Implementation of High Performance Hardware Architecture of OpenSURF Algorithm on FPGA

Xitian Fan, Chenlu Wu, Wei Cao, Xuegong Zhou, Shengye Wang, Lingli Wang*
State Key Laboratory of ASIC and System, Fudan University
No. 825 Zhangheng Road, Shanghai, 201203, China
* caow@fudan.edu.cn

***Abstract*—This paper proposes a high performance hardware architecture of Speeded Up Robust Features (SURF) algorithm based on OpenSURF. In order to achieve high processing frame rate, the hardware architecture is designed with several characteristics. Firstly, a sliding window method is proposed to extract feature points in parallel at selected scale levels. As a result, the time cost in feature extraction can be greatly reduced. Secondly, data reuse strategy is proposed in orientation generation and descriptor generation to reduce the memory access times. In this way, $3.87\times$ and $2.25\times$ speedup are achieved respectively. Thirdly, the integral image is segmented to buffer in different memory blocks in order to support multiple data accessing in one clock cycle, which will further reduce the whole calculating time of our implementation. The hardware architecture is implemented on an XC6VSX475T FPGA with 156 MHz and its maximal frame rate for VGA format image can reach 356 frames per second (fps), which is 6.25 times frame rate of OpenSURF running on a server with a Xeon 5650 processor, and 6 times the reported frame rate of the recent implementation on three Vritex4 FPGAs [8].**

I. INTRODUCTION

Image matching and object recognition are complicated tasks in the field of computer vision because objects are generally dependent on background illumination, camera shooting angle as well as the noise caused by image sensors [1]. In order to retrieve images or detect objects in the ways that are invariant to these factors, many algorithms have been proposed [2][3][4]. Among them, Scale Invariant Feature Transform (SIFT) [2] and Speeded-Up Robust Feature (SURF) [4] are widely used in a variety of applications, such as image mosaic [9], object detection [10], 3D image reconstruction [11] and etc. However, one of the critical problems of these two algorithms is the high computation complexity. Although SURF has lower computation complexity and can achieve almost the same performance in terms of accuracy comparing with SIFT [4], it is still time-consuming and needed to be accelerated for practical applications. Generally speaking, the acceleration of SURF algorithm can be divided into two categories: software acceleration and hardware acceleration. As to software acceleration, multi-cores CPUs and GPUs are adopted as acceleration platforms; the acceleration is achieved by exploiting the parallelisms between different cores. Similar implementations can be found in [12][13].

As to hardware acceleration, several realizations have been proposed in [5][6][7][8]. In [5], a configurable and scalable feature extraction architecture has been proposed, in which descriptors are generated by an embedded PowerPC CPU. When processing 630x457 resolution images on an FPGA, 1.6 seconds are needed to extract feature points and calculate the correspondent descriptors. D. Bouris et al. [6] introduce another approach. In their implementation, frame rate up to 56 fps can be achieved for VGA format images. But the drawback of their approach is that *descriptor generation* stage has not been realized. In [7], an implementation of modified SURF algorithm has been reported. They propose a new rotation invariant descriptor called O-DAISY to replace the SURF descriptor. The implementation results show that with the image resolution of 640×480, 406 fps can be achieved. However, the time to generate feature points is not taken into account. Another hardware implementation of the SURF algorithm has been reported by T. Sledevic and A. Serachis [8]. In their system, frame rate of 60 fps for VGA format images can be achieved. But due to the hardware resource limitation, interpretation step in the *feature extraction* is neglected; the *orientation* and *descriptor generation* stages of SURF algorithm are simplified. However, the effects of these simplifications on the matching accuracy have not been analyzed.

In this paper, a high performance hardware architecture of SURF algorithm is presented. Compared with the previous works that have been realized on an FPGA, our contributions are:

1) A sliding window method is proposed to implement the feature extraction stage, which can significantly reduce the calculation time and improve the performance.
2) In the stage of orientation generation and descriptor generation of SURF algorithm, data reuse strategy is exploited and implemented. As a result, more parallelisms can be achieved and the performance can be further improved.
3) A new integral image buffering method is proposed, with which multiple data can be accessed in one clock cycle at the same scale.

The rest of this paper is organized as follows: Section II briefly reviews the SURF algorithm. Section III presents the designed hardware architecture of SURF algorithm. Experiment results on a Xilinx Virtex6 FPGA device are demonstrated in Section IV. Finally, Section V draws the conclusions.

II. OVERVIEW OF SURF ALGORITHM

The SURF algorithm can mainly divided into four stages: integral image generation, feature extraction, orientation generation and descriptor generation. These stages will be briefly described as follows [4].

This research is supported by National Natural Science Foundation of China (61131001,61171011) and National 863 Program of China (2009AA012201)

978-1-4799-2198-0/13/$31.00 ©2013 IEEE

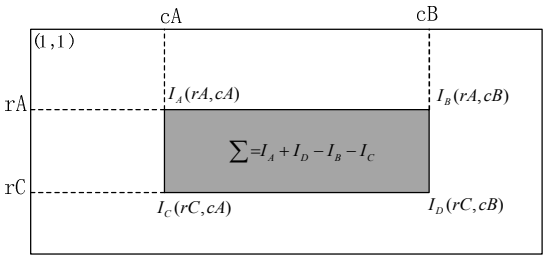

Fig. 1. Calculating summation of all pixels in a rectangle region [8].

