A Complete Real-Time Feature Extraction and Matching System Based on Semantic Kernels Binarized

Michael Schaffner, P. A. Hager, L. Cavigelli, Z. Fang, P. Greisen, F. K. Gürkaynak, A. Smolic, H. Kaeslin, L. Benini

A Complete Real-Time Feature Extraction and Matching System Based on Semantic Kernels Binarized

M. Schaffner[1,2], P. A. Hager[1], L. Cavigelli[1], Z. Fang[1], P. Greisen[1], F. K. Gürkaynak[1], A. Smolic[2], H. Kaeslin[1], and L. Benini[1]

[1] ETH Zurich, Switzerland
{schaffner,greisen,kgf,kaeslin,benini}@iis.ee.ethz.ch
{phager,lukasc,fangz}@student.ethz.ch

[2] Disney Research Zurich, Switzerland
{smolic}@disneyresearch.com

Abstract. Feature extraction and matching is an important step in many current image and video processing algorithms. In this work, we designed and implemented an efficient feature extraction and matching system for sparse point correspondence search in stereo video. Our system is based on the recently proposed *Semantic Kernels Binarized* (SKB) algorithm, which showed superior performance with respect to other algorithms in our evaluation. The feature extraction stage has been prototyped in 180 nm technology and the complete system with two feature extraction pipelines (left and right view) together with the matching unit have been implemented on a Stratix IV FPGA where it delivers a performance of up to 42 frames per second on 720p video. Especially due to the high throughput of up to 25 k matched descriptors per frame, our system compares favourably with recent hardware implementations of similar algorithms.

Keywords: Features, Matching, Binary Descriptor, Semantic Kernels Binarized (SKB), Stereo Video Processing, Real Time, VLSI, ASIC, FPGA

1 Introduction

Current image and video processing pipelines commonly rely on image features in order to calculate sparse point correspondences among several images or frames. Over the past decade, numerous different algorithms and variations thereof have been devised. Earlier methods such as SIFT [14] and SURF [5] are costly to compute and the calculated descriptors consist of many floating point entries that require a significant amount of memory – which renders them less attractive for embedded devices or hardware implementations. This led to the development of more efficient *binary* descriptors such as BRIEF [8], BRISK [13], FREAK [4] and *Semantic Kernels Binarized* (SKB) [26]. These are less expensive to compute (e.g. BRIEF and SKB directly compute the binary descriptor pattern by means of intensity comparisons or thresholding), require less storage, and can be matched very efficiently using the Hamming distance.

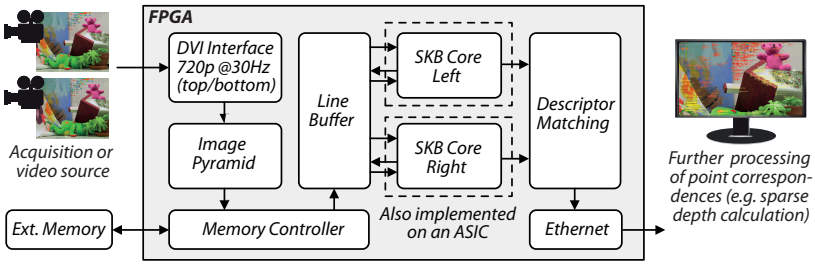

Fig. 1. Overview of the stereo video feature detection system with two SKB cores and one matching unit. The 'Teddy' image shown here is from the dataset provided by [20].

In this work, we consider the calculation of a sparse disparity map from stereo video, as this is a crucial ingredient for certain video processing methods such as automatic stereo-to-multiview conversion [22]. Our system implements the SKB algorithm [26], as it provides competitive results in the restricted setting of stereo vision and is amenable to efficient hardware implementations. We present a hardware architecture of the whole feature extraction and matching system (a system overview is shown in Figure 1). The feature extraction core has been implemented and fabricated in 180 nm CMOS technology, and the whole system has been implemented on a Stratix IV FPGA.

1.1 Related work

There exist many FPGA and ASIC implementations of SIFT and SURF, such as [6, 7, 12, 18, 21, 23]. But to our knowledge, the following two are the only other FPGA/ASIC implementations of complete feature extraction and matching systems which are based on binary descriptors. J. S. Park, *et al.* [16] implemented a variant of BRIEF with FAST keypoints [17] on an ASIC that has a throughput 94.3 full-HD frames per second with 512 extracted feature points per frame. J. Wang, *et al.* [25] propose a FPGA system which is also based on BRIEF, but with SIFT keypoints. In their implementation, they achieve a feature throughput of 60 fps for 720p video with 2k detected points. Both of these works are single view systems matching the extracted descriptors to the ones extracted from the previous frame. In contrast, our system operates on the left and right views simultaneously, and the constrained stereo camera set-up is exploited in order to match the descriptors from both views on-the-fly. Further, our system is capable of extracting and matching a much larger number of features per frame (15 k - 25 k).

1.2 Summary of contributions

We have developed a hardware architecture for real-time SKB feature extraction and matching from 720p stereo video for real-time stereo vision applications. Instead of using a two-dimensional integral image[1] to compute the filter responses, we use a local one-dimensional integral image in order to overcome the large memory entries and the

[1] The 2D integral image is defined as $\mathbf{II}_{xy} = \sum_{x' \leq x} \sum_{y' \leq y} \mathbf{I}_{x'y'}$ [24].

associated bandwidth that a two-dimensional integral image entails. Since the feature extraction part is able to extract a large amount of interest points (up to 25 k) per frame, we developed a high throughput matching engine able to match interest points at this rate. The system has been implemented on a Stratix IV based FPGA evaluation board where it runs with up to 142 MHz, delivering a throughput of 42 fps (stereo video). Finally, we compare our work with other state-of-the-art implementations.

This chapter is an extension of our conference paper [19], where we developed and implemented the feature extraction core of the presented system.

1.3 Chapter Organization

The system overview and our version of the SKB algorithm is explained in Section 2, and a performance simulation of the implemented configuration is shown at the end of this section. The hardware architecture is explained in Section 3.1, the implementation results and comparisons are given in Section 4, and Section 5 concludes the chapter.

2 Algorithm Details

When searching for sparse point correspondences in an image pair using features, the three main steps that have to be performed are the *interest point detection*, the *descriptor calculation* and the *descriptor matching*. In the following, we summarize these steps of our version of the SKB algorithm and point out the differences to the original [26]. A performance comparison with other descriptors is given at the end of this section.

2.1 Interest Point Detection

Similar to the original SKB implementation, we use a variant of the simple *Difference of Boxes* (DoB) filter to detect interest points in the image. However, our system builds upon the original DoB version of CenSurE [2] instead of the modified SUSurE DoB [9] that is used in the SKB paper. The CenSurE detector basically performs a pixel-dense scan on all scales, whereas the SUSurE detector uses a scan line sparsification to leave out pixel positions which are not likely to lead to an extremal filter response. The SUSurE detector is about $3\times$ faster than the CenSurE in software [9], but it exhibits a data dependent, irregular flow which inhibits parallel processing of different filter scales. The DoB filter is a simplified *Laplacian of Gaussian* (LoG) filter and its response is given by the subtraction of the pixel sums within two quadratic boxes with side length $2n+1$ and $4n+1$. The advantage of this simplification is that it can be efficiently calculated using an integral image [24], since the area of a box can be obtained by only adding/subtracting the integral image values at the box corners. The image is filtered using different sizes of this filter, thereby forming a volume of filter responses which is also denoted as *scale space*. As noted in [5, 21], the largest number of interest points is found on the first few scales. Therefore, and because and in our application we do not require a large scale invariance, the scale space is limited to the first 8 scales (i.e. $n \in [1,2,\ldots,N_{max}=8]$).

The DoB filter responses in the scale space are checked for extremal points in a local $3 \times 3 \times 3$ neighborhood using *non-maximum suppression* (NMS), and a weak response threshold t_{wk} is applied in order to filter out non-robust interest points. Since a part of the NMS neighborhood is unknown at the scale space border, maxima are not allowed to occur there. Interest points are therefore only detected on 6 out of 8 scales. Note that the integrated box areas must be properly normalized [2] such that comparisons among different scales are possible. The accuracy of the interest point coordinates could be enhanced by additional interpolation of the maxima location, but since the DoB filters are evaluated pixel-dense on each scale this is not done here. Further, no Harris corner test is performed in our implementation, since this operation is very costly and often not necessary in this application [26]. The output of the interest point detection is a set of tuples $\{(x_i, y_i, s_i)\}_{i \in \{0,1,\ldots,M\}}$, where x_i and y_i are the integer image coordinates and s_i is the index of the scale of a particular interest point i.

