J Sign Process Syst (2016) 85:83–99
DOI 10.1007/s11265-015-0986-4

Analyzing the Performance-Hardware Trade-off of an ASIP-based SIFT Feature Extraction

Nico Mentzer · Guillermo Payá-Vayá · Holger Blume

Received: 30 October 2014 / Revised: 6 February 2015 / Accepted: 19 February 2015 / Published online: 11 March 2015

Abstract One of the key problems in the field of Computer Vision is recovering the geometry from multiple views of the same scene. Once the homography of two images is known, the motion of a stereo camera system can be determined, images can be rectified or image registration can be performed. A feature-based approach to determine the homography between two images bases on the extraction and matching of SIFT features (SIFT, Scale-Invariant Feature Transform). By extracting image features from varying images of one scene and finding corresponding image features in both images, the homography of the scene can be determined. The extraction of image features, which provide sufficient quality for computation of the homography of a scene, leads to an algorithm complexity, that prevents real-time applications on conventional CPUs. Therefore, we present and discuss an application-specific instruction-set extensions for a Tensilica Xtensa LX5 ASIP to accelerate a SIFT feature extraction (ASIP, Application-Specific Instruction-set Processor). In total, the complete SIFT feature extraction, executed on an extended processor is accelerated by a factor of x125 compared to the baseline processor. At the same time, the accuracy of the SIFT features is preserved. In addition, the proposed processor extensions maintain the full flexibility of an ASIP for a fast integration of further feature extractors.

Keywords ASIP · Scale-invariant feature transform · SIFT · Tensilica · LX5

1 Introduction

Nowadays, object detection, object recognition, and object tracking are mandatory for a large part of computer vision applications. For advanced driver assistance systems (ADAS), the goal is not only to detect objects, but also to interpret scenes. Therefore, the underlying algorithms rely on 3D information obtained by two or more camera views of the same scene [22]. Both consecutive images and synchronous images from two cameras offer information of the scene [17], which is used to feed safety applications in vehicles. However, these algorithms rely on a robust set of corresponding 2D image points of the same unknown 3D points. A feature-based approach to solve the problem of establishing point correspondences in images is SIFT (Scale-Invariant Feature Transform) [21]. Video surveillance of outdoor premises, e.g. airport aprons, is aimed at safety and security. Many complex maneuvers involving aircraft, vehicles and persons are carried out at such places. A system named ASEV that automatically assesses situations for airport surveillance is presented in [14]. The scene analysis bases on the extraction and matching of SIFT-image features not just to analyze a scene, but also to improve security against aircraft collisions or detect servicing activities. High resolution cameras, powerful CPUs and broadband wireless communication of mobile devices have shown great potential in mobile visual search [12]. To reduce the latency of wireless communication, data

N. Mentzer (✉) · G. Payá-Vayá · H. Blume
Institute of Microelectronic Systems, Leibniz Universität Hannover, Appelstrasse 4, 30167 Hannover, Germany
e-mail: nico.mentzer@ims.uni-hannover.de

G. Payá-Vayá
e-mail: guipava@ims.uni-hannover.de

H. Blume
e-mail: blume@ims.uni-hannover.de

reduction is required in the form of an image descriptor. This trend has received dedicated efforts in MPEG-7 standardization, namely, Compact Descriptor for Visual Search (CDVS) [13]. Distributed systems such as smart cameras are, mobile devices and advanced driver assistance systems in vehicles call for sufficient performance and a limited power consumption at the same time. Those features require special architectures to reach the design goals and to be flexible, simultaneously.

Evaluated as an excellent performing approach in detection and recognition quality [23], the computation complexity of SIFT is much higher than other feature-based object recognition algorithms such as SURF [4]. Furthermore, SIFT features are invariant to scale, rotation and illumination, and thus suitable for many applications. In addition to the high computational cost, SIFT also has high memory requirements, which prevents a real-time application for state-of-the-art standard processors. General purpose architectures (e.g., GPUs or state-of-the-art CPUs) provide sufficient processing power for image processing but their power consumption prevents their use in low power systems. Therefore, specialized architectures are necessary to guarantee the required processing performance and to meet the power restrictions.

SIFT is one algorithm providing sufficient quality in feature extraction from which a number of variations have been derived. Hence, other feature extractors such as SURF [4] show commonalities with SIFT. Dedicated hardware architectures for SIFT-feature extraction with a low power consumption are available [18], but they suffer from less flexibility for a fast adaption of future algorithms. Compared to hardware based systems, software based extraction of image features provides the opportunity to make fast code adaptions to implement future feature extractors, that might have lower complexity with similar good algorithmic quality at the same time. Due to the necessity for accelerated SIFT processing and the requirement for a flexible platform, Application-Specific Instruction-set Processors (ASIP) represent a promising approach with sufficient specialization and flexibility. By accelerating essential and widely used elements of general digital image processing and specialized feature-based object recognition tasks, the mandatory processing speed can be reached and simultaneously future features extractors can be integrated quickly.

Design and implementation of a custom instruction-set for the computation of SIFT features using a commercially available ASIP development environment by Tensilica [16] is presented in this paper. The paper is an extension of a previously published paper [22], at the SAMOS conference 2014, and gives a more detailed insight to the complexity of the SIFT algorithm and its implementation as well as its evaluation for ASPIs. The computationally most intensive arithmetic tasks are determined by detailed profiling and are accelerated by special Tensilica Instruction-set Extensions (TIE). These extensions will enable a significantly higher rate of SIFT feature extraction by accelerating a set of arithmetic functions. The full flexibility of a programmable platform is obtained and the full accuracy of the SIFT algorithm is ensured simultaneously. Compared to SIFT implementations on non-programmable hardware architectures, it will be shown that:

- it is possible to reach a significant speed-up of the SIFT feature extraction using a set of accelerated general computer vision hardware instructions and specific SIFT feature extraction instructions implemented with Tensilica Instruction-set Extensions and
- that the full flexibility of an ASIP for a fast integration of future feature extractors is preserved.

This paper is organized as follows: After a review of related work in Section 2, Section 3 introduces the SIFT algorithm and gives a short overview about the algorithm's complexity. In Section 4, a detailed description of the application-specific instruction-set processor with its given accelerators. The results and the evaluation are presented in Section 5. The paper concludes with Section 6.

2 Related Work

The extraction of image features for recovering the geometry from multiple views of the same scene is a key problem in the field of computer vision. An essential contribution to solve this problem was introduced in 2004 by Lowe with the SIFT feature extractor [21]. The SIFT algorithm achieves invariance of features to changes in scale and rotation and, therefore, it provides fundamental characteristics for implementing a feature-based solution for extracting and matching image features between two images. In the last years, investigations have been started to speed-up feature extraction and feature matching by reducing the complexity of the algorithm and by reducing the dimension of the feature descriptor. In 2004, Ke et al. presented PCA-SIFT [19], which reduces the dimension of the feature descriptors in order to accelerate the matching of features. Furthermore, in several implementations, the initial SIFT algorithm has been adopted [8, 10], but at the expense of accuracy, which leads to a decrease in feature quality. In the following years, a lot of alternatives to SIFT have been introduced (GLOH [23], SURF [4], FAST [25], BRISK [20], ORB [26], BRIEF [7], FREAK [1]), however, SIFT was evaluated as the hightest performing approach in detection and recognition quality [23].

The algorithmic quality entails a complexity which prevents a real-time application on conventional single-core CPUs for high definition images. In 2008, Zhang et al. [32]

presented an OpenMP-multicore implementation for SIFT-feature extraction. With eight cores (2x Intel Quad-Core 2, 2.33 GHz), a frame rate of 45 FPS could be achieved for VGA images. Wu et al. proposed the SiftGPU implementation in 2007 [30], which was considered as a GPU reference implementation for a few years. For a VGA input image, a frame rate of 27 FPS was achieved for a NVIDIA 8800GTX GPU. In 2013, Yonglong et al. [31] presented a CUDA-implementation for SIFT and reached a frame rate of 35 FPS for VGA images for a NVIDIA Telsa C2050 device. In the same year, Wang et al. [29] introduced an OpenCL implementation, which processed 74 FPS on a NVIDIA C2050 GPU with 512×512 pixel per image. Contemporary GPUs offer sufficient processing speed and flexibility for complex computer vision applications, but their high power consumption prevents their usage in low power applications. A SIFT-feature extraction for smartphones was presented by Wang et al. [28] in 2013. The ARM-v7a processor with a mobile GPU as accelerator reached 6 FPS in average with an power consumption of approximately 1.5 W for images with a size of 320×240 pixel. For this implementation, the OpenCL framework was used. In addition to a low power consumption, the SIFT-HW presented in 2012 by Huang et al. [18] provides high processing speed (30 FPS for VGA images), but the lack of flexibility prohibits its use in a still rapidly evolving algorithm field, such as feature extraction.

Bonato et al. [6] introduced a mixed architecture for the extraction of SIFT features. A highly parallelized hardware architecture was presented to accelerate computationally high intensive tasks, e.g., the building of pyramids, feature-point detection or stability checks. The generation of the descriptor itself is computed on an Altera NIOS II softcore processor. This mixed architecture runs on an Alter Stratix II FPGA and achieves a processing rate of 30 fps with almost the same accuracy as the reference CPU implementation for QVGA image size.