A. Integral Image Calculation

Integral image is the novel method to improve the performance of the subsequent steps for SURF algorithm. It is defined by (1). By using the integral image, it is efficient to calculate the summation of pixels in an upright rectangle area of image. As an example illustrated in Fig. 1, whatever the size of this area is, the summation of pixels in the gray area can be formulated with one addition and two subtractions by (2).

$$I(x,y) = \sum_{r=1}^{y}\sum_{c=1}^{x} img(r,c) \tag{1}$$

$$\sum = I_A + I_D - I_B - I_C \tag{2}$$

B. Feature Extraction

SURF algorithm detects feature points base on the scale-space analysis. The scale space is divided into several octaves, each of which consists of four intervals. Each interval represents the determinant responses of Hessian matrix that calculated by three kinds of box filters (denoted as D_{xx}, D_{yy} or D_{xy}), which is the approximation of the second order Gaussian derivatives (denoted as G_{xx}, G_{yy} or G_{xy}). Every interval corresponds to a scale level and we denote the scale as *s* in this paper. Table I shows the relationship between scale *s* and the size of box filter in each octave (see [4] for more detail).

In order to detect feature points, the determinants of Hessian matrix expressed as (3) for every sampling point $\mathbf{x} = (x, y, s)$ are calculated in an approximation way, as show in (4) where w^2 is a weight coefficient equal to 0.91. The D_{xx}, D_{yy} and D_{xy} are computed respectively by filtering the correspondent image region with one of the box filters as demonstrated in Fig. 2. In these box filters, the white, gray and black pixels are referred to 1, 0 and -2 for D_{xx} and D_{yy}, and 1, 0, -1 for D_{xy}. In [1], w^2 is changed from original value 0.91 to 0.875 as the latter is

$$H(\mathbf{x}) = \begin{pmatrix} G_{xx}(\mathbf{x}) & G_{xy}(\mathbf{x}) \\ G_{xy}(\mathbf{x}) & G_{yy}(\mathbf{x}) \end{pmatrix} \tag{3}$$

$$\mathcal{H}(\mathbf{x}) = \det(H) = D_{xx}D_{yy} - w^2 D_{xy}^2 \tag{4}$$

TABLE I. THE RELATIONSHIP BETWEEN SCALE AND THE SIZE OF BOX FILTER

Octave	2				4				...
Size of Box filter	9	15	21	27	15	27	39	51	...
Scale (*s*)	1.2	2	2.8	3.6	2	3.6	5.2	6.8	...

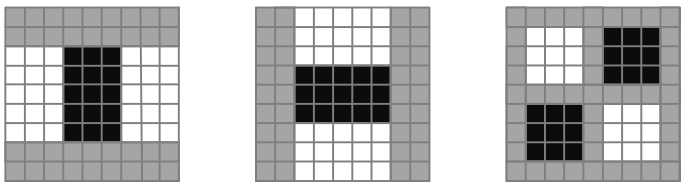

Fig. 2. D_{xx}, D_{yy} and D_{xy} box filter with the size of 9x9 when scale s equal to 1.2 [4].

more suitable to be calculated by hardware. After the calculation of $\mathcal{H}(\mathbf{x})$, threshold discard and non-maximum suppression in a $3 \times 3 \times 3$ neighborhood are carried out to extract the local extremes. Then interpretations according to (5) are performed in order to get sub-pixel precision.

$$\hat{\mathbf{x}} = -\frac{\partial^2 \mathcal{H}^{-1}}{\partial \mathbf{x}^2}\frac{\partial \mathcal{H}}{\partial \mathbf{x}} \tag{5}$$

C. Orientation Generation

In order to get rotation invariant, each feature point is assigned by a "principle orientation", which is generated as follows: Firstly, a circle with a radius of $6s_0$ (s_0 is the scale of detected feature point) is placed around the feature point. And then within the circle region, Haar wavelet responses in vertical and horizontal direction of 109 sampling points (the sampling step is s_0) are calculated. The size of each Haar-wavelet is set to be $4s_0$. Secondly, all pairs of Haar wavelet responses are weighted with a Gaussian ($\sigma = 2s_0$) centered at the feature point. The normalized results then are summed up by a sliding sector window of $\pi/3$ with the step of $\pi/18$. In our implementation, the sliding step of this window is set to be $\pi/12$. As a consequence, there are totally 24 pair summation results of sliding sector windows and each pair yields a local orientation vector. The longest vector is chosen to describe the "principal orientation" of the correspondent feature point.

D. Descriptor Generation

In this stage, a rectangle with the size of $20s_0$ is placed around the feature point and then rotated along its orientation. The rectangle is further divided into 4×4 sub-regions with the size of $5s_0$. In each sub-region, 25 pairs of Haar wavelet responses with the size of $2s_0$ are calculated. For simplicity, we mark Haar wavelet response that is parallel to the orientation as h_p and the perpendicular to the orientation as h_v. Then in each sub-region, h_p and h_v are weighted with a Gaussian ($\sigma = 3.3s_0$) centered at the feature point. After that, all the Haar wavelet responses in each sub-region can be summed up to form a four-dimensional vector $(\sum h_p, \sum h_v, \sum|h_p|, \sum|h_v|)$. Sixteen four-dimensional vectors make up a 64-dimensional descriptor and then scale normalization is employed to get the final unit descriptors.

III. PROPOSED HARDWARE ARCHITECTURE

This section is dedicated to describe the proposed hardware architecture of OpenSURF. Compared with original SURF introduce in section II, OpenSURF makes some major modification in two aspects. Firstly, in the stage of descriptor generation, the size of rectangle created to gen-

erate 64-dimensional descriptor is changed from $20s_0$ to $24s_0$. Secondly, the size of 16 sub-regions in the $24s_0 \times 24s_0$ rectangle is changed to be $9s_0 \times 9s_0$. As a result, every two adjacent sub-regions overlap to each other. Similar to the procedure described in Section II, the designed hardware architecture is divided into four major components: integral image generation, feature extraction, orientation generation and descriptor generation. In each component, optimizations in terms of hardware resource reduction and processing time reduction are performed. Details will be described as follows.

