

Research Article

FPGA Implementations of Algorithms for Preprocessing of High-Frame-Rate and High-Resolution Image Streams in Real-Time

Uroš Hudomalj*, Christopher Mandla and Markus Plattner

Max Planck Institute for Extraterrestrial Physics, Germany

hudomalj@mpe.mpg.de; mandla@mpe.mpg.de; plm@mpe.mpg.de

*Correspondence: hudomalj@mpe.mpg.de

Received: 29th January 2021; Accepted: 10 February 2021; Published: 1st April 2021

Abstract: This paper presents FPGA implementations of image filtering and image averaging – two widely applied image preprocessing algorithms. The implementations are targeted for real-time processing of high-frame-rate and high-resolution image streams. The developed implementations are evaluated in terms of resource usage, power consumption, and achievable frame rates. For the evaluation, Microsemi's Smartfusion2 Advanced Development Kit is used. It includes a SmartFusion2 M2S150 SoC FPGA. The performance of the developed implementation of image filtering algorithm is compared to a solution provided by MATLAB's Vision HDL Toolbox, which is evaluated on the same platform. The performance of the developed implementations are also compared with FPGA implementations found in existing publications, although those are evaluated on different FPGA platforms. Difficulties with performance comparison between implementations on different platforms are addressed and limitations of processing image streams with FPGA platforms discussed.

Keywords: *image processing; real-time image processing; FPGA; Camera Link; resource usage; power consumption; frame rate; processing data rate*

1. Introduction

Cameras are nowadays indispensable gadgets used in a plethora of applications, ranging from uses in automotive industry for autonomous driving [1,2], for security and surveillance [3,4], in medicine [5], for product quality assurance in manufacturing [6], as well as observations of both space [7] and earth¹ [8]. Most of these applications require images received by the camera to be processed in real-time for them to be relevant, useful and to prevent loss of image frames [9]. The real-time processing means that the processing of data has to be completed in the time available between the two successive input sample [10]. In case of image stream this means that an image has to be processed before the next frame arrives. If this processing criterion is not met, frames could be dropped, thus causing the loss of information. For most of the mentioned applications a missed frame could mean catastrophic consequences. A real-time processing system with such strict processing criteria is referred to as a hard real-time system [8,11].

In order to extract the needed information from a captured image, different processing algorithms have to be applied. These vary from primitive operations like noise reduction to more

¹ NASA's ECOSTRESS Detects Amazon Fires from Space. 2019. <https://www.jpl.nasa.gov/news/news.php?feature=7490>

advanced techniques like feature extraction, pattern recognition and object recognition. Task-specific algorithms are constructed for a certain application. However, algorithms for image preprocessing are widely used in various applications. The preprocessing algorithms enhance the features of interest in the image, simplifying higher-level processing [12]. Nevertheless, processing of images in real-time is getting more and more difficult because we are constantly striving to improve our applications by developing more complex algorithms [13] as well as capturing larger images [14] and at increasing frame rates [15], which altogether results in increased processing requirements. Therefore, the most important requirement for a real-time imaging system is the achievable processing data rate. Unfortunately, traditional methods for processing images from a camera, i.e. with a single or multiple processors, are insufficient to provide high enough processing data rates. Therefore, other options have to be considered especially for embedded applications where processing resources like power consumption, weight, space, cooling, etc. are limited, as for example in smartphones [16]. The solution for achieving image processing in real-time while also satisfying other design constraints, is to design systems with dedicated functionality of image processing. Field Programmable Gate Arrays (FPGAs) present a platform which satisfies the mentioned design requirements. Moreover, FPGAs are modifiable, thus can easily be adapted to a specific image processing application [10]. If an FPGA is not included in the system which receives the images, a simple solution is to insert the FPGA between the camera and the receiver. In this case the FPGA must acquire the image sent by the camera via an interface (I/F), process it and then send it to the intended receiver via the same or even a different I/F.

This paper presents FPGA implementations of two standard image preprocessing algorithms, namely image filtering and image averaging. The former is used to emphasize certain features of an image or remove others. The latter reduces the noise levels by averaging multiple images from a sequence.

The developed implementations allow real-time processing of high-frame-rate and high-resolution image streams. They are developed in Simulink² using HDL library, which provides automatic generation of HDL code with HDL coder³. This is a popular method of designing imaging applications due to its simplicity and straightforward verification of the implemented algorithms as their functionality can be compared with MATLAB's functions from the Image Processing Toolbox⁴. MATLAB already provides a solution for implementing the image filtering in an FPGA with its Vision HDL Toolbox⁵. Nevertheless, the developed implementation presents an alternative design using fewer hardware resources while also allowing shorter intervals of blanking signals of the image stream. Moreover, MATLAB does not provide solutions for algorithms which need information about multiple frames, like the image averaging algorithm.

The implementations are evaluated in terms of resource usage, power consumption, and achievable frame rates. The performance of the developed implementation of the image filtering algorithm is compared with the solution provided by MATLAB's Vision HDL Toolbox, which is implemented and evaluated on the same platform. For the performance evaluation, an existing imaging system is used. It consists of an industrial graded camera GO-2400M PMCL [17] by Jai and a personal computer (PC) with a PCIe frame grabber board [18] from Silicon Software as the image receiver. Since the system does not include an FPGA, a Smartfusion2 Advanced Development Kit from Microsemi [19] is inserted between the camera and the receiver. The development board includes a SmartFusion2 M2S150 SoC FPGA. The camera uses the Camera Link I/F. Therefore, a Camera Link receiver and transmitter are implemented in the FPGA as described in [20]. To verify the correctness of the implementations at the fastest achievable data rates of the camera, it is configured to use the full Camera Link I/F [21]. The complete setup for the verification is illustrated in Fig. 1.

