Сучасний стан наукових досліджень та технологій в промисловості. 2025. № 3 (33)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

UDC 681.326
DOI: https://doi.org/10.30837/2522-9818.2025.3.189

O. SHKIL, O. FILIPPENKO, D. RAKHLIS, I. FILIPPENKO, V. KORNIIENKO

OPTIMIZATION OF SOFTWARE CODE FOR HIGH-LEVEL SYNTHESIS DURING HARDWARE IMPLEMENTATION OF THE COMPUTATIONALLY-LOADED ALGORITHMS

The **subject matter** of the work is the impact of code optimization methods of highly intensive algorithms, used in digital signal processing, on hardware costs and performance when implemented on different platforms. The goal of the work is to conduct a comparative analysis of the impact of the effects of three C-code optimization approaches: loop unrolling, switching to fixed-point arithmetic, and their combinations, on performance and hardware costs when implementing matrix multiplication, fast Fourier transform, and wavelet transform algorithms using high-level synthesis (HLS) tools on system-on-chip (SoC) platforms, personal computers (PCs), and single-board computers. The following **tasks** were solved in the article: implementation of highly intensive algorithms based on selected hardware platforms and using HLS; comparison of execution time of algorithms with and without different optimization methods; comparison of hardware costs for algorithms' implementations with and without different variants of optimization; formulate conclusions about the impact of different C-code optimization methods on performance and hardware costs on different target platforms. The following **methods were** used: C/C++ code optimization methods, diagnostic experiments using high-level synthesis tools to implement digital signal processing algorithms on the selected hardware platform, and statistical data collection using Python. The following **results** were obtained: for algorithms based on arithmetic operations, code optimization provided up to 30% reduction in execution time on ARM platforms. For algorithms based on the Fourier transform, complex optimization reduced execution time by up to 90% on processor devices. For programmable logic (FPGA), none of the optimization methods provided a significant execution acceleration. However, the transition to fixed arithmetic reduced hardware costs by 40–80% regardless of the algorithm type. **Conclusions**. The choice of a C code optimization strategy significantly impacts the efficiency of algorithm implementation on processor architectures. In contrast, optimizing the data types used plays a key role for FPGAs. In contrast, for FPGAs, optimizing the data types used plays a key role.

Keywords: embedded systems; high-level synthesis; C code optimization; System-on-Chip.

Introduction

In modern embedded systems, in particular, in high-performance solutions based on system-on-chip (SoC) technology, optimization of algorithms implemented in a high-level programming language is becoming increasingly important in order to efficiently use available hardware resources. In the process of developing such systems, an important task is to achieve a balance between computing speed, hardware resource consumption, and data processing accuracy.

In digital signal processing systems (DSP), algorithms that have a high computational load attract special attention. Such algorithms include matrix multiplication, fast Fourier transform (FFT), and wavelet transform. These algorithms have numerous applications ranging from computer vision and image processing to telecommunications and real-time signal analysis.

These algorithms are considered computationally intensive due to their complexity and scale of data processing. For example, matrix multiplication in the classical version is characterized by computational complexity $O\left(N^2\right)$, which leads to an exponential increase in the number of operations as the size of the input matrices increases. Even the use of optimized variants, such as the Coppersmith-Winograd method with a complexity of about $O\left(N^{2.5}\right)$, does not eliminate the significant burden on computing resources. The fast Fourier transform, which is widely used in spectral analysis, has a complexity of $O\left(N \cdot \log\left(N\right)\right) Nlog\left(N\right)$ but performs a large number of operations with complex numbers, which creates an additional load on arithmetic units. Similarly, the wavelet transform involves multilevel signal decomposition, which is implemented by successive iterations and requires efficient memory and data flow management, especially in real time.

Implementation of such algorithms in high-level synthesis environments, where the functionality is described in C/C++, opens up wide opportunities for applying various optimization methods. Among the most common methods are looping, switching to fixed-point arithmetic, and a combination of these. Such optimization techniques can reduce the logic depth, improve

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2025. No. 3 (33)

parallelization, reduce power consumption, and improve performance, especially when implemented on field-programmable gate arrays (FPGAs), embedded ARM processors, or single-board computers.