A. Overall Hardware Architecture

The overall hardware architecture of SURF algorithm is depicted in Fig. 3. In the designed system, the greyscale images are sent through the 10 gigabit Ethernet and buffered in the DDR3 memory. For data synchronization requirement, a small buffer is added to isolate two clock domains. The data in the small buffer are sent to a *serial-to-parallel* unit (denoted as *SP* in Fig. 3).

The *SP* unit produces *N* (*N* is the number of input channel and equals to 4 in the implementation) channels data for the first stage module to generate the integral image. The generated results are used in two ways. One is for the feature extraction module and the other is to be buffered in the memory, which contains four sub-storage blocks. When there is any feature point being detected, stage 3 starts to generate the feature orientations. The results of stage 3 are input to the stage 4 where the feature descriptors are calculated. The output results including the feature points, the correspondent orientations and descriptors are written to the *Send Data Buffer* module, where the Ethernet controller reads the results and sends them out.

Fig. 4 shows the operation flow of the hardware architecture. As what will be discussed in the following sub-sections, feature extraction module is not the most time consuming stage any more. The time of feature extraction can be concealed by the integral image generation module and descriptor generation module. On the other hand, the processing time of the orientation generation module can also be concealed. As a result, the total operating time of the hardware architecture is dependent on the time cost in integral image calculation and the descriptors generation.

B. Resource Optimization of Integral Image Generation

Integral image provides an efficient way to compute the sum of pixels' values in any upright rectangular region. In SURF [4], operations including box filter and Haar wavelet calculations are based on the integral image, which make SURF to be more computation efficient than SIFT. However, the large word length of integral image may seriously impact the performance of designed

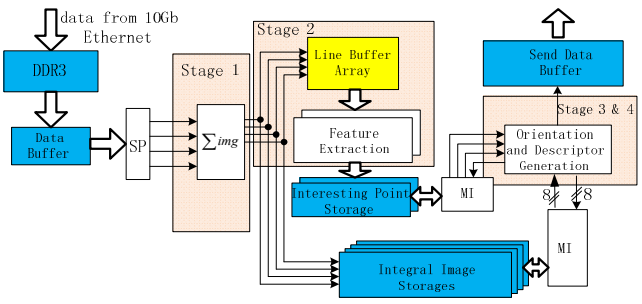

Fig. 3. Overall architecture implements SURF algorithm (MI is the memory interface)

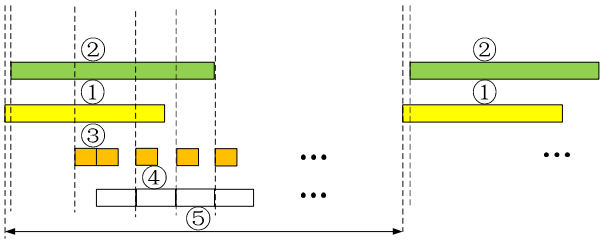

Fig. 4. Operation flow of design architecture (①:Integral image calculation; ②: Feature extraction; ③: Orientation generation; ④: Descriptors generation; ⑤: Time of processing the whole image).

hardware, especially for those implementations that need to store the whole integral image on FPGA. For example, as to an 8 bit VGA format image, the word length of the integral image is 27 bits. To store a frame of such integral image, $27 \times 640 \times 480$ bits or 7.9 Mb memory are required. That may be unacceptable in many cases. To deal with this problem, [17] proposed a technique that not only reduce the word length of integral image but also maintain the accuracy of the calculated results. According to [17], if the width and the height of the maximal rectangular is w_{max} and h_{max}, the word length L_{ii} of the final integral image is decided by (6).

$$(2^{L_{ii}} - 1) \geq (2^{L_i} - 1)w_{max}h_{max} \tag{6}$$

L_i is the word length of input image. Although there are some intermediate overflowing result during the process of computing integral image, final summation of pixels in any rectangular whose size is less than $w_{max} \times h_{max}$ is still correct. In our implementation, the maximal size of box filter is 51×51. This box filter can be broken down into three smaller box filters with the size of 17×26 for D_{xx} and 26×17 for D_{yy}. In this way, the word length of integral image is 17 bits if L_i equal to 8 according to (6), which result in 37% of memory reduction.

Inspired by [18], The hardware architecture of integral image is illustrated in Fig. 5, in which four rows of data are input simultaneously and four accumulators and adders are used to generate the integral image. *Address Generator* module is designed to generate the read addresses as well as the write addresses for *Last Row Memory* module. A bypass control signal is also generated by the *Address Generator* in order to select zero as one of the input of the first adder when *in_data_1* is the first row of the input image.

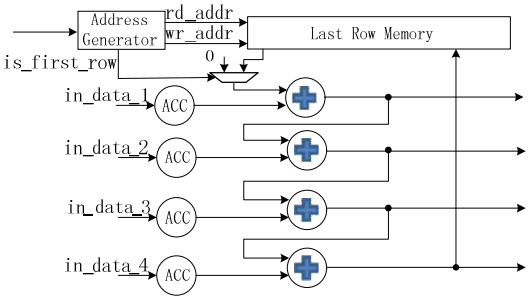

Fig. 5. Hardware architecture for integral image generation

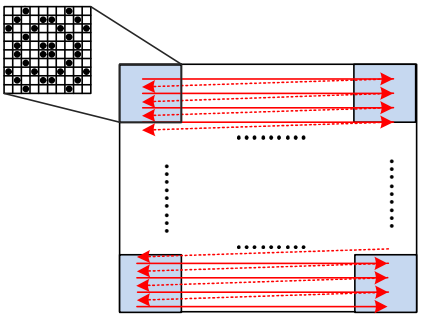

Fig. 6. Scanning through the whole image to calculate determinants by sliding window method.