² MathWorks. Simulink - Simulation and Model-Based Design. 2019. <https://www.mathworks.com/products/simulink.html>

³ MathWorks. HDL Coder - MATLAB & Simulink. 2019. <https://www.mathworks.com/products/hdl-coder.html>

⁴ MathWorks. Image Processing Toolbox. 2019. <https://www.mathworks.com/products/image.html>

⁵ MathWorks. Vision HDL Toolbox. 2019. <https://www.mathworks.com/products/vision-hdl.html>

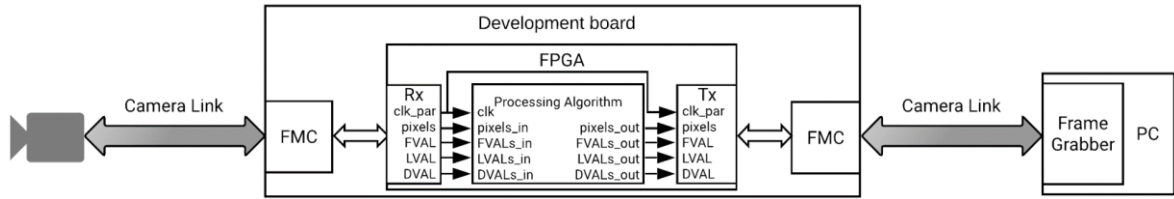


Figure 1. Setup for verification of algorithm implementations.

The paper extends the work presented in [22] by comparing the performance of the developed algorithm implementations in terms of processing data rates and power consumptions not only with the MATLAB’s solution for image filtering but also with the performance of other published FPGA implementations of the same algorithms. It also elaborates on the difficulties with conducting such a comparison.

2. Implementations of Preprocessing Algorithms

The following subsections describe the implementations for image filtering and image averaging. The implementations presented take into consideration the properties of the camera and the FPGA.

The camera in the full Camera Link I/F configuration transmits 8 pixels with a bit depth of 8 bits each clock cycle in parallel together with synchronization signals. The pixels are transmitted row by row. To achieve real-time processing of the image stream, the implementations process 8 pixels in parallel at the same frequency as they are transmitted. The clock frequency of the pixels is limited to 37.125 MHz due to the camera [21], the FPGA [23] and the custom Camera Link receiver/transmitter IP properties [20]. The maximum resolution of the camera is 1216 by 1936 pixels. At the frequency of 37.125 MHz, such image size is transmitted at a data rate of 2.2 Gbps [21].

2.1. Image Filtering

Image filtering is implemented as a two-dimensional convolution in the spatial domain of the image $f(x,y)$ with the filter mask $h(x,y)$ of size $2a+1$ by $2b+1$ according to (1).

$$r(x,y) = \frac{1}{M \cdot N} \sum_{m=-a}^a \sum_{n=-b}^b f(x+m,y+n) h(m,n) \tag{1}$$

To process 8 pixels in parallel, 8 convolutions have to be calculated in parallel, which is shown in the circuit in Fig. 2. For simplicity, implementation with a filter of size 3x3 is presented.

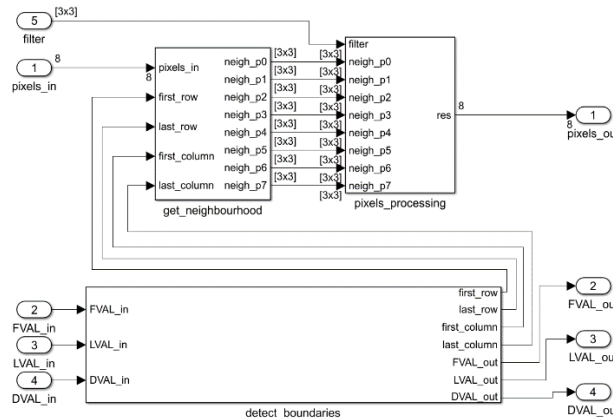


Figure 2. Top view of image filtering circuit.

In the circuit from Fig. 2, the component “get_neighbourhood” stores the arriving 8 pixels in memory as well as reads out 8 neighbourhoods of 3x3 pixels that are needed for the convolution for 8 pixels. The component also detects if any of the pixels of the neighbourhood are outside the image boundaries in which case it pads them with zeros. The boundaries are detected by the component “detect_boundaries” based on the frame synchronization signals. The component “detect_boundaries” delays the synchronization signals so that the processed data also has the right synchronization.

The convolution of a pixel is realized as shown in Fig. 3. First, a row of the pixel neighbourhood and the corresponding row of the filter coefficients are extracted. The extracted values are multiplied element-wise in parallel and summed together (implemented as an adder tree). This is repeated for the remaining rows. The results of the summed multiplications of rows are summed together (again implemented as another adder tree) to get the processed pixel value. These operations are done at a clock frequency three times higher than the rest of the circuit in order to process the incoming data at the same rate as the new data is arriving. Processing at a higher frequency allows lowering the number of resources needed, i.e. only 3 multipliers and 4 adders are needed for each of the 8 pixels processed in parallel. The same result could be achieved by direct multiplication and later addition of all neighbouring pixels at the same clock frequency as the rest of the circuit. However, such an implementation requires 9 multipliers and 8 adders for each pixel processed in parallel.

