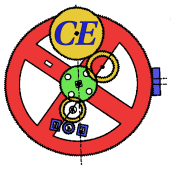

Computer Engineering
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ce.et.tudelft.nl/

CE-MS-2010-11

M.Sc. Thesis

Design and Implementation of Real-Time High-Definition Stereo Matching SoC on FPGA

Lu Zhang B.Sc.

Abstract

Stereo matching has been widely used in many fields, such as viewpoint interpolation, feature detection system and free-view TV. However, the long processing time of stereo matching algorithms has been the major bottleneck that limits their real-time applications. During the past decades, many implementation platforms and corresponding algorithm adaptations are proposed to solve the processing time problem. Although notable real-time performances have been achieved, these works rarely satisfy both real-time processing and high stereo matching quality requirements.

In this thesis, we propose an improved stereo matching algorithm suitable for hardware implementation based on the VariableCross and the MiniCensus algorithm. Furthermore, we provide parallel computing hardware design and implementation of the proposed algorithm. The developed stereo matching hardware modules are instantiated in an SoC environment and implemented on a single EP3SL150 FPGA chip. The experimental results suggest that our work has achieved high speed real-time processing with programmable video resolutions, while preserving high stereo matching accuracy. The online benchmarks also prove that this work delivers leading matching accuracy among declared real-time implementations, with only 8.2% averaged benchmark error rate. We have also achieved 60 frames per second for 1024 × 768 high-definition stereo matching, which is the fastest high-definition stereo matching to the best of our knowledge.

Design and Implementation of Real-Time High-Definition Stereo Matching SoC on FPGA

Thesis

submitted in partial fulfillment of the
requirements for the degree of

Master of Science

in

Embedded Systems

by

Lu Zhang B.Sc.
born in Shandong, China

This work was performed in:

Computer Engineering Group
Department of Microelectronics & Computer Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Delft University of Technology

Delft University of Technology
Department of
Microelectronics & Computer Engineering

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science for acceptance a thesis entitled **"Design and Implementation of Real-Time High-Definition Stereo Matching SoC on FPGA"** by **Lu Zhang B.Sc.** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: 2010-September-03

Chairman: _____
Prof.dr.ir. Koen Bertels

Advisors: _____
Dr.ir. Gauthier Lafruit

_____
Dr.ir. Georgi Kuzmanov

Committee Members: _____
Dr.ir. Rene van Leuken

_____

Abstract

Stereo matching has been widely used in many fields, such as viewpoint interpolation, feature detection system and free-view TV. However, the long processing time of stereo matching algorithms has been the major bottleneck that limits their real-time applications. During the past decades, many implementation platforms and corresponding algorithm adaptations are proposed to solve the processing time problem. Although notable real-time performances have been achieved, these works rarely satisfy both real-time processing and high stereo matching quality requirements.

In this thesis, we propose an improved stereo matching algorithm suitable for hardware implementation based on the VariableCross and the MiniCensus algorithm. Furthermore, we provide parallel computing hardware design and implementation of the proposed algorithm. The developed stereo matching hardware modules are instantiated in an SoC environment and implemented on a single EP3SL150 FPGA chip. The experimental results suggest that our work has achieved high speed real-time processing with programmable video resolutions, while preserving high stereo matching accuracy. The online benchmarks also prove that this work delivers leading matching accuracy among declared real-time implementations, with only 8.2% averaged benchmark error rate. We have also achieved 60 frames per second for 1024×768 high-definition stereo matching, which is the fastest high-definition stereo matching to the best of our knowledge.

Acknowledgments

This thesis work has been performed at IMEC (Leuven, Belgium) with support from the Computer Engineering group of TU Delft (Delft, the Netherlands). I want to take this opportunity to thank everyone who has helped me during this thesis work period.

First I would like to thank Dr. Gauthier Lafruit, Prof. Diederik Verkest and Prof. Kees Goossens for their agreement to offer me the opportunity to do my master's thesis project at IMEC. Their rich experience and scientific insights have also put the whole research and development in the correct direction - dedicated hardware implementation of high performance stereo matching.

During the thesis work period, I thank Ke Zhang for his ingeniously developed VariableCross stereo matching algorithm and guiding me into the stereo vision world. Ke also inspires me a lot during the hardware architecture and computing flow design. I thank Bart Vanhoof for his advices of selecting and using the suitable FPGA and development tools. Bart also contributes a lot to the camera interface and image rectification implementations on FPGA. I thank Luc Rynders for his collaboration on viewpoint interpolation analysis and corresponding software developments. I would like to thank Prof. Georgi Kuzmanov for his invaluable suggestions for improvements and all the hours he put in proofreading my thesis. Along the way Qiong Yang and Bart Masschelein also spent time on teaching me background knowledge and proposing insightful advices. I also would thank Gauthier Lafruit, Francesco Pessolano and Johan De Geyter for their excellent management.

Special thanks give to Prof. Tian Sheuan Chang and Gauthier Lafruit. Without their support and contributions this thesis work cannot make our desired achievements. Prof. Chang has proposed the Mini-Census transform, which is applied in our implementation to improve the robustness of stereo matching and reduce the memory consumption. I am also grateful to have his inspiration and encouragement. Gauthier's enthusiasm and ambition is the propulsion of the whole research and development. I owe cordial appreciation for his constant guidance, trust and support.

The recent two years' study and research in Delft and Leuven has been a wonderful experience. TU Delft and IMEC give me the opportunity to interact with outstanding and talented people from all over the world. I want to express my gratefulness to all my friends for their love, support and sharing all the happy time with me in the two fantastic European towns. I also deeply thank my family for their remote and emotional support and encouraging.

Finally I thank all my thesis defense committee members including Prof. Koen Bertels, Prof. Georgi Kuzmanov, Dr. Gauthier Lafruit and Prof. Rene van Leuken.

Lu Zhang B.Sc.
Delft, The Netherlands
2010-September-03

Contents

List of Figures

List of Tables

1 Introduction

Stereo matching has been, and continues to be one of the most active research topics in computer vision. The task of stereo matching algorithm is to analyze the images taken from a stereo camera pair, and to estimate the displacement of corresponding points existing in both images in order to extract depth information (inversely proportional to the pixel displacement) of objects in the scene. The displacement is measured in number of pixels and also called *Disparity*; disparity values normally lie within a certain range, the *Disparity Range*, and disparities of all the image pixels form the disparity map, which is the output of a stereo matching processing. An example with the *Teddy* benchmark image set is shown in Figure 1.1. In the figure the disparities are visualized as grayscale intensities, and the brighter the grayscale, the closer (to the stereo cameras) the object. Therefore the disparity map encodes the depth information

Figure 1.1: A disparity map example
(a): Image taken by the left camera. (b): Image taken by the right camera. (c): The ground truth disparity map associated with the left image.

of each pixel, and once we infer the depth information by means of stereo matching, we are able to obtain the 3D information and reconstruct the 3D scene using triangulation. Since stereo matching provides depth information, it has great potential uses in 3D reconstruction, stereoscopic TV, navigation systems, virtual reality and so on.

1.1 Motivation

The motivation of this thesis research and design work is to facilitate eye-gazing video conferencing with the support of high quality and real-time stereo matching. As many of the audience may have experienced, a common desktop video communication through

Skype and a webcam often lacks natural eye contact. The reason is obvious: providing natural eye contact requires the communication participants looking directly into the camera, but unfortunately, traditional video conferencing devices with a camera positioned above the screen (as shown in Figure 1.2) fail to satisfy this requirement. The angular displacement (noted as α in Figure 1.2) between the single camera and

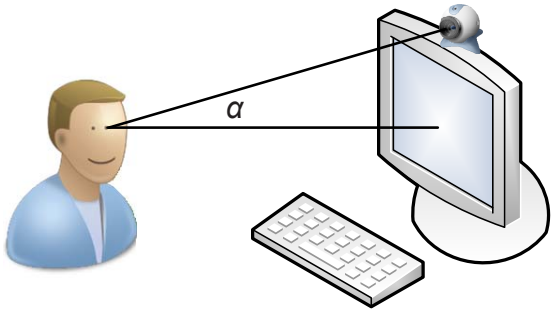

Figure 1.2: The eye gazing angle problem

monitor center causes one participant cannot look at the camera simultaneously while looking at the remote participant shown on the display. The lack of natural eye contact leads to awkward and uncomfortable feelings, and sometimes causes distrust between meeting participants. There are several approaches to correct the eye gazing problem, one remarkable one is the *Eye Contact Silhouette* system provided by Digital Video Enterprises (DVE). As shown in Figure 1.3, the camera is physically positioned behind

Figure 1.3: The Eye Contact Silhouette system

the display, which means that when the local participants look at the display showing remote participants, they are also looking directly into the camera and providing true eye contact (with zero gazing angle) to the far end sites. However, this arrangement is bulky and expensive, and more importantly it does not fit well to our commodity computer peripherals. A more elegant solution is to provide a computer-synthesized video which looks as if it were captured from a camera directly behind the monitor screen.

Our proposed solution is shown in Figure 1.4. It requires two cameras positioned on the top and bottom (or left and right) of the display, which provide the computer with stereo vision. The depth information in the computer's vision (meeting participants)

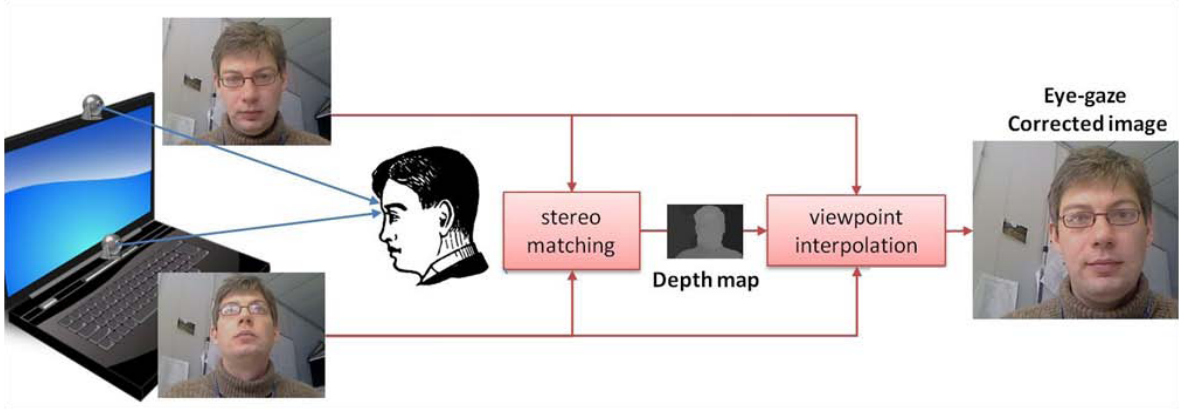

Figure 1.4: IMEC proposed eye gazing solution

is therefore obtained by means of stereo matching, and the in-between view from the virtually centered camera position is provided by a view interpolation algorithm. In this thesis we consider the two cameras are horizontally aligned, i.e., left and right. When the cameras are vertically aligned, they can also be considered as left and right with image rotation processing.

The full proposed eye-gazing video pipeline contains image acquisition, camera calibration, image rectification, stereo matching, view interpolation and video display. This thesis work focuses on the stereo matching part, which introduces the major processing bottleneck as discussed in the following section.

1.2 Problem Definition

In the processing pipeline, stereo matching is normally the most time consuming part. Due to high computational complexity, solutions that obtain satisfactory synthesizing quality under the constraint of real-time execution are quite rare, e.g. a recent three-camera viewpoint interpolation prototype system [32] proposed by Sharp Corporation only reaches a couple of frames per second for low-resolution images. In order to provide real-time and high-quality interpolation results, the stereo matching bottleneck must be solved first, especially for high frame rate and high definition (e.g. 50 frames per second with 1024×768 video size) applications such as business eye-gazing video conference systems.

Recent stereo matching design and implementations utilize various processing platforms such as high-performance multi-core CPU, GPU, DSP, FPGA and ASIC. Meanwhile, many stereo matching algorithms and adaptations are also proposed for efficient implementations on these platforms. Nevertheless, these work rarely satisfies both real-time and high matching accuracy requirements. For example, a recent implementation on a

GeForce8800 GTX GPU proposed by Zhang et al. [50] preserves high matching quality with an averaged benchmark error rate 7.65%, but only reaches 12 frames per second with 450 × 375 image resolution. In contrast, the FPGA implementation proposed by Jin et al. [23] achieves 230 frames per second stereo matching with VGA (640 × 480) resolution, but the matching accuracy degrades a lot with averaged benchmark error rate 17.24%.

Furthermore, frames to match in stereo matching come from two different cameras and hence might exhibit very different luminance or radiometric variations, incurring a severe matching cost bias to which some stereo matching cost functions are very sensitive. Some of the recently proposed algorithms are not robust in this condition.

1.3 Solutions and Contributions

To satisfy both real-time and high-accuracy stereo matching requirements, especially for high-definition view interpolation applications, we propose a hardware friendly high-accuracy stereo matching algorithm and a dedicated hardware implementation approach. The prototyping and evaluation design are implemented on a single EP3SL150 FPGA, and the implemented algorithm is based on *Adaptive Cross-Based Local Stereo Matching (VariableCross)* proposed by Zhang et al. [49] with our hardware friendly adaptations. Compared with the other two kinds of implementations (on CPU and GeForce8800 GTX GPU) of the same algorithm, this design has achieved significant speedup regarding processing time (speedup factor ranges from 41 to 162 compared with the CPU implementation), while preserving high matching accuracy.

Our main achievements are twofold: on one side, we improved the state-of-the-art VariableCross [49] algorithm regarding its accuracy and robustness; on the other side, we provided a high-performance and high-quality hardware design of the proposed algorithm. Summarized contributions of this thesis work are listed below:

We proposed an improved algorithm that:

- Preserves high matching accuracy with 8.2% averaged benchmark error rate.
- Maintains robust to radiometric distortions and bias differences introduced by stereo cameras. The robustness is obtained with the support of *Mini-Census Transform* proposed by Chang et al. [4].

We implemented the algorithm in hardware that:

- Achieves real-time stereo matching with various and run-time programmable resolutions e.g., 450 × 375 @ 193FPS, 640 × 480 @ 105FPS and 1024 × 768 @ 60FPS, independent on the disparity range.
- Proposes fully pipelined and scalable hardware structure. The hardware scales with the maximum disparity range under consideration, and the scalability offers hardware resource savings with small disparity range applications. On the FPGA

EP3SL150 we have evaluated implementations with maximum disparity range of 16, 32 and 64.

- Implements industry standard on-chip interconnections (*Avalon System Interconnect Fabric*). This enables our design to work properly with other Avalon-compatible processing modules, for example the Altera's *Video and Image Processing (VIP) Suite*.
- Provides fully functional SoC reference design with run-time configurable parameters and disparity map visualizations on a standard DVI monitor.

The whole design is implemented on the DE3 evaluation board, which contains the EP3SL150 FPGA and rich peripherals e.g., DDR2 SDRAM, USB and camera interfaces. A DVI extension board is also employed for the disparity map display. To verify the hardware implementation on FPGA, the stereo matching hardware processing starts with rectified stereo images from an external frame buffer (DDR2 SDRAM) and finally output disparity maps to a result frame buffer. The disparity maps are uploaded to a PC for verification and benchmarking. To visualize disparity maps, the result frame buffer is removed and the disparity maps are output to a monitor. To enable follow-up work and view interpolation prototyping, the complete setup is also proposed, as shown in Figure 1.5.

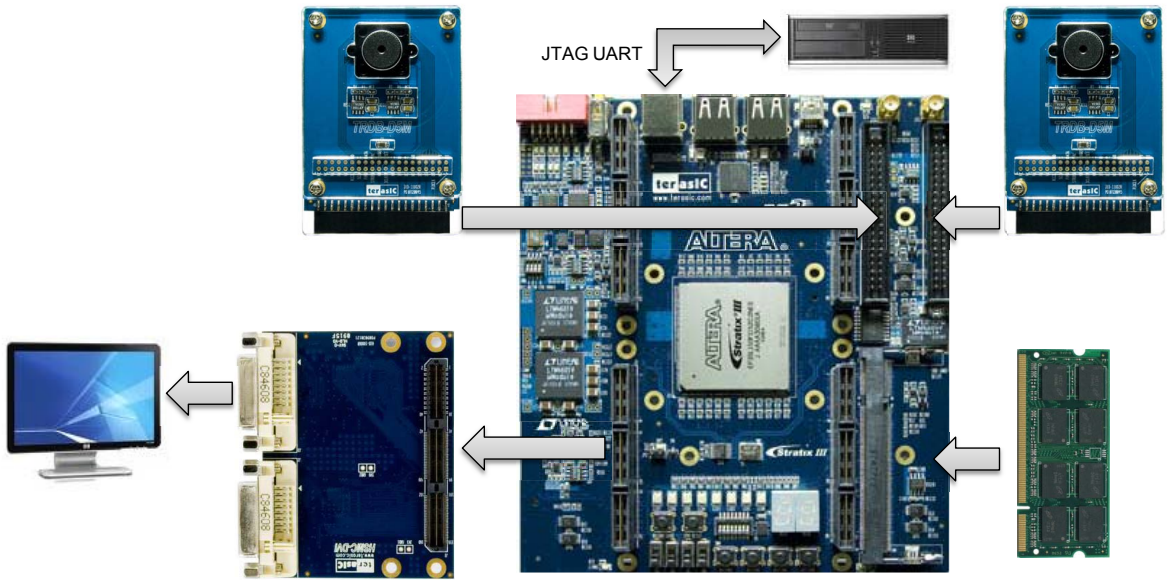

Figure 1.5: Proposed view interpolation setup with FPGA

1.4 Overview of Chapters

The content of the thesis text is organized as follows: Chapter 2 introduces the background knowledge of stereo matching computing flow, related work and our design

considerations. In Chapter 3, we present the proposed stereo matching algorithm and corresponding hardware module design in a top-down approach. Chapter 4 provides two reference SoC designs with the stereo matching modules on our selected FPGA and corresponding hardware/software tasks. In chapter 4, we also estimate the external memory storage and bandwidth utilizations. Chapter 5 evaluates the whole design and implementation according to our design considerations, and provides comparison to related work. Finally, Chapter 6 summarizes the thesis and suggests valuable optimization techniques and interesting application developments as future work based on our stereo matching results.

2 Background and Related Work

In the past two decades, various stereo matching algorithms have been proposed and they were summarized and evaluated by Scharstein and Szeliski [35]. In this notable work, these proposed stereo matching algorithms are categorized into two major types: local area based methods and global optimization based methods. In local methods, the disparity evaluation at a given pixel is based on similarity measurement performed in a finite window. The similarity metric is defined by a *Matching Cost* and all costs in the local window are often aggregated to provide a more reliable and robust result. On the other hand, global methods define global cost functions and solve an optimization problem. Global algorithms typically do not perform an aggregation step, but rather seek a disparity assignment that minimizes a global cost function.

In this thesis we are particularly interested in local stereo matching methods, which generally have low computation complexity and less storage requirement; and therefore they are suitable for real-time and embedded implementations. In Section 2.1 we introduce the image rectification required by efficient local stereo matching; In Section 2.2 fundamental basis of the stereo matching computation flow and our chosen algorithm are discussed. We present our design considerations and corresponding algorithm adaptations in Section 2.4 and conclude this chapter with related work and comparisons.

2.1 Epipolar Geometry and Image Rectification

For a given pixel in the left image, the stereo matching algorithm is to seek the corresponding one in the right image, for example in Figure 2.1 the object point p appears in both left and right views as pixel $p(x, y)$ and $p'(x', y')$, respectively. The two pixels are defined as a correspondence pair, and the displacement of their locations in the stereo view is the disparity; in this case the disparity is equal to $(x - x')$. In Figure 2.1, the distance between the stereo cameras' focal axis is defined as the *Baseline*. To enable efficient stereo matching, it is necessary that the two input images are *Rectified*, fulfilling *Epipolar Geometry*. In epipolar geometry each pixel in the left image finds its correspondence in the right image (if exists) on a specific line, called the *Epipolar Line*. If the two images are rectified, the epipolar lines coincide with the image rows and run parallel to the baseline; so that the correspondence search only needs to be performed along a 1-dimension scanline. The concept is shown in Figure 2.2. Image rectification according to epipolar geometry is an important processing step before the stereo matching algorithm, enabling less complex implementation of the correspondence searching. Detailed rectification algorithm and implementations are not further discussed in this thesis, and we therefore assume in the sequel input stereo images are rectified.

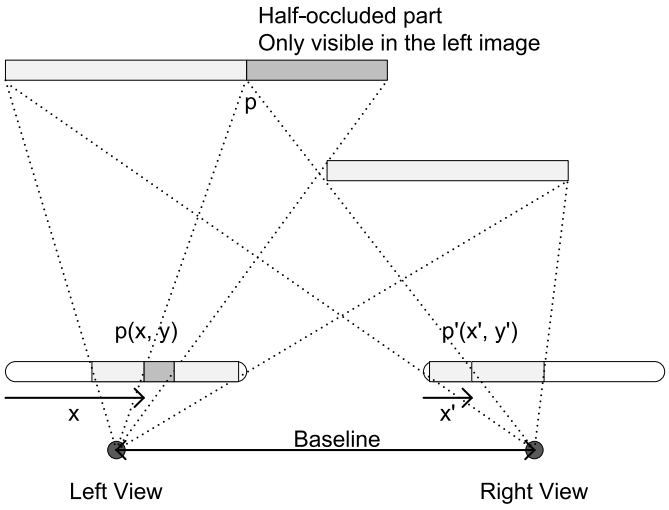

Figure 2.1: Correspondences and the half occlusion problem

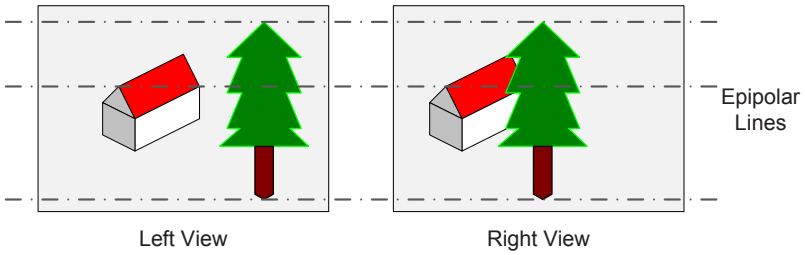

Figure 2.2: Rectified images and epipolar lines

2.2 Stereo Matching Computation Flow

In general, stereo matching algorithms perform the following steps:

1. Matching cost computation.
2. Cost (support) aggregation.
3. Disparity computation / optimization
4. Disparity refinement.

The four steps are introduced in the following subsections. In local algorithms, emphasis is on the matching cost computation and cost aggregation steps.

2.2.1 Matching Cost Computation

Thanks to image rectification, searching for correspondences is reduced to 1-dimensional. In addition, the disparity values lie within a certain disparity range;

so the basic matching method is performed pixel-to-pixel in a horizontal line segment, as shown in Figure 2.3. In this example the disparity range is $[0 \ldots d_{max}]$, and $d_{max} = 7$ in this figure. The disparity range is determined by the distance between the scene

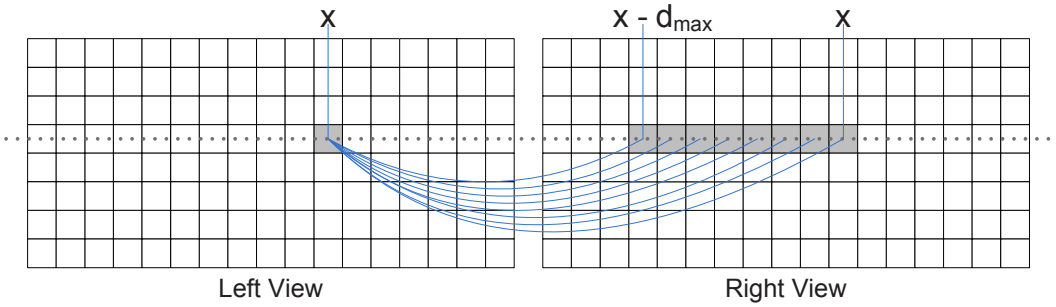

Figure 2.3: Basic pixel-to-pixel matching method

objects and the camera baseline, and the length of the baseline itself. To determine the matched correspondences in a stereo pair, the matching costs are defined to evaluate the probability of a correct match; the smaller the costs, the higher the probability. For pixel-to-pixel matching, the matching cost is commonly defined as *Absolute Difference, Squared Difference, Hamming Distance etc.*, on both gray and color images. The Equation 2.1 defines the matching cost measured by absolute intensity difference; where $I(\cdot)$ returns the grayscale intensity of a pixel in the left image and $I'(\cdot)$ returns the intensity of a right image pixel. The two pixels under consideration are related by a disparity hypothesis d.

$$Cost(x, y, d) = |I(x, y) - I'(x - d, y)| \tag{2.1}$$

In this simple case, the correspondence is determined by the minimum cost value, and the associated d is the disparity of the pixel $p(x, y)$ in the left image.

Hamming distance cost is normally computed based on transformed images. The transform is a pre-processing (often a filter) step before the matching cost computation. In our implementation we adopted the *Mini-Census Transform* proposed by Chang et al.[4]. Details of this transform algorithm are introduced in Chapter 3.

To avoid ambiguity, in the following context we refer the matching cost between two single pixels as the *Raw Matching Cost*. Raw matching costs in a restricted area are often aggregated to improve the matching accuracy.

2.2.2 Area-Based Matching Cost Aggregation

Typically, to increase the reliability of cost evaluation and the robustness to image noise, local stereo matching methods choose to aggregate (accumulate) raw matching costs over a *Support Region*. The most basic support region is a square window, as shown in Figure 2.4. In this example, a fixed 3×3 support region is built for a given pixel in the left image and all raw costs in this region are aggregated. The same computation applies to each pixel in the disparity range in the right image.

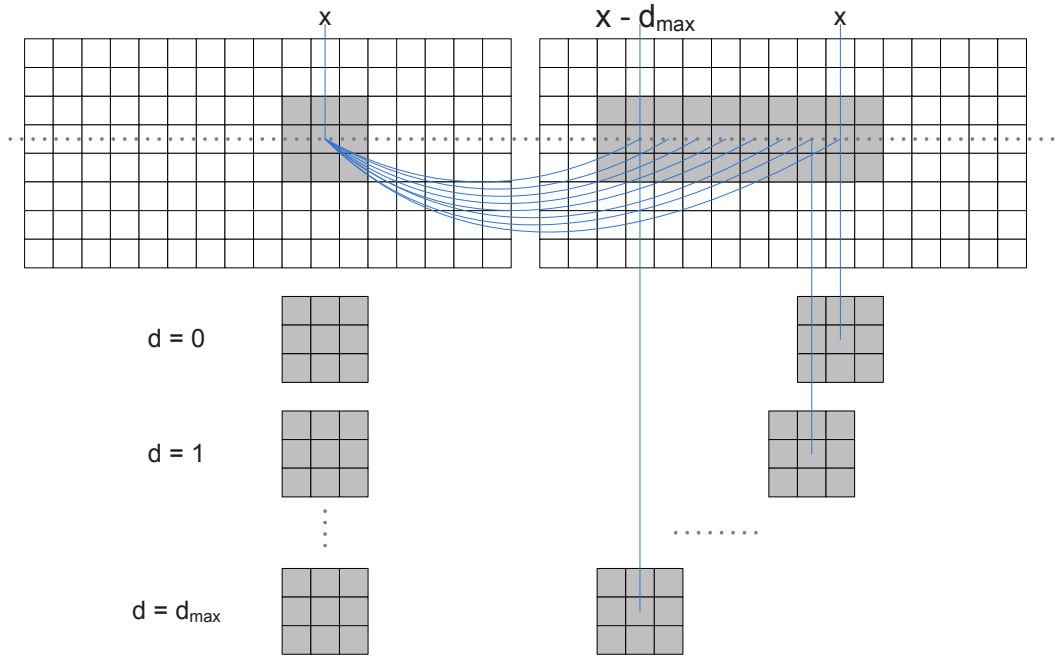

Figure 2.4: Cost aggregation over a fixed window

The well known challenge for area-base local stereo matching is that a local support region should be large enough to contain sufficient intensity variation for reliable matching, while it should be small enough to avoid disparity discontinuity inside the region. Therefore state-of-the-art stereo matching algorithms often shape the support window in order to include only pixels with the same (unknown) disparity. Veksler [42] found a useful range of window sizes and shapes while evaluating the aggregated cost. Zhang et al. [49] adaptively shaped a support window with variable crosses in order to bound the pixels near arbitrarily shaped disparity discontinuities. Yoon and Kweon [48] also assigned a support weight to the pixel in a support window based on color similarity and geometric proximity. The Figure 2.5 illustrates a simple shaped support region example. The support regions are built around the anchor pixel in the left image, and the pixels in the disparity range in the right image. Normally for evaluating each hypothetical pair, only the overlapped area of the two support windows is considered to take both images into account.

