A Binary Adaptable Window SoC Architecture for a StereoVision Based Depth Field Processor

Andy Motten, Luc Claesen
Expertise Centre for Digital Media
Hasselt University – tUL – IBBT
Wetenschapspark 2, 3590 Diepenbeek, Belgium
{firstname.lastname}@uhasselt.be

Abstract— **This paper presents a novel binary fully adaptable window for incorporating in a stereo matching System-on-Chip (SoC) architecture. This architecture is fully scalable and parameterizable to allow for custom SoC implementations, as well as rapid prototyping on FPGAs. For each window a binary mask window is generated which selects the supporting pixels in the cost aggregation phase of the SAD algorithm. This selection is performed using color similarity and spatial distance metrics. Hardware resource utilization for a fixed window and an adaptable window cost aggregation is compared based on FPGA logic element use.**

Keywords-component; stereo matching; adaptable window; cost aggregation; computer vision; System-on-Chip; FPGA;

I. Introduction

Stereo matching has long been an important research topic in computational video. Many stereo matching algorithms have been investigated and published. A good comparison between different algorithms can be found in the review paper of Lazaros et al. [1], and the review paper of Scharstein and Szeliski [2].

Local and window based methods calculate the differences between the left and right images from a small part of the images. They produce a decent depth image result and are suitable for real-time applications. Selection of the ideal window size is important: it should be large enough to contain distinguished features, but small enough to keep depth discontinuities.

Recently more advanced local based methods make use of color information to select the optimal support window. A good overview of these methods can be found in Wang [3].

The adaptive-weight algorithm proposed by Yoon [4] adjusts the support weight of each pixel in a fixed sized window. The support weights are depending on the color and spatial difference between each pixel in the window and the center pixel. Dissimilarities are computed based on the support weights and the plain similarity scores. Their experiment indicates that a local based stereo matching algorithm can produce depth maps similar to global algorithms.

Tombari [5] extends the adaptive-weight algorithm of Yoon [4] by using information from segmentation. It allows inclusion of connectiveness of pixels and segment shapes, instead of relying only on color and spatial distance.

Cooperative stereo algorithms optimize initially calculated disparity matches by evaluating the disparity results of the neighboring pixels. An example of such a cooperative algorithm that makes use of color information can be found in the paper of Brockers [6]. He proposes to use color information in a local window to align support areas with local borders to keep object boundaries while using a cooperative stereo method.

Implementations on hardware of these methods are difficult to find. Most stereo vision implementations on a custom SoC or a FPGA make use of a fixed cost aggregation window ([7], [8] and [9]) or an adaptable rectangular cost aggregation window [10].

This paper presents a novel binary full adaptable window for incorporating in a stereo matching System-on-Chip (SoC) architecture. For each window a binary mask window is generated which selects the supporting pixels in the cost aggregation phase of the SAD algorithm. This selection is performed using color similarity and spatial distance metrics. The architecture is an extension on the architecture presented in [11]. It consists of a multiple parallel on-chip memory architecture, an adaptable window SAD matching cost computation and a tree based minima calculation with registers to store intermediate results. This architecture is fully scalable and parameterizable to allow for custom SoC implementations, as well as rapid prototyping on FPGAs.

Hardware resource utilization for a fixed window and a binary adaptable window cost aggregation is compared based on FPGA logic element use and depth map quality obtained from the Tsukuba stereo pair [2].

II. Stereo Matching Architecture

A. Basics and Requirements

The stereo matching algorithm takes two undistorted and rectified images that have been taken by two cameras that have a vertical alignment and a horizontal offset (see Fig. 1). Objects will appear on both images on the same horizontal line (the epipolar line). The horizontal distance between the same objects on the left and right images is called the disparity.

The research described in this paper is directly funded by Hasselt University through the BOF framework.

Objects that are close to the cameras will have a larger horizontal disparity than objects which are far away. The goal of the stereo matching algorithm is to measure the disparity between all pixels in the image.