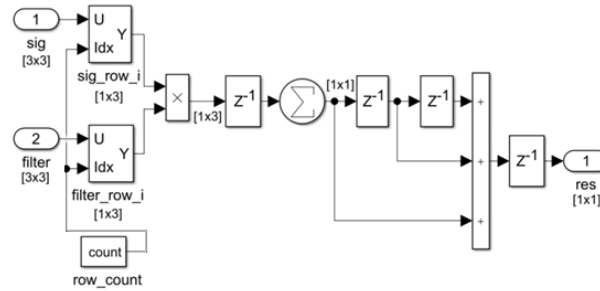


Figure 3. Circuit for calculation of 2D convolution.

The FPGA does not include hardware multipliers and adders for floating point operations. Therefore, the filter coefficients are stored with 16 bit fixed floating point precision. The intermediate calculations are done with fixed floating point precision with a minimum number of bits which do not influence the 8 bit output precision.

2.2. Image Averaging

Image averaging is implemented as a running average filter. It is applied to each pixel of the image. As mentioned, 8 pixels are processed in parallel each clock cycle, i.e. filter is applied to 8 pixels in parallel.

$$y(n) = y(n-1) + \frac{1}{L}[x(n) - x(n-L)] \quad (2)$$

If image averaging is implemented according to (2), four memory accesses are needed each clock cycle for each pixel, i.e. to store the current averaged output pixel $y(n)$, to read the previous output pixel $y(n-1)$, to store the incoming pixel $x(n)$ and to read its L -th previous value $x(n-L)$. The internal memory of the FPGA in the form of SRAM can store at most 4.5 Mbits [19], meaning it is too small to store even one image at maximum resolution, which takes up 18.8 Mbits. Therefore, external memory in form of DDR needs to be used. Memory access time limits the maximum achievable processing data rate. To maximize the throughput of the DDR memory, a circuit for accessing it with multiple overlapping bursts in a sequence is designed according to [20]. A throughput of 5.32 Gbps is achieved, which is the maximum possible throughput of the DDR memory of the development board used [24]. At a pixel data rate of 2.23 Gbps, a throughput of 8.92 Gbps of the DDR memory is needed, which is more than available. A solution is to simplify the filter by assuming $x(n-L) \approx y(n-1)$, resulting in (3). The assumption is valid if the pixels are not changing much through time.

$$y(n) = (1 - \frac{1}{L})y(n-1) + \frac{1}{L}x(n) \quad (3)$$

For efficient hardware implementation, the simplified filter form is implemented as shown in Fig. 4. It is assumed that the number of images to average L is a power of 2. With this assumption, the division is replaced with a simple bit shift.

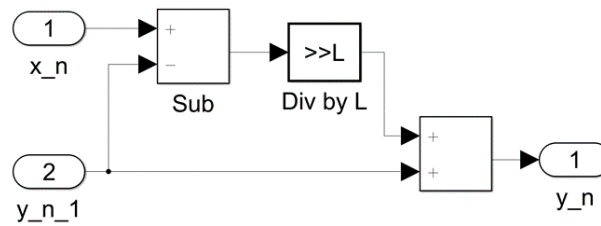


Figure 4. Circuit for implementing simplified version of running average filter.

3. Evaluation of Implemented Algorithms

In the following subsections are presented evaluations of resource usage, power consumption, and achievable frame rates for the developed implementations of image filtering and image averaging as well as the MATLAB's image filtering implementation from the Vision HDL Toolbox using the presented imaging system (Fig. 1). Before their evaluation, the correctness of the functionality of the implementations is first verified by simulating the generated circuits and comparing the output images with images processed with MATLAB's functions from the Image Processing Toolbox. The standard 8 bit, black-and-white Lenna image from [25] is used for the verification process. The image is resized and cropped to a resolution of 1936 by 1216 pixels to correspond to the maximum image resolution of the camera used.

The resource usage is evaluated based on the report of the synthesis tool from Microsemi's development environment Libero [26]. Some of the allocated resources do not differ for different implementations and algorithms, like the number of input/output connections, the number of clock conditioning circuits, oscillators, etc. Therefore, only the allocation of logic and memory resources are considered in detail. The former consist of look-up tables (LUTs) and multiply-accumulate blocks (MACCs); the latter of D flip-flops (DFFs) and RAM blocks.

The power consumption is estimated based on the resource usage using Microsemi's power estimator tool [27]. To get the full power consumption, the power consumption of DDR memory needs to be added when it is used. The power consumption of the DDR memory is estimated based on Micron's power estimator tool⁶ [28] for the DDR memory used [29].

The achievable frame rates are evaluated for the maximum frequency at which the implementations can operate and image size of 1216 by 1936 pixels, which is the largest image size that the used camera can capture. For the evaluation, the duration of the blanking signals between individual lines of an image and between frames of the image stream are kept to their minimum at which the implementations still work correctly.

3.1. Image Filtering

Results of verification of image filtering is presented in Fig. 5 for a Gaussian low-pass filter with standard deviation of 1 and size of 7x7. The figure includes the result of the simulated circuit (top left) and the referenced filtered image (top right). The reference image is acquired with MATLAB's functions 'fspecial' and 'imfilter'. The figure also includes the histogram of the absolute difference between the two results (bottom).

The histogram in Fig. 5 shows that not all the pixels have the same value. The source of the difference lies in the rounding of the coefficients. For the hardware implementation they are rounded to a 16 bit fixed floating-point representation. If they were implemented with double-precision floating-point format, there would be no difference. However, such an implementation would require a lot more hardware resources, e.g. floating-point multipliers and larger memories. Moreover, the histogram of the absolute difference between the simulated and the reference image shows that the maximum difference between using the double-precision and 16 bit fixed floating-point is 3 and altogether below 10% of all pixels differ. This is sufficiently small error for all practical applications.