All mentioned platforms provide either sufficient processing performance, a low power consumption or the desired flexibility, but exclusively two out of those three features. To find a solution for satisfying all three characteristics, Application-Specifc Instruction-set Processors (ASIP) represent a promising approach among those requirements. ASIPs, especially the Tensilica Xtensa ASIP, have shown their capability in many digital image processing applications. Low power consumption combined with the high flexibility of a programmable processor and the possibility to accelerate specific algorithm processing bottlenecks, provides a promising base for computer vision applications on ASIPs. Banz et al. [3] achieved a speed-up of over x130 compared to a basic software implementation for the semi-global matching algorithm and reached a frame rate of 20 fps at VGA image size with a customized ASIP. Furthermore, Beucher et al. [5] demonstrated the capabilities of an instruction-set extension with a real-time ASIP implementation for motion-compensated frame rate conversion. The benefit of application-specific instruction-set processors is also discussed by Fontain et al. [15]. He presents a multi-core ASIP for 3D target tracking with a speed-up of x22 compared to a general purpose processor.

This paper presents a general set of customized instruction-set extensions that achieve a sufficient processing speed for SIFT feature extracting and fulfill the low power constraints. Furthermore, full flexibility for future feature extraction algorithms is retained by using application-specific instruction-set processors. The results are evaluated on the Tensilica Xtensa LX5 ASIP.

3 Scale-Invariant Feature Transform

This section is devided into two parts. The algorithm for extracting SIFT features [21] is presented by a detailed description of the four major algorithm steps. Furthermore, a complexity analysis is given in the second part of this section.

3.1 Algorithm Description

The SIFT algorithm consists of four parts. First, the Gaussian pyramid and the Difference of Gaussian (DoG) pyramid are computed. This step is followed by the feature-point detection, which includes identification and refinement of the feature-points and a stability check. The third calculation step is orientation assignment followed by the determination of the description of each feature.

3.1.1 Building of Pyramids

For a stable localization of feature-points, a scale-space with a Gaussian kernel is used. The scale-space $L(x, y, \sigma)$ is defined as the convolution of a Gaussian kernel $G(x, y, \sigma)$ with various scales and an input image $I(x, y)$:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \tag{1}$$

The Gaussian kernel is defined as

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2}$$

in which σ is the scale parameter and depends on the number of pyramid scales. The next octave of the pyramid is derived from the previous octave by downsizing the Gaussian image with twice the initial value of the scale parameter σ. The downsizing is realized by a subsampling which halves the image width and height. The total number of octaves depends on the size of the initial input image.

After the calculation of the Gaussian pyramid the Difference of Gaussian (DoG) pyramid $D(x, y, \sigma)$ is constructed by subtracting adjacent Gaussian images within an octave:

$$\begin{aligned} D(x, y, \sigma) &= (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) \\ &= L(x, y, k\sigma) - L(x, y, \sigma) \end{aligned} \tag{3}$$

The parameter k depicts the next higher scale of the Gaussian filtered images in one octave. In Fig. 1, the construction of the pyramids is shown.

3.1.2 Feature-Point Detection

The detection of feature-points is realized by analyzing the DoG pyramid. If a central pixel of a $3 \times 3 \times 3$ pixels neighborhood is a local minimum or a local maximum, the pixel is detected as a possible key point. The neighborhood consists of eight pixel surrounding a central pixel and the 3×3 pixels of both adjacent DoG pyramid scales, as illustrated in Fig. 2.

Each feature-point has to fulfill several stability checks, otherwise the candidates are rejected. Therefore, a local refinement for location and scale is applied. The 2^{nd} degree Taylor expansion of the scale-space function $D(x, y, \sigma)$ is used as a quadratic approximation for location refinement:

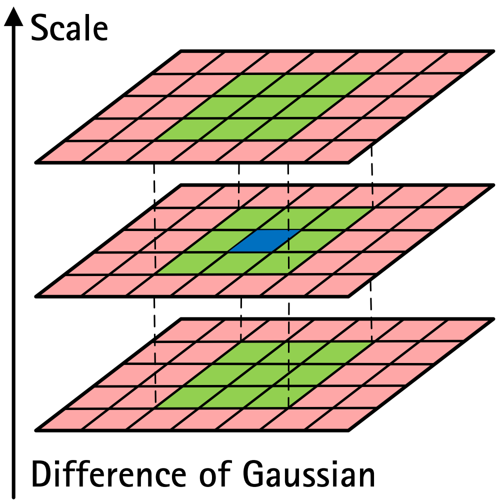


Figure 2 Key point detection: A central pixel is detected as a key point, if it is a local minimum or local maximum in a $3 \times 3 \times 3$ neighborhood of the DoG pyramid.

$$D(\hat{\boldsymbol{x}}) = D + \frac{1}{2}\frac{\partial D^T}{\partial \boldsymbol{x}}\hat{\boldsymbol{x}} \tag{4}$$

The function value at this refined position is checked for sufficient contrast by comparing it to an empirical value introduced in [21]. In a second stability check, all key point candidates with instabilities from small amount of noise are

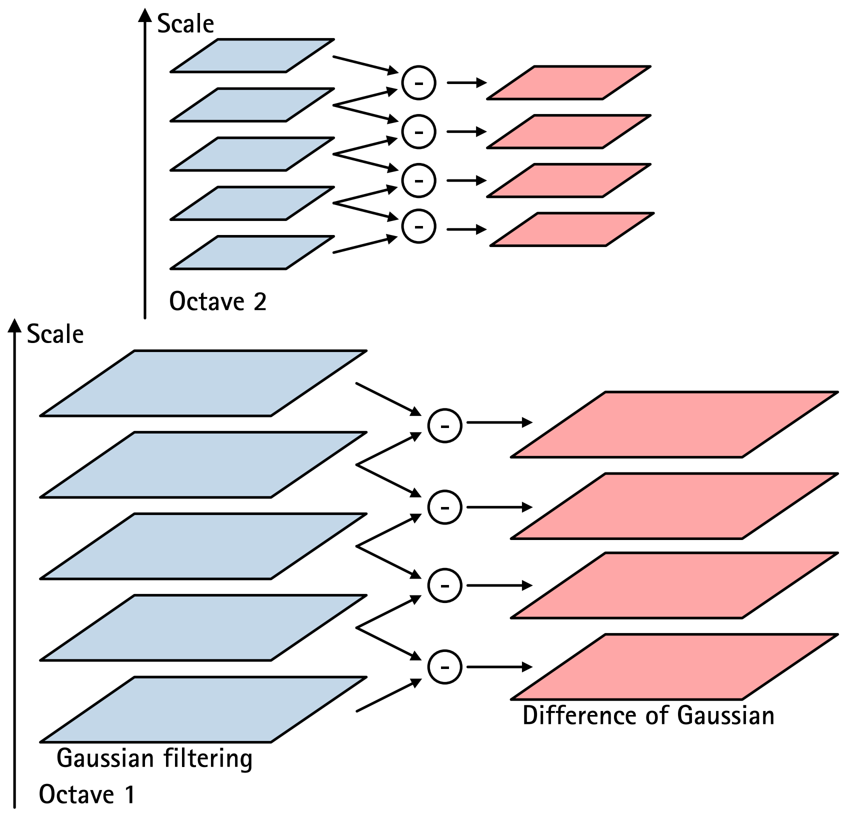


Figure 1 Calculation of pyramids: Resulting images of the Gaussian pyramid are the convolutions of a scale dependent Gaussian kernel with the adjacent images of lower scales. The input image for the next octave is the Gaussian filtered image that has twice the initial value of the scale parameter σ. The images of the DoG pyramid are computed by subtracting adjacent Gaussian images of one octave.

rejected by analyzing the principal curvature. The principal curvature is computed by the Hessian matrix $\boldsymbol{H}$ and its eigenvalues for a key point location. Instead of this computationally expensive task, it is sufficient to calculate the ratio of the eigenvalues and compare them with an empirical factor ε by [21]:

$$\frac{Tr(\boldsymbol{H})^2}{Det(\boldsymbol{H})} < \varepsilon \tag{5}$$

where $Tr(\boldsymbol{H})$ is the trace of the Hessian Matrix $\boldsymbol{H}$ and $Det(\boldsymbol{H})$ the determinant.

3.1.3 Orientation Assignment

To achieve invariance to image rotation, a local orientation is assigned to each key point. The local region of the Gaussian filtered image, with the scale close to the refined key point position, is used to compute an orientation histogram. For each pixel within this region, a gradient magnitude $m(x, y)$ and an orientation $\theta(x, y)$ are computed:

$$m(x, y) = \sqrt{(\partial L_x)^2 + (\partial L_y)^2} \tag{6}$$

$$m(x, y) = \sqrt{(\partial L_x)^2 + (\partial L_y)^2} \tag{7}$$

with

$$\partial L_x = L(x + 1, y) - L(x - 1, y) \tag{8}$$

$$\partial L_y = L(x, y + 1) - L(x, y - 1) \tag{9}$$

By choosing 36 equal bins for the orientation histogram, each bin covers 10° of the circular sector. Depending on $\theta(x, y)$ the magnitudes $m(x, y)$ of the pixel in the local neighborhood are accumulated in the bins. The peak in the histogram is called the orientation of a key point.

