Real-Time Sorting and Lossless Compression of Data on FPGA

Valery A. Kokovin[1], Saygid U. Uvaysov[2] and Svetlana S. Uvaysova[3],
[1]State University "Dubna", branch "Protvino", Protvino, Moscow reg., Russia
[2]Moscow Technological University (MIREA), Moscow, Russia
[3]National Research University Higher School of Economics, Moscow, Russia
[1]kokovin@uni-protvino.ru, [2]uvaysov@yandex.ru, [3]uvay@yandex.ru

***Abstract*—The method of data lossless compression with preliminary sorting in real time is considered in the paper. The compression method is based on the analysis of the frequency distribution of the incoming data stream, the selection of the constant by sorting and, on the basis of this, subsequent compression. The combination of sorting and data compression proposed in the article allows saving processing time and dynamically manage the network controller load. The hardware algorithm implementation on FPGA for sorting and compressing data during stream processing of information is considered. The solution makes it possible to implement an algorithm in the form of an IP core, with the ability to adapt it to the characteristics of solving its task of compressing data, thereby increasing system performance. This algorithm can be used to create embedded applications with limited computing resources and time-critical requirements. The device, based on the method considered, showed stable operation in the task of processing data from a multichannel system of high-speed sensors. This algorithm can be applied in solving problems of creation the telecommunications networks of distributed control systems, data processing subsystems Internet of Things and Internet of Robotic Things.**

***Keywords*—distributed control system, FPGA, lossless compression, sorting of data, telecommunication system**

I. Introduction

The solution of the problem of technological processes (TP) automation, when these processes have a distributed territorial and algorithmic character, is connected with the construction of a distributed control system, which necessarily includes a telecommunications system (TCS). Many TPs, and therefore their control systems, are characterized by a large amount of sensory information coming from control objects and rapid algorithm implementation. Such TPs can be occurred in the chemical, engineering and other industries, as well as in scientific complexes [1].

As in any CS, the response time of a distributed CS to a control action is the determining criterion in the development of this type systems. The data transfer speed between separate subsystems and their volume in the TCS largely determine the reaction rate of the entire control system. To improve the TCS performance, various methods of compressing transmitted data are used. If data are transferred to form control actions on the control object, for example, information from sensors or events, then lossless compression methods are used. In the case of data plan information transmission, for example, an image or sound for the process operator, during compression the losses are admissible.

Over the past decade, a new direction has emerged, called the Internet of Things (IoT), which is a distributed system of compact embedded applications, connected by wireless or wired networks [2]. Today, on the IoT basis, a new direction is successfully developing, which by means of network technologies unites robots or robotic devices. This direction, called the Internet of Robotic Things (IoRT), is aimed at implementing robotic technologies, by extending the functionality of IoT devices. The paper [3] presents the IoRT concept, which highlights the tremendous flexibility in the development and implementation of new applications for networked robotics to achieve the goal of providing distributed computing resources as the main utility. Robotic devices can track events, collect data from a variety of sensors from various sources and use the intellectual capabilities of their calculators to determine the best actions.

The task of data compression in applications of the listed directions is especially relevant because of their limited resources. At the same time, often the developer faces the problem of choosing between many criteria and optimizing solutions. In the proposed paper, the device implemented on a fairly efficient method of data lossless compression is considered. It is implemented in hardware on the basis of Field-Programmable Gate Array (FPGA). The proposed compression method is based on the frequency distribution analysis of the incoming data stream, the selection of a constant by sorting and on the basis of this, subsequent compression.

The paper is as follows. Section 2 provides an overview of sorting and compression methods for lossless data implemented on an FPGA. Section 3 discusses the device implementation, discusses the features of the structure and algorithms of the solved problem. The obtained results and their discussion are given in Section 4. Finally, the conclusion is given in Section 5.

978-1-5386-3498-1/18/$31.00 © 2018 IEEE

II. Methods of Hardware Sorting and Data Lossless Compression Implemented on FPGA

There are a large number of methods for sorting and compressing data [4], [5] and [6]. The article [7] is devoted to the use of stream processing in intellectual systems. A detailed analysis of lossless and lossy compression methods is given in [8] [9], and the main ideas and concepts embodied in these methods are also described. Lossless compression methods are the most interesting, since in distributed control systems control data are usually transmitted to the control algorithm implementation. The compression efficiency is an important characteristic of compression. By efficiency, we will take not only the compression ratio (the ratio between the initial data size and the data after compression size), but also the speed of the compression device - the compressor. The compressor can be implemented both programmatically and hardware.

