Video-Rate Stereo Vision on Reconfigurable Hardware

by

Ahmad Darabiha

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science in the
Graduate Department of Electrical and Computer Engineering,
University of Toronto

Video-Rate Stereo Vision on Reconfigurable Hardware

Master of Applied Science, 2003

Ahmad Darabiha

Graduate Department of Electrical and Computer Engineering

University of Toronto

ABSTRACT

This thesis describes the implementation of a stereo depth measurement algorithm in hardware on Field-Programmable Gate Arrays (FPGAs). This system generates 8-bit, sub-pixel disparities on 256 by 360 pixel images at video rate (30 frames/sec). The algorithm implemented is a multi-resolution, multi-orientation phase-based technique called Local Weighted Phase-Correlation. Hardware implementation speeds up the performance approximately 60 to 900 times that of the same algorithm running in software. In this thesis, we describe the programmable hardware platform, the base stereo vision algorithm and the design of the hardware. We include various trade-offs required to make the hardware small enough to fit on our system and fast enough to work at video rate. We show the depth map results from the functioning hardware. Although this research is specifically focused on phase-based stereo vision FPGA realizations, most of the design issues are common to other DSP and Vision applications.

Acknowledgements

First and foremost, I would like to thank my advisors, Jonathan Rose and James MacLean. They made my graduate study an invaluable and exciting experience. I learned a lot from Jonathan's insight, wisdom and his passion for "producing knowledge" as the goal of the research. James also provided a continuous source of encouragement and confidence. Our interesting conversations and detailed discussions were fundamental to the success of this project.

I am grateful to Professor Allan Jepson, Professor Parham Aarabi and Professor Mireille Brouck for volunteering to be in my defense committee and for reading my thesis.

I would like to give special thanks to Dave Galloway and Marcus van Ierssel for their contribution to this work in the form of TM-3A maintenance and support. Without their help this work would have never been possible.

All my friends and the fellow grad students in LP392 made our lab an enjoyable and productive room. I would like to specially thank Rubil Ahmadi, Anish Alex, Jason Anderson, Igor Arsovski, Reza Azimi, Navid Azizi, Tomasz Czajkowski, Mehrdad Eslami, Yadollah Eslami, Tooraj Esmailian, Joshua Fender, Vincent Gaudet, Warren Gross, Marcus van Ierssel, Paul Kundarewich, Kostas Pagiamtzis, Lesley Shannon and Andy Ye, in the alphabetical order.

I am grateful beyond measure to my family, in particular to my parents for their encouragements. They have always kindly respected my decisions and supported me to achieve my goals. I am also grateful to the members of my extended family in Mississauga for their kindness and support during the entire course of my studies.

Table of Contents

CHAPTER 1

Introduction

Vision enables humans to navigate and gather information about the surrounding environment. Detecting a human face in a scene, recognizing objects and understanding people's emotional moods from their facial expressions are only a few examples of vision tasks that we all do automatically in everyday life. Although these tasks are performed with minimum effort, analysis and simulation of these processes are highly complex.

In the last three decades researchers have tried to give vision capabilities to automated systems such as robots, which would lead to more intelligent systems with a wider range of capabilities. This is very much a work-in-progress: the way human brain processes visual information is not yet exactly understood, and image processing tasks are usually computationally intensive.

In building vision systems, designers have essentially two options: developing the vision algo-

rithms in software and running them on a standard processor, or designing custom hardware specially tailored for the application. Custom designed hardware can perform operations faster and also allows taking advantage of parallelism existing in many image processing algorithms, but it is rarely tried in practice because the resources required are not usually available at an affordable price and the design process takes a long time. These facts have usually limited researchers to the first option, which is to develop algorithms that can be run as fast as possible on standard processors.

In the last few years, a third solution for vision system designers has become viable due to the rapid growth in capacity and speed of programmable hardware. Programmable hardware has also been used successfully in some non-vision signal processing applications [35]. This method allows designers to configure the chip according to the specifications of the algorithm cheaply and quickly because it eliminates the most expensive and time consuming part of Application Specific Integrated Circuit (ASIC) Design, namely, IC fabrication. It also reduces the debugging time because one can typically re-compile the design in few hours and re-configure the FPGAs in less than a second.

In this research, the goal is to explore the feasibility of using a programmable hardware platform to implement one important feature of the human vision system: to estimate the depth structure of a three-dimensional scene from two images seen by left and right eyes. This task is known as “stereo vision”. A robot equipped with a real-time stereo vision system can estimate the distance of an objects around it and use this information to avoid collision, or as an input to an object recognition module. By exploiting the useful features of reconfigurable hardware along with parallelism, our goal is to speed up the depth reconstruction process such that the system will be able to “see” in stereo in real time (30 frames per second).

The nature of this research neccessitates focusing both on software and hardware. We investigate different techniques in software already proposed for disparity measurements and choose a high performance algorithm that is also appropriate for hardware implementation. The algorithm we have implemented is called “Local Weighted Phase-Correlation algorithm” [11]. The key issue is to modify the original algorithm to make it efficiently implementable in hardware

with minimal loss in algorithm performance. In hardware, the focus will be on designing a circuit to run the algorithm in video rate (30 frames/sec). We have implemented our system on a reprogrammable board called Transmogrifier-3A (TM-3A) [18][28] developed at the University of Toronto. In the circuit design process, we exploit the parallelism existing in the algorithm and at the same time fit the whole computations to the limited hardware resources available on the TM-3A.

This thesis is organized as follows. Chapter 2 provides the background on stereo vision algorithms in software and hardware. It describes stereo disparity measurement methods and focuses on the specific method used in our stereo system. In hardware, it introduces the FPGAs and some example systems based on reconfigurable chips. Chapter 3 describes the implementation of the Local Weighted Phase-Correlation disparity measurement algorithm on programmable hardware. In Chapter 4, the depth map results from hardware system are presented and compared with the results from the original software based algorithm. Chapter 5 presents conclusions and future work.

CHAPTER 2

Background

The nature of this research requires knowledge of both stereo vision techniques and hardware design issues. This chapter provides the background for both of these areas. Section 2.1 describes the theoretical basis of stereo vision. It includes a brief description of major approaches and then focuses on the approach adopted in this work. Section 2.2 introduces programmable hardware and describes the architecture of a general Field-Programmable Gate Array (FPGA). It then looks at a few hardware systems that are based on FPGA devices. Finally, Section 2.3 reviews previous work on implementing vision and image processing applications on reprogrammable hardware.

2.1 Stereo Vision Basics

Stereo vision is the task of reconstructing depth information encoded within multiple images. In this thesis, by stereo vision we mean reconstructing the depth information only from two images, a task also known as *binocular stereopsis*. All the stereo vision techniques stem from

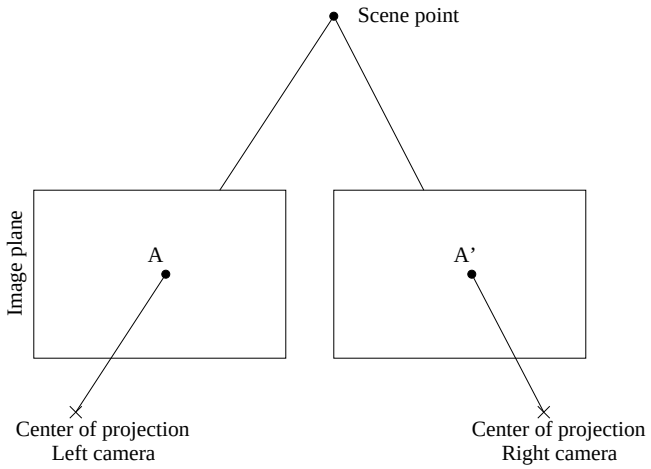


Figure 2.1: A and A' are corresponding points originated from one scene point

the fact that when two images are taken from different view points of a scene, the projection of a 3D point will have different locations on the two images. The shift between the corresponding projected points, also referred to as disparity, can be used to estimate the distance of the 3D point to the cameras. Figure 2.1 shows a pair of cameras and the corresponding points from the same scene point.

The goal of any stereo vision system is to establish the correspondence between the two points arising from the same element in the scene. This problem is called *stereo matching* or the *correspondence problem*. Once matching points are detected, one can simply extract disparity as the shift between these points and then estimate the scene point distance by simple calculations. Figure 2.2 illustrates how distance is calculated from disparity. In this figure and through this thesis, we have made two assumptions for the camera set up: 1) the cameras have the same focal length; and 2) the cameras are vertically aligned and have parallel optical axes. In Figure 2.2, the disparity, *d*, is defined as the shift between two corresponding points:

$$d = u - u' \tag{2.1}$$

and then the distance, *Z*, can be calculated as:

$$Z = \frac{fT}{d} \tag{2.2}$$

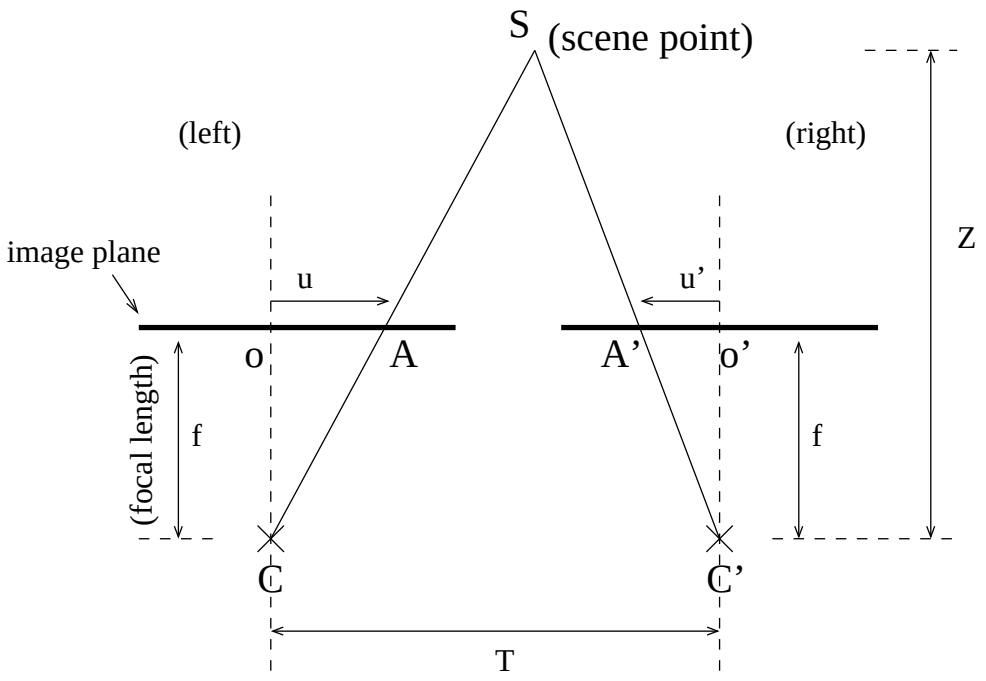


Figure 2.2: Calculating depth from disparity

where f is the camera focal length and T is the distance between the optical centers of the two cameras. Eq. (2.2) shows that given a fixed focal length and camera separation, distance is inversely proportional to disparity. So, the bigger the disparity, the closer the scene point is to the cameras.

To solve the correspondence problem, the first approach is to pick one pixel in one image and then search through a 2-D region around that pixel location in the other image to find the corresponding point. However, it can be shown that a 1-D search is sufficient due to the epipolar constraint. This constraint guarantees that if A is the projection of a scene point in one image, then the corresponding point, A', in the other image will lie on a straight line, *epipolar line*, which is the intersection of the image planes with a plane that contains point A and the two centers of projection.

In a pair of cameras that are vertically aligned and have parallel optical axes, the epipolar line will coincide with a scan line of the image (Figure 2.3). This property makes the search process simpler compared with non-horizontal epipolar lines. This will be of great advantage for a hardware implementation because in hardware the pixel stream is usually received in hori-

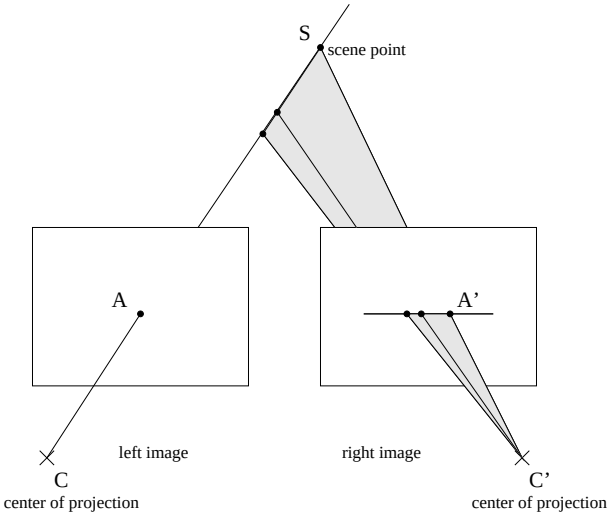


Figure 2.3: Epipolar Constraint

zontal order.

2.1.1 Stereo Matching Techniques

Stereo matching is the task of finding corresponding points between two images. This task is usually complicated by several factors such as lack of texture, occlusion and lighting variations. Regions of the image with not enough texture will make it difficult to find corresponding points. Also, sometimes a scene point is visible in one image but is occluded in the other image. In these cases, there is no corresponding point to find. In addition, sometimes the lighting varies between different view points such that the same scene point will have different gray scale intensities in the two images. This will add some noise to the final disparity map results.

Researchers have proposed various techniques to solve and improve the performance of stereo matching. The key question that will affect the matching performance and strategy is the selection of matching primitives. Depending on the primitive used, matching techniques can be categorized into three major groups: 1) Intensity-Based; 2) Feature-Based; and 3) Phase-Based.

Intensity-based techniques use the pixels intensity or brightness as the matching primitive. These techniques assume that the image intensity corresponding to a 3-D point remains the same in binocular images. In this method the intensity of each pixel in one image is compared

with the intensity of a range of pixels in the other image in order to find the most similar pixel. To improve the performance, some intensity-based techniques use window-based comparison instead of point-to-point comparison. The important issue in intensity-based techniques is the correct selection of window size: Small windows may not have enough image structure, and hence, lead to false matches; Large windows might lose fine image structures that are much smaller than the size of the window. Some techniques such as an adaptive matching window approach [21] and coarse-to-fine approach [24][5] have been proposed to overcome the window size problem. The fundamental weakness of intensity-based techniques, regardless of the comparison approach, is that they are sensitive to brightness variations in binocular images.

Feature-based techniques use sparse primitives such as corners, edges [4] or straight line segments [3]. In feature-based techniques, the input images are usually passed through pre-processing blocks which detect features such as edges or corners. The matching block then finds the corresponding features in two images and assigns disparities to them. The major limitation of all feature-based techniques is that they can not generate dense disparity maps and hence they often need to be used in conjunction with other techniques.

In *Phase-based* techniques the disparity is defined as the shift necessary to align the phase value of band-pass filtered versions of the two images. In [10], phase-based methods are shown to be robust when there exists smooth lighting variations between stereo images. It also shows that phase is predominantly linear and hence reliable approximations to disparity can be extracted from phase displacement. It is interesting to note that neurophysiological data also imply the importance of phase information. These data suggest that the first stage of stereo disparity processing in the visual cortex in cats is thought to use a phase-based approach [8]. The first step in any phase-based method is to extract the phase from input images. One commonly used approach is to pass the input images through complex-valued quadrature pair filters. The phase of the complex-valued output of these filters is used as the primitive for stereo matching.

Figure 2.4 shows a sample pair of stereo images from a scene and the result of stereo matching in the form of a depth map. In the depth map image, the distance is encoded by grey scale: the closer objects are brighter. In the next section, we will explain one particular phase-based

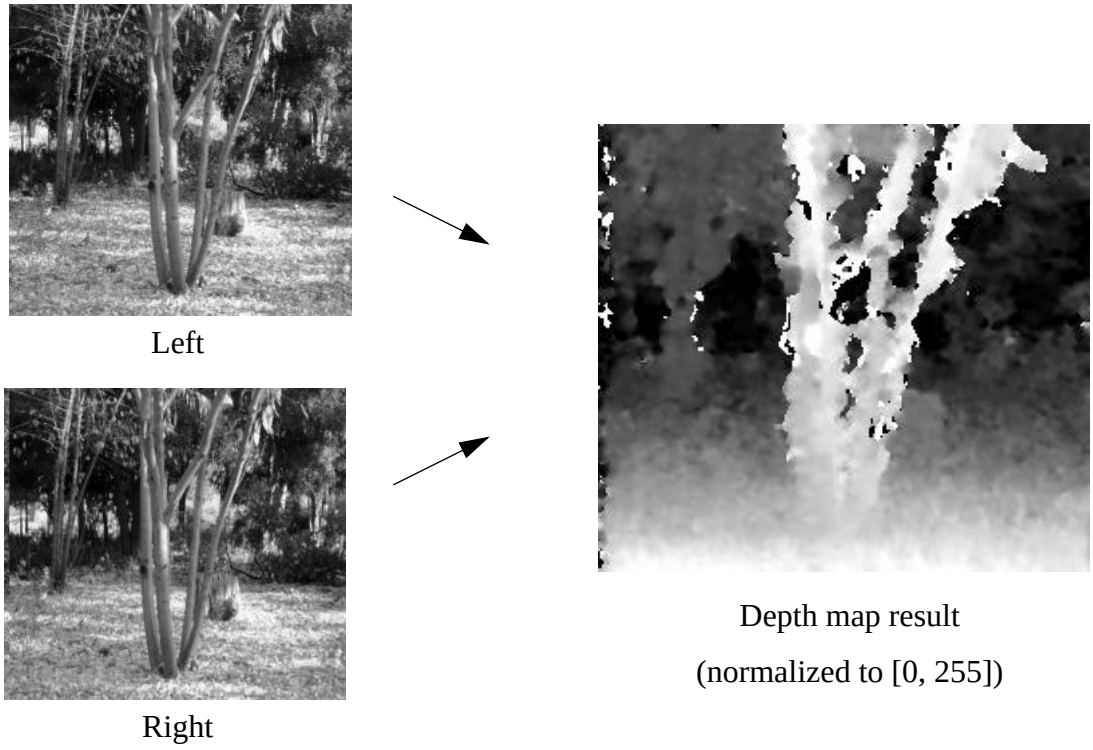


Figure 2.4: Tree stereo images and a sample depth map
(courtesy of Bob Bolles, SRI International)

method, *Local Weighted Phase-Correlation*, which is the basis of the stereo system developed in this research.

