FP-SLIC: A Fully-Pipelined FPGA Implementation of Superpixel Image Segmentation

Adnan Ghaderi, Carl Ahlberg, Magnus Östgren[1], Fredrik Ekstrand, and Mikael Ekström

School of Innovation, Design and Engineering, Mälardalen University, Västerås, Sweden
{adnan.ghaderi, carl.ahlberg, fredrik.ekstrand, mikael.ekstrom}@mdu.se, [1]mon15008@student.mdu.se

Abstract

A superpixel segment is a group of pixels that carry similar information. The Simple Linear Iterative Clustering (SLIC) is a well-known algorithm for generating superpixels that offers a good balance between accuracy and efficiency. Nevertheless, due to its high computational requirements, the algorithm does not meet the demands of real-time embedded applications in terms of speed and resources. This paper proposes a fully-pipelined FPGA architecture based on SLIC, dubbed FP-SLIC, that exhibits 1) a simplified and efficient algorithm of reduced computational complexity that facilitates algorithm development for FPGAs, 2) a fully pipelined FPGA design operating at 40MHz with a throughput of one pixel per cycle, and 3) a memory-efficient architecture that eliminates the requirement for external memory. FP-SLIC shows promising BSDS500 benchmark results, especially considering boundary recall for less than 1000 superpixels, where it performs better than related works, while, at the same time, accomplishing a throughput of 259 frames per second (fps).

Index Terms

FPGA, Superpixel, Segmentation, SLIC, Image Processing.

I. Introduction

The term superpixel refers to a group of neighboring pixels with similar information [1]. Superpixel segmentation divides an image into smaller segments (Fig. 1), which can be used in pre-processing to reduce the number of processing points. Reducing the number of processing points is desirable in many applications, especially with large datasets. The superpixel approach has been adopted in several applications, such as increasing the speed of stereo vision [2] or decreasing the execution time of classification [3]. Among the existing methods for generating superpixels, SLIC [4] is one of the most promising approaches, offering a good balance between accuracy and efficiency. However, achieving real-time performance (30 fps) as image resolution increases, with CPU frequencies hitting the ceiling, is challenging. Field Programmable Gate Arrays (FPGA) are commonly used in real-time image processing applications due to their parallel properties, and in this paper a Fully Pipelined SLIC approach (FP-SLIC) to optimize superpixel segmentation on an FPGA is presented. This paper includes the following contributions:

- A simplified and efficient SLIC algorithm to reduce computational complexity and facilitate the development of algorithm for FPGAs.
- A fully pipelined FPGA design operating at 40MHz with a one pixel per cycle throughput, making it one of the fastest approaches available.
- A memory-efficient architecture that eliminates the need for external memory.

The remainder of the paper is arranged as follows: Section II provides a brief overview of related work, followed by a more detailed description of superpixel segmentation in Section III. After that, the proposed algorithm for real-time superpixel segmentation on an FPGA is discussed in section IV. Then the experimental setup with results are presented in Section V and Section VI, respectively. Finally, the results are discussed in Section VII followed by conclusions and future work in Section VIII.

II. Related Work

There are various algorithms for generating superpixels. Stutz et al. [5] discuss and compare the wide variety of superpixel segmentation algorithms. The Simple Linear Iterative Clustering (SLIC), proposed by Achanta et al. [4] is based on K-means clustering of the five-dimensional CIELAB color and image XY-plane space [6]. The Simple Non-Iterative Clustering (SNIC) is an improved version of SLIC proposed by Achanta et al. [7]. Contrary to SLIC, SNIC is non-iterative and requires less memory. However, SNIC relies on connectivity right from the beginning, and pixels are visited randomly, so it needs a complex priority queue data structure. The Superpixels Extracted via Energy-Driven Sampling (SEEDS) proposed by Van den Bergh et al. [8] is another method for generating superpixels. It is based on a simple hill-climbing optimization in which an initial grid of perfect square superpixels continuously modifies its boundaries to maximize an energy function.

Meyer [9] suggests a seeded watershed method for growing clusters through priority queues. Neubert and Protzel improved this algorithm with the Compact Watershed algorithm [10]. The Entropy Rate Superpixel Segmentation (ERS) [11] improves

