

# An FPGA-Based Real-Time System for 3D Stereo Matching, Combining Absolute Differences and Census with Aggregation and Belief Propagation

Kyprianos Papadimitriou, Sotiris Thomas, and Apostolos Dollas

School of ECE, Technical University of Crete  
Akrotiri, Chania, 73100, Greece  
{kpapadim, sthomas, dollas}@mhl.tuc.gr

**Abstract.** The implementation of 3D stereo matching in real time is an important problem for many vision applications and algorithms. The current work, extending previous results by the same authors, presents in detail an architecture which combines the methods of Absolute Differences, Census, and Belief Propagation in an integrated architecture suitable for implementation with Field Programmable Gate Array (FPGA) logic. Emphasis on the present work is placed on the justification of dimensioning the system, as well as detailed design and testing information for a fully placed and routed design to process 87 frames per sec (fps) in  $1920 \times 1200$  resolution, and a fully implemented design for  $400 \times 320$  which runs up to 1570 fps.

**Keywords:** stereo matching, correspondence problem, real-time, field programmable gate arrays, absolute differences, census, belief propagation

## 1 Introduction

Stereo vision is a research area in which progress is made for some decades now, and yet emerging algorithms, technologies, and applications continue to drive research to new advancements. The purpose of stereo vision algorithms is to construct an accurate depth map out of two or more images of the same scene, taken under a slightly different angle/position. In a set of two images one image has the role of the reference image while the other is the non-reference one. The basic problem of finding pairs of pixels, one in the reference image and the other in the non-reference image that correspond to the same point in space, is known as the correspondence problem and has been studied for many decades [1]. The difference in coordinates of the corresponding pixels (or similar features in the two stereo images) is the disparity. Based on the disparity between corresponding pixels and on stereo camera parameters such as the distance between the two cameras and their focal length, one can extract the depth of the related point in space by triangulation. This problem has been widely researched by



the computer vision community and appears not only in stereo vision but in other image processing topics as well such as optical flow calculation [2]. The range of applications of 3D stereo vision cannot be underestimated, with new fields of application emerging continuously, such as in recent research on shape reconstruction of space debris [3].

The class of algorithms which we study falls into the broad category of producing dense stereo maps. An extensive taxonomy of dense stereo vision algorithms is available in [4], and an online constantly renewed comparison can be found in [5], containing mainly software implementations. In general, the algorithm searches for pixel matches in an area around the reference pixel in the non-reference frame. This entails a heavy processing task as for each pixel the 2D search space should be exhaustively explored. To reduce the search space, a constraint called epipolar line can be applied. This constraint aims at reducing the 2D area search space to a 1D line by assuming that the two cameras are placed on the same horizontal axis (much like the human eyes) and that the corresponding images do not have a vertical displacement, thus the pixels which correspond to the same image location are only displaced horizontally. The epipolar line constraint is enforced through a preprocessing step called rectification, which is applied to the input pair of stereo images. In this work we concentrate on the stereo correspondence algorithm and not on the rectification step, assuming that images are rectified prior to processing. We present an FPGA-based implementation that is scalable and can be adjusted to the application at hand, offering great speed-up over a software implementation. Essentially, we extend our work published in [6], by including more results and a detailed analysis on aspects related to performance and resource utilization. We should note here that stereo matching is embarrassingly parallel and thus someone would reasonably expect getting high performance gains from a custom hardware implementation. Hence, our contributions go beyond solely achieving high performance results, and these are:

- an analysis showing how the use of aggregation alleviates the need to employ the more computationally demanding Sum of Absolute Differences (SAD) algorithm while still maintaining good results;
- an analysis on how to dimension the combination of the Absolute Differences (AD) and the Census algorithms with aggregation in a single hardware implementation;
- the FPGA-based architecture with detailed tradeoff analysis in the use of its primitive resources (Block RAM, Flip-Flops, logic slices), which justifies the use of FPGAs in the field of stereo vision;
- a placed-and-routed design allowing real-time processing up to 87 fps for full HD  $1920 \times 1200$  frames in a medium-size FPGA;
- a modification at the final phase of design cycle that improved by 1.6x the system performance;
- a detailed cost vs. accuracy analysis and on-FPGA RAM usage for design optimization.



The chapter is organized as follows: Section 2 discusses previous work, focusing mainly on hardware-related studies, and with a more up-to-date comparison of recent research results vs. those in our previous work [6]. Section 3 describes the algorithm and its individual steps. Section 4 analyses the benefits of mapping the algorithm to an FPGA with emphasis on dimensioning, and especially on the usefulness of aggregation in addition to AD and Census vs. the SAD algorithm. An in-depth discussion of our system is given in Section 5, including the implementation of belief propagation. Section 6 has the system performance and the usage of resources, and Section 7 summarizes the chapter.

## 2 Relevant Research

In recent years there is considerable work on 3D stereo vision, and in particular on hardware systems to support real-time 3D stereo vision. Most of these results are with FPGA technology, although there exist approaches with Digital Signal Processors (DSP) and Graphics Processor Units (GPU). Due to its intrinsic heavy parallelization and pipelining, it is one of the most promising candidates that can benefit from hardware implementation. However, several factors should be considered when it comes to develop such a design. The main factors are the maximum resolution supported and whether processing can be done at real-time; a rate of 30 fps is desirable for the human eye, but higher rates might be useful in industrial applications. Table 1 consolidates representative implementations in different technologies, with information on the maximum resolution and processing rate.

**Table 1.** Implementation of 3D Stereo Vision in different technologies

Ref	Resolution	Disparity	fps	Technology	Year
[7]	$160 \times 120$	32	30	DSP	1997
[8]	$320 \times 240$	20	150	Spartan-3	2007
[9]	$320 \times 240$	16	75	Virtex-II	2010
[10]	$320 \times 240$	16	574	GPU	2010
[6]	$400 \times 320$	64	1570	Virtex-5	2013
[11]	$640 \times 480$	128	30	4 Stratix S80	2006
[12]	$640 \times 480$	64	230	Virtex-4	2010
[13]	$640 \times 480$	54	>30	Spartan-6	2013
[14]	$640 \times 480$	60	507.9	Virtex-6	2014
[14]	$1024 \times 768$	60	199.3	Virtex-6	2014
[15]	$1920 \times 1080$	300	30	Spartan-6	2011
[6]	$1920 \times 1200$	64	87	Virtex-5	2013



The work in [7] was one of the earliest ones to combine the development of cost calculation with the Laplacian of Gaussian in a DSP. More recently, several works developed different algorithms in fully functional FPGA-based systems ranging from relatively simple [8, 9] to more complex ones [11, 12, 15, 14]. The authors of [12, 15] have designed full stereo vision systems incorporating the rectification preprocessing step. The work in [10] provides designs of an algorithm based on census transform in three different technologies, i.e. CPU, GPU and DSP; the maximum performance was obtained with the GPU. The authors in [16] introduced a local stereo matching scheme, making use of a guided filter for weighted cost aggregation to achieve impressive results relative to the quality of local algorithms. Their implementation on a GPU achieved real-time performance with 23 fps on average. The authors of [17] compare FPGA and GPU implementations of stereo vision to expose the trade-off between the flexibility but relatively low speed of an FPGA, and the high speed and fixed architecture of the GPU; that work highlights the relative strengths and limitations of the two systems. An interesting work reviewing algorithms suitable for low-cost FPGA implementation was published in [18], concluding that the memory footprint of the algorithm is the most important consideration given the limited on-chip memory of FPGAs; three different algorithms were demonstrated as a part of a real-time self-contained stereo vision system based on a Xilinx Spartan 6.