3.1.4 Feature Description

In the final step of the SIFT algorithm, the distinctive descriptor for invariance against illumination and viewpoint is generated. In a local neighborhood, the image gradient magnitudes and orientations are accumulated to histograms of $n \times n$ subregions (see Fig. 3). Each histogram consists of 8 bins. With $n = 4$ subregions, the resulting descriptor has 128 elements. To achieve a rotation invariance the histogram is rotated by the orientation of the feature-point that was computed in step 3. A final normalization step concludes the feature-point extraction.

3.2 Complexity Analysis

Regarding the number of operations and the memory accesses that have to be executed, the extraction of SIFT features is a challenging task. On the example of pyramid building and the feature-point detection, the number of operations and the memory consumption is determined for the non-optimized reference algorithm. The orientation assignment and the feature description are highly dependent on the number of found features and on the scene that is shown in the input image. Since it is only possible to determine an inaccurate estimation for the number of operations those two algorithmic steps are therefore omitted from this section.

3.2.1 Number of Operations

The building of pyramids consists of two steps: first, the building of the Gaussian pyramid and second, the building of the DoG pyramid. For both pyramids, it is possible to ascertain an upper bound of the number of operations.

The size of the Gaussian pyramid is determined by the size of the input image $width \times height$ and the number of scales, which is an algorithmic parameter, that have to be set in advance (see Fig. 1). The scales of every octave have their own specific size of filter kernel $N \times N$:

$$N = 2 \cdot \lceil 4 \cdot \sigma_s \rceil + 1 \tag{10}$$

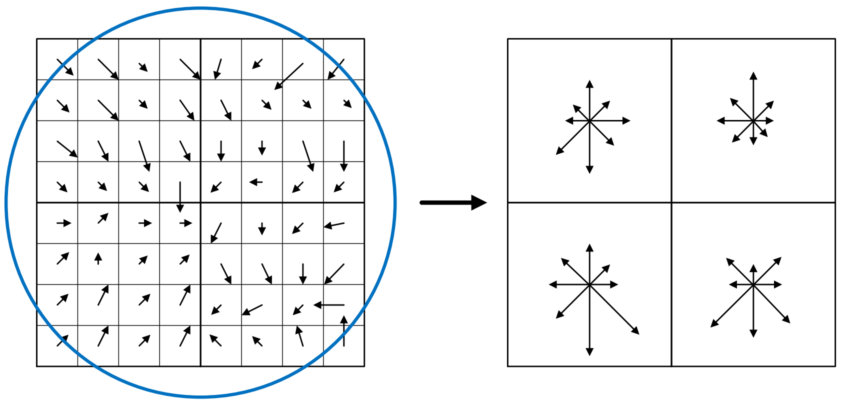


Figure 3 Keyoint descriptor: The image gradient magnitude and orientation are accumulated to histograms of $n \times n$ subregions (here $n = 2$) with 8 bins each. The gradient magnitudes are Gauss-weighted before accumulation.

with

$\sigma_s = \sigma_{init} \cdot k^{(s-1)} \cdot \sqrt{k^2 - 1}$

and

$k = 2^{\frac{1}{\#scales}}$

The number of octaves for both pyramids are dependent on the size of the input image:

$$\#octaves = \lfloor ld(MIN(width, height)) - 2 \rfloor \quad (11)$$

With the standard SIFT parameters by Lowe ([21]) ($\#scales = 3$, $\sigma_{init} = 1.6$) following setup for the building of pyramids arises:

- *#octaves* with six scales for the Gaussian pyramid
- *#octaves* with five scales for the DoG pyramid
- *#scales* + 2 different sizes of Gaussian kernels

Each $N \times N$ filter leads to $N \times N$ multiplications and $N \times N - 1$ additions, whereas each $N \times N$ filter has to be applied for *#octaves* times for different images sizes. Assuming an image size of 800×640, for this setup, there are approximately $1.2 \cdot 10^9$ multiplications and $1.2 \cdot 10^9$ additions for the Gaussian pyramid and $3.4 \cdot 10^6$ subtractions for the DoG pyramid. To find candidates for the feature-points, each pixel of the DoG pyramid has to be compared with 26 pixels in a local neighborhood. In total, there are $88.7 \cdot 10^6$ comparisons to be executed. Assuming, that one operation can be executed in one cycle, a system clock of 2.5 GHz is necessary, to reach one frame per second, just for executing the arithmetic operations of the first two algorithmic steps. This high clock frequency expresses the necessity for algorithmic optimization and parallelization, to reduce the number of operations in order to reach the design goal of SIFT feature extraction on multiple frames per second.

3.2.2 Memory Requirement

SIFT is a data-greedy algorithm, that requires a certain amount of memory to store temporal result. Typically, the number of feature-points, that will be found during the computation, will be below 1 % of the pixel of the input. Choosing a PGM input image [1] with a resolution of 800×640 pixel, 512 kB memory is necessary to store the image and about 737 kB for the resulting feature-points. To store the Gaussian pyramid and the DoG pyramid, $17 \sim 17$ MB memory is necessary, which is a factor of 33 compared to the input image.

Compared to other algorithms, which can be implemented on a streaming architecture [24] to reduce the number of operations and to avoid a large amount of temporal results, the proposed work dispenses this approach since the amount of data which needs to be held inside the processor in conjunction with the controlling overhead exceeds its internal storage capacity. Differing architectural concepts, e.g. dedicated hardware architectures on FPGAs, provide possibilities of massive parallel data processing with memory onchip to store temporal results, which leads to SIFT-streaming concepts which skips the storage of the DoG-pyramid.

To compute the orientation and the SIFT-descriptor itself, random memory accesses without any pattern have to be performed, which slow down the extraction of the SIFT-feature extraction. As soon as the feature-points candidates are found, the DoG pyramid can be discarded again, but both pyramids have to be accessible during the computation, which leads to a high memory consumption.

4 Application-Specific Instruction-Set Extension for SIFT Feature Extraction

The large number of operations resulting from the complexity of SIFT and the design goal to reach a certain frame rate is a challenge in the digital video processing, even for medium-sized VGA images. Control intensive parts of SIFT, e.g., stability checks or histogram generation, lead to branching and therefore to stalls in the processor pipeline. Furthermore, SIFT is a memory intensive algorithm, due to the large amount of intermediate results, such as the image pyramids and the arbitrary memory accesses during the orientation assignment and descriptor generation. To meet the high performance demands of the SIFT algorithm, the number of cycles can be reduced significantly by extending the processor with special functional units. This leads to a significant acceleration of SIFT processing on an application-specific instruction-set processor.

In a detailed algorithm analysis, the computationally intensive processing steps have been ascertained and the most cycle consuming parts have been split into three problem-setups: First, there is a large number of operations to execute; second, branching causes idle cycles in the processor pipeline; and third, long memory accesses delay the read stage in the processor pipeline.

After an introduction to the Tensilica LX5 processor, selected extensions of the processor are presented, related to the three problem-setups mentioned above.

4.1 Tensilica LX5 Processor

The baseline processor for this investigation is the Tensilica Xtensa LX5 ASIP. The Xtensa LX5 consists of a 32-bit architecture, which is configurable in several hardware options. Furthermore, the instruction set is extendable by

[1]PGM - Partable GrayMap; image file format for storing image data without any compression

self-defined instructions and register files using a verilog-like description language called TIE (Tensilica Instruction Extension) [27].

Tensilica offers multiple options to configure the 32-bit baseline processor. The SIFT-ASIP utilizes an integrated, fully pipelined, 32-bit multiplier, which requires two execution cycles. Furthermore, an integrated integer divider, which computes quotient and remainder (modulo operation) for 32-bit integer numbers, is implemented. Depending on the input bit patterns, the operation takes two to 13 cycles for a division. The non-pipelined hardware divider leads to a processor stall until a division is completed. The generation of the feature-point descriptor requires the resolution and dynamic of single-precision floating point numbers. Therefore, a floating point unit is added to the baseline processor to avoid massive cycle consumption by emulating floating point operations on fixed point hardware. A basic single-precision operation requires up to 100 cycles compared to a floating point operation, if it is emulated on fixed point hardware instead of using a floating point unit.

The Xtensa LX5 processor offers the possibility of using special *states*. States are meant for maintaining temporarily used data inside the processor. Direct register accesses rather than long memory accesses improves the performance of the processor. States behave like self-defined registers that can be read and written by special functional units. The advantages of states are demonstrated in the `FIR`-und `ISQRT`-unit. Hereinafter, states in figures are displayed through color coding.

The processor can be configured with a 5-stage pipeline or 7-stage pipeline. The 5-stage pipeline consists of the stages `instruction fetch`, `register-read`, `execute`, `execute/memory access` and `register-write`. A 7-stage pipeline has one extra stage for instruction fetch and, in addition, an extra stage for memory accesses. Those two extra stages are meant for processors with a clock rate that would be limited by slow memory accesses. By choosing a 7-stage pipeline, slow memory accesses can be executed in two pipeline stages instead of reducing the clock rate. The pipeline can be customized by adding new special functional units into the execution state of the pipeline. Those special functional units are defined by the designer and, conceivably, the critical path in a special functional unit can decrease the processor's clock rate. To prevent a reduction of the clock rate, the execution of special functional units for complex operations can occupy two execution pipeline stages. A scheduling of the operation defines which part of the operation is performed in which pipeline stage.

