Journal Pre-proof

A Fully Pipelined FPGA Accelerator for Scale Invariant Feature Transform Keypoint Descriptor Matching

Luka Daoud, Muhammad Kamran Latif, H.S. Jacinto, Nader Rafla

PII: S0141-9331(19)30080-8
DOI: https://doi.org/10.1016/j.micpro.2019.102919
Reference: MICPRO 102919

To appear in: *Microprocessors and Microsystems*

Received date: 8 February 2019
Revised date: 16 September 2019
Accepted date: 16 October 2019

Please cite this article as: Luka Daoud, Muhammad Kamran Latif, H.S. Jacinto, Nader Rafla, A Fully Pipelined FPGA Accelerator for Scale Invariant Feature Transform Keypoint Descriptor Matching, *Microprocessors and Microsystems* (2019), doi: https://doi.org/10.1016/j.micpro.2019.102919

A Fully Pipelined FPGA Accelerator for Scale Invariant Feature Transform Keypoint Descriptor Matching

Luka Daoud, Muhammad Kamran Latif, H S. Jacinto, Nader Rafla

Electrical and Computer Engineering
Boise State University
Boise, ID 83725, USA

Abstract

The scale invariant feature transform (SIFT) algorithm is considered a classical feature extraction algorithm within the field of computer vision. The SIFT keypoint descriptor matching is a computationally intensive process due to the amount of data consumed. In this paper, we designed a fully pipelined hardware accelerator architecture for the SIFT keypoint descriptor matching. It was implemented and tested on a field programmable gate array (FPGA). The proposed hardware architecture is able to properly handle the memory bandwidth necessary for a fully-pipelined implementation and hit the roofline performance model achieving the potential maximum throughput. The fully pipelined matching architecture was designed based on consine angle distance approach. It was optimized for 16-bit fixed-point operations and implemented on hardware using Xilinx Zynq-based FPGA development board. Our proposed architecture showed a noticeable reduction of area resources compared with its counterparts in the literature maintaining high throughput by alleviating the memory bandwidth restrictions. The results showed reduction in device-resources up to 91% in LUTs and 79% of BRAMs. Our hardware implementation is 15.7× faster than the comparable software approach.

Keywords: Scale Invariant Feature Transform, SIFT, Matching algorithm, FPGA, Pipeline, Acceleration, High Level Synthesis, HLS.

1. Introduction

Object recognition using feature-based algorithms are generally computationally intensive. The scale-invariant feature transform (SIFT) algorithm proposed in 1999 by David Lowe [1], is a classical and well-known algorithm within the field of computer vision. SIFT algorithm is a feature-based algorithm that can be applied in object recognition. The best candidate match for a SIFT keypoint is found by identifying its nearest-neighbor in the

Email addresses: LukaDaoud@u.boisestate.edu (Luka Daoud), MuhammadLatif@u.boisestate.edu (Muhammad Kamran Latif), SheltonJacinto@u.boisestate.edu (H S. Jacinto), nrafla@boisestate.edu (Nader Rafla)

keypoint database. The matching process often involves operating on data-at-rest but more recently real-time applications using feature-based object recognition have gained popularity. Feature extraction based object recognition is an approach commonly applied in several varying applications such as medical imaging [2], satellite imaging [3], facial recognition [4], and the landing of unmanned aerial vehicles (UAVs) [5].

Various steps in the extracting SIFT descriptors often require the use of complex software routines that require intensive computations [1]. However, in a running scenario of keypoint extraction, the extraction only occurs once per test image. The limitations of keypoint descriptor matching thus requires that matching must be performed every time a test image is compared with a possible match in the database. Each time the database needs to be accessed, the overall matching time for the test image increases as the overall size of the database grows.

The SIFT descriptor matching is based on the nearest-neighbor algorithm [1] where, for a single test keypoint descriptor match, the Euclidean distances [6] of the test descriptor are calculated between each descriptor in the descriptor database. The calculated distances are then sorted such that the minimum and second minimum distances are found. A positive match between the test descriptor and the descriptor database is found if the Euclidean distance ratio is above a pre-set threshold, suggested by David Lowe in [1].

Since a SIFT keypoint descriptor is an array of 128 elements, calculated based on all pixels of an image around the centered keypoint in a 16×16 sliding window. The generated descriptor by this method can be defined mathematically as:

$$d_k^\alpha = \{f_{k,1}^\alpha \ , \ f_{k,2}^\alpha \ , \ \ldots\ldots \ , \ f_{k,128}^\alpha\} \quad .$$

The Euclidean distance between two descriptors, d_k^α and d_m^β, is thus calculated:

$$\sum_{i=1}^{128} \frac{(f_{k,i}^\alpha - f_{m,i}^\beta)^2}{(f_{k,i}^\alpha + f_{m,i}^\beta)}.$$

In the process of matching a descriptor, d_k^α, with a database, the Euclidean distances of d_k^α in relation to the database's descriptors is calculated. The process of calculating Euclidean distances is computationally intensive however, resource consumption can effectively be reduced by changing the calculation of Euclidean distance. Instead of using a conservative approach of calculating the Euclidean distance as mentioned, a cosine angle distances can be calculated between the descriptors [7]. Since SIFT descriptors are normalized during keypoint extraction, calculating the angular distances by taking the arc-cosine of the dot-products of normalized descriptors prove to be a close approximation for Euclidean distances [7]. Utilizing a method of angular distance will significantly reduce the hardware resource consumption.

If an image of m descriptors is represented by a matrix of size $m \times 128$, there is a recurrent redundancy of memory access for descriptors of an image and the descriptor database. Memory access times for descriptors further vary based on locality thus, in a software approach, memory access time becomes variant that may impact on both timing and resource overheads for the SIFT descriptors matching.

In this paper, our proposed SIFT descriptors matching architecture is designed and implemented with the purpose of accelerating the matching process, handling the memory bandwidth limitation, and reducing the area resources. The contributions of this paper are summarized as following:

- A hardware implementation of the SIFT keypoint descriptor matching based on cosine angle distance on FPGA including:
  - A fully pipelined architecture.
  - Minimal resource utilization.
  - High throughput hardware accelerator.
- Resulting analysis of memory bandwidth usage and its effect on the overall computational performance.

The rest of this paper is organized as follows: Section 2 summarizes the literature review and the related work of SIFT descriptors matching on accelerating platforms. Section 3 provides background and related definitions along with the software approach of the matching algorithm based on calculating cosine angle distances. Section 4 studies the computation and memory bandwidth optimization. Section 5 presents our proposed matching architecture on FPGA. Section 6 evaluates our proposed matching architecture and provides the experimental results. Finally, Section 7 concludes the paper.

2. Related Work

This section particularly focuses on different approaches of calculating nearest-neighbor distances for descriptors matching algorithms on FPGA-based accelerators. It also provides a brief overview of the matching implementations on other platforms.

There have been several hardware-based implementations of descriptors matching on FPGA [8, 9, 10, 11]. Most recently, Vourvoulakis et al. [8] proposed an FPGA-based architecture for SIFT descriptors matching based on the calculation of the distances between the descriptors in the database. The similarity between the descriptors was determined based on the minimum value of SAD (Sum of Absolute Distances) calculators. Their implementation was based on comparing the currently extracted descriptor with 128 previously detected ones to find a potential match. The authors proposed a moving window of 16 descriptors to fit the entire matching architecture on an FPGA. In their implementation, a total 8 clock-cycles were required to calculate 128 SAD values to report a potential match using a single matching core that required significant memory resources.