Some local stereo matching algorithms simply sum up all the raw matching costs in a support region, for example the *Sum of Absolute Differences (SAD), Sum of Squared Differences (SSD) and Sum of Hamming Distances*. These algorithms are clearly broken down into step 1 and 2 as mentioned above. On the other hand, some of the algorithms merge step 1 and 2 and define a new matching cost based on the support region, such as *Normalized Cross-Correlation (NCC)*. The Table 2.1 summarizes involved computations for these commonly used matching cost aggregation. In the table, the $\sum_{x,y}(\cdot)$ denotes aggregation over the (overlapped) support region; $I(\cdot)$ returns the intensity of a pixel and $BitVec(\cdot)$ gives the bit vector associated with a pixel after

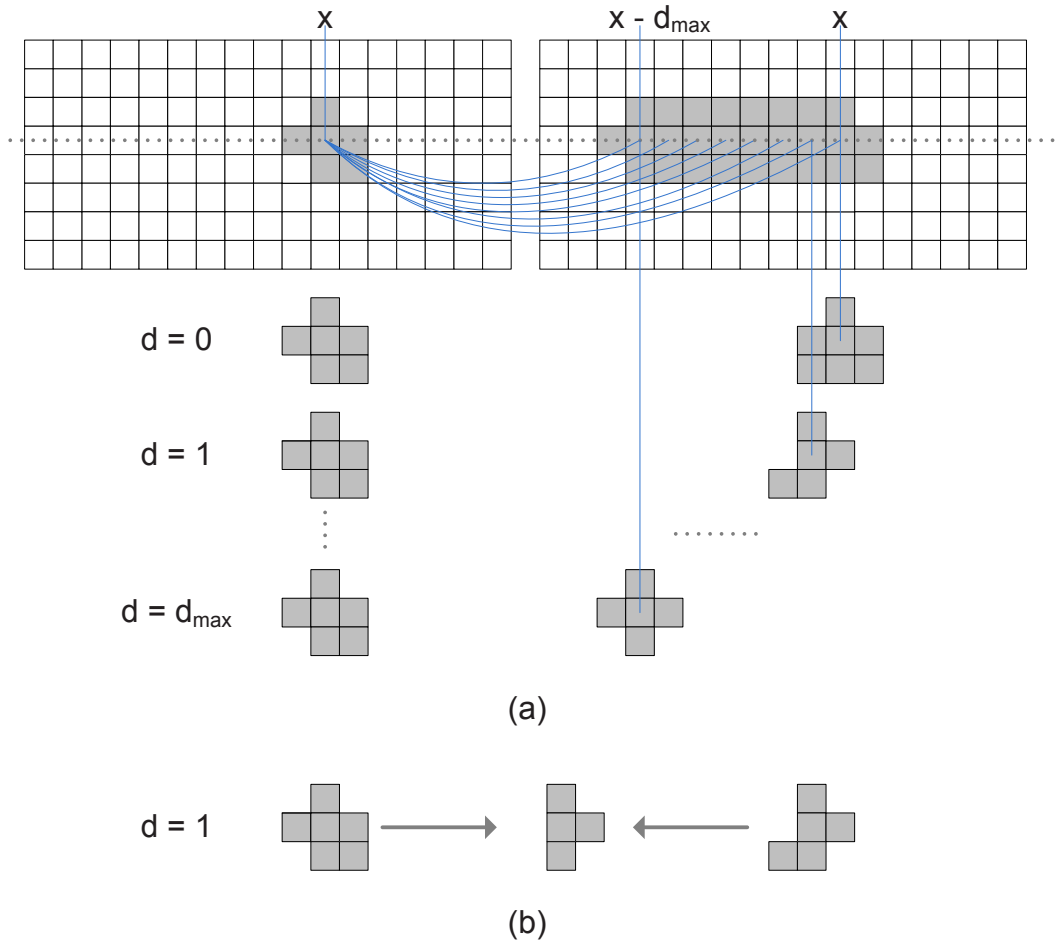

Figure 2.5: Cost aggregation over an shaped window
(a): Shaped window for pixels in the left and right image. (b): Overlapped area of the windows associated with a hypothetical pair.

Matching Cost	Computations
Sum of Absolute Differences	$\sum_{x,y} \lvert I(x,y) - I'(x-d,y) \rvert$
Sum of Squared Differences	$\sum_{x,y} (I(x,y) - I'(x-d,y))^2$
Sum of Hamming Distances	$\sum_{x,y} \text{Hamming}(\text{BitVec}(x,y), \text{BitVec}'(x-d,y))$
Normalized Cross-Correlation	$\frac{\sum_{x,y}(I(x,y) \cdot I'(x-d,y))}{\sqrt{\sum_{x,y}(I(x,y)^2 \cdot I'(x-d,y)^2)}}$

Table 2.1: Matching cost computations and aggregations

a certain transform, e.g. Census. More extensive discussions and evaluations about state-of-the-art matching cost computations/aggregations are found in [19].

2.2.3 Disparity Estimation

Disparity estimation is to choose at each pixel in the left image the disparity d associated with the minimum cost value. Generally, this involves a local "Winner-Takes-All" (WTA) optimization at each anchor pixel $p(x, y)$ in the left image and the pixels in the disparity range in the right image. Therefore the basic concept of stereo matching is explained by the pseudo code listed in Figure 2.6. Though straightforward, the

```
FOR (y IN 0 TO ImageHeight - 1)
     FOR (x IN 0 TO ImageWidth - 1)
        FOR (d IN 0 TO d_max)
            IF (Cost(x, y, d) < MinCost(x, y, d))
                MinCost(x, y, d) = Cost(x, y, d)
                d_p(x, y) = d
            END IF
        END FOR
    END FOR
END FOR
```

Figure 2.6: Stereo matching pseudo code

embedded computing loops cause the major bottleneck for real-time implementations, and one of the contributions of this thesis work is to solve this problem with parallel and pipelined computing.

A limitation of this disparity estimation is that uniqueness of matches is only guaranteed for one image (the left image), while pixels in the right image might get matched to multiple points. To compensate for this problem, a similar matching process is also performed based on the right image, and the disparity map of the right image is also generated. The uniqueness and consistency tests are performed in the disparity refinement step, combining both disparity maps.

2.2.4 Disparity Refinement

The correspondence may not exits for each pixel in one image due to *Half-Occlusion*, as shown in Figure 2.1. *Consistency Check* (comparing left-to-right and right-to-left disparity maps) is often employed to detect occluded areas and perform uniqueness tests, and detected mismatches can be fixed by surface fitting or by distributing neighboring valid disparity estimates [35].

If disparity results pass consistency check, some additional techniques are used to further the disparity reliability and accuracy, for example *Sub-Pixel Estimations* [37] or local high-confidence *Disparity Voting* [28] in a support region. After this step, a final median filter is also helpful for removing speckle noises and smoothing the disparity map.

Disparity refinement normally contributes considerable improvements to the WTA-estimated disparity map, so in our final implementation we applied consistency check, local disparity voting and median filtering.

2.3 Related Work

To enable efficient stereo matching computation, especially for real-time applications, various approaches have been developed during the past two decades. In recent years, notable achievements have been made in either matching accuracy and processing speed. We categorize them according to the implementation platforms.

General purpose CPUs are often used as the first algorithm prototyping platform, and many efficient aggregation methods are proposed to accelerate the CPU processing. Tombari et al. [39] proposed an efficient segmentation-based cost aggregation strategy, which achieves a frame rate of 5 FPS for 384 × 288 images with 16 disparity range on an Intel Core Duo clocked at 2.14 GHz. But it drops to only 1.67 FPS with 450 × 375 images and 60 disparity range. Later, Salmen et al. [34] optimized a dynamic programming algorithm on a 1.8 GHz PC platform and achieved a frame rate of 5 fps for 384 × 288 and 16 disparity range stereo matching. Kosov et al. [26] combined a multi-level adaptive technique with a multi-grid approach that allows the variational method to reach 3.5 FPS with 450 × 375 images and 60 disparity range. Zhang et al. [49] proposed the VariableCross algorithm and orthogonal integral image technique to accelerate the aggregation over irregularly shaped regions. This work achieves 7.14 FPS with 384 × 288 and 16 disparity range, but only 1.21 FPS with 450 × 375 and 60 disparity range. Though real-time performances are generally poor, CPU implementations often achieve very good accuracy with less than 10% averaged error rates. Error rates of the above mentioned four implementations are 8.24%, 8.83%, 9.05% and 7.60%, respectively.

With GPU implementation, a global optimizing real-time algorithm was developed by Yang et al. [46] on a GeForce 7900 GTX GPU. The authors used hierarchical belief propagation and reached 16 FPS for 320 × 240 images with 16 disparities. This work presented high matching accuracy with 7.69% averaged error rate, but clearly its frame rates and image resolution are limited. Another GPU implementation was introduced by Wang et al. [43]. It is based on an adaptive aggregation method and dynamic programming and reaches a frame rate of 43 FPS for 320 × 240 images and 16 disparity levels on an ATI Radeon XL1800 graphics card. Compared with the work by Yang et al. [46], it achieves higher frame rates, but its accuracy degrades to 9.82%. Later, the authors of VariableCross algorithm [49] also implemented the same algorithm on GeForce GTX 8800 [50], which achieves 100.9 FPS with 384 × 288 images and 16 disparity range. However, the frame rate drops to 12 FPS with 450 × 375 resolution and 64 disparities. Most recently, Humenberger et al. [20] optimizes the Census Transform on GeForce GTX 280 GPU, and reaches 105.4 FPS with 450 × 375 resolution and 60 disparity range. This work also maintains high accuracy with averaged error rate of 9.05%. Although notable performances have been achieved with GPU implementations, GPUs are not efficient in power and memory consumption and

hence not suitable for highly integrated embedded systems. Furthermore, these works rarely report the performances with high-definition stereo matching.

Chang et al. [3] investigated the performance of using DSP as the implementation platform. In this work, the authors proposed a 4×5 jigsaw matching template and the dual-block parallel processing technique to enhance the stereo matching performance on a VLIW DSP. The DSP implementation achieved 50 FPS with disparity range of 16 for 384×288 stereo matching. Nevertheless, its accuracy degrades a lot, with above 20% averaged error rate and its frame rate drops to 9.1 with 450×375 images and 60 disparity range.

Implementations with CPUs, GPUs and DSPs are all software based approaches. The major problem with software is that the computing logic and data paths are fixed and not configurable. As a result, they are not optimized for available algorithms. Furthermore, their performances also require very high device clock frequency and memory bandwidth. To overcome the software limit, recent works also investigated the performance of dedicated hardware design. Chang et al. [4] proposed a high performance stereo matching algorithm with Mini-Census and Adaptive Support Weight (MCADSW) and also provided its corresponding real-time VLSI architecture. The design is implemented with UMC 90nm ASIC, and reaches 42 FPS with 352×288 images and 64 disparity range. It also preserves high matching accuracy (unknown average) with benchmark image sets. Jin et al. [23] proposed a fully pipelined hardware design using Census Transform and Sum of Hamming Distances. This work has achieved 230 FPS with 640×480 resolution and 64 disparities. However, its averaged error rate is 17.24%. Though the two hardware implementations achieve remarkable performance, the image resolutions are limited by the on-chip memories, due to relatively high storage requirements or less efficient hardware design. In Chapter 5, we provide a comparison between our implementation and all of the above mentioned references.

2.4 Design Considerations and Targets

In order to enable real-time processing speed while preserving high stereo matching accuracy, we considered the following aspects to choose an algorithm and make adaptations for our implementation.

1. Matching accuracy

   Our target implementation should achieve less than 10% averaged error rate.

2. Robustness

   Frames to match in stereo matching come from two different cameras and hence might exhibit very different luminance or radiometric variations, incurring a severe matching cost bias to which some stereo matching cost functions are very sensitive. Our implementation should also be robust, i.e., none or slight matching accuracy variation in this situation.

3. Real-time Implementations

   Our design targets for at least 50 frames per second with programmable resolutions, and high-definition stereo matching is desired.

4. Implementation Complexity

   Our design should avoid complex computations such as divisions and square root calculations.

5. Implementation Scalability

   The proposed hardware structure should scale well with the maximum allowed resolution and disparity range. Our desired target is HDTV resolution.

Compared with software approaches, the major benefit of using FPGA or ASIC is the great flexibility in logic and data path design, which offers extreme parallelism in data flow and pipelined processing. Therefore they enable highly efficient and optimized designs and implementations. Based on the related work and our investigation, we believe customized hardware design is the most suitable approach for our proposed algorithm. Since FPGAs provide great resources for parallel computing and fast prototyping, we choose FPGA to implement our algorithm. To achieve a fast prototyping and enable future developments, we select Terasic DE3 development board as the development platform. The board contains Altera EP3SL150 FPGA and rich peripherals like JTAG UART port, DDR2-SDRAM, camera interfaces and DVI extension card and so on.

Stereo Matching Algorithm: Software-Hardware Co-Design

3

In this chapter, we introduce the proposed stereo matching algorithm which is modified from *Adaptive Cross-Based Local Stereo Matching* proposed by Zhang et al. [49]. We modify this algorithm in order to make it more robust and more hardware friendly. Key to the best processing throughput is to fully pipeline all processing steps, i.e. with source image pixels come progressively in scanline order, a new income pixel gets its disparity at the end of the pipeline after a certain pipeline latency, and valid disparities also come successively in scanline order, synchronized with the input pixel rate. To enable fully pipelined implementation, parallelization is also important to provide all required data at the entrance of a pipelined processing module.

Our proposed algorithm and corresponding hardware pipeline design are introduced together in a top-down approach. Section 3.1 discusses the overview of the whole stereo matching processing and proposed data-level parallelism. The full pipeline is divided into three major processing blocks, i.e., *pre-processing, stereo matching and post-processing*. In hardware their functions are performed by corresponding *Pre-Processor, Stereo-Matcher and Post-Processor*, which are introduced in three subsections of this chapter.

3.1 Overview of the Stereo Matching Pipeline

Overview of the stereo matching processing flow is presented in Figure 3.1, corresponding to the aforementioned pseudo algorithm code (see Figure 2.6). Clearly the embedded computing loops cause the major problem for efficient implementations. If design follows this diagram, e.g., using state machines, raw stereo image data has to be read repeatedly into the processing module, which not only prolongs processing time but also exhausts external memory bandwidth.

According to our research, the disparity evaluation loop can be unrolled and all following processing modules are also parallelized accordingly. The parallelized data flow is illustrated in Figure 3.2. In the parallelized computing, matching costs with different hypothetical disparities are computed and aggregated mutually independently. As shown in the figure, multiple concurrent computing paths are employed to enable the parallelization. In principle any number of parallel paths are supported by the algorithm, but in practice this number is usually limited by the available hardware resources.

Our proposed hardware design follows the parallelized data flow, with pipelined processing at each algorithmic module. Although the cameras and image rectifications are shown in the figure, they are not implemented with this thesis work. The purpose

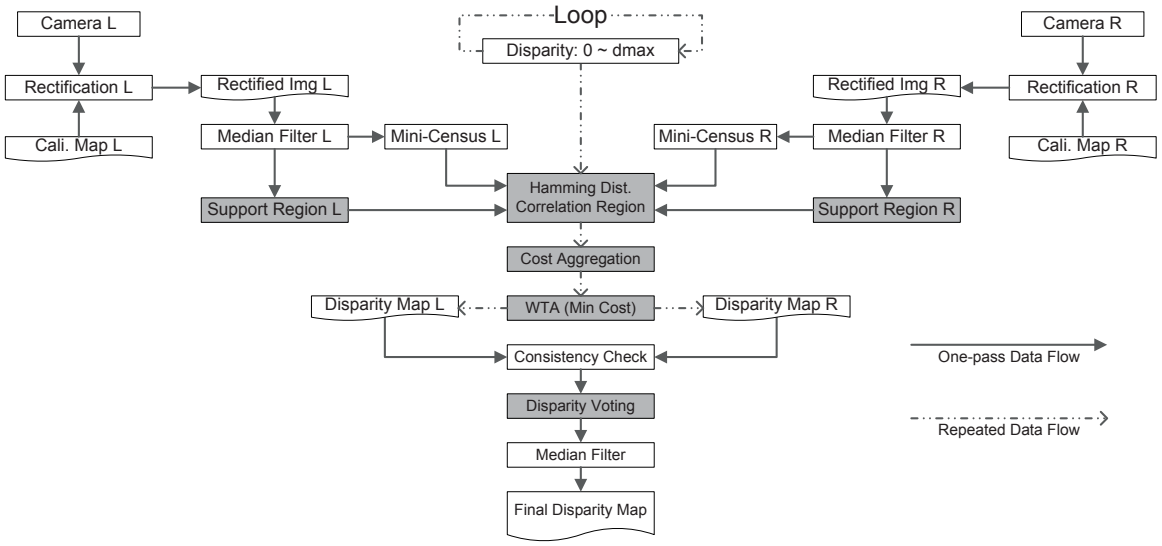

Figure 3.1: Sequential stereo matching processing flow
Modified functions by this thesis work are shaded

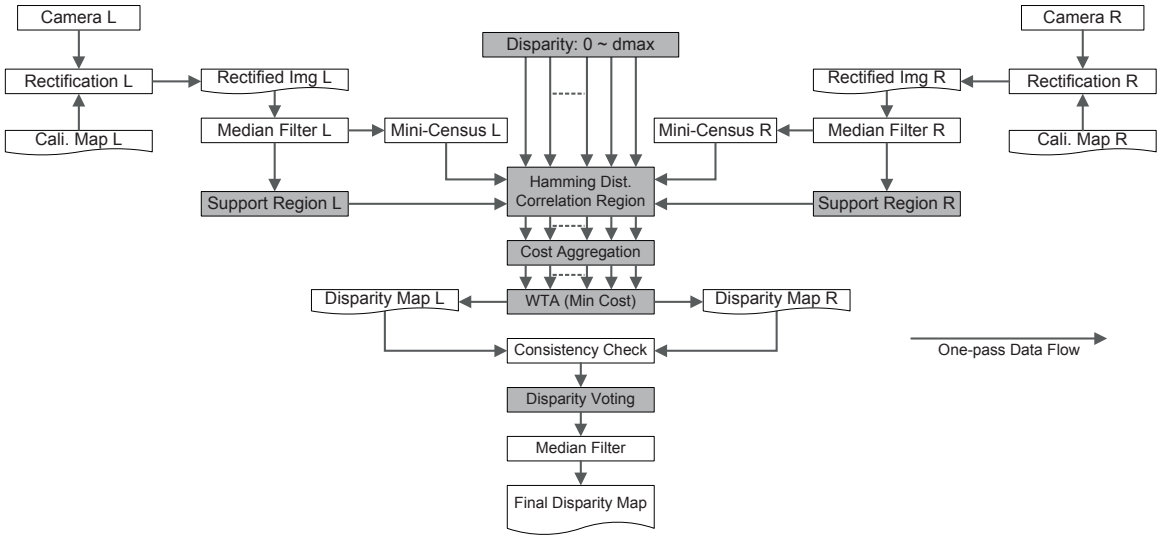

Figure 3.2: Parallelized stereo matching processing flow
Modified functions by this thesis work are shaded

of their presence is to show that the left and right images are processed independently before the actual stereo matching starts; and their pixel coordinates must be synchronized to ensure a valid stereo matching processing. Processing modules shown in Figure 3.2 are categorized into three higher level modules, i.e., the Pre-Processor, Stereo-Matcher and Post-Processor. Their functions and inputs/outputs are shown in Figure 3.3. DMAs and Avalon interfaces are implemented for SoC integrations, which

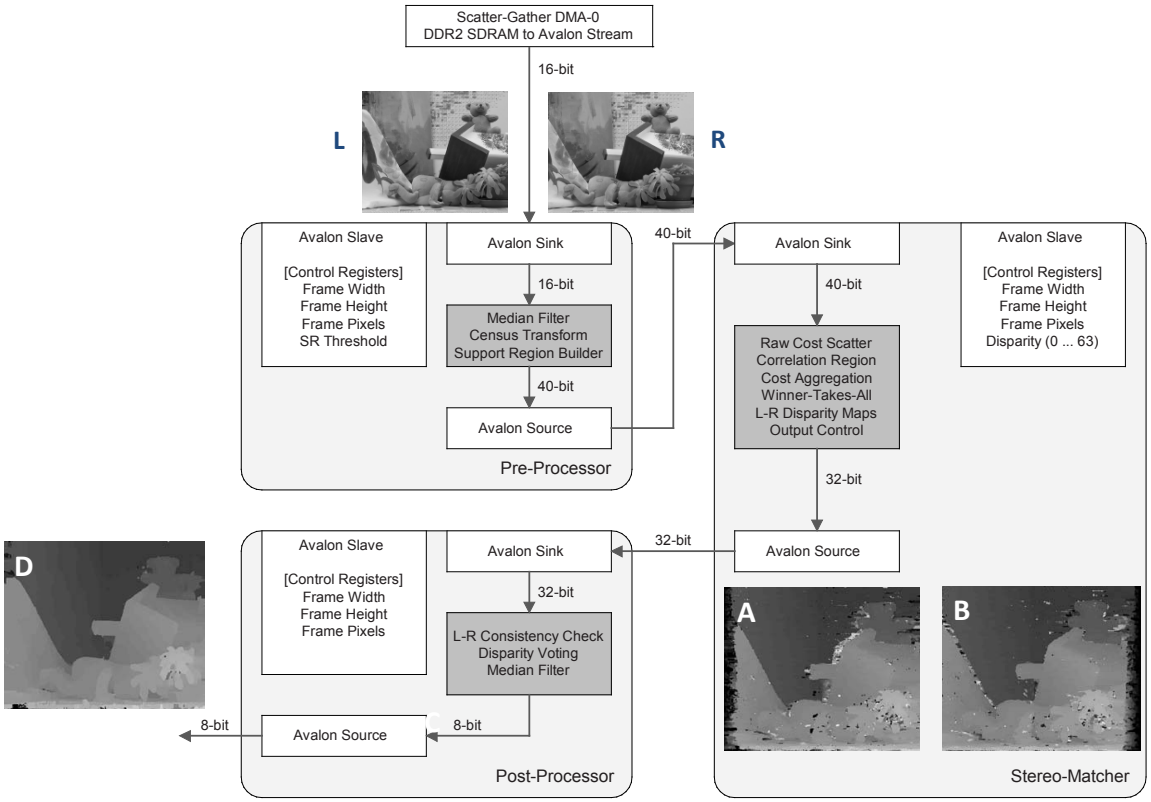

Figure 3.3: Stereo matching pipeline and functions

will be introduced in Chapter 4. The hardware pipeline processing sequence is explained below.

1. The DMA transfers a pair of stereo frames to the Pre-Processor, with only luminance intensities. Synchronized luminance pixels from the left and right image are packed into one 16-bit data word.

2. The Pre-Processor performs median filter, census transform and support region construction on left and right frame independently and sends its transformed images to the Stereo-Matcher.

3. The Stereo-Matcher first computes the raw matching cost and overlapped support region at each pixel according to different hypothesis disparities. Then all the costs in an overlapped support region are aggregated; costs with different hypothesis disparities are processed independently in parallel.

4. Aggregated costs of all disparity hypothesis are presented to the WTA module by the cost aggregation module, and the best disparity is selected by a tree-based minimum value searching structure.

5. In the Stereo-Matcher, two disparity maps are generated for the left and right image respectively, denoted as image A and B in Figure 3.3.

6. The Post-Processor receives two disparity maps and support region data from the Stereo-Matcher, performs L-R consistency check, local disparity voting and median filter to detect occluded regions and improve the disparity map quality. The Post-Processor outputs the final disparity maps.

Each processor also contains several control registers for run-time parameter configuration, these parameters include image resolution, disparity range and some threshold values. In the following sections, we present the proposed algorithm functions together with the corresponding hardware design. We also provide the latency calculations based on 100MHz operation clock with the selected EP3SL150 FPGA. If some functions become critical paths with another device technology, it is required to implement more pipeline stages.

3.2 Design of the Pre-Processor Pipeline

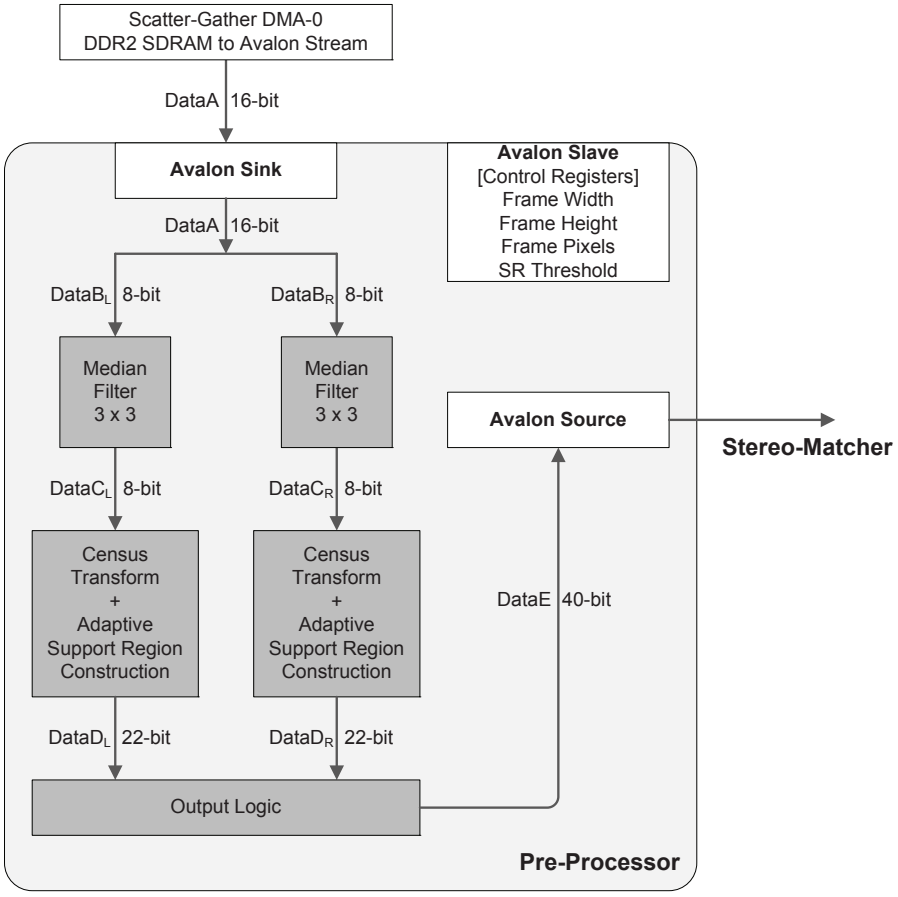

Figure 3.4: Pre-Processor internal processing pipeline

The internal processing pipeline of Pre-Processor is shown in Figure 3.4. Each data word in the pipeline is comprised of several fields. Since most processing modules require data from left and right video frames simultaneously for parallel computing, the corresponding data word also contains both. In the Pre-Processor, left and right frames are processed independently, and corresponding data words are marked with 'L' or 'R' subscripts.