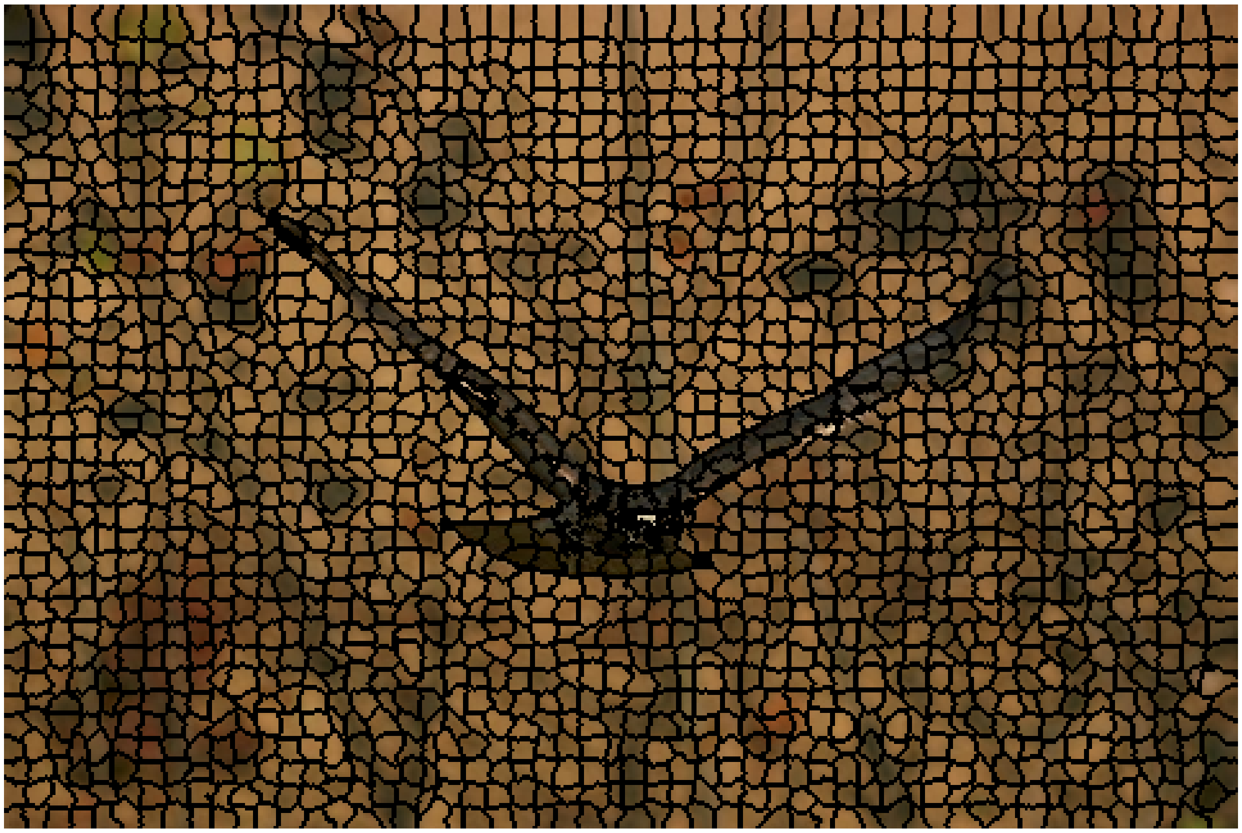


Fig. 1: FP-SLIC (1600 SPs, BSDS500:70011)

the accuracy of superpixel segmentation, in comparison to previously mentioned algorithms, but without being capable of real-time performance (30 fps).

Superpixel segmentation has been demonstrated as a competent pre-processing technique in image processing; for instance, Miyama [2] increased the speed of stereo vision by superpixel segmentation, and Gu et al. [3], decreased the execution time of classification.

Only few have addressed the hardware realization of superpixel segmentation. Hong et al. [12] proposed a Subsampled SLIC (S-SLIC) algorithm to reduce the memory bandwidth using pixel subsampling. S-SLIC was designed in 16nm FinFET technology and accomplishes efficient computation of superpixels at 30 fps on 1920×1080 images. In addition, S-SLIC provides 250× better energy efficiency than a SLIC implementation on a mobile Graphics Processing Unit (GPU). Miyama [13] proposed SS (Simplified SEEDS) on an FPGA that divided the image into a lattice shape and used an energy function to update the boundary. The design, which used external memory, process 0.43 pixels every clock cycle, achieving a throughput of 42.2 fps at 30MHz on 640×480 images.

III. SLIC Superpixel segmentation

SLIC is one of the most popular and efficient approach to generate superpixels. The algorithm converts RGB to $CIELAB$ space and then clusters pixels in the five-dimensional space [*labxy*] based on their color information and their distance from image plane centers. In SLIC, initial cluster centers are located on a uniform grid and distances are calculated as a sum of the $CIELAB$ color space [6] and the Euclidean distances in the XY plane [14]. To avoid placing clusters on edges, initial centers are perturbed to the lowest gradient position within the cluster neighborhood. The calculation of the image gradients is defined as follows:

$$\begin{aligned} G(x,y) &= ||\mathbf{I}(x+1,y)-\mathbf{I}(x-1,y)||^2 \\ &+ ||\mathbf{I}(x,y+1)-\mathbf{I}(x,y-1)||^2 \end{aligned} \tag{1}$$

where $\mathbf{I}(x,y)$ is the lab vector for a pixel at position (x,y) and $||.||$ is the L_2 norm. The distance D_s, to measure the distance between pixel i to center k, is calculated as follows:

$$D_s = d_{lab} + \frac{m}{S} d_{xy} \tag{2}$$

$$d_{lab} = \sqrt{(l_k - l_i)^2 + (a_k - a_i)^2 + (b_k - b_i)^2} \tag{3}$$

$$d_{xy} = \sqrt{(x_k - x_i)^2 + (y_k - y_i)^2} \tag{4}$$

where d_{lab} is the color distance and d_{xy} is the spatial distance. Furthermore, the compactness of a superpixel can be adjusted by changing m. In Eq. (2), S is the grid spacing, measured as: $S = \sqrt{\frac{N_p}{K}}$, where N_p is the number of pixels in the image and K is the desired number of superpixels. There is a $2S \times 2S$ search area around the center on the XY plane for each pixel; after measuring the D_s within the search area, the pixel will be assigned to a center with the smallest distance. After all pixels are assigned to the nearest center, the cluster center is updated by calculating the average of $labxy$ vector for all pixels belonging to the corresponding center.

IV. Proposed Algorithm

The efficiency of the SLIC makes it very attractive, but resource demanding. This section describes a fully-pipelined FPGA architecture of SLIC, called FP-SLIC that allows to implement SLIC on the FPGA in a resource efficient way.

A. Architecture

Original SLIC implementation on hardware is challenging, as it requires extensive resources. The proposed FP-SLIC uses a modified design approach that uses less resources to overcome this challenge. In SLIC, each pixel is read multiple times per iteration, while FP-SLIC adopts a pixel perspective architecture, similar to S-SLIC [12]. In this architecture, processing one pixel at a time decreases the required amount of memory. Furthermore, in FP-SLIC, three more modifications compared to SLIC have been made.

Firstly, the SLIC algorithm works in the CIELAB color space and converting the pixels from/to the RGB color space to/from CIELAB space at the beginning and end of the process comprises of multiple division, multiplication, and exponential operations. These mathematical operations require substantial memory and computational resources in hardware design. Therefore, FP-SLIC is implemented to use RGB color space to avoid resource-heavy mathematical calculations. Secondly, the SLIC algorithm uses the Euclidean distances (Eq. (2), Eq. (3), Eq. (4)) to measure the distance of pixels from the superpixel center. The square root operation used for distance calculations demands heavy resources, notably when used in multiple instances in one clock cycle. FP-SLIC instead utilizes a computation-efficient distance metric, the Manhattan distance, to avoid this problem. The Manhattan distance between two points $(X1, Y1)$ and $(X2, Y2)$ is equal to $|X1 - X2|$ + $|Y1 - Y2|$. Thus, Eq. (2), Eq. (3) and Eq. (4) were updated as follows:

$$D_s = d_{rgb} + \frac{m}{S} d_{xy} \tag{5}$$

$$d_{rgb} = |(r_k - r_i)| + |(g_k - g_i)| + |(b_k - b_i)| \tag{6}$$

$$d_{xy} = |(x_k - x_i)| + |(y_k - y_i)| \tag{7}$$

Finally, to save resources associated with gradient calculation (Eq. (1)), initial centers are not perturbed. Hence, FP-SLIC enters the K-means iteration stage with a uniform cluster grid.