2.1.2 Local Weighted Phase-Correlation Algorithm

The stereo system developed in this research is based on a phase-based stereo matching technique called “Local Weighted Phase-Correlation” [11] (In this thesis, we also refer to it as *LWPC* algorithm.). This algorithm decomposes the input images into multiple scales. All the scales are then passed through multiple G2-H2 quadrature filter pairs, each of them oriented in unique directions using steerable filters [13]. The left and right pairs of filter outputs for each scale and each orientation are passed through a correlation block which assigns similarity measures, or voting functions, between one pixel and its shifted versions in the other image. These voting functions are then combined over all the scales and orientations to build the

overall voting value. The shift corresponding to the location of the peak response will be declared as an estimate for the disparity. The LWPC algorithm can be summarized in four steps (Figure 2.5):

1. Create a Gaussian pyramid [13] with total number of M scales for both left and right images. Then, decompose each scale of the pyramid using oriented quadrature-pair filters. Assuming that $K_j(x)$ is the filter impulse response of the j^{th} orientation, we can write the complex-valued output of the convolution of $K_j(x)$ with each scale of left and right images, $I_l(x)$ and $I_r(x)$, as:

$$O_l(x) = \rho_l(x)e^{i\varphi_l(x)} \quad , \quad O_r(x) = \rho_r(x)e^{i\varphi_r(x)} \tag{2.3}$$

   in the polar representation, where $\rho(x) = |O(x)|$ is the amplitude and $\phi(x) = \arg[O(x)]$ is the phase of the complex response.

2. Compute the voting function $C_{(j,s)}(x,\tau)$ as:

$$C_{(j,s)}(x,\tau) = \frac{W(x) \otimes [O_l(x)O_r{}^*(x+\tau)]}{\sqrt{W(x) \otimes |O_l(x)|^2}\sqrt{W(x) \otimes |O_r(x)|^2}} \tag{2.4}$$

   where $W(x)$ is a smoothing, small, localized window and τ is the pre-shift of the right filter output.

3. Combine the voting functions $C_{(j,s)}(x,\tau)$ over all orientations, $1 \le j \le F$, and scales, $1 \le m \le M$, where F is the total number of orientations and M is the total number of scales:

$$S(x,\tau) = \sum_{j,m} C_{(j,m)}(x,\tau) \tag{2.5}$$

4. For each position x, find the τ value corresponding to the peak in the real part of $S(x, \tau)$ as a good estimate for the true disparity.

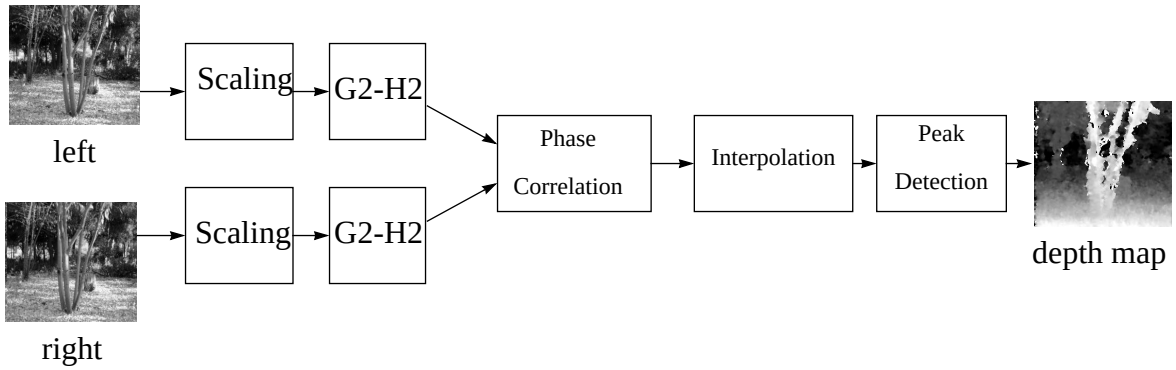


Figure 2.5: Block diagram of Local Weighted Phase-Correlation disparity matching algorithm

Two major features make this algorithm a good candidate for hardware implementation: First, it is primarily composed of linear operations which are easier to implement. Second, there is no iteration or any explicit coarse-to-fine control strategy. This property makes the real-time flow of data possible through the hardware system. In the next sections, we will describe the general architecture of programmable hardware and the Transmogrifier-3A board which is used as the platform to implement stereo vision system.

2.2 Field-Programmable Gate Arrays (FPGAs)

An FPGA is a chip that allows its user to control and reprogram the functionality of its logic circuits. All FPGAs consist of three major components [6]: 1) logic blocks; 2) I/O blocks; and 3) programmable routing as shown in Figure 2.6. To implement a circuit on an FPGA, each logic block is programmed to perform a small portion of the logic required by the circuit and each I/O block is programmed to act as an input or output as required by the circuit. The programmable routing is also configured to make all the necessary connections between logic blocks and also from logic blocks to I/O blocks.

When compared to custom hardware, the first advantage of an FPGA is shorter manufacturing cycle and the ability to modify the existing systems only by re-compiling the design instead of re-fabricating the chip. The second advantage is that using FPGA is cheaper: Building an ASIC costs in the order of hundreds of thousands of dollars, while FPGAs cost less than a few

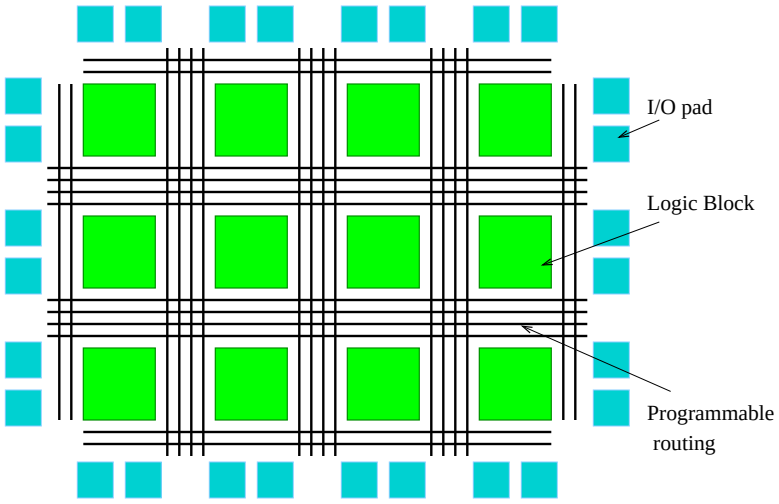


Figure 2.6: Architecture of a generic FPGA

thousand dollars.

The processing power of an FPGA is directly proportional to the processing capabilities of its logic blocks and the total number of logic blocks available in the array. Currently most of the commercial FPGAs use logic blocks that contain one or more Look-Up Tables (LUTs). A k-input LUT can implement any binary function of k logic inputs. Figure 2.7 shows the architecture of a simple LB containing one 4-input LUT and one flip flop for storage. Modern FPGAs also contain blocks of on-chip memory. For example, the chips used in this work contain 160 blocks of 4kbits of RAM and 38,000 LUT-flip flop pairs. Current commercial FPGAs can have arrays containing as many as 93,000 LUTs and flip-flops in a single FPGA. For the designs that are too large to fit on a single FPGA, a group of FPGAs connected with a pro-

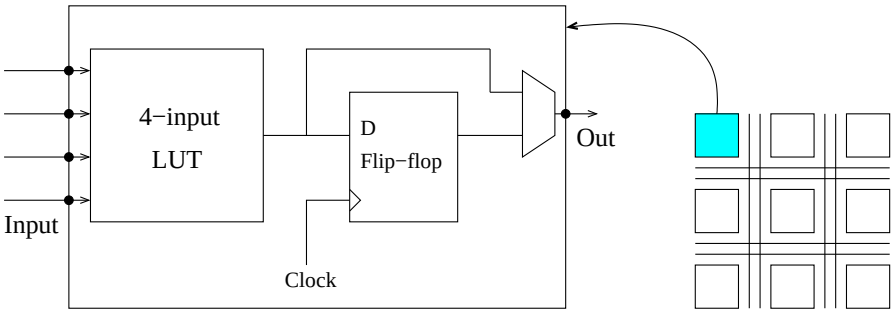


Figure 2.7: Simplified architecture of a logic block with one 4-input LUT and one flip flop. The flip flop at the output of LUT allows storage of the LUT output and hence implementing sequential circuits.

grammable interconnection network can be used.

2.2.1 Reconfigurable Systems

A variety of reconfigurable systems have been developed that contain more than one FPGA device. These systems are used for implementing designs that do not fit on a single FPGA. They typically consist of FPGAs, some method of programmably interconnecting FPGAs, and might have external memory resources or video interface circuitry. The PAM system [31] consists of a 4x4 mesh of Xilinx XC3090 chips with each FPGA connected to each of its four Manhattan neighbors through 16 direct connections. This system has a total of 4 MB of RAM. The Splash-2 parallel processor board [2] contains 16 processing elements (PEs) fully connected via a 16x16 crossbar switch which is controlled by a 17th processing element. Each PE consists of a Xilinx XC4010 FPGA with 256K 16-bit words of memory. The PARTS reconfigurable computer [32] is a single PCI board that utilizes a variety of Xilinx chips with a maximum total number of about 35,000 4-input LUTs.

2.2.2 Transmogrifier-3A

The Transmogrifier-3A (TM-3A) [15],[28] is a reconfigurable platform built at the University of Toronto and is used in our work to implement video rate stereo depth reconstruction. The TM-3A is a programmable hardware architecture containing four Xilinx Virtex 2000E [32] FPGAs. Each Xilinx Virtex 2000E chip contains 38,000 LUTs, which in comparison is 20 times more than the number of LUTs in a Xilinx XC4028. Virtex 2000E chips also provide 640 Kbits of on-chip memory. Each FPGA on the TM-3A is connected to the other three FPGAs via a 98-bit bus. Each chip is also connected to a 256K x 64bit synchronous external SRAM, an I/O connector and a nibble bus which allows communication with a housekeeping FPGA. The housekeeping chip communicates with the host computer for download and control functions. A video encoder and decoder chip [26] is included on the TM-3A board to give the ability to receive NTSC video and also send output results directly to a display. TM-3A can operate at frequencies of up to 100 MHz.

The TM-3A can be used to implement designs that are larger than the capacity of one single FPGA. A set of TM-3A software tools provide support for design compilation, routing between FPGAs, system management, debugging and user interface. The board can commu-

nicate with a program running on a UNIX workstation using the ‘ports package’ [29]. The ports package facilitates design stimulus, data transfer and design debugging. TM-3A also provides two other tools for debugging: 1) JTAG boundary scan which reports the values of I/O pins on all FPGAs; and 2) a circuit debugging program called tm3step that allows running the system clock for a given number of cycles and displays the content of any flip flop in the design.

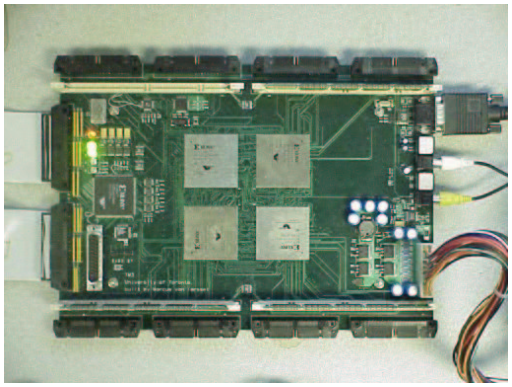


Figure 2.8: Transmogrifier-3A board

2.2.3 VHDL Language

This section provides a brief introduction to the hardware description language used in this work to build our hardware system. VHDL is the VHSIC Hardware Description Language. VHSIC is an abbreviation for Very High Speed Integrated Circuit. This language can describe the behavior and structure of electronic systems, but is particularly suited as a language to describe the structure and behavior of digital electronic hardware designs, such as ASICs and FPGAs.

VHDL allows the behavior of complex electronic circuits to be captured into a design system for automatic circuit synthesis or for system simulation. Similar to high-level languages such as Pascal, C and C++, VHDL includes features useful for structured design techniques and offers a set of control and data representation features. One important difference between VHDL and other programming languages is that VHDL provides features that allow describing concurrent events. This is because the hardware described by VHDL is inherently concurrent in its operation.

As an example, the following VHDL code describes a simple comparator. This comparator receives two 8-bit inputs, A and B, and generates a 1-bit output, EQ, based on whether A is equal to B or not.

```
------------------------------
-- Eight-bit comparator
------------------------------
library ieee;
use ieee.std_logic_1164.all;
entity compare is
   port (A, B: in std_logic_vector(0 to 7);
         EQ: out std_logic);
end compare;

architecture my_compare of compare is
begin
   EQ <= '1' when (A = B) else '0';
end my_compare;
```

In the code above, the VHDL keywords are highlighted in bold face type. A complete reference for VHDL and a guide to VHDL for synthesis can be found in [17] and [27] respectively.

2.3 Vision Applications on Reconfigurable Systems

The increase in capacity and speed of FPGAs in the last few years has led to a variety of reconfigurable systems being used for accelerating computationally intensive machine vision and graphics applications. The Transmogrifier-2A [24], the predecessor of TM-3A, was used as a platform for video rate face detection [21] and texture mapping [35]. In INRIA [9], a 4 x 4 matrix of small FPGAs is used to perform the cross-correlation of two 256 x 256 images in 140 msec.

A small number of hardware based stereo systems have also been developed in the past few

years. A stereo engine was developed on PARTS system based on the census transform, an intensity-based stereo matching method mainly consisting of bit-wise comparisons and additions [36]. The PARTS stereo engine generates dense disparity maps of size 240 x 320 pixels at video rate. In [15], a combination of FPGA and Digital Signal Processors (DSPs) is used to perform edge-based stereo vision. It uses FPGAs to perform low level tasks such as edge detection and DSPs for high level image processing tasks. There are also stereo systems that are based on custom designed hardware as opposed to reprogrammable hardware: in [21], a system is described that is composed of three boards built from discrete components plus a C40 DSP-array board and a real-time OS board. This system performs sum-of-absolute-difference correlation in 30 frames/seconds on images of size 200 x 200 pixels.

These video rate hardware-based stereo systems are principally implementing intensity-based or a combination of intensity and feature-based matching techniques. As we mentioned in Section 2, phase-based matching techniques have a better performance than intensity-based techniques in the presence of brightness variations in binocular images. Phase has also the advantage of linearity and stability. To date, no phase-based video rate stereo system is believed to be implemented on hardware other than the one described in this thesis. This is likely due to the large amount of computation required for extracting and manipulating the phase.

The following chapter describes development of a phase-based multi-resolution multi-orientation stereo vision system on an FPGA platform that generates a 256 x 360 pixel depth map in video-rate and with 8-bit, sub-pixel accuracy. The basis of the algorithm is Local Weighted Phase-Correlation technique [11], as explained in Section 2.1.2, which is one of the highest performance algorithms to date for disparity matching. By implementing this algorithm on FPGAs, we achieve a speed-up factor of approximately 60 to 900 over its software implementation with the same parameters.

CHAPTER 3

Hardware Design

When implementing a complex algorithm on reprogrammable hardware, the key issue is that there is a fixed amount of hardware available in each FPGA. These hardware resources include logic capacity, on-FPGA and off-FPGA available memory, memory access bandwidth and chip-to-chip communication bandwidth. Achieving the best overall performance requires efficient usage of all hardware resources.

This chapter describes implementation of Local Weighted Phase-Correlation (LWPC) algorithm as introduced in Chapter 2. In this work, for parallel and efficient hardware implementation of the stereo depth measurement, some modifications are introduced to the original LWPC algorithm. Three major modifications are: 1) employing fixed-point data representation instead of floating-point representation; 2) changing the location of the smoothing Gaussian windows; and 3) using $L_1 - norm$ instead of $L_2 - norm$ in calculating phase correlation. To analyze the effect of these modifications on the final performance of the system, we first built a software model that emulates the behavior of hardware. Once all the design parameters were

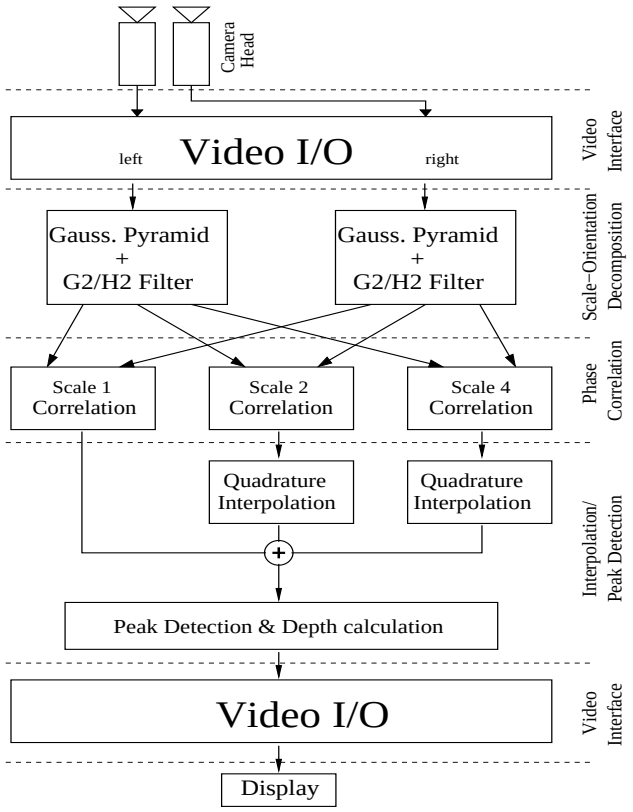


Figure 3.1: High Level Stereo System Architecture. Two video inputs are received from parallel cameras and are passed through multi-scale multi-resolution LWPC algorithm. The Video Output Interface sends the depth map results to display.

decided in the emulation version of the algorithm, we started to design the hardware system. In this chapter, we first explain the major building blocks of the system and the distribution of tasks over the four FPGAs available on TM-3A board. Then, we will discuss the advantages and effects of modifications on the overall system performance.