3.2.1 Median Filter

We have applied a 3 × 3 median filter to each raw frame pixel to remove impulsive noises. The function of a median filter is shown in Figure 3.5. Input (DataA) to the median filter module is a systolic stream carrying a combined data word with both left and right luminance pixels in progressive scanline order (see Figure 3.6). Computations on the left and right frames (DataB) are performed independently in

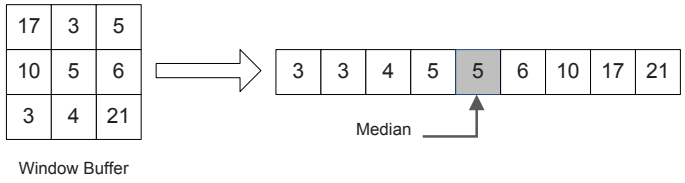

Figure 3.5: Median filter

parallel, and the output from the median filter module is the filtered data stream (DataC), in synchronization with the input pixel rate.

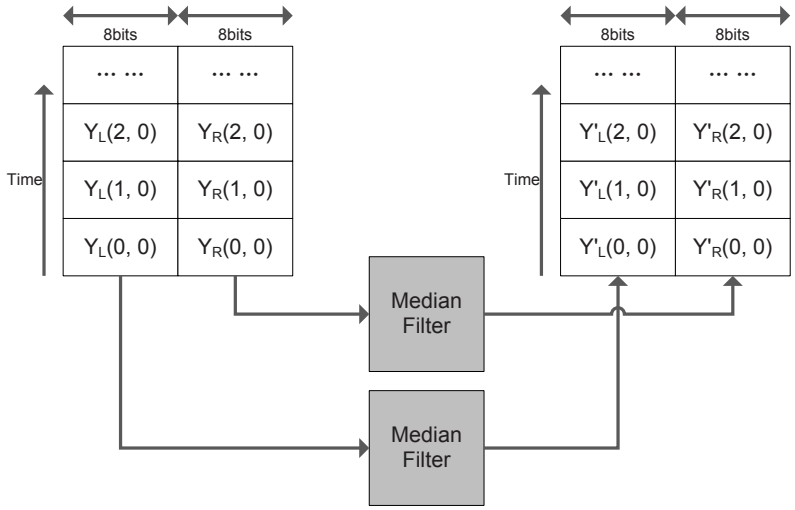

Figure 3.6: Median filter data stream

To get the median value from a 3×3 array, we first prepare all the 9 pixels in a window buffer and afterwards they are captured simultaneously by a follow-up sorting module, which delivers the median value in a few pipeline cycles. The pixel buffer window slides over the whole frame and provides a new 3×3 pixel array in every valid pipeline cycle, therefore after the initial latency for preparing the first window content and sorting the 9 values, we get continuous filtered pixels synchronized with the input pixel rate.

Because the pixel stream comes in scanline order, and a 3×3 filter window is required for each pixel, 2 horizontal lines and 3 shift register arrays of length 3 are used to form a valid filter window. The structure and interconnections of line buffers and shift registers are illustrated in Figure 3.7. In the implementation, each line buffer stores one scanline of 8-bit luminance pixels, and each *WinReg* contains one 8-bit luminance pixel. The *Row Index Counter* cyclically counts from 0 to (frame width - 1) and provides read/write addresses for the line buffer memory blocks. Because the line buffer memory has a 2-cycle read latency, its write address and write data are also delayed by 2 cycles to match the processing sequence. This module prepares all the required pixels for computing the median for the center pixel $p(x, y)$, buffered in *WinReg4*.

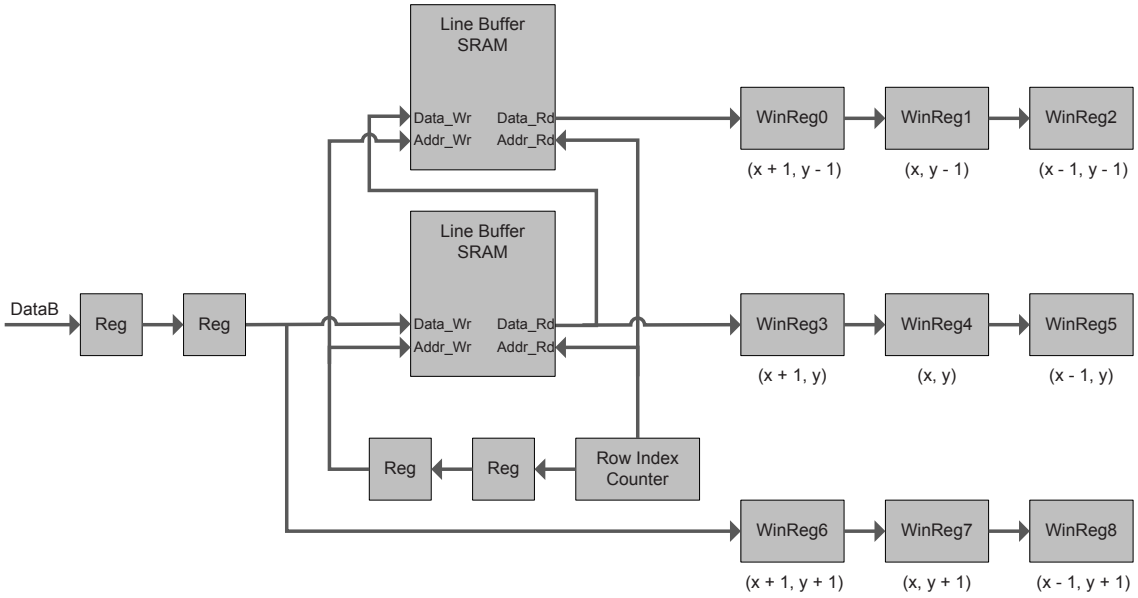

Figure 3.7: Line buffers and registers in a median filter

The depth of a line buffer is determined by the maximum allowed frame width. A 1024 × 8 SRAM block supports any frame width no larger than 1024. For an input pixel $p(x, y)$ from DataB stream, it has to wait for

$$2\ (read\ latency) + frame\ width + 2\ (WinReg3\ and\ WinReg4)$$

cycles to arrive in *WinReg4*. As pixels come in continuously, the 3 × 3 pixel window also slides over the whole frame in scanline order (see Figure 3.8; the four boundary pixel rows/columns are not filtered, by-passing the sorting module) and provides all neighboring pixels of each center pixel simultaneously for the following sorting module.

The 9 luminance values in a filter window are captured by a sorting module to get the median of them. We have adopted the median sorting structure proposed by Vega-Rodrguez et al. [41] and implemented suitable pipeline stages. The implemented median sorting module is shown in Figure 3.9. Each basic building block in this systolic array is a combination of a comparator and two multiplexers, which performs a basic sorting function on two unsigned values. The full array is divided in 3 pipeline stages so the desired median is available after 3 valid pipeline cycles.

Median filtering is performed on the left and right frames independently in parallel, and the output streams of this module are filtered stereo frames (DataC in Figure 3.4). The pipeline latency of the median filter module is given by Equation 3.1.

$$Latency_{mf} = frame\ width + C_{mf} \tag{3.1}$$

In the equation C_{mf} is a small constant value depending on the above mentioned read latency, shift register cycles and actual implemented pipeline stages. Quantitative reports of the latencies are presented in Chapter 5.

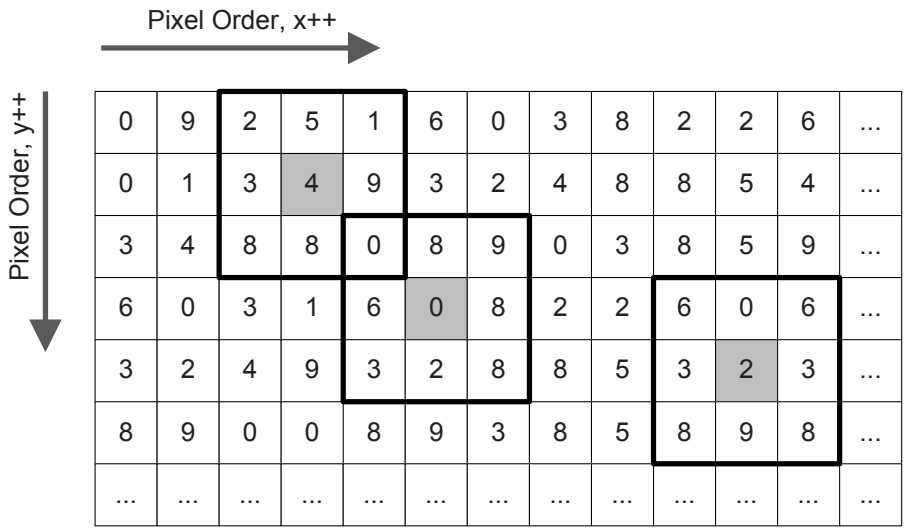

Figure 3.8: Slide window for a median filter

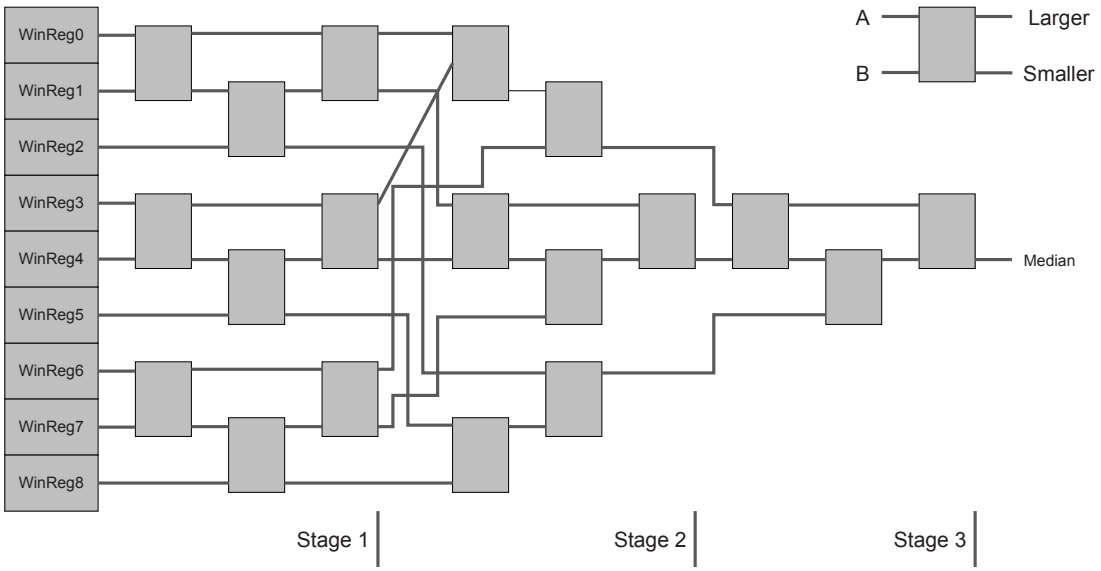

Figure 3.9: Median filter sorting array

3.2.2 Census Transform and Adaptive Support Region Construction

3.2.2.1 Census Transform

Census transform is a process extracting relative information for the center pixel from its neighbors, and its output is a bit vector encoding the relative information. Census transform has a few variants depending on different patterns of selected neighbor pixels; the one we have adopted is the *Mini-Census* algorithm proposed by Chang et al. [4],

and the transform is illustrated in Figure 3.10.

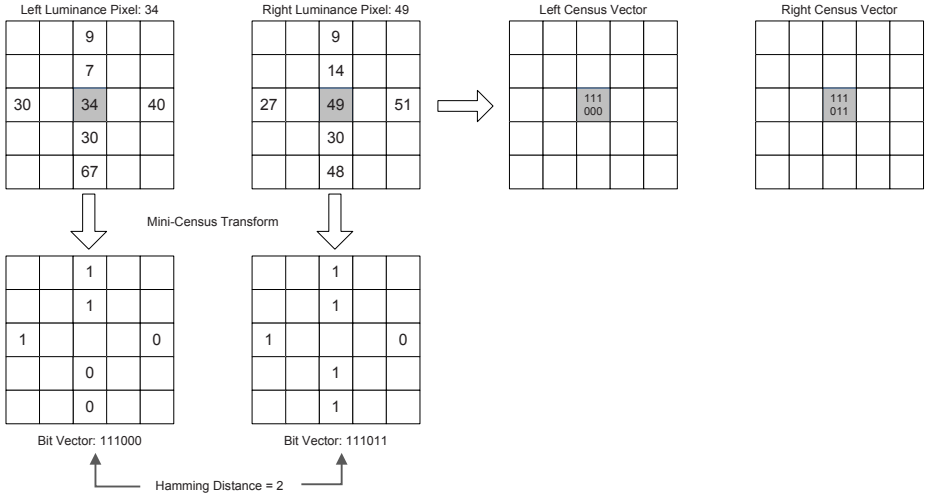

Figure 3.10: Mini-Census transform

In the transform, each square in the census window represents a (median filtered) 8-bit luminance pixel, but only the pixel locations marked with values are considered for the two example center pixels. If the luminance of a neighbor pixel under consideration is larger than that of the center pixel, the corresponding bit is set to '0', otherwise '1'. All of the comparison results form a 6-bit vector as the output of a mini-census transform. In the later *Raw Cost Scatter* module, the cost function is to compute the hamming distance between two hypothesis pixel's mini-census vectors.

Using census transform and hamming distance as cost function has been proved to be very robust to radiometric distortions [19], and much more hardware-friendly than other methods e.g., *Rank filter* and *Normalized cross-correlation (NCC)*. Another important advantage of census transform is that it saves two valuable bits by converting an 8-bit luminance pixel to a 6-bit census vector. In SAD based cost computation, the maximum raw cost of two pixels is 255 (8-bit storage), while in hamming distance calculations the maximum raw cost is only 6 (3-bit storage). If also consider the cost aggregation over a support region, the mini-census transform significantly reduces the memory requirements.

3.2.2.2 Adaptive Support Region Construction

Aggregating matching cost in an adaptive support region is the key idea of the Variable-Cross algorithm. The approach of adaptive support region construction is to decide an upright cross for each pixel $p(x_p, y_p)$ in the input frame. As shown in Figure 3.11, this pixel-wise adaptive cross consists of two orthogonal line segments, intersecting at the anchor pixel p. The cross of each pixel is defined by a quadruple$\{h_p^-, h_p^+, v_p^-, v_p^+\}$ that denotes the left, right, up, and bottom arm length, respectively (see Figure 3.12). Since the cross-based support region is a general concept, there are a variety of approaches to

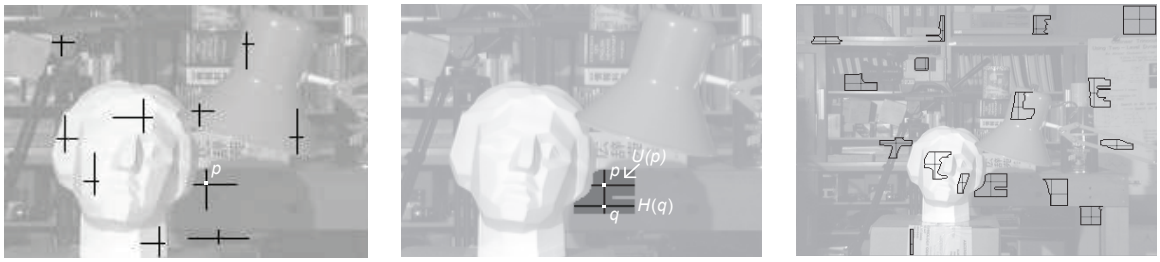

Figure 3.11: Local adaptive cross and support regions on an image
(a) A pixel-wise adaptive cross defines a local support skeleton for the anchor pixel, e.g., p. (b) A shape adaptive full support region $U(p)$ is dynamically constructed for the pixel p, integrating multiple horizontal line segments $H(q)$ of neighboring crosses. (c) Sample shape adaptive local support regions, approximating local image structures appropriately.

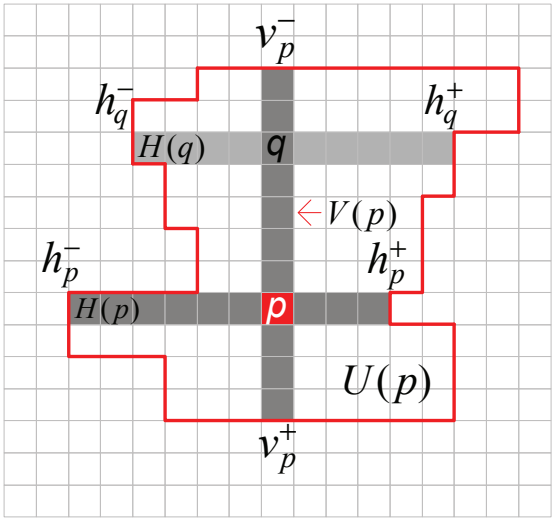

Figure 3.12: Adaptive support region representation
It shows the configuration of a local upright cross $H(p) \cup V(p)$ for the anchor pixel p, and the constructed full support region $U(p)$. Pixel q is on the vertical segment $V(p)$.

decide the four adaptive arm lengths. Here we implement an efficient approach based on luminance similarity: for each anchor pixel $p(x_p, y_p)$ in the frame, luminance difference evaluations are performed in the four directions on its consecutive neighboring pixels in order to find out the largest span r^* in each direction. The computation of r^* is formulated in Equation 3.2.

$$r^* = \underset{r \in [1,L]}{Max} \big(r \prod_{i \in [1,r]} \delta(p, p_i) \big) \tag{3.2}$$

In the equation $p_i = (x_p - i, y_p)$ and L is a preset maximum allowed directional arm length; $\delta(p_1, p_2)$ is an indicator function evaluating the luminance difference between

the pixel p_1 and p_2 based on a given threshold value τ.

$$\delta(p_1, p_2) = \begin{cases} 1, & |Y(p_1) - Y(p_2)| \leq \tau \\ 0, & \text{otherwise} \end{cases} \tag{3.3}$$

The value of τ is a run-time configurable parameter that controls the confidence level of luminance similarity. It is set empirically by the end user according to different applications. Its effect on the benchmark images are discussed in Chapter 5.

Based on the four arms $\{h_p^-, h_p^+, v_p^-, v_p^+\}$, the adaptive cross of pixel p are given by Equation 3.4.

$$\begin{cases} H(p) = \{(x, y) | x \in [x_p - h_p^-, x_p + h_p^+], y = y_p\} \\ V(p) = \{(x, y) | x = x_p, y \in [y_p - v_p^-, y_p + v_p^+\} \end{cases} \tag{3.4}$$

The shape adaptive full support region $U(p)$ of each frame pixel is therefore built by integrating all the horizontal segments $H(p)$ of the pixels residing on the vertical segment $V(p)$.

The output of the support region construction module is a data word containing the four arm lengths $\{h_p^-, h_p^+, v_p^-, v_p^+\}$, which are used in the *Cost Aggregation* and *Disparity Voting* modules.

3.2.2.3 Combined Hardware Processing

Because both of the mini-census transform and the adaptive support region construction are window-based processing, and they all operate on the same data stream (filtered stereo frames), they are designed to share the same line and window buffers. The left and right frames are again processed independently in parallel.

In order to compute all the four arm lengths simultaneously, $2 \times L_{max}$ lines of the filtered frame are stored in line buffers and a shift register matrix of $(2 \cdot L_{max} + 1) \times (2 \cdot L_{max} + 1)$ is used for providing all the neighboring pixels for the centered anchor pixel. The configurations and interconnections of line buffer memory blocks are similar with these in the median filter module (see Figure 3.7), with different number of blocks. One important hardware parameter in this module is the maximum allowed arm length L_{max}, which is set to 15 in our implementation. Its effect on the matching accuracy is duscussed in Chapter 5. To make clear illustrations in Figure 3.13 L_{max} is set to 7.

In Figure 3.13, the centered pixel's mini-census transform and support region are ready to be computed. Computing the census transform is straightforward: the luminance values of shaded pixels are compared with the center pixel's luminance concurrently and the census vector is obtained by packing the result bits in a certain order. The four arm lengths in the center pixel's adaptive cross are also computed in parallel; take the h_p^+ for example, first all 7 pixels in the anchor pixel's right direction are evaluated concurrently and the corresponding $\delta(p, p_i)$ values are obtained. Then the packed $\delta(p, p_i)$ values are sent to a priority encoder that performs the function of Equation 3.2.

Because the input data stream to this module carries consecutive video frames, pixels in this window are possibly not form the same frame. To solve the problem, pixel row

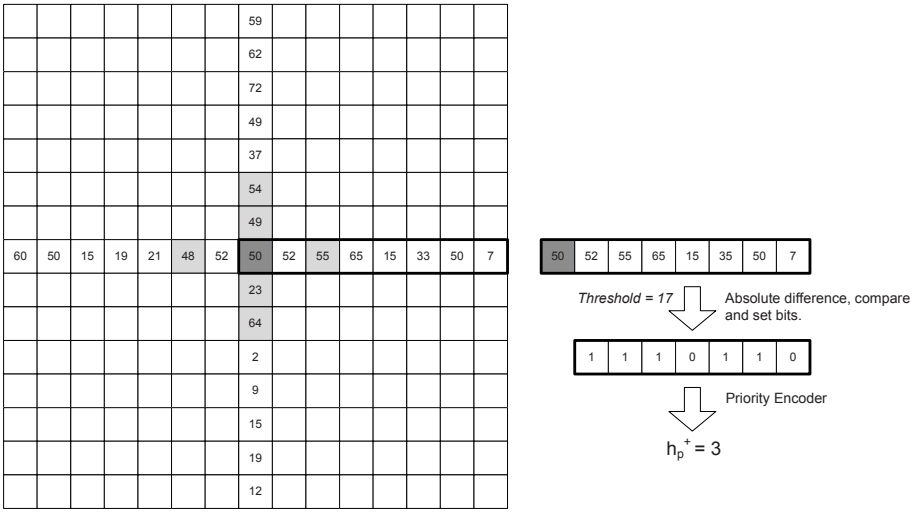

Figure 3.13: Census transform and adaptive support region construction window

and column counters are also implemented in this module. The outlier pixels in this window are masked before computing the census vector and support regions.

The pipeline latency of the census transform and adaptive support region construction module is given by Equation 3.5.

$$Latency_{ctsr} = 15 \times frame\ width + C_{ctsr} \tag{3.5}$$

3.2.3 Output Logic

The output logic packs all the data words produced by the Pre-Processor and prepare additional data fields required by the following Stereo-Matcher processor. Content of the 40-bit output data word is listed below, from most significant bit to least. A data field followed by 'L' indicates a value from the left frame, and 'R' indicates a value from the right frame, with the same pixel coordinates.

- 4-bit all '0'
- 6-bit mini-census vector L
- 8-bit horizontal arms (h_p^- and h_p^+) L
- 6-bit mini-census vector R
- 8-bit horizontal arms ($h_{p'}^-$ and $h_{p'}^+$) R
- 8-bit vertical support region segment (v_p^- and v_p^+) L

Because we set the maximum allowed arm length to 15, a 4-bit word is sufficient to represent any value in the quadruple$\{h_p^-, h_p^+, v_p^-, v_p^+\}$. The 4-bit all '0' bits are simple used to pad the data word width to a multiple of 8, as required by the Avalon bus.

3.3 Design of the Stereo-Matcher Pipeline

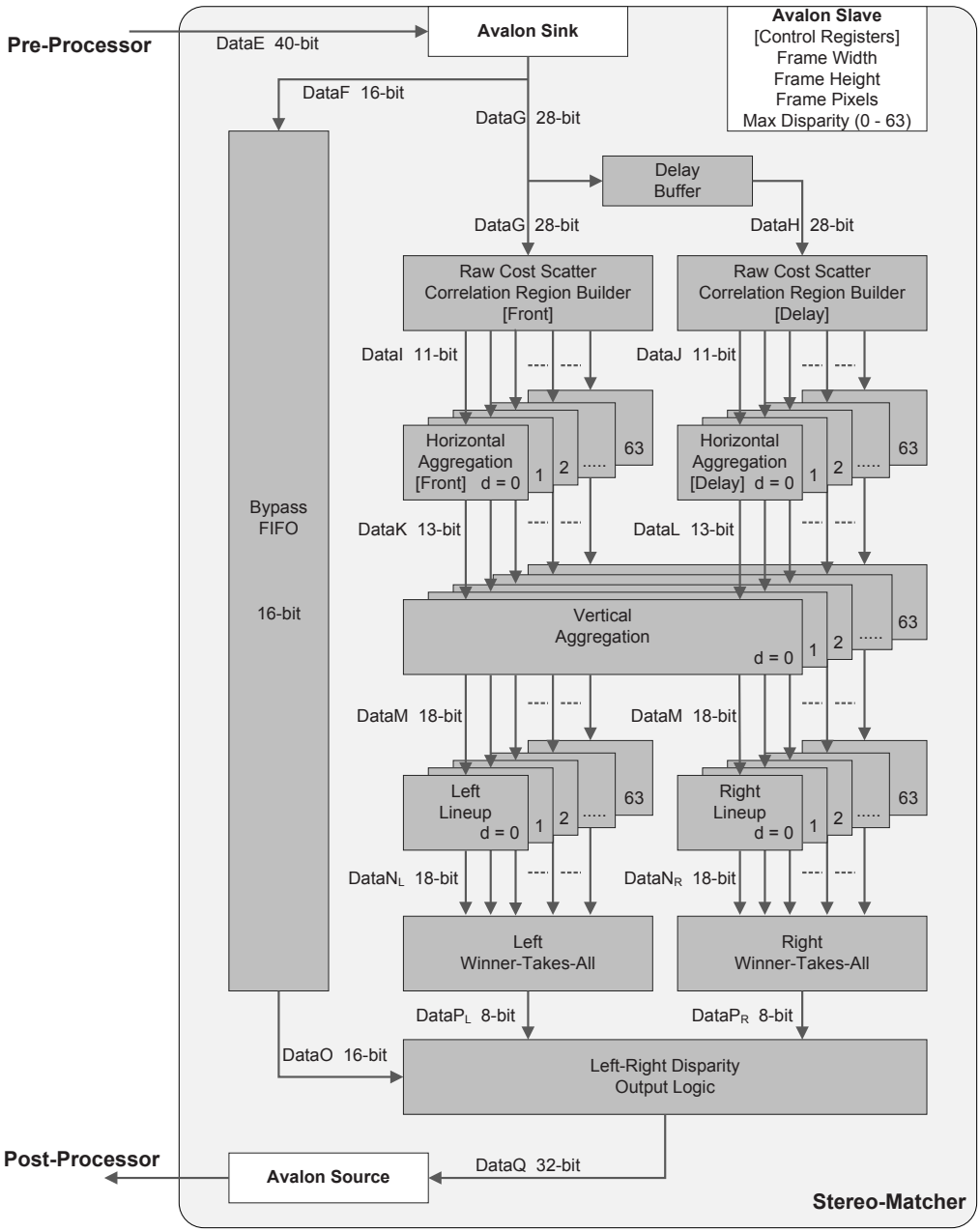

Figure 3.14: Stereo-Matcher processor internal processing pipeline

Overview of the internal processing pipeline of the Stereo-Matcher processor is shown in Figure 3.14. The Stereo-Matcher processor operates on the data stream coming from Pre-Processor (DataE). The data from the Pre-Processor is always associated with a new stereo frame pair and passes through each pipeline stage in the Stereo-Matcher. Once the final results (L-R disparity maps) are available, the Stereo-Matcher sends

them to the Post-Processor. In the figure, before the vertical aggregation step there are two major data paths, denoted by *Front* and *Delay* respectively. They work for the data reuse technique implemented in the vertical aggregation. After the vertical aggregation step, there are still two major data paths, but now they are associated with the left and right disparity map respectively.

