STEREO VISION SYSTEM MODULE FOR LOW-COST FPGAS FOR AUTONOMOUS MOBILE ROBOTS

A Thesis

Presented to

the Faculty of California Polytechnic State University

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Connor Citron

August 2014

COMMITTEE MEMBERSHIP

TITLE:	Stereo Vision System Module for Low-Cost FPGAs for Autonomous Mobile Robots
AUTHOR:	Connor Citron
DATE SUBMITTED:	August 2014
COMMITTEE CHAIR:	Professor John Seng, Ph.D., Department of Computer Science
COMMITTEE MEMBER:	Professor Franz Kurfess, Ph.D., Department of Computer Science
COMMITTEE MEMBER:	Professor Chris Lupo, Ph.D., Department of Computer Science

Abstract

Stereo Vision System Module for Low-Cost FPGAs for Autonomous Mobile Robots

Connor Citron

Stereo vision uses two adjacent cameras to create a 3D image of the world. A depth map can be created by comparing the offset of the corresponding pixels from the two cameras. However, for real-time stereo vision, the image data needs to be processed at a reasonable frame rate. Real-time stereo vision allows for mobile robots to more easily navigate terrain and interact with objects by providing both the images from the cameras and the depth of the objects. Fortunately, the image processing can be parallelized in order to increase the processing speed. Field-programmable gate arrays (FPGAs) are highly parallelizable and lend themselves well to this problem.

This thesis presents a stereo vision module which uses the Sum of Absolute Differences (SAD) algorithm. The SAD algorithm uses regions of pixels called windows to compare pixels to find matching pairs for determining depth. Two implementations are presented that utilize the SAD algorithm differently. The first implementation uses a 9x9 window for comparison and is able to process 4 pixels simultaneously. The second implementation uses a 7x7 window and processes 2 pixels simultaneously, but parallelizes each SAD algorithm for faster processing. The 9x9 implementation creates a better depth image with less noise, but the 7x7 implementation processes images at a higher frame rate. It has been shown through simulation that the 9x9 and 7x7 are able to process an image size of 640x480 at a frame rate of 15.73 and 29.32, respectively.

ACKNOWLEDGMENTS

Thanks to:

- John Seng for his support and guidance through this project.
- My Mom and Dad for their love and support (and letting me use them as "rubber duckies" when talking through problems and accomplishments).
- Cal Poly Robotics Club and its past and present members for helping to foster an enjoyment and passion for all things robotic during the time spent with them.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Stereo vision uses two adjacent cameras to create a three dimensional image. This is similar to how human eyes work. A depth map can be created by comparing the offset of a pair of corresponding pixels of the two cameras. This depth map is a three dimensional representation of the real world. Mobile robots can use stereo vision to improve their awareness of their surroundings.

The point cloud made from the pixels of the depth map in combination with one of the actual images allows for object detection and object identification. As opposed to infrared laser scanning, which can only be used indoors, stereo vision can be used anywhere there is adequate lighting. The data obtained from a stereo vision system can be used to map or recreate objects and places from the environment [12].

Some of the earliest research of stereo vision was used with industrial robots [23]. In the 1980s, the challenge of industrial robots needing to avoid unexpected obstacles was addressed with stereo vision in order to detect those objects quickly and to determine how far the robot would need to adjust its course to prevent accidental collisions [24].

As stereo vision systems become more essential for mobile robots, embedded stereo vision systems become more important. Embedded stereo vision systems allow for smaller robots to achieve the same capabilities as their larger counter-

parts [8].

One problem faced with stereo vision systems is the amount of information that needs to be processed to allow for real time operations, which can make the robot perform slowly [31]. Smaller image sizes will help speed up performance, but at the cost of the resolution of the objects.

Most of the image processing is independent of the image which allows for parallelization when processing each image. In the 1990s, research into using field programmable gate arrays (FPGAs) with stereo vision began to gain momentum due to the parallelizability of FPGAs [15]. In the 2000s and onward is when FPGAs became more practical for higher speeds and higher image resolutions for real time mobile robot applications [20].

Mobile robots such as autonomous quadrupeds are able to use stereo vision to navigate difficult terrain while avoiding obstacles in their path [30].

The stereo vision system module presented in this paper is used on a FPGA Atlys board [3] and is shown to work with two different types of implementations of the Sum of the Absolute Differences (SAD) algorithm.

Background information on stereo vision and the SAD algorithm used in stereo vision implementations in this paper can be found in Chapter 2. Related work is presented in Chapter 3. The implementations of the system used on the FPGA board is described in Chapter 4. Experiments and results are presented in Chapter 5. Finally, the conclusion and future work are in Chapter 9 and Chapter 10, respectively.

Chapter 2

Background

This chapter presents some general information on stereo vision that should be useful for understanding the decisions that were made in developing this stereo vision system module.

2.1 Computer Stereo Vision Overview

Computer vision is concerned with using computers to understand and use information that is within visual images [17]. There are many different types of computer vision, which range from using one image to multiple images in order to obtain information. One image cannot provide the depth of the objects within the image.

Stereo vision uses multiple images of the same scene, taken from different perspectives, in order to construct a three dimensional representation of the objects in the images [14]. Comparing multiple images together for their similarities and differences allows for the depth to be obtained.

Binocular stereo [22] involves comparing a pair of images. These images are normally acquired simultaneously from a scene. By searching for corresponding pairs of pixels between the two images, depth information can be determined [22]. Pixel based comparisons can require substantial amount of computational power and time. Certain assumptions are made because of the resources required. Cam-

era calibration and epipolar lines [22] are common assumptions. Camera calibration refers to the orientation of the cameras to each other. Epipolar lines are lines that can be drawn through both images that intersect corresponding points. Ideally, the epipolar lines will go horizontally through the images. For example, take two images of the same scene that are 640x480 pixels in size. Each image contains 307,200 pixels, which is over 600,000 pixels between the two images for one frame. For a real-time application, say 30 frames per second, becomes over 18 million pixels between the two images that would need to be processed every second.

Computational requirements for real-time applications can be reduced in several ways. First, lowering the number of pixels in the images will reduce the number of pixel comparisons in each second. Images at a size of 320x240 pixels would require a quarter of the number of computations, but at the cost of losing detail of the objects in the images. Also, reducing the number of frames per second will decrease the amount of computing needed. Going much below 30 frames per second is noticeable to a person and can be annoying to observe a low frame rate. A robot on the other hand, depending on its task and how fast it is moving, might only need a few frames per second (e.g. 10) in order to function within desired parameters. Image resolution could be more important than frame rate for a robot if object details are more important than frames per second.

Figure 2.1 represents a simplified illustration of binocular stereo vision. The two cameras are held at a known fixed distance from each other and are used to triangulate the distance of different objects in the images they create. The points U_L and U_R in the left and right images, respectively, are 2D representations of the point P in 3D space. By comparing the offset between U_L and U_R in the two images, it is possible to obtain the distance of point P from the cameras [2].

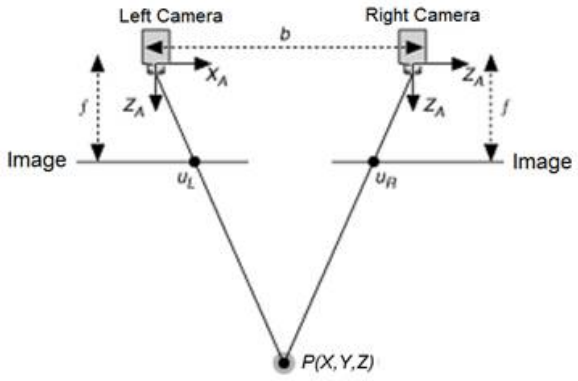


Figure 2.1: Simplified binocular stereo vision system [2].

The closer an object is to the stereo vision system, the greater the offset of corresponding pixels will be. If an object is too close to the system, it is possible for one camera to see part of an object that the other camera cannot. The farther an object is away from the stereo vision system, the smaller the offset of corresponding pixels. If an object is far enough away, it is possible for an object to be in almost the exact same location in both images. You can show this to yourself by holding a finger up close to your face, close one eye, and then alternate between which eye is open and which eye is closed. Your finger should appear to move a noticeable amount. Next, hold your finger as far away from you as you can and again alternate between which eye is open and which is closed. You should notice that your finger appears to move significantly less than it did when your finger was close to your face. That is how stereo vision works. The distance of an object is inversely proportional to the amount of offset between the two images.

2.1.1 Parallelism in Stereo Vision

Processing images for stereo vision allows for a high degree of parallelism. Locating the corresponding position of a pair of pixels is independent of finding another corresponding pair of pixels. This independence allows for the ability to process different parts of the same images at the same time, as long as there is hardware to support it.

Field programmable gate arrays (FPGAs) allow for a higher degree of parallel processing to be implemented compared to using the CPU on a computer. In Section 4 the amount of parallel processing used for the stereo vision module presented in this paper is discussed.

2.2 Stereo Vision Algorithms

Stereo vision algorithms can be placed into one of 3 categories: pixel-based methods, area-based methods, and feature-based methods [8]. Pixel-based methods utilize pixel by pixel comparisons. They can produce dense disparity maps, but at the cost of higher computation complexity and higher noise sensitivity [8]. Area-based methods utilize block by block comparisons. They can produce dense disparity maps and are less sensitive to noise, however, accuracy tends to be low in areas that are not smooth [8]. Feature-based methods utilize features, such as edges and lines for comparisons. They cannot produce dense disparity maps, but have a lower computational complexity and are insensitive to noise [8].

There are a lot of different stereo vision algorithms [28]. In the taxonomy of [28], 20 different stereo vision algorithms were compared against each other using various reference images. Many algorithms used are based on either the

Sum of Absolute Differences (SAD) or correlation algorithms [21].

An algorithm that is similar to SAD is the Sum of the Square Differences (SSD). Both of these algorithms produce similar results and contain around the same amount of error [8]. SAD was chosen over the other algorithms to implement in this paper because it is highly parallilizable and is simpler to implement in hardware. SSD requires squaring the difference between corresponding pixels and summing it up. Squaring a number requires more over head and more hardware than just taking the absolute value of the difference of a corresponding pair.

2.2.1 Sum of the Absolute Differences (SAD) Algorithm

SAD is a pixel-based matching method [21]. Stereo vision uses this algorithm to compare a group of pixels called a window from one image with a window in another image to determine if the corresponding center pixels match. The SAD algorithm, shown in Equation 2.1 [21], takes the absolute difference between each pair of corresponding pixels and sums all of those values together to create a SAD value. One SAD value by itself does not give any useful information about the two corresponding center pixels. Several SAD values will be calculated from different candidate windows for each reference window. Out of the all the SAD values calculated for the reference window, the SAD value with the smallest value (all of them are greater than or equal to 0 because of the absolute part in the equation) is determined to contain the matching pixel. Figure 2.2 shows that for one reference window, there are several candidate windows. The line that the candidate windows are chosen from is called an epipolar line.

$$\sum_{(i,j)\in W} |I_1(i,j) - I_2(x+i, y+j)| \tag{2.1}$$

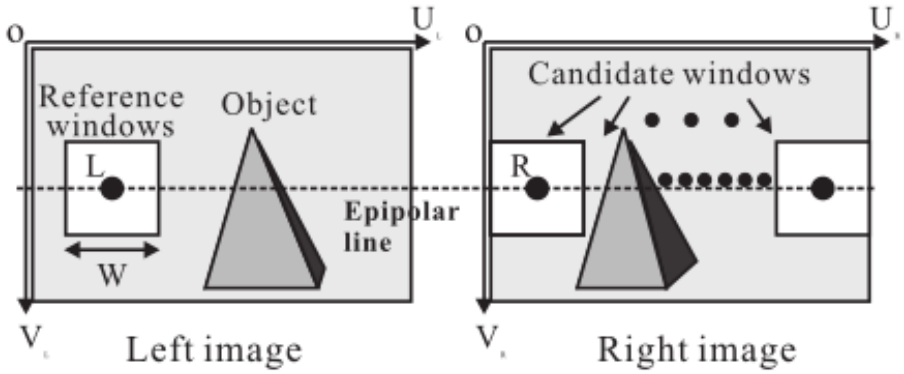


Figure 2.2: Searching for corresponding points between the two images [18].