At the same time, the effectiveness of the same optimization methods depends significantly on the computing platform. For example, optimization techniques that demonstrate a significant reduction in execution time on ARM processors may not have the same effect on FPGAs, where the nature of computing operations is different. This justifies the need for a detailed comparative analysis of the impact of software optimization methods on performance and hardware costs for each type of implementation platform.

It is also important to consider that the effectiveness of different optimization methods is closely related to the nature of the algorithm itself. In the case of predominantly arithmetic computations, optimization methods that reduce the number of unnecessary operations and optimize memory access are critical. Whereas for algorithms based on transformations, special attention should be paid to those optimization methods that affect the efficient organization of multi-level data processing and minimize delays in processing large amounts of information.

Embedded systems that are designed to perform specialized application tasks are usually characterized by limited computing power, memory capacity, and energy efficiency. Given these limitations, it is important to carefully plan and optimally utilize available hardware resources, such as embedded memory and specialized computing modules, when developing such systems.

There are two main ways to improve the performance of an embedded system, which is determined by reducing the execution time of the target algorithm: hardware and software.

The hardware way involves upgrading the computing platform, but this often contradicts the economic and energy constraints inherent in many classes of embedded systems. Instead, software optimization involves improving the runtime environment and the target program code. If the choice of the execution environment is determined by the initial technical specification, the focus is on optimizing the implementation of the algorithm itself.

Universal approaches to software optimization include the use of more efficient algorithms with minimization of redundant calculations and memory accesses, optimization of data structures, involvement of low-level program code with specialized libraries, and reduction of calculation accuracy while maintaining the required functionality. It is worth noting that the effectiveness of the proposed approaches depends on the specific problem statement and the chosen algorithm [1].

In the case of using Xilinx ZYNQ-7000 SoCs as the hardware basis, the architecture of which integrates the Processing System (PS) and Programmable Logic (PL), it is advisable to consider optimizing the software implementation in both PS and PL. This approach allows for flexible distribution of the computational load between general-purpose processors and hardware-accelerated modules.

Thus, the object of research is methods for optimizing the high-level description of computationally intensive algorithms. The subject of the study is the impact of the selected optimization methods on hardware costs and performance when implementing these algorithms on different platforms.

The purpose of the article is to compare the impact of three approaches to C-code optimization, namely loop unrolling, transition to fixed-point arithmetic, and their combination, on the efficiency of implementing matrix multiplication, fast Fourier transform, and wavelet transform algorithms using high-level synthesis tools on SoC, PC, and single-board computers.

Literature review

Paper [2] explores the potential of using Large Language Models (LLM) for automated adaptation of program code to HLS-compliant. One of the significant contributions of the researchers is the implementation of a framework for correcting program compilation errors, which is guided by the LLM and combines automatic code generation with various hardware-oriented optimization methods. According to the conclusions, an experimental evaluation on 24 applications demonstrates that the proposed framework significantly exceeds the indicators of successful adaptation of program code compared to traditional scripts and direct application of LLM.

Study [3] analyzes the potential of code optimization specific to high-level synthesis to achieve higher performance and energy efficiency compared to traditional implementations in the field of High Performance Computing (HPC). The authors propose a library of optimized primitives for high-level synthesis with an analysis of individual optimization techniques aimed at reducing memory access time, scalability of the architecture, and optimization of the execution pipeline.

ISSN 2522-9818 (print)
Сучасний стан наукових досліджень та технологій в промисловості. 2025. № 3 (33) ISSN 2524-2296 (online)

The analysis procedure includes the study of individual optimization techniques and their impact on high-performance hardware computers.

In [4], a new FPGA implementation of the Strassen algorithm for matrix multiplication is considered, which demonstrates significant performance advantages over traditional approaches. The authors of the paper test the algorithm implementation on Alveo U50 and U280 hardware platforms for matrix sizes up to 256 elements. They also study the effect of the matrix data type on the speed of the proposed and implemented architecture.