The major focus of the architecture presented in this paper is the implementation of a fully adaptable window to replace the fixed window cost aggregation algorithm. The architecture that is presented has been developed in a scalable and parameterized way. In this way a custom depth field processor SoC module can be generated and tuned to the available resources, the implementation technology and application at hand.

The architecture should be easy to pipeline in order to balance the hardware usage with respect to the speed of the implementation technology. This requirement is necessary to guarantee a real time stereo vision system.

B. General Structure

The architecture consists of five processing blocks. The first block captures the synchronized stereo pixel stream and places it into multiple on-chip parallel memories. This stream can result from synchronized cameras, frame buffers or precaptured data streams on mass storage media. This data stream is a synchronized source of pixels coming from the two images where the first elements of the stream are the top left pixels in the left and right images and the following elements are corresponding pixels on the same line. The second block uses the information in the on-chip parallel memory to place a correctly oriented window of the image in a register. These first two blocks can be seen as more general and applicable to many (and more advanced) vision applications and are explained in more detail in [11]. The third block generates support weights for each pixel in the window using a color similarity measurement. These weights are used in block four to adapt the cost aggregation phase of the Sum of the window stored in the memory architecture. The fifth and last block calculates the minimum SAD result. This data flow can be seen on Fig. 2.

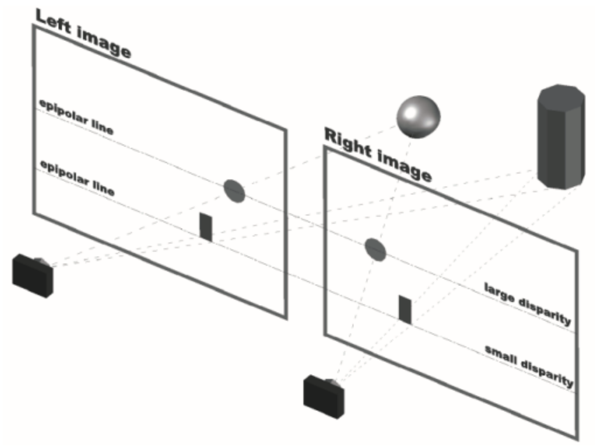


Figure 1. Stereo Vision Setup

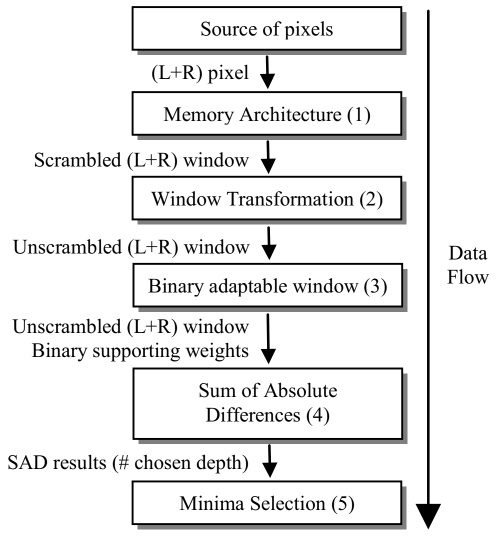


Figure 2. General Architecture

C. Memory Architecture

Due to the sequential way in which digital video data are presented, video signal processing architectures are traditionally built around line buffers. In the line buffers a number of the most recent scan lines are kept on-chip. Line buffers could be implemented as shift registers, but are currently efficiently implemented as on-chip memory blocks with dedicated addressing logic, such that they are used as FIFOs, typically one FIFO per scan line. At the outputs of the FIFOs, corresponding column pixels for the recent scan lines are accessed. These can then be stored in a shift register array with the size of the area of interest for window based video operations such as filtering, edge detection, sharpening, resampling etc.

However, the traditional scan line based FIFO architecture does not fully exploit the parallelism that is available with multiple on-chip memory blocks.