In order to ensure parallel data processing and to support long data words, new register files can be inserted into the pipeline. The register file can be customized regarding width and size. In this work, the pipeline has been extended with a new 64-bit register file `GRF`, in addition to the 32-bit base core register file `AR`. All modifications of the basic processor pipeline are shown in Fig. 4.

This work proposes three types of special functional units. Firstly, there are data handling units (e.g., `STORE64`, `MOVE64`, or `SHIFT64`) for the 64 bit register file. Secondly, there are basic computer vision special functional units (`FIR`) and thirdly, there are special SIFT-units, which accelerate arithmetic functions (e.g., `ISQRT`, `ATAN`, `SINE`, or histogram computations). All three types of units are designed to process data in one or two pipeline execution stages. An overview of the instruction-set extension is given in Table 1. In the following part, each special functional unit for accelerating the SIFT processing is presented separately.

4.2 Gaussian Pyramid

The first calculation step in the SIFT feature extraction is the building of the image pyramids through multiple Gaussian filterings (see Eq. 1). Dependent on the input image size, there are $\lceil log_2(MIN(width, height)) - 2 \rceil$ octaves. For each octave, five Gaussian filterings with different kernel sizes have to be computed. Due to the fact that a Gaussian filter is a separable and symmetric FIR-filter, the computation can be divided into horizontal and vertical processing steps to save operations. In addition, a buffer for storing intermediate results is implemented. For a VGA

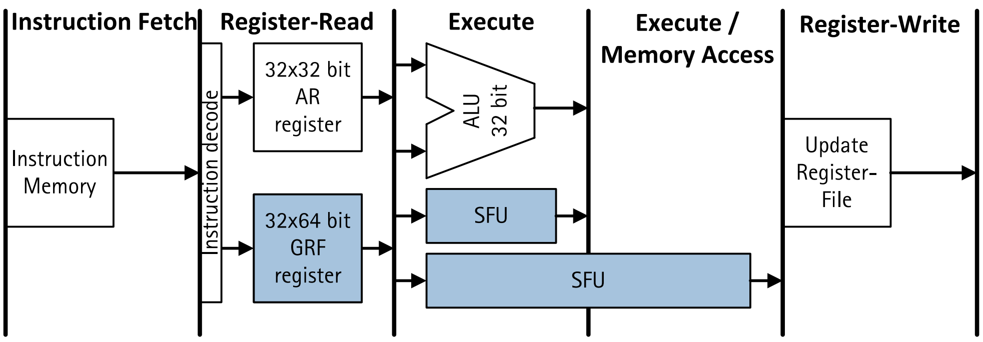


Figure 4 Pipeline scheme of the Xtensa LX5 processor for an arithmetic instruction execution showing the new register file and special functional units.

Table 1 Implemented special functional units of the Xtensa LX5 processor for accelerating the SIFT feature extraction and number of pipeline execution cycles.

Category	Instruction	Description	#EX	Used in
Data Handling	LOAD64	Load 64 bit into register	1	General
	STORE64	Store 64 bit word to memory	1	
	MOVE64	Move 64 bit to different register	1	
	SHIFT64	Shift 16 bit subwords by 1 subword	1	
Special Functional Units (Base CV)	FIR	Symmetric and separable FIR-filter vertical and horizontal unit	1	Building of Pyramids
Special Functional Units (SIFT)	ISQRT_INIT	Initialization step	1	Orientation Assignment and Feature Description
	ISQRT	Calculation of integer square root	1	
	ATAN	Calculation of arctangent	1	
	SINE_INIT	Initialization step	1	Feature Description
	SINE	Calculation of sine and cosine	2	
	HIST_VAL	Calculation of histogram entries	2	
	HIST_POS	Calculation of histogram positions	1	

input image, there are 31 Gaussian filterings to perform and therefore, 62 FIR-filter steps. The overall number of cycle reduction in this processing step has a large impact on the overall SIFT processing performance. The design of the special functional unit for the horizontal and vertical filter step is identical, but the loading of image data differs. Due to the image processing direction, the vertical processing direction uses four special functional units for parallel image processing. The block diagram of the special functional unit is shown in Fig. 5.

In each execution step, one pixel is streamed into the FIR-unit and one resulting pixel is computed. The pixel values are temporarily saved in a shift register, implemented by states, for the next computation step. After four iterations, the four resulting pixels have to be stored with a special data handling command. The additional *mode*-input specifies the kernel size of the filter matrix. Applying a one hot code, the associated multiplexer chooses the correct input pixel. All spare kernel entries are multiplied by zero. The correct filter coefficients are locally stored in a table with the *mode*-entrance as coefficient selection. A 7 × 7 FIR-unit is shown in Fig. 5 for clarification. Each scale of the image pyramid has a different filter kernel size, which depends on the initial Gaussian sigma and the number of scales (see Eq. 10). For the original SIFT parameter set of [21], the maximum filter kernel size is 27 × 27, which is used for the evaluation

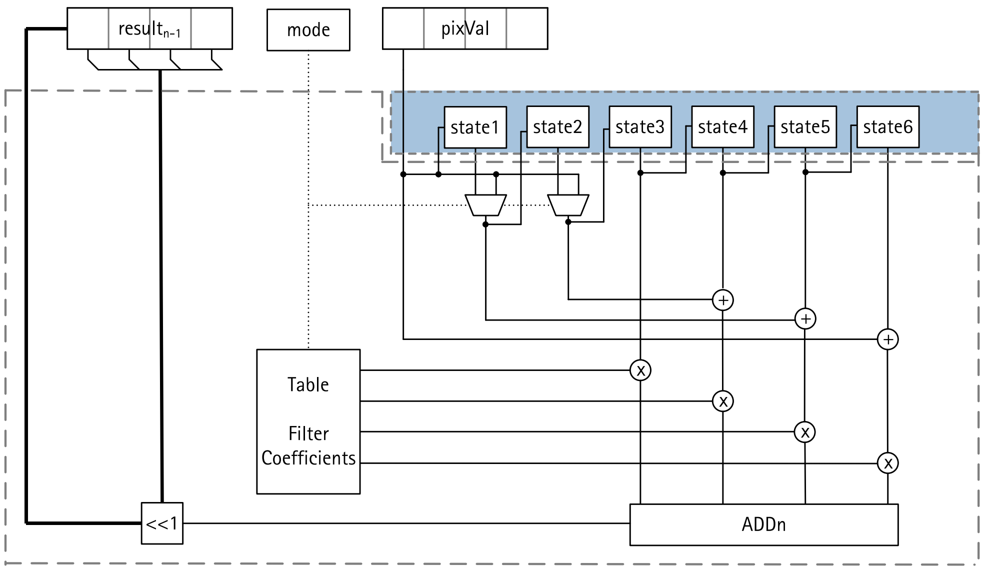


Figure 5 FIR unit for calculation of a Gaussian filtering. A single pixel is processed in each execution cycle. The *mode* input selects the size of the filter kernel and the different kernel coefficients. Special state registers are shown as color coded.

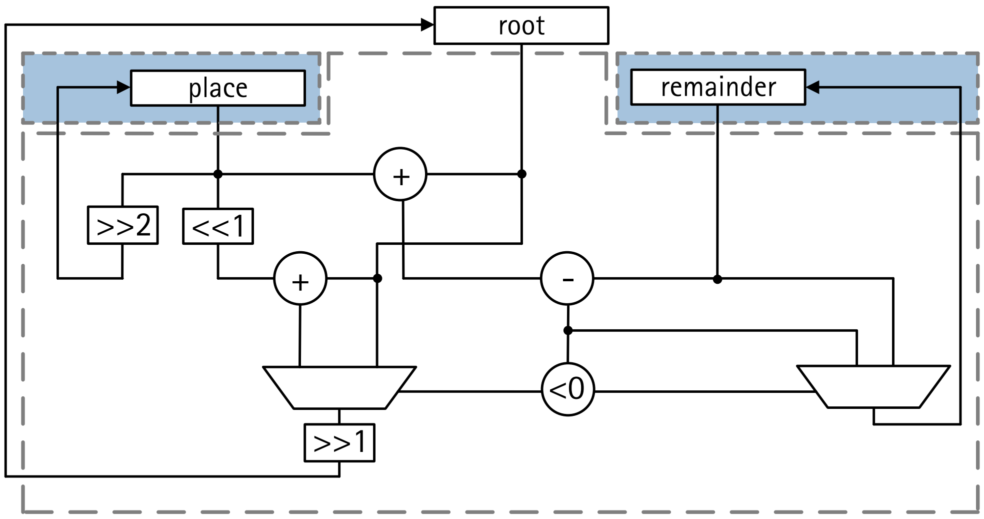


Figure 6 Special functional unit for an iterative computation of the integer square root. After initializing the process variables *place* and *remainder*, the operation has to be executed $\frac{n}{2}$ times to compute the result of a n bit number.

presented later. The FIR-unit provides an accuracy of 8 bit integer and 8 bit fraction, which is sufficient to ensure the precision of later computations.

4.3 Calculation of Orientation

During the orientation assignment, the pixel wise image orientation representation has to be changed from the cartesian to the polar form. For each conversion, the magnitude (Eq. 6) and the angle (Eq. 7) have to be computed by a square root and an arctangent operation, respectively.

