Computations of cross-correlation functions on a single board Raspberry Pi computer

V A Faerman¹, M P Shvetsov² and A V Tsavnin²

¹ Tomsk State University of Control Systems and Radioelectronics, Tomsk, Moscow Technological Institute, 40 Lenina ave., Tomsk, 634050, Russia
² Tomsk Polytechnic University, 30 Lenina ave., Tomsk, 634050, Russia

E-mail: fva@fb.tusur.ru

Abstract. The paper discusses the implementation of correlation algorithm for time delay estimation on a Raspberry Pi single-board computer. The implemented correlation algorithm is based on Fourier transform. In the course of the study, we applied two alternative solutions for the software implementation of discrete Fourier transform. The first solution stands on FFTW library and uses general-purpose quad-core ARM Cortex A53 processing unit. The alternative method uses VideoCore IV graphic processing unit and is implemented via firmware GPU_FFT library. We have performed a computational experiment on a Raspberry Pi 3B to determine which solution is more preferable for the implementation of correlator. After a comparative study we figured out that estimated processing time is highly dependent on computations parameters and input signals. For small FFT window sizes CPU is proved to be a preferable option. However, for large FFT windows GPU allows significantly accelerating the computations. At some point, you can achieve even better performance by using batching and GPU for direct FFT and CPU for inverse FFT. According with the results, we have concluded that both alternatives have their own potential advantages and particular drawback. We also establish, that Raspberry Pi 3 B computer with HiFiberry extension can be used as a real-time correlator for audio signals.

1. Introduction

In recent decades, there has been a pertaining tendency to expand the scope of practical application of computer technologies. This is due to the increasing computing power of devices, as well as their low cost and pervasive availability. The appearance of single-board computers in the previous decade and their further development made it possible to transfer software for signal processing—traditionally implemented on personal computers—to mobile platforms as single-board computers [1].

The paper describes the problem of software implementation of a digital correlator on a Raspberry Pi 3B single-board computer. The choice fell on this platform due to its availability and architectural features that allow using the graphics coprocessor to calculate spectral transformations [2]. Within the framework of the study, the performance time estimation of the device when operating in the simple correlator mode were made, as well as the prospects of using the device for the implementation of more complex algorithms of correlation analysis were concerned. Special focus is on discussion of the possibility to use the device for real-time correlation signal processing.

2. Materials and Methods

The main goal of the study is to implement a digital correlator on a Raspberry Pi 3B single-board computer and to experimentally evaluate its performance. To achieve the goal, we have to solve the following tasks:
• implement an algorithm for calculating the signals cross-correlation function based on the Short Time Discrete Fourier Transforms (DFT);
• integrate into the program the implementations of DFT that use various processing units on the board;
• perform a series of computational tests to determine the performance of the computer in different processing modes and for different durations of the input signals;
• interpret the results of the tests and discuss the prospects of using the device in the systems of correlation processing of audio signals.

2.1. Real-time correlation analysis
Correlation processing is one of the most effective tools for solving the practical Time Delay Estimation (TDE) problem described in [3]. TDE, in turn, is an important technique to solve a wide range of problems associated with determining the location of the remote object producing and broadcasting signal. It should be noted that the correlation analysis is not the only TDE method, but it is often considered the most versatile and effective [4, 5].

A detailed description of the algorithm for calculating the cyclic correlation using the convolution theorem is given in the corresponding section in [6]. When processing real-time data, as well as analyzing long-term recordings, segmentation for the correlator’s inputs is applied, followed by averaging the output for each segment. A detailed mathematical description of this problem was proposed in [7]. The functional diagram of the simple device for correlation signal processing is in figure 1.

Figure 1. Functional diagram of the correlator. Following designations are used: FFT – forward fast DFT; DFT−1 – inverse fast DFT; [·]* - complex conjugation operator; X – elementwise multiplication operator; + - elementwise addition operator.