Lentaris et al. [9] implemented a pipelined architecture for SIFT descriptor matching using the Euclidean norm for computing distances between descriptors. In their implementation, a finite state machine fetches all the descriptors from the test image $d^{\alpha}_{k,i}$ and the descriptors from the database $d^{\beta}_{k,i}$ stored in memory one by one. The descriptor pair is passed to a chi-square distance state, where the similarity of the two descriptors was evaluated by calculating the distance between them. The distance calculating state consists of

128 chi-square (χ^2) calculators and each calculator performs $(d^{\alpha}_{k,i} - d^{\beta}_{k,i})^2/(d^{\alpha}_{k,i} + d^{\beta}_{k,i})$ calculation where i is the i^{th} element of the 128-dimensional vector. Each multiplier and divider in chi-square state is 16-bit and produces a 32-bit result. The output from χ^2 calculators is summed using linear systolic array and the result is passed to matching state to keep tracking of the two best matches. At the end of the database, the distance ratio of these two matches is compared with a fixed threshold to accept or reject the best match. Their used technique of the matching algorithm by calculating the Euclidean distance necessitated more resources than our approach as explained in Section 6.

Wang et al. [10] proposed an embedded System-on-Chip for features detection and matching. Their system extracts binary robust independent elementary features (BRIEF) [12] descriptors from the detected SIFT ones. Unlike SIFT descriptors that has 128 elements, the BRIEF descriptor is a vector of 64 elements. The BRIEF matching detection was performed by calculating the distances between two BRIEF descriptors. A successful match is reported if the calculated distance is smaller than a minimum threshold [10].

Kapela et al. [11] presented a hardware-software platform in which fast retina keypoint (FREAK) [13] descriptors were extracted in software and matched by calculating the Hamming distance which was implemented on Xilinx Zynq-7000 FPGA. Their proposed matching core included multiple Hamming distance calculator circuits that are running in parallel to calculate the distance between the descriptors. The overall performance of their system depends on the number of the Hamming distance cores. Additionally, the number of LUTs and registers increases proportionally with the number of Hamming calculators.

Condello et al. [14] presented an OpenCL-based feature matching algorithm that made use of the capabilities of GPUs to speedup the matching process for speeded-up robust feature (SURF) descriptors. The matching algorithm uses Euclidean distances to calculate the nearest neighbors for a test descriptors with the others in the database. They implemented their matching core on NVIDIA's GTX275, which has a theoretical peak of 2760 GFlops. However, the latency of the global memory access affected on the computation power of the GPU where it limited the memory reuse during the distance computation step of the matching process.

Fassold et al. [15] used NVIDIA's Tesla K20 GPU for the SIFT descriptor matching by calculating the nearest-neighbors between descriptors using Euclidean distance. Their implementation of the matching architecture on the GPU achieved 13 milliseconds for a set of 2,800 descriptors.

The matching algorithm for most of the implementations is based on calculating the nearest-neighbor distances between the current feature and the features in the database. To the best of our knowledge, this paper is the first attempt for hardware implementation of SIFT matching algorithm on FPGA, where the matching technique is based on calculating the nearest-neighbor distances using cosine angle distance rather than using the traditional descriptor distance calculations. The following part of this paper moves on to describe in greater detail the SIFT matching algorithm based on cosine angle distance technique.

3. Matching Algorithm based on Cosine Angle Distance

3.1. Nomenclature and Definitions

An image is a 2-D array of pixels that carry information and keypoint descriptors are highly distinctive features in an image. A SIFT descriptor is a vector of 128 elements that describe a scale-invariant local image region. It can be given as d_k^α, where k and α are the k^{th} descriptor in an image $\tilde{\alpha}$.

$$d_k^\alpha = \{f_{k,1}^\alpha \ , \ f_{k,2}^\alpha \ , \ \ldots\ldots \ , \ f_{k,128}^\alpha\},$$

where $f_{k,i}^\alpha$ is the i^{th} element of the k^{th} descriptor of image $\tilde{\alpha}$ and $(0 \leq \ f_{k,i}^\alpha \leq 1)$. So, descriptors of an image $\tilde{\alpha}$ that has a m set of descriptors is described as d^α:

$$d^\alpha = \begin{bmatrix} d_1^\alpha \\ d_2^\alpha \\ \vdots \\ d_m^\alpha \end{bmatrix} = \begin{bmatrix} f_{1,1}^\alpha & f_{1,2}^\alpha & f_{1,3}^\alpha & \cdots & f_{1,128}^\alpha \\ f_{2,1}^\alpha & f_{2,2}^\alpha & f_{2,3}^\alpha & \cdots & f_{2,128}^\alpha \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ f_{m,1}^\alpha & f_{m,2}^\alpha & f_{m,3}^\alpha & \cdots & f_{m,128}^\alpha \end{bmatrix}.$$

The dot-product operation of two descriptors, d_l^α and d_s^β, is denoted as $dp_{l,s}^{\alpha,\beta}$, calculated in Equation 1.

$$dp_{l,s}^{\alpha,\beta} = d_l^\alpha \odot d_s^\beta = \sum_{i=1}^{128} f_{l,i}^\alpha \cdot f_{s,i}^\beta \tag{1}$$

Thus, $dp_k^{\alpha,\beta}$ is a dot-product of the k^{th} descriptor of image $\tilde{\alpha}$, with each descriptor of image $\tilde{\beta}$, defined as

$$dp_k^{\alpha,\beta} = \begin{bmatrix} dp_{k,1}^{\alpha,\beta} \\ dp_{k,2}^{\alpha,\beta} \\ \vdots \\ dp_{k,n}^{\alpha,\beta} \end{bmatrix} = d_k^\alpha \odot \begin{bmatrix} d_1^\beta \\ d_2^\beta \\ \vdots \\ d_n^\beta \end{bmatrix} = \begin{bmatrix} d_k^\alpha \odot d_1^\beta \\ d_k^\alpha \odot d_2^\beta \\ \vdots \\ d_k^\alpha \odot d_n^\beta \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{128} f_{k,i}^\alpha \cdot f_{1,i}^\beta \\ \sum_{i=1}^{128} f_{k,i}^\alpha \cdot f_{2,i}^\beta \\ \vdots \\ \sum_{i=1}^{128} f_{k,i}^\alpha \cdot f_{n,i}^\beta \end{bmatrix}.$$

Therefore, the dot-product of all descriptors of image $\tilde{\alpha}$ and image $\tilde{\beta}$ can be denoted as $dp^{\alpha,\beta}$, defined by

$$dp^{\alpha,\beta} = \begin{bmatrix} dp_1^{\alpha,\beta} \\ dp_2^{\alpha,\beta} \\ \vdots \\ dp_m^{\alpha,\beta} \end{bmatrix} = \begin{bmatrix} dp_{1,1}^{\alpha,\beta} & dp_{2,1}^{\alpha,\beta} & dp_{3,1}^{\alpha,\beta} & \ldots & dp_{m,1}^{\alpha,\beta} \\ dp_{1,2}^{\alpha,\beta} & dp_{2,2}^{\alpha,\beta} & dp_{3,2}^{\alpha,\beta} & \ldots & dp_{m,2}^{\alpha,\beta} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ dp_{1,n}^{\alpha,\beta} & dp_{2,n}^{\alpha,\beta} & dp_{3,n}^{\alpha,\beta} & \ldots & dp_{m,n}^{\alpha,\beta} \end{bmatrix}^T.$$