In stereo vision, epipolar lines are created from the two cameras capturing images from the same scene. Figure 2.3 shows the epipolar line that point X must be on in the corresponding images. This is useful because if the epipolar lines are known for both images, then it is possible to know the line that two corresponding points are on. It reduces the problem of finding the same two points from a 2D area to a 1D line. Now, if the epipolar lines in both images are horizontal as they are in Fig. 2.2 as opposed to them being at a diagonal as they are in Fig. 2.3, then Eq. 2.1 reduces to Equation 2.2. For cameras that are not perfectly aligned, rectification is often used in order to align epipolar lines between images [19]. However, many stereo vision algorithms will assume that the epipolar lines are rectified to simplify the overall processing required.

$$\sum_{(i,j)\in W} |I_1(i,j) - I_2(x+i,j)| \tag{2.2}$$

The disparity is the amount of offset between two corresponding pixels. The disparity range is the number of pixels that the candidate window will move through the image and is represented by the value 'x' in Eq. 2.2. It corresponds to the amount of SAD values that will be calculated for each pixel. Figure 2.4

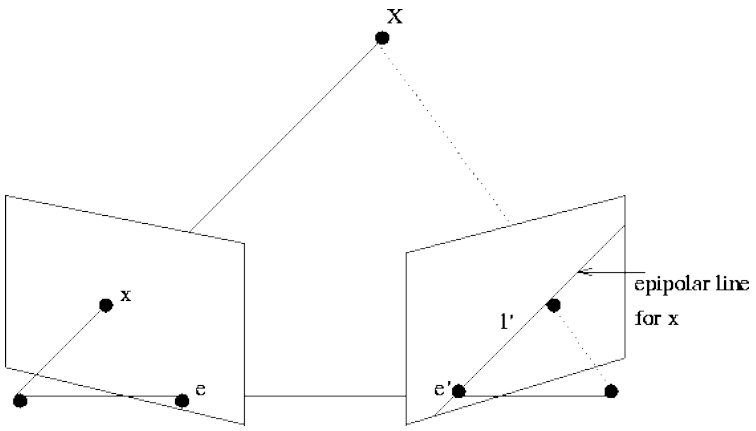
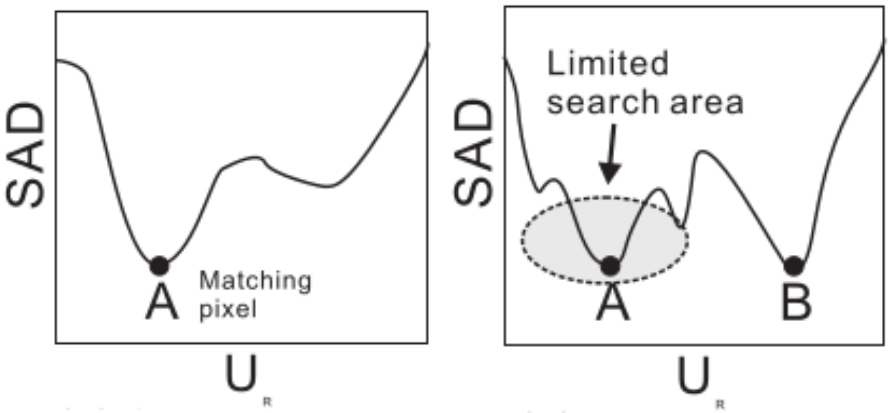


Figure 2.3: The epipolar line that point X is on for both images [9].

shows two types of SAD search methods. Fig. 2.4a selects the overall SAD value with the lowest value to be the matching pixel. However, Fig. 2.4b limits the search region to a specific area. This helps to avoid issues of similar looking areas that are not near the reference window from being falsely identified as matching. The downside to this method is that if an object gets too close, meaning it would have high disparity values, and if the search region is not large enough, then the distance of the object will be miss classified. It is important to determine a window size and a search region that fit desired parameters.

For example, Figure 2.5a shows a reference (template) window from one image. Figure 2.5b shows the candidate (search) area in from the other image. The disparity range is 3, or 0 to 2. There are three 3x3 windows within the search region in Fig. 2.5b. From left to right the three search windows have their center pixel as 4, 6, and 5, respectively.

Comparing corresponding pixels in the template window with the first search window (S0) gives the absolute differences for all 9 pixels going from left to right and top to bottom of 8, 1, 1, 2, 1, 0, 1, 2, and 2. So the SAD value for S0



(a) Global SAD search (b) Local SAD search

Figure 2.4: The SAD between a reference window and several candidate windows [18].

is 18, which is obtained by adding up all nine of those values. The SAD value for the second search window (S1) is 6 and the last search window (S2) is 13. The template window has the smallest SAD value with S1. Therefore the center pixel in S1 is determined to be the corresponding pixel for the center pixel in the template window. The disparity value is 1 (how far the matching search window was shifted to the right). The disparity value, along with many others, is used to create a disparity map. Each disparity value in the disparity map is at the same relative location that the center pixel of its corresponding template window is located.

1	2	3
4	5	6
7	8	9

(a) Template Window

9	1	2	4	5
2	4	6	5	3
8	6	7	8	7

(b) Search Region

Figure 2.5: Template (reference) window and search (candidate) window.

Chapter 3

Related Work

There are several different ways to implement a stereo vision system. Many stereo vision systems are implemented on field-programmable gate arrays (FPGAs). FPGAs allow for parallelization when processing images. Systems that use FPGAs generally can achieve a high frames per second with a decent or good image quality, but most of these systems are expensive.

FPGA Design and Implementation of a Real-Time Stereo Vision System [21] uses an Altera Stratix IV GX DE4 FPGA board to process the right and left images that come from the attached cameras. It uses the Sum of Absolute Differences (SAD) algorithm to compute distances. This system allows for real time speeds, up to 15 frames per second at an image resolution of 1280x1024. However, the Altera Stratix IV GX DE4 FPGA board costs over $4,000 [5], which makes the system impractical for non-high budget projects.

Improved Real-time Correlation-based FPGA Stereo Vision System [16] uses a Xilinx Virtex-5 board to process images. It uses a correlation-based algorithm, which is based on the Census Transform, to obtain the depth in images. The algorithm is fast, but there are some inherent weaknesses to it. This system can run at 70 frames per second for images at a resolution of 512x512. Unfortunately, the Xilinx Virtex-5 board costs more than $1,000 [6], which is still expensive for such users as club projects and other users on a budget.

Low-Cost Stereo Vision on a FPGA [26] uses a Xilinx Spartan-3 XC3S2000 board. It uses the Census Transform algorithm for image processing. This allows images with a resolution of 320x240 to be processed at 150 frames per second. The total hardware for the low-cost prototype used in [26] costs just over $1,000, which is a bit too pricy for a lot of projects.

An Embedded Stereo Vision Module For Industrial Vehicles Automation [13] uses a Xilinx Spartan-3A-DSP FGPA board. It uses an Extended Kalman Filter (EKF) based visual simultaneous localization and mapping (SLAM) algorithm. The accuracy of this system directly varied with speed and distance of detected object. The Xilinx Spartan-3A-DSP FGPA board is around $600 [8], which is less expensive than the other systems presented so far.

Several commercial stereo vision systems exist presently [13]. Most of them are quite capable of producing good quality depth maps of their surroundings. However, the cost of these products can be relatively expensive, especially from a club or hobbyist standpoint. The Bumblebee2 [4] from Point Grey is able to produce disparity maps at a rate of 48 frames per second for an image size of 640x480, but it costs somewhere around $1,000 or so. Having been involved with the Cal Poly Robotics Club for 6 years and seen the budgets each project in the club usually gets, $1,000 would be most of a project's budget for the year. That kind of money could be better spent elsewhere on a project.

During the course of this thesis, a stereo vision surveillance application paper [27] was published that used the Digilent Atlys board [3]. A stereo camera module, VmodCAM [7], can be purchased with the Atlys board and was also used. The Atlys board is relatively inexpensive, at least by the standards presented thus far, at $230 for academic use. With the VmodCAM included, the price goes up to around $350, which is still a significant cost savings over other

FPGA boards presented. The cost and capacity of the board are why the Atlys board was selected for use in this thesis (the selection was independent of the surveillance paper). The surveillance paper used the AD Census Transform to calculate distance. Their disparity map data from the board was displayed on monitor through the board's HDMI output. The output image is rather noisy, but it is very easy for a human to understand what is in the image and it can show the depth of objects relatively close to the cameras.

Chapter 4

Implementation

This chapter presents the implementation and architecture of the stereo vision system module presented in this paper.

4.1 Architecture Overview

The stereo vision module in this paper is composed of three main parts: SAD, minimum comparators, and a wrapper, which goes around the previous two that takes in image data and outputs disparity values.

The code for the following sections is located on github under: `https://github.com/cccitron/mastersThesis`.

4.1.1 Sum of the Absolute Differences Architecture

Two versions of the SAD algorithm have been implemented in this paper. The first uses a 9x9 window and the other one uses a 7x7 window. Figure 4.1 shows the top level entity of the SAD implementation used. Both versions have a clocked input (clk_I) and a one bit data input (data_I) to notify the algorithm to begin calculating the SAD value. The template_window_I and search_window_I between the two versions differ because 49 (7x7) or 81 (9x9) bytes are sent to the sadAlgorithm entity. The data out signal (data_O) notifies when the calculation is complete and is ready for the next set of input. The calculated SAD value is

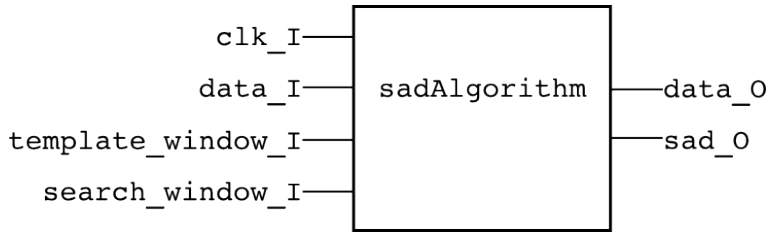


Figure 4.1: The top level SAD algorithm implementation.

sent out of the entity through sad_O.

There is a slight variation between the standard SAD algorithm and how it is implemented in this stereo vision system. Instead of subtracting two pixel values and then taking the absolute difference between them, these implementations in this paper find which corresponding pixel has a greater value and then sends the two pixels to the subtracter. See Appendix A and Appendix B for the code used. The subtracter then takes the greater value and subtracts from it the lesser value and returns the difference, "sub". The value sub will always be greater than or equal to zero, which is equal to the absolute difference of the two corresponding pixels. This process allows for the absolute value to be obtained without having to deal with signed values and the additional bits needed to account for the signed portion of the negative values.

4.1.1.1 State Diagram

Inside the sadAlgorithm entity from Fig. 4.1, the state machine from Figure 4.2 controls the SAD algorithm. The state machine begins at state SO and initializes all the values used in it to 0. It then proceeds to S1 where the state machine remains on standby until data_I becomes '1'. In S2, the counter starts at 0, the subtraction between corresponding pixel values begins, and on the next clock

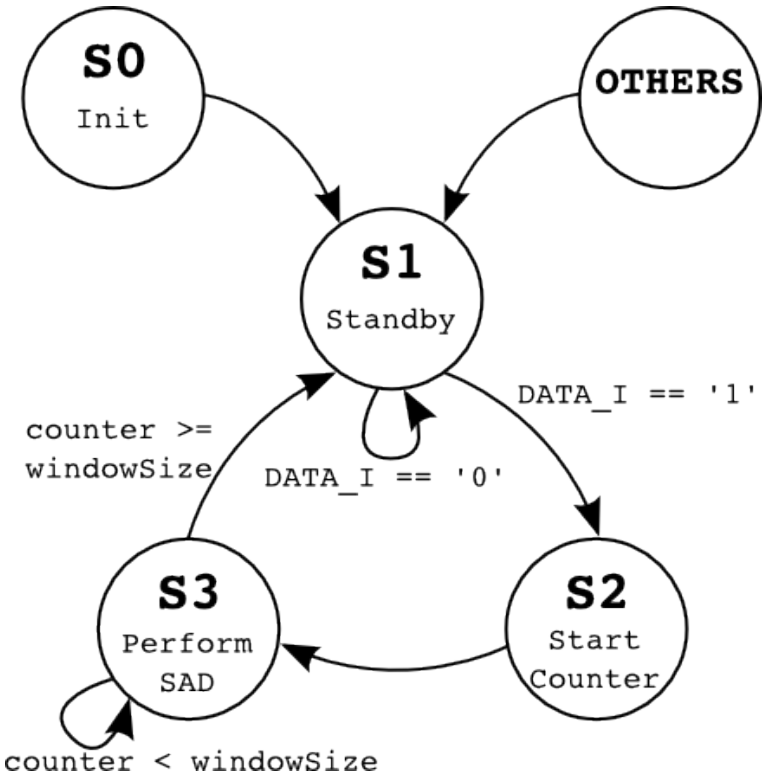


Figure 4.2: The state machine for implementing the SAD algorithm.

cycle, the state will be S3. While in S3, the counter is incremented by 1 every clock cycle. S3 is where the SAD algorithm is performed. After the counter is equal to windowSize of 7 for the 7x7 or 81 for the 9x9 (see Section 4.1.1.2 and Section 4.1.1.3 for details) the SAD calculation is complete. The state machine sets data_O to '1' to notify the SAD wrapper that the calculation is complete and the state moves to S1 and waits for the next set of input.

4.1.1.2 9x9 Window

The 9x9 window implementation operated with 4 pixels processed in parallel. Every pixel has 16 SAD operations processed in parallel. So there are 64 SAD entities in this implementation. However, each SAD calculation has a higher

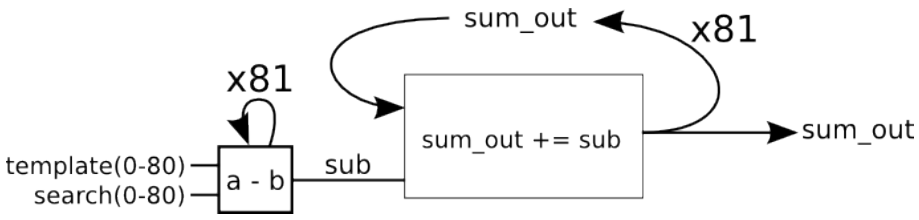


Figure 4.3: Architecture overview of the SAD algorithm with the 9x9 window implementation.

degree of serialization than the 7x7 window implementation in order to reduce space to fit on the Atlys board [3]. Figure 4.3 shows a simplified version of this process. For each of the 81 clock cycles, the difference between corresponding pixels is calculated. Beginning one clock cycle after the differences start to be calculated the difference, sub, sum_out is added to itself and sub. This process also occurs 81 times, one addition for each clock cycle. The state machine in Figure 4.2 stops the calculation for sum_out after the full SAD value has been summed up.