The supported disparity is an important parameter that scales with the image resolution. As shown in Table 1, disparity for medium resolutions should be between 64 and 128; this was the case for our functional prototype as well. Our system surpasses all previous systems in terms of performance. The system we implemented in a Xilinx Virtex-5 FPGA sustains a processing rate of 1570 fps for  $400 \times 320$  frames. To the best of our knowledge this is far better than any published work. For  $640 \times 533$  resolution we achieved a processing rate of 589 fps, while we support  $1920 \times 1200$  resolution at a rate of 87 fps. Moreover, our analysis differs from other publications in the sense that we study the way FPGA primitive resources suit the characteristics of each stage of the stereo vision algorithm.

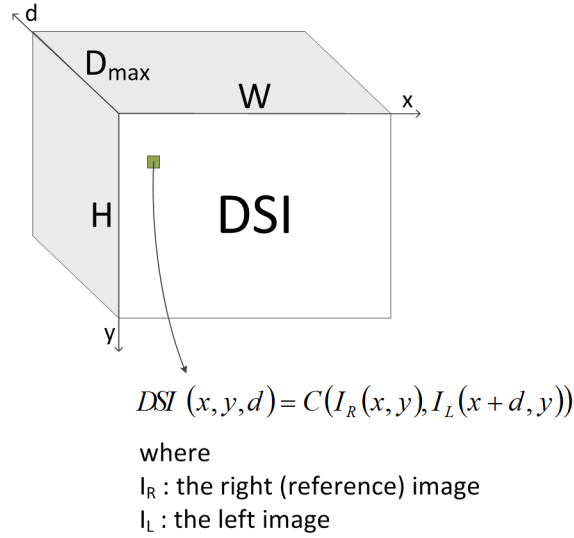
A more recent version of our system, aiming at a low-cost embeddable design has been published in [13]. This design is substantially smaller in FPGA resources vs. the current work, however, the design in [13] has fewer capabilities, including a 15% loss of 3D stereo matching capability in the near depth of field, which comes from limitations in the number of pixels among which the disparities are computed (54 in [13] vs. 64 in the present work), and the number of frames per second was deliberately lowered to 30 in order to reduce power consumption; however, with a higher clock rate a higher fps rate could be achieved.

### 3 The Algorithm

A typical approach in stereo matching is to employ a local algorithm which matches corresponding pixels in the image pair. This local algorithm computes matching costs between a pixel in the reference frame and a set of pixels in the



target frame and selects the match with the minimum cost. This is known as the Winner-Take-All (WTA) strategy, according to which the algorithm selects the match with the global minimum cost in the search space. Essentially, this process is equivalent to computing a 3D cost matrix (called Disparity Search Image or DSI, shown in Figure 1) of size  $W \times H \times D_{max}$ , - where  $W$  the frame width,  $H$  the frame height and  $D_{max}$  the size of the search space - and selecting the index of the minimum in the  $D_{max}$  dimension. To improve the results, usually a cost aggregation step that acts on the DSI is interjected between the cost computing and match selecting steps. Post-processing steps can further refine the resulting disparity map.



**Fig. 1.** Disparity Search Image (DSI) volume

Our algorithm consists of the cost computation step implemented by the Absolute Difference (AD) census combination matching cost, a simple fixed window aggregation scheme, a left/right consistency check and a scan-line belief propagation solution as a post processing step. Each step of the algorithm will be explained below, whereas the justification for the choice of this combination of algorithms will become evident from quantitative data in Section 4.

The AD measure is defined as the absolute difference between two pixels,  $C_{AD} = |p_1 - p_2|$ , while census [19] is a window based cost that assigns a bit-string to a pixel and is defined as the sum of the Hamming distance between the bit-strings of two pixels. Let  $W_c$  be the size of the census window. A pixel's bit-string is of size  $W_c^2 - 1$  and is constructed by assigning 1 if  $p_i > p_c$  or 0 otherwise, for  $p_i \in Window$ , and  $p_c$  the central pixel of the window. The two

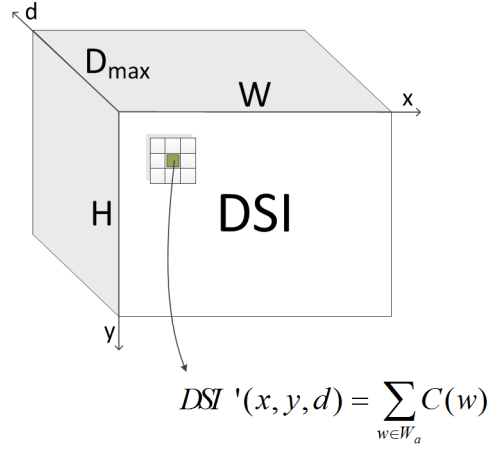


costs are fused by a truncated normalized sum:

$$C(p_R, p_T) = \max\left(\frac{C_{AD}(p_R, p_T)}{C_{AD}^{Max}} + \frac{C_{Census}(p_R, p_T)}{C_{Census}^{Max}}, \lambda_{trunc}\right) \quad (1)$$

where  $\lambda_{trunc}$  is the truncation value given as parameter. This matching cost encompasses image local light structure (census) as well as information about the light itself (AD), and produces better results than its parts alone, as was shown in [20]. At object borders, the aggregation window necessarily includes costs belonging to two or more objects in the scene, whereas ideally we would like to aggregate only costs of one object. For this reason, truncating the costs to a maximum value helps at least limiting the effect of any outliers in each aggregation window [4].

After initializing the DSI volume with AD-Census costs, we perform a simple fixed window aggregation on the  $W \times H$  slices of the DSI, illustrated in Figure 2. This is based on the assumption that neighbouring pixels (i.e. pixels belonging in the same window) most likely share the same disparity (depth) as well. Although this does not stand for object borders and slanted surfaces, it produces good results. On the other hand, one should select carefully the size of the aggregation window  $W_a$ , as large windows tend to lead to an edge fattening effect in object borders while small aggregation windows lead to loss of accuracy in the inside area of an object itself, which results in a noisy output.