In accordance with figure 1, data at the input of the correlator comes from a * .wav - file stored in the SD card of the computer. At first, we parse the file and fill the buffer with data. Then we form a data segment and calculate its correlation. After the first segment, we apply the same procedure for every consequent segment comprising the data file. The result of each iteration of the computation cycle is the modification of the data in the output buffer. After each iteration of the algorithm, the data from the output buffer can be displayed using the visualization tools or saved in external memory as a file.

The most computationally expensive operations are forward and inverse FFTs. So the execution time of the correlation function calculation cycle is largely determined by the execution time of these transforms [8].

2.2. Hardware
The study used a common Raspberry Pi 3B single-board computer, equipped with a sound interface HiFiberry DAC + ADC as HAT [9]. The hardware composition of the device is shown in fig. 2.
The computational capabilities of the Raspberry Pi 3B are determined by the SoC solution BCM2837 [10]. At the heart of this Broadcom chip is an ARM architecture. The crystal incorporates a cluster of four Cortex A53 processor cores [10], as well as a VideoCore IV graphics coprocessor [11] based on two DSP cores. The calculation of forward and inverse FFTs can be implemented on the Raspberry Pi 3B using either a CPU at 1.2 GHz or massively parallel computations on a GPU at 0.4 GHz.

The use of the GPU for FFT seems a priori preferable due to the high degree of parallelism inherent in the Cooley-Tukey algorithm [8]. However, it is not really always the case. The communication between CPU to the GPU is quite time-consuming and takes about 0.1 ms, according to the estimates in [12].

2.3. Software
The widely used FFTW and GPU_FFT libraries were used as a basis for the implemented software. The FFTW library is designed to perform transformations using a CPU and is actually an up-to-date industrial standard [13]. The specialized library GPU_FFT is offered as firmware by the Raspberry Pi vendor and is designed specifically to accelerate FFT using the Raspberry Pi graphics cores [12]. However, since both BCM2835 and BCM2837 use almost identical graphics cores, the GPU_FFT library remains applicable on Raspberry 3 B without significant rework [13].

The current versions of the libraries used in the research are FFTW 3.3.8 [14] and GPU_FFT 3.0 [15]. In order to make the software solution convenient for further research, a set of wrapper functions in the C++ language were implemented for these libraries. It resulted in somewhat unification if the input and the output data for both libraries used.

The GPU_FFT library allows unrelated data arrays to be grouped into batches immediately before calling the calculation function. It reduces the amount of time spent on context switches when processing large data sets [12]. A practical implication in our research is that when calculating the correlation, you can produce two forward FFTs with a single function call.

The software implementation of correlation processing was carried out as it described at subsection 2.1. The structure of the created software solution is shown in figure 3.
3. Results and Discussion
As stated before, the computational efficiency of the algorithm is basically determined by the efficiency of the implementation of the forward FFT and inverse FFT operations. The objective of the experimental study is to obtain estimates of the computation time of correlation functions on the CPU and on the GPU for the various duration of inputs signals as well as for data segments of different sizes. An auxiliary objective is to determine the conditions under which computing on the CPU or on the GPU demonstrates the best performance.

3.1. Experimental setup
As mentioned earlier, the calculation of the correlation function requires two forward and one inverse FFT. Thus, using the developed software, four different options for calculating the correlation function can be proposed:

- all FFTs are executed on the CPU;
- all FFTs are performed on the GPU;
- forward FFTs are performed on the CPU, reverse FFTs are performed on the GPU;
- forward FFTs are performed on the GPU, reverse FFTs are performed on the CPU.

The input data originated from a software random number generator and then were encapsulated into *wav files in Mathcad Prime 4.0. In the course of the study, the identical sampling frequency $f_d = 44100$ Hz and the number of bits per sample $B = 16$ were used for all produced recordings. The total number of samples in a recording is denoted by $L$. The number of samples in each data segment is denoted by $N$. Let us note that in figure 1 $N$ corresponds to the size of the FFT window and it is actually equal to the segment size. The data was organized as shown in figure 4.
Figure 4. Data structure and segmentation. The total amount of data in the record is \( L \) samples. The recording is divided into \( Q \) equal segments, each of \( N \) samples. Since the use of the FFT algorithm imposes a constraint on the value of \( N \), it was chosen from the powers of two. In the course of the study, \( L \) ranged from 65536 to 4194304 samples, \( N \) varied from 1024 to 524288 samples.