Figure 4.4 illustrates the pipeline used in a SAD calculation for the 9x9 window version. It takes 81 clock cycles to take the differences between all 81 pairs of pixel values. After the first difference is calculated, the differences can then be summed up. The summing also takes 81 clock cycles and ends one cycle after the last difference is calculated. This results in the SAD algorithm taking 82 clock cycles.

The code for the 9x9 window implementation can be found on github:
`https://github.com/cccitron/mastersThesis/tree/master/makestuff/libs/libfpgalink-20120621/hdl/fx2/vhdl/sad_buffer_9x9`

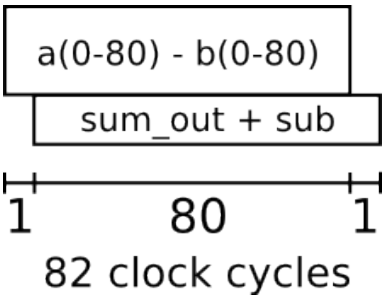


Figure 4.4: Pipeline architecture of the SAD algorithm with the 9x9 window implementation.

4.1.1.3 7x7 Window

The 7x7 window implementation operated with 2 pixels processed in parallel. Each pixel has 16 SAD operations processed in parallel. There are only 32 SAD operations occurring in parallel, as opposed to 64 that were performed in parallel in Sec. 4.1.1.2. The 7x7 window size has 32 pixels less than the 9x9 version for each window in every SAD calculation. The process was able to utilize a higher degree of parallelization. The increased parallelism takes up more space on the board than the serial version from Sec. 4.1.1.2. Figure 4.5 shows a simplified version of this process. Each clock cycle during 7 cycles, the difference, sub, between corresponding pixels is calculated. One clock cycle after the differences begin to be calculated, sum_out is added to itself and the value sub. This process also occurs 7 times, one set of addition each clock cycle. The state machine in Figure 4.2 stops the calculation for sum_out after the full SAD value has been summed up.

The main difference between this implementation and the 9x9 window implementation from Sec. 4.1.1.2 is that the calculation of the differences between corresponding pixels is parallelized to calculate 7 absolute values at once. The

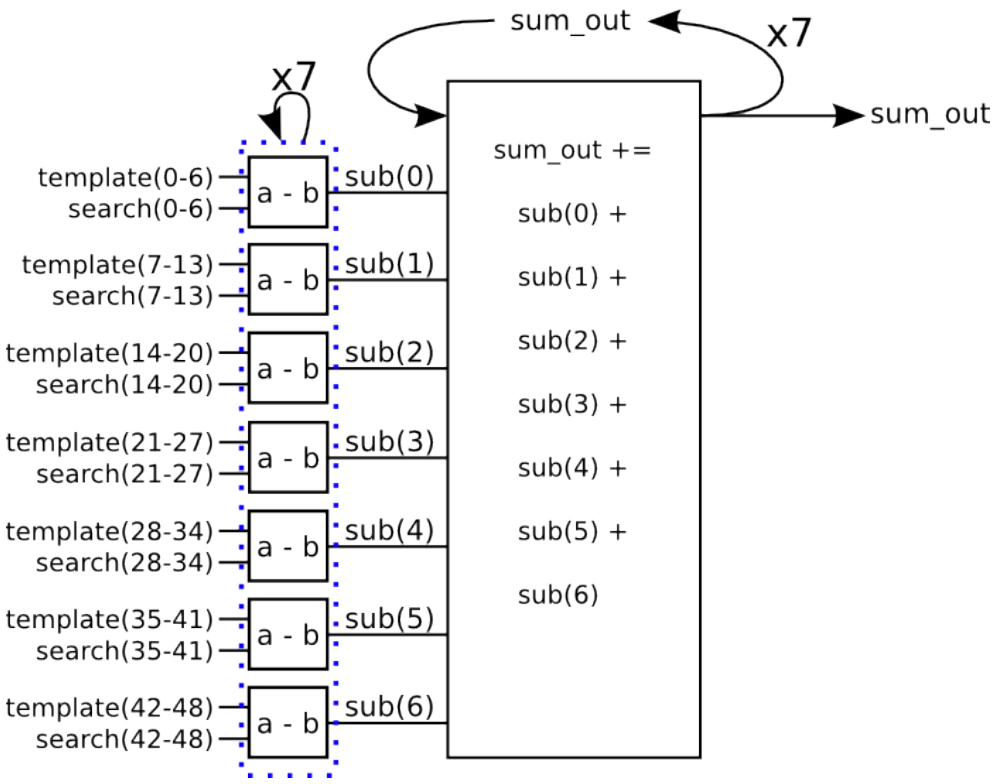


Figure 4.5: Architecture overview of the SAD algorithm with the 7x7 window implementation.

dotted box in Figure 4.5 represents all 7 of the subtraction calculations occurring 7 times in the SAD calculation. Instead of requiring 49 clock cycles to calculate all the differences, it only takes 7 clock cycles. All 7 of the differences that were calculated are added to sum_out each clock cycle.

Figure 4.6 shows the pipeline used for the 7x7 window version. It takes 7 clock cycles to calculate the differences between all 49 pairs of pixel values. After the first set of differences is calculated, the differences can begin to be summed up. The summing also takes 7 clock cycles and ends one cycle after the last difference is calculated. This results in a total of 8 clock cycles.

The code for the 7x7 window implementation can be found on github:

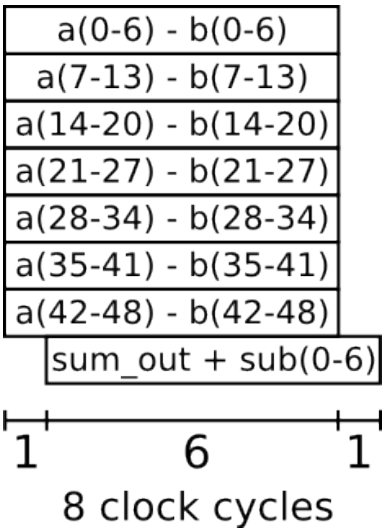


Figure 4.6: Pipeline architecture of the SAD algorithm with the 7x7 window implementation.

`https://github.com/cccitron/mastersThesis/tree/master/makestuff/libs/libfpgalink-20120621/hdl/fx2/vhdl/sad_buffer_parallelSAD_7x7`

4.1.2 Minimum Comparator Architecture

The purpose of the minimum comparator is to find the lowest value of two input values and output the lowest value. The top level implementation of the minimum comparator is shown in Figure 4.7. The process is synchronous, noted by the clock clk_I. The index, pos0_I and pos1_I, of the SAD values sad0_I and sad1_I, respectively, ranges from 0 to 15, which gives a disparity range of 16.

Appendix C shows the code for the minimum comparator. If sad1 is less than sad0, then sad1 and its index, pos1, are returned, otherwise sad0 and pos0 are returned. Using a less than comparison is supposed to take up less hardware than a greater than or equal to comparison [10]. This is useful because 15 minimum comparators (see Figure 4.8) are used for each pixel that is processed in parallel.

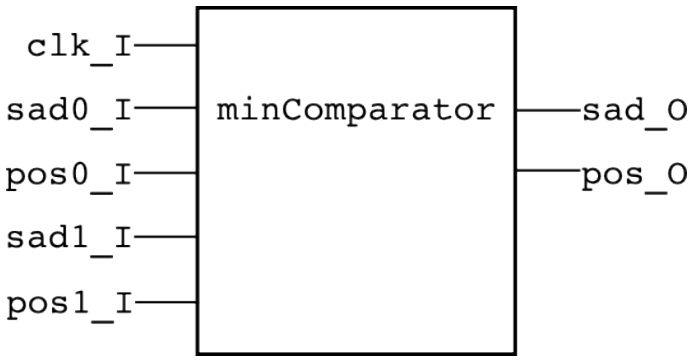


Figure 4.7: The top level minimum comparator implementation.

So 30 minimum comparators were used for the 7x7 window implementation and 60 minimum comparators were used for the 9x9 window implementation. Constructing the minimum comparator in this way accounts for cases where 2 SAD values are equal to each other. The SAD value with the lower index is always assigned to the sad0_I input and the higher indexed SAD value goes to the sad1_I input. Therefore, if 2 values are equal, the SAD value with a lower index, and a lower disparity, will be returned.

Multiple minimum comparators were put together to create a tree, as shown in Figure 4.8, in order to quickly determine which SAD value was the lowest. This process is used to find the index of the lowest SAD value out of the 16 SAD values calculated for each pixel. A normal serial comparison of 16 values would take 15 comparisons, or 15 clock cycles, if one comparison occurred each clock cycle. Having 15 comparators allows the number of SAD values needed to be reduced by half each clock cycle. Using a tree of comparators drops the comparison time from 15 clock cycles to only 4 clock cycles. It is almost a 4 times speed up.

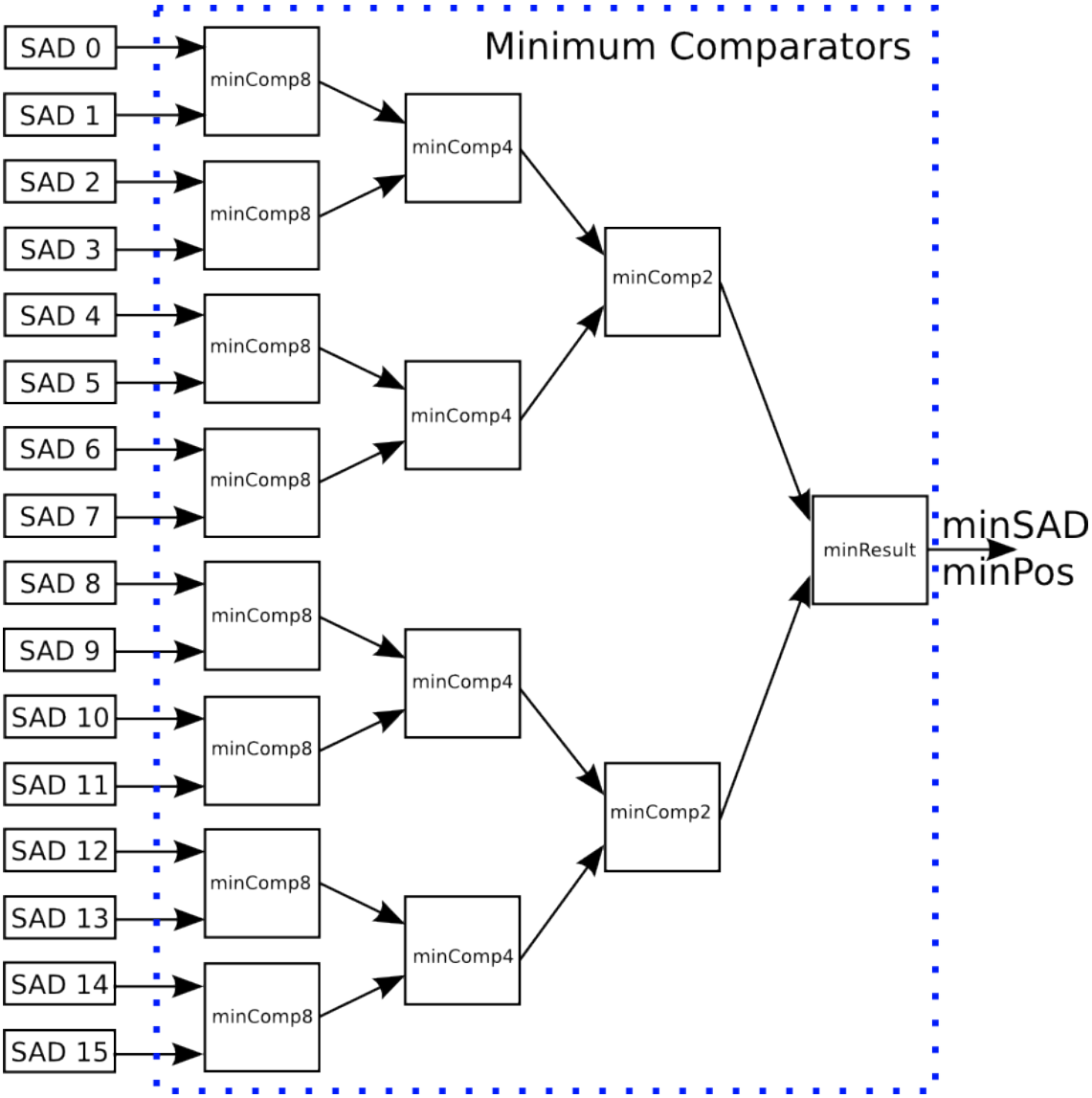


Figure 4.8: The minimum comparator tree designed to quickly find the minimum value and corresponding index out of the 16 SAD values that are calculated for one pixel.

