Hindawi Publishing Corporation
VLSI Design
Volume 2012, Article ID 602737, 17 pages
doi:10.1155/2012/602737

Research Article

Hardware Design Considerations for Edge-Accelerated Stereo Correspondence Algorithms

Christos Ttofis and Theocharis Theocharides

Department of Electrical and Computer Engineering, University of Cyprus, P.O. Box 20537, 1678 Nicosia, Cyprus

Correspondence should be addressed to Christos Ttofis, ttofis.christos@ucy.ac.cy

Received 25 February 2012; Accepted 23 March 2012

Academic Editor: Muhammad Shafique

Stereo correspondence is a popular algorithm for the extraction of depth information from a pair of rectified 2D images. Hence, it has been used in many computer vision applications that require knowledge about depth. However, stereo correspondence is a computationally intensive algorithm and requires high-end hardware resources in order to achieve real-time processing speed in embedded computer vision systems. This paper presents an overview of the use of edge information as a means to accelerate hardware implementations of stereo correspondence algorithms. The presented approach restricts the stereo correspondence algorithm only to the edges of the input images rather than to all image points, thus resulting in a considerable reduction of the search space. The paper highlights the benefits of the edge-directed approach by applying it to two stereo correspondence algorithms: an SAD-based fixed-support algorithm and a more complex adaptive support weight algorithm. Furthermore, we present design considerations about the implementation of these algorithms on reconfigurable hardware and also discuss issues related to the memory structures needed, the amount of parallelism that can be exploited, the organization of the processing blocks, and so forth. The two architectures (fixed-support based versus adaptive-support weight based) are compared in terms of processing speed, disparity map accuracy, and hardware overheads, when both are implemented on a Virtex-5 FPGA platform.

1. Introduction

Depth extraction from stereoscopic images is a vital step in several emerging embedded applications, such as robot navigation, obstacle detection for autonomous vehicles, and space and avionics [1]. Depth information is extracted by solving the challenging problem of stereo correspondence, which aims at finding corresponding points (or conjugate pairs) between the input stereo images (usually referred to as reference image I_r and target image I_t). Given that the input images are rectified, the correspondence of a pixel at coordinate (x, y) can only be found at the same vertical coordinate y, and within a maximum horizontal bound, called *disparity range* D (d_m–d_M) [2]. The disparity is computed as the absolute difference between the coordinates of the corresponding pixels in I_r and I_t. The disparities of all corresponding pixels form a disparity map, which once estimated, can be used to extract the depth of the scene by using triangulation [1].

In general, stereo correspondence algorithms mostly follow four steps: (1) matching cost computation, (2) cost (support) aggregation, (3) disparity computation/optimization, and (4) disparity map refinement [2]. Moreover, they are classified into two broad categories: *global* and *local* [2]. Global algorithms can produce very accurate results but are slower and computationally more demanding compared to local algorithms, due to their iterative nature and high memory needs [3]. On the other hand, local algorithms are faster and less computationally expensive, hence suitable for the majority of embedded stereo vision applications.

Local algorithms determine the disparity of pixel p in I_r by finding the pixel of I_t that yields the lowest score of a similarity measure (see [3] for a review) computed on a support (correlation) window centered on p and on each of the d_M–d_m candidates defined by the disparity range. Early local algorithms have followed a simple approach that uses a fixed (typically square) support window during the cost aggregation step. This approach, however, is prone to errors

as it blindly aggregates pixels belonging to different disparities, resulting in incorrect disparity estimation at depth discontinuities and regions with low texture [3]. Recent local algorithms improve this basic approach by using multiple-window and adaptive-support weight (ADSW) methods [4]. The latter methods represent state-of-the-art in local stereo correspondence algorithms, as they can generate disparity maps that approach the accuracy of global algorithms [3]. These methods operate by assigning different weights to the pixels in a support window based on the spatial and color distances to the center pixel [4], or on information extracted from image segmentation [5, 6]. In this way, they aggregate only those neighboring pixels that are at the same disparity. Despite their improvements in accuracy, ADSW algorithms are generally slower than other local algorithms. A survey for different approaches can be found in [2, 3].

The real-time and low-power constraints of most embedded stereo vision applications complicate the realization of stereo correspondence algorithms. Software implementations usually struggle to meet these constraints, and as a result, some form of hardware acceleration or even a complete custom hardware implementation is preferred [3]. Hardware acceleration of stereo correspondence algorithms has been done extensively using Digital Signal Processors (DSPs) and Graphics Processing Units (GPUs) [3]. These systems, however, involve architectures that are generally not suitable for embedded applications. DSPs do not provide enough computational power to achieve parallel processing, while GPUs consume excessive power. The strong embedded requirements of such applications imply the use of dedicated hardware architectures such as ASICs and FPGAs, which can provide the necessary computational power and the energy efficiency.

The majority of dedicated hardware architectures implement fixed-support algorithms, providing adequate frame rates. However, numerous embedded vision applications require not only real-time processing speed, but also reliable depth computation. As such, there have been a few attempts recently to implement more complex algorithms in hardware such as ADSW algorithms, but these algorithms are computationally more expensive and thus the frame rate suffers, especially when high resolution images are used. It is imperative that the key to providing a successful realization of an embedded stereo vision system is the careful design of the core architecture so as to provide a good tradeoff between the processing speed and the quality of the resulting disparity map.

This paper presents an approach to the acceleration of hardware implementations of stereo correspondence algorithms for embedded stereo vision systems, by using edge information. The presented approach lies on the extraction of feature points (edges) that are used to restrict the stereo correspondence algorithm only to specific features of the input images (rather than to all image points), thus resulting in a considerable reduction of the search space. The paper presents and analyzes design issues and considerations associated with the edge-directed approach when it is applied to different hardware implementations of stereo correspondence algorithms, including both fixed-support and even more complex ADSW algorithms.

The paper extends our initial work on a hardware implementation of an edge-directed fixed-support stereo correspondence algorithm presented in [7]. In particular, the extended work generalizes the idea of using edge information as part of the stereo correspondence process and presents design principles and considerations that arise when integrating edge information on the architecture of an ADSW algorithm as well. The paper also explores the effects of different edge detection algorithms on the ADSW-based architecture, showing the impact of each algorithm on the quality of the disparity maps produced by that architecture. Furthermore, the paper compares the two architectures (fixed-support based versus ADSW-based) in terms of processing speed, disparity map accuracy and hardware overheads, when both are implemented on a Virtex-5 FPGA platform. The experimental results indicate that the presented edge-directed approach for accelerating hardware implementations of stereo correspondence algorithms exhibits high potential for applications with hard real-time constraints, as both architectures achieve remarkable speedups relative to existing hardware architectures that implement equally complex algorithms. The architecture based on the fixed-support algorithm can meet the real-time requirements of such applications, even for high-resolution images. Moreover, the combination of the edge-directed approach with the ADSW algorithm enables the realization of accurate disparity computation systems that can also satisfy the real-time requirements of many embedded vision applications such as pedestrian and obstacle detection.

The rest of this paper is organized as follows. Section 2 discusses related work and Section 3 presents the edge-based stereo correspondence process. Section 4 presents the proposed edge-directed disparity map computation architectures. Section 5 presents the experimental framework and results, and Section 6 concludes the paper, discussing future work.

2. Related Work

There exists a large number of implementations in the literature that solve the stereo correspondence algorithm. Several existing works feature algorithms running on general purpose processors, as well as clusters and multiprocessor systems [8–10]. However, most of these implementations require high-end computational equipment in order to compute the disparity map in real-time, especially as the image resolution increases. As a result, the real-time processing of those implementations is limited only to small-sized images (smaller than VGA). Alternatively, specialized hardware, such as Intel MMX [11], Graphics Processing Units (GPUs) [12, 13], and Digital Signal Processors [14–16], has been used to accelerate the disparity map computation. [11] attempts to improve the accuracy of the simple SAD correlation technique by using a multiple window approach that decreases errors at object boundaries. This however requires a lot of computational resources. [12, 13] present GPU-based implementations that achieve high frame rates

that address the computational needs of stereo vision algorithms; however, they are not currently suitable for embedded and mobile applications due to their power demands. Approaches implemented on high-end fixed point DSPs [14–16] consume lower power; however, they do not provide the parallelism of FPGAs and ASICs, thus are less suitable for stereo vision applications with hard real-time requirements. There exist also attempts to implement stereo vision algorithms on the Cell processor [17, 18], but are subject to restrictions imposed by the Cell platform, mainly due to the limited memory of the Synergistic Processing Elements (SPEs).

The last decade features an emergence in dedicated hardware architectures, as a means to address the aforementioned constraints found in software and specialized hardware approaches. Dedicated hardware architectures, implemented on ASICs and FPGAs, can exploit the parallelism inherent in the stereo correspondence process and optimize memory access patterns to provide fast computation. Most of dedicated hardware implementations have been implemented on FPGA platforms, taking advantage of the reconfiguration offered by these platforms, in order to exploit the intrinsic parallelism of the stereo correspondence algorithm. A stereo depth measurement system on an FPGA is introduced in [19]. It generates disparities on 512×480 images at 30 fps, implementing a window-based correlation search technique. Another system presented in [20] yields 20 fps on 640×480 image sizes. However, the memory access pattern utilized does not provide scalability and performance optimization, limiting the overall performance of the system. In [21], a survey of some FPGA implementations, [22–24], is presented, and the survey highlights the common use of the SAD similarity measure, a relatively simple technique suitable for hardware implementations. Furthermore, [22–24] make extensive use of parallelism and pipelining in order to achieve real-time performance. [24] also raises the impact of the image size on the performance of the disparity map computation; it claims 5063 fps using very small (64×64) images. The frame rate, however, decreases exponentially as the image size increases.