The authors of the study [5] consider the growing role of software components in complex heterogeneous embedded systems, where development time, flexibility, and reuse are important factors. The authors note that due to the regularity of processing multimedia and DSP applications, statically scheduled devices, in particular VLIW (Very Long Instruction Word) processors, are promising options for such systems compared to dynamically scheduled processors, such as modern superscalar General Purpose Processors (GPPs).

Paper [6] presents a promising study of the use of high-level synthesis tools for the development of hardware accelerators focused on digital signal processing tasks with an emphasis on fast Fourier transform. The authors emphasize the importance of behavioral models for achieving efficient and high-performance hardware architectures. They also analyze the proposed architecture in terms of performance and hardware costs.

The authors of [7] consider the shortcomings of traditional multiplication algorithms for very large numbers and their impact on the performance of circuits, in particular in digital filters. The authors propose the use of the Schönhage-Strassen Algorithm (SSA) to optimize the operation of filters with a finite impulse response (FIR), which are the main components of many DSP processors.

The study [8] proposed an efficient hardware architecture for the two-dimensional discrete Fourier transform (SDFT), which is focused on minimizing the use of hardware resources and optimizing performance for real-time tasks. The proposed architecture significantly reduces the need for hardware resources compared to the Park method.

The authors of [9] consider the process of code optimization at the compilation stage. The authors emphasize that code optimization tries to improve the target code without changing its output or causing side effects. The paper describes classical optimization techniques, such as eliminating common nested expressions, removing dead code, and collapsing constants, which are widely used in compilers. The article also analyzes the challenges faced by compiler developers and the latest code optimization methods for such systems.

The study [10] considers the improvement of the key encapsulation mechanism based on a ring with a restricted polynomial of degree N (NTRU-KEM). The authors point out that, despite its high reliability, NTRU-KEM has increased storage and computing requirements compared to classical cryptography, which leads to significant memory consumption and performance degradation. This paper proposes a hardware-software co-design approach that allows for the customization of computations to meet variable requirements for running time and number of iterations. The main contribution of the work is the development of a new hardware acceleration technique focused on optimizing the use of the SoC bus. Experimental results confirm the effectiveness of the proposed approach.

Paper [11] presents the POLSCA compilation system, which improves the process of automatic optimization of nested affine cycles for high-level synthesis. The system performs a preliminary decomposition of the project to balance code complexity and parallelism, and automatically modifies memory interfaces to better support HLS tools. This avoids the need to manually add directives and simplifies integration. Experiments on the Polybench/C benchmarks have shown that the approach provides an average speed up of 1.5 times.

Study [12] proposes a Design Space Exploration (DSE) method for high-level synthesis that takes into account information from the scheduling stage. The authors use this information to improve the efficiency of a genetic algorithm implemented on the basis of their own HLS tool. The proposed approach allows finding more Pareto-optimal solutions compared to methods that do not take into account scheduling data. The method outperforms the traditional Genetic Algorithm (GA) approach, reducing the execution time by a factor of four and achieving 95.7% of optimal solutions while exploring only 0.18% of the Pareto space.

The authors of [13] also propose an approach to studying DSE in high-level synthesis based on Contrastive Learning (CL) to determine the dominance relationship between projects. Unlike traditional methods that evaluate absolute values of productivity and cost, the proposed method classifies projects by their relative efficiency. The CL-method is integrated with three

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2025. No. 3 (33)

modern DSE approaches, which significantly reduces the number of syntheses without losing the quality of the results. Comparative experiments have shown the advantage of classification methods over regression methods in predicting Pareto dominance.

The study [14] presents a method for automatic optimization of HLS designs based on domain-specific knowledge and does not require quality assessment models or metaheuristics. The proposed approach automatically selects efficient directive configurations for source code, which is especially useful for users without hardware experience. The method has been tested on more than 100 examples from benchmarks and GitHub running on Xilinx ZCU104 FPGA. On average, we achieved a speedup of ×7.2 compared to manually optimized projects and ×1.35 compared to overprovisioning methods. Comparison with modern DSE methods showed similar quality of results, but at a speed that exceeds traditional approaches by 100 to 1000 times.