In the SIFT matching algorithm the cosine inverse (arc-cosine), denoted by ci, of each dot-product operation is calculated. Similarly, $ci^{\alpha,\beta}$ is the arc-cosine of $dp^{\alpha,\beta}$, defined as

$$ci^{\alpha,\beta} = \begin{bmatrix} ci_1^{\alpha,\beta} \\ ci_2^{\alpha,\beta} \\ \vdots \\ ci_m^{\alpha,\beta} \end{bmatrix} = \begin{bmatrix} ci_{1,1}^{\alpha,\beta} & ci_{2,1}^{\alpha,\beta} & ci_{3,1}^{\alpha,\beta} & \dots & ci_{m,1}^{\alpha,\beta} \\ ci_{1,2}^{\alpha,\beta} & ci_{2,2}^{\alpha,\beta} & ci_{3,2}^{\alpha,\beta} & \dots & ci_{m,2}^{\alpha,\beta} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ ci_{1,n}^{\alpha,\beta} & ci_{2,n}^{\alpha,\beta} & ci_{3,n}^{\alpha,\beta} & \dots & ci_{m,n}^{\alpha,\beta} \end{bmatrix}^T .$$

3.2. Software Approach of the SIFT Matching Algorithm

The SIFT matching algorithm iterates through several steps to check if a match of a single descriptor of image $\tilde{\alpha}$ corresponds with another descriptor in image $\tilde{\beta}$. The efficient design of a matching algorithm depends largely on the platform in which implementation is to occur. In this section a software approach is detailed with a description of the resulting implementation of the SIFT matching algorithm based on our proposed angular distance measure between descriptors.

Algorithm 1 Software approach for SIFT descriptor matching.

Input: *kth* descriptor of image α with size 1×128
Database descriptors of image β with size $m \times 128$
Output: Matched result for *kth* descriptor with the database descriptors

for $j = 1$ to m **do**
  for $k = 1$ to 128 **do**
    $p[i][j] += A[i][k] \cdot B[j][k]$;
  end for
  $[sort\_vals, index] = sort(arccos(p[i][j]))$;
  if $sort\_vals(1) < (threshold * sort\_vals(2))$ **then**
    return *Match Found*
  else
    return *No Match Found*
  end if
end for

Algorithm 1 provides the computational software flow of the SIFT matching algorithm for the k^{th} descriptor, d_k^{α}, of image $\tilde{\alpha}$ with the database descriptors of image $\tilde{\beta}$, d^{β}, where image $\tilde{\beta}$ has n descriptors, represented as

$$d^{\beta} = \{d_1^{\beta} \ , \ d_2^{\beta} \ , \ \dots\dots \ , \ d_n^{\beta}\}.$$

The first step of the SIFT matching algorithm is to calculate the dot-product of the descriptor, d_k^{α}, with each descriptor in the database according to Equation 1. The result of the dot-product operation is a vector, $dp_k^{\alpha,\beta}$, of n elements, shown as

$$dp_k^{\alpha,\beta} = [dp_{k,1}^{\alpha,\beta} \ , \ dp_{k,2}^{\alpha,\beta} \ , \ \dots\dots \ , \ dp_{k,n}^{\alpha,\beta}].$$

The following step is to take the arc-cosine of each element in $dp_k^{\alpha,\beta}$, saving the result in memory or cache, presented mathematically as

$$ci_k^{\alpha,\beta} = [ci_{k,1}^{\alpha,\beta} \ , \ ci_{k,2}^{\alpha,\beta} \ , \ \ldots\ldots \ , \ ci_{k,n}^{\alpha,\beta}].$$

The resulting output array, $ci_k^{\alpha,\beta}$, is sorted in ascending order where the first and second minimums are calculated.

David Lowe defined a threshold criteria [1], typically 0.6, to determine matching success. Matching success is determined by the match between the k^{th} descriptor, d_k^{α}, of image $\tilde{\alpha}$ with the database descriptors, d^{β}, of image $\tilde{\beta}$, according to Equation 2.

$$\begin{cases} minimum < (0.6 \times second\_minimum) & \text{Match} \\ otherwise & \text{No Match} \end{cases} \tag{2}$$

The calculations listed are repeated for each descriptor in image $\tilde{\alpha}$ to determine the matching features in image $\tilde{\beta}$. From Algorithm 1, the SIFT matching algorithm requires an equally large number of calculations and memory resources; quickly showing large time dependency due to both calculation and memory access latency.

4. Proposed Optimization of Memory Bandwidth

In this section, we study the impact of the memory bandwidth on the overall performance of the matching process and explore an optimization scheme to fully utilize the computation core and the memory bandwidth.

4.1. Roofline Performance Model

Image descriptors are streamed to the SIFT descriptor matching algorithm subsystem via an attached memory to the computing core. The total memory bandwidth plays a vital role in achieving maximum performance for a given system. In order for the matching core to start processing, one descriptor for each image, $\tilde{\alpha}$ and $\tilde{\beta}$, should be ready at the input ports of the matching core. We assume that the k^{th} descriptor of image $\tilde{\alpha}$ is always ready at the input port of the computational core. Since each descriptor is composed of 128 elements[1], each data transfer between memory and the computational core is 256 bytes.

In order to study the effect of the memory bandwidth in the overall performance of the system, let's assume that only one computational core exists in the system, that is pipelined and works at 100 MHz. To execute one operation, a full descriptor (256 bytes) should be ready at the input port of the computational core. Hence, the memory bandwidth take part in the system throughput. For example, if the memory bandwidth reaches 32 bytes per clock-cycle (3.2 GB/s), the computational core will wait for 8 clock-cycles to completely receives a single descriptor to start the process. This will achieve 12.5 Mega operation/second (M op/s). When the memory bandwidth increases to 6.4 GB/s, similarly, the performance

[1]Each element would be nominally composed of a 16-bit fixed point for the angular distance method.

increases to 24 M op/s. As long as the memory bandwidth increases, the performance increases. However, the maximum attainable throughput stops at its maximum peak when the memory bandwidth reaches 256 bytes per clock-cycle (25.6 GB/s) at the input port of the computation core. As the memory bandwidth increases above this limit, more data is present at the input port of the computation core but only 256 bytes are processed at a time. Figure 1 shows the effect of the memory bandwidth on the system performance. The speed of the computational core increases with increasing the memory bandwidth till it reaches the boundary of the peak performance.

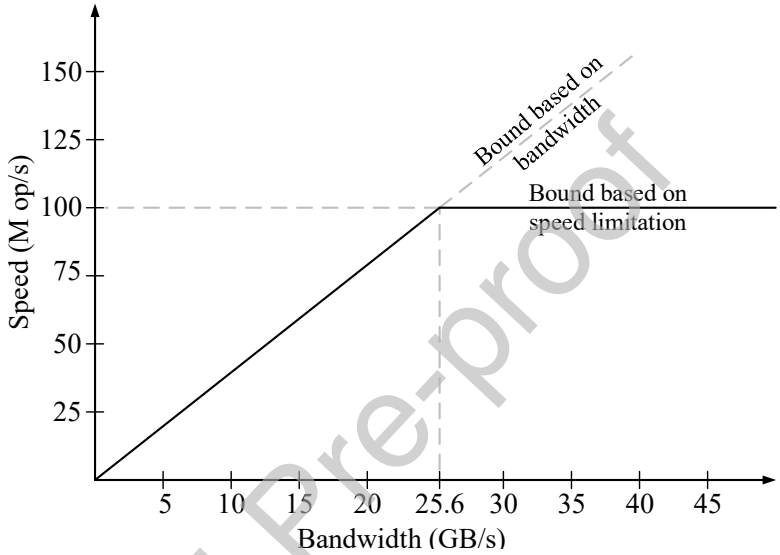


Figure 1: Performance and memory bandwidth effect.

