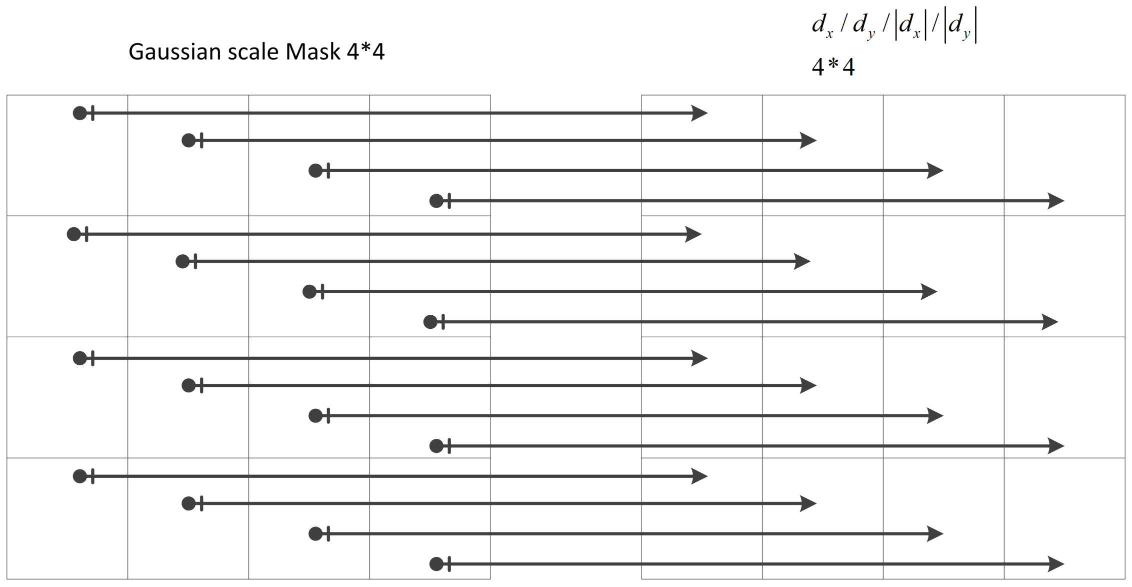

Figure 4.12: Gaussian scale mask

in the following Matching module to determine the items in certain picture.

4.5 Descriptor matching module

After interest point descriptor calculation, it is first pushed into a FIFO. Based on the interest point detection, the interest point does not appear continuously. The descriptor extractor module may be idol for some time because there is no neighbor matrix enters descriptor extraction module.

The matching module first fetches one descriptor from the FIFO, and then computes the similarity of pre-built item library and current descriptor to determine whether an object is detected. In this design we use traffic sign a stop sign as the object to detect in the video. According to SURF algorithm, a given item or a picture is needed for the matching operation in another picture. We choose a standard stop sign picture as the reference. Actually, based on the rule of speed and resources, another system of calculating the stop sign's descriptors paralleling with current system is a waste of resource. We choose to use matlab to extract the interest points descriptors from a standard stop sign, and transfer the descriptor from double format to 48bit width fixed point format. Also, all of the 128 descriptors information is stored in ROMs built from FPGA local block RAM. The basic matching procedure is shown in Figure 3.4.

When a descriptor enters the matching module, the reference descriptor in library ROM also be read by matching module. The entering descriptor needs to be compared with every descriptor in the library ROM. The processing speed is unacceptable if the matching operation for descriptor is completely serial. A structure optimization of library is to divide the library ROM into 8 independent ROMs as a tradeoff between resource and speed. According to the descriptor entering sequence, totally 128 descriptors are distributively stored in the 8 library ROMs. The storing order is to averagely break a descriptor of length 64 into 8 groups, so it will take 8 clock cycles to read all the descriptor out of the ROM rather than 64 clock cycles, 8 times faster than before. The architecture is shown in Figure 4.13.

