High-Speed SIFT Descriptor Generation with 36 Small-Region Division and Logic-Synthesis Evaluation

Ayumu Mitsumoto Tetsuo Hironaka

Department of Computer and Network Engineering
Graduate School of Information Sciences, Hiroshima City University
3-4-1 Ozuka-Higashi, Asaminami-ku, Hiroshima-shi, Hiroshima 731-3194, Japan
mj66019@e.hiroshima-cu.ac.jp hironaka@hiroshima-cu.ac.jp

Abstract— SIFT's rotation operation is sequential and therefore becomes a bottleneck in hardware implementations. We divide the descriptor region into 36 small regions, compute orientation histograms independently, and sum them into 17 subregion histograms, enabling 36-way parallel processing. This enables fast execution and high matching accuracy under accuracy-oriented parameter settings. Furthermore, we present the RTL design of the feature descriptor generator and perform logic synthesis with FreePDK45, achieving up to 31.8× speedup compared with the method proposed in previous research.

I. Introduction

Feature extraction plays an important role in various fields such as autonomous driving, robot vision, and medical image processing. Among these, the Scale-Invariant Feature Transform (SIFT) [1] has been widely used as a feature extraction algorithm that provides scale invariance, rotation invariance, and high matching accuracy [4]. However, SIFT incurs high computational and memory costs, and in particular, the operation that rotates the region surrounding the keypoint to align with the direction of maximum intensity change is sequential, becoming a processing speed bottleneck in hardware implementations [2][9]. To address this issue, research aimed at accelerating SIFT feature descriptor generation has been progressing [10][3][7][8][5].

J. Jiang, X. Li, and G. Zhang [3] proposed an architecture that approximates the rotation operation by dividing the descriptor region into 17 subregions and using orientation histograms for each subregion. Although this method relaxed the sequential dependency of the rotation process, their design prioritized speed improvement, leading them to set the number of scales in the Gaussian pyramid, the Gaussian filter's standard deviation σ, and the descriptor region size to smaller values, which resulted in a significant drop in matching accuracy. A Gaussian pyramid is a multi-level set of images generated by stepwise smoothing with Gaussian filters, where a larger σ represents a coarser scale. Therefore, reducing the number of levels or the value of σ narrows the range of scale representation, and further shrinking the descriptor region decreases the amount of information in the descriptor, together causing accuracy degradation. In the present study, by reproducing and comparing J. Jiang, X. Li, and G. Zhang's architecture under parameter settings that emphasize accuracy, we confirmed that matching accuracy close to that of conventional SIFT can be achieved. However, due to the nature of the process in which pixel gradient information is accumulated into orientation-specific histograms for each subregion, the expansion of the descriptor region under these settings makes high-speed processing difficult when employing their sequential histogram-addition method.

B. Liu et al. [5] proposed an architecture that adopts a polar-coordinate-based circular descriptor region, aiming primarily at optimizing energy efficiency for high-frame-rate scenarios. In their method, the circular descriptor region is finely divided into fan-shaped segments, which are then integrated and reordered according to the main orientation to achieve rotation invariance. Furthermore, they reduce the dimensionality of the output feature vector to improve energy efficiency, and they also confirmed that matching accuracy is significantly improved compared with the work of J. Jiang, X. Li, and G. Zhang. However, according to their evaluation results, the matching accuracy fluctuates considerably under certain conditions such as scaling, suggesting that robustness of the features remains an issue. Consequently, their method also faces a similar challenge to that of Jiang et al., namely that when parameter settings prioritize accuracy, high-speed processing becomes difficult.

To overcome this challenge, we propose an architecture that increases processing parallelism by dividing the descriptor region into 36 small regions, thereby achieving fast feature descriptor processing while maintaining accuracy. Each small region independently computes an orientation histogram, and these histograms are then summed and reordered to form the 17 subregion histograms, enabling processing with up to 36 parallel units while approximating the rotation operation. This allows for both fast execution and high matching accuracy even under

parameter settings that prioritize accuracy. Furthermore, we provide a detailed RTL design of the feature descriptor generator, which is significantly affected by the expansion of the descriptor region within the proposed architecture, and, through timing analysis using a logic synthesis tool, quantitatively evaluate the acceleration effect of the proposed architecture in a hardware implementation.

