STATUS OF THESIS

Title of thesis | MINIMIZATION OF RESOURCE UTILIZATION FOR A REAL-TIME DEPTH-MAP COMPUTATIONAL MODULE ON FPGA

I NGO HUY TAN ,

hereby allow my thesis to be placed at the information Resource Center (IRC) of Universiti Teknologi PETRONAS (UTP) with the following conditions:

1. The thesis becomes the property of UTP
2. The IRC of UTP may make copies of the thesis for academic purposes only.
3. This thesis is classified as

[ ] Confidential

[√] Non-confidential

If the thesis is confidential, please state the reason:

_____

_____

_____

The contents of the thesis will remain confidential for _____years.

Remarks on disclosure:

_____

_____

_____

Endorsed by

_____ _____

Signature of Author

Permanent address:

Tao Hamlet, Mao Dien Village, Thuan Thanh District, Bac Ninh Province, Vietnam.

Date: _____

Signature of Supervisor

Assoc. Prof. Dr. Nor Hisham Hamid

Date: _____

UNIVERSITI TEKNOLOGI PETRONAS

MINIMIZATION OF RESOURCE UTILIZATION FOR A REAL-TIME DEPTH-MAP COMPUTATIONAL MODULE ON FPGA

By

NGO HUY TAN

The undersigned certify that they have read, and recommend to the Postgraduate Studies Programme for acceptance this thesis for the fulfillment of the requirements for the degree stated.

Signature: _____

Main Supervisor: Assoc. Prof. Dr. Nor Hisham Hamid

Signature: _____

Co-Supervisor: Mr. Patrick Sebastian

Signature: _____

Head of Department: Assoc. Prof. Dr. Nor Hisham Hamid

Date: _____

MINIMIZATION OF RESOURCE UTILIZATION FOR A REAL-TIME DEPTH-MAP COMPUTATIONAL MODULE ON FPGA

By

NGO HUY TAN

A Thesis

Submitted to the Postgraduate Studies Programme
as a Requirement for the Degree of

MASTER OF SCIENCE

ELECTRICAL AND ELECTRONICS ENGINEERING

UNIVERSITI TEKNOLOGI PETRONAS

BANDAR SRI ISKANDAR

PERAK

SEPTEMBER 2011

DECLARATION OF THESIS

Title of thesis | MINIMIZATION OF RESOURCE UTILIZATION FOR A REAL-TIME DEPTH-MAP COMPUTATIONAL MODULE ON FPGA

I NGO HUY TAN ,

hereby declare that the thesis is based on my original work except for quotations and citations which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at UTP or other institutions.

Witnessed by

_____ _____

Signature of Author
Permanent address:
Tao Hamlet, Mao Dien Village, Thuan Thanh District, Bac Ninh Province, Vietnam.

Signature of Supervisor
Assoc. Prof. Dr. Nor Hisham Hamid

Date: _____ Date: _____

ABSTRACT

Depth-map algorithm allows camera system to estimate depth in many applications. The algorithm is computationally intensive and therefore more effective to be implemented on hardware such as the Field Programmable Gate Array (FPGA). However, the recurring issue in FPGA implementation is the resource limitation. The issue is normally resolved by modifying the algorithm. However, the issue can also be addressed by implementing hardware architectures without the need to modify the depth-map algorithm. In this thesis, five different depth-map processor architectures for the sum-of-absolute-difference (SAD) depth-map algorithm on FPGA at real-time were designed and implemented. Two resource minimization techniques were employed to address the resource limitation issues. Resource usage and performance of these architectures were compared. Memory contention and bandwidth constrain were resolved by using self-initiative memory controller, FIFOs and line buffers. Parallel processing was utilized to achieve high processing speed at low clock frequency. Memory-based line buffers were used instead of register-based line buffers to save 62.4% of logic element (LEs) used, but require some additional dedicated memory bits. A proper use of registers to replace repetitive subtractors saves 24.75% of LEs. The system achieves SAD performance of 295 mega pixel disparity per second (MPDS) for the architecture with 640x480 pixel image, 3x3 pixel window size, 32 pixel disparity range and 30 frames per second. The system achieves SAD performance of 590 MPDS for the 64 pixels disparity range architecture. The disparity matching module works at the frequency of 10 MHz and produces one pixel of result every clock cycle. The results are dense disparity images, suitable for high speed, low cost, low power applications.

ABSTRAK

Algoritma peta-dalaman membolehkan sistem kamera untuk menganggarkan kedalaman di banyak aplikasi. Algoritma ini pengkomputeran intensif dan kerana itu lebih berkesan untuk dilaksanakan pada peranti keras seperti Field Programmable Gate Array (FPGA). Namun, masalah berulang dalam perlaksanaan FPGA adalah keterbatasan sumber daya. Masalah ini biasanya diselesaikan dengan mengubah algoritma. Masalah ini juga boleh diatasi dengan arsitektur peranti keras melaksanakan tanpa perlu mengubah algoritma kedalaman-map. Dalam tesis ini, lima berbeza kedalaman-peta arsitektur prosesor untuk jumlah perbezaan mutlak (SAD) kedalaman-peta algoritma pada FPGA pada real-time direka dan dilaksanakan. Dua sumber daya teknik minimisasi dipekerjakan untuk menangani masalah keterbatasan sumber daya. Penggunaan sumber kuasa dan prestasi arsitektur ini dibandingkan. Pertengkaran memori dan pengendalian bandwidth diselesaikan dengan menggunakan pengendali memori self-inisiatif, FIFO dan garis buffer. Pemprosesan selari digunakan untuk mencapai kelajuan pemprosesan tinggi pada frekuensi clock yang rendah. Memori yang berasaskan 'buffer line' digunakan dan bukan 'register line' dapat mengurangkan penggunaan elemen logic (LE) sebanyak 62.4%, namun memerlukan beberapa bit memori tambahan khusus. Penggunaan tepat register untuk menggantikan subtractor dapat menjimatkan 24.75% penggunaan LE. Sistem dapat mencapai prestasi SAD perbezaan 295 mega piksel sesaat (MPDS) untuk arkitektur dengan imej 640x480 piksel, 3x3 saiz piksel tingkap, jurang piksel 32 dan 30 frame sesaat. Ia mencapai prestasi SAD 590 MPDS untuk arkitektur dengan disparitas 64 piksel. Modul perbezaan bekerja pada frekuensi 10 MHz dan menghasilkan satu piksel setiap kitaran jam. Hasilnya adalah gambar perbezaan yang padat, sesuai untuk kelajuan tinggi, kos rendah dan aplikasi yang menggunakan kuasa rendah.

ACKNOWLEDGEMENTS

I owe my deepest gratitude to my parents and my family members for their immortal love and support.

It is my pleasure to thank my supervisor Dr. Nor Hisham Bin Hamid for his help, courage, and support in many ways. My sincere gratefulness goes to my co-supervisor Mr. Patrick Sebastian for his continuous help, encouragement, and guidance.

Thanks and gratitude must be given to the graduation assistantship scheme and the Electrical and Electronics Department of the Universiti Teknologi PETRONAS for this great opportunity. Thanks extended to all the postgraduate office members for their help.

Heartfelt gratitude is extended to my father Mr. Ngo Huy Thuan, my mother Mrs. Vu Thi Tinh, my brother Ngo Huy Tu, other family members and relatives, who without their love and support it would have been impossible for me to complete this work.

My sincere appreciation goes to my dear friend Ms. Zahraa Elhassan Mohamed Osman for her valuable help and support.

Last but not least, I would like to thank all my friends and colleagues who have made this journey very special and unforgettable.

DEDICATION

To My Beloved Parents and Family

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Image Processing

Machine vision systems are image processing systems that are able to simulate the human vision system by analyzing the scene captured in a digital image. By using several image processing operations, a machine vision system is able to acquire information such as size, shape, color, material, location and type of an object and even the action it is performing. Building a general purpose computational vision system is a challenging task. However, researchers have been successful in designing algorithms and building systems that deal with some specific tasks of the human vision system [1].

Image processing operations are classified into high and low level operations [2]. Low level operations involve algorithms that implement simple arithmetic calculations such as addition, subtraction and multiplication and they are performed on all pixel data of the image. Low level image processing systems are calculation and data intensive systems since they need to process a massive amount of data from millions of pixels of the image. Processing each pixel normally requires only the data in a small neighborhood around that pixel. Thus, the calculation can be done in parallel [2] for many local regions at the same time and the algorithm can be implemented on a parallel processing system for very high throughput. Examples of low level image processing algorithms are edge detection, down sampling, contrast enhancement, image rectification and depth-map calculation.

High level image processing tasks include the algorithms such as object recognition, motion detection and face identification. They are complex algorithms and often involve techniques in the field of artificial intelligence. High level image processing algorithms are implemented after the image has been processed by several low level algorithms. They acquire information generated by low level algorithms and intend to "understand" such information. High level image processing tasks often require less computational power than low level tasks because they are applied on selected portions of the image rather than uniformly across the entire image [2]. They work well on fast serial computational platform like personal computer (PC) and general purpose processing system.

1.2 Three Dimensional Vision

Among the low level image processing tasks, depth-map algorithm allows computer vision system to estimate depth – the distance in the third dimension of the space. The ability of perceiving depth is important to generate information for high level image processing tasks. Example of this is in the robotic navigation system. With the perception of depth, a robotic system is able to navigate the road and avoid obstacles.

The ability to perceive three dimensions of the space – which are length, width and depth – is desired in many artificial vision systems. In image processing, objects are characterized by their size, shape and color. The size of the projection of an object on a camera sensor depends on the size of the object itself and the distance from the object to the camera. Thus, it is important to calculate the length, width and depth concurrently. The ability of perceiving depth plays a vital role in modern vision systems. It has potential applications in robotic navigation, 3D imaging, camera surveillance systems and object recognition.

By mimicking the natural vision system of human and animals, artificial 3D vision systems have been implemented with two cameras [3-5]. They are called *stereo vision systems*. A typical depth estimation system consists of two cameras with overlapping field of view and a processing unit [4, 6]. Two images of the same scene are captured by two cameras from two different viewpoints. The projection of an

object is displaced on an image compared to the other image. One of the two images is made as a reference image and the other is made as a displaced image. A search algorithm is applied on the displaced image to find the *match* for a pixel of the reference image. Such displacement of a pixel (in the unit of micrometer) is recorded in the form of pixel intensity on the result image. The resulting image of a depth perceiving system is a set of the displacement values for every pixel of the reference image. The displacement value of a pixel is also called the disparity value and the set of displacement values for the reference image is called the disparity map or disparity image. To implement a 3D (stereo) vision system, it is necessary to develop a processing algorithm which performs the search operation to find matching pixels between the two images. Such an algorithm is called a *depth-map algorithm* or *disparity algorithm*.

1.3 Depth-map Algorithms

Stereo vision system provides information of depth in the form of pixel intensity. It allows estimation of distance between system's cameras and the objects. The closer object has higher displacement and therefore, it has higher disparity value. Objects at infinity distance have zero disparity. This will be illustrated in chapter 2. The disparity algorithm is applied on the two images to find the textural matches. It is classified under the low level category of image processing algorithms [2].

There are several different depth-map algorithms [4]. One which calculates the disparity map using the intensity of the input images is called the intensity based algorithm. The common intensity based algorithms include *absolute difference algorithm* (AD), *sum-of-absolute difference* (SAD), *squared intensity difference* (SD) and *sum-of-squared difference* (SSD) [4]. Other depth-map algorithms are based on gradient [7] of the pixel intensity or census transform [8]. These algorithms will be described in more detail in chapter 2 – literature review.

1.4 Processing Platforms

Due to the computations and data intensive property of the low level image processing algorithms, the depth-map algorithm requires high computational power and memory bandwidth. Several attempts have been made to implement the depth-map algorithm on different computing platforms [6, 9]. These processing platforms include general purpose processor (GPP), digital signal processor (DSP), field programmable gate array (FPGA) and application specific integrated circuit (ASIC). Systems implemented on a general purpose processor often take thousands of clock cycles to calculate the disparity value of one pixel. For example, to calculate the disparity value for a pixel in a 3x3 window with 32 disparity range, it would require 3x3x32 = 288 subtractions, 31x2 = 62 comparisons, 288 additions to calculate total difference, 288 additions to calculate total similarity and other calculations for index increment and read/write process. Each instruction requires four clock cycles. Hence, it takes at least (288x3 + 62)x4 = 3704 clock cycles to calculate the disparity value of one pixel. In actual measurement, it took approximately 1.15 seconds to calculate the disparity map for a pair of images with the size of 500x375 pixels. The program implemented simple SAD disparity algorithm using C programming language, ran on an Intel's core i5 computer, 4GB DDR3 RAM, 1000 MB/s memory bandwidth, Windows 7 operating system. Because of the high flexibility, general purpose processor can be used to test any complex algorithm. Digital signal processor (DSP) can deliver higher throughput compared to general purpose processor running at the same clock frequency. This platform is ideal to implement high level tasks such as multiplication and division. However, parallelism provided by DSP is very limited. Field programmable gate array (FPGA) can take advantage of parallelism inherent in many low level image processing tasks. With relatively lower clock frequency than a DSP, FPGA can deliver higher throughput. The fastest implementation of disparity algorithm is by using application specific integrated circuit (ASIC). However, ASIC is a fixed circuit with no flexibility. It is good for end-user products rather than a testing platform.

1.5 Field Programmable Gate Array (FPGA)

Many applications require real-time speed of the depth estimation system [5]. Software based systems like PC and DSP are too slow for these applications [9]. In the recent years, it is realized that FPGA is a good platform to implement low level image processing algorithm like depth-map algorithm for research and testing purposes [5, 6, 10]. The hardware nature of FPGA allows image processing systems implemented on it to achieve performance comparable with ASIC while maintaining flexibility with the reconfigurable capability. FPGA is also more cost effective than ASIC since it is mass produced. However, FPGAs have a limited amount of resources. For a typical Altera's FPGA, the available resources are the number of logic elements, the number of internal dedicated memory bit, the maximum external memory bandwidth, the number of dedicated multipliers and the number of phase locked loop (PLL) circuits. Image processing system implemented on FPGA should fit within the available resources. Otherwise, multiple FPGAs need to be connected to each other with a complex handshaking and data transferring scheme. Consequently, resource limitation is an issue when implementing depth-map algorithm on FPGA.

In this research, the Altera's DE2-70 FPGA board was used because it has sufficient resources for implementation of depth-map algorithm at low cost. The Altera's DE2-70 development board includes a Cyclone II FPGA with 68,000 LEs, 1,152,000 internal dedicated memory bits, 300 nine-bit embedded multipliers and 4 phase lock loop circuits (PLL). It also has 32 MB of dynamic random access memory (DRAM), 2 MB of static RAM (SRAM) and 8 MB of flash memory.

1.6 Problem Statement and Objectives

Due to the resources limitation problem, implementing disparity algorithm on FPGA is a challenging task [11]. As shown in subsection 1.4, typical depth estimation system employs hundreds of subtractors, comparators and adders. These operations are built using logic gates and registers, which are implemented using logic elements of the FPGA. Depending on the window size and disparity range, large amount of internal memory bits is used for different types of line buffers. In the FPGA market,

the number of logic elements and memory bits is directly proportional to the cost of the FPGA. Therefore, resource saving is important to reduce the cost of FPGA-based system. High-end FPGAs (FPGAs with abundant resources and high computational power) are more expensive than low-end FPGAs (FPGAs with limited resources). Keep in mind that the depth-map algorithm is not the only algorithm being implemented in modern vision systems. When a vision system, which employs many sophisticated algorithms, exhausts the resource on a single FPGA, the solutions are to add another FPGA, modify the algorithm or modify the architecture. Adding other FPGA to the system requires the system to be partitioned into smaller circuits [1]. Besides that, transferring large amount of data and control signals with limited bandwidth and strict timing requirement between two FPGA is a difficult issue [1]. It is desired that the whole system is well designed to fit into a single FPGA.

Apart from the problem of limited logic elements and memory bits on FPGA, the finite bandwidth of external memory is another issue for depth-map system as well as other low level image processing systems. This is due to the fact that the disparity algorithm is data intensive. For instance, to calculate the disparity value of one pixel, it requires the data of (32+2)x3+3x3 = 111 neighbor pixels from two images (with the window size of 3x3 and disparity range of 32 pixels). Memory bandwidth is also considered as an important resource on FPGA boards. When talking about resource minimization for FPGA, we imply that memory bandwidth reduction is included.

In many applications, disparity image is required at *real-time*. Real-time systems are the systems which finished processing and deliver output before new inputs and new commands enter the system. In image processing field, real-time systems often refer to the systems that can deliver a throughput of more than 30 frames per second (fps). This real-time requirement puts more troubles on the task of designing depth perceiving systems since a real-time system demands more resource and memory bandwidth than a slower system. With the concern to the resource limitation of FPGA and real-time requirement, we have made a hypothesis that "*Depth-map algorithm can be implemented on FPGA with better architecture to save resources and achieve real-time performance without modifying the algorithm or losing the quality of the resulting image*". With that supposition, the research in this thesis implemented a

SAD depth-map system with look-up table and data reuse techniques to achieve real-time performance with little resource usage.

Objectives of the research are:

- To develop different hardware architectures for the SAD depth-map algorithm
- To implement parallel processing techniques to increase throughput and achieve real-time performance.
- To implement the designed depth-map systems on FPGA and apply resource minimization techniques to minimize resource usage.
- To evaluate the resource usage and performance of these systems and compare the implemented architectures with other systems in the literature for resource usage and performance.

1.7 Scope and Contribution

The main contribution is a depth-map processor architecture that achieves processing speed of more than 30 frames per second and minimizes resource usage. The system was implemented with two types of line buffers, a specifically designed memory controller for image processing purpose, and a data reuse technique that reuse the available data through the calculation of many disparity pixels. That data reuse technique is proven to save large amount of resource of the FPGA.

The work does not try to come up with a new depth-map algorithm or modification of existing algorithm. It rather implements a simple SAD algorithm on FPGA with different architectures to minimize resource usage and achieve real-time performance. The supported image size is 640x480 pixels. The window size is 5x5 pixels and the maximum disparity range is 64 pixels. Among the three important aspects of system design which are speed, resource usage and power consumption, this research focuses on speed and resource utilization only, because the power consumed by a FPGA chip is somewhat difficult to measure and is almost unchanged for various designs.

1.8 Thesis Organization

This thesis begins with chapter 1, which gives a brief introduction on image processing, depth-map algorithm and processing platforms. Hypothesis, objectives and scope of the work is also presented in this the first chapter. In chapter 2, a literature review on the implementation of a depth-map system is presented. The basic theory of depth-map system is discussed. The main focus of this thesis is in chapter 3, where the proposed architectures and resource minimization techniques are described. Chapter 4 presents the result of the work. Comparisons between our different architectures and between our system and other systems are also made in this chapter. Chapter 5 concludes the work done in this research and proposes possible further work.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

A typical depth-map processing system includes an image acquisition module, a pre-processing unit, a disparity processing unit and other modules such as memory and display modules. The block diagram of a depth-map processing system is shown in Figure 2.1.

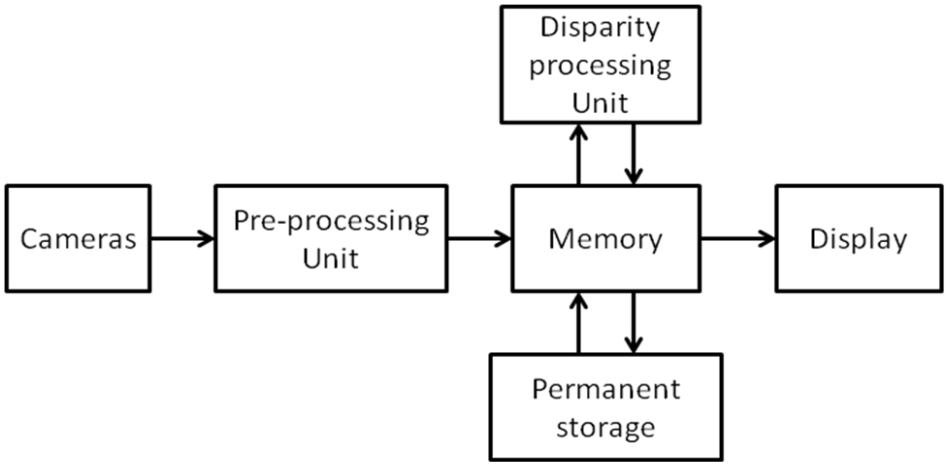

Figure 2.1 The fundamental components of a depth-map processing system

In Figure 2.1, the cameras capture and digitalize images from at least two camera viewpoints. Output of the camera units should be in digital format. Otherwise, there should be an analog to digital converter (ADC) which converts the analog image into digital form.