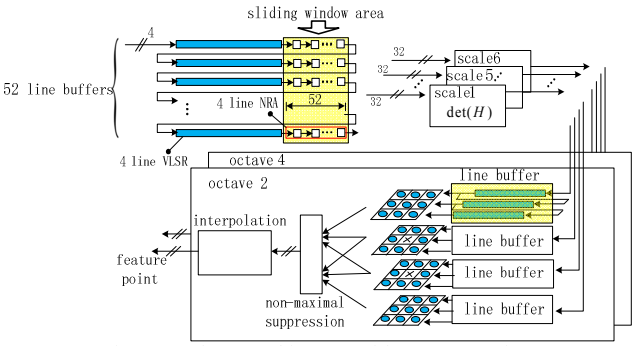

Fig. 7. Hardware architecture of feature extraction

C. Acceleration of Feature Extraction

Feature extraction is regarded to be the most time consuming stage in SURF when implemented by software [16]. In order to accelerate this stage, we proposed a parallel sliding window method to calculate the determinants of Fast-Hessian matrix at different scales. Sliding window can be regarded as a data selecting mask. In each mask, 32 points can be selected to calculate the determinant of Fast-Hessian. Fig. 6 shows an example of a 10×10 sliding window selecting 32 points for determinant calculation at scale $s = 1.2$. The black dots are the positions to select the data. This window shifts from the left of the image to the right and from the top to the bottom. After scanning through the whole image, all of the determinants of the image at correspondent scale are calculated.

For practical purpose, the hardware architecture only realizes the first 6 scales (the first two octaves) of OpenSURF as most of the feature points are detected at the first two octaves and cost less FPGA resources. As illustrated in Fig. 7, 52 lines buffer are used in order to buffer 52 rows of integral image data. In each line buffer, there are two components. The first component is *VLSR* (variable-length shift register) and the second component is *NRA* (normal register array). *VLSR*s are used to support different resolution image, while the *NRA*s are used to construct a 52×52 two-dimensional register array (denoted as sliding window area in Fig. 7), where the 6 sliding windows select the correspondent 32 points for determinant calculation. In Fig. 7, *N* (*N* is the number of input channel and equal to 4 in the implementation) rows of integral image are input to the line buffers and shifted from left to right. In the sliding window area, 32 points of each sliding window are selected and input to the determinant calculation units, where the determinants are calculated and sent to the non-maximum suppression unit. After local extremes are generated, feature points are calculated by interpolation module.

Taking advantage of the sliding window method, feature points are extracted at 6 scales in parallel. As a result, processing time of feature detection module can be significantly reduced. For a VGA format image, nearly $640 \times 480/N$ clock cycles are need. In the designed system with clock frequency of 156 MHz, the feature detection stage only cost 0.49 ms which is nearly 7 times speed-up of feature detection on the GF8800GTX GPU [16].

D. Data Reuse in Orientation Generation

As described in Section II, the Haar wavelet responses of 109 sampling points (the black points in Fig. 8) have to be calculated in the stage of orientation generation. To compute a pair of Haar wavelet responses, 8 sampling points should be accessed. As a result, the memory to store integral image will be accessed for 109×8 times when calculating a feature point's orientation. However, as illustrated in Fig. 8, many data of integral image can be reused. For example, to compute the pair of Haar wavelet responses of point E, 8 sampling points including A, B, C, D, F, G, H and I will be accessed. On the other hand, point D, F, G and I will be accessed again when the Haar wavelet responses of point H are calculated. By using this data reused strategy, the memory accesses times will be reduced from 109×8 to 15×15, which is equivalent to 74% time reduction.

Fig. 9 shows the implementation of data reused strategy when calculating the Haar wavelet responses. It consists of two components: Five lines shift registers with the depth of 15 and an 8-input adder tree. The 5 lines of shift registers

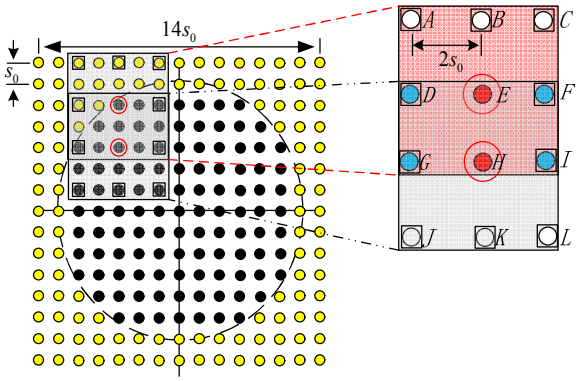

Fig. 8. Data reuse is applied in orientation generation.