The rest of this paper is organized as follows. Section II explains the algorithm of SIFT feature descriptor processing, which forms the basis of the proposed architecture. Section III describes the overall structure of the proposed feature descriptor processing architecture and the parallelization method by small-region division. Section IV presents the detailed design of the feature descriptor generator. Section V reports the evaluation results in terms of accuracy as well as speed, area, and power based on logic synthesis. Finally, Section VI concludes this paper.

II. Algorithm of SIFT Feature Descriptor Processing

SIFT feature descriptor processing mainly consists of three steps: main orientation estimation, descriptor generation, and normalization. In this section, we explain these three processes, which form the basis of the proposed architecture.

A. Main Orientation Estimation

In the main orientation estimation, the gradient information of the pixels around a keypoint is used to compute an orientation histogram, as shown in Fig. 1. The horizontal axis represents 36 bins obtained by dividing the orientation into 10° intervals, and the vertical axis represents the sum of gradient magnitudes accumulated in each bin. Here, "orientation histogram" denotes a histogram of gradient orientations weighted by gradient magnitudes.

In this process, the local region around the keypoint is first convolved with a Gaussian kernel, and then the gradient magnitude of each pixel is added to the corresponding bin. The angle of the bin with the maximum value is determined as the main orientation, and in the subsequent processing steps, the descriptor region is rotated according to this orientation to achieve rotation invariance.

B. Feature Descriptor Generation

Next, in the normalized coordinate system based on the main orientation, the local region centered at the keypoint is divided into multiple subregions. As shown in Fig. 2(a), the local region is rotated according to the main orientation indicated by the arrow, and the rotated region is divided into 4×4 subregions. As illustrated in the transition from Fig. 2(b) to Fig. 2(c), the gradient magnitudes of pixels within each subregion are accumulated into the corresponding orientation bins of the histogram. Each

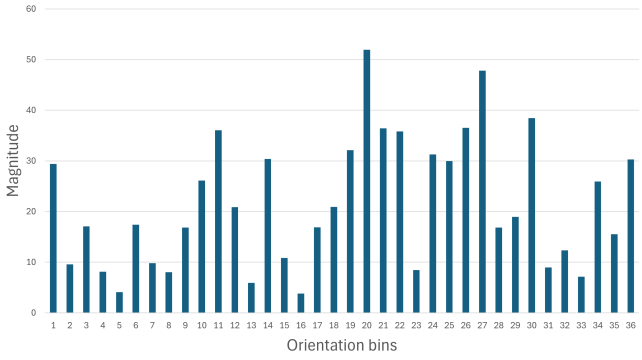

Fig. 1. Example of an orientation histogram for main orientation estimation.

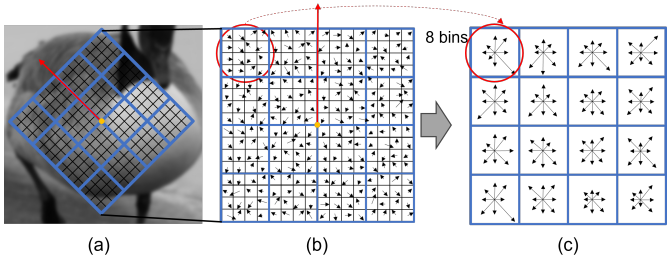

Fig. 2. Computation of orientation histograms from the descriptor region: (a) descriptor region rotated according to the main orientation. (b) gradient information of pixels within the descriptor region. (c) orientation histograms with 8 bins for each subregion.

subregion has an orientation histogram with 8 bins. In this process, the contribution of each pixel is weighted by a Gaussian window centered at the keypoint so that pixels closer to the center are given larger weights. Furthermore, for both the subregion to which a pixel belongs and its neighboring subregions, as well as the orientation bin and its neighboring bins, linear interpolation is performed in proportion to the distance and orientation difference. As a result, since the 4×4 subregions each have 8 bin values, a 128-dimensional feature vector is constructed. However, the rotation operation based on the main orientation can only be performed sequentially, which becomes an obstacle to acceleration in hardware implementations. In the next section, we propose a feature descriptor processing architecture that resolves this issue.