3.1 System Overview

The architecture of the stereo vision system is described in Figure 3.1. This architecture is derived from LWPC algorithm illustrated in Chapter 2. It consists of four major units: the Video Interface Unit, the Scale/Orientation Decomposition Unit, the Phase-Correlation Unit and the Interpolation/Peak-Detection Unit. Each of these units are implemented on one of the Xilinx Virtex 2000E chips available on the Transmogrifier-3A.

A brief overview of the hardware units is as follows: After being buffered, image streams from

left and right cameras are passed, in the form of pixel grey scale values, through Gaussian pyramid scaling and steerable G2-H2 filters. The outputs of left and right G2-H2 filters are then merged in a correlation block which computes the voting functions based on the similarity of the left filter output with shifted versions of the right filter output. At the end of the correlation unit, we have a series of three dimensional arrays, each corresponding to a specific scale and orientation. The first two dimensions of these arrays are X and Y coordinates of the pixel in the image and the third dimension has voting function values for each of the candidate disparities, ranging from 0 to the maximum disparity. Since there are multiple scales of image at this stage, we need to interpolate the voting function arrays in X, Y and also in disparity domain, τ, in order to combine all the voting functions results. As the final step, for each pixel location, the disparity index, τ_{max}, corresponding to the maximum voting function, is detected. By performing linear interpolation on voting function results, sub-pixel accuracy can be obtained for disparity estimates. This estimate is then translated to a depth value using Eq. (2). The following sections will describe each of the major units of the hardware stereo system in more detail.

3.2 Video Input Interface Unit

This unit receives composite NTSC video signals from two CCD cameras (Figure 3.2). The analog NTSC input is converted to digital RGB signals using the designated NTSC decoder chip [23] on the TM-3A board. Realization of a video-rate stereo vision system requires two simultaneous video signals from two cameras. To do this, we need two video input channels on board, each receiving video from one camera. But there is only one NTSC decoder on the TM-3A board and it has only one video input channel available at a time.

To solve the single video input channel problem, we alternate the source selector of the decoder chip after each frame is grabbed such that in each second 15 frames from the left camera and 15 frames from the right camera are received. The drawback of this solution is that the overall processing rate of the stereo system in practice is half of the standard video rate (15 frames/sec). However, this limitation is only arising from the fact that TM-3A is a general purpose reprogrammable board with one video input channel. If there was a way to receive two 30 frame/sec video signals in parallel, the rest of the current stereo system would

Figure 3.2: The camera head with two 1/2” CCD cameras. The focal length of the camera lenses is 12mm and the separation between the optic centers of the cameras is 70mm.

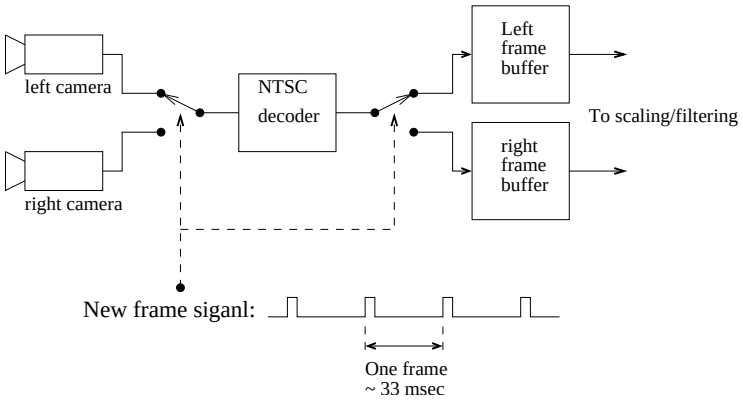


Figure 3.3: Using source selector to alternate between left and right cameras

have been able to generate disparities in full video rate. In fact, in the current design, the input image buffers are read and processed in 30 frames/sec even though the input image changes every 15 frames/sec. Figure 3.3 describes the source selector alternation after each frame.

At the end of the Video Input Interface Unit, the input interlaced images of 256 by 360 pixels are written into left and right frame buffers. The Scale/Orientation Decomposition Unit will, then, read these frame buffers in non-interlaced order and will process these image streams as explained in next section.

3.3 Scale-Orientation Decomposition Unit

This unit consists of two major sub-blocks: the Scaler block and the G2-H2 Filter block. It reads from the frame buffers and after scaling and filtering, sends the outputs to the Phase-Correlation Unit. For parallel processing of left and right image streams, two instantiations of the scaler and filtering sub-blocks are used to separately process the two image streams.

3.3.1 Scaler Block

As discussed in Chapter 2, combining the matching data from multiple scales of the stereo images improves the overall performance of stereo matching. This improvement is because each scale leads to one correct disparity and some possibly false disparities. By combining the matching data across all the scales, the false disparities can be reduced [11].

Figure 3.4 shows the architecture of the Scaler Block. This block down-samples the original image in two steps, each time by a factor of 2 in both horizontal and vertical directions. To avoid aliasing, which can happen as a result of down-sampling, we pass the input image through a low-pass anti-aliasing filter. In our system, a three-tap Gaussian FIR filter is used as an anti-aliasing filter in both horizontal and vertical directions. The Scaler Unit outputs two Gaussian pyramids, one for left image and one for right image.

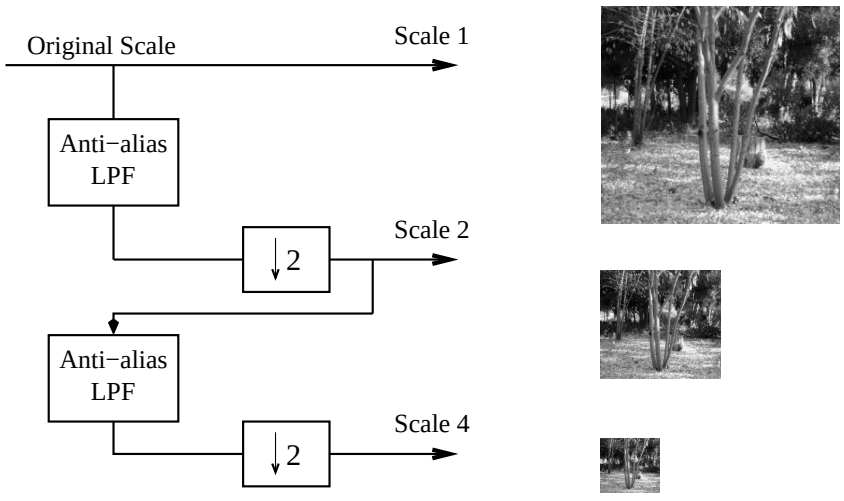


Figure 3.4: Two-step scaler architecture with anti-alias low-pass filters and 2D down sampling

3.3.2 G2-H2 Filter

G2-H2 Filter is the second block of the Scale/Orientation Decomposition Unit. As discussed in Chapter 2, G2/H2 filters are a common approach for phase extraction. G2/H2 is a complex-valued quadrature-pair filter. The real and imaginary parts of this filter have the same amplitude spectra, but they are $90°$ out of phase. In other words, G2 can be expressed as the second derivative of a Gaussian function and H2 is its Hilbert transform. To extract more features from the input images, we apply two different orientations of G2/H2 filters (The original LWPC algorithm uses three directions, but we decompose in two directions due to the space considerations.). In [13], it is shown that G2/H2 filters are "steerable" which means any arbitrary orientation of G2 or H2 filters can be expressed as a linear combination of a set of basis filters. The basis set for G2 filter has three filters: G_{2a}, G_{2b} and G_{2c}, while H2 has a basis set with four filters: H_{2a}, H_{2b}, H_{2c} and H_{2d}. In hardware, we have implemented all the seven basis filters and then, by combining the basis filter outputs by proper coefficients, we construct two oriented filters in $45°$ and $-45°$ degrees. One advantage of implementing basis filters is that any other filter orientation can be constructed with minimum extra cost.

Figure 3.5 shows the architecture of the G2/H2 filter block. In this block, the main computation is performed in the basis filters. Each basis filter G_{2a}, G_{2b}, ..., H_{2d} is originally a 9×9 FIR filter. However, for hardware implementation purpose, since the first and last coefficients are negligible compared with the other coefficients, we implemented them as 7×7 filters. In the next paragraphs, we will describe the implementation of the basis filters in hardware.

One important feature of G2/H2 filters is that they are X-Y separable. An X-Y separable filter has an impulse response that can be expressed as the product of two functions: one which only depends on *x*, and one which only depends on *y*. Consider a separable filter with impulse response *K[x,y]*, which can be expressed as:

$$K[x, y] = F[x] \cdot G[y] \tag{3.1}$$

Then, the 2D convolution of image *I[x,y]* with *K[x,y]* can be written as:

$$I[x, y] \otimes K[x, y] = (I[x, y] \otimes F[x]) \otimes G[y] \tag{3.2}$$

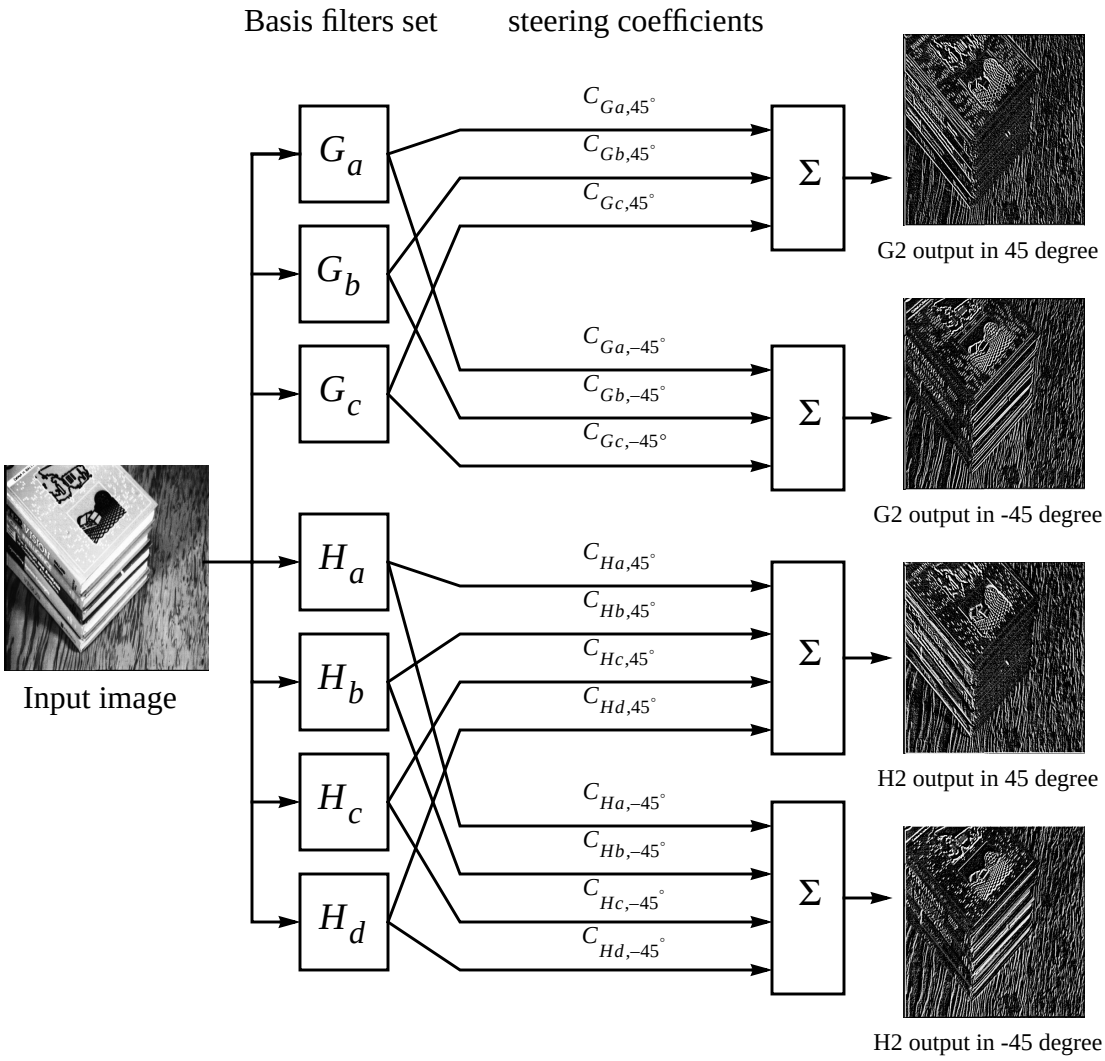


Figure 3.5: Architecture of the G2/H2 filter block. Any orientation of G2/H2 filters can be calculated by a linear combinati of a set of seven basis filters.

or

$$I[x, y] \otimes K[x, y] = (I[x, y] \otimes G[y]) \otimes F[x] \quad \textbf{(3.3)}$$

This property allows efficient implementation of G2/H2 filters because the convolution of the input image with an $N \times N$, X-Y separable kernel can be replaced with two separate 1-D convolutions with a horizontal $1 \times N$ vector and a vertical $N \times 1$ vector. This feature reduces the filter complexity from $O(N^2)$ to $O(2N)$. We call the convolution with $1 \times N$ vector the X-filter stage and the convolution with $N \times 1$ vector the Y-filter stage.

Hardware implementation of FIR filters requires two steps: 1) creating delay buffers, 2) multiplying with constant coefficients and building an adder tree. The X-filter and Y-filter are both FIR, and, thus have similar architectures in hardware. Their only difference is that since the video input data is arriving in rows, implementation of the X-filter delay buffers is simpler and smaller in comparison with the Y-filter delay buffers. An *N*-tap FIR X-filter requires only $N-1$ delay elements, while a Y-filter with the same size requires roughly $W \cdot (N-1)$ delay elements, where W is the number of pixels in one image scan line.

The design shown in Figure 3.5 includes seven pairs of X-filters and Y-filters, all of them processing the same input image. In this figure, all the basis filters have the same input. This property suggests sharing of delay buffers among all the basis filters. As Eq. (3.2) and Eq. (3.3) show, we can perform X-filtering and Y-filtering in arbitrary order without affecting the final convolution results. We have chosen to perform Y-filtering first (as in Eq. (3.3)).

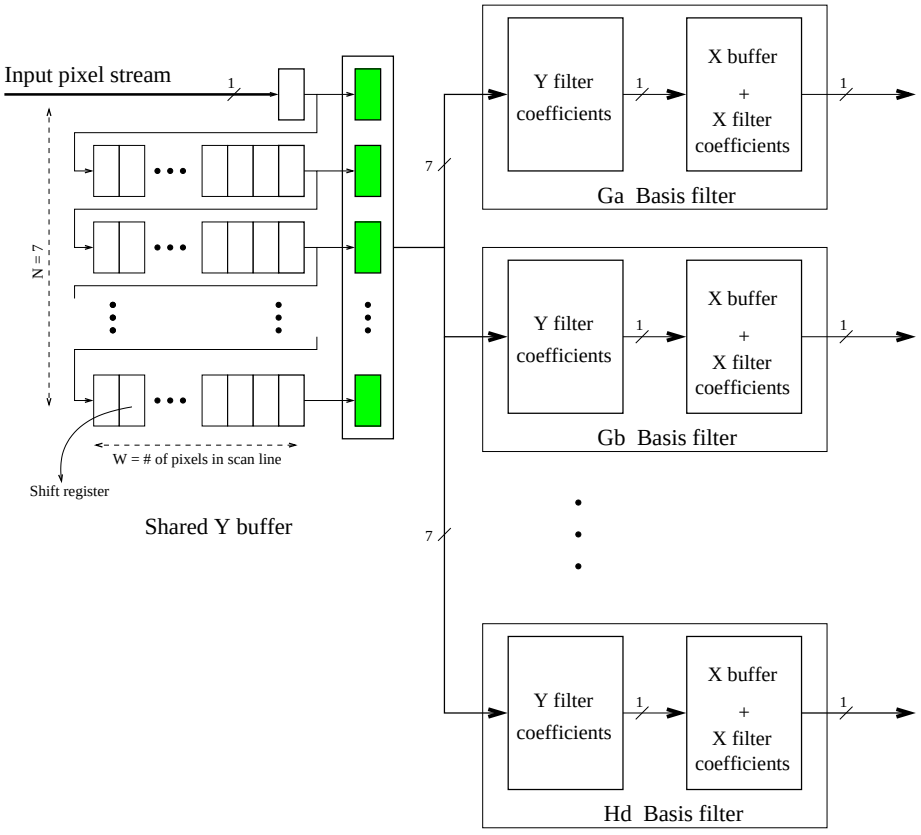


Figure 3.6: Sharing Y buffer for all seven Y filters of the basis filters. The Y buffers are by far larger than X buffers. So, sharing the Y buffer among all the basis filters saves more hardware as opposed to sharing X buffers.

By implementing the Y-filters first, the vertical buffer for Y-filtering can be shared between the Y-filters of all basis filters. Since the Y buffer is much larger than the X buffer, the amount of hardware saving as a result of Y-buffer sharing is much higher in comparison with X-buffer sharing. Figure 3.6 shows the internal architecture of basis filters.