In this paper, an on-chip memory architecture (see Fig. 3) is used that allows parallel access to all pixels located in a chosen window. A window is defined here as a rectangular region of interest of the image. This region should be read out in one clock cycle for real-time operation and in order to avoid that memory access becomes a bottleneck in the data path.

On every clock cycle, exactly one on-chip memory block is written to. An Address Management Unit (AMU) is needed to keep track of which on-chip memory to write to (called the base memory) and of which address to write to (called the base address).

On every clock cycle, reading the on-chip memory contents for a chosen window is performed in parallel.

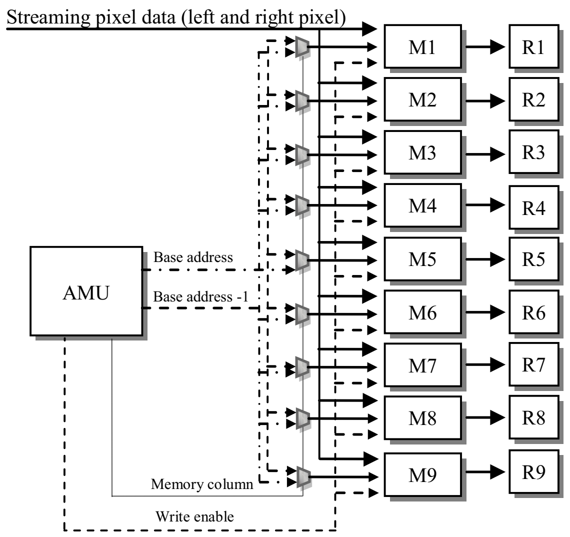


Figure 3. Parallel Memory Implementation

D. Window Transformation

The window retrieved from the memories is not immediately usable for further processing. Although all window pixels are available at the same clock cycle, they still need to be reorganized in a way that the window orientation in the memory represents the window orientation on the image. Due to the addressing order of the AMU, the pixels contained in a memory window are scrambled in comparison with the image window.

For the architecture with a fixed window cost aggregation, only a column transformation is needed for the complete window. For the architecture with the binary adaptable window cost aggregation, a column transformation is needed for the complete window and a complete transformation is needed for the center pixel.

E. Binary Adaptable Window

When using a fixed window shape, implicitly depth continuity across this window is assumed. This assumption is not correct at depth edges, where the center pixel depth is different from some (or the majority) of the surrounding pixels depths. A more conservative assumption is to assume depth continuity across pixels with similar color. The adaptive weight algorithm proposed by Yoon [4] gives a support weight to each pixel in a fixed squared window. A derivative of this method is implemented in this architecture. To save SoC resources, the support weights are chosen binary, where '0' means that this pixel will give no support to the matching window and '1' means that this pixel gives support to the matching window. There is no gradient between these two extremes.

This can be seen as a masking window that selects the pixels that will be used in the cost aggregation phase of the SAD algorithm. Only the pixels where the masking window is '1' (black) will be taken into account (see Fig. 4). Note that this masking window can arrange for any shape possible within the original window, connectiveness between pixels and segment shape are not taken into account.

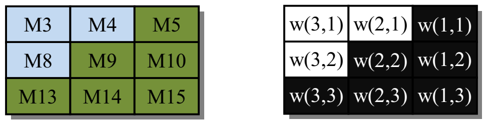


Figure 4. Window Content (left) and Resulting Window Mask (right)

For every sub window we want to match, a binary mask window needs to be generated. With the first version, this mask is generated by comparing the blue difference and red difference chroma components (in the YCrCb colour space) between each pixel in the window with the center pixel. If the combined chroma differences are larger than a certain threshold, the corresponding entry in the mask window will be '0', otherwise it will be '1' (1).

$$w = \begin{cases} 0 & if\ \Delta chroma > threshold \\ 1 & otherwise \end{cases} \tag{1}$$

The second version uses the same chroma difference combined with a distance parameter (2 and 3). This distance parameter is a pre-calculated value stored in a look-up table. It acts as a regulator to trim medium chroma matches from pixels that are located further from the center. The current implementation of this version doesn't make use of multiplications (2), it instead uses shift operations (3) in order to implement this version efficiently into hardware. This way, almost no extra resources are needed to include the distance parameter.

