A 42fps Full-HD ORB Feature Extraction Accelerator with Reduced Memory Overhead

Rongdi Sun*, Peilin Liu, Jun Wang, Cecil Accetti, Abid A. Naqvi
School of Electronic Information and Electrical Engineering
Shanghai Jiao Tong University
Shanghai, China
*Email: the.sun@sjtu.edu.cn

***Abstract*—This paper proposes a hardware accelerator of Oriented FAST and Rotated BRIEF (ORB) algorithm with its full implementation. The accelerator is optimized in computational logics and memory organizations to achieve real-time performance and good matching accuracy. The architecture consists of a loose-coupled pipeline, in which keypoints are detected in scale-level parallel while descriptors are built in task-level parallel. In this way, the workload in different pipeline stages are balanced. Because of the loose-coupled architecture employed, large amounts of image pyramid data are offloaded to external memory and fetched again. To relieve the external memory bandwidth requirement, image pyramid data are stored every other scale and a shared data-reuse structure is also introduced for patch loading. Besides, we design a score recorder to refine the keypoints in different regions adaptively. As a result, the relatively balanced distribution of keypoints in a frame is in favor of establishing keypoint correspondences. The proposed architecture is implemented on a Zynq-family FPGA with 100MHz and can extract 1000 features in full-HD images at 42fps. An external memory bandwidth of 1.96Gbps and internal memory of 583.4Kbits are suitable for practical embedded applications.**

Keywords*—*feature extraction; hardware accelerator; ORB

I. Introduction

Image feature extraction is a fundamental task in the field of computer vision. Recently, high-level features learned by deep neural network have gained great success in object recognition and detection. Still, handcrafted features are widely used in a variety of applications such as 3D reconstruction, visual odometry and simultaneous localization and mapping (SLAM), where feature correspondences are built in image sequences to generate a map. Scale-Invariant Feature Transform (SIFT) [1] is one of most prominent low-level features invariant to scale, rotation, noise and illumination. Likewise, Speeded-UP Robust Feature (SURF) [2] is proposed with lower computational complexity and keeps almost the same performance compared with SIFT. Both of them detect repeatable keypoints in scale space and generate descriptors based on Histogram of Gradients (HoG).

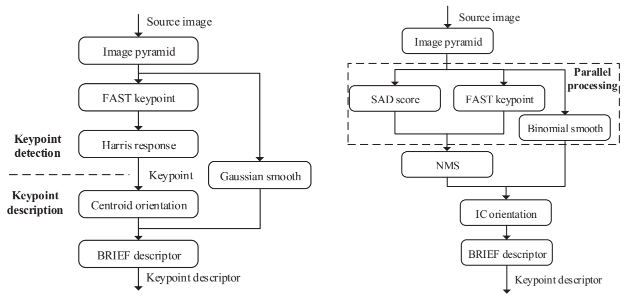
Despite high-quality features obtained by SIFT/SURF, it is difficult to implement them in real-time due to intensive computation and memory access. To address the problem, researchers have presented binary descriptors. Binary descriptors compare the intensity of pixel pairs with certain distribution in a local image patch, which cost less memory space and can be efficiently matched using XOR operations. A combination of Features from Accelerated Segment Test (FAST) [3] and Binary Robust Independent Elementary Features (BRIEF) [4] is widely used for keypoint detection and feature description respectively since it achieves comparable performance with SIFT. This solution is further refined as Oriented FAST and Rotated BRIEF (ORB) [5] to achieve scale and rotation invariance. ORB algorithm has been successfully applied in sparse visual SLAM system [6].

To develop vision applications on embedded platforms such as IoT devices, unmanned aerial vehicles and smart robots, many researchers have explored different architectures to perform binary feature extraction in real-time [7] [8] [9] [10]. In [7], an image recognition accelerator has been proposed, which shares a unified data path to perform FAST and BRIEF algorithms, but the time of pixel loading is not taken into account. In [8] [10], full-pipelined FAST and BRIEF architectures have been implemented on FPGA, yet they do not address scale and rotation invariance. To the best knowledge of us, the work in [9] is the first hardware solution based on ORB method. For ORB algorithm, both oFAST keypoint detection and rBrief descriptor generation access large amount of image data from multiple scales. However, the option that multi-scale images are stored on chip results in much resource overhead and also limits the scalability.

This paper presents a hardware architecture that efficiently implements real-time ORB feature extraction. Compared with previous works realized on FPGA, our contributions are:

- A loose-coupled architecture is proposed to balance the workload. It detects keypoints on multiple scales in parallel and computes a descriptor for each keypoint one by one.
- A score recorder is integrated with non-max suppression to filter low-quality keypoints and adaptively control the keypoint distribution over an image.
- Smoothed images of every other scale are stored to external memory, which reduces massive internal memory size and the access size of external memory as well.
- A shared data-reuse structure for both orientation estimation and descriptor generation is designed to further decrease the memory bandwidth requirement.