C. Normalization

Finally, the obtained feature vector is normalized to make it robust against illumination changes and outliers. First, the L_2 norm is computed using all elements, and each component is divided by this norm to normalize the entire feature vector. Next, to prevent abnormally large

components from dominating the descriptor, each component is clipped to 0.2 (after the first normalization), and then L_2 normalization is applied again.

Through the above processes, the SIFT algorithm describes features from the gradient information around a keypoint.

III. Proposed Feature Descriptor Processing Architecture

A. Overall Structure of the Proposed Architecture

The proposed architecture shown in Fig. 3 takes as input the gradient information of the descriptor region **grad** of size $M \times M$ and a **valid** signal indicating the start of processing, and outputs the 136-dimensional feature vector **feature_vec**. The squares in the figure represent functional blocks, each of which operates sequentially, starting after the preceding block has completed. When **valid** is high, processing begins: using the input gradients, **find_keypoint_orientation** estimates the main orientation of the keypoint, and **generate_subregion_histogram** computes the orientation histograms for the 17 subregions. Because these two processes have no data dependency, they operate in parallel.

The resulting histograms are then reordered by bin indices according to the main orientation via **reordering_histogram**, and the 16 subregion histograms (excluding the center) are reordered via **reordering_subregion**. This sequence of operations approximates the rotation operation.

Finally, **histogram_to_vector** applies L_2 normalization to the reordered histograms to generate **feature_vec**.

In this study, we focus on the **generate_subregion_histogram** process and introduce a method of subdividing the descriptor region into smaller regions to increase parallelism. The details of this small-region-based parallelization, used in **generate_subregion_histogram**, are described in the next subsection.

B. Parallelization Method by Subregion Division

As shown in Fig. 4, the proposed architecture divides the $M \times M$ descriptor region in (a) into 36 small regions in (b). Each small region has size $\lceil M/6 \rceil \times \lceil M/6 \rceil$, and the shaded portions in the figure indicate areas where multiple regions overlap. In the proposed architecture, each small region independently computes an orientation histogram, and by summing these histograms, the architecture constructs the orientation histograms for the 17 subregions. Specifically, by summing the eight-bin orientation histogram values computed in small regions ①, ②, ⑦, and ⑧ in Fig. 4(b), we obtain the histogram corresponding to subregion ⑦ in Fig. 4(a). Because there are no dependencies between small regions, processing can be performed in up to 36 parallel units, yielding a substantial acceleration.

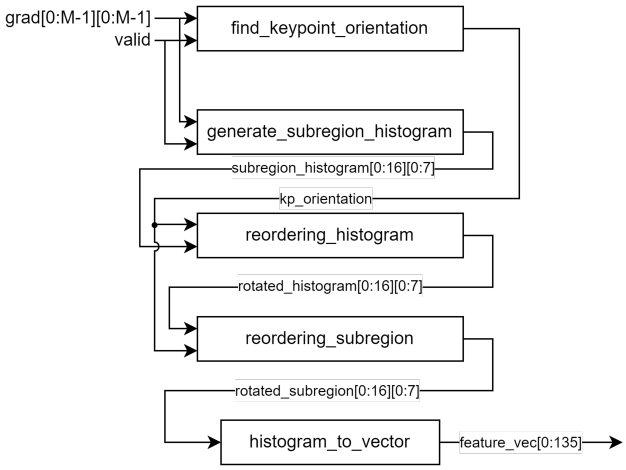

Fig. 3. Proposed Feature Descriptor Processing Architecture.

In the next section, we describe in further detail the architecture of the **generate_subregion_histogram** feature descriptor generator that incorporates the small-region division.