As a local stereo matching algorithm, the key emphasis of its processing is to find out the best correspondence for a pixel $p(x, y)$ in the left frame, from a local area in the right frame. Because the stereo frames are already rectified, this local area shrinks to a 1-dimensional segment of pixels located on the same scanline (see Figure 3.15). Given

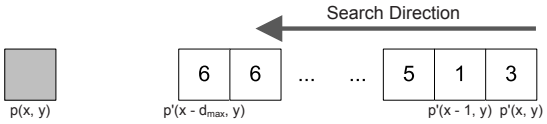

Figure 3.15: 1-dimensional local search area
The number at each pixel location indicates the raw matching cost

a pair of hypothetical correspondences, i.e., $p(x, y)$ in the left frame and $p'(x', y')$ in the right frame, we define the following variables before inferring the best correspondence. Here the coordinates of p and p' are correlated with a hypothetical disparity d ranging from 0 to d_{max}(the maximum disparity under consideration), i.e., $x' = x - d$ and $y' = y$.

- *Raw Matching Cost*: the Hamming distance between their Mini-Census vectors i.e., $HammingDist\big(cv_p(x, y), cv_{p'}(x', y')\big)$
- *Correlation Region*: the overlapped area of the two pixels' support regions
- *Aggregated Matching Cost*: the accumulated raw matching costs in their correlation region i.e., $AggCost(x, y, d)$
- *Aggregated Pixel Count*: the total number of pixels in their correlation region i.e., $PixCount(x, y, d)$
- *Averaged Matching Cost*: the aggregated cost divided by the aggregated pixel count in their correlation region i.e.,

$$AvgCost(x, y, d) = \frac{AggCost(x, y, d)}{PixCount(x, y, d)} \tag{3.6}$$

Therefore the best correspondence is defined by the hypothetical pair that gives the minimum averaged matching cost. The hypothetical disparity d that leads to the minimum averaged cost is recorded as the result disparity $d_p(x, y)$ for pixel $p(x, y)$. All the computed disparities form the disparity map for the left frame.

For a given pixel $p(x, y)$ in the left frame, cost aggregations in correlation regions with different hypothetical disparities are independent on each other; so it is possible to compute them all in parallel, and all the aggregated costs and pixel counts are available

simultaneously as the output of all parallel computing threads. Therefore the minimum averaged cost is picked out using the *Winner-Takes-All* method. To make this parallel computing successful, two key problems have to be solved:

1. Raw matching costs are provided for all computing threads simultaneously.

2. All computing threads are fully pipelined to match the input pixel data rate.

Solutions to the two problems are explained in Section 3.3.1 and Section 3.3.2 respectively. The Stereo-Matcher processor provides disparity maps for both left and right stereo frames, which are sent to the Post-Processor for *L-R Consistency Check* that detects mismatched correspondences caused by occlusion. The modules for generating both disparity maps are introduced in Section 3.3.3. We also present a method with aggregation data reuse technique in Section 3.3.2, which significantly reduces the required line buffer memories.

3.3.1 Raw Cost Scatter and Correlation Region Builder

The task of this module is to provide raw matching costs for all hypothetical disparities simultaneously. Input data stream to this module carries the census vectors $cv_p(x, y)$ and $cv_{p'}(x', y')$ for pixel $p(x, y)$ in the left frame and pixel $p'(x', y')$ in the right frame, respectively. Because the x' coordinate of pixel p' ranges from x to $x - d_{max}$ for all hypothetical disparities, an array of shift registers are used to buffer the full disparity range for pixel p. Assuming $d_{max} = 3$, as shown in Figure 3.16, in a certain pipeline

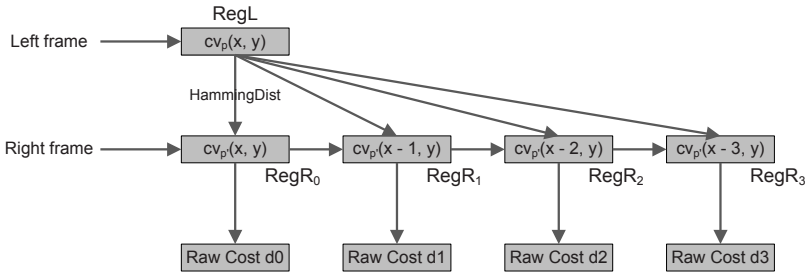

Figure 3.16: Scattering raw matching costs using shift register arrays

cycle t, the census vector $cv_p(x, y)$ arrives in the $RegL$ while in the same cycle the census vector $cv_{p'}(x, y)$ also arrives in $RegR_0$; meanwhile all the census vectors stored in the $RegR$ array shift to the right for one position. Therefore the desired Hamming distances $HammingDist\big(cv_p(x, y), cv_{p'}(x-d, y)\big)$, with d ranging from 0 to 3, are available simultaneously by comparing all registers' content in $RegR$ array with the content in $RegL$.

Similarly, the horizontal arms (h_p^-, h_p^+, $h_{p'}^-$, $h_{p'}^+$) for both p and p' are buffered using the same technique. Since the correlation support region is defined as the overlapped area

of two pixel's support regions, the horizontal arms in each correlation support region (h_{pc}^-, h_{pc}^+) are obtained by selecting the shorter arm.

$$h_{pc}^- = \begin{cases} h_p^-, & h_p^- \leq h_{p'}^- \\ h_{p'}^-, & \text{otherwise} \end{cases} \quad h_{pc}^+ = \begin{cases} h_p^+, & h_p^+ \leq h_{p'}^+ \\ h_{p'}^+, & \text{otherwise} \end{cases}$$

As a result, this module provides 4 parallel streams of data, carrying the raw matching costs and horizontal arms in scanline order for 4 hypothetical disparities. The 4 streams are scattered into 4 parallel data paths, which lead to the follow-up cost aggregation modules. In the next cost aggregation step we use fixed vertical arms for correlation region, i.e., $v_{pc}^- = v_{pc}^+$. The adaptive vertical arms v_p^- and v_p^+ are used in the later disparity voting step; they are put in the FIFO to bypass the cost aggregation modules.

The latency of this module is a small constant, depending on the implemented pipeline stages.

$$Latency_{rcs} = C_{rcs} \tag{3.7}$$

3.3.2 Parallelized Cost Aggregations

After the raw cost scatter module, cost aggregations are able to be performed on all raw cost streams in parallel, and mutually independently by $(d_{max} + 1)$ parallel computing threads. For each thread, aggregating all the raw costs in a correlation region is achievable through the line buffer plus register matrix configuration used in the Pre-Processor. But the arbitrarily shaped 2D support region makes this solution not hardware friendly. We solve this problem by decomposing the 2D aggregation into two orthogonal 1D aggregations, i.e., a horizontal step followed by a vertical one (see Figure 3.17).

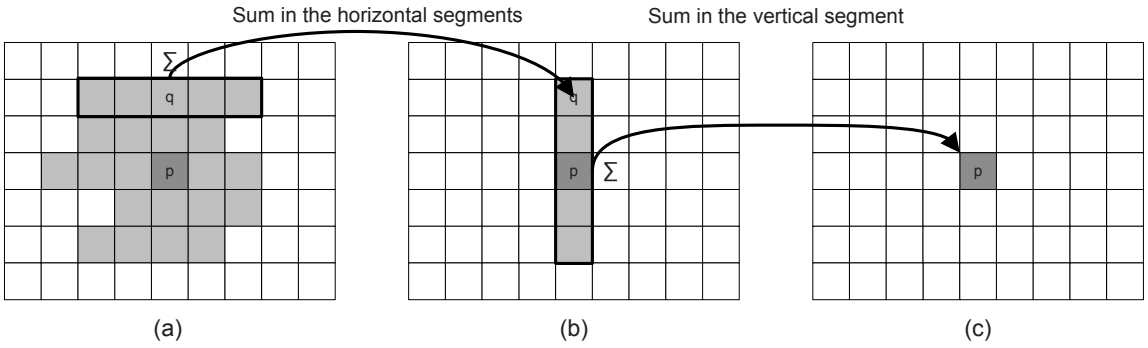

Figure 3.17: Two orthogonal 1D aggregations

Figure 3.17 (a) shows the correlation region of pixel p (shaded region), and q indicates a pixel lying in the vertical segment of p. First we aggregate all the raw costs in the horizontal segment of pixel q. As pixel q slides along the vertical segment of pixel p, we obtain the aggregated cost in the horizontal segment of each q. Next we aggregate all the horizontally aggregated costs along the vertical segment of pixel p, as shown in

Figure 3.17 (b - c). The aggregated cost by this two orthogonal 1D process is exactly identical with the aggregation result in the 2D region, and meanwhile this technique also simplifies the hardware implementation.

Accordingly we divide one cost aggregation module into two sub-modules, namely *Horizontal Cost Aggregation* and *Vertical Cost Aggregation*; they are introduced in the following two sub-sections.

3.3.2.1 Horizontal Cost Aggregations

The input data stream to a horizontal aggregation module carries the raw matching cost and horizontal arms (h_{pc}^{-}, h_{pc}^{+}); and the output of this module is the partially aggregated cost in the horizontal segment of each left frame pixel.

The horizontal partial aggregation for each pixel is performed using integral costs. As

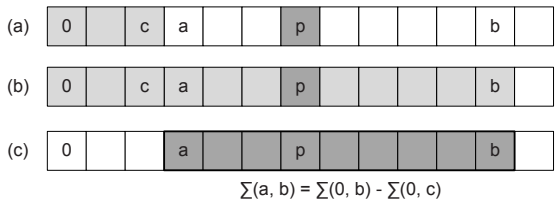

Figure 3.18: Integral costs and horizontal cost aggregation

shown in Figure 3.18, assuming pixel p has horizontal arms $h_{pc}^{-} = 3, h_{pc}^{+} = 5$, we first calculate the integral for each pixel in a scanline, and then the horizontally aggregated cost of pixel p equals to the difference of the two integrals indicated by h_{pc}^{-} and h_{pc}^{+}.

Since the input raw costs come in scanline order, the integral computing unit is implemented with an accumulator, which sends its output to an array of shift registers for concurrently accessing the two integrals randomly chosen by h_{pc}^{-} and h_{pc}^{+}. The hardware logic diagram is shown in Figure 3.19, assuming the maximum arm length L_{max} is 15. A shift register array of length $(2 \times L_{max} + 1)$ is needed for buffering all possible integrals to determine the horizontally aggregated cost. Once the raw cost of pixel p arrives in $Reg14$, its horizontal arms h_{pc}^{-} and h_{pc}^{+} also come out from the delay line; therefore the two boundary integrals indicated by the two arms are selected by the two *(L + 1)-to-1* multiplexers. The difference of the two multiplexer outputs is the desired horizontally aggregated cost, and the pixel count in the horizontal segment is obtained by $h_{pc}^{-} + h_{pc}^{+}$.

The latency of this module is mainly determined by the maximum arm length L_{max}, because an incoming raw cost has to arrive in $Reg14$ before its associated aggregation being able to be computed.

$$Latency_{hagg} = L_{max} + C_{hagg} \tag{3.8}$$

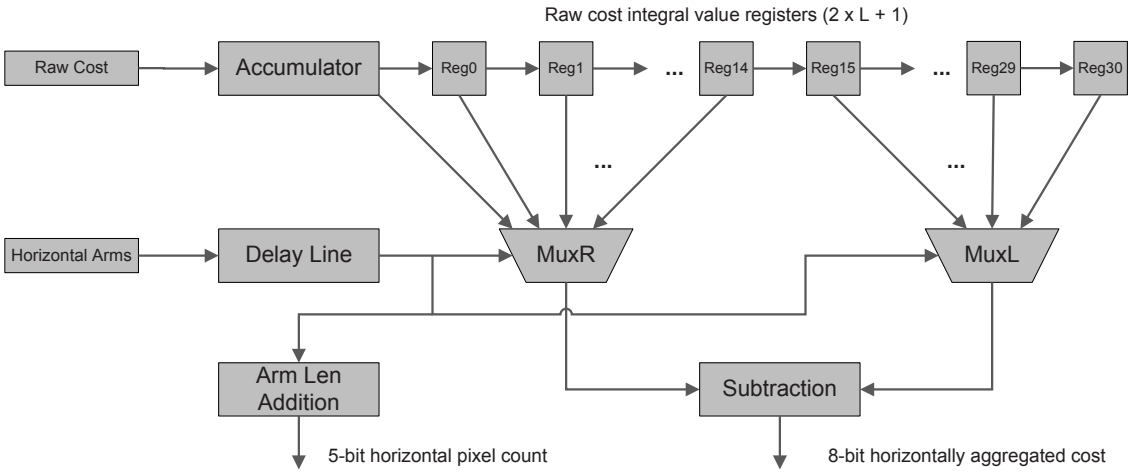

Figure 3.19: Horizontal cost aggregation logic

3.3.2.2 Vertical Cost Aggregations

The input data stream to a vertical aggregation module carries the horizontally aggregated costs and pixel counts associated with a certain hypothetical disparity; and the output of this module is the fully aggregated cost and pixel count in the correlation region of each left frame pixel.

Because the input data still comes in scanline order, and the vertical aggregation requires a vertical data vector for each pixel, line buffers are employed for converting the input data stream into a vertical vector stream. In contrast with the horizontal aggregation, vertical aggregation modules use fixed aggregation span for each pixel instead of the adaptive vertical arms. This approach has significantly reduced the line buffers needed for each vertical aggregation module, with negligible influence on the disparity map quality. Based on our experiments, a vertical aggregation span (V_{span}) of 5 (see Figure 3.17 (b)) gives the best trade-off between matching accuracy and hardware complexity. The vertical aggregation span covers the anchor pixel's scanline and the adjacent lines defined by v_{pc}^{-} and v_{pc}^{+}. For fixed aggregation span $v_{pc}^{-} = v_{pc}^{+} = v_{pc}$ and we define V_{span} with Equation 3.9.

$$V_{span} = v_{pc}^{-} + v_{pc}^{+} + 1 = 2 \times v_{pc} \tag{3.9}$$

The V_{span} is a hardware parameter that determines the depth of the *Delay Buffer* and pipeline latency. The effect of V_{span} on matching accuracy is introduced in Chapter 5.

Moreover, with fixed vertical aggregation span a more efficient data reuse technique is applicable. The concept of this technique is illustrated in Figure 3.20. In the figure the pixels $p(x, y)$, $q(x, y + 1)$ and $r(x, y + 2)$ are located in the same vertical column. And their correlation regions are shaded in the illustration. Obvioudly, the aggregated

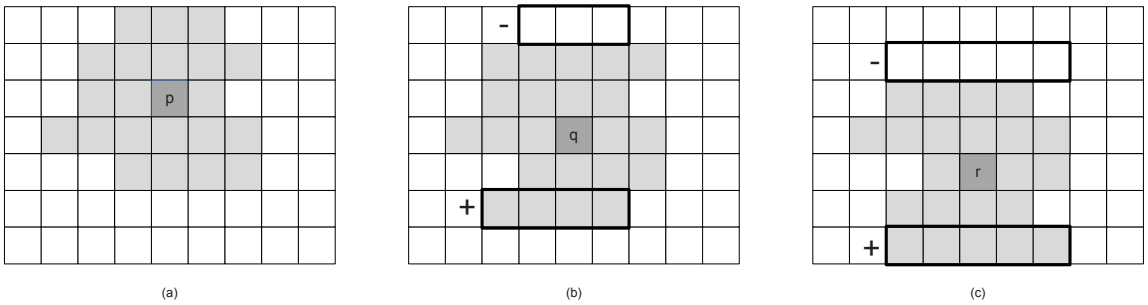

Figure 3.20: Aggregation data reuse in vertical aggregation
(a): pixel $p(x, y)$ and its correlation region. (b): pixel $q(x, y + 1)$ and its correlation region. (c): pixel $r(x, y + 2)$ and its correlation region

costs of pixel $q(x, y + 1)$ and $r(x, y + 2)$ fulfill the Equation 3.10.

$$\begin{aligned} AggCost(x, y+1, d) &= AggCost(x, y, d) \\ &+ AggCost_h(x, y+3, d) \\ &- AggCost_h(x, y-2, d) \\ AggCost(x, y+2, d) &= AggCost(x, y+1, d) \\ &+ AggCost_h(x, y+4, d) \\ &- AggCost_h(x, y-1, d) \end{aligned} \tag{3.10}$$

In Equation 3.10 the $AggCost_h(\cdot)$ returns the horizontally aggregated cost in the bordered region. Therefore the vertically aggregated cost is reused for vertically adjacent pixels.

Without the data reuse method, each vertical aggregation module only requires 4 line buffers to provide a vertical vector, and the fully aggregated cost in a correlation region is given by summing up the values in the vertical vector. Figure 3.21 shows the line buffers and aggregation logic configuration without data reuse.

With the data reuse method, the line buffer memories are required to store only one line of data. The horizontal aggregations are provided by two data paths accordingly. As shown in Figure 3.14, a *Delay Buffer* is applied to provide the delayed data flow. In case the vertical aggregation span is 5, the delay buffer should be large enough to store 5 stereo image lines. In contrast with the distributed line buffers in vertical aggregation modules, the delay buffer is not duplicated; as a result the overall on-chip memory consumption is significantly reduced by applying the data reuse method. Figure 3.22 shows the line buffer and aggregation logic configuration with data reuse.

The expense coming with the data reuse method is that it requires two blanking lines after each frame in order to generate the valid output of the two bottom lines in the frame. Nevertheless two lines is trivial for common image sizes, especially for high-definition images and videos. The latency of this module is given by Equation 3.11.

$$Latency_{vagg} = 2 \times frame\ width + C_{vagg} \tag{3.11}$$

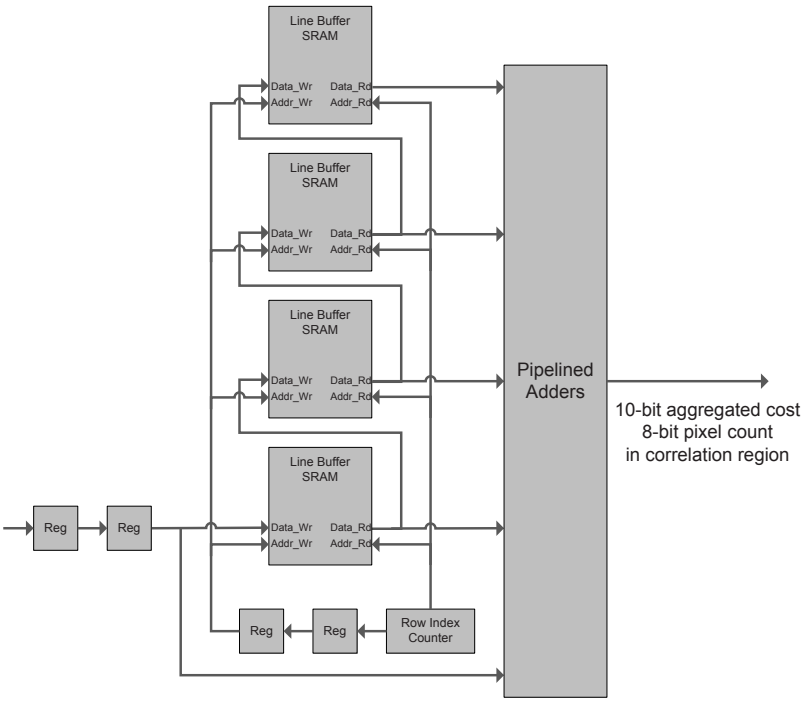

Figure 3.21: Vertical cost aggregation without data reuse

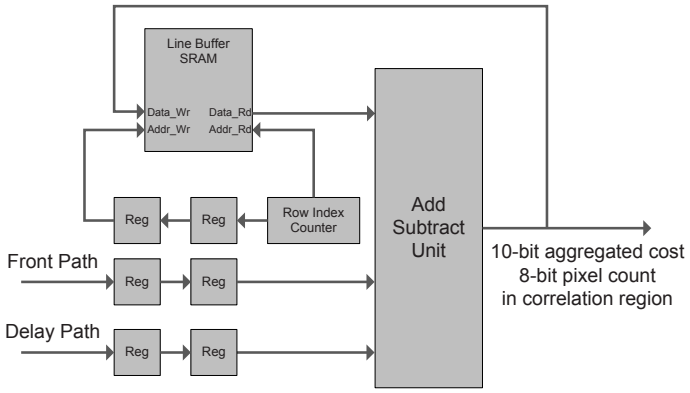

Figure 3.22: Vertical cost aggregation with data reuse

After the initial pipeline latency, in each follow-up pipeline cycle an aggregation module delivers aggregated cost and pixel count in a correlation region associated with a certain disparity. The aggregation modules are fully pipelined so that in each cycle the result is also associated with a new pixel, in scanline order. The data flow is illustrated in Figure 3.23, assuming $d_{max} = 4$ and 4 parallel computing threads are implemented.

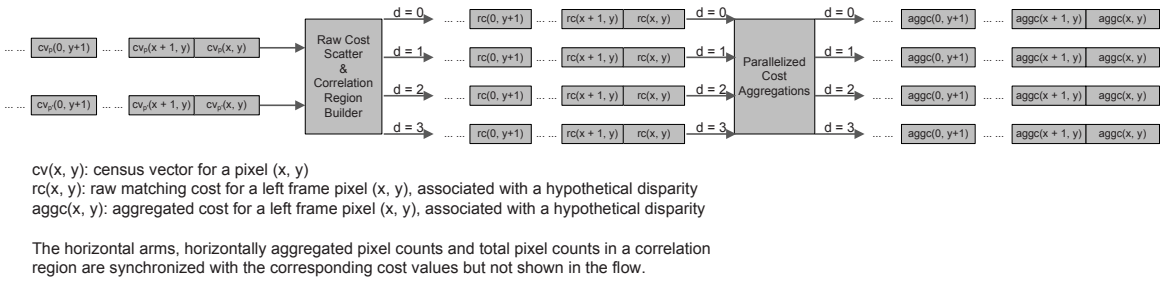

Figure 3.23: Pipelined data flow

3.3.3 Winner-Takes-All and L-R Disparity Maps

The averaged cost $AvgCost(x, y, d)$ associated with each hypothetical d is ready to be computed with the input streams to this module. A straightforward approach to get an averaged value is to implement a divider for each thread. However dividers are quite expensive for FPGA in terms of occupied area and hardware resources; moreover, a divider also causes result precision problems. So instead of using dividers in each parallel thread, we propose using *multiply-subtract* functions to determine the minimum averaged cost. The idea is based on the following equivalence.

$$\frac{AggCost(x, y, d_0)}{PixCount(x, y, d_0)} < \frac{AggCost(x, y, d_1)}{PixCount(x, y, d_1)}$$

$$\Leftrightarrow$$

$$AggCost(x, y, d_0) \times PixCount(x, y, d_1) < AggCost(x, y, d_1) \times PixCount(x, y, d_0)$$

So given $(d_{max} + 1)$ pairs of $AggCost(x, y, d)$ and $PixCount(x, y, d)$, a tree structure with *multiply-subtract* functions and multiplexors selects the minimum $AvgCost(x, y, d)$, as shown in Figure 3.24, assuming $d_{max} = 4$. Here each select

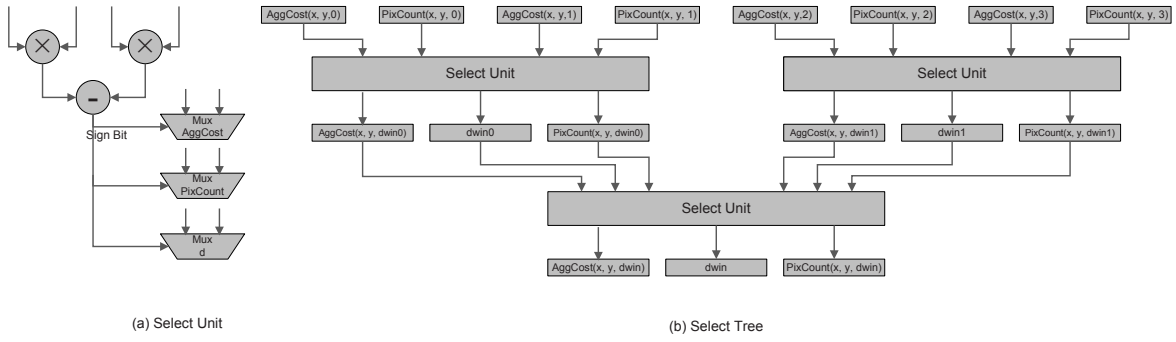

Figure 3.24: Winner-Takes-All select tree

unit is comprised of a multiply-subtract unit and three multiplexers that track the $WinnerAggCost, PixCount$ and associated d respectively. The multiply-subtract unit

maps perfectly to the dedicated DSP block in our FPGA, with a pipeline latency of only 1 cycle.

In a certain pipeline cycle t, the computing threads give $(d_{max}+1)$ aggregated costs and pixel counts based on pixel $p(x,y)$ in the left frame and the set of pixels in the right frame: $p'(x-d_{max},y)$ ... $p'(x,y)$. In the next cycle $t+1$, aggregated costs are given by pixels $p(x+1,y)$ and $p'(x-d_{max}+1,y)$ ... $p'(x+1,y)$. As a result, the aggregated costs based on pixel $p'(x,y)$ in the right frame and the set of pixels in the left frame: $p(x,y)$ ... $p(x+d_{max},y)$ are available from cycle t to cycle $t+d_{max}$. This reversed

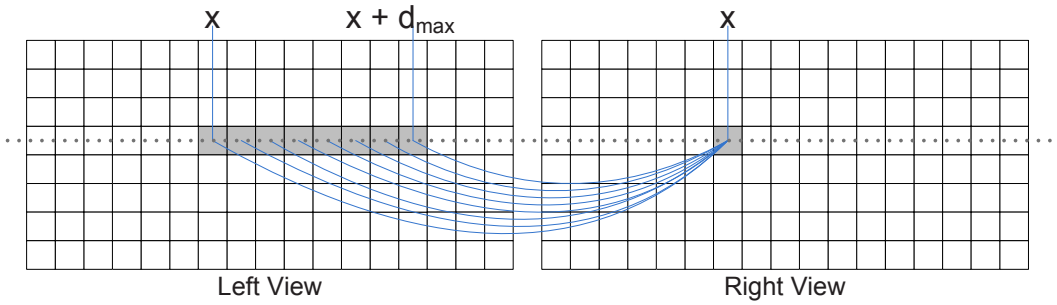

Figure 3.25: Compute the disparity map of the right image

search (see Figure 3.25) provides the result disparity $d_{p'}(x,y)$ for pixel $p'(x,y)$ in the right frame. Similarly, $d_{p'}(x,y)$ is selected by the minimum averaged cost associated with pixel $p'(x,y)$ and the set of pixels $p(x,y)$ ... $p(x+d_{max},y)$. The right disparity map is generated for a *L-R Consistency Check*, i.e., if pixel $p(x,y)$ in the left frame finds its best correspondence $p'(x-d_p(x,y),y)$ in the right frame, then $p'(x-d_p(x,y),y)$ should also map back to pixel $p(x,y)$ with its disparity $d_{p'}\big(x-d_p(x,y),y\big)$. This relationship is formulated by Equation 3.12.

$$d_{p'}\big(x-d_p(x,y),y\big)=d_p(x,y) \tag{3.12}$$

The consistency check is performed in the Post-Processor, and here it is important to generate the right disparity map. Selecting the disparity for pixel $p'(x,y)$ is similar to the process of selecting the disparity for pixel $p(x,y)$, only with different set of aggregated costs and pixel counts. As discussed above, the required aggregated costs and pixel counts for pixel $p'(x,y)$ come consecutively in several pipeline cycles, so we place a *Lineup Buffer* before the right winner-takes-all module to store them and provide all the required ones simultaneously in a certain cycle. To keep the result disparities for $p(x,y)$ and $p'(x,y)$ synchronized, we also use a similar lineup buffer for the left frame data. Configuration of the lineup buffers are shown in Figure 3.26, with $d_{max}=4$ and 4 parallel threads. In the configuration, each column is a small dual-port memory block that buffers the aggregated costs and pixel counts generated in 4 consecutive cycles. The aggregation results are written into them with a circular addressing pattern. After 4-cycle latency, all the required data for pixel $p'(x,y)$ is available in the right lineup buffer, with an incremental address pattern; meanwhile, all data for pixel $p(x,y)$ is also available in the left lineup buffer, but with the same read address. Read address patterns for the left and right lineup buffers are shaded in Figure 3.26.