The rest of this paper is organized as follows: Section II reviews the ORB algorithm and introduces several hardware-



(a) The primitive ORB algorithm (b) The modified ORB algorithm

Fig. 1. The ORB algorithm flow and its optimization.

oriented optimizations. Section III presents the architecture details of the proposed accelerator. Implementation on FPGA and simulation experiments are demonstrated in Section IV. Section V gives a summary of our design and discuss the future work.

II. Review of ORB Algorithm

The ORB algorithm flow is shown in Fig.1a. It first detects FAST corners in multiple scales of the source image. Then non-max suppression (NMS) is performed to filter low-quality corners. After that the orientation of each keypoint is estimated by Intensity Centroid (IC) method. Finally descriptors are generated using BRIEF rotated by orientations. With some modification and reorder of the pipeline, the algorithm flow is optimized to improve processing speed and reduce hardware overhead, as shown in Fig.1b. Each step and its optimization is briefly described in the following.

A. Keypoint Detection

The FAST detection [3] operates on a 16-pixel Bresenham circle around the candidate pixel P with intensity of I_p. If there exists n contiguous pixels on the circle brighter than I_p plus a threshold t or darker than $I_p - t$, then the center pixel P will be classified as a corner, as illustrated in Fig.2. The original FAST detector build a decision tree by ID3 algorithm to classify pixels. However, we still perform FAST detection by its definition because it is efficient to implement in hardware and cost less memory. As suggested in [5], a scale built by downsampling the source image is also used to produce mulit-scale features. Besides, the optimal value of n is set to 9, which produces best repeatability.

B. Non-max Suppression

In [5], all pixels passed FAST the segment test are ordered by their Harris corner response and those with lower response are filtered. However, Harris response requires plenty of gradient and multiplication operations when computing the auto-correlation function of a local image patch. Instead, we define the score as the sum of the absolute difference (SAD) between the corner's intensity and the intensity of pixels on the continuous bright or dark arc. Only if the score of a candidate

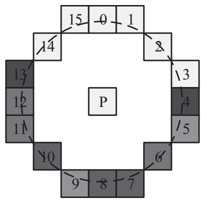


Fig. 2. The simplified computing patterns.

corner is higher than that of its 8-connected neighbors, or in the 3×3 window, it will be denoted as a keypoint.

C. Orientation Estimation

The orientation of a keypoint is estimated by the intensity centroid in a local patch. The horizontal component m_x and vertical component m_y of intensity centroid can be computed through sum of intensity $I(x, y)$ weighted by coordinate (x, y), where the origin locates at the keypoint:

$$\begin{cases} m_x = \sum_{x=-r}^{r} xI(x,y) \\ m_y = \sum_{y=-r}^{r} yI(x,y) \end{cases} \tag{1}$$

where r is radius of the local patch, set to 15 by default. Then the orientation θ of the patch is defined by:

$$\theta = \arctan(\frac{m_y}{m_x}) \tag{2}$$

where $\arctan$ is quadrant-aware. In our hardware implementation, the latency of orientation estimation will be hidden when loading image patch before descriptor generation.

D. Descriptor Generation

Since BRIEF descriptor is sensitive to noise, the image patch p used for pixel sampling should be smoothed first. Here we use a 5×5 binominal coefficient filter to approximate a Gaussian filter, in which convolution can be simply replaced by shift and addition operations. Then the descriptor is expressed as a 256 bit vector. Each bit of the vector is the result of a comparison $\tau(p; x, y)$ between the intensity of two pixels $p(x)$ $p(y)$ sampled from the patch centered at the keypoint:

$$\tau(p;x,y) = \begin{cases} 1 & p(x) < p(y) \\ 0 & p(x) \geq p(y) \end{cases} \tag{3}$$

ORB algorithm utilizes a learning method to select those pixel pairs with high variance and low correlation to achieve better performance. To improve orientation invariance, the coordinates of learned pixel pairs need to be rotated by the orientation of the keypoint before the binary test.