IV. Detailed Design of the Feature Descriptor Generator

A. Overall Structure of the Feature Descriptor Generator

Fig. 5 shows the architecture of the **generate_subregion_histogram** feature descriptor generator. It receives as input the gradient information **grad** of the $M \times M$ region around the keypoint and a **valid** signal, and outputs 17 orientation histograms, each with 8 orientation bins. First, the input **grad** is sliced by the **WindowSlice** block into regions of size $M/3 \times M/3$. Next, each sliced region is further divided, within the small-region computation block **calc_small_region**, into four overlapping small regions, and an orientation histogram is computed for each small region. Here, both **WindowSlice** and **calc_small_region** operate in nine-way parallel. The 36 small regions computed in nine-way parallel are then sent to the subregion accumulation block **SubregionAccumulator**.

The descriptor region is divided, as shown in Fig. 4(a), into 17 subregions: one central region and 16 peripheral regions that include overlap. In **SubregionAccumulator**, for each subregion, the corresponding four small-region histograms' bin values are summed. This series of operations yields the orientation histograms for the 17 subregions.

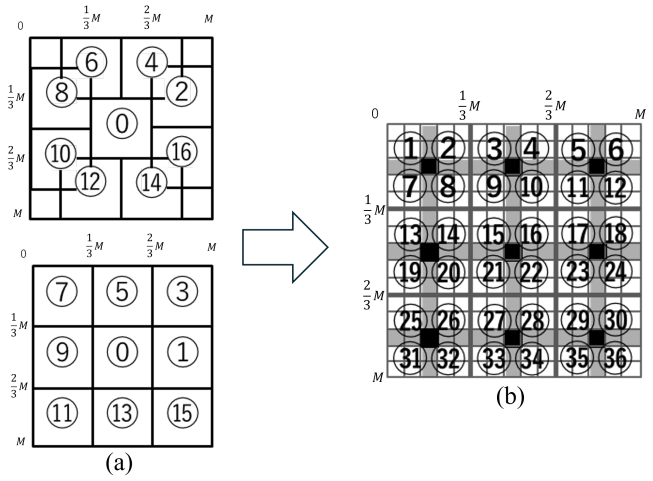

Fig. 4. Small-region division of the descriptor region. (a) Division of the $M \times M$ descriptor region into 17 subregions. (b) Division into 36 small regions.

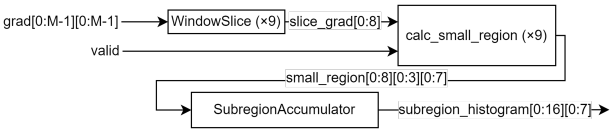

Fig. 5. Architecture of the **generate_subregion_histogram** block.

The following sections explain, in detail, the architectures of the small-region computation block **calc_small_region** and the subregion accumulation block **SubregionAccumulator**.

B. Small-Region Computation Block

Fig. 6 shows the architecture of the small-region computation block **calc_small_region**. This architecture comprises three processing stages: a signal-selection stage, a weighting stage, and a histogram-computation stage.

In the signal-selection stage, first, the input gradient information **grad** for the $M/3 \times M/3$ region is divided—still overlapping—into four small regions of size $\lceil M/6 \rceil \times \lceil M/6 \rceil$ by four **WindowSlice** units. Next, each divided data block is selected using a multiplexer (MUX) that takes as its control signal the coordinates of the small region. The small-region coordinate register **Coord** is controlled by the **valid** signal that indicates the start of processing, and the same coordinate values are used for all four small regions. Also, because **grad** comprises paired data of gradient magnitude and orientation, the **OriMagSplitter** splits it into gradient magnitude data **mag** and gradient orientation data **ori**.

In the weighting stage, the gradient magnitude **mag** is

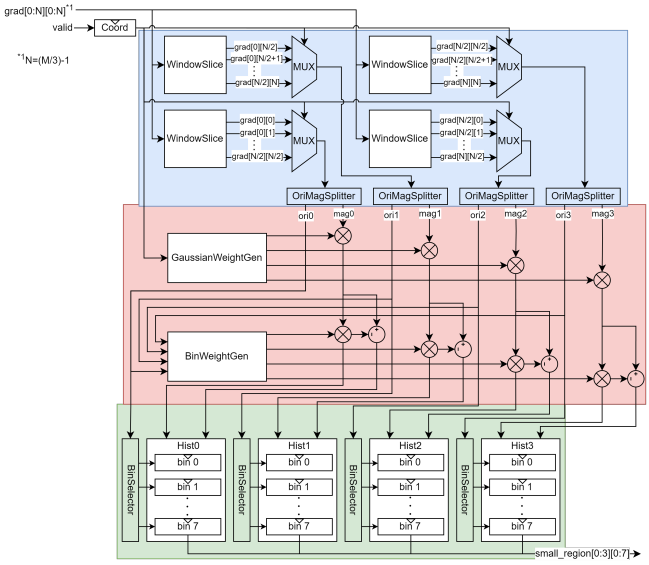

Fig. 6. Architecture of the **calc_small_region** block.