In this paper, we implemented the matching core on Zedboard [16]. The platform includes two DDR3 memory components. The multi-protocol DDR controller is configured for 32-bit wide accesses to a 512 MB address space. For 32-bit data width access of the DDR memory, 64 bits (8 bytes) are accessed in one clock-cycle. This limits the performance of the matching core to 100/32 M op/s for 100 MHz running clock, where the core waits for 32 clock-cycles to receive a complete descriptor to start its operation. However, by optimizing the memory access, we can achieve the peak performance of the platform, 100 M op/s, as illustrated in section 4.2.

4.2. Memory Access Optimization

Due to the maximum memory bandwidth limitations presented by the hardware platform, our goal is to increase throughput by executing one dot-product operation every clock-cycle. To assure one dot-product operation can be computed every clock-cycle, a new descriptor must be valid every clock-cycle. In order to alleviate the memory bandwidth bottleneck, an internal memory (cache) is used for storing 32 descriptors. Since the time to calculate 32 dot-product operations is 32 clock-cycles (one operation per clock-cycle), within that time period, one complete descriptor can be fetched from external memory. The newly

fetched descriptor will execute a dot-product operation with each descriptor in the internal cache (32 descriptors stored in internal cache). The result of the fetching optimization operations allows calculating 32 dot-product operations in 32 clock-cycles. While executing the 32 dot-products of the current descriptor, a new descriptor is received and the process is repeated until the entirety of descriptors of image $\tilde{\beta}$ is completed.

Therefore, in order to alleviate the memory bandwidth restriction, the descriptors of image $\tilde{\alpha}$ are divided into blocks of 32 descriptors each. Each block is passed to an internal cache and the dot-product operation is executed with one block and the entirety of descriptors of image $\tilde{\beta}$. Architecturally, two first-in first-out (FIFO) buffers are used to store the descriptors from external memory as a linear cache, with a total fetch time of 1024 clock-cycles per block[2]. The result from the block latency is that the highest throughput can be achieved when image $\tilde{\beta}$ has more than 32 descriptors. To further alleviate computation time and memory requirements for the SIFT descriptors matching, the architecture must be fully compatible with the platform. In our approach, the matching architecture is implemented such that full utilization of the core is achieved.

5. Proposed Matching Algorithm Architecture on Hardware

FPGA is generally utilized for accelerating computational processes by increasing concurrent operations. It further increases the overall throughput of the system by pipelining and overlapping the instructions. The goal of this work is to accelerate the SIFT descriptors matching on FPGA and efficiently handling memory bandwidth limitations which is often seen in software implementation, as explained in Section 4.

In our proposed architecture, the descriptors of image $\tilde{\alpha}$ and image $\tilde{\beta}$ are streamed from external memory into an internal FIFO. Each descriptor is composed of 128 elements and its location, (x, y), in the image. Although each element should be represented as a double-precision floating-point to increase the accuracy, such floating-point adder circuits [17] is more complicated and consumes more resources. Therefore, for further optimization, each element of the descriptor is represented as a 16-bit fixed-point value and a total of 32-bits for its location, leading to an individual descriptor size of 2080 bits.

The proposed SIFT matching architecture [18] consists of four main sub-cores and two internal caches to alleviate memory bottlenecks; all of which are fully pipelined and implemented onto FPGA:

- Dot_Product.
- Cosine_Inverse.
- Minimum Search (MIN_FIND).
- Match_Check.

[2] 1024 clock-cycles comes from the previous 32 clock-cycles to fetch a single descriptor by the number of descriptors per block, $32 \times 32 = 1024$.

- Descriptor Cache (DES_MEM).
- Minimum(s) Cache (MIN_MEM).

For making use of high-level synthesis design [19], Xilinx System Generator ® is used for designing and implementing the *Dot_Prod* and the *Cosine_Inverse* blocks as IP cores.

Figure 2 describes a block diagram of our proposed SIFT descriptor matching accelerator core. In this architecture, the descriptors are streamed from the memory to the matching core. The internal caches (buffers) are used for keeping the descriptors to alleviate the memory bottleneck and fully utilize the matching core. This will allow a complete descriptor available at every clock-cycle, where the external memory bandwidth is optimized for increasing the throughput as explained in Section 4.2.

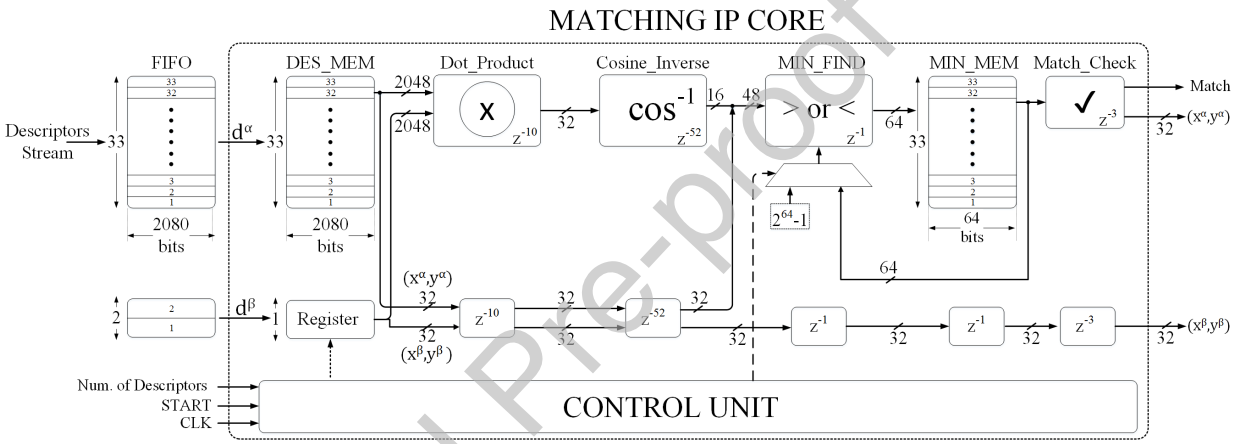


Figure 2: Matching core architecture for the proposed SIFT descriptors matching detailing the sub-modules and caches necessary for operation as controlled by the control unit. Z^{-n} is a shift register with n pipeline stages.

Since the descriptor size is 260 bytes (including the coordinates) and the memory bandwidth is 8 bytes per clock, it takes 33 clock-cycles to fetch a complete descriptor. In order to match the fetching time, 33 descriptors of image $\tilde{\alpha}$ were saved in a separate descriptors cache (DES_MEM), where each descriptor of image $\tilde{\beta}$ falls through in 33 clock-cycles and saved into Register.

The outputs of the DES_MEM and the Register are split into descriptors that are handed to the Dot_Product core and their coordinates that are passed through shift register with a corresponding number of the pipeline stages. The output of the Dot_Product is passed to a Cosine_Inverse core of 52 pipeline stages approximating the resulted output to 16-bits.

To calculate the minimum and second minimum on the fly, the previous minimum and second minimum are retrieved from the minimum(s) cache (MIN_MEM)[3] and passed to the minimum-search core (MIN_FIND) along with the current calculated cosine-inverse.