Magnitude Representation - For computing the magnitude of the representation conversion, the integer square root (ISQRT) is utilized in combination with a range adjustment. The integer square root is defined as the largest integer smaller than the full precision square root. With the iterative method from [9], full accuracy can be assured. After an initialization step and assuming an n bit input number, the two cycle special functional unit needs $\lceil \frac{n}{2} \rceil$ iterations to compute the ISQRT.

The `ISQRT` unit has three input variables (see Fig. 6) of which *place* and *remainder* are process variables and need to be initialized. By using states for the process variables, the iterative process doesn't suffer from long memory access. The workload in one iteration of three shifts, two additions, one subtraction and one comparison for multiplexing are executed in one cycle.

Angle Representation - The computation of the orientation direction is based on the approximation of the arctangent. In [2], an octant approximation $atan(\frac{x}{y}) = atan_{octant}$ is presented in which four different second-order polynomial approximations are used to determine the angle:

$$\begin{aligned}
atan_{1,8} &= \frac{y}{x}\left[1.0584 - sign(y)\cdot 0.273\cdot \frac{y}{x}\right]\\
atan_{2,3} &= \frac{\pi}{2} - \frac{x}{y}\left[1.0584 - sign(x)\cdot 0.273\cdot \frac{x}{y}\right]\\
atan_{4,5} &= sign(y)\cdot\pi + \frac{y}{x}\left[1.0584 + sign(y)\cdot 0.273\cdot \frac{x}{y}\right]\\
atan_{6,7} &= -\frac{\pi}{2} - \frac{x}{y}\left[1.0584 - sign(x)\cdot 0.273\cdot \frac{x}{y}\right] \qquad (12)
\end{aligned}$$

The special functional unit (see Fig. 7) has a two-stage structure. In the first step, the four polynomials are evaluated in parallel. The correct value is selected in the second step. This selection is based on the octant of the two input values.

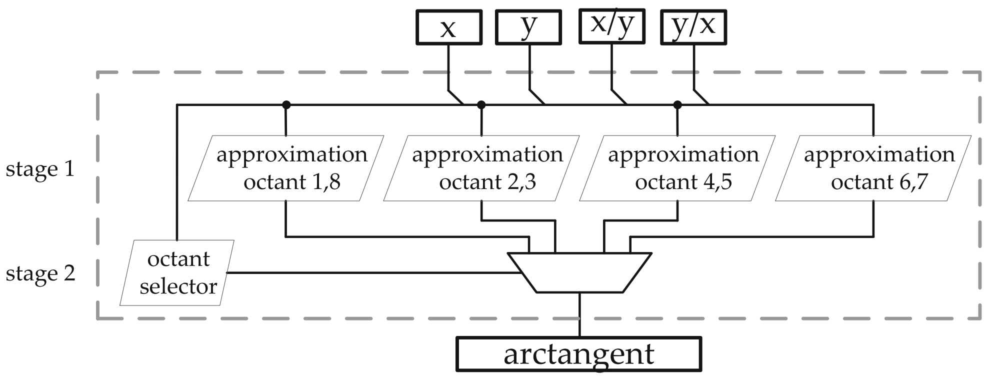


Figure 7 Special functional unit for the two stages computation of the arctangent. In the first step four second-order polynomial approximations are computed in parallel. Afterwards depending of the input value's octant, the correct value is selected.

The four polynomials require two divided values as input values. These computationally intensive operations are accelerated by special Tensilica dividers. The ATAN-unit itself needs one cycle for a 32-bit integer result, which is an angle representation.

4.4 Calculation of Descriptor

The last step of the SIFT feature extraction is the generation of the descriptor. The descriptor is composed of multiple histogram entries. Based on the polar representation of the image gradient, the histogram entries and their position in the histogram have to be computed. To accelerate the computation, two special functional units HIST_VAL and HIST_POS have been included in the pipeline. Furthermore, each histogram is rotated by its previously computed orientation. This computation step requires the computationally expensive functions sine and cosine. To accelerate these functions, the Xtensa LX5 processor is extended with the special functional unit SINE_INIT and SINE for trigonometrical computations.

Calculation of Histogram Values - To avoid boundary effects in which the descriptor abruptly changes, trilinear interpolation is used to distribute the value of each gradient sample into the adjacent histogram samples. Therefore, each histogram entry is multiplied by a weight of $1 - d$ for each dimension (row, column and orientation), where d is the distance of the sample from the central value of the histogram entry. Three dimensions result in eight possible Gaussian weighted histogram values:

$$\begin{aligned}
coef[1] &= mag \cdot d\_r \cdot d\_c \cdot d\_o\\
coef[2] &= mag \cdot d\_r \cdot d\_c \cdot (1-d\_o)\\
coef[3] &= mag \cdot d\_r \cdot (1-d\_c) \cdot d\_o\\
coef[4] &= mag \cdot d\_r \cdot (1-d\_c) \cdot (1-d\_o)\\
coef[5] &= mag \cdot (1-d\_r) \cdot d\_c \cdot d\_o\\
coef[6] &= mag \cdot (1-d\_r) \cdot d\_c \cdot (1-d\_o)\\
coef[7] &= mag \cdot (1-d\_r) \cdot (1-d\_c) \cdot d\_o\\
coef[8] &= mag \cdot (1-d\_r) \cdot (1-d\_c) \cdot (1-d\_o)
\end{aligned}$$

where mag is the Gaussian weight and d_r, d_c and d_o depict the distances of the three dimensions row, column and orientation, respectively. All histogram coefficients can be computed in parallel by implementing a tree of 14 multipliers with max. two multipliers in chain. In this case, 14 multipliers have to be implemented. A reasonable trade-off between computation time versus resource usage is a two step computation for the histogram values. In Fig. 8, the special functional unit HIST_VAL is shown, which occupies two stages in the processor pipeline. The special functional unit requires four input values, which are routed through a connection matrix for the first multiplication stage, which consists of eight multipliers. The results are fed back to the connection matrix as inputs for the second multiplication stage. As depicted, each multiplier has two multiplexers to choose the correct input value. By accepting two cycles for this computation, six multipliers can be saved compared to a one cycle solution of this computation.

Calculation of Histogram-Positions - Not every histogram coefficient is selected to fit in the resulting descriptor. Dependent of the position in the subregions, a histogram coefficient is accepted or declined, which leads to intense branching in the computation. To avoid performance loss due to branching, a validation mask is generated for each set of eight histogram coefficients that are mentioned above. In the special functional unit HIST_POS, the position of a histogram coefficient and its validation is computed. With the widely used parameter set of [21], the histogram consists of 128 entries 8 bit each. To decode 128 entries in a histogram, 7 bits are required. In addition there is one validation bit needed (see Fig. 9). To avoid branching, the memory area for the histogram is doubled to 256 entries. The invalid coefficients are accumulated in the lower 128 entries of the histogram, which are skipped in the later SIFT processing. Only the valid entries are accumulated in the correct positions.

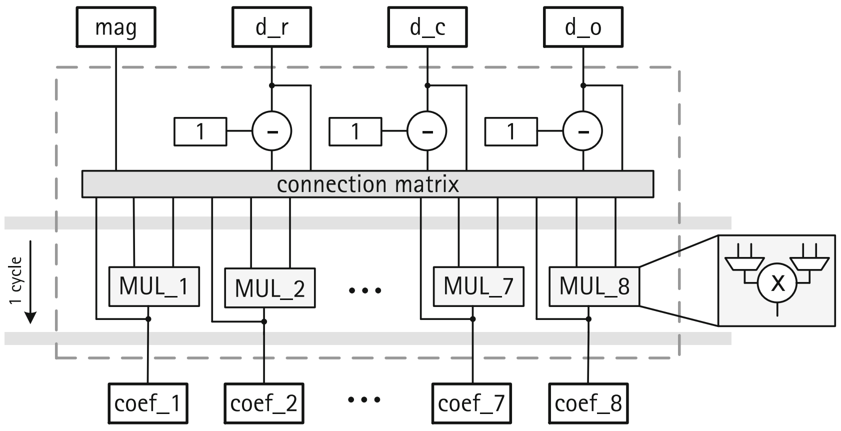


Figure 8 Special functional unit for computing the histogram entries. The four input unit consists of eight multipliers and needs two cycles to generate eight coefficients of 8 bits each. The results of the first multiplication step are fed back to the connection matrix.