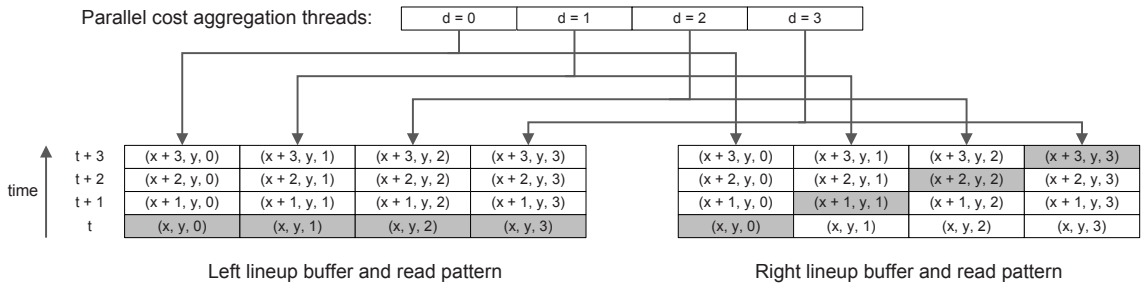

Figure 3.26: Lineup buffers and read patterns

Readout data from the left and right lineup buffers are sent to left and right winner-takes-all modules respectively, and the computed *winner* left-right disparities are synchronized at the output ports of the two modules.

The latency of this winner-takes-all and L-R disparity map generator is a small constant, depending on the implemented pipeline stages.

$$Latency_{wta} = C_{wta} \tag{3.13}$$

3.3.4 Output Logic

The final output of this Stereo-Matcher to the Post-Processor is a 32-bit data word, including the following fields.

- 8-bit horizontal arms (h_p^- and h_p^+) L
- 8-bit vertical arms (v_p^- and v_p^+) L
- 8-bit disparity map result L i.e., $d_p(x, y)$
- 8-bit disparity map result R i.e., $d_{p'}(x, y)$

The right disparity map is used for checking mismatched correspondence in the *L-R Consistency Check* module, and the support regions for the left frame pixels are used in the *Disparity Voting* module to improve the disparity map quality. The L-R disparity maps generated so far is shown in Figure 3.27.

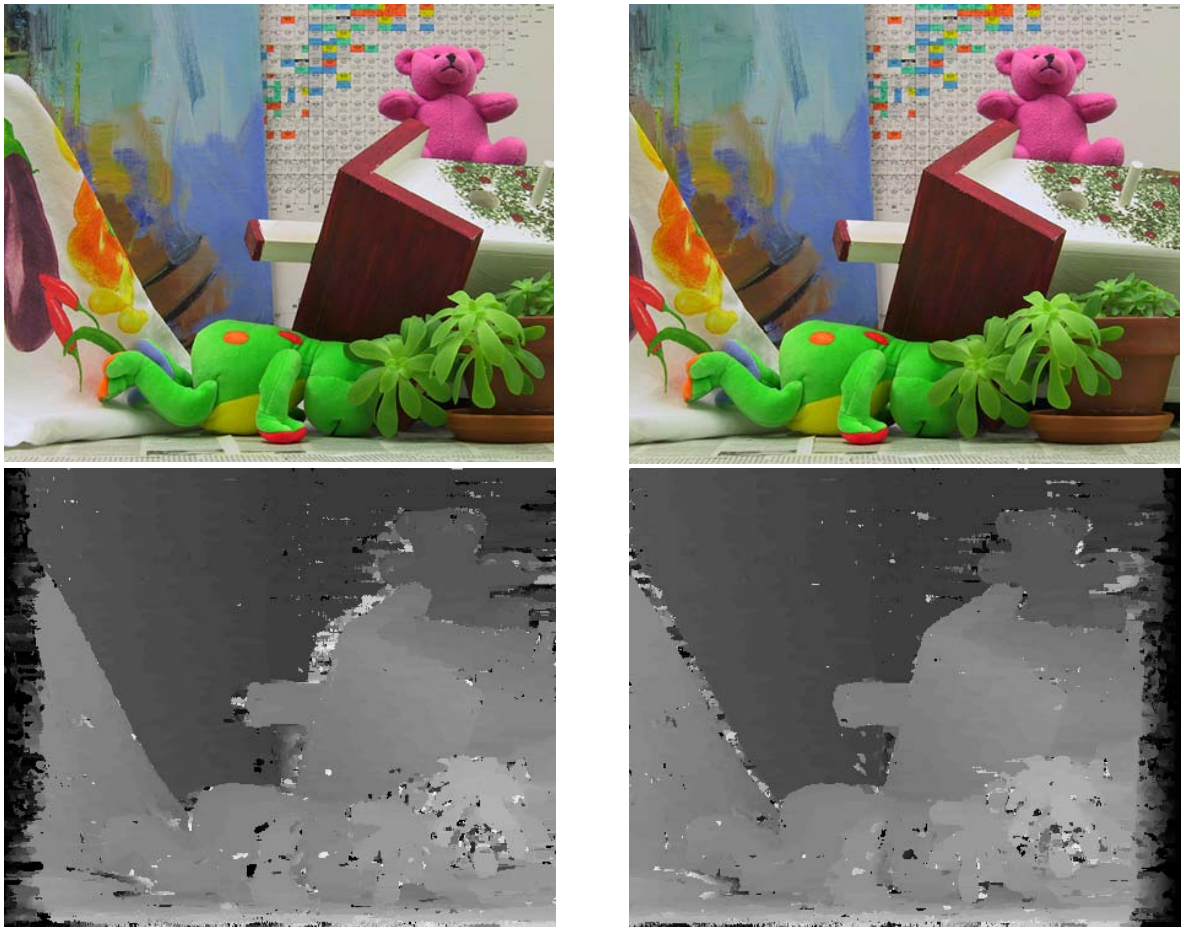

Figure 3.27: L-R disparity maps generated by the Stereo-Matcher

3.4 Design of the Post-Processor Pipeline

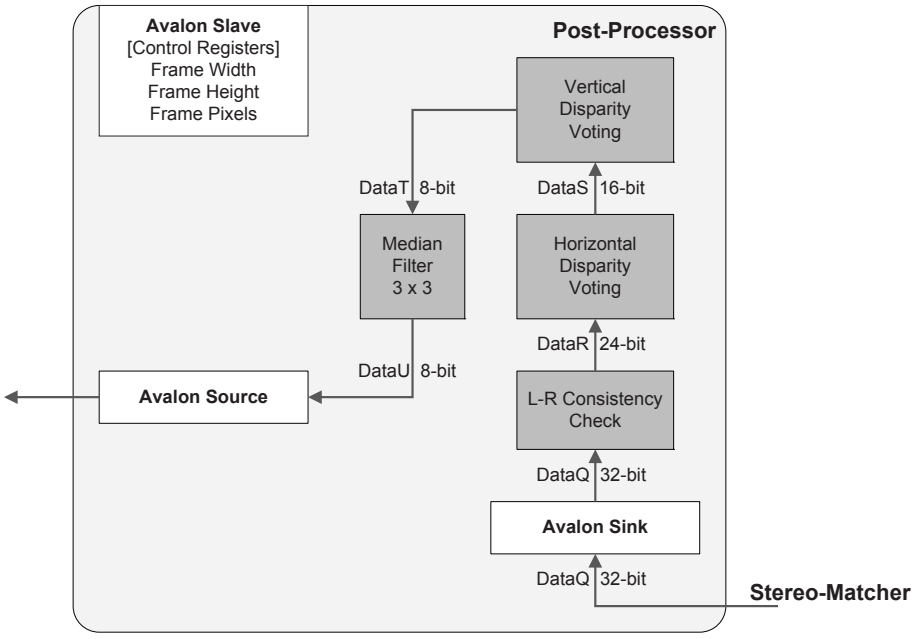

Figure 3.28: Post-Processor internal processing pipeline

The internal processing pipeline of the Post-Processor is shown in Figure 3.28. It takes the L-R disparity maps and support regions of the left frame pixels defined by the quadruple$\{h_p^-, h_p^+, v_p^-, v_p^+\}$. The right disparity map is used to check disparity consistency, and correct mismatched correspondences in the left disparity map. The double-checked left disparity map is sent to the following disparity voting modules to further refine its quality. Finally the refined left disparity map passes through a median filter to remove its speckles. The median filter gives the final disparity map associated with the left frame, which is used for higher level applications such as view interpolation or feature detection.

3.4.1 L-R Consistency Check

The task of the L-R consistency check module is to testify whether the two disparity maps satisfy Equation 3.12 i.e.,

$$d_{p'}\big(x - d_p(x,y), y\big) = d_p(x,y)$$

If $d_p(x,y)$ and the corresponding $d_{p'}\big(x - d_p(x,y), y\big)$ satisfies this equation, $d_p(x,y)$ is considered as a valid disparity; otherwise $d_p(x,y)$ is regarded as a mismatch caused by occlusions. In the latter case $d_p(x,y)$ is replaced by a closest valid disparity.

The input data stream to this module contains synchronized disparities of the left and right frame respectively, as shown in Figure 3.29. To check Equation 3.12 for each

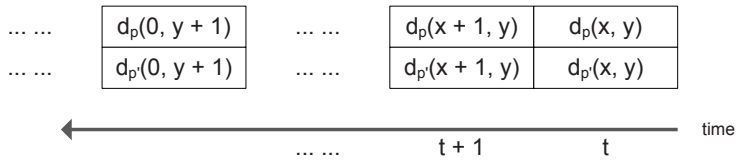

Figure 3.29: Synchronized L-R disparity data word

$d_p(x, y)$ in the left disparity map, we first buffer a series of both left-right disparities, as shown in Figure 3.30. The depth of each disparity buffer should be no less than $(d_{max} + 1)$. When $(d_{max} + 1)$ disparities are buffered, the *Read Address Generator* provides the read address for $d_{p'}\big(x - d_p(x, y), y\big)$, according to $d_p(x, y)$. The *Consistency Check* block checks whether the two disparities are identical; if so, $d_p(x, y)$ is considered as valid, and registered as the latest valid disparity; otherwise, $d_p(x, y)$ is replaced by the last registered value. The *Delay Line* is used to compensate for the address generator and buffer read latency.

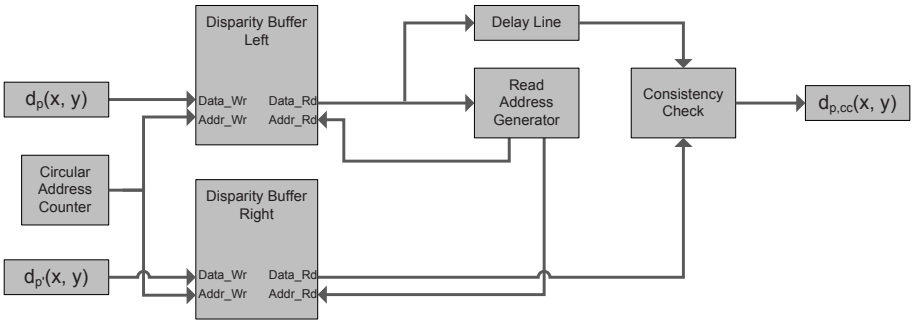

Figure 3.30: L-R consistency check logic

The latency of this L-R consistency check module is determined by the maximum allowed disparity range and a few constant pipeline stages.

$$Latency_{lrcheck} = D + C_{lrcheck} \tag{3.14}$$

Output of this L-R consistency check module is the corrected left disparity map $d_{p,cc}(x, y)$, and the right disparity map is discarded. The corrected left disparity map is shown in Figure 3.31.

3.4.2 Disparity Voting

Disparity voting is to search for the most frequent disparity in a pixel's support region as the statistically optimized disparity for the pixel. Precisely, this corresponds to build a histogram for all disparities $d_{p,cc}(x, y)$ in the support region of pixel $p(x, y)$ (see Figure 3.32), and pick out the disparity associated with the peak histogram bin. The philosophy behind this computation is that pixels contained in a local support

Figure 3.31: Left disparity map after consistency check

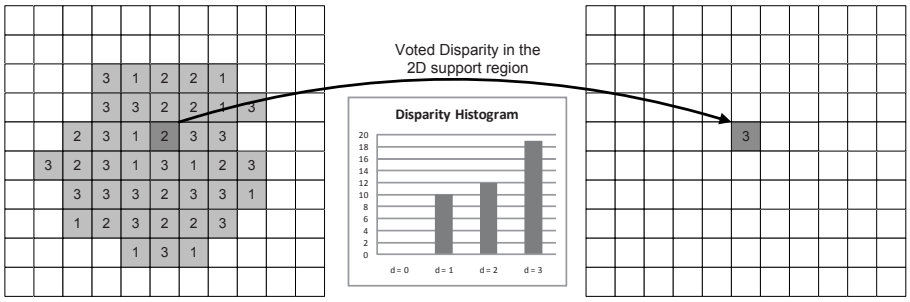

Figure 3.32: Disparity histogram and voted disparity

region mostly originate from the same scene patch, and hence they very likely share similar disparities. Experiments show that this processing has considerably improved the disparity map quality.

Like cost aggregations, the disparity voting also performs on an arbitrarily shaped 2D support region, which complicates the hardware design. To simplify this problem, here we again decompose the voting in a 2D region into 2 $\times$ 1D voting processes (see Figure 3.33); i.e., the *Horizontal Disparity Voting* module first searches for the most frequent disparity in each horizontal segment in the support region of pixel $p(x, y)$, then the *Vertical Disparity Voting* picks out the most frequent one in the vertical segment. This decomposing method does not necessarily provides the same result as the voting in the 2D region, but it is able to give the same result in most cases. Experiments show that it only provides a slightly different disparity map compared with the 2D voting result.

3.4.2.1 Horizontal Disparity Voting

The horizontal disparity voting module is comprised of $(d_{max} + 1)$ horizontal voting units. Each of them is responsible for building the histogram bin for a certain dispar-

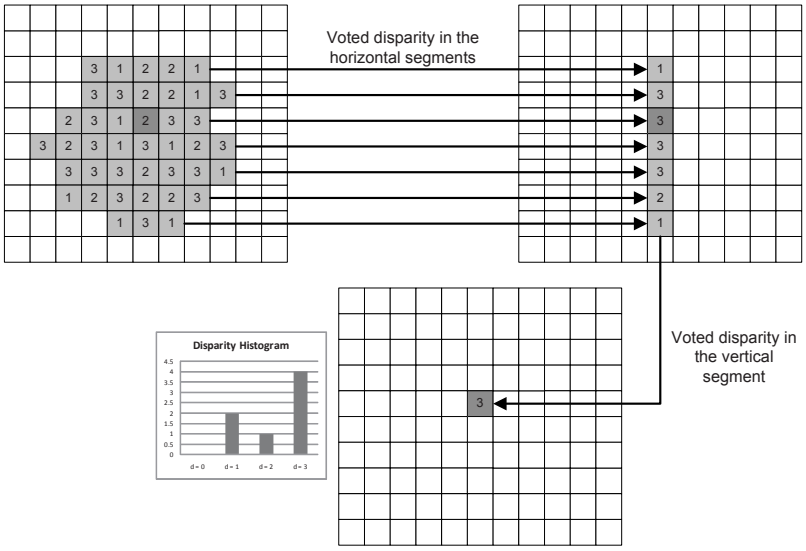

Figure 3.33: 2 × 1D orthogonal voting method

ity. The voting unit for disparity 0 is shown in Figure 3.34, assuming the maximum arm length $L_{max} = 15$. The input to such a unit is a stream containing the consistency-

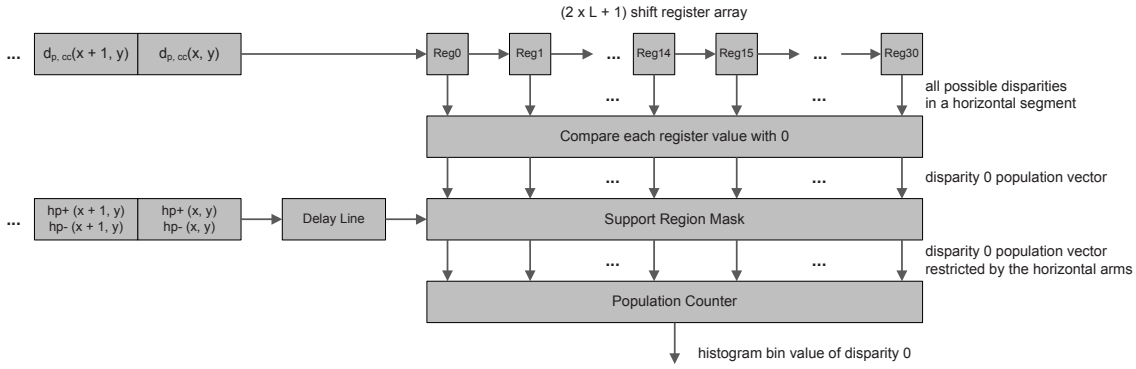

Figure 3.34: Horizontal voting unit for disparity 0

checked disparity $d_{p,cc}(x, y)$ and the corresponding horizontal arms (h_p^-, h_p^+). The disparities are buffered in a shift register array that has the capacity to store all the possible disparities in the horizontal segment specified by h_p^- and h_p^+, which are accessible simultaneously when $d_{p,cc}(x, y)$ arrives in $Reg15$. In the voting unit for disparity 0, all shift register values are compared with 0, and the related result bit is set to '1' if equal, otherwise '0'. All outputs of the comparators form a *population vector* that has the same number of '1' bits as the number of disparity 0. In the next *Support Region Mask* stage, the '1' bits that do not belong to the $h_p^- \leftrightarrow h_p^+$ segment are inverted by a mask vector given by h_p^- and h_p^+. As a result, the mask unit outputs the population vector for disparity 0 in the horizontal segment of pixel $p(x, y)$; and the actual number is obtained by the follow-up *Population Counter* that counts the number of '1' bits in

the population vector.

Similarly, each of the $(d_{max} + 1)$ horizontal voting units gives the histogram bin value of the corresponding disparity. So the disparity associated with the histogram peak is obtained using a comparator tree. The latency of this module is determined by the maximum arm length L_{max} and a few constant pipeline stages.

$$Latency_{hvote} = L_{max} + C_{hvote} \tag{3.15}$$

The output of the horizontal disparity voting module is the stream that carries the voted disparity in the horizontal segment of each pixel $p(x, y)$.

3.4.2.2 Vertical Disparity Voting

The vertical disparity voting module employs the same voting units as the ones used in the horizontal disparity voting unit, and each unit counts the number of a certain disparity in the vertical segment of pixel $p(x, y)$. Because the incoming data come in scanline order, disparities in the vertical segment of pixel $p(x, y)$ are presented using line buffers instead of shift register array. The line buffer configurations and voting logic are shown in Figure 3.35, assuming the maximum arm length $L_{max} = 15$. The

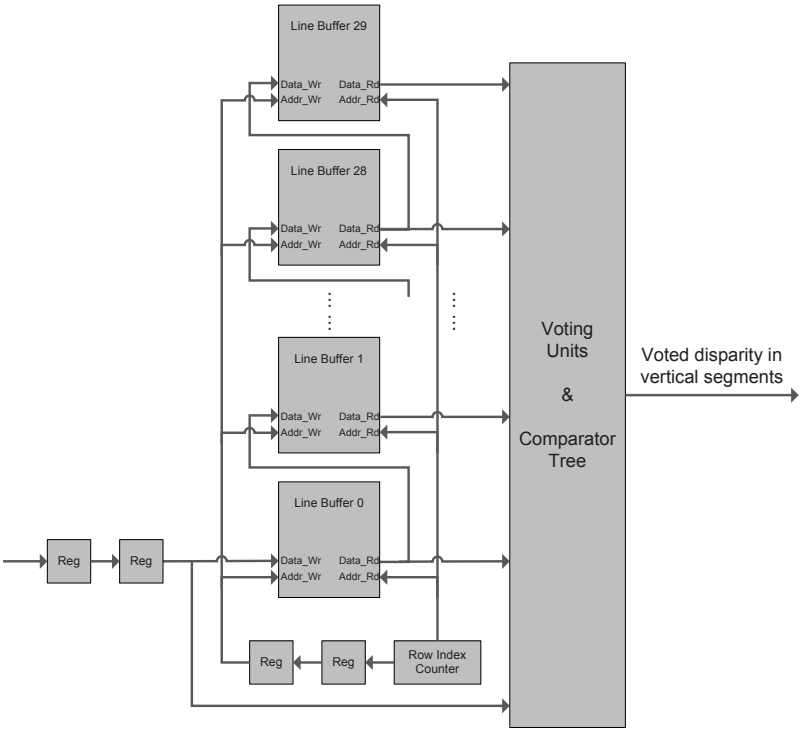

Figure 3.35: Vertical voting logic and line buffers

latency of the vertical voting module is given by Equation 3.16.

$$Latency_{vvote} = 15 \times frame\ width + C_{vvote} \tag{3.16}$$

The vertical disparity voting module gives the result of the 2 × 1D voting method, and the disparity map after this stage is considerably improved. The voted disparity map is shown in Figure 3.36. To compare with the 2D voting method, the 2D voted result is also presented.

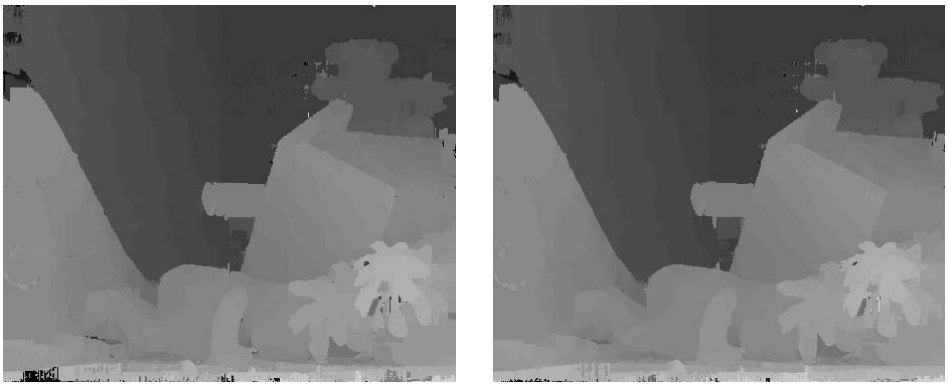

Figure 3.36: Voted disparity maps

3.4.3 Median Filter and Output

The final median filter, though not critical, is employed to to increase the reliability and accuracy by removing some spike points in the disparity map. The median filter concludes the whole pipeline of the Post-Processor and the final disparity map associated with the left frame is output (See Figure 3.37).

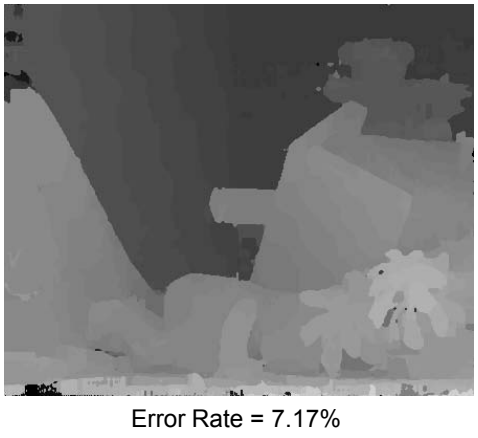

Figure 3.37: Final disparity map

Stereo Matching SoC Implementations on FPGA

4

We propose two SoC implementations to verify and evaluate the stereo matching pipeline design. Their major difference is whether to use a result frame buffer to store the final disparity maps in the external memory or not. Figure 4.1 illustrates the implementation with result frame buffer and two *Scatter-Gather DMAs (SG-DMA)* to transfer data between the FPGA and the external memory. Another implementation