[3]Initially, the MIN_MEM is empty; the minimum and second minimum values are delivered and temporarily considered as the maximum values ($0 \times FFFF$).

For each new descriptor block of image $\tilde{\alpha}$, MIN_MEM should be flushed. Therefore, a multiplexer is used to pass a constant value of $(0 \times FFFF)$ when a new block of descriptor is delivered at DES_MEM. Otherwise, it passes the current value(s) in MIN_MEM. This is controlled by the Control Unit, seen in Figure 2.

The Control Unit present in the proposed SIFT matching architecture allows several operations to run concurrently by using a scheduling method; increasing overall throughput of the system. An example of scheduled concurrent operation is when the execution of a dot-product operation occurs on the final descriptor of image $\tilde{\beta}$, the DES_MEM is simultaneously filled with the subsequent descriptor of image $\tilde{\alpha}$ such that the following descriptor of image $\tilde{\beta}$ will already have a new reference descriptor block. The Control Unit is aware of the total number of descriptors and the processing time of each core, which make it able to handle the control signals to receive new descriptors, enable the internal cores, and control the internal caches.

5.1. Dot_Product Core

The SIFT matching algorithm begins by calculating the product of each element of the k^{th} descriptor of image $\tilde{\alpha}$ with the corresponding element of image $\tilde{\beta}$. Since the descriptor is comprised of 128 elements, 128 multiplications are required to calculate them in parallel. The output from each prior multiplication is sequentially added to obtain the resulting dot-product by the Dot_Product core which composed of 128 multiplication cores and seven levels[4] of tree adders. The multipliers necessary for the Dot_Product core were implemented into the FPGA's digital signal processing (DSP) slices with 3 pipeline stages. The seven levels of adders necessary for sequential addition are equivalent to 7 pipeline stages. Figure 3 shows the internal architecture of the Dot_Product core with a total of 10 pipeline stages.

5.2. Cosine Inverse Core

In order to implement the cosine-inverse in hardware, a coordinate rotation digital computer (CORDIC) core provided by Xilinx using System Generator for DSP [20] is used. The Cosine_Inverse core is shown in Figure 4. It is composed of two CORDIC cores: one to calculate the square root of an input and another to find the polar coordinates of a feature, labeled in Figure 4 as Square_Root and Polar_Sys, respectively. Internal to the Square_Root and Polar_Sys core are 37 and 11 pipeline stages, respectively. The calculation of $1 - x^2$ as an input to the Square_Root core is completed with 4 pipeline stages, with a total of 52 pipeline stages for the Cosine_Inverse core alone.

5.2.1. Polar_Sys Core

The Polar_Sys core within the Cosine_Inverse core has two inputs, u, and v of a Cartesian system, and two outputs, magnitude, ρ, and angle, θ. The relation between these Cartesian system, (u, v) and the polar system, $(\rho,\ \theta)$ is simply described for a right triangle with

[4]Seven levels of adders are required since $log_2 128 = 7$, meaning for each tree we can compute a segment of the 128 elements.

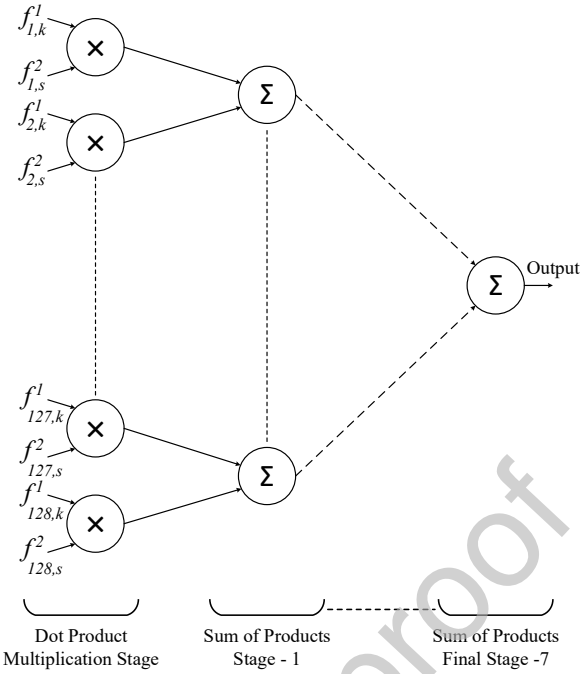


Figure 3: Dot product core.

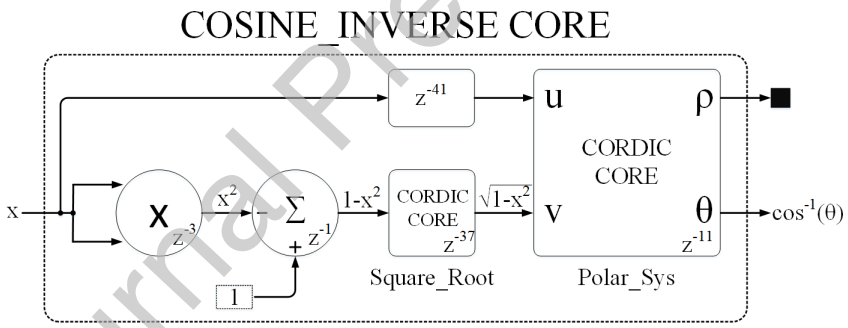


Figure 4: Internal depiction of the "Cosine_Inverse" core, including square root calculation and the polar coordinate translation module. Z^{-n} is a shift register with n pipeline stages.

hypotenuse ρ and sides u and v as $\rho = \sqrt{u^2 + v^2}$ and $\theta = tan^{-1}(v/u)$. To obtain the arc-cosine, only the calculation of θ is necessary, thus only computing $\theta = cos^{-1}x$, where x is an input into the Cosine_Inverse needs calculation. The two inputs of the Polar_Sys core, (u, v) are thus obtained as $u = x$ and $v = \sqrt{1 - x^2}$.

5.3. Minimum Search (MIN_FIND) Core

After calculating the arc-cosine for each descriptor in the database, two minimum values are determined, serving to highlight the database descriptors that are potential candidates for similarity within the image's descriptor under consideration. In software approach to descriptor matching, the output values are stored into a memory then a sorting algorithm is applied to find the minimum and second minimum values. In hardware approach to

descriptor matching, a typical sorting algorithm is resource-inefficient due to memory needs thus, in our design, the MIN_FIND core is designed to find both the minimum and second minimum values on the fly. This is done by retrieving the previous minimum and second minimum values from (MIN_MEM) and compared with the current calculated cosine-inverse. The pseudo-code representing the hardware operation of the MIN_FIND core is shown in Algorithm 2.

Algorithm 2 Comparison scheme for calculating minimum and second minimum values to highlight database descriptors as potential candidates for similarity with the image descriptor.

Input: Output of the cosine inverse (curr_val), recent minimum value from the memory (prev_min), and recent second minimum value (prev_sec_min).