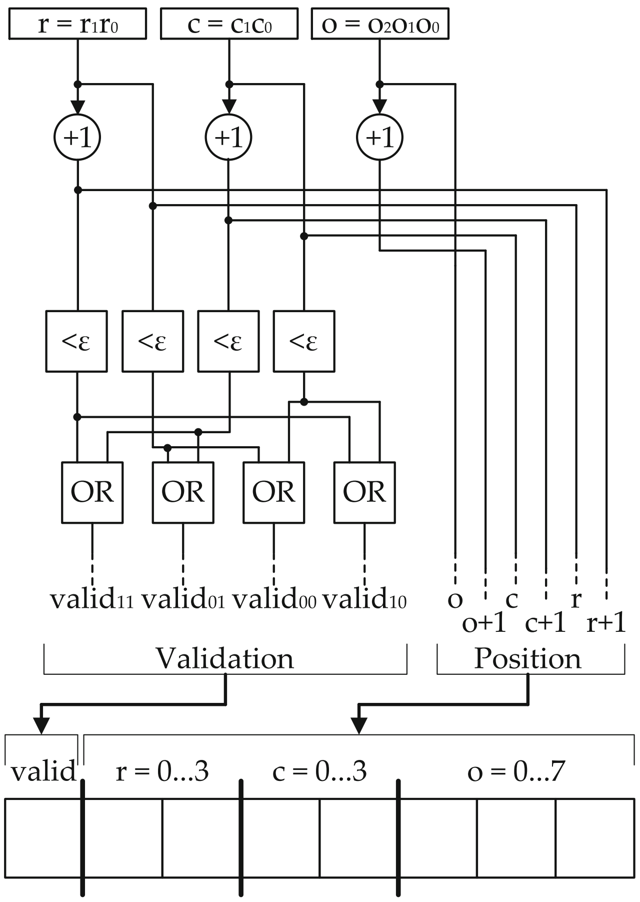


Figure 9 Special functional unit for computing the histogram position and its validations. The position in the histogram is generated by the three dimensions of the trilinear interpolation. By comparing thresholds the validation of each position is ensured.

Sine Approximation - The orientation's sine and cosine are required for the computation of the histogram rotation. To accelerate the computation of those trigonometric functions, the processor is extended with the special functional unit SINE_INIT and SINE (see Fig. 10a and b). To save resources, the similarity between sine and cosine and the geometry of the sine function are exploited. The control input *mode* can toggle between sine and cosine computation. Furthermore, only the positive half-wave of the sine is determined and the resulting sign of the value is set by a range-multiplexer. The sine approximation is a quadratic polynomial approximation computed in two stages to gain more precision:

$$temp = abs(inVal) \cdot \left(\frac{4}{\pi} - \frac{4}{\pi^2} \cdot abs(inVal)\right) \tag{13}$$

$$sine = 0.225 \cdot \left(temp^2 - temp\right) + temp \tag{14}$$

The input value $inVal$ for this operation is a scaled float number, which is converted to an integer. The result of this operation is a 32-bit integer number. It takes two execution stages in the pipeline to perform the computation. The initialization step consists of the mode selection (sine/cosine) and the preparation of the input value for the computation (see Fig. 10a). The first approximation step is performed as well. Due to the low complexity of the initialization and the resulting short execution time, it is useful to move the MAC-operation of Eq. 13 to the initialization step. Thereby, an additional execution step in the pipeline is avoided, however, both functional units are highly interlaced and cannot be used standalone in different context. For fast data accesses in the sine computation itself, all necessary values are stored in states. As shown in Fig. 10b, the remaining operations for the sine computation are distributed to two execution stages and to the register-write stage of the pipeline.

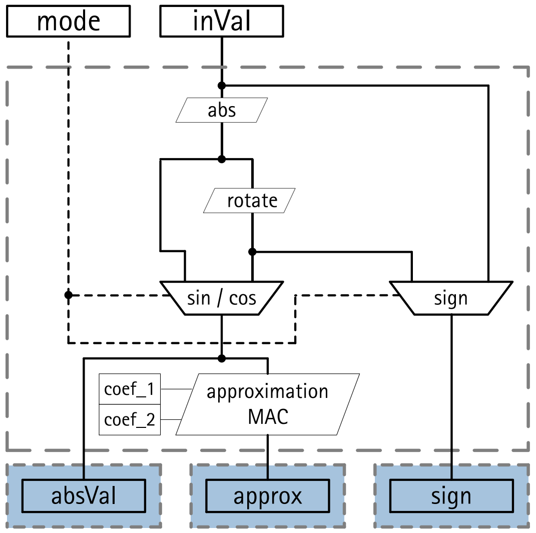
5 Evaluation

This section is divided into five parts. First, the speed-up provided by the proposed arithmetic functional units is presented. Based thereon, in the second part the speed-up for the four major algorithmic steps are discussed. In part three, the relation of image size, number of feature-points and number of execution cycles is presented. In part four, the gate count for the extended processor is shown. Finally, the section ends with the presentation of an efficiency measure for the SIFT-ASIP.

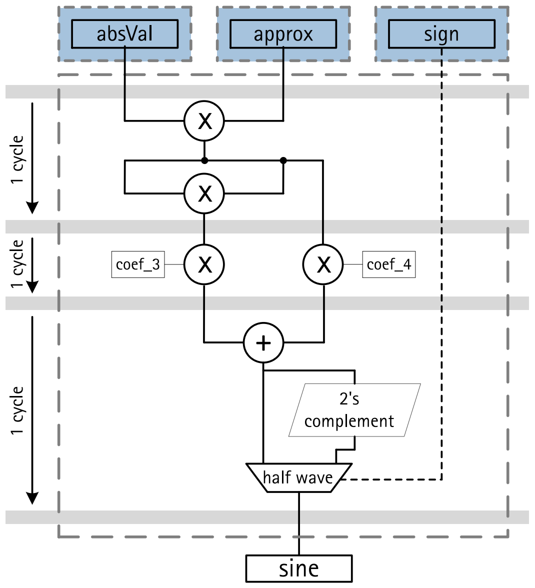
5.1 Acceleration of Arithmetic Functions

In Table 2, an overview of the special functional units of accelerated arithmetic functions is presented. To show the speed-up of the accelerated arithmetic functions, the extensions are compared with the standard libraries and the reference code, executed on the baseline processor. The baseline processor makes no use of a single precision floating point unit, no special multiplier nor divider units. Each function call for the accelerated implementation invokes at least one special functional unit and some data pre- and post-processing for the faster computation of an arithmetic function.

The arctangent computation and the integer square root computation are invoked approximately 2,100 times in average for one detected feature-point. Depending on the local neighborhood of a feature-point, a varying number of pixel is included for the calculations and consequently, a deterministic cycle count cannot be given. Reducing the cycle count from 5,552 cycles to 30 cycles for one arctangent computation and from 366 cycles to 22 cycles for one integer square root computation, the total cycle count could be reduced by 12.3 Mio. cycles per feature just for these two computations.



(a) Special functional unit for initialization of the sine and cosine computation. In addition to the initialization step, the MAC-operation of equation 13 is executed by this functional unit. Reused data is stored in states for fast access by the following special functional unit.



(b) Special functional unit for computing sine and cosine. All operations are performed in two execution stages and the register-write stage of the pipeline. The input's sign is used to determine which half-wave is selected as output.

Figure 10 Special functional units for computing sine and cosine - initialization step and computation step. **a**. Special functional unit for initialization of the sine and cosine computation. In addition to the initialization step, the MAC-operation of Eq. 13 is executed by this functional unit. Reused data is stored in states for fast access by the following special functional unit. **b** Special functional unit for computing sine and cosine. All operations are performed in two execution stages and the register-write stage of the pipeline. The input's sign is used to determine which half-wave is selected as output.

The processor extension for the sine function reduces the sine computation to 6 cycles instead of 3,772 cycles, which means a speed-up factor of x629. With a number of 500 feature-points for one image, this instruction-set extension saves 1.9 Mio. cycles per frame, compared to the baseline processor.

The generation of the histogram values for the descriptor is highly dependent on the extracted number of feature-points. Furthermore, the region size out of which the histogram is generated, depends on the Gaussian scale, in which the feature-point was detected. Therefore, the number of invocations for histogram generation varies from feature-point to feature-point. On average, the special functional units for the histogram generation are invoked approximately 1,500 times for one feature-point. With a number of 500 feature-points in one image, this results in 750,000 invocations per frame. With a speed-up of x89 for the histogram generation, the number of cycle reduction has a great impact on the SIFT feature extraction.

Table 2 Cycle count for accelerated arithmetic functions for the Xtensa LX5 processor compared with the cycle count for the baseline processor.

Function	C-Reference [#cycles]	Accelerated [#cycles]	Speed-Up	#Invocations (per feature)
Arctangent	5,552	30	185	$\sim 2,100$
Integer square root	366	22	16	$\sim 2,100$
Sine	3,772	6	629	2
Histogram generation	$\sim 16,000,000$	$\sim 180,000$	89	$\sim 1,500$
Gaussian Filtering	4,906,469,155	2,595,499	1,890	1

For each function, the cycle count of one invocation is presented. Furthermore, the average number of invocations for one feature-point is given. The resolution of the input image for the Gaussian Filtering is 800 × 640 pixel. The filter size of the Gaussian filtering is 15 × 15.

Table 3 Cycle count for SIFT feature extraction for the baseline processor and the proposed extended processor.

Algorithm	Baseline Processor	Extended Processor	Speed-Up
Building of Pyramids	48,336,848,554	41,023,999	1,178
Feature-point Detection	999,349,517	216,614,138	5
Orientation Assignment	1,527,402,586	28,013,088	55
Feature Description	9,053,651,191	195,270,505	46
Total	59,917,251,848	480,921,730	125

The resolution of the input image is 800 × 640 pixel.

The special functional unit for a `FIR` filter has to be evaluated for a complete Gaussian filtering to make it comparable. By processing an image of 800 × 640 pixel with a filter size of 15 × 15 for one selected filter step, the extended processor achieves a speed-up of x1,890 as opposed to the non-accelerated code execution for the same filter step. In total, the Gaussian pyramid consists of five different filterings. The overall speed-up for the computation of the Gaussian pyramid is presented in the following section.