Paper [15] presents Stream-HLS, a new methodology and framework for automated creation of high-performance hardware architectures based on C/C++ or PyTorch code. The solution is built on the Multi-Level Intermediate Representation (MLIR) infrastructure and addresses key HLS issues, including support for multi-core applications, global loop scheduling, and graph pipelining. Stream-HLS automatically generates an optimized dataflow architecture and host code for FPGAs using an analytical performance model. Experiments on standard and real-world problems (Transformers, Convolutional Neural Networks (CNN), Multilayer Perceptrons (MLP)) demonstrate an acceleration of up to 79.4× compared to state-of-the-art solutions and up to 10.6× compared to manual optimization.

The authors of [16] propose an approach to the design of Application-Specific Instruction-set Processors (ASIP), fully implemented at the level of the ANSI C programming language standard using HLS tools. The authors combine the description of the Central Processing Unit (CPU) and the hardware accelerator to synthesize the entire system together, which reduces the area and power consumption due to optimal resource allocation. HLS also provides the ability to automatically generate different ASIP variants with different balance between performance, area, and power consumption. The results show that the proposed approach outperforms traditional methods, providing lower overhead, higher performance, and a significant reduction in power consumption compared to a basic RISC-V processor and state-of-the-art analogs.

The presented review covers a wide range of modern research in the field of high-level synthesis, optimization of hardware architectures, and intelligent design automation methods that are relevant for high-performance computing systems, embedded solutions, and digital signal processing.

Therefore, the issue of optimizing the code of highly loaded algorithms used in digital signal processing in order to reduce hardware costs and increase performance when implemented on different target platforms remains relevant.

Problem statement

It is necessary to analyze how different ways of optimizing the C-code of the algorithm description in the high-level synthesis of embedded systems on a chip affect the performance and hardware costs of the resulting devices. The impact of the proposed optimization methods will also be analyzed for classic PCs and single-board computers.

Different types of computationally intensive algorithms will be used as examples:

– matrix multiplication (classical algorithm, computational complexity of which is $O\left(n^{(3)}\right)$, Winograd algorithm, computational complexity of which is $O\left(n^{(2,5)}\right)$;

– series decomposition using harmonic and wave functions (fast Fourier transform (FFT transform, computational complexity of which is $O(NlogN)$) and wavelet transform, computational complexity of which is $O(NlogN)$.

These algorithms are analyzed for objects of different sizes.

Taking into account the tasks, we introduce the following notations:

1. $A=\{A_1, A_2, ..., A_m\}$ – a set of algorithms, where eachA_i is a computationally intensive algorithm (e.g., matrix multiplication, FFT, wavelet transform).

2. $D=\{D_1, D_2, ..., D_n\}$ – set of objects, where each D_j is the size of the data volume for a particular algorithm. For matrix multiplication algorithms, these are matrix sizes, for FFT the length of the

Сучасний стан наукових досліджень та технологій в промисловості. 2025. № 3 (33)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

transformation, for wavelet transform the size of the input buffer and the level of decomposition.

3. $O=\{O_1,O_2,...,O_k\}$ – a number of ways to optimize C code (looping, fixed point, combined optimization).

4. $P=\{P_1,P_2,...,P_l\}$ – a variety of platforms on which the algorithms will be tested, including classic PCs, single-board computers, and SoCs.

Thus, the implementation of the algorithm A_i on the object D_j with the optimization O_k on the platform P_l can be described as:

$$R_{ijkl}=Run\left(A_i,D_j,O_k,P_l\right). \quad (1)$$

The objective of the study **is to** investigate the impact of different ways to optimize O_k on performance and hardware costs when implementing $A_1,A_2,\ldots,A_m$ algorithms on platforms $P_1,P_2,...,P_l$.

The classical ways of optimizing C/C++ code were chosen, namely, loop unrolling, transition to fixed-point arithmetic, and a hybrid version that combines both approaches. The implementation of fixed-point arithmetic was performed using the fixed_16_16 type. This data