Scientific Visualization, 2024, volume 16, number 4, pages 102 - 108, DOI: 10.26583/sv.16.4.09

Development and Application of Algorithms of Video Stream Filtering and Processing on Programmable Logic Integrated Circuits FPGA

Yu.A. Salkov[1,B], O.N. Tretiyakova[2,A], D. N. Tuzhilin[3,B]

[A] Moscow Aviation Institute (National Research University) MAI
[B] LLC "PROMIS LAB" of the group of companies "Lasers and Equipment TM"

[1] ORCID: 0009-0002-4928-3171, salkov@laser-app.ru
[2] ORCID: 0000-0003-0256-4558, tretiyakova_olga@mail.ru
[3] ORCID: 0000-0002-8570-1732, tuzhilin@laserapr.ru

Abstract

The paper is devoted to the details of implementation of machine vision algorithms on FPGA. The algorithms were implemented using the Vitis HLS high-level synthesis tool. The main parameter in the realization of the algorithms is speed; since it is necessary to proceed processing of the incoming video stream with minimum delay.

Keywords: system-on-chip, ZYNQ, machine vision techniques, video processing, FPGA image filtering, instrumentation technology, moving average, mass center, HLS.

1. Introduction

In the process of manufacturing of semiconductor parts, it is necessary to track the focus the laser beam accurately on surface of wafer. To adjust the position of laser focus directly during the wafer laser process, an optical tracking system was developed [1] (Figure 1), which consists of a laser diode, a linear camera, and a controller. The laser diode beam shines at an angle onto the surface of the workpiece and is reflected from it. Reflected beam of the laser diode returns to the linear camera, thus forming a peak with its position on the camera sensor.

The frame from the camera transfers to the controller of the optical tracking system, where received frame is processed by moving average filter, after which the filtered frame is processed to find a peak position according to the specified parameters, by the formula for finding the center of mass.

Since the focus adjustment of the laser is performed directly during the laser processing process, the optical tracking system (OTS) controller needs to quickly receive and process the incoming video stream, so all machine vision algorithms and the servo motor control algorithm were implemented on FPGA. Implementing algorithms on FPGA allows to achieve higher performance compared to conventional computing architectures (such as x86, ARM or RISC-V). Thus, it becomes highly specialized architecture for specific tasks because algorithms are implemented on FPGA.

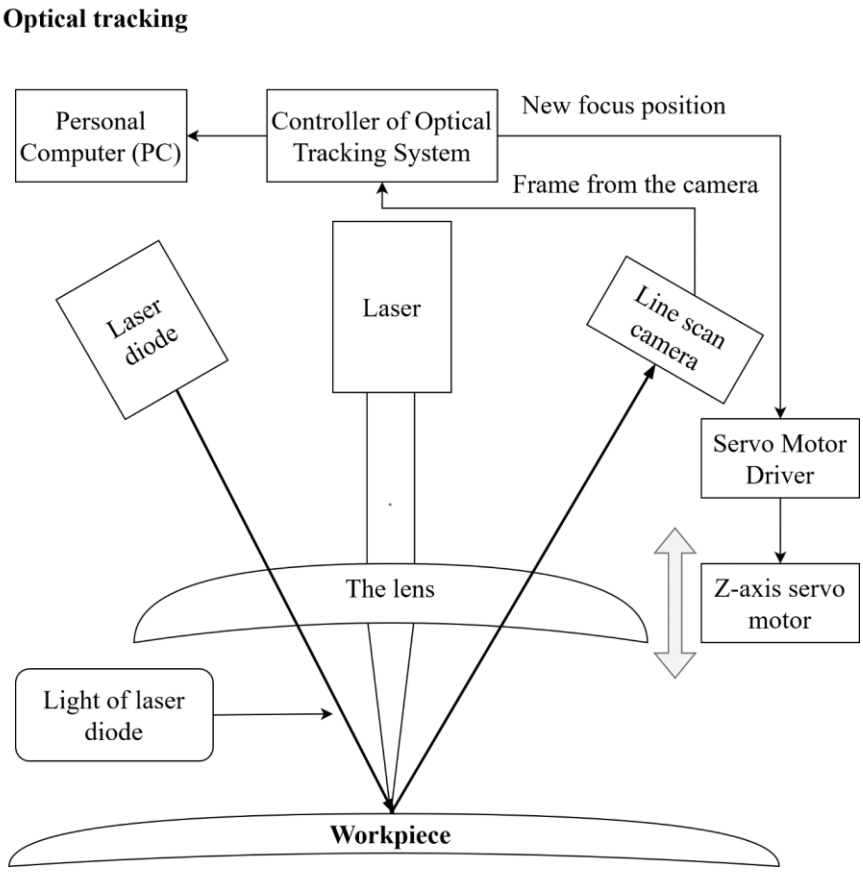

Figure 1: Schematic diagram of the optical setup of tracking system.