$$w = \begin{cases} 0 & if\ (\Delta chroma * distance\ parameter) > threshold \\ 1 & otherwise \end{cases} \tag{2}$$

$$w = \begin{cases} 0 & if\ (\Delta chroma \ll distance\ parameter) > threshold \\ 1 & otherwise \end{cases} \tag{3}$$

Figure 5 presents the distance parameter for implementation with the multiplication operator compared with the implementation with the shift operator. In the later case, the regulation possibility of the distance parameter becomes less nuanced but can be implemented very efficiently.

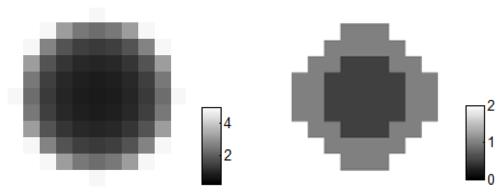


Figure 5. Distance parameter (left: for usage with the multiplication operator, right: for usage with the shift operator)

F. Sum of Absolute Differences

The Sum of Absolute Differences (SAD) calculates the differences between two selected subwindows. It is a measurement of similarity between two parts of an image. The main building block is the calculation of the absolute difference (AD). Different methods exist to calculate this. The method chosen for this architecture calculates first the difference of the two numbers, if this result is negative, the two's complement of the result is taken (Fig. 6).

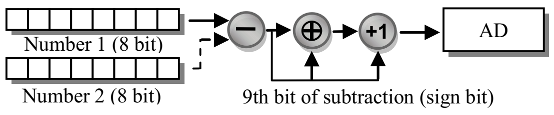


Figure 6. Absolute Difference Calculation

The summation part or cost aggregation step of the SAD uses a window and depth parallel approach to calculate the sum of absolute differences for all depths and all windows in one clock cycle.

Three main methods are integrated in this architecture. The first method is a SAD block calculation (see Fig. 7). This method calculates the SAD values immediately from the absolute differences of the pixels in the subwindow. For a 3x3 subwindow with a depth of 2; 27 AD calculations and 24 adders are needed.

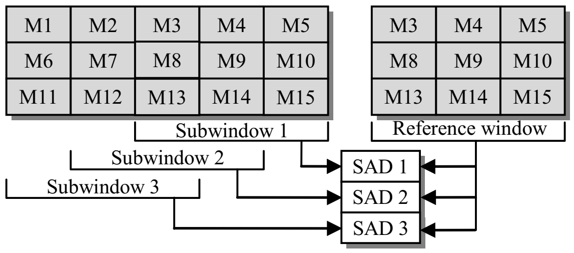


Figure 7. SAD Block Calculation

The second method calculates first the column SAD and places these results in registers (see Fig. 8). These registers are kept in buffers during the next processing round so that they can be reused during several clock periods (7, 8). Second the column SAD's from several processing rounds are summed to calculated the SAD values (4, 5, 6). For a 3x3 subwindow with a disparity range of 2; 9 AD calculations, 12 adders and 3 registers files with a size equal to the window width are needed.

