Accelerated Real-Time Local Maxima Detection in Video Streams Using FPGA Technology

Anindhita Nayazirly Sukarno*, Yahwista Salomo†, Trio Adiono, Infall Syafalni,
Nana Sutisna, and Rahmat Mulyawan
* Institut Teknologi Bandung, Indonesia
E-mail: zirlysukarno@gmail.com
† Institut Teknologi Bandung, Indonesia
E-mail: yahwistasalomo@gmail.com

***Abstract*—In this paper, we present a successful implementation of a local maxima filter on a Zybo Z7-20 and PYNQ Z1 FPGA using their two HDMI ports in real-time. The proposed system uses the HDMI ports to capture video frames with a resolution of 640x480 pixels. The local maxima filter is then applied to the captured frames in real-time, allowing for the detection of peaks in the image data. The filter uses a sliding window approach to determine the local maxima, and a threshold value is set to identify and retain only the most significant peaks in the image. The system was implemented using SystemVerilog Hardware Description Language (HDL). The system was developed in the Xilinx Vivado design suite and the results show that the proposed system is able to process video frames at a rate of 60 frames per second with high accuracy and low latency. The proposed implementation using SystemVerilog presents a more efficient and flexible solution for image processing applications on FPGA, making it a promising approach for real-time image processing.**

I. Introduction

Image processing has become an integral part of modern technological systems, with applications ranging from medical diagnostics to autonomous vehicles [1]. Real-time processing of high-resolution video streams poses significant challenges, particularly in resource-constrained environments [2].

The identification of local maxima in image frames is a fundamental operation in many computer vision tasks, including feature detection and segmentation [3]. However, performing this operation in real-time on high-resolution video streams (e.g., 640x480 at 60 fps) requires substantial computational resources [4].

Traditional software-based approaches often struggle to meet the stringent timing requirements of real-time video processing [5]. While GPU-accelerated solutions offer improved performance, they typically consume significant power and may not be suitable for embedded systems [6].

There is a need for a hardware-accelerated solution that can perform real-time local maxima detection on high-resolution video streams while maintaining low latency and power efficiency [7]. Existing FPGA-based solutions often lack the necessary throughput or flexibility to handle varying image complexities [8].

This paper presents a novel FPGA-based system for real-time local maxima detection in video streams. Our key contributions include:

- A highly parallelized architecture capable of processing 640x480 pixel frames at 60 fps [9].
- An adaptive processing algorithm that dynamically adjusts to image complexity [10].
- An efficient memory management scheme optimized for row-wise image processing [11].
- A pipelined design that maximizes hardware utilization and minimizes latency [12].
- Comprehensive performance analysis and comparison with existing solutions [13].

Our proposed system demonstrates significant improvements in processing speed and efficiency, paving the way for advanced real-time image processing in resource-constrained environments [14].

II. Local Maxima Algorithm

The process of identifying 3x3 local maxima in an image involves a siterative procedure that examines each pixel in relation to its surrounding neighbors. This algorithm, which shares similarities with techniques described by Gonzalez and Woods [1] in their seminal work on digital image processing, can be delineated as follows:

- Pixel Iteration: The algorithm systematically traverses each pixel within the image matrix, constructing a 3x3 window centered on the current pixel under evaluation. This approach is reminiscent of convolution operations used in various image processing tasks [3].
- Comparative Analysis of Neighboring Pixels: For each target pixel, a rigorous comparison is conducted against its eight adjacent pixels within the defined 3x3 window. A pixel is designated as a local maximum if and only if its value surpasses those of all its neighbors. In scenarios where the target pixel's value is equivalent to one or more of its neighbors, a more nuanced approach is adopted: the target pixel's status as a local maximum is determined by the output status of its neighboring pixels, introducing a level of contextual decision-making [15].

- Image Analysis: The algorithm rigapplies this evaluation process to every pixel within the image bounds, ensuring a thorough analysis of the entire visual field. This approach is critical for maintaining the integrity of the local maxima detection across the image [10].
- Output Generation and Representation: The culmination of this process yields a set of coordinates corresponding to all identified local maxima. This output can be represented in various forms, such as a binary image or a list of coordinate pairs, depending on the specific requirements of subsequent processing stages [6].