B. Algorithm Validation

For FPGAs, as algorithms are realized as physical electronic circuits, iterative processes propose a challenge. One approach is to use one and the same iteration circuit for all iterations. This way, the amount of resources does not depend on the number of iterations. However, memory resources are required store intermediate results, which can prove difficult for larger images, and only one iteration can be processed at a time, resulting in a poor frame rate. Another approach is to define a fixed number of iterations, each with its own set of resources. As an iteration is performed on partial image information, i.e. a number of lines, the result can be passed to the next iteration, before the full image has been processed. Hence, a processing pipeline can be setup, where iterations are processed in parallel, on different lines of the image. By taking advantage of the parallel processing capacity of the FPGA high frame-rate and low latency can be achieved. However, this approach is only viable if the number of iterations can be kept low.

In order to validate the soundness of the proposed algorithm and to investigate its performance under increasing the number of iterations, an initial experiment was carried out, using a software implementation. The experiment was based on the Berkeley Segmentation Data Set and Benchmarks 500 (BSDS500) [15] with 205 test images. Fig. 2 shows the changes in Boundary Recall (BR) and Undersegmentation Error (UE) as the number of iterations increase. The first thing to notice is that most of the improvements is achieved during the first iterations, indicating that the number of iterations can be cut, allowing for a parallel implementation approach.

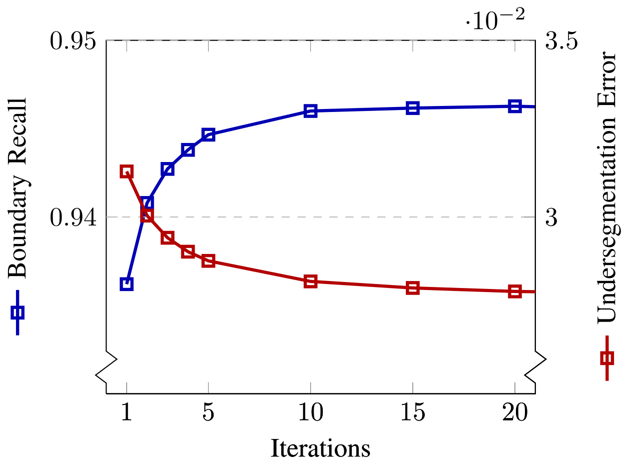


Fig. 2: BSDS500 results as a function of iterations (1600 SPs)

BR is a part of ground truth edges within a determined distance from a superpixel boundary to assess boundary adherence and is defined as:

$$BR = \frac{TP}{TP + FN} \tag{8}$$

where TP and FN are the numbers of true positive and false negative boundary pixels in the determined distance, respectively. High BR illustrates better boundary adherence.

The UE is a fraction of the summation of an overlapping boundary area to the entire image [16], and is expressed by:

$$UE = \frac{1}{N} \cdot \left[\sum_{G \in GT} \left(\sum_{SP:SP \cap S \neq 0} min(SP_{in}, SP_{out}) \right) \right] \tag{9}$$

where, N is the number of pixels, G is a superpixel segment in the GT (ground truth), SP_{in} and SP_{out} are the superpixels of SP inside and outside of S respectively. Lower UE implies better performance.

C. FPGA Implementation

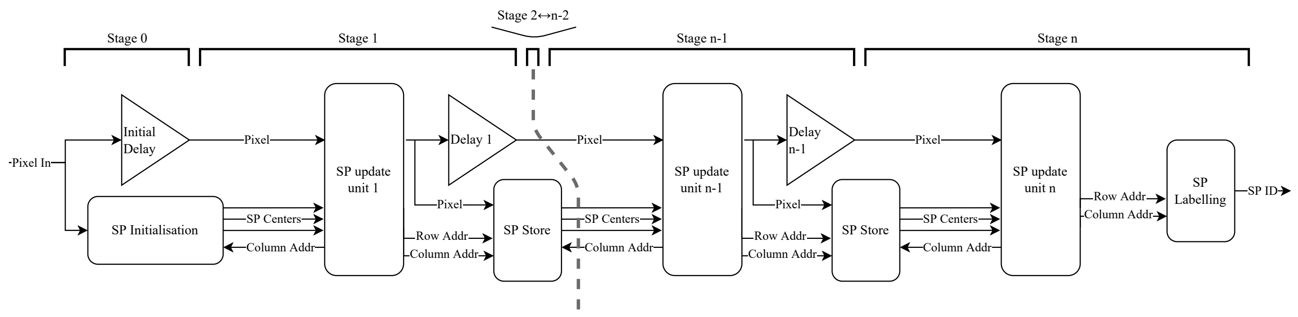


Fig. 3: Block diagram overview of SP-SLIC.

The FPGA architecture of FP-SLIC consists of multiple stages with dedicated hardware for each stage, as shown in Fig. 3. The stages can work simultaneously, and they start processing data once they have received a valid pixel. The FIFO-based delay between stages helps to ensure that a pixel arrives at a particular stage at the correct time. The delay unit, the superpixel store, and the superpixel update unit are the three main components of the design.

1) Delay Unit: The delay units are designed as FIFO ring buffers in a block RAM. Compared to the other units in the FP-SLIC design, the delay units consume more of FPGA's resources. The unit's inputs and outputs are an AXI stream Slave and AXI stream Master, respectively. The unit gets RGB pixels as input and sends out the pixels after the desired delay. The image size, grid spacing of the initial centers (S), and the number of superpixels are essential parameters to determine the length of a delay. For example, consider a $S \times S$ square of pixels for each center. In the initial stage, the center is determined by assigning a pixel in the center of the square. In contrast, the centers in the middle stages are the average color and position

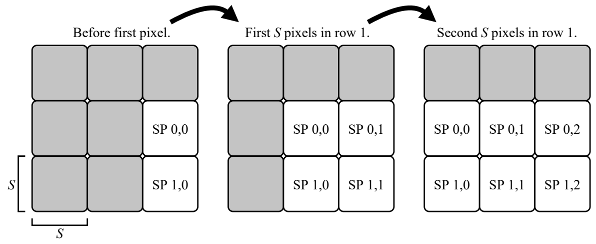


Fig. 4: The sliding window changes for the first few pixels.