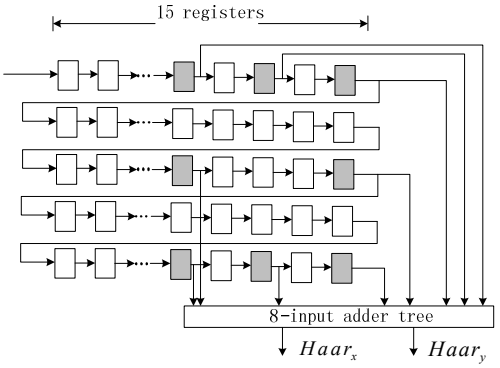

Fig. 9. The architecture of Haar Wavelets generation unit

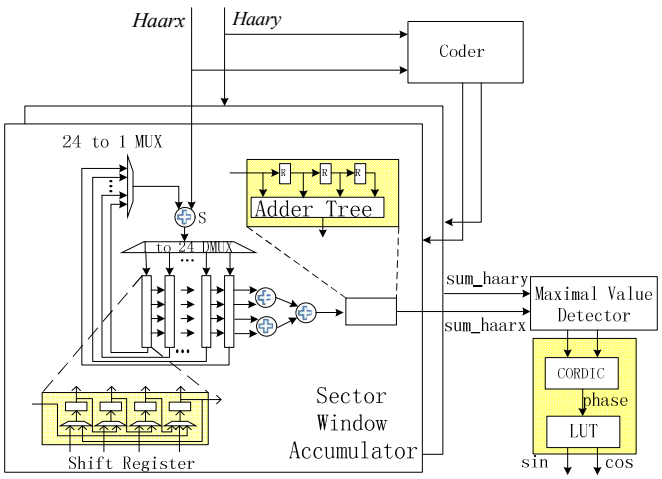

Fig. 10. The architecture of orientation generation

are connected head-to-tail to buffer the first 5 rows of sampling points while the 8-input adder tree is used to calculate the Haar wavelet responses.

The architecture of orientation generation is illustrated in Fig. 10. It contains four components: *Coder*, two *Sector Window Accumulator*s, *Maximal Value Detector* and *SinCos Generator*. The Haar wavelet responses generated by *Haar Wavelet Generation* unit are sent to the *Coder* module as well as the *Sector Window Accumulator* module. The function of the *Coder* module is to generate the control signals for the multiplexer and de-multiplexer in *Sector Window Accumulator* module. As described in Section II, the summation of Haar Wavelet responses in 24 sliding sector windows have to be calculated in order to compute the orientation. To do this, 24 *Shift Register* Modules are used to store the intermediate accumulation results of 24 sub-sectors that each covers $\pi/12$. Then the intermediate results are shifted out to two adder trees to calculate the accumulation of Haar Wavelet responses in every sector window that cover $\pi/3$ (four sub-sector windows are needed). The *Maximal Value Detector* detects the maximal input value and then sends it to the *CORDIC* module to generate the phase of the correspondent maximal responses. In the implementation, Look-Up-Table method is used to generate the sine and cosine values according to the generated phases.

E. Data Reuse in Descriptor Generation

As described in Section II, in the stage of descriptor generation, the rectangle centered on the feature point should be rotated along its orientation. The rotation behavior will lead to irregular redistribution of the sampling points that are used in the calculation of Haar wavelet responses. This problem will make data of integer image reuse become too complicated to realize. However, according to the OpenSURF, every two adjacent sub-regions overlap to each other in the rotated rectangle, as sub-region 1 and sub-region 2 illustrated in Fig. 11. Considering the fact that Haar wavelet responses in each sub-region should be weighted with the Gaussian ($\sigma = 2.5s_0$) centered on the sub-region. In order to reuse the weighted Haar Wavelet responses in the overlapping area, two-dimensional Gaussian kernel G can be represented as the convolution of two one-dimensional Gaussian kernels $G_h * G_v$. G_v is for vertical weighting and the G_h is for horizontal.

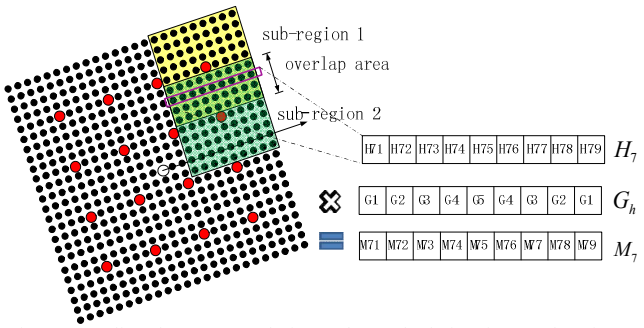

Fig. 11. Appling data reuse technique when calculating feature descriptor.

If we denote the *sub-region 1* and *sub-region 2* in Fig. 11 as $SR1$ and $SR2$, then each sub-region can be expressed as:

$$SR1 = (H1, H2, H3, H4, H5, H6, H7, H8, H9)^T \quad (7)$$

$$SR2 = (H6, H7, H8, H9, H10, H11, H12, H13, H14)^T \quad (8)$$

If we denote the result of *SR*1 and *SR*2 weighting with Gaussian kernel G as *SRG*1 and *SRG*2, then *SRG*1 and *SRG*2 can be expressed as:

$$SRG1 = SR1 * G = SR1 * (G_h * G_v) = (SR1 * G_h) * G_v \quad (9)$$

$$SRG2 = SR2 * G = SR2 * (G_h * G_v) = (SR2 * G_h) * G_v \quad (10)$$

As some elements in *SR*1 and *SR*2 are the same (*H*6 to *H*9 in (7) and (8)), the correspondent elements of the intermediate results named as $SR1 * G_h$ and $SR2 * G_h$ in the (9) and (10) can be reused, as M_7 illustrated in Fig. 11. By this technique, the speedup of the calculation of feature descriptors can achieve a factor of 2.25. This achievement is based on the times of accessing memory reduction. However, if we can further reduce the clock cycles cost in accessing every integral image data on memory, the calculation of feature descriptors can be further speeded up. The following sub-section introduces the approach.