**Fig. 2.** Example of 3x3 fixed window aggregation of DSI costs

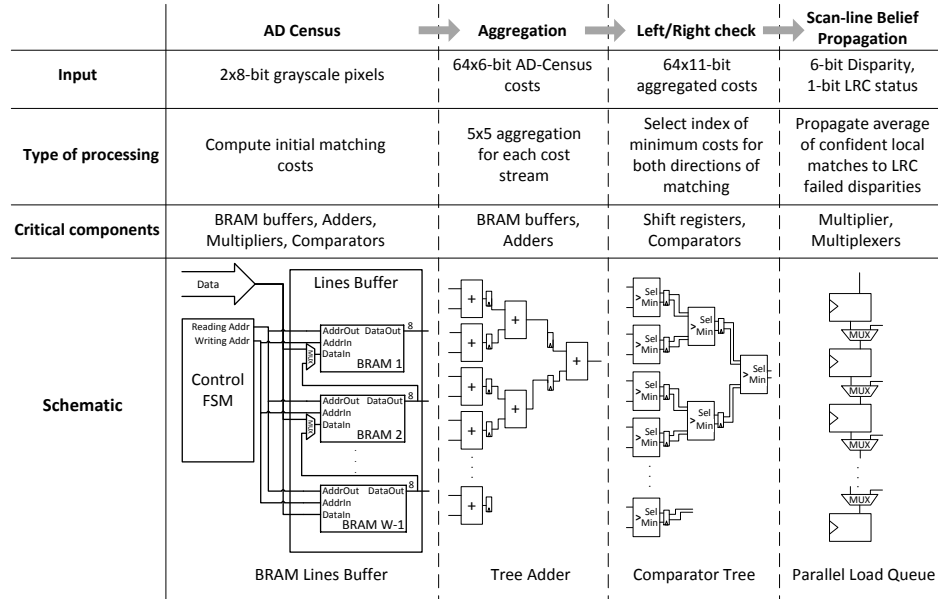
Finally, we perform a left/right consistency check (LRC check) which repeats the match selection step but with the opposite frame as reference and compares the new disparity image with the original one. This process allows to detect mismatches due to occlusions (areas of the scene that appear only in one frame).



Using the mismatches detected, our scan-line belief propagation solution propagates local confident matches along the scan-line, by accumulating matches that passed the LRC check in a queue (called confident queue due to that it stores only disparities that passed the LRC check), and propagating them to local matches classified as occlusions in a neighborhood queue.

## 4 Dimensioning of the FPGA Architecture

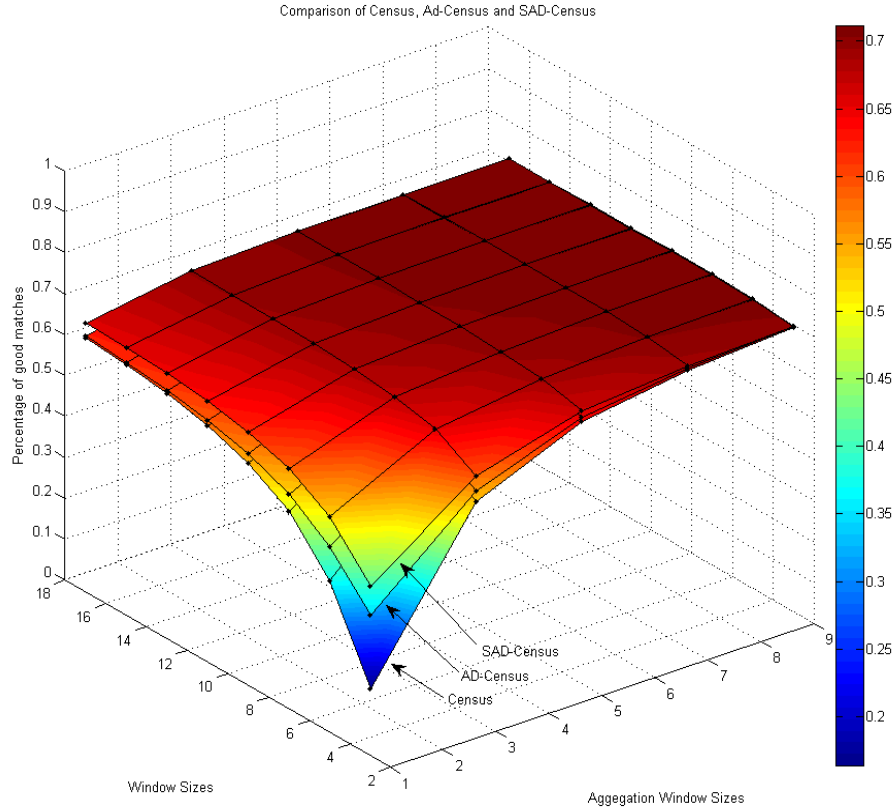
The algorithm can be mapped on an FPGA very efficiently due to its intrinsic parallelism. For instance, the census transform requires  $W_c^2 - 1$  comparisons per pixel to compute the bit-string. Aggregation also requires  $W_a^2$  additions per cost. For each pixel we must evaluate 64 possible matches by selecting the minimum cost. These operations can be done in parallel. The buffer architecture requires memories to be placed close to each other, as they shift data between them in a very regular way. FPGA memory primitives (BRAMs) are located in such a way they facilitate this operation. Figure 3 shows the critical components for the different steps of the algorithm. Our system shares the pixel clock of the cameras and processes the incoming pixels in a streaming fashion as long as the camera clock does not surpass the system's maximum frequency. This way we avoid building a full frame buffer; we instead keep only the part of the image that the algorithm is currently processing.



**Fig. 3.** Algorithm stages and critical components that fit well into FPGA regular structures



It is important to assess the need for flexibility regarding the algorithm parameters, and the gains of such a setup. First and foremost, we are seeking to build a system that is frame-agnostic. Stating differently, we aim at supporting a series of frame sizes within a range of choices; we regard this feature as obligatory. However, a limit on the maximum frame width was imposed for reasons explained in Section 6. In addition, all the algorithmic parameters are adjustable; the maximum disparity search range  $D_{max}$ , the census window size  $W_c$ , and the aggregation window size  $W_a$ . We chose to structure our system in a modular way in order to easily add/remove features. Features such as scanline belief propagation and aggregation can be turned on or off selectively by the user. Figure 4 shows performance results without aggregation and with various aggregation window sizes.