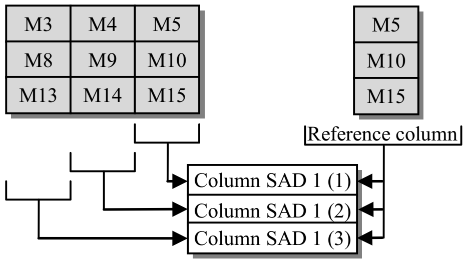


Figure 8. SAD Column Calculation

$$SAD\ 1 = \sum_{i=1}^{3} Column\ SAD\ i(1) \tag{4}$$

$$SAD\ 2 = \sum_{i=1}^{3} Column\ SAD\ i(2) \tag{5}$$

$$SAD\ 3 = \sum_{i=1}^{3} Column\ SAD\ i(3) \tag{6}$$

$$Column\_SAD\ 3 = Column\_SAD\ 2 \tag{7}$$

$$Column\_SAD\ 2 = Column\_SAD\ 1 \tag{8}$$

The third method calculates first the AD and places them directly into registers without combining them into a column SAD (see Fig. 9). These registers are also kept in buffers during the next processing round so that they can be reused during several clock periods (12, 13). Second the column SAD's from several processing rounds are summed to calculated the SAD values (9, 10, 11). For a 3x3 subwindow with a disparity range of 2; 9 AD calculations, 24 adders and 3 registers files with a size equal to the window width multiplied with the window height are needed.

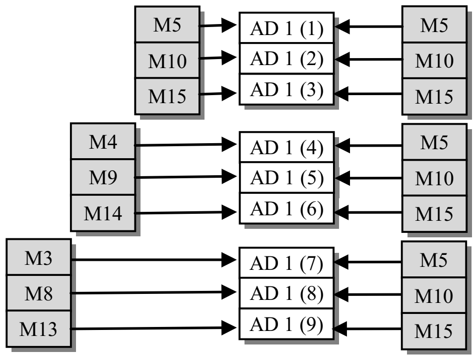


Figure 9. SAD Single Calculation

$$SAD\ 1 = \sum_{i=1}^{3} AD\ 1(i) + \sum_{i=1}^{3} AD\ 2(i) + \sum_{i=1}^{3} AD\ 3(i) \tag{9}$$

$$SAD\ 2 = \sum_{i=4}^{6} AD\ 1(i) + \sum_{i=4}^{6} AD\ 2(i) + \sum_{i=4}^{6} AD\ 3(i) \tag{10}$$

$$SAD\ 3 = \sum_{i=7}^{9} AD\ 1(i) + \sum_{i=7}^{9} AD\ 2(i) + \sum_{i=7}^{9} AD\ 3(i) \tag{11}$$

$$AD\ 3 = AD\ 2 \tag{12}$$

$$AD\ 2 = AD\ 1 \tag{13}$$

Method one and three allow access to all single AD calculations, which make them suitable to switch to a binary adaptable cost aggregation window. Equation 14, 15 and 16 show the result of switching method three.

$$SAD\ 1 = \sum_{i=1}^{3} w(1,i) * AD\ 1(i) + \sum_{i=1}^{3} w(2,i) * AD\ 2(i) + \sum_{i=1}^{3} w(3,i) * AD\ 3(i) \tag{14}$$

$$SAD\ 2 = \sum_{i=4}^{6} w(1,i) * AD\ 1(i) + \sum_{i=4}^{6} w(2,i) * AD\ 2(i) + \sum_{i=4}^{6} w(3,i) * AD\ 3(i) \tag{15}$$

$$SAD\ 3 = \sum_{i=7}^{9} w(1,i) * AD\ 1(i) + \sum_{i=7}^{9} w(2,i) * AD\ 2(i) + \sum_{i=7}^{9} w(3,i) * AD\ 3(i) \tag{16}$$

Since the weights in this architecture are '0'or '1', the multiplication in (14, 15 and 16) can be replaced by an AND operator. This will accommodate for an efficient hardware implementation.

G. Minima Selection

The minima selection is based on an iterative minima tree calculation (see Fig. 10). The SAD results are pair wise compared, while each time the lowest value is stored in a register. Afterwards, these registers are pair wise compared and stored in registers. These steps are repeated until one value remains. Storing of the intermediate results in registers makes this method interesting for pipelining.