F. Acceleration of Descriptor Generation

In the implementation, Block RAMs (Random Memory Access) are used to buffer the integral image. In order to accelerate the descriptor generation, enabling descriptor generation module to access memory for multiple data in one clock cycle is a good approach. To do this, the memory used to buffer the integral image is split up into 4 parts. Each part is termed as sub-storage block and denoted as $mem(n), n \in \{0,1,2,3\}$. For *J*-th row of the integral image, the associated sub-storage block $mem(i)$ is decided by (11).

$$i = \left\lfloor \frac{J \bmod 8}{2} \right\rfloor \quad (11)$$

The reason of using (11) is based on the fact below: In the descriptor module, the size of Haar wavelet is $2s_0$. Eight sampling points located in three rows will be accessed when calculating the Haar wavelet responses, as illustrated in Fig. 12. According to OpenSURF, the value of s_0 falls into the range of $[2, 6]$ when we only select the

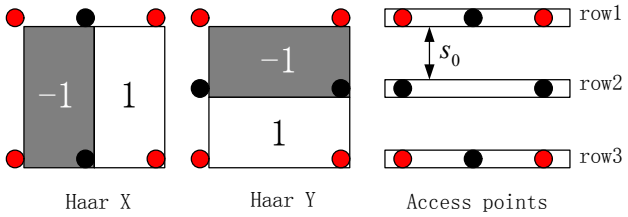

Fig. 12. Haar wavelet calculation needs to access 8 sampling points located in 3 different rows.

first 6 scales. As a result, only when the s_0 falls into the interval [3.75, 4.75] that the sampling points in *row1* and *row3* will locate in the same sub-storage block. Otherwise, sampling points in *row1*, *row2* and *row3* will locate in different ones. If each sub-storage block has dual ports, it only takes two clock cycles to access 8 sampling points when the rounding of s_0 is not equal to 4 or 5. Otherwise, three clock cycles are required.

By splitting the integral image to be buffered in multiple sub-storage blocks, the calculation time of Haar wavelet responses decrease from 4 clock cycles to 2 or 3 clock cycles, which equivalent to 2 or 1.33 times speedup.

The designed hardware architecture of descriptor generation module consists of two components. One is to compute the 24×24 Haar wavelet responses for very feature point. The other is to generate the 64-dimensional descriptor. To calculate a pair of Haar wavelet responses, 8 sampling addresses are generated simultaneously by an address generator to read integral image data from sub-storage blocks. The Haar wavelet responses are calculated by an 8-input adder tree. After that, the generated responses are sent to the descriptor calculation unit, which is depicted in Fig. 13. To generate the final 64-dimensional descriptor, every component of the input Haar wavelet responses are firstly buffered by 15 registers which are connected head to tail as a shift register. Four *Multiply Accumulate (MAC)* units are used to perform Gaussian weighting and the calculation of 4-dimensional vector in every sub-region. In this way, 18 *Processing Elements (PE)* in Fig. 13 generate the 64-dimensional descriptor, which is a unit 64-dimensional descriptor after processing in the *vector normalization* module.

IV. Experiment Results

We implement the proposed hardware architecture with a custom design board which contains a Xilinx XC6CSX475T FPGA, an 8GB DDR3 memory and a 10-gigabit Ethernet interface. The selected FPGA has 297600 Slice LUTs, 1064 Block RAM36s and 2016 DSP48Es. The resources of the FPGA are enough to implement the whole design. Detail resource utilization is listed in Table II.

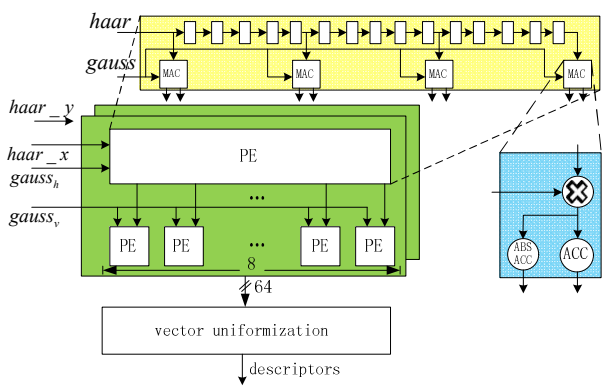

Fig. 13. Hardware architecture of descriptors generation

A. Precision Evaluation

In [21], an evaluation criterion of matching performance has been proposed, which uses the curve of *recall* versus *1-precision* to evaluate. According to [21], the number of correctly matched points relative to the number of corresponding matched points is defined as recall:

$$recall = \frac{\#correct\ matches}{\#correspondence} \tag{12}$$

The *1-precision* is defined as the ratio of the number of false matches to the total number of matches. The matched points including the false matches and the correct matches, as formulated in (13).

$$1 - precision = \frac{\#\ false\ matches}{\#correct\ matches + \#\ false\ matches} \tag{13}$$

As the whole system is designed with fixed point, we have done several evaluations before the proposed hardware architecture being designed, especially for the word lengths of sine and cosine values in the orientation generation stage as well as the number of octaves we selected to extract features. For evaluation purpose, several sets of images obtained from [22] are tested. The results are depicted in Fig. 14, which shows how large the matching performance changed as the number of octave and the word length of sine and cosine value decreased. Fig. 14 (a) shows that if we considerate the hardware resource utilization, it is reasonable to select two octaves to extract feature points with the sacrifice of a little matching performances. Because the more octaves are used, the more hardware resources are consumed. Fig. 14 (b) shows that when the word length of sine and cosine values increase from 2 bits to be 8 bits or 16 bits, the matching performance can change greatly, while the performance of 8 bits or 16 bits is just slightly poorer than that of float point. For saving

TABLE II. Resource Utilization of the Selected FPGA

FPGA	LUTs	DSP48Es	BRAMs
Integral Image Generation	219/297600 (0.07%)	12/2016 (0.59%)	1/1064 (~0%)
Feature Extraction	61345/297600 (20.61%)	mulators (39.78%)	0/1024 (0%)
Orientation Generation	8483/297600 (2.85%)	323/2016 (16.02%)	0/1046 (0%)
Descriptor Generation	15578/297600 (5.23%)	47/2016 (2.33%)	0/1064 (0%)
Whole System	107873/297600 (36.24%)	1185/2016 (58.77%)	295/1064 (27.72%)