2. Hardware configuration and development tools

A system-on-chip (SOC) from Xilinx, the AMD Zynq 7000 SoC ZC702 debug board, was taken to implement the OTS controller. This board is redundant for SOC controller realization. Meantime it has all necessary interfaces and allows developing the controller without limiting the number of logic gates.

To register the position of the reflected laser beam, a linear monochrome camera transmitting video via camera link protocol [2] was taken. To receive the video stream, a special interface board for FPGA with a camera link connector was purchased.

The vitis-HLS high-level synthesis tool [3] was used in the development of the OTS controller. High-level synthesis tools allow developing individual functional units by writing code in high-level languages (C++, Python, etc.) rather than in hardware description languages (Verilog, HDL, etc.). These tools allow the functional module developer to abstract from the register transfer level (RTL) and develop code at the data handling level.

Vitis-HLS has an adapted Open-CV library, so the development of functional blocks related to video processing becomes much easier due to the availability of ready and adapted solutions. In this development the open-CV functions were not used due to the features of the OTS controller operation.

3. Development of filtering algorithm

As mentioned above, the OTS controller must quickly receive and process the incoming video stream. Before searching for a peak on the frame, the frame should be filtered. A modified moving average algorithm [4] was developed to filter the frame. This algorithm was chosen due to the fact that the peak on the frame is clearly expressed and it is necessary to filter out unnecessary noise to correctly determine peak position.

The filter input is an image of 2*2048 pixels resolution, the frame is monochrome, each pixel is encoded with 8 bits. The first line of the frame is written to the frame buffer without

changes. From the beginning of the arrival of the second frame line, filtering starts using a sliding window, the width of the window is set by the user, and the height of the window is fixed - 2 pixels. From the beginning of filtering the window is filled with zeros, because of this some information at the beginning of the image is lost, which is not critical for this project. The filtered second line of the image is written to the frame buffer, thus in one frame there are both "raw" data and filtered data, due to this it is possible to display both graphs at once in the visualizing program. When visualizing data in the form of a graph, the X-axis shows the index of the pixel in the frame and the Y-axis shows the brightness of the pixel. Figure 2 shows the algorithm of moving average filter operation.

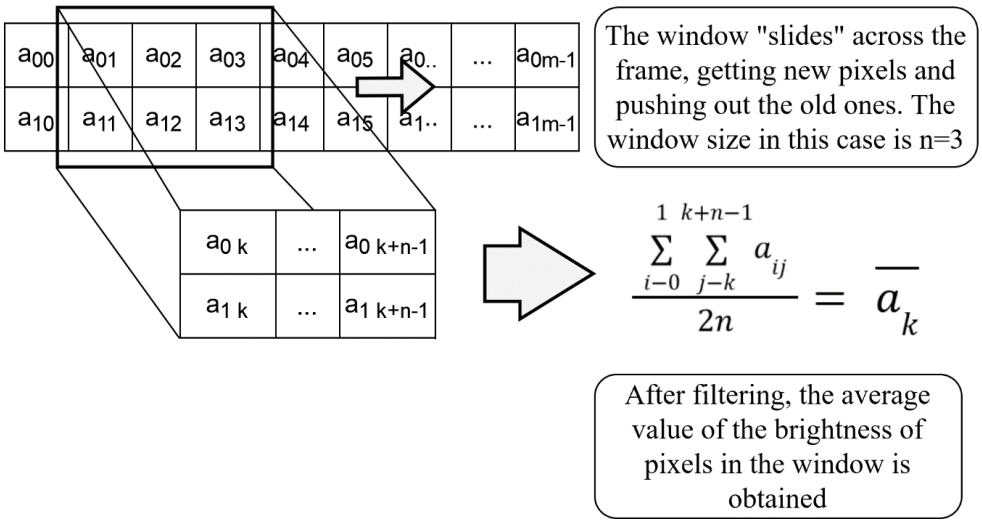

Figure 2: Filtering algorithm and formula.