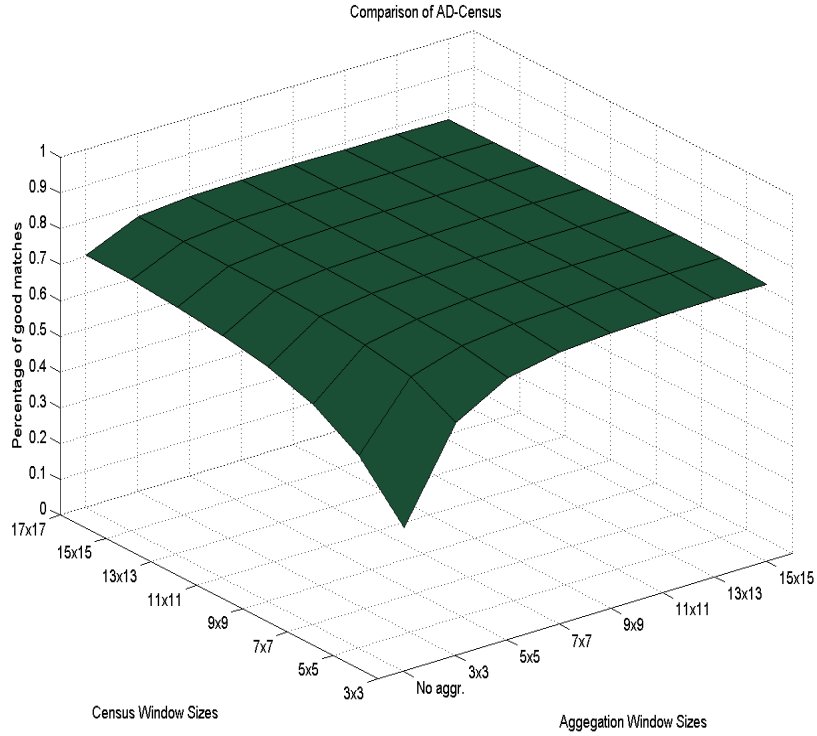


**Fig. 4.** Comparison of Census, AD and SAD performance with various aggregation window sizes



In order to develop an efficient architecture it is important to understand how resources are used. In terms of sheer performance when no aggregation is used, the SAD algorithm is the best, and so it would seem that it is best to implement it in hardware. No aggregation means that the aggregation window is of size 1. , when computational cost is considered, the SAD algorithm is by far more expensive than the alternatives, as it has approximately  $2 \times W^2$  comparisons. It is therefore useful to consider cost vs. performance when we introduce aggregation to the system, where we notice that the system-level performance with aggregation comes close to the SAD performance, but at a lower computational cost.

The next question to answer in the development of a useful architecture is the tradeoff between the census window size vs. the aggregation window size. The effects of different census window sizes and different aggregation window sizes is illustrated in Figure 5, which contains in a more clear form the information from Figure 4 for the algorithms which were actually implemented in our system.



**Fig. 5.** Quality results in terms of the good matches for different census and aggregation window sizes, when using the AD-Census



We analyzed the influence of the algorithm’s parameters on the quality metric of percentage of good matches, over six (6) datasets of Middlebury’s 2005 database [5]. We have settled on a  $W_c = 9 \times 9$  sized census window, a  $W_a = 5 \times 5$  sized aggregation window and a  $D_{max} = 64$  disparity search range; these values offer a good trade-off between overall quality and computational requirements. We followed a similar procedure to determine all the other secondary parameters as well, such as the confident neighborhood queue size and the neighborhood queue size of the scan-line belief propagation module, and the LR check threshold of the LR consistency check module [10].

There are negligible gains if we choose a larger  $W_c$  or  $W_a$ . The maximum achievable percentage of good matches was 78,36% for AD-Census ( $W_c=7$ ,  $W_a=13$ ), therefore there is no actual benefit by choosing a large aggregation window. It is thus our choice to fix the window sizes in our implementation. Our design remains generic in any parameter aspect but it is not reconfigurable at runtime. This decision simplifies our hardware design. For purposes of evaluation and experimental verification of the design we designed our system using Xilinx FPGAs, namely, a Virtex 5 XC5VLX110T as well as a Spartan 3 1000, setting the parameters accordingly to fit the FPGA device at hand.

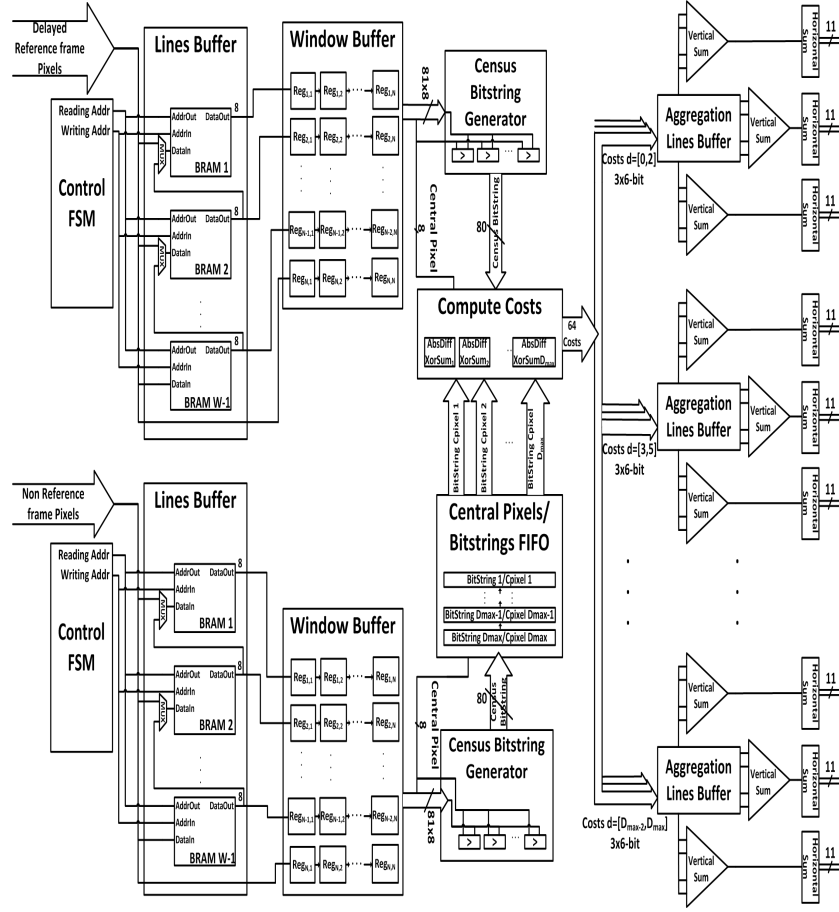
Last but not least, we need to consider what happens with occluded pixels from one or the other camera. It is therefore useful to allow for some resources to be used for Belief Propagation (BP), as shown in Section 5. Belief propagation (which uses results from the Left-Right consistency check) does not consume significant resources but it solves the problem of uncertainty due to occluded pixels which would result if only one camera were used as the only reference image.

## 5 Design and Implementation

The system in Figure 6 receives two 8-bit pixel values per clock period, each one for the corresponding image in the stereo pair. A window buffer is constructed for each data flow in two steps. Lines Buffer stores  $W_c - 1$  scan-lines of the image, each in a BRAM, conceptually transforming the single pixel input of our system to a  $W_c$  sized column vector. Window Buffer acts as a  $W_c$  sized buffer for this vector, essentially turning it into a  $W_c^2$  matrix. This matrix is subsequently fed into Census Bitstring Generator of Figure 6, which performs  $W_c^2 - 1$  comparisons per clock, producing the census bit-string. Central pixels/Bit-strings FIFO stores 64 non-reference census bit-strings and window central pixels, which along with the reference bit-string and central pixel are driven to 64 Compute Cost modules. This component performs the XOR/summing that is required to produce the Hamming distance for the census part of the cost, along with the absolute difference for the AD part and the necessary normalization and addition of the two. The maximum census cost value is 80 as there are 81 pixels in the window excluding the central pixel from calculations. Likewise, the maximum AD cost value is 255 as each pixel is 8 bits wide. As the two have different ranges, we scale the census part from the 0-80 range to a 0-255 range, by turning it into an



8-bit value. To produce the final AD-Census cost we add the two parts together, resulting in a 9-bit cost to account for overflow. Truncating this cost to 6-bit produces a slight improvement in quality as discussed in Section 3, and also reduces buffering requirements in the aggregation step.