A compare results 128 distance value after subtracting library descriptor from current frame value. Each of the descriptors from current frames generates a distance vector of length 128, which compares with the previous distance vector and keeps the smaller elements in the distance vector. The smallest distance vector is then sorted from small to large. The module selects the sum of the smallest 30 distances to compare with the threshold. If the distance sum is less than the threshold, the stop sign is detected. Just like the Figure 4.14 demonstrated.

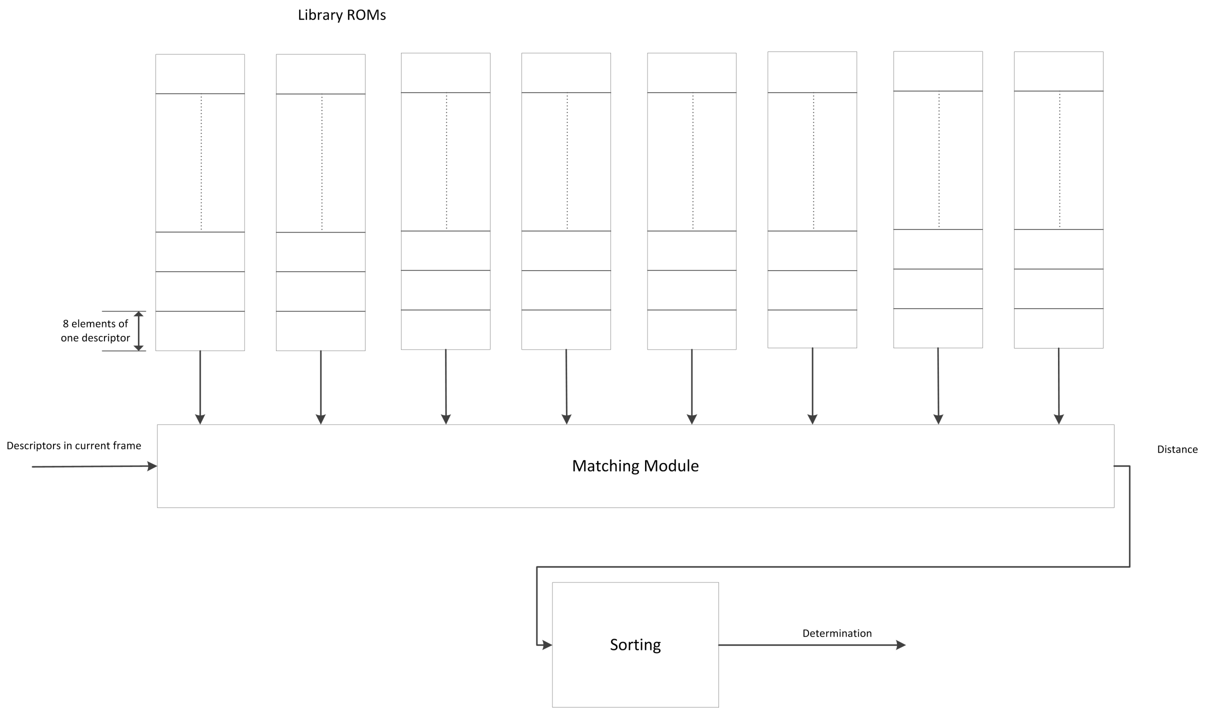

Figure 4.13: Matching module

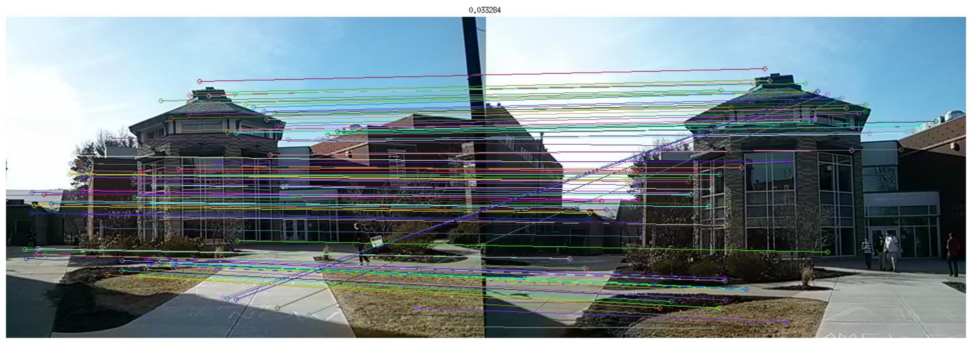

Figure 4.14: Matching result

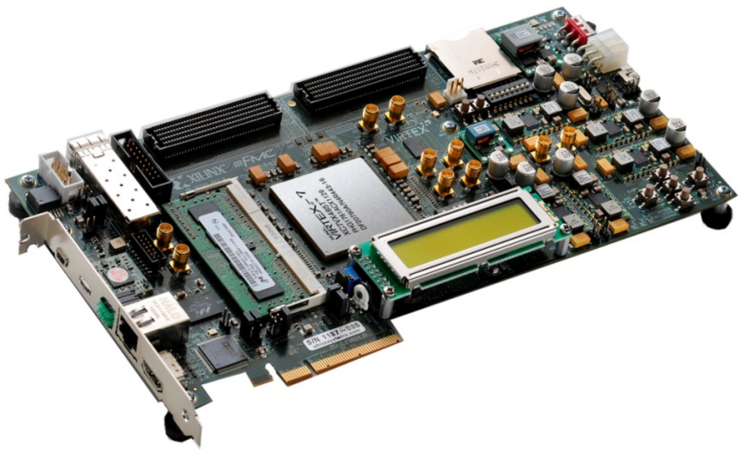

Figure 4.15: KC705 platform

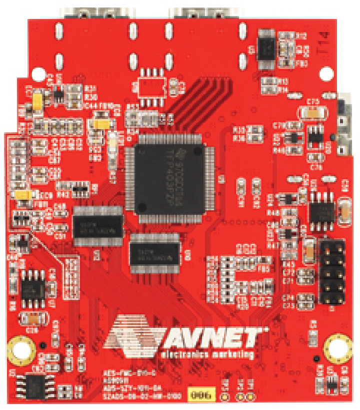

Figure 4.16: AVNET DVI FMC module

4.6 Platform and results

The Whole system is built on a Kintex 7 KC 705 Xilinx FPGA development board shown in Figure 5.1. We use a desktop as the DVI video Frame source. The input video resolution is 800*600. The video signals are transmitted through AVENT DVI I/O FMC daughter board shown in Figure 5.2. The output video signals are also sent through ANVET DVI I/O board to a monitor in resolution 800*600. FPGA detects stop signs of every frame preciously.

Resources	Used	Available	Utilization
Number of Slices Registers	108581	407600	26%
Number of Slice LUTs	179559	203800	88%
Number of bonded IOBs	173	500	34%
Number of Block RAM36s	182	445	40%
Number of RAM18s:	80	890	8%
Number of DSP48E1s	244	840	29%

Table 4.2: Resource utilities in the design

The FPGA SURF design tested in a frame rate of 60fps and resolution of 800*600, successfully detect the stop sign in real time, high-lighted the pixels represent the stop sign. SURF implemented on FPGA speeded up considerably comparing with the original CUP-based SURF, which will be a respectable help for the future video industrial development.

Chapter 5

Conclusions

In this thesis, we describe two object detection algorithms, differentiation in stationary background and feather detection algorithm Speeded up robust feature. We simulate and develop the FPGA structure of differentiation in stationary background also the FPGA SURF implementation. To reduce the resources utilities and computational complexity, also accelerate the processing speed, pipeline and parallel architecture are introduced in these two designs. We do every effort to optimize the structures to make a balance between processing speed and chip resource utilities. A moving object can successfully be detected and the traffic sign detection using SURF FPGA are able to process on an FPGA with real-time SVGA video at 60fps.