As a rule, widespread compression algorithms relate to universal algorithms. The algorithm universality can be attributed to the dignity, but the emergence of a large number of methods and algorithms for data compression [8] suggests that the most effective compression algorithm is specialized for solving a particular problem. Given the fact that not only the compression ratio but also the compressor speed is important, it can be noted that there are more and more devices on the market performing data compression on the hardware platform, usually on FPGA. For example, in hardware compressors [10], the Huffman and RLE algorithms are used to compress the image in the JPEG standard [11]. These compressors are implemented as IP (Intellectual Property) - FPGA based cores by Altera [12] and Xilinx [13]. By purchasing an IP-core, the developer can adapt it to the specifics of solving the task data compression task, thereby increasing the performance of its system. In addition, it becomes possible to change the algorithm of data compression "on the fly", changing the FPGA configuration, that makes it possible to adjust to the changing system operating conditions.

III. The Device Implementation

As it was said above, the effective solutions implementation raises for developers the problem to choose the architecture and algorithms of embedded applications in limited resources conditions. That is, the developer actually faces the problem of choosing between many criteria and optimizing solutions. We can distinguish the following criteria for optimizing data processing (sorting), compressing and transmitting the result over the network in embedded systems implemented on controllers and FPGAs:

- Performance speed as a function of the complexity of implementing the data processing algorithm
- Minimizing energy consumption in data processing as a function of performance speed
- Minimizing energy consumption as a function of the used volume FPGA
- Минимизация потребления энергии как функция использованного объема FPGA.
- Efficient network controller load (performance criterion without downtime)
- Data compression rate as a function of time.

In multicriteria selection problems, it is not always necessary to find the maximum or the minimum function value. In some cases, it is necessary to average the value of a certain parameter, i.e. to find a compromise solution.

A. Formulation of the problem

Let the input of the device realized on the FPGA (Fig. 1) receive a serial data stream with a dimension of bytes. The binding to the byte size is due to the subsequent transfer of data through the network controller via Ethernet protocols, Profibus et al. The size of the data stream is unlimited. It is necessary to perform the following operations in real time:

- to calculate the frequency of appearance of a single value data, i.e. determine the frequency distribution of incoming data;
- to sort the number of occurrences of certain data values in ascending order;
- to allocate a maximum in the frequency distribution of data (the constant for compression by the method proposed in [9]);
- to perform data compression.

The device for sorting and compressing data (Fig.1) consists of the following blocks:

- Input Register – input register, which receives data stream from external devices (for example, sensors or byte-by-byte video output from a camera);
- RAM 256x8 – the built-in FPGA memory bank is used to store the number of occurrences in the input data stream of identical values. The addresses of the memory cells are the data values (signals *data_r* [7..0]), coming from *the Input Register.* The memory data input is connected to the counter output COUNTER (signals *cnt_out* [7..0]);
- COUNTER – counter for counting the number of occurrences in the input data stream with the same *data_nmb* values. The counter input is connected to the RAM output by the signals *ram_out* [7..0], and from the output the data *cnt_out* [7..0] comes again to the memory input with the same address;
- SORTER – sorter of data values from the memory RAM 256x8 per 256 cells;
- FIFO – buffer memory to store input data for sorting time. Input signals *data_r* [7..0], output - *data_FIFO* [7..0];
- COMPRESSOR – data compression based on the selected constant after sorting At the module input, the data stream from FIFO (signals *data_FIFO* [7..0]) and signals of selected constants *data_const* [7..0];
- Control Unit – block to form the synchronous series, which provide the specified sequential-parallel operation of the entire device.