Other dedicated hardware architectures suitable for real-time disparity map computation are given in [25–31]. These works attempt to implement high-performance stereo correspondence algorithms that fail to achieve real-time performance in software. The FPGA-based architectures presented in [25, 26] implement local fixed-support algorithms using the SAD similarity measure and compute intermediate-sized disparity maps at a rate of 768 and 600 fps, respectively. The system in [27] is based on a phase-based computational model, as an alternative to feature correspondence and correlation techniques. That work exploits the parallel computing resources of FPGA devices to produce dense disparity maps for high-resolution images at 52 fps. The supported disparity range, however, is not practical for embedded stereo vision systems, which traditionally utilize much larger disparity ranges, especially when high-resolution images are used. Another implementation of a computationally complex disparity algorithm that is based on locally weighted phase correlation is presented in [28] and utilizes 4 FPGAs to produce dense disparity maps of size 256×360 at the rate of 30 fps. A more recent FPGA implementation of a real-time stereo vision system is presented in [29]. That system generates dense disparity maps based on the Census transform. Lastly, the hardware implementation presented in [31] performs a modified version of the Census transform in both the intensity and the gradient images, in combination with the SAD correlation metric (SAD-IGMCT algorithm), achieving 60 fps on 750×400 images.

The majority of the aforementioned implementations adopt local fixed-support algorithms to achieve real-time performance, as these algorithms can be greatly benefited by the use of parallel and straightforward structures, which are key factors available in dedicated hardware implementations. ADSW algorithms, which traditionally achieve better disparity map accuracy, have been rarely implemented on dedicated hardware architectures. As mentioned above, these algorithms are computationally more expensive compared to fixed-support algorithms and hence require complex and hardware-unfriendly operations. To the best of our knowledge, only the work in [30] implements a local ADSW stereo correspondence algorithm. That work proposes an accurate, hardware-friendly disparity estimation algorithm called mini-census ADSW, and its corresponding real-time VLSI architecture that achieves 42 fps on 352×288 image sizes. There has been a good effort in [30] to reduce the computational complexity of the ADSW and achieve real-time performance. However, that work achieves real-time performance for relatively small-sized images.

This work investigates the integration of edge-directed information into hardware architectures of stereo correspondence algorithms, as a means to restrict the stereo correspondence process only to the specific features (edges) of the input images (rather than to all image points), thus reducing the search space considerably. The proposed approach targets real-time embedded vision applications and can benefit both fixed-support and ADSW algorithms. When applied to fixed-support algorithms, it leads to very efficient implementations that can effectively address the hard real-time constraints of such applications even when using high resolution images and large disparity ranges. When applied to ADSW algorithms, the proposed approach provides a good tradeoff between accuracy of the disparity maps and processing speed; the reduction of the search space leads to real-time implementations for image sizes which are larger than the ones used in [30]. A comparison between the proposed work and other existing disparity map computation systems is given in Table 7.

3. Edge-Based Stereo Correspondence Process Overview

3.1. Summary of the Algorithm. In this section we present the overall flow of the edge-based stereo correspondence process. The description is generic and applies to both fixed-support and adaptive-support weight (ADSW) algorithms. In general, there are three major tasks associated with an edge-based stereo correspondence algorithm: edge detection, stereo correspondence, and interpolation. Figure 1 illustrates

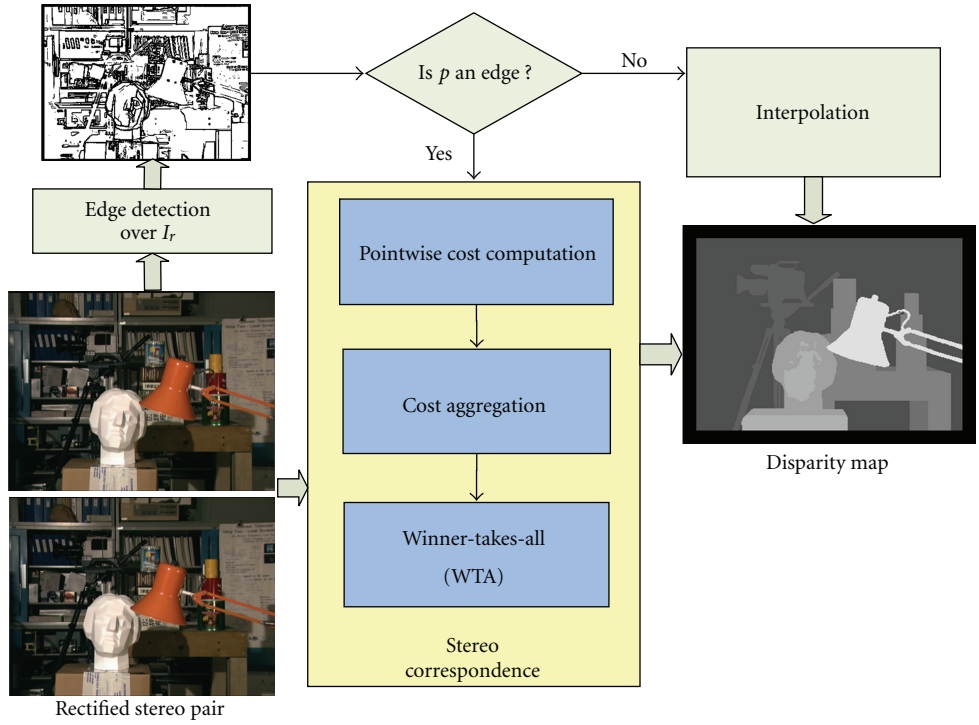

Figure 1: Steps of the edge-based stereo correspondence process.

the algorithm's steps towards computing a disparity value for each pixel p in I_r. The algorithm starts by determining whether a pixel p corresponds to an edge or not. For each pixel p corresponding to an edge, the algorithm extracts an $m \times m$ correlation (or support) window W_r centered on p in I_r, and an $m \times m$ support window W_t centered on q in I_t (the coordinate of q is $(x + d, y)$, where d lies in the range d_m to d_M). The algorithm then computes a pointwise score for any pixel $p_i \in W_r$ corresponding to $q_i \in W_t$ and aggregates the scores spatially over the support windows.

Fixed-support and ADSW stereo correspondence algorithms differ in the way they perform the pointwise score computation and aggregation steps. The fixed-support algorithm computes a pointwise score for any pixel $p_i \in W_r$ corresponding to $q_i \in W_t$ as the absolute different (AD) of p_i and q_i. After the computation of the AD values, the final aggregated cost is computed by summing all pointwise scores as

$$C_{x+d,y} = \sum_{p_i \in W_r, q_i \in W_t} \mathrm{AD}(p_i, q_i), \quad d_m \le d \le d_M. \tag{1}$$

Equation (1) holds only for the case of a fixed-support algorithm. In the case of an ADSW algorithm, each pointwise score is computed by multiplying the absolute difference of p_i and q_i by a weight coefficient $w'_r(p_i, p_c)$ and a weight coefficient $w'_t(q_i, q_c)$. These weight coefficients are assigned to each pixel in a support window, based on the pixel's spatial distance, as well as on its distance in the CIELAB color space with regard to the central pixel (p_c or q_c) in the window. The final aggregated cost is computed by summing up all the weighted pointwise scores, and normalizing by the weights sum as in (2), where, d_p and d_c are the Euclidean distance between two coordinate pairs and two triplets in the CIELAB color space, respectively, and γ_p and γ_c are two parameters of the algorithm

$$C_{x+d,y} = \frac{\sum_{p_i \in W_r, q_i \in W_t} w'_r(p_c, p_i) \cdot w'_t(q_c, q_i) \cdot \mathrm{AD}(p_i, q_i)}{\sum_{p_i \in W_r, q_i \in W_t} w'_r(p_c, p_i) \cdot w'_t(q_c, q_i)}, \quad d_m \le d \le d_M,$$
$$\text{where } w'_{r,t} = \exp\left(-\frac{d_p(p_i, p_c)}{\gamma_p} - \frac{d_c(I_{r,t}(p_i), I_{r,t}(p_c))}{\gamma_c}\right). \tag{2}$$

Both the fixed-support and ADSW algorithms compute an aggregated cost for all disparity levels in the range $[d_m\ d_M]$, and the best disparity for the pixel p is found by locating the disparity with the minimum aggregated cost through a winner-takes-all (WTA) approach. In the case where pixel p does not correspond to an edge, the disparity is obtained by an interpolation method.

3.2. Discussion. The edge-based stereo correspondence algorithm described above applies an edge detection process over the input image I_r prior to performing the correlation step. Edge detection returns locations in the image that indicate the presence of an edge [1]. These locations, described by the *edge points*, determine the outline (i.e., perimeter) of an object and distinguish it from the background and other objects [1, 32]. The correlation window W_r in I_r is moved only to the edges (and not to each possible pixel) along the working scanline, resulting in a considerable reduction in

the search space. As a result, the correlation step computes only a disparity value for the pixels that correspond to an edge, while the disparity values of the remaining pixels are computed using interpolation, which however is considered computationally less expensive compared to stereo correspondence.

Another consideration of the edge-based stereo correspondence process is whether it uses the edges only as directive points or as matching primitives as well (instead of using pixel intensities). Since edges are encoded using only one bit per pixel, rather than 8 bits (as used in grayscale images), matching edges instead of pixel intensities reduce the computational space and complexity of the search and match operations, as well as the data path requirements. However, this can be applied only to the fixed-support algorithm. Computing pointwise scores using binary data returns only zeros or ones, and this would invalidate the weights in the ADSW algorithm.

4. Hardware Realization of Edge-Accelerated Disparity Map Computation Architectures

This section presents the hardware architectures of two different edge-based stereo correspondence algorithms: an SAD-based fixed-support algorithm and an ADSW-based algorithm. We present design principles and considerations about the implementation of these algorithms on hardware, and discuss issues related to the memory structures needed, the organization of the processing blocks, the parallelism exploited, and so forth.

4.1. Disparity Map Computation Hardware Architecture Based on the Fixed-Support Algorithm

4.1.1. Design Issues and Requirements. This architecture implements an SAD-based, fixed-support algorithm and uses edges both as directive points and as matching primitives. This requires that the architecture should be able to perform edge detection on both input images. Moreover, the use of binary data during the matching process reduces the computation of the SAD values (cost computation step) to a hamming distance operation that can be directly implemented in hardware using only addition and 1-bit subtraction operations. The simplicity of the logic required during the cost computation step enables the parallel design of the architecture, so that to target a large number of search and match operations performed in a single cycle. However, the amount of parallelism that can be extracted depends on the ability of the edge detector to generate edge points (as the matching process is performed using edges). In this work we integrate the Sobel detector mainly due to its simple hardware structure, which can be easily parallelized to provide more than one edge points per single cycle. While the use of edge information reduces the search space and simplifies the logic required, the hardware implementation of the algorithm is not straightforward, but the overall design poses significant challenges in terms of the memory structures used in order to account for the irregular rates of data due to the edge only computation. The architecture requires a clever arrangement of the on-chip memory in order to be able to process multiple support windows centered at the edge points, while skipping the nonedge points found between successive edges (every clock cycle).