⁶ Micron Technology Inc. System Power Calculators. 2019. <https://www.micron.com/support/tools-and-utilities/power-calc>

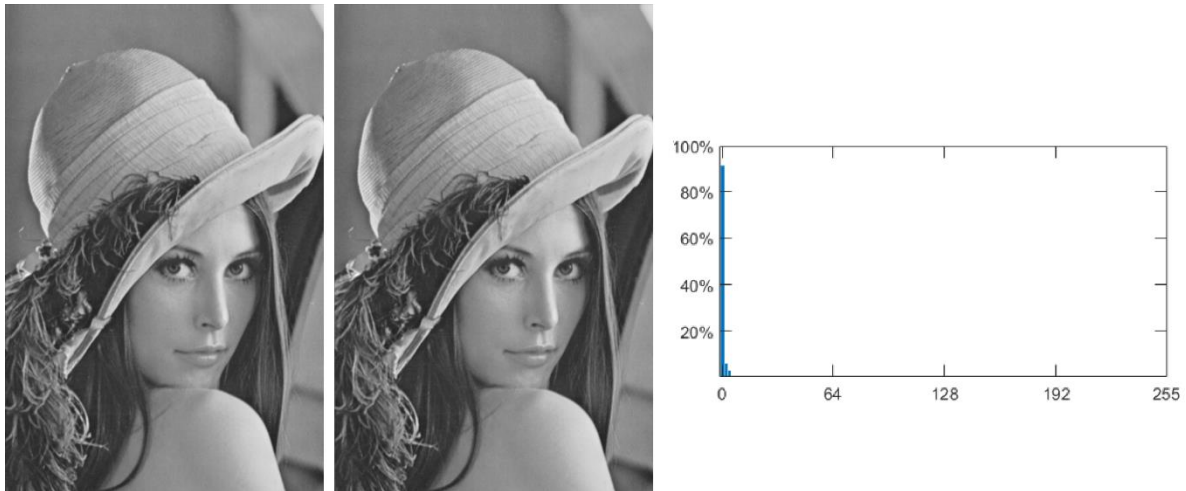


Figure 5. Verification of developed image filtering. Left image – simulated result of implemented image filtering; middle image – reference output; right – histogram of their absolute difference.

Evaluation of image filtering is conducted for filters of sizes from 3x3 up to 15x15. Fig. 6 and Fig. 7. show the relative resource usage for the developed implementation (denoted with “_d”) and MATLAB’s version (denoted with “_m”) for logic and memory resources, respectively.

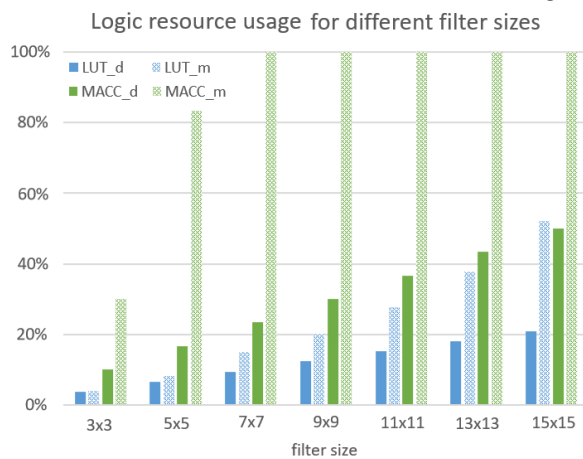


Figure 6. Logic resource usage for different filter sizes for developed (“_d”) and MATLAB’s (“_m”) implementation of image filtering.

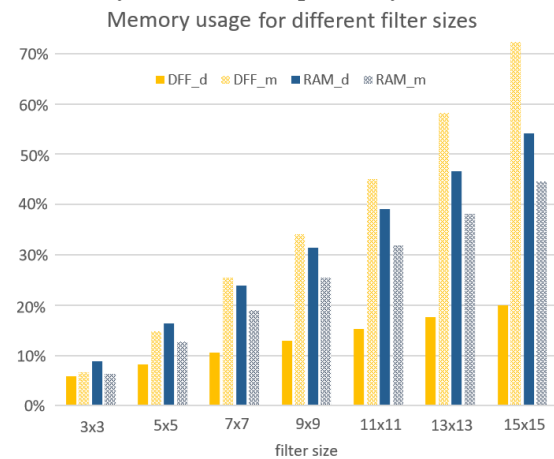


Figure 7. Memory usage for different filter sizes for developed (“_d”) and MATLAB’s (“_m”) implementation of image filtering.

From Fig. 6 it can be seen that with MATLAB’s implementation all the available MACC blocks get used up already for filter size 7x7. This is because their implementation does multiplications on all filter coefficients in parallel, meaning the number of multipliers needed increases with the square of the filter size. Once there are no more MACC blocks available, the multipliers are implemented in fabric, increasing fabric logic allocation with a square as well. On the other hand, the developed implementation uses a number of multipliers proportionate to the filter size by doing the multiplications at an increased frequency. As it can be seen in the figure, this way less logic resources are used.

A similar increase as with the logic resources can be seen in the usage of DFF for both implementations in Fig. 7. This is because they are used to store the intermediate results of the calculations. On the other hand, RAM blocks increase linearly in both implementations as they are only used for storing the number of rows corresponding to the filter size. One can note that MATLAB’s implementation is more efficient in respect of allocated RAM as it uses relatively 25% less RAM blocks than the developed implementation. This is because of a difference in the implementations of zero padding, which results in different allowed minimal blanking intervals between image lines.