After that, the digital image is fed into the pre-processing unit. The pre-processing unit performs image rectification and transformation from color images to grayscale images. Image rectification is important to increase the accuracy and reliability of the disparity algorithm. This step will be discussed more detail in the subsection 2.3.1 – Image rectification. The disparity processing unit implements the depth-map algorithm and the result is put into the memory for display, storage or further processing. This is the main processing unit of a 3D vision system. Each stereo vision system is characterized by the disparity algorithm and the architecture implemented in this module.

This chapter reviews the fundamental theory and previous works for each of these parts in the system in the flow from input to the output. Available methods for stereo camera calibration are presented. The pre-processing procedures along with a detailed description of the depth-map algorithms theory are stated. The implementation of the depth-map algorithms on different platforms such as personal computer (PC), digital signal processor (DSP), (FPGA) and application specific integrated circuit (ASIC) will be discussed. This chapter will also talk about the resource minimization techniques for FPGA platform. For each of the above issues, several previous works are highlighted as examples.

2.2 Cameras Calibration

The cameras module in Figure 2.1 is characterized by the way the cameras are aligned. There are two ways of calibrating the cameras, one is the cameras system with fixed optical axes and the other one is camera system with rotatable axes. We will discuss these two camera calibration techniques in sections 2.2.1 and 2.2.2.

2.2.1 Fixed optical axes camera system

For the fixed optical axes systems [5, 11], the optical axes of the two cameras are aligned in parallel, with a distance l between two optical centres. The Figure 2.2 shows an example of a fixed optical axes camera system.

Figure 2.2 Fixed optical axes camera system [1]

In Figure 2.2, two cameras are mounted to an aluminum bar, pointing to the same direction. The two cameras are able to take pictures of the same scene from two different viewpoints. The pictures are then used in the processing unit to produce the depth-map image. Figure 2.3 describes how the depth image is estimated using this type of camera calibration.

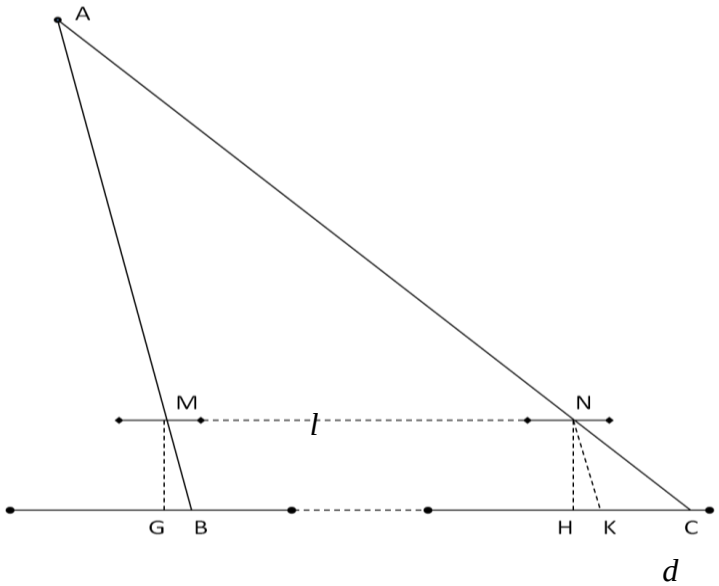

Figure 2.3 Fixed optical axes stereo vision system geometry

In Figure 2.3, M and N are the two optical centres. Let l = MN be the distance between two optical centres M and N. B and C are the projection of a point A on the focal plane of the left and right cameras, respectively. Lets NK // AB. Since the two triangles ABC and NKC are similar, we have:

$$\frac{AB}{NK} = \frac{BC}{KC}$$

$$AB = \frac{BK + KC}{KC} \times \sqrt{NH^2 + HK^2} \tag{2.1}$$

Lets KC = d is the displacement of the projection of A on the right image compared to the left image. We also have MG = NH is the focal length f, and GB = HK. So,

$$AB = \frac{l + d}{d} \times \sqrt{f^2 + GB^2} \tag{2.2}$$

GB is the distance from the pixel being evaluated to the center of the image. So, finding GB is straightforward. The problem of finding the distance from A to the left camera (AB) is resolved by finding the displacement d. Since the size of a pixel on the camera's sensor is known, it is only necessary to find the number of pixels that the projection of A has been shifted. Then KC = d = μ.P, where μ is the size of a pixel and P is the number of pixels between the two points K and C.

The equation (2.2) leads to

$$d = \frac{l\sqrt{f^2 + GB^2}}{AB - \sqrt{f^2 + GB^2}} \tag{2.3}$$

The term AB in the denominator implies that the objects closer to the camera give large displacement while distant objects give smaller displacement. On the disparity image, distant objects look darker than the objects close to the cameras.

This type of camera calibration is popular in many researches [8, 11]. Divyang K. Masrani in [11] used this camera calibration technique to test his method of expanding disparity range using temporal information available in a video sequence without recalculating the disparity of a number of pixels in a frame. To formulate the disparity algorithm, Christos Georgoulas in [5] drew the camera system geometry in the way that the camera lens centers are at the back and the image plane is in between the object and the camera centers. Although this camera geometry model is applicable with the assumption that the distance from the lens to the image plane is insignificant compared to the distance from the object to the camera, but to make it more accurate, in this thesis, the camera system was drawn as shown in Figure 2.3 with the lens in between the object and the image plane. In [6], Stefania Perri created a coordinate system where he separate the three components longitude *x*, latitude *y* and depth *z*. In his paper, the depth component *z* was calculated. But there no big difference between (2.2) and the method described in [6] because by using a simple coordinate transformation technique, the method in [6] becomes equivalent to (2.2).

2.2.2 Rotatable optical axes camera system

In rotatable axes stereo vision systems [12], the cameras are rotated about two vertical axes. This is done by mounting the cameras on two servo motors as shown in Figure 2.4. The ability to capture images of closer objects is an advantage of this camera system. By rotating about the vertical axes, the two cameras are able to point to a closer common view.

Figure 2.4 Rotatable optical axes camera system [12]

Figure 2.5 describes the concept of the rotatable optical axes camera system. *X*, *Y* and *Z* are the points on the objects being observed. *M* and *N* are the optical centers. *EF* and *PQ* are the vertical projections of the sensor plane.

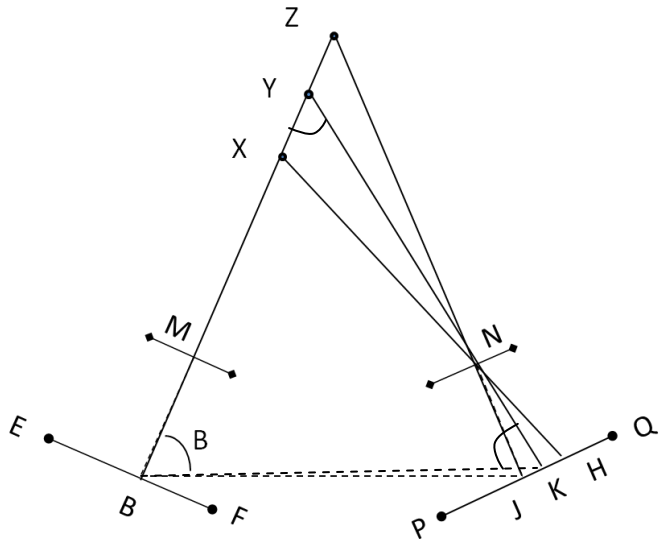

Figure 2.5 Rotatable optical axes stereo vision system geometry

B is the projection of *X*, *Y* or *Z* on the left camera sensor. *H*, *K* and *J* are the projection of *X*, *Y* and *Z* on the right camera sensor, respectively. A disparity matching algorithm is applied on the two images. If *H* matches with *B* then *B* is the projection of *X*. If *K* is the match point of *B* then *B* is the projection of *Y*, and so on.

Assume that *B* and *K* are matched. In the triangle *BYK* we have

$$\frac{YK}{\sin B} = \frac{BY}{\sin K} = \frac{BK}{\sin Y} \tag{2.4}$$

$$BY = \frac{BK \sin K}{\sin Y} \tag{2.5}$$

Because $Y = (\pi - B - K)$, let $BK = l$ be the distance between the two projection of *Y* on the camera system, then

$$BY = \frac{l \sin K}{\sin(\pi - B - K)} = \frac{l \sin K}{\sin(B + K)} \tag{2.6}$$

Angle detectors are used to measure the angle *B* and *K*. After that, the distance from the object to the camera is calculated using formula (2.6). Kazuhiro Shimonomura [12] installed on two cameras on two stepper motors. Another two stepper motors were used for rotating the cameras on the horizontal axes (looking up and down) in an attempt to mimic human eyes.

The fixed optical axes stereo vision system exhibits a minimum distance where the system is unable to estimate the distance of any point closer than that, because the projection of that point falls outside the maximum disparity range [1]. The rotatable optical axes system can estimate the distance of very close objects [12]. However, a precise mechanical system is required. System response is slow due to the mechanical rotation. It also raises controlling difficulty and computational complexity. Human eye is a special case of rotatable optical axes stereo vision system, where the projections of the object always fall into the two *macular*. It is able to estimate the distance from as close as few centimeters to infinity.

In this research, the fixed optical axes camera calibration technique was used to

avoid the difficulty of building a complex mechanical system. The problem of minimum visible distance is solved by reducing the distance between two cameras and increasing the disparity range. As shown in equation (2.2), when the distance between two cameras, *l*, is reduced, AB is reduced. When *d* increases, AB will be reduced. Increasing the disparity range requires additional resource on FPGA. But for this project, it is more cost effective than building a complex mechanical system.

$$AB_{\min} = \frac{l + d_{\max}}{d_{\max}} \times \sqrt{f^2 + GB^2}$$
$$AB_{\min} \approx \frac{l}{d_{\max}} \times \sqrt{f^2 + GB^2} \qquad (2.7)$$

Equation (2.7) is derived from equation (2.2) to show that the minimum visible range can be reduced by increasing the disparity range d_{max} and reducing the distance between two camera, *l*.

2.3 Pre-processing

The pre-processing unit in a general 3D vision system may perform several tasks such as contrast enhancement, image rectification and color to grayscale conversion. But in this work, only the image rectification and color to grayscale conversion will be discussed. Contrast enhancement is not discusses because the depth-map algorithm make decision about the depth based on the displacement of the objects rather than the contrast in the color.

2.3.1 Image Rectification

Depth-map algorithms attempt to find matching patterns of one image on the other image by searching on a horizontal line of that image. Thus, it is necessary that the images are rectified so that the two projections of an object are brought to the same horizontal line (epipolar line) on both images. If the cameras are well calibrated, this step is not necessary. But in most cases, the cameras are not well aligned. Sometime

they are inclined, tilted or both. Image rectification is a general term referring to processes such as image rotation, image shifting and image deformation so that a pixel of the same scene is in line with the pixel of that scene on the second image. According to Changming Sun in [13], image rectification can improve speed and reliability of depth-map estimation process. Richard I. Hartley [14] presented a mathematical theory for image rectification. Zezhi Chen et al. [15] reported a practical image rectification method, which does not require any camera calibration. The technique minimized the pixel information loss along the epipolar line and never split a region of the image.

In this research, when two cameras were installed, it was found that the two images were not on the same horizontal line. One image was shifted a few lines below the other image. In order to bring the objects to the same epipolar line, some lines on the top of an image were cut and the whole image was shifted up to be the same as the other image.

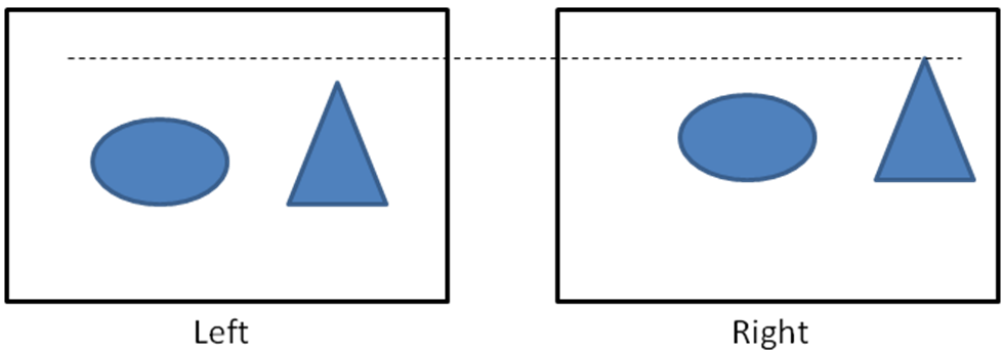

Figure 2.6 Original images before rectification

Figure 2.6 shows the original pair of images that we received from the cameras. Notice that the objects are not on the same horizontal scan-line. The top of the triangle in the left image is a few pixels lower than one in the right image. If disparity algorithm was applied on that horizontal line, it would find no matching texture between the left and the right images. In our system, the input images were shifted a few lines to bring them to the same epipolar line as shown in Figure 2.7

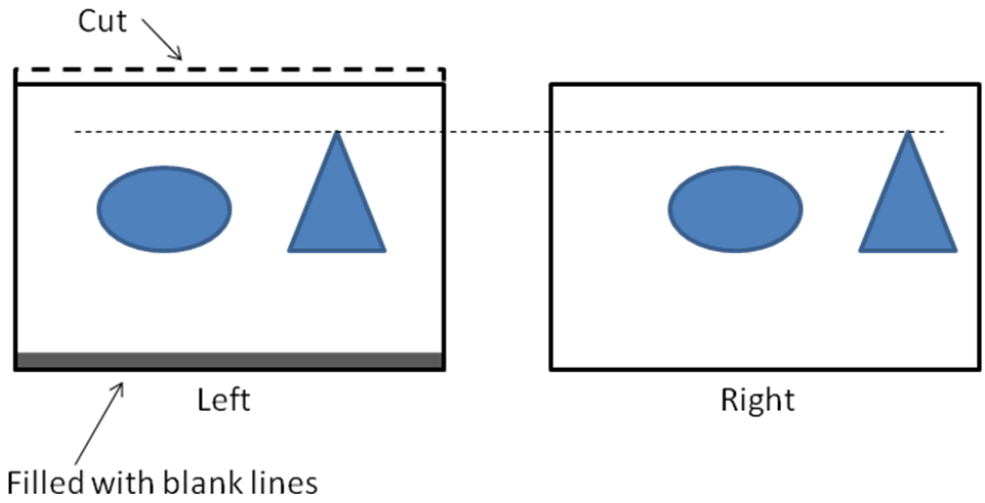

Figure 2.7 Rectified images

Figure 2.7 shows only the left image being shifted and the right image remains the same. In our actual system, the left image was shifted half distance up and the right image was moved half the distance down to keep the image size as close to 640x480 pixels as possible. This image rectification step does not affect the quality of result image but rather it makes the image smaller by a few lines.

2.3.2 Conversion from RGB Format to Grayscale Image

Depth-map algorithms use either the intensity of the pixels or the gradient between pixels to calculate the depth-map image [4]. In both cases, it requires two grayscale images. If the outputs from cameras are color images, the color images should be converted to grayscale before being used in the disparity module. To generate the grayscale image from color image, the *luminosity* method takes the average of three colors with the weight of each color regards to the perception of human eyes. Different values of these weighting factors were used in the current systems and software [16]. For example, Photoshop 5.0 used the weighting factors of.

$$Y = 0.2126R + 0.7152G + 0.0724 B \quad (2.8)$$

Another popular formula for the luminance model is [17]

$$Y = 0.2989*R + 0.5870*G + 0.1140*B \quad (2.9)$$

For images based on subtractive color space (Cyan, Magenta, Yellow), the conversion can be done by converting CMY to RGB color space before calculating the gray value. The chosen method for our system will be described in section 3.1.1.

2.4 Depth-Map Algorithms

Currently, there are a number of depth-map algorithms available [18-21]. They have a common purpose of finding the displacement between the projections of the same object on two images. Depth-map algorithms can be classified using the localism of the calculating window, which are called local and global algorithms. Besides, they can also be classified using their matching cost function or gradient versus intensity based techniques [4]. This section will describe these types of depth-map algorithms in detail.

2.4.1 Local, global and hierarchical algorithms

Disparity matching algorithms are classified into global or local (window-based) algorithms. There are also some algorithms that take advantages of these two main classes, such as cooperative algorithm or hierarchical algorithm. Of course, they exhibit more computational complexity. The local based algorithms are based on the "winner takes all" basis for each pixel. It performs disparity search for a pixel of the *reference image* and when a local maximum of similarity is found, the corresponding pixel on the *search image* is considered a match with the pixel on the reference image (see Figure 2.8). The disparity value of a pixel depends only on the gray level of the pixels in its window. The local based algorithm does not take into account the smoothness of the disparity value for other pixels of the image explicitly. However, it makes implicit smoothness assumption in each window. D. Chaikalis et al. [22] has implemented this local search algorithm to calculate disparity value for integral

photography (IP) image and video compression purpose. In [3], Bongsoon Kang et al. implemented local disparity algorithm on hardware with vertical strip structure to minimize resource usage.

The global algorithms make explicit smoothness assumption by combining data and smoothness terms in the matching cost expression. The disparity is found when the global cost expression in (2.10) is minimized.

$$E(d) = E_{data}(d) + \lambda E_{smooth}(d) \quad (2.10)$$

Global algorithms perform well on gray continuity area but have a serious limitation on the edge of the object where the disparity value changes abruptly.

The hierarchical algorithms perform disparity search from coarse levels to fine levels. The disparity result of coarser levels is used as constraint to calculate the disparity value of finer levels [9]. To reduce the computational cost, the images are down sampled to a smaller size for coarse disparity calculation. Pascal Fua in [9] presented a coarse to fine algorithm to calculate the disparity map with multiple cameras. In his paper, an interpolation algorithm was used in the post-processing step to fill the textureless and occluded areas.

2.4.2 Depth-map algorithms classification using matching cost function

Disparity algorithms are also categorized using their matching cost. Matching cost is a function of similarity or difference with the disparity value *(d)* acting as a variable. Generally, disparity algorithms try to find the maxima of the similarity function *f(d)* or the minima of the difference function. Figure 2.8 illustrates the similarity function. When the maximum of similarity between two windows is found, its corresponding value of disparity (d_{match}) is the solution to the depth-map algorithm.