Depending on the value of the weak response threshold t_{wk}, different amounts of interest points are detected. It is desirable to adjust this threshold, such that each frame of a video yields around the same number of points. Since the DoB filter responses are basically differences of image areas, their magnitudes correlate with the average gradient magnitude[2] in the image. Although this relationship may be used to set the threshold to a suitable value, we found that this does not stabilize the number of points well over a video sequence since the image content itself has a large influence on how many points are detected. A feedback loop with simple rules on how to adapt the threshold has proven to be much more effective. In our version of the DoB detector, we use the following rules

$$t_{wk} := \begin{cases} \text{if } I < I_{lo2} & , \max\left(t_{wk} - \Delta t_{wk2},\ t_{wk}^{min}\right) \\ \text{else if } I < I_{lo1} & , \max\left(t_{wk} - \Delta t_{wk1},\ t_{wk}^{min}\right) \\ \text{else if } I > I_{hi2} & , \min\left(t_{wk} + \Delta t_{wk2},\ t_{wk}^{max}\right) \\ \text{else if } I > I_{hi1} & , \min\left(t_{wk} + \Delta t_{wk1},\ t_{wk}^{max}\right) \end{cases}, \tag{1}$$

where I_{lo2}, I_{lo1}, I_{hi1}, I_{hi2} are the decision boundaries, and Δt_{wk1}, Δt_{wk2} are the step sizes, and t_{wk}^{min}, t_{wk}^{max} are the minimal and maximal threshold values (used for saturation). Figure 2a shows the behavior of the feedback loop when applied to a test video sequence containing three scene cuts. Using 4 decision boundaries instead of only 2 allows to use larger step sizes if the number of interest points is far away from the target. Note that the step size should not be chosen too large since this may cause an oscillatory behavior. A similar functionality to control the population of detected interest points is provided by the *AdjusterAdapter* class in OpenCV [1].

[2] The average gradient magnitude is given by $m_{avg} = \frac{1}{X \cdot Y} \sum_{x=1}^{X} \sum_{y=1}^{Y} \left\| \nabla \mathbf{I}_{xy} \right\|$.

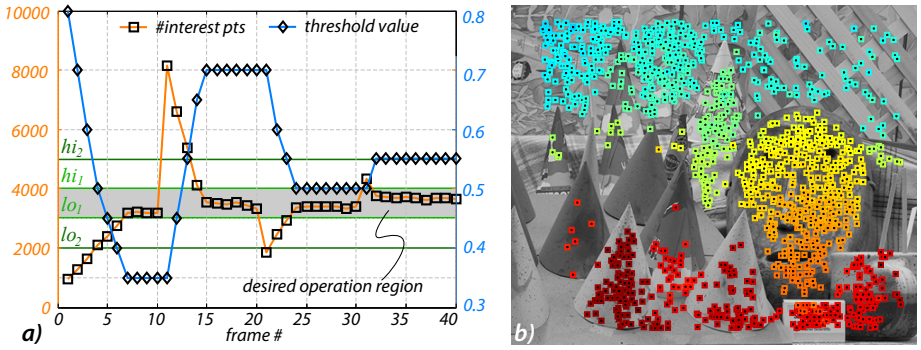

Fig. 2. a) Behavior of the weak response threshold adjustment scheme in the interest point detection step (every 10 frames there is a scene change in this video sequence). The following parameter set has been used in order to stabilize the number of points between $3k$ and $4k$: $(I_{lo2}, I_{lo1}, I_{hi1}, I_{hi2}) = (2k, 3k, 4k, 5k)$, $(\Delta t_{wk1}, \Delta t_{wk2}) = (0.05, 0.1)$ and $\left(t_{wk}^{min}, t_{wk}^{max}\right) = (0.1, 1.0)$. b) example for a sparse disparity estimation using SKB 256 bit type B descriptors (this is the left image and the estimated disparity has been color-coded). Note that there are almost no outliers even tough no post-processing step like RANSAC has been performed.

2.2 SKB Descriptor Calculation

The SKB descriptor makes use of a set of sixteen 4×4 filter kernels (also called *semantic kernels*, shown in Figure 9b) which are evaluated at 16 positions within a normalized support region around an interest point. This leads to 256 values which are binarized using a certain thresholding scheme. In [26] they propose three different binarization variants A, B and C where A leads to a 256 bit descriptor and B and C lead to a 512 bit descriptor. Here we use the fast variant A where the 256 values are binarized by comparing them against 0, i.e. only the sign bit is kept. In [26] they also define two different support regions (type A and B) out of which we use the larger 16×16 region (B), as our experiments show that it performs slightly better (Figure 3).

The (x_i, y_i, s_i) tuples from the interest point detector are used to calculate the coordinates of the support region of that interest point. Bilinear interpolation is then used to resample the support region such that it fits into the normalized frame of 16×16 pixels (the normalization factors are given by the ratio of the outer DoB box size of the actual scale and the smallest scale i.e. $(4 \cdot s_i + 1)/5$). Note that we do not perform any rotational alignment as this is not necessary in the case of stereo matching. In order to facilitate the resampling step, we precompute an image pyramid by successive down sampling of the input image by a factor of two. Depending on the scale factor of an interest point i, the nearest pyramid level is selected as the pixel source (this concept is also known as *mipmapping* [3]). This has the advantage that aliasing artifacts are reduced in the resampled patches and that the accessed image patch is always contiguous.

2.3 Descriptor Matching

Generally speaking, feature matching in this context is the process of finding the optimal assignment of feature points $i \in \{1,2,\ldots,I\}$ extracted from the left stereo image to feature points $j \in \{1,2,\ldots,J\}$ extracted from the right stereo image. I.e. we have to solve an assignment problem of the form

$$\mathscr{S} = \arg\min_{\mathbf{S}} \sum_{i,j} (S_{ij} \cdot C_{ij}), \tag{2}$$

where C_{ij} are the elements of a $I \times J$ matching cost matrix $\mathbf{C}$; and $S_{ij} \in \{0,1\}$ are the elements of a $I \times J$ assignment matrix $\mathbf{S}$, which contains exactly $\min(I,J)$ nonzero elements, and at most one nonzero entry per row and column. As the feature descriptors are binary vectors in this case, the cost of matching a descriptor $\mathbf{d}_i$ from the left image with a descriptor $\mathbf{d}_j$ from the right image can be efficiently calculated using the *Hamming Distance* between the two vectors, i.e.,

$$C_{ij} = \sum_{k=1}^{N} \left(d_{ik} \text{ xor } d_{jk}\right), \tag{3}$$

where k is the bit index in the vectors. However, the assignment problem itself is computationally challenging if a globally optimal solution has to be computed. E.g., the *Hungarian Method* which is often employed to solve this kind of problems has a complexity of $\mathscr{O}\left(I^3\right)$, where I is the expected number of descriptors in the left and right image. Furthermore, a direct solution of the assignment problem (2) without any further constraints may still lead to suboptimal solutions, since the geometrical relationship between the two images is completely ignored. For example the solution with the least overall cost could assign a feature point in the top right corner of the left image to a feature point in the bottom left corner of the right image, which completely violates the geometry of a rectified stereo setup. It is therefore common to constrain the search to a small window, based on the knowledge about the camera setup. In our work, we adopt a *greedy* nearest-neighbour (NN) search which is computationally efficient and – as will be seen in the performance evaluation in Section 2.4 – which provides sufficient accuracy. An additional *Random Sample Consensus* (RANSAC) [10] step could be performed to eliminate wrong matches, but this is currently not done in our hardware implementation. The details of the window search are given below.

Windowed Nearest-Neighbour Search In this work, we match points from the left image to the right, i.e., the list of descriptors from the left image is traversed, and for each point i we exhaustively search a corresponding point j in a window in the right image. This window is defined by $(\Delta y, \Delta x_1, \Delta x_2)$, and the point j has to fulfill the following set of constraints

$$\{|y_i - y_j| < \Delta y, \quad x_j - x_i < \Delta x_1, \quad x_i - x_j < \Delta x_2\}. \tag{4}$$

Δy is the maximum y disparity, Δx_1 is the maximum negative- and Δx_2 the maximum positive disparity. Two different Δx values are used here, since the positive and negative

disparity ranges in stereo setup are dependent on the alignment of the cameras and are often not equal. Δy accounts for disparities in y direction that occur if the image planes of the two cameras are not completely coplanar[3]. The current descriptor $\mathbf{d}_i$ from the left image is compared to all descriptors $\mathbf{d}_j$ within the matching window (4) using the hamming distance. The point with descriptor $\mathbf{d}_j$ with the lowest hamming distance is deemed the correspondence of the point i if the hamming distance lies below the matching threshold t_{match} (usually $\in \{10,\ldots,30\}$ here). Once a match has been found, the points j and i are removed from the descriptor lists. This method is *greedy*, but it has a complexity of only $\mathcal{O}(I \cdot C \cdot \log(J))$, where C is the average number of points in the matching window (assuming that the descriptors from the right image have been sorted according to their position, and that a point lookup has complexity $\mathcal{O}(\log(J))$).