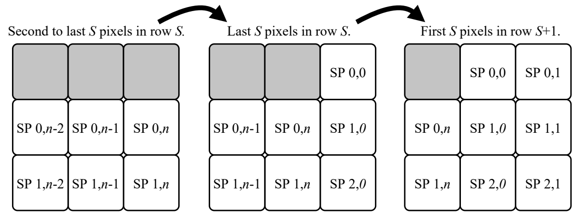


Fig. 5: The sliding window changes at the end of row S.

information of all pixels associated with the superpixel. An incoming pixel can get assigned to the closest superpixel in the 3×3 region around the square, so nine superpixel centers need to be ready before the new pixel arrives. Since centers are assigned by their location in the initial stage, they are not required to wait for all pixels of the corresponding superpixel to update. Thus, the minimum delay in the initial stage consumes less memory than other stages and is given by:

$$\text{Initial stage delay} = W \cdot S + \frac{W}{2} \cdot S \tag{10}$$

Where, W is the image width. For other stages, since the center will be ready only after the last pixel of the corresponding superpixel arrives, the delay would be more than the initial stage. Therefore, the minimum delay for other stages is given by:

$$\text{Middle stages delay} = 3 \cdot W \cdot S \tag{11}$$

2) Superpixel Store Unit: The superpixel store units are responsible for keeping the values of the superpixel centers. Each superpixel store is composed of six banks of the center store, where one bank stores the values of the superpixel centers of a single row, i.e., $\frac{W}{S}$ values. In the current stage, banks are responsible for writing and updating centers, and when writing continues after the right amount of delay, the next stage begins reading centers from center stores. Each superpixel in a bank represents an average of summed pixels belonging to the superpixel. The information of pixels saved in the banks comprises red, green, blue, column, and the row of the pixel. As soon as a new superpixel arrives, the values in the banks are reset.

3) Superpixel Update Unit: This unit plays a prominent role in allocating incoming pixels to the closest superpixel center among the nine possible centers. The nine squares in a 3×3 big window hold nine superpixel centers. Three small windows of each column slide to the left when new centers are loaded. Whenever a new pixel arrives, the distance from the current pixel to all nine available superpixel centers is measured simultaneously. After comparing all the distances, the index information of the smallest distance is obtained. The index information determines the row and column of the possible superpixel center. On a sliding window, the columns of superpixels shift every S clock cycle. A SOF (Start of Frame) and EOL (End of Line) serve to synchronize a counter to the control shifting process. The SOF and EOL information is part of an AXI stream pixel information. Whenever the superpixel update unit receives the first pixel of the frame, simultaneously, the sliding window shifts to the left for the first time (Fig. 4, middle).

Shifting in the same row continues until the last pixel of row S is arrived (Fig. 5, left). When a new pixel of a new superpixel row arrives, the top left in each row of superpixels will be shifted one row upwards to the top right (Fig. 5, middle). After the arrival of another pixel, the column will be shifted to the left again. This procedure will continue until the last pixel arrives. In Fig. 4 and Fig. 5, the cells in the sliding window with the wrong centers would fill with maximum distance; thus, they will not be selected by mistake.

4) Initial and Label Stages: The initial and label stages are determined as stages 0 and n respectively, as shown in Fig. 3. The initial stage comprises a simple form of a superpixel store unit. Unlike the superpixel store unit in the middle stages, there are no summation, averaging, and division operations in the initial stage. The store in the initial stage holds the information of the middle pixel in the $S \times S$ square. The superpixel update unit of stage 1 can start its process whenever two rows of

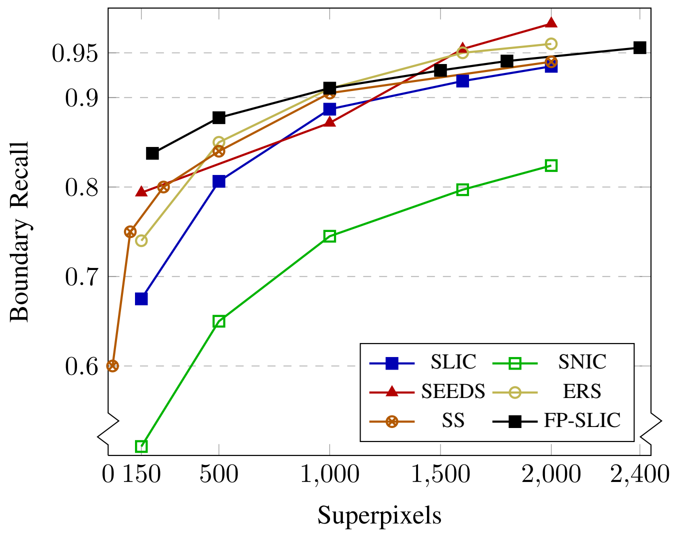


Fig. 6: BSDS500 evaluation comparison – Boundary Recall

superpixels are determined in an initial stage. The minimum delay in preparing two rows of superpixel is shown in Eq. (10). Finally, in the last stage, the row and column addresses generated by the update unit are used to label a pixel with the superpixel ID.

$$\text{superpixel ID} = SPs_{row} \times A_{row} + A_{col} \tag{12}$$

where SPs_{row} is the number of superpixels per row, A_{row} and A_{col} are the row and column addresses, respectively. As pixels are processed in the label stage, the superpixel ID, the SOF, and EOL signals of corresponding pixels are sent out as an AXI stream. The output can be prepared based on superpixels instead of pixels. In this case, the labeling unit can be replaced by one superpixel store unit to compute the final value of superpixel centers.