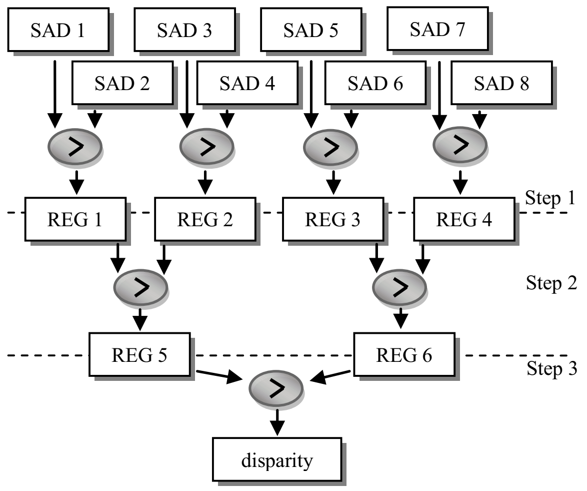


Figure 10. Iterative minima tree calculation for a maximum disparity range of eight

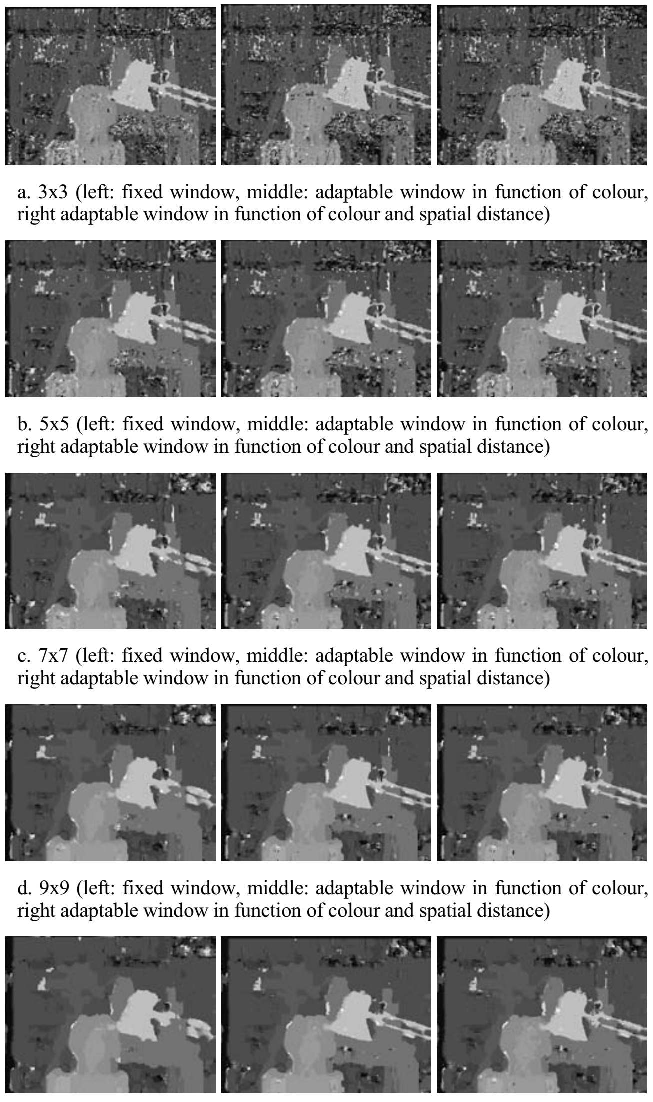
III. IMPLEMENTATION AND RESULTS

Matlab has been used to generate, out of the chosen parameters and the high level architecture, a complete stereo matching architecture for both simulation and hardware generation from a high level description. This allows an initial check of the stereo matching architecture in Matlab before implementation on the actual hardware. Using this framework, comparison between different stereo matching parameters and architectures can be rapidly performed.

The architecture and methods presented in this paper have been implemented on an FPGA system, based on an Altera Cyclone II with 68.416 logic elements and 250 memory blocks. The source of the input stream is a flash memory containing two 24 bit RGB color images and the depth information is stored on an external frame buffer that is connected to an LCD screen. Both fixed window as well as binary adaptive window SAD are compared with each other in function of their logic elements usage and the quality of the produced depth maps. The Quartus-II version 9.0 logic synthesis and fitter tool has been used for the hardware implementation.