Streamed video comes to the filter, two pixels are transmitted at each clock pulse, thus, when the second line of the image is received and the filtering of the frame begins, two pixels from each line are written to the window at once.

Due to the fact that this memory is a two-port memory, it is impossible to perform read or write operations with more than two memory cells at one moment of time. This imposes serious limitations on the work with memory, because in a clock comes data about two pixels of the frame at once, and if it is already the second line of the frame, then in the same clock it is necessary to read data from the buffer, to put them in the averaging window.

The vitis vision library class "xf::cv::LineBuffer" is used to create and work with memory. This class allows us to fine-tune the size and type of memory of the frame buffer. When creating the buffer, it needs to specify the size of elements to be stored in it. To meet the limitation on the number of one-time operations on the buffer it was decided to store two pixels in one buffer cell at a time. Due to this solution, only two operations on the frame buffer are performed at one time: the operation of writing new pixels and the operation of reading pixels of the first line of the frame.

When filtering a frame, two pixels from both frame lines are written into the filter window at once, "displacing" old pixels. Window averaging algorithm performs reading all cells of the window, summed up of values and obtain average calculated. Thus, usage of conventional block two-port memory for the averaging algorithm is impossible, because it will spend too much time to obtain data from all cells of the window. The averaging window is a set of shift registers. Shift registers allows obtaining data from all memory cells in a single clock cycle. It allows the operation of shifting the data in any direction. Amount of registers in FPGA are much smaller than block memory, so the realization of the frame buffer by means of registers does not make much sense.

The shift register window is implemented using the "xf::cv::Window" class, which is also part of the vitis vision library. When a window is created, its height and width are set, as well as the size of the data stored in the window.

The overall filtering algorithm is shown in Figure 3, in the figure red color indicates read operation from frame buffer and blue color indicates write operations.

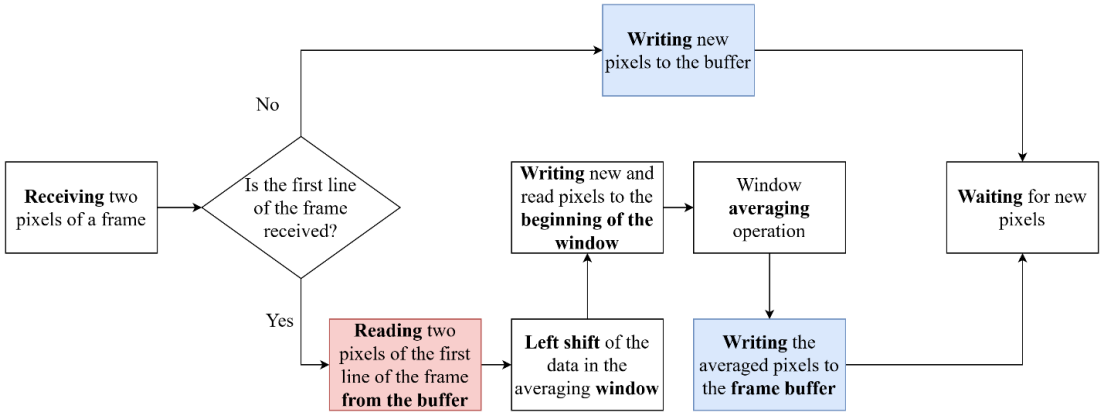

Figure 3. Algorithm of frame filtering.

During configuring the system, it is important to be able to set different sizes of the smoothing window. It allows fine-tuning of the controller. But when implementing any algorithms on FPGA it is impossible to change the size of already defined arrays, this is due to the fact that during synthesis and placement and routing all necessary resources are reserved and there is no possibility to add or release them in the future [5]. Therefore, to realize variable window size, it was decided to initially create a window of the maximum size to be able to vary the window size from 0 (no filtering occurs) to the maximum size.