V. Experimental Setup

The proposed FP-SLIC was implemented on the Zynq UltraScale+ ZCU104 board using VHDL. In the FPGA design, shown in Fig. 3, three stages were used, the initial stage, the middle stage, and the label stage. In this design, the VDMA IP core from Xilinx is responsible for streaming the pixels in and out. Just as the initial experiment, Section IV-B, the experiment was based on BSDS500 205 test images. The framework proposed by Stutz et al. [5] was used for evaluation of the architecture. FP-SLIC with two iterations and the compactness of 80 is compared with the original SLIC, SEEDS, ERS, SNIC, and SS algorithms in terms of Boundary Recall (BR) and Undersegmentation Error (UE) metrics and their run-time for a varying number of superpixels. The SLIC, SEEDS and ERS parameter values have been left unchanged in the framework. For SNIC, the author code[1] with compactness equal to 50 was used. Since there is no public code, the results presented in the original paper concerning the SS algorithm [13] were used.

VI. Results

This section presents the quality of FP-SLIC compared to SLIC, SEEDS, ERS, SNIC, and SS algorithms using BR and UE metrics. Then, resource usage of the FPGA implementation of FP-SLIC, the accelerator performance, and the possibility of supporting a real-time application are described.

A. Quality Evaluation

Fig. 6 and Fig. 7 show the BR and UE metrics changes as the number of superpixels increased on the BSDS500 dataset images. The BR increases with the number of superpixels, and the UE is reduced with the number of superpixels.

[1]https://www.epfl.ch/labs/ivrl/research/snic-superpixels/

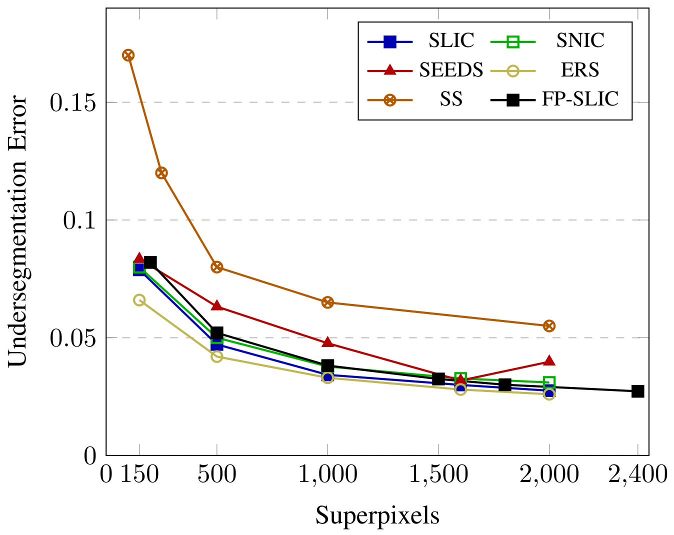


Fig. 7: BSDS500 evaluation comparison – Undersegmentation Error

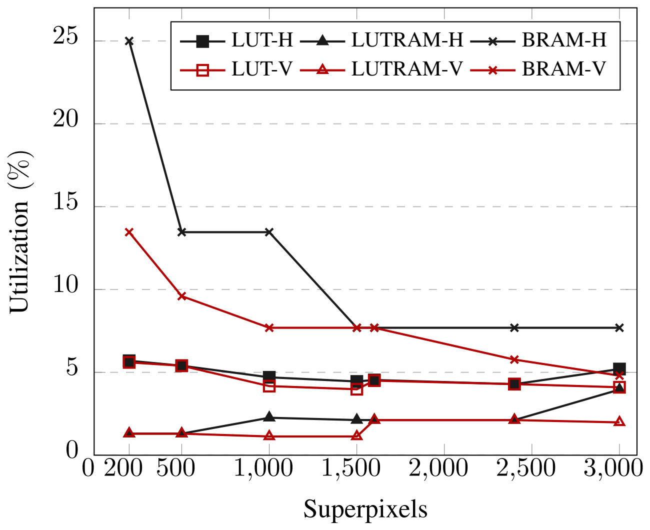


Fig. 8: FPGA resources of V(321×481) and H(481×321) images

B. FPGA Resource Utilization

The usage of FPGA resources depends on image size, the orientation of the image, and the number of superpixels. In FP-SLIC, the three primary resources are LUT, $LUTRAM$, and $BRAM$. Fig. 8 compares the required resources for images with both orientations (**H**orizontal (landscape) 481×321 and **V**ertical (portrait) 321×481) and various numbers of superpixels. Compared to images with a higher number of superpixels, images with a lower number of superpixels consume a considerable amount of resources. Essentially, an image with fewer superpixels will produce larger superpixels that will contain more pixels. As a result, the increase in the number of pixels within superpixels will result in allocating more memory in $Delay$ units to store the pixels. The $Delay$ unit implemented in BRAM is one of the main components in the design that occupies most of the resources on the FPGA. The specific delay for each stage is calculated in Eq. (10) and Eq. (11). Thus, having a bigger width will increase the delay, and the required memory for storing the pixels would also increase. Therefore, in Fig. 8, the resources for horizontal images are higher than for vertical images. Table I specifies the resource utilization for a 321×481 image and 2000 superpixels. The results are presented for 1, 2, and 3 iterations and the maximum frequency of each iteration to meet the timing constraint.

C. Accelerator Performance

A pixel arrives at every clock cycle in FP-SLIC, and the processor fetches a new instruction at every clock cycle. The two iterations of FP-SLIC achieves 40MHz for 481×321 and 35MHz for 640×480 image sizes. With a size of 321×481 or

TABLE I: FP-SLIC resource utilization, ZCU104, 321 × 481 pixels, 2000 SPs

Resource	Utilization and Utilization (%)						Available
LUT	2231	0.97	11546	5.01	19964	8.66	230400
$LUTRAM$	0	0	2160	2.12	4320	4.25	101760
FF	448	0.10	1647	0.36	2840	0.62	460800
$BRAM$	12	3.85	18	5.77	24	13.46	312
Iterations/MHz	1/100		2/40		3/40		