4.1.3 SAD Wrapper

The SAD wrapper is the entity that encompasses the SAD algorithms and minimum comparators. The wrapper receives a clock signal through clk_I and a reset signal through rst_I. It gets the template image data through templ_I and receives the search image data through search_I. The write_t_I and write_s_I notify the wrapper when new data is actively being sent from the template and search images, respectively. It was designed to allow data from both images to be sent to the wrapper in parallel or serially. The SAD wrapper is able to buffer the next pair of row data while the SAD algorithms are running. The h2fReady_I and f2hReady_I are used to communicate when data is ready to be sent between the host and FPGA board. The sw_I signal connects the 8 switches on Atlys board to the wrapper in order to display desired data on the 8 LEDs, led_O. The outputs templ_O for template image region, search_O for search image region, sad_O for the SAD values calculated from the current template and search image regions, and disp_O for the disparity values found from the minimum comparators are able to be read from by the user. In the current implementations, templ_O, search_O, and sad_O were used for debugging purposes only while disp_O was used to create the depth map.

4.1.4 Top Level

Figure 4.10 shows the top most level, the FPGA board itself, and its interaction with the computer. This setup was used to produce the disparity map images in Chapter 5. The computer transferred the pixel data from both images to the FPGA board because current implementations were unable to hold all the image data at one time (see Ch. 5). The FPGA board used the SAD wrapper entity

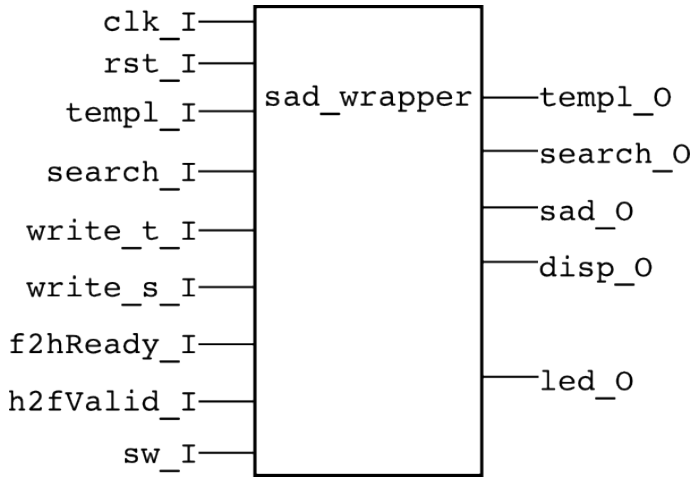


Figure 4.9: The SAD wrapper that encompasses the SAD algorithm and minimum comparator. It interacts with the top level.

within a top level entity. The top level handled the communication between the board and the computer. The board transferred the disparity values to the computer.

The implementation of the SAD wrapper and its internal entities were designed to be able to work with any FPGA that has enough resources to hold it (see Table 5.4). Figure 4.11 shows the SAD wrapper inside a top level entity. The top level gives the SAD wrapper image data and the SAD wrapper sends the top level disparity values. Those values are then transmitted to the computer. See Sec. 4.2 for the communication process. The implementation in Figure 4.11 represents the 9x9 window implementation. For the 7x7 window implementation, there are only 2 SAD and 2 minimum comparators, as opposed to 4 each.

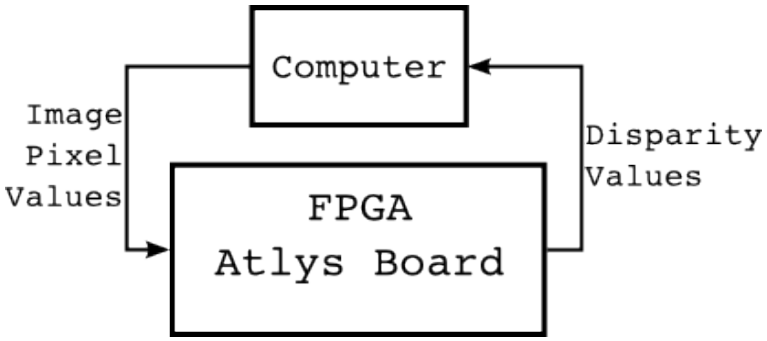


Figure 4.10: The setup for testing the SAD module on the FPGA board. The computer sent the image pixel data to the FPGA board and the FPGA board sent the disparity values to the computer.

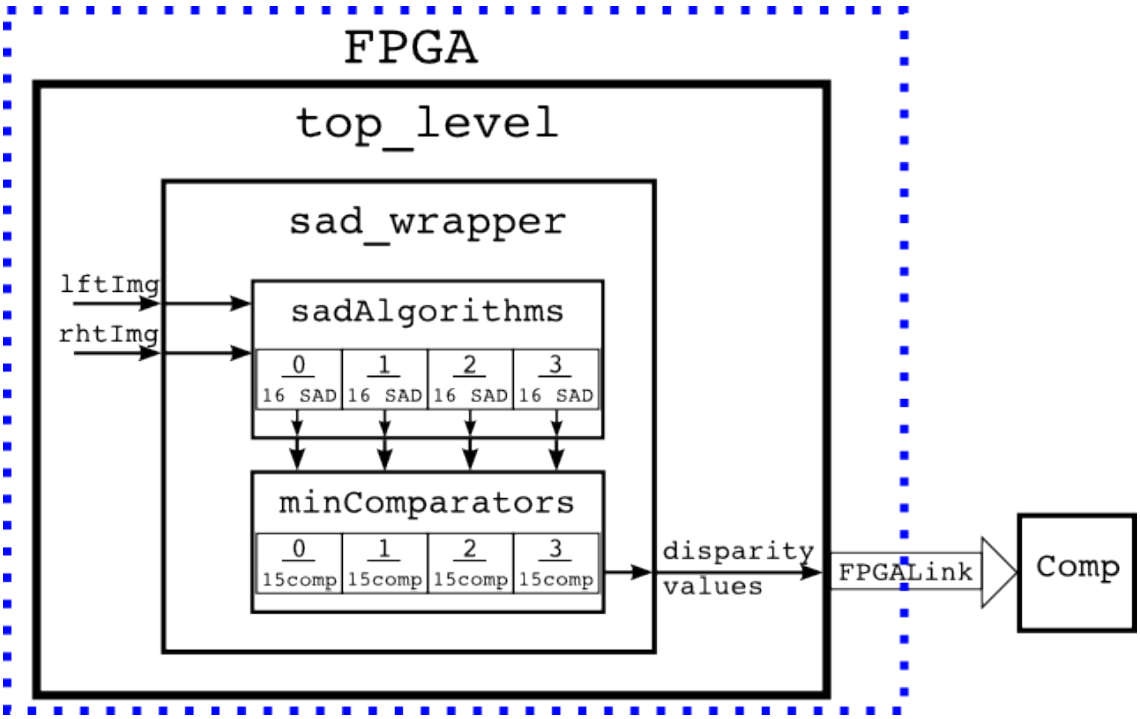


Figure 4.11: The overview of the structure used for implementing the 9x9 window. The 7x7 window has two less SAD and minComp each.

4.2 FPGALink

FPGALink [25] was used to facilitate communications between the computer (host) and the FPGA (Atlys board) over USB. An overview of how the FPGALink works between the host and FPGA is shown in Figure 4.12. The FPGALink has two possible communication modules to choose from, FX2 and EPP. According to [25], FX2 has an observed throughput up to around 26 MB/s, while EPP has a observed throughput up to around 1.26 MB/s. FX2 was used due to its higher throughput.

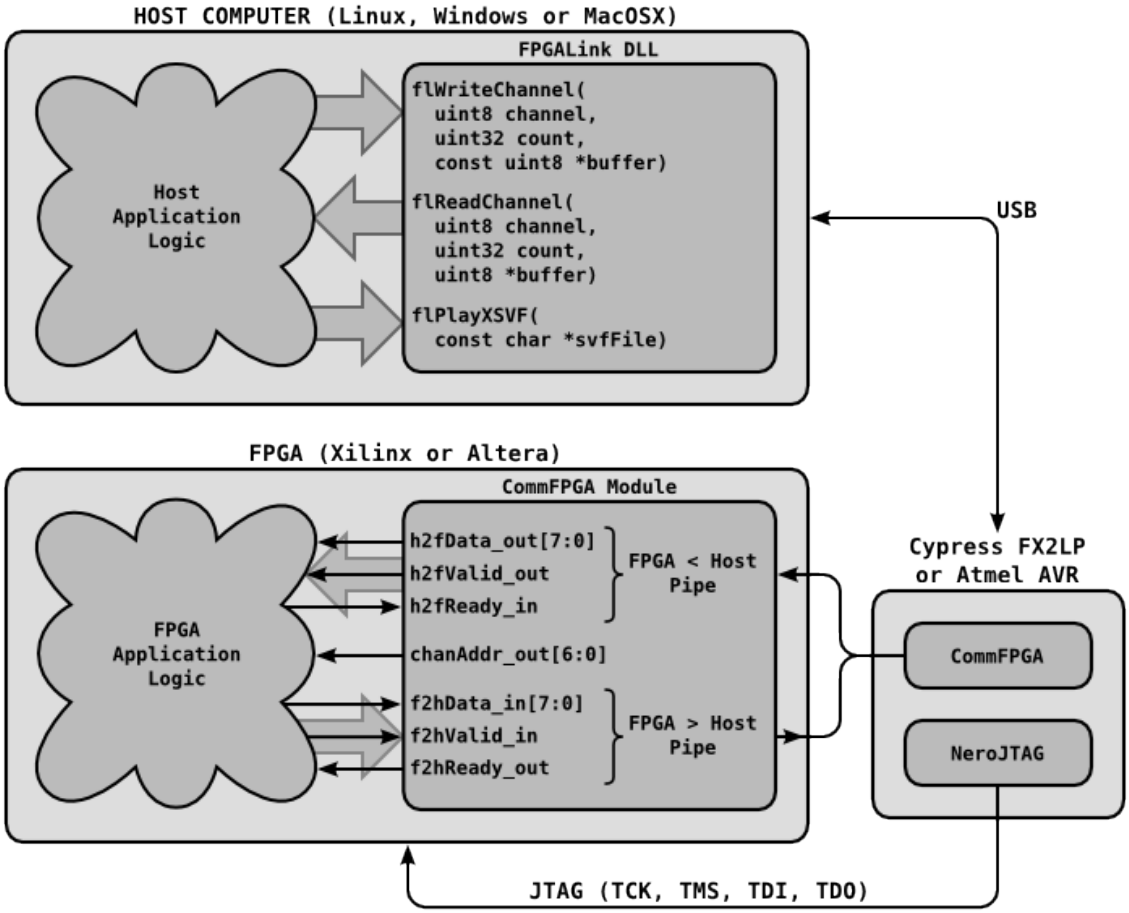


Figure 4.12: Overview of FPGALink communications between host computer and FPGA [25].

Chapter 5

Experiments and Results

This chapter presents the experiments and results of running the SAD algorithm on the FPGA board in comparison to running the algorithm on a general purpose CPU core. The code used on the FPGA board was written in VHDL and the code used on the CPU core was written in C. Disparity map images, testbench simulations, and counting clock cycles were used to compare the accuracy and speed of the VHDL implementations to the C implementation.

5.1 Methodology

The experiments and results presented in this section used the FPGA Atlys board and a desktop computer that has an i7 CPU 950 at 3.07 GHz, 16 GB of RAM, and runs Ubuntu 64-bit. Due to hardware and timing issues with the DDR2 memory chip on the FPGA board, the board was unable to hold all of the data for both images. Part of the rows of both images were sent to the FPGA board from the computer. The board processed the data and sent the disparity values back to the computer. The computer then provided the board with the next set of partial row data and so on until the entire disparity map was created. The images were processed from top to bottom, left to right, where the column width was based on the number of pixels processed in parallel. This was used to test the quality and accuracy of the disparity maps from the FPGA implementation

in comparison to computer implementation in C. The transfer time of sending both images to the FPGA board and getting back the disparity map was not a real-time solution since it took at least 20 seconds to complete the whole process. To test for the maximum possible frames per second the SAD wrapper could perform at, clock cycle counting on the FPGA board and testbench simulations were used to obtain the time it should take to produce a disparity map on the board for a 100 MHz clock. The Atlys board has a 100 MHz clock on it, which determined the clock cycle of 10 ns in the testbench simulations. However, for the experiments that produced the disparity image maps where the image data was sent from the computer to the board, a 48 MHz clock was used, which was the clock frequency of the FPGALink. This was done in order to synchronize the input data with the SAD wrapper.

5.2 Window Size Selection

The size of the window (e.g. 9x9 pixels) affected the quality of the disparity map (see Figure 5.1) and the number of computations required to create the disparity map. The 3x3 window size used in Figure 5.1a was the fastest out of the window sizes shown since each SAD calculation only had 9 pairs of pixels to process. The 13x13 window in Figure 5.1f has 169 pairs of pixels, which required 160 more calculations per SAD value. However, the 13x13 window has the least amount of noise in its disparity map image, but it loses some detail as shown by comparing the neck of the lamp in the foreground of the image to the lamp necks in the other images. Table 5.1 shows the number of pixels needed based on the window size and the number of pixels processed in parallel. As the window size increased, more resources were needed on the FPGA board. The 7x7 and 9x9 window sizes were selected because they provided a good compromise on

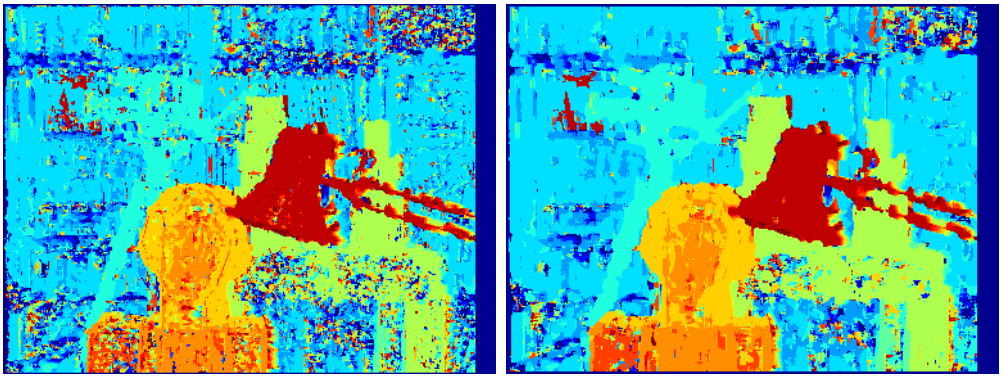
Window Size	**# of pixels/ window**	**# of pixels/ disparity value**	**# of pixels/ 2 disparity values**	**# of pixels/ 4 disparity values**
3x3	9	108	114	126
5x5	25	200	210	230
7x7	**49**	**308**	**322**	**350**
9x9	**81**	**432**	**480**	**486**
11x11	121	572	594	638
13x13	169	728	754	806

Table 5.1: Number of 1 byte pixels based on the window size and number of pixels processed in parallel for producing disparity values simultaneously for a disparity range of 16.

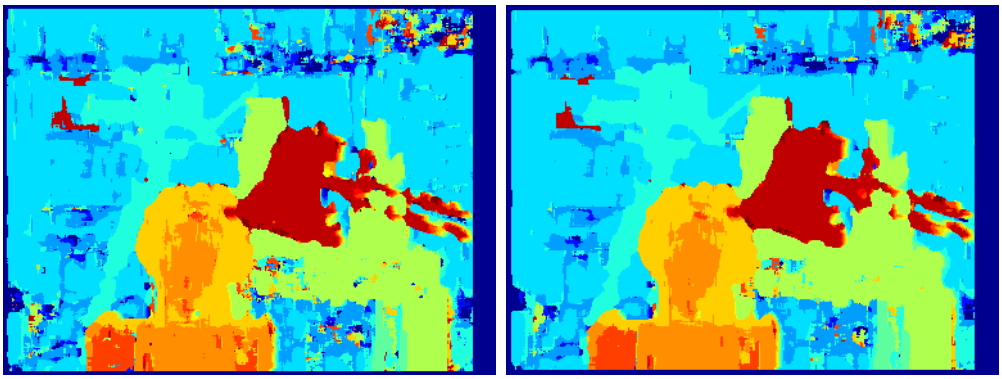
the amount of noise in the disparity maps to the amount of hardware resources required for implementation.

5.3 Resource Utilization on FPGA

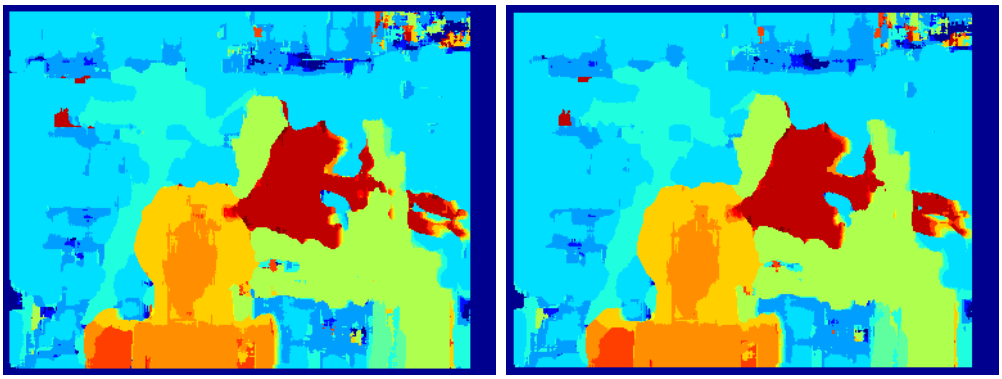
The disparity range used to obtain the disparity value for each pixel affects the number of SAD entities and the number of minimum comparator entities required. Table 5.2 shows the direct correlation between the disparity range and number of those entities needed. A lower disparity range of 8 requires fewer resources, but does not work well for objects that get close to the pair of cameras. A disparity range of 32 will give better results with objects that are closer; however, the resource requirements increase. The disparity range of 16 was selected since it provided a compromise of resource space to minimum detectable object distance.



(a) SAD 3x3 Window Disparity Map (b) SAD 5x5 Window Disparity Map



(c) SAD 7x7 Window Disparity Map (d) SAD 9x9 Window Disparity Map



(e) SAD 11x11 Window Disparity Map (f) SAD 13x13 Window Disparity Map

Figure 5.1: Window size comparisons for disparity maps [11] of the Tsukuba image pair [29].

Disparity Range	**# of SAD entities/ pixel in parallel**	**# of Min. Comp. entities/ pixel in parallel**
8	8	7
16	**16**	**15**
32	32	31

Table 5.2: Number of SAD algorithm entities and minimum comparators entities needed per pixel processed in parallel based on the disparity range.

For a disparity range of 16, Table 5.3 shows the amount of SAD algorithm and minimum comparator entities needed to processing different numbers of pixels in parallel. Processing 2 or 4 pixels in parallel allowed for the speed needed while having a resource utilization size that fits on the Atlys board.

See Table 5.4 for resource utilization. The bold 7x7 and 9x9 rows were the resource utilization of the implementations presented in Chapter 4. The bold 7x7 window implementation used slightly more resources on the FPGA board than the bold 9x9 window implementation due to the additional space requirements of the parallelized SAD algorithms. Both bold implementations used more resources than the other implementations of their same window size. The bold 7x7 implementation processed half the amount of pixels in parallel, but each of its SAD algorithm entities has 7 times as many subtracters than the other 7x7 implementation, which allowed it to run faster while taking up more space. The bold 9x9 implementation processed more pixels in parallel than the other 9x9 implementations and therefore needed more resources. There is plenty of space on the board for other top level entity designs for these SAD modules.

# of pixels in parallel	**# of SAD entities**	**# of Min. Comp. entities**
1	16	15
2	**32**	**30**
4	**64**	**60**
6	96	90

Table 5.3: Number of SAD algorithm and minimum comparator entities needed based on the number of pixels processed in parallel for a disparity range of 16.

Window Size	**# of pixels in parallel**	**SAD Alg. Parallelized**	**# of Slice Registers, out of 54,576**	**# of Slice LUTs, out of 27,288**
7x7	4	No	8,729 (15%)	12,215 (44%)
7x7	**2**	**Yes**	**10,990 (20%)**	**17,329 (63%)**
9x9	**4**	**No**	**10,969 (20%)**	**15,359 (56%)**
9x9	2	No	8,306 (15%)	12,414 (45%)
9x9	1	No	7,108 (13%)	10,426 (38%)

Table 5.4: Resource utilization on the FPGA Atlys board for different window implementations.

5.4 Testbench Simulation

See Figure D.1 and Figure D.2 in Appendix D for the testbench simulations for the 9x9 window and 7x7 window implementations, respectively.

The signal h2fvalid_i, near the top of the figures, went high when image data was sent to the SAD wrapper. For the 9x9 implementation, a cycle was created from the number of clock cycles it took for the SAD algorithms and minimum comparators to run. The signal data_out went low when the SAD algorithms were running and went high after the SAD values were sent to the minimum comparators. For the 7x7 implementation with parallelized SAD, the SAD algorithms and minimum comparators finished in fewer clock cycles than it took to write in the next pair of row data. The cycle was created from the number of clock cycles it took to write in the row data. The SAD wrapper was designed to allow both the template image data and the search image data to be sent to the wrapper at the same time, if desired, thus reducing the amount of time taken to get all necessary data into the wrapper. When the signal f2hready_i goes high, it meant the disparity values were sent out of the SAD wrapper.

For the testbench simulations, it was assumed that the data being sent to the SAD wrapper was supplied when the desired data was needed. So there was no delay to slow the process down. On a full FPGA implementation, the image data would be stored in the DDR2 memory chip on the Atlys board from the cameras and then sent to the SAD wrapper. According to Digilent, the DDR2 can run at up to an 800 MHz data rate [3]. However, for reading data out of the DDR2, it takes around 22 to 32 clock cycles to begin receiving data after sending the read command to it [1]. By configuring the DDR2 as 2 64-bit bi-directional ports, each image would have its own port and data could be clumped together

for 8 bytes of data being read from it at a time. Using several BRAMs within the FPGA chip as an intermediary buffer between the DDR2 and SAD wrapper while reading enough data from the DDR2 at a time should be sufficient to prevent timing delays.

The 9x9 window implementation has its next data rows buffered while the SAD algorithms were running. The SAD algorithms took longer to run than it did to fill up the next rows. This means there were several clock cycles that could be used if the incoming data was slower than ideal speeds.

The 7x7 window implementation with parallelized SAD, for its maximum speed, needed a pixel of data sent into the SAD wrapper every clock cycle. In the simulation, h2fvalid_i was always high, showing that data was always being sent into it.

5.5 Pixel Parallelization

The 7x7 window implementation used a parallelized SAD algorithm (see Section 4.1.1.3). Due to the additional resources needed, only 2 pixels were processed in parallel instead of 4. The reduced number of pixels in parallel was made up for from the speed up of the SAD algorithm. In order to verify the parallelized SAD algorithm produced a speed up greater than reduction of pixels processed in parallel, a 7x7 window implementation of the 9x9 window implementation (see Section 4.1.1.2) was introduced. This version of the 7x7 implementation processed 4 pixels in parallel (4 pix //) and its SAD algorithms were not parallelized.

5.6 FPGA Clock Cycle Runtimes

A clock cycle counter was introduced to 3 implementations, the 9x9 with 4 pixels in parallel, the 7x7 with 2 pixels in parallel and parallel SAD algorithms, and the 7x7 with 4 pixels in parallel (4 pix //). The 7x7 (4 pix //) was tested to demonstrate how the 7x7 with 2 pixels in parallel has a higher frame rate due to the parallelized SAD algorithms.

Table 5.5 shows the recorded clock cycle counts for the Tsukuba (384x288) and Venus (434x383) image pairs for the SAD algorithms and minimum comparators. The VmodCAM (640x480) clock cycles were calculated based on the series of equations in Equation 5.1. For all implementations, the minimum comparators took 4 clock cycles from when it got the 16 SAD values to when it output the disparity value for the lowest SAD value. The 7x7 with parallelized SAD has additional clock cycles due to how fast its SAD algorithms ran in comparison to data load times of the next row into the SAD wrapper.

Eq. 5.1 produced the same SAD Cycles and Min. Comp. Cycles as the values recorded from the FPGA in Tbl. 5.1 for the Tsukuba and Venus image pairs. Eq. 5.1 also was used to calculate the total cycles required per iteration. Testbench simulations in Appdx. D were used to determine the additional clock cycles needed per iteration. The simulations also verified the number of clock cycles required for data input, SAD algorithm, and minimum comparators.

Equations (5.1a) to (5.1g) are the parameters for the rest of the equations. Equation (5.1h) gives the height of the disparity map image. The width of the disparity map image, Equation (5.1j), needs to be divisible by Equation (5.1e), so Equation (5.1i) is used to reduce it to a divisible amount. The number of pixels in the disparity map image is calculated in Equation (5.1k). The base number

of iterations, Equation (5.1l), is the minimum number of iterations required to create the disparity map. Additional iterations, Equation (5.1n), occur from the SAD algorithm proceeding through the images in the form of columns, top to bottom, left to right. There are a certain amount of iterations at the top of each column that are not used because a winSize of rows must be loaded in order to obtain the disparity values. Equation (5.1o) is the total number of iterations needed to process the pair of images. Equation (5.1p) is the total number of cycles for the SAD algorithms. Equation (5.1q) is the total number of cycles for the minimum comparators. Equation (5.1r) represents the extra cycles that occur only for the 7x7 implementation with the parallelized SAD algorithms. For that implementation, the SAD algorithm is faster than the data can be loaded, so its iterations take longer than just the number of SAD algorithm and minimum comparator cycles. Equation (5.1s) is the total number of cycles for the entire pair of images to be processed on the FPGA.

The 9x9 implementation was able to have its next rows of 27 pixels each buffered within the 82 clock cycles it took for the SAD algorithms to finish. There were a total of 86 clock cycles per iteration.