Based on the resource usages, power consumptions for both implementations are calculated. They are shown in Fig. 8. The power consumptions are calculated for pixel clock frequency of 37.125

MHz. The developed implementation cannot be implemented in the used system for filter sizes larger than 9x9 due to the frequency limitations of the FPGA, which is 400 MHz. Nevertheless, the power consumptions of filters larger than 9x9 are still included in the figure (marked with a light pattern) for complete analysis.

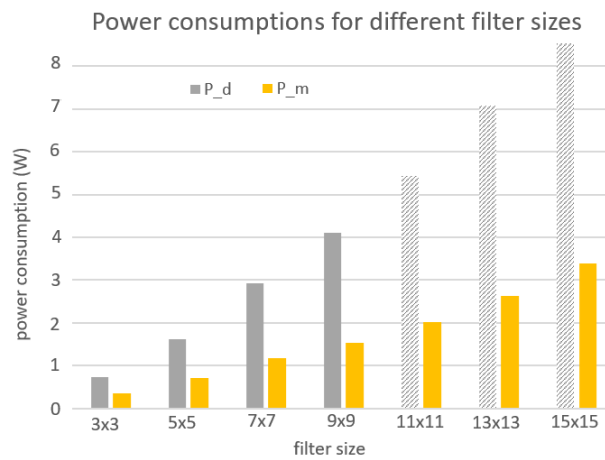


Figure 8. Power consumption for different filter sizes for developed (“_d”) and MATLAB’s (“_m”) implementation of image filtering operating at 37.125 MHz.

The power consumption is proportionate to the number of resources and the frequency at which they operate [30]. As the number of resources increases with a square for the MATLAB’s implementation, the power consumption follows the trend as it can be seen in Fig. 8. Moreover, also the developed implementation follows a square trend because the resources increase linearly but also the frequency of operation of multipliers. The important thing to note here is that power consumption of the developed implementation is larger than MATLAB’s implementation. The reason is that the developed implementation does not use for a factor of the filter size less resources than MATLAB’s as one might first think. In order to operate the multipliers at an increased frequency, a control circuit is needed. The control circuit increases the resource usage. Moreover, it operates at the increased frequency, resulting in a larger overall power consumption of the implementation.

The achievable frame rates for the used FPGA are shown in Fig. 9. The figure shows that frame rate of MATLAB’s implementation is slightly decreasing. This is because it needs to have the blanking interval between the lines of an image at least as long as the filter size. This issue is more noticeable in cases where the ratio of the filter and image sizes is larger. However, the developed implementation does not have this limitation. It operates at the minimum blanking interval needed for detecting transitions of the synchronization signals, namely one clock cycle. Although its achievable frame rate is still decreasing. It decreases proportionally with the filter size. That is because the clock frequency of pixels is limited to the ratio of the maximum frequency of the FPGA and the filter size.

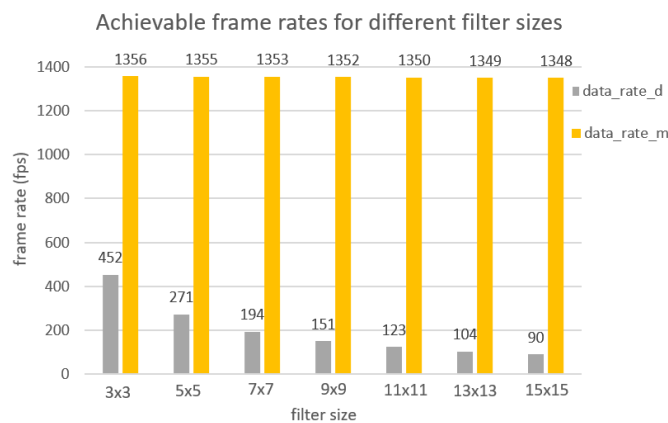


Figure 9. Achievable frame rates for different filter sizes for developed (“_d”) and MATLAB’s (“_m”) implementation of image filtering for image size of 1216 by 1936 pixels.

3.2. Image Averaging

For verification of image averaging, the parameter L of the filter is set to 8. Moreover, a set of nosy images is generated with MATLAB using function 'imnoise'. The noise is set to have a Gaussian distribution with zero mean value and variance of 0.02. An example of a nosy image is shown at the left of Fig. 10, followed by the result of the simulated circuit (middle image) and the reference image (right image). The reference image is represented by the not-simplified version of the running average filter. By comparing the three images, it can be seen that the noise level is decreased due to the use of the running average filter. Fig. 10 also includes the histogram of the absolute difference between the simulated and the reference image. From the histogram, a discrepancy between the two filters is noticeable. Nevertheless, the difference is small enough to be neglected for most practical purposes. Therefore, these results validate the simplification and implementation of the averaging algorithm.



Figure 10. Verification of developed image averaging. Left image – generated nosy image; middle image – simulated result of implemented image averaging; right image – reference output; and histogram of absolute difference between the simulated and the reference image.

Evaluation of the resource usage for the implementation of the image averaging algorithm can be seen in Table 1. Based on it, the power consumption when operating at 37.125 MHz is calculated to be 971 mW. As already indicated, the achievable frame rate is limited by the access data rate to the external DDR memory, resulting in a maximum frame rate of 135.93 fps.

Table 1. Evaluation of image averaging algorithm

| Resource | LUT | DFE | MACC | RAM |
|-----------------------------|-----|-----|------|-----|
| Percentage of Resource Used | 2% | 1% | 23% | 2% |

4. Performance Comparison with other Published Works

To further evaluate the performance of each of the developed algorithm implementations, they are compared with different FPGA implementations of the same algorithms from other published works. The performance is evaluated in terms of processing data rates and power consumptions. Moreover, since the two metrics also depend on the available resources of the platform used in the tests, the ratio of processing data rate versus power consumption is also considered because it better reflects the efficiency of individual implementations.