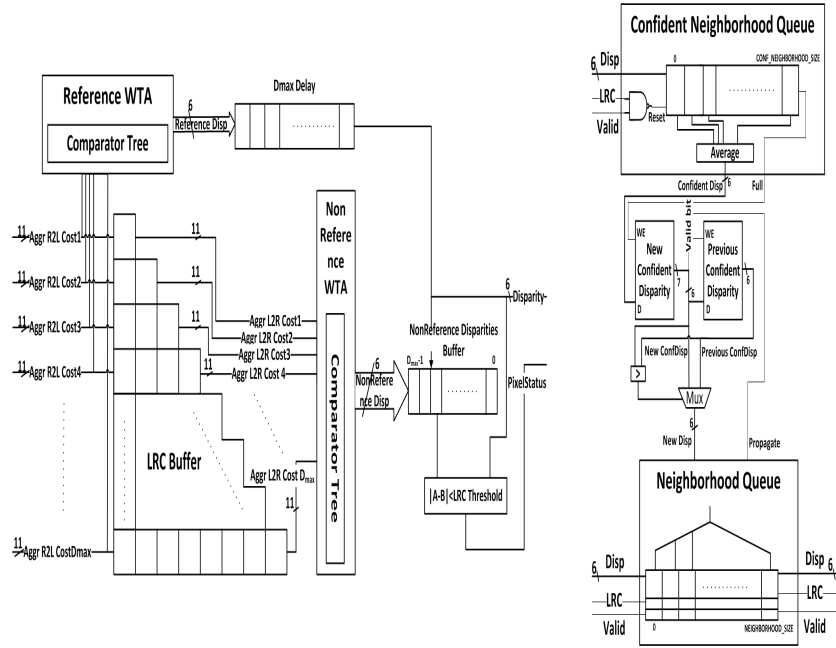
**Fig. 6.** Datapath of the cost computation (left side) and aggregation (right side)

For the aggregation stage, 22 line buffers (Aggregation Lines Buffer in Figure 6) are used for 64 streams of 6-bit costs, each lines buffer allocated to 3 streams. BRAM primitives are configured as multiples of 18K independent memories, so we maximize memory utilization by packing three costs per BRAM, accepting a maximum depth of 1024 per line. Like the Lines Buffers at the input, they conceptually transform the stream of data to  $W_a$  sized vertical vectors. Each vector is summed separately in the Vertical Sum components and driven to



delay adders (Horizontal Sum), which output  $X(t) + X(t - 1) + \dots + X(t - 4)$ . At the end of this procedure we have 64 aggregated costs.

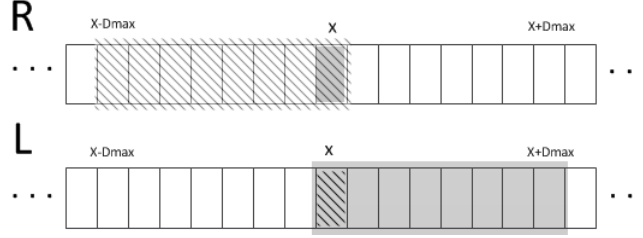
Following the aggregation of costs, the LRC component illustrated in Figure 7, filters out mismatches caused by occlusions; its operation is illustrated in Figure 8. The architecture of LRC is based on the observation that by computing the right-to-left disparity at reference pixel  $p(x, y)$ , we have already computed the costs needed to extract the left-to-right disparity at non-reference pixel  $p'(x, y)$ . The LRC buffer is a delay in the form of a ladder that outputs the appropriate left-to-right costs needed to extract the non-reference disparity. The WTA modules select the match with the best (lowest) cost using comparator trees. The reference disparity is delayed in order to allow enough time for the non-reference disparities space to build up in NonReference Disparities Buffer and then it is used to index said buffer. Finally, a threshold in the absolute difference between  $Disp_{RL}(x, y)$  and  $Disp_{LR}(x, y)$  indicates the false matches detected.



**Fig. 7.** Datapaths of the left/right consistency check(left side) and scan-line belief propagation(right side)

The datapath for the scan-line belief propagation algorithm is shown in Figure 7. The function of this component is based on two queues: the Confident Neighborhood Queue and the Neighborhood Queue. As implied by its name, the Confident Neighborhood Queue places quality constraints on its contents, meaning that only disparities passing the LR consistency check are written in it.





**Fig. 8.** Top row has pixels of the Right image, whereas bottom row has pixels of the Left image. The grey pixel in the center of the Right image has a search space in the Left image shown with the broad grey area. To determine the validity of  $Disp_{RL}(x, y)$ , we need all left-to-right disparities in the broad grey area, thus we need right-to-left costs up to  $x + D_{max}$ . The same stands for the diagonal shaded pixel in the center of the Left image.

Furthermore, at each cycle it calculates the average of the confident disparities, as this value will ultimately be propagated to non confident ones in the neighborhood queue. This average is calculated by a constant multiplier, using fixed point arithmetic and rounding to reduce any number representation errors. On the other hand, the Neighborhood Queue simply keeps track of local disparities and their LR status. When the Propagate signal is asserted (active when a new confident disparity is calculated and stored in the New Confident Disparity register), the NewDisp is written to all records with a false LRC flag. NewDisp is selected to be Previous Confident Disparity when this value is smaller than New Confident Disparity, otherwise it is assigned New Confident Disparity, effectively propagating confident background depths.

We report, below, some noticeable points regarding system operation:

- A small frame of unknown disparities is formed around the final disparity image, where due to the image boundaries, the windows cannot be formed and thus disparities cannot be computed.
- Searching for matches near the image boundary leads to reduced or trivial disparity search spaces. In such cases our system finds the best match in the possible range. Left/Right Consistency check filters out any incorrect matches due to small search spaces.
- Belief propagation is activated on each end of line regardless of the LRC indication, in order to fill in the occluded right area of the image.
- In order for high resolutions to work acceptably, the user needs to either increase the  $D_{max}$  accordingly or alternatively to reduce the inter-camera spacing. The second option is preferable as it has no computational consequences, whereas increasing  $D_{max}$  has a cost on FPGA resources. Moreover, as the features of the images are now more spaced out in pixel distances, the other parameters of the system need also to be adjusted in order to maintain output quality.



## 6 Performance Evaluation and Resource Utilization

The system which was described, above, can process one pixel pair per clock period, after an initial latency. The most computationally intensive part of the main stage of the algorithm lies in the XOR/sum module of the AD census, which computes the XOR/sum of 64 80-bit strings at the same time. A similar situation stands for the WTA module, which performs 64 11-bit simultaneous comparisons. We cope with both bottlenecks through fully pipelined adder/comparator trees in order to increase the throughput. After the initial implementation, we added extra pipeline stages to further enhance performance. Below we present the differences between the initial (unoptimized) design, and the second (optimized) design. Table 2 shows the performance results of the system implemented in a Xilinx Virtex-5 FPGA. The maximum clock after place and route for the unoptimized design is 131MHz, while for the optimized design is 201MHz. Table 3 has the differences in resource utilization between the two designs; it demonstrates that for a speed improvement of over 50%, the resource utilization penalty is rather small. Based on data we gathered from the tools, the critical path lies on a control signal driving the FSM of the aggregation line buffers; 16,6% is attributed to logic while the rest 83,4% of the delay is due to routing.