To minimize the impact of random disturbing factors, each computational experiment was repeated ten times in a row. The two highest measurements were excluded, and the remaining were averaged.

### 3.2. Performance evaluation

Key results of computational experiments are tabulated below. All tables use the following designations:

- \( T_{\text{FFTW}} \) is the computation time on the CPU using the FFTW library;
- \( T_{\text{GPU_FFT}} \) is the GPU computing time using the GPU_FFT library without batching;
- \( T_{\text{GPU_FFT*}} \) is the GPU computation time using the GPU_FFT library using batching;
- \( T_{\text{RT}} \) is the duration of an sound recording containing \( L \) samples per channel

\[
T_{\text{RT}} = f_d \cdot L;
\]

- \( T_{\text{MIN}} \) is the minimum computing time achieved.

The ratio between the duration of the audio recording and the computation time will be called time factor and denoted as

\[
F = T / T_{\text{RT}}.
\]

If the ratio \( F < 1 \) is satisfied, then real-time computations in the current conditions could be potentially implemented. At the same time, the practical feasibility of real-time computations also depends on such factors as the size of the ADC buffer, the time of generation and routing of interrupts, and others.

Batching was carried out using the technique described in [16]. The transform function was called once for a complex sequence that was composed of two real sequences. Further processing of the results was carried out as described in the source.

Table 1 presents estimates of the time to calculate the correlation function for different \( N \) using CPU and GPU.
Table 1. Correlation function computation time on Raspberry Pi.

| $L$, samples | $N$, samples | $T_{\text{FFTW}}$, sec | $T_{\text{GPU,FFT}}$, sec | $T_{\text{GPU,FFT}^*}$, sec | $T_{\text{FFTW}} / T_{\text{GPU,FFT}^*}$, sec | $T_{\text{MIN}} / T_{\text{RT}}$ |
|--------------|--------------|--------------------------|---------------------------|---------------------------|---------------------------------|-----------------|
| 65 536 (1 486 ms) | 1024 | 0.0642 | 0.1010 | 0.0703 | 0.9131 | 0.0432 |
| | 2048 | 0.0584 | 0.0977 | 0.0695 | 0.8397 | 0.0393 |
| | 4196 | 0.0581 | 0.0992 | 0.0714 | 0.8129 | 0.0391 |
| | 8192 | 0.0666 | 0.1045 | 0.0756 | 0.8810 | 0.0448 |
| | 16384 | 0.0718 | 0.1098 | 0.0797 | 0.9009 | 0.0483 |
| | 32768 | 0.0806 | 0.1020 | 0.0741 | 1.0888 | 0.0498 |
| | 65536 | 0.1004 | 0.1185 | 0.0861 | 1.1658 | 0.0579 |
| 327 680 (7 430 ms) | 1024 | 0.3052 | 0.5312 | 0.3510 | 0.8693 | 0.0411 |
| | 2048 | 0.3052 | 0.4992 | 0.3460 | 0.8821 | 0.0411 |
| | 4196 | 0.2604 | 0.4955 | 0.3526 | 0.7384 | 0.0350 |
| | 8192 | 0.2918 | 0.5129 | 0.3696 | 0.7895 | 0.0393 |
| | 16384 | 0.3037 | 0.5221 | 0.3786 | 0.8020 | 0.0409 |
| | 32768 | 0.3175 | 0.4635 | 0.3370 | 0.9423 | 0.0427 |
| | 65536 | 0.3692 | 0.4795 | 0.3492 | 1.0573 | 0.0470 |
| 1 638 400 (37 152 ms) | 1024 | 1.6952 | 2.8331 | 1.8366 | 0.9230 | 0.0456 |
| | 2048 | 1.4883 | 2.6566 | 1.8230 | 0.8164 | 0.0401 |
| | 4196 | 1.4417 | 2.6219 | 1.8566 | 0.7765 | 0.0388 |
| | 8192 | 1.6079 | 2.7056 | 1.9450 | 0.8267 | 0.0433 |
| | 16384 | 1.6713 | 2.7497 | 1.9915 | 0.8392 | 0.0450 |
| | 32768 | 1.7343 | 2.4179 | 1.7564 | 0.9874 | 0.0467 |
| | 65536 | 1.8832 | 2.4416 | 1.7772 | 1.0596 | 0.0478 |