2.4 Descriptor Performance

Evaluation Setup We developed a similar framework as K. Mikolajczyk, *et al.* [15] in order to compare the descriptor performance of different SKB variants against the implementations of SIFT [15], SURF [5] and BRIEF [8]. The difference to the framework of K. Mikolajczyk, *et al.* is that we use the Middlebury stereo test set [11], since we are interested in the performance on rectified stereo content. In the framework of K. Mikolajczyk, *et al.*, the test set comprises images with blur and JPEG distortion artifacts, different exposures, and camera rotations – which is useful to evaluate descriptors for applications where a lot of invariance is required (such as image in stitching or object recognition), but it does not model a stereo-camera setup.

The SIFT, SURF and BRIEF descriptors were all parameterized using the default configuration. The SIFT and SURF descriptors have dimensions 128 and 64, respectively, and the Euclidean distance is used to calculate their matching cost. The BRIEF descriptor has dimension 256 and the matching cost is calculated using the Hamming distance. As explained in Section 2.2, the SKB descriptor comes in different flavors, out of which the computationally more efficient 256 bit types *A* and *B* are evaluated. The fixed point model of the hardware implementation is included in the evaluation as well (type *B*). The descriptors were all evaluated on the same DoB interest points and are upright, since no orientation information is extracted in the interest point detector. The descriptor performance is assessed using *1 - precision* vs. *recall* plots. This is a parametric plot, where the matching threshold t_{match} is swept from 0 to the maximum matching cost. The precision is defined as the percentage of false matches, and the recall is the ratio of the number of correct matches and the number of existing true matches. The dataset [11] provides the ground truth in the form of accurate depth maps. We use these maps in order to transform interest points from one view to the other to check for true correspondences. In this evaluation, we always used *view1* and *view5* of the scenes. The recision-recall curves are averages over the whole test set.

Results and Discussion The evaluation results are shown in Figure 3. In Figure 3a the Hungarian method has been used to calculate a globally optimal matching, whereas

[3] This is also called *Keystone distortion*.

in Figure 3b a greedy, windowed NN matching has been used. In Figure 3c, the two different SKB types and our fixed-point implementation are compared (the curve for BRIEF is shown as a reference, here). We can observe from Figure 3a and b that SKB outperforms all other descriptors, and has a very steep precision-recall curve, i.e., at low thresholds, many accurate point correspondences are found. This observation is in line with the claim of F. Zilly, *et al.* [26]. When comparing the results from Figure 3a and Figure 3b we can see that a globally optimal matching performs better – especially in the region with higher thresholds. There is no 'bending down' of the precision-recall curve, as can be observed in Figure 3b. But we also note that in the low threshold region there is almost no difference. Since we intend to operate in this region, the greedy NN matching is a perfectly feasible choice. We can see in Figure 3c that among the SKB types A and B, the latter performs slightly better, and our fixed-point implementation of SKB-B shows no performance degradation.

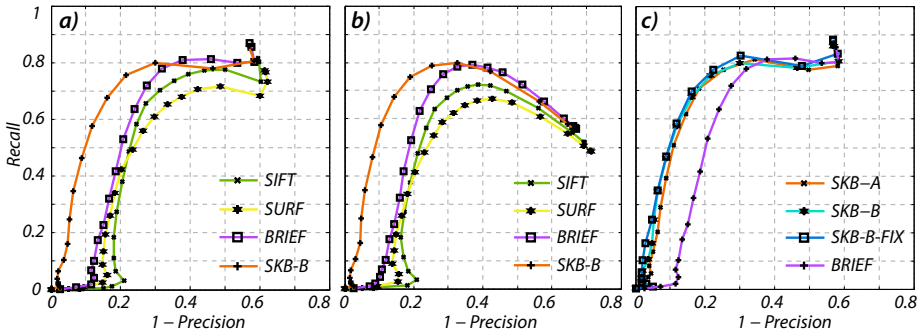

Fig. 3. Matching performance simulation with the stereo test set from [11]. The *recall* is the ratio of correct matches and existing correspondences between left and right image, and *1-precision* is the percentage of false matches [15]. a) globally optimal matching, b) greedy NN matching (left to right), c) comparison of different SKB types and our fixed point implementation

3 Hardware Architecture

The complete feature extraction and matching system given in Figure 1 was developed in several distinct stages. Initial work concentrated on designing an efficient *SKB core* in hardware. The resulting core (described in Section 3.2) was implemented as a custom ASIC using a relatively mature 180 nm technology from UMC. In the later stages of the project, the development was moved to the FPGA based prototyping system Terasic DE4-530 which is based on an Altera Stratix IV FPGA (EP4SGX530KH40C2).

The presented system has been designed to be part of a much larger image processing pipeline currently in development at our institute. The overall system adds some additional constraints on the feature extraction and matching system presented here. The first constraint is the need to process stereoscopic images in a top-bottom format.

In order to take advantage of existing video infrastructure, frames of a stereoscopic video that consist of a left and right image are usually encoded as a single image of an existing video format. Two widely used formats are side-by-side, which as the name implies places the two images next to each other, and top-bottom, which places one image on top of the other. The second constraint is the need for a relatively high number of interest points (and corresponding descriptors) per frame. This is because our target application multiview synthesis [22] requires many point correspondences in the order of 5 k. Since the images are processed in scan-line fashion, the system must be able to cope with clusters of interest points, which demands a throughput which is higher than the average number of points that is detected per frame. Further, not all interest points are going to lead to a match – depending on the matching threshold t_{match}, this number is about 60% of the detected interest points.

One of the fundamental decisions in the entire system has been to process streaming video data and to avoid storing entire image frames within the processing core. The challenge in the design is to balance the amount of storage in the system with the practical I/O bandwidth limitations.

In the following, we will explain individual components of the feature extraction and matching system shown in Figure 1. It consists of three main parts: The input buffering and image pyramid calculation, the detection of interest points and calculation of the descriptors, and finally the matching unit that performs the matching and sends the results out via Ethernet for further processing.

3.1 Image Pyramid and Line Buffer

The architecture of the image pyramid calculation and the line buffer is shown in 4a and b. The incoming 720p video is converted to gray scale and the image pyramid consisting of a 360p and 180p resolution level is calculated and stored in the off-chip RAM[4]. The image pyramid is calculated by averaging four neighboring pixels, which results in a reduction of two in both dimensions. This averaging is implemented using two adders and a delay line per reduction step. Since the two views in the input video arrive sequentially after each other in top-bottom format, only two reduction instances are required. The calculated pixel streams are then parallelized and buffered such that long bursts can be written to the off-chip memory. In our system, the external memory is a PC2-6400 DIMM, and a local memory controller interface provides a 256 bit wide access at 200 MHz clock rate, resulting in a peak data rate of 1.6 GByte/s. The image pyramid calculation block itself works with the input pixel clock (72.5 MHz), and the off-chip bandwidth for 3 pyramid levels and 2 views amounts to 72.6 MByte/s at 30 fps.

The *line buffer* keeps a sliding window of each of the pyramid levels on-chip, such that the requests from the *SKB cores* can be served with low latency and without generating additional overhead to the off-chip memory. Per view, a *DoB feed* unit transfers the lowest level of the image pyramid to the *interest point detection* stage of the *SKB cores*. The corresponding sliding window of the lowest pyramid level comprises 62 rows. Of this total, 48 rows are needed to provide the 1D integral image to the *DoB feed*

[4] Since we use only eight scales in our implementation, three pyramid levels are sufficient.

in column-wise chunks (see Section 3.2). We use the remaining 14 rows for prefetching, since the cores work on overlapping image stripes of 48 rows height, and these stripes are spaced 14 pixels apart.