While there are multiple works investigating the topic of implementation of the two presented algorithms in a FPGA, only a few provide an evaluation of the achievable processing data rates and/or power consumptions. Therefore, only the latter can be considered here for comparison. The works [31–34] have been found to include an investigation of the implementation of image filtering algorithm, while performance of the implementation of image averaging has only been found to be investigated in [33].

4.1. Approaches to Performance Comparison

The main issue faced when conducting a performance comparison of image processing algorithm implementations, is inconsistent format of the reported evaluation between different

published works. For example, it is common that as a performance metric, frame rate is reported rather than processing data rate. The two metrics are related as:

$$\text{processing data rate} = \text{frame rate} * \text{image size} * \text{bit depth} \quad (4)$$

While it is simpler to understand what frame rate represents, it is more informative to express the processing data rate when comparing algorithm implementations. The image size and/or bit depth are application specific and not algorithm dependent. Although there are some typical values for them, they can be freely chosen for a specific application.

A similar issue with inconsistent metrics is present with power consumption. It is defined as the average power consumption during processing of a frame, or in other words, the average energy invested for processing of a single frame. But different publications include different contributions to the considered power consumption. One contribution comes from the so-called static power consumption when the platform running the algorithm is idle. The other contribution is referred to as dynamic and stems from running the implemented algorithm itself. In the evaluation in this paper, we consider their sum as the power consumption of the algorithm because of two main reasons. Firstly, it is difficult to completely separate evaluation of the different power contributions. And secondly, the algorithm implementation can be adapted to use the advantages of the targeted platform, thus it inherently also includes its characteristics like the static power consumption.

Moreover, to have a meaningful comparison it is important to consider implementations with the same parameters, e.g. same filter sizes and precision of the calculated results. However, due to specific platform limitations and characteristics (e.g. available hardware multipliers), this might not necessarily be feasible to achieve. All this has a consequence that the comparison of the implementations is at least to some degree dependent on the platform used for evaluation.

4.2. Results of Performance Comparison

When applying the conversion of the different approaches to the performance comparison discussed in the previous section to the data available from [31–34], the Tables 2 and 3 are produced. It should be noted that for [31,32] the power consumption has been estimated based on the reported resource usage and power estimator tools for the used platforms, i.e. ⁷ and ⁸ respectively. Moreover, the comparison of different implementation of image filtering is conducted for filter size of 9 by 9.

Table 2. Implementation performance comparison for image filtering

| Implementation | FPGA platform used | Processing data rate [Gbps] | Power consumption [W] | Efficiency [Gbps/W] |
|----------------|--|-----------------------------|-----------------------|---------------------|
| Developed | Microsemi Smartfusion2 Advanced Development Kit | 2.8 | 4.1 | 0.7 |
| [31] | GiDEL ProcStar III with Altera Stratix III E260 FPGA | 1.0 | 2.3 | 0.4 |
| [32] | GiDEL ProcStar IV with Altera Stratix IV E530 FPGA | 1.2 | 3.2 | 0.4 |
| [33] | Xilinx Zynq UltraScale+ MPSoC ZCU102 | 2.4 | 3.0 | 0.8 |
| [34] | Xilinx XC4VLX160 | 0.85 | Not available | Not available |

Table 3. Implementation performance comparison for image averaging

| Implementation | FPGA platform used | Processing data rate [Gbps] | Power consumption [W] | Efficiency [Gbps/W] |
|----------------|---|-----------------------------|-----------------------|---------------------|
| Developed | Microsemi Smartfusion2 Advanced Development Kit | 5.1 | 0.97 | 5.3 |
| [33] | Xilinx Zynq UltraScale+ MPSoC ZCU102 | 2.4 | 3.1 | 0.77 |

⁷ Intel. PowerPlay Early Power Estimator Download Stratix III Devices. 2020.

https://www.intel.com/content/www/us/en/programmable/support/support-resources/operation-and-testing/power/st3-power_estimator_download.html

⁸ Intel. Stratix IV and Stratix V Early Power Estimator Download. 2020.

<https://www.intel.com/content/www/us/en/programmable/support/support-resources/operation-and-testing/power/st4-5-estimator-download.html>

From the unified performance evaluation metrics shown in the two tables, it can be concluded that the developed algorithm implementations presented in this paper are amongst the most efficient in terms of achievable processing data rates per consumed power compared to implementations from other publications. Moreover, the developed implementations also achieve the highest processing data rates. Nevertheless, it would be presumptuous to attribute these performance metrics only to the algorithm implementations. The platforms used need to be taken into account as well, as already noted.

5. Conclusion

This paper presented FPGA implementations of two standard image preprocessing algorithms, namely image filtering and image averaging. The implementations allow real-time processing of high-frame-rate and high-resolution image streams. The implementations were developed in Simulink and verified by simulating the generated circuits and comparing the output images with images processed with MATLAB's functions from the Image Processing Toolbox. Moreover, the implementations were also verified by testing them in an imaging system consisting of Microsemi's Smartfusion2 Advanced Development Kit to which an industrial camera was connected via the Camera Link interface. The developed implementations were evaluated in terms of resource usage, power consumption, and achievable frame rates. The performance of the developed implementation of the image filtering algorithm was also compared with a solution provided by MATLAB's Vision HDL Toolbox, which was also implemented and evaluated on the same imaging system.