4.1.2. General Structure. The block diagram of the architecture is shown in Figure 2 and consists of two major hardware units: the Edge Detection Unit (EDU) and the Disparity Computation Unit (DCU). It also consists of a Memory Controller and Control Unit (MCCU) that optimizes memory access based on the algorithm requirements and coordinates data transfers and handshakes between the EDU and the DCU. The EDU converts the rectified input image pair (grayscale images) into a pair of binary images (black and white) that characterize only the edges of the initial images. The black and white images are then fed into the DCU, which performs correlation with the objective to compute the disparity map of the input image pair. While the disparity maps generated by the DCU are sparse (disparity estimates are provided only for points corresponding to edges), the system can generate dense disparity maps as well, by integrating interpolation methods as part of the MCCU. In this work, we use the simple and fast nearest neighbor interpolation method, as our emphasis has been on performance in embedded scenarios.

The EDU and the DCU, which communicate through the use of internal memory (FIFO queues), are pipelined, and thus operate concurrently. They are also provided with scanline buffers, which temporarily store the pixels needed to perform convolution (in the case of the EDU), or correlation (in the case of the DCU). This reduces the clock cycles required to load image data from the input port, by exploiting the fact that working windows moved over the image use overlapping pixels. The scanline buffers are organized into FIFO structures and their size depends on the size of the working window (3×3 for the EDU, $m \times m$ for the DCU) and the width of the image, N. The delay to fill the scanline buffers is proportional to the I/O bandwidth.

4.1.3. EDU Architecture Overview. The Edge Detection Unit (EDU) integrated to the system implements the Sobel edge detector, which performs a 2D spatial gradient measurement on the input grayscale images using a pair of 3×3 convolution masks. The masks hold data values between -2 and 2; thus the overall convolution can be implemented in hardware using shifters instead of multipliers. We integrated this detector mainly due to its simple hardware structure, which can be easily parallelized to provide more than one edge points per single cycle. This is important, as the ability of the DCU to perform parallel computations depends on the ability of the ECU to provide multiple edge points per cycle. The architecture of the EDU is shown in Figure 2(a). The EDU employs hardware features, such as parallelism and pipelining, in an effort to parallelize the repetitive calculations involved in the Sobel operation, and uses optimized memory structures in order to reduce the memory reading redundancy. The detector architecture consists of an I/O

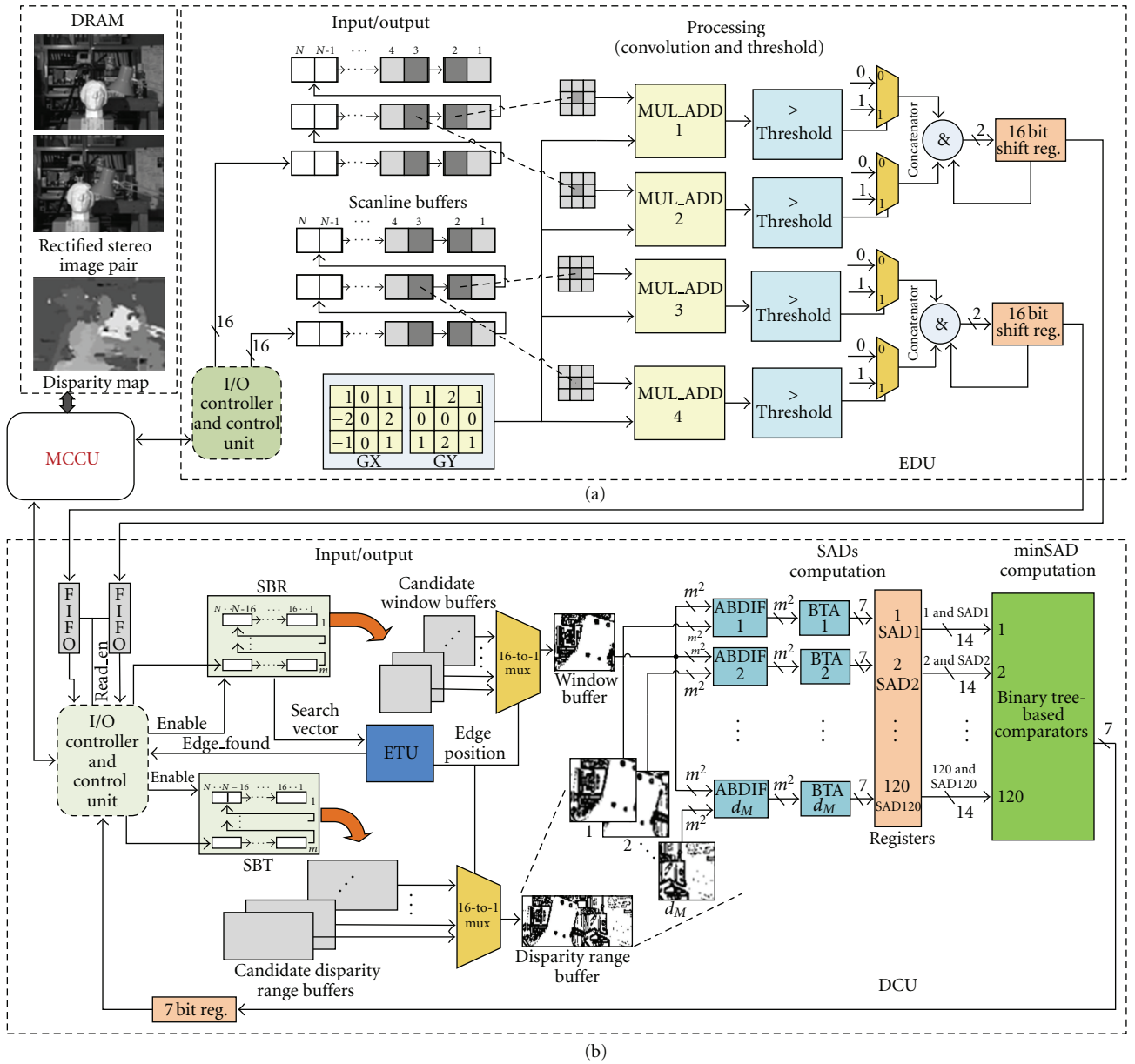

Figure 2: Block diagram of the edge-accelerated hardware architecture that implements the fixed-support algorithm. (a) Edge Detection Unit, (b) Disparity Computation Unit.

controller that reads/writes pixel data from/to the I/O port, on-chip memory for storing pixel data (scanline buffers) and the convolution mask values, a series of convolution units (MUL_ADD units), as well as comparators. The architecture is pipelined into 2 stages: *INPUT/OUTPUT* and *PROCESSING*. In the *INPUT/OUTPUT* stage the I/O controller fetches 4 pixels from the I/O port (2 pixels per image), in a row-wise fashion, and forwards them to the input ports of the scanline buffers (16 bits to each buffer), which have a FIFO structure. During the *PROCESSING* stage, the scanline buffers produce four successive 3×3 windows per cycle, which are convoluted with the 3×3 Sobel masks by the MUL_ADD units. This pipeline stage returns 1-bit pixel intensity values, which are concatenated into two 16-bit registers. The values from the 16-bit registers are forwarded to the output port (FIFO queues of the DCU) once every 8 clock cycles.

4.1.4. DCU Architecture Overview. The DCU has been designed in such a way as to calculate sparse disparity maps covering disparity ranges up to 120 pixels and correlation window sizes from 3×3 to 11×11. The architecture of the DCU is shown in Figure 2(b) and involves three major steps: *INPUT/OUTPUT*, *SADs_COMPUTATION*, and MIN*SAD_COMPUTATION*. The architecture consists of an I/O controller that reads/writes data from/to the I/O port, on-chip memory (FIFO queues, scanline buffers and window buffers) for temporarily storing the edge data, and a unit that searches and finds the position of the edge to be processed

(edge tracking unit (ETU)). It also consists of a collection of adders and subtractors to compute the SAD values (1) for all disparity levels, a unit to compute the minimum SAD value, multiplexers, and intermediate pipeline registers. Apart from reading/writing data from/to the I/O port, the I/O controller also acts as a control unit, coordinating memory accesses from the on-chip memory, as well as data transfers and handshakes between the components of the DCU.

The DCU unit was designed with emphasis on parallelism, targeting a large number of search and match operations performed in a single clock cycle. This is facilitated by the simplicity of the adders and subtractors used (due to the use of binary data), and by the organization of the adders and comparators in tree structures. Due to its pipelined and parallel structure, the DCU presents good scalability in terms of correlation window size and disparity range. Particularly, it can compute the SAD values for all possible positions of the shifting window with a maximum size of 11×11 in two clock cycles, and the minimum SAD value for a maximum of 120 disparity levels in three cycles. However, once the pipeline fills up and the FIFO queues are not empty, the DCU can provide a disparity value at the output every clock cycle.