Two *descriptor feed* units fetch the image patches requested by the *descriptor calculation* part of the *SKB cores* from the correct image pyramid level. Out of the eight scales that are used in the interest point detection, only the scales 2 - 7 can actually have interest points due to the non-maximum suppression (NMS). Since always the nearest mipmap is selected, these scales all fall onto the second and third pyramid levels. Therefore, the *descriptor feed* units only need access to these two levels. The sliding window comprises of 56 rows on the first, and 48 on the second level. This is enough to accommodate the largest descriptor that is supported ($\sim$30 pixels), plus a margin large enough such that the requested descriptor patches do not fall outside of the sliding window[5].

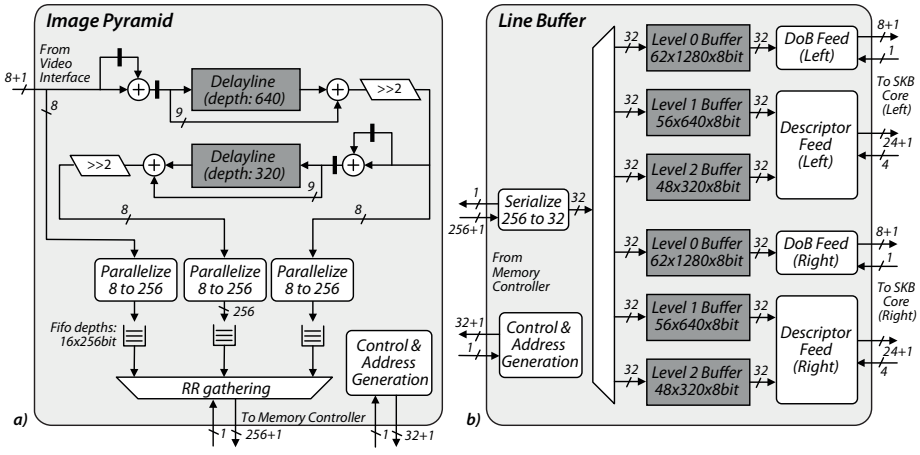

Fig. 4. Details of the image pyramid a), and line buffer blocks of the feature matching system b)

3.2 SKB Core

A top-level diagram of the *SKB core* architecture is shown in Figure 5. It is composed of two main blocks which perform the interest point detection and the descriptor calculation. The *interest point detection* unit performs a dense scan over the whole image and is constantly supplied with image data by the *DoB feed* unit of the *line buffer*. The detected interest points are temporarily stored in a FIFO before they are fetched by the *Descriptor Calculation Units* (*DCUs*). Note that the interest points are distributed sparsely over the whole image. The *DCU* has been designed to handle up to 12.5 k descriptors. Depending on the desired descriptor throughput, several instances can be

[5] The descriptor calculation lags behind the interest point detection and thus it can happen that requested descriptor patches lie outside of the current sliding window if there are not enough rows in the buffer.

operated in parallel. The FIFO serves to compensate local variations in throughput. Based on the position and scale of a certain interest point, the *DCUs* request the corresponding image patch from the *descriptor feed* units in the *line buffer*. The resulting descriptors, their position, and their scale are then sent on to the *matching unit*.

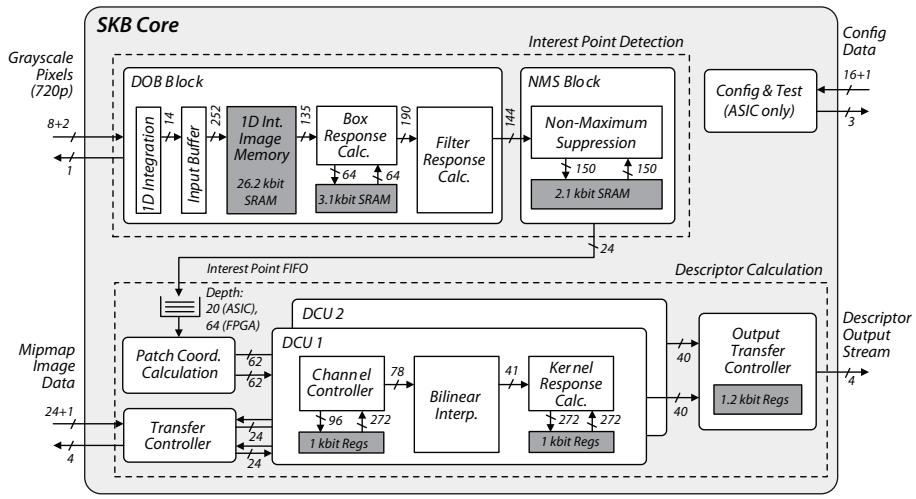

Fig. 5. Block diagram of the *SKB core* with two *descriptor calculation units*: The *SKB core* is supplied with raw 8 bit gray-scale image pixels. In a first step it searches for interest points within the image. In a second step, the surroundings of these points are described. For this description step, image patches around the interest points are transferred to the *SKB core* as well. The final descriptors are sent back to the *matching unit*.

Interest Point Detection A total of eight DoB responses have to be evaluated for each pixel – one for each scale in the scale space. Each DoB filter response can be decomposed into a linear combination of an inner and an outer box filter response. These box filter responses are usually calculated with the aid of a 2D integral image, which allows to compute the sum over an arbitrary rectangular area by accessing only four values in the integral image [24].

The integral image requires significant amounts of memory. While a conventional 720p image using 8 bit gray values requires 7.4 Mbit, the corresponding integral image requires large 28 bit entries which results in 25.8 Mbit. When the integral image is stored off-chip, this results in a large bandwidth as eight values have to be accessed per DoB filter response. For $N_{max} = 8$ scales and an effective image size of $x_{eff} \times y_{eff} = 1248 \times 688$ pixel this results in a bandwidth of $x_{eff} \times y_{eff} \times N_{max} \times 8 \times 28$ bit/frame $\approx$ 1.54 Gbit/frame. The effective image dimensions are given by $x_{eff} = x_{res} - 4 \times N_{max}$ and $y_{eff} = y_{res} - 4 \times N_{max}$, respectively.

One option to reduce this bandwidth is to transfer and locally store whole blocks of the integral image in order to leverage the spatial overlap among subsequent filters

[18]. In our implementation we use only a one-dimensional local integral image. Our approach builds on the observation in [9] where they show how a box filter response can be calculated recursively, provided that a dense scan is performed on the whole image. When scanning from left to right, a box filter response can be updated by adding the new pixels the box covers on the right side, and subtracting the pixels that are no longer covered by the box on the left side (Figure 6a).

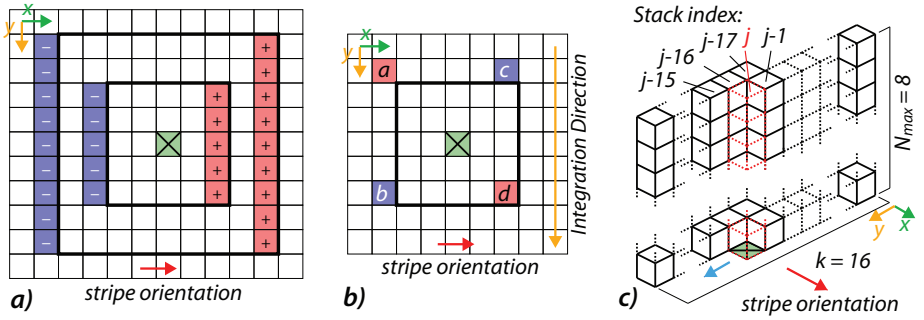

Fig. 6. Interest point detection details: a) Recursive DoB filter calculation for the green pixel position. b) 1D integral image box response update. c) NMS evaluation: the most recently added stack has index j.

However, the number of additions is still linearly dependent on the filter size. In our architecture, we additionally make use of the observation that the pixel groups that have to be added or subtracted are always continuous pixel columns. It is therefore possible to use a *one-dimensional integral* image which enables the calculation of 1D-sums of arbitrary length along the columns in constant time (two memory accesses and one subtraction). This allows us to update a box response by accessing only four corner values (a, b, c, d) as shown in Figure 6:

$$B_i = B_{i-1} - (b-a) + (d-c) = B_{i-1} + (a-c) + (d-b). \tag{5}$$

The terms can be reordered such that only differences between two values in the same row need to be added. Note that the one-dimensional integral image can be easily constructed locally as the integration direction is orthogonal to the scanning direction – i.e. if the image is processed in stripes of a certain height h, the integration amounts to the addition of h values, and is completely independent of the *width* of an image. This enables a hardware architecture that only needs to store a sliding window of the original image in a *line buffer*, and the memory bandwidth can be reduced considerably compared to a naive implementation using a two-dimensional integral image.