Table 2 presents the estimates of correlation function computing time when processing signals of duration $L$ on the CPU and GPU.

From the experimental results presented in Table 1 and in Figure 5 is evident that the calculation on the GPU demonstrates greater efficiency with an increase in $N$. This is due to the fact that with a larger volume of data segments and large FFT windows, context is switched less frequently, which has a positive impact on the overall performance.

For the same reasons, the execution time of a pair of forward FFTs with batching is lower than the execution time of it without batching. In the latter case, for each pair of forward FFTs, the transform is called twice, which requires two context switches. Aside from it, functions implemented in the GPU_FFT library has APIs focused only on working with complex sequences. So the real transforms were implemented at the wrapper level.

Evidently from figure 6, the duration of the input recording is more than 20 times longer than the time spent on its processing. This means that the computing power of the Raspberry Pi is sufficient to implement a real-time correlator for audio signal processing.
Table 2. Correlation function computation time on Raspberry Pi.

| N, samples | L, samples | \( T_{\text{FFTW}} \), sec | \( T_{\text{GPU}, \text{FFT}} \), sec | \( T_{\text{GPU}, \text{FFT}*} \), sec | \( T_{\text{FFTW}} / T_{\text{GPU}, \text{FFT}*} \), sec | \( T_{\text{MIN}} / T_{\text{RT}} \) |
|------------|------------|----------------|----------------|----------------|----------------|----------------|
| 8192       | 65536      | 0.0666         | 0.1040         | 0.0756         | 0.8810         | 0.0448         |
|            | 327680     | 0.2918         | 0.5281         | 0.3696         | 0.7895         | 0.0393         |
|            | 524288     | 0.5237         | 0.8482         | 0.5895         | 0.8884         | 0.0440         |
| 32768      | 1048576    | 1.1017         | 1.7480         | 1.2132         | 0.9081         | 0.0463         |
|            | 2097152    | 2.0194         | 3.4389         | 2.3978         | 0.8422         | 0.0425         |
|            | 3145728    | 3.1154         | 5.3611         | 3.7404         | 0.8329         | 0.0437         |
|            | 4194304    | 3.9389         | 6.8225         | 4.7620         | 0.8272         | 0.0414         |
| 65536      | 65536      | 0.0806         | 0.1032         | 0.0752         | 1.0718         | 0.0506         |
|            | 327680     | 0.3175         | 0.4620         | 0.3370         | 0.9423         | 0.0427         |
|            | 524288     | 0.5080         | 0.7309         | 0.5332         | 0.9529         | 0.0427         |
| 32768      | 1048576    | 1.0837         | 1.5151         | 1.1052         | 0.9806         | 0.0456         |
|            | 2097152    | 2.0551         | 2.9633         | 2.1615         | 0.9508         | 0.0432         |
|            | 3145728    | 3.2168         | 4.5434         | 3.3140         | 0.9707         | 0.0451         |
|            | 4194304    | 4.2619         | 5.9345         | 4.3286         | 0.9846         | 0.0448         |
| 65536      | 65536      | 0.1004         | 0.1182         | 0.0861         | 1.1658         | 0.0579         |
|            | 327680     | 0.3692         | 0.4788         | 0.3492         | 1.0573         | 0.0470         |
|            | 524288     | 0.5708         | 0.7514         | 0.5480         | 1.0415         | 0.0461         |
| 65536      | 1048576    | 1.2317         | 1.5390         | 1.1224         | 1.0973         | 0.0472         |
|            | 2097152    | 2.4182         | 3.0477         | 2.2228         | 1.0879         | 0.0467         |
|            | 3145728    | 3.6908         | 4.6246         | 3.3729         | 1.0942         | 0.0473         |
|            | 4194304    | 4.8426         | 6.0218         | 4.3919         | 1.1026         | 0.0462         |

