Image Processing on the Edge using an Entry-Level FPGA

Armin Zunic, Emir Sokic, Nedim Osmic, Isam Vrce, Almir Salihbegovic
Faculty of Electrical Engineering, University of Sarajevo
Sarajevo, Bosnia and Herzegovina
{azunic1, esokic, nosmic, ivrce1, almir.salihbegovic}@etf.unsa.ba

***Abstract*—This paper explores the application of FPGA programmable structures in the field of digital image signal processing (ISP). FPGAs offer high flexibility, speed and parallelism, making them ideal for general digital signal processing (DSP), as well as specific ISP tasks. The paper utilizes standard ISP algorithms such as morphological operations, filtering and edge detection to compare practical implementations of FPGA and CPU-based compute engines. Through illustrative examples and empirical results, we demonstrate the distinct advantages of employing FPGA for these use-cases, and contrast them with traditional CPU approaches, clearly showing FPGA capacity to significantly accelerate execution. The challenges that arise from resource-limited IOT-class hardware configurations are highlighted in the paper, namely resource optimization, memory management and maximal frequency.**

***Keywords*—FPGA, IOT, ISP, Parallelism, Acceleration**

I. Introduction

With the rapid advancement of technology, there is a significant demand for the extraction of useful information from visual data, using image processing methods [1]. Depending on the method or algorithm of image processing, different parts or steps are preferred to be executed on separate processors, i.e. in parallel. This can ultimately result in a completely parallel structure of the processing workflow.

FPGA (Field-Programmable Gate Array) represents a unique type of chip characterized by low-level programmability. Due to the advancement of programmable FPGA technologies, there has been a growing interest in utilizing these structures for DSP and ISP. During the past decade, considerable research has been dedicated to the exploration of FPGA-based image processing. Parallelization and digital image processing at the fundamental level are particularly highlighted in [1]. In articles [2] and [3] the execution speeds of image processing algorithms on FPGAs, CPUs (Central Processing Units) and GPUs (Graphics Processing Units) are compared. The focus is on filters, essential components of any image processing operation. Both FPGA-optimized OpenCV libraries and proprietary implementations of algorithms are compared in [2], where they showed that custom-made implementations are superior with increased filter size. When it comes to the complexity and scope of algorithms, [3] stands out over [2]. They have compared 36 image processing algorithms categorized into 6 groups, and showed that for each group GPUs and FPGAs outperform CPUs. The conclusion drawn from both studies is that FPGAs are optimal in terms of both maximizing execution speed and energy efficiency.

The main aspects of digital image processing using FPGAs are discussed in paper [4], while [5] also discusses the Canny edge detector example. The described algorithm offers a detailed overview of the resource utilization used by the hardware employed. The authors in [6] describe the resource consumption of the FPGA system in more detail, including the maximum frequency and energy consumption. They concluded that the energy consumption for filter implementations was much higher than for the same algorithms described in [3], due to the hardware used.

The purpose of our study is to compare the performance of image processing algorithms on an entry-level FPGAs versus CPUs, with all their pros and cons. Execution times for identical algorithms will be evaluated on both FPGA and CPU platforms and benchmarked against relevant studies in this research field using established metrics. The primary goal is to explore and demonstrate the capabilities of FPGA-based digital image processing, particularly on cost-effective IOT hardware, highlighting its potential for real-time applications. It will be demonstrated that the parallelization offered by programmable FPGA structures can significantly accelerate certain digital image processing algorithms, potentially matching or even surpassing the speeds of much more expensive hardware such as desktop or server-class CPUs.

The paper is structured as follows. Section II presents the used FPGA board, while image vizualization is discussed in Section III. Implementation of image processing algorithms on FPGA is presented in Section IV. Results are discussed in Section V, with the conclusion provided in the last section.

II. FPGA Development System TangNano20K

Xilinx and Altera are two main manufacturers of FPGA devices [7]. In contrast, our work will be based on an FPGA chip from the GoWin Semiconductor Corp, and Sipeed development board TangNano20K. Both are known for efficiency and speed, yet attractive pricing structure. For comparison, it may be seen from Table I that TangNano20K board offers the lowest cost per flip-flop (FF), look-up-table (LUT) and memory (BSRAM). However, as these entry-level devices are not optimized for image processing applications, the implementation of image acquisition, processing and visualization on them is more challenging.

2024 International Symposium ELMAR | 979-8-3503-7542-8/24/$31.00 ©2024 IEEE | DOI: 10.1109/ELMAR62909.2024.10694571

Figure 1. Tang Nano 20k FPGA development board.