In a hardware realization of an algorithm, floating point multiplication and division is usually expensive in terms of the amount of logic resources available on the chip. One common solution is to replace floating-point operations with fixed-point operations of an appropriate width. The width should be large enough to introduce acceptable quantization error according to the constraints of the algorithm. In Figure 3.6, input pixel values are in 8-bit grey scale, so all the shift registers in the delay buffers are 8 bits wide. The X-filter and Y-filter coefficients are quantized to 8-bit signed precision. The final outputs of oriented filters are presented in signed 16-bit values. A full detail analysis of the fixed-point representation for each stage of the design will be given in Section 3.7.

Besides different structures for delay buffers, the rest of the architecture of X-filter and Y-filter is similar. Figure 3.7(a) shows the design of an FIR 7-tap X-filter with coefficients C_1, C_2, ..., C_7. One important feature of X and Y decomposition of G2/H2 filters is that the coefficients are either symmetric or anti-symmetric, e.g. for a 7-tap FIR filter: $C_1 = \pm C_7$, $C_2 = \pm C_6$ and $C_3 = \pm C_5$. This property is used in the design to reduce the number of constant multiplications

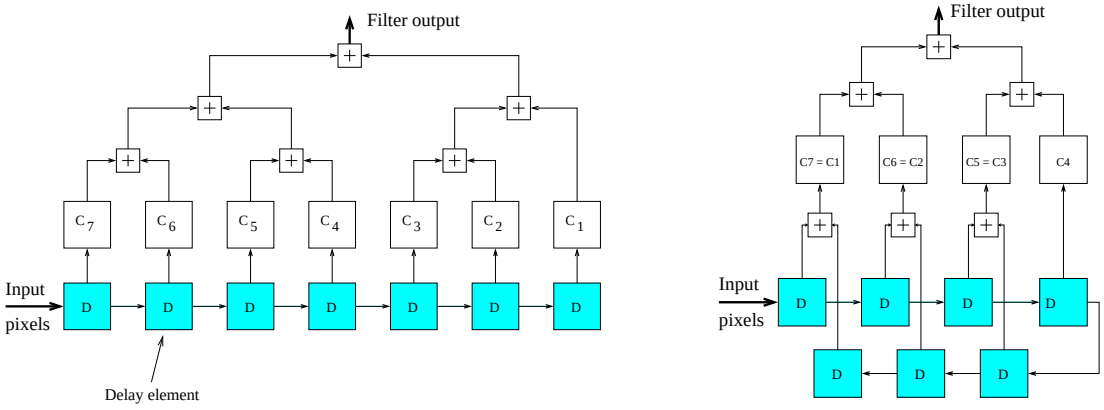


Figure 3.7: (a) Original architecture of the horizontal 7-tap FIR filter. (b) Revised architecture with symmetric coefficients. (For anti-symmetric coefficients the first level of adders can be replaced with subtractors.)

from 7 to 4, as shown in Figure 3.7(b). In this architecture, the pairs of input samples are added or subtracted before being multiplied by the coefficients. Note that the total number of adders in both designs (a) and (b) is identical.

The Scale/Orientation Decomposition Unit produces two complex valued band-pass outputs (in directions $45°$ and $-45°$) for each scale of the pyramid. Since the pyramid has three scales, there are totally 6 complex-valued outputs from this unit. Note that there are two parallel streams of signals: one for left image stream and one for right stream. So, each scale and each orientation has a pair of left and right signals that should be sent to the Phase-Correlation Unit. The complex valued signals consist of real and imaginary parts, each represented by 16-bit signed values.

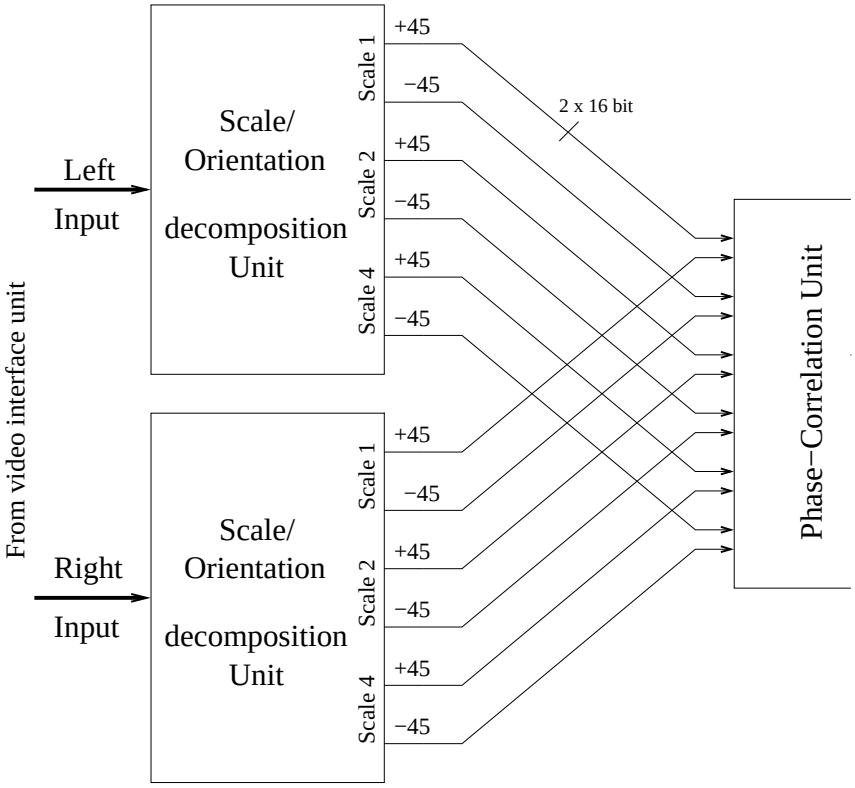


Figure 3.8: Communication between Scale/Orientation Unit and Correlation Unit

3.4 Phase-Correlation Unit

This unit is the heart of the vision system where the filter outputs of left and right images are

merged. As discussed in Chapter 2, the final goal of the stereo matching is to find two corresponding points in the stereo images. In general, finding the corresponding point requires search over the whole image, but by aligning the cameras and using the epipolar constraint, we can shrink the search window to a 1D horizontal window. This means that for each pixel in one image, the corresponding pixel in the other image lies on the same scan line and within a maximum distance.

To find the best match, we compute a similarity function for each pixel in the left image and the horizontally shifted locations of that pixel in the right image. The similarity function results are then combined across all scales and orientations. The shift value which produces the highest similarity will be detected as the best match.

Figure 3.9 shows the high level architecture of the correlation unit for one pair of left and right images. The value of D in Figure 3.9 represents the maximum allowed disparity between stereo images, or the size of the searching window. In our work, we have limited the value of D to 20 pixels in the finest scale, based on the hardware resources available. In the coarser

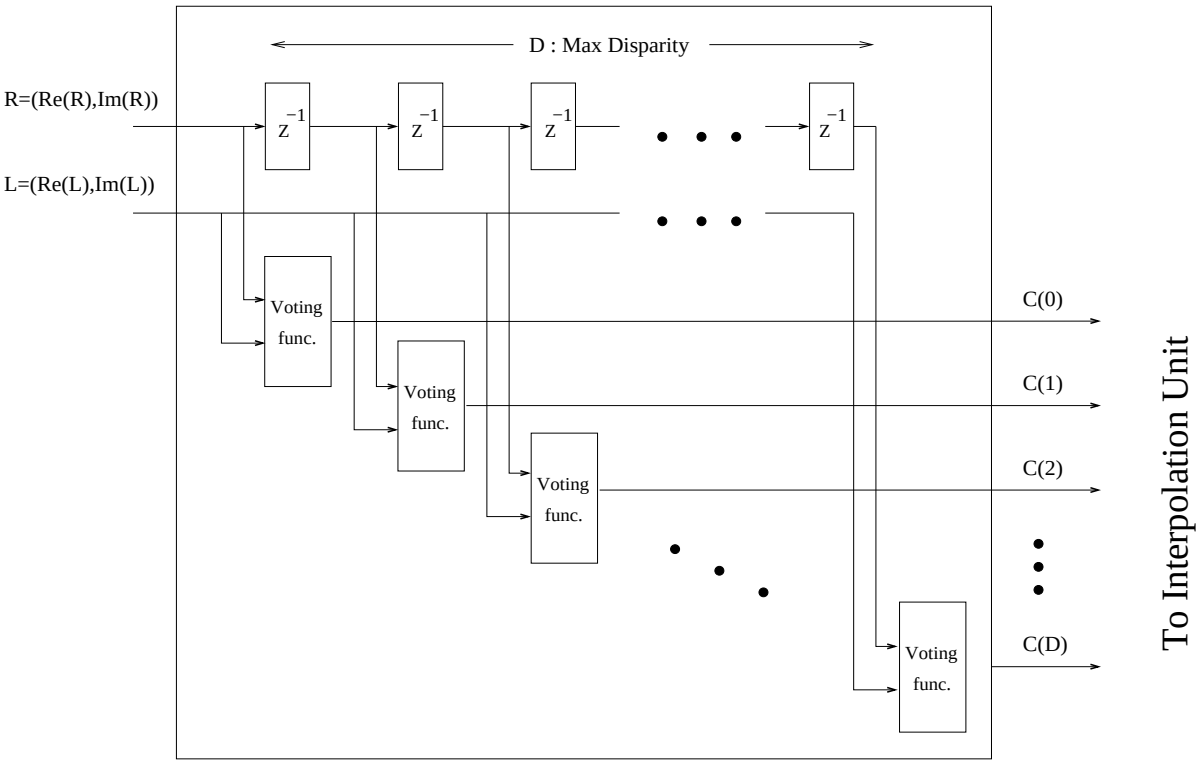


Figure 3.9: High level architecture of Phase-Correlation Unit for one pair of left and right images. In this architecture, all the voting function blocks are operating in parallel.

scales, the number of shifts are also scaled by the same factor such that in scale 2, there are 10 shifts and in scale 4, there are 5 shifts. Choosing larger values for maximum allowed disparity, D, will result in a larger search window which proportionally decreases the minimum detectable distance from objects to the camera head. On the other hand, increasing the value of D will increase the size of Phase-Correlation Unit and Interpolation Unit almost proportionally.

The similarity function, or voting function, implemented in this work, is based on the voting function proposed in LWPC algorithm:

$$C(x,\tau) = \frac{W(x) \otimes [O_l(x)O_r{}^*(x+\tau)]}{\sqrt{W(x) \otimes |O_l(x)|^2}\sqrt{W(x) \otimes |O_r(x)|^2}} \quad \textbf{(3.4)}$$

Based on this equation, to compute voting function C in location x of the image and for candidate disparity of τ, we need to convolve the Gaussian window, $W(x)$, with the inner product of $O_l(x)$ and $O_r{}^*(x+\tau)$, where $O_l(x)$ is the complex-valued G2/H2 filter output for left image and $O_r{}^*(x+\tau)$ is the conjugate of right image G2/H2 filter output shifted by τ pixels horizontally. The result is then divided by square root of convolution of $W(x)$ with the square of the amplitude of both $O_l(x)$ and $O_r(x)$.

Efficient calculation of the voting function, $C(x,\tau)$, in hardware is critical to the development of the whole system because of two facts: 1) this block contains several non-constant multiplications, square roots and dividers which are all expensive in terms of logic resources available on FPGA. 2) Several identical voting function blocks should be implemented in parallel. To have a rough estimate of the total number of this block, we have 20, 10 and 5 voting functions in scale 1, 2 and 4 respectively, which adds to a total of 35 blocks per orientation. Since this whole process is done on two separate orientations, in total there are 70 blocks of voting function. So, any saving in the implementation of this block will be magnified 70 times. In the next paragraphs, we will describe the techniques we have applied to shrink the size of voting function block, while introducing minimal error compared with original voting function.

As discussed in Chapter 2, at true disparity, the real part of $C(x,\tau)$ should be maximum and its imaginary part should be close to zero. It means ideally all the true matches should be detect-

able just by finding the τ at which $Re[C(x,\tau)]$ is maximum, although computing both real and imaginary parts of $C(x,\tau)$ will help to reject wrong matches. Based on this property, we only compute the real part of $C(x,\tau)$ in Eq. (3.4) in our system. In this equation, the Gaussian window, *w,* and denominator are always real-valued. So, to compute $Re[C(x,\tau)]$, we just compute $Re[O_l(x)O_r{}^*(x+\tau)]$, as follows:

$$Re[O_l(x)O_r{}^*(x+\tau)] = Re[O_l(x)]Re[O_r(x+\tau)] - Im[O_l(x)]Im[O_r(x+\tau)] \quad \textbf{(3.5)}$$

Implementation of Eq. (3.5) requires only two real multipliers, as opposed to four multipliers for full complex-valued multiplication to compute both real and imaginary parts of the voting function.

Figure 3.10 shows how $Re[C(x,\tau)]$ can be derived from $O_l(x)$ and $O_r(x)$ with basic mathematical operations. It requires seven multipliers, one square root block, one divider and three adders plus three parallel blocks of Gaussian window. Here is a rough estimate of number of LUTs required to implement the block illustrated in Figure 3.10 Assume that all multipliers are 8x8 bit and square root and divider blocks need the same number of LUTs as multipliers. Implementation of each multiplier requires around 60 LUTs and each adder requires 8 LUTs. So, architecture of Figure 3.10 needs 566 LUTs for basic mathematical operations. Consider-

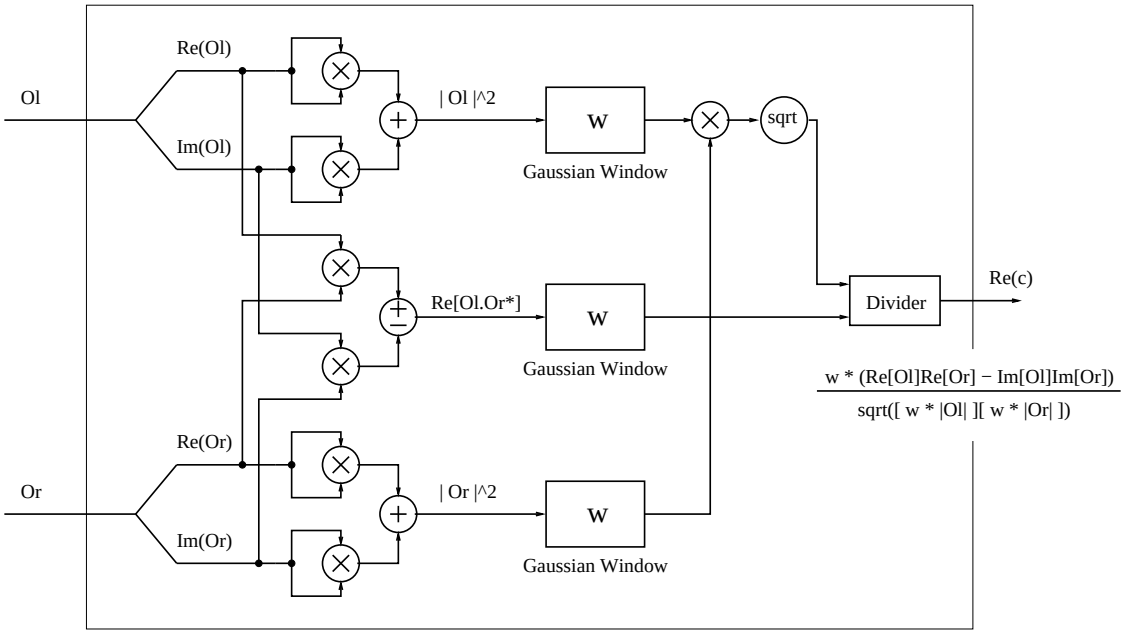


Figure 3.10: Realization of the real part of voting function in (3.4)

ing the fact that this bock needs to be replicated 70 times, the total number of LUTs would be 39,620 which is more than the 38,000 LUTs totally available in the Virtex2000E chip. Note that we have not taken into account implementation of any Gaussian Windows, delay elements or control logics yet. The 8-bit square-roots and dividers are also more costly than multipliers, making this problem even worse.

3.4.1 Location of Gaussian Window

To shrink the size of this block, we have slightly modified the method of calculating $Re[C(x,\tau)]$, as shown in Figure 3.11. In this revised architecture, we have pushed the Gaussian windows to the end of the block after divider. In fact, instead of applying the localized Gaussian window, *W(x)*, to the amplitudes and inner product of left and right outputs, we apply the Gaussian window to the result of the divider. While this is not the mathematical equivalent of the original method, we expect it will have similar smoothing effect because the *W(x)* window in our system is a small 3-tap FIR filter and the left and right outputs are band-limited (and hence their values do not vary dramatically in the window). This change allows us to extract the common portion of computations between several blocks such that we only perform them once. In fact, the architecture shown in Figure 3.11 is close to the local phase difference

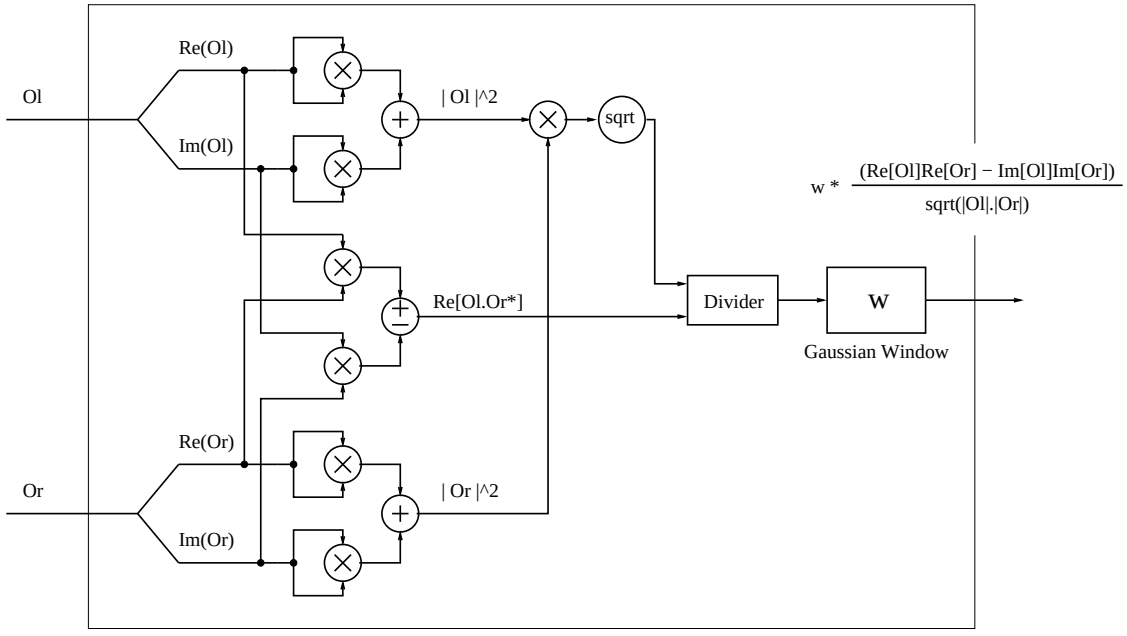


Figure 3.11: Revised voting function architecture: The Gaussian window is moved to the end of block

method proposed in [10].