The evaluation of the two implementations of image filtering showed that the developed one uses fewer resources than MATLAB's version. However, its power consumption is larger due to the processing at higher frequency. Reason being that the additional control logic needed for processing at the increased frequency is also operating at that increased frequency, resulting in an overall larger power consumption even though fewer resources are used. The processing at higher frequency is also the reason for decreasing the achievable frame rate with increasing filter size, in contrast to MATLAB's implementation. Nevertheless, the developed implementation has the advantage of operating at the minimum blanking interval between the lines of an image, i.e. one clock cycle, in comparison with MATLAB's version where the interval needs to be at least as long as the filter size. Despite the lower performance of the developed implementation of image filtering compared to MATLAB's version, the advantage of the smaller resource usage can prove to be crucial in complex applications, e.g. in state-of-the-art object recognition with convolutional neural networks (CNNs). For real-time object detection with CNNs, multiple image filters need to operate in parallel, which can only be achieved with small designs of filtering.

Furthermore, the implementation of image averaging showed that the complexity of the algorithms, which can be implemented in a system, is also greatly limited by the available access data rate to the external DDR memory. Nevertheless, it was demonstrated how this limitation can be overcome by simplifying the algorithm by making reasonable assumptions, e.g. that pixels do not change much through time which can be achieved by capturing images with a high enough frame rate.

The performance of the developed implementations was also compared to performance of the algorithm implementations from other published works. The comparison showed that the developed implementations perform well in comparison with others. Nevertheless, it should be noted that the performance is not only dependent on the algorithm but also on the platforms used for the evaluation. Therefore, for a more informative comparison of the different algorithm implementations, the same platform should be used as it was done with the MATLAB's solution. However, in some cases even this would not result in a completely clear comparison because some implementations can better take advantage of and be adapted to the specific platform characteristics. This just further shows how large and divers the design space for real-time imaging applications is, and that when designing a real-time imaging applications, the algorithm implementations and the platform used should be considered together to find the optimal solution.

References

- [1] Benjamin Ranft and Christoph Stiller, "The Role of Machine Vision for Intelligent Vehicles", *IEEE Transactions on Intelligent Vehicles*, Print ISSN: 2379-8904, Online ISSN: 2379-8858, pp. 8–19, Vol. 1, No. 1, March 2016, DOI: 10.1109/TIV.2016.2551553.
- [2] Yi Yang, Hengliang Luo, Huarong Xu and Fuchao Wu, "Towards Real-Time Traffic Sign Detection and Classification", *IEEE Transactions on Intelligent Transportation Systems*, Print ISSN: 1524-9050, Online ISSN: 1558-0016, pp. 2022–2031, Vol. 17, No. 7, July 2016, DOI: 10.1109/TITS.2015.2482461.
- [3] Soharab Hossain Shaikh, Khalid Saeed and Nabendu Chaki, "Moving object detection using background subtraction", In: *Moving Object Detection Using Background Subtraction*. SpringerBriefs in Computer Science, Cham, Switzerland: Springer, 2014.
- [4] Susmita A. Meshram and Rani S. Lande, "Traffic surveillance by using image processing", *2018 International Conference on Research in Intelligent and Computing in Engineering (RICE)*, Published by IEEE, San Salvador, pp. 1–3, August 2018, DOI: 10.1109/RICE.2018.8627906.
- [5] Geoff Dougherty, *Digital Image Processing for Medical Applications*, 1st ed. Cambridge, UK: Cambridge University Press, ISBN: 9780521860857, 2009.
- [6] Bruce G. Batchelor, Eds., *Machine vision handbook*, Cham, Switzerland: Springer, ISBN: 978-1-84996-170-7, 2012.
- [7] Kazunori Akiyama, Antxon Alberdi, Rebecca Azulay, Anne-Kathrin Baczko, Mislav Baloković *et al.*, "First M87 Event Horizon Telescope Results. III. Data Processing and Calibration", *The Astrophysical Journal Letters*, p. 52, 2019, DOI: 10.3847/2041-8213/ab0e85.
- [8] Sebastian Beulig, "Real-time anomaly and pattern recognition in high-resolution optical satellite imagery for future on-board image processing on Earth observation satellites", PhD Thesis, Universität der Bundeswehr München, Germany, 2014, Available: <https://athene-forschung.unibw.de/doc/97069/97069.pdf>.
- [9] Bud Brown, "Camera Connections", *Vision Systems Design*, April 1, 2008, Available: <https://www.vision-systems.com/non-factory/environment-agriculture/article/16739233/camera-connections> (accessed on 22.9.19).
- [10] Chao Li, Souleymane Balla-Arabe and Fan Yang-Song, *Architecture-Aware Optimization Strategies in Real-time Image Processing*, New Jersey, USA: John Wiley & Sons, Inc., ISBN: 978-1-786-30094-2, DOI: 10.1002/9781119467243, 2017.
- [11] Kang G. Shin and Parameswaran Ramanathan, "Real-time computing: a new discipline of computer science and engineering", *Proceedings of the IEEE*, Print ISSN: 0018-9219, pp. 6–24, Vol. 82, No. 1, January 1994, DOI: 10.1109/5.259423.
- [12] Rafael C. Gonzalez and Richard E. Woods, *Digital Image Processing*, 2nd ed. New Jersey, USA: Prentice-Hall, January 2002.
- [13] Nadia Nacer, Bouraoui Mahmoud and Mohamed Hedi Bedout, "FPGA Architecture for Facial-Features and Components Extraction", *International Journal of Computer Science, Engineering and Information Technology*, Vol. 3, No. 2, pp. 51-64, DOI: 10.5121/ijcseit.2013.3204, 2013.
- [14] Michael Schoberl, Jurgen Seiler, Siegfried Foessel and Andre Kaup, "Increasing imaging resolution by covering your sensor", *2011 18th IEEE International Conference on Image Processing*, Brussels, pp. 1897–1900, , DOI: 10.1109/ICIP.2011.6115839, September 2011.
- [15] M. Fukuzawa, H. Hama, N. Nakamori and M. Yamada, "A high-frame-rate embedded image-processing system by using a one-chip DSP", *2008 IEEE International Conference on Electrical and Computer Engineering*, Dhaka, Bangladesh, pp. 412–416, DOI: 10.1109/ICECE.2008.4769242, 2008.
- [16] Khairul Muzzammil Saipullah, Ammar Anuar, Nurul Atiqah Ismail and Yewguan Soo, "Measuring Power Consumption for Image Processing on Android Smartphone", *American Journal of Applied Sciences*, Print ISSN: 1546-9239, pp. 2052–2057, Vol. 9, No. 12, DOI: 10.3844/ajassp.2012.2052.2057, December 2012.
- [17] Jai, "GO-2400-PMCL datasheet", 2019, Available: https://www.jai.com/uploads/documents/English-Manual-Datasheet/GO-Series/Datasheet_GO-2400-PMCL.pdf.
- [18] SiliconSoftware, "Datasheet SiliconSoftware microEnable 5 marathon VCL", 2019, Available: https://silicon.software/pdf-generator/?product_id=763.
- [19] Microsemi, "UG0557: User Guide SmartFusion2 SoC FPGA Advanced Development Kit revision 4", 2016.
- [20] Uroš Hudomalj, "Real-Time Processing of Image Streams", Master's thesis, University of Ljubljana, Ljubljana, Slovenia, 2019, Available: <https://repozitorij.uni-lj.si/Dokument.php?id=124554&lang=eng>.
- [21] Jai, "User Manual: GO-2400M-PMCL and GO-2400C-PMCL", 2019, Available: https://www.jai.com/uploads/documents/English-Manual-Datasheet/GO-Series/Manual_GO-2400-PMCL.pdf.
- [22] Uroš Hudomalj, Christopher Mandla, Markus Plattner, "FPGA Implementations for Real-Time Processing of High-Frame-Rate and High-Resolution Image Streams", *2020 IEEE International Conference on Computing*,