III. The ORB Accelerator

This section presents the architecture of the proposed ORB-based accelerator. The accelerator consists of six function units to implement the most features of ORB algorithm: image downsampling, image smoothing, keypoint detection, non-max

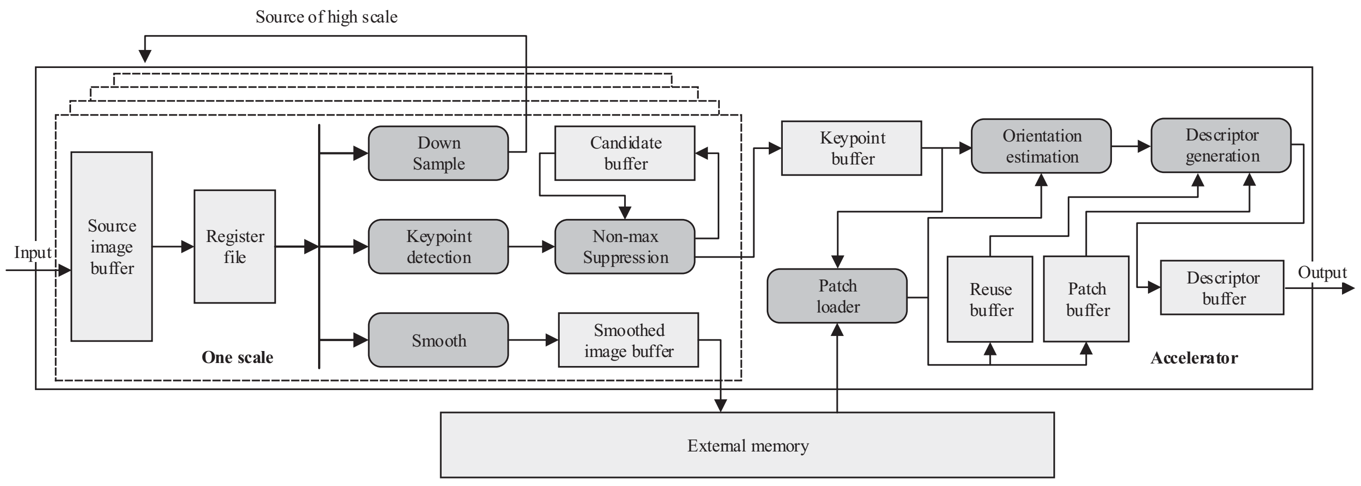


Fig. 3. The ORB accelerator architecture

suppression, orientation estimation and descriptor generation. The details of each function unit and several hardware-oriented optimizations are described in the following.

A. Overall Hardware Architucture

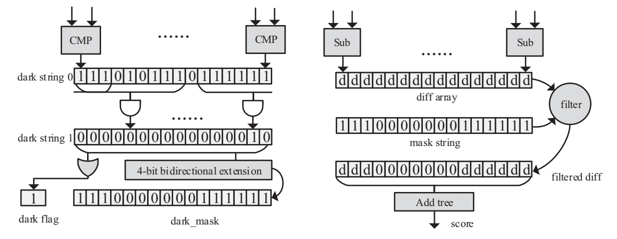
The overall architecture is shown in Fig. 3. Through a streaming interface, the gray-scale image is first written into a buffer. The buffer is composed of seven FIFOs connected end to end. Each FIFO has a depth of image width W (W is 1920 in the implementation) to store a row of pixels. A 7×7 2D register file is connected with the output ports of the buffer. Data in the previous FIFO are read and written to the next FIFO and the register file simultaneously on every clock cycle. Such memory organization can be regarded as a sliding window moving over the entire frame. The register file provides the ability to access 7×7 pixels within a local region, which is particularly advantageous for typical image processing algorithms.

Following the register file, function units including image smoothing, image downsampling and keypoint detection fetch data from the register file concurrently with respective patterns. These modules(the part in the dotted line in Fig.3) are duplicated to process in 4 scales in parallel. Source images except the fourth scale are downsampled by bilinear interpolation, and the result serves as the source of the higher scale. Moreover, the source images in the first and the third scale are also smoothed by convolution with binominal kernals and offloaded to external memory. The keypoint detection module not only extracts keypoint candidates but also computes their scores. Then the keypoint candidates are filtered by NMS, and written into the keypoint buffer. Based on a keypoint's position in a frame, the patch loader fetches a patch of data from the external memory and writes them to the reuse buffer and the patch buffer. Both of them provide data for the orientation estimation and the descriptor generation. The patch loader and the reuse buffer cooperate to realize the data reuse scheme.