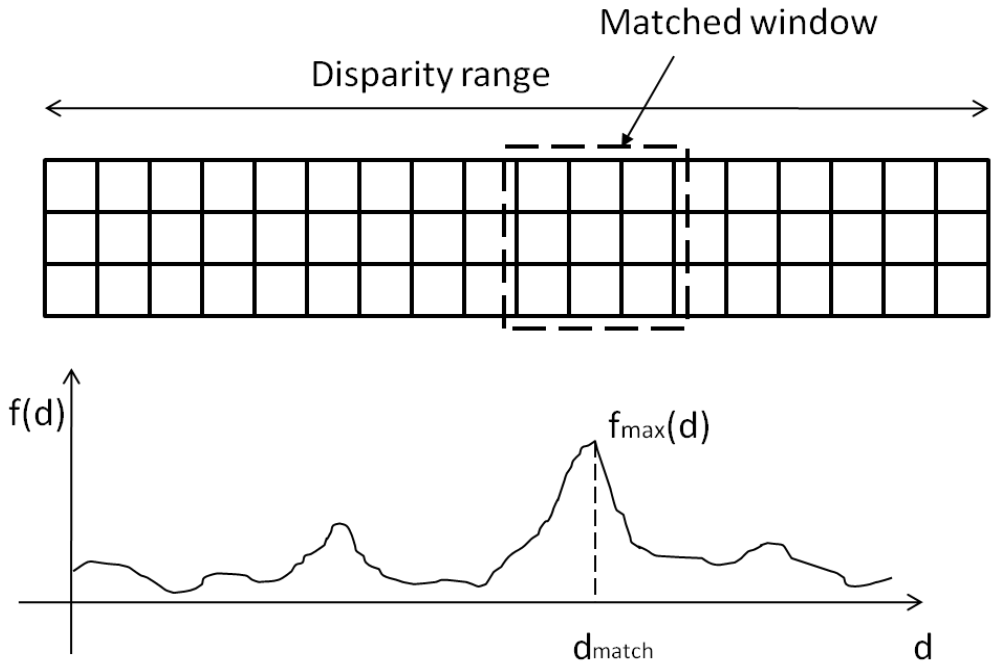

Figure 2.8 Matching cost function

The common matching cost functions are *square-intensity-difference* (SD), *sum-of-squared-difference* (SSD), *absolute difference* (AD) and *sum-of-absolute-difference* (SAD). For example, the matching cost functions of the SAD algorithm with the window size of 3x3 pixels is

$$f(d) = \sum_{i=1}^{9} | E_i(d) | \tag{2.11}$$

where *f* is the matching cost function, *d* is the current disparity value and E_i is the difference in intensity between two pixels at the same position of the two comparing windows. The two windows are considered "match" when the disparity value *d* minimizes the sum of absolute difference function. The corresponding value of *d* is mapped into the disparity image. Bongsoon Kang, *et al*. [3] implemented the AD algorithm on FPGA that find the disparity value by maximizing the similarity function. The system achieves the frame rate of up to 15 frames per second (fps). D. Chaikalis, *et al.* [22] implemented the SAD algorithm on hardware. With the frame size of 1024x768 pixels and the disparity range of 64 pixels, the system reach real-

time performance of 31 fps. Stefania Perri *et al*. [6] presented a hardware system using SAD algorithm with the frame size of 512x512 pixels and the maximum disparity range is 255 pixels. The system works at the frame rate of 25.6 fps with the clock frequency of 286 MHz.

2.4.3 Gradient versus intensity based algorithms

The disadvantage of intensity based matching cost functions is that they are sensitive to the camera gain and bias. If the two cameras have different gain (different light sensitivity), the intensity value of each pixel may vary and the algorithm becomes vulnerable. There are algorithms that take the gradient of pixel values as matching cost function. These algorithms are insensitive to camera gain [4, 7].

2.4.4 Dense and spare depth-maps

Some depth-map algorithms do not calculate the disparity value for all pixels on the image [9]. Instead, they only calculated the disparity value for the pixels at the edge of the object. The disparity value is then propagated through the image by using interpolation, diffusion or voting mask. Disparity value of the pixel at the edge of the object often has high confidence level. These algorithms are claimed to generate less errors. The algorithms with high criteria of truthful disparity reject most of the uncertain disparity pixels. They are called the sparse disparity algorithms. Spare depth-map algorithms were used in by Pascal Fua [9] and by F. Solari *et al*. [23]

Opposed to sparse disparity algorithm is the dense disparity algorithm which calculates the disparity value for all the pixels in the image. A method for dense depth-map estimation was implemented in by Luis Alvarez *et al*. [24] Each of these algorithms has different applications. For example, robotic system requires spare but high confident disparity map for road navigation. 3D image reconstruction system requires dense disparity image which has the disparity value of all pixels on the image, even the textureless and occluded areas. To generate denser depth-map from a sparse depth-map image, J. Ralli *et al*. [25] proposed a method of spare disparity

densification using gradient based voting mask. The algorithm works with the assumption that on the same object, disparity value changes gradually and on different objects, disparity value is discontinuous and separated by the edge of the object.

2.5 Implementation of depth-map algorithms

Depth-map algorithm was implemented on various processing platforms. In general, as described in section 1.4, there is a trade-off between flexibility and speed of a system. Pascal Fua [9] implemented his algorithm on a workstation (PC), a DSP and also a *Connection Machine*. On the workstation, it takes approximately 2 minutes 30 seconds to process images with the resolution of 256x256 pixels and the disparity range of 50 pixels. In other word, the frame rate is only 0.4 frames per minute. Of course, his software program is highly flexible. On the multi-DSP 96002 board, the system takes 15 seconds to complete processing a frame. Thus, the frame rate is 4 frames per minute. In [26] Dustin Lang and James J. Little programmed the graphic hardware to perform SAD algorithm and achieve the frame rate of 10 fps for the image of 640x480 pixels or 25 fps for the image of 320x240 pixels.

Implementation of depth-map algorithm on FPGA is popular as presented in [1, 3, 6, 22]. Divyang K. Masrani *et al.* [1] implemented the phase correlation depth-map algorithm on a platform using four Altera's Stratix FPGAs. The system is able to process a pair of images with the resolution of 640x480 pixels. The disparity range is up top 128 pixels and the frame rate is 30 fps. B. Kang *et al.* [3] used Altera's APEX20K1000-EBC652-3 FPGA to perform absolute different (AD) depth-map algorithm with the image size of 320x240 pixels, 64 pixels disparity range and achieved the frame rate of 15 fps. S. Perri, *et al.* [6] have implemented SAD algorithm on Xilinx FPGA for the image size of 512x512 pixels, disparity range of 255 pixels and the frame rate is 25.6 fps. D. Chaikalis, *et al.* [22] had used the Xilinx Virtex XCV-2000E FPGA to implement SAD depth-map algorithm. The system works with the frame size of 1024x768 pixels, 8x8 pixels window size and 64 pixels disparity range. The frame rate of this system is 31 fps.

2.6 Resource Minimization Techniques

As pointed out in chapter 1, resource minimization is one of the most challenging problems for FPGA implementation of depth-map algorithm. Researchers have been focusing on modifying the depth-map algorithm to minimize the amount of computational load and therefore minimize the resource usage [8, 10, 11]. Pixels with small change of depth are not recalculated but disparity value is taken from the previous frame. Because the disparity values of some pixels are propagated from previous frames, it takes time (in term of a few frames) for the disparity map to be filled up. This leads to slow convergence and high failure rate. Divyang K et al. presented a method for expanding the disparity range without increasing the computational load and therefore save resource usage [11].

Another possible method of resource minimization is by designing a resource saving architecture. The depth-map algorithm is not changed but instead, the system architecture is modified. The system preserves fast response to the change of depth while resource usage is reduced. To the best of the author's knowledge, there is currently no work done on this area. For that reason, the research in this thesis focus on developing a new hardware architecture which implements a regular SAD based depth-map algorithm with reduced resource utilization.

2.7 Summary

In this chapter, a literature review on camera calibration, pre-processing, depth-map algorithms and resource minimization techniques were presented. There are two methods for camera calibration, which are fixed optical axes system and rotatable optical axes system. Mathematical model of each camera calibration technique was developed. The fixed optical axes method was chosen because it does not require complex mechanical system. Depth-map algorithms are classified as local, global and hierarchical algorithm. They can also be classified using matching cost function, intensity and gradient, or spare or dense depth-map. The algorithm used in this research is local method with the intensity based sum-of-absolute-different (SAD) matching cost function. The result is a dense depth-map image. Resource utilization

on FPGA can be minimized by modifying the depth-map algorithm or designing different architectures. This work implements the original depth-map algorithm and designs a few different architectures in order to minimize resource utilization.

CHAPTER 3

THE PROPOSED ARCHITECTURE

In this chapter, the design and functionality of a stereo vision system is presented. We then propose several techniques to improve system performance and reduce resource utilization. Lastly, these techniques were implemented in five different disparity matching architectures to compare resource usage and performance. The highlight of this work is the effect of these techniques on resource utilization and the performance of different proposed architectures.

3.1 Design of a Stereo Vision System

Figure 3.1 shows the block diagram of our system. There are three major blocks being implemented on FPGA. The first block of the system consists of two camera interface modules, which capture the pixel data from two cameras and feed into two FIFOs. Pixel data is then written into external memory by a multi-ports memory controller. The cameras need to be configured to appropriate parameters such as the parameters which set the frame size, brightness, and sequence of the output. This task is taken by a camera configuration module inside the camera interface module. The second block is the multi-ports memory controller. It manages all the memory access requests from other modules and ensures no memory contention. The third block represents the processing unit with a disparity matching module. Other blocks are clock circuitry module and VGA controller module. In this section, we will describe our system according to the flow of data.

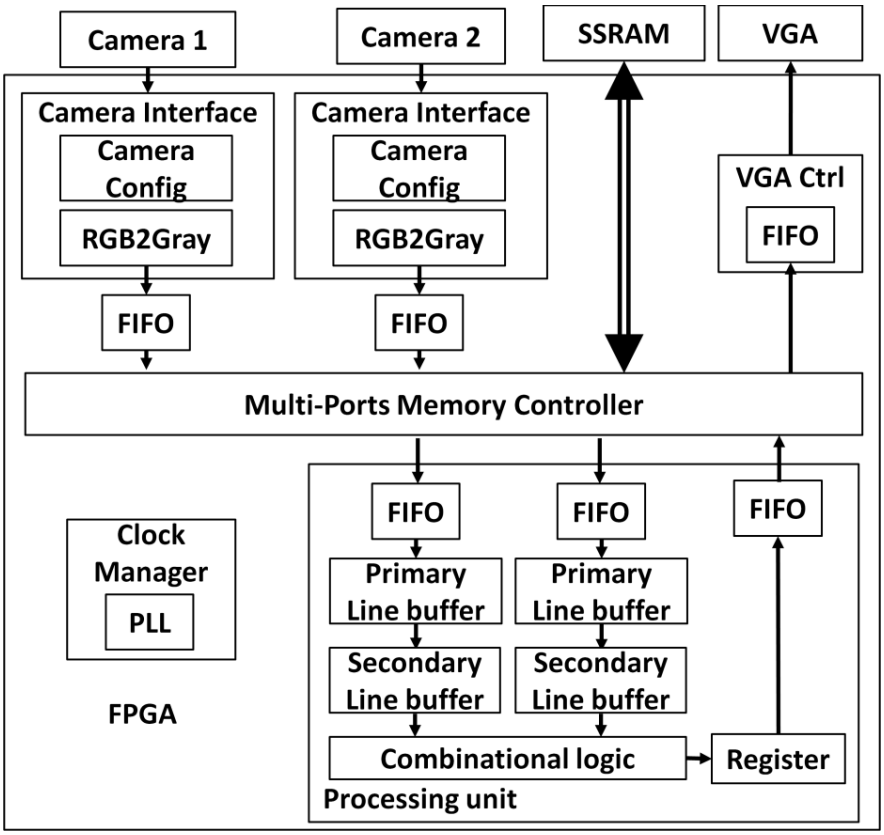

Figure 3.1 Block diagram of the system

3.1.1 Image Preparation

This section discusses the image capturing procedure and how the color image is converted to grayscale image in the camera interface module. Output of the camera is in RGB format and RGB to grayscale conversion is needed because the SAD depth-map algorithm requires two grayscale images to be processed. In this design, two Terasic's THDB-D5M [27] cameras were used as input. The cameras were chosen because they have an interface which is compatible with the DE2-70 board. Figure 3.2 shows the D5M camera pixel array which contains 2592 columns and 1944 rows in the active area. Surrounding the active area of the image is a frame of boundary and dark pixels. In default operation, only pixels in the active region is read, giving the image of 2592x1944 pixels (Figure 3.2).

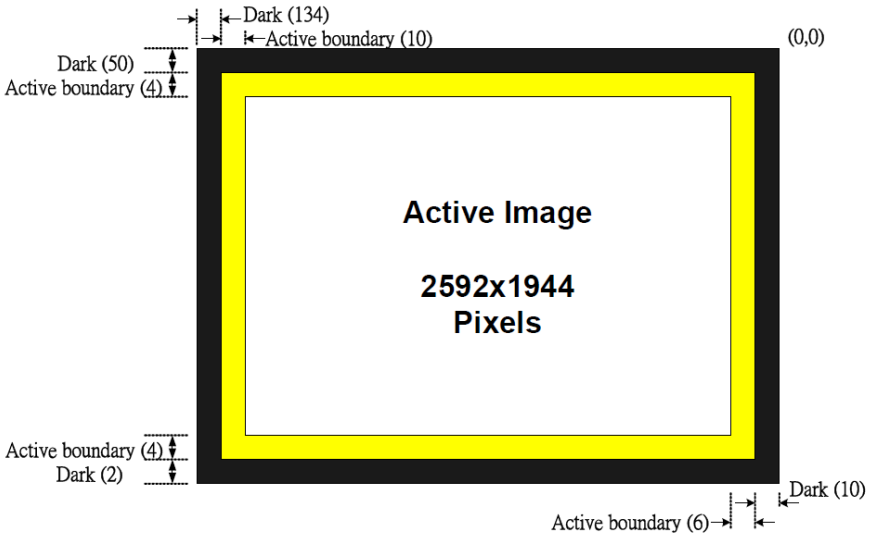

Figure 3.2 Pixel array description [27]

The D5M camera is a charge coupled device. When the image sensor is exposed to the light, the charge level will increase proportionally to the light intensity and the exposure time. On the image sensor, each pixel is covered with a color mask. There are three types of color masks, corresponding to three colors: red, green and blue. The charge level of the pixels under a red color mask is affected only by the red light. And the charge level of the pixels under green or blue mask is influenced only by the green or blue light, respectively. That charge level is converted into electrical voltage in the analog signal chain, and then digitalized by a 12-bit analog-to-digital converter. Output of the camera is a series of 12-bit binary numbers. One pixel is output every clock cycle. The color of a pixel is determined by the time of its occurrence with regard to the frame valid (*FVAL*) and line valid (*LVAL*) signal. A frame is defined when the *FVAL* signal is high. The rising edge of *FVAL* signal is the start of a new frame. And the falling edge of *FVAL* sets the end for the current frame. The time interval between the end of the old frame and the start of a new frame is called the vertical blanking. An image line is defined when the *LVAL* signal is high. When *LVAL* signal is low, there will be no output to the data port of the camera. This time interval is called the horizontal blanking. When both *FVAL* and *LVAL* signals are high, one 12 bits output pixel is latched to the data bus at every rising edge of the camera clock. The readout timing is shown in Figure 3.3 below.

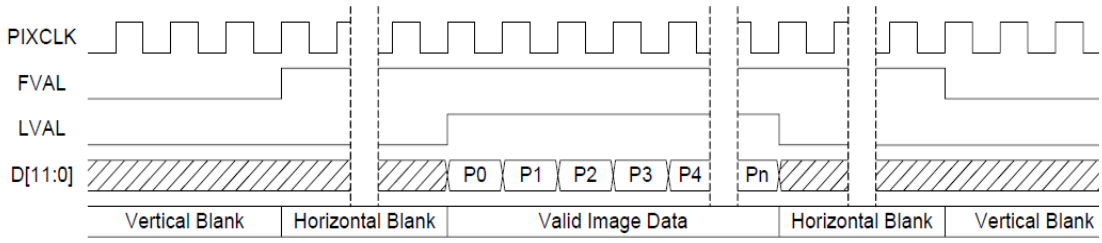

Figure 3.3 Pixel output timing [27]

In Figure 3.3, *PIXCLK* is the camera clock. *FVAL* and *LVAL* are the frame valid and the line valid signals. *D[11:0]* is the twelve-bit data bus. *Pi* is the pixel number in a line. There are 2592 pixels in a line and 1944 lines in a frame. But on the above figure, only six pixels were shown when *LVAL* is high. And only one line was shown when *FVAL* is high. There is a horizontal blank between two lines and a vertical blank between two frames. Since a pixel is latched at the positive edge of the *PIXCLK*, the camera interface module was designed to capture it at the negative edge of *PIXCLK*.

The pixel color map and the read out sequence are shown in Figure 3.4 below.

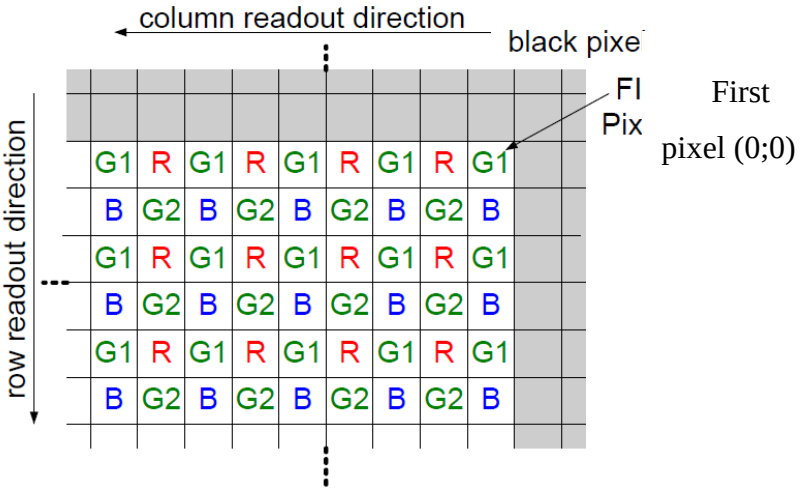

Figure 3.4 Pixel color map and read out direction [27]

Figure 3.4 shows that the first row from the top consists of two colors *G1* and *R*. And the pixels on the second row are alternating between *B* and *G2*. The first pixel being read when the *FVAL* signal goes high is the pixel at the top right corner (*G1*). And the image pixels are read out in a sequence from right to left and top to bottom as shown in the figure. A *line* and a *pixel* signal were used to identify *R*, *G1*, *B* and *G2*

components. At the first pixel, the *line* and *pixel* signals both have the value of 0. The *pixel* signal value is altered every clock cycle and the *line* signal is altered after each line. As a result, the *G1* color appears at the output of the camera when {*line*,*pixel*} = 00_2. The *R* component is available at the output when {*line*,*pixel*} = 01_2. The *B* component is there when {*line*,*pixel*} = 10_2. And the *G2* color should be captured when {*line*,*pixel*} = 11_2.

The D5M camera has 256 internal-16-bits-programmable registers that control many aspects of the camera such as image size, PLL, read mode and black level. These registers can be programmed through a serial port. The rows are defined by row start, row size, bin, skip, and row mirror registers. Similarly, the columns are defined by column registers. These registers were set to appropriate values so that the camera outputs an image of 960 rows and 1280 columns of Bayer colors (R, G, B). Note that this is not the default image size. The reason of changing from 2592x1944 pixels image size to 1280x960 pixels is to fit with the display size and the processing unit. From 1280x960 pixels of red, green and blue colors, a grayscale image with the resolution of 480x640 pixels was created. The camera output pixel is 12 bits, but to make the design simple, only 8 most significant bits were used. The four least significant bits were truncated. This actually reduces the resolution of each pixel. But 8 bits depth is sufficient for most applications. The camera interface module was designed such that for the odd rows (1,3,5...) of the color image, it will capture only the R element and ignore the G1 element. An internal memory block of 640x8 bits was used to store 640 pixels in one row of the Red component before the Blue and Green components arrive. B and G2 components are latched into the camera interface module in the next row of input. A temporary register were used to store the B color pixel for one clock cycle. The RGB to grayscale conversion is performed for each pixel as soon as the G2 component is fed into the camera interface module. This is done in a combinational logic circuit and the output is read at the negative edge of PIXCLK.

The common formula to convert from RGB to grayscale is [17]

$$Y = 0.2989*R + 0.5870*G + 0.1140*B \quad (3.1)$$

The green has the highest weight because human eye is more sensitive to green. However, machine does not have different sensitivity for different colors. Therefore, all three colors should be treated equally. When the colors are treated the same, the luminance formula will be

$$Y = 1/3\ R + 1/3\ G + 1/3\ B \quad (3.2)$$

But implementing multiplication or division on FPGA is very time consuming and resource demanding. Thus, the equation (3.2) was modified so that the system uses only addition and shift operation. The formula used is

$$Y = 1/4\ R + 1/4\ G + 1/4\ B \quad (3.3)$$

$$Y = (R + G + B)/4 \quad (3.4)$$

The result image using formula (3.3) or (3.4) is 25% darker than the one using (3.2) because the sum of intensity is divided by 4 rather than 3. But this does not pose significant effect to the calculation of disparity value. Division by 4 can be made as a 2-bits shift right operation or simply truncating the two least significant bits. Equation (3.4) was used instead of (3.3) because significant bits may be lost if divisions (or shift right) were performed before addition. In the following example, the three 8-bits numbers are added and the result is divided by 4.