Output: Updated minimum value (min) and second minimum value (sec_min).

if $curr\_val < prev\_min$ **then**
  $min = curr\_val$;
  $sec\_min = prev\_min$;
else if $curr\_val < prev\_sec\_min$ **then**
  $min = prev\_min$;
  $sec\_min = curr\_val$;
else
  $min = prev\_min$;
  $sec\_min = prev\_sec\_min$;
end if
return *min, sec_min*

5.4. Match_Check Core

The final step of the SIFT matching algorithm is to check if an actual match occurs by passing the minimum and second minimum values to the Match_Check core. The Match_Check core then applies Equation 2 to check matching between the descriptor of image $\tilde{\alpha}$ with another descriptor within image $\tilde{\beta}$. The hardware design of the Match_Check core consists of 3 pipeline stages *without* the need for any multiplier. The multiplication procedure of the second minimum with 0.6 is hidden in an addition process since $(0.6)_d = (0.10011)_b$ can be used as a constant value. Therefore, $minimum \times (100000)_b$ is compared with $second\ minimum \times (10011)_b$ to check matching between the two descriptors, previously mentioned in Algorithm 1. Figure 5 shows a three pipelined stages of adder circuits to implement Equation 2, where the multiplication of a number with the constant value $(10011)_b$ is done by adding the number to itself after it is shifted to the left one time and the result is added to the same number after it is shifted to the left four times. The shifting process was done by appending the right side of the number with extra zero-bits.

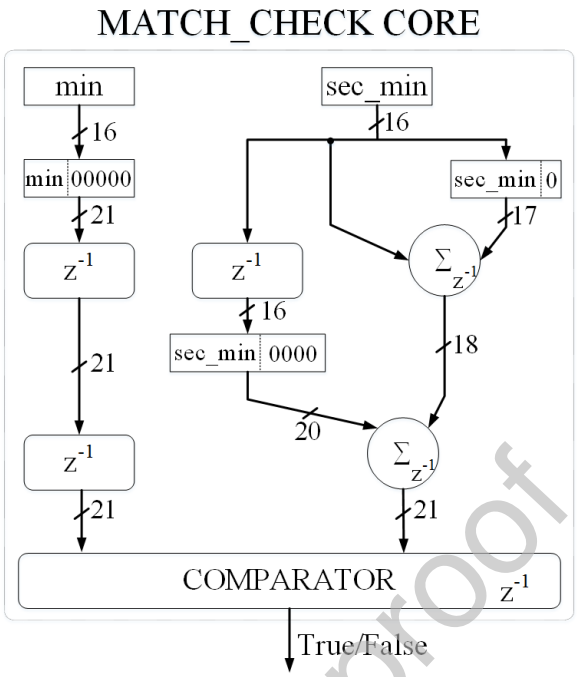


Figure 5: Matching Check core.

6. Experimental Results and Evaluation

In order to evaluate the proposed hardware architecture of the the SIFT descriptors matching, we used a Xilinx® Zynq-7000-based Zedboard. The Zedboard has a programmable logic and two ARM Cortex-A9 co-processors. The SIFT hardware matching algorithm core was implemented into the Zedboard's programmable logic while a single ARM Cortex-A9 processor was only used for simulation within the Xilinx® Software Development Kit.

6.1. Experimental Setup

In order for the SIFT matching core to start processing, descriptors d^{α} and d^{β} of both images $\tilde{\alpha}$ and $\tilde{\beta}$ should be ready at the input ports of the matching core. The descriptors for both images were initially stored onto a SD card used within the Zedboard, whose contents were then loaded into the external DRAM by the on-board processing system (PS). To check the matches between two images, the PS initializes and facilitates direct memory access (DMA) transfer of the provided descriptors from the DRAM to the descriptor buffers.

The PS then initiates the matching process with the matching core over advanced extensible interface (AXI) whereupon the matching core is provided with the number of descriptor blocks, number of descriptors per block, and a start signal. The Xilinx® EDA design Suite was used to synthesize and implement the design, including the matching core, DMA, and FIFO buffers. The Xilinx® Software Development Kit was used to read descriptors for both images from SD card into memory, and pass them to the fabric buffers/descriptor(s) caches. The fabric clock of the Zedboard has a range of 100 kHz to 250 MHz, however the AXI DMA has a maximum frequency of 150 MHz or 120 MHz for AXI4 and AXI4-Lite, respectively

[21]. Due to the frequency limitations of the Zedboard's systems, all experiments were run at a nominal frequency of 100 MHz.

6.2. Experimental Results

The SIFT matching algorithm was fully synthesized and implemented onto the Zedboard's fabric with a 135 MHz maximum frequency with a normal clock-speed of 100 MHz[5]. The implemented SIFT matching core has only one computational element of the "Dot_Product", "Cosine_Inverse", "MIN_FIND", and "Match_Check " cores. The matching core includes a "Control_Unit", additional internal memories for Descriptors and Minimum(s) Caches, and other registers for pipelining and synchronizing the data flow of the algorithm. Table 1 summarizes the overall resource utilization for individual components, with the "Others" category collecting registers and multiplexers used for synchronizing descriptors and control signals.

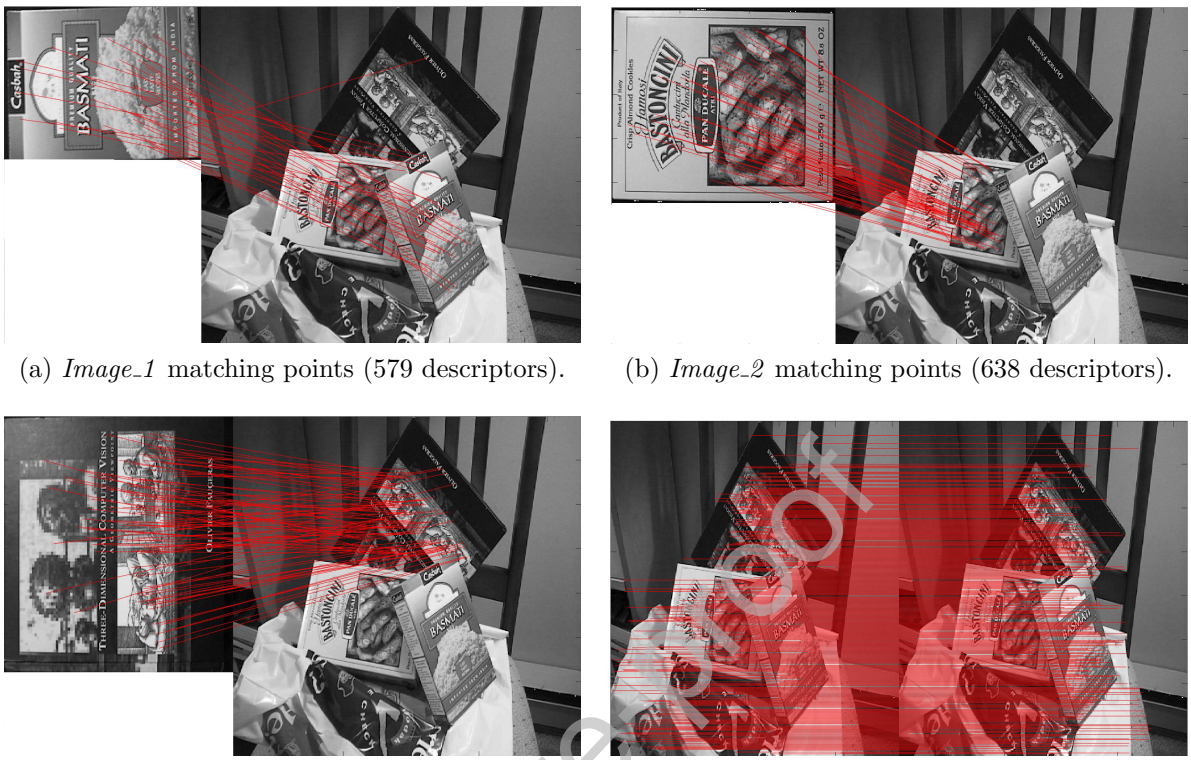
Table 1: Our proposed SIFT matching algorithm architecture utilization report for Zedboard FPGA implementation.