The entire input data stream is conventionally divided into packets that do not exceed the maximum transmission unit (MTU) for the selected network controller protocol. Compliance of this condition saves data transmission time and does not fragment the transmitted packet. Accordingly, all the

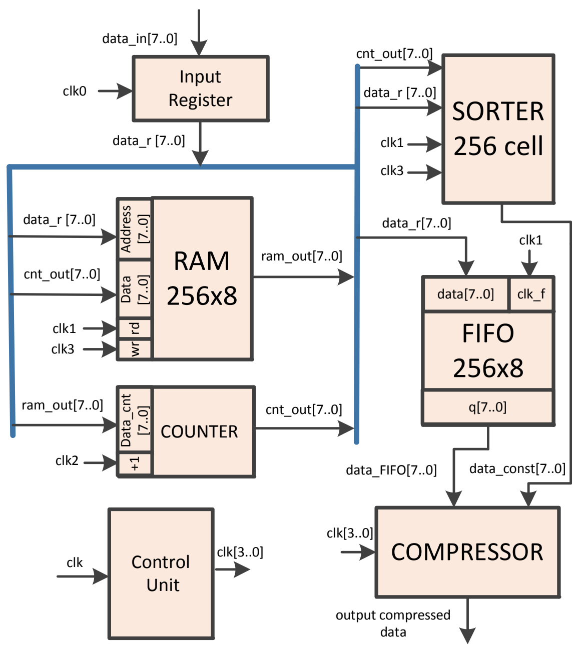

Fig. 1. Structure of the device for sorting and compressing data on FPGA

above operations for sorting and compressing data are valid for this amount. This partitioning allows optimizing the use of the network controller and performing its efficient download. If the controller is used to transfer data from several streams, it is necessary to adaptively control the process of compressing the input data for each stream.

For definiteness, we take the number of bytes N above which sorting and compression operations are performed, not exceeding 256 ($N \leq 256$).

B. Definition of Frequency Distribution of Data

The solution of the problem of determining the frequency distribution of input data is a sorting preparation operation. For this purpose, RAM memory size of 256x8 and counter COUNTER are used. Each new data byte from the Input Register to the address input of the memory block (Fig.1) loads the contents of the corresponding memory cell into the counter. Further, the increased counter value is transmitted to the input of the SORTER module, and again it is written to the memory cell with the former address, i.e. the memory contents are incremented. According to the size assumption of the data packet (256 bytes), the maximum value in each memory cell will not exceed 256 (if the entire packet contains bytes with one value). Thus, at each step n (n = 1, 2, 3..N), the SORTER module sorts not the value of the input data byte, but the number of bytes with the same values *data_nmb*.

C. Sorting Data

In general, sorting is an ordering a certain amount of arbitrary width data. This data can contain information and an index (a characteristic) that is used to sort. The information and the index can be combined with a data value. Let us sort the data represented by the entries $D(x_1)$, $D(x_2), \ldots, D(x_n)$ with the indices $x_1, x_2, \ldots, x_n$, where $n = \{1,2,\ldots,N\}$. The indices should be related by the order relation «<», so that for any three indices (for example, *a, b* and *c*) two conditions are met:

- exactly one of the possibilities $a<b$, $a=b$, $a>b$ is true;
- if $a<b$ and $a<b<c$, then $a<c$ [12].

The sorting task is to arrange the data in such a way that indices of this data are located in a non-decreasing order:

$$D(x_1) \leq D(x_2) \leq \ldots \leq D(x_n)$$

The data was sorted according to the algorithm proposed in [15]. This algorithm is based on the sorter (SORTER) implementation in the form of intellectual cells that interact with each other through the formation of one-bit signals (events). To solve the problem of sorting the considered device, the algorithm (given in [15]) is modified.

The sorter is a set of cells (in our case ≤ 256), which operates according to certain rules. The SORTER input provides two-byte data *srt_word*, which in the high-order byte contain the value *data_r* [7..0], and in the least significant bit of byte *cnt_out* [7..0], which are the *data_nmb* index. In each cycle, SORTER operates according to the following algorithm [Fig.2]:

- The SORTER sequentially receives immediately to all the cells the data with the indexes. Before sorting all cells are empty.
- If the *high_byte* of any cell coincides with the value of the next input byte *data_r* [7..0], then this cell is excluded by shifting the contents of all the cells located below upwards.
- Then, each occupied cell compares the value of its least significant byte with the value of index