TABLE III. COMPARISON WITH PREVIOUS WORKS

SURF version	Clock (MHz)	Resolution	Word length of image	FPS	Feature Number	Scale	Software/Hardware Implementation	Feature Detection(FD)/Orientation & Descriptor Generation(ODG)	Multi-Resolution Support
Schaeferling[1]	100	640x480	8	~2	~49	8	S+H	FD+ODG	No
Bouris [6]	200	640x480	8	56	NA	NA	H	FD	No
Fisher [7]	125	640x480	NA	406	NA	1	H	ODG	Yes
Sledevic [8]	25	640x480	4	60	100	6	H	FD+ODG	No
Proposed	156	640x480	8	356	100	6	H	FD+ODG	Yes

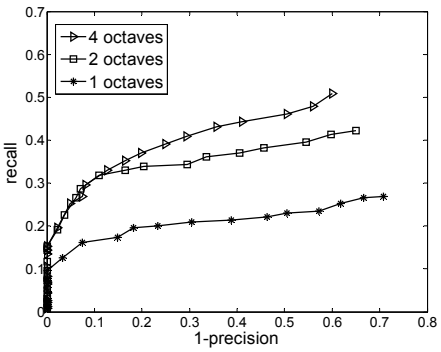

Fig. 14 (a). Results of evaluation by using differentnumber of octave

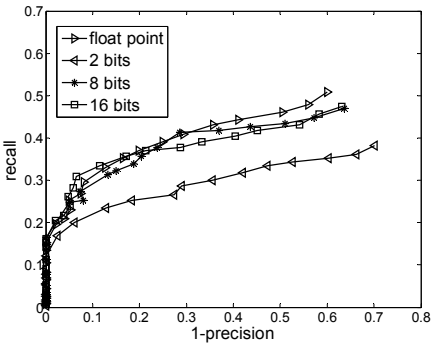

Fig. 14 (b). Results of evaluation by decreasing the word length of sine and cosine value.

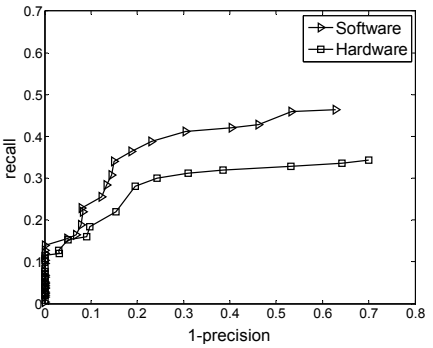

Fig. 15. Matching performance comparison between software and hardware.

hardware resources purpose, it is reasonable to select 8 bits to be the word lengths of sine and cosine values generated in the orientation stage.

The obtained images are also been tested by the proposed hardware on FPGA. The matching results of one set images are showed in Fig. 16, where each line in the two images represents a pair of matching points. From Fig. 16, we can see that except several mismatches, most of points can match correctly. In order to evaluate the matching performance of our hardware system at different threshold, we have run the software implementation of OpenSURF for comparison purpose. By varying the matching threshold, the curves of *recall* versus *1-precision* of hardware implementation and software implementation are shown in Fig. 15, where we can see that the recall of the hardware implementation is not as good as that of software implementation. The reasons are based on the following aspects. Firstly, we use only two octaves to extract the feature points, which will inevitably lead to performance degradation as proved above in Fig. 14 (a). Secondly, the whole system is designed with fixed point. Some truncation error may propagate and accumulate and lead to matching performance degradation. Such as the sine and cosine values generated in the orientation stage. We also have analyzed them above and showed the results in Fig. 14 (b). However, according to the test of our system, although the recall of the hardware implementation is a little poorer than that of the software implementation at a certain matching threshold, the overall accuracy of matching between different images is acceptable.

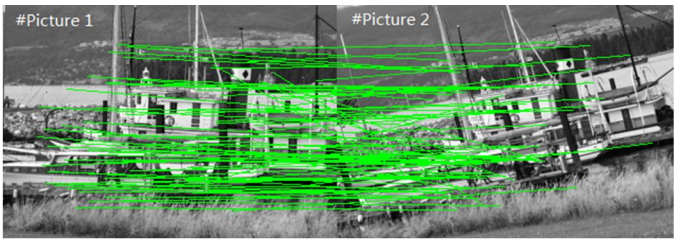

Fig. 16. Matching between two rotating images (color in electronic edition)

B. Performance Evaluation

We evaluate the performance of our design in two aspects. The first one is the performance comparison with server. The second one is the performance comparison with the previous works.

We run OpenSURF in Linux operating system on a server, which is integrated with a Xeon 5650 processor (2.66GHz, 6 cores and 12 threads). Considering the fact that performance in terms of frame rate of implementing SURF algorithm is dependent on the number of detected interesting points, for fairly comparison purpose, we just force the OpenSURF programs and the hardware to detect only a certain number of feature points and generate the correspondent descriptors. When we select the number of interesting points to be 100, the performances of the server and the proposed hardware on FPGA are 57 fps and 356 fps respectively. The results show that the design hardware on FPGA can achieve 6.25 times speedup compared with the software performance.

Comparisons between previous works and the proposed hardware architecture are listed in Table III. As we have discussed in Section I, reference [7] and [8] have modified and simplified some key procedures of SURF algorithm. For example, in reference [7], the feature descriptor is totally changed; in reference [8], the orientation generation and descriptor generation module are greatly simplified. On the contrary, our hardware architecture is designed consistent with the OpenSURF. As shown in Table III, when processing 100 feature points, our hardware architecture can gain 356 fps, which is nearly 6 times of the recent implementation proposed in [8]. Besides, our proposed hardware architecture is more flexible because multi-resolution images that do not exceed the size of 640×480 can be supported.