4.1.5. DCU Memory Architecture and Edge Tracking Process. To keep a constant flow of data in the pipeline, the DCU must be able to locate one edge in the reference image every clock cycle, while discarding the non-edge points found between successive edges. At the same time, the DCU must have parallel access to the $m \times m$ window surrounding the edge found in the reference image, as well as to the corresponding d_M windows from the target image. For these reasons, each scanline buffer used in the DCU consists of a series of 16-bit registers and can store m scanlines from an input image. We avoid using 1-bit registers in order to facilitate more parallelism and to make the process of discarding the non-edge points fast. The 16-bit registers are organized into FIFO structures and allow parallel access to their elements. Specifically, the scanline buffers for the reference image (SBR) output 16 successive $m \times m$ windows (stored in the candidate window buffers), while the scanline buffers for the target image (SBT) output 16 successive $m \times (m + d_M)$ windows (stored in the candidate disparity range buffers). The SBR also outputs a 16-bit vector (search vector) from positions $1 + w$ to $16 + w$ of the $(w + 1)$th scanline, where $w = (m-1)/2$. The search vector is being searched for potential edge points by the edge tracking unit (ETU), which is the connection point between the *INPUT/OUTPUT* stage and the remaining stages. The ETU works by locating an edge and its corresponding position in the 16-bit search vector every clock cycle. The positions of the edges found during the searching process are used to select the window and disparity range buffers (among the 16 candidates) corresponding to the edge points found; the selected buffers become the input of the next pipeline stage. It must be noted that the ETU requires from 1 cycle (in the best case) to 16 cycles (in the worst) to locate all edges in the search vector. During this period, the *edge_found* signal is set to 1 and the scanline buffers are disabled, so that the content of the candidate window and candidate disparity range buffers remains constant. When all edges in the search vector are located, the ETU sets the *edge_found* signal to 0. This informs the I/O controller to fetch new edges from the input FIFO queues and to shift the scanline buffers to the right.

4.1.6. DCU Pipeline Stages Description. The major pipeline stages of the DCU are described below.

(i) *INPUT/OUTPUT.* This pipeline stage fetches pixel data from the I/O port, executes the edge tracking process, and selects the windows corresponding to the edges found. The data fetched from the input port (the two 16-bit vectors produced by the EDU) is stored into FIFO queues. The I/O controller reads data from the input FIFO queues (if they are not empty) and forwards data to the scanline buffers (16-bits to each scanline buffer), until the first m scanlines from both images are stored into the scanline buffers. After the scanline buffers are filled, the I/O controller reads new pixel data from the queues only if the *edge_found* signal is set to 0 by the ETU. While the *edge_found* signal is set to 1, the scanline buffers are disabled, and during this period the edge tracking process described above is performed. If there is new data available at the input during this period, this data is written to the input queues. Furthermore, during this pipeline stage, the I/O controller writes the disparity value computed in the previous cycle to the output port.

(ii) *SADs_COMPUTATION.* The SAD values for all disparity levels are computed during this pipeline stage. The stage consists of d_M absolute difference (ABDIF) units, which compute the absolute difference between the $m \times m$ correlation window (stored in the Window Buffer) and the d_M $m \times m$ windows (stored in the disparity range buffer). Each ABDIF unit receives as input two m^2-bit vectors, whose elements are the edge points of the correlation windows, and consists of m^2 1-bit subtractors that compute the absolute difference of the edge points. The output of each ABDIF unit is an m^2-bit vector, which is next added bitwise using binary tree adders (BTA). Given the 11×11 maximum supported correlation window size, and 1-bit pixel intensities, the maximum value of the addition operation cannot be greater than 121. As such, the outputs of the BTA units are 7-bit values.

(iii) *MINSAD_COMPUTATION.* The SAD values for all disparity levels in the range $[1 : d_M]$ are compared with each other in order to compute the minimum value and its disparity. The comparison is carried out by a collection of 7-bit comparators and registers, arranged in tree structure to reduce the delay of the longest path. As stated previously, this stage was further divided into 3 pipeline stages in order to meet the targeted operating frequency (100 MHz). Figure 3 shows the circuit that computes the minimum SAD value and its disparity. Each minSAD unit receives as input two 14-bit vectors, each of which is a concatenation of an SAD value (7 bits) and its corresponding disparity (7 bits—up to 120 disparity levels). The minSAD unit compares the two SAD values and outputs the minimum of them along with

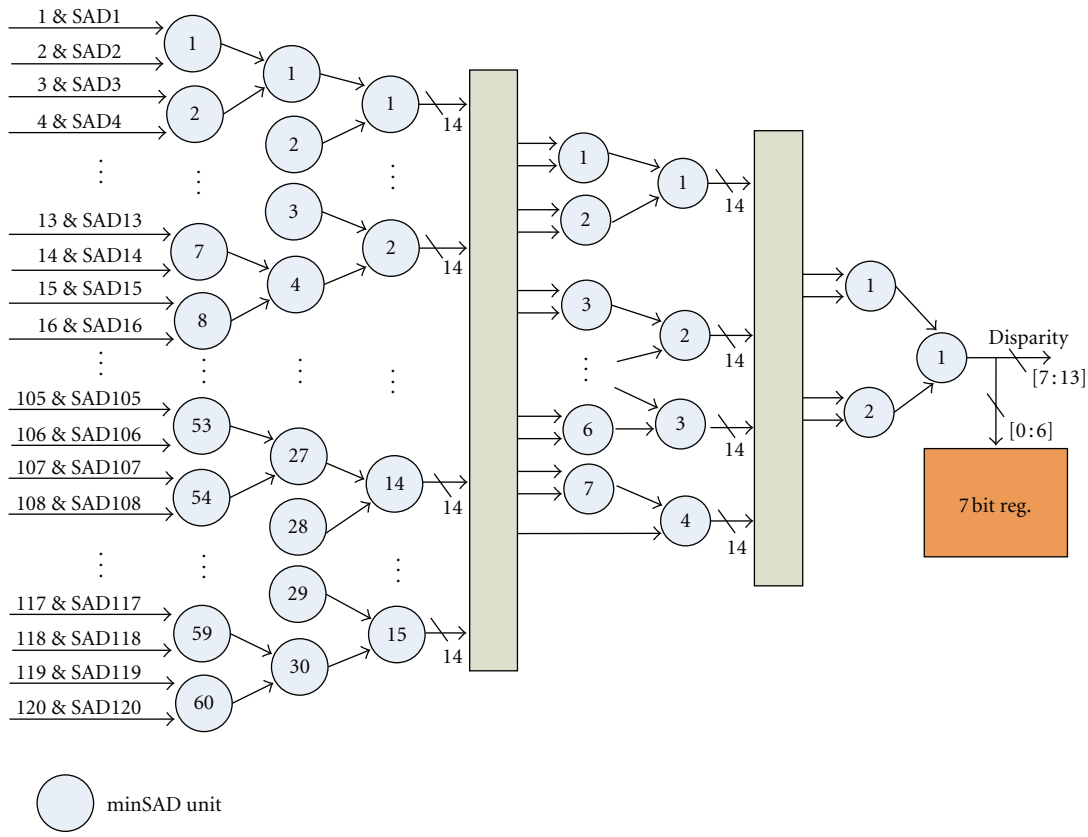

Figure 3: Minimum SAD value computation.

its disparity. The entire circuit for computing the minimum SAD value and its disparity consists of multiple minSAD units, arranged in a structure of a binary tree of $\log_2(d_M)$ levels.

4.2. Disparity Map Computation Hardware Architecture Based on the ADSW Algorithm

4.2.1. Design Issues and Requirements. This architecture implements an ADSW algorithm, which is computationally more expensive compared to the fixed-support algorithm both in terms of calculations and memory requirements. This is attributed in a significant way to the complex equation involved in the algorithm (2). In this work, we adopted some hardware optimization techniques that aim to simplify the algorithm and make it hardware-friendly and suitable for embedded constraints. We first eliminate the use of the spatial distance during the weight computation step based on our observation that it affects the accuracy of the disparity maps slightly. We also use YUV instead of CIELAB color representation during the computation of the weight coefficients. This allows the use of unsigned integers instead of signed floating-point integers, which are complex and hardware-unfriendly. Furthermore, the computation of the color distance between two YUV triplets is performed using Manhattan rather than Euclidean distance. In this way, the square and square root operations are replaced by simple absolute difference and addition operations. In addition, the $\exp(-x)$ function is approximated by the $\lfloor 2^{8-x} \rfloor$ function, which assigns a maximum weight of 256 if the color distance is zero and a weight of 0 if the color distance is greater than 8. This function simplifies the circuits that implement the multiplication of the weight coefficients with the pointwise scores, as multiplications are reduced to left shift operations. The cost function is further simplified by setting γ_c to a power of 2 (32 in our case). This converts the division to a right shift operation. Lastly, the denominator of (2) is approximated by the nearest power of 2 during the cost aggregation step, allowing the division to be replaced by a right shift operation.

4.2.2. General Structure. The architecture implementing the ADSW algorithm consists of two major pipeline stages: the *Input Stage* (IS) and the *Calculation Stage* (CS). The IS consists of a memory controller, 2 RGB to YUV color space converters (rgb2yuv) and 2 RGB to grayscale converters (rgb2gray). The memory controller fetches the RGB color values corresponding to the support windows W_r and W_t in a column-wise fashion (1 pixel value per input image every clock cycle) from the external memory. Those values are then converted to grayscale by the two rgb2gray units, and to their corresponding 8-bit YUV representation by the rgb2yuv units. The image values computed by the IS are temporarily stored into on-chip buffers (memory arrangements (MAs)),

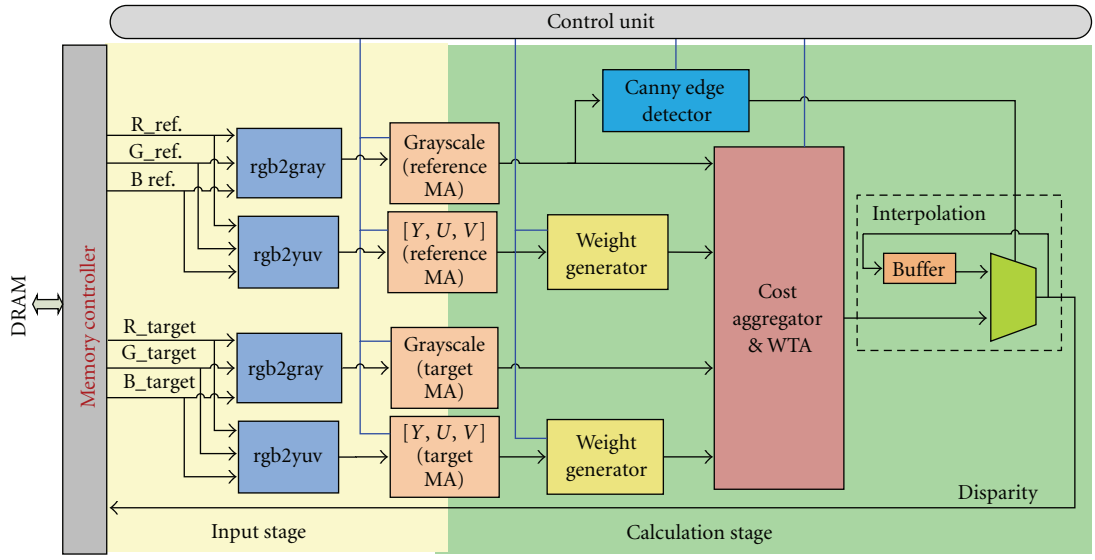

Figure 4: The architecture of the ADSW algorithm.