TABLE I. COST-EFFECTIVENESS COMPARISON OF DIFFERENT FPGA

	Xilinx Zynq 7020	Xilinx Zynq ZCU102	Altera Stratix 3	**Tang Nano 20k**
LUTs	53k	600k	142k	20k
FFs	106k	550k	135k	15k
BSRAM	4.9Mb	32.1Mb	5.5Mb	0.8Mb
Cost ($)	1100	3200	750	25
Cost/LUTs ($)	20.75	5.33	5.28	**1.25**
Cost/FFs ($)	10.37	5.82	5.55	**1.67**
Cost/Mb ($)	224.49	99.69	136.36	**31.25**

These hardware compute platforms are classified by the number of LUTs that the on-board FPGA chip contains. The work will focus mainly on the TangNano20K development board, which hosts an FPGA from the Gowin 2A series, namely the GW2AR18-QN88. The FPGA has 20k LUTs, 15k FFs, 41kb Shadow Static Random Access Memory (SSRAM), 828kb Block Static Random Access Memory (BSRAM), 64Mb 32bits Synchronous Dynamic Random Access Memory (SDRAM), 48 18×18 Multipliers, 2×Phase-locked loops (PLLs) and 8×I/O bank.

The board provides 40 pins, 34 of which are General-Purpose Input/Output (GPIO). The rest are power supplies. All pins are shown in Fig. 1.

The TangNano20K comes with a rich set of peripherals, including: 64Mbit QSPI Flash, MicroSD card slot, HDMI output, RGB LCD connector, pins for speakers or headphones, RGB LED (WS2812), 6 programmable LEDs, 2 programmable buttons, one USB Type-C port, 5V/0.5A power supply via USB-C, clock generator (MS5351), RISC-V SOC (BL616), PCM amplifier (MAX98357A). Other useful software tools for developing applications with the chosen board are GoWin FPGA Designer - GoWin EDA (which allows

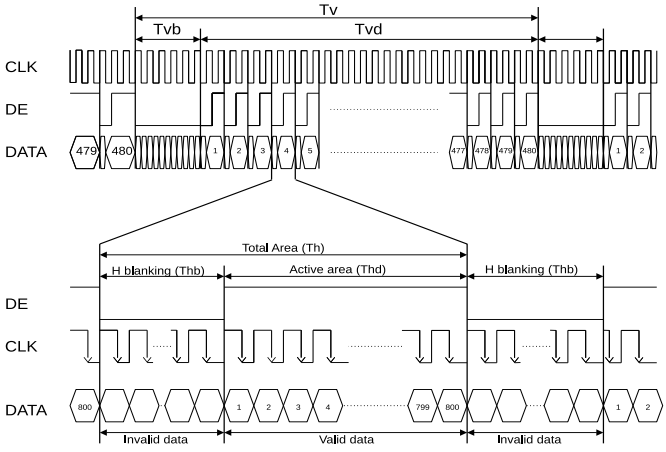
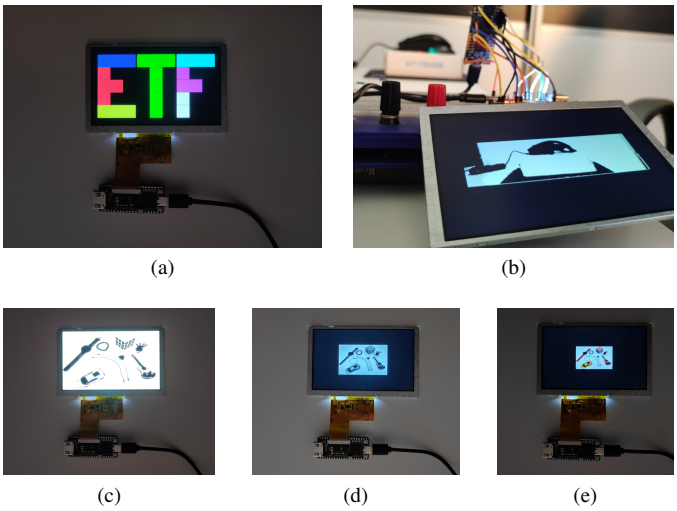


Figure 2. Timing diagram of relevant signals for the LCD display.



(a) (b) (c) (d) (e)

Figure 3. Vizualization of: a) synthetized image, b) image acquired by camera and processed on FPGA, c) binarized image from memory, d) grayscale image from memory, e) color image from memory.

writing programs/configurations for the FPGA chip, its synthesis, routing, bitstream generation and transfer), DSim Cloud (for the testing, simulation and verification of the FPGA chip design, performance analysis - particularly speed or execution time), and GTKWave (displaying signals).