Fig. 11 show the depth maps of the Tsukuba stereo pair [2] with a binary adaptive subwindow compared with a square subwindow. The results indicate that the quality of the resulting depth map increases when using a binary adaptive subwindow. Even with smaller window sizes, small details around the edges are noticeable improved. With larger window sizes the smoothing effect stays while preserving small details around the edges. This improvement comes not for free, the logic element usage with a binary adaptive subwindow increases linearly with the size of the subwindow (see Fig. 12). The main reason for the large resource usage difference between the two cost aggregation versions is the switch from SAD Column to SAD Single calculation.

When comparing the results of a binary window implementation using only a color metric and a binary window implementation which a color and space distance metric (see Fig. 11). The depth image becomes less smooth and the borders are better preserved. Note that by using only shift operators, almost no extra resources are needed to add the space distance metric to the color metric.



a. 3x3 (left: fixed window, middle: adaptable window in function of colour, right adaptable window in function of colour and spatial distance)

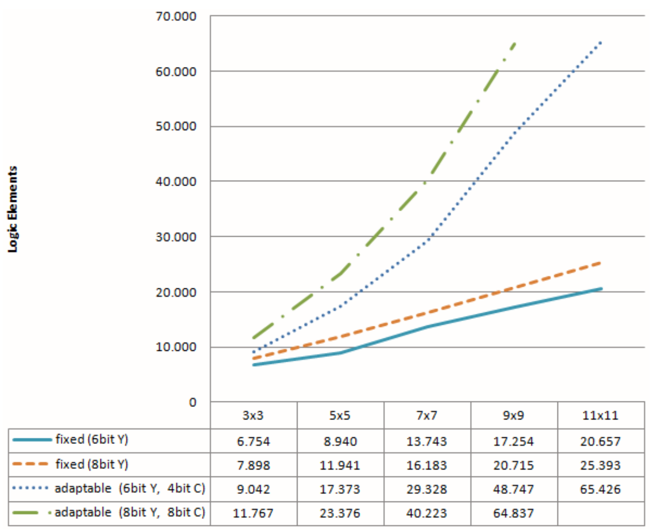
b. 5x5 (left: fixed window, middle: adaptable window in function of colour, right adaptable window in function of colour and spatial distance)

c. 7x7 (left: fixed window, middle: adaptable window in function of colour, right adaptable window in function of colour and spatial distance)

d. 9x9 (left: fixed window, middle: adaptable window in function of colour, right adaptable window in function of colour and spatial distance)

e. 11x11 (left: fixed window, middle: adaptable window in function of colour, right adaptable window in function of colour and spatial distance)

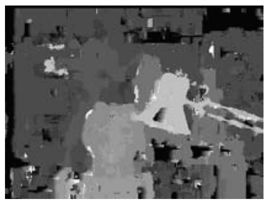
Figure 11. Depth map quality of the Tsukuba stereo pair [2] in function of window size and aggregation window type

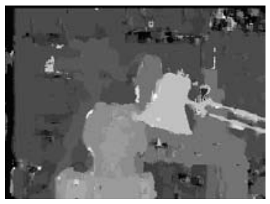


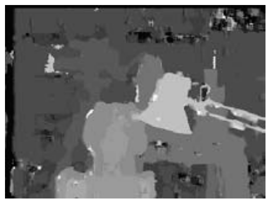
	3x3	5x5	7x7	9x9	11x11
fixed (6bit Y)	6.754	8.940	13.743	17.254	20.657
fixed (8bit Y)	7.898	11.941	16.183	20.715	25.393
adaptable (6bit Y, 4bit C)	9.042	17.373	29.328	48.747	65.426
adaptable (8bit Y, 8bit C)	11.767	23.376	40.223	64.837	

Figure 12. Logic element usage in function of window size, color resolution and aggregation window type

The color resolution is an important parameter to reduce hardware resource usage. Fig. 13 shows that a 6 bit luma component with a 4 bit chroma component is sufficient to get acceptable results while limiting resource usage. Although pixel resolution will improve the quality of the resulting depth map, hardware resource usage rises rapidly (see Fig. 12).