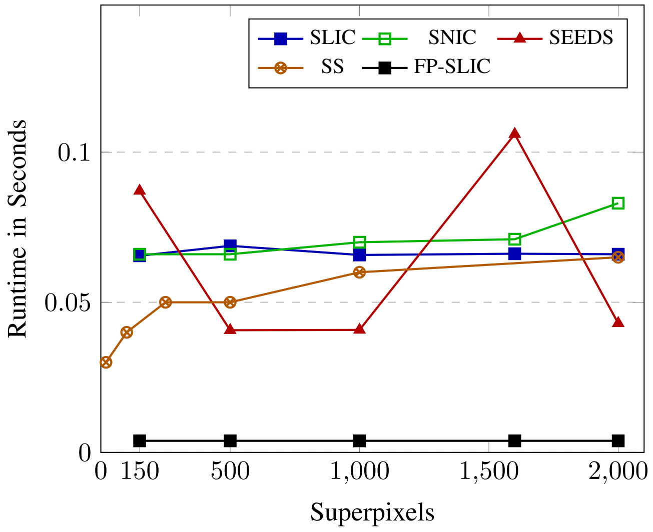


Fig. 9: Evaluation comparison – Run Time in seconds

TABLE II: Full processing times of different platforms for different sizes

Implementation	Run-time (ms) for image size 320 × 240	640 × 480
SLIC (CPU) [4]	32	126
gSLIC (GPU) [17]	9	21
S-SLIC (ASIC) [12]	-	19.8
FP-SLIC (FPGA)	1.9	8.7

481 × 321 images, the throughput of the FPGA is 259.06 fps, which is 8.63× more than the requirement to achieve real-time performance (30 fps). Furthermore, the FPGA can process one frame in 3.86 ms. Fig. 9 presents the run-time of FP-SLIC and other state-of-the-art methods. ERS had a run-time of $\approx$ 650 ms, which was much longer than other algorithms; therefore, the run-time of ERS is not included. The run-time of CPU-based algorithms varied with different machines. In Fig. 9, the SLIC, SNIC, and SEEDS results were run on a PC with an Intel Core i7-8850H (2.60 GHz) CPU, and the run-time of FPGA implementation of SS is derived from the original paper [13].

In comparison with CPU, ASIC (Application Specific Integrated Circuits), and GPU implementations of SLIC, run-time of our architecture on the FPGA improved dramatically. For example, the CPU implementation of SLIC needs 126 ms to segment a 640 × 480 image. In addition, gSLIC (a GPU-based implementation of SLIC) [17] with the same performance as the SLIC with an NVIDIA GTX460 graphic card is 6× faster than the CPU-based implementation of SLIC for the 640 × 480 image. Moreover, S-SLIC (ASIC-based) needs 19.7 ms for the corresponding image size. However, FP-SLIC implementation on the FPGA is, 14.48×, 2.4×, and 2.27× faster than CPU-based SLIC, gSLIC, and S-SLIC respectively, for the 640 × 480 image size. Table II shows the required time to segment an image with these four platforms for different image sizes.

VII. Discussion

The performance of FP-SLIC is evaluated with respect to BR, UE, run-time, and FPGA resources. First, the quality of FP-SLIC is assessed in comparison with SLIC, SEEDS, SNIC, and SS. Fig. 6 shows that FP-SLIC has better BR than SLIC and SNIC and slight improvement compared to other methods. In addition, Fig. 7 illustrates that the UE of FP-SLIC is better than the SEEDS and SS algorithms and has the same performance compared to the others. Both graphs indicate that FP-SLIC did not sacrifice performance, despite modifications. Secondly, the software implementation of our architecture demonstrates the quality of our algorithm as the number of iterations increases. For example, Fig. 2 illustrates that with the lower number of iterations, FP-SLIC achieves acceptable accuracy. Thirdly, FPGA resources of FP-SLIC presented for 1, 2, and 3 iterations in Table I. Furthermore, we examined the effects of the number of superpixels and the image orientation on the utilization of FPGA resources. According to Fig. 8, larger superpixels and horizontal images use more resources than smaller superpixel sizes and vertical images, respectively. Finally, Table I shows that FP-SLIC achieves 100MHz with one iteration. This indicates that FP-SLIC is one of the fastest approaches among existing algorithms, which could meet real-time performance for high-quality images. Also, the two iterations of FP-SLIC compared well with various methods considering using different platforms. In Fig. 9, the run-time of FP-SLIC is the lowest (3.86 ms), implying faster approach than the others. The results show that FP-SLIC has significant speed other than CPU-based approaches. Also, $2.4\times$ and $2.27\times$ faster than GPU-based SLIC, and S-SLIC respectively.

VIII. Conclusion and Future work

This paper presents FP-SLIC, a superpixel algorithm that shows promising evaluation results for the BSDS500 segmentation benchmark while benefiting from FPGA acceleration. Limiting the number of iterations with efficient memory usage enables a fully pipelined design that does not rely on off-chip memory, albeit introducing a minor negative impact on evaluation results. For example, on a Xilinx Zynq UltraScale+ ZCU104, a two iteration FP-SLIC achieves 40MHz, which for the BSDS500 images results in 259 fps. This is $8.63\times$ more than the requirement for real-time performance (30 fps). This architecture is expected to be used in computer vision applications to accelerate the processing of images.

References

[1] X. Ren and J. Malik, "Learning a classification model for segmentation," in *Computer Vision, IEEE International Conference on*, vol. 2. IEEE Computer Society, 2003, pp. 10–10.

[2] M. Miyama, "Fast stereo matching fully utilizing super-pixels," *Journal of Computer and Communications*, vol. 6, no. 8, pp. 15–27, 2018.

[3] F. Gu, H. Zhang, and C. Wang, "A classification method for polsar images using slic superpixel segmentation and deep convolution neural network," in *IGARSS 2018-2018 IEEE International Geoscience and Remote Sensing Symposium*. IEEE, 2018, pp. 6671–6674.

[4] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk, "Slic superpixels," Tech. Rep., 2010.

[5] D. Stutz, A. Hermans, and B. Leibe, "Superpixels: An evaluation of the state-of-the-art," *Computer Vision and Image Understanding*, vol. 166, pp. 1–27, 2018.

[6] M. R. Luo, *CIELAB*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 1–7.

[7] R. Achanta and S. Susstrunk, "Superpixels and polygons using simple non-iterative clustering," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 4651–4660.

[8] M. Van den Bergh, X. Boix, G. Roig, B. de Capitani, and L. Van Gool, "Seeds: Superpixels extracted via energy-driven sampling," in *European conference on computer vision*. Springer, 2012, pp. 13–26.