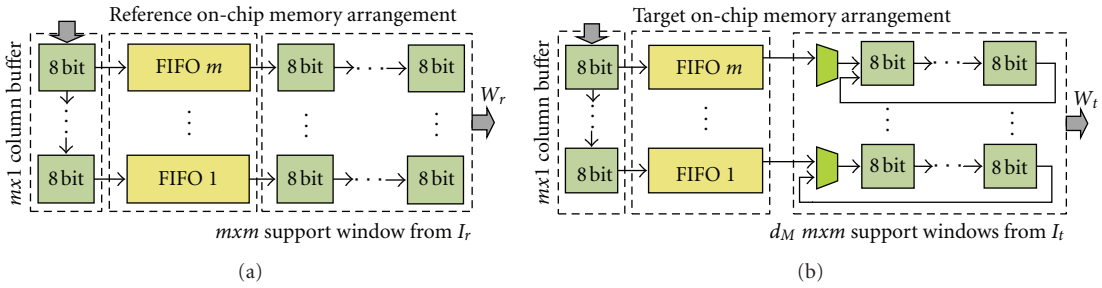

Figure 5: On-chip memory arrangements for the reference and target image data.

ensuring that there is always sufficient data for the CS, which is responsible for the calculation of the disparities. The overall system architecture also consists of a control unit that coordinates data transfers and handshakes between the different system units. Figure 4 shows a block diagram of the architecture and the data flow between units.

4.2.3. Memory Architecture. The IS and CS stages communicate through the use of on-chip buffers (memory arrangements (MAs)), which temporarily store the pixels required to perform correlation between W_r in I_r and the d_M candidate support windows W_t in I_t. The system is provided with 4 MAs per input image, which store the *Y, U, V* color values and the grayscale values. Figures 5(a) and 5(b) show the architectures of the on-chip MAs for the reference and target image, respectively. Both MAs consist of a column buffer, a series of FIFO queues, and window buffers. The column buffer consists of m 8-bit registers that store the pixels of an entire column of a support window. It receives one pixel per clock cycle and outputs a column every m cycles. The output column is stored in a series of m FIFO queues (1 pixel per queue), which are used to allow the memory controller to continuously fetch data from the external memory to the on-chip MAs (given that there is free space in the queues) irrespective of the data consumption rate of the CS. The CS consumes data at irregular rates; it consumes a column in d_M cycles if p is an edge and in a single cycle if p is not an edge.

The data from the queues is forwarded to the window buffers, which form the inputs of the CS. The window buffer of the reference MA consists of $m \times m$ 8-bit registers, while the window buffer of the target MA consists of $m \cdot (m + d_M - 1)$ registers (d_m is set to 0). The use of registers allows parallel access to the window buffers; after an initial delay of $m \cdot (m + d_M - 1)$ cycles per scanline (dominated by the cycles needed to fill in the window buffer of the target MA), both on-chip MAs can provide an $m \times m$ window per cycle. The window buffer of the target MA is organized in a cyclic structure and is provided with a series of multiplexers at its input, which determine whether the input comes from the FIFO queues or from the rightmost registers of the window buffer. This structure is adopted to enable data reuse.

4.2.4. Calculation Stage (CS). The CS consists of an Edge Detection Unit, two units for the generation of the weight coefficients (weight generators), a unit that computes the aggregated costs and selects the disparity with the minimum cost, and a unit responsible for the nearest neighbor (NN)

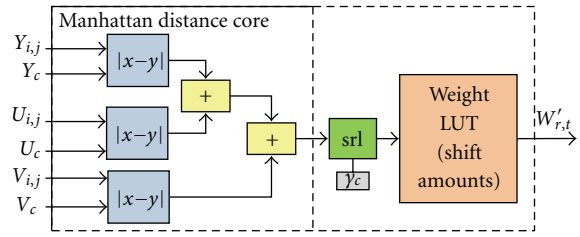

Figure 6: Circuit that computes the weight of a single pixel.

interpolation step. The Edge Detection Unit implements the Canny operator, which performs image smoothing using Gaussian convolution, vertical and horizontal gradient and angle calculation, nonmaximum suppression and thresholding. The detector employs hardware features such as parallelism and pipelining in order to provide an edge point every single clock cycle.

The weight generator computes the weight coefficients $w'_{r,t}$ for a support window $W_{r,t}$ in parallel, based on the YUV color values fetched from the $[Y,U,V]$ MAs, and by utilizing m^2 instances of the circuit shown in Figure 6. That circuit consists of a Manhattan distance core and a weight table (LUT). Since the multiplication of the pointwise scores by the weight coefficients (cost aggregation step) is performed using shifters instead of multipliers, each location x of the LUT stores the shift amount corresponding to the weight coefficient $\lfloor 2^{8-x} \rfloor$. This shift amount is equal to the binary logarithm of $\lfloor 2^{8-x} \rfloor$, except from values of x greater than 8, for which a binary logarithm does not exist. In that special case, the corresponding entries in the LUT are set to a number, which is large enough so that the result of a shift operation by that number is equal to zero. The weight coefficient $w'_{r,t}(i, j)$ is looked up in the LUT using the color distance generated by the Manhattan distance core as index.

The architecture of the cost aggregator and WTA unit is shown in Figure 7. The unit has been design in a parallel manner and utilizes m^2 absolute different circuits that compute the pointwise scores between corresponding pixels in W_r and W_t. Those scores are then shifted by the shift amounts corresponding to the weight coefficients w'_r and w'_t using a series of left shifters (equivalent to multiplying the scores by w'_r and w'_t). The final aggregated cost is computed by summing the outputs of the left shifters using a tree adder, and then normalizing (dividing) it by the weights sum, which, before being used for division, is rounded to the nearest power of 2 by using tree comparators. This enables a cost-effective implementation of the division using a right shifter. Finally, the WTA unit is responsible for selecting the disparity with the minimum cost.

5. Experimental Results

5.1. Experimental Platform and Methodology. The architectures presented in this paper have been implemented on the ML505 Evaluation Platform, which features a Virtex-5 LX110T FPGA. The basic features of the edge-directed disparity map computation architectures are listed in Table 1, while more details are presented in the following subsections. We evaluated the architectures using rectified synthetic and real-world data, initially stored in the compact flash memory card. The synthetic data includes stereo images from the Middlebury database [2] and the pedestrian data-set in [33], and the real-world data includes stereo images taken in the lab. The images were loaded into the on-board DRAM using the Microblaze soft-processor and were used as input to the systems shown in Figures 2 and 4, respectively. The resulting disparity maps were directed to a TFT monitor. The evaluation results of the synthetic images are shown in Figure 8, while the evaluation results of the real-world images are shown in Figure 9. The architectures were compared in terms of processing speed, disparity map accuracy, and FPGA resource utilization. They were also compared against their equivalent hardware systems that do not integrate the edge detectors in order to emphasize on the benefits of the edge-based approach adopted by the architectures.

Table 1: Features of the FPGA Prototypes.

Feature	Fixed-support-based architecture	ADSW-based architecture
Adopted algorithm	SAD-based Fixed-support algorithm	Adaptive-support weight algorithm
Image size	1280 × 1024 pixels (8-bit grayscale)	800 × 600 pixels (24-bit RGB color images)
Disparity range	120	64
Support window size	11 × 11	11 × 11
Device	Virtex-5 LX110T FPGA	Virtex-5 LX110T FPGA
Frequency	100 MHz	155 MHz
Memory bandwidth	32 bits/cycle	48 bits/cycle
Processing speed	150 fps	30 fps

5.2. Disparity Map Quality-Impact of Edge Detection. To evaluate the quality of the disparity maps generated by the proposed edge-directed hardware architectures, and to examine the impact of the edge detection algorithm in the overall system quality, we use Middlebury stereo pairs, as well as stereo pairs from [33], for which the ground truth disparity maps are known and measure the incorrect disparity estimates using the percentage of bad pixels, a commonly accepted metric [2].