Here are two major techniques we use to extract the common factors in order to reduce the size of the block:
First, by moving the Gaussian windows to the end of the block, the three Gaussian windows are reduced to one window. In fact, this Gaussian window can be pushed even further back to after addition of the voting functions of different orientations because of linearity of the Gaussian windowing operation. This will reduce the total number of Gaussian windows from 210 to 35.

Second, the sub-block before Gaussian window of Figure 3.11 can be re-arranged as shown in Figure 3.12. In this new design, each of the two inputs, $O_l(x)$ and $O_r(x)$, are normalized first such that they both have unity length. The inner product of these unity length vectors is then calculated and sent to Gaussian window. At first, having two normalizing blocks as in Figure 3.12 might seem not to be helpful because it takes 3 dividers and one square root block more than its original version. But it has a main advantage: These normalization blocks are identical

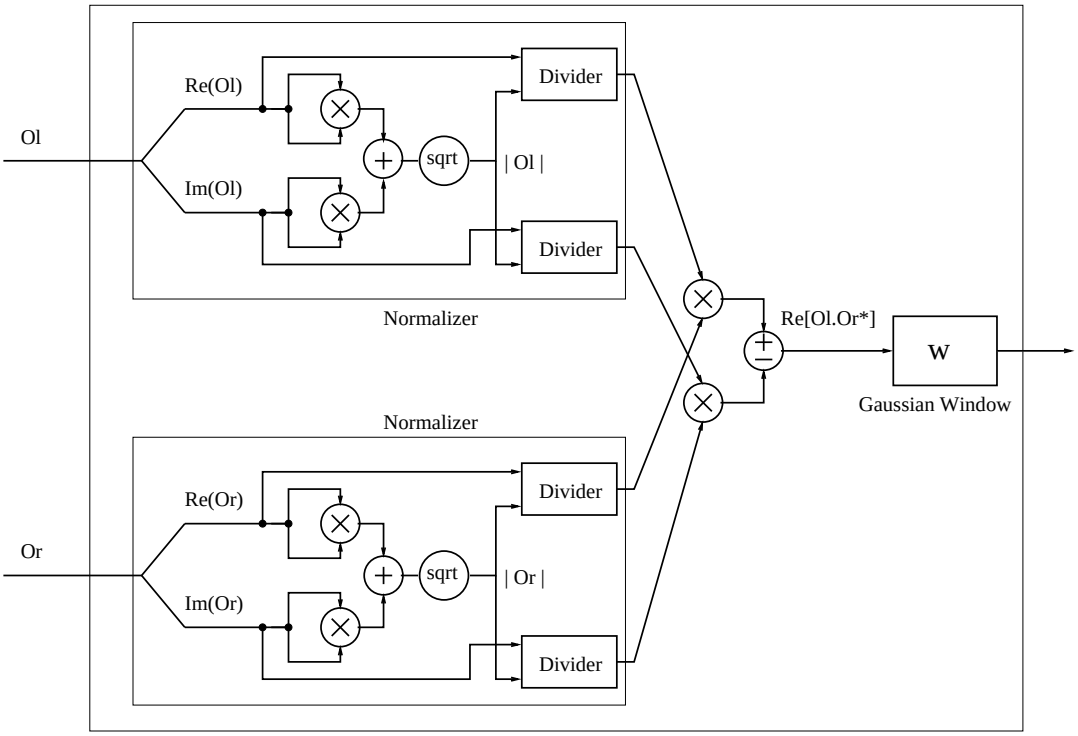


Figure 3.12: Re-arrangement of the architecture of Figure 3.11 with two normalizer blocks, one inner product and one Gaussian window unit.

for all the shifted versions of the image (Figure 3.9). It means instead of having normalization units in every voting function block, we can share them across all the shifted versions of voting functions. This sharing technique is illustrated in Figure 3.13. Extracting the normalization block from the voting function blocks reduces the computational complexity inside each block. Since all disparities in each scale and orientation need only one pair of normalization blocks, the total number of normalization blocks is 12, as opposed to 140 normalization blocks needed before adopting sharing technique.

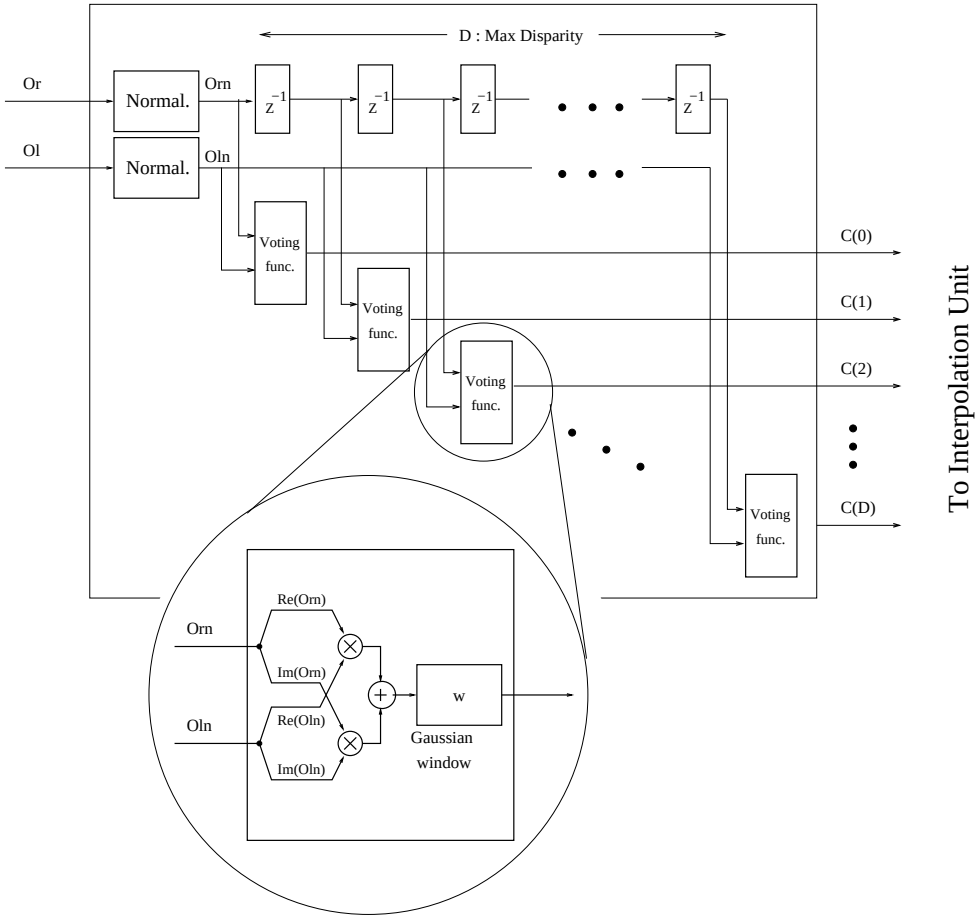


Figure 3.13: Extracting the normalizer blocks and sharing them across all voting functions reduces the complexity inside each voting function block.

3.4.2 Normalization

For further reduction in the size of the Phase-Correlation Unit, we have modified the architecture of normalization blocks. The major portion of the normalization block is computing the

amplitude of complex-valued inputs. Amplitude of a complex value, A, can be expressed as the $L_2 - norm$ of the 2D vector A, with $Re(A)$ and $Im(A)$ as its elements:

$$L_2 - norm: \qquad \|A\|_2 = \sqrt{Re(A)^2 + Im(A)^2} \tag{3.6}$$

Hardware implementation of $\|A\|_2$ is expensive because it requires two multipliers, one square root and one adder. Instead, we have replaced $\|A\|_2$ with $L_1 - norm$ of vector A, $\|A\|_1$, defined as:

$$L_1 - norm: \qquad \|A\|_1 = |Re(A)| + |Im(A)| \tag{3.7}$$

where $|x|$ means absolute value of x. Figure 3.14 shows the effect of replacing $\|A\|_2$ with $\|A\|_1$ on the normalized output vectors. In the $L_2 - norm$ method, all the normalized vectors are located on the unit circle in the Real-Imaginary plane but in $L_1 - norm$, they are projected on a square, as shown in Figure 3.14. This is because the sum of absolute real and imaginary parts of normalized vectors are always unity and therefore they all become projected to straight lines which form a square instead of a unit circle. This technique provides enough accuracy for our application and can also be used in similar applications. To improve the accuracy and still avoiding implementation of $\|A\|_2$, there is one potential solution: Since $Re(A_{u1})$ and $Im(A_{u1})$ in Figure 3.14 are always limited between -1 and 1, one can use a memory block as a look up table with appropriate size to replace current A_{u1} vectors with those closer to values on the

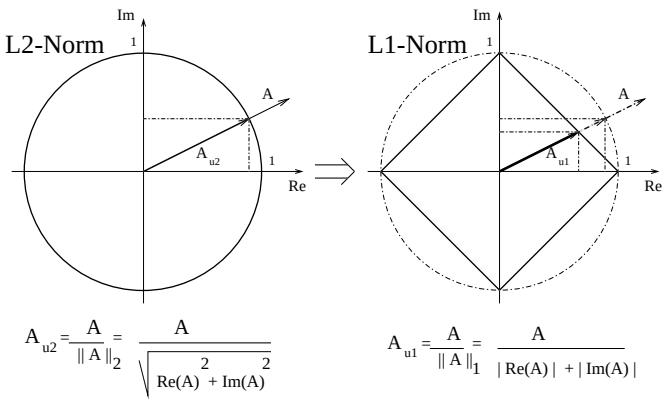
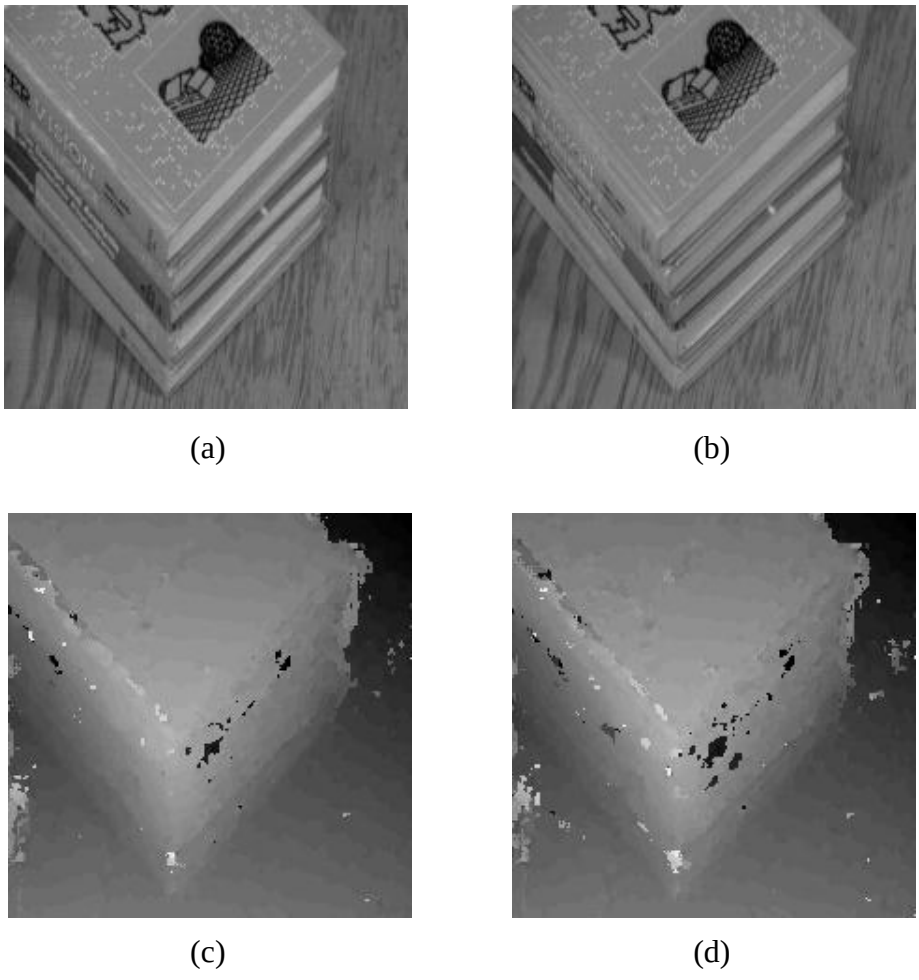


Figure 3.14: Effect of using $L_1 - norm$ instead of $L_2 - norm$

unit circle. The important point is that these look up tables can be built using on-chip memory which is available in most of the current FPGA devices without the need to use logic elements of the device.

Figure 3.15 shows the effect of the first two major modifications we have made to the original LWPC algorithm. Figure 3.15(a) and (b) are a 'books' stereo images. In (c), the depth result from original algorithm is shown. In (d), the depth map after changing the location of Gaussian window and replacing $L_2 - norm$ with $L_1 - norm$ is shown. In most of the regions, the two images have the same depth values, but as we were expecting, the depth map in (d) contains



(a) (b)

(c) (d)

Figure 3.15: Effect of using $L_1 - norm$ instead of $L_2 - norm$ and changing the location of Gaussian window on the final depth map results of the 'books' stereo images. (a) Left image. (b) Right image. (c) Depth map by using $L_2 - norm$ and original location for Gaussian window. (d) Depth map by using $L_1 - norm$ and Gaussian window moved to the end of voting function block.

slightly more noise compared with (c). This noise comes from the approximations we have made in order to make the hard implementation efficient and feasible.

Table 1 summarizes the number of mathematical operations required in the different architectures of the phase-correlation block. In this table Gaussian windows are considered as a building blocks without counting the components inside the Gaussian window. In fact Gaussian windows in our system are 3x3 X-Y separable low-pass FIR filters. So, implementing each Gaussian window requires a set of X-buffer and Y-buffer plus constant coefficients and adders tree. These resources are not included in table 1.

Architecture	Original from LWPC algorithm	Gaussian windows moved	Gaussian windows moved + normalizer sharing	Sharing + L1 normalizer
Multipliers	490	490	188	140
Dividers	70	70	24	24
Square roots	70	70	12	-
Adders	210	210	82	82
Gaussian Windows	210	70	35	35

TABLE 1. Summary of the number of basic blocks for different architectures of Phase-Correlation Unit

3.5 Interpolation/Peak-Detection Unit

The Interpolation/Peak-Detection Unit is the last unit before Video Output Interface. This unit is implemented on the fourth FPGA available on the board and consists of two major blocks, the Interpolation Block, and the Peak Detection Block.

3.5.1 Interpolation Block

This block interpolates two coarser scale voting functions, $C_{(j,2)}(x,\tau)$ and $C_{(j,3)}(x,\tau)$, in both x and τ domains such that they can be combined with the finest scale voting function, $C_{(j,1)}(x,\tau)$. It performs quadrature interpolation in the τ domain and constant interpolation in x domain. The quadrature interpolation performed in this work is as follows: Assume a function $f(n)$, for $0 \le n \le N$. We build $g(m)$, $0 \le m \le 2N$, by quadrature interpolation of $f(n)$ based on this

formula:

$$g(m) = \begin{cases} f(n) & if \quad m = 2n \\ 0.375 f(n) + 0.75 f(n+1) - 0.125 f(n+2) & if \quad (m = 2n+1) \wedge (m \neq M-1) \\ (-0.125) f(N-2) + 0.75 f(N-1) + 0.375 f(N) & if \quad m = M-1 \end{cases} \quad \textbf{(3.8)}$$

The interpolated voting functions are then added together to produce the overall voting function $S(x, \tau)$. The interpolation from scale 2 to scale 1 is performed in one step, but interpolation from scale 4 to scale 1 is broken to two steps, each step up-sampling by a factor of two.

3.5.2 Peak-Detection Block

The input to this block is a three dimensional array with the first two dimensions the same size as the size of the image (256 x 360) and the third dimension of size D (maximum disparity). Each element $S(x, y, \tau)$ in this array is the combined voting function for location (x, y) of the image and for the candidate value of τ, as the disparity. The goal of the Peak Detection block is to find, for each pixel location (x, y), the τ value which maximizes $S(x, y, \tau)$. The index value of τ is sent to output as the estimated disparity for pixel (x, y) of the image.