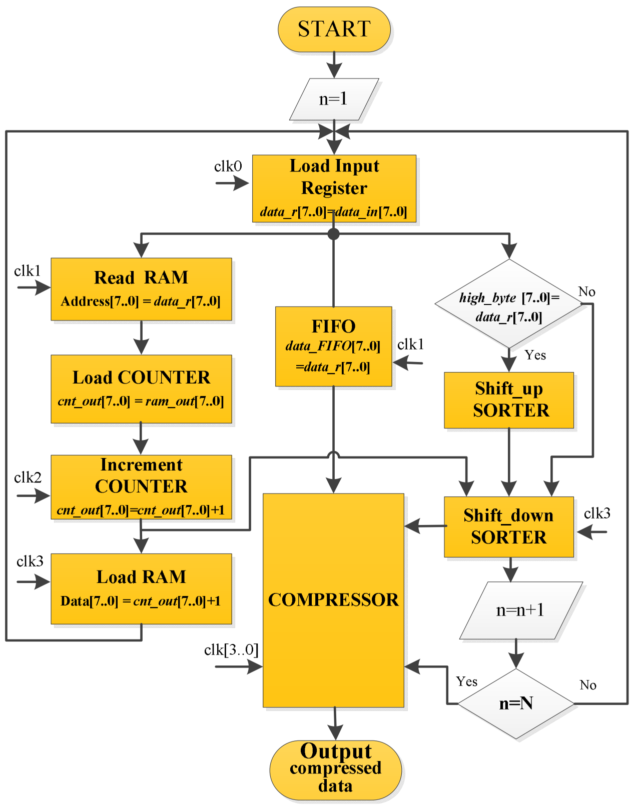
START
n=1
clk0
Load Input Register
data_r[7..0]=data_in[7..0]
clk1
Read RAM
Address[7..0] = data_r[7..0]
Load COUNTER
cnt_out[7..0] = ram_out[7..0]
clk2
Increment COUNTER
cnt_out[7..0]=cnt_out[7..0]+1
clk3
Load RAM
Data[7..0] = cnt_out[7..0]+1
FIFO
data_FIFO[7..0] =data_r[7..0]
clk1
high_byte [7..0]= data_r[7..0]
No
Yes
Shift_up SORTER
Shift_down SORTER
clk3
COMPRESSOR
clk[3..0]
n=n+1
n=N
Yes
No
Output compressed data

Fig. 2. The algorithm of the device for sorting and compressing data without losses.

data_nmb of the word *srt_word*. If *data_nmb* is smaller than the cell index, then the content of all cells is shifted, beginning with the current one and lower. The value of the word *srt_word* is written to the freed cell.

- The sorting cycle continues until the parameter *n* is equal to *N*.
- If *n* = *N*, then the most significant byte value of the lowest occupied cell SORTER (*data_const* [7..0]) will be fed to the module COMPRESSOR.

Further, the constant value and the data stored in the FIFO are fed to the module COMPRESSOR.

D. Lossless compression data

In paper [9] the operation algorithm of the data compression device is described. The input array data (volume of 88 bytes) arrives in parallel. The data array is a set of the technological timer channel signals of the linear accelerator URAL30 of the

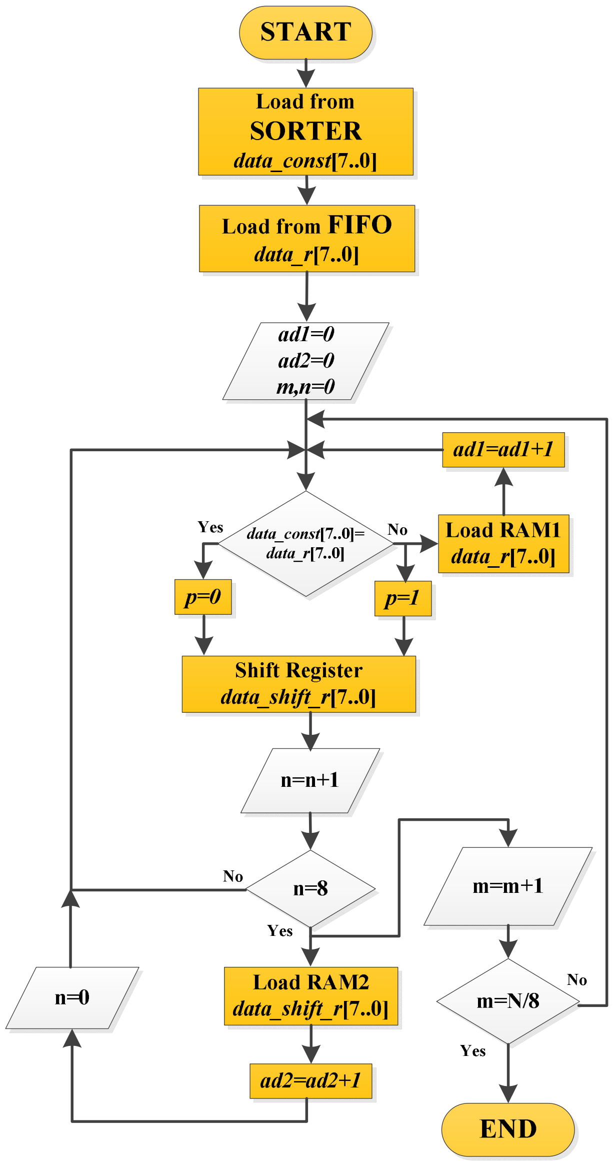
START
Load from SORTER data_const[7..0]
Load from FIFO data_r[7..0]
ad1=0
ad2=0
m,n=0
ad1=ad1+1
Yes
data_const[7..0]= data_r[7..0]
No
Load RAM1 data_r[7..0]
p=0
p=1
Shift Register data_shift_r[7..0]
n=n+1
No
n=8
Yes
m=m+1
n=0
Load RAM2 data_shift_r[7..0]
m=N/8
No
Yes
ad2=ad2+1
END