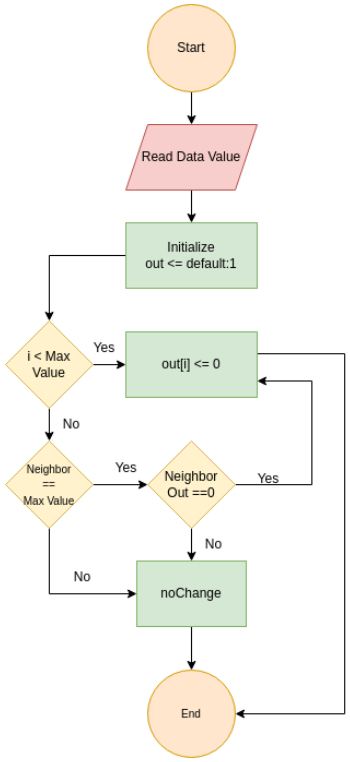

Fig. 1. Schematic representation of the local maxima algorithm (single iteration)

Fig. 10 elucidates the intricate timing management within the process block. The counter initiates its sequence upon the deactivation of the reset signal. The write_address and read_address signals are strategically delayed in relation to the counter, implementing a form of pipeline processing to optimize throughput. In this context, the notation 'd' denotes raw data input, 'o' represents the intermediate output values awaiting processing, and 'n' signifies the resultant values destined for storage in bram_out_curr.

A. Mathematical Representation of the algorithm

Below is the iterative algorithm for local maxima detection. We can represent this algorithm mathematically as follows: Let I be the input image of size $M \times N$, and O be the output image of the same size. Initialize O with all elements set to 1:

$$O_{i,j} = 1 \quad \forall i \in [1, M], j \in [1, N] \tag{1}$$

Define the neighborhood $\mathcal{N}(i,j)$ of pixel (i,j) as:

$$\mathcal{N}(i,j) = (i',j') : |i - i'| \leq 1, |j - j'| \leq 1, (i',j') \neq (i,j) \tag{2}$$

The algorithm then iteratively updates O until convergence: For each pixel (i,j):

Find the maximum value in the neighborhood:

$$\max_{local} = \max_{(i',j') \in \mathcal{N}(i,j)} I_{i',j'} \tag{3}$$

If $I_{i,j} < \max_{local}$:

$$O_{i,j} = 0 \tag{4}$$

If $I_{i,j} = \max_{local}$:

$$O_{i,j} = \min_{(i',j') \in \mathcal{N}(i,j), I_{i',j'} = I_{i,j}} O_{i',j'} \tag{5}$$

The algorithm continues until O no longer changes between iterations.

III. Proposed Design

A. System Architecture and Design

1) Image Processing Architecture: Both the input and output frames adhere to a 640x480 pixel resolution with a 24 bits per pixel (bpp) color depth, consistent with standard VGA specifications [4]. However, the system introduces an intermediate step where the received frame undergoes grayscale conversion to 6 bpp. This reduction in color depth not only optimizes memory usage but also simplifies subsequent processing steps, a common technique in image pre-processing [5]. Following the local maxima computation, the output is remapped to the full 24 bpp color space for transmission, ensuring compatibility with standard display devices. The system's memory architecture is built around three specialized block RAMs, each serving a distinct purpose in the processing pipeline: Input Array: Stores the incoming frame data from the DVI decoder. Output Array: Houses the intermediate results of the local maxima algorithm. Transmission Buffer: Contains a duplicate of the final output array, ready for transmission. This tripartite memory structure facilitates efficient data flow and minimizes latency, a crucial factor in real-time image processing systems [5].

2) Processing Unit: The processing flow incorporates a conditional branching mechanism: upon completion of the looping process, the final processed data is committed to a designated BRAM for subsequent transmission. However, if the process remains incomplete, the intermediate output is written to a buffer RAM and reprocessed. This iterative approach ensures the algorithm's convergence, a critical aspect in handling complex image structures [1]. The loop count increments with each iteration, with data read out sequentially until the process concludes. As illustrated in Fig. 3, each loop cycle necessitates (BRAM_Depth + 6) clock cycles to conclusively verify the finished signal, a design choice that balances processing thoroughness with system latency [2].

Fig. 4 provides an in-depth view of the processor's architectural scheme. The design incorporates sophisticated shift

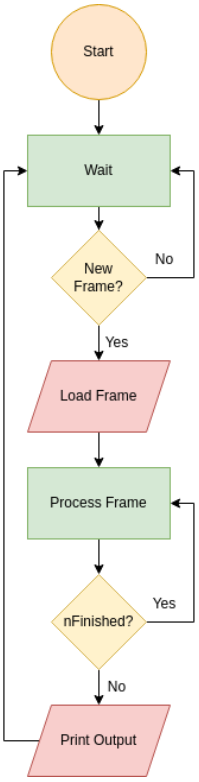

Fig. 2. Flowchart depicting the data processing scheme

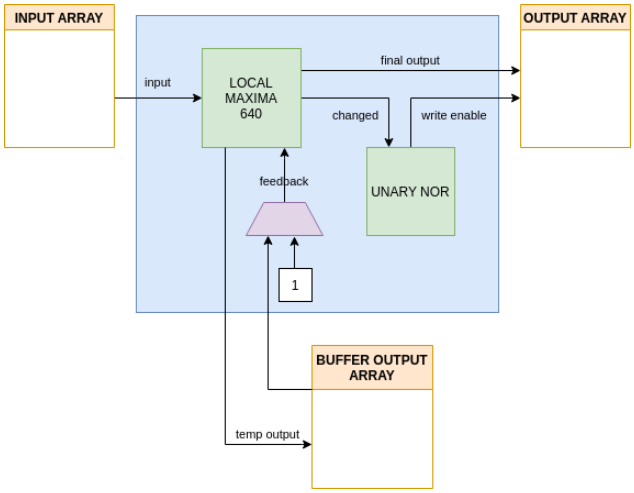

Fig. 3. Detailed block diagram of the advanced processor architecture

registers: at system initialization, out_buf_0 and out_buf_1 are preset to one, while data_buf_0 and data_buf_1 are initialized to zero. This initialization strategy ensures proper boundary handling, a crucial aspect in image processing algorithms [15]. In each clock cycle, the data in the shifters undergoes an upward shift. The local maxima combinational algorithm receives input from both the out_buf and data_buf shifters, with results subsequently buffered in the out_bram_curr memory. A notable feature of each of the 640 local_maxima combinational blocks is the inclusion of a specialized output port. This port is designed to detect variations from the previous value, triggering a reprocessing of the frame if changes are detected.

Fig. 5 offers a detailed view of the combinational logic within each local maxima processing block. Each module is designed to process 9 bits of image data from the BRAM alongside 9 bits of prior output data. The module then determines the output value, denoted as value_o. The 'changed' signal plays a crucial role in the system's adaptive behavior, determining whether additional processing iterations are necessary.

3) DVI Receiver: The DVI receiver subsystem is responsible for capturing and preprocessing the incoming video stream. Fig. 6 illustrates its complex architecture. The receiver integrates several key components:

4) DVI Transmitter: The DVI transmitter subsystem, depicted in Fig. 7, is responsible for converting the processed image data back into a format suitable for DVI transmission.

The DVI transmitter operates across multiple clock domains to meet the stringent timing requirements of the DVI standard:

A 25 MHz clock serves as the pixel clock, facilitating the output of 640x480 pixels at 60 frames per second, adhering to standard VGA timing specifications [4].

B. Performance Considerations and System Optimization

1) Pipelining and Parallelism: The system leverages both pipelining and parallelism to achieve high throughput. The 640 parallel "Local Maxima" combinational blocks represent a form of spatial parallelism, allowing the system to process an entire row of pixels simultaneously. This approach significantly reduces the time required to process each frame [12].

2) Memory Management: The use of three separate block RAMs (input array, output array, and transmission buffer) is a key feature that enables efficient data management. This approach minimizes memory conflicts and allows for simultaneous read and write operations, a technique that Ienne and Leupers [8] highlight as crucial for high-performance embedded systems. The row-based memory organization, with a depth of 480 and varying widths, is optimized for the specific requirements of image processing. This design choice facilitates burst reads and writes, reducing memory access latency and improving overall system performance [11].

3) Adaptive Processing: The inclusion of a 'changed' signal in the local maxima blocks, triggering reprocessing when necessary, represents an adaptive approach to image processing. This feature ensures that the system can handle varying levels of image complexity, dynamically adjusting its processing to maintain output quality. Such adaptive techniques are increasingly important in real-world image processing applications, as noted by Bovik [5].

IV. Hardware Implementation

As can be seen in Fig. 8 the system was implemented on Pynq Z1 (Digilent). The DVI source used for testing was Raspberry Pi 4 (Adafruit). The Raspberry Pi was used to play videos and stream camera. The DVI sink used for testing was

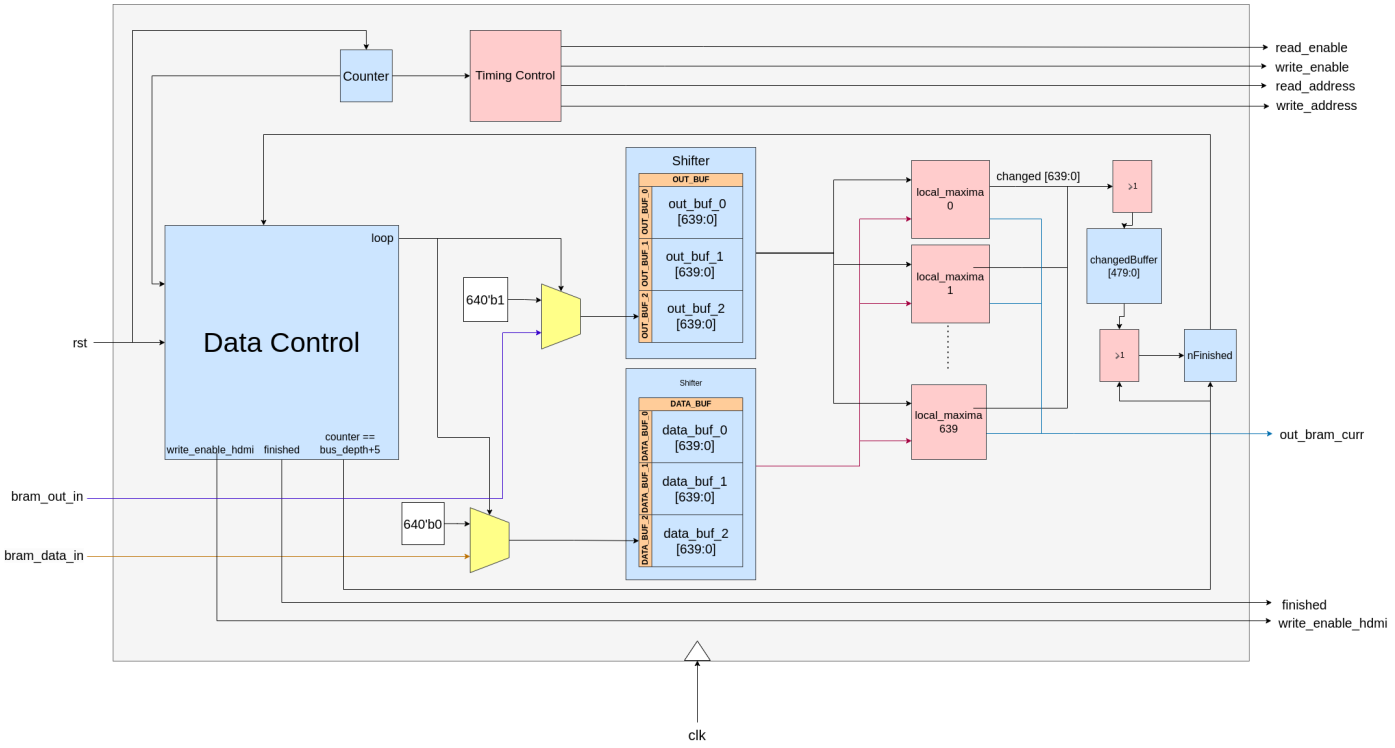

Fig. 4. Complete block diagram of the processor's internal structure

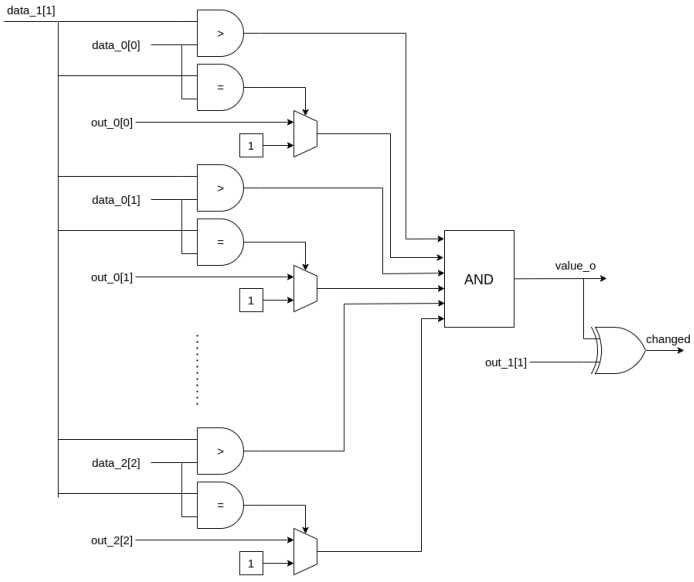

Fig. 5. Detailed combinational logic diagram of the local maxima processing unit

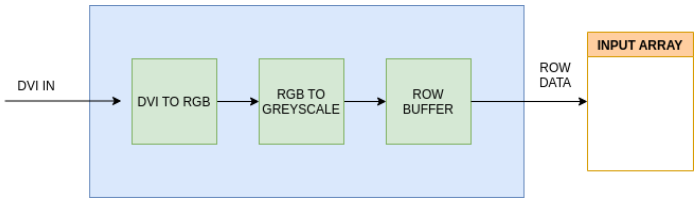

Fig. 6. Detailed block diagram of the advanced DVI receiver subsystem

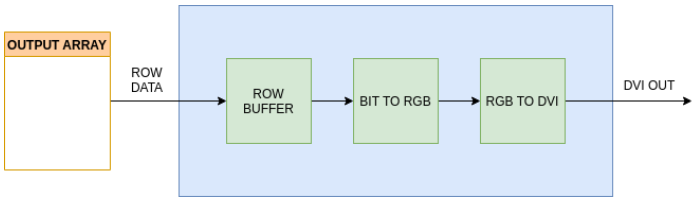

Fig. 7. Comprehensive block diagram of the advanced DVI transmitter subsystem

a generic display monitor. The Raspberry Pi unfiltered display was also mirrored to another display monitor for comparison. Fig. 9 shows a display of the two monitors. Video of the demo can be accessed at https://youtu.be/rrs4vBUoiV0.

Fig. 10 shows the expected behaviour of a single frame processed once in the loop. Note, Xilinx's bram requires two clock cycle to be registered.

V. Results and Discussion

A. Latency

The overall latency of the system is fixed as the time distance between two frames which are 16.67 milliseconds. The operating frequency of the datapath used is 25Mhz, which in its worst case scenario would require

$$Latency = 40ns * (BITWIDTH) * (BITDEPTH + 6)$$

or 12.4416 milliseconds to finish an operation.

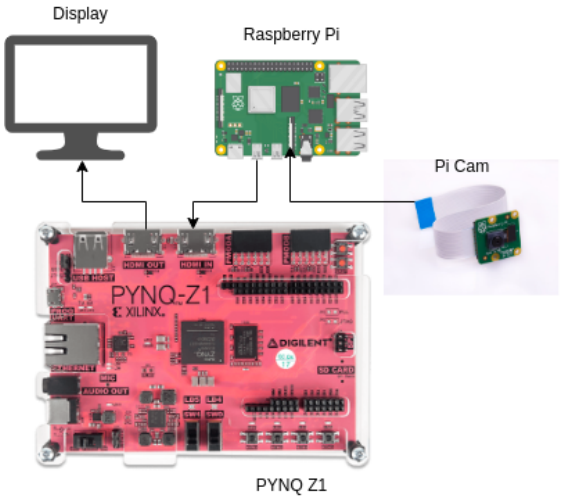

Fig. 8. Test setup

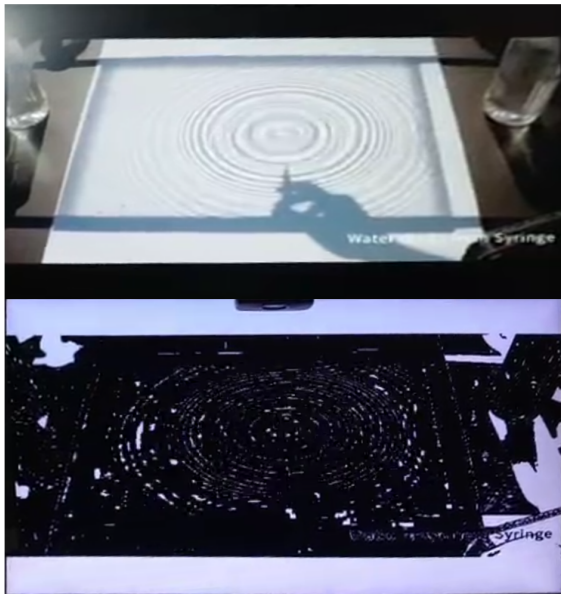

Fig. 9. Example of system input (top picture) and output (bottom picture)

TABLE I
RESOURCE UTILIZATION

Resource	Utilization
LUT	30581
LUTRAM	785
FF	24772
BRAM	88.50
IO	18
BUFG	6
MMCM	2

Note: LUT: Look-Up Table; FF: Flip-Flop;
BRAM: Block RAM; IO: Input/Output; BUFG: Global Clock Buffer;
MMCM: Mixed-Mode Clock Manager

Strictly speaking of the datapath delay path, ignoring other peripheral such as BRAM, ILA, or VIO, the critical path of the datapath of local maxima with 640 width, 480 depth, and 6 data width is 10.632ns. Which means the maximum internal operating frequency that can be achieved is around 94Mhz. with the worst possible scenario, could finish the operation in 3.3069 milliseconds and process 300 frames per second.

B. Scalability

The local maxima processing block are designed with scalability in mind, the relating hdl codes uses three parameter as its inputs, which are **DATA_WIDTH** to specify how many parallel data you want to be processed each clock, **DATA_DEPTH** to specify how many clock cycles per loop you want, and **BIT_WIDTH** to specify the number of bits used as the comparator.

C. Comparison of Equivalent Implementations

To contextualize the performance of our FPGA-based local maxima detection system, it's valuable to compare it with equivalent implementations on different platforms. This comparison highlights the advantages and trade-offs of various approaches.

1) CPU-based Implementation: Traditional CPU-based implementations, such as those using MATLAB or OpenCV, offer flexibility but often struggle with real-time performance for high-resolution video streams. For instance, Neubeck and Van Gool [16] reported processing times of 13.5 ms for a 512x512 image using an optimized algorithm on a 2.8 GHz Pentium 4. While this approaches real-time performance, it doesn't match our FPGA implementation's capability to process 640x480 frames at 60 fps (16.67 ms per frame).

VI. CONCLUSION AND FUTURE WORK/

The proposed system represents an approach to real-time local maxima detection in video streams. By leveraging advanced FPGA capabilities, including parallel processing, pipelined architectures, and efficient memory management, the system achieves high-performance image processing while maintaining the flexibility necessary for future enhancements. The design's modular architecture, with clearly defined input, processing, and output stages, provides a solid foundation for future research and development. As the field of real-time image processing continues to evolve, systems like this will play an increasingly important role in applications ranging from computer vision and robotics to medical imaging and autonomous vehicles.