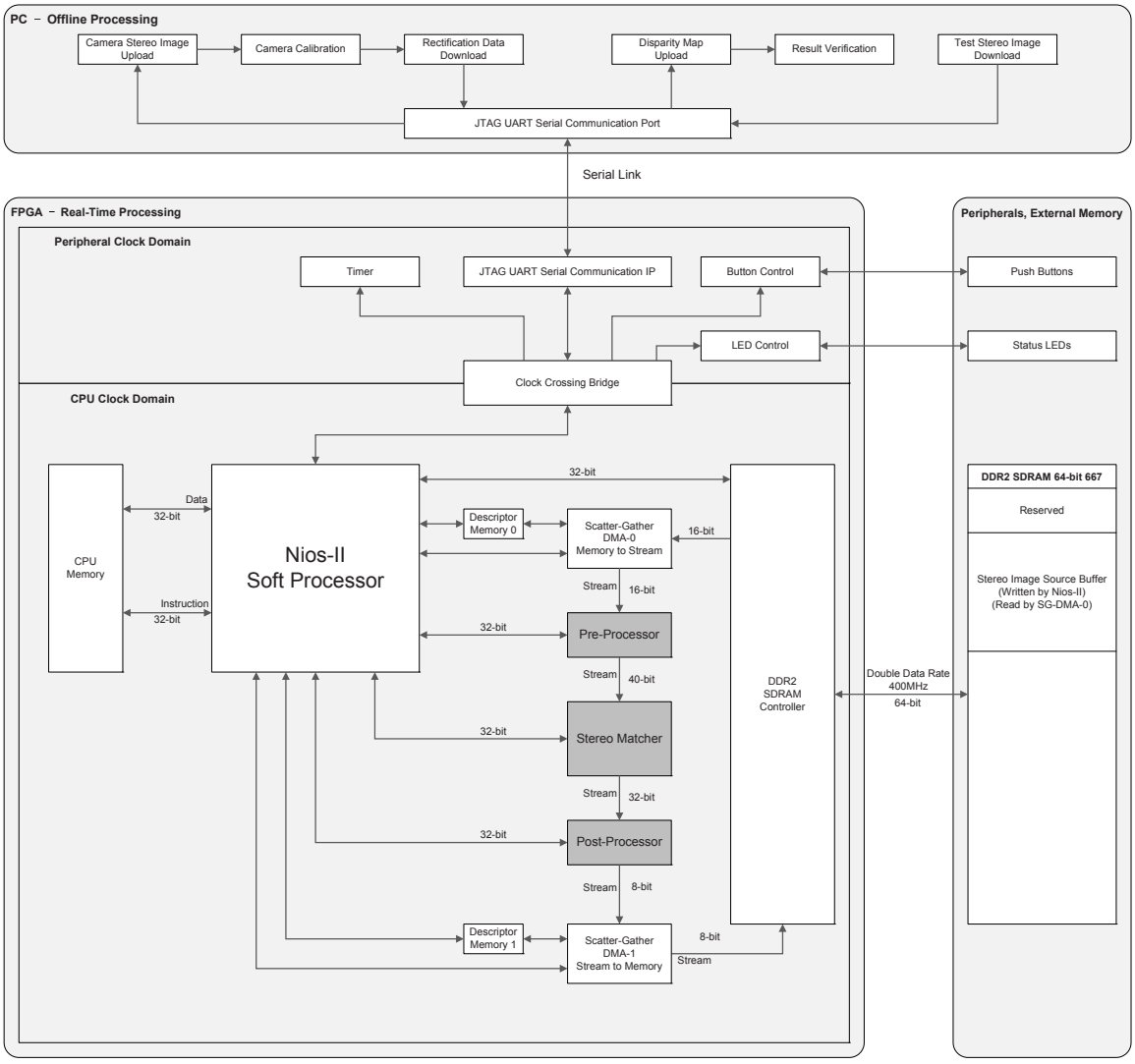

Figure 4.1: Proposed SoC with both source and result frame buffer

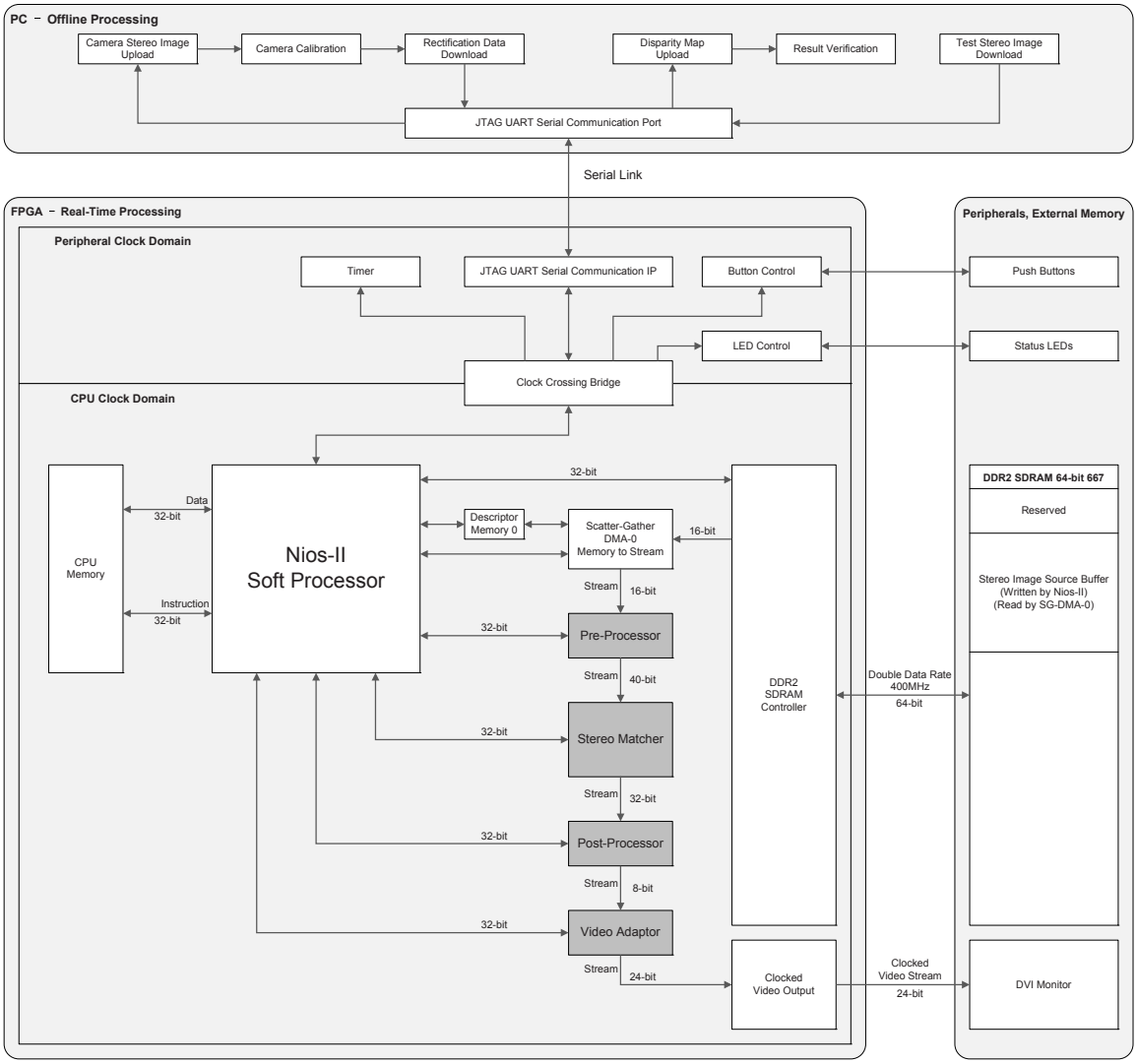

Figure 4.2: Proposed SoC with source frame buffer and display

without the result frame buffer but with a DVI-display interface is shown in Figure 4.2. Their functional differences are introduced in Section 4.2 and Section 4.3. The proposed system is divided into two computing platforms, the PC and the FPGA board, which will be introduced in Section 4.1. We also discuss the system interconnections, memory and bandwidth usage estimations in this chapter.

4.1 Overview of the Proposed System

The communication between the PC and the FPGA is based on a *JTAG UART* link. JTAG UART is a simplified version of the standard UART interface and implemented over a USB connection. As a simplified UART, the JTAG UART does not provide

adjustable baud rates and its data rate is limited. As a result, this link is not suitable for real-time image and video communications; but due to its simplicity, we choose it as a transceiver for non-real-time validation.

4.1.1 The PC Tasks

The desktop PC works as the host development and evaluation platform. In the development phase, the EDA software e.g., Quartus-II and Nios-II IDE running on the PC configures and debugs the FPGA hardware and software respectively. In the evaluation phase, main tasks of the PC are shown in the top part of Figure 4.1 and Figure 4.2.

The first task of the PC is to calibrate the stereo cameras. A PC program receives a sequence of stereo image pairs captured by the cameras through the JTAG UART link, analyzes the images and extracts the required rectification data. The essential output of the calibration program is the rectification matrices, which are used to rectify real-time stereo videos to make them satisfy epipolar geometry. The rectification matrices are downloaded to the DDR2-SDRAM on the FPGA board through the JTAG UART link again. Since the focus of this design work is stereo matching, image acquisition and rectification modules are not implemented.

Another task of the PC is to verify and evaluate the FPGA processing results. A PC program downloads test image pairs to the FPGA board and retrieves the resulting disparity maps, which are compared with the results produced by PC software algorithm to verify the FPGA hardware development.

4.1.2 The FPGA Tasks

The FPGA chip contains the core subject of this design and implementation, which is categorized into two parts: digital hardware and embedded software. The whole design on FPGA is built into an Altera SOPC[1] environment, which is a Nios-II soft processor integrated with memories, peripherals and system interconnections. Altera has provided abundance of SOPC component cores e.g., block memory, JTAG UART, DMA and SDRAM controller together with its *SOPC Builder* tool. The SOPC components are connected with each other through the *Avalon System Interconnect Fabric* [7]. Our proposed stereo matching hardware is also developed as SOPC Builder compatible components connected with other SOPC peripherals as well as the Nios-II processor.

Most hardware IPs, including our stereo matching cores, provide a memory-mapped interface that is accessible by software running on the Nios-II processor. Several internal registers are employed to control each core and monitor its working status. The FPGA is responsible for all the real-time processing including stereo matching and result display.

[1] System on a Programmable Chip

4.2 Introduction to the SoC Hardware

As shown in Figure 4.1 and Figure 4.2, the FPGA hardware is split in two clock domains: the CPU clock domain and the peripheral clock domain. In our implementation the CPU clock domain is working at 100MHz, which contains more time-critical components than those in the 50MHz peripheral clock domain. This split allows Quartus-II to optimize the hardware placement and routing. Components in the two clock domains communicate with each other through a clock crossing bridge.

In the peripheral clock domain, the push-button and LED interface are implemented using Altera *Parallel IO Core*; they are responsible for receiving configuration signals from the push-buttons and showing the system status on the LEDs. The JTAG UART core is connected with the PC through a USB cable and exchanges non-real-time data with the PC-side program. A timer is also implemented in this domain to work as a performance evaluation tool. In our future developments, the camera interfaces and rectification module will also be implemented in this domain. The rectified stereo frames are written to the DDR2-SDRAM for the follow-up stereo matching processing.

The CPU clock domain contains the Nios-II soft processor that regulates the whole system behavior. It is equipped with a dedicated on-chip memory block that contains complete embedded software instructions and run-time program data. The DDR2-SDRAM interface is implemented using Altera *DDR2-SDRAM High-Performance Controller Core*, which is a memory-mapped slave component in the SOPC and provides accesses for multiple masters. In addition, a PLL is also implemented in this controller by SOPC Builder; the PLL works as the clock source for the CPU clock, peripheral clock and DDR2 memory interface clock.

Our proposed stereo matching hardware pipeline lies in the CPU clock domain, comprised of the aforementioned three coprocessors: *Pre-Processor*, *Stereo-Matcher* and *Post-Processor*. Each of them has one memory-mapped slave port and several unidirectional streaming ports. The memory-mapped ports provide accesses for Nios-II CPU to configure computing parameters e.g., frame resolution, thresholding values at run-time. The streaming ports work as data highway for transporting image/video data and processing results around these blocks.

The three coprocessors are not granted direct accesses to the DDR2-SDRAM controller, but provided by the SG-DMAs. Since the stereo matching cores mostly require continuous pixel data reads/writes, implementing a random-addressing master interface for them to access the external memory is not necessary, which complicates the stereo matching hardware design and makes it more error-prone. Compared with traditional DMA cores, which only queue one transfer command from the CPU at a time, the SG-DMA is able to read a series of descriptors stored in the *Descriptor Memory* and execute data transactions accordingly, without additional intervention from the CPU. The descriptor series are organized as a linked list and created by the CPU at the beginning of an automatic transaction sequence; each descriptor specifies the source, destination and how much data to be transferred. Besides traditional memory-to-memory transfers, an SG-DMA also supports memory-to-data stream, data stream-to-memory transfers, which perfectly match the proposed continuous stereo matching data flow.

In Figure 4.1, two SG-DMAs are implemented to transfer the stereo images and disparity maps respectively. This implementation is used to measure the processing speed with both source and result frame buffers. Since the stereo matching coprocessors are all fully pipelined, they do not have internal processing bottlenecks; so their actual performance depends on the availability of the data source or sink, which is determined by the efficiency of SG-DMAs and the DDR2-SDRAM controller and possible external memory accessing competitions. Once the data source or the sink is not ready for providing/receiving new data, the full stereo matching pipeline also has to stall in order to wait for them.

The system shown in Figure 4.2 aims at showing the disparity maps directly on a standard desktop monitor. Off-the-shelf DVI monitors support several standard refresh rates e.g., 640 × 480 @ 75Hz and 1024 × 768 @ 60Hz with certain pixel clocks. Without a disparity map frame buffer, the stereo matching processing must be synchronized with the corresponding pixel clock. In practice the stereo matching cores still work at 100MHz and the *Clockd Video Output* core is used for the synchronization with a clock-domain crossing FIFO. It back-pressures the stereo matching pipeline when the FIFO is full. This implementation removes the result frame buffer and as a result the external memory accessing competitions are reduced. Potentially this implementation is able to offer better processing throughput compared with the Figure 4.1 system, but if the display refresh rate is less than the pipeline frame rate, the pipeline has to stall.

4.3 Introduction to the SoC Software

The embedded software runs on the Nios-II CPU, a general-purpose RISC processor designed by Altera for FPGA implementations. Software in our proposed system is responsible for the following tasks:

- Communicate with the PC program.

  Communications with the PC program exchange non-real-time data between PC and FPGA through the JTAG UART interface.

- Control and monitor the SOPC peripherals.

  This is achieved by accessing peripheral registers through memory-mapped interfaces.

- Generate and update descriptor memories for corresponding SG-DMAs.

  An Scatter-Gather DMA performs multiple transactions as specified by the descriptors stored in its descriptor memory. After one descriptor is processed, or after a linked list of descriptors are processed, the SG-DMA core generates an interrupt request to Nios-II CPU, and the CPU calls the related *Interrupt Service Routine (ISR)* to update its descriptor memory if necessary.

- Configure stereo matching parameters.

Stereo matching parameters e.g., frame resolution and thresholding values are configurable at run-time. Control signals are specified by the user through push-buttons and handled by the push-button interrupt service routine.

- Evaluate the system performance.

  System performance e.g., stereo matching frame rates are measured by the timer and analyzed by the embedded software.

The software footprint is small enough to be completely implemented on-chip, and a dedicated memory block is employed for all the instruction and data storages. The embedded software is implemented with a round-robin with interrupts architecture [36]. In this architecture, interrupt service routines deal with the most urgent requests of the hardware and set flags; the main loop polls flags and does any follow-up processing indicated by the flags.

In our system, interrupt service routines mainly serve for SG-DMAs and push-buttons. Since SG-DMAs have to deal with the most urgent stereo matching data, their interrupts are assigned with the highest priority. The 2 SG-DMAs all have the same priority and they are served based on *first-come, first-served* policy. Other tasks in the main loop all have the same priority which is lower than all interrupts; different tasks are handled in a round-robin fashion. The flow chart of task code in the main loop is shown in Figure 4.3.

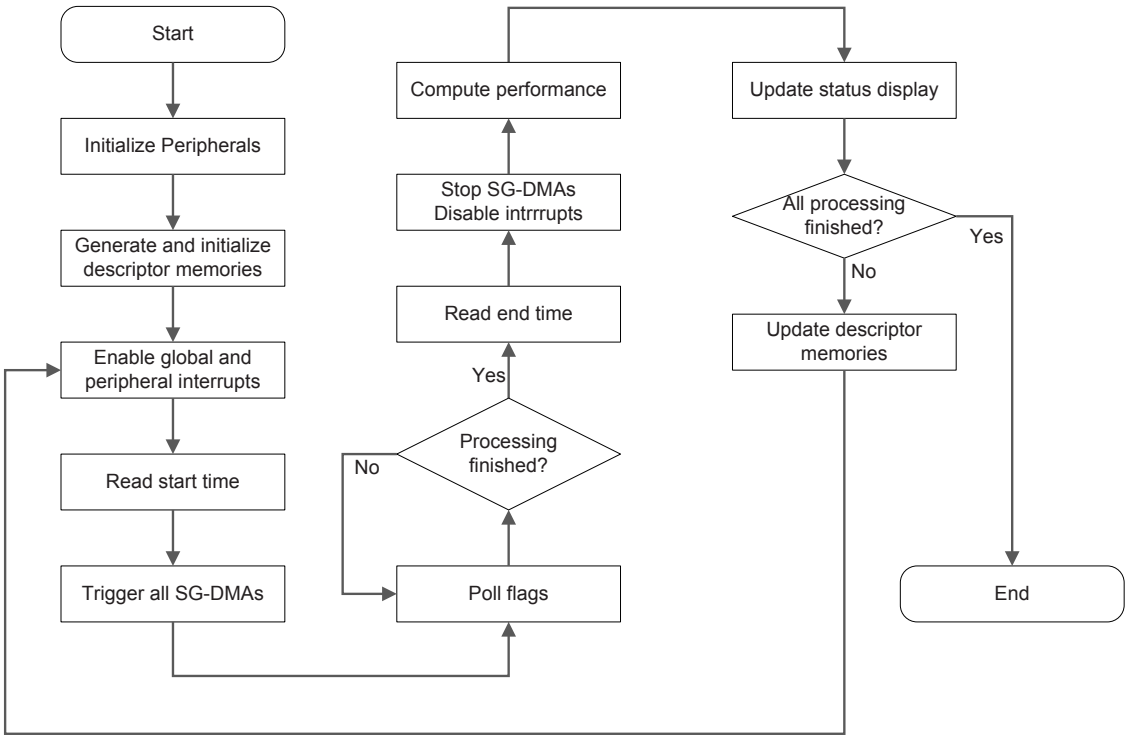

Figure 4.3: Flow chart of the main loop tasks

4.4 System on a Chip Interconnections

We have chosen the *Avalon System Interconnect Fabric* as the SoC interconnections. It is a collection of interconnect and logic resources defined by Altera, which connects components in an SOPC Builder system. The interconnect fabric is comprised of two standards [7]: Avalon memory-mapped (Avalon-MM) and Avalon streaming (Avalon-ST). We have implemented both for different data transfer requirements. Their properties are introduced in the following subsections.

4.4.1 Avalon Memory Mapped Interconnect

The Avalon-MM interconnect is a partial crossbar switch connecting memory-mapped slaves and masters. Unlike a full crossbar switch, which allows any number of masters to connect any number of slaves, the partial crossbar only make connectivities between masters and a selected number of slaves. Non-active connections in a full crossbar are removed to reduce the usage of FPGA logic and routing resources. Also in contrast with a shared bus structure, a partial crossbar allows concurrent transactions issued by different masters, as long as they do not access the same slave. For masters accessing the same slave simultaneously, the SOPC Builder generated arbiter logic performs slave side arbitration. By default, the arbitration algorithm provides equal fairness for all masters accessing a particular slave, and accesses are granted in a round robin fashion. Prioritized arbitration is also supported by assigning more arbitration shares to a particular master in the SOPC Builder. Avalon-MM interconnect employs *Dynamic Bus Sizing* [7] to resolve data width differences between masters and slaves.

In our system, for example, the CPU Memory is connected with Nios-II processor through a dedicated Avalon-MM link, which is always accessible for the processor regardless of other going-on transactions. So during the stereo matching processing, the CPU is still able to perform some tasks e.g., updating a descriptor memory, without interfering the stereo matching flow. In contrast, the DDR2-SDRAM Controller is accessed by many masters, sometimes concurrently; therefore the access has to be granted according to masters' priorities. In our current implementation, equal fairness is applied for all masters.

Avalon-MM interconnections in our system mainly take the following responsibilities:

- Work as data and instruction buses between the Nios-II CPU and memories connected to it.
- Allow Nios-II to access its peripherals and transfer control commands and status information between them.
- Transfer non-real-time data, e.g., data from the JTAG UART link to DDR2-SDRAM.

4.4.2 Avalon Streaming Interconnect

An Avalon-ST link creates a point-to-point connection between a streaming source and a streaming sink port to accommodate the requirements of high-bandwidth, low latency, uni-directional traffic associated with networking, video and DSP applications. Since Avalon-ST link carries continuous data streams, it does not require an address signal on the sink port. Also because its data flow is always point-to-point and uni-directional, there is no need to distinguish read or write requests. Minimally, an Avalon-ST interconnect only requires two signals: *data* and *valid*, as shown in Figure 4.4; the sink port only samples data from the source port when both *data* and *valid* signals are asserted.

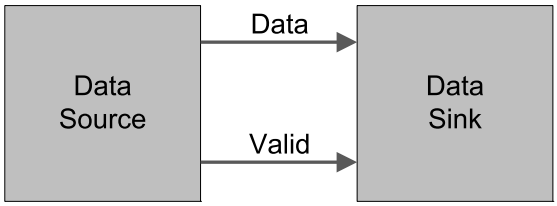

Figure 4.4: The minimum Avalon-ST interconnect

In our system, the *ready* signal is also implemented in Avalon-ST links to provide back pressure. The *ready* signal is asserted by a sink port when it is ready to accept data, otherwise *ready* is de-asserted to pause the data sending from its source port. Each Avalon-ST link in our system is comprised of *data*, *valid* and *ready* signals (see Figure 4.5). Because our stereo matching hardware is fully pipelined, it never stops processing internally as long as its source is valid and sink is ready.

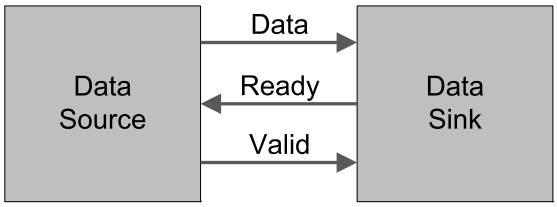

Figure 4.5: Avalon-ST interconnect in our system

Avalon-ST interconnects are responsible for transporting stereo frames , intermediate stereo matching results and final disparity map frames between processing cores and the DDR2-SDRAM. All Avalon-ST streams are transferred in progressive scanline order.

4.5 Storage System

Hierarchically, the storage system contains three levels: registers, on-chip SRAM and off-chip SDRAM. Registers guarantee 0-cycle access latency for all operations, so they

are responsible for holding the most time-critical data. In addition, since a group of registers are accessible concurrently, they are also used for data to be processed in parallel. The on-chip SRAMs are implemented as memory blocks (also called block RAMs) in our Stratix-III FPGA and used for the following purposes:

- Cache, program and data memory for the Nios-II processor
- Descriptor memories for the SG-DMAs
- Scanline data buffers in the stereo matching coprocessors
- FIFOs for peripherals if necessary e.g., in the Clocked Video Output

The external DDR2-SDRAM is a 1GB DDR2-667 dual-rank SO-DIMM module, with each rank organized as 64M × 64bit. The SDRAM core is clocked at same frequency with the CPU clock (100MHz), therefore the data bus between FPGA and DDR2-SDRAM is working at 200MHz. Because DDR2-SDRAM transfers data at both the rising and falling clock edge, the local data width provided by the DDR2-SDRAM controller is 256bits (2 × 2 × 64). However because there exists several masters in the system accessing the only SDRAM, and also because SDRAM has internal access latencies, the DDR2-SDRAM still works not as efficient as the on-chip block RAMs. Higher efficiency of using the SDRAM is achieved when long read bursts or long write bursts are issued to the same *SDRAM row*, which normally implies a continuous addressing space. To use the SDRAM more efficiently, we pack data words to be processed concurrently into one larger data word, and read/write them continuously as required by scanline processing order. For example, the 16-bit data word read by Scatter-Gather DMA-0 (see Figure 4.1) is comprised of 8-bit luminance pixel from the left video frame and 8-bit luminance pixel with the same coordinates from the right frame, illustrated in Figure 4.6.

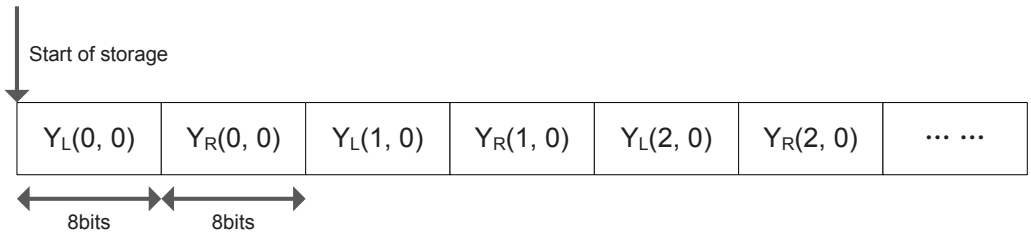

Figure 4.6: Packed stereo frame data storage

Utilization of the DDR2-SDRAM storage is listed below.

- Stereo frame buffer

  Minimally this storage only requires space for 2 stereo frame pairs to provide alternating accesses for the stereo cameras and stereo matching cores. The required memory space is:

$$Stereo\ frame\ buffer\ size = 2 \times frame\ width \times frame\ height \times (2 \times 8bits) \quad (4.1)$$

For a VGA resolution (640 × 480) video this amounts to 1.17M bytes. In case that the raw stereo frames are not required to be preserved, the stereo cameras directly feed packed stereo frame data to stereo matching blocks and this piece of storage becomes zero. In another case if a sequence of continuous frame pairs is required for evaluating the stereo matching performance, this storage also linearly scales accordingly:

$$Stereo\ frame\ buffer\ size = N \times frame\ width \times frame\ height \times (2 \times 8bits) \quad (4.2)$$

Where N indicates the desired number of frame pairs.

- Disparity map buffer

  This storage, if exist, is similar to the stereo video source part, except that here it only requires 1 frame of 8-bit disparities for each processed stereo frame pair. The required memory space is:

$$Disparity\ map\ buffer\ size = N \times frame\ width \times frame\ height \times 8bits \quad (4.3)$$

  Where N indicates the number of disparity map frames, which also equals to the number of processed stereo frame pairs .

4.6 Bandwidth Utilization Estimations

Bandwidth of a DDR2 SDRAM module is given by Equation 4.4:

$$Bandwidth = 2\ (double\ data\ rate) \times data\ bus\ width \times data\ bus\ frequency \quad (4.4)$$

In our implementation, the DDR2-SDRAM *data bus width* is 64-bit and the *data bus frequency* is 200MHz. Therefore the bandwidth of the DDR2-SDRAM is 2 × 64 × 200MHz = 25.6Gbits/s. To avoid ambiguity, in the context of the thesis *Bandwidth* refers to the maximum or theoretical bit rate of a communication channel, and *Throughput (or Bandwidth Utilization)* indicates the average bit rate of actual data being transferred. Throughput of the DDR2-SDRAM is calculated by Equation 4.5:

$$Throughput = memory\ bandwidth \times efficiency \quad (4.5)$$

The *efficiency* is mainly determined by the DDR2-SDRAM controller and the accessing pattern of applications. The worst-case efficiency happens when the application is frequently switching between short reads and short writes and memory addressing is forcing a *row* to be opened and closed on every transaction [6]. In contrast, the best-case scenario arises with long read bursts and long write bursts, which implies a continuous address space to be accessed. The following techniques are applied to improve the memory accessing efficiency.

- Data words to be processed in parallel are packed into a larger word (e.g., Figure 4.6) to form a continuous addressing space.

- All data words are stored and accessed in continuous scanline order to minimize possible memory row breaks.
- The DDR2-SDRAM controller is implemented using Altera DDR2-SDRAM High-Performance Controller Core and some parameters are configured to improve its efficiency e.g., the maximum burst length and address bus ordering (chip select, bank, row and column).

As shown in Figure 4.1, only the Nios-II processor and SG-DMAs have access to the DDR2-SDRAM controller. Nios-II only uses the external memory for non-real-time data, e.g., prepare the stereo frame buffer content at the beginning and upload disparity maps to the PC finally; so its influence on the real-time throughput is negligible. In contrast stereo matching cores have to access the external memory for real-time processing, and the required throughputs by different coprocessors are listed below:

- Pre-Processor

  SG-DMA-0 reads stereo frame pairs from the external memory. Its throughput requirement is given by

  $$Stereo\ throughput = frame\ rate \times frame\ width \times frame\ height \times 16bits \quad (4.6)$$

  In the future developments, if stereo video directly comes from cameras, this requirement becomes zero for external memory.

- Post-Processor

  SG-DMA-1 writes disparity maps to the external memory. Its throughput requirement is:

  $$Disparity\ throughput = frame\ rate \times frame\ width \times frame\ height \times 8bits \quad (4.7)$$

  If disparity maps are directly output to display devices e.g., Figure 4.2, this requirement also becomes zero for external memory.

- Stereo-Matcher

  The Stereo-Matcher does not require the external memory as data storage.

The overall bandwidth utilization is therefore estimated by Equation 4.8 and Equation 4.9 for the two SoC implementations respectively.

$$Estimated\ throughput_1 = frame\ rate \times frame\ width \times frame\ height \times 24bits \quad (4.8)$$

$$Estimated\ throughput_2 = frame\ rate \times frame\ width \times frame\ height \times 16bits \quad (4.9)$$

Design Evaluation and Experimental Results

5

Corresponding to the design considerations discussed in Section 2.4, in this chapter we evaluate our design in the following aspects.

- The proposed stereo matching algorithm
- FPGA implementation and scalability
- Real-time performance

For the proposed algorithm, we evaluate its accuracy with several hardware and software design parameters and its robustness. We also evaluate our FPGA implementation and the design scalability in this chapter. The real-time performance of our design is evaluated based on both ModelSim simulation and FPGA implementation. Finally the comparison between our implementation and related work is provided and discussed.