In our previous work in [7], we have investigated the impact of three different edge detectors (Sobel, Canny, and Evolvable [34]) on the accuracy of the disparity maps generated by an SAD-based, fixed-support algorithm, in order to select the best detector to be integrated to the system described in [7]. As we discussed in our previous paper, we have selected the Sobel detector, both for its simplicity on an FPGA and also for its good accuracy. In this paper, we also investigate the impact of the same detectors on the ADSW algorithm. The results about the percentage of bad pixels for the Tsukuba and Venus stereo image pairs are given in

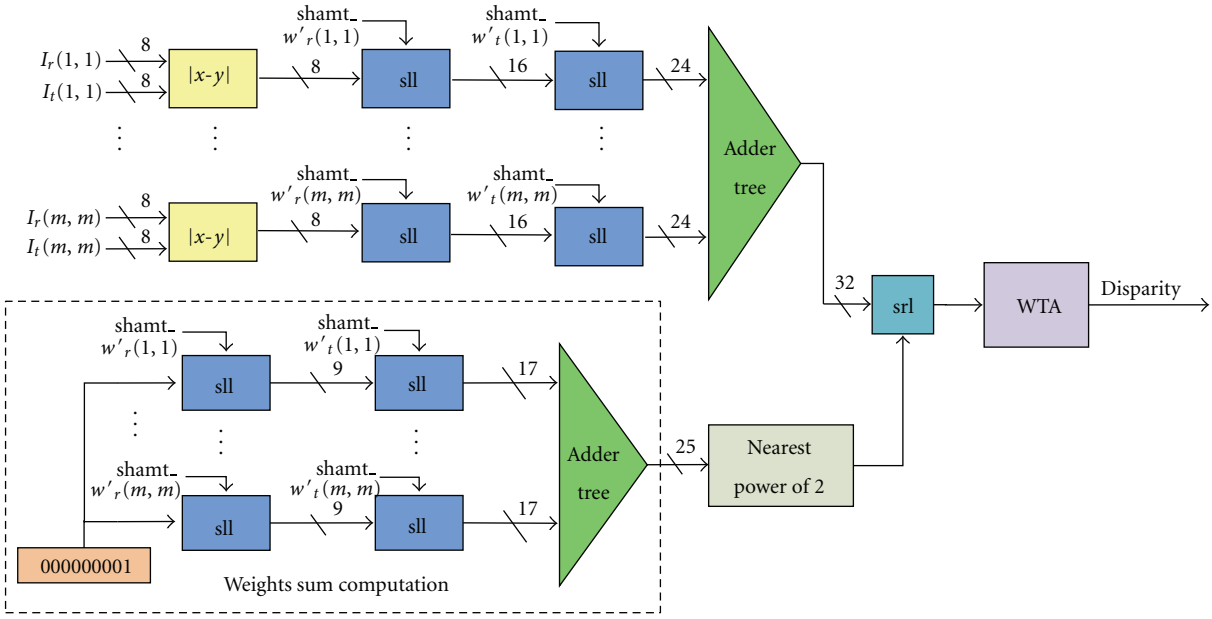

Figure 7: Architecture of the cost aggregator and WTA.

Table 2: Quality comparison for different edge detectors.

Algorithm	Percentage of bad pixels (%)					
	Canny		Sobel		Evolvable	
	Tsukuba	Venus	Tsukuba	Venus	Tsukuba	Venus
Fixed-support	10.30	17.03	8.73	16.73	11.04	12.81
ADSW	7.39	8	5.78	14.61	8.10	8.90

Table 3: Different system configurations.

System architecture	Adopted method
1	SAD-based fixed support without edge detector
2	SAD-based fixed support with Sobel edge detector
3	SAD-based adaptive support weight without edge detector
4	SAD-based adaptive support weight with Canny edge detector

Table 2. The average percentages of data reduction associated to the results presented in the table are 82.95%, 70.23%, and 57.67% for the Canny, Sobel, and Evolvable detectors, respectively.

As can be seen, the Canny detector achieves on average better accuracy (lower percentage of bad pixels) compared to the other two edge detectors when is being applied to the ADSW algorithm, but the quality of the disparity maps is lower when using the Canny detector in the fixed-support algorithm. This difference in the behavior of the Canny detector in the two algorithms lies in the fact that the fixed-support algorithm utilizes edges both as directive points and also as matching primitives. As a result, the fixed support algorithm works better if the edge detector preserves not only strong edges, but also "busy" texture regions (edge regions with numerous small edge elements), which aid the matching process. The Canny detector does not work well with the fixed support algorithm as it is less likely to be fooled by noise, and more likely to detect true weak edges. As such, we have selected the Canny detector to be integrated into the ADSW-based disparity map computation architecture.

For a detailed quality analysis, we compare the quality of the disparity maps generated by the edge-directed architectures to the disparity maps generated by their equivalent hardware architectures that do not integrate the edge detectors (stand-alone fixed-support and ADSW architectures). The stand-alone fixed-support architecture has a similar structure with the edge-directed architecture shown in Figure 2 (without the EDU, the ETU and the candidate window and disparity range buffers), which is able to process 8-bit pixels instead of edges. The stand-alone ADSW architecture has a similar structure with the edge-directed architecture shown in Figure 4 (without the edge detection and interpolation units). For simplicity purposes, we will refer to each hardware system configuration as "System 1–4," as defined in Table 3.

The results extracted by comparing the disparity maps of Systems 1–4 to the ground truth disparity maps are presented in Table 4 for two sample image pairs. The resulting disparity maps are also given in Figure 10 for a qualitative comparison. Obviously, the disparity maps generated by the systems that implement the ADSW algorithm are more accurate, and particularly at depth discontinuity regions and at regions with repetitive patterns, where ADSW algorithms traditionally work better. The results also indicate that the use of the edge detectors does not impact negatively the accuracy of the disparity maps, but, in some cases, they improve the accuracy and especially at depth borders. This is because

Figure 8: Evaluation results for synthetic images. From left to right: Tsukuba, Venus, and pedestrian stereo images. From top to bottom: reference image, target image, output of the Sobel detector, output of the Canny detector, disparity maps generated by the fixed-support-based architecture, disparity maps generated by ADSW-based architecture, ground truth disparity maps.

the stand-alone architectures suffer from changes in pixels intensities, while the edge-directed architectures potentially struggle this problem by limiting the correspondence search to specific reliable features in the images (edges in our case). It must be noted that the aforementioned results are based on a nearest neighbor interpolation method. It is anticipated that the proposed edge-directed hardware systems can produce disparity maps with better quality by integrating more complex interpolation methods such as bilinear and bicubic interpolation; this however is left as part of future work.

5.3. Processing Speed. The processing speed of the edge-directed disparity computation architectures described in

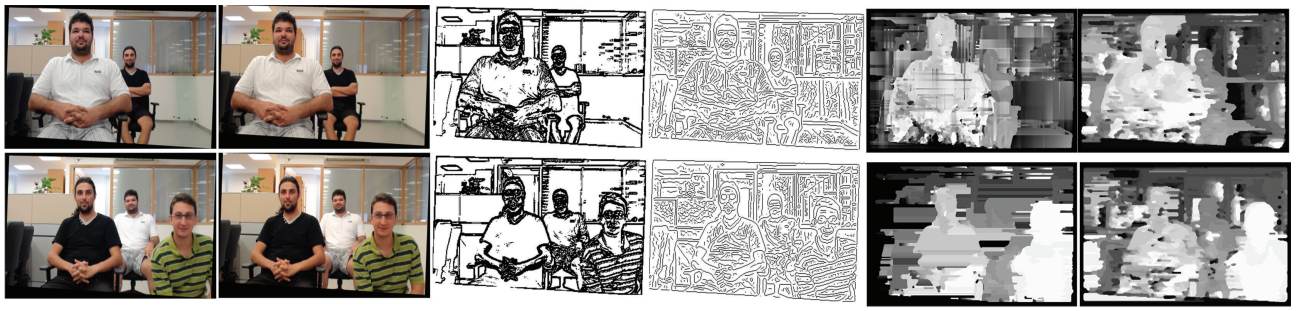

Figure 9: Evaluation results for real-world images. From left to right: reference image, target image, output of the Sobel detector, output of the Canny detector, disparity map generated by the fixed-support-based architecture, disparity map generated by the ADSW-based architecture.

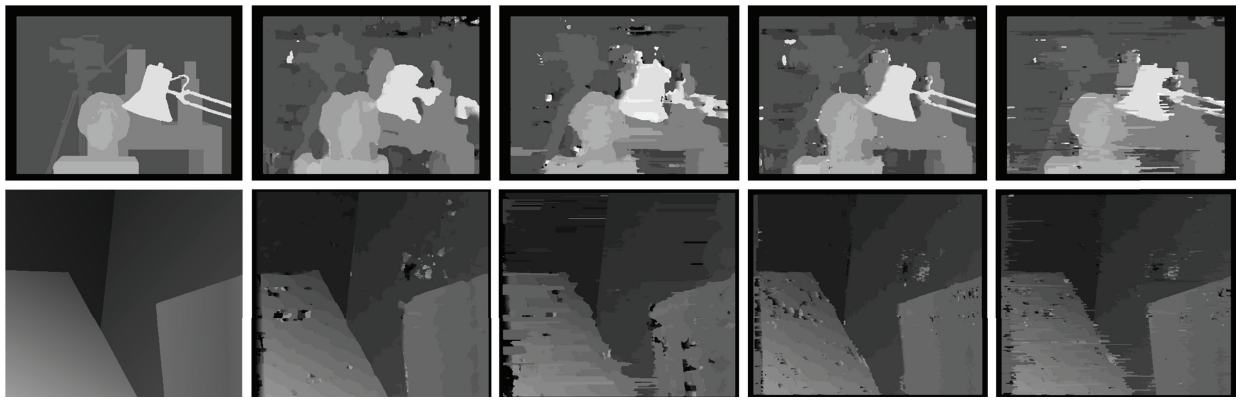

Figure 10: Disparity maps of the Tsukuba and Venus image pairs for the system configurations listed in Table 3. From left to right: ground truth disparity map and disparity maps generated by System 1, System 2, System 3, and System 4, respectively.

Section 4 is measured in frames per seconds (fps) and is affected by several algorithmic and implementation-specific parameters. The algorithmic parameters include the amount of data reduction (the percentage of non-edge points over the total image points) obtained from the edge detectors, the image size and the representation of the input images (grayscale or color), as well as the disparity range. The implementation-specific parameters include the operating frequency, the I/O bandwidth from the external memory, and the available FPGA resources.

The architecture based on the fixed-support approach (System 2) has lower requirements in terms of input bandwidth, as it processes grayscale stereo images. Therefore, the external memory bandwidth can be exploited to fetch multiple pixels per clock cycle compared to the architecture that adopts the ADSW algorithm (System 4), which fetches one pixel per input image. The frame rate of System 2 is also independent from the support window size and the disparity range, as that system utilizes the edge information not only as directive points, but also as matching primitives, thus exploiting more parallelism. System 2 can compute the aggregated costs for all d_M support windows in a single clock cycle. System 4 requires d_M clock cycles to compute the costs for all disparity levels as that system processes color stereo images and uses grayscale pixels as matching primitives. Thus, it can exploit much less parallelism when assuming a constant amount of FPGA resources; particularly, it processes a support window every clock cycle, but requires d_M clock cycles to compute a disparity value in the case where the processing pixel represents an edge. System 4, however, performs the matching process on a smaller number of edge points, since it is directed by the Canny detector, which achieves larger percentage of data reduction compared to the Sobel detector. With respect to the I/O bandwidth and the image size, the performance of both system architectures decreases as the image size increases and the I/O bandwidth decreases, since in the former case there is more data to be processed, while, in the latter case, the amount of data flowing into the system limits the system throughput.