Concluding, this paper has presented a successful design implementation on the Zybo Z7-20 and PYNQ Z1 using the HDMI output of Raspberry Pi. The proposed local maxima filter is a parallel 640 data blocks with a 10.632ns critical path, and is capable of much higher throughput than is currently achieved. Further works may include using it as a red blood cell counter or other image processing applications that require local maxima. With these improvements, the proposed design

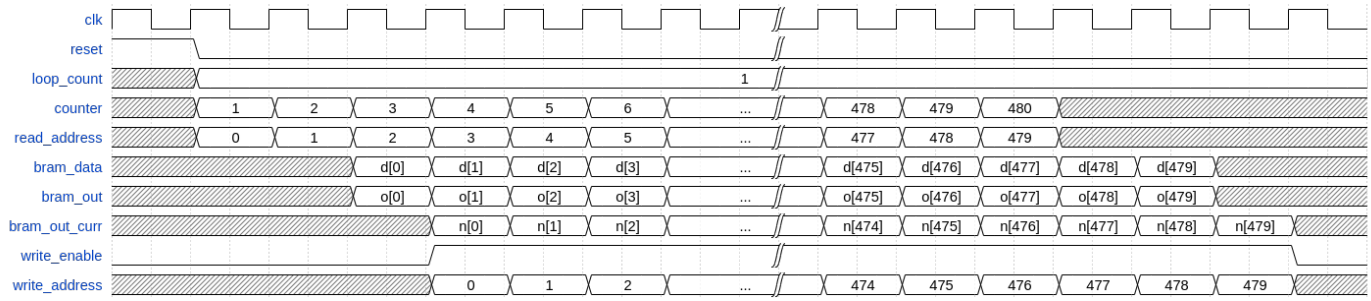

Fig. 10. Local maxima processor timing diagram for one loop

could provide a platform for more complex image processing tasks and applications.

While our current implementation demonstrates promising results in controlled environments using a Raspberry Pi as the video source, we acknowledge the need for more extensive testing in real-world conditions. Future work will focus on evaluating the system's performance across a wider range of scenarios, including varying lighting conditions, dynamic scenes, and different video resolutions. Additionally, we plan to conduct rigorous tests on the system's robustness in handling noisy or degraded video streams. This will involve introducing artificial noise, compression artifacts, and other forms of signal degradation to assess the algorithm's resilience. We also aim to explore adaptive thresholding techniques to enhance the system's ability to maintain accurate local maxima detection in suboptimal video conditions. These investigations will provide valuable insights into the system's applicability in diverse real-world applications, such as surveillance, autonomous vehicles, and medical imaging, where video quality can vary significantly.

References

[1] R. C. Gonzalez and R. E. Woods, *Digital image processing*. Pearson, 2018.

[2] N. Kehtarnavaz, *Real-Time Digital Signal Processing: Based on the TMS320C6000*. Newnes, 2015.

[3] R. Szeliski, *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[4] K. Jack, *Video demystified: a handbook for the digital engineer*. Newnes, 2007.

[5] A. C. Bovik, *Handbook of Image and Video Processing*, 2nd. Academic Press, 2010.

[6] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*. O'Reilly Media, Inc., 2008.

[7] D. G. Bailey, *Design for embedded image processing on FPGAs*. John Wiley & Sons, 2011.

[8] P. Ienne and R. Leupers, *Customizable Embedded Processors: Design Technologies and Applications*. Morgan Kaufmann, 2006.

[9] K. Asanovic, R. Bodik, B. C. Catanzaro, *et al.*, "The landscape of parallel computing research: A view from berkeley," *Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley*, 2006.

[10] M. Sonka, V. Hlavac, and R. Boyle, *Image processing, analysis, and machine vision*. Cengage Learning, 2014.

[11] S. Hauck and A. DeHon, *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation*. Morgan Kaufmann, 2007.

[12] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 6th. Morgan Kaufmann, 2019.

[13] L. Wanhammar, *DSP Integrated Circuits*. Academic Press, 1999.

[14] P. Pirsch, *Architectures for digital signal processing*. John Wiley & Sons, 1995.

[15] R. Jain, R. Kasturi, and B. G. Schunck, *Machine vision*. McGraw-Hill New York, 1995.

[16] A. Neubeck and L. Van Gool, "Efficient non-maximum suppression," in *18th International Conference on Pattern Recognition (ICPR'06)*, IEEE, vol. 3, 2006, pp. 850–855.