a. 4 bit Y, 4 bit C


b. 6 bit Y, 4 bit C


c. 6 bit Y, 6 bit C

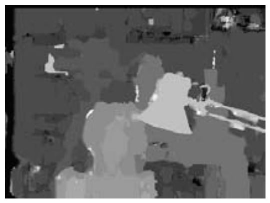

d. 8 bit Y, 8 bit C

Figure 13. Depth map quality of the Tsukuba stereo pair [2] in function of the color resolution with a binary adaptable window size of 11x11 in fucntion of colour and spatial distance

IV. Conclusions

A novel binary adaptive window cost aggregation calculation architecture is presented that allows for depth discontinuity preservation within the reference window. For each window a binary mask window is generated which selects the supporting pixels in the cost aggregation phase of the SAD algorithm. This selection is performed using color similarity and spatial distance metrics. The results of the implementation indicate that the binary adaptive window implementation improves the disparity map quality with both small as large window sizes. It is shown that this architecture can be implemented efficiently into a SoC design, however the resource usage rises rapidly with increased window size.

This architecture is chosen with particular attention given to pipelining and parallelism possibilities and is fully scalable to allow for custom SoC implementations, as well as rapid prototyping on FPGAs.

The system reported in this paper has been implemented on static image input from flash memory. Future work focuses on incorporating live video streams and optimization of the operating frequency.

References

[1] N. Lazaros, G. C. Sirakoulis, and A. Gasteratos, “Review of stereo vision algorithms: From software to hardware,” International Journal of Optomechatronics, vol. 2 (4), 2008, pp. 435-462.

[2] D. Scharstein, and R. Szeliski, “A taxonomy and evaluation of dense two-frame stereo correspondence algorithms,” International Journal of Computer Vision, vol. 47 (1), 2002, pp. 7-42.

[3] L. Wang, M. Gong, M. Gong, and R. Yang, “How far can we go with local optimization in real-time stereo matching,” Proc. Third Int. Symp. On 3D Data Processing, Visualization, and Transmission, 2006, pp. 129-136.

[4] K.J. Yoon, and I.S. Kweon, “Adaptive support-weight approach for correspondence search,” IEEE Trans. PAMI, vol. 28 (4), 2006, pp. 650-656.

[5] F. Tombari, S. Mattoccia, and L. Di Stefano, “Segmentation-based adaptive support for accurate stereo correspondence,” in Advances in Image and Video Technology, Lecture Notes in Computer Science, Berlin: Springer-Verlag, vol. 4872, 2007, pp. 427-438.

[6] R. Brockers, “Cooperative stereo matching with color-based adaptive local support,” Conference on Computer Analysis of Images and Patterns, 2009, pp. 1019-1027.

[7] C. Murphy, D. Lindquist, A. M. Rynning, T. Cecil, S. Leavitt, and M. L. Chang, “Low-cost stereo vision on an FPGA,” International Symposium on Field-Programmable Custom Computing Machines, 2007, pp. 333-334.

[8] K. Ambrosch, M. Humenberger, and W. Kubinger, “SAD-based stereo matching using FPGAs,” in Embedded Computer Vision, Advances in Pattern Recognition, K. Branislav, ed. London: Springer-Verlag, 2009, pp. 121-138.

[9] J. Yi, J. Kim, L. Li, J. Morris, G. Lee, and P. Leclercq, “Real-time three dimensional vision,” in Advances in Computer Systems Architecture, Lecture Notes in Computer Science, Berlin: Springer-Verlag, vol. 3189 2004, pp. 309-320.

[10] M. Hariyama, and M. Kameyama, “Pixel-serial and window-parallel VLSI processor for stereo matching using a variable window size,” Interdisciplinary Information Sciences, vol. 7 (2), 2001,pp. 289-297.

[11] A. Motten, and L. Claesen, “An on-chip parallel memory architecture for a stereo vision system,” unpublished.