The 7x7 implementation with parallelized SAD was unable to have its next rows, 23 pixels each, buffered within the 8 clock cycles it took for the SAD algorithms to finish. The limiting factor was the time it took to buffer each pair of rows, so there were 23 clock cycles per iteration. The 4 clock cycles for the minimum comparators was hidden within the 23 clock cycles.

$$height = 288 \quad \text{(image height)} \tag{5.1a}$$

$$width = 384 \quad \text{(image width)} \tag{5.1b}$$

$$winSize = 9 \quad \text{(9x9 window size)} \tag{5.1c}$$

$$dispRange = 16 \quad \text{(disparity range 0-15)} \tag{5.1d}$$

$$parPix = 4 \quad \text{(pixels processed in parallel)} \tag{5.1e}$$

$$sadCyc = 82 \quad \text{(\# of cycles for SAD algorithm)} \tag{5.1f}$$

$$minCyc = 4 \quad \text{(\# of cycles for minimum comparator)} \tag{5.1g}$$

$$dispH = 280 = height - (winSize - 1) \tag{5.1h}$$

$$lessPix = 23 = (dispRange - 1) + (winSize - 1) \tag{5.1i}$$

$$dispW = 360 = (width - lessPix) - (width - lessPix)\%parPix \tag{5.1j}$$

$$dispPixels = 100,800 = dispH * dispW \tag{5.1k}$$

$$baseIters = 25,200 = dispPixels/parPix \tag{5.1l}$$

$$colAdd = 89 = dispWidth/parPix - 1 \tag{5.1m}$$

$$addIters = 712 = colAdd * (winSize - 1) \tag{5.1n}$$

$$totIters = 25,912 = basIters + addIters \tag{5.1o}$$

$$sadTotCyc = 2,124,784 = sadCyc * totIters \tag{5.1p}$$

$$minTotCyc = 103,648 = minCyc * totIters \tag{5.1q}$$

$$extraCyc = 0 = totIters * 0 \tag{5.1r}$$

$$totClkCyc = 2,228,432 = sadTotCyc + minTotCyc + extraCyc \tag{5.1s}$$

Image Size (WxH)	**Disparity Image Size**	**Window Size**	**SAD Cycles**	**Min. Comp. Cycles**	**Extra Cycles**	**Total Clock Cycles**
384x288	360x280	9x9	2,124,784	103,648	0	2,228,432
384x288	362x282	7x7	416,976	208,488	573,342	1,198,806
384x288	360x282	7x7 (4 pix //)	1,295,700	103,656	0	1,399,356
434x383	408x375	9x9	3,202,756	156,232	0	3,358,988
434x383	412x377	7x7	631,136	315,568	867,812	1,814,516
434x383	412x377	7x7 (4 pix //)	1,972,150	157,772	0	2,129,922
640x480	616x472	9x9	6,060,784	295,648	0	6,356,432
640x480	618x474	7x7	1,186,512	593,256	1,631,454	3,411,222
640x480	616x474	7x7 (4 pix //)	3,695,700	295,656	0	3,991,356

Table 5.5: Number of clock cycles counted when a pair of images were processed on the FPGA for the SAD algorithms and the minimum comparators.

The 7x7 (4 pix //) implementation was able to have its next rows of 25 pixels each buffered within the 50 clock cycles it took for the SAD algorithms to finish. There were a total of 54 clock cycles per iteration.

Table 5.6 is a continuation of Tbl. 5.5. It shows the time it took to process a frame and the frame rate with a clock speed of 48 MHz and 100 MHz. The 48 MHz was the transfer rate of the FPGALink between the computer and the

Window Size	**Total Cycles**	**Sec/ Frame @ 48 MHz**	**Sec/ Frame @ 100 MHz**	**Frames/ Sec @ 48 MHz**	**Frames/ Sec @ 100 MHz**
9x9	2,228,432	0.04642	0.02228	21.54	44.87
7x7	1,198,806	0.02497	0.01199	40.05	83.42
7x7 (4 pix //)	1,399,356	0.02915	0.01399	34.31	71.46
9x9	3,358,988	0.06997	0.03359	14.29	29.77
7x7	1,814,516	0.03780	0.01815	26.46	55.11
7x7 (4 pix //)	2,129,922	0.04437	0.02130	22.54	46.95
9x9	6,356,432	0.1324	0.06356	7.553	15.73
7x7	3,411,222	0.06578	0.03411	14.07	29.32
7x7 (4 pix //)	3,991,356	0.08314	0.03991	12.03	25.05

Table 5.6: Frame rates that are possible for the number of clock cycles taken per image.

FPGA board. It was used in order to not introduce timing issues for the data given to the SAD wrapper. The image processing on the FPGA board always takes a consistent amount of clock cycles, relative to the image sizes, so the frame rate can be calculated for a clock speed of 100 MHz, which is the clock frequency of the clock on the Atlys board.

5.6.1 Frame Rate

The smaller the image size, the higher the frame rate, as shown in Figure 5.2. Once the number of pixels in an image goes below 300,000 for the 7x7 window implementation or 170,000 for the 9x9 window implementation, the frame rate reaches 30 frames per second. For robots, a frame rate of 10 should be sufficient for most tasks. Both the 9x9 and 7x7 window implementations were shown be above 10 frames per second for an image size of 640x480. Therefore it is possible to use larger image sizes and still get at least 10 frames per second.

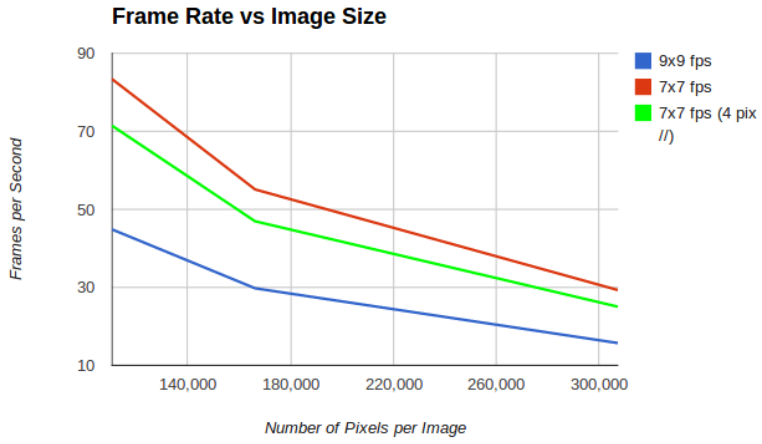


Figure 5.2: Frame rate comparison of different image sizes.

5.7 Test Image Pairs

In this section, FPGA disparity maps were compared to disparity maps created using C code. Part of the SAD algorithm implementation in C is shown in Appendix E. The C SAD version was performed completely in serial, so 1 pixel was processed at a time. The images the C version produced were used to compare disparity map quality and runtime of the VHDL algorithm for the FPGA board.

Python was used to convert the grayscale images into text files. Each row was separated by a new line. Each column was separated by a blank space. The C code read in the data from the text files, performed the SAD algorithm on the data, and wrote the disparity map data to a text file. The disparity map text file was read by another Python script and converted into a disparity map image. The time comparisons focused on the total time it took the SAD algorithm to run and disparity map data to be generated from the SAD values.

Table 5.7 shows the frames per second (FPS) comparisons for the Tsukuba and Venus image pairs between C and VHDL implementations. The C code was compiled with gcc using optimization O2 and ran on a single processor core. The 7x7 window implementation with parallelized SAD and 9x9 window implementation are the 7x7 and 9x9 shown for the VHDL code in the table. The 7x7 VHDL implementation averaged around 12.80 times faster than the C 7x7 version. The 9x9 VHDL implementation was around 11.12 times faster than the C 9x9 version.

5.7.1 Data Overflows

The size of the data used for storing logic and values in hardware was defined during the coding process. In the SAD algorithm, it was possible for the SAD value to become much greater than the individual pixel values. For example, the pixel values range from 0 to 255, or 8 bits, while some SAD values could be over 4,095 and need to be stored in more than 12 bits. Most SAD values were under 4,096, so the SAD algorithm used 14 bits to account for any values from 0 to 16,383. Figure 5.3 shows what can happen when the data size allotted for the SAD algorithm was not large enough (e.g. only having 10 bits for storage). The data used was unsigned, so when it went above the highest supported value, it went back down to 0 and continued from there.

Image	Window Size	Code	Sec/frame	FPS	Speed up
Tsukuba	7x7	C	0.1532	6.527	1
Tsukuba	7x7	VHDL	0.01199	83.42	12.78
Tsukuba	9x9	C	0.2454	4.075	1
Tsukuba	9x9	VHDL	0.02228	44.87	11.01
Venus	7x7	C	0.2327	4.297	1
Venus	7x7	VHDL	0.01815	55.11	12.83
Venus	9x9	C	0.3776	2.648	1
Venus	9x9	VHDL	0.03359	29.77	11.24

Table 5.7: Tsukuba and Venus image pairs comparison runtimes for C code and FPGA testbench simulations. The disparity range is 16 for both.

Since most of the values were below 4,096, a measure was put in place to reduce the amount of bits needed during the minimum comparisons. If a SAD value was greater than 4,095, then 4,095 was returned for that calculated SAD value because the greater that the value is the less likely that search pixel will be the matching pixel to the corresponding template pixel. In Figure 5.4 and Figure 5.5, there are a total of 64 pixels that differ between all of the disparity map images. All of the differing pixels were in the 9x9 Venus disparity image, which were from those SAD values being over 4,095 and were handled accordingly. In the disparity map images, the objects closer to the cameras have a warmer color while objects farther away have a cooler color.

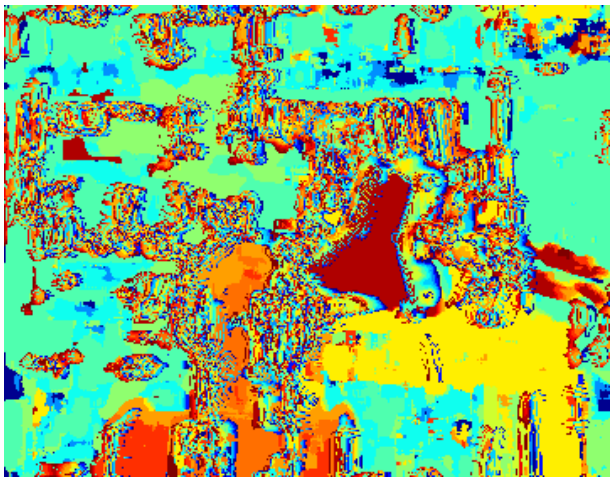


Figure 5.3: Data overflow for Tsukuba image pair [29].

5.7.2 Tsukuba

In Figure 5.4a and Figure 5.4b, the Tsukuba image pair is shown. Figure 5.4 shows how the 7x7 window implementation is slightly noisier than the 9x9 window implementation. There were no differences between the corresponding disparity maps of the C code to the VHDL code.

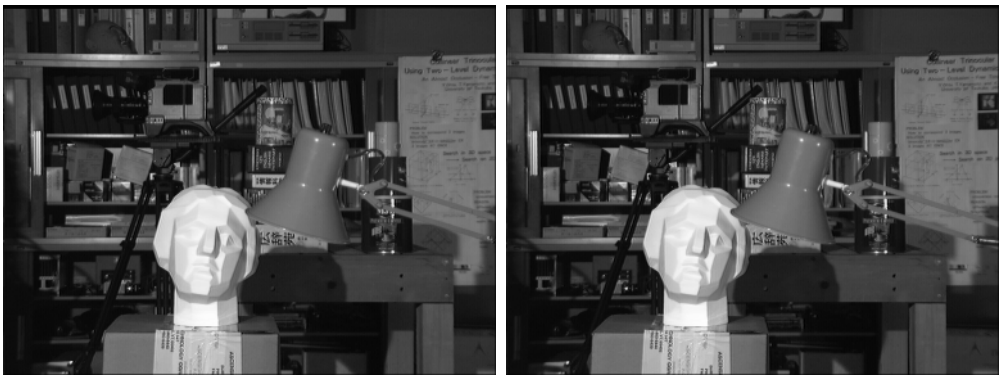
5.7.3 Venus

In Figure 5.5a and Figure 5.5b, the Venus image pair is shown. In the image pair, the newspaper articles are flat and slanted, relative to the cameras. This gradual slope, also present in the background, can be difficult for the SAD algorithm to deal with; however, the algorithm was able to give a fairly accurate representation of the depth in the image. It also caused the gradient pattern shown in the disparity maps. The 7x7 window depth maps have more noise than the 9x9 window depth maps. The only difference between the corresponding disparity maps of the C code to the VHDL code were with the 9x9 implementations. There

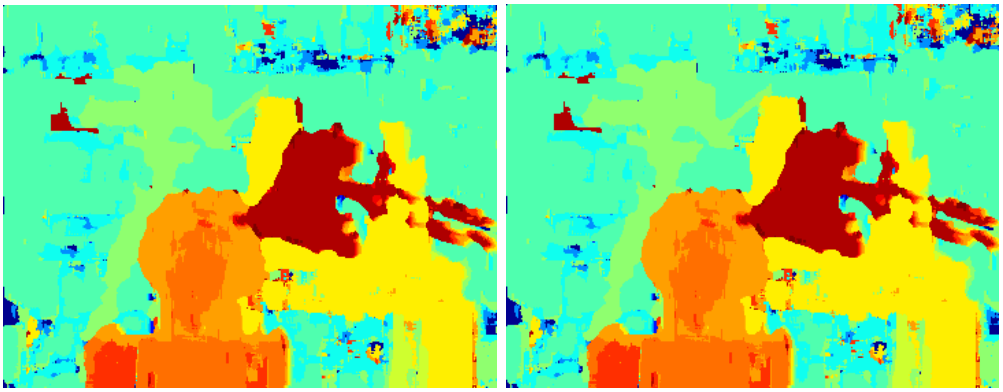
were 64 pixels that differed between the C and FPGA 9x9 implementations. The SAD values that correspond to those pixels on the FPGA were above 4,095, so they were cut off at 4,095. The 64 pixels out of 155,324 pixels only gives 0.0412% error.