first weighted with Gaussian weights, and then weighted according to the distance between the gradient direction and the center of each bin.

In the histogram-computation stage, the weighted values from the weighting stage are added, via the **BinSelector**, to two active bins. Specifically, one bin receives the weighted value directly, and the other bin receives the weighted value subtracted from the pre-weighted value. By doing so, we reduce the number of multiplications by one. Through this sequence of operations, the orientation histograms for the four small regions are computed.

C. Subregion Accumulation Block

SubregionAccumulator consists of 17 identical instances of the block in Fig. 7, one per subregion. Fig. 7 illustrates the architecture that computes the orientation histogram of a subregion from four small regions. Each instance first receives as input the four small-region histograms **hist** corresponding to its target subregion. Within **Bin0_Slice** through **Bin7_Slice**, it extracts the bin values from those four histograms.

Next, it sums the values of the same bin across the four histograms for each of the eight orientations, and outputs the result as **subregion_histogram**. With 17 such instances in **SubregionAccumulator**, this process runs in 17-way parallel, thereby computing the orientation histograms for all 17 subregions simultaneously.

V. Evaluation

In this section, we evaluate the accuracy of the proposed architecture, as well as the cycle count, cell area, and

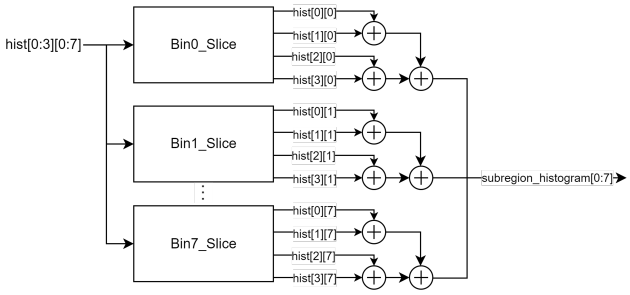

Fig. 7. Computation of a subregion orientation histogram from four small regions.

power consumption of the feature descriptor generator that incorporates small-region division. For the accuracy evaluation, we implemented both the conventional SIFT algorithm and the proposed architecture using floating-point arithmetic and compared their performance. Therefore, before presenting the matching accuracy results, we first describe the SIFT algorithm used for accuracy evaluation.

A. SIFT Algorithm Used for Accuracy Evaluation

SIFT feature extraction consists of two main processes: keypoint detection and feature descriptor generation. Among these, the keypoint detection algorithm and parameters were kept identical for both the baseline and the proposed architecture. In this study, to achieve stable and high-precision matching, we adopted the parameters reported by Lowe [1] that yield high matching accuracy: the number of octaves was set to 3, the number of scales to 6, the initial standard deviation to $\sigma_0 = 1.6$, and the increment factor to $k = 2^{1/3}$. The σ values for each scale are $\{\sigma_0, k\sigma_0, k^2\sigma_0, k^3\sigma_0, k^4\sigma_0, k^5\sigma_0\}$, and keypoints are detected at scales 1, 2, and 3 during Gaussian/DoG pyramid generation and extrema detection. In the Gaussian pyramid, the blurring range was set to $\pm 2.5\sigma$. For localization in extrema detection, the threshold was set to 0.015 to eliminate low-contrast keypoints with small DoG values, and the eigenvalue ratio r of the Hessian matrix was set to 10 to suppress edge responses. Note that subpixel position estimation was not included in this study.