Core	**LUT**	**FF**	**DSP**	**BRAM**
Dot-Product & Cosine Inverse	3382	3557	132	0
Minimum Search	82	66	0	0
Matching Check	42	283	0	0
Control Unit	92	2132	0	0
Descriptor Cache	0	0	0	29
Minimum(s) Cache	0	0	0	1
Others	112	327	0	0
Total	3710	6365	132	30

In order to evaluate our SIFT matching core based on cosine angle distance approach, several experiments were conducted. The experiments were run on four different images, *image_1*, *image_2*, *image_3* and *image_4* where each image has 579, 538, 882, and 1021 descriptors, respectively. We chose *image_4* as the database image which the other three images were checked against for potential matches. To check the correctness of the matching points, *image_4* was used for testing the self-matching ability of the SIFT matching core as developed. Figure 6 shows the matching points between the selected images with the database.

For comparison of the descriptor matching time using a traditional software approach, the SIFT matching algorithm was executed on a 64-bit Intel® Core 2 Duo CPU running at 3.16 GHz using MATLAB® 2017a, following the original design described by Section 3.2. By using our proposed hardware SIFT matching core, it took 6.08, 6.75, 9.11, and 10.46 milliseconds for images with 579, 638, 882, and 1021 descriptors, respectively, whereas the software approach took 71.6, 77.4, 105.6 and 163.9 milliseconds, respectively, for the

[5]Used in this context due to limitation of the DMA core and AXI4-Lite provided by the Vivado toolset.



(a) *Image_1* matching points (579 descriptors). (b) *Image_2* matching points (638 descriptors).

(c) *Image_3* matching points (882 descriptors). (d) *Image_4* matching points (1021 descriptors).

Figure 6: Matching points for selected images with different number of descriptors

same set of images. The differences in time taken for both the software and our proposed hardware approach can be summarized that the software approach takes a quadratic increase in computational time with an increasing number of descriptors, compared with a linear increase for our proposed algorithm.

Our SIFT matching core accelerated the computational time of the selected images by $(11.5 \sim 15.7)\times$ for $(579 \sim 1021)$ descriptors. The double-floating point operations within the SIFT matching algorithm in software were approximated to 16-bit fixed-point operations in hardware with 98% of matched-descriptors detected with decreasing error for increasing fixed-point resolution. For further analysis, we compared our proposed hardware architecture of the SIFT matching algorithm with a similar approach accelerated on a graphics processing unit (GPU) in [15]. The Authors in [15] used both NVIDIA's Tesla K20 GPU and their local Intel Xeon® 2.7 Ghz Quad-Core CPU with a set of 2,800 descriptors for both the test image and database set. The authors noted that their matching algorithm takes 13 milliseconds on GPU and 80 milliseconds on CPU. Our hardware approach takes 10.46 milliseconds for 1021 descriptors using only one computation core, compared with the 2,496 cores present in the K20. It is noteworthy that by increasing the computational cores with fixed memory bandwidth, the throughput of our system is not affected due to memory bandwidth limitation. In this presented work, we achieved the maximum performance by

hitting the roofline performance model, i.e. achieving high throughput with the maximum allowed memory bandwidth and one computational core.

In addition, it is important to compare our proposed cosine angle distance approach for descriptor matching with other implementations in the literature. There are several matching techniques based on different approaches such as calculating the Chi-square distances [9], Sum of Absolute Differences (SAD) [8], and calculating Hamming distances [11].

The implementation in [9] used 64 cores to calculate distance metrics in parallel, requiring many multiplications and divisions. The comparison of resources to our proposed implementation shows a reduction of 91% of LUTs, 79% of BRAM. Apart from hardware resource utilization, our proposed SIFT matching core has the capability to check matchpoints within 9.28 milliseconds while [9] takes 10 milliseconds for 900 descriptors, due to pipelining and maximization of memory bandwidth within the computation core.

The implementation in [8] used 16 SAD (Sum of Absolute Differences) calculators to calculate the absolute difference between descriptors. The 16 SAD values are passed through 4 levels of comparators to obtain the minimum SAD value as a potential candidate among 16 descriptors. They used 2 separate RAMs to store the intermediate descriptors.The comparison of resources compared to our proposed implementation results in a reduction of about 92% of LUTs, 98% of BRAM and 75% DSP area.

The implementation in [11] used a variable number of Hamming distance calculators in the matching core. For implementation using 2 cores of Hamming distance a considerable saving in the FPGA resources were obtained. Compared to 2 cores to calculate Hamming distance, our implementation of the matching core using the cosine angle distance approach saved 37% of LUTs and 89% of BRAM. As the number of Hamming distance calculation cores increases, the number of resources utilization increases. Table 2 shows the utilized resources of our proposed SIFT matching core using cosine angle distance versus other matching methods such as Chi-square distance [9], SAD calculators [8] and Hamming distance [11].

Table 2: Utilized resources of our architecture core vs other implementation in the literature.

Parameter	**[9]**	**[8]**	**[11]**	**Proposed**
FPGA used	Virtex6	Cyclone IV	Zynq-7000	Zynq-7000
Image Size	512 × 384	640 × 480		512 × 384
Type of descriptors	SIFT	SIFT	FREAK	SIFT
# of descriptors	900	–	–	882
LUTs (% saved)	42662 (91%)	51068 (92%)	5967 (37%)	3710
DSP (% saved)	104 (-27%)	528 (75%)	–	132
BRAM (% saved)	142 (79%)	213 (85%) [6]	294 (75%)	30
Clock Frequency (MHz)	172	100	100	100

[6]The authors reported 1697 kbits of memory, which is equivalent to 213 BRAM in the best case of memory utilization.

Compared to these recent works [9, 8, 11] of implementation of matching cores on hardware, our proposed architecture consumes significantly fewer resources with acceptable matching accuracy (98%).

7. Conclusions

In this paper, a fully pipelined accelerator for a keypoint descriptor matching scheme for the SIFT object recognition algorithm was designed and implemented on FPGA where the matching core was constructed of four main computational sub-modules and two local caches. Utilizing a close construction and 16-bit fixed-point calculations helped alleviate memory bandwidth restrictions in order to achieve maximum throughput. An experimental system was designed on a Xilinx® Zedboard where the matching core was implemented on the programmable fabric and the Zynq processing system initialized the matching process. Our proposed SIFT matching architecture consumes fewer resources and accelerates the matching process where 9.11, 6.75, and 6.08 milliseconds elapsed for calculating the matching points of 882, 638, and 579 descriptors, respectively, with an image of 1021 descriptors. Our proposed SIFT matching hardware implementation additionally utilized 91% fewer LUTs and 79% fewer BRAM when comparing with the state of the art hardware matching core. Future work includes the extension of the hardware architecture into a fully pipelined vision system with increased number of computation cores.

Declaration of Competing Interest

All authors have participated in (a) conception and design, or analysis and interpretation of the data; (b) drafting the article or revising it critically for important intellectual content; and (c) approval of the final version.

This manuscript has not been submitted to, nor is under review at, another journal or other publishing venue.