5.7.4 Cones

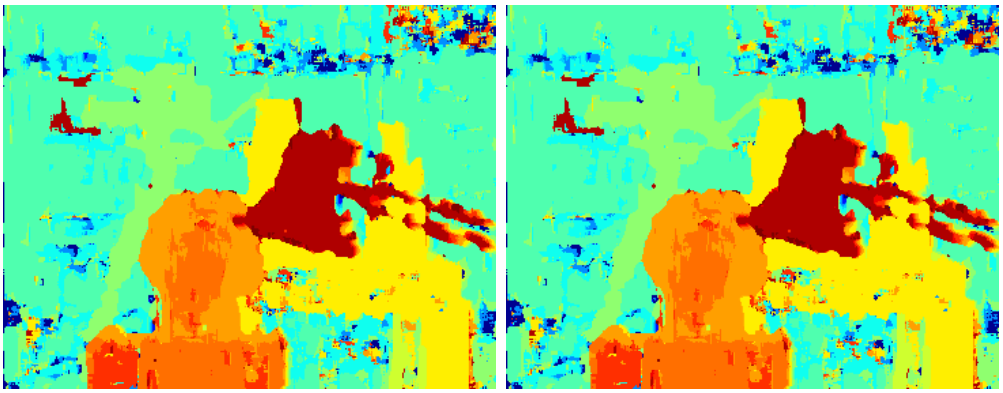
In Figure 5.6a and Figure 5.6b, the Cones image pair is shown. Figure 5.6 shows the issue of objects being too close to the stereo cameras. The closer an object is to the stereo cameras, the greater its disparity value is and the greater the disparity range needs to be. Using the SAD algorithm with a 9x9 window and a disparity range of 60 (as opposed to the range of 16 used on the FPGA board) produces the results in Figure 5.6c. When the disparity range is not high enough, the disparity map in Figure 5.6d is produced.



(a) Left Tsukuba Grayscale Image (b) Right Tsukuba Grayscale Image



(c) C 9x9 Disparity Map (d) FPGA 9x9 Disparity Map

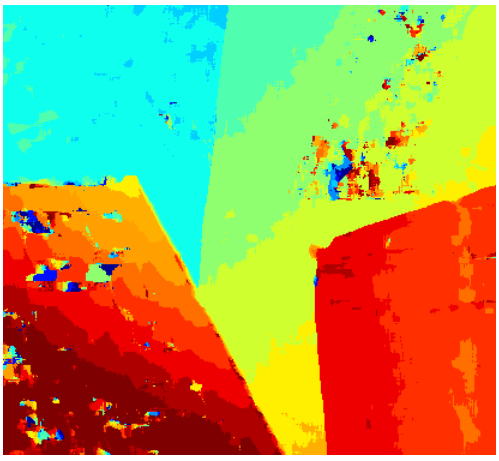


(e) C 7x7 Disparity Map (f) FPGA 7x7 Disparity Map

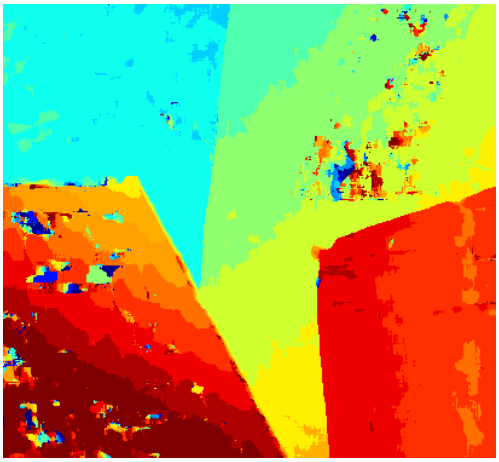
Figure 5.4: Disparity map comparison of the Tsukuba image pair [29].

(a) Left Venus Grayscale Image

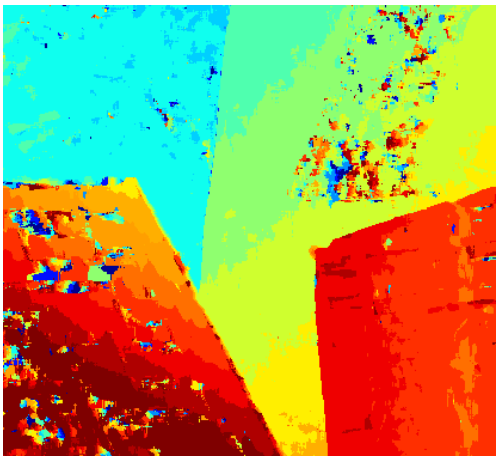
(b) Right Venus Grayscale Image



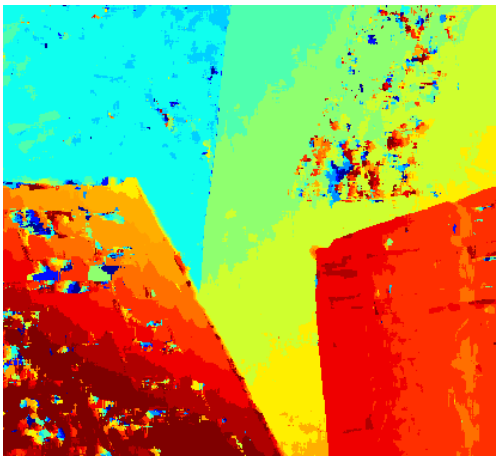
(c) C 9x9 Disparity Map



(d) FPGA 9x9 Disparity Map



(e) C 7x7 Disparity Map

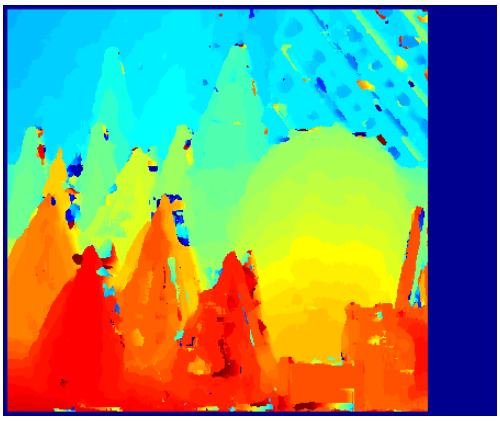


(f) FPGA 7x7 Disparity Map

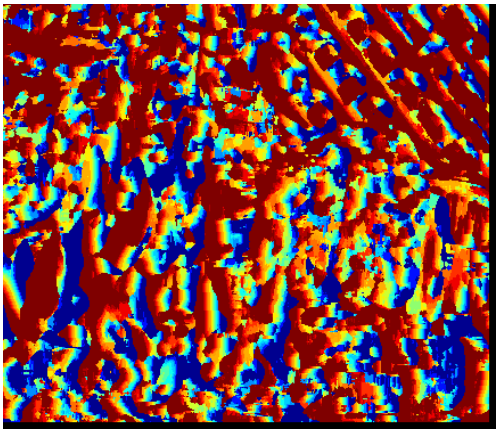
Figure 5.5: Disparity map comparison of the Venus image pair [29].

(a) Left Cones Grayscale Image

(b) Right Cones Grayscale Image



(c) 9x9 at Disparity Range of 60 [11]



(d) 9x9 at Disparity Range of 16

Figure 5.6: Disparity map comparison of the Cones image pair [29].

Chapter 6

Conclusions

For image processing, the more operations that can be parallelized, the faster an image can be processed. However, as parallelism is increased, the amount of hardware required is also increased. It could be possible to parallelize a SAD algorithm to the point where it only takes a few clock cycles to process the whole image (i.e. every SAD calculation for an image pair occurring simultaneously). Unfortunately, the area required on an FPGA would be a lot more than what was implemented in this paper, especially since the implementation in this paper was only able to process up 4 pixels simultaneously. The hardware cost to obtain the higher levels of FPGAs would be very cost prohibitive and not something a club or hobbyist could readily use for a robotics project. There does come a point where the frames per second of disparity maps exceeds the rate the other parts of the robot can process, which is an unnecessary cost. So the FPGA board only needs to be able to handle a SAD implementation up to a certain frame rate and image quality, which depends on the requirements of the application for the robot.

Between the 9x9 window implementation and the 7x7 window implementation with parallelized SAD, unless a higher frame rate is needed, the 9x9 window is better than the 7x7 window. While 7x7 has a higher frame rate, 9x9 produces a better quality disparity map with less noise and requires fewer hardware resources. The 7x7 implementation with parallelized SAD only processed 2 pixels

in parallel, but it was around 17% faster than the 7x7 implementation without parallelized SAD that processed 4 pixels in parallel. The 9x9 window implementation processed 4 pixels in parallel and its SAD algorithms were not internally parallelized.

The number of clock cycles required to process an image for the different implementations were obtained from the FPGA board and compared to the number of clock cycles measured from testbench simulations. The number of clock cycles matched up and showed that the number of cycles per iteration were calculable. On the Atlys board, the VmodCAM can supply the board with an image size of 640x480 pixels. It has been shown that the 9x9 implementation was able to process an image pair of that size at up to 15.73 frames per second. The 7x7 implementation with parallelized SAD was shown to be able to process that image pair size at up to 29.32 frames per second. The 7x7 frame rate could be higher if the data supplied to the SAD wrapper was further parallelized to decrease the number of clock cycles required to send data to it.

This modular implementation of the SAD algorithm has the potential to be used for FPGA implementations in autonomous mobile robotic applications.

Chapter 7

Future Work

The next steps are to get a fully functional stereo vision implementation on the Atlys board that uses the SAD module presented in this paper. The Atlys board has a 1 Gbit DDR2 memory chip, which could be used to buffer the images from the VmodCAM stereo camera module [3]. The left and right images from the VmodCAM could be buffered to the DDR and then sections of the buffered images could be sent to the SAD module to obtain the disparity values. With the correct timing and buffering, both or one of the images and the disparity map can then be sent off board to a computer on a robot to use the image and depth data to navigate and interact with the world.

After a fully functional implementation on the Atlys board is working, a custom FPGA board could be designed and manufactured. The custom board only needs the functionalities of the Atlys board in order to communicate with the computer, obtain images from the stereo cameras, buffer the images, and process the images on the FPGA IC. A custom board without the extra peripherals on the Atlys board has the potential to further reduce the cost of a stereo vision FPGA board. Also, the stereo cameras could be built into the board to reduce the cost of hardware needed for connections.

Furthermore, replacing the FPGA IC used on the Atlys board with one having a clock frequency higher than 100 MHz or more space while keeping the cost of

the IC around the same price range as the Atlys board FPGA IC is a way to speed up the SAD calculation time and increase the frame rate.

When all is said and done, having robots readily able to have better and less expensive "eyes" to perceive the world around them in greater depth will allow for more practical applications, uses, experiments, and expansion of our knowledge in this growing field.

Bibliography

[1] Spartan-6 FPGA Memory Controller. www.xilinx.com/support/documentation/user_guides/ug388.pdf, August 2010.

[2] 3D Imaging with NI LabVIEW. http://www.ni.com/white-paper/14103/en/, August 2013.

[3] Atlys Spartan-6 FPGA Development Board. http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,400,836&Prod=ATLYS&CFID=5602753&CFTOKEN=ff475ab97d889237-27A8B1C4-5056-0201-02F29D3CC5564ED6, July 2014.

[4] Bumblebee2. http://ww2.ptgrey.com/stereo-vision/bumblebee-2, July 2014.

[5] Digi-Key. http://www.digikey.com/product-detail/en/DK-DEV-4SGX230N/544-2594-ND/2054809?cur=USD, July 2014.

[6] Digi-Key. http://www.xilinx.com/products/boards_kits/virtex5.htm, July 2014.

[7] VmodCAM - Stereo Camera Module. http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,648,931&Prod=VMOD-CAM, July 2014.

[8] XtremeDSP Starter Platform Spartan-3A DSP 1800A Edition.

http://www.xilinx.com/products/boards-and-kits/HW-SD1800A-DSP-SB-UNI-G.htm, July 2014.

[9] Epipolar geometry. http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/OWENS/LECT10/node3.html, Accessed July 18, 2014.

[10] http://web.cecs.pdx.edu/~mperkows/CLASS_VHDL/VHDL_CLASS_2001/Safranski_Alex/HW1.htm, Accessed July 29, 2014.

[11] Siddhant Ahaja. Correlation based similarity measures-Sum of Absolute Differences (SAD). https://siddhantahuja.wordpress.com/2009/05/11/correlation-based-similarity-measures-in-area-based-stereo-matching-systems/, May 2009.