$R = 00101101_{(2)}$ $G = 01001011_{(2)}$ $B = 00110110_{(2)}$

If formula (3.3) is applied, the result will be

$$Y1 = 001011_{(2)} + 010010_{(2)} + 001101_{(2)} = 00101010_{(2)}$$

If formula (3.4) is applied, the result will be

$$Y2 = (00101101_{(2)} + 01001011_{(2)} + 00110110_{(2)}) >> 2$$

$$Y2 = 00101011_{(2)}$$

Comparing Y1 and Y2, the least significant bit is lost when the formula (3.3) is applied because the shift right operation is performed before the addition. Only the grayscale pixel is latched into a FIFO, waiting to be written into the memory. In this way, a huge amount of memory bandwidth is saved for not storing the raw image data and re-accessing it again. In fact, if the two cameras were well synchronized, the grayscale pixel could be sent directly to the processing unit without being written to the memory. However, the two cameras that we are using are not sending the pixel data at the same time. So the two input images need to be stored in memory prior to being processed.

3.1.2 FIFO

Because the cameras, the SRAM, the processing unit and the VGA are working at different clock frequency, FIFO is an essential component in this system. It has two roles, i.e. passing data to other clock domains and serving as a buffer for immediate access. This is a nice two-in-one component.

The synchronization of multiple changing signals between two different clock domains is a difficult task [28]. This is because of the metastability phenomenon [28] that happens when the signal is sampled in the other clock domain and the different propagation delay of different signals. Metastability is caused when a signal is sampled during its transition. If a data signal transition violates a register's set up time, t_{SU}, or hold time, t_H, requirements, the output of the register may go into a metastable state [29]. In metastable state, the signal may take an amount of time longer than the clock to output time (t_{CO}) of the register to be stable. The signal can be resolved to a new value or goes back to its old value after the metastability period. If the signal is not resolved after t_{CO}, and if it is read to another register or fed into a combinational logic, it will cause a system failure because the data is unknown and the output will be unpredictable. In Figure 3.5, the occurrence of the clock edge during the signal transition (between t_{SU} and t_H) causes the first signal (Output A) to be metastable and resolved to new value after t_{CO}. The second signal (Output B) is resolved to its old value after t_{CO}.

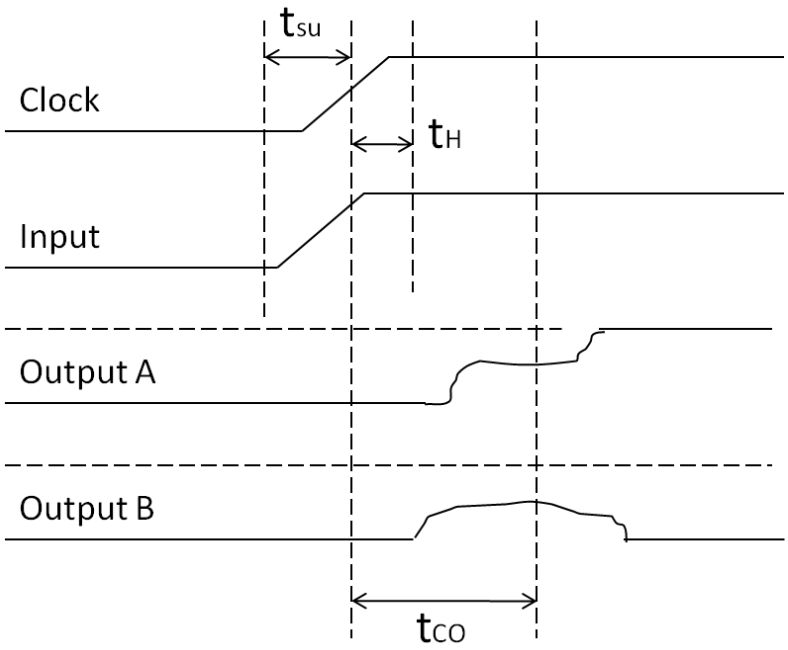

Figure 3.5 Metastability timing

Circuit failure due to metastability is minimized by introducing a synchronization registers chain (Figure 3.6). Synchronization register chain is a sequence of two or three registers close to each other, clocked by the same clock frequency. Output of one register is connected to the input of the next register in the chain. In Figure 3.6, there are three registers. The one in the Clock1 Domain is responsible for sending data. Two registers in Clock2 Domain are the receiving registers. The first register in Clock2 Domain acts as the synchronization register that takes the data from the sending clock domain. Data is then passed into the second register in the synchronization chain. If there are more registers in the synchronization chain, they will be connected in series. Output of the last register is used in the combinational logic circuit of the receiving clock domain. If there are *n* registers (*n* = 2,3,4…) in the synchronization chain, it will take *n* clock cycles to pass data from the sending clock domain to the combinational logic of the receiving clock domain. It is said that the latency of this synchronization chain is *n* clock cycles.

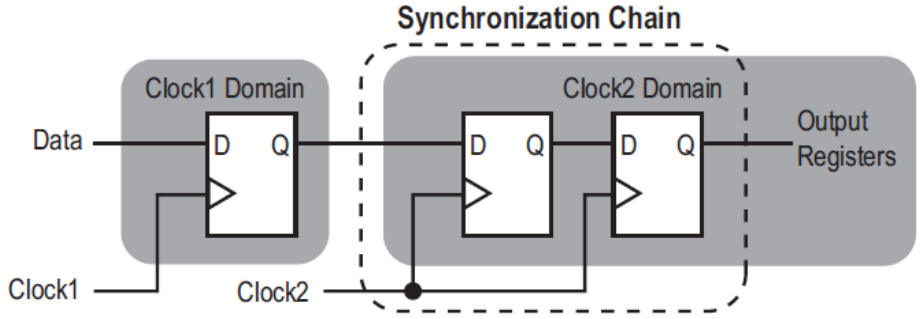

Figure 3.6 Synchronization chain [29]

The purpose of adding registers in the synchronization chain is to give the metastable signal additional time (a few more clock cycles) to resolve to a known value before it is used in the rest of the design. When more registers are added to the synchronization chain, probability of system failure due to metastability will be lower. This increases the mean time between failures (MTBF) of the system. A higher MTBF (such as hundreds or thousands of years between metastability failures) indicates a more robust design. However, adding a register adds an additional latency stage to the synchronization logic, so designers must evaluate whether that is acceptable. In our system, two registers are used in each synchronization chain. This is justified because our system is not a life critical medical system or nuclear power device which requires very high stability. Processing speed is important in our system and long latency is not preferred. Besides, two registers synchronization chain is typical in most system.

Synchronization chain works well for a single bit signal. But when passing multiple bits of data or control signals to other clock domain, a small skew between the signals could cause the two signals to be synchronized into different clock cycles within the new clock domain [28]. For example, a data register with two control signal “load” and “enable”. If the two signals are synchronized into different clock cycle, it would cause the data not to be loaded to the register.

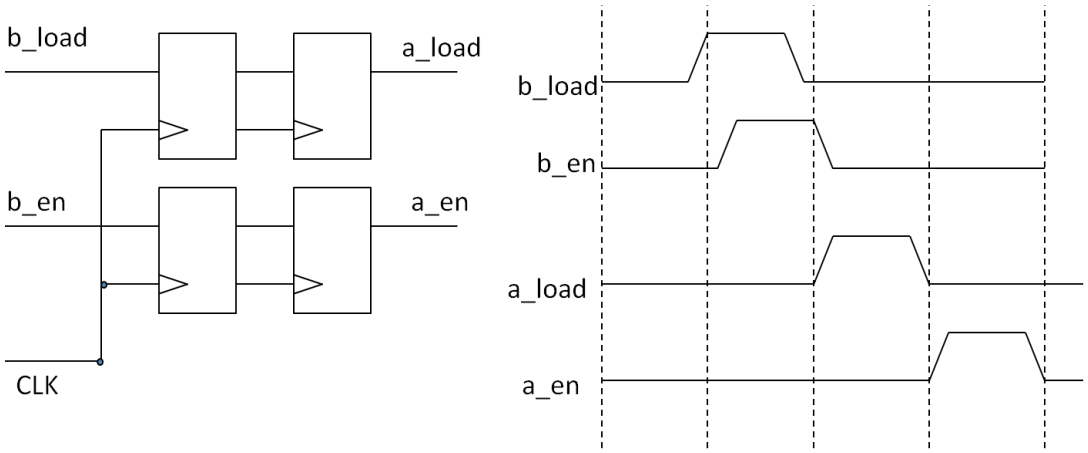

Figure 3.7 Problem of passing multiple control signals between clock domains

In Figure 3.7, the *a_load* and *a_en* signals are supposed to be asserted in the same clock cycle. However, due to the skew between *b_load* and *b_en*, a clock edge that occurs right between the two transitions of the two signals sampled them into two different clock cycles. In the third clock cycle, *a_load* is asserted but *a_en* is low, so the data is not loaded into the register. In the fourth clock cycle, *a_en* is high but *a_load* is low, the data is also not loaded. In our system, 8 bits pixel data needs to be transferred from the camera to the memory controller, from the memory controller to the processing unit and to the VGA controller module. If a clock edge occurs at the transition of the data word, the pixel data might be sampled into a random number.

To resolve the above problems, FIFO was used in our designs to safely pass multi-bit data words from one clock domain to another. FIFO is a dual port internal memory. One port is controlled by the sender which puts data into the memory as fast as one data word per write clock cycle [30]. The other port is controlled by the receiver, which pulls data out of memory one data word per read clock cycle (see Figure 3.9). Data signals are given sufficient time to be stable inside the FIFO before it is used in the receiving clock domain. Conceptually, the task of designing a FIFO like a dual port memory seems to be easy. The difficulty associated with doing FIFO design is related to generating the FIFO pointers and finding a reliable way to determine full and empty status of the FIFO. The FIFO consists of a memory array, a read pointer and a write pointer. When the reset signal is asserted low, both pointers will be reset to zero, the empty signal is asserted high. Otherwise, the write pointer

always points to the next location to be written and the read pointer always points to the next location to be read. On a FIFO-write operation, the memory location that is pointed to by the write pointer is written, and then the write pointer is incremented to point to the next location to be written. After the last location in the FIFO memory array is written, the write pointer wraps back to the first location.

When the FIFO is reset, assertion of the empty signal indicates that there is no data in the FIFO. Attempting to retrieve data from the output of the FIFO is not allowed. As soon as the first word location in the FIFO is written, the empty signal is cleared. The read pointer always points to the next location to be read. So the receiver logic doesn't have to use two clock cycles to read a data word. If the receiver first had to increment the read pointer before reading a FIFO data word, the receiver would clock once to output the data word from the FIFO, and clock a second time to capture the data word into the receiver. That would be needlessly inefficient.

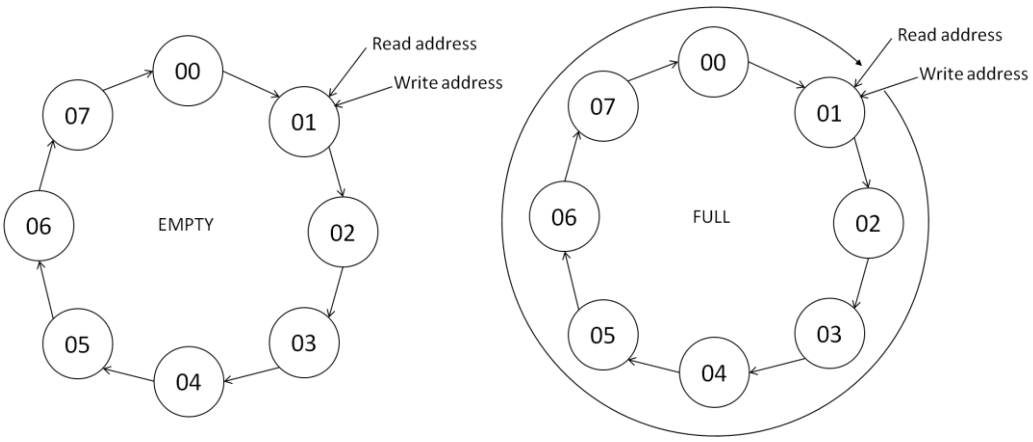

Figure 3.8 FIFO full and empty condition

In Figure 3.8, the read pointer is incremented to the next location after the data word is read. When it caches up with the write pointer, it will point to the last word in the FIFO. The FIFO is now determined to be empty and the empty signal should be asserted. So the empty condition is simply determined by comparing the read pointer and the write pointer. If they are equal then the FIFO is empty. However, when the write pointer finishes one round and caches up to the read pointer again, the two

pointers are also equal but the FIFO is full (Figure 3.8). This is a problem. The FIFO is either empty or full when the pointers are equal, but which? To distinguish between the full and empty status, one extra bit was added to each pointer. When the write pointer increments past the final FIFO address, it toggles the most significant bit (MSB) while setting other bits to zero. The FIFO is empty when the both pointers are equal, including the MSBs. If the MSBs are different but the other bits of the two pointers are equal, it means the write pointer has wrapped one more time than the read pointer, and the FIFO is full.

As mentioned above, to determine the full and empty status of a FIFO, the read pointer and the write pointer need to be compared. But they are generated in two different clock domains, so the read pointer must be synchronized to the write clock domain and the write pointer must be synchronized to the read clock domain. To do that, synchronization register chains were used. However, the pointer itself is a multiple bits signal. If the pointer is a binary number, when two or more bits change at a time (let say from $0011_{(2)}$ to $0100_{(2)}$), due to the skew between the signals, it might become any number after being sampled in the receiving clock domain. The sampled pointer will point to an unpredictable word location in the range of the FIFO. Solution for this problem is by using Gray code counter to design the pointer instead of binary counter. Because the gray code counter only change one bit at a time, so if the clock edge occurs at the transition of the Gray code counter, the synchronized value will either be the old value or the new value. The FIFOs full and empty signals are generated by using the pessimistic value of the synchronized pointers. So the FIFO is never overflow or underflow. For this design, the FIFO described in [30] was used with some modification of word length and number of location as will be described later.

FIFOs are used mainly for passing data to other clock domain. But in our design, FIFO also serves as a line buffer. It provides transparent access to the memory because the memory access request from any module can be served immediately by reading/writing to the FIFOs. Accessing external memory has latency. For this design, two clock cycles are needed to initiate the read operation. Also, many modules are trying to access the memory at the same time. Without FIFO, all modules would have

to work at the same clock frequency. It takes three clock cycles to read one memory location to serve a module, another three clock cycles to serve the second module, and so on. During that time, all other modules have to be idle and wait for the SRAM. In our system, some modules such as the camera and the VGA have high priority and require immediate access to the memory. Thus, an internal line buffer is necessary to buffer the data before writing to a slower external memory. The FIFO described in [30] has 8 word locations. But with 8 locations, the FIFO will be full/empty too quickly; causing excessive overhead clock cycles are needed to access the SRAM. The FIFO was modified to have 32 locations so that it can serve the purpose of being a line buffer better. Also, because the SRAM word length is 32 bits. So the FIFOs word length was modified to be 32 bits to match with the SRAM.

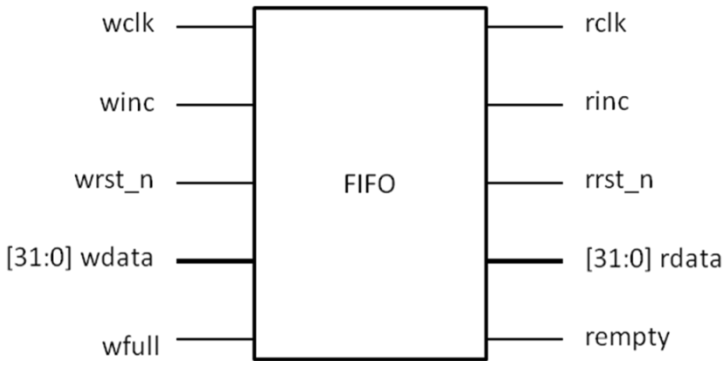

Figure 3.9 FIFO interface

Figure 3.9 shows the interface of the FIFO. There are five sub-modules in the FIFO. The top-level module of the FIFO does not implement any function but just wiring the other sub-modules together. When *wrst_n* signal is asserted low, the write pointer will be reset to 0 and the *rempty* signal is asserted high. If *wrst_n* is not asserted, the *winc* signal is high, and the FIFO is not full (*wfull* = 0), the FIFO will write one 32-bits data word to its memory at the rising edge of the *wclk*. At the same time, the write pointer will advance to the next memory location. Similarly, when *rrst_n* is low, *rinc* is high, and *rempty* is low, the FIFO will pull one data word out every *rclk* clock cycle.

The FIFO includes five sub-modules, which are *fifomem, rptr_empty, wptr_full, sync_r2w* and *sync_w2r*. Among the sub-modules, the *fifomem* module is a dual port

memory. It will write one data word to the location pointed by the write pointer when there is a valid write command. On the read port, the output data is assigned to the memory location pointed by the read pointer. So the output data is always available for reading. After a successful read and the FIFO is not empty, the read pointer will advance one location.

The *rptr_empty* module generates the binary code read pointer which is used in the *fifomem* module to address the data in the memory. It generates the Gray code read pointer to be used for synchronization to the write clock domain and compare with the write pointer. It receives the synchronized write pointer from the *sync_w2r* module, compare with the Gray code read pointer to generate the *rempty* signal.

The *wptr_full* module generates write pointer in binary code and also in Gray code. It compares the Gray code write pointer with the synchronized Gray code read pointer taken from the *sync_r2w* module to generate the *wfull* signal.

The *sync_r2w* module takes the read pointer from the *rptr_empty* module and synchronizes it to the write clock domain through a chain of two registers. This is to avoid system failures caused by metastability which might occur during signal synchronization. Similarly, the *sync_w2r* module synchronizes the write pointer to the read clock domain.

When pixel data is taken from the cameras and written to the external memory (SRAM), the write port of the FIFO is controlled by the camera and the read port of the FIFO is controlled by the multi-ports memory controller. When data is read from the memory and fed to other modules such as the processing unit or the VGA controller, the multi-ports memory controller will take control of the write port and the receiving module will control the read port.

3.1.3 Multi-Ports Memory Controller

FPGA has a tiny amount of internal memory bits. This little memory is good only for implementing FIFOs and line buffers. It is not possible to store a whole image or several frames inside the FPGA. Therefore, an external memory is a necessity. In this

system, there are three images that need to be stored in memory, two input images and one output disparity image. Each image has 8 bits depth and 640x480 pixels resolution. Thus, a minimum of 921600 bytes are required to store all three images. The two MBs SRAM chip on the Altera's DE2-70 board is used to store all three images.

The SRAM has a command bus and a 32 bits bi-directional data bus. Data is shared between the camera, the disparity matching module and the VGA controller module. Therefore, a memory controller was designed to control the SRAM through the command bus, capture the data during read operation, feed data to the memory during write operation and manage the time sharing between access requests. Figure 3.10 shows the interface of the SRAM. The SRAM has 19 address bits, 32 bits word, divided into 4 bytes which can be written individually.