Figure 5. Relation between processing times \((T)\) and window size \((N)\).
Figure 6. Relation between the time factor \((F)\) and window size \((N)\).

From the results of the experiment presented in table 2 and in figures 7 and 8 is evident that the duration of the analyzed recordings does not significantly affect the overall performance. However, it should also be noted that for short signals \((L\) and \(N\) are of the same magnitude), the performance is relatively low for the CPU. This is due to the microarchitectural features of the processor, namely, the instruction pipelining and pushing data into the cache. The local minima in figure 8 are also explained by the conformity between the total cache size (16 KiB of L1 and 512 KiB of L2) and the amount of analyzed data (1280 KiB for \(L = 327680\)).

Figure 7. Relation between processing times \((T)\) and signal length \((L)\) for various window sizes \((N)\): a) \(N = 8192\); b) \(N = 65536\).
3.3. GPU vs CPU

Further study is devoted to comparing the performance of computations on the GPU and the CPU. Figure 9 shows the graphs of computational acceleration \( A \) using the video core:

\[
A = \frac{T_{FFT,W}}{T_{GPU,FFT}}
\]

Figure 8. Relation between processing times \( (T) \) and signal length \( (L) \).

Figure 9. GPU acceleration rate \( (A) \) for various experiments: a) relation between \( A \) and \( N \) for various \( L \); a) relation between \( A \) and \( L \) for various \( N \).
Figure 9a shows that the efficiency of using the GPU increases with increasing FFT window size. This is due to the features of the Cooley-Tukey FFT algorithm: as \( N \) increases, the parallelism of the task increases therefore massively parallel computations became more relevant. Figure 9b shows that the acceleration does not depend on the duration of the analyzed signals. The exception is those values of \( L \) for which the efficiency of computations on the CPU is maximum due to the conformity of the input data volume and the cache size.

In order to choose the optimal computing strategy, the contribution to the processing time from the use of the CPU and GPU was estimated independently for the forward and inverse FFT. Table 3 summarizes the results of the experiments. The table uses the following notation: to the left of the "|" symbol is the implementation of the forward FFT is indicated, on the right is the inverse FFT.

**Table 3.** Correlation function computation time on Raspberry Pi.

| \( L \), samples | \( N \), samples | \( T_{\text{FFTW}} \) sec | \( T_{\text{GPU,FFT}} \) sec | \( T_{\text{FFTW}} \) | \( T_{\text{GPU,FFT}} \) |
|-----------------|----------------|----------------|----------------|----------------|----------------|
| 524 288 (11 888 ms) | 1024 | 0.4890 | 0.5090 | 0.5446 | 0.5614 |
| | 2048 | 0.4310 | 0.4753 | 0.5110 | 0.5535 |
| | 4196 | 0.4173 | 0.4774 | 0.5081 | 0.5638 |
| | 8192 | 0.4647 | 0.5237 | 0.5419 | 0.5895 |
| | 16384 | 0.4816 | 0.5362 | 0.5505 | 0.6026 |
| | 32768 | 0.5080 | 0.5206 | 0.5228 | 0.5332 |
| | 65536 | 0.5853 | 0.5708 | 0.5626 | 0.5480 |
| | 131072 | 0.7053 | 0.6816 | 0.5923 | 0.5682 |
| | 262144 | 0.8034 | 0.7440 | 0.6660 | 0.6048 |
| | 524288 | 0.8739 | 0.8197 | 0.7197 | 0.6846 |