[12] K. Al Mutib, M. Al Sulaiman, H. Ramdane, M.M. Emaduddin, and E.A. Mattar. Active Stereo Vision for Mobile Robot Localization and Mapping Path Planning. In *Computer Modelling and Simulation (UKSim), 2012 UKSim 14th International Conference on*, pages 293–299, 2012.

[13] P. Ben-Tzvi and Xin Xu. An embedded feature-based stereo vision system for autonomous mobile robots. In *Robotic and Sensors Environments (ROSE), 2010 IEEE International Workshop on*, pages 1–6, Oct 2010.

[14] M.Z. Brown, D. Burschka, and G.D. Hager. Advances in computational stereo. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(8):993–1008, Aug 2003.

[15] P. Corke and P. Dunn. Real-time stereopsis using FPGAs. In *TENCON '97. IEEE Region 10 Annual Conference. Speech and Image Technologies for Computing and Telecommunications., Proceedings of IEEE*, volume 1, pages 235–238 vol.1, 1997.

[16] Jingting Ding, Xin Du, Xinhuan Wang, and Jilin Liu. Improved real-time correlation-based FPGA stereo vision system. In *Mechatronics and Automation (ICMA), 2010 International Conference on*, pages 104–108, Aug 2010.

[17] M. Gosta and M. Grgic. Accomplishments and challenges of computer stereo vision. In *ELMAR, 2010 Proceedings*, pages 57–64, Sept 2010.

[18] M. Hariyama, N. Yokoyama, M. Kameyama, and Y. Kobayashi. FPGA implementation of a stereo matching processor based on window-parallel-and-pixel-parallel architecture. In *Circuits and Systems, 2005. 48th Midwest Symposium on*, pages 1219–1222 Vol. 2, Aug 2005.

[19] Li He-xi, Wang Guo-rong, and Shi Yong-hua. Application of Epipolar Line Rectification to the Stereovision-Based Measurement of Workpieces. In *Measuring Technology and Mechatronics Automation, 2009. ICMTMA '09. International Conference on*, volume 2, pages 758–762, April 2009.

[20] N. Isakova, S. Basak, and A.C. Sonmez. Fpga design and implementation of a real-time stereo vision system. In *Innovations in Intelligent Systems and Applications (INISTA), 2012 International Symposium on*, pages 1–5, 2012.

[21] N. Isakova, S. Basak, and AC. Sonmez. FPGA design and implementation of a real-time stereo vision system. In *Innovations in Intelligent Systems and Applications (INISTA), 2012 International Symposium on*, pages 1–5, July 2012.

[22] Seunghun Jin, Junguk Cho, Xuan Dai Pham, Kyoung-Mu Lee, Sung-Kee Park, Munsang Kim, and J.W. Jeon. FPGA Design and Implementation

of a Real-Time Stereo Vision System. *Circuits and Systems for Video Technology, IEEE Transactions on*, 20(1):15–26, Jan 2010.

[23] J. Y S Luh. Industrial robot that moves around unexpected obstacles with a minimum distance. In *Decision and Control, 1983. The 22nd IEEE Conference on*, volume 22, pages 1458–1463, 1983.

[24] J. Y S Luh and John A. Klaasen. A Three-Dimensional Vision by Off-Shelf System with Multi-Cameras. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-7(1):35–45, 1985.

[25] Chris McClelland. FPGALink: Easy USB to FPGA Communication. `http://www.makestuff.eu/wordpress/software/fpgalink/`, March 2011.

[26] C. Murphy, D. Lindquist, AM. Rynning, Thomas Cecil, S. Leavitt, and M.L. Chang. Low-Cost Stereo Vision on an FPGA. In *Field-Programmable Custom Computing Machines, 2007. FCCM 2007. 15th Annual IEEE Symposium on*, pages 333–334, April 2007.

[27] G. Rematska, K. Papadimitriou, and A Dollas. A low cost embedded real time 3D stereo matching system for surveillance applications. In *Bioinformatics and Bioengineering (BIBE), 2013 IEEE 13th International Conference on*, pages 1–6, Nov 2013.

[28] D. Scharstein, R. Szeliski, and R. Zabih. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. In *Stereo and Multi-Baseline Vision, 2001. (SMBV 2001). Proceedings. IEEE Workshop on*, pages 131–140, 2001.

[29] Daniel Scharstein and Richard Szeliski. Stereo. `http://vision.middlebury.edu/stereo/`, April 2012.

[30] Xuesong Shao, Yiping Yang, and Wei Wang. Obstacle crossing with stereo vision for a quadruped robot. In *Mechatronics and Automation (ICMA), 2012 International Conference on*, pages 1738–1743, 2012.

[31] Zhao Yong-guo, Cheng Wei, and Liu Guang-liang. The Navigation of Mobile Robot Based on Stereo Vision. In *Intelligent Computation Technology and Automation (ICICTA), 2012 Fifth International Conference on*, pages 670–673, 2012.

Appendix A

Absolute Difference 9x9 Window Code Snippet

```
-- Assign greater value to more and smaller to less
IF (search_window(ndx) < template_window(ndx)) THEN
    more <= template_window(ndx);
    less <= search_window(ndx);
ELSE
    less <= template_window(ndx);
    more <= search_window(ndx);
END IF;

-- Subtraction IP CORE, sub = more - less
subber : subtr_core
   PORT MAP (
       a => more,
       b => less,
       s => sub
    );
```

Appendix B

Absolute Difference 7x7 Window Code Snippet

```
-- Assign greater value to more and smaller to less
-- Loop is unrolled in hardware, 7 assignments occur simultaneously
FOR i IN 0 TO 6 LOOP
    IF (search_window(ndx+(7*i)) < template_window(ndx+(7*i))) THEN
        more(i) <= template_window(ndx + (7*i));
        less(i) <= search_window(ndx + (7*i));
    ELSE
        less(i) <= template_window(ndx + (7*i));
        more(i) <= search_window(ndx + (7*i));
    END IF;
END LOOP;

-- Subtraction IP CORE, sub(i) = more(i) - less(i)
g_differ_10 : FOR i IN 0 TO 6 GENERATE
    i_subber : adder_10
        PORT MAP (
            a => more(i),
            b => less(i),
            s => sub(i)
        );
END GENERATE g_differ_10;
```

Appendix C

Minimum Comparator Code

```
-- Constantly assign inputs
sad0 <= sad0_I;
pos0 <= pos0_I;
sad1 <= sad1_I;
pos1 <= pos1_I;

-- Comparison
PROCESS(clk_I)
begin
    IF (RISING_EDGE(clk_I)) THEN
        IF (sad1 < sad0) THEN
            sad_out <= sad1;
            pos_out <= pos1;
        ELSE
            sad_out <= sad0;
            pos_out <= pos0;
        END IF;
    END IF;
END PROCESS;

-- Constantly assign outputs
sad_O <= sad_out;
pos_O <= pos_out;
```

Appendix D

Testbench Simulations

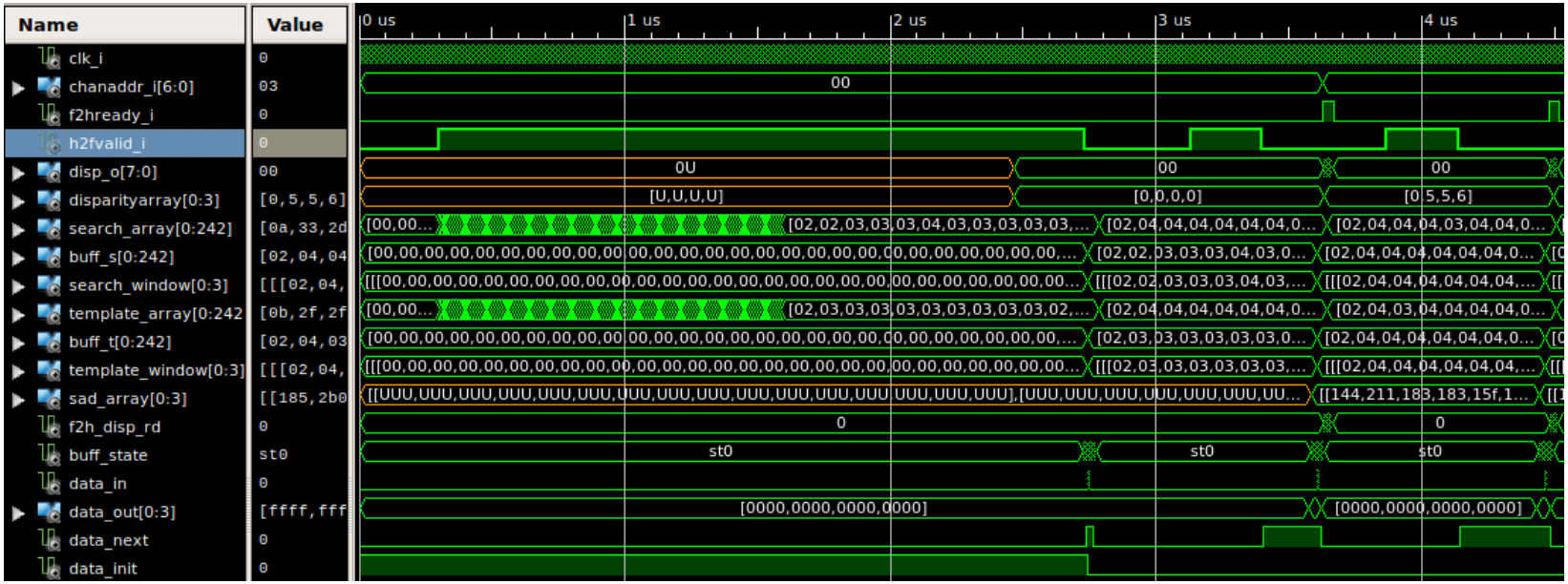


Figure D.1: Testbench simulation for the 9x9 window implementation.

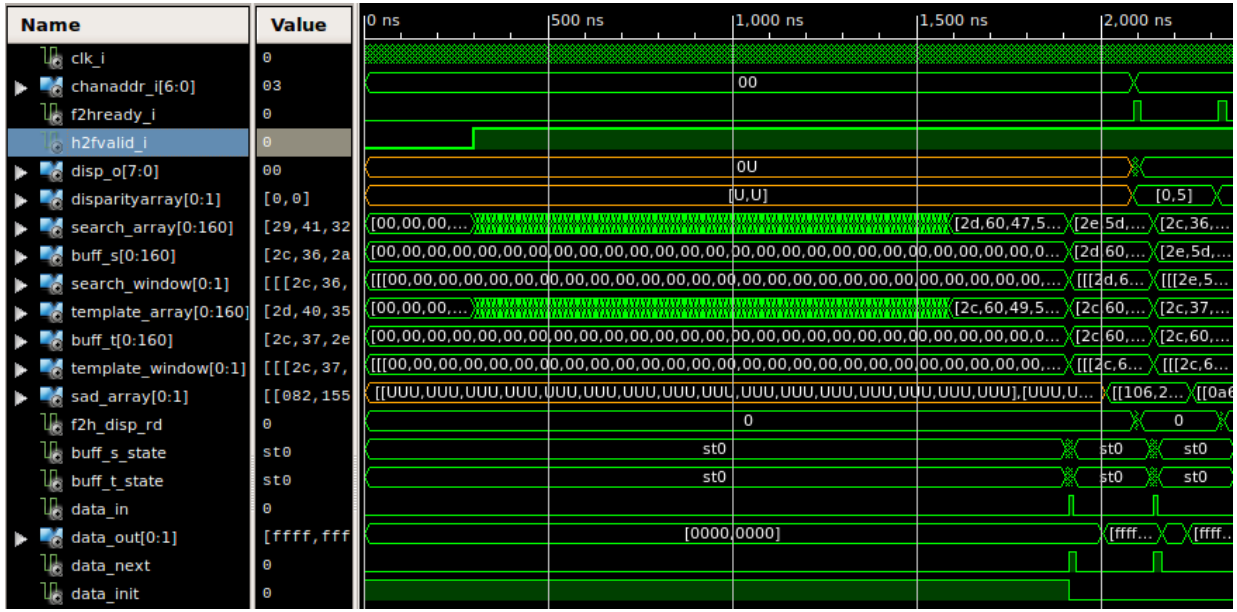


Figure D.2: Testbench simulation for the 7x7 window implementation.

Appendix E

C Serial SAD Algorithm

```
// SAD Algorithm Code Snippet
for (i = 0; i < dispH; i++) {
    for (j = 0; j < dispW; j++) {
        memset(sadArray, 0, sizeof(int) * dispRange);
        for (k = 0; k < dispRange; k++) {
            for (m = -win; m <= win; m++) {
                for (n = -win; n <= win; n++)
                    sadArray[k] += abs(arrR[i+m+win][n+j+win] -
                        arrL[i+m+win][n+j+k+win]);
            }
        }
        minPos = 0;
        minVal = sadArray[0];
        for (pos = 1; pos < dispRange; pos++)
            if (sadArray[pos] < minVal) {
                minVal = sadArray[pos];
                minPos = pos;
            }
        arrDisp[i][j] = minPos;
    }
}
```

The full code for the C SAD Algorithm can be found on github:

`https://github.com/cccitron/mastersThesis/tree/master/pythonSAD`