B. String-based FAST-9 Detection

Previous works in [7] [9] implement FAST corner detection based on optimized string searching algorithms with early rejections. Inspired by these methods, we also generate binary strings to perform corner detection. Two identical test units (TU) without searching and early rejection, shown in Fig.4a, are used to judge whether a pixel is a dark or bright corner respectively. Take dark test as an example, it contains 16 parallel comparators. If the intensity of one pixel in Breshenman circle is less than that of the center pixel by a threshold, then the comparator outputs "1", otherwise "0". The result of these 16 comparators form a *dark_string0*, indicating the state of this 16 circle pixels. For the *dark_string0*, 9 continuous bits are tested by AND operation. This procedure is duplicated from the MSB and turned back when met with the LSB. The output of 16 AND operations again form a 16-bit *dark_string1*. If there exists at least one "1" in *dark_string1*, it means the center pixel passes the dark test and the *dark_flag* is set to "1". In the meanwhile, the *dark_string1* is left and right circle shifted by 1 to 4 bits. These nine strings are fused by OR operation to generate a *dark_mask* used for score computation. Similarly, the bright test unit produces a *bright_flag* and a *bright_mask*. The final *corner_flag* and *mask* are defined as the OR by bit of the results from the two TUs. As a result, if the *corner_flag* is "1", meaning the center pixel is a corner, the *mask* will include at least 9 continous "1" bits with the rest of "0", and the "1" bits represent those dark or bright circle pixels.

On the other hand, A score unit (SU) as seen in Fig.4b is also used to compute the score of the pixel based on result of the two TUs. The absolute values of difference between the circle pixels' intenstiy and the intensity of the center pixel constitute a *diff_array*. The 16 elements in the *diff_array* are then filtered by the *mask* to reserve those pixels on the dark or bright arc, denoted as *filtered_diff*. Instead of multiplication, the *filtered_diff* is calculated by the AND operations between the *diff_array* and the *mask*. Then an adder tree is employed to sum the differences of intensity to obtain the final score of the



(a) Data path for test unit (b) Data path for score unit

Fig. 4. The architecture of string-based FAST-9 detection

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

(a) ID number of pixels

1	4	6	4	1	
4	16	24	16	4	
6	24	36	24	6	/256
4	16	24	16	4	
1	4	6	4	1	

(b) Binominal kernels

Fig. 5. The templetes for image smoothing

pixel. Finally, for each pixel under test, the FAST detection module outputs a *corner_flag* and its *score*.

C. Resource Optimization in Downsampling and Smoothing

To detect keypoints in multiple scales, an image pyramid is built by donwsampling on lower levels. Moreover, images of each scale are smoothed to improve the robustness to noise of descriptors. Both of the downsampling and smoothing process operate in a 5×5 window. For the convenience of description, the ID number of each pixel in the window is shown in Fig.5a. For the pixel located at the center of the window, the smoothed result is calculated by convolving all pixels with the binominal kernels as in Fig.5b. To reduce the hardware cost, the convolution can be converted to add and shift operations:

$$\begin{cases} t_0 = \sum I_n & n \in \{0,4,20,24\} \\ t_1 = (\sum I_n) \ll 2 & n \in \{1,3,5,9,15, \\ & 19,21,23\} \\ t_2 = ((\sum I_n) \ll 2) + \sum I_n & n \in \{2,10,14,22\} \\ t_3 = (\sum I_n) \ll 4 & n \in \{6,8,16,18\} \\ t_4 = ((\sum I_n) \ll 4) + ((\sum I_n) \ll 3) & n \in \{7,11,13,17\} \\ t_5 = ((\sum I_n) \ll 5) + ((\sum I_n) \ll 2) & n = 12 \\ I_s = (\sum_{m=0}^{5} t_m) \gg 8 \end{cases} \tag{4}$$

where t_0 to t_5 are intermediate values, I_n is the intensity of pixels with ID number n and I_s is the intensity of center pixel after smoothing.

The downsampling with a scale factor of 1.25 is calculated by bilinear interpolation. Therefore, every five pixels will generate four downsampled pixels. As seen in Fig.6, among the five pixels two adjacent source pixels weighted by kernels are added to produce a downsampled pixel. This process is also realized by adders and shifters:

$$\begin{cases} I_{d0} = I_0 \\ I_{d1} = ((I_1 \ll 1) + I_1 + I_2) \gg 2 \\ I_{d2} = ((I_2 \ll 1) + (I_3 \ll 1)) \gg 2 \\ I_{d3} = (I_3 + (I_4 \ll 1) + I_4)) \gg 2 \end{cases} \tag{5}$$

where I_0 to I_4 and I_{d0} to I_{d3} intensity of source pixels and downsampled pixels respectively. The results of horizontal downsampling are then taken as input to vertical downsampling to compute the final downsampled image.