Since the maximum disparity, D, in our system is set to 20 pixels, the disparity output can have integer values from 0 to 20. It means that τ has an accuracy of about 5 bits. This will be translated to 5-bit accuracy in the final results of depth measurements. To improve the overall accuracy of the system, we have employed a sub-pixel accuracy sub-block inside the Peak Detection Block which was not in the original LWPC method. The sub-pixel accuracy block picks the $S(x, y, \tau)$ that has the maximum value. In addition, it picks that point's left and right neighbors in the τ domain, e.g. $S(x, y, \tau - 1)$ and $S(x, y, \tau + 1)$ and fits these three points to a quadratic curve. The point corresponding to the peak of the quadratic curve is declared as the sub-pixel accuracy disparity. Assuming that $S(x, y, \tau) = a$, $S(x, y, \tau - 1) = b$ and $S(x, y, \tau + 1) = c$, we calculate the sub-pixel accuracy disparity, τ_{sub}, as the sum of an integer value, τ, and a fractional shift, Δ_τ, defined as:

$$\tau_{sub} = \tau + \Delta_\tau, \quad \Delta_\tau = \frac{c - b}{4a - b - c} \quad \textbf{(3.9)}$$

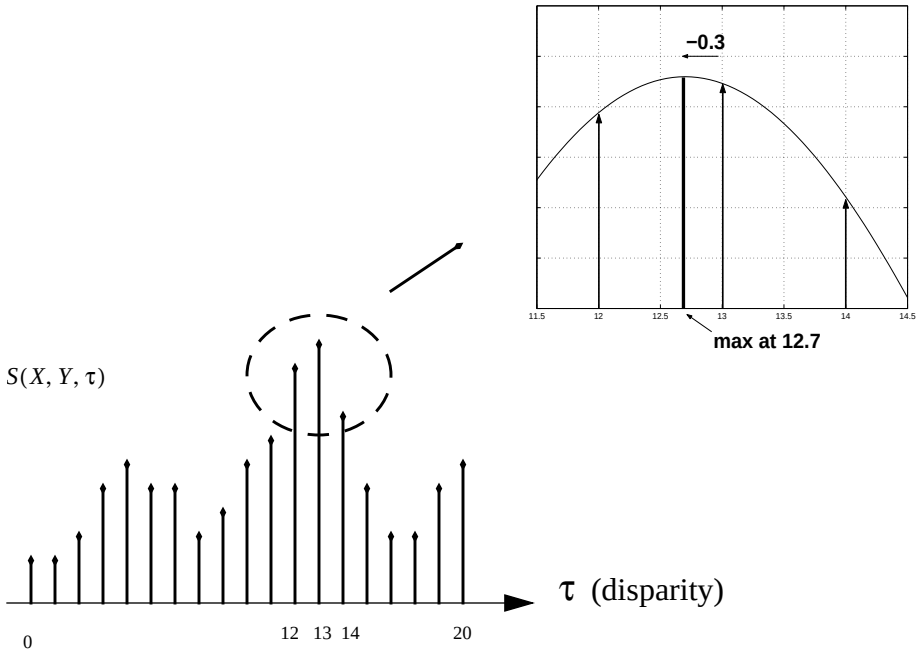


Figure 3.16: Sub-pixel accuracy in disparity is obtained by adding Δ_τ to the integer-valued τ.

The value of Δ_τ adds a fraction to original value of τ to move it towards the neighbor with value closer to the peak. We compute Δ_τ with 3 bit precision after the decimal point that after being added to the original 5-bit integer τ produces final τ_{sub} results with 8-bit accuracy. (Figure 3.16)

The final disparity results are written in the external SRAM bank which will be read by the Video Output Unit to be displayed on a monitor.

3.6 Video Output Unit

The Video Output Unit sends three sets of information to the display: 1) original input video, 2) depth map results; and 3) mouse pointers on both original and disparity map results.

The original input video and final depth map are always buffered in a 2MB external SRAM connected to FPGA #3 of the TM-3A board. The Video Output Unit reads these two buffers and sends them to the monitor such that the lower half of the display shows the original image and the upper half shows the result of stereo algorithm. In the depth map results, the depth is

encoded in grey scale values from 0 to 255: the higher grey scale value (brighter pixels) indicates a pixel close to cameras. Lower grey scale values indicate pixels farther from camera.

In addition to displaying the dense depth map, our system allows the user to check the distance of any arbitrary pixel in the image just by clicking the mouse pointer on that pixel. This feature is added by a two-way communication between the TM-3A board and the host machine: the user moves the mouse which is connected to the host machine. A program on the host machine continuously reads the mouse pointer coordinates and sends them to the TM-3A board through the ports package, which allows communication of host with the board. The Video Output Unit receives the mouse pointer coordinates and superimposes two small squares as pointers on the original and depth map images on the monitor which is directly connected to TM-3A board. When the user right clicks the mouse, the TM-3A is informed through the ports package. The TM-3A is then responsible to read the depth map buffer and send the depth value of that pixel back to the host machine. After the click, a window on the host machine displays the X and Y coordinates, plus the distance of the pixel from the camera in centimeters.

3.7 Fixed-Point Representation Analysis

Floating-point mathematical operations require extensive resources to implement in hardware. Since we know the required scale and precision in each stage of the stereo algorithm, we can implement it in a far more efficient fixed-point representation. However, there is always a trade-off between accuracy of fixed-point representation and the hardware cost: minimum quantization error requires using wide fixed-point representations, but wider signals require larger circuits to perform mathematical operations (dividers, multipliers, adders, etc.), in addition to larger data path circuits, larger memory and higher chip-to-chip communication bandwidth. In this section, we analyze the effect of replacing original floating-point operations with fixed-point representations of different size.

Figure 3.17 shows the cost of implementing parallel (one clock cycle) multipliers and dividers versus the input width of the multiplier or divider. If the required throughput permits one can use serial multipliers and dividers that cost less but generate only one output every N clock cycles (N>1). But regardless of using parallel or serial operators, the number of LUTs to build

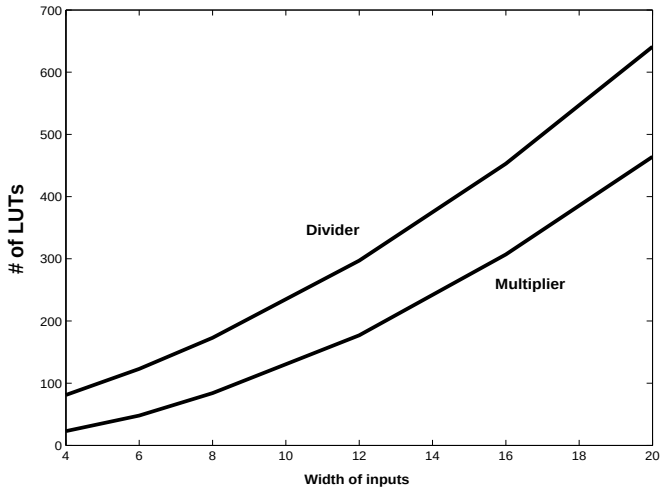


Figure 3.17: Divider and Multiplier Hardware Cost

a multiplier or divider increases approximately with the square of the input width.

The key issue is that the assignment of fixed-point width to signals should be analyzed for every block of the algorithm separately. This is because fixed-point precision in each stage of the system has a different effect on the final results. So, we need to make good decisions for the precision of the variables and operations in each stage of the algorithm. This analysis requires both knowledge of the target hardware and the algorithm itself. Few tools have been developed to solve this problem. For example, [7] is a framework for automatically determining fixed-point precision of floating-point calculations.

In the design process of our system, we have done a detailed analysis of the effect of fixed-point width of signals on the final results in order to find the proper width assignment. Before the detailed discussion of these effects, there are several issues that should be mentioned regarding the selection of proper width for the fixed-point representations:

- As we discussed in Chapter 2, there is a limited amount of hardware resources in TM-3a, including number of LUTs and flip flops in each FPGA, on-chip and off-chip memory, memory access bandwidth and chip-to-chip communication bandwidth. This fact sets an upper limit on the fixed-point precision of signals. So, as the first rule, we need to make sure that the selected fixed-point widths meet the hardware requirements.

- Usually the final accuracy of the system is more sensitive to the precision of some signals as opposed to others. For efficient usage of the hardware resources, we try to assign wider bit representation to those signals that affect the final results more.
- Decisions should be based on the relative size and number of the blocks: those blocks that are replicated so many times have larger effect on the total size of the circuit. So, when minimizing the size of blocks, the priority is for bigger and more frequent blocks.
- This problem is in general an optimization problem with complicated constraints and cost functions. While we have tried to assign appropriate representations to signals, we can not guarantee that this is the optimum setting.

We have conducted a set of emulations in order to find the approximate sensitivity of the overall performance to the precision of different signals. The task of assigning proper precisions to all signals is theoretically a multi-variable optimization problem with a complex cost function. In this research we are not claiming we have reached the globally optimum settings but our solution is efficient enough to meet our design constraints.

In this section we show the results of analysis of the precision in four major points of the system: 1) X-filters outputs, 2) Y-Filters outputs, 3) normalizer block outputs, and 4) voting function outputs. Our methodology is as follows: We start from the first point, X-filters outputs, sweep its precision over a wide range, assuming that all the other points in the system are in full precision. For each precision of the X-filter output, we calculate the quantization error by comparing the final disparity result of limited X-filter output precision with final disparity results of full precision X-filter outputs. After creating the graph of error versus precision, we choose the proper precision before the graph becomes flat (assuming this precision is less than the upper limit imposed by hardware resources). The reason for this choice is that when we reach the flat part of the graph, it means that we are increasing the precision but it does not improve the quality of final results.

After choosing the X-filter outputs precision, we keep this precision fixed and sweep the Y-filter outputs, still assuming that next points are in full precision. We continue this method until the last point, where all the precisions are fixed except the last one.

Figures 3.18, 3.19, 3.20, 3.21 show the effect of precisions in X-filters outputs, Y-filters outputs, normalization block outputs and voting function outputs on the final disparity results respectively. In these figures, the horizontal axis is the width at a specific point in the system and the vertical axis is the mean square error in the final disparity outputs as a result of replacing full precision signals with fixed-point representation. For all the three test images in these figures (Books, Tree, Lamp), the maximum disparity is 20 pixels.

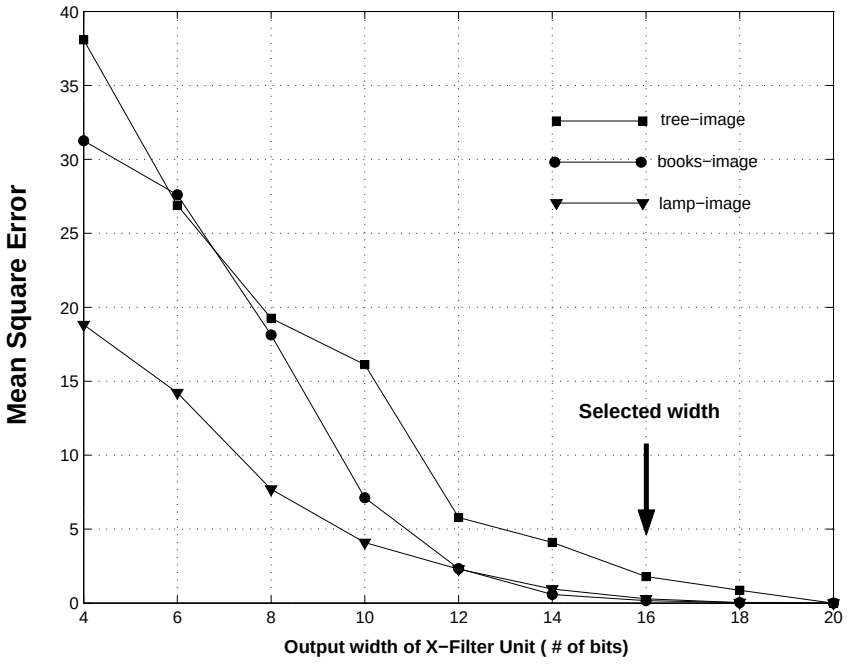


Figure 3.18: Fixed-point analysis for the X-filter block output width. Based on this analysis, we represent X-filter outputs with 16-bit signed values.

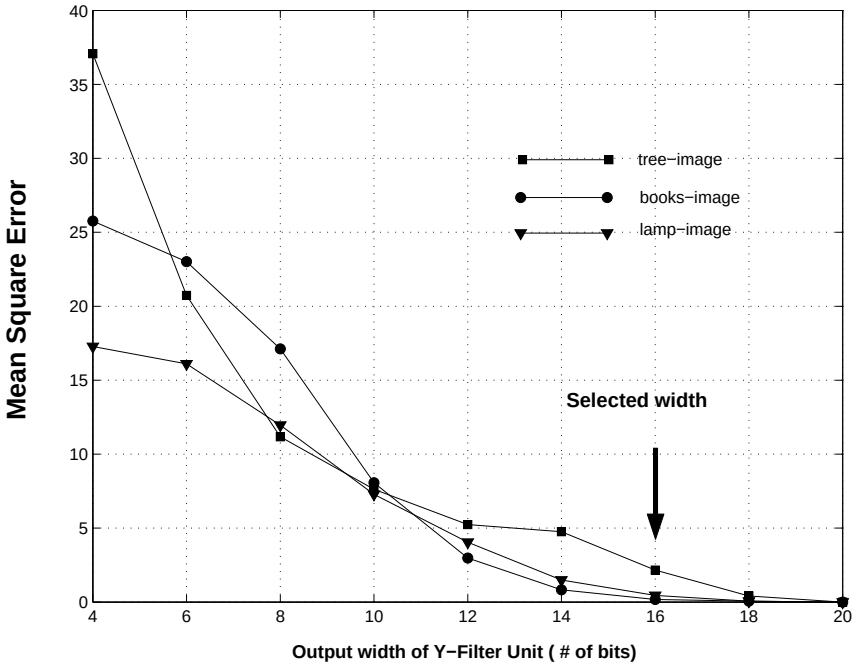


Figure 3.19: Fixed-point analysis for the Y-filter block output width. Based on this analysis, we represent Y-filter outputs with 16-bit signed values.

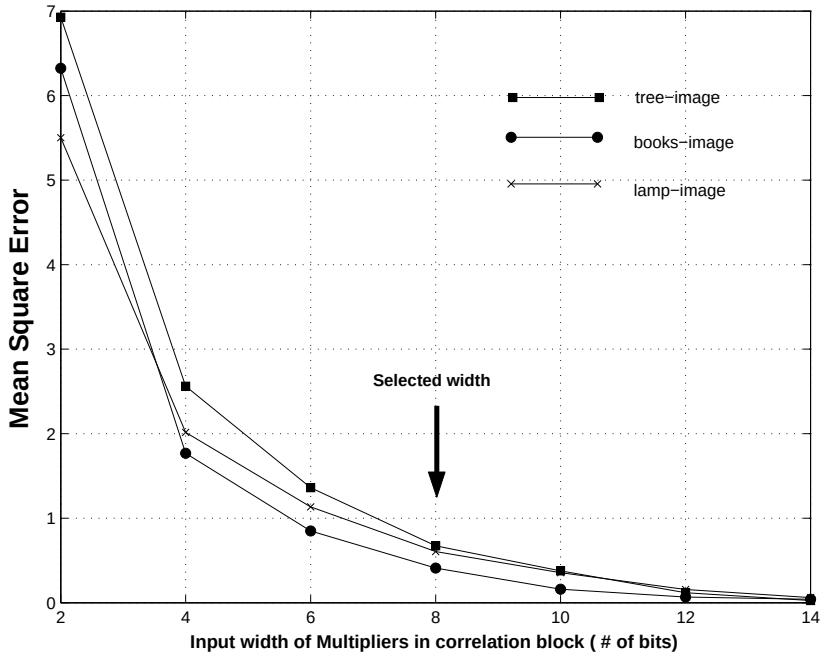


Figure 3.20: Fixed-point analysis for the output width of the Normalizer Block (inside Phase-Correlation Unit). Based on this analysis, we represent Normalized complex values with 8-bit signed values.

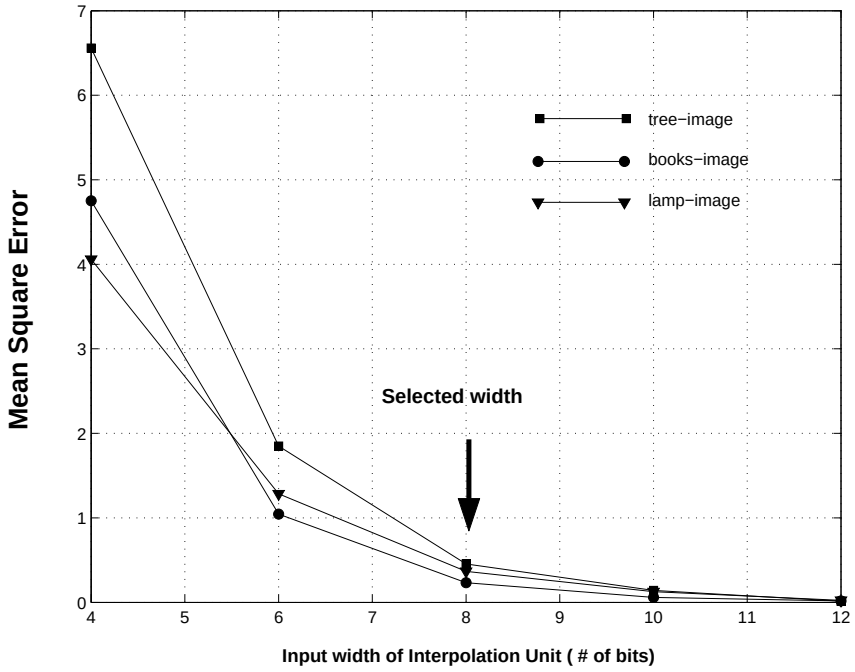


Figure 3.21: Fixed-point analysis for the input width of the Interpolation Unit (output of Phase-Correlation Unit). Based on this analysis, we represent Interpolation Unit inputs with 8-bit signed values.

3.8 Chip-to-Chip Communication

As we discussed in previous section, a key issue in hardware design is chip-to-chip communication. Each FPGA in TM-3A is connected to the other three FPGAs with a 98-bit bus. In our design, we have managed to transfer all the data between the chips using less than 98bits. For example, from Scale/Orientation Unit to Phase-Correlation Unit we needed to transfer 24,16-bit signals arriving at a rate of 8 Mega sample/sec. To fit this wide signals all in 98-bit bus, we used a simple Time Division Multiplexing (TDM) technique. In this technique, we use one 16-bit 48 Mhz channel to transfer a group of six parallel 16-bit 8 Msample/sec signals (Figure 3.22).