During processing the image is scanned on all scales in parallel, as otherwise several scanning passes through the same image would be required. The *line buffer* contains sliding windows of the input images and supplies the *DoB* block in the *SKB core* with a constant stream of raw image data. The image is processed in overlapping stripes of height $h = 4 \times N_{max} + k$, where $4 \times N_{max} = 32$ is the minimum neighborhood required

for eight scales, and k is the number of effectively calculated box filter responses within one column of the stripe. In order to enable non-maximum suppression in a $3 \times 3 \times 3$ neighborhood, the inner part of evaluated responses of a stripe need to be overlapped by another two pixels, i.e., subsequent stripes have a relative offset of $k-2$ rows. A larger value of k reduces the overhead due to the overlap among subsequent stripes, but it also increases the size of the local integral image buffer. In our implementation we use a value of $k = 16$. This results in a total bandwidth of $(4 \times N_{max} + k) \times x_{res} \times \left\lceil 1 + \frac{y_{eff}-k}{k-2} \right\rceil \times 8\,\text{bit} \approx 24.1\,\text{Mbit}$ per frame, and the local integral image buffer has to hold at least $(4 \times N_{max} + k) \times (4 \times N_{max} + 1) = 1584$ entries.

A detailed block diagram of the *DoB* unit is shown in Figure 7. The input is supplied in column-major format, which enables simple 1D integration along the columns. Note that with $k = 16$, it is sufficient to transfer one pixel to the *SKB core* per cycle to reach a frame rate above 30 frames per second with a clock frequency of 100 MHz.

When $k = 16$, the 1D-integration of a column takes 48 cycles, and the downstream circuitry is designed to calculate all $N_{max} \times k = 128$ DoB responses during this time. Some of the inner and outer boxes of the DoB-filter across different scales coincide, such that only 12 out of the total 16 boxes have to be effectively evaluated in the case of eight scales. The *1D integral image memory* is organized as a ring buffer and can hold 48 rows each with a width of 34 pixels. The rows of this memory are segmented into nine dual-port memories to ensure a collision free, parallel access for the box filter response calculation. Note that, the values are accessed in such a way that the difference of two values in one row (see previous section) can be immediately calculated at the memory output, which reduces the multiplexing overhead of the subsequent logic.

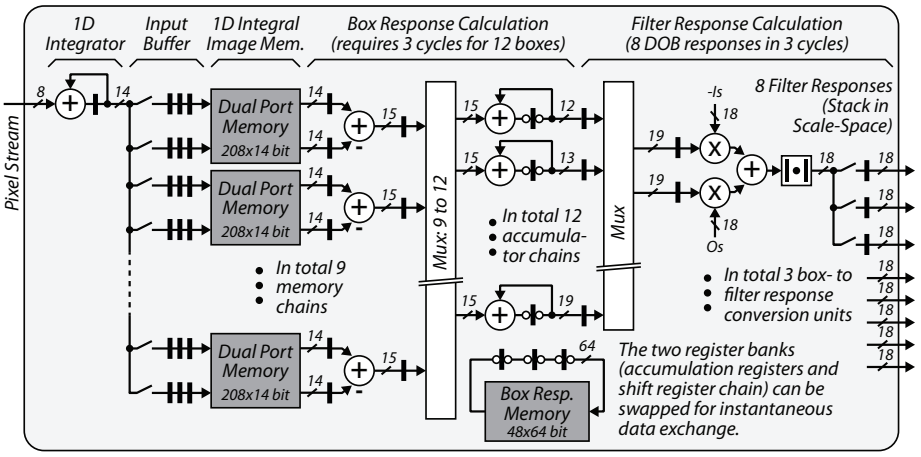

Fig. 7. The DoB filter block receives a dense pixel stream and calculates the DoB responses, which are output as stacks of 8 values in the scale space (j-th stack in Figure 6c). The constants I_n and O_n are the appropriate normalization factors for the inner and outer boxes of scale n.

The previous box filter responses for the recursive calculation are stored in another memory block. In one computation cycle, the $k = 16$ sets of twelve box responses are sequentially loaded, updated and stored again. For faster loading and storing, as illustrated in Figure 7, two interchangeable register banks are used: while one bank is in accumulator bank configuration, the other bank is organized as a shift register such that the intermediate values can be shifted to and from the memory. In three cycles, all 12 accumulators update their box response according the method presented in (5). Finally, the weighted sums among the intermediate box responses are formed to get the DoB response. For this purpose, three units are used which are able to calculate eight DoB responses in three cycles.

The detailed block diagram of the *NMS (Non Maximum Suppression) unit* is given in Figure 8. It receives the filter responses stack-wise from the *DoB unit* and performs NMS stack-wise on the local scale space volume. It is important to note that the $26 \times (N_{max} - 2)$ comparisons for one $3 \times 3 \times (N_{max} - 2)$ stack-neighborhood are not computed at once, but in a sequential manner as indicated in Figure 6c: the incoming stack j is compared to the upper left half of its neighborhood which has already been calculated, and the intermediate comparison results are stored. At the same time, the stack $j - 17$ is compared to the lower right half of its neighborhood, and these results are combined with the corresponding intermediate values in order to get the final result. Compared to a naive approach where the full neighborhood of a stack is stored in order to perform the NMS comparisons, this sequential approach requires to keep track of only the last $k + 1 = 17$ scale space stacks, including their intermediate comparison results. This is about $2\times$ less memory, since the intermediate results amount to only 6 bits per stack (maxima do not occur on the lowest and highest scale). After suppressing the non-maximum responses, the weak response t_{wk} threshold is applied to the remaining interest point candidates. Finally, the coordinates of the points passing this test are written to the interest point FIFO. We found that a depth of 64 works well in this setting.

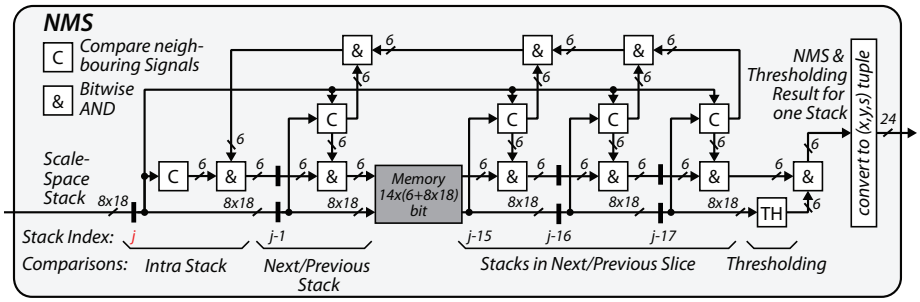

Fig. 8. The *NMS* block receives a stack of eight DoB responses per pixel position (j-th stack in Figure 6c), performs the NMS, and applies the weak response threshold. The 8 DoB responses of a single stack are stored in 8×18 bit wide registers and the intermediate comparison results in 6 bit wide registers. Overall, a local history of the last 17 scale space stacks is stored.

Interest Point Description In contrast to the interest point detection, the descriptor calculation operates on sparse data. It has to operate fast enough such that – on average – it is able to process all interest points in an image. Our evaluations have shown that several thousand interest points per frame are detected when setting the weak response threshold within a reasonable range of $t_{wk} \in [0.1, 1.0]$. The *interest point description* block is designed to be scalable and consists of several parallel *Descriptor Calculation Units* (DCUs) which can process up to 12.5 k descriptors per frame each. If the application requires a high throughput, this can be easily achieved by instantiating the appropriate amount of *DCUs* and adjusting the bandwidth of the image memory accordingly. The current implementation uses two *DCUs* which results in an aggregated throughput of 25 k descriptors per frame.

The *interest point description* block takes interest points from the FIFO buffer and assigns each one of them to a *DCU*, which then acquires the required data from the nearest mipmap level in the *line buffer* through the *transfer controller*. Then the received data is interpolated to a normalized 16×16 image patch which is convolved with several filter kernels and the responses are binarized using a threshold before the result is written to the output buffer. These steps are described in more detail below.

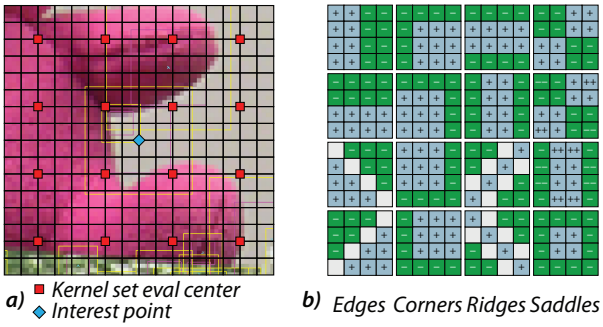

Fig. 9. Descriptor details: a) The descriptor support region is overlaid over an image patch showing the centers around which the set of semantic kernels depicted in b) is evaluated.