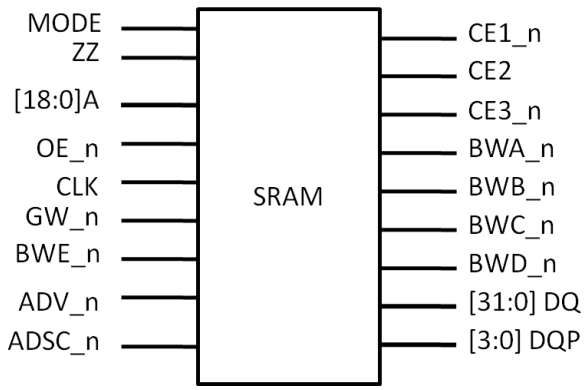

Figure 3.10 The SRAM interface

The designed interface of our multi-ports memory controller is shown in Figure 3.11. When the asynchronous reset signal is asserted, the memory controller will go to an idle state. The *oSRAM_ADSC_N* signal is asserted high, making the SRAM to be idle. The *r_data_avai1* signal and the *w_port_avai5* signal will be asserted low, telling other modules that accessing to the SRAM is not available. There are four read ports and four write ports being implemented on the memory controller. However, in the above figure, only two ports are shown. The memory controller decides when a port gets access to the external memory. Notice that the read and write ports are made separately to achieve simple interface and convenience. If a module requires both read

and write operation, it will be provided one read port and one write port, this is simpler than using a bi-directional port. After reading/writing to a port, the memory controller takes one clock cycle to go back to the idle state, and one clock cycle to jump to the next port.

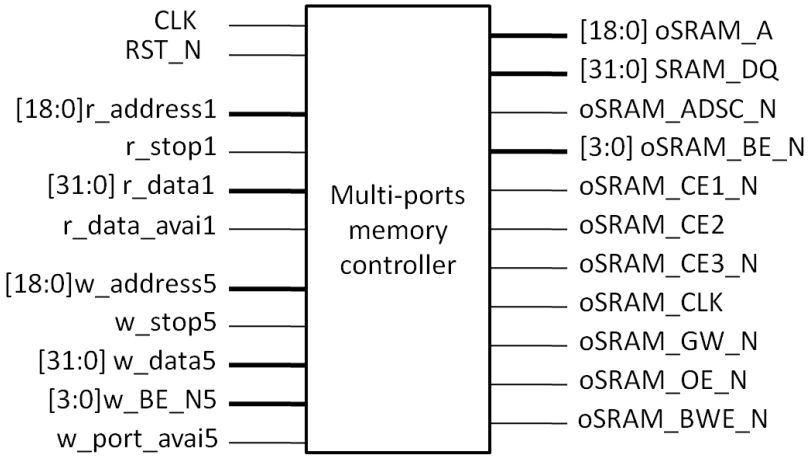

Figure 3.11 Multi-ports memory controller interface

The memory controller has four read ports and four write ports. The number of ports is not a problem in this design. For each read port, there is a 19 bits input address bus, a 32 bits output data bus, a data available signal (*r_data_avai1*) and a stop signal to stop the read sequence. The write port has an address bus, a data bus, a port available signal (*w_port_avai6*), a stop signal and a byte write enable signal (w_BE_N6) to tell which byte in the memory location should be written to. Each port is connected to a FIFO. The read ports of the memory controller are connected to the write ports of FIFOs. And the write ports are connected to the read port of the FIFOs. The ports are examined sequentially. If a read FIFO is not full, it will be filled up without any request. And if a write FIFO is not empty, it will be cleared by taking the data from the FIFO and write to the SRAM.

During a read cycle, *oSRAM_ADSC_N* is set to low to enable the SRAM. A data enable signal (*SRAM_D_ena*) is set to low. This sets the data port of the SRAM controller to high impedance, activates the output enable signal of the SRAM (*oSRAM_OE_N*) and disable the write enable signal (*oSRAM_BWE_N*). The SRAM will take control of the data bus. The first address is presented to the on the address

bus. In the next state, the second address is presented to the address bus. So, the address is being pipelined into the memory. In the third state, after two clock cycles since the first address is presented on the address bus, data available signal (*r_data_avai1*) is asserted, telling the receiving module that the data will be available to be captured in the next rising edge of the SRAM clock. After two and half clock cycles since the first address is presented, the data is available on the data bus. It is captured to a temporary register (*SRAM_DR*). After another half a clock cycle (total of three clock cycles), the data from *SRAM_DR* can be written into the FIFO. It takes three clock cycles to get the first data word. But the address was pipelined into the memory in every state. So the next data can be capture at the fourth clock cycle. To read *n* memory location, it takes *n+2* clock cycles. The read cycle is stopped when the *r_stop1* signal is asserted high by the receiving module. This signal is connected to the write full (*wfull*) signal of the FIFO. It means, when the FIFO is full, the read cycle will be stopped.

In a write cycle, *oSRAM_ADSC_N* is set to low to enable the SRAM. *SRAM_D_ena* signal is set to high. As soon as the byte write enable signal *oSRAM_BWE_N* is asserted low, it will overrule the *oSRAM_OE_N* signal. The SRAM immediately turns its data port to high impedance regardless of the value of *oSRAM_OE_N*. The memory controller takes control of the data port and the write cycle begins. Writing is a one clock cycle operation. So, the address and corresponding data should be presented to the SRAM at the same time with the assertion of *oSRAM_BWE_N* signal. The write operation is done in one clock cycle. However, a pre-writing state was added to the memory controller to make a handshaking procedure with the sending module. In this pre-writing state, the memory controller captures the first address and set the write port available signal (*w_port_avai5*) to high. When the *w_port_avai5* is asserted high, the sending FIFO will start feeding data to the memory controller, because the *w_port_avai5* was connected to the read increment (*rinc*) signal of the FIFO. *N+1* clock cycles are required to write *n* locations in the memory. The write operation is finished when the FIFO is empty. This was done by connecting the read empty signal (*rempty*) of the FIFO to the write stop signal (*w_stop5*) of the memory controller.

3.1.4 The Processing Unit

The processing unit was initially designed as shown in Figure 3.12. But it was improved later as will be described in section 3.2. This module consists of two input FIFOs that read the pixel data from left and right images, which were stored in memory by the camera interface module. The pixel data is then fed into two line buffers, which were originally designed as register-based line buffer. The purpose of using line buffer is to save memory bandwidth usage as will be described in sub-section 3.2.3. Initially, the system was designed to implement SAD algorithm with 3x3 pixels window and 32 pixels disparity range. So, the first line buffer (let's call it primary line buffer) has three outputs, equivalent to three lines on the input image. The primary line buffer has 1280 registers and is the same for both images. In Figure 3.12, the primary line buffer is shown in the top right corner block. Data from the first line buffer's outputs is passed into a second register-based line buffer (secondary line buffer). The purpose of the second line buffer is to select a segment on a row of image data to be processed. The left secondary buffer has only three registers for each line (equal to the window size), while the right secondary buffer has 34 registers for the disparity range of 32 pixels. Figure 3.12 shows the left and right secondary buffer at the top left corner. When new data is fed into the line buffer and previous data is shifted to the left, it is similar as a window being moved from left to the right on the input image.

The subtraction D[j][k], sum of absolute difference SAD[k], total similarity Sim[k] and comparators (CMP) are modeled as combinational logic circuit, shown at the lower block in Figure 3.12. where $0 \leq j \leq 8$ and $0 \leq k \leq 31$. At every rising edge of the clock, a pixel is shifted to the next location in the left side of the register array and a new pixel is fed from the line buffer.

$$L[i][k] <= L[k][i+1] \qquad i = 1,2,3 \qquad 0 \leq k \leq 3 \tag{3.5}$$

$$R[i][k] <= R[k][i+1] \qquad i = 1,2,3 \qquad 0 \leq k \leq 3 \tag{3.6}$$

A new pixel is fed from the FIFO to the beginning of the first line buffer. When a FIFO is empty, it will be filled with the successive pixels from the same row on the

image. The effect of the above actions is that the 3x3 window will slide from left to right on the left image, and the 3x34 pixels window will slide from left to right on the right image.

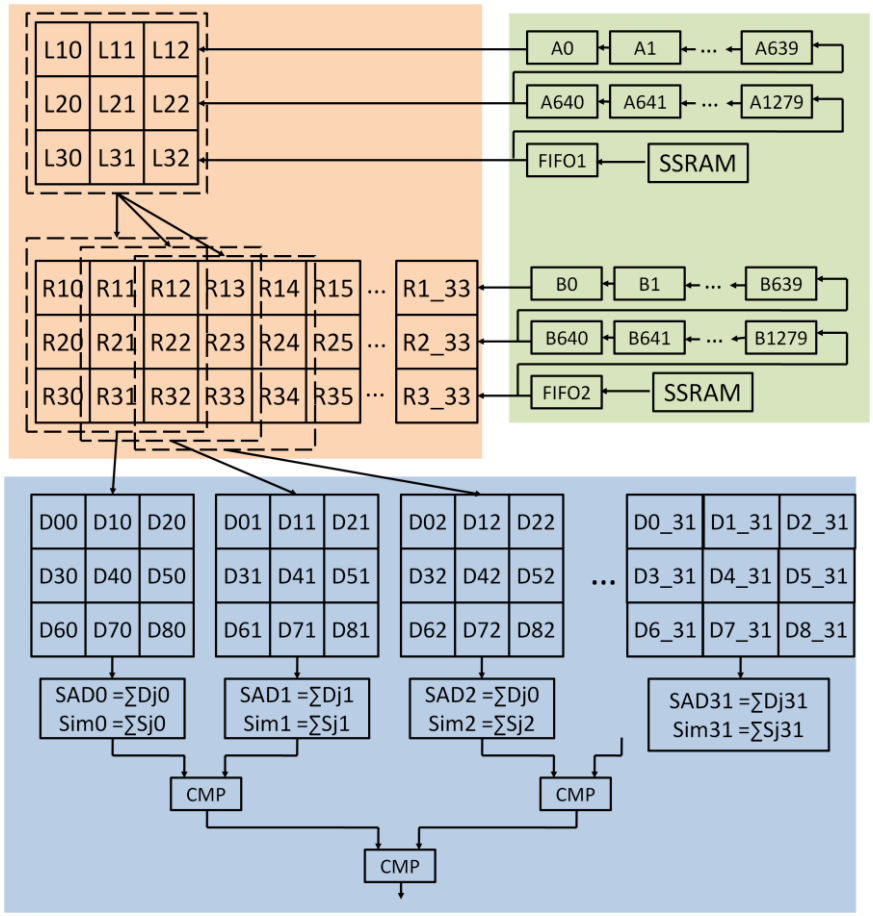

Figure 3.12 The processing unit data path

In Figure 3.12, D[j][k] is the absolute difference between two pixels.

$$D[0+w][k] = |L1[w]-R1[k+w]| \qquad w = 0,1,2 \qquad (3.7)$$

$$D[3+w][k] = |L2[w]-R2[k+w]| \qquad w = 0,1,2 \qquad (3.8)$$

$$D[6+w][k] = |L3[w]-R3[k+w]| \qquad w = 0,1,2 \qquad (3.9)$$

$$0 \le k \le 31$$

There are 9x32 = 288 absolute subtractions being performed in parallel. The absolute differences are fed into the sum of absolute difference calculation. All SAD[k] calculations and comparisons are also done in parallel.

$$SAD[k] = \sum D[j][k] \qquad 0 \le k \le 31 \text{ and } 0 \le j \le 8 \tag{3.10}$$

If the difference in intensity between two pixels is less than a threshold value T, the two pixels are considered similar. Users can set this threshold value through the switches on the FPGA board. Typically, T is set to 10. The number of similar pixels is added up for each window, having the value from 0 to 9.

$$S[j][k] = \begin{cases} 1 & if\ D[j][k] < T \\ 0 & if\ D[j][k] > T \end{cases} \qquad 0 \le j \le 8 \text{ and } 0 \le k \le 31 \tag{3.11}$$

$$Sim[k] = \sum_{j=0}^{j=8} S[j][k] \tag{3.12}$$

These sums of absolute difference (SAD[k]) are compared to find the smallest value (min SAD[k]). The window on the right line buffer which produces min(SAD[k]) and max similar pixels (max(Sim[k])) is consider to be “matched” with the left window. Its corresponding displacement is the disparity value. The amount of displacement is written into the disparity image as pixel intensity. All of these operations are models as combinational logic circuit. The comparators are designed in binary tree. So, the level 1 comparison has 16 comparators, level 2 has 8 comparators and so on. There are five levels of comparison before the disparity value is found. It takes only one clock cycle to pass the data from the line buffers to the disparity register at the last comparator.

3.2 Proposed Resource Minimization Techniques

The important resources on the DE2-70 FPGA board includes logic elements, memory bandwidth, internal memory bits, dedicated multipliers and phase locked loop (PLL) circuits. In a depth-map system, logic elements and memory bandwidth

are the two most valuable resources. This section discusses the resource minimization techniques using different system architectures to save LEs and memory bandwidth.

3.2.1 Minimization of Logic Elements

In this section, three methods to minimize logic element usage are proposed. The first method is to implement depth-map architecture with single search direction instead of searching on both left and right direction to find the disparity match. The second technique is to use memory based line buffer to replace register based line buffer. The third technique to reduce the number of logic elements usage is by replacing the repetitive calculation with a register which store previously calculated result. This technique is also called the data reuse technique.

3.2.1.1 *Single Search Direction Method*

The disparity matching module uses large amount of resources. The design shown in Figure 3.12 uses approximately 69% the number of logic elements (LE) available in the Altera's Cyclone II FPGA. However, there is room for optimization. To the author's knowledge, most of the previous designs (if not all) take a window on an image as reference and make a search on both direction of the scan line on the second image [9, 11]. This is a waste of time if the system was implemented on general purpose processor and waste of resource if the system was implemented on hardware. As shown in Figure 2.3, the projection of A on the right image is shifted to the right compared to the projection of A on the left image for all position of A within the field of view of the camera. Thus, if we take the left image as reference, it is not necessary to search on the left side of the referent point on the right image. The system shown in Figure 3.12 takes the left image as reference and searches on the right direction of the right image only. With a disparity range of 32 pixels, this system has the actual search range equivalent to other designs with 64 pixels disparity range. Basically, it saves almost half of the resource usage for the same performance. Because with the disparity range reduces from 64 to 32 pixels, the number of subtractions required is reduced from (64+2)x3x3=576 to (32+2)x9=306 subtractions. The number of

comparators required is also reduced by the same percentage. In our actual system, the number of logic elements used is cut down from 33,482 LEs to 17,632 LEs.

3.2.1.2 *Memory Based vs. Register Based Line Buffer*

The design in Figure 3.12 uses four line buffers. Two primary line buffers have 1280 registers each. The secondary line buffer of the left image has a block of 3x3 registers. And the secondary line buffer of the right image has a block of 3x34 registers. Registers are created using logic gates. Therefore, this method of designing the line buffer requires large number of logic elements (LEs) to create the registers for a long line buffer. Logic element is a precious resource on FPGA because it is very limited and it is useful for many purposes, such as creating combinational logic circuits, dedicated registers and memory. On the Altera's Cyclone II FPGA, there are 68,416 LEs and 250 M4K RAM blocks with 1,152,000 memory bits available. The design shown in Figure 3.12, which has 3x3 pixels window size and 32 pixels disparity range, uses up to 46,933 LEs (69% of total LEs available). The two primary line buffers use 30,272 LEs, which is equivalent to 44% of the total LEs available. Meanwhile, it used only 13,120 dedicated memory bits (1% of memory bits available), mainly for the FIFO's memory and the Red component buffer in the camera interface module. Another way of designing line buffer is by using on-chip dedicated RAM blocks. The line buffer works as an 8x1280 memory block with sequential read and write accesses. The line buffer has one input write pointer and two output-read pointers as shown in Figure 3.14. The read and write pointers are increased at the positive edge of the clock if a pixel data is required by the disparity matching module. The first read pointer (rp0) is always 1280 locations behind the write pointer (wp) and the second read pointer (rp1) is 640 locations behind the write pointer. The third output of the line buffer is taken directly from the input. With the memory-based line buffer, it requires only 146 LEs for the read and write pointer instead of 30,272 LEs. However, it needs 40,960 dedicated memory bits to store pixel data (5% of available memory bit). Thus, we moved from using critical resource to a less critical one. Memory bit also require less chip area than a logic register.

The question here, why don't we use memory-based line buffer for the secondary line buffer also? It would save even more resource? Notice that a read pointer is required to access a location on the memory-based line buffer. For the secondary line buffer, all pixel data need to be read every clock cycle. Thus, if the secondary line buffer was designed as memory based, it would require (34+3)x3 = 111 read pointers to access these pixels. The pointer is made from registers (LEs) as well. Therefore, it will require even more LEs to make the read pointer than to store image pixel data. Implementing secondary line buffer using registers is more effective than using dedicated memory bits.

3.2.1.3 *Data Reuse Technique*

In Figure 3.12, D20 is the absolute difference of R12 and L12. So, a subtraction circuit was used to calculate D20. Similar for D10 and D00:

$$D10 = |L11-R11| \quad ; \qquad D00 = |L00-R00| \tag{3.13}$$

But the value of L12 is shifted to L11 after one clock cycle and the value of R12 is shifted to R11 at the same time. So, the subtraction L11 – R11 has the same value as the subtraction of L12 – R12 at the previous clock cycle. It is not necessary to repeat the subtraction of the same values three times. Instead of using three subtractors to calculate D20, D10 and D00, two registers can be used together with only one subtractor for D20.

$$D20 = |L12-R12|$$

$$D10 <= D20; \qquad D00 <= D10; \tag{3.14}$$

This can reduce two third the number of subtractors. On the actual system, this step shows a reduction of 24.75% LE usage of the whole system.

3.2.2 System Improvement

With the resource usage being minimized, there is extra resource on the Cyclone II FPGA for further improvement of system performance. The system in Figure 3.12 was improved to 64 pixels disparity range and 5x5 pixels window size. The advantage of increasing disparity range is that it will reduce the minimum distance which the camera system is able to estimate the depth. In other words, the system can "see" closer objects. This is shown in equation (2.7). The drawback is the increasing of system complexity and resource usage. With the disparity range being doubled, it requires twice the number of registers on the right line buffer, from Ri[0] to Ri[65] (i=1,2,3), twice the number of absolute subtractors to find D[j][k] for j=2,5,8; $0 \leq k \leq 64$. As in (3.14), it also requires two times the number of registers to store D[j][k] for j=0,1,3,4,6,7 and $0 \leq k \leq 64$. The number of adders to calculate SAD[k] and Sim[k] is also increased by two times. On the comparison stage, only the number of comparators on the first level (level 1) was doubled. Other levels remain the same. However, one more level of comparison is required, making up a total of six levels. The critical path is slightly longer, thus, the max clock frequency is reduced. But the real-time requirement is still met.

Increasing the window size from 3x3 pixels to 5x5 pixels makes the depth-map algorithm perform better on textureless area. It also increases the sensitivity of the comparators for Sim[k] since the resolution of Sim[k] is increased from 9 values to 25 values. But, the algorithm will perform slightly worse on the intensity discontinuous regions (edges of objects). The system complexity was also increased. The primary line buffer was increased from two lines with 1280 locations to 4 lines with 2560 locations. L[i][k] and R[i][k] are increased to 5 lines. The window of D[j][k] was expanded to 5x5 pixels. The adders in (3.10) and (3.12) become more complex because the number of terms in each adder increased from 9 to 25. The resource minimization technique in (3.14) shows greater advantage on a larger window size.

3.2.3 Minimization of Memory Bandwidth Usage

Memory bandwidth is a crucial resource in image processing as it can degrade overall system performance. Memory contention is potentially a critical problem in many systems. In our design, only one SRAM chip with the capacity of 2 MB was used. We have implemented two types of line buffers, FIFO and a specialized memory controller to resolve memory contention.

3.2.3.1 *Line Buffers*

The first step that should be taken to reduce memory contention is by minimizing the memory bandwidth and the number of memory accesses required by the system. The disparity matching module was designed in such a way that it will work at the clock frequency of 10 MHz and calculate one disparity pixel every clock cycle. Because the window size is 5x5 pixels, 25 pixels are required for each window. Since the disparity range is 64 pixels, one window on the left image and 68 windows on the right image are required to calculate one pixel of disparity. Thus, the amount of data required in each clock cycle is equivalent to (68+1) x 25 = 1725 pixels. In other word, 1.68 MB of data is required in a clock cycle or 16451 GB/s. However, each pixel was used several times in the calculation. Pixels from the fifth to the 64th on the right image are used five times in five adjoining windows of one disparity calculation. The 1st, 2nd, 3rd, 4th pixels and the 68th, 67th, 66th, 65th pixels were used 1, 2, 3 and 4 times in the calculation, respectively. So, only 365 pixels are required from the two images. These 365 pixels were accessed 1725 times in a clock cycle.