Fig. 3. The algorithm of the COMPRESSOR module.

TABLE I
DESCRIPTION AND VALUES OF THE MARKER BITS

Bit	Bit description	Marker bit values
Bit 0	the first byte of input data	*p*=1, *data_const*[7..0] ≠ *data_in*[7..0]
Bit 1	the second byte of the input data	*p*=0, *data_const*[7..0] = *data_in*[7..0]
Bit 2	third byte of input data	*p*=1, *data_const*[7..0] ≠ *data_in*[7..0]
Bit 3	fourth byte of input data	*p*=0, *data_const*[7..0] = *data_in*[7..0]
Bit 4	fifth byte of input data	*p*=1, *data_const*[7..0] ≠ *data_in*[7..0]
Bit 5	sixth byte of input data	*p*=0, *data_const*[7..0] = *data_in*[7..0]
Bit 6	seventh byte of input data	*p*=1, *data_const*[7..0] ≠ *data_in*[7..0]
Bit 7	eighth byte of input data	*p*=0, *data_const*[7..0] = *data_in*[7..0]

The marker fixes the sequence of bytes: if the next sorted byte does not coincide with the constant *data_const*[7..0], then the byte is written in RAM1 and "1" is written to the corresponding marker bit (*p*=1, *data_const*[7..0] ≠ *data_in*[7..0]), otherwise "0" is written (*p*=0, *data_const*[7..0] = *data_in*[7..0]). The marker bytes are written in RAM2.

IHEP accelerator complex [16]. The first stage of the encoder's job is to sort the data for a zero value. Based on the results of this comparison, a new variable-length word is formed. At the second stage of the algorithm, a channel frame is created containing two service bytes (markers) and significant bytes of channel signals. Next, the formed channel frames are loaded into the network controller. The algorithm is implemented in the form of an IP core based on the FPGA of Altera and showed good and stable results in the monitoring of the technological timer [9].

For the device under consideration (Fig.1), the encoders algorithm COMPRESSOR adds the ability to dynamically select the constant *data_const* [7..0] and double-encode the stream. Fig.3 shows the algorithm of the compressor. The main idea of the algorithm is to form two memory areas from the initial data packet: the area of sorted bytes that do not coincide in value with the *data_const* [7..0] constant (memory RAM1 with *ad1* address field) and the marker area (RAM2 memory with address field *ad2*).The marker fixes the sequence of bytes: if the next sorted byte does not coincide with the constant *data_const* [7..0], then the byte is written in RAM1 and "1" is written to the corresponding marker bit (Tab.1), otherwise "0" is written. Markers are stored in RAM2.

The variable *p* arrives at the serial input of the Shift Register and takes two values *p*={0,1}. Thus, after each comparison, a marker code is generated in the shift register.
The number of markers depends on the number of bytes in the source data packet.

IV. RESULTS AND DISCUSSION

The design of the considered device is implemented on FPGA family Cyclone IV. Approximately 6000 logical elements (that is <23% of the device capacity) and three banks of built-in memory M9K (256x32) are used. The maximum incoming serial data frequency is 120 MHz. It is possible to optimize the project in order to increase the processing frequency, but it is necessary to match the processing frequency and the output of the network controller output.