The interest points are read from the FIFO and the *patch coordinate calculation* block determines the parameters for the bilinear interpolator, the mipmap level and the coordinates and dimensions of the patch to be fetched from that mipmap level. The *patch coordinate calculation* unit then dispatches this information together with the interest point to an available *DCU*.

The mipmap is selected according to the scale parameter s of the interest points. A strategy that always selects the lower mipmap level would ensure that no information is lost, since then the image patches would always be downscaled. In our implementation, we use a different strategy that always selects the nearest mipmap (in scale). This has the benefit that the amount of data to be transferred to the bilinear interpolators is reduced by around a factor of two when compared to the first strategy. And although some information is lost since some patches have to be magnified, we found that the matching

performance does not degrade significantly. Moreover, this strategy causes the DCUs to never access the lowest mipmap level (corresponding to the original image). This does not imply that the lowest mipmap level does not have to be stored – it is still needed by the *DoB unit* – but it does not have to be accessible by the DCUs, and the size of the sliding window buffer can be set to the minimum size dictated by the *DoB unit*.

Each *DCU* has a local pixel buffer that temporarily stores the pixel data used by the interpolator. This reduces the required throughput of the interface significantly – in some cases certain pixels are used up to nine times within a few cycles. Whenever there is enough free space in the buffer, the *DCU* requests another two lines of the mipmap image patch from the *transfer controller*. The *transfer controller* acknowledges the requests whenever it has the capacity, and passes the request on to the *line buffer* while still receiving the response to an earlier request (this overlapping of requests ensures a high utilization of the interface bandwidth). A detailed block diagram of the datapath of one *DCU* is shown in Figure 10. First, the acquired patch from the nearest mipmap is resampled with a normalization factor in $[0.75, 1.5)$ using bilinear interpolation to complete the normalization of the support region. In a second step, the resulting row-wise stream of single pixels of the resulting 16×16 support region (Figure 9a) is convolved with the 16 semantic filter kernels shown in Figure 9b). Each *DCU* is able to process one interpolated pixel per cycle, which involves the evaluation of 16 kernel updates. Since the normalized image patch is processed in scan line order, it is necessary to keep track of four sets of 16 temporary kernel responses. Whenever the convolution of a set of filters is completed, it is binarized using a threshold, and the resulting part of the descriptor is written to the *output buffer*.

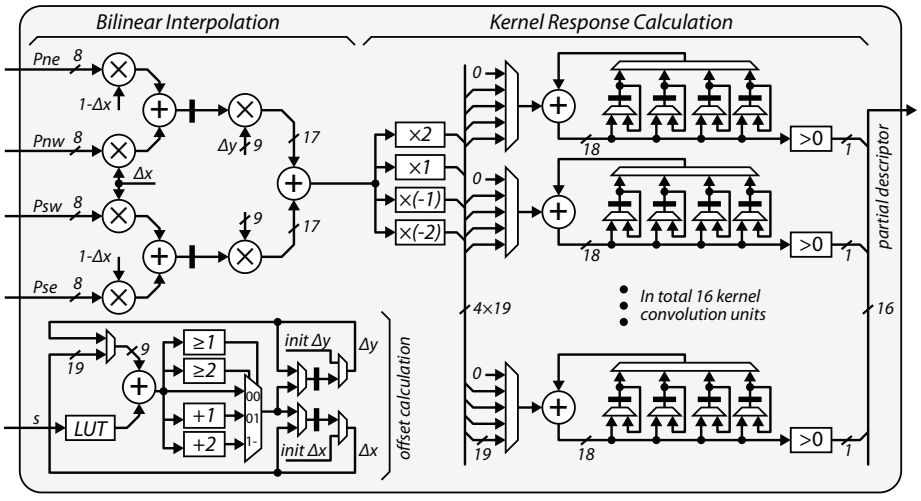

Fig. 10. The descriptor calculation unit (shown without the channel controller) contains a bilinear interpolator and 16 kernel response units. Note that the weights of the semantic kernels can be implemented without multipliers.

The raw descriptor calculation throughput of one *DCU* is one descriptor in 256 cycles which translates into around 12.5 k descriptors per frame (slightly less than 13 k due to control overhead). However this assumes that image data is always present for the bilinear interpolation. This is not always the case since the size of the image patches (on the mipmap levels) that are processed vary up from 16×16 to 25×25 pixels (a factor of 2.44 in size). The effective throughput may thus be dependent on the speed of this interface. In our implementation, the interface can deliver 24×100 Mbit/s of pixel data, which depending on the patch size corresponds to between 15 k and 38 k image patches in the worst and best case including control overhead. This rate is sufficient to supply one *DCU* continuously. However, to cope with feature point clusters, it is important to have a higher throughput than what the average case suggests. This is necessary to keep the interest point FIFO size within reasonable limits. If the FIFO is too small, some of the interest points can be dropped when the density of interest points gets too large. In our design we use two *DCUs* which together are able to process at least 15 k descriptors in the I/O-limited case, and up to 25 k in the computationally limited case.

Output Interface The output interface transmits each interest point together with its descriptor, i.e., the 24 bit triplet (x, y, s) and the 256 bit descriptor as a data packet over a 4 bit wide bus[6] requiring 71 cycles. A *DCU* requires at least 256 cycles to calculate a descriptor. The output interface guarantees that all results can be transmitted within these 256 cycles. Each *DCU* contains an output buffer that can store two finished descriptors in order to bridge time periods where the *output transfer controller* is busy transmitting a descriptor from a different *DCU*.

3.3 Descriptor Matching

In this system, we are matching interest points from the left image to the right image with a variant of the greedy, windowed nearest neighbour search explained earlier in Section 2.3. Since the matching process is basically an exhaustive search within the matching window, it is crucial to sort the descriptors from the right image appropriately. Otherwise the whole memory needs to be scanned for each point from the left image. In software, this is often done by using a range-tree variant (e.g. k-d tree). Here we use a different, hardware friendlier approach similar to a direct mapped cache which has a much simpler data structure management.

Interest Point Sorting and Matching In the worst case our right image will contain 25 k interest points, which is very sparse and corresponds to less than 3% of the number of pixels for a 720p image. In order to reduce both the memory and the search overhead, we have decided to use a binning method where we subdivide the entire image into uniformly sized bins. Each bin is allowed to store a small number of descriptors. In our implementation, after an exhaustive evaluation, we have decided to use a bin size of 4×32 pixels, with eight descriptor slots per bin. Assuming 25 k uniformly distributed

[6] This particular organization is a left-over from the initial part of the project where the *SKB core* was implemented in an external ASIC which had I/O constraints.

descriptors per image, the average number of descriptors per bin is around 3.3. We use more slots per bin since sometimes clusters of interest points occur, and this may lead to a bin overflow. Our evaluations have shown that when using eight descriptor slots, the percentage of dropped descriptors is usually low and around 2%.

In order to match a descriptor from the left image, only the bins covered by the matching window have to be accessed, as illustrated in Figure 11. When determining the size of the matching window for the nearest neighbour search, the setup of the stereo video system has to be taken into account. The exact matching window size is dependent on the geometric setup of the two cameras. Our implementation is able to handle a maximum window size of $(\Delta y, \Delta x1, \Delta x2) = (15, 31, 255)$ which works fine for most stereo setups.

Since the calculated descriptors follow a scan-line pattern from the top-left to the bottom-right of the image[7], it is possible to use a two-dimensional cache of bins. Considering the interest point detection step-size of 14 pixels and a maximum matching window height of 30 pixels, the cache should at least span $30 + 2 \times 14 = 58$ pixels in y dimension (14 pixels to write the descriptors from the current stripe, and 14 pixels for the stripe being matched). This value is rounded up to 64 rows in our implementation.