[9] F. Meyer, "Color image segmentation," in *1992 international conference on image processing and its applications*. IET, 1992, pp. 303–306.

[10] P. Neubert and P. Protzel, "Compact watershed and preemptive slic: On improving trade-offs of superpixel segmentation algorithms," in *2014 22nd international conference on pattern recognition*. IEEE, 2014, pp. 996–1001.

[11] M.-Y. Liu, O. Tuzel, S. Ramalingam, and R. Chellappa, "Entropy rate superpixel segmentation," in *CVPR 2011*. IEEE, 2011, pp. 2097–2104.

[12] I. Hong, I. Frosio, J. Clemons, B. Khailany, R. Venkatesan, and S. W. Keckler, "A real-time energy-efficient superpixel hardware accelerator for mobile computer vision applications," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2016, pp. 1–6.

[13] M. MIYAMA, "Fpga accelerator for super-pixel segmentation featuring clear detail and short boundary," *IIEEJ Transactions on Image Electronics and Visual Computing*, vol. 5, no. 2, pp. 83–91, 2017.

[14] P.-E. Danielsson, "Euclidean distance mapping," *Computer Graphics and image processing*, vol. 14, no. 3, pp. 227–248, 1980.

[15] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik, "Contour detection and hierarchical image segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 33, no. 5, pp. 898–916, 2010.

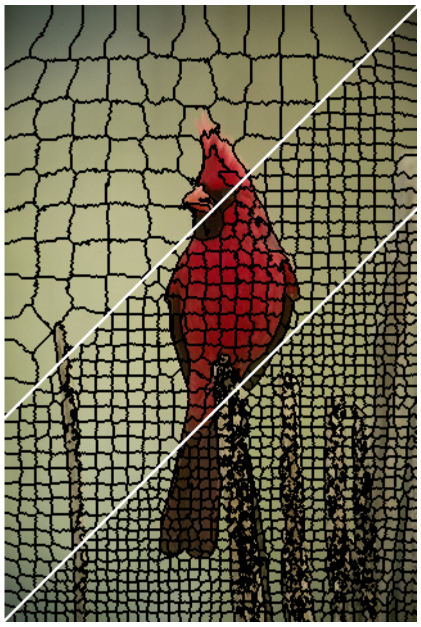
[16] P. Neubert and P. Protzel, "Superpixel benchmark and comparison," in *Proc. Forum Bildverarbeitung*, vol. 6, 2012, pp. 1–12.

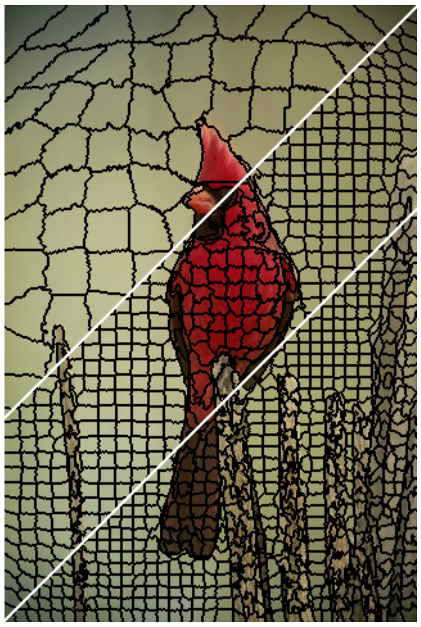
[17] C. Y. Ren and I. Reid, "gslic: a real-time implementation of slic superpixel segmentation," *University of Oxford, Department of Engineering, Technical Report*, pp. 1–6, 2011.

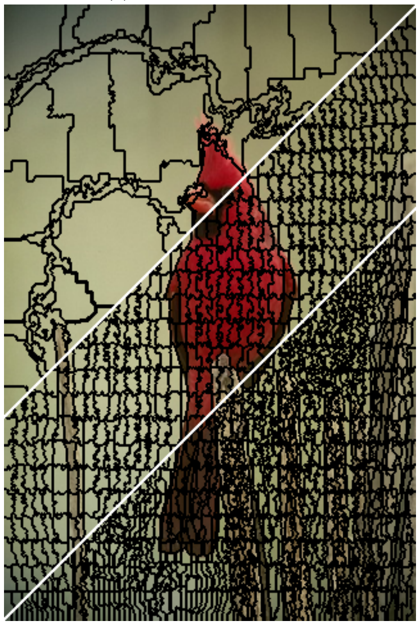
Appendix

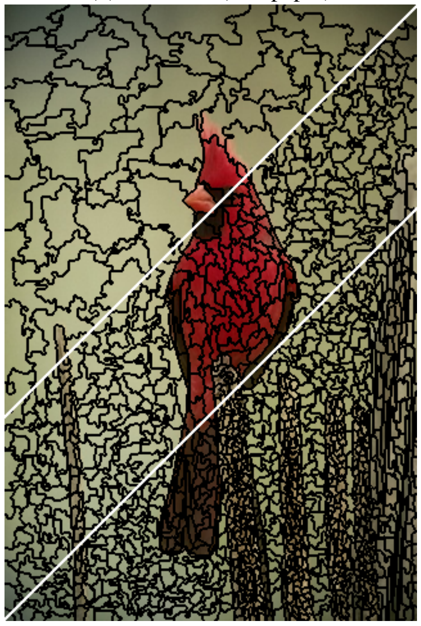
To better visualize superpixel segmentation, figures 10 and 11 demonstrate the output images of various algorithms using 150, 1000, and 2000 superpixels.