D. 3×3 NMS with Score Recorder

We design a four-stage keypoint filter to perform NMS as seen in Fig.7. After the FAST test, all pixels are sent to the NMS module in row-wise. These pixels are labeled with a flag indicating whether they are corners or not. The data processing flow of our proposed method is described as follows. First, M0 checks the flag and compares the score of the pixel in register B4 with its left neighbor pixel in register B3. Only if B4 and B3 are both corners and the score of B4 is lower than B3, will M0 set B4 to non-corner and output it to register B3. In the next cycle, the pixel in B3, which has been tested by M0, is compared with its right neighor in B4 by M1 in the same way. As a result, M1 outputs the suppressed result of B3. Then M2 tests whether the pixel in B1 is a keypoint by checking its flag and comparing its score with three nearest pixels A0 to A2 in the previous row. The result of B1 is written into a candidate buffer, and the buffer is read synchronously to keep the pixels in B2 and A2 are always vertically adjacent. At last, M3 tests A1 with three nearest pixels below and outputs the final 3×3 NMS result. In general, M0 and M1 together perform NMS on the center pixel with its two left and right neighbors in the same row, while M2 and M3 perform NMS on the pixel with its six top and bottom neighbors.

When stored to the local keypoint buffer, the pixel after NMS is recorded in a table according to its score as seen in Fig.8. The record table has 8 entries with two fields per each. The high-bit regions are initialized with pre-computed score steps (S0 to S7) ranging from the $t \times 9$ to the max, where t is the FAST threshold. The low-bit regions are counters used to record the number (N0 to N7) of keypoints higher than the corresponding scores. For example, if a corner with score higher than S2 but lower than S3, then counters for S0, S1 and S2 will all get increased by 1. A number threshold t_n and a pointer ptr are also attached to the score table. In the beginning, ptr points to S0 (or N0). When N0$\geq t_n$, ptr moves to S1 and this process is the same for S1 to S7. In practice, we

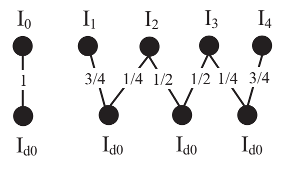


Fig. 6. The templetes for image smoothing

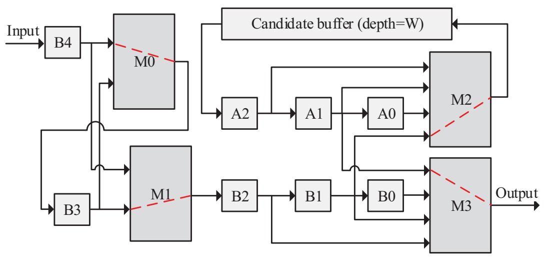


Fig. 7. The architecture of 3×3 NMS

divide a row into 8 segments and utilize two tables to record these segments in turn. When a table completes recording pixels of current segment, the final local score threshold t_s is determined:

$$t_s = \begin{cases} S[ptr] & N[ptr] > t_n/2 \\ (S[ptr] + S[ptr-1])/2 & N[ptr] \leq t_n/2 \end{cases} \tag{6}$$

Then corners fetched from the local keypoint buffer are then further refined, meaning that only those with score higher than t_s are retained and written into the global keypoint buffer. The global keypoint buffer contains four FIFOs for one scale per each.

E. Data Reuse in Patch Loading

Both orientation estimation and descriptors generation operate on a local image patch. Since keypoints are detected in row-wise, local patches required by two consecutive keypoints in the same row of a scale may be overlapped. Fig.9 shows an example of patch overlap, in which the distance d between the previous keypoint kp_0 and the current one kp, is less than the patch width $2r+1$. As a result, part of the local patch of kp_0, labeled by R can be reused for kp when computing its orientation and descriptor, and the rest part F with a size of $d \times (2r+1)$ ($2r+1$ is the patch height) will be fetched from the external memory. Therefore, the amount of external memory access can be reduced by the amount of data in overlapped region R.

The architecture of the patch loader is shown in Fig.10. As the smoothed images of the first and third scale are stored in external memory, patch loading are completed in Four steps. First, the boundary of the required local patch is calculated based on the keypoint coordinates, and compared with the previous boundary to compute the overlap and non-overlap region. After that, if the keypoint is located in the second

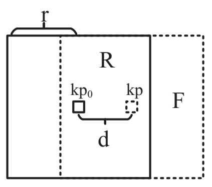


Fig. 9. Patch overlap

or fourth scale, the non-overlap region needs to be mapped on the lower scale by multiplication with the scale factor. Then the patch data are fetched from either the reuse buffer or the external memory. If the non-overlap region has been scaled, the data from external memory are downsampled to change the input patch to higher scale. At last, data from the external memory, the downsample module or the reuse buffer are written in to reuse buffer for next keypoint and transmitted to orientation estimation and the patch buffer as well.