III. VISUALIZING IMAGES ON EXTERNAL DISPLAY

The simulator is adept at precisely interpreting particular signals and tracking their progression over time, however, it is the visual display that provides users with a comprehensive understanding of the entire process. The TangNano20K development board features an RGB LCD connector that facilitates the connection of the screen and the board. The screen used in this work is the 5" Sipeed SH500J01Z with 800x480 resolution. The typical frequency for displaying individual pixels on the LCD screen needs to be 33.3 MHz, which ensured a screen refresh rate of 60 Hz for the entire image. It was particularly challenging to ensure the desired pixel frequency so that the image on the screen is clear and flicker-free. In order to ensure a stable image, the FPGA board needed to produce the required signals, namely horizontal and vertical synchronization, data enable signal and signal for data (timing diagram presented in Fig. 2).

A program that "draws" the image on the LCD is written in System Verilog RTL, using "ray chasing" method – each pixel is computed at exact moment when used, so alleviating the need for memory resources, be they in the form Line or Frame Buffer. Thus, the just-in-time computed value of each pixel is immediately displayed on the screen. An example of this is shown in Fig. 3 a). In summary, the image is generated very efficiently with minimal resource utilization.

A. Displaying an image read from memory

Binary Image - In the case of displaying a binary image, where each pixel occupies exactly one bit, then for an image size of 800x480, it will take 384000 bits of BSRAM, which

will fit in the available BSRAM. Thus the image can be displayed on the screen in full resolution. The display of the binary image read from memory is shown in Fig. 3 c).

Grayscale Image - To begin with, a single block of BSRAM offers 18,432 bits. Given that the selected entry-level FPGA chip contains 46 BSRAM blocks, the total BSRAM storage adds up to 847,872 bits. While it is feasible to store data of any size in the blocks, the arrangement may vary, and the total memory capacity might not be fully utilized. Given that each pixel in a grayscale image uses 8 bits, blocks of 2Kx8 are utilized (each block holds 16,384 bits). With 46 such blocks, the total available storage is 753,664 bits. Thus, the maximum image size that can be displayed on the screen is 368x256, as in Fig. 3 d).

Full Color Image Display - The screen supports the RGB565 color format, which means that each pixel occupies exactly 16 bits. Consequently, in this case, the maximum image size will be 284x176, as shown in Fig. 3 e).

The primary challenge was maintaining the required pixel refresh frequency due to pixel address calculations during image display. While memory usage was maximized, other resources were minimally used. Achieving the f_{max} target was straightforward, allowing a design focused on combo logic to reduce power consumption, aligning with the energy-efficient edge IoT context. Displaying images from memory used less than 1% of Lookup Tables and Flip-Flops, with ample timing margins.

B. Displaying an image acquired from camera

This work also covered the aspect of real-time image processing using a camera (OV2640) as input and a screen as output. The TangNano20K does not have a direct camera input, thus we opted to use an external DVP (Digital Video Port) as in Fig. 3 b). Synchronizing the camera and screen, while executing digital image processing algorithms in real-time, presented a particular challenge. Data handoffs are implemented using source-synchronous, eye-centered interfacing method. The camera keeps streaming data into FPGA using the classic "Push" interface. On the other hand, the Image/Video Processor takes data out of the Pixel Buffer, which is a "Pull" interface. To make it more-efficient in terms of LUT utilizaton, the processing algorithm takes advantage of blanking intervals.

Listing 1. Formulas for Calculating Pixels for Sobel Edge Detector

```
assign f5 =
  (pixel6 >> 2) + (pixel8 >> 2) + (pixel9 >> 2);
assign f6 =
  (pixel1 >> 2) + (pixel2 >> 2) + (pixel4 >> 2);
assign f7 =
  (pixel4 >> 2) + (pixel7 >> 2) + (pixel8 >> 2);
assign f8 =
  (pixel2 >> 2) + (pixel3 >> 2) + (pixel6 >> 2);

assign output_pixel = {8{~(
       ((f5 >= f6) & ((f5 - f6) < threshold))
     |  ((f6 >= f5) & ((f6 - f5) < threshold))
     &  ((f7 >= f8) & ((f7 - f8) < threshold))
     |  ((f8 >= f7) & ((f8 - f7) < threshold))
)}};
```

This means that, in order not to drop data, the Pixel Buffer absorbs the resulting discrepancy between ingress and egress throughput, despite having the same clock rate and data width, both in and out. It does it by running Half Full most of the time, which yields pixel elasticity of a half-buffer depth. The RGB LCD back-end does not maintain the full Store-and-Forward Frame Buffer (FB) either. Instead, it provides the minimal amount of Cut-Through Pixel Buffering, essentially just enough to smooth out the occasional bursts of video output from the processor. Substantial memory saving is realized in this way compared to the standard video buffering methods.