- Electronics Communications Engineering (ICCECE)*, pp. 211–216, August 2020, DOI: 10.1109/ICCECE49321.2020.9231119.
- [23] Microsemi, "DS0128: Datasheet IGLOO2 FPGA and SmartFusion2 SoC FPGA revision 12", 2018.
- [24] Microsemi, "Application Note AC422: SmartFusion2 - Optimizing DDR Controller for Improved Efficiency - Libero SoC v11.7", 2016.
- [25] Mike Wakin, "Standard test images - Lena/Lenna", May 13, 2003, Available: <https://www.ece.rice.edu/~wakin/images/lena512.bmp> (accessed on 22.9.19).
- [26] Microsemi, "Libero SoC v12.0 and later", 2019, Available: <https://www.microsemi.com/product-directory/design-resources/1750-libero-soc> (accessed on 22.9.19).
- [27] Microsemi, "Microsemi Power Estimator (MPE) - v10d: SmartFusion2 and IGLOO2", 2015.
- [28] Jeff Janzen, "Calculating memory system power for DDR SRAM", *Micron designline*, Vol. 10, No. 2, 2001.
- [29] Micron Technology, Inc., "Datasheet for MT41K128M16 DDR3L SDRAM", 2015.
- [30] T Grzes, V Salauyou and I Bulatova, "Power estimation methods in digital circuit design", *Optoelectronics, Instrumentation and Data Processing*, pp. 576–583, Vol. 45, No. 6, 2009.
- [31] Jeremy Fowers, Greg Brown, Patrick Cooke and Greg Stitt, "A performance and energy comparison of FPGAs, GPUs, and multicores for sliding-window applications", *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays - FPGA '12*, ACM Press, Monterey, California, USA, p. 47, 2012, DOI: 10.1145/2145694.2145704.
- [32] Patrick Cooke, Jeremy Fowers, Greg Brown and Greg Stitt, "A Tradeoff Analysis of FPGAs, GPUs, and Multicores for Sliding-Window Applications", *ACM Transactions on Reconfigurable Technology and Systems*, Print ISSN: 1936-7406, Online ISSN: 1936-7414, pp. 1–24, Vol. 8, No. 1, March 2015, DOI: 10.1145/2659000.
- [33] Murad Qasaimeh, Kristof Denolf, Jack Lo, Kees Vissers, Joseph Zambreno *et al.*, "Comparing Energy Efficiency of CPU, GPU and FPGA Implementations for Vision Kernels", *15th IEEE International Conference on Embedded Software and Systems*, Nevada, United States, pp. 1-8, May 2019.
- [34] Shuichi Asano, Tsutomu Maruyama, Yoshiki Yamaguchi, "Performance comparison of FPGA, GPU and CPU in image processing", *2009 IEEE International Conference on Field Programmable Logic and Applications*, Prague, Czech Republic, pp. 126–131, August 2009, DOI: 10.1109/FPL.2009.5272532.



© 2021 by the author(s). Published by Annals of Emerging Technologies in Computing (AETiC), under the terms and conditions of the Creative Commons Attribution (CC BY) license which can be accessed at <http://creativecommons.org/licenses/by/4.0>.