During window averaging operation, iteration over the window is performed using loop “for”, vitis HLS has a special pragma for "unroll loops" (pragma HLS unroll, i.e. performing an operation in a loop over all iterated objects at once). This pragma allows us to perform the entire cycle in one clock cycle, which significantly speed up data processing. To "unroll" the loop, a constant number of iterations must be specified in the loop declaration, otherwise the pragma will not apply to the loop. Figure 4 shows an example of the unroll pragma, where maxWindowSize is a constant variable and windowSize can be changed by the user at runtime. It is possible to create loops with a variable number of iterations by accepting if(i < ...) in the loop body, which will be executed in one clock cycle.

```
int avg = 0;

for(int i = 0; i < windowSize; ++i)
{
    avg += window[i];
}

avg /= windowSize;
```

```
int avg = 0;

for(int i = 0; i < maxWindowSize; ++i)
{
    #pragma HLS unroll
    if(i < windowSize) avg += window[i];
}

avg /= windowSize;
```

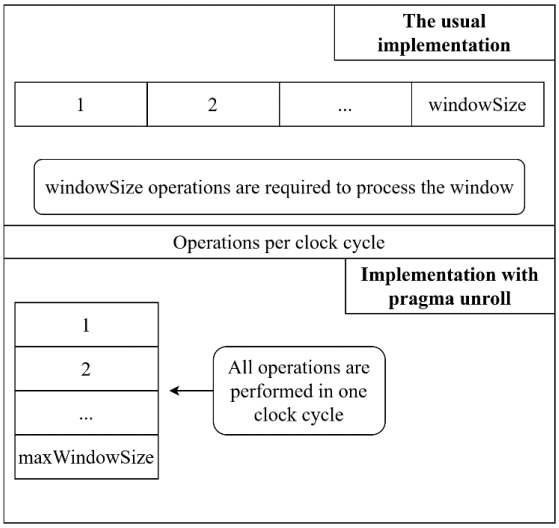

Figure 4: Description of unroll pragma operation.

As a result, we managed to develop an optimized rapid algorithm for filtering the incoming video stream. The filter outputs the result as soon as a full frame is received, thus the speed of data processing is mostly limited by the speed of video stream transmission by the camera.

4. Development of an algorithm for peak search on the frame

In order to maintain the controller performance, it was decided to implement the peak search algorithm together with the video stream filtering algorithm. Thus, two algorithms working one after the other are realized in one functional block. After the window filtering operation, the peak data is updated, and immediately after receiving the whole frame the new peak position is calculated. The delay between receiving the full frame and calculating the new peak position is the minimum as possible.

The algorithm to search center mass of peak itself is a center mass search formula, where the brightness of a particular pixel (m_i) was taken instead of the mass, and the pixel index (r_i) was taken instead of the point coordinates. The algorithm for finding the center mass is presented in Figure 5.

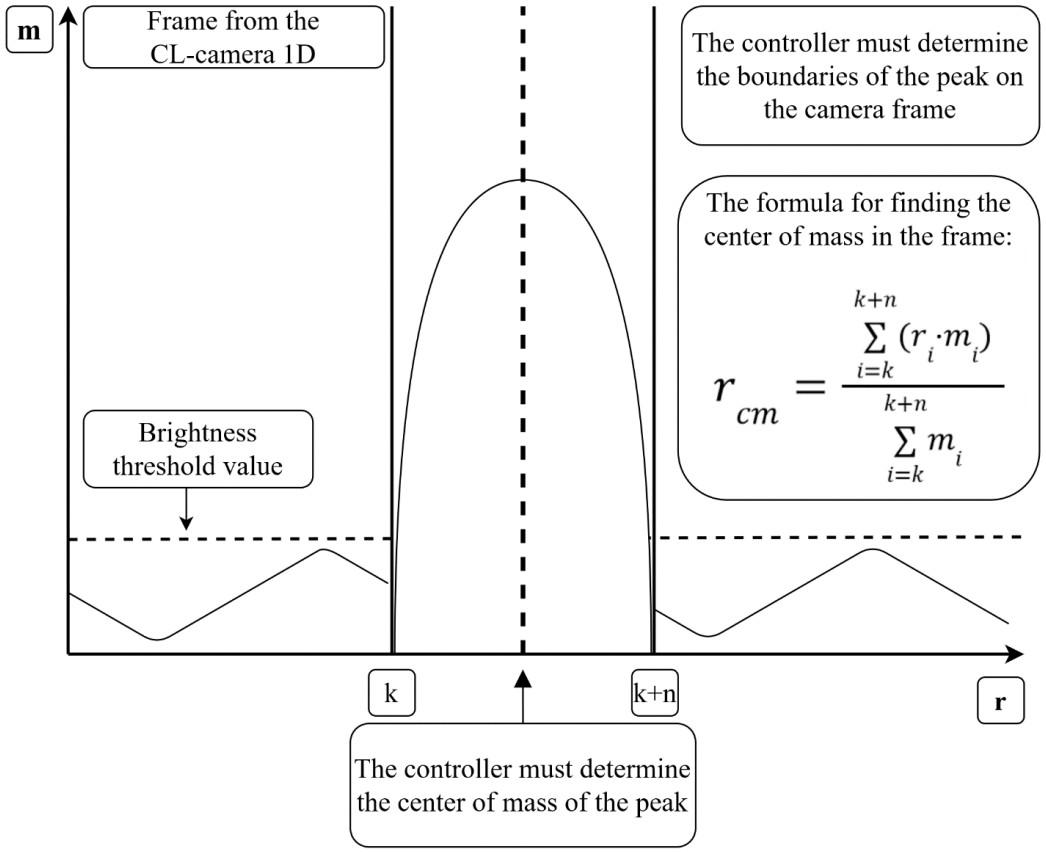

Figure 5: Algorithm for finding the mass center.

Peak search is performed on all non-zero filtered pixels of the frame. For peak search the user sets parameters such as: maximum and minimum peak width, peak pixel brightness threshold, maximum peak brightness, etc. Immediately after obtaining the average value from the averaging window, the data is updated. If the brightness of a pixel is zero, but the pixels before it was not zero, the calculation of the peak center masses is performed and the obtained peak data is checked with the set parameters. The first peak satisfying the set parameters is considered suitable and the algorithm finishes its work. The output of the functional block receives data on the peak position and size.

When implementing the peak search algorithm, it is necessary to compare the current peak value with the set parameters, this is done using the if operator with a large number of comparisons (i.e. if (a < b && b == c && ...)). Using the logical AND (&& in C++) in one if operator in a large number leads to a long critical path, it is due to the fact that in C++ when using the && operator it is guaranteed that the condition to the left of the operator will be checked first and if it is not true, the second condition will not be checked. When the critical path increases, the speed of data processing decreases greatly due to the reduced frequency of the whole controller. In this connection, all operators of logical AND have been replaced by bitwise AND (&) operators, while calculating bitwise AND allows to check all conditions at once and only after calculating the result of bitwise AND.

As a result of the peak search algorithm development, it was possible to implement an optimized and high-performance algorithm that produces results almost immediately after a full frame is acquired. As with the implementation of the filtering algorithm, the performance of the algorithm is also limited by the speed of data transmission from the camera.

5. Conclusion

As a result of algorithms realization, a functional block of filtering and peak search on video stream was developed. This block processes the incoming frame with a minimum time of 42 μs, this time includes the time of receiving the frame. The function block has flexible settings for filtering and peak search on the frame.

Acknowledgments

The authors express their gratitude to the management of the "Lasers and Equipment TM" group of companies for their assistance in providing material and technical support for conducting experimental research and modeling the considered process.

References

1. Kondratenko, V.S.; Saprykin, D.L.; Tretyakova, O.N.; Tuzhilin, D.N. Development of the automatic control system of the focus adjustment for the technology of laser micromachining of materials (in Russian) // Instrumentation. 2022. № 4. C.26-31.

2. Camera Link standard [Electronic resource] // Imagelabs URL: https://www.imagelabs.com/wp-content/uploads/2010/10/CameraLink5.pdf (date of address: 07.07.2024)

3. Ryan Kastner, Janarbek Matai, and Stephen Neuendorffer. Parallel Programming for FPGAs [Electronic resource] // Kastner Research Group URL: https://kastner.ucsd.edu/wp-ontent/uploads/2018/03/admin/pp4fpgas.pdf (date of address: 07.07.2024)

4. Orlova Irina Anatolievna, Ustyukov Dmitry Igorevich, Efimov Alexey Igorevich Features of realization of spatial image filters on FPGA // Izvestiya TulSU. Technical Sciences. 2018. №2. 195-206.

5. Digital synthesis: a practical course // Antonov A. A., Barabanov A. V., et al. V., Barabanov A., et al. / Edited by A. Y. Romanov, Y. V. Panchul. - Moscow: DMK Press, 2020. 111.