V. CONCLUSION

In this paper, we propose a high performance hardware architecture to implement SURF algorithm. Our hardware architecture is consistent with the OpenSURF and therefore, good matching accuracy can be achieved. We proposed a sliding window method to extract feature points in parallel for different scale levels and as a result, the time of extracting feature points can be greatly reduced. Besides, in the implementation stage of orientation generation and descriptor generation, data reuse strategy is proposed to reduce the time to compute Haar wavelet responses. Moreover, the memory to buffer integral image is split up into several sub-storage blocks, which can enable multiple data accessing in one clock cycle and contribute to the improvement of performance.

Each stage of the proposed hardware architecture can operate in parallel. When implementing on Virtex6 FPGA, the hardware architecture can achieve 6.25 times and 6 times speedup compared with midrange server and the recent FPGA implementation respectively.

REFERENCES

[1] M. Schaeferling and G. Kiefer, “Flex-SURF: A Flexible Architecture for FPGA-Based Robust Feature Extraction for Optical Tracking Systems,” In *International Conference on Reconfigurable Computing and FPGAs, ReConFig 2010*, 13-15 Dec. 2010, pp. 458-463.

[2] D. G. Lowe, “Distinctive image features from scale-invariant key points,” In *Proc. IEEE-RAS Int. Conf. Humanoid Robots*, Dec. 2006, pp. 530-535.

[3] M. S. Sarfraz and O. Hellwich, “Head pose estimation in face recognition across pose scenarios,” In *VisApp 2008: Proceedings of the 3rd International Conference on Computer Vision Theory and Applications*, HJ Araujo, Ed., 2008, pp. 235-242.

[4] H. Bay, A. Ess, and T. Tuytelaars, “Speeded-Up Robust Features (SURF),” *Computer Vision and Image Understanding,* v. 110, no. 3, pp.346-359, June, 2008.

[5] M. Schaeferling; Kiefer, G., "Object Recognition on a Chip: A Complete SURF-Based System on a Single FPGA," In *International Conference of Reconfigurable Computing and FPGAs*, *ReConFig 2011*, Nov. 30 Dec. 2011, pp. 49-54.

[6] D. Bouris, A. Nikitakis, I. Papaefstathiou, "Fast and Efficient FPGA-Based Feature Detection Employing the SURF Algorithm," In *18th International Symposium on Field-Programmable Custom Computing Machines (FCCM 2010)*, *2010*, pp. 3-10.

[7] J. Fischer, A. Ruppel, F. Weisshardt, and A. Verl, “A rotation invari- ant feature descriptor O-DAISY and its FPGA implementation,” In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, *2011*, pp. 2365–2370.

[8] T. Sledevic and A. Serackis, “SURF algorithm implementation on FPGA,” In *13th Biennial Baltic Electronics Conference (BEC)*, *2012*, pp. 291–294.

[9] J. Hong, W. Lin, H. Zhang, and L. Li, “Image Mosaic Based on SURF Feature Matching,” In *1st International Conference on Information Science and Engineering, 2009*, pp. 1287–1290.

[10] M.-L. Wang and H.-Y. Lin, “Object recognition from omnidirectional visual sensing for mobile robot applications,” In *IEEE International Conference on Systems, Man and Cybernetics, 2009*, pp. 1941–1946.

[11] M. P. Segundo, L. Gomes, O. R. P. Bellon, and L. Silva, “Automating 3D reconstruction pipeline by surf-based alignment,” In *19th IEEE International Conference on Image Processing (ICIP), 2012*, pp. 1761–1764.

[12] Z. Fang, D. Yang, W. Zhang, H. Chen, and B. Zang, “A comprehensive analysis and parallelization of an image retrieval algorithm,” *2011 IEEE International Symposium on Performance Analysis of Systems and Software*, pp. 154–164, 2011.

[13] N. Cornelis and L. Van Gool, “Fast scale invariant feature detection and matching on programmable graphics hardware,” In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, 2008*, pp. 1–8.

[14] 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,” In *International Conference on Field-Programmable Technology, 2009*, pp. 30–37..

[15] J. Qiu, Y. Lu, T. Huang, and T. Ikenaga, “An FPGA-Based Real-Time Hardware Accelerator for Orientation Calculation Part in SIFT,” In *Fifth International Conference on Intelligent Information Hiding and Multimedia Signal Processing, 2009*, pp. 1334–1337.

[16] N. Cornelis and L. Van Gool, “Fast scale invariant feature detection and matching on programmable graphics hardware,” In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, 2008*, pp. 1–8.

[17] Belt, H. J W, "Word length reduction for the integral image," In *Proceedings of IEEE International Conference on Image Processing, 2008*, pp.805-808.

[18] S. Ehsan, A. F. Clark, K. D. McDonald-Maier, "Novel Hardware Algorithms for Row-Parallel Integral Image Calculation," In *Proceedings of Digital Image Computing: Techniques and Applications, 2009,* pp.61-65.

[19] K. Mizuno, H. Noguchi, G. He, Y. Terachi, T. Kamino, H. Kawaguchi, and M. Yoshimoto, “Fast and Low-Memory-Bandwidth Architecture of SIFT Descriptor Generation with Scalability on Speed and Accuracy for VGA Video,” In *International Conference on Field Programmable Logic and Applications, 2010*, pp. 608–611.

[20] 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 Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 3, pp. 340–351, 2012.

[21] 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, 2005.

[22] K. Mikolajczyk. Local feature evaluation dataset. http://www.robots.ox.ac.uk/~vgg/research/affine/.