To evaluate the processing speed of the edge-based architectures, we compare them with their equivalent system architectures that do not integrate the edge detectors (stand-alone architectures). We identify the speedup of the proposed architectures compared to the stand-alone architectures and provide results when increasing the input image size and the disparity range, in order to illustrate the impact of the edge detector in all cases. We used synthetic stereo images as benchmarks from [2, 33] in order to extract the processing speed of the systems mentioned above. The results are given in Tables 5 and 6.

As it can be seen, the processing speed is inversely proportional to the image size in all cases. Another particular

Table 4: Comparison of disparity map quality between stand-alone and edge-directed architectures.

Image pair	Percentage of bad pixels											
	System 1*			System 2			System 3*			System 4		
	nonocc. (%)	all (%)	disc. (%)	nonocc. (%)	all (%)	disc. (%)	nonocc. (%)	all (%)	disc. (%)	nonocc. (%)	all (%)	disc. (%)
Tsukuba	8.45	10.4	29.5	8.78	10.5	27.4	7.95	9.54	28.9	7.91	8.60	26
Venus	6.33	7.95	46.0	12.3	13.7	32.3	4.41	5.53	31.7	8.78	9.66	27.9

*Disparity map quality results for System 1 and System 3 were extracted using Matlab simulation.

Table 5: Frame rate versus image size for Systems 1–4 (disparity range = 64, support window size = 11 × 11).

Image size	*w/o* edge detector*		*w*/edge detector	
	Fixed-support	ADSW	Fixed-support	ADSW
320 × 240	1225	34	2570	174
640 × 480	315	8	645	45
800 × 600	203	5	412	30
1024 × 768	124	3	252	18
1280 × 1024	75	1	150	7

*The frame rates for the systems without the edge detectors are projected frame rates.

Table 6: Frame rate versus disparity range for Systems 1–4 (image size = 800 × 600, support window size = 11 × 11).

Disparity range	*w/o* edge detector*		*w*/edge detector	
	Fixed-support	ADSW	Fixed-support	ADSW
16	203	21	412	102
32	203	10	412	59
64	203	5	412	30
128	203	3	412	16

*The frame rates for the systems without the edge detectors are projected frame rates.

observation is that the architectures based on the ADSW approach are slower even if they operate on higher operating frequencies and have larger memory bandwidth; this is due to the high computational complexity of the ADSW algorithm and due to the limited parallelism that can be exploited by such an algorithm in a resource constraint implementation platform such as an FPGA. The most important observation is that the use of the edge detection offers significant speedups, when compared to the stand-alone architectures for all image sizes. The impact of the edge detector is even more emphasized in the architecture based on the ADSW algorithm, as the data reduction of the Canny detector is larger compared to the Sobel. While the use of the edge detector in System 4 yields a speedup of ∼5, the speedup obtained in System 2 is ∼2. This is because System 2 was designed with more parallelism and provides a disparity value every clock cycle, irrespective of whether or not the pixel being processed is an edge. As a result, the edge detector in System 2 needs to provide multiple edges per clock cycle, so the bottleneck in this case is the Edge Detection Unit. Of course, the EDU in System 2 can further be parallelized, so the real limitation in terms of parallelism is the external memory I/O. On the other hand, System 4 exploits parallelism only on the level of a support window. If the pixel being processed is an edge, it requires d_M cycles to compute the disparity, so the edge detector used in System 4 needs to provide only one edge every clock cycle. Table 6 indicates that the disparity range affects only the ADSW-based system architectures, due to the limited parallelism that can be exploited on the targeted FPGA. This however does not necessarily imply that the architecture is not scalable. The parallelism is limited by the amount of hardware resources. Of course, the ADSW-based architecture could be implemented to process multiple support windows in parallel on a larger FPGA. This would increase the frame rate as well.

Table 7 presents a comparison between existing implementations ([12, 13, 15, 16, 19, 20, 22–31]) and the proposed FPGA prototypes (System 2 and System 4). Performance is provided in frames per second (fps), as well as in points × disparity estimates per second (PDS). The proposed edge-directed architecture that implements the fixed-support algorithm (System 2) achieves a PDS of 23592×10^6 and is the fastest implementation listed in the table, when considering the input image size and the maximum disparity range supported. System 4 achieves a PDS of 922×10^6 for an image size of 800×600 and a disparity range of 64. Such PDS seems sufficient considering that the algorithm implemented by System 4 is much more complex compared to the algorithms implemented by the other implementations in the table; only [30] implements an ADSW algorithm and achieves lower performance rates. Conclusively, performance results indicate that the proposed edge-directed approach for accelerating hardware implementations of stereo correspondence algorithms exhibits high potential for applications with hard real-time constraints. System 2 can meet the real-time requirements of such applications, even for high-resolution images. Moreover, the combination of the edge-directed approach with the ADSW algorithm enables the realization of accurate disparity computations systems that can also satisfy the real-time requirements of many embedded vision applications such as pedestrian and obstacle detection.

5.4. Hardware Overheads. The proposed FPGA systems were evaluated for relevant metrics such as area and operating frequency. Table 8 gives the overall hardware demands of the FPGA prototype that is based on the fixed-support algorithm (System 2). The table also lists the hardware overheads associated with each of the implemented components.

Table 7: Comparison of PDS performance for various systems and methods.

Work	Image size	Disparity Range	Frame rate (fps)	PDS (10^6)	Algorithm	Platform
Yang and Pollefeys [12]	512 × 512	32	N/A	289	Block-matching (SAD)	ATI Radeon 9800 graphics card
Yang et al. [13]	384 × 288	16	12.77	22.2	Hierarchical belief propagation	NVIDIA Geforce 7900 GTX graphics card
Khaleghi et al. [15]	160 × 120	30	20	11.5	Census transform based	BlackFin processor-ADSP-BF561 (600 MHz)
Zinner and Humenberger [16]	450 × 375	60	11.8	119.5	Census transform based	Texas Instruments TMS320C6414 DSP
Hile and Zheng [19]	512 × 480	32	30	235.9	Block-matching (SAD)	N/A
Miyajima and Maruyama [20]	640 × 480	80	26	639	Block-matching (SAD)	ADM-XRC-II (40 MHz)
Arias-Estrada and Xicotencatl [22]	320 × 240	16	71	87.2	Block-matching (SAD)	XCV800HQ240-6 (66 MHz)
Lee et al. [23]	640 × 480	64	30	589	Block-matching (SAD)	XC2V8000 (10 MHz)
Hariyama et al. [24]	64 × 64	64	5063	1327.2	Block-matching (SAD)	APEX20KE (86 MHz)
Georgoulas and Andreadis [25]	800 × 600	80	550	21120	Block-matching (SAD)	EP4SGX290 (511 MHz)
Ambrosch et al. [26]	450 × 375	100	600	10125	Block-matching (SAD)	EP2S130 (110 MHz)
Díaz et al. [27]	1280 × 960	29	52	1885	Phase based	Custom FPGA, Xilinx Virtex-II (65 MHz)
Darabiha et al. [28]	256 × 360	20	30.3	55.2	LWPC (phase correlation)	TM-3A board (Xilinx Virtex-4 2000E FPGA)
Jin et al. [29]	640 × 480	64	230	4522	Census transform	Virtex-5 XC4VLX200-10 FPGA (93.1 MHz)
Chang et al. [30]	352 × 288	64	42	272.5	Minicensus adaptive support weight	UMC 90ns Std. Cell
Ambrosch and Kubinger [31]	750 × 400	60	60	1080	SAD-IGMCT	Altera Stratix I (133 MHz)
Proposed (System 2)	1280 × 1024	120	150	23592	Edge-based fixed support weight block matching (SAD)	ML505 Evaluation Board with Virtex-5 LX110T FPGA (100 MHz)
Proposed (System 4)	800 × 600	64	30	922	Edge-based adaptive support weight block matching (SAD)	ML505 Evaluation Board with Virtex-5 LX110T FPGA (155 MHz)

Table 8: Hardware overheads of the edge-directed fixed-support-based architecture (System 2) (image size = 1280 × 1024, max disparity range = 120, window size = 11 × 11).

Design unit	Slice LUTs (69120)	Slice registers (69120)	Block RAMs (148)	DSP48Es (64)	Frequency (MHz)
Edge Detection Unit (EDU)	2812 (∼4%)	1008 (∼3%)	0	0	134.6
ABDIF and BTA	129 (∼0.2%)	0	0	0	N/A
Tree Comparators	1760 (∼2.5%)	329 (∼0.5%)	0	0	174.3
Disparity Computation Unit (DCU)	47331 (∼68%)	31863 (∼46%)	0	0	109
Microblaze system and external Memory CONTROLLER	7754 (∼11%)	8562 (∼12%)	30 (∼20%)	6 (∼9%)	161.2
Entire system	**66882** (∼**83.8**%)	**41559** (∼**60**%)	**30** (∼**20**%)	**6** (∼**9**%)	**100**

The entire system utilizes ∼83.8% of the FPGA slice LUTs and ∼60% of the FPGA slice registers. The more hardware demanding component is the DCU, as obviously anticipated, due to the large amount of computations performed by that unit. The EDU requires only a small amount of resources, despite the fact that it processes two pixels per input image. This is attributed to the simplicity of the circuits that compute the convolution operation (using shifters instead of multipliers). Table 9 gives the overall hardware demands of the FPGA prototype that is based on the ADSW algorithm (System 4). That prototype utilizes ∼73.1% of the FPGA LUTs and ∼64.8% of the FPGA slice registers and can operate

Table 9: Hardware overheads of the edge-directed ADSW-based architecture (System 4) (image size = 800 × 600, max disparity range = 64, support window size = 11 × 11).