For the feature descriptor generation process of the baseline SIFT used for comparison, we employed the same algorithm as described in Section II. Following the work of I. Rey-Otero and M. Delbracio [6], the Gaussian kernel window size was set to 6σ, and the side length of the descriptor region was set to $1.25 \times 6\sigma$. In our proposed architecture, for computational convenience while keeping the descriptor region size close to that of conventional SIFT, the side length was set to $\{27, 39, 51\}$ at scales 1, 2, and 3, respectively, which are multiples of 3 and odd numbers.

B. Matching Accuracy Evaluation

Fig. 8 shows a comparison of accuracy for each method. The top of the figure indicates the type of image and the applied transformation. The horizontal axis represents the strength of the transformation, and the vertical axis represents the matching accuracy between the original image (1) and the transformed image. In the bottom portion of the figure, "Original" denotes an implementation of conventional SIFT using the parameters adopted in this work, and "Proposed" denotes an implementation of the feature descriptor processing proposed in this study. In this evaluation, both methods were implemented in programs using 32-bit floating-point arithmetic, and features were extracted for assessment.

Feature similarity was computed using Euclidean distance, and correspondences were established using a threshold $k = 0.8$. Matching accuracy was measured on the Mikolajczyk dataset [4]. From Fig. 8, it can be confirmed that the accuracy of the Proposed method is significantly improved compared to the results reported by J. Jiang, X. Li, and G. Zhang. Moreover, although there is a tendency for the Proposed method's accuracy to drop sharply when the Original method's accuracy approaches 80%, overall the Proposed method achieves accuracy close to that of the Original.

As another example of rotation-approximation methods, the evaluation results reported by B. Liu et al. [5] for their polar-coordinate-based approach show significant fluctuations in matching accuracy under zoom + rotation and viewpoint change transformations. In contrast, our Proposed method maintains stable accuracy under the same transformations, demonstrating superior robustness in preserving rotation and scale invariance.

C. Logic Synthesis Evaluation of the Feature Descriptor Generator

Table I shows the results of logic synthesis for both the RTL of the proposed feature descriptor generator and the RTL reproducing the module corresponding to the feature descriptor generator of J. Jiang, X. Li, and G. Zhang's method [3], using the FreePDK45 (an open-source 45 nm process technology) and Synopsys Design Vision W-2024.09 with a target clock of 200 MHz.

In the reproduced implementation, processing an $M \times M$ region ($M = 27, 39, 51$) requires $\{732, 1524, 2604\}$ cycles. In contrast, in our architecture, the computation and integration of the 36 small regions complete in $\{26, 50, 82\}$ cycles, respectively. Therefore, the proposed method operates $\{28.2, 30.5, 31.8\}$ times faster than the reproduced method for each value of M. On the other hand, the cell area is approximately three times larger for each M, and the power consumption is nearly five times greater. These increases are believed to result from the larger total number of histograms and the increased number of additions and multiplications, as well as from non-optimized

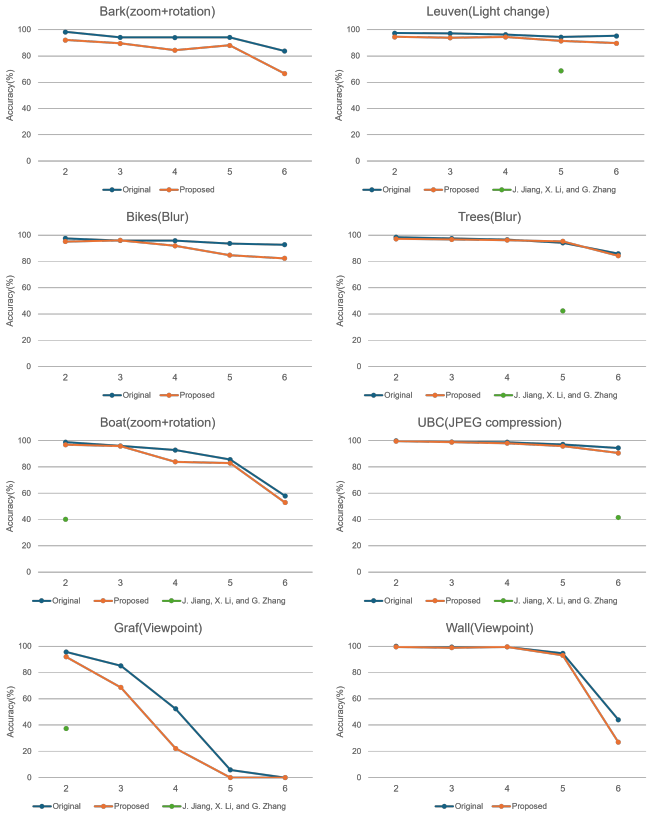

Fig. 8. Accuracy comparison of each method.

bit-width settings.