To take advantage of the data reuse, ten register-based line buffers were introduced (the secondary line buffer). These line buffers allow immediate access to any pixel in the line buffer at any time. Five line buffers of the left image store 5 pixels each. And five line buffers of the right image store 68 pixels on each line buffer. In the next clock cycle, disparity value of the next pixel needs to be calculated. The new neighborhood pixels and search range is also shifted one pixel to the right on the same scan line. This was done on the line buffer, the pixel data was shifted one pixel to the left every clock cycle as shown in Figure 3.13.

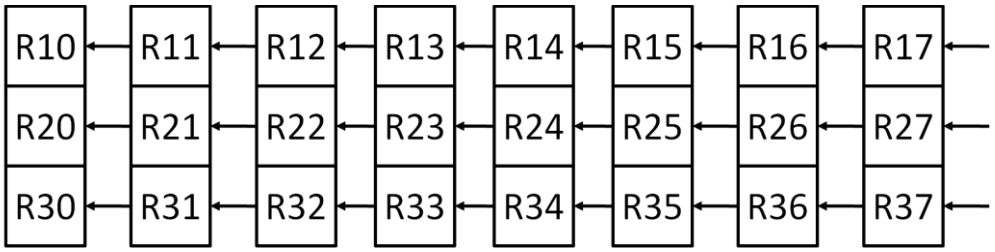

Figure 3.13 Shifting pixel data on the secondary line buffer

A pixel on the left image is used five times in five successive disparity calculations and a pixel on the right image is used 64 times in 64 successive disparity calculations. The line buffers store the previous pixel data. Therefore, among the 365 pixels, there are only 10 new pixels from 10 lines are required every clock cycle.

Also, data from previous rows can be used in the calculation of the next row of disparity. Each line is reused 5 times. For each image, a memory-based line buffer was used to store four rows of image pixels. The memory-based line buffer has one input and 5 outputs. It is actually an on chip memory block with one write pointer and five read pointers. The distance between two adjoining read pointers is 640 pixels, equivalent to one line on the image. When a pixel is read, the pointer advances to the next location. This is different with the register-based line buffer where there is no read/write pointer but the pixel data is shifted to the next register. Outputs of the memory-based line buffer are sequential, that means, only one pixel can be read from each output at a time. With the memory-based line buffer (the primary line buffer), the number of pixels required every clock cycle is reduced from 10 pixels to only two pixels per clock cycle, one for each image. Figure 3.14 shows the two types of line buffers being used together to minimize memory bandwidth required. The right side of Figure 3.14 represents the primary line buffer, which was implemented as memory. The left side of figure 3.14 shows the secondary line buffer. It is connected to the primary line buffer through 3 read ports (*rptr1* to *rptr3*). The reason of using two types of line buffers is because of the resource consumption as explained in subsection 3.2.1.

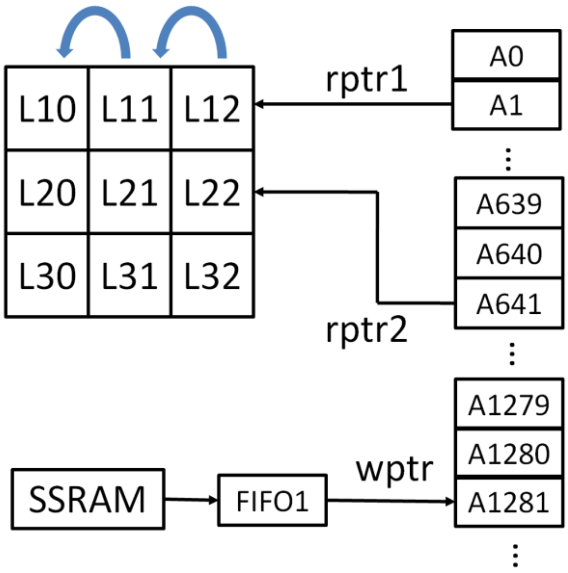

Figure 3.14 Two types of line buffer

Furthermore, four 8-bit pixels are combined and written into one 32-bit memory location. Thus, each memory access can read four pixels. The number of memory accesses required by the input of the disparity matching module is reduced to only 2/4 = 0.5 accesses/clock cycle.

3.2.3.2 *Specially designed memory controller*

Although the memory bandwidth requirement was greatly reduced by using line buffers, memory contention is still a major problem. A naive approach would design the memory controller with a read/write-request input signal. The memory controller will be idle while waiting for other modules to send their access request signals to initiate read/write operations. This naive approach suffers from memory contention when there are two or more requests with high priority issued at the same time. In our system, the two cameras keep feeding data into the FIFOs every clock cycle of the camera clock. The camera clock is generated by a phase locked loop (PLL) inside the camera itself. If the camera clock is stopped, several frames will be lost. Therefore, the camera FIFOs must be cleared before both of them overflow. The VGA controller needs to be supplied with continuous data for a correct display. The worst case happens when the two camera FIFOs are full and the VGA FIFO is empty at the same

time, because these three modules cannot wait for each other. The disparity matching module has lower priority since it may be stopped. In our approach, we used a *self initiate* memory controller. This memory controller always tries to fill/clear the FIFOs without any request from other modules. The FIFOs are examined sequentially. The one needs to be filled is filled up before it underflows and the one that needs to be cleared is cleared out before it overflows. The technique works as long as the external memory bandwidth is larger than the total memory bandwidth required by all modules plus overhead.

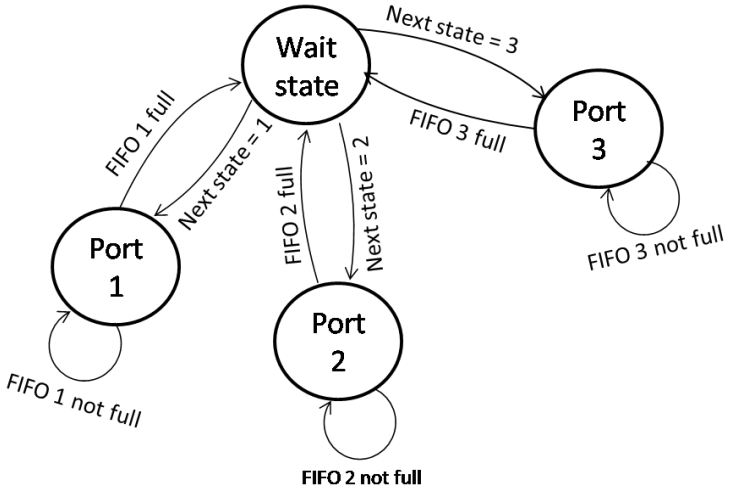

Figure 3.15 State machine of the multi-port memory controller.

Figure 3.15 shows the state machine of this memory controller. For our system, with the line buffers implemented, the two cameras require a peak memory bandwidth of 48x2 = 96 MB/s, the processing unit requires 30 MB/s and the VGA controller module requires 28 MB/s memory bandwidth. The total required memory bandwidth is 96 + 30 + 28 = 154 MB/s. The memory controller needs two clock cycles to initiate a read operation and one clock cycle to initiate a write operation. Each read/write burst can fill/clear a FIFO of 32 locations long. Thus, the overhead is approximately 2/32 = 6.3%. The external memory chip must have a minimum bandwidth of 154 x 1.063 = 163.7 MB/s. A static RAM (SRAM), which can deliver 200 MB/s at clock frequency of 50 MHz, was used to ensure no memory contention.

Thus, to resolve the memory contention, register based and memory based line buffers were used to minimize the amount of required memory bandwidth. A specially designed memory controller was used to avoid concurrent multiple access requests.

3.3 Summary

A stereo vision system was designed with two cameras acting as inputs. The cameras were configured to produce a frame size of 640x480 pixels. A detail output timing and color pattern of the camera was described in section 3.1.1. The camera interface module starts converting the color image into grayscale image when the second line of the image (Green and Blue component) is fed from the camera. FIFOs and line buffers were used to minimize external memory bandwidth requirements and to transfer data across multiple clock domains. When data is passed from a clock domain to the other, metastability phenomenon causes system failure. Metastability rate is proportional to the clock frequency of the two clock domain [29]. Thus, FIFO is needed to transfer the data safely. Synchronization register chain was used to avoid metastability of the FIFO's control signals. The line buffer reduces the memory bandwidth requirement since it allows data of the recent used pixels to be used several times in the FPGA. Two types of line buffers were implemented with different uses. If only few (3 to 5) pixels in a line buffer are required in a clock cycle, memory based line buffer shows great resource saving advantage. When all pixels in the line buffer are required every clock cycle, a register based line buffer shows better efficiency. In our system, the FIFO also acts as a short line buffer. In between the memory controller and the VGA controller, there is no dedicated line buffer but the FIFO behaves as a small data pool. For the disparity processing module, the SAD depth-map algorithm was implemented. The key idea of the algorithm is to find the maximum point of the total similarity function and minimum point of the total difference function with regard to disparity value. Disparity value is inversely proportional to the distance between camera system and the object. Resource minimization techniques were employed in five different hardware architectures. To minimize resources usage, line buffers were changed from register based line buffer

to memory based line buffer whenever possible. The repetitive subtractions were replaced by registers which store previous result for subsequent uses. The memory controller was specially designed for image processing tasks. It reads the image data and feeds into FIFO automatically without initiate signal from the receiving module. Thus, memory contention is eliminated.

CHAPTER 4

RESULT AND DISCUSSION

The disparity matching module was implemented with five different architectures. The advantages and disadvantages of these architectures are discussed. Results in term of memory bandwidth usage, logic element used and internal memory bits usage are described. Experimental results were taken to demonstrate the functionality the proposed system. These architectures are compared in term of resource usage, speed (frame rate), maximum frequency and timing analysis. Comparison is made between the five architectures implemented in this research, and between our architecture and the other systems discussed in the literature review. Because the SAD depth-map algorithm was not modified, our system maintains the same image quality as other systems implementing the same algorithm.

4.1 Five Implemented Architectures

In overall, the system consists of two cameras connected to two camera interface modules. Each of them is then connected to a FIFO before transferring data to a memory controller (as shown in Figure 3.1). Image data is stored in an external SRAM chip. The processing unit requests data from the memory and generates disparity result. Output disparity image is displayed on a VGA monitor. The difference between our five architectures is in the processing unit, where the resource minimization techniques were implemented.

4.1.1 The first architecture

The implementation of the first architecture was described in chapter 3. The first architecture performs SAD depth-map algorithm for the window size of 3x3 pixels, 32 pixels disparity range. The processing unit of this architecture was explained in sub-section 3.1.4. An illustration of the processing unit is shown in Figure 3.12.

In figure 3.12, the arrow from A1 to A0 indicates that data is shifted from register A1 to register A0. On the secondary line buffer (top right block), data is shifted from register L12 to L11 and then to L10. The first architecture was implemented without resource minimization techniques. It uses registers for both primary and secondary line-buffers. The repetitive subtractors were not replaced with registers. Therefore, it takes the large number of logic elements compared to the other four architectures for a relatively small window size and small disparity range. The whole design uses 46,933 LEs, which is 69% the total logic element available on the Altera's Cyclone II FPGA. The two primary line buffers (each line buffer contains 3 lines of the image) use 30,275 LEs (44.24% available LEs). The disparity combinational logic block uses 14,471 LEs (21.15%) and approximately 2000 LEs in other modules. Among the 14,471 LEs used in the disparity module, a large part of it is used for the secondary line buffer. Because the secondary line buffer is inside the disparity module, Quartus II software does not have a reliable tool to calculate the number of LEs used by the secondary line buffer separately. Beside the logic elements, this architecture also uses 13,120 memory bits (about 1% of available memory bit) and one out of four dedicated phase locked loop circuits (PLL). The memory bits were used in the FIFOs and the Camera interface module to store the Red color pixel value as explained in subsection 3.1.1. Only one phase locked loop circuit was used to generate three different frequency clock signals for the disparity matching module, the SRAM controller module and the camera interface module. The maximum allowed clock frequency of the disparity matching module is 28.72 MHz. But it requires that the disparity matching module work at only 10 MHz for the system to achieve real-time performance. If the clock frequency is set to 28.7 MHz, this system can achieve the frame rate of more than 80 fps.

Figure 4.1 is one of the two images taken by the cameras.

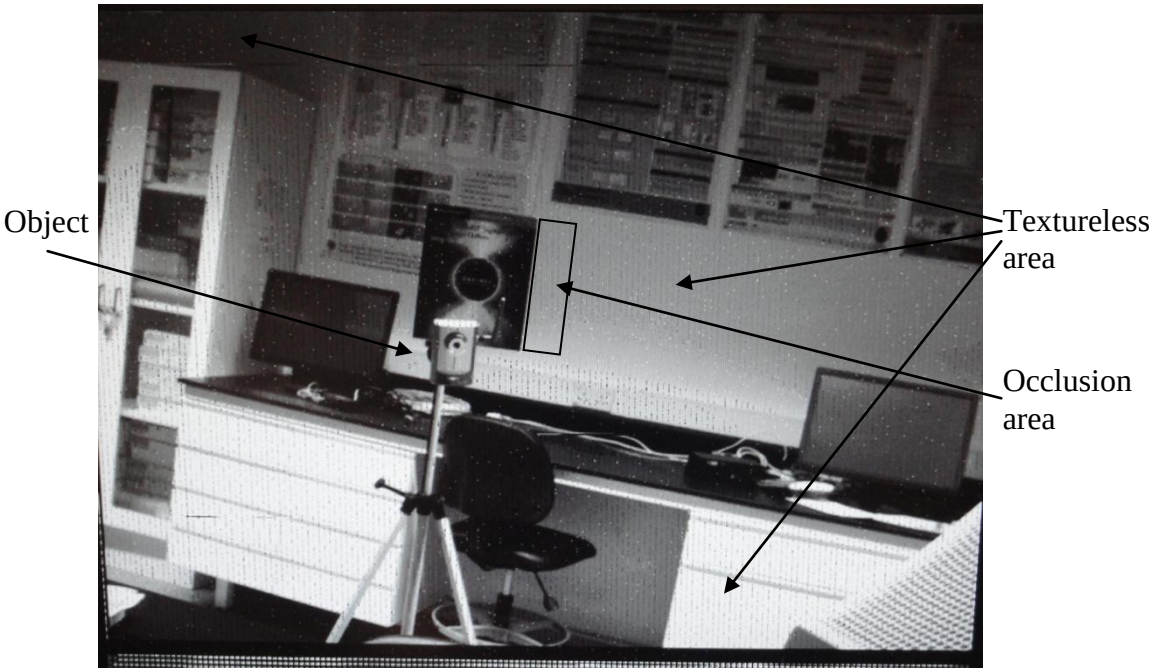

Figure 4.1 Input image with object at 2 meters

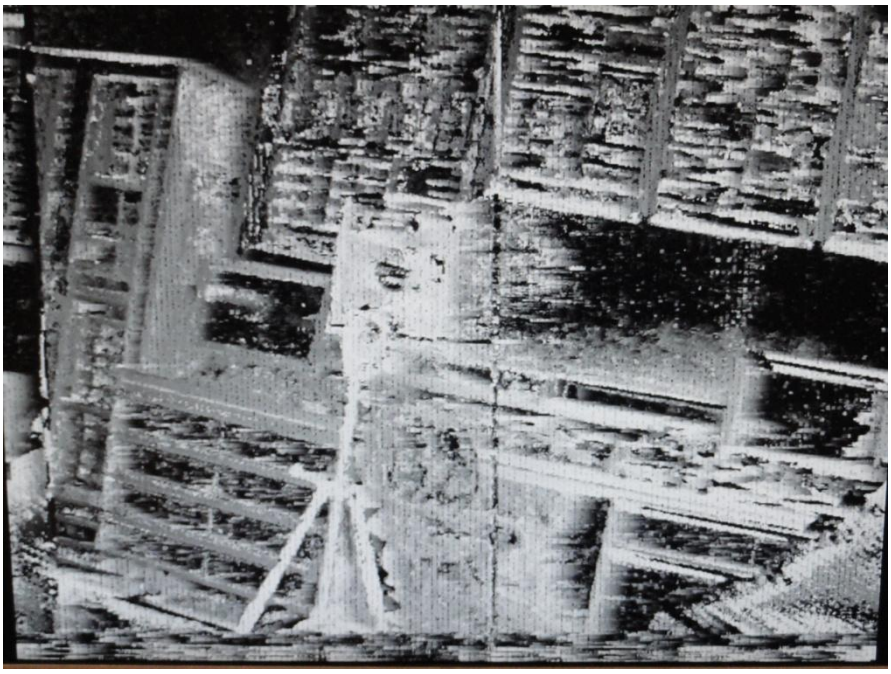

Figure 4.2 Disparity image of object calculated by the first architecture

In the experiment, the object (the tripod with a webcam and a piece of paper on it) was placed approximately more than 2 meters away from the system's cameras. The

object has to be placed at that distance because the first architecture has a disparity range of 32 pixels. According to equation (2.7), small disparity range (d_{max}) results in large minimum visible range (AB_{min}). Experiment shows that the system starts giving correct disparity value when the object is more than 2 meters away from the cameras. Figure 4.2 shows that the system is able to calculate the disparity image. In Figure 4.2, the webcam is brighter than the background since it is closer to the cameras. In other word, it has higher disparity value than the background. The camera case and the webcam appear clearly on the resulting image at higher intensity than the surrounding area. Thus, we can conclude that the system is working.

There is some exception of the disparity value on the resulting image. It can be noticed that the input image contains also textureless and occlusion area. Textureless area is an area larger than the window size where there are no changes in gray value across a horizontal line. In other word, there is nothing in that area. Because the depth-map algorithm works by comparing the gray value on a horizontal line of the images, if the gray values are identical on a horizontal line, the depth-map algorithm is unable to estimate the disparity value. This is an inevitable limitation of the algorithm. In Figure 4.2, the textureless areas are filled with black. Because two cameras look at a scene from two different angles, there will be some areas which appear in one camera but not in the other camera. Those areas are called occlusion areas. Occlusion area always occurs beside a front object. Since it doesn't appear on both cameras, the depth-map algorithm is unable to calculate the depth for it.

4.1.2 The second architecture

When the distance from the camera to the scene is less than the minimum visible range, the system cannot produce correct disparity value. Thus, it is necessary to increase the disparity range so that the AB_{min} can be reduced. But the first architecture uses 69% of the total logic element available on the Cyclone II FPGA. When we tried to implement a system with 64 pixels disparity range, the total resource requirement was greater than the available resource. Therefore, it is needed to design a better architecture that minimizes resource usage before increasing disparity range. The

second architecture is an attempt to minimize resource usage. To achieve this goal, the second system was implemented based on the first architecture. The block diagram of this system is the same as the first architecture, which is shown in Figure 3.1. Except the primary line buffer was implemented using dedicated memory bits instead of register. The primary line buffer of the first architecture is shown in Figure 4.3 and the primary line buffer of the second architecture is shown in Figure 4.4.

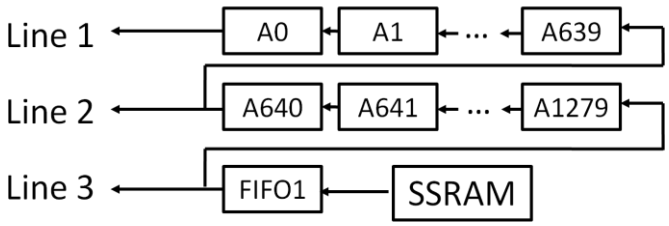

Figure 4.3 Register based primary line buffer

Note that in Figure 4.3, data is shifted from the right to the left of the line buffer and new data is fed from the FIFO to the right most register. In Figure 4.4, the primary line buffer was designed as a dual port memory block, which is accessed using one write pointer and three read pointers. Instead of shifting data from A1 to A0, fixed memory locations are used. The read pointer will advance to the next location after a read operation.