**Table 2.** Design clock and processing rates for the optimized vs. unoptimized design in Virtex XC5VLX110T FPGA for various resolutions

	100x83	384x320	644x533	1024x853	1600x1333	1920x1200
Unoptimized design (131MHz)	15,783fps	1,066fps	384fps	150fps	61fps	56fps
Optimized design (201MHz)	24,216fps	1,635fps	589fps	230fps	94fps	87fps

**Table 3.** Resource utilization of the unoptimized vs. optimized design in Virtex XC5VLX110T FPGA for  $D_{max} = 64$ ,  $W_c = 9$ ,  $W_a = 5$

	Slices (%)	LUTs (%)	Flip-Flops (%)	BRAMs (%)
Available	17,280	69,120	69,120	148
Unoptimized design (131MHz)	13,556 (78%)	37,107 (53%)	39,565 (57%)	59 (39%)
Optimized design (201MHz)	14,239 (82%)	37,986 (55%)	41,792 (60%)	59 (39%)

We should point out here that  $W_c$ ,  $W_a$  and  $D_{max}$  parameters are related with tasks carried out in parallel, thus they do not affect system performance but only resource utilization. Table 4 has the distribution of resources along with the percentage breakdown in each type of resources for the optimized design.



**Table 4.** Resource utilization of the optimized design in Virtex XC5VLX110T FPGA for  $D_{max} = 64$ ,  $W_c = 9$ ,  $W_a = 5$

	LUTs (%)	Flip-Flops (%)	BRAMs (%)
Available	69,120	69,120	148
Total consumed	37,986/69,120 (55%)	41,792/69,120 (60%)	59/148 (40%)
AD Census	25,135/37,986 (66%)	29,167/41,792 (70%)	8/59 (14%)
Aggregation	6,547/37,986 (17%)	7,312/41,792 (17%)	51/59 (86%)
Left/Right Check	4,638/37,986 (12%)	4,734/41,792 (11%)	0/59 (0%)
Scanline Belief Propagation	543/37,986 (1.5%)	634/41,792 (1.5%)	0/59 (0%)

We conducted several experiments by varying the parameters in each stage so as to assess system performance in terms of scalability and increase in resource utilization. Tables 5, 6, 7 have the FPGA resource results for different parameter values. It is obtained that the design clock is not affected. At the same time we observe that once a parameter increases, the amount of resources needed for a resource category might increase drastically while another category is not affected, i.e. in Table 6 as  $W_c$  increases, the amount of flip-flops and LUTs increases as opposed to the amount of BRAMs which remains unchanged.

**Table 5.** Impact of  $D_{max}$  on resource utilization and performance, when  $W_c = 9$  and  $W_a = 5$

$D_{max}$	LUTs(%)	Flip-Flops(%)	BRAMs(%)	Max Clock
16	10,284(14%)	12,531(18%)	30(20%)	201.207MHz
32	19,148(27%)	22,687(32%)	30(20%)	201.045MHz
64	37,986(54%)	41,792(60%)	59(39%)	201.518MHz

**Table 6.** Impact of  $W_c$  on resource utilization and performance, when  $D_{max} = 64$  and  $W_a = 5$

$W_c$	LUTs(%)	Flip-Flops(%)	BRAMs(%)	Max Clock
5	21,637(31%)	21,866(31%)	59(39%)	201.086MHz
7	29,813(43%)	31,840(46%)	59(39%)	201.113MHz
9	37,986(54%)	41,792(60%)	59(39%)	201.518MHz

Aggregation of the costs consumes most of our BRAM resources, as we have to construct  $D_{max} \times W_a$  cost line buffers (a total of  $D_{max} \times W_a \times FrameWidth \times CostSize$  bits must be buffered). BRAM primitives of Virtex 5 FPGAs support only certain aspect ratios. The vendor tool employs these primitives to construct bigger memories, using an allocation algorithm. Memories with different



widths/depths from those ratios are mapped to the closest possible solution but may not use the resources optimally. Memories with ratios of  $1 \times 16K$  (16,384 elements of 1-bit),  $2 \times 8K$ ,  $4 \times 4K$ ,  $9 \times 2K$ ,  $18 \times 1K$ ,  $36 \times 512$  are guaranteed to utilize a single 18K primitive and thus use the resources optimally.

**Table 7.** Impact of  $W_a$  on resource utilization and performance, when  $D_{max} = 64$  and  $W_c = 9$

$W_a$	LUTs(%)	Flip-Flops(%)	BRAMs(%)	Max Clock
1 (off)	28,505(41%)	33,047(47%)	9(6%)	201.005MHz
3	34,618(50%)	38,660(55%)	31(20%)	201.167MHz
5	37,986(54%)	41,792(60%)	59(39%)	201.518MHz

In addition, very large frame sizes cause parameter bloating. In specific, images with  $1800 \times 1500$  resolution require at least  $D_{max} = 180$  for achieving satisfactory results in terms of quality (without altering the current camera baseline). While keeping the other parameters constant ( $W_c = 9$ ,  $W_a = 5$ ), such a large  $D_{max}$  would require buffering  $180 \times 5 \times 1800$  elements in the aggregation stage.

Due to the above we decided to put a limit on the image width. Restricting the frame width to 1024 pixels allowed us to:

- Pack at least two lines per 18K BRAM using a  $18 \times 1K$  BRAM primitive configuration. For each cost line we allocate  $9 \times 1024$  bits.
- Avoid excessive parameter bloating.

Using AD-Census, the costs are 9-bit long as described earlier. This benefits our design as BRAM primitives can be used optimally in a  $18 \times 1K$  configuration. Using pure Census, cost size is reduced to 7-bits. We can maximize BRAM usage by using 9-bit costs, so we have room to increase census window size  $W_c$  up to  $21 \times 21$ , with little additional cost to resource usage.