IV. Implementation of machine vision algorithms

Parallel execution of kernel-based algorithms on FPGAs requires multiple memory readers. Using 9 readers for a 3x3 kernel increases memory usage and limits image size. Reducing to 3 readers improves memory efficiency, allows for larger images, and reduces LUT usage. Thus, execution time depends mostly on f_{max}, which is influenced by computational load.

Besides the Sobel algorithm, detailed in the next section, we implemented morphological operations and the Gaussian filter. Erosion and dilation were performed on FPGAs using AND and OR gates. The Gaussian filter was implemented through convolution with Gaussian kernels for image smoothing.

A. Implementation of Sobel edge detector on FPGA

The Sobel edge detector is an algorithm based on the gradient method. This method finds edges using a horizontal mask (HM) and a vertical mask (VM). Examples of such 3x3 masks are as follows:

$$\text{HM} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}, \quad \text{VM} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

These masks are convolved with the input image. Assume that a 3x3 segment of the input image is given as a matrix $P = \{P_{ij}\}$ $(i = 1, \ldots, 3,\ j = 1, \ldots, 3)$.

The horizontal gradient is given as follows

$$G_x = (P_7 + 2 \cdot P_8 + P_9) - (P_1 + 2 \cdot P_2 + P_3) \qquad (1)$$

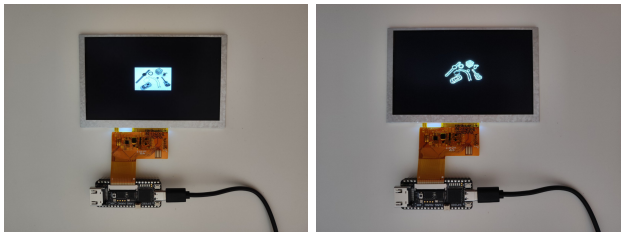
and the vertical gradient is given by

$$G_y = (P_3 + 2 \cdot P_6 + P_9) - (P_1 + 2 \cdot P_4 + P_7) \qquad (2)$$

The standard Sobel edge detector algorithm represents the resultant gradient as: $G = |G_x| + |G_y|$. If the gradient G of a specific pixel (e.g. P_4) is greater than a set threshold, then that pixel is considered as an edge; otherwise, it does not constitute an edge.

The lines of code implementing the equations (1) and (2) in System Verilog RTL are given in Listing 1.

The image size that could be processed with the Sobel edge detector, using the maximum amount of BSRAM, is 222x138 pixels. The original image and the image after applying the Sobel operator are shown in Fig. 4 a) and Fig. 4 b) respectively.



(a) (b)

Figure 4. a) Original image on FPGA, b) Edge-detection using FPGA.

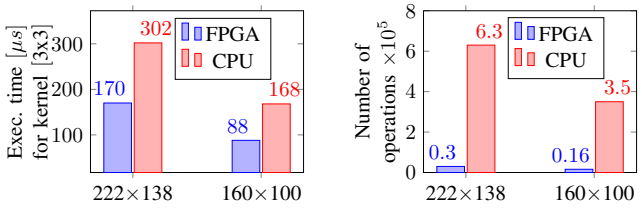


Figure 5. Comparison of execution times and number of operations for Sobel edge detection on grayscale image (FPGA vs CPU).

B. Implementation of Sobel edge detector on CPU

The Sobel operator is implemented using the PC with AMD Ryzen 5 3500U processor, 2.10 GHz, and the C++ OpenCV library (v.4.5.4). This results in images with marked horizontal and vertical gradients. These are combined into a binary image that represents the resulting gradient.

V. Experimental results and discussion

The time required to apply the Sobel edge detector on an image size of 222x138 with a 3x3 kernel on the FPGA at a frequency of 180 MHz is given as $T_{222\times138,3\times3,Sobel} \approx 170.2\mu s$. In the experiment, every image processed by either the FPGA or the CPU is in 8-bit format. This calculation can also be confirmed via the simulator. The average execution time of the algorithm described on the CPU was 302.4 μs. The FPGA and CPU execution time using the same setup, but on a smaller image size 160x100, is 88.89μs and 168.8 μs respectively (Fig. 5). The FPGA implementation of the Sobel operator is clearly superior to the CPU.