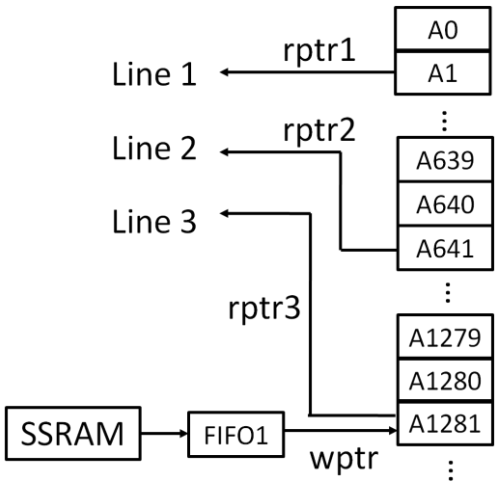

Figure 4.4 Memory based primary line buffer

Compilation report shows that the line buffer module uses only 146 logic cells (0.21% LEs available) for the read/write pointers instead of 30,275 LEs (44% LEs available) of the first architecture's line buffer. The secondary line buffer and the disparity combinational logic claim 14,604 logic elements, which is same as the first architecture. So, the total LEs used is 17,632 LEs, which is a reduction of 62.43% compared to the first architecture. As a replacement for the logic elements in the line buffer, 40,960 bits (3.6%) of dedicated memory were used. The whole system uses 54,080 memory bits, which is equivalent to 5% of the total internal memory. This is an increase of 312% compared to the first architecture. However, memory bit is a less critical resource than the logic element. Thus, it makes sense to use memory bits instead of logic elements to design the primary line buffer. As reported by Quartus II compilation software, the maximum allowed clock frequency for this architecture is 13.8 MHz. Thus, the system has the potential to process 41 frames per second and meet the real-time requirement. Since there is no change in the depth-map algorithm compared to the first architecture, the second system produces the same image result as the first one. The AB_{min} is still 2 meters and any object with a distance closer than that will give incorrect disparity value.

4.1.3 The third architecture

When the resource is minimized with the memory based line buffer as discussed in section 4.1.2, there is sufficient resource to implement a SAD algorithm with larger disparity range. The third architecture was implemented based on the second architecture with the disparity range increased from 32 pixels to 64 pixels and the window size remains 3x3 pixels. The processing unit of this architecture is shown in Figure 4.5. Compared to the first architecture in Figure 3.12, the right secondary line buffer has been changed from 34 registers (R1_0 to R1_33) to 66 registers (R1_0 to R1_65). The number of absolute subtractors is also increased by two times, from Dj_0 to Dj_63 instead of Dj_0 to Dj_31 ($0 \leq j \leq 8$). And the number of SAD and Sim is also increased by two times.

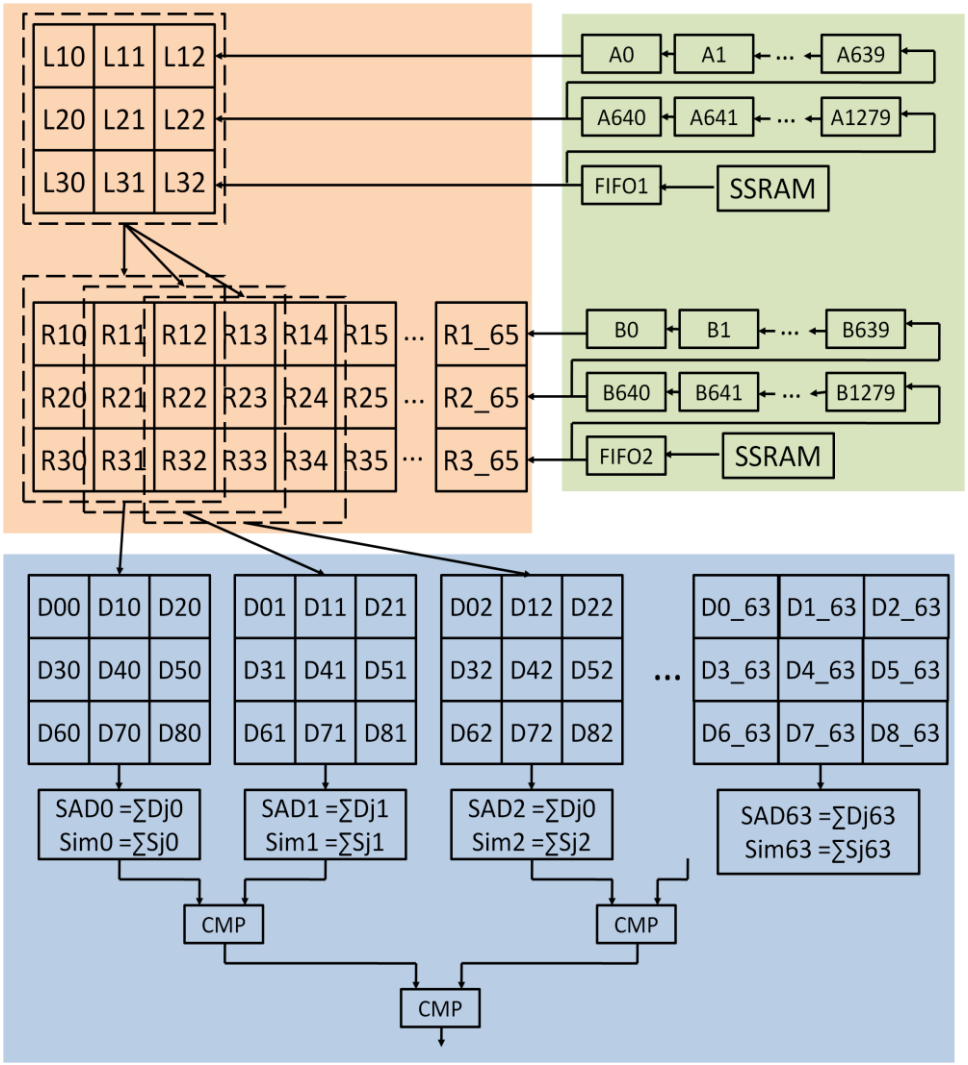

Figure 4.5 The processing unit data path of the 64 pixels disparity range architecture

According to equation (2.7), increasing the disparity range d_{max} reduces the minimum visible range AB_{min}. Thus, the system can estimate disparity value for closer objects. With the disparity range rises to 64 pixels, the minimum visible range was reduced to 92 cm (approximately 2 meters for the 32 pixels disparity range architecture). However, increasing the disparity range escalates the computational power requirement. The system demands more resources and become more complex. In addition, the maximum permitted clock frequency is trimmed down because of the longer critical path. Compilation report shows that the system requires a total of 33,482 LEs, which is an increase of 89.9% compared to the second architecture. Since the FIFOs and line buffers are unchanged, the total memory bit usage is the same as

the second architecture. The maximum allowed clock frequency for the processing unit is 12.17 MHz. This sets the maximum frame rate of the system to 39 fps, which satisfies the real-time requirement.

Experiment was set up with the object closer to the cameras than for the first architecture. The tripod (Object 1) is 1.06 meters from the camera and the mouse pad (Object 2) is 1.56 meters from the camera. Figure 4.6 shows an input image for this system.

Figure 4.6. Input image with object at 1 meter

With the disparity range increases to 64 pixels, the system is now able to estimate the depth at a close distance. The term AB_{min} in equation (2.7) has been reduced. In Figure 4.7, the Object 1 has high disparity value; it appears much brighter than the Object 2. For another comparison, a set of images has been taken with two subjects. In Figure 4.8, the person A was standing about one meter from the cameras and the person B was standing approximately 1.8 meters from the camera. It can be observed from Figure 4.9 that the disparity image of the person A is brighter than the person B.

Disparity value of some areas in the background was not calculated correctly because the background consists of many large textureless, colorless areas.

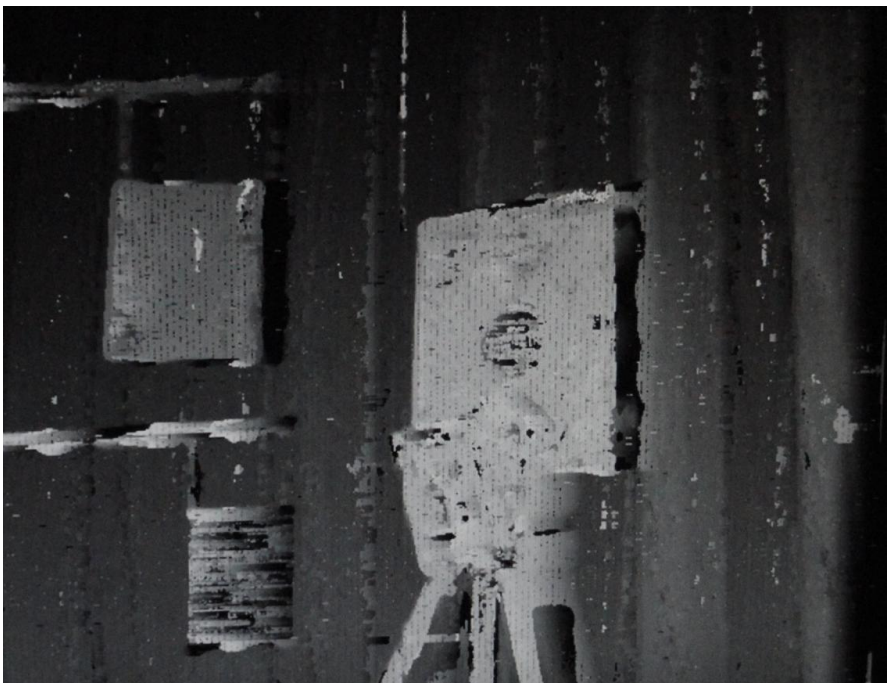

Figure 4.7 Disparity image of object calculated by the third architecture

Figure 4.8 Input image with subjects at 1 meter and 1.8 meters

Figure 4.9 Disparity image of subjects calculated by the third architecture.

4.1.4 The forth architecture

When image processing is implemented on FPGA, it is common that the result from one calculation is used in other similar calculations of the next pixels. If the is used several times in many calculations, the technique is called data reuse. The forth architecture was implemented with data reuse technique to further minimize resource usage of the third architecture. It was designed based on the third architecture with the repetitive subtractors for D[j][k] calculations being replaced by shift registers (equation (3.14)). The calculation is done for the first pair of input window and the result is passed to a chain of two registers after a clock cycle to avoid repeating the same calculation. Figure 4.10 shows the combinational logic block of the third architecture and Figure 4.11 shows the combinational logic block of the fourth architecture.

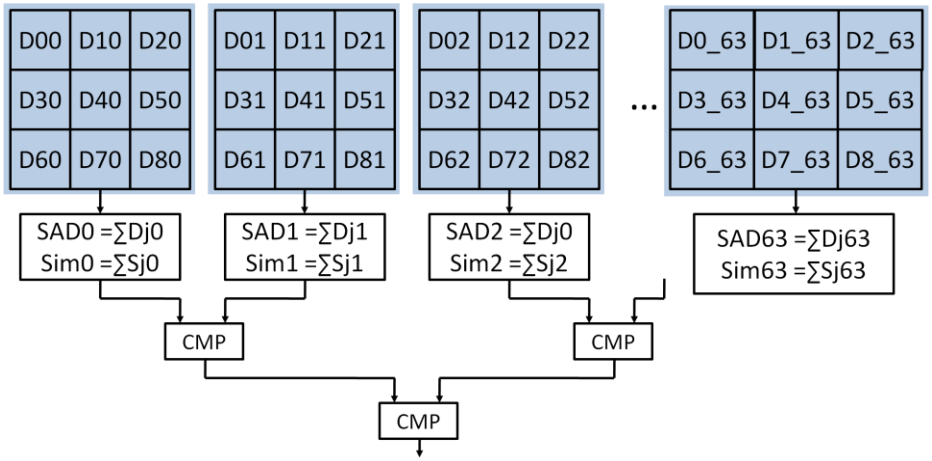

Figure 4.10 Combinational logic block without data reuse technique

In Figure 4.10, each D[j][k] is an eight-bit subtractor. In Figure 4.11, two out of three subtractors in a row have been replaced by registers and data is shifted from the first subtractor to the register after each clock cycle.

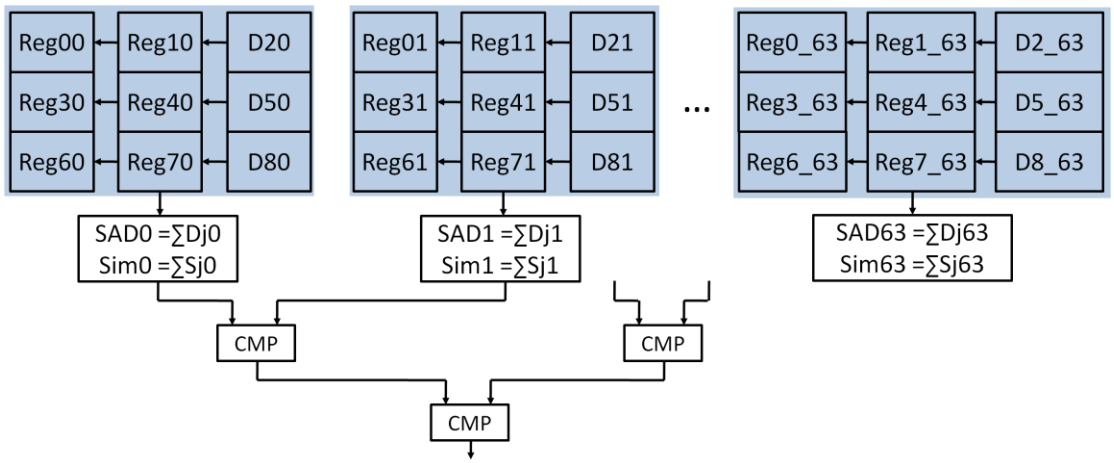

Figure 4.11 Combinational logic block with data reuse technique

Total logic element usage was reduced to 25,195 LEs. This is a reduction of 24.75% LEs compared to the third architecture. The maximum clock frequency is reduced to 11.69 MHz and the frame rate is decreased to 37 fps. There is no difference in the result disparity image since the technique only store the previous subtraction result instead of recalculating it.

4.1.5 The fifth architecture

As shown in Figure 4.2, Figure 4.7 and Figure 4.9, depth-map algorithm does not perform well on textureless areas. When the textureless area is larger than the window size, it cannot calculate the total similarity and total difference effectively. This problem is more pronounced on system with high resolution image and small window size. But if the window size is too large, it will include many textures with different depth and also occlusion areas. Performance of the system at the edge of the objects will be degraded. Thus, the window size should not be too small compared to the textureless area and not too large compared to the size of a texture. The ideal condition to generate correct disparity image is to process high resolution images with window size relatively equal to the texture size. The fifth architecture tries to increase the window size from 3x3 pixels to 5x5 pixels for comparison. It is the improved version of the forth architecture with window size increased to 5x5 pixels. The disparity image was calculated in a coarser level. When the window size is increased to 5x5 pixels, five lines of image data need to be stored in the line buffer. The number of subtractors and the size of the combinational logic circuit are also increased dramatically. For the 3x3 pixels window architecture, only nine subtractors are required for each pair of input windows. But 25 subtractors are required for each pair of input windows in the 5x5 pixels window architecture. Thus, the number of logic element was increased from 25,195 LEs in the fourth architecture to 54,338 LEs in the fifth architecture. In percentage, the number of LEs has risen by 115.7%. Because five lines of image data were stored in the line buffer, the number of required memory bits rises from 54,144 bits to 275,328 bits, which is an increment of 408%. The larger combinational logic circuit of the 5x5 pixel window architecture has reduced the maximum permitted clock frequency of the disparity processing unit to only 11.01 MHz. The system can achieve a frame rate of 35 fps. For the actual implemented system, the clock was set at 10 MHz and the frame rate was 30 fps. If a higher frame rate is required by the application, it is necessary that the critical path is broken into shorter path with intermediate registers in between. This is the pipelining technique, which will speed the system up with the cost of some additional resource and a carefully designed controlling/handshaking signal. Pipelining system is out of the scope of this research.

For the experimental result, the input images are still the same as in Figure 4.6 and Figure 4.8. The resulting images are shown in Figure 4.12 for input image with object and Figure 4.13 for input image with subjects.

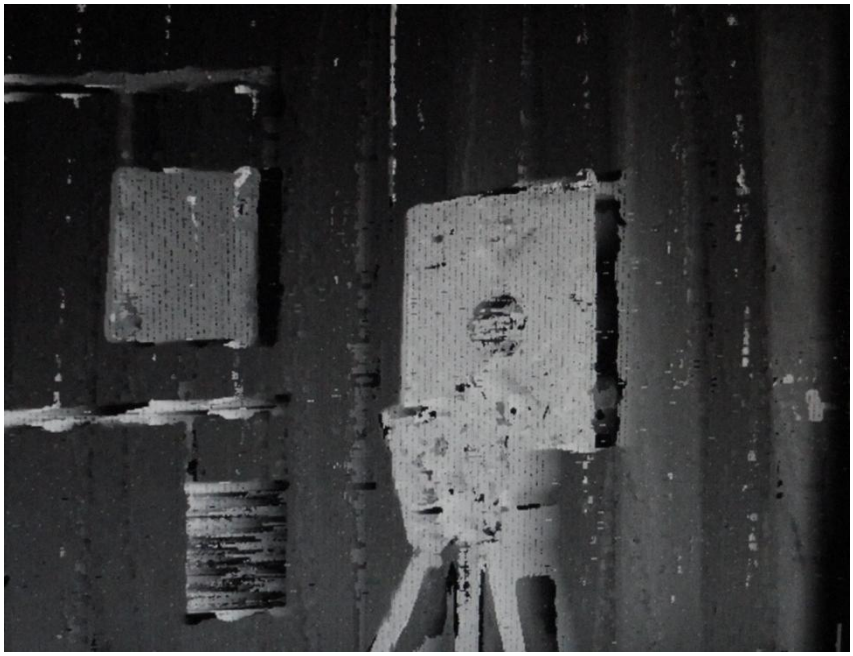

Figure 4.12 Disparity image of object calculated by the fifth architecture

Figure 4.13 Disparity image of subjects calculated by the fifth architecture.

Large window size architecture works well with high resolution image. It is a need for future research to discuss about high resolution, large window size depth-map system. However, the image size in our system is 640x480 pixels, which is relatively small as the cameras being used are wide angle cameras. Thus, the size of a texture is relatively small compare to the window size. Therefore, the affect of large window size in image quality is not much significant. Changing from 3x3 pixels window to 5x5 pixels window largely increases the resource usage, as shown in Table 4.1. For the 5x5 pixels window architecture, the percentage of resource reduction when replacing repetitive subtractors with register (equation (3.14)) would be higher. In other word, the technique is more effective on large window architecture. But, there is not enough resource on the Altera's Cyclone II FPGA to implement a system without resource minimization to compare with the fifth architecture.

4.1.6 Discussion

In term of memory bandwidth requirement, the memory bandwidth is so critical that the system wouldn't work without the bandwidth minimization techniques described in section 3.2.2. Thus, an optimized memory access scheme was used for all five architectures in this thesis. The memory bandwidth requirements are the same for all architectures, which equals to 163.7 MB/s as calculated in section 3.2.2 and it is independent of window size and disparity range.

The differences between these architectures are in the design of line buffers, disparity combinational logic circuit, size of the window and the disparity range. The first architecture implement disparity algorithm with 3x3 pixels window size, 32 pixels disparity range and register based line buffer. The second architecture change the primary line buffer from register based to memory based line buffer to save resources. The third architecture increases the disparity range from 32 pixels to 64 pixels to reduce the minimum visible range. The fourth architecture implements data reuse technique by replacing repetitive subtractors with registers to minimize resource usage of the third architecture. The fifth architecture increase window size from 3x3 pixels to 5x5 pixels for comparison.

Table 4.1 compares the five architectures implemented in this research. The comparison is made for window size, disparity range, type of line buffer, subtractor-register replacement, resource usage and speed.