F. Acceleration of Orientation and Descriptor Computation

In ORB algorithm, the intensity centroid of a keypoint is calculated in the region centered at the keypoint with a radius of r ($r = 15$ by default) to estimate its orientation. This time-consuming procedure involves large amount of multiplication and add operations, as well as triangle function operations. However, when writing data into the patch buffer prepared for descriptor generation, the output data from the patch loader can also be used to compute the orientation. Thus, the time cost on orientation estimation is hidden behind during patch loading. Fig.11 shows the architecture of orientation estimation. The coordinate unit generates the x and y components of the input pixel's coordinate. Two identical MAC units computes the x and y components of intensity centroid in parallel. After traversing all pixels in the circular region with a radius of 15, the cordic unit calculates tangent value of the orientation angle in 20 cycles. Then the tangent value is converted to an ID number as the final orientation by searching the LUT. In our configuration, the 2D coordinate system is split into 32 different orientations.

When orientation estimation is done, the same image patch is stored in both reuse buffer and patch buffer. Therefore, the time cost on descriptor generation is reduced by half by accessing the two buffers concurrently. As shown in Fig.12, the sine value $\sin\theta$ and cosine values $\cos\theta$ of the orientation angle

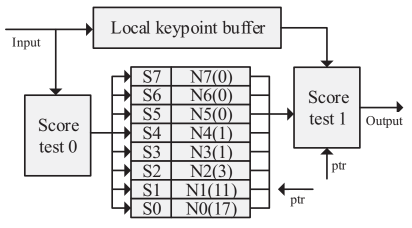


Fig. 8. The architecture of score recorder

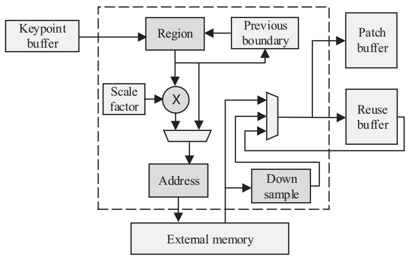


Fig. 10. The architecture of patch loader

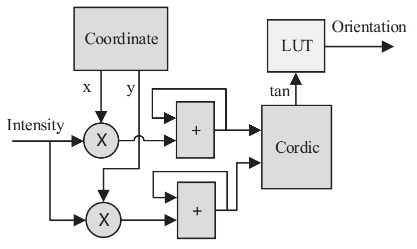


Fig. 11. The architecture of orientation estimation

are indexed by the orientation ID number. Then the coordinates (x, y) of a pair of pixels are rotated:

$$\begin{cases} x_r = x\cos\theta - y\sin\theta \\ y_r = y\cos\theta + x\sin\theta \end{cases} \tag{7}$$

where (x_r, y_r) is the rotated coordinates of one pixel. The addresses of pixel a and b is figured out by (x_r, y_r) to access the intensity values in the reuse buffer and the patch buffer respectively. The result of intensity comparison is shifted into the descriptor register. Hence a 256-bit descriptor is generated in 256 cycles.

IV. Experimental Results

The proposed accelerator is programmed with Verilog HDL and synthesized for Xilinx Zynq-706 evaluation board. It hosts an ARM dual-core CPU, 2GB DDR3 RAM and an XC7Z045 FPGA with enough resources to implement the whole design. Table I shows the resource utilization of the FPGA for ORB implementation. The detector module includes the source buffer, the register file, the keypoint detection unit, the smoothing unit, the downsampling unit, the NMS unit and the candidate buffer in one scale. The descriptor module contains the patch loader, the reuse buffer, the patch buffer, the descriptor buffer, the orientation estimation unit and the descriptor generation unit. For real-time verification, images are captured by a CMOS sensor, transmitted to the FPGA board and processed to generate ORB features. Then the features are sent to a PC through a USB channel. The feature matching operations between to images are completed on the PC. For the convenience of test on a common dataset, images from [11] are downloaded into the DRAM prepared for feature extraction.

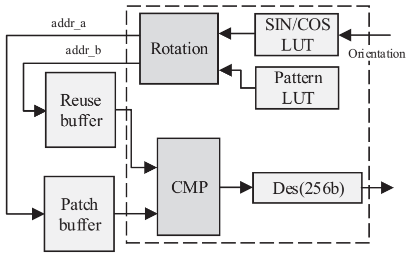


Fig. 12. The architecture of descriptor generation

TABLE I
FPGA synthesis result of the ORB accelerator

Module	LUTs	FFs	BRAMs	DSP48s
Detector module(one scale)	3568	2122	5	0
Global keypoint FIFO	173	236	4	0
Descriptor module	2398	2037	4	18
Total	11363	8403	28	18

A. Memory cost and BW requirement

This section evaluates the internal memory cost and the external memory bandwidth requirement with optimizations. Table II shows the sizes of memory required by the proposed accelerator to support 1080p processing. The default patch width for descriptor generation is set to 31. In the worst case, it has to be scaled to 43 when the orientation angle is $45°$. Because the source image is scanned in row-wise, the patch buffer should have to store 43 lines of pixels per scale for descriptor generation, which is unacceptable in embedded systems. With offloading smoothed images to the external memory, the internal memory size is dramatically reduced to only 583.4kb in our implementation.