Figure 3.23 shows the flow of data through the FPGAs and the number of bits used to commu-

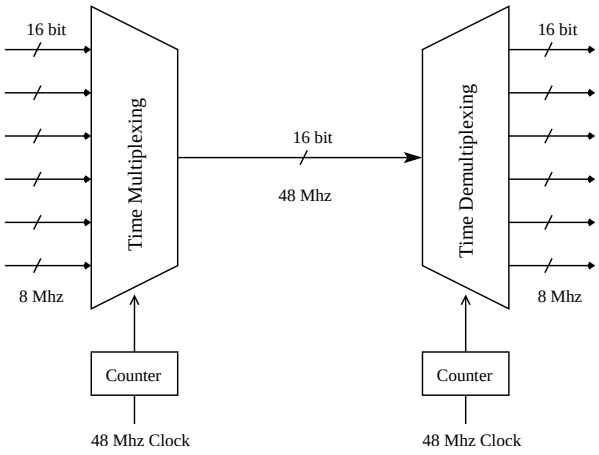


Figure 3.22: Time Multiplexing/Demultiplexing in chip-to-chip communication

nicate between each pair of FPGAs. The major portion of data in this system flows clockwise, starting from FPGA #3, Video Interface Unit, and returning back again to this unit. The data transfer between FPGA #2 and FPGA #0 does not follow the global circular flow of data. It is mainly to balance the processing load among all the chips. In other words, the Scale/Orientation Decomposition Units is 'borrowing' a portion of LUTs and flip fops from FPGA #0, which is designated to Interpolation/Peak-detection Unit.

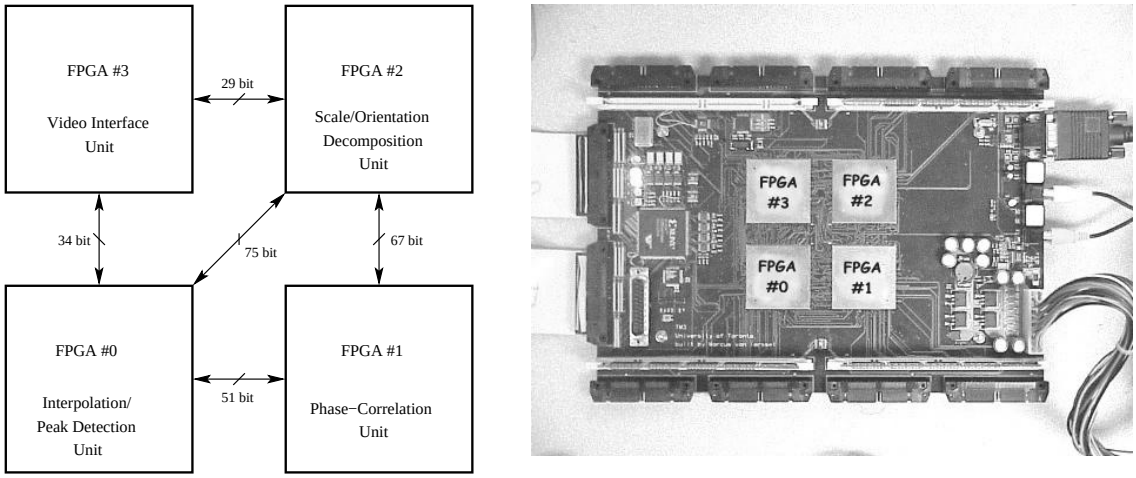


Figure 3.23: Data transfer between four FPGAs

3.9 Summary

Table 2 lists the hardware resources used in each unit of the stereo system in terms of number of Look Up Tables (LUTs), flip flops, slices and the amount of on-chip fast memory. Each slice in Xilinx Virtex 2000E contains two LUTs and two flip flops [33]. Since Virtex 2000E FPGA contains 38,000 LUT-flip flop pairs, it has 19,200 slices. Table 2 indicates that although in all of the four chips, except the video interface chip, almost all the slices are used, the percentage of used LUTs or flip flops is at most 88%. This means some slices are used partially and still have some LUTs of flip flops available. While we could still use more LUTs and flip flops, it is usually with the cost of not meeting timing constraints of the design such that the maximum frequency in which the circuit can perform correctly drops to less than 50 Mhz, which is our target clock frequency.

Unit Name	# of 4-input LUTs	% of 4-input LUTs	# of flip-flops	% of flip-flops	# of slices	% of total slices	On-chip memory (bits)	% of total memory
Video Interface (FPGA #3)	169	1%	71	1%	105	1%	-	-
Scale/Orient. Decomp. (FPGA #2)	23,151	60%	18,020	46%	19,198	99%	614,400	93%
Phase-correlation (FPGA #1)	16,709	43%	30,961	80%	19,198	99%	-	-
Interpolation (FPGA #0)	26,615	69%	33,974	88%	18,048	94%	172,032	26%

Table 2: Hardware resources for each unit

The stereo system is functioning at the clock frequency of 50 Mhz and produces 256 x 360 pixels 8-bit, sub-pixel accuracy depth map at 30 frames per second. In Chapter 4, we present the results of the implemented system. It includes the functionality of the stereo system as well as the depth measurement results.

CHAPTER 4

Implementation Results

4.1 Functionality and performance

The FPGA stereo system developed in this research performs multi-resolution, multi-orientation depth extraction based on the LWPC algorithm. This system can produce a dense disparity map of size 256 x 360 pixels with 8-bit, sub-pixel accuracy disparity results at the rate of 30 frames/sec. One common comparison metric used to measure the throughput of stereo vision systems is the PDS (Points times Disparity per Second) measurement defined as:

$$PDS = \frac{n \times m \times D}{T} \tag{4.1}$$

where the image size is $n \times m$ pixels and D is the maximum disparity evaluated in a total time of T seconds. Using this metric, our system achieves a performance of 55 million PDS, which is among the highest rates reported. Table 3 compares the performance of our vision system

with several other high performance hardware stereo vision engines.

	$n \times m$	D	T (msec)	PDS (x 10^6)	algorithm	platform
INRIA[9]	256 x 256	32	280	7.5	Intensity Correlation	PeRLe-1 board (23 Xilinx XC3090 FPGAs)
PARTS engine[32]	240 x 320	24	23.8	77	Census	16 Xilinx 4025 FPGAs
CMU stereo machine [21]	200 x 200	30	33	36	sum of absolute difference	special purpose hardware (C40 DSP + real time processor)
This Work	256 x 360	20	33	55	LWPC (Phase based)	TM-3A board (four Xilinx Virtex 2000E FPGAs)

TABLE 3. Summary of reported stereo vision system performances

As this table shows, our stereo system ranks after PARTS engine in terms of PDS metric. While the PDS metric reflects the density and the speed of the system, it does not measure the accuracy of the implemented algorithm. The important feature of our system in comparison with other hardware stereo machines is its high accuracy phase-based algorithm. To realize a phase-based algorithm in video rate, the system performs the equivalent of more than 10 billion 16 x 16 bit multiplications per second and the four Virtex 2000E devices communicate at a data rate of up to 200 Mbytes/sec.

In comparison with software implementation, we have run the Matlab version of the stereo algorithm on a Sun UltraSPARC-III 750 MHz processor with 2.5 GB of memory. On this platform, producing the depth map for each pair of frames takes about 30 seconds which is about 900 times slower than our FPGA-based stereo system. Although Matlab implementation is not the fastest possible software solution, we expect that the software optimizations (implementing the algorithm with C/C++ and create a stand-alone application) will improve the software speed by a factor of less than 5 on the same hardware platform. This is because our Matlab implementation is heavily vectorized and is mainly using the built in Matlab functions which are designed to be fast and efficient. So, depending on the architecture and the maximum frequency of the processor that runs the software algorithm, the overall speed-up from software to reconfigurable hardware is approximately 60 to 900.

4.2 Depth Measurement Results

We have conducted two sets of tests to evaluate the performance of disparity matching and distance estimation. First, we test the system with the synthesized images and, second, with natural images received from the cameras. The advantage of testing with synthesized images is that since the input is not from a camera, any type of noise associated with the imaging system (e.g. noise in image sensors, camera alignment, brightness variations in left and right images, etc.) will not affect the stereo matching performance.

4.2.1 Synthesized Images

The first example in Figure 4.10 shows the depth map from a pair of random stereograms. Random stereograms are pairs of images with random grey scale values such that the pixels in one image are the same as the other image but shifted based on the intended distance. So, this pair of images underlies a synthesized 3-D surface. Figure 4.10(a) and (b), show a random stereogram with a depth ground truth as given in Figure 4.10(c) or a 3D surface as in Figure 4.10(d), which is called a “wedding cake” [15]. This surface has four levels of depth with 0, 4, 8 and 12 pixel disparities and is symmetrical in both horizontal and vertical directions from the view point of the right image. Since the ground truth in this figure is shown from the left image view point, all the objects are skewed to right. Figure 4.10(e) shows the depth map obtained from applying original software algorithm and Figure 4.10(f) shows the result obtained by our hardware stereo engine which is close to the software results. As this figure shows, the hardware system is capable of reconstructing smooth surfaces similar to the results of the software and ground truth depth. To compare the performance of software and hardware methods quantitatively, we have calculated the Mean Square Error (MSE) between software disparity results and the ground truth, and also between hardware disparity results and the ground truth. For example, in Figure 4.10, the MSE for software disparity compared with ground truth is 1.13 $pixel^2$, while the MSE for hardware is 1.53 $pixel^2$. In both cases, the maximum allowed disparity was limited to 20 pixels. As we were expecting, the hardware results have larger errors compared to software. This larger error is because of all the modifications we have done to the original software method (e.g. the fixed-point operations and the changes in the architecture as explained in Chapter 3) to build the system in the limited hardware

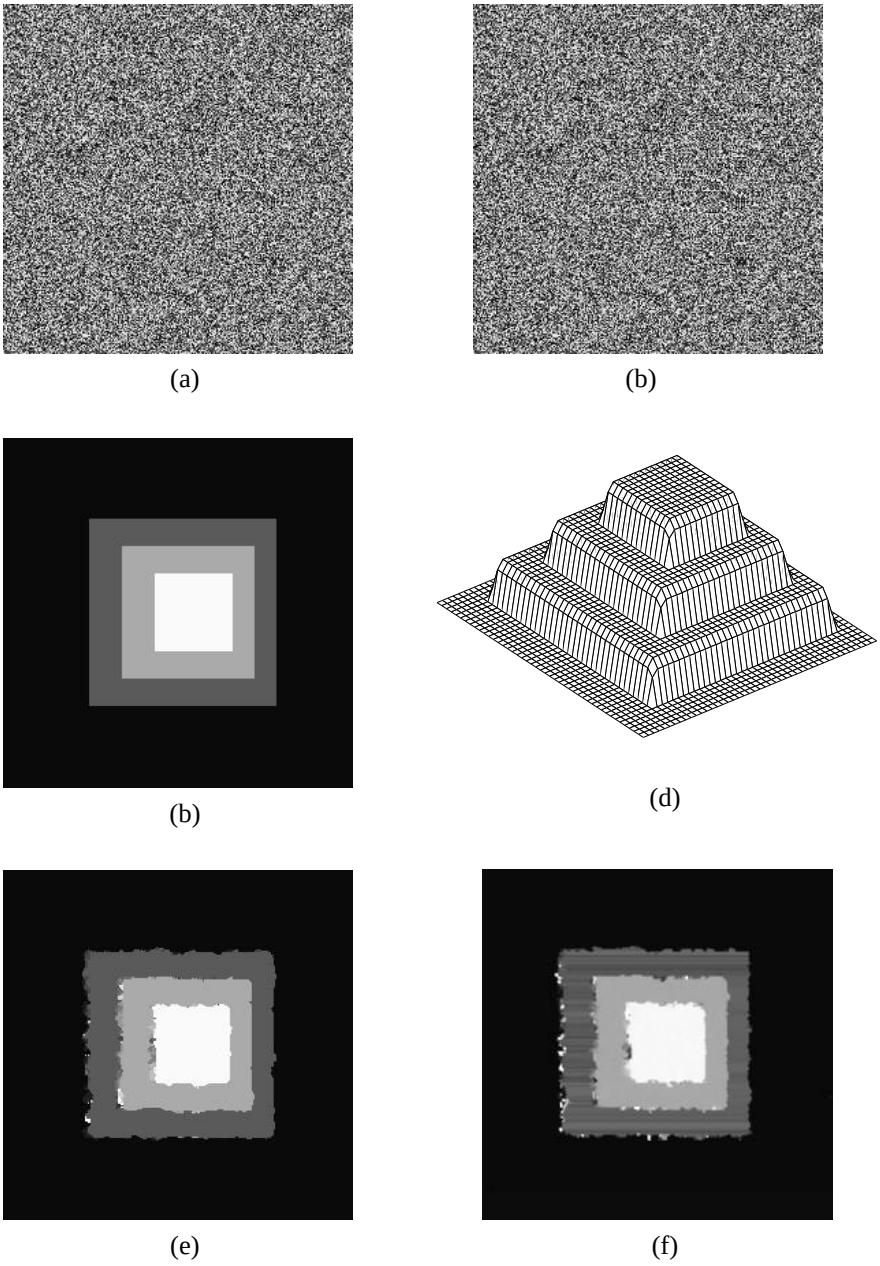


Figure 4.10: Overall system performance on Wedding-cake random stereograms: (a), (b) The stereogram (c) Ground truth depth (d) 3D structure of ground truth (e) depth from original software (f) depth from hardware vision system

resources available. This increase in error is of course rewarded by the speed up as the result of hardware implementation.

In this work we have used MSE to compare the performance of hardware and software methods. One important point is that while the MSE measure indicates the amount of error

between the stereo pre-shift results and the ground truth pre-shift values, it is not the best possible method to quantify this error. The reason is that MSE measure assumes a Gaussian distribution for the error but by looking at the histogram of the errors in pre-shift results, we can see that it is a combination of two distributions: The first is a very sharp Gaussian distribution around zero with a variance of less than one pixel. The second belongs to a relatively small number of pixels with errors uniformly distributed over all possible pre-shifts. So, in order to characterize the error in pre-shift values, a more accurate method would be to separate the two underlying distributions and describe them separately.

As we can see in Figure 4.10(e) and (f), at the edges of the surface, where there is a depth discontinuity, both software and hardware methods introduce noise. This is, in fact, a common problem in stereo matching: at the depth discontinuties, some points are only visible in one image but occluded in the other image. So, the algorithm fails to find corresponding points. It can be noticed that in Figure 4.10(e) and (f), most of error matchings happen in the left-side edges of the wedding cake as opposed to right-side edges. As we explained, this is because in our algorithm we pick each pixel in left image and then look for its match in the right image. The pixels close to the left-side edges in left view of wedding cake are not visible in the right view, so, the correct match can not be found. But the pixels close to the right-side edges in the left view are still visible from the right point of view, so, the noise is mush less in these regions. Several solutions have been proposed to overcome the depth discontinuity problem. One straight forward solution is called left-to-right and right-to-left validation [14]. This technique performs two sets of matchings: first time, it searches for the matches of left image pixels in the right image, and second time, it searches for the matches of right image pixels in the left image. By combining the results of these two searches, this technique rejects the invalid matches and hence improves the performance. The system implemented in this work does not perform any such post processing operation. All the depth map results shown in this chapter are raw disparity data received from the Interpolation/Peak-Detection Unit. A technique such as left-to-right and right-to-left validating can be integrated with the current system with an estimated cost of 15% to 20% increase in the amount of required logic.

Figure 4.11 shows another example of stereograms with results from software and hardware. In the underlying surface, the background has 3 pixels disparity, the other three levels have 6,

8 and 12 pixels disparity. In this figure, the MSE for hardware disparity is 1.99 $pixel^2$, while the MSE for software disparity is 1.59 $pixel^2$. It can be seen in both software and hardware results that as the size of depth discontinuity becomes larger, the amount of noise on the edges becomes more visible.