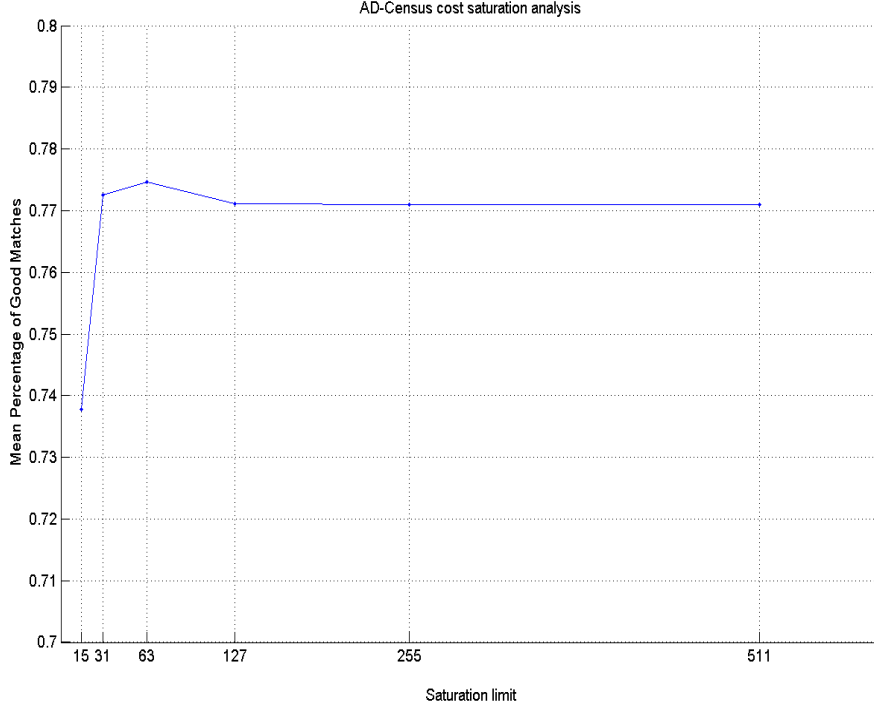
If the cost size is less than 9-bits or if the frame width is less than 1024 we can pack more lines. This aspect of our design is also parametric, as depending on the frame size and cost size, each BRAM can fit up to 6 lines in a  $36 \times 512$  BRAM configuration.

In an effort to reduce BRAM consumption even further, we performed a cost size-accuracy tradeoff experimental analysis, depicted in Figure 9. AD-Census was redefined as:

$$ADCensus' = \min(ADCensus, SaturationValue) \quad (2)$$

Selecting saturation values to be power of 2, can reduce cost size and thus fit more data into the BRAMs that implement the aggregation buffers. Our analysis shows that there is a slight benefit in doing so: for a saturation value of 63 (cost size reduced to 6-bits), and for the default  $W_c$  and  $W_a$  values of 9 and 5 respectively, we observe a 0.5% improvement over the cost without saturation.





**Fig. 9.** Cost size/accuracy analysis. The peak value shifts to the right as the true maximum cost increases.

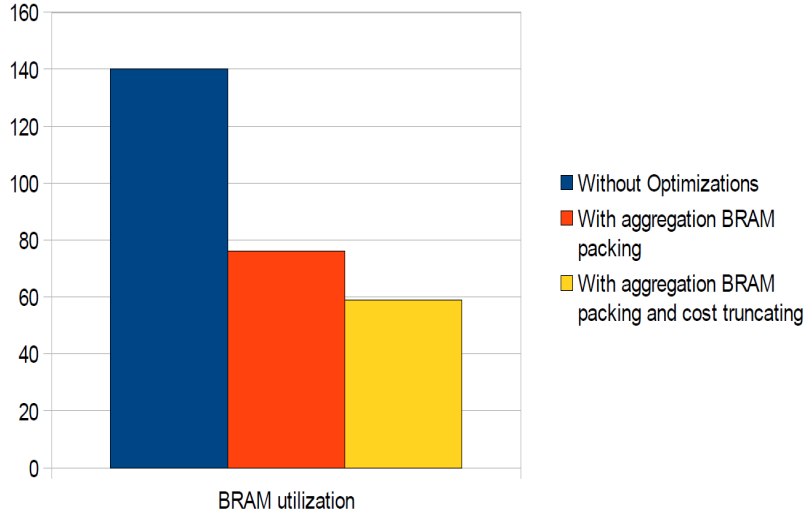
This is an important result because it puts our quality almost on par with a  $W_c = 11$  and  $W_a = 5$  parameter set. This slight improvement is attributed to the reduction of the influence of outliers within the aggregation window by truncating the cost. With 6-bit costs, we can pack 3 streams of costs per Aggregation Lines Buffer, thus reducing BRAM consumption even further. Note that all the results presented so far with regard to FPGA resource utilization, correspond to designs incorporating the previous optimizations.

Figure 10 shows the effect of optimizations on BRAM utilization for  $W_c = 9$ ,  $W_a = 5$ ,  $D_{max} = 64$  and a maximum frame width of 1024 pixels. Operating with small frame sizes allows for optimal algorithm performance.

We performed extensive verification of our designs Figure 11 has the set of images we used to test our prototype. We entered stereo images and we compared the software and the FPGA output over the ground truth. The SW version aimed to support the validation phase; we developed it in Matlab prior to the FPGA design. In terms of the physical setup for the verification, Figure 12 shows the methodology we followed to validate the FPGA system. The values of the pixels in the output of the FPGA processing were subtracted from the values of the



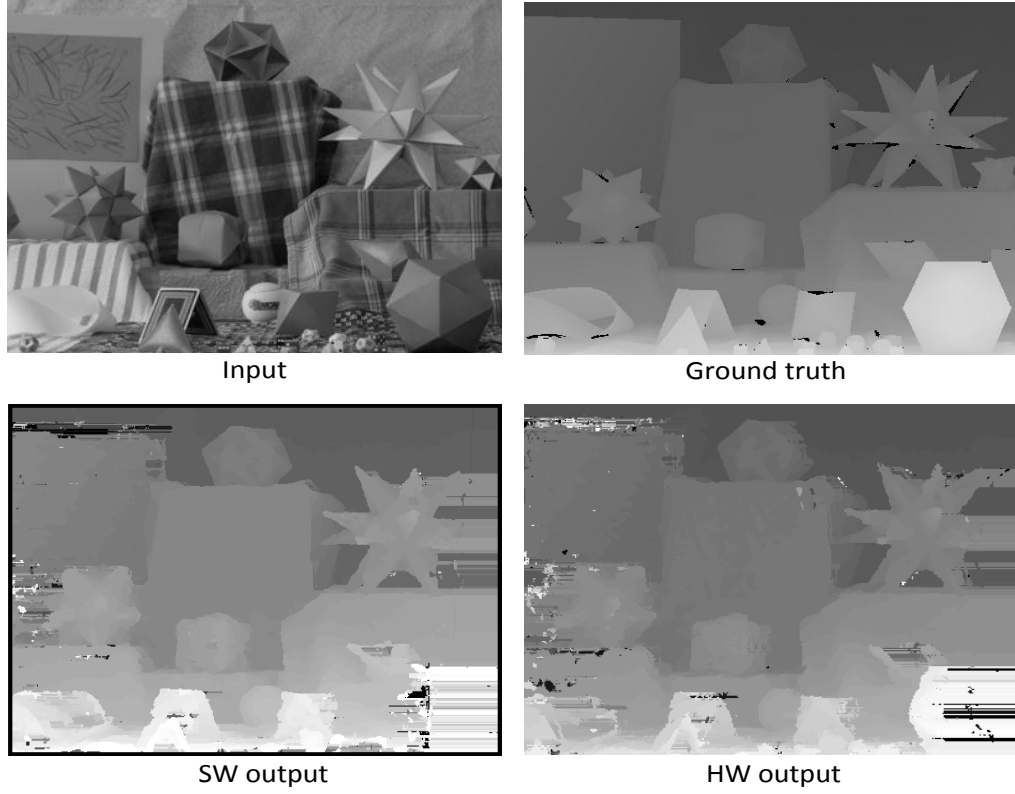
pixels in the output of the SW, pixel-per-pixel so as to create an array holding their differences. We obtained that SW and HW produced similar results. The error lines are attributed to a slightly different selection policy in the WTA process of the LRC stage. In particular, when it comes to compare two equal cost values, our SW selects one cost value randomly, while our HW selects always the first one. This variation occurs early in the algorithmic flow, thus it is not only propagated but it is also amplified in the belief propagation module where local estimates of correct disparities are spread to incorrectly matched pixels along the scan-line. Finally, the errors at the borders that occur in both SW and HW outputs as compared with the ground truth, are due to the unavoidable occlusions at the image borders.