5.1 Evaluation of the Proposed Algorithm

As introduced in Section 3.2.2.3 and Section 3.3.2.2, there are two important hardware parameters in our design, i.e., the maximum support region arm length L_{max} and the vertical aggregation span V_{span}. The two parameters have impact on the logic resource and memory usage, and should be determined at design time. One software parameter that is programmable at run time is the support region threshold τ, which is configured by the Nios-II software. Figure 5.1 and Figure 5.2 show their influence on the stereo matching accuracy. In the two figures the support region threshold is set to 17. The curve marked with $V_{span} = A$ indicates vertical aggregation with fully adaptive support region. The figures show that the stereo matching algorithm produces high accuracy with L_{max} ranging from 15 to 20, and V_{span} from 5 to fully adaptive; the error rate only varies a little. Because larger L_{max} and V_{span} require more logic and memory resource, we set the two hardware parameters to 15 and 5 respectively.

With L_{max} and V_{span} fixed, the software parameter τ is still programmable at run time; its value is configured by the user with push-buttons on the FPGA board or control commands sent from a host PC. Its effect on the stereo matching accuracy is shown by Figure 5.3 and Figure 5.4. In the figures, we care most about the NonOcc error rates, which should be less than 10% in out target implementation. From the figures we notice that the matching accuracy varies with this software parameter and reaches the minimum error rates when $15 \leq \tau \leq 20$. By default we set τ to 17. The benchmarked

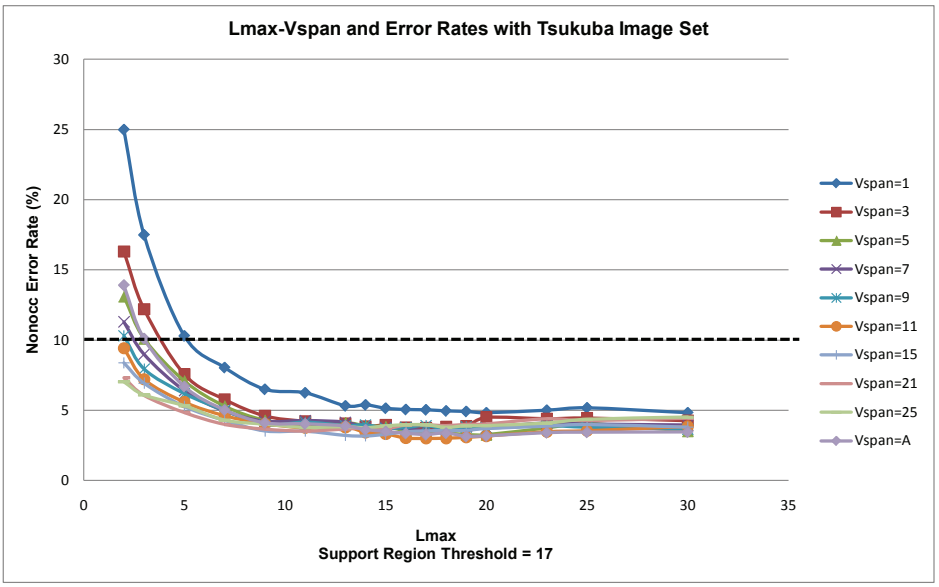

Figure 5.1: Lmax-Vspan and error rates with Tsukuba image set

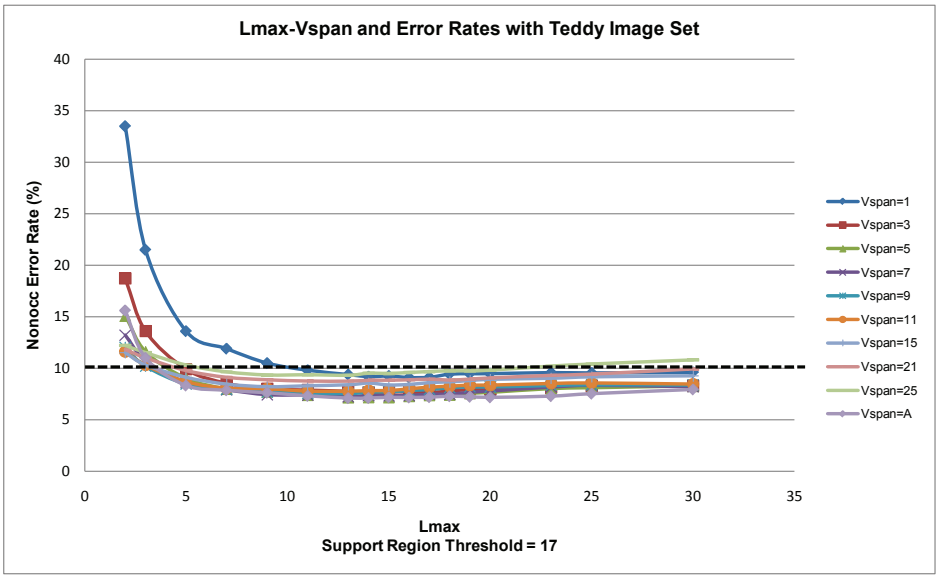

Figure 5.2: Lmax-Vspan and error rates with Teddy image set

results given in Section 5.4 are computed with the parameter configuration given by Equation 5.1.

$$L_{max} = 15, V_{span} = 5, \tau = 17. \tag{5.1}$$

The left benchmark images, truth disparity maps and our resulted disparity maps are shown together in Figure 5.5.

Our proposed algorithm is also very robust to luminance bias and radiometric differences. Figure 5.6(a - b) shows the stereo image set with a luminance bias of +50 on the

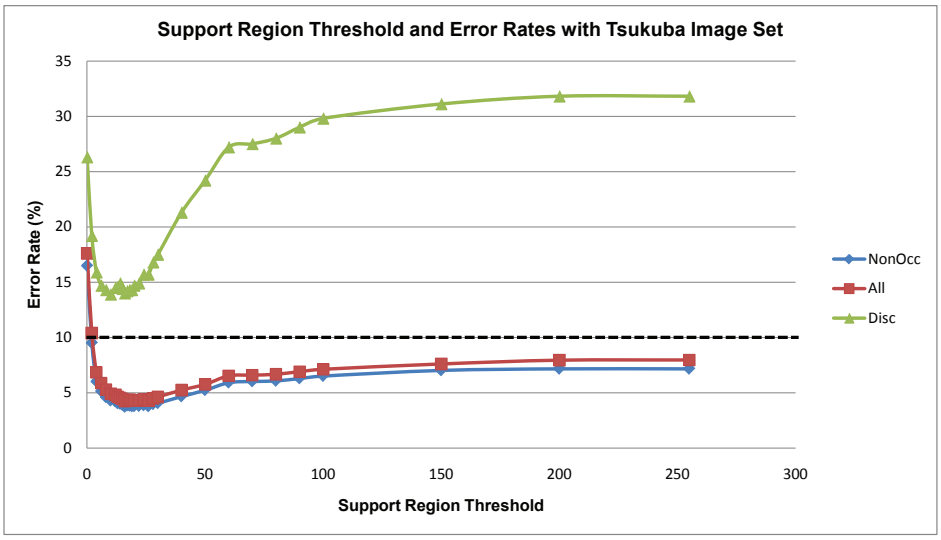

Figure 5.3: Support region threshold and error rates with Tsukuba image set

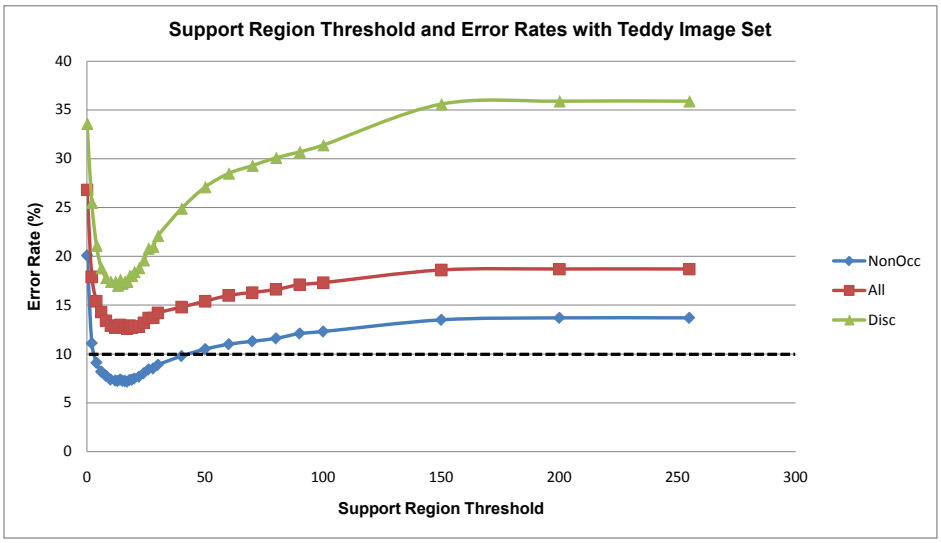

Figure 5.4: Support region threshold and error rates with Teddy image set

right camera and the resulted disparity map (Figure 5.6(c)) computed by the Variable-Cross algorithm [49]. Obviously the stereo matching result severely degrades caused by the bias. In contrast, Figure 5.6(d) is the disparity map computed by our proposed algorithm, which is only a bit affected by the luminance bias. The robustness is obtained by the adopted mini-census transform [4] and corresponding Hamming distance raw cost function.

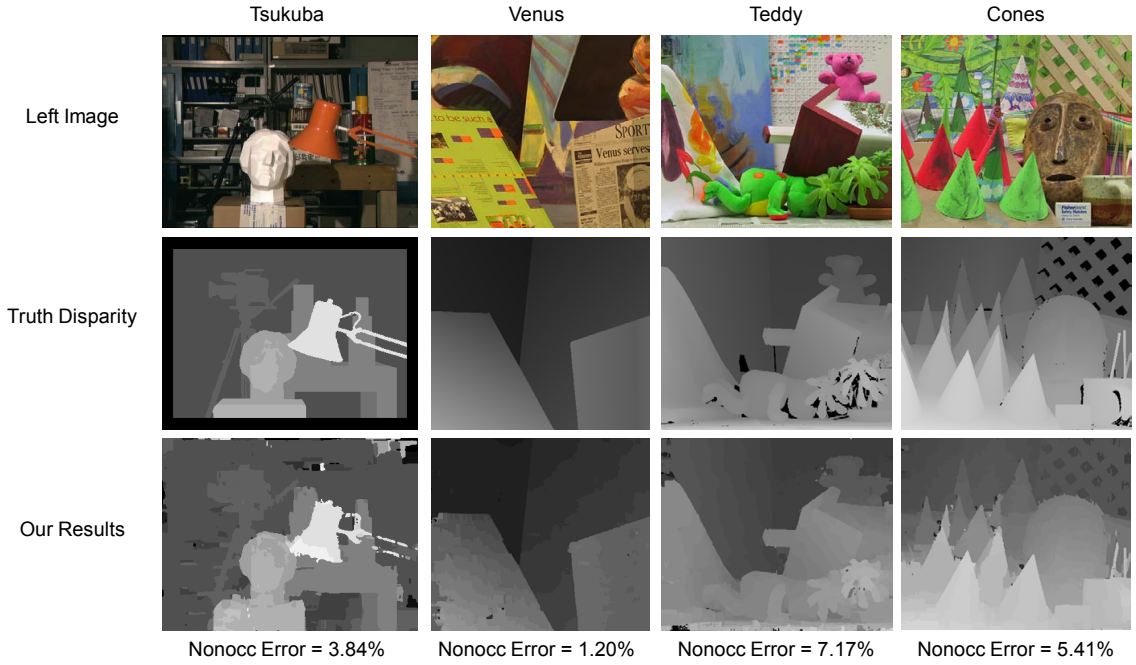

Figure 5.5: Truth disparity maps and our results

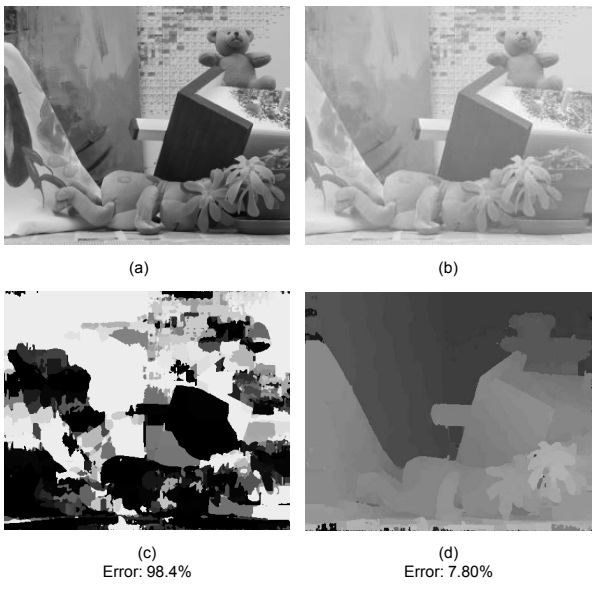

Figure 5.6: Luminance biased images and the resulted disparity maps
(a): Image taken by the left camera. (b): Image taken by the right camera with +50 luminance bias. (c): Resulted disparity map by the VariableCross algorithm. (d): Resulted disparity map by the proposed algorithm

5.2 Evaluation of the FPGA Implementation

5.2.1 Introduction to the FPGA Hardware Resource

FPGA EP3SL150 is chosen as our implementation platform; this FPGA belongs to Altera 65-nm *Stratix III L* family that provides balanced logic, memory and multiplier ratios for mainstream applications [11]. Unlike custom ASICs, the hardware resources of an FPGA chip are determined at the time it leaves factory. Therefore, knowing the features and amount of its programmable hardware is essential to any design work. Its hardware resources that are essential to our design are introduced below, and their availability is summarized in Table 5.3.

- Programmable logic array blocks (LABs)

  The basic programmable logic array block if EP3SL150 is known as *adaptive logic module (ALM)* that supports logic functions, arithmetic functions and registers. The high-level block diagram of an ALM is shown in Figure 5.7. Each ALM

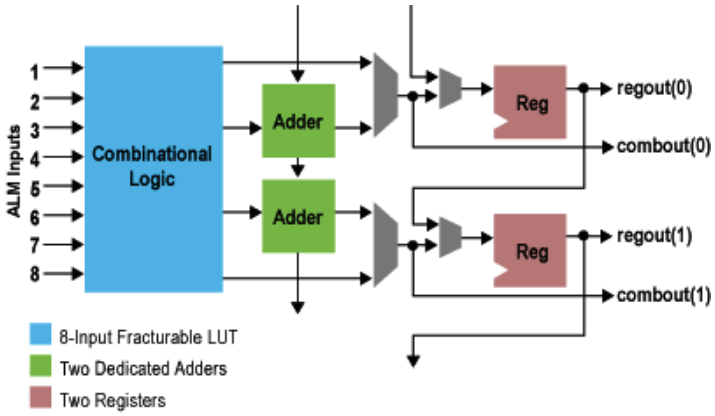

Figure 5.7: High-level block diagram of Stratix-III ALM

  is equipped with a fracturable 8-input look-up table (LUT), two dedicated 1-bit adders, two dedicated 1-bit registers and additional logic enhancements. One ALM is equivalent to 2.5 traditional 4-input LUT based *Logic Elements (LEs)*. Utilization of ALMs is performed by Quartus-II software at compile time, and in most cases users do not have to manually adjust them.

- Memory logic array blocks (MLABs)

  MLAB, which adds SRAM memory to the above mentioned LAB, is a superset of the LAB and includes all LAB features. Each ALM in an MLAB is also able to work as a 320-bit SRAM block, and its possible aspect ratio configurations are listed in Table 5.1 [8]. MLABs are distributed over the whole FPGA chip and very suitable for implementing small delay lines and shift registers.

- Dedicated memory blocks (BRAMs)

EP3SL150 contains two types of dedicated memory blocks: M9K and M144K. As their names indicate, one M9K block contains 9K SRAM bits and one M144K contains 144K bits. Each memory block supports single port, simple dual port[1] or true dual port[2] mode and multiple aspect ratio configurations (see Table 5.1); different configurations also affect their achievable performances. BRAMs are suitable for general purpose memory applications, and in our system they are employed for processor memory, descriptor memory, FIFO, frame line buffer etc.

Feature	MLABs	M9K Blocks	M144K Blocks
Maximum Performance	600 MHz	580 MHz	580 MHz
Aspect Ratios	16 × 8 16 × 9 16 × 10 16 × 16 16 × 18 16 × 20	8 K × 1 4 K × 2 2 K × 4 1 K × 8 1 K × 9 512 × 16 512 × 18 256 × 32 256 × 36	16 K × 8 16 K × 9 8 K × 16 8 K × 18 4 K × 32 4 K × 36 2 K × 64 2 K × 72

Table 5.1: EP3SL150 on-chip memory features

- Digital signal processing blocks (DSPs)

  EP3SL150 has dedicated high-performance digital signal processing (DSP) blocks optimized for DSP applications. The fundamental building block is a *Two-Multiplier Adder* comprised of a pair of 18-bit × 18-bit multipliers followed by a 37-bit addition/subtraction unit, as shown in Figure 5.8. Each Stratix III DSP block contains four *Two-Multiplier Adder* units and other logic enhancements for different computing configurations. Stratix III DSP blocks support various computa-

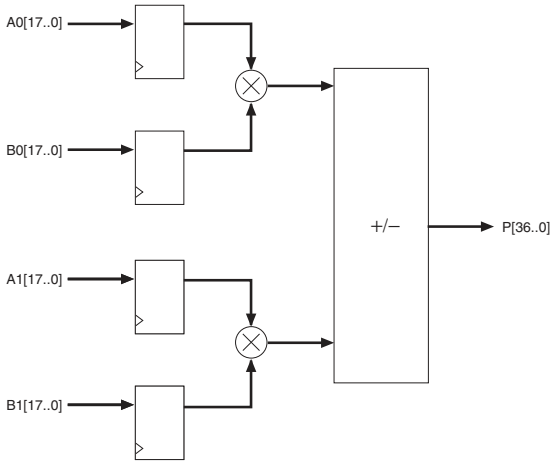

Figure 5.8: Basic Two-Multiplier Adder DSP unit

tion modes including multiplication, multiply-add/subtract, multiply-accumulate

[1]One read port and one write port
[2]Two read/write ports

and so on. The available resources under different configurations are summarized in Table 5.2. Among these configurations, we have widely implemented the

Configuration	DSP Blocks	Independent Input and Output Multiplication Operators				Two-Multiplier Adders
		9 × 9 Multipliers	12 × 12 Multipliers	18 × 18 Multipliers	36 × 36 Multipliers	18 × 18 ± 18 × 18
Availability	48	384	288	192	96	192

Table 5.2: EP3SL150 DSP block configurations

Two-Multiplier Adder mode to compute Equation 5.2, which is applied in the Winner-Takes-All step.

$$Result[36..0] = A[17..0] \times B[17..0] - C[17..0] \times D[17..0] \tag{5.2}$$

The overall available resources that are important to this design are summarized in Table 5.3. To make our design fit the FPGA and scalable for higher-density devices,

Resource	Logic Array Blocks		Memory Array Blocks		Dedicated SRAMs			Dedicated DSPs	
	ALMs	LEs	MLAB Blocks	MLAB Kbits	M9K Blocks	M144K Blocks	Dedicated RAM Kbits	DSP Blocks	Two-Multiplier Adders
Availability	57K	142.5K	2850	891	355	16	5499	48	192

Table 5.3: EP3SL150 hardware resource summary

one important principle is to properly map our algorithm to different functional units and balance their consumption ratios. For example, general shift registers are achievable using either normal ALMs or MALBs, but the synthesis tool Quartus-II is not intelligent enough to make the optimal implementation all the time. In this case we have to clearly specify which kind of resource to be used.

More available resources of the selected FPGA, e.g., phase-locked loops (PLLs), error correction coding (ECC) and high-speed I/O support, are not introduced here. They are either not used or automatically handled by Quartus-II.

5.2.2 FPGA Implementation Report

In the stereo matching pipeline implementation, the Pre-Processor is not aware of any disparity range and the Post-Processor is only slightly affected by the maximum allowed disparity range. In contrast, utilized hardware resource by the Stereo-Matcher is mainly determined by the maximum allowed disparity range. As shown in Figure 3.14, the number of processing modules in the Stereo-Matcher scales with the maximum allowed disparity range. In practice, the disparity range is determined by the distance between the scene objects and the stereo camera baseline, and the length of the baseline itself. So it varies with the target application and corresponding camera setup. In our implementation, the Post-Processor is set to deal with a maximum disparity range of 64, and the Stereo-Matcher is tested with maximum disparity range of 16, 32 and 64 respectively.

Besides disparity range, the image size also determines the hardware resource utilization, especially for the data line buffers. As introduced in Section 5.2.1, the data

line buffers are implemented with the M9K memory blocks, which are configured with different data widths according to the computing requirements. With the EP3SL150 FPGA we target for videos with frame width no larger than 1024, so the depths of the line buffer memories are also set to 1024 (1K). The line buffer depth only limits the frame width; the frame height is not limited and it only affects the processing time.

	Combinational ALUTs		Memory ALUTs		Registers		DSP Blocks		SRAM Bits	
	Total: 113,600	Util.	Total:	Util.	Total:	Util.	Total: 384	Util.	Total: 5,630,976	Util.
Pre-Processor	3,310	3%	288	1%	2,075	2%	0	0%	417,792	7%
Post-Processor	12,211	11%	536	1%	10,263	9%	0	0%	393,216	7%
Stereo-Matcher 16	8,874	8%	4,864	9%	18,813	17%	60	16%	589,824	10%
Stereo-Matcher 32	17,136	15%	9,728	17%	37,245	33%	124	32%	884,736	16%
Stereo-Matcher 64	33,639	30%	19,456	34%	74,109	65%	252	66%	1,474,560	26%
Stereo-Matcher 64 SoC1	60,296	53%	20,288	36%	94,891	84%	257	67%	3,771,247	67%
Stereo-Matcher 64 SoC2	60,816	54%	20,288	36%	94,980	84%	257	67%	3,752,121	67%

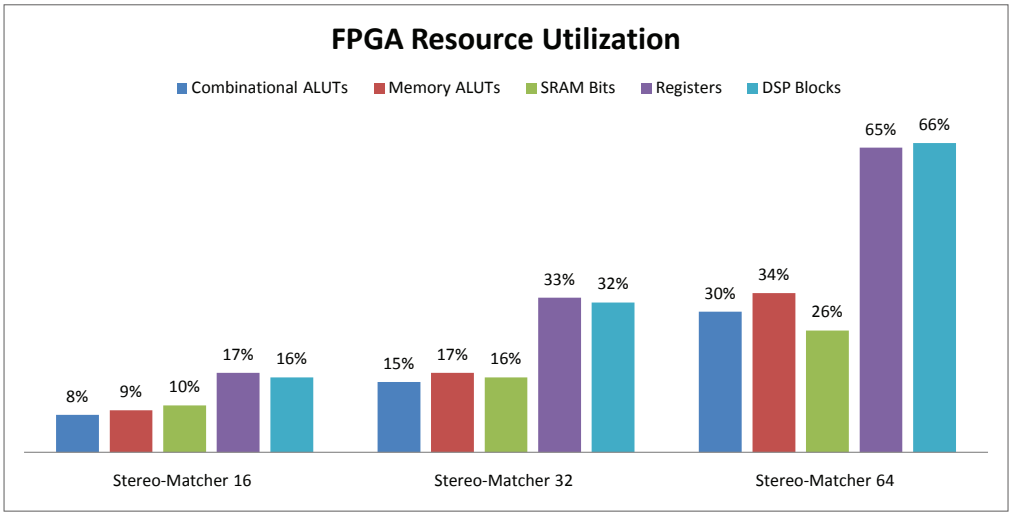

Table 5.4: EP3SL150 hardware resource utilization summary

The FPGA hardware resources utilized by different processing modules are summarized in Figure 5.4. The values following Stereo-Matcher indicate the number of parallel computing threads implemented in the Stereo-Matcher processor, which equals the maximum allowed disparity of corresponding implementation. It clearly shows that the hardware utilization scales with the number of parallel computing threads. The design scalability is illustrated in Figure 5.5. The scalability figure shows that the implementation scales nearly linearly with the parallel computing threads, but there are different scaling factors associated with the corresponding hardware resource. In the current implementation the required registers (flip-flops) and dedicated DSP blocks are limiting resources for a larger scale implementation. Balancing techniques include replacing some dedicated hardware DSPs with LUTs, and reducing the usage of pipeline registers.

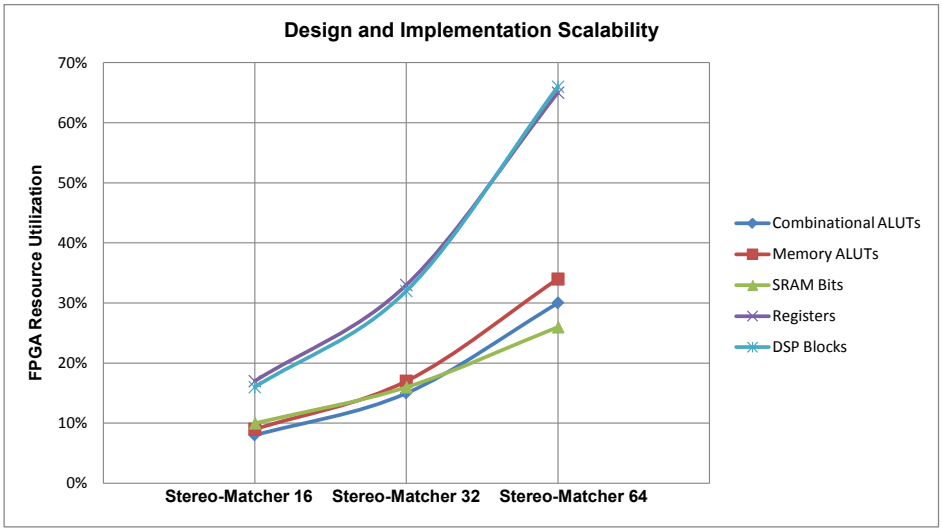

Table 5.5: Design and implementation scalability

5.3 Evaluation of the Real-Time Performance

We evaluate the real-time performance with two approaches: ModelSim simulation and FPGA implementation. The simulation verifies the functional correctness of our design model and the implementation demonstrates its applicability and actual performance in practice.

5.3.1 Performance Evaluation with Simulation

The ModelSim simulation provides the theoretical processing time and latencies. We have already discussed the hardware processing latencies in Chapter 3, and they are summarized in Table 5.6. The full pipeline requires a latency about 34 lines of valid

	Processing Module	Pipeline Latency	Frame Resolution				
			384 × 288	450 × 375	640 × 480	800 × 600	1024 × 768
Pre-Processor	Median Filter	Frame Width + 6	390	456	646	806	1030
	Census Transform and Support Region Builder	Frame Width × 15 + 19	5779	6769	9619	12019	15379
	Output Logic	0	0	0	0	0	0
Stereo-Matcher	Raw Cost Scatter and Correlation Region Builder	4	4	4	4	4	4
	Horizontal Aggregation	17	17	17	17	17	17
	Vertical Aggregation	Frame Width × 2 + 3	771	903	1283	1603	2051
	Lineup and WTA	72	72	72	72	72	72
	Output Logic	2	2	2	2	2	2
Post-Processor	L-R Consistency Check	69	69	69	69	69	69
	Horizontal Voting	23	23	23	23	23	23
	Vertical Voting	Frame Width × 15 + 9	5769	6759	9609	12009	15369
	Median Filter	Frame Width + 6	390	456	646	806	1030
Total		Frame Width × 34 + 230	13286	15530	21990	27430	35046

Table 5.6: Stereo matching pipeline latency summary
The latency is measured in cycles and targets for 100MHz with EP3SL150 FPGA

pixel cycles. For video processing, after the initial pipeline latency, valid disparity

pixels continuously come out of the pipeline in progressive scanline order. Thus the throughput of the full pipeline is determined by the frame resolution and the vertical blanking lines required by the aforementioned vertical aggregation step. The pipeline cycles for processing a full video frame is summarized in Table 5.7.