5.2 Acceleration of Algorithmic Steps

In Table 3, the cycle count for the implementation of the SIFT feature extraction on the baseline processor and the extended processor is presented. The baseline processor is a core configuration without a floating point unit or other Tensilica provided accelerators. The resolution of the input image is 800 × 640 pixel, which results in seven octaves for the Gaussian pyramid. Each algorithmic step according to Section 3 is evaluated on its own. In addition, the overall speed-up of the SIFT feature extraction is given.

Utilizing all the special functional units presented above, the resulting speed-ups are x1178 for the construction of the image pyramids, x5 for detection of the feature-points, x55 for the orientation-assignment and x46 for the generation of descriptors. Compared to the SIFT implementation on the basic Xtensa LX5 core, the number of total processing cycles, and thus the processing speed, could be reduced by a factor of over x125 for the extended processor.

The increased speed-up for pyramid building compared to the speed-up of the remaining algorithmic steps can be explained by the fact, that just the first algorithmic step consists of a very regular calculation. In the remaining algorithmic steps, irregular memory accesses and strong branching prohibit a larger speed-up factor for this processor configuration.

5.3 Image Dependent Effects

The four arithmetic parts of the SIFT-algorithm can be divided into two categories: image size dependent and feature-point related. The building of pyramids and a small part of feature-point detection can be categorized as image size dependent computation time, whereas the cycle count of the orientation assignment, feature descriptor generation and parts of the feature-point detection are highly dependent of the extracted number of feature-point candidates.

To show the relation between different image sizes and a varying number of extracted feature-points, three different test cases are presented:

1. 800 × 640 pixel, 1 feature-point
2. 800 × 640 pixel, 256 feature-points
3. 320 × 240 pixel, 256 feature-points

With this test configuration, the partition between image size dependent cycle count and feature-point related cycle count can be shown (see Table 4).

The cycle count for the building of pyramids is proportional to the number of pixels for the later pyramids. The factor of x6.6 between the image sizes of the different configurations can be found approximately by comparison of

Table 4 Cycle count of the extended processor for the four algorithmic steps for different sizes of the input image in relation to the number of found feature-points.

	320 × 240, 256 FP	800 × 640, 256 FP	800 × 640, 1 FP
Building of Pyramids	6,280,360	41,023,999	41,023,999
Feature-point Detection	31,111,397	216,614,138	216,283,876
Orientation Assignment	24,082,018	28,013,088	94,243
Feature Description	190,521,195	195,270,505	1,321,832
Total	251,994,970	480,921,730	258,723,950

the cycle count. The feature-point detection is also strongly dependent on the input image size. In addition, there is a small cycle count, dependent on the number of found feature-points, which is below 1 % of the total cycle count for this algorithmic step and hence, can be ignored. Comparing the cycle count for the orientation assignment and the feature description, it can be seen that this calculation step is proportional to the number of found feature-points. The difference between the cycle count of the small and large input image with identical number of feature-points results from the different input images. Because of the images' differing content, the size of the local neighborhood of pixels for the orientation assignment differs for those two test cases.

5.4 Silicon Area - Equivalent Gate Count

The equivalent gate count (without memories) for using a 45 nm standard CMOS technology is presented in Table 5. Due to the architectural extensions, a total area increase from 93,000 gates to 456,976 gates is observed, which is a factor of x4.9. The detailed breakdown of the gate count relies on the pre-synthesis estimations, provided by Tensilica tools.

One special functional unit for computing the FIR-filtering requires 20,389 gates. For parallel image processing and the reduction of memory accesses, the processor is extended with four of those units, which results in 26 % of the total additional area. In total, there are 56 multipliers and 56 adders mapped to 81,556 gates for the `FIR`-units.

For the `HIST_VAL`-unit, an additional gate count of 125,223 gates is required. The functional unit consists of eight multipliers, which are shared in two consecutive execution stages of the pipeline. This sharing reduces the required silicon area by prohibiting the execution of consecutive `HIST_VAL` instructions. Even though the number of multipliers in the `FIR`-unit is four times larger than the number in the `HIST_VAL`-unit, the gate count of the `HIST_VAL`-unit is 1.5 times larger than the gate count of the `FIR`-units. This large difference can be explained by the fact, that the multipliers in the `FIR`-units have constant values as inputs. In contrast to conventional multipliers, basic arithmetic functions with constant inputs can be synthesized with a much smaller gate count.

In Table 6, the synthesis results of the Synopsys design tools for 45 nm standard CMOS technology are shown. Each processor setup related to the single algorithmic steps can perform as a standalone processor and includes some identical overhead for full functionality, e.g., the `GRF` register file. Therefore, the area results differ from the estimated results by the Tensilica design tool. For a TSMC 45 nm technology, the processor with all extensions reaches a frequency of 400 MHz and uses silicon area of 0, 591 mm^2.

Depending of the number of extracted feature-points the accelerated processor will reach a frame rate of 1.58 fps for an input image of 320 × 240 pixel or appr. 1 fps for an input image of 800 × 640 pixel. The heterogeneous system the of [6] reaches a frame rate of 30 fps for an image size of 320 × 240 pixel and therefore, it outperforms the ASIP presented in this work, but the ASIP features full flexibility for the implementation of further image feature extractors.

Table 5 Core area requirements of the accelerated ASIP for a generic 45nm GS process - estimations by Tensilica design tools.

Category	Extension	Area [Gates]	
Register Files	Standard RF	1,196	
	Custom RF	16,958	5 %
States		33,804	9 %
Data Handling	LOAD64	257	< 1 %
	STORE64	257	< 1 %
	MOVE64	53	< 1 %
	SHIFT64	0	0 %
Special Functional Unit (Base CV)	FIR (x4)	81,556	22 %
Basic Functional Unit (SIFT)	ISQRT_INIT	0	0 %
	ISQRT	567	< %
	ATAN	19,463	5 %
	SINE_INIT	5,485	2 %
	SINE	19,606	5 %
	HIST_POS	1,150	< 1 %
	HIST_VAL	125,223	34 %
Total additional area		363,976	100 %
LX5 baseline processor		93,000	
Total Area		456,976	

Table 6 Core area requirements of the accelerated ASIP for a TSMC 45nm GS process - synthesis with Synopsys design tools.

Core Configuration	Area [Gates]	Area [mm^2]	Max. Freq. [MHz]
Baseline Porcessor	77,440	0.120	625
Building of Pyramids	227,522	0.355	416
Feature-point Detection	95,123	0.148	416
Orientation Assignment	107,649	0.168	500
Feature Description	216,544	0.338	500
Full Core	379,087	0.591	400

Table 7 Speed-up and relative additional hardware to determine the efficiency of the instruction set extension of the extended processor. A value of 1 expresses the meaning off no speed-up with no extra hardware resources. For an overview of the available special functional units, see Table 1.

Configuration	Speed-Up	Add. Hardware	Efficiency ϵ
Baseline Processor	1	1	1
+ B-ISE[a]	1.01	1.24	0.82
+ B-ISE + FIR	5.53	3.19	1.73
+ B-ISE + FIR + ISQRT[b] + ATAN	6.42	3.59	1.79
+ B-ISE + FIR + ISQRT + ATAN + SINE + HIST[c]	124.59	5.40	23.06

[a]B-ISE - Baseline Instruction Set Extension: new register file, defined stages and special functional units for data handling.

[b]ISQRT - ISQRT_INIT and ISQRT.

[c]HIST - HIST_VAL and HIST_POS.

5.5 Efficiency

In order to evaluate the quality of the extended processor, the efficiency $\epsilon = \frac{benefit}{effort}$ is determined. The *profit* equals the reached speed-up by the presented instruction set extensions (ISE) (see Table 1), whereas the *effort* reflects the additional hardware resources, that have to be spent by utilizing the special functional units.

By definition, the baseline processor features an efficiency $\epsilon = 1$. With an initial effort of the gate count of baseline processor, the reached speed-up is 1. In relation to the execution cycles of the baseline processor, it is possible to calculate a speed-up for the SIFT feature extraction, taking into account which part of the instruction set extension is used. Furthermore, the additional hardware depicts the relative additional hardware of each accelerated algorithmic step, compared to the base processor (see Table 7).

The base instruction set extension (B-ISE), which consists of the stages and new register file, reaches an efficiency of 0.82. This value can be explained by the fact, that additional hardware is necessary, but there is no considerable speed-up reached. The B-ISE is just the groundwork for the following ISE and is not intended to accelerate the SIFT feature extraction. By adding the FIR-ISE to the basline processor and the B-ISE, the efficiency rises to 1.73, which confirms, that the acceleration of the SIFT execution outweighs the additional hardware consumption. Furthermore, the additional processor extension with the ISQRT-ISE and ATAN-ISE increases the efficiency slight. The fully extended core with all instruction set extensions is 23.06 times more efficient compared to the baseline processor, which results from a speed-up of 124.59 by using 5.40 times more hardware than to the initially core. By this efficiency measure, the impact of new instruction set extensions can be evaluated in relation to the additional hardware, which is an important aspect in evaluating the quality of the extended processor.