The maximum execution frequency of the Sobel edge detector in [8], on a more powerful Xilinx Spartan 3 (2.5× Block RAM capacity, 75k of LUTs) is 190 MHz, which is comparable to our 180 MHz. In addition, the frequency on the FPGA is fixed for any image size and kernel size. Thus, increasing the kernel size with the current image size, the execution time for processing on the FPGA remains the same, while that time on the CPU increases. However, it is important to mention that the increase of the kernel size results in higher memory demands on the FPGA chip.

Other image processing algorithms have been implemented and tested (Table II). For morphological operations (e.g. dilatation/erosion), reduction of the image size and increasing the kernel size yields to shortening FPGA times, which is not the case for CPU implementation. A similar conclusion applies to the Gaussian filter. In addition, for a kernel size of 5x5, the execution times on the FPGA and CPU are nearly identical. It is important to note that the working frequency of the CPU was an order of magnitude higher compared to the maximum execution frequency of the FPGA implementations. The same goes for energy efficiency. When comparing the number of operations required for algorithms, FPGA are much dominant (e.g. Fig. 5). In addition, as the complexity of the method increases, or it is suitable for parallel implementation, the FPGA structures outperform the CPU.

TABLE II. Comparison of Image Processing Algorithms Execution Time on FPGA and CPU

Algorithm	**FPGA res. vs time**		**CPU res. vs time**	
	630x390	488x302	630x390	488x302
Morph. oper. [3x3, 5x5]	867μs	496μs	82μs	67μs
	252x156	172x106	252x156	172x106
Gauss. filter [3x3, 5x5]	144μs	66μs	48μs	42μs
	222x138	160x100	222x138	160x100
Sobel oper. [3x3]	170μs	88μs	302μs	168μs

Conclusion

This paper presented an implementation of image acquisition, processing and visualization on a cost-effective FPGA architecture and highlighted its prospective advantages. Through the analysis of implemented algorithms, it is evident that parallel processing capabilities facilitate rapid and efficient operations, crucial for real-time applications. Optimization of algorithms on a resource-constrained hardware demonstrated superiority over mainstream software-based desktop CPU solutions.

Acknowledgement

Authors would like to thank Chili.CHIPS*ba for support and partial funding of this project.

References

[1] D. Bailey, *Design for Embedded Image Processing on FPGAs*. IEEE Press, Wiley, 2011.

[2] C. Brugger, L. Dal'Aqua, J. A. Varela, C. De Schryver, M. Sadri, N. Wehn, M. Klein, and M. Siegrist, "A quantitative cross-architecture study of morphological image processing on cpus, gpus, and fpgas," in *2015 IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE)*, pp. 201–206, IEEE, 2015.

[3] M. Qasaimeh, K. Denolf, J. Lo, K. Vissers, J. Zambreno, and P. H. Jones, "Comparing energy efficiency of cpu, gpu and fpga implementations for vision kernels," in *2019 IEEE international conference on embedded software and systems (ICESS)*, pp. 1–8, IEEE, 2019.

[4] D. G. Bailey, "The advantages and limitations of high level synthesis for fpga based image processing," in *Proceedings of the 9th International Conference on Distributed Smart Cameras*, pp. 134–139, 2015.

[5] H. S. Neoh and A. Hazanchuk, "Adaptive edge detection for real-time video processing using fpgas," *Global Signal Processing*, vol. 7, no. 3, pp. 2–3, 2004.

[6] M. I. AlAli, K. M. Mhaidat, and I. A. Aljarrah, "Implementing image processing algorithms in fpga hardware," in *2013 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AEECT)*, pp. 1–5, IEEE, 2013.

[7] A. HajiRassouliha, A. J. Taberner, M. P. Nash, and P. M. Nielsen, "Suitability of recent hardware accelerators (dsps, fpgas, and gpus) for computer vision and image processing algorithms," *Signal Processing: Image Communication*, vol. 68, pp. 101–119, 2018.

[8] S. Halder, D. Bhattacharjee, M. Nasipuri, and D. K. Basu, "A fast fpga based architecture for sobel edge detection," in *Progress in VLSI Design and Test: 16th International Symposium, VDAT 2012, Shibpur, India, July 1-4, 2012. Proceedings*, pp. 300–306, Springer, 2012.