In addition, if the network controller transmits data from several sorting and data compression devices, it is necessary to dynamically control the processing in order to efficiently load the controller. The processing of input data goes in parallel in the SORTER and COMPRESSOR modules. When processing the first packet in SORTER, the COMPRESSOR module does not work. The maximum computed compression rate in the COMPRESSOR module when using the presented algorithm is about 20. The algorithm works most effectively on arrays of homogeneous data (for example, data from the CMOS matrix of the video camera).

V. Conclusion

The Experienced operation of the sorting and data compression device has shown stable operation in the task of data processing from a multichannel system of high-speed sensors with a total incoming traffic volume of about 7 Gb/s [17]. It should be noted that the task of determining the frequency distribution of input data is used in many applied problems: the construction of histograms in the recognition of images, the development of recording electronics for physical experiments, etc., which increases the efficiency of the device.

The disadvantage of the COMPRESSOR module work of this device is that at the output in the worst case there can be a data array which size exceeds the input one. The modules using the RLE compression method have the same flaw.

References

[1] V.Komarov, G.Antonichev, L.Kim, V.Kokovin, N.Krotov, V.Kuznetsov, Yu.Milichenko, N.Radomsky, and V.Voevodin, “Modernization of U-70 general timing system”, Proceedings of ICALEPCS-2005, Geneva, Switzerland, October 10-14, 2005.

[2] Sutikno T., Jidin A., and Basar M. F. Simple realization of 5-segment discontinuous svpwm based on FPGA // International Journal of Computer and Electrical Engineering. Vol. 2. No. 1. Feb. 2010

[3] P. P. Ray, “Internet of robotic things: Concept, technologies, and challenges,” IEEEAccess, vol. 4, pp. 9489–9500, 2016.

[4] R. Marcelino, H. Neto, and J. Cardoso. Sorting Units for FPGA-Based Embedded Systems. In Distributed Embedded Systems: Design, Middleware and Resources, volume 271 of IFIP International Federation for Information Processing, pages 11–22. Springer Boston, 2008.

[5] J. Ouyang, H. Luo, Z. Wang, J. Tian, C. Liu, and K. Sheng, “Fpga implementation of gzip compression and decompression for idc services,” in Field-Programmable Technology (FPT), 2010 International Conference on. IEEE, 2010, pp. 265–268

[6] J. A. Pérez-Celis, J. Martínez-Carranza, A. Morales-Reyes, C. Feregrino-Uribe, and R. Cumplido, “An fpga architecture to accelerate the burrows wheeler transform by using a linear sorter,” in Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International. IEEE, 2016, pp. 156–161.

[7] V.A. Kokovin and A.N. Sytin The processing of information from sensors in intelligent systems / Journal of Physics: Conference Series, 2017. Vol. 803, N. 1, 012075. doi:10.1088/1742-6596/803/1/012075

[8] D.Vatolin, A.Ratushnjak, M.Smirnov, and V.Jukin, “Metody szhatija dannyh. Ustrojstvo arhivatorov, szhatie izobrazhenij i video”-M.: DIALOG-MIFI, 2003.-p.384.

[9] Kokovin V., Uvaysov S., and Uvaysova S. Lossless Compression Algorithm For Use In Telecommunication Systems, Control and Communications (SIBCON), IEEE, 2016 DOI: 10.1109/SIBCON.2016.7491839

[10] http://www.cast-inc.com/ip-cores/images/jpeg-c/index.html

[11] http://jpeg.org/

[12] https://www.altera.com/

[13] http://www.xilinx.com/

[14] Donald E. Knuth. The art of computer programming, volume 3: (2nd ed.) Sorting and Searching, Addison Wesley, 1998, ISBN: 0-201-89685-0

[15] Joshua Vasquez, “Sort faster with FPGAs”, https://hackaday.com/2016/01/20/a-linear-time-sorting-algorithm-for-fpgas/#comments

[16] S. Ivanov, "Accelerator complex u70 of ihep: present status and recent upgrades", Proceedings of RuPAC-2010, Protvino, 2010.-pp. 27-31.

[17] A.Yu. Kalinin, V.A. Kokovin, V.I. Kryshkin, and V.V. Skvortsov " An absolute intensity beam monitor", Instruments and Experimental Techniques, 2016, Vol. 59, No. 4, pp. 536–538