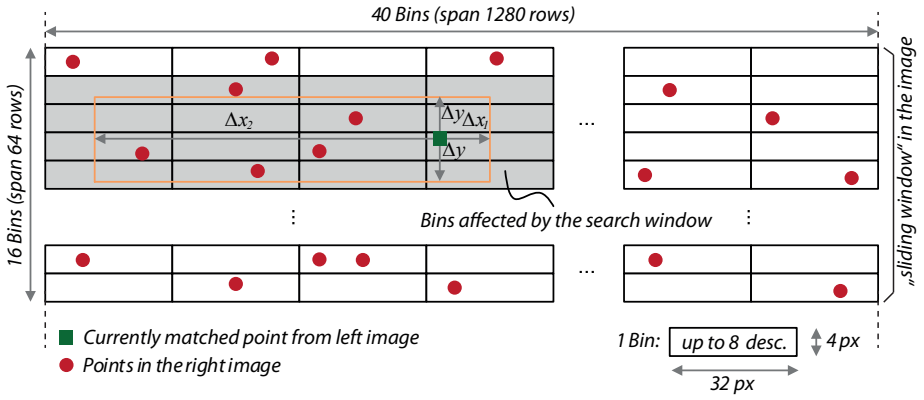

Fig. 11. The image is divided into bins of 4×32 pixels. Each such bin can hold a maximum of eight interest points. A sliding search window (in gray) is used to detect a match of an interest point in the left image (green) to possible candidates in the right image (red).

Matching Block Details A block diagram of the *feature matching unit* is shown in Figure 12. The *feature matching* unit consists of a descriptor FIFO for the left image, a binning memory for descriptors from the right image, a valid bit memory for the bins, a *compare unit* and two control units that control the binning and matching processes.

[7] The calculated descriptors do not necessarily follow a strict order due to the parallel *DCU*s, and since the interest point detection works column-wise in a narrow strip.

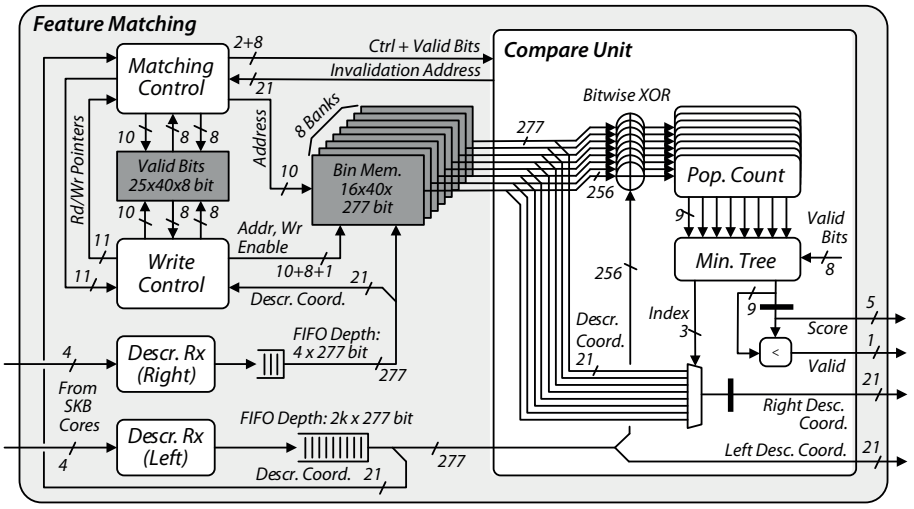

Fig. 12. The *feature matching unit* contains a large descriptor FIFO for the left image, a binning memory with associated valid bits for the right image, and a *compare unit* that performs 8 descriptor comparisons in parallel.

The *write control* unit contains a counter that keeps track of the sliding window position in the image. The coordinates of the incoming descriptors are checked against this position, and if the point lies within the sliding window, the *write control* unit checks the valid bits of the corresponding bin (there are eight valid bits per bin, one for each descriptor slot). If there is a free slot, the descriptor is written to that bin. Otherwise it is discarded. If the descriptor lies below the sliding window, the uppermost row of the sliding window is invalidated, moved to the bottom, and the row counter is incremented. This procedure is repeated until the descriptor coordinates lie within the sliding window.

The *matching control* unit fetches the coordinate from the left-image descriptor currently present at the output of the FIFO, and based on the sliding window parameters, it determines which bins have to be accessed. The depth of the descriptor FIFO for the left image has been set to 2 k which is the expected number of descriptors in a 64×1280 pixel window in the image (assuming 25 k uniformly distributed descriptors). The required throughput of the *feature matching* block in terms of comparisons per second (cps) is given by the expected number of descriptors in the left image and the expected number of descriptors in the matching window in the right image:

$$25\,\mathrm{k} \times 25\,\mathrm{k} \times \frac{30 \times (31+255)}{720 \times 1280} \times 30\,\mathrm{fps} = 1.74\,\mathrm{Mcps}. \tag{6}$$

Assuming a clock frequency of 100 MHz, this corresponds to a throughput of around 1.74 comparisons per cycle. But since the descriptors can be clustered, it is necessary to provide enough throughput margin such that the *feature matching* block does not start lagging behind. In our implementation, we access one whole bin in parallel and perform 8 comparisons per cycle in order to stay on the safe side. These descriptors

are sent to the *compare unit*, where the Hamming distances between the left image descriptor and all right image descriptors is calculated. Next, the index of the descriptor with the smallest Hamming distance is determined. After all bins have been loaded and compared, the coordinates of the descriptor pair with the smallest Hamming distance is output – given that the Hamming distance is below the matching threshold t_{match}. The valid bit corresponding to the right descriptor is then cleared.

4 Results

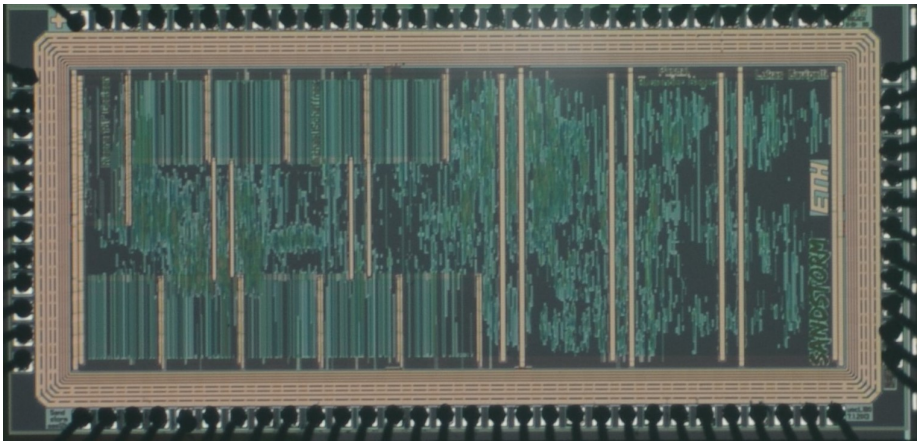

Fig. 13. Microphotograph of the SANDSTORM chip

4.1 ASIC Implementation of the Core

As mentioned earlier, in the initial phase of the project, the *SKB core* for one view has been prototyped on an ASIC which was named SANDSTORM and which has been fabricated in 180 nm CMOS technology (a die photograph is shown in Figure 13). Table 1 shows the key figures. At 100 MHz, one core is able to process a 720p video stream at 30 fps with 15 k-25 k descriptors per frame (depending on the distribution of the descriptor scales). As opposed to the FPGA implementation, the ASIC has a shorter interest point FIFO (20 instead of 64), does not automatically adjust the weak response threshold, and it contains an additional configuration block. This block allows to adjust parameters such as the weak response threshold, and to read out statistical information on detected and dropped interest points, as well as memory BIST results.

For comparison, Table 1 also lists the specifications of the BRIEF implementation of J. S. Park, *et al.* [16]. Note that their work has been designed for an object recognition application with different throughput requirements (1080p images, 512 descriptors per frame). Further, their implementation also contains the matching part and a large descriptor buffer (4096 entries) for testing purposes.

Table 1. Measurement results of the SANDSTORM chip. The results from J. S. Park, *et al.* [16] are also listed as a reference. *The abbreviations IPD, FE and FM stand for *Interest Point Detection*, *Feature Extraction* and *Feature Matching*.

Physical Characteristics	this work	J. S. Park, *et al.* [16]
Technology	UMC 180 nm (1P6M)	130 nm (1P6M)
Core Voltage	1.8 V	1.2 V
Package	CQFP 120	?
# pads	82 (I:40, O: 26, PWR: 16)	?
Core Area	3.08 mm^2	10.24 mm^2
Complexity (with SRAM)	254 kGE (2.4 mm^2)	861 kGE
Logic (std. cells)	193 kGE (1.8 mm^2)	78.3 kGE
On-chip SRAM	29 kbit	1024 kbit
Operating Frequency	100 MHz	100 MHz
Power Dissipation	146 mW (core), 38 mW (pads)	182 mW
Performance		
Functionality	IPD + FE*	IPD + FE + FM*
Throughput (single frames)	30 fps (720p)	94.3 fps (1080p)
Max. Desc./Frame	15 k-25 k	512

4.2 FPGA Implementation of the System

The synthesis results for an Altera Stratix IV EP4SGX530KH40C2 of our feature extraction and matching system are given in Table 2 (without memory controller, video and Ethernet interface). On this FPGA, the system runs with up to 142 MHz, delivering a throughput of 42 fps with 15 k-25 k descriptors per frame (depending on the distribution of the descriptor scales).