(a) BSDS500-196027


(b) FP-SLIC (this paper)


(c) SLIC


(d) SEEDS


(e) ERS

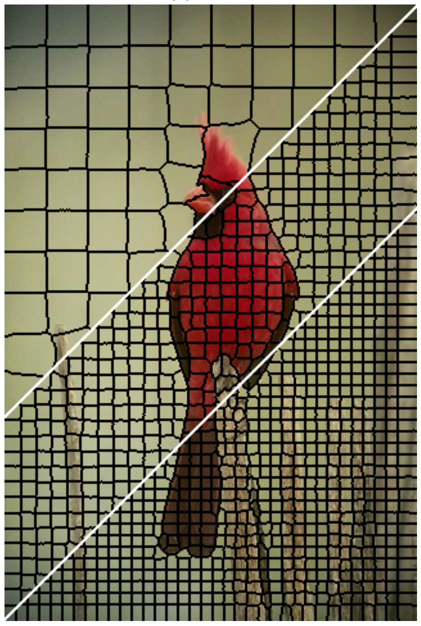
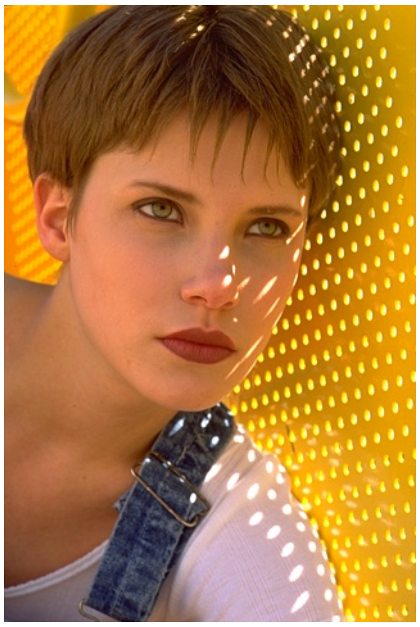
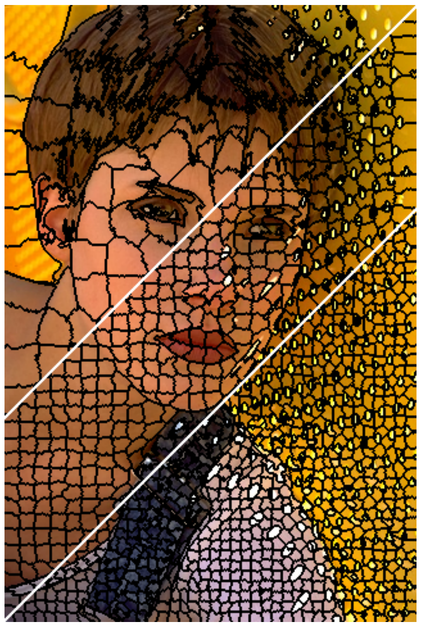
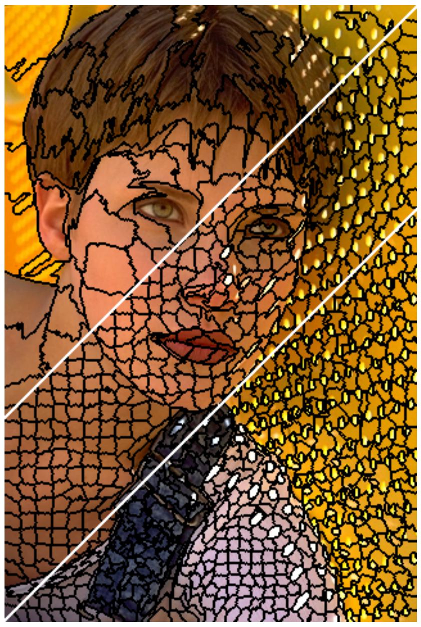
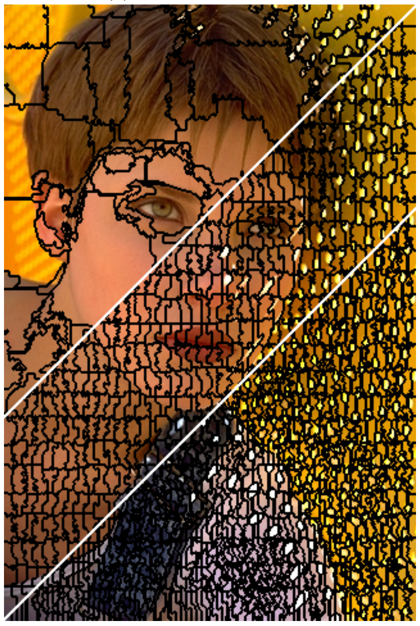

(f) SNIC

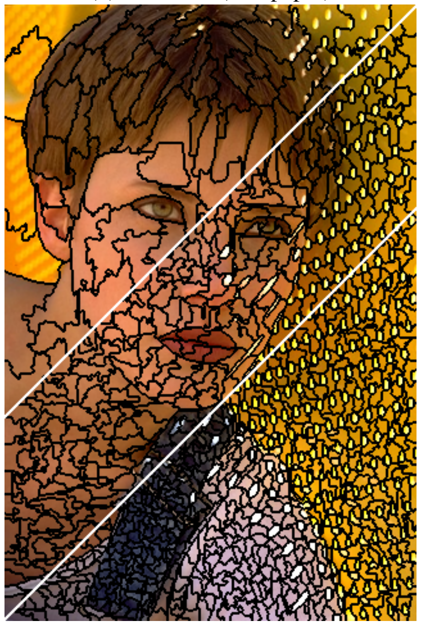
Fig. 10: BSDS500:196027, Two iterations, SPs: 150 - 1000 - 2000


(a) BSDS500-388006


(b) FP-SLIC (this paper)


(c) SLIC


(d) SEEDS


(e) ERS

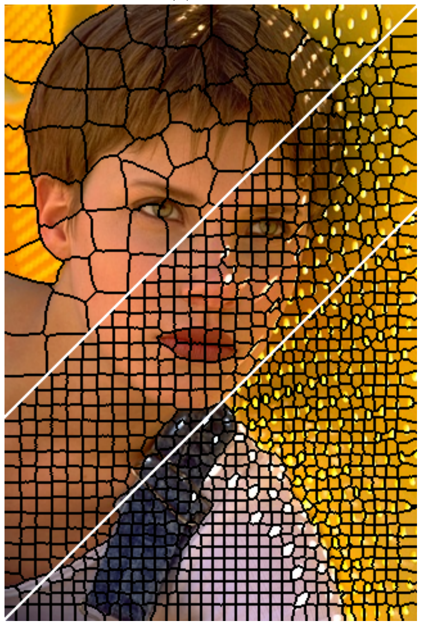

(f) SNIC

Fig. 11: BSDS500:388006, Two iterations, SPs: 150 - 1000 - 2000