To reduce the requirement of external memory bandwidth, we propose to store the smoothed images every other scale and reuse patch data when computing descriptors. Table III compares the size of access to the external memory with proposed optimizations. The simulation is performed on 5 images from the dataset [11], and the number of keypoints is limited to 1000. From the fifth column we can see the access size of external memory is reduced by 42.77% with the two proposed schemes on average. Normalized by the number of pixels in an image, the average access size is 4.13bytes/pixel approximately. Therefore, to process full-HD images in 30f-ps, the bandwidth requirement of external memory is about 1.96Gbps, only 15.3% of the available bandwidth for 16-bit DDR3 at 400MHz.

TABLE II
The breakdown of internal memory cost

Buffer	w/o ext MEM(kb)	w ext MEM(kb)
Source buffer	310	310
Smoothed image buffer	60	60
Candidate buffer	97.5	97.5
Reuse buffer	0	14.5
Patch buffer	1904	14.5
Keypoint buffer	44	44
Descriptor buffer	28.2	28.2
Others	14.8	14.8
Total	2442.7	583.5

TABLE III
THE SIZE OF ACCESS TO THE EXTERNAL MEMORY WITH THE PROPOSED OPTIMIZATIONS

Image	Size (byte)	No opt Access size (MB)	Low-scale storage(A) Access size (MB)	A+Data resue Access size (MB)
bikes	700000	4.649	2.997	2.666
boat	578000	4.161	2.797	2.285
cars	565494	4.109	2.776	2.412
graf	512000	3.897	2.689	2.247
ubc	512000	3.897	2.689	2.243
Average	573499	4.143	2.790 (67.34%)	2.371 (57.23%)

B. Performance evaluation

We compare the performance of our design with both the software solution on PC and previous hardware solutions.

First we program ORB algorithm using the OpenCV library on an Intel i5-4670 processor at 3.40GHz with 8GB DRAM. For a fair comparison, we set the same parameters as the hardware accelerator for the software program. Both of them extract 1000 features in 4 scales with a scale factor of 1.25 and a patch size of 31. Table IV shows the processing time of software solution and our accelerator. For VGA images with 1000 keypoints, the performance of software and our hardware solutions are 74.8ms and 23.8ms respectively. Actually, due to the hybrid pipeline architecture, the processing speed of the proposed accelerator mainly depends on the number of keypoints. Thus, for a given number of keypoints, the performance improvement of the accelerator over the software solution will increase when processing high-resolution images.

Table V lists the comparison of different hardware architectures. Previous works in [7] [8]and [10] implement only FAST and BRIEF algorithm with no support for scale and rotation invariance. The work in [10] has the least internal memory size since it is designed for VGA image processing. In reference [9], the orientation estimation is greatly simplified, which results in loss of matching accuracy. Compared with previous works, our design implements the most features of ORB algorithm. A significant reduction in internal memory cost is obtained by offloading large amount of image pyramid data to external memory. It achieves 42fps for full-HD processing at moderate 100MHz operating frequency and generates 1000 features per image which is suitable for practical applications.

TABLE IV
PROCESSING SPEED OF SOFTWARE AND HARDWARE FOR VGA IMAGES

Image	SW(ms)	HW(ms)
bikes	62	24.23
boat	93	24.02
cars	47	23.59
graf	94	23.87
ubc	78	23.27
Average	74.8	23.8

TABLE V
PERFORMANCE COMPARISON WITH RELATED WORKS

Work	Memory (kb)	Clock (MHz)	RI/OI	Number of features	Speed (fps)
[7]	1024	200	N/N	512	94.3
[8]	660	100	N/N	-	48
[9]	1640	200	3/16	-	135
[10]	194	111	N/N	-	141(VGA)
This work	583.4	100	4/32	1000	42

C. Accuracy evaluation

To evaluate the quality of features generated by the proposed accelerator, we adopt the *recall* versus *1-precision* method presented in [12]. Recall is the number of correctly matched regions with respect to the number of corresponding matched points between two images:

$$recall = \frac{\#correct\ matches}{\#correspondences} \tag{8}$$

The number of false matches relative to the total number of matches is represented by 1-precision:

$$1 - precision = \frac{\#false\ matches}{\#correct\ matches + \#false\ matches} \tag{9}$$

We run simulations on software and hardware with the same configuration as the previous. The features generated on hardware are transmitted to PC for matching. The matching strategy is based on nearest neighbor (NN) searching. Fig.13 shows the curves of recall versus 1-precision of software and hardware implementations as the matching threshold varies. The matching result of hardware is slightly worse than that of software. From the description in Section II and Section III, we can conclude several reasons for matching performance decrease. First, we use FAST response instead of Harris response as the score of a keypoint when filtering low-quality keypoints, and the latter one is more effective to