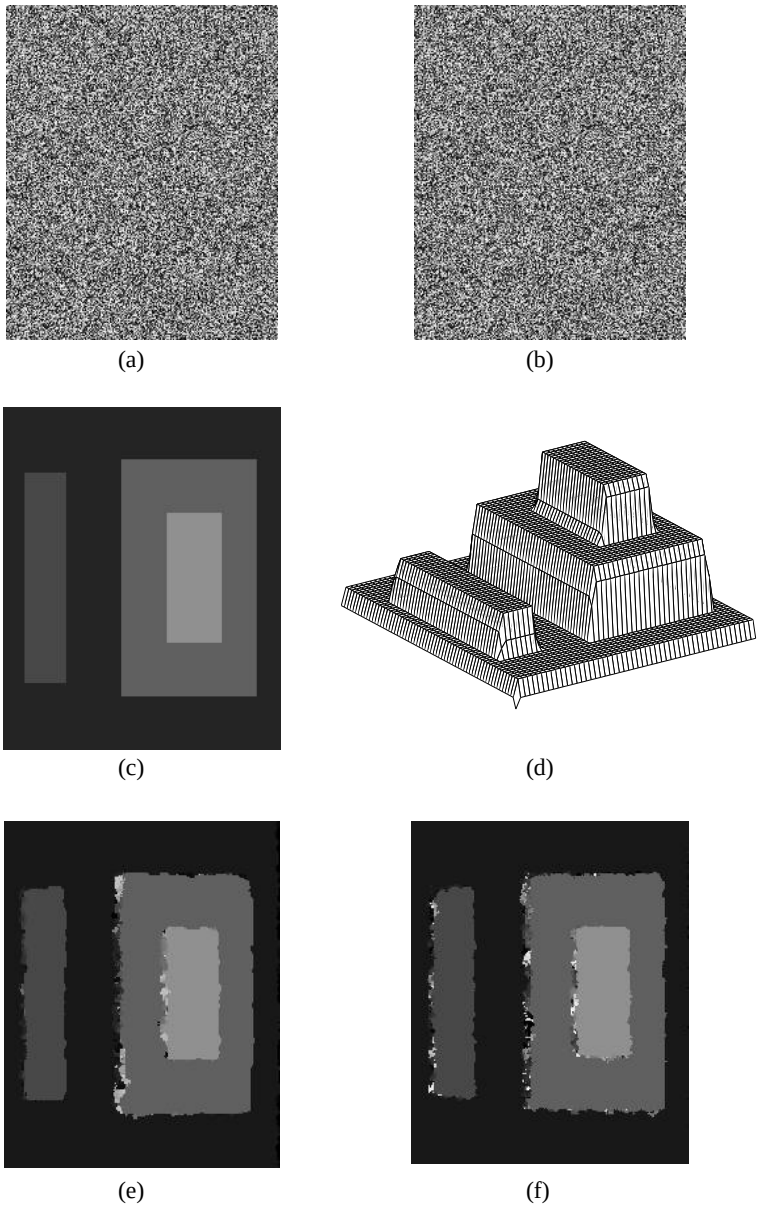
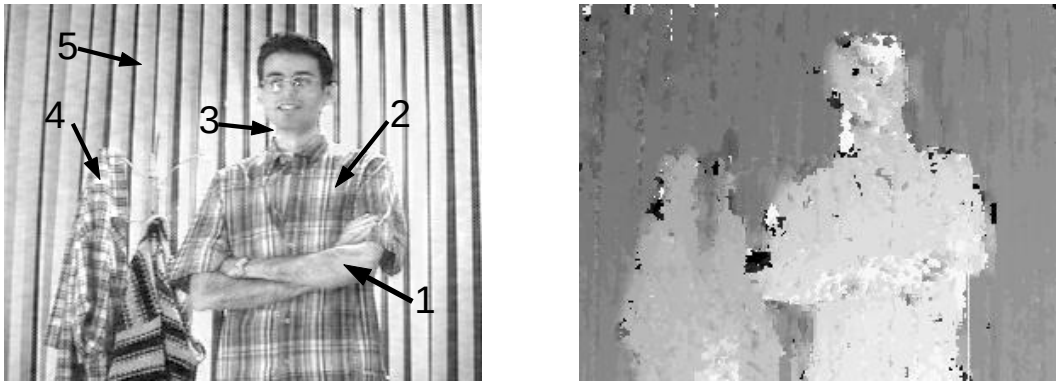


Figure 4.11: Overall system performance on a pair of random stereograms: (a), (b) Random-dot stereogram (c) Ground truth depth (d) 3D structure of ground truth (e) Depth from original software (f) Depth from hardware vision system



(a) (b)

Figure 4.12: (a) Input from right camera, (b) Output disparity map

Point #	1	2	3	4	5
Ground truth distance (cm)	300	315	320	365	410
Distance from stereo system (cm)	309	320	276	355	402
Error %	3%	1.6%	13.7%	2.7%	1.9%

TABLE 4. Distance measurements for five points of Figure 4.12

4.2.2 Natural Images

Figure 4.12 shows a sample natural image and the depth map generated by the hardware. As we can see, the background in this image has a constant distance from the camera, which is consistent with the plain grey background in the depth map result. The coat rack and the person in the image are closer to the camera respectively. Similar to synthesized test images, at depth discontinuties of the scene (such as point #3), the depth results are not reliable. Although the depth map result, as in Figure 4.12(b), qualitatively illustrates the performance of the stereo system, for quantitative evaluation, we have measured the distance of 5 points in the scene and compared it with the results obtained by hardware. Table 4 shows these two sets of distance values. In points 1, 2, 4 and 5, the error in the distance results is less than 5% of the absolute value of the distance.

Figures 4.13, 4.14, 4.15 and 4.16 show more samples of the stereo inputs and distance outputs in different scenes and in different lighting conditions. In Figure 4.16, the texture of the jacket on the coat rack is not visible in the original image. This is the main reason of noise in that region in the distance map.

It is important to note that in our current design, we have assumed that the two cameras have identical focal length and are vertically aligned such that no rectification [30] or any other pre-processing is required. But this assumption is not always true. There is usually a slight mis-alignment between the cameras and even sometimes in the position of the CCD plane inside the camera. This mis-alignment introduces some error in the disparity results of the natural images. Similar to the pre-processing stages, in this work there is no post-processing stage such as left-to-right/right-to-left validation or smoothing/gap filling implemented in hardware. These processing units can be integrated with the future versions of our system with some extra cost as we will explain in the next chapter.

(a) (b)

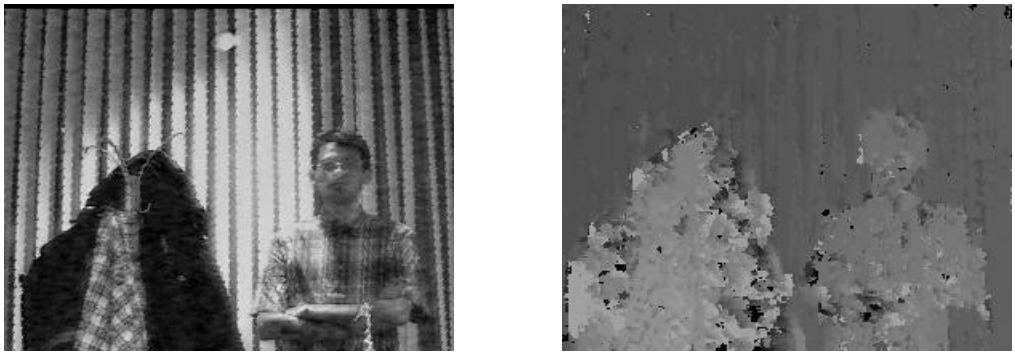
Figure 4.13: (a) Input from right camera, (b) Output disparity map

(a) (b)

Figure 4.14 (a) Input from right camera, (b) Output disparity map

(a) (b)

Figure 4.15 (a) Input from right camera, (b) Output disparity map



(a) (b)

Figure 4.16 (a) Input from right camera, (b) Output disparity map

CHAPTER 5

Conclusions & Future Work

In this work, we have built a hardware system that is capable of imaging its surroundings and reconstructing the 3D structure of the scene. It views the scene through a pair of stereo cameras and generates dense depth map results at a video rate of 30 frames/sec. The depth map obtained from video rate stereo vision can provide valuable information for higher level vision tasks such as object detection and pattern recognition which can be used in a variety of applications such as robotics, navigation, security, monitoring and automotives.

The stereo system developed in this research has two main features: First, it runs at video rate which is approximately 60 to 900 times faster than its software implementation, and second, it uses a high performance phase based stereo matching algorithm called “Local Weighted Phase-Correlation Algorithm”. The combination of these two features shows the significance of this work. While some other video rate vision systems have been reported before, we believe that this work is the first implementation of a complex algorithm such as LWPC in real

time.

Video rate performance in this work is achieved by implementing the algorithm on Field-Programmable Gate Arrays (FPGAs). Hardware implementation accelerates the stereo vision system by performing several parts of the algorithm fast and in parallel. On the other hand, reprogrammability of the FPGAs allows for faster and cheaper design cycle of the system compared to Application Specific Integrated Circuit (ASIC) design.

One important issue in this research has been the efficient translation of the original software algorithm to its corresponding hardware system. Some blocks of the original system were modified in order to reduce the size of the hardware system while introducing acceptable error.

There are several possibilities to continue this research. We categorize them into three groups. The first involves expanding the current stereo vision system to improve its performance. Secondly, we can look at other high performance stereo vision algorithms, third, investigating the feasibility of using better design tools to shorten the design cycle. In the following paragraphs, we will explain these suggestions in more detail.

Similar to other semiconductor devices, the capacity and speed of FPGAs are growing very quickly. Employing bigger and faster FPGAs will allow us to add more features to our current stereo system in order to improve the performance. The reprogrammability of the FPGAs also helps to modify the current system with minimum cost. When additional hardware resources become available, we would suggest these expansions respectively:

- Adding a post processing block such as left-right right-left validation to reject the invalid matches. This feature will almost double the size of correlation and interpolation units, but will also remove most of the noise resulted from occlusions or depth discontinuties. In fact, a slower version of left-right right-left validation can be added to the current stereo system with a small hardware cost: performing left-right matching and right-left matching on alter-

nating frames. By combining the depth results of each two consecutive frames, we can detect and reject the invalid matches. However, this solution will reduce the overall frame rate to 15 frames/sec from current 30 frames/sec.

- An alternative to the left-right, right-left validation is the idea of using results from previous frame while calculating the depth in current frame. At the present, our system starts from "scratch" to find the depth for every frame, yet we don't expect the world's depth structure to change dramatically from one frame to the next. Carrying over information from previous frames will allow us to improve the accuracy of the depth results dynamically after each frame.
- Adding pre-processing blocks such as rectification or noise filtering before applying the stereo matching algorithm. Depending on the accuracy of the camera pair calibration and alignment, the pre-processing blocks can improve the quality of final results dramatically.
- Performing G2/H2 filters in three directions($45°$, $-45°$ and $0°$) as opposed to the two directions ($45°$ and $-45°$) implemented in this work. This addition will increase the size of the Decomposition Unit and also the data transfer between this unit and the Phase-Correlation Unit by almost 50%, but will make the algorithm more robust and will extract more texture from the images.
- Introducing a confidence measure for depth values to indicate the goodness of fit between left and right matching points. The confidence measure shows the reliability of each depth value. Later stages of a vision system can use this information by assignning more weight to the depth values with higher confidence measure. For the LWPC algorithm, it is shown in [19] that the magnitude of the voting function, $C(x, \tau)$, provides the confidence measure and its value is bounded between 0 and 1.
- Increasing the size of search window in order to increase the maximum allowed disparity. This will expand the range of the stereo system by decreasing the minimum possible distance. Currently, the maximum disparity is limited to 20 pixels, which translates to a minimum detectable distance of about 2.5 meters. We estimate that choosing larger horizontal window size will linearly increase the size of Phase-Correlation and Interpolation Units.

- Finally, processing images larger than the current size of 256 x 360 pixels (higher resolution but with the same field of view). Dealing with higher resolution images will affect the hardware requirements in different ways. First, more pixels per image means less number of clock cycles available to process each pixel, which in turn requires faster and hence usually larger components (e.g. faster multipliers and dividers). Second, by increasing the resolution, the resulting disparity for an object in a fixed distance from the camera head will be larger (in terms of the number of pixels, assuming that the focal length of the cameras do not change). So, if we want to keep the minimum distance fixed, a wider search window will be required. This will again increases the hardware requirements, as explained in previous paragraph. Finally, larger images will of course require larger frame buffers and faster external memory accesses.

One other possible direction for future work is to investigate other stereo matching algorithms. This is very important to start with appropriate algorithm since some algorithms are naturally parallel and hence can be translated to parallel hardware implementation more easily and resulting in higher speed-ups.

The last possibility is to improve the design methodology by using design tools that allow automation of some design tasks that are currently performed manually. Among these tasks is the translation of high-level system description to a low-level hardware description language that can be then synthesized and translated to bit streams to program FPGAs. Recently, some commercial tools have been introduced that link the Digital Signal Processing (DSP) systems described in a system level design tool such as MATLAB or Simulink with Hardware Description Language (HDL) development tools [1][32]. These automation tools can be used to develop the sub-blocks of a multi-FPGA system although at present, the overall design needs to be tailored manually to the specifications of the hardware platform. Another design automation tool has also been introduced to find the optimum fixed-point precisions when replacing floating-point operations with their fixed-point counterparts [7].

In summary, this thesis has described an implementation of a state-of-the-art stereo disparity

algorithm. It is hoped that this system will find use as a module in a larger vision system and also that it demonstrates the practical and desirable nature of reconfigurable hardware for implementation of vision algorithms.

Bibliography

[1] Altera, DSP Builder User Guide, v2.0.0, *http://www.altera.com/literature/ug/ug_dsp_builder.pdf*, 2002.

[2] J.M. Arnold, D.A. Buell, D.T. Hoang, D.V. Pryor, N. Shirazi, M.R. Thistle, The Splash 2 Processor and Applications, *IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages: 482-485, 1993.

[3] N. Ayache and B. Faverjon, Efficient registration of stereo images by matching graph descriptions of edge segments, *Intern. J. Comput. Vis. 1(2)*:107:131, 1987.

[4] H.H. Baker and T.O. Binford, Depth from edges and intensity based stereo, *Proc. 7th Intern. Joint Conf. Artif. Intell.*, Vancouver, pp. 631-636, August 1981.

[5] S.T. Barnard, Stochastic stereo over scale, *International Journal of Computer Vision*, 3:17-32, 1989.

[6] S. Brown, R. Francis, J. Rose, Z. Vranesic, *Field-Programmable Gate Arrays*, Kluwer Academic Publishers, May 1992.

[7] M. L. Chang and S. Hauck, Precis: A design-time precision analysis tool, *10th IEEE Symposium on Field-Programmable Custom Computing Machines*, April 2002.

[8] G.C. DeAngelis, I. Ohzawa and R.D. Freeman, Depth is encoded in the visual cortex by a specialized receptive field structure, *Nature*, 352:156-159, 1991.

[9] O. Faugeras, et al., Real time correlation based stereo: algorithm, implementations and applications, Research Report 2013, INRIA Sophia-Antipolis, 1993.

[10] D.J. Fleet, A.D. Jepson and M. Jepson, Phase-based disparity measurement, *CVGIP: Image Understanding*, 53(2):198-210, 1991.

[11] D.J. Fleet, Disparity from local weighted phase-correlation, *IEEE International Conference On Systems, Man, and Cybernetics*, page(s): 48 - 54 vol.1, 1994.

[12] D.J. Fleet and A.D. Jepson, Stability of phase information, *IEEE Trans. PAMI*, 15(12):1253-1268, 1993.

[13] W.T. Freeman and E.H. Adelson, The design and use of steerable filters, *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, Volume: 13, Issue: 9, Page(s): 891 -906, Sept. 1991.

[14] Pascal Fua, A parallel stereo algorithm that produces dense depth maps and preserves image features, *Machine Vision and Applications*, INRIA research report 1369, 6(1), 1993,

[15] W.E.L. Grimson, *From Images to Surfaces: a computational study of human early visual system*, MIT press, Cambridge, MA, 1981.

[16] K.M. Hou, et al., A reconfigurable and flexible parallel 3D vision system for a mobile robot, *Proc. Computer Architectures for Machine Perception*, pp. 215 - 221, Dec 1993.

[17] IEEE, *IEEE standard VHDL Language Reference Manual, std 1076-1993*, New York, 1993.

[18] M. Van Ierssel, D. Galloway, P. Chow, J. Rose, The Transmogrifier-3a: Hardware and Software for a 3 Million Gate Rapid Prototyping System, *Micronet Annual Workshop*, 2001.

[19] M. Jenkin and A. Jepson, Recovering local surface structure through local phase difference measurements, *CVGIP: Image Understanding* 59: 72-93, 1994.

[20] F. Jutand, S. Maginot, N. Demassieux, H. Maitre, ENSTA Single chip VLSI architecture for a real time stereo vision processor, *ICASSP-88.*, Pages: 1965 - 1968, vol.4, 1988.

[21] T. Kanade, A. Yoshida, K. Oda, H. Kano, M. Tanaka, A stereo machine for video-rate dense depth mapping and its new applications, *IEEE conf. on Computer Vision and Pattern Recognition*, pp 196-202, 1996.

[22] R. McCready, *Real-Time Face Detection on a Configurable Hardware Platform*, Master's thesis, University of Toronto, 2000.

[23] T. Kanade and M. Okutomi, A stereo matching algorithm with an adaptive window: theory and experiment, *IEEE International Conference on Robotics and Automation*, Vol. 2, pages: 1088 -1095, 1991.

[24] D. Lewis, D. Galloway, M. Van Ierssel, J. Rose, P. Chow, The Transmogrifier-2: A 1 Million Gate Rapid Prototyping System, *in IEEE Trans. on VLSI*, Vol. 6, No. 2, pp 188-198, 1998.

[25] D. Marr and T. Poggio, A theory of human stereo vision, AI lab, MIT, 1977.

[26] Philips Semiconductors, *http://www-us.semiconductors.philips.com/cgi-bin/pldb/pip/SAA7111H*

[27] A. Rushton, *VHDL for Logic Systhesis*, John Wiley, 1998.

[28] TM-3 documentation, *http://www.eecg.utoronto.ca/~tm3/*

[29] TM-3 documentation, *http://www.eecg.utoronto.ca/~tm3/ports.ps*

[30] E. Trucco, A. Verri, *Introductory Techniques for 3-D Computer Vision,* Prentice Hall, 1998.

[31] J.E. Vuillemin, P. Bertin, D. Roncin, M. Shand, H.H. Touati and P. Boucard, Programmable Active Memories: reconfigurable systems come of age, *IEEE Trans. on VLSI Systems*, 4:56-69, March 1996.

[32] J. Woodfill and Von Herzen, Real-time stereo vision on the PARTS reconfigurable computer, *The 5th Annual IEEE Symposium on Field-Programmable Custom Computing Machines Proceedings*, Pages:201-210, 1997.

[33] Xilinx data sheets, *http://direct.xilinx.com/bvdocs/publications/ds022-1.pdf*

[34] Xilinx, System Generator for DSP Reference Guide, v2.2, *http://www.xilinx.com/ipcenter/dsp/ref_guide.pdf*, July 2002.

[35] A.G. Ye, *Procedural Texture Mapping on FPGAs*, Master's thesis, University of Toronto, 1999.

[36] R. Zabih and J. Woodfill, Non-parametric Local Transforms for Computing Visual Correspondence, *Proceedings of 3rd European Conference on Computer Vision*, pp. 150-158, May 1994.