| 2,097 152 (47 554 ms) | 1024 | 2.1837 | 2.2365 | 2.3511 | 2.3692 |
| | 2048 | 1.9140 | 2.0861 | 2.1994 | 2.3446 |
| | 4196 | 1.8543 | 2.0374 | 2.1107 | 2.3191 |
| | 8192 | 2.0194 | 2.2385 | 2.2860 | 2.3978 |
| | 16384 | 1.9902 | 2.2054 | 2.2394 | 2.4111 |
| | 32768 | 2.0551 | 2.1111 | 2.1350 | 2.1615 |
| | 65536 | 2.4182 | 2.3350 | 2.3188 | 2.2228 |
| | 131072 | 2.9096 | 2.8055 | 2.3826 | 2.2722 |
| | 262144 | 3.2051 | 2.9391 | 2.5591 | 2.2882 |
| | 524288 | 3.2025 | 3.0367 | 2.5164 | 2.3292 |

To quantify the effect on performance, the following measure was introduced:

\[
\Delta = 100\% \cdot \frac{(T_{\text{FFTW}} - T_{\text{GPU,FFT}})}{T_{\text{MIN}}},
\]

which was applied independently for forward and backward FFT. The graphs of the characteristics \( \Delta_{\text{FRW}}(N) \) and \( \Delta_{\text{INV}}(N) \) obtained in this way are shown in figure 10.
Figure 10. Processing time difference on CPU and GPU normalized by processing time for various window sizes ($N$): a) for forward FFT; b) for inverse FFT. Positive values mean that GPU is outperforming CPU and vice versa.

A number of observations can be made in figure 10. Firstly, we can confirm the previously determined position about the advantages of GPUs for large window sizes ($N > 65536$). Secondly, the contribution of the forward FFT to the total computation time is somewhat larger than the contribution of the inverse FFT. It can be explained by the fact that although two transformations of real sequences are equivalent to one transformation of a complex sequence, their implementation requires preprocessing the input arrays and post-processing the results. Each of these operations requires arithmetic operations on $N$ complex array elements. Thirdly, most likely, when processing long-term signals, the use of the GPU provides some advantages; however, experimental results are not sufficient to support this conclusion.

4. Conclusion

The article describes the correlator implementations based on the Raspberry Pi 3B single-board computer. The core operation of the processing algorithm implemented on the computer is calculating the correlation function.

The studied alternatives differ in the way they perform spectral Fourier transformations. One of the methods is based on the use of a quad-core ARM Cortex A53 CPU and the FFTW library. Another method uses massively parallel computations using VideoCore IV GPU and the GPU_FFT library. To compare each of the options, we carried out computational experiments on signal processing of stereo signals, simulating the conditions of using a Raspberry computer as a TDE unit in a leak detector [10]. The provided results prove that common single board computers could be used in some engineering applications as vibration diagnostics, discussed in [17], or as a simple alternative to FPGA coder, described in [18].

Considering the Raspberry Pi 3B as a platform for the implementation of a device for locating pipeline leaks by assessing the time difference of arrival of the acoustic emission signal, the following conclusions can be drawn. In the course of the experiments, the computing power of the Raspberry Pi 3B single-board computer is proved to be sufficient for the implementation of the real-time correlator. Practically it means that creating a correlating device on that basis is highly possible. Further research by the authors will be aimed at implementing the processing of data continuously received through the Hifiberry audio interface.
Despite the fact that the use of the GPU did not result in a dramatic increase in performance when using windows less than 65536 samples, its use is still potentially beneficial for implementation of the time-frequency correlator proposed in [19]. The most computationally expensive operation is the inverse FFT for multiple frequency intervals, which can be effectively solved by batching and massively parallel computations on the GPU. This will provide a high acceleration even with relatively small sizes of the transform window. The latter will open the possibility of implementing a time-frequency correlator on a Raspberry computer operating in a soft real time.