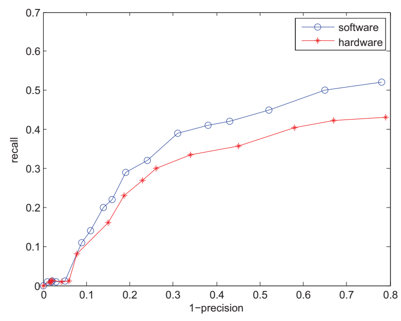


Fig. 13. The matching performance of software and hardware solutions

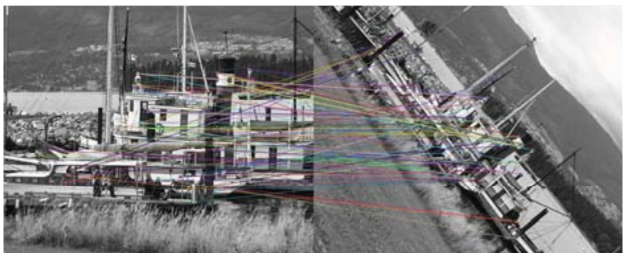


Fig. 14. The matching results of the proposed accelerator

remove keypoints along edges. Second, the truncation errors derived from the conversion to fix-point format in orientation estimation will accumulate and propagate along the data path, and decrease the quality of descriptors. Third, when loading image patch from a lower scale, the downsampling of the non-overlapped region brings in little intensity errors, which also has a effect on orientation estimation and descriptor generation. Despite the matching performance degradation, the overall matching accuracy of the features generated by the accelerator is acceptable for practical use. Fig.14 shows an example of feature matching based on the accelerator, the number of keypoint is limited to less than 100 by setting a high threshold for the convenience of display.

V. Conclusion

This paper proposes a hardware architecture for ORB algorithm with full implementation. Several trade-offs are made to reduce resource overhead and remain good feature matching accuracy. The data path of the accelerator is loose-coupled by the keypoint FIFO. Keypoints are extracted in multiple scales concurrently while descriptors are built on keypoints, which balances the workload in different pipeline stages. In addition, large amounts of smoothed image data are stored to external memory and fetched again for descriptor genration. Therefore, the internal memory size is less than 36% of the previous work [9]. To relieve the external memory bandwidth requirement, smoothed images are stored every other scale. Then, a shared data-reuse structure is introduced for patch loading. These two schemes reduce the access size of the external memory by 42.77%. Besides, a score recorder is used to filter low-quality keypoints adaptively, resulting in a more balanced distribution of keypoints in a frame. The proposed architecture is implemented on FPGA at 100MHz. It can extract 1000 features from 1080P images at 42fps. An external memory bandwidth of 1.96Gbps and internal memory of 583.4Kbits are suitable for real-time embedded applications.

References

[1] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[2] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," *Computer vision and image understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[3] E. Rosten, R. Porter, and T. Drummond, "Faster and better: A machine learning approach to corner detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 1, pp. 105–119, Jan 2010.

[4] M. Calonder, V. Lepetit, M. Ozuysal, T. Trzcinski, C. Strecha, and P. Fua, "Brief: Computing a local binary descriptor very fast," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 7, pp. 1281–1298, July 2012.

[5] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International conference on computer vision*. IEEE, 2011, pp. 2564–2571.

[6] R. Mur-Artal, J. Montiel, and J. D. Tardós, "Orb-slam: a versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[7] J.-S. Park, H.-E. Kim, and L.-S. Kim, "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*, vol. 23, no. 5, pp. 832–845, 2013.

[8] M. Fularz, M. Kraft, A. Schmidt, and A. Kasinski, "A high-performance fpga-based image feature detector and matcher based on the fast and brief algorithms," *International Journal of Advanced Robotic Systems*, vol. 12, 2015.

[9] W. Zhu, L. Liu, G. Jiang, S. Yin, and S. Wei, "A 135-frames/s 1080p 87.5-mw binary-descriptor-based image feature extraction accelerator," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 26, no. 8, pp. 1532–1543, Aug 2016.

[10] O. Ulusel, C. Picardo, C. B. Harris, S. Reda, and R. I. Bahar, "Hardware acceleration of feature detection and description algorithms on low-power embedded platforms," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, Aug 2016, pp. 1–9.

[11] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. Van Gool, "A comparison of affine region detectors," *International journal of computer vision*, vol. 65, no. 1-2, pp. 43–72, 2005.

[12] K. Mikolajczyk and C. Schmid, "A performance evaluation of local descriptors," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 10, pp. 1615–1630, Oct 2005.