**Fig. 10.** BRAM resource utilization with the optimized aggregation buffer structure

The actual prototype in the Virtex-5 FPGA can process images of up to  $400 \times 320$  resolution. We put our efforts on building a high-speed stereo matching design, rather than solving the I/O issue. Instead, at off-line time we send images into internal BRAMs through a serial protocol; once the entire image is stored in the BRAMs the design starts processing it. We used hardware means for measuring the time to complete the FPGA processing, and by performing experiments on different images - mainly from Middlebury database - we achieved a processing rate of 1570fps. To the best of our knowledge this outperforms any published FPGA-based system.



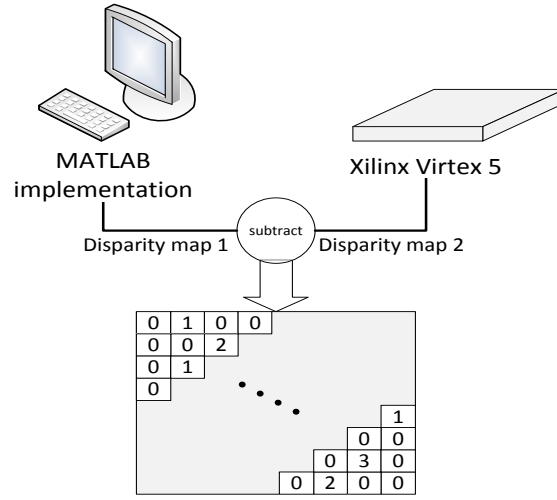


**Fig. 11.** Top row has Moebius  $400 \times 320$  input dataset from Middlebury database and the ideal (ground truth) result. The bottom row has algorithm's output from SW and HW implementations.

## 7 Conclusions and Future Work

In this chapter we presented the architecture and implementation of a real time stereo matching algorithm in an FPGA, which utilizes efficiently the strengths of this device. We validated the system on a prototype, and we exceeded real-time requirements by a large margin for various parameter configurations. In the future we will place our efforts to more advanced aggregation schemes as they are the key to better quality disparity maps for local methods. Another point that would benefit from further research is the scan-line belief propagation method, which we plan to augment to two dimensions and thus eliminate its streaking artifacts. Also we plan to complete the stereo vision core algorithm with rectification and full camera integration.





**Fig. 12.** Validation methodology

## 8 Acknowledgement

This work has been partially supported by the General Secretariat of Research and Technology (G.S.R.T), Hellas, under the project AFORMI- Allowing for Re-configurable Hardware to Efficiently Implement Algorithms of Multidisciplinary Importance, funded in the call ARISTEIA of the framework Education and Lifelong Learning (code 2427).

## References

1. D. H. Ballard and C. M. Brown, *Computer Vision*. Englewood Cliffs, NJ, US: Prentice-Hall, 1982.
2. D. Marr, *Vision*. San Francisco, CA, US: Freeman, 1982.
3. S. Di Carlo, P. Prinetto, D. Rolfo, N. Sansonne, and P. Trotta, "A novel algorithm and hardware architecture for fast video-based shape reconstruction of space debris," *EURASIP Journal on Advances in Signal Processing*, vol. 2014, no. 1, pp. 1–19, 2014.
4. D. Scharstein and R. Szeliski, "A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms," *International Journal of Computer Vision*, vol. 47, no. 1-3, pp. 7–42, April-June 2002.
5. <http://vision.middlebury.edu/stereo/eval/>.
6. S. Thomas, K. Papadimitriou, and A. Dollas, "Architecture and Implementation of Real-Time 3D Stereo Vision on a Xilinx FPGA," in *IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, October 2013, pp. 186–191.
7. K. Konolige, "Small Vision Systems: Hardware and Implementation," in *Proceedings of the International Symposium on Robotics Research*, 1997, pp. 111–116.



8. C. Murphy, D. Lindquist, A. M. Rynning, T. Cecil, S. Leavitt, and M. L. Chang, "Low-Cost Stereo Vision on an FPGA," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, April 2007, pp. 333–334.
9. S. Hadjitheophanous, C. Ttofis, A. S. Georgiades, and T. Theocharides, "Towards Hardware Stereoscopic 3D Reconstruction, A Real-Time FPGA Computation of the Disparity Map," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE)*, March 2010, pp. 1743–1748.
10. 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*, vol. 114, no. 11, pp. 1180–1202, November 2010.
11. D. K. Masrani and W. J. MacLean, "A Real-Time Large Disparity Range Stereo-System using FPGAs," in *Proceedings of the IEEE International Conference on Computer Vision Systems*, 2006, pp. 42–51.
12. S. Jin, J. U. Cho, X. D. Pham, K. M. Lee, S.-K. Park, and J. W. J. Munsang Kim, "FPGA Design and Implementation of a Real-Time Stereo Vision System," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 20, no. 1, pp. 15–26, January 2010.
13. G. Rematska, K. Papadimitriou, and A. Dollas, "A Low-Cost Embedded Real-Time 3D Stereo Matching System for Surveillance Applications," in *IEEE International Symposium on Monitoring and Surveillance Research (ISMSR), in conjunction with the IEEE International Conference on Bioinformatics and Bioengineering (BIBE)*, November 2013.
14. M. Jin and T. Maruyama, "Fast and Accurate Stereo Vision System on FPGA," *ACM Transactions on Reconfigurable Technology and Systems (TRETs)*, vol. 7, no. 1, pp. 3:1–3:24, February 2014.
15. <http://danstrother.com/2011/01/24/fpga-stereo-vision-project/>.
16. C. Rhemann, A. Hosni, M. Bleyer, C. Rother, and M. Gelautz, "Fast Cost-Volume Filtering for Visual Correspondence and Beyond," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2011, pp. 3017–3024.
17. R. Kalarot and J. Morris, "Stereo Vision Algorithms for FPGAs," in *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, June 2010, pp. 9–15.
18. S. Mattoccia, "Stereo Vision Algorithms for FPGAs," in *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2013, pp. 636–641.
19. R. Zabih and J. Woodfill, "Non-parametric Local Transforms for Computing Visual Correspondence," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 1994, pp. 151–158.
20. X. Mei, X. Sun, M. Zhou, S. Jiao, H. Wang, and X. Zhang, "On Building an Accurate Stereo Matching System on Graphics Hardware," in *Proceedings of the IEEE International Conference on Computer Vision Workshops (ICCV)*, November 2011, pp. 467–474.