Table 4.1 Resource usage of different architectures

Architecture used	First	Second	Third	Fourth	Fifth
Window size (pixels)	3x3	3x3	3x3	3x3	5x5
Disparity range (pixels)	32	32	64	64	64
Primary line buffer	register	memory	memory	memory	memory
Replace subtractors with registers?	No	No	No	Yes	Yes
LEs used	46,933	17,632	33,482	25,195	54,338
LEs saved	-	62.43%	-	24.75%	-
Memory bits used	13,120	54,080	54,080	54,144	275,328
Max frequency (MHz)	28.7	13.8	12.2	11.69	11
Max frame rate (fps)	80	41	39	37	35

Figure 4.14 visualizes the LEs usage of five architectures. It can be observed that when a resource minimization technique is applied, the number of LEs usage drops (the second and fourth architecture). And when the window size or disparity range increases, the system demands more logic elements.

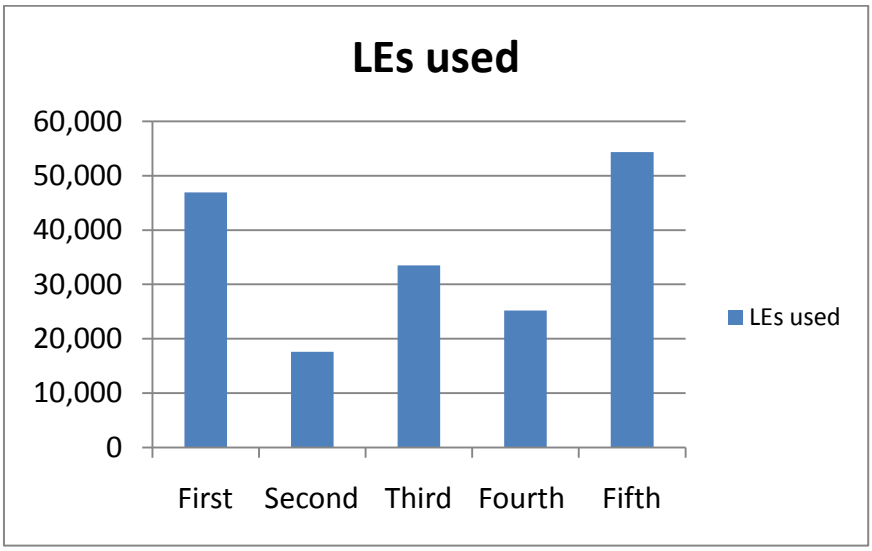

Figure 4.14 Logic element used in each architecture

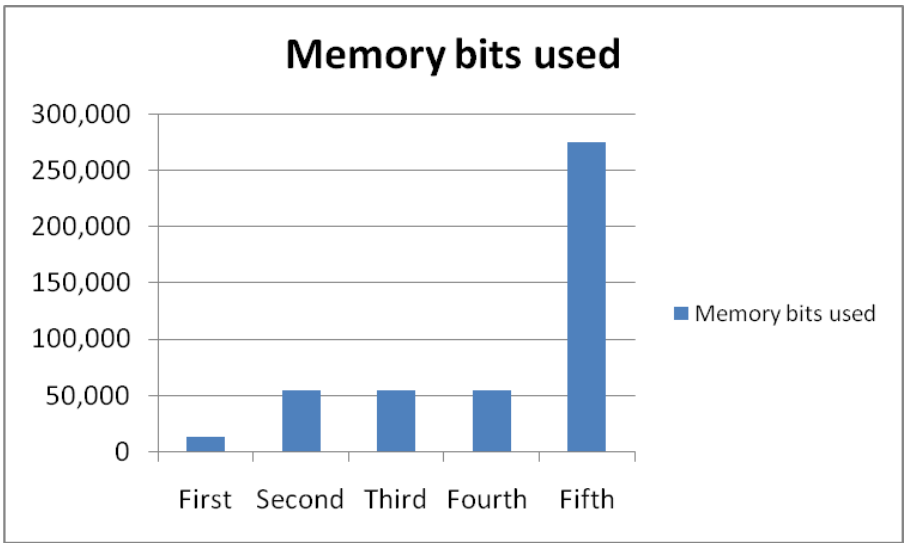

Figure 4.15 Internal memory bit usage

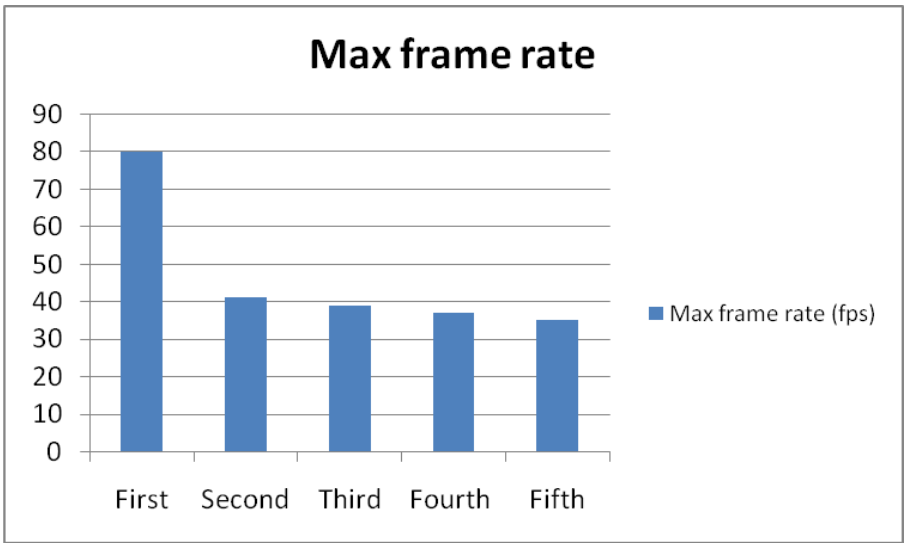

Figure 4.16 Frame rate comparison between five architectures

As shown in Figure 4.15, the amount of internal memory bit usage depends on the window size. When the window size changes from 3x3 pixels to 5x5 pixels, the length of the primary line buffer increases. Therefore, memory bit usage is largely increased. In Figure 4.16, the frame rate is slightly reduced when the system become more complex.

The fourth architecture is recommended over other four architectures because it has large disparity range, uses a minimized amount of resource and produces a resonable image quality. The large disparity range allows it to have shorter minimum visible range. The minimized resource techniques let it work in a co-operation with other image processing algorithms on a single FPGA without resource constrain.

4.2 Comparison between proposed architecture and other works

Table 4.2 summarizes the characteristic of the fifth architecture and other system implemented by Pascal Fua [9], Divyang K. Masrani [11], Bongsoon Kang [3], D. Chaikalis [22], Stefania Perri [6] and Dustin Lang [26]. In table 4.2, we are comparing between our architecture and the first three architectures [3, 9, 11], which implement the same SAD depth-map algorithm with similar window size and disparity range. The last three architectures implement different algorithms. They are provided for additional information. As shown in Table 4.2, our system was implemented on a DE2-70 development board, which has an Altera's Cyclone II FPGA. According to Altera's classification, Cyclone II is a low cost FPGA [31]. That means, our computation platform is much cheaper, slower and has less logic elements than other FPGAs listed in Table 4.2. Especially compared to Divyang's work, where he used four Stratix FPGAs connected to each other. Stratix is considered high end, expensive FPGA. For Pascal Fua, he used a work station and a DSP, which are of course much slower than FPGA. Therefore, he achieved the processing speed of only 4 frames per minute.

Table 4.2 Comparison between proposed system and other systems

Author	Algorithm used	Platform used	Frame size	Window size	Disparity range	Frame rate
Proposed system	SAD	Altera DE2-70 (Cyclone II) FPGA	640x480	5x5	64	35 fps
D. Chaikalis, et al	SAD	Xilinx Virtex XCV-2000E	1024x768	8x8	64	31 fps
Stefania Perri, et al	SAD	Xilinx FPGA	512x512	5x5	255	25.6 fps
Dustin Lang and James J. Little	SAD	Graphic hadware	640x480	5x5	54	10 fps
Pascal Fua	phase corelation + interpolation	work station DSP	-	3x3, 5x5, 7x7	50	0.4 f/min 4 f/min
Divyang K. Masrani, W. James MacLean	phase corelation	Four Altera's Stratix FPGAs	640x480	multi-scale	128	30 fps
Bongsoon Kang, et al	Absolute difference (AD)	Altera APEX20K1000-EBC652-3	320x240	11x11	64	15 fps

Our frame size is comparable with other works. Only Chaikalis used larger frame size, which is 1024x768 pixels. Our window size and disparity range are on average. Only Bongsoon Kang and D. Chaikalis used larger window size. However, increasing the window size does not improve the image quality significantly as shown in Figure 4.9. Besides, it makes the system performance worse at the edge of objects. Increasing disparity range reduces the minimum visible range as shown in equation (2.7) but with the cost of large amount of resources. As shown in table 4.3, most of the works used disparity range of 32 to 64 pixels.

We achieved the fastest processing speed among these works. Processing speed is important in real-time applications such as robotic navigation and camera surveillance. Figure 4.17 illustrates the difference between the processing speeds of these systems.

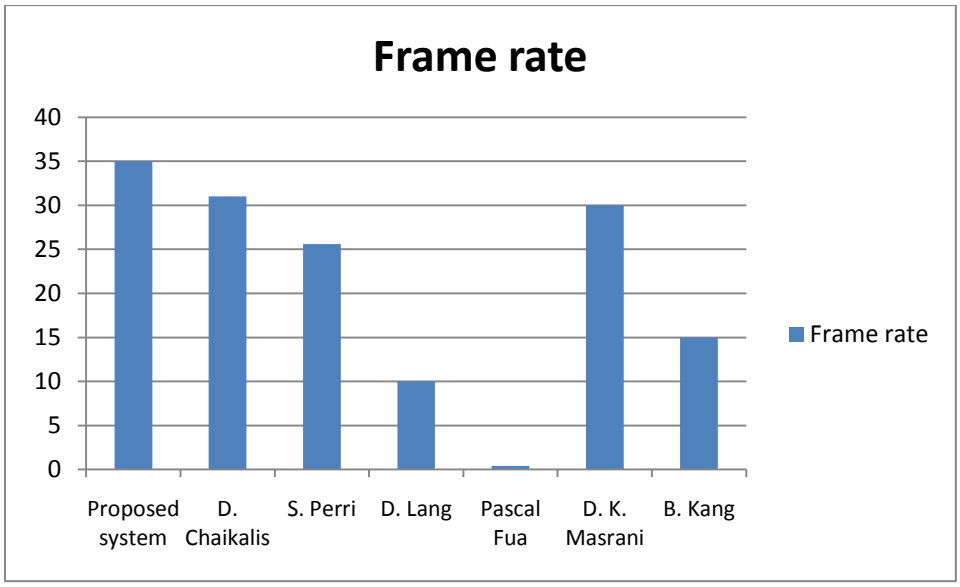

Figure 4.17 Frame rate of different systems

In Figure 4.17, we compared our fifth architecture with other systems. The fifth architecture is actually the slowest architecture among our five designs. The fourth architecture achieves the processing speed of 37 fps and the third one is 39 fps.

4.3 Summary

In this chapter, results of five different architectures implemented in this research were discussed. Comparing between these architecture, the first architecture has the highest processing speed but produces poor image quality because the subjects were closer to the camera than the minimum visible range. The resource usage is most inefficient in the first architecture. Image quality and efficiency of resource utilization was improved through the second to fifth architecture. The fifth architecture implemented the depth-map algorithm with the largest window size, largest disparity range and most efficient resource usage. All resource minimization techniques were applied on this architecture. In comparison with other works in the literature, our system was implemented on the cheapest and slowest FPGA. It has average image size, window size and disparity range but achieves the fastest processing speed.

CHAPTER 5

CONCLUSION

Stereo vision is part of low level image processing tasks. The computational intensive property makes it difficult to be implemented on general purpose processor or DSP. FPGA is the best choice to implement depth-map algorithm for research and testing purpose because it offers high parallel processing speed as well as reprogrammable ability. When depth-map algorithm is implemented on FPGA, resource limitation of FPGA is an issue. To overcome this problem, either the depth-map algorithm or the system architecture needs to be modified.

In this thesis, a design of a stereo vision system was presented. The system was implemented with two resource minimization techniques and five different architectures. The objective of minimizing resource usage without modification of depth-map algorithm was achieved. Real-time processing speed was achieved.

On the designing stage, the fixed optical axes camera calibration method was chosen over the rotatable optical axes system. The original SAD depth-map algorithm was used. The algorithm strives to find the maximum of similarity function and the minimum of difference function. From there, it gets the corresponding disparity value. The system was implemented on an Altera's DE2-70 development board and using Verilog hardware description language. A special multi-ports memory controller was designed which effectively eliminate memory contention. To transfer data between different clock domains, FIFO was used. The primary line buffer using dedicated memory bits proved to be more resource saving than a register-based line buffer. For the secondary line buffer, the register based line buffer uses less resource and is simpler than a memory based line buffer. The disparity matching module was designed with five different architectures to evaluate the resource usage and performance. The resource usage efficiency is better in the second architecture

compared to the first architecture because the memory based line buffer was implemented. And it is better in the fourth architecture compared to the third one since the data reuse technique was implemented.

In the disparity matching module, some of the absolute subtractions are repeated several times after each clock cycle. Therefore, a memory unit was used to store the result of the previous subtractions to minimize recalculations, which uses the logic element unnecessarily. Substituting the subtractors with a register to store the previous calculation results significantly reduces the logic elements used and it can be applied not only for depth-map processor system but also for other image processing systems. The method is more efficient if applied on complex systems such as disparity system with multi-scale, multi-dimensional disparity calculation. Increasing the disparity range will reduce the minimum visible range of the system but on the expense of resource requirement and system complexity. Increasing the window size makes the algorithm perform better on the textureless area but it degrades the performance at the edge of objects and requires larger amount of resources.

The depth-map system presented in this thesis proves to be a working prototype with minimized resource usage. It was implemented on a low-end FPGA, which is Altera's Cyclone II. The disparity matching module achieves real-time performance of 30 fps. It produces 295 MPDS for the 32 pixels disparity range architecture and 590 MPDS for the 64 pixels disparity range architecture. The clock frequency of the disparity module is only 10 MHz. That implies, the systems processing speed can be improved if a higher frame rate is required by the application. To improve the system speed, it is necessary that the pipelining technique is utilized. In comparison with other work in the literature, our system has average complexity in term of image size, window size and disparity range. It achieves the highest processing speed with the least amount of resources.

It is recommended that further work implements pipelining technique to further improve the processing speed. The work done in this research uses a low quality camera which has low signal to noise ratio, the resulting image has a lot of noise. Therefore, we recommend that further works use better cameras and a noise reduction circuit if possible. The depth-map algorithm is a low level image processing

algorithm; it cannot work alone in a marketable vision system. Thus, this architecture should be included in a more complete vision system.

REFERENCES

[1] D. K. Masrani, "Expanding Stereo-Disparity Range in an FPGA-system While Keeping Resource Utilisation Low," Master of Applied Science, Electrical and Computer Engineering, University of Toronto, Toronto, 2006.

[2] S. Ullman, *High-level vision: object recognition and visual cognition*: MIT Press, 2000.

[3] B. Kang, *et al.*, "Design of three-dimensional real-time disparity system using stereo images," *Current Applied Physics*, vol. 4, pp. 31-36, 2004.

[4] D. Scharstein and R. Szeliski, "A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms," *International Journal of Computer Vision*, vol. 47, pp. 7-42, 2002.

[5] C. Georgoulas, *et al.*, "Real-time disparity map computation module," *Microprocessors and Microsystems*, vol. 32, pp. 159-170, 2008.

[6] S. Perri, *et al.*, "SAD-Based Stereo Matching Circuit for FPGAs," in *Electronics, Circuits and Systems, 2006. ICECS '06. 13th IEEE International Conference on*, 2006, pp. 846-849.

[7] L. Ze-Nian and H. Gongzhu, "Analysis of disparity gradient based cooperative stereo," *Image Processing, IEEE Transactions on*, vol. 5, pp. 1493-1506, 1996.

[8] M. Humenberger, *et al.*, "A fast stereo matching algorithm suitable for embedded real-time systems," *Computer Vision and Image Understanding*, vol. 114, pp. 1180-1202, 2010.

[9] P. Fua, "A parallel stereo algorithm that produces dense depth maps and preserves image features," *Machine Vision and Applications*, vol. 6, pp. 35-49, 1993.

[10] V. Simhadri and Y. Ozturk, "RASCor: An associative hardware algorithm for real time stereo," *Computers & Electrical Engineering*, vol. 35, pp. 459-477, 2009.

[11] D. K. Masrani and W. J. MacLean, "A Real-Time Large Disparity Range Stereo-System using FPGAs," in *Computer Vision Systems, 2006 ICVS '06. IEEE International Conference on*, 2006, pp. 13-13.

[12] K. Shimonomura, *et al.*, "Binocular robot vision emulating disparity computation in the primary visual cortex," *Neural Networks*, vol. 21, pp. 331-340.

[13] C. Sun, "Uncalibrated three-view image rectification," *Image and Vision Computing*, vol. 21, pp. 259-269, 2003.

[14] R. I. Hartley, "Theory and Practice of Projective Rectification," *International Journal of Computer Vision*, vol. 35, pp. 115-127, 1999.

[15] Z. Chen, *et al.*, "A new image rectification algorithm," *Pattern Recognition Letters*, vol. 24, pp. 251-260, 2003.

[16] G. Hoffmann. *Luminance Models for the Grayscale Conversion*. Available: http://www.fho-emden.de/~hoffmann/gray10012001.pdf

[17] M. Grundland and N. A. Dodgson, "Decolorize: Fast, contrast enhancing, color to grayscale conversion," *Pattern Recognition*, vol. 40, pp. 2891-2896, 2007.

[18] P. Anandan, "A computational framework and an algorithm for the measurement of visual motion," *IJVC*, vol. 2(3), pp. 283-310, 1989.

[19] M. J. Hannah, "Computer Matching of Areas in Stereo Images," PhD, Stanford University, 1974.

[20] T. Kanade, "Development of a video-rate stereo machine," presented at the Image Understanding Workshop, 1994.

[21] M. Tekalp, *Digital Video Processing*. Upper Saddle River, NJ: Prentice Hall, 1995.

[22] D. Chaikalis, *et al.*, "Hardware implementation of a disparity estimation scheme for real-time compression in 3D imaging applications," *Journal of Visual Communication and Image Representation*, vol. 19, pp. 1-11, 2008.

[23] F. Solari, *et al.*, "Fast technique for phase-based disparity estimation with no explicit calculation of phase," *Electronics Letters*, vol. 37, pp. 1382-1383, 2001.

[24] L. Alvarez, *et al.*, "Dense Disparity Map Estimation Respecting Image Discontinuities: A PDE and Scale-Space Based Approach," *Journal of Visual Communication and Image Representation*, vol. 13, pp. 3-21, 2002.

[25] J. Ralli, *et al.*, "A method for sparse disparity densification using voting mask propagation," *Journal of Visual Communication and Image Representation*, vol. 21, pp. 67-74, 2010.

[26] D. Lang and J. J. Little. (2004, *GLStereo: Stereo Vision Implemented in Graphics Hardware*. Available: http://www.cs.toronto.edu/~dstn/papers/glstereo.pdf

[27] Terasic. *THDB-D5M Terasic D5M Hardware specification*. Available: http://www.cse.hcmut.edu.vn/dce/celab/altera/index2.php?option=com_docman&task=doc_view&gid=9&Itemid=36

[28] C. E. Cummings, "Synthesis and Scripting Techniques for Designing Multi-Asynchronous Clock Designs," *Synopsys Users Group Conference, San Jose, CA, 2001*, vol. Section MC1, 3rd paper.

[29] Altera. *Understanding Metastability in FPGAs*. Available: http://www.altera.com/literature/wp/wp-01082-quartus-ii-metastability.pdf

[30] C. E. Cummings, "Simulation and Synthesis Techniques for Asynchronous FIFO Design," *Synopsys Users Group Conference*, vol. Section TB2, 2nd paper, March 2002.

[31] Altera. *Altera Devices*. Available: http://www.altera.com/products/devices/dev-index.jsp