However, the detection is still limited for both of the two algorithms, for example, the background of differentiation algorithm cannot be changed or the detection performance will be unacceptable. Also the detection of FPGA SURF is still limited for the item to detect since the library ROM size is not enough for more than two items' descriptor. That means Only one items can be detected during the process.

We also notice that the chip area usage may be further optimized for less area with higher speeded up. For example, the bit width for the data in SURF may be reduced for an acceptable accuracy loss.

We also glad to see the processing speed of SURF FPGA achieves a real time processing speed in SVGA resolution in 60 fps, much faster than CPU based SURF, include the descriptor matching operation. The pipeline structure effectively achieves high frame rate video processing.

In conclusion, video object detection algorithms based on FPGA are effective, high performance methods that is well suitable for resource and power limited condition, especially in automobile realm, which reduce the processing speed from 3 to 4 seconds per frame to real time processing.

Bibliography

[1] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *Computer Vision–ECCV 2006*. Springer, 2006, pp. 404–417.

[2] J. Zhao, S. Zhu, and X. Huang, "Real-time traffic sign detection using surf features on fpga," in *High Performance Extreme Computing Conference (HPEC), 2013 IEEE*. IEEE, 2013, pp. 1–6.

[3] Z. Zhang, C. Cao, R. Zhang, and J. Zou, "Video copy detection based on speeded up robust features and locality sensitive hashing," in *Automation and Logistics (ICAL), 2010 IEEE International Conference on*. IEEE, 2010, pp. 13–18.

[4] M. Zhou and V. K. Asari, "Speeded-up robust features based moving object detection on shaky video," in *Computer Networks and Intelligent Computing*. Springer, 2011, pp. 677–682.

[5] J. Svab, T. Krajník, J. Faigl, and L. Preucil, "Fpga based speeded up robust features," in *Technologies for Practical Robot Applications, 2009. TePRA 2009. IEEE International Conference on*. IEEE, 2009, pp. 35–41.

[6] A. Broggi, C. Caraffi, R. I. Fedriga, and P. Grisleri, "Obstacle detection with stereo vision for off-road vehicle navigation," in *Computer Vision and Pattern Recognition-Workshops, 2005. CVPR Workshops. IEEE Computer Society Conference on*. IEEE, 2005, pp. 65–65.

[7] K. Kluge and S. Lakshmanan, "A deformable-template approach to lane detection," in *Intelligent Vehicles' 95 Symposium., Proceedings of the*. IEEE, 1995, pp. 54–59.

[8] R. Labayrade, D. Aubert, and J.-P. Tarel, "Real time obstacle detection in stereovision on non flat road geometry through" v-disparity" representation," in *Intelligent Vehicle Symposium, 2002. IEEE*, vol. 2. IEEE, 2002, pp. 646–651.

[9] M. Bertozzi and A. Broggi, "Gold: A parallel real-time stereo vision system for generic obstacle and lane detection," *Image Processing, IEEE Transactions on*, vol. 7, no. 1, pp. 62–81, 1998.

[10] L.-W. Tsai, J.-W. Hsieh, C.-H. Chuang, Y.-J. Tseng, K.-C. Fan, and C.-C. Lee, "Road sign detection using eigen colour," *IET computer vision*, vol. 2, no. 3, pp. 164–177, 2008.

[11] A. De La Escalera, L. E. Moreno, M. A. Salichs, and J. M. Armingol, "Road traffic sign detection and classification," *Industrial Electronics, IEEE Transactions on*, vol. 44, no. 6, pp. 848–859, 1997.

[12] Q. Zhu, M.-C. Yeh, K.-T. Cheng, and S. Avidan, "Fast human detection using a cascade of histograms of oriented gradients," in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 2. IEEE, 2006, pp. 1491–1498.