VI. Conclusion

In this study, to address the sequential nature of the rotation operation, which constitutes a bottleneck in SIFT feature descriptor processing, we proposed an architecture that divides the descriptor region into 36 small regions, independently computes orientation histograms in each small region, and then assembles the histograms of the 17 subregions. This structure enables processing in up to 36 parallel units and, even under parameter settings that prioritize accuracy, achieves both rapid processing and high matching accuracy. We performed a detailed RTL design of the feature descriptor generator, which is the core of the proposed method, and conducted logic synthesis using the FreePDK45, achieving a maximum speedup of 31.8×. Meanwhile, cell area increased by roughly 3× and power consumption by about 5×; these increases are primarily attributed to the larger number of histograms, the increase in arithmetic units, and insufficient bit-width optimization. Going forward, in addition to RTL implementation of the entire architecture, we will pursue further improvements in area efficiency and high throughput through bit-width optimization, memory resource reuse techniques, and arithmetic scheduling optimization.

TABLE I
Comparison of feature descriptor generators.

Method	M	**Power (mW)**	**Total Cell Area (μm^2)**
Reproduced [3]	27	18.33	133,307
	39	18.51	174,733
	51	18.27	227,912
Proposed	27	91.00	435,707
	39	79.88	492,989
	51	110.04	703,021

Acknowledgements

This work was also supported through the activities of VDEC, d.lab, The University of Tokyo, in collaboration with NIHON SYNOPSYS G.K.

References

[1] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vis.*, vol. 60, no. 2, pp. 91–110, 2004.

[2] F.-C. Huang, S.-Y. Huang, J.-W. Ker and Y.-C. Chen, "High-Performance SIFT Hardware Accelerator for Real-Time Image Feature Extraction," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 3, pp. 340–351, 2012.

[3] J. Jiang, X. Li and G. Zhang, "SIFT hardware implementation for real-time image feature extraction," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 24, no. 7, pp. 1209–1220, 2014.

[4] K. Mikolajczyk and C. Schmid, "A performance evaluation of local descriptors," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 10, pp. 1615–1630, 2005.

[5] B. Liu et al.,"An energy-efficient SIFT based feature extraction accelerator for high frame-rate video applications," *IEEE Trans. Circuits Syst. I*, vol. 69, no. 12, pp. 4930–4943, 2022.

[6] I. Rey-Otero and M. Delbracio,"Anatomy of the SIFT Method," *Image Processing On Line*, vol. 4, pp. 370–396, 2014.

[7] J. Vourvoulakis, J. Kalomiros, J. Lygouras,"Fully pipelined FPGA-based architecture for real-time SIFT extraction," *Microprocessors and Microsystems*, vol. 40, pp. 53–73, 2016.

[8] S. -A. Li, W. -Y. Wang, W. -Z. Pan, C. -C. J. Hsu and C. -K. Lu,"FPGA-Based Hardware Design for Scale-Invariant Feature Transform," *IEEE Access*, vol. 6, pp. 43850–43864, 2018.

[9] L. -C. Chiu, T. -S. Chang, J. -Y. Chen and N. Y. -C. Chang,"Fast SIFT Design for Real-Time Visual Feature Extraction," *IEEE Trans. Image Process.*, vol. 22, no. 8, pp. 3158–3167, 2013.

[10] L. Yao, H. Feng, Y. Zhu, Z. Jiang, D. Zhao and W. Feng,"An architecture of optimized SIFT feature detection for an FPGA implementation of an image matcher," *Proc. Int. Conf. Field-Programmable Technol.*, pp. 30–37, 2009.