	Pipeline Cycle Count	Frame Resolution				
		384 × 288	450 × 375	640 × 480	800 × 600	1024 × 768
Valid Area	Frame Width × Frame Height	110592	168750	307200	480000	786432
Vertical Blanking	Frame Width × 2	768	900	1280	1600	2048
Total	Frame Width × (Frame Height + 2)	111360	169650	308480	481600	788480

Table 5.7: Stereo matching processing speed summary
The speed is measured in cycles and targets for 100MHz with EP3SL150 FPGA

Our ModelSim simulation exactly verifies the latency and processing cycles. Therefore the theoretical processing frame rate is calculated by dividing the pixel clock with the total pipeline processing cycles, as shown in Table 5.8.

Pixel Clock	Theoretical Frame Rates (FPS)				
	384 × 288	450 × 375	640 × 480	800 × 600	1024 × 768
	111360	169650	308480	481600	788480
25M	224	147	81	51	31
50M	448	294	162	103	63
75M	673	442	243	155	95
100M	897	589	324	207	126
150M	1346	884	486	311	190

Table 5.8: Theoretical frame rates (FPS) with different pixel clocks

5.3.2 Performance Evaluation with FPGA

We evaluate the real-time performance of our FPGA implementation with the two proposed SoC architecture presented in Chapter 4. The first SoC (see Figure 4.1 SoC1) accesses the external DDR2-SDRAM as both source and result frame buffer and the whole stereo matching pipeline and corresponding DMAs work at 100MHz. To measure the processing time with this implementation, we first download a pair of rectified stereo frames to the DDR2-SDRAM through the JTAG UART link; then the Nios-II CPU triggers the stereo matching processing, which repeatedly computes the same frame pair for a certain number of times. The software control is illustrated in Figure 4.3. The computing start and end times are recorded by the on-chip timer and therefore the corresponding frame rates are obtained. The measured frame rates with SoC1 and 100MHz operating clock are summarized in Table 5.9. As discussed in Section 4.6, in our implementation the DDR2-SDRAM bandwidth is 25.6Gbits/s; corresponding throughput and external memory bandwidth utilization are computed by Equation 4.8 and Equation 4.5.

FPGA Frame Rates (FPS) with SoC1 and 100MHz Clock					
Frame Sizes	**384 × 288**	**450 × 375**	**640 × 480**	**800 × 600**	**1024 × 768**
	111360	169650	308480	481600	788480
Frame Rates	296	193	116	80	47
Throughput (Gbits/s)	0.791	0.786	0.859	0.925	0.889
Bandwidth Utilization	3.09%	3.07%	3.35%	3.61%	3.47%

Table 5.9: FPGA frame rates (FPS) with SoC1 and 100MHz Clock

Comparing the data shown in Table 5.8 and Table 5.9, we notice that the measured frame rates are about one third of the theoretical maximum attainable performance. This is because the external memory accessing efficiency is not optimized, which is determined by a number of factors, such as randomness of addresses, refresh rate, turnaround times between reads and writes, and burst lengths. During the stereo matching processing, the SG-DMA-0 in SoC1 (see Figure 4.1) continuously reads stereo pixels from the DDR2-SDRAM and the SG-DMA-1 continuously writes pixel disparities back to it. Such back-to-back reads and writes are considered to be the most inefficient way of using the DDR2-SDRAM. There are at least two approaches to solve this problem, one is to increase the allowed burst transfer size of the DMAs, and the other is to add FIFOs before and after the processing pipeline to avoid the frequent switches between reading and writing the DDR2-SDRAM. Since the proposed stereo matching pipeline requires continuous pixel reads and writes, FIFOs can be used to pre-fetch data and buffer results for long burst transfers. We have put the first approach into practice and it contributes to the performance in Table 5.9. Other possible optimizations will also be implemented in our future developments.

With SoC2 (see Figure 4.2) and a desktop DVI monitor, we have evaluated our design and implementation with several standard display specifications. In this case, the stereo matching pipeline and the SG-DMA-0 still work at 100MHz, but the output disparity map is synchronized with various pixel clocks. This implementation does not need a result frame buffer, therefore possible external memory accessing competitions are relieved. This implementation has the potential to provide higher frame rates than the SoC1. Our reported performance 1024×768 @ 60FPS is based on this implementation. The implementation passes tests with all frame sizes and frame rates listed in Table 5.10; corresponding throughput and external memory bandwidth utilization are computed by Equation 4.9 and Equation 4.5.

FPGA Frame Rates (FPS) with SoC2 and Standard Pixel Clocks					
Frame Sizes	**640 × 480**	**640 × 480**	**800 × 600**	**800 × 600**	**1024 × 768**
	308480	308480	481600	481600	788480
Pixel Clock (MHz)	25.17	31.5	40	49.5	65
Frame Rates	60	75	60	75	60
Throughput (Gbits/s)	0.296	0.370	0.462	0.578	0.757
Bandwidth Utilization	1.15%	1.45%	1.81%	2.26%	2.95%

Table 5.10: FPGA frame rates (FPS) with SoC2 and standard pixel clocks

The performance evaluations show that the proposed implementation is able to deliver real-time high-definition stereo matching with very low external memory bandwidth

utilization. It is also flexibility adaptive to various standard pixel clocks. On one hand, the external memory accessing efficiency can be improved to increase the achievable frame rates; on the other hand, the reserved bandwidth can also be used for other applications such as image rectification and viewpoint interpolation. To the best of our knowledge, we have achieved the fastest processing speed (60 frames per second) for high-definition (1024 × 768) stereo matching.

5.4 Comparison to Related Work

A well known stereo matching algorithm benchmarking metric is presented on the Middlebury Stereo Evaluation website. A disparity map produced by a stereo matching algorithm is compared with the ground truth disparity map; if the disparity values are not identical in the two maps, it is regarded as a bad matched pixel. Besides comparing *all* pixels in the image, two more additional metrics are also given in the evaluation results i.e., *nonocc* and *disc*, which denotes *non-occluded regions* and *near occluded regions* respectively. The *near occluded regions* are formerly referred to as *near discontinuities*, and qualify as the most difficult areas for stereo matching algorithms to solve.

The benchmarked performance of our implemented algorithm is shown and compared with other work in Table 5.11. The algorithm is based on VariableCross [49] shown in the table. Although the benchmark provides a fair comparison among error rates

Stereo Matching Error Rates (Bad Matches%)													
Image Set	*Tsukuba*			*Venus*			*Teddy*			*Cones*			*Average Bad Pixel Rate*
Image Size	*384 x 288*			*434 x 383*			*450 x 375*			*450 x 375*			
Disparity Range	*16*			*20*			*60*			*60*			
Evaluation Method	*nonocc*	*all*	*disc*	*nonocc*	*all*	*disc*	*nonocc*	*all*	*disc*	*nonocc*	*all*	*disc*	
VariableCross [49]	1.99	2.65	6.77	0.62	0.96	3.20	9.75	15.1	18.2	6.28	12.7	12.9	7.60
RealtimeBFV [50]	1.71	2.22	6.74	0.55	0.87	2.88	9.90	15.0	19.5	6.66	12.3	13.4	7.65
RealtimeBP [46]	1.49	3.40	7.87	0.77	1.90	9.00	8.72	13.2	17.2	4.61	11.6	12.4	7.69
Chang et al. 2010 [4]	N/A	2.80	N/A	N/A	0.64	N/A	N/A	13.7	N/A	N/A	10.1	N/A	N/A
Proposed	**3.84**	**4.34**	**14.2**	**1.20**	**1.68**	**5.62**	**7.17**	**12.6**	**17.4**	**5.41**	**11.0**	**13.9**	**8.20**
FastAggreg [39]	1.16	2.11	6.06	4.03	4.75	6.43	9.04	15.2	20.2	5.37	12.6	11.9	8.24
OptimizedDP [34]	1.97	3.78	9.80	3.33	4.74	13.0	6.53	13.9	16.6	5.17	13.7	13.4	8.83
RealtimeVar [26]	3.33	5.48	16.8	1.15	2.35	12.8	6.18	13.1	17.3	4.66	11.7	13.7	9.05
RTCensus [20]	5.08	6.25	19.2	1.58	2.42	14.2	7.96	13.8	20.3	4.10	9.54	12.2	9.73
RealTimeGPU [43]	2.05	4.22	10.6	1.92	2.98	20.3	7.23	14.4	17.6	6.41	13.7	16.5	9.82
Jin et al. 2010 [23]	9.79	11.56	20.29	3.59	5.27	36.82	12.5	21.5	30.57	7.34	17.58	21.01	17.24
Chang et al. 2007 [3]	21.5	21.7	48.7	16.5	17.8	29.9	26.3	33.6	35.1	24.2	32.4	31.0	N/A

Table 5.11: Stereo matching algorithm error rate benchmark
The algorithms and corresponding implementations are ordered according to the averaged bad pixel rate

of different algorithms, it still has some insufficiencies, e.g., it does not evaluate the algorithm's robustness to luminance bias and radiometric differences; and more importantly, the processing speed is not evaluated in this benchmark. To improve the robustness of VariableCross algorithm and enable efficient hardware implementations, we make some trade-offs so the result is reasonably a bit worse than VariableCross. Nevertheless, the modified algorithm still demonstrates even better performance than VariableCross on the *Teddy* and *Cones* image sets, which feature higher resolution and

disparity range. On the other hand, VariableCross is implemented on CPU and achieves very limited frame rates; while the proposed algorithm with FPGA implementation has significant improvement regarding processing speed.

Table 5.12 shows the processing speed comparisons between our implementation and other reported real-time implementations. The processing speed of different systems is

Stereo Matching Frame Rates (Frames per second)				
	Implementation	**Disparity Range**	**Frame Rate**	**MDE/s**
Jin et al. 2010 [23]	1 x FPGA Virtex-4 XC4VLX200-10	64	230 @ 640 x 480	4521
Proposed	**1 x FPGA EP3SL150**	**64**	**60 @ 1024 x 768**	**3019**
RTCensus [20]	GPU GeForce GTX 280	60	105.4 @ 450 x 375	1067
Chang et al. 2010 [4]	ASIC UMC 90nm	64	42 @ 352 x 288	272
RealtimeBFV [50]	GPU GeForce GTX 8800	64	12 @ 450 x 375	129
Chang et al. 2007 [3]	DSP TMS320C64x	60	9.1 @ 450 x 375	92
RealTimeGPU [43]	GPU Radeon XL1800	16	43 @ 320 x 240	53
RealtimeVar [26]	CPU Pentium 2.83GHz	60	3.5 @ 450 x 375	35
RealtimeBP [46]	GPU GeForce GTX 7900	16	16 @ 320 x 240	20
FastAggreg [39]	CPU Core Duo 2.14GHz	60	1.67 @ 450 x 375	17
OptimizedDP [34]	PC 1.8GHz	60	1.25 @ 450 x 375	13
VariableCross [49]	CPU Pentium IV 3.0GHz	60	1.21 @ 450 x 375	13

Table 5.12: Stereo matching algorithm frame rates
The algorithms and corresponding implementations are ordered according to the MDE/s

given in frames per second (FPS) and, more meaningfully, in million disparity evaluations per second (MDE/s).

$$MDE/s = Image\ Width \times Image\ Height \times Disparity\ Range \times FPS \tag{5.3}$$

Clearly, our implementation demonstrates very high frame rates compared to other implementations. Our implementation and the one proposed by Jin et al. [23] are both fully pipelined design and the frame rates are not limited by the computing pipeline itself. The frame rate difference between the two implementations is caused by different measurement methods. In addition, our implementation achieves real-time high-definition performance and higher matching accuracy compared with Jin's work. For FPGA implementations, the achievable resolution is usually limited by the available on-chip memories, used as line buffers etc. Thanks to the mini-census transform, the required on-chip matching cost storage is reduced a lot. Moreover, we have also applied data-reuse technique in the cost aggregation step to further reduce the memory consumption. On the other hand, CPU and GPU based implementations do not have resolution limitations, however, they hardly achieve real-time performance with high-definition images. To the best of our knowledge, our implementation has achieved so far the best processing speed on high-definition images.

6 Conclusions and Future Work

This chapter summarizes the contributions of this thesis and concludes the thesis by suggesting valuable optimizations and future research and development directions.

6.1 Conclusions

This thesis has proposed an improved stereo matching algorithm suitable for hardware implementation based on the VariableCross and the MiniCensus algorithm. Furthermore, we provide parallel computing hardware design and implementation of the proposed algorithm. The experimental results have proved that our work has achieved high speed real-time processing with programmable video resolutions, while preserving high stereo matching accuracy. The online benchmarks also suggest that this work has achieved leading matching accuracy among declared real-time implementations. To the best of our knowledge, we have achieved the fastest processing speed (60 frames per second) for high-definition (1024×768) stereo matching.

The implementations have satisfied our design targets discussed in Section 2.4. Regarding the algorithm, the experimental results suggest that the combination of Mini-Census transform and the VariableCross stereo matching algorithm not only delivers very high matching accuracy, but also remains robust to radiometric and luminance differences. The hardware design proves that the cost aggregations with different disparity hypothesis are suitable for parallelization and pipelined systolic array processing. We have also proposed various modifications that are more hardware efficient than the VariableCross algorithm, without degrading the matching accuracy. These modifications simplify the hardware implementation so that more parallel computing threads and high-definition processing are supported on a chosen FPGA chip.

6.2 Summary of Chapters and Contributions

The proposed stereo matching algorithm and real-time implementation are motivated by a viewpoint interpolation application, which provides computer synthesized viewpoint for eye-gazing video conferences. The concept and proposed setup for the eye-gazing viewpoint interpolation are presented in Chapter 1. The focus and major contribution of this thesis work is that we have solved the major computing bottleneck, stereo matching, in the viewpoint interpolation processing pipeline.

Chapter 2 has introduced state-of-the-art stereo matching algorithms and various computing platforms being used for efficient implementations, which include high performance CPU, GPU, DSP, FPGA and ASIC. Nevertheless, these related work either

suffers from low stereo matching accuracy or only partially meets real-time requirements, especially for high definition image/video processing. To achieve both high speed and quality stereo matching, we have presented our design considerations and proposed solutions for a more advanced implementation. Finally the Altera EP3SL150 FPGA and the Terasic DE3 development board are selected as our implementation platform and we have successfully achieved the desired matching accuracy and real-time performance. Comparisons with previously related work are presented in Section 5.4.

Chapter 3 has elaborated the algorithm-hardware co-design development scheme being used in this thesis work. We first propose the data-level (also known as loop-level) parallel computing architecture for concurrently evaluating hypothetical disparities in the disparity range. Then we present the proposed algorithm and corresponding hardware pipeline design in a top-down approach. The full stereo matching pipeline is divided into three major processing modules, i.e., pre-processing, stereo matching and post-processing. In hardware their functions are performed by three co-processors i.e., Pre-Processor, Stereo-Matcher and Post-Processor, respectively. Massive data-level parallelizations are mainly implemented in the Stereo-Matcher processor, corresponding to the maximum allowed disparity range. In the other two co-processors data and bit level parallelism are also applied, for example the left and right images are processed in parallel whenever possible. Besides parallel computing, all co-processors are also fully pipelined in each processing step, which is the key to enable the highest throughput.

In Chapter 4 we have provided two reference SoC designs utilizing our stereo matching co-processors, soft processor (Nios-II) and various on-chip peripherals including DMAs, DDR2-SDRAM controller and block memories. The SoC1 design accesses the external DDR2-SDRAM as both source and result frame buffer, which is a practical use case with other higher level applications. The resulted disparity maps in DDR2-SDRAM provides by SoC1 are also uploaded to the PC to verify the hardware processing results. In contrast, the SoC2 design avoids the result frame buffer and directly outputs resulted disparity maps on a standard desktop DVI monitor. Because of accessing competitions, refresh time and turnaround between read and write problems, the external memory bandwidth is hardly used very efficiently. So we try to use the SoC2 implementation to achieve better real-time performance than SoC1. The SoC2 design and implementation also prove that our stereo matching IPs is perfectly compatible with the Avalon system interconnect standards and work properly with other Avalon-compatible components such as the IPs provided by Altrea's Video and Image Processing (VIP) Suite. In this chapter we also estimated the memory and bandwidth utilizations with several derived formula, which provide reference for future optimizations and application developments.

Chapter 5 evaluates the overall design and implementation in various aspects corresponding to the design considerations proposed in Chapter 2. The evaluation results show that this work has achieved our desired matching accuracy, robustness, real-time performance and design scalability. It also shows several potential problems with the current implementation, which will be solved or optimized in our future research and developments.

6.3 Future Work and Application Developments

Regarding the current FPGA implementation, there are at least two optimization plans to further improve its real-time performance.

1. Add FIFOs between DMAs and DDR2-SDRAM controller to avoid frequent switches between reads and writes.

   As shown in Section 5.3.2, the DDR2-SDRAM bandwidth utilization is very low. This accessing efficiency can be improved to increase the achievable frame rates; alternatively, the reserved bandwidth can also be used for other applications such as image rectification and viewpoint interpolation.

2. Balance the hardware resource utilizations

   Although the proposed architecture is scalable, the FPGA implementation reports in Section 5.2 expose that the consumed FPGA resources have different scaling factors according the implemented parallel computing threads. Since the chosen FPGA has fixed hardware resources, balanced resource utilizations are achievable by replacing some dedicated hardware DSPs with LUTs, and reducing the usage of pipeline registers. FPGA vendors also provide products with various resource allocations, targeting different application types. So it is also possible to choose a more suitable FPGA chip for this implementation or use ASIC design to circumvent the hardware resource limitations.

To enable a more complete function set in a single chip, the image acquisition and camera rectification modules are also suggested to be implemented together with the stereo matching pipeline. If camera and rectification modules are also Avalon-compatible, we believe the source frame buffer (in DDR2-SDRAM) can also be removed and the processing pipeline including image acquisition, rectification and stereo matching does not require any external memory access.

With real-time stereo matching processing, various applications are feasible with the stereo matching results, the disparity maps. The eye-gazing viewpoint interpolation is an potential application, which also motivates this thesis research. Other possible applications include free-view TV, object tracking and gesture controlled devices and video games.

Bibliography

[1] K. Ambrosch, M. Humenberger, W. Kubinger, and A. Steininger. SAD-based stereo matching using FPGAs. *Embedded Computer Vision*, pages 121–138, 2009.

[2] M.Z. Brown, D. Burschka, and G.D. Hager. Advances in computational stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 993–1008, 2003.

[3] N. Chang, T.M. Lin, T.H. Tsai, Y.C. Tseng, and T.S. Chang. Real-time DSP implementation on local stereo matching.

[4] N.Y.C. Chang, T.H. Tsai, B.H. Hsu, Y.C. Chen, and T.S. Chang. Algorithm and Architecture of Disparity Estimation With Mini-Census Adaptive Support Weight. *Circuits and Systems for Video Technology, IEEE Transactions on*, 20(6):792–805, 2010.

[5] P.P. Chu. *RTL hardware design using VHDL: coding for efficiency, portability, and scalability*. Wiley-IEEE Press, 2006.

[6] Altera Corporation. *The Efficiency of the DDR & DDR2 SDRAM Controller Compiler*, December 2004. http://www.altera.com/literature/wp/wp_ddr_sdram_efficiency.pdf.

[7] Altera Corporation. *Avalon Interface Specifications*, April 2009. http://www.altera.com/literature/manual/mnl_avalon_spec.pdf.

[8] Altera Corporation. *TriMatrix Embedded Memory Blocks in Stratix III Devices*, May 2009. http://www.altera.com/literature/hb/stx3/stx3_siii51004.pdf.

[9] Altera Corporation. *Embedded Peripherals IP User Guide*, July 2010. http://www.altera.com/literature/ug/ug_embedded_ip.pdf.

[10] Altera Corporation. *Nios II Processor Reference Handbook*, July 2010. http://www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf.

[11] Altera Corporation. *Stratix III Device Handbook*, July 2010. http://www.altera.com/literature/hb/stx3/stratix3_handbook.pdf.

[12] Altera Corporation. *Video and Image Processing Suite User Guide*, July 2010. http://www.altera.com/literature/ug/ug_vip.pdf.

[13] A. Darabiha, J. Rose, and W.J. MacLean. Video-rate stereo depth measurement on programmable hardware. 2003.

[14] A. Fusiello, V. Roberto, and E. Trucco. Efficient stereo with multiple windowing. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 858–863. Citeseer, 1997.

[15] M. Gong, R. Yang, L. Wang, and M. Gong. A performance study on different cost aggregation approaches used in real-time stereo matching. *International Journal of Computer Vision*, 75(2):283–296, 2007.

[16] M. Hariyama, T. Takeuchi, and M. Kameyama. VLSI processor for reliable stereo matching based on adaptive window-size selection. In *PROC IEEE INT CONF ROB AUTOM*, volume 2, pages 1168–1173, 2001.

[17] H. Hirschmueller, P.R. Innocent, and J. Garibaldi. Real-time correlation-based stereo vision with reduced border errors. *International Journal of Computer Vision*, 47(1):229–246, 2002.

[18] H. Hirschmueller and D. Scharstein. Evaluation of cost functions for stereo matching. In *IEEE Conference on Computer Vision and Pattern Recognition, 2007. CVPR'07*, pages 1–8, 2007.

[19] H. Hirschmueller and D. Scharstein. Evaluation of stereo matching costs on images with radiometric differences. *IEEE transactions on pattern analysis and machine intelligence*, pages 1582–1599, 2008.

[20] M. Humenberger, C. Zinner, M. Weber, W. Kubinger, and M. Vincze. A fast stereo matching algorithm suitable for embedded real-time systems. *Computer Vision and Image Understanding*, 2010.

[21] H. Jeong and S.C. Park. Trellis-based systolic multi-layer stereo matching. In *IEEE Workshop on Signal Processing Systems, 2003. SIPS 2003*, pages 257–262, 2003.

[22] Y. Jia, X. Zhang, M. Li, and L. An. A miniature stereo vision machine (MSVM-III) for dense disparity mapping. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004*, volume 1, 2004.

[23] S. Jin, J. Cho, XD Pham, KM Lee, S.K. Park, M. Kim, and JW Jeon. FPGA Design and Implementation of a Real-time Stereo Vision System. *IEEE Transactions on Circuits and Systems for Video Technology*, 2010.

[24] T. Kanade, A. Yoshida, K. Oda, H. Kano, and M. Tanaka. A stereo machine for video-rate dense depth mapping and its new applications. In *cvpr*, page 196. Published by the IEEE Computer Society, 1996.

[25] P. Kauff, N. Brandenburg, M. Karl, and O. Schreer. Fast hybrid block-and pixel-recursive disparity analysis for real-time applications in immersive tele-conference scenarios. In *Proc. of WSCG*, volume 9. Citeseer, 2000.

[26] S. Kosov, T. Thormahlen, and H.P. Seidel. Accurate real-time disparity estimation with variational methods. *Advances in Visual Computing*, pages 796–807, 2009.

[27] M. Kuhn, S. Moser, O. Isler, F.K. Gurkaynak, A. Burg, N. Felber, H. Kaeslin, and W. Fichtner. Efficient ASIC implementation of a real-time depth mapping stereo vision system. In *Proceedings of the of the 46th IEEE Midwest International Symposium on Circuits and Systems (MWSCAS03)*, pages 1478–1481. Citeseer.

[28] J. Lu, G. Lafruit, and F. Catthoor. Anisotropic local high-confidence voting for accurate stereo correspondence. In *Proceedings of SPIE*, volume 6812, page 68120J, 2008.

[29] J. Lu, S. Rogmans, G. Lafruit, and F. Catthoor. High-speed stream-centric dense stereo and view synthesis on graphics hardware. In *IEEE 9th Workshop on Multimedia Signal Processing, 2007. MMSP 2007*, pages 243–246, 2007.

[30] D. Masrani and W. MacLean. A real-time large disparity range stereo-system using FPGAs. *Computer Vision–ACCV 2006*, pages 42–51, 2006.

[31] Y. Miyajima and T. Maruyama. A real-time stereo vision system with FPGA. In *Field-Programmable Logic and Applications*, pages 448–457. Springer, 2003.

[32] S. Mukai, D. Murayama, K. Kimura, T. Hosaka, T. Hamamoto, N. Shibuhisa, S. Tanaka, S. Sato, and S. Saito. Arbitrary view generation for eye-contact communication using projective transformations. In *Proceedings of the 8th International Conference on Virtual Reality Continuum and its Applications in Industry*, pages 305–306. ACM, 2009.

[33] Sungchan Park and Hong Jeong. Real-time Stereo Vision FPGA Chip with Low Error Rate. 2007.

[34] J. Salmen, M. Schlipsing, J. Edelbrunner, S. Hegemann, and S. Luke. Real-Time Stereo Vision: Making More Out of Dynamic Programming. In *Computer Analysis of Images and Patterns*, pages 1096–1103. Springer, 2009.

[35] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International journal of computer vision*, 47(1):7–42, 2002.

[36] D.E. Simon. *An embedded software primer*. Addison-Wesley, 1999.

[37] Q. Tian and M.N. Huhns. Algorithms for subpixel registration. *Computer Vision, Graphics, and Image Processing*, 35(2):220–233, 1986.

[38] F. Tombari, S. Mattoccia, L. Di Stefano, and E. Addimanda. Classification and evaluation of cost aggregation methods for stereo correspondence.

[39] F. Tombari, S. Mattoccia, L. Di Stefano, and E. Addimanda. Near real-time stereo based on effective cost aggregation. In *19th International Conference on Pattern Recognition, 2008. ICPR 2008*, pages 1–4, 2008.

[40] Middlebury University. *Middlebury Stereo Evaluation - Version 2*, 2010. http://vision.middlebury.edu/stereo/eval/.

[41] M.A. Vega-Rodríguez, J.M. Sánchez-Pérez, and J.A. Gómez-Pulido. An FPGA-based implementation for median filter meeting the real-time requirements of automated visual inspection systems. In *Proceedings of the 10th Mediterranean Conference on Control and Automation*. Citeseer, 2007.

[42] O. Veksler. Fast variable window for stereo correspondence using integral images. 2003.

[43] L. Wang, M. Liao, M. Gong, R. Yang, and D. Nister. High-quality real-time stereo using adaptive cost aggregation and dynamic programming. 2006.

[44] J. Woodfill and B. Von Herzen. Real-time stereo vision on the PARTS reconfigurable computer. In *IEEE Symposium on FPGAs for Custom Computing Machines*, volume 4. Citeseer, 1997.

[45] J.I. Woodfill, G. Gordon, and R. Buck. Tyzx deepsea high speed stereo vision system. 2004.

[46] Q. Yang, L. Wang, R. Yang, S. Wang, M. Liao, and D. Nister. Real-time global stereo matching using hierarchical belief propagation. In *The British Machine Vision Conference*, pages 989–998, 2006.

[47] R. Yang, M. Pollefeys, and S. Li. Improved real-time stereo on commodity graphics hardware. In *Conference on Computer Vision and Pattern Recognition Workshop, 2004. CVPRW'04*, pages 36–36, 2004.

[48] K.J. Yoon and I.S. Kweon. Adaptive support-weight approach for correspondence search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(4):650–656, 2006.

[49] K. Zhang, J. Lu, and G. Lafruit. Cross-based local stereo matching using orthogonal integral images. *IEEE Transactions on Circuits and Systems for Video Technology*, 19(7):1073–1079, 2009.

[50] K. Zhang, J. Lu, G. Lafruit, R. Lauwereins, and L. Van Gool. Real-time accurate stereo with bitwise fast voting on CUDA. *5th IEEE workshop on embedded computer vision, held in conjunction with ICCV 2009.*