The authors have no affiliation with any organization with a direct or indirect financial interest in the subject matter discussed in the manuscript

References

[1] D. G. Lowe, Object Recognition from Local Scale-Invariant Features, in: Proceedings of the Seventh IEEE International Conference on Computer Vision, Vol. 2, 1999, pp. 1150–1157.

[2] H. B.-S. Lee D-H, Lee D-W, Possibility Study of Scale Invariant Feature Transform (SIFT) Algorithm Application to Spine Magnetic Resonance Imaging, PLoS ONE 11 (4).

[3] Y. Jiang, Y. Xu, Y. Liu, Performance evaluation of feature detection and matching in stereo visual odometry, Neurocomputing 120 (2013) 380 – 390, image Feature Detection and Description.

[4] J. Kriaj, V. truc, N. Paveic, Adaptation of SIFT Features for Face Recognition under Varying Illumination, in: The 33rd International Convention MIPRO, 2010, pp. 691–694.

[5] A. Cesetti, E. Frontoni, A. Mancini, A. Ascani, P. Zingaretti, S. Longhi, A Visual Global Positioning System for Unmanned Aerial Vehicles Used in Photogrammetric Applications, Journal of Intelligent & Robotic Systems 61 (1) (2011) 157–168.

[6] B. Kolman, D. R. Hill, Elementary Linear Algebra, Pearson Education, 2004.

[7] G. Qian, S. Sural, Y. Gu, S. Pramanik, Similarity Between Euclidean and Cosine Angle Distance for Nearest Neighbor Queries, in: Proceedings of the 2004 ACM Symposium on Applied Computing, SAC '04, ACM, New York, NY, USA, 2004, pp. 1232–1237.

[8] J. Vourvoulakis, J. Kalomiros, J. Lygouras, Fpga-based architecture of a real-time sift matcher and ransac algorithm for robotic vision applications, Multimedia Tools and Applications 77 (8) (2018) 9393–9415.

[9] G. Lentaris, I. Stamoulias, D. Soudris, M. Lourakis, HW/SW Codesign and FPGA Acceleration of Visual Odometry Algorithms for Rover Navigation on Mars, IEEE Transactions on Circuits and Systems for Video Technology 26 (8) (2016) 1563–1577.

[10] J. Wang, S. Zhong, L. Yan, Z. Cao, An Embedded System-on-Chip Architecture for Real-time Visual Detection and Matching, IEEE Transactions on Circuits and Systems for Video Technology 24 (3) (2014) 525–538.

[11] R. Kapela, K. Gugala, P. Sniatala, A. Swietlicka, K. Kolanowski, Embedded platform for local image descriptor based object detection, Applied Mathematics and Computation 267 (2015) 419 – 426, the Fourth European Seminar on Computing (ESCO 2014).

[12] M. Calonder, V. Lepetit, M. Ozuysal, T. Trzcinski, C. Strecha, P. Fua, BRIEF: Computing a Local Binary Descriptor Very Fast, IEEE Transactions on Pattern Analysis and Machine Intelligence 34 (7) (2012) 1281–1298.

[13] A. Alahi, R. Ortiz, P. Vandergheynst, FREAK: Fast Retina Keypoint, in: 2012 IEEE Conference on Computer Vision and Pattern Recognition, 2012, pp. 510–517.

[14] G. Condello, P. Pasteris, D. Pau, M. Sami, An OpenCL-based feature matcher, Signal Processing: Image Communication 28 (4) (2013) 345 – 350, special Issue: VS&AR.

[15] H. Fassold, H. Stiegler, J. Rosner, M. Thaler, W. Bailer, A GPU-accelerated two stage visual matching pipeline for image and video retrieval, in: Content-Based Multimedia Indexing (CBMI), 2015 13th International Workshop on, IEEE, 2015, pp. 1–5.

[16] Xilinx Inc., ZC702 Evaluation Board for the Zynq-7000 XC7Z020 User Guide (September, 2015).

[17] L. Daoud, D. Zydek, H. Selvaraj, A Survey on Design and Implementation of Floating Point Adder in FPGA, in: Progress in Systems Engineering, Springer, 2015, pp. 885–892.

[18] L. Daoud, M. K. Latif, N. Rafla, SIFT Keypoint Descriptor Matching Algorithm: A Fully Pipelined Accelerator on FPGA, in: Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, ACM, 2018, pp. 294–294.

[19] L. Daoud, D. Zydek, and H. Selvaraj, A Survey of High Level Synthesis Languages, Tools, and Compilers for Reconfigurable High Performance Computing, in: Advances in Systems Science, Springer, 2014, pp. 483–492, , DOI: 10.1007/978-3-319-01857-7_47.

[20] Xilinx Inc., Vivado Design Suite Reference Guide: Model-Based DSP Design Using System Generator (May, 2019).

[21] Xilinx Inc., AXI DMA v7.1: LogiCORE IP Product Guide (October, 2017).

Luka Daoud received the B.S. degree in Electrical Engineering from Fayoum University, Egypt in 2007, and M.S. degree in Electronics and Communications Engineering from Egypt-Japan University of Science and Technology (E-JUST), Alexandria, Egypt in 2012. Luka is currently a Ph.D. candidate in Electrical and Computer Engineering at Boise State University, Boise, Idaho, USA. His main research focuses on hardware security, Network-on-Chip (NoC), high performance computing, and High Level Synthesis (HLS) design.

Muhammad Kamran Latif received the B.Sc. degree in electrical engineering from the University of Engineering and Technology, Lahore, Pakistan, in 2013 and his M.Sc. in electrical engineering from Boise State University (BSU), Idaho, USA in 2017. He is currently pursuing the Ph.D. degree from Boise State University, Idaho, USA. His current research interests include applications of power electronics to power systems, applications of computer engineering in smart grids and cyber-physical security of power distribution systems.

HS. Jacinto received his B.S. degree in electrical and computer engineering from Boise State University, Boise, Idaho, USA, in 2017, and is currently a Ph.D. candidate in electrical and computer engineering from Boise State University, Boise, Idaho, USA. From 2015 to 2017 he worked with Idaho National Labs in conjunction with the Advanced Energy Lab conducting research on self-powered wireless sensor networks and their security. He has moved over to the Air Force Research Lab Quantum Information Science group in 2018 under a fellowship to work on quantum information processing systems, integrated quantum photonics, and quantum control. His main research focuses on quantum network hardware cybersecurity, quantum informatics, and adaptive hardware anti-tamper and encryption technologies for use in the field of hardware security to create a secure platform for an upcoming quantum era.

Nader I. Rafla, PhD., P.E., received his M.S.E.E. and Ph.D. in Electrical Engineering from Case Western Reserve University, Cleveland, Ohio in 1984 and 1991, respectively. His doctoral research concentrated on object recognition and localization from multi sensor data: range image, force-torque, and touch. From 1991 to 1996 he was an Associate Professor at the Department of Manufacturing Engineering at Central State University, Wilberforce, Ohio. In January 1997, he joined the newly developed electrical and computer engineering department at Boise State University where he is currently Chair and Associate Professor, heading the development of the B.S. and M.S. programs. Dr. Rafla's areas of expertise are: Systems on a Programmable Chip (SoPC): reconfigurable computing and neuromorphic architectures; evolvable hardware and genetic algorithms; cryptography and cybersecurity; Implementation and design of secure hardware architectures for the next-generation embedded systems and their applications that have immediate impact on real-world problems and related technical challenges such as data fusion, hardware security, and hardware acceleration.

508

Daoud:

Latif:

Jacinto:

Rafla: