DOI: https://doi.org/10.15276/aait.05.2022.16
UDC 004.383

Image buffering in application specific processors

Anatoliy M. Sergiyenko[1)]
ORCID: http://orcid.org/0000-0001-5965-1789; asergy@bigmir.net. Scopus Author ID: 27868137900
Vitaliy O. Romankevich[1)]
ORCID: http://orcid.org/0000-0003-4696-5935; romankev@scs.kpi.ua. Scopus Author ID: 57193263058
Pavlo A. Serhiienko[1)]
ORCID: http://orcid.org/0000-0003-3030-0074; paulsrgnk002@gmail.com. Scopus Author ID: 57204497516
[1)] National Technical University of Ukraine “Igor Sikorsky KPI”. 37, Peremohy Ave. Kyiv, 03056, Ukraine

ABSTRACT

In many digital image-processing applications, which are implemented in field programmable gate arrays, the currently processed image's frames are stored in external dynamic memory. The performance of such an application depends on the dynamic memory speed and the necessary requests quantity during algorithm’s runtime. This performance is being optimized through field programmable gate arrays - implemented buffer memory usage. But there is no common method for the formal buffer memory synthesis with preset throughput, input and output data sequence order and minimized hardware costs. In this article, the features of image input and processing based on Field Programmable Gate Array are considered. The methods of building buffer circuits in field programmable gate arrays, due to which the intensity of data exchanges with external memory is reduced, are analyzed. The method of synthesizing pipeline circuits with specified performance characteristics and the data sequence order is given, which is based on the mapping of the spatial synchronous data flows into the structure implemented in the field programmable gate arrays. A method of designing buffer schemes is proposed, which is based on the mapping of spatial synchronous data flows into local memory in the form of chains of pipeline registers. The method helps to organize the data flow of at the input of built-in pipeline units of image processing, in which the data follow in a given order, and to minimize the amount of buffer memory. The method ensures the use of dynamically adjustable register delays built into the field programmable gate arrays, which increases the efficiency of buffering. This method was tested during the development of an intelligent video camera. The embedded hardware implements a video image compression algorithm with a wide dynamic range according to the Retinex algorithm. The same time it selects characteristic points in the image for the further pattern recognition. At the same time, multiple decimation of the frame is performed. Due to the multirate buffering of the image in the field programmable gate arrays, it was possible to avoid using of external dynamic memory.

Keywords: Field programmable gate array; spatial synchronous data flows; image processing; buffer memory

***For citation*: Sergiyenko A. M., Romankevich V. O., Serhiienko P. A. Image buffering in application specific processors. *Applied Aspects of Information Technology.* 2022; Vol.5 No.3: 228–239. DOI: https://doi.org/10.15276/aait.05.2022.16**

INTRODUCTION

Advances in FPGA (Field Programmable Gate Array) technology have made them a platform for implementing various computer vision algorithms [1], [2]. The majority of the algorithms are impossible to be operated in real time on the general-purpose processors and the graphics accelerators application is too expensive for it. Therefore, these algorithms are best suited for hardware implementation [3]. For example, there is a significant volume increase of security cameras-generated data happened. In many systems with a limited communication throughput exists a need in remoted image processing in which perform the image collection, processing and intelligent cameras [4]. They have embedded FPGAs telecommunications [5]. Since security demands are increasing, the processing complexity will only increase. Also the real time image processing algorithms, such as those used in high-speed vehicle control [6] require the hardware with high bandwidth and low latency. That demands are also satisfied with the use of FPGA. Therefore, a deep neural network, applied in FPGA, is able of data video stream processing with latency of 31,85ms [7].

The last generation FPGAs represent an attractive alternative for image processing acceleration, as FPGAs contain ARM processors and programmable logic for computationally intensive operations acceleration. However, transferred twice

FPGA have no more than ten MB of built-in memory. Because of these limited capabilities of on-board memory, the external DDR-RAM is used for image frames [8]. This leads to high memory accessing intensively, which has potentially high latency, and, as a result, to performance deterioration or high hardware costs due to the access paralleling to multiple memory chips and the power consumption increase, because DDR is highly energy consuming. This prevents the FPGA application, especially in large frames processing with algorithms, which have intensive communication with memory.

As shown in the above examples, for image processing tasks it is common to process video streams at high clock rates and preferably with minimized latency. For this purpose, on the one hand, in FPGA should be used pipeline equipment that is able to work with the input flows of image pixels, and on the other hand, the buffering of intermediate images cannot be excessive. Such a hardware architecture can start image processing in the very moment when the required pixels set is accumulated and keep processing in a pipelined manner, providing both high throughput and low latency.

In this article examines the organization of memory access during pipeline image processing in order to minimize the use of external FPGA resources, which is a prerequisite for minimizing the latent delay. The approach to the buffer memory organization in FPGA is proposed, which ensures the data flows in the required order between hardware functional units that are configured in FPGA. This approach, of course, does not exclude the external memory access, but it is applicable to any kind of internal memory in FPGA.

LITERATURE REVIEW

Image processing algorithms, as usual, consist of several sequentially performed functions. Each of these functions reads the required pixels quantity from the image frame's memory unit, processes them and records the result pixel into another area of this memory or the other memory block. Since the image frame occupies relatively large volume, the hierarchical memory is used in image processing systems on FPGA. The bottom level of this memory consists of pipeline registers, middle level – buffer memory blocks sized of a few kilobytes and the external memory (DRAM) forms the top level.

The internal FPGA memory has low access latency, but its capacity is relatively low. In contrast, the external memory has higher capacity, but higher latency and lower throughput. In addition, accessing DRAM consumes significantly more power than accessing FPGA memory.

Thus in the field of image processing FPGA architecture it is essential topic – finding – the balance between built-in buffer and system performance.

For flow architecture, it is accepted that the FPGA receives image pixels, row by row, in the order as they are captured by image sensor. On-chip buffers in FPGA are used to store multiple frame lines to access a specific window or aperture, which is being processed. Yet the most widely used and comprehensible paradigm remains that the processing algorithm reads data from any place in the frame, processes it and writes the results back to the frames memory. For this purpose, a high throughput external memory is required and also buffer memory blocks with a volume-optimized design.

The simplest approach to increase memory throughput – is to have a few parallel memory blocks. Similarly, it is possible to implement a memory with an extra-large data word length that stores several adjacent pixels. But in these cases, besides of several external memory chips, it is necessary to have many separate FPGA pins for addresses and data output, which is often unacceptable.

This problem can be solved by organizing several cache memory blocks in the FPGA. By dividing the address space into multiple banks, for example using one memory bank for odd addresses and one for even addresses, adjacent addresses can be accessed simultaneously. For instance, four banks can be used to access four pixels in the area 2X2. Furthermore, to effectively access pixels in aperture the address may be encoded in the manner that is proposed in [9].

In pipelined random access data processing, one process can write results to one memory bank, and another can read data from the second bank. When the processing of the next frame is completed, the banks switch roles. At the same time, a third memory bank is used for better synchronization [10]. Still, such switching of banks adds one excessive period to the latency of the algorithm and has the consequence of increasing the hardware costs of the system and the use of more FPGA contacts.

A more practical approach is running the memory at a higher clock frequency than the rest of the system. Double Data Rate (DDR) memory is one example of memory that allows data to be per clock.

Normally modern high-capacity FPGAs has dedicated pins and a built-in access controller for external dynamic DDR memory of the latest generations [11]. At the same time, the project simulates multiport memory due to access time slots. In addition, blocks of buffer memory are required for writing and reading, since dynamic memory has high throughput only when transferring data rows from neighboring cells. Unfortunately, in many projects, DDR memory is also crucial for supporting the operating system of the processor embedded in the FPGA, and therefore the bandwidth of this memory drops during processing images.

If the data follows sequentially, then it is worth using buffers of type first in – first out (FIFO) type, the cells which store blocks of data, and the pixels at the output are selected at the local address [12]. One common form of intermediate data storage is row buffering. Consider the calculation of a function from nine pixel values in the aperture. According to the algorithm, nine pixels have to be read from the frame memory for each aperture position in each clock interval, and each pixel have to be read nine times as the aperture scans the image. Pixels that are next to each other horizontally are required in consecutive system clocks, so can be buffered and held in registers. This reduces the number of readings to three pixels per clock. The row buffer stores the pixel values of previous rows to avoid re-reading the pixel values (Fig. 1).

Each line buffer actually delays pixel input by one line. An obvious implementation of such a digital delay is the use of an N-stage shift register, where N is the width of the image. The RAM block (BlockRAM) in the FPGA can be configured as a FIFO buffer according to the circular buffer scheme. In addition, several parallel line buffers can be implemented as one buffer, but with a larger data bit rate [13].

Buffers of different lengths should be designed for different image sizes. In work [14], it is proposed to use a universal buffer that can be adjusted to the size of the frame and aperture with the possibility of dynamic reconfiguration. A similar buffer is described in [15], which is additionally capable of transposing the position of pixels in the window, as well as performing image correction at its edges.

Works [16, 17] presented general methods of designing a flow structure for image processing with an aperture as in the example in Fig. 1. At the same time, the functions that are sequentially performed in the algorithm are displayed in the corresponding processing blocks, which are separated from each other by buffer blocks that store several adjacent lines. The interconnections between processing blocks and buffer blocks are buses that correspond to the arcs of the data flow graph (DFG) of the algorithm.

It should be noted that such a scheme executes the algorithm specified on the network of Kahn processes [18]. In such a model, the algorithm is divided into several functions, data between which is transferred through data flows implemented as FIFO. Due to this, the intermediate data is reused many times without referring to the external memory. This organization of calculations is also recommended when programming graphic functions of the OpenCL library in FPGAs [19]. Yet it is worth noting that the network of Kahn processes is not protected from blocking.

Therefore, the **problems** are the following:

1. Optimization of data exchange between FPGA computing resources and external memory by improving the design of the corresponding buffer memory.

2. The designed buffer memory blocks should have a balance between cost and efficiency, optimized in size according to the size of the image being processed.

3. Such memory blocks must be made in the form of a cyclic FIFO buffer.

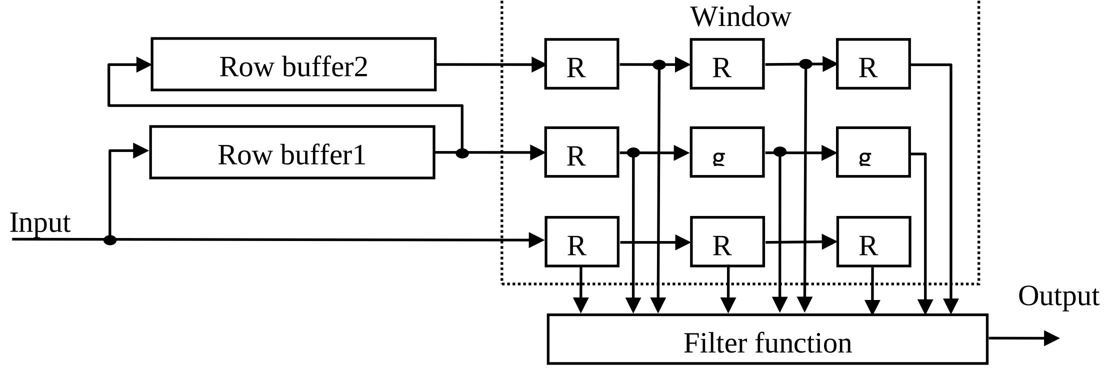

Fig.1. **A typical image frame processing scheme**
Source: **compiled by the authors**

THE PURPOSE OF THE ARTICLE

When processing images, the best strategy is to perform data calculations that are stored as much as possible in buffer memory blocks inside the FPGA. At the same time, processing is carried out in pipelined processing blocks, and buffer memory blocks are FIFO buffers. But the question of how best to organize data input and transfer of intermediate results between pipeline processing units remains open. In addition, the task of organizing calculations when the period of data tracking does not coincide with the period of the clock signal, although it is a multiple of it, has not been solved. FPGAs manufactured by Xilinx have the ability to organize FIFO buffers on the basis of logical tables (look-up tables), which have dynamic depth reconfiguration. These are SRL16 elements. But there is no method how they can be effectively applied for image processing. These questions are answered in the next section.

Therefore, the **aim of the research** is the following:

To propose a method for designing built-in buffer memory blocks for FPGAs, suitable for working with pipeline processing blocks, including clock-asynchronous calculations.

MATERIALS AND RESEARCH METHODS

The used methods

In the study of data buffering algorithms and their mapping to hardware, the method of mapping of the spatial synchronous data flows (SDF) was used [1]. The method considers periodic algorithms in which the data for processing follows with a period of P clocks, which also includes image-processing algorithms. Such an algorithm is presented as a spatial SDF in space with coordinates of the type of operation, the place of its execution, and the moment of its execution. The graph of the resulting calculator, which executes the algorithm in the pipeline mode with a period of P clocks, is obtained by mapping the spatial SPD into the subspace of structures, and the schedule of execution of operators into the time subspace.

Experimentation and prototyping

The resulting conveyor device was described in the VHDL language directly according to the spatial SDF. The device was modeled to check the correctness of the algorithm execution in the VHDL simulator. For the final test of the device's functionality, its description was compiled in the Lattice FPGA CAD and the resulting firmware was loaded into the ECP3-70 FPGA, which is located on the Lattice HDR-60 board, which is equipped with a video sensor.

METHODS OF SYNTHESIS OF BUFFER SCHEMES BASED ON SPATIAL SPD FOR DATA FLOW PROCESSING

The method of synthesis of pipeline schemes for data flow processing

The general approach to the development of a functional scheme at the level of register transfers is as follows: a set of resources (adders, multiplication blocks, registers, etc.) is selected, a schedule of algorithm operations is drawn up, and operations are assigned to resources. For this, they find a set of necessary registers and a resource-switching network. This approach is also used to design a circuit with FIFO buffers. At the same time, the chains of registers built in the scheme are replaced by appropriate FIFOs, such as SRL16. However, the chains of registers in such a scheme appear randomly and therefore, there are significantly fewer of them than possible, and thus, the registers in the scheme are used inefficiently. Additionally, the delay property of the SRL16 elements, which changes dynamically, is not used.

In [20], a method of designing pipeline computers is proposed by displaying a mapping of spatial data flows (SDF), which is represented in the resource-time space in the form of a spatial SDF. The method makes it possible to simultaneously make a schedule, minimize the number of processor elements (PEs), and search for an effective system of connections between this PEs. Here, PE means an elementary calculator with or without memory, for example, an adder, a multiplexer with a register, a FIFO, etc.

At the first stage of the synthesis according to the indicated method of the vertex-operator of the homogeneous spatial SDF together with arcs are located in three-dimensional space as sets of vectors $\boldsymbol{K}_i$ and $\boldsymbol{D}_j$ 3 taking into account the conditions given in [21]. At the same time, the coordinates of the vector $\boldsymbol{K}_i = (s,q,t)^{\mathrm{T}}$ mean the number s of PE, where the operator is executed, type q of PE and frequency component t, which is equal to the clock number in the algorithm execution period. Vectors $\boldsymbol{K}_i$ with equal time component form one layer and therefore are performed simultaneously. Time component $T(\boldsymbol{D}_j)$ of vector $\boldsymbol{D}_j = \boldsymbol{K}_i - \boldsymbol{K}_l$ equals to the delay between operators' execution, vertex vectors $\boldsymbol{K}_i$, $\boldsymbol{K}_l$ of that are adjacent.

Minimization of the number of PEs is carried out by fulfilling the requirements $|\boldsymbol{K}_{s,q}| \rightarrow L$, i.e., the number of vertices displayed in the s-th PE goes to

L, where L is the period of execution of the algorithm, clocks.

At the second stage, the spatial SDF is being balanced, which consists in adding delay vertices to the arcs of the graph until the time components of all vectors $\boldsymbol{D}_j$ are equal to 0 or 1. After that, the spatial SDF is optimized by mutual permutations of vectors-vertices from one layer in order to minimize the number of registers and the number of multiplexer inputs in the resulting structure and/or using other strategies, for example, resynchronization [20] or using genetic programming [21]. Also, the number of registers is minimized by gluing delay vertices from the same tier that store the same operand.

At the third stage, the obtained optimized spatial SDF is displayed in the graph of the computer structure by gluing vertices vectors with the same coordinates s,q.

Spatial is transformed into the schedule of execution of operators, using the property that the time component of vector Ki is equal to the moment of execution of the operator, regardless of the number of the execution period. At the same time, you can avoid building the structure and schedule if you immediately describe the computer circuit in the VHDL language [22].

This method is formalized, gives correct structural solutions that execute the algorithm with a given period in pipeline mode. Therefore, it makes sense to create a methodology for the development of FIFO register buffer devices based on this method.

The method of synthesis of buffer circuits for processing one-dimensional data flows

In local image processing, two-dimensional signal, as usual, transforms into single-dimensional. So let us consider the case of buffer schematic synthesis for single-dimensional data flow processing.

Consider some SDF subgraph, which executes in a register buffer. This subgraph performs operand x transmission from the source $\boldsymbol{K}_{i1}$ to the recipients $\boldsymbol{K}_{j1}$, $\boldsymbol{K}_{l1}$ via edges $\boldsymbol{D}_{j1} = \boldsymbol{K}_{j1} - \boldsymbol{K}_{i1}$, $\boldsymbol{D}_{l1} = \boldsymbol{K}_{l1} - \boldsymbol{K}_{i1}$, and an operand y from the source $\boldsymbol{K}_{i2}$ to the recipients $\boldsymbol{K}_{j2}$, $\boldsymbol{K}_{l2}$ via edges $\boldsymbol{D}_{j2}= \boldsymbol{K}_{j2} - \boldsymbol{K}_{i2}$, $\boldsymbol{D}_{l2}= \boldsymbol{K}_{l2} - \boldsymbol{K}_{i2}$, respectively (Fig. 2a). In order to the graph executes in the register buffer, no less than all its input vertices must have the same spatial coordinate's p.

On algorithm execution in the register, buffer operands x and y in each clock are sent to the next register buffer. It is equivalent to that in balanced SDF these operands are transmitted in each clock to the next graph layer and to the next delay vertices row, which is being mapped to a register buffer. In the other words, the adjacent delay vertices chains $\boldsymbol{K}_{Di}$ while the $R(\boldsymbol{K}_{Di})$ coordinates grow steadily and are placing across parallel straight lines, which are placed with the same inclination to the ot axis. When the sequences of the delay vertices $R(\boldsymbol{D}_{j1})$, $R(\boldsymbol{D}_{l1})$, $R(\boldsymbol{D}_{j2})$, $R(\boldsymbol{D}_{l2})$ are passed, the operands x and y are put on the corresponding subgraph vertices (Fig. 2b). The result register buffer schematic is shown on the Fig.2c.

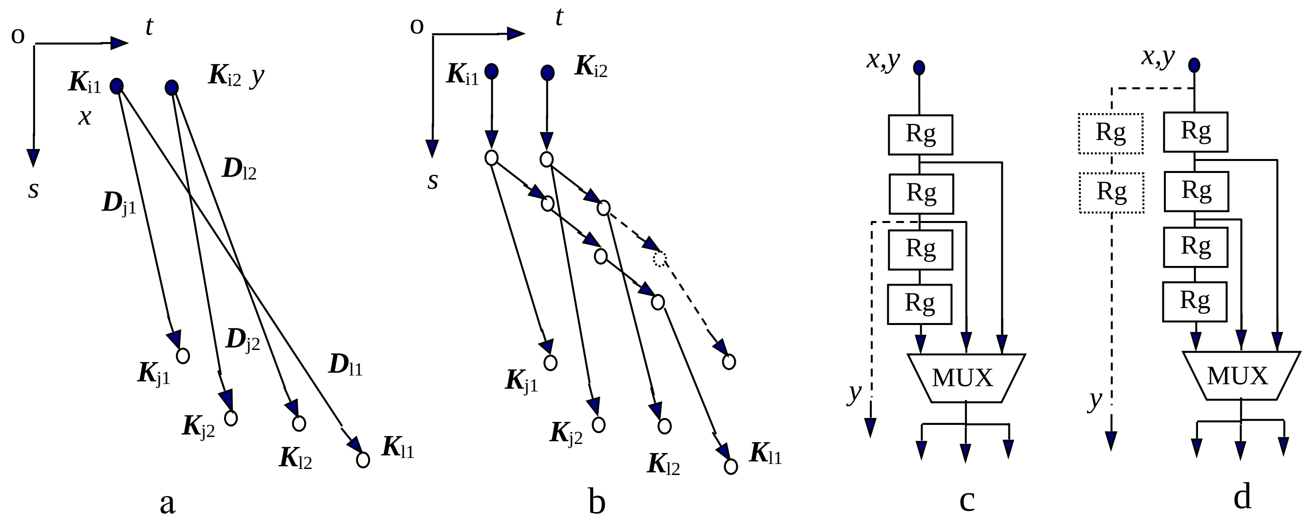

Fig.2. **SDF subgraph mapping into a register buffer:**
a – SDF before balancing; b – balanced SDF; c – graph in fig b mapped into a schematic; d – equivalent schematic mapped to register delays as in SRL16
***Source:* compiled by the authors**

The resulting buffer has a single output, therefore, an additional restriction is imposed on the subgraph of the SDF – no more than one arc leading to the output vertex should come from the delay vertices that belong to the same layer and are displayed in the buffer. Under this condition, the output multiplexer of the buffer will connect to only one register of this buffer at a time. Otherwise, the buffer must have more than one output or output multiplexer, as shown by the dotted line in Fig. 2c.

The SRL16 element has the additional clock signal accept input, controlling which enables holding up the operands propagation in buffer's registers. With the use of this input it is possible to spare some registers if the $R(\boldsymbol{D}_j)$ number is larger than the quantity of the registers in such a module. Fig.3 shows the example of the SDF transform of the one from Fig. 2b in order to make an additional delay of the operands, which are received in the vertices $\boldsymbol{K}_{l1}$, $\boldsymbol{K}_{l2}$, for a clock. Such a delay corresponds to the vectors $\boldsymbol{D}_j$, which are placed in parallel to the *ot* axis.

The design method of pipeline calculators with register buffers looks like this: Initial data – SDF, execution period of algorithm L and other optimization parameters. The method is performed in the same way as described in [21], with the exceptions described below.

At the first stage of synthesis, it is necessary to select subgraphs of the SDF corresponding to the transfer of operands between computer resources with time delays and/or shuffling of operands, which are supposed to be mapped into separate register buffers.

On the second step, it is required to balance the dependencies edges using the intermediate delay vertices. The number of intermediate delay vertices for all arcs is reduced if possible.

Place the delay vertices on parallel straight lines that are at an angle to the time axis or parallel to this axis in such a way that adjacent delay vertices differ in time coordinate by one clock. Fulfill the requirements for the correct placement of vertices, including the requirement to implement a buffer with one input and one output. If it is not possible to obtain a buffer with one output, the chain of delay vertices is split so that they are mapped into additional buffers (see. Fig. 2d).

Map the dependencies edges, together with respective delay vertices, which are incident to the vertices-receivers, into registers buffers. When compiling the computer control algorithm, if only arcs are displayed in the buffer that are at an angle to the time axis, then operands are written to the buffer registers in each cycle, and if there are arcs parallel to this axis, then writing to these registers is prohibited in the corresponding cycles (Fig. 3).

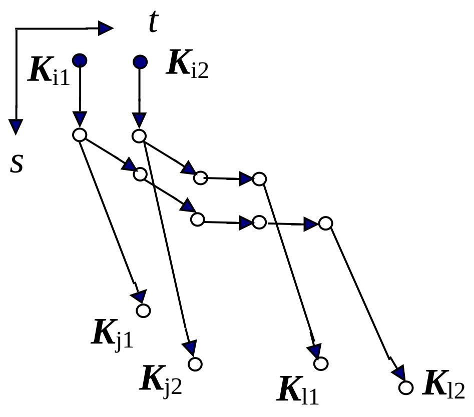

Fig.3. **Spatial SDF, which corresponds to SRL16 module with the accept input**
Source: **compiled by the authors**

In any case, the number of registers in the buffer can be minimized by applying the left edge scheduling optimizing algorithm [23]. Still, in such a case the hardware expenses for additional multiplexors increase significantly.

On the third step the computing device is described in VHDL or Verilog language and is compiled into a FPGA configuration which contains FIFO buffers on SRL16 elements, that correspond the dedicated SDF subgraphs.

Consider the example of designing a data-shuffling buffer according to the z-shaped traversal rule, which is used in image encoders according to the H264 standard [24]. If 16 input data signals arrive on the input of such a buffer in a natural sequential order, they are brought to the output according to the following sequence: 0,1,4,8,5,2,3,6,9,12,13,10,7,11,14,15. Usually such a buffer is built based on two-port rapid access memory, which leads to irrational resources usage and high latency between receiving input data and results output.

A balanced spatial SDF of the functioning algorithm of such a buffer, which is prepared according to the developed method is displayed on the Fig. 4. Here with small circles the input and output vertices are represented, while the big circles – are one-clock delay vertices. According to the spatial SDF method, all the delay vertices, which are situated on the same horizontal line, map into a single pipeline register. The description of such a buffer in VHDL code is given below.

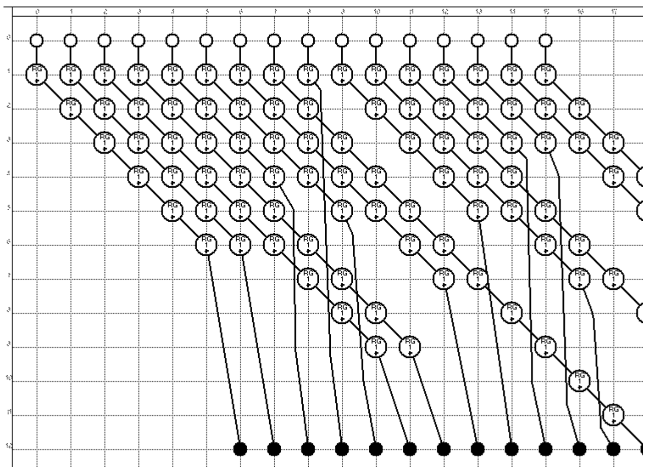
Fig. 4. **Balanced spatial SDF of a shuffle buffer**
***Source:* compiled by the authors**

```
entity BUFZ4x4 is
   port(CLK : in STD_LOGIC;
      START : in STD_LOGIC;     -- start of data
      DI : in STD_LOGIC_VECTOR(11 downto 0); --input data
      DO : out STD_LOGIC_VECTOR(11 downto 0) ); --output data
end BUFZ4x4;
architecture BUFSRL16 of BUFZ4x4 is
   type TARRAY16 is array (0 to 15) of bit_vector(11 downto 0);
   type TN is array(0 to 15) of natural range 0 to 10;
   signal srl16:TARRAY16;
- register array of SRL16
   constant ntable :TN:=(5,5,3,0,4,8,8,6,4,2,2,6,10,7,5,5); -- reg outputs of SRL16
   signal regnumber, addr:natural range 0 to 15;
begin
   FSM:process(CLK,RST)  begin      -- period counter
      if CLK'event and CLK='1' then
         if START='1' then  addr<=0;  else addr<=(addr+1) mod 16; end if;
      end if;
   end process;
   regnumber <=ntable(addr);
--mapping clock number to reg output of SRL16
   SRL16_BUF:process(CLK)  begin
 --SRL16 description
      if CLK'event and CLK='1' then
         srl16<=DI & srl16(0 to 14);
-- FIFO shift
      end if;
   end process;
   DO<= srl16(regnumber);            -- output of the shuffled datum from the register  SRL16
end BUFSRL16;
```

The resulting buffer when configured in the Xilinx Virtex-7 FPGA has minimal hardware costs – 19 LUTs, of which 12 LUTs are SRL16 elements. This buffer can be used at a clock frequency of up to 900 MHz. Such results show the effectiveness of the developed method.

The method of synthesis of buffer circuits for processing two-dimensional data streams

A two-dimensional array representing an image frame is stored in external DRAM and sent to the FPGA for processing pixel by pixel according to the frame scan law or some other rule. This data flow must be temporarily stored in an internal buffer, from which the data belonging to a certain aperture is read (Fig. 1).

According to the spatial SDF method, the position of a pixel in a DDR can be encoded by a vector

$$\boldsymbol{K}_i = (s_{1i}, s_{2i}, q, t)^{\mathrm{T}},$$

where s_1, s_2 are coordinates of the pixel in the frame in the row and column, respectively; q is the type of storage device; t is the moment in time under consideration.

When transferring a frame to the FPGA buffer memory, its pixels are displayed in the data flow elements

$$\boldsymbol{K}_j' = (s, q', t')^{\mathrm{T}},$$

where s – is the number of line when transmitting several pixels at the simultaneously.

According to the systolic processors designing theory [25] and the parallel processor structures synthesis method [26], such a mapping $\boldsymbol{K}_i' = R(\boldsymbol{K}_i)$, must be injective, linear and monotonous. Here, injectivity means that no two pixels can be stored in the same memory cell or transmitted over the communication line at the same time. Linear and monotonic mapping preserves the dependence of pixel precedence.

If frame transmission is used according to the law of line scanning when data is transmitted through one line, and the time is counted in clocks, then the display function is as following:

$$R(s_{1i}, s_{2i}, q, t)^T = (0, q', Ns_{1i} + s_{2i} + t)^T, \quad (1)$$

where N is frame’s width.

The data buffer makes it possible to obtain from the signal $\boldsymbol{K}_j'$ the currently processed pixels with the coordinates $\boldsymbol{K}_j''$, which belong to the aperture (Fig. 1). Thus, the buffer functioning algorithm is described with the set of vector-edges

$$D_j = K_j'' - K_i'. \quad (2)$$

The sets of vectors $\boldsymbol{K}_j'$, $\boldsymbol{K}_j''$ and $\boldsymbol{D}j$ form the spatial SDF. According to this SDF the buffer schematic is synthesized just as is described in the previous section.

Consider an example of designing a buffer for filtering with a filter with an aperture of 3×3. Let the image frame have dimensions of 5×8, thereby, $N=8$ and contains of pixels $x_{i,j}$ (see Fig. 5). The pixels of the frame are being read by rows and they are used to form the data flow $x_k = x_{i,j}$, where $k = Ni + j$. Vertex vectors $\boldsymbol{K}_k'$ correspond to flow elements. Operator vertex $\boldsymbol{K}''$ collect the data from nine pixels and therefore it is connected via vectors-edges $\boldsymbol{D}_k = \boldsymbol{K}_k' – \boldsymbol{K}''$ with the flow vertices, which mark the aperture's pixels.

Balanced spatial SDF of the buffer functioning algorithm, which is prepared with the method, is presented on Fig.6. Here the delay vertices, which are mapped into two rows buffers, highlighted by rectangles. The rest of the delay vertices are mapped into the respective pipeline registers. The resulting structure is matching the structure shown in Fig.1. If necessary, the N = 8 parameter can be replaced with any number, as well as the aperture size.

Therefore, the proposed method allows formalized construction of buffer circuits for image processing. In this example, the processing block executing the function *f(x(k))* receives input data in one clock cycle. If necessary, the method can provide data presentation in arbitrary cycles, for example, when thinning data during image decimation.

This method was applied during the development of an intelligent video camera that processes images with a wide dynamic range [27]. At the same time, the method made it possible to abandon the use of external memory for saving image frames, to ensure the development of image processing pipeline blocks with an arbitrary order of incoming data from buffer memory blocks.

Simulation and hardware implementation results

According to the simulation results of the method of designing buffer circuits, an experimental sample of the technical vision system for receiving the video stream from the HDR sensor, processing it and outputting it to the display via the HDMI interface is simulated. The model is implemented in a prototype built based on the Helion-60 board of the Helionvision company, on which the Lattice ECP-3 LFE3-70EA FPGA is installed.

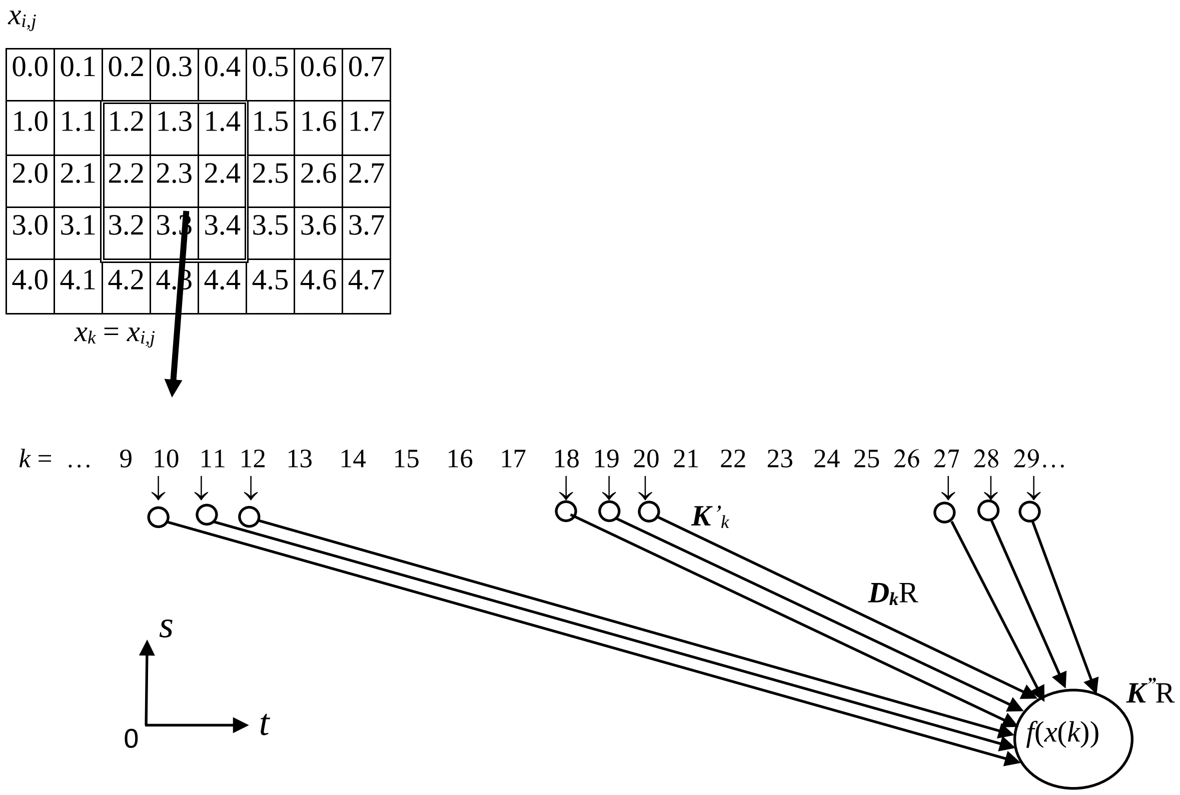

Fig. 5. **Display of the frame in the spatial SDF of the buffer**
Source: **compiled by the authors**

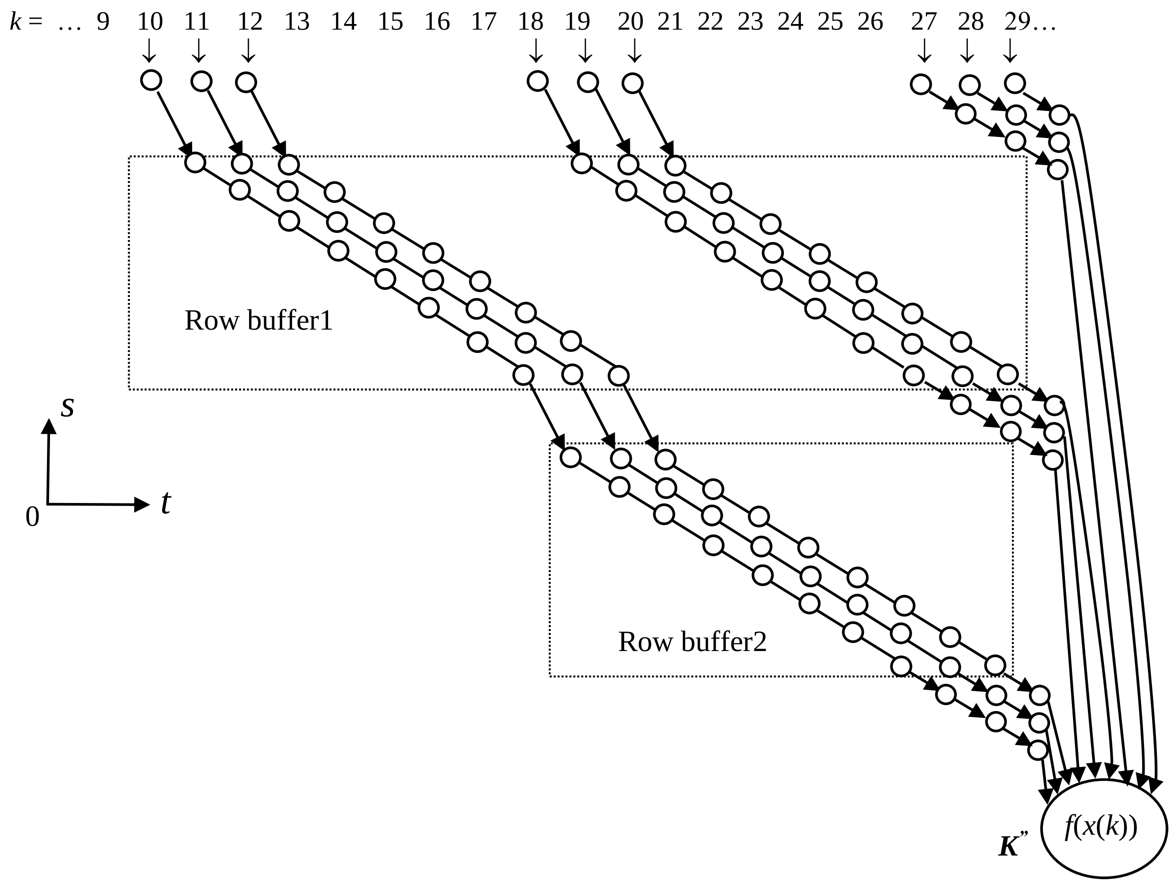
k = … 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29…
Row buffer1
Row buffer2
s
t
0
K''
f(x(k))

Fig. 6. **Balanced spatial SDF of buffer**
***Source:* compiled by the authors**

The component modules of the system and their parameters are given in Table 1.

The DECOMP block decompresses the input 12-bit video stream to 20 bits.

The GIST block calculates the distribution of pixels by brightness and outputs to the NORM1 block the parameters that are necessary for image normalization. The COL_FILTR5 block performs reverse color filtering (debayerization), i.e., restores image colors using interpolation.

The MEDIANF_3X3 filter performs median filtering of the image, improving the signal-to-noise ratio. The BW block extracts the brightness component from the video signal. The HDR_FILTR53 block, thanks to adaptive filtering, compresses the dynamic range of the video signal from 18 to 8 bits for the brightness component. The COLOR_REST block restores the colors for the compressed image.

Table 1. **Component modules of the technical vision system**

Module name	Purpose	Q-ty LUT	Q-ty Triggers	Q-ty Adders	Q-ty Mem. blocks	Max clock frequency, MHz	Latency
DECOMP	HDR-video decompressor	46	60	-	-	280	3
GIST	Histogram building	892	59	-	-	139	N·m
COL_FILTR5	Debayerization	279	779		4	169	2N + 6
MEDIANF_3X3	Median filter	9689	3906	-	6	61	N + 3
BW	Convert to black and white image	22	10	-	-	270	1
HDR_FILTR53	HDR-compressor	8400	3313	37	4	41	2N + 16
COLOR_REST	Color restoring	156	490	3	-	28	1
NORM1	Normalization block	76	60	1	-	164	1

***Source:* compiled by the authors**

After compiling the technical vision system project, the results are shown in Table 2. Analysis of the design results of memory blocks showed that their required number is quite small.

Table 2. **Project compiling results**

FPGA elements	Quantity	Available	%
Configurable logic blocks (CLB slices)	8563	33264	25.7
Triggers	5694	66528	8.6
Multiplexors 18x18	34	128	26.6
Memory blocks 1024x18	56	240	23.3

Source: **compiled by the authors**

CONCLUSIONS

A review of the ways in which image data can be put into the FPGA for processing has shown that the best data storage strategy is one that involves storing as much of the data as possible in the FPGA's buffer memory blocks. Therefore, the development of methods for organizing such buffers is demanded. The proposed method of synthesis of blocks of buffer memory is based on the method of mapping spatial SDF into computing resources. At the same time, buffer memory registers are used as resources. The new method consists in converting a two-dimensional representation of an image into a one-dimensional one, constructing a spatial SDF and describing it in a hardware description language such as VHDL. In contrast to known methods of designing buffer circuits, the method makes it possible to carry out their development in a formalized manner with the minimization of hardware costs, directing the synthesis to obtain buffers of the FIFO type or random access memory or register memory, ensuring a predetermined order of data input and output. The level of formalization of the method makes it possible to implement it in automated design systems.

REFERENCES

1. Wang, J., Zhong, S., Yan, L. & Cao, Z. “An embedded system-on-chip architecture for real-time visual detection and matching”. *IEEE Trans. Circuits Syst. Video Technol*. 2014; 24: 525–538. DOI: https://doi.org/10.1109/TCSVT.2013.2280040.

2. Mondal, P., Biswal, P.K. & Banerjee, S. “FPGA based accelerated 3D affine transform for real-time image processing applications”. *Computers & Electrical Engineering*. 2016; 49: 69–83. DOI: https://doi.org/10.1016/j.compeleceng.2015.04.017.

3. Perri, S., Frustaci, F., Spagnolo, F. & Corsonello, P. “Design of real-time FPGA-based embedded system for stereo vision”. *In Proceedings of the 2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. Florence: Italy. 2018. p. 1–5. DOI: https://doi.org/10.1109/ISCAS.2018.8351886.

4. Conti, F., Rossi, D., Pullini, A., Loi, I. & Benini, L. “PULP: A ultra-low power parallel accelerator for energy-efficient and flexible embedded vision”. *Journal of Signal Processing Systems*. 2016. p. 339–354. DOI: https://doi.org/10.1007/s11265-015-1070-9.

5. Stevanovic, U., Caselle, M., Cecilia, A., Chilingaryan, S., Farago, T., Gasilov, S., Herth, A., Kopmann, A., Vogelgesang, M., Balzer, M., Baumbach, T. & Weber, M. A. “Control system and streaming DAQ platform with image-based trigger for X-ray Imaging”. *IEEE Transactions on Nuclear Science*. 2015; 62: 911–918. DOI: https://doi.org/10.1109/TNS.2015.2425911.

6. Guo, C., Meguro, J., Kojima, Y. & Naito, T. “A multimodal ADAS system for unmarked urban scenarios based on road context understanding”. *IEEE Transactions on Intelligent Transportation Systems*, 2015; 16: 1690–1704. DOI: https://doi.org/10.1109/TITS.2014.2368980.

7. Ma, Y., Cao, Y., Vrudhula, S. & Seo, J. S. “An automatic RTL compiler for high-throughput FPGA implementation of diverse deep convolutional neural networks”. *Proceedings of the 27th International Conference on Field Programmable Logic and Applications (FPL)*. Ghent: Belgium. 2017. p. 1–8. DOI: https://doi.org/10.23919/FPL.2017.8056824.

8. Dessouky, G., Klaiber, M.J., Bailey, D.G. & Simon, S. “Adaptive dynamic on-chip memory management for FPGA-based reconfigurable architectures”. *Proceedings of the 24th International Conference on Field Programmable Logic and Applications (FPL)*. Munich: Germany. 2014. p. 1–8. DOI: https://doi.org/10.1109/FPL.2014.6927471.

9. Kim, K. & Kumar, V. K. P. “Parallel memory systems for image processing”. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. San Diego: California, USA. 1989. p. 654–659. DOI: https://doi.org/10.1109/CVPR.1989.37915.

10. Khan, S., Bailey, D. & Sen Gupta, G. "Simulation of triple buffer scheme (comparison with double buffering scheme)". *Proceedings of the 2nd International Conference on Computer and Electrical Engineering (ICCEE 2009).* Dubai: UAE. 2009; 2: 403–407. DOI: https://doi.org/10.1109/ICCEE.2009.226.

11. Churiwala, S. "Designing with Xilinx® FPGAs: Using Vivado". *Springerz. Switzerland.* 2017. DOI: https://doi.org/10.1007/978-3-319-42438-5.

12. Sedcole, P., Cheung, P. Y. K., Constantinides, G. A. & Luk, W. "Run-time integration of reconfigurable video processing systems". *IEEE Transactions on VLSI Systems.* 2007; 15 (9): 1003–1016. DOI: https://doi.org/10.1109/TVLSI.2007.902203.

13. "ChipScope Pro 10.1 Software and Cores User Guide". Vol. UG029, Xilinx Inc. – Available from: https://docs.xilinx.com/v/u/en-US/chipscope_pro_sw_cores_10_1_ug029. – [Accessed: 07.09.2021].

14. Shi R., Wong J. S. J. & So, H. K. "High-throughput line buffer microarchitecture for arbitrary sized streaming image processing". *J Imaging*. 2019; 5(3): 34. DOI: https://doi.org/10.3390/jimaging5030034.

15.Bailey D. G. & Ambikumar A. S. "Border handling for 2D transposes filter structures on an FPGA". *Journal of Imaging*. 2018; 4(12): 138. DOI: https://doi.org/10.3390/jimaging4120138.

16. Ikarashi, Y. Ragan-Kelley, J. Fukusato, T. Kato, J. & Igarashi, T. "Guided optimization for image processing pipelines". *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC).* 2021. p. 1–5. DOI: https://doi.org/10.1109/VL/HCC51201.2021.9576341.

17. Özkan, M. A., Reiche, O., Hannig, F. & Teich, J. "FPGA-based accelerator design from a domain-specific language". *Proceedings of the 26th International Conference on Field Programmable Logic and Applications (FPL).* Lausanne: Switzerland. 2016. p. 1–9. DOI: https://doi.org/10.1109/FPL.2016.7577357.

18. Lee, E. A. & Neuendorffer, S. "Concurrent models of computation for embedded software". *IEE Proc.-Comput. Digit. Tech.* March 2005; 152 (2): 239–250. DOI: https://doi.org/10.1049/ip-cdt:20045065.

19. Waidyasooriya, H. M. A., Haiyama, M. & Uchiyama, K. "Design of FPGA-based computing systems with OpenCL". *Springer.* 2018. DOI: https://doi.org/10.1007/978-3-319-68161-0.

20. Sergiyenko, A., Serhienko, A. & Simonenko, A. "A method for synchronous dataflow retiming". *IEEE First Ukraine Conference on Electrical and Computer Engineering (UKRCON*). 2017. p. 1015–1018. DOI: https://doi.org/10.1109/UKRCON.2017.8100404.

21. Sergiyenko, A., Serhienko, A. & Romankevich, V. "Genetic programming of pipelined datapaths for FPGA". *IEEE 40th International Conference on Electronics and Nanotechnology (ELNANO*). 2020. p. 802–806. DOI: https://doi.org/10.1109/ELNANO50318.2020.9088773.

22. Maslennikow, O. & Sergiyenko, A. "Mapping DSP algorithms into FPGA". *International Symposium on Parallel Computing in Electrical Engineering (PARELEC'06).* 2006. p. 208–213. DOI: https://doi.org/10.1109/PARELEC.2006.51.

23. Ruvald Pedersen M. & Madsen, J. "Optimal register allocation by augmented left-edge algorithm on arbitrary control-flow structures". *NORCHIP 2012*. 2012. p. 1–6. DOI: https://doi.org/10.1109/NORCHP.2012.6403107.

24. Richardson, I. E. G. "H.264 and MPEG-4 video compression. Video coding for next-generation multimedia". *Wiley*. 2003. p. 281. DOI: https://doi.org/10.1002/0470869615.

25. Cong, J. & Wang, J. "PolySA: Polyhedral-based systolic array auto-compilation". *IEEE/ACM International Conference on Computer-Aided Design (ICCAD).* 2018. p. 1–8. DOI: https://doi.org/10.1145/3240765.3240838.

26. Sergiyenko, A. & Simonenko, V. "The displaying of periodic algorithms into field programmable gate arrays". *Electronic Modelling*. 2007; 29 (2): 49–61.

27. Sergiyenko, A., Serhiienko, P. & Zorin, J. "High dynamic range video camera with elements of the pattern recognition". *IEEE 38th International Conference on Electronics and Nanotechnology (ELNANO).* 2018. p. 435–438. DOI: https://doi.org/10.1109/ELNANO.2018.8477556.

Conflicts of Interest: the authors declare no conflict of interest

Received 22.08.2022
Received after revision 04.10.2022
Accepted 18.10.2022

DOI: https://doi.org/10.15276/aait.05.2022.16
УДК 004.383

Буферизація зображень при їх обробці у спеціалізованих процесорах

Сергієнко Анатолій Михайлович[1)]
ORCID: http://orcid.org/0000-0001-5965-1789; asergy@bigmir.net. Scopus Author ID: 27868137900
Романкевич Віталій Олексійович[1)]
ORCID: http://orcid.org/0000-0003-4696-5935; romankev@scs.kpi.ua. Scopus Author ID: 57193263058
Сергієнко Павло Анатолійович[1)]
ORCID: http://orcid.org/0000-0003-3030-0074; paulsrgnk002@gmail.com. Scopus Author ID: 57204497516
[1)] Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», пр. Перемоги, 37. Київ, 03056, Україна

АНОТАЦІЯ

У багатьох застосунках для цифрової обробки зображень, які реалізовані у програмованих логічних інтегральних схемах, кадри зображення, що обробляються, зберігаються у зовнішній динамічній пам'яті. Продуктивність такого застосунку залежить від швидкодії динамічної пам'яті та необхідної кількості звертань до неї під час виконання алгоритму. Ця продуктивність оптимізується завдяки використанню буферної пам'яті, яка реалізована у програмованих логічних інтегральних схемах. Але не існує загального методу, який би дав змогу формально синтезувати буферну пам'ять з заданими пропускною здатністю, порядком слідування вхідних і вихідних даних та мінімізованими апаратними витратами. В роботі розглядаються особливості вводу й обробки зображень у спеціалізованих процесорах на базі програмованих логічних інтегральних схемах. Аналізуються методи побудови буферних схем у програмованих логічних інтегральних схемах, завдяки яким зменшується інтенсивність обмінів даними з зовнішньою пам'яттю. Приводиться метод синтезу конвеєрних схем з заданими характеристиками продуктивності та порядком слідування даних, який ґрунтується на відображенні просторового графу синхронних потоків даних у структуру, що реалізована в програмованих логічних інтегральних схемах. Запропонований метод проектування буферних схем, який заснований на відображенні просторового графу синхронних потоків даних у локальну пам'ять у вигляді ланцюжків конвеєрних регістрів. Метод дає змогу організувати потік вхідних даних на вхід вбудованих конвеєрних блоків обробки зображень, в якому дані слідують у заданому порядку, а також мінімізувати об'єм буферної пам'яті. Метод забезпечує використання в програмованих логічних інтегральних схемах вбудованих динамічно регульованих регістрових затримок, що підвищує ефективність буферизації. Метод було перевірено при розробці інтелектуальної відеокамери, яка виконує алгоритм стиснення відеозображення з широким динамічним діапазоном за алгоритмом Retinex і одночасно виділяє характерні точки у зображенні для подальшого розпізнавання образів. При цьому виконується багатократна децимація кадра. Завдяки багатократній буферизації зображення у програмованих логічних інтегральних схемах, вдалось уникнути застосування зовнішньої динамічної пам'яті.

Ключові слова: Програмовані логічні інтегральні схеми, граф синхронних потоків даних; обробка зображень; буферна пам'ять

ABOUT THE AUTHORS

Anatolij M. Sergienko - Doctor of Engineering Sciences, Professor, Professor of Department of Computer Engineering. National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", 37, Peremogy Av. Kyiv, 03056, Ukraine
ORCID: http://orcid.org/0000-0001-5965-1789; asergy@bigmir.net. Scopus Author ID: 27868137900
Research field: Computer architecture; hardware synthesis; digital signal processing

Сергієнко Анатолій Михайлович - доктор технічних наук, професор , професор кафедри обчислювальної техніки. Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», пр. Перемоги, 37. Київ, 03056, Україна

Vitalij O. Romankevich - Doctor of Engineering Sciences, Professor, Professor of System Programming and Special Computer System Department. National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", 37, Peremogy Av. Kyiv, 03056, Ukraine
ORCID: http://orcid.org/0000-0003-4696-5935; romankev@scs.kpi.ua. Scopus Author ID: 57193263058
Research field: Dependability of fault-tolerant multiprocessor control systems; self-testing of multiprocessor systems

Романкевич Віталій Олексійович – доктор технічних наук, професор, професор кафедри Системного програмування та спеціальних комп'ютерних систем. Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», пр. Перемоги, 37. Київ, 03056, Україна

Pavlo A. Serhiienko - PhD student, Assistant of Department of System Programming and Specialized Computer Systems. National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", 37, Peremogy Av. Kyiv, 03056, Ukraine
ORCID: http://orcid.org/0000-0003-3030-0074; paulsrgnk002@gmail.com. Scopus Author ID: 57204497516
Research field: Pattern recognition in images; embedded high-performance manycore systems in FPGA

Сергієнко Павло Анатолійович – аспірант, асистент кафедри Системного програмування та спеціальних комп'ютерних систем. Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», пр. Перемоги, 37. Київ, 03056, Україна