6 Conclusions

In this paper, the design and implementation of a suitable instruction-set extension for SIFT feature extraction based on the Tensilica Xtensa LX5 ASIP is presented. The results show, that the proposed hardware extensions lead to a significant speed-up factor of x125 when compared with the base processor. The speed-up was obtained by considering several techniques while designing the processor extensions. By reducing the number of operations, avoiding branching and eliminating memory accesses, the number of cycles could be significantly reduced. This improvement can be achieved while maintaining the flexibility of the enhanced processor by implementing general hardware extensions as well as a specialized set of SIFT-specific instructions. In total, the silicon area requirement increases by a factor of x4.9 while the number of cycles decreases by a factor of x125 for an input image with a resolution of 800 $\times$ 640 pixel.

Acknowledgments This work was partially supported by the European Commission under the ECSEL Joint Undertaking in the scope of the DESERVE [11] project.

References

1. Alahi, A., Ortiz, R., & Vandergheynst, P. (2012). Freak: fast retina keypoint. In *2012 IEEE conference on computer vision and pattern recognition (CVPR)*, IEEE (pp. 510–517).
2. Alvarez, J.S. (2012). *Streamlining digital signal processing: a tricks of the trade guidebook*: Wiley - IEEE Press.
3. Banz, C., Dolar, C., Cholewa, F., & Blume, H. (2011). Instruction set extension for high throughput disparity estimation in stereo image processing. In *Application-specific Systems, architectures and processors (ASAP)*, IEEE (pp. 169–175).
4. Bay, H., Ess, A., Tuytelaars, T., & Van Gool, L. (2008). Speeded-up robust features (SURF). *Computer Vision and Image Understanding*, *110*(3), 346–359.

5. Beucher, N., Blanger, N., Savaria, Y., & Bois, G. (2009). High acceleration for video processing applications using specialized instruction set based on parallelism and data reuse. *Signal Processing Systems*, *56*(2-3), 155–165.
6. Bonato, V., Marques, E., & Constantinides, G.A. (2008). A parallel hardware architecture for scale and rotation invariant feature detection. *IEEE Transactions on Circuits and Systems for Video Technology*, *18*(12), 1703–1712.
7. Calonder, M., Lepetit, V., Strecha, C., & Fua, P. (2010). BRIEF: Binary Robust Independent Elementary Features. In *Proceedings of the 11th European conference on computer vision: part IV, ECCV'10* (pp. 778–792). Berlin, Heidelberg: Springer.
8. Chiu, L.C., Chang, T.S., Chen, J.Y., & Chang, N.Y.C. (2013). Fast SIFT design for real-time visual feature extraction. *IEEE Transactions on Image Processing*, *22*(8), 3158–3167.
9. Crenshaw, J.W. (1998). Integer square roots. http://www.embedded.com/electronics-blogs/programmer-s-toolbox/4219659/Integer-Square-Roots.
10. Deng, W., Zhu, Y., Feng, H., & Jiang, Z. (2012). An efficient hardware architecture of the optimised SIFT descriptor generation. In D. Koch, S. Singh, & J. Trresen (Eds.), *FPL*, IEEE (pp. 345–352).
11. DESERVE: DEvelopment platform for safe and efficient dRiVE (2012). http://www.deserve-project.eu.
12. Duan, L.Y., Gao, F., Chen, J., Lin, J., & Huang, T. (2013). Compact descriptors for mobile visual search and mpeg cdvs standardization. In *2013 IEEE international symposium on circuits and systems (ISCAS)*, IEEE (pp. 885–888).
13. Ekmekcioglu, E., Worrall, S., Velisavljevic, V., De Silva, D., & Kondoz, A. International Organisation for Standardisation ISO/IEC JTC1/SC29/WG11 Coding of Moving Pictures and Audio.
14. Fenzi, M., Ostermann, J., Mentzer, N., Payá-Vayá, G., Blume, H., Nguyen, T.N., & Risse, T. (2014). Asev?automatic situation assessment for event-driven video analysis. In *2014 11th IEEE international conference on advanced video and signal based surveillance (AVSS)*, IEEE (pp. 37–43).
15. Fontaine, S., Goyette, S., Langlois, J.M.P., & Bois, G. (2008). Acceleration of a 3D target tracking algorithm using an application specific instruction set processor. In *ICCD*, IEEE (pp. 255–259).
16. Gonzalez, R.E. (2000). Xtensa: a configurable and extensible processor. *IEEE Micro*, *20*(2), 60–70.
17. Hartley, R., & Zisserman, A. (2003). *Multiple view geometry in computer vision*, 2nd edn. New York: Cambridge University Press.
18. Huang, F.C., Huang, S.Y., Ker, J.W., & Chen, Y.C. (2012). High-Performance SIFT hardware accelerator for real-time image feature extraction. *IEEE Transactions on Circuits and Systems for Video Technology*, *22*(3), 340–351.
19. Ke, Y., & Sukthankar, R. (2004). Pca-sift: A more distinctive representation for local image descriptors. In *Proceedings of the 2004 IEEE computer society conference on computer vision and pattern recognition, CVPR 2004*, IEEE, (Vol. 2 pp. II–506).
20. Leutenegger, S., Chli, M., & Siegwart, R.Y. (2011). BRISK: Binary Robust Invariant Scalable Keypoints. In *Proceedings of the 2011 international conference on computer vision, ICCV '11* (pp. 2548–2555). Washington: IEEE Computer Society.
21. Lowe, D.G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, *60*(2), 91–110.
22. Mentzer, N., Pay-Vay, G., Blume, H., von Egloffstein, N., & Ritter, W. (2014). Instruction-Set extension for an ASIP-based SIFT feature extraction. In *International conference on embedded computer systems: architectures, modeling, and simulation (SAMOS XIV), 2014*.
23. Mikolajczyk, K., & Schmid, C. (2005). A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *27*(10), 1615–1630.
24. Ravasi, M., Tenze, L., & Mattavelli, M. (2002). A scalable and programmable architecture for 2-d dwt decoding. *IEEE Transactions on Circuits and Systems for Video Technology*, *12*(8), 671–677.
25. Rosten, E., Porter, R., & Drummond, T. (2010). Faster and better: a machine learning approach to corner detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *32*(1), 105–119.
26. Rublee, E., Rabaud, V., Konolige, K., & Bradski, G. (2011). Orb: an efficient alternative to sift or surf. In *2011 IEEE International conference on computer vision (ICCV)*, IEEE (pp. 2564–2571).
27. Systems, C.D. (2014). Xtensa LX5 microprocessor data book. Cadence Design Systems.
28. Wang, G., Rister, B., & Cavallaro, J.R. (2013). Workload analysis and efficient opencl-based implementation of sift algorithm on a smartphone. In *Proceedings in IEEE global conference signal and information processing (GlobalSIP)* (pp. 759–762).
29. Wang, W., Zhang, Y., Guoping, L., Yan, S., & Jia, H. (2013). *Clsift: An optimization study of the scale invariance feature transform on gpus*. *IEEE*, (pp. 93–100).
30. Wu, C. (2007). SiftGPU: a GPU implementation of scale invariant feature transform (SIFT). http://cs.unc.edu/ccwu/siftgpu.
31. Yonglong, Z., Kuizhi, M., Xiang, J., & Peixiang, D. (2013). Parallelization and optimization of sift on gpu using cuda. In *2013 IEEE 10th international conference on high performance computing and communications & 2013 IEEE international conference on embedded and ubiquitous computing (HPCC_EUC)*, IEEE (pp. 1351–1358).
32. Zhang, Q., Chen, Y., Zhang, Y., & Xu, Y. (2008). Sift implementation and optimization for multi-core systems. In *IEEE international symposium on parallel and distributed processing, IPDPS 2008*, IEEE (pp. 1–8).

Nico Mentzer received his diploma in electrical engineering from Leibniz Univerität Hannover, Germany, in 2011. Since then he is working as a research engineer at the Institute of Microelectronic Systems in Hanover, Germany. Currently, he is working towards a Ph.D. degree in the field of algorithms and architectures for digital image processing.

Guillermo Payá-Vayá obtained his Ing. degree from the School of Telecommunications Engineering, Universitat Politécnica de Valencia, Spain, in 2001. During 2001-2004, he was a member of the research group of Digital System Design, Universitat Politécnica de Valencia, where he worked on VLSI dedicated architecture design of signal and image processing algorithms using pipelining, retiming, and parallel processing techniques.

In 2004, he joined the Department of Architecture and Systems at the Institute of Microelectronic Systems, Leibniz Universität Hannover, Germany, and received a Ph.D. degree in 2011. He is currently Junior Professor with the Institute of Microelectronic Systems, Leibniz Universität Hannover, Germany. His research interests include embedded computer architecture design for signal and image processing systems.

Holger Blume received his diploma in electrical engineering in 1992 at the University of Dortmund, Germany. In 1997 he achieved his Ph.D. with distinction from the University of Dortmund, Germany. Until 2008 he worked as a senior engineer and as an academic senior councilor at the Chair of Electrical Engineering and Computer Systems (EECS) of the RWTH Aachen University.

In 2008 he got his postdoctoral lecture qualification. Holger has been Professor for 'Architectures and Systems' at the Leibniz Universität Hannover, Germany, since July 2008 and manages the Institute for Microelectronic Systems. His present research includes algorithms and heterogeneous architectures for digital signal processing, design space exploration for such architectures as well as research on the corresponding modeling techniques.