Design unit	Slice LUTs (69120)	Slice registers (69120)	Block RAMs (148)	DSP48Es (64)	Frequency (MHz)
rgb2gray	0	0	3 (∼4.7%)	0	N/A
rgb2yuv	261 (∼0.4%)	0	0	0	N/A
Weight generator	7056 (∼10.2%)	0	0	0	N/A
Cost aggregator	30691 (∼44.4%)	434 (∼0.6%)	0	0	391
WTA	83 (∼0.1%)	61 (∼0.1%)	0	0	429
On-chip MA (left)	228 (∼0.3%)	2304 (∼3.3%)	0	0	632
On-chip MA (right)	644 (∼0.9%)	2574 (∼3.7%)	0	0	582
Canny detector	2060 (∼3%)	947 (∼1.4%)	0	0	165
Microblaze system & external memory Controller	7754 (∼11.2%)	8562 (∼12.4%)	6 (∼9.4%)	30 (∼20.3%)	161
Complete system	**50539** (∼**73.1**%)	**44802** (∼**64.8**%)	**12** (∼**18.8**%)	**30** (∼**20.3**%)	**155**

at 155 MHz. The slice LUTs are dominated by the cost aggregator and the weight generators, which consume ∼44% of the available LUTs. The slice registers are dominated by the on-chip MAs, which consume ∼28% of the available slice registers.

A closer look in Tables 8 and 9 leads us to the following useful observations.

(i) Even though System 2 exploits parallelism aggressively in order to compute the cost values for all support windows in the range $[1\ d_M]$ in a clock cycle, it requires only ∼10% more FPGA LUTs than those required by System 4. This indicates that the edge-based approach for stereo correspondence is much more efficient (in terms of hardware usage) when applied to a fixed-support algorithm, mainly due to the fact that it performs correlation using binary data (edges), thus requiring very simple circuits during the cost computation and aggregation steps.

(ii) While the Sobel detector is computationally less demanding than Canny, it requires ∼1% more FPGA LUTs. This is due to the fact that the EDU is able to process 4 pixels per clock cycle (2 per input image); thus, the hardware resources have been replicated by a factor of 4. The Canny detector, on the other hand, outputs only a pixel per cycle.

(iii) The EDU operates at a slower clock frequency compared to the Canny detector. It could be possible to divide the EDU into more pipeline stages so that to increase its frequency, at the expense of further hardware usage. However, this would be meaningless considering that the DCU works at a slower frequency. Moreover, the frame rates achieved by System 2 are extremely high, and therefore any further increase of the operating frequency to achieve higher frame rates would not justify the extra hardware cost.

6. Conclusion

Stereo correspondence is an important algorithm in several embedded vision applications that require real-time computation of depth information. This paper presented an overview of the use of edge information, as a means to accelerate hardware implementations of stereo correspondence algorithms. The paper analyzed design issues and considerations associated with the edge-directed approach when it was applied to different hardware implementations of both fixed-support and even more complex ADSW algorithms.

Our immediate plans involve the integration of other interpolation methods into the proposed architectures in order to achieve better disparity map accuracy. We also plan to investigate the applicability of the presented edge-directed approach to global stereo correspondence algorithms, which return better quality on the disparity map. Furthermore, the FPGA prototypes will be extended to full-custom ASIC designs in order to evaluate large-scale performance and power.

References

[1] B. Cyganek and J. P. Siebert, *Introduction to 3D Computer Vision Techniques and Algorithms*, Wiley, John & Sons, 2009.

[2] 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, 2002.

[3] M. Z. Brown, D. Burschka, and G. D. Hager, "Advances in computational stereo," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 8, pp. 993–1008, 2003.

[4] K.-J. Yoon and I.-S. Kweon, "Adaptive support-weight approach for correspondence search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 4, pp. 650–656, 2006.

[5] F. Tombari, S. Mattoccia, and L. Di Stefano, "Segmentation-based adaptive support for accurate stereo correspondence," in *Proceedings of the 2nd Pacific Rim Conference on Advances*

in Image and Video Technology, vol. 4872 of *Lecture Notes in Computer Science*, pp. 427–438, Springer, December 2007.

[6] M. Gerrits and P. Bekaert, "Local stereo matching with segmentation-based outlier rejection," in *Proceedings of the 3rd Canadian Conference on Computer and Robot Vision (CRV '06)*, p. 66, June 2006.

[7] C. Ttofis, S. Hadjitheophanous, A. S. Georghiades, and T. Theocharides, "Edge-directed hardware architecture for real-time disparity map computation," in *Proceedings of the IEEE Transactions on Computers*, 2012.

[8] K. Mühlmann, D. Maier, J. Hesser, and R. Männer, "Calculating dense disparity maps from color stereo images, an efficient implementation," *International Journal of Computer Vision*, vol. 47, no. 1–3, pp. 79–88, 2002.

[9] S. Forstmann, Y. Kanou, J. Ohya, S. Thuering, and A. Schmitt, "Real-time stereo by using dynamic programming," in *Proceedings of the Computer Vision and Pattern Recognition Workshop (CVPRW '04)*, p. 29, June 2004.

[10] L. D. Stefano, M. Marchionni, and S. Mattoccia, "A fast area-based stereo matching algorithm," *Image and Vision Computing*, vol. 22, no. 12, pp. 983–1005, 2004.

[11] H. Hirschmüller, P. R. Innocent, and J. Garibaldi, "Real-time correlation-based stereo vision with reduced border errors," *International Journal of Computer Vision*, vol. 47, no. 1–3, pp. 229–246, 2002.

[12] R. Yang and M. Pollefeys, "A versatile stereo implementation on commodity graphics hardware," *Real-Time Imaging*, vol. 11, no. 1, pp. 7–18, 2005.

[13] Q. Yang, L. Wang, R. Yang, S. Wang, M. Liao, and D. Nister, "Real-time global stereo matching using hierarchical belief propagation," in *Proceedings of the The British Machine Vision Conference*, 2006.

[14] R.-P. M. Berretty, A. K. Riemens, and P. F. Machado, "Real-time embedded system for stereo video processing for multi-view displays," in *Proceedings of the Stereoscopic Displays and Virtual Reality Systems XIV*, Proceeding of SPIE, San Jose, Calif, USA, January 2007.

[15] B. Khaleghi, S. Ahuja, and Q. M. J. Wu, "An improved real-time miniaturized embedded stereo vision system (MESVS-II)," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops*, pp. 1–8, June 2008.

[16] C. Zinner and M. Humenberger, "Distributed real-time stereo matching on smart cameras," in *Proceedings of the 4th ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC '10)*, pp. 182–189, New York, NY, USA, September 2010.

[17] S. Cavanag and M. Manzke, "Real time disparity map estimation on the cell processor," in *Proceedings of the Eurographics Ireland Workshop 2009*, pp. 67–74, Trinity College Dublin, December 2009.

[18] J. Liu, Y. Xu, R. Klette, H. Chen, W. Rong, and T. Vaudrey, "Disparity map computation on a cell processor," in *Proceedings of the IASTED International Conference on Modelling, Simulation, and Identification (MSI '09)*, Beijing, China, October 2009.

[19] H. Hile and C. Zheng, "Stereo video processing for depth map," Tech. Rep., University of Washington, 2004.

[20] Y. Miyajima and T. Maruyama, "A real-time stereo vision system with FPGA," in *Proceedings of the 13th International Conference on Field-Programmable Logic and Applications (FPL '03)*, pp. 448–457, Lisbon, Portugal, 2003.

[21] L. Nalpantidis, G. Sirakoulis, and A. Gasteratos, "Review of stereo matching algorithms for 3D vision," in *Proceedings of the 16th International Symposium on Measurement and Control in Robotics (ISMCR '07)*, pp. 116–124, Warsaw, Poland, June 2007.

[22] M. Arias-Estrada and J. M. Xicotencatl, "Multiple stereo matching using an extended architecture," in *Proceedings of the Field-Programable Logic and Applications*, vol. 2778, pp. 203–212, Springer, 2003.

[23] S. H. Lee, J. Yi, and J. Kim, "Real-time stereo vision on a reconfigurable system," in *Proceedings of the Embedded Computer Systems: Architectures, Modelling and Simulation*, vol. 3553, pp. 299–307, Springer, 2005.

[24] M. Hariyama, Y. Kobayashi, M. Kameyama, and N. Yokoyama, "FPGA implementation of a stereo matching processor based on window-parallel-and-pixel-parallel architecture," in *Proceedings of the IEEE International 48th Midwest Symposium on Circuits and Systems, (MWSCAS '05)*, vol. 2, pp. 1219–1222, Covington, Ky, USA, August 2005.

[25] C. Georgoulas and I. Andreadis, "A real-time occlusion aware hardware structure for disparity map computation," in *Proceedings of the Image Analysis and Process (ICIAP '09)*, vol. 5716, pp. 721–730, 2009.

[26] K. Ambrosch, M. Humenberger, W. Kubinger, and A. Steininger, "A SAD-based stereo matching using FPGAs," in *Proceedings of the Embedded Computer Vision part II*, pp. 121–138, Spinger, 2009.

[27] J. Díaz, E. Ros, R. Carrillo, and A. Prieto, "Real-time system for high-image resolution disparity estimation," *IEEE Transactions on Image Processing*, vol. 16, no. 1, pp. 280–285, 2007.

[28] A. Darabiha, J. MacLean, and J. Rose, "Reconfigurable hardware implementation of a phase-correlation stereoalgorithm," *Machine Vision and Applications*, vol. 17, no. 2, pp. 116–132, 2006.

[29] S. Jin, J. Cho, X. D. Pham et al., "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, 2010.

[30] 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," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 20, no. 6, pp. 792–805, 2010.

[31] K. Ambrosch and W. Kubinger, "Accurate hardware-based stereo vision," *Computer Vision and Image Understanding*, vol. 114, no. 11, pp. 1303–1316, 2010.

[32] M. Raman and H. Aggarwal, "Study and comparison of various image edge detection techniques," *The International Journal of Image Processing*, vol. 3, no. 1, pp. 1–12, 2009.

[33] "A Framework for Evaluating Stereo-Based Pedestrian Detection Techniques," http://www.cdvp.dcu.ie/datasets/pedestrian_detection/.

[34] Z. Vasicek and L. Sekanina, "An evolvable hardware system in Xilinx Virtex II Pro FPGA," *International Journal of Innovative Computing and Applications*, vol. 1, no. 1, pp. 63–73, 2007.