The results of the state-of-the art feature extraction and matching system of J. Wang, *et al.* [25] is also shown on Table 2 for comparison. Their system is based on BRIEF, and also works on 720p images. The difference is that they work on single views at 60 fps, with 2 k descriptors per frame (the descriptors are matched between subsequent frames). Their implementation uses about the same amount of DSP slices and on-chip memory bits, and uses about 2× less logic resources and around 3× less registers. But the fact that our system has a much higher descriptor extraction and matching throughput puts this into perspective.

5 Conclusions

First and foremost, we have shown that SKB outperforms all other evaluated descriptors on stereo content, and is thus a prime choice for efficient hardware accelerators. The presented system is able to extract and match SKB features from 720p stereo video in real-time, with 15 k-25 k descriptors per frame. We have used several innovations to

allow on-the-fly computation, and have reduced the amount of intermediate data storage and necessary I/O bandwidth without compromising the detection quality. Our system was designed for an application where a large amount of matched interest points will be needed. The system still remains competitive when compared to other state-of-the-art implementations, even though it is able to extract and match nearly an order of magnitude more interest points.

Table 2. Key figures of the FPGA implementation of our feature extraction and matching system (without memory controller, video and Ethernet interface). *This is for the worst case corner (slow 85C model). The pyramid calculation block runs on the pixel clock of the video interface (72.5 MHz). The synthesis results from J. Wang, *et al.* are also listed for comparison.

Physical Characteristics	this work	J. Wang, *et al.* [25]
Target FPGA	Altera Stratix IV (EP4SGX530KH40C2)	Xilinx Virtex 5 (XC5VLX110T)
Maximum Clock Frequency	142 MHz*	159 MHz
LUTs	33.6 k/425 k (7.9%)	17 k/69 k (25%)
Registers	30.7 k/425 k (7.2%)	11.5 k/69 k (17%)
Memory bits	4.15 Mbit/21 Mbit (19.5%)	4.6 Mbit/5.33 Mbit (86%)
DSPs	64/1024 (6.25%)	52/64 (81%)
Performance		
Functionality	IPD + FE + FM	IPD + FE + FM
Throughput (720p)	42 fps (stereo)	60 fps (single)
Desc./Frame	15 k-25 k	2 k

References

1. OpenCV Documentation (2014), `http://docs.opencv.org/`, Accessed May 2014
2. Agrawal, M., Konolige, K., Blas, M.: CenSurE: Center Surround Extremas for Realtime Feature Detection and Matching. In: Forsyth, D., Torr, P., Zisserman, A. (eds.) Computer Vision ECCV 2008, Lecture Notes in Computer Science, vol. 5305, pp. 102–115. Springer Berlin Heidelberg (2008)
3. Akenine-Möller, T., Haines, E., Hoffman, N.: Real-Time Rendering. AK Peters (2008)
4. Alahi, A., Ortiz, R., Vandergheynst, P.: FREAK: Fast Retina Keypoint. In: IEEE Conference on Computer Vision and Pattern Recognition. pp. 510–517 (2012)
5. Bay, H., Tuytelaars, T., Van Gool, L.: SURF: Speeded Up Robust Features. In: Leonardis, A., Bischof, H., Pinz, A. (eds.) Computer Vision ECCV 2006, Lecture Notes in Computer Science, vol. 3951, pp. 404–417. Springer Berlin Heidelberg (2006)
6. Bonato, V., Marques, E., Constantinides, G.: A Parallel Hardware Architecture for Scale and Rotation Invariant Feature Detection. IEEE Transactions on Circuits and Systems for Video Technology 18(12), 1703–1712 (2008)

7. Bouris, D., Nikitakis, A., Walters, J.: Fast and Efficient FPGA-Based Feature Detection Employing the SURF Algorithm. In: IEEE Annual International Symposium on Field-Programmable Custom Computing Machines. pp. 3–10 (2010)
8. Calonder, M., Lepetit, V., Ozuysal, M., et al.: BRIEF: Computing a Local Binary Descriptor Very Fast. IEEE Transactions on Pattern Analysis and Machine Intelligence 34(7), 1281–1298 (2012)
9. Ebrahimi, M., Mayol-Cuevas, W.: SUSurE: Speeded Up Surround Extrema Feature Detector and Descriptor for Realtime Applications. In: Workshop on Feature Detectors and Descriptors: The State Of The Art and Beyond" as part of IEEE Conference CVPR (June 2009)
10. Hartley, R., Zisserman, A.: Multiple View Geometry. Cambridge University Press, 2nd edn. (2003), ISBN-13: 978-0-521540-051-3
11. Hirschmüller, H., Scharstein, D.: Evaluation of Cost Functions for Stereo Matching. In: IEEE Conference on Computer Vision and Pattern Recognition. pp. 1–8 (2007)
12. Jeon, D., Kim, Y., Lee, I., et al.: A 470 mV 2.7 mW Feature Extraction-Accelerator for Micro-Autonomous Vehicle Navigation in 28 nm CMOS. In: IEEE International Solid-State Circuits Conference Digest of Technical Papers. pp. 166–167 (2013)
13. Leutenegger, S., Chli, M., Siegwart, R.: BRISK: Binary Robust Invariant Scalable Keypoints. In: IEEE International Conference on Computer Vision. pp. 2548–2555 (2011)
14. Lowe, D.: Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision 60(2), 91–110 (2004)
15. Mikolajczyk, K., Schmid, C.: A Performance Evaluation of Local Descriptors. IEEE transactions on Pattern Analysis and Machine Intelligence pp. 1615–1630 (2005)
16. Park, J.S., Kim, H.E., Kim, L.S.: A 182 mW 94.3 f/s in Full HD Pattern-Matching Based Image Recognition Accelerator for an Embedded Vision System in 0.13-CMOS Technology. IEEE Transactions on Circuits and Systems for Video Technology 23(5), 832–845 (2013)
17. Rosten, E., Porter, R., Drummond, T.: Faster and Better: A Machine Learning Approach to Corner Detection. IEEE Transactions on Pattern Analysis and Machine Intelligence 32(1), 105–119 (Jan 2010)
18. Schaeferling, M., Kiefer, G.: Object Recognition on a Chip: A Complete SURF-Based System on a Single FPGA. In: International Conference on Reconfigurable Computing and FPGAs. pp. 49–54 (2011)
19. Schaffner, M., Hager, P., Cavigelli, L., et al.: A Real-Time 720p Feature Extraction Core Based on Semantic Kernels Binarized. In: IFIP/IEEE 21st International Conference on Very Large Scale Integration. pp. 27–32 (Oct 2013)
20. Scharstein, D., Szeliski, R.: High-Accuracy Stereo Depth Maps using Structured Light. In: IEEE Conference on Computer Vision and Pattern Recognition. vol. 1, pp. I–195 (2003)
21. Sledevic, T., Serackis, A.: SURF Algorithm Implementation on FPGA. In: Biennial Baltic Electronics Conference. pp. 291–294 (2012)
22. Stefanoski, N., Wang, O., Lang, M., et al.: Automatic View Synthesis by Image-Domain-Warping. IEEE Transactions on Image Processing 22(9), 3329–3341 (Sept 2013)
23. Svab, J., Krajnik, T., Faigl, J., et al.: FPGA based Speeded Up Robust Features. In: IEEE International Conference on Technologies for Practical Robot Applications. pp. 35–41 (2009)
24. Viola, P., Jones, M.: Rapid Object Detection using a Boosted Cascade of Simple Features. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. vol. 1, pp. I–511–I–518 vol.1 (2001)
25. Wang, J., Zhong, S., Yan, L., et al.: An Embedded System-on-Chip Architecture for Real-time Visual Detection and Matching. IEEE Transactions on Circuits and Systems for Video Technology 24(3), 525–538 (March 2014)
26. Zilly, F., Riechert, C., Eisert, P., et al.: Semantic Kernels Binarized - A Feature Descriptor for Fast and Robust Matching. In: Conference for Visual Media Production. pp. 39 –48 (nov 2011)