References

[1] Zubarev Yu B, Vityazev V V and Dvorkovich V P 1999 Digital signal processing as a real time informatics [Cifrovaja obrabotka signalov — informatika real'noho vremeni]. Cifrovaja obrabotka signalov 1-3 5-17 (in Russian)
[2] Gajjar R 2015 Raspberry Pi Sensors (Mumbai: Packt Publishing)
[3] Carter G C 1985 Time delay estimation In Urban H G (eds) Adaptive Methods in Underwater Acoustics (Springer)
[4] Juang B H and Chen T 1998 Highlights of statistical signal and array processing IEEE Signal Processing Magazine 15(3) 21-64
[5] Faerman V A and Avramchuk V S 2020 Comparative study of basic time domain time-delay estimators for locating leaks in pipelines International Journal of Networked and Distributed Computing 8(2) 49
[6] Ifeachor E and Jervis B 2001 Digital Signal Processing: A Practical Approach (2nd Edition) (Prentice Hall)
[7] Faerman V A and Avramchuk V S 2020 Algorithms for real-time leak noise correlator [Algoritmichesko e obespechenie korrelyacionnogo techeiskatelja, funkcionirujushchego v rezhime real'noho vremeni] Doklady TUSUR 23(1) 20-26 (in Russian)
[8] Avramchuk V S, Luneva E E and Cheremnov A G 2014 Optimization of calculating time-frequency correlation function on the CPU Proc. of Int. Conf. on Mechanical Engineering, Automation and Control Systems (16-18 October, Tomsk, Russia) IEEE
[9] HiFiBerry DAC+ADC 2020 Datasheet for HiFiBerry DAC+ADC. Retrieved from: https://www.hifiberry.com/docs/hardware/using-dynamic-microphones-with-the-dac-adc/
[10] Van Loo G 2014 ARM Quad A7 core datasheet (rev. 3.4). Retrieved from: https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2836/QA7_rev3.4.pdf.
[11] VideoCore IV 3D 2013 VideoCore IV 3D. Architecture reference guide. Retrieved from: https://docs.broadcom.com/doc/12358545
[12] Holme A 2014 Accelerating Fourier transforms using the GPU. Raspberry Pi Blog. Retrieved from: https://www.raspberrypi.org/blog/accelerating-fourier-transforms-using-the-gpu/
[13] Qihao H, Weaver V and Segee B 2018 Comparing power and energy usage for scientific calculation with and without GPU acceleration on a Raspberry Pi Model B+ and 3B Proceedings of Int'l Conf. Internet Computing and Internet of Things (July 30-August 2, Las Vegas) IEEE pp 3-9
[14] FFTW 2020 FFTW library v. 3.3.8 Retrieved from: http://www.fftw.org/
[15] GPU_FFT 2020 GPU_FFT library v. 3.0 Retrieved from: http://www.aholme.co.uk/GPU_FFT/Main.htm
[16] Matusiak R 2001 Implementing Fast Fourier Transform Algorithms of Real-Valued Sequences with the TMS320 DSP Platform, Texas Instruments application report (SPRA291). Retrieved from: https://www.ti.com/lit/an/spra291/spra291.pdf
[17] Avramchuk V S, Kazmin V P, Faerman V A and Le V T 2017 The time-frequency method of signal analysis in internal combustion engine diagnostics Journal of Physics: Conference Series 803(1) 012012
[18] Mytsko E A, Malchukov A N, Zuev I V, Ryzhova S E and Kim V L 2017 FPGA design of the fast decoder for burst errors correction Journal of Physics: Conference Series 803(1) 012015
[19] Avramchuk V S, Yefremov A A and Goncharov V I 2014 New calculation method of correlation function applied to leak localization Proc. of Int. Conf. on Mechanical Engineering, Automation and Control Systems (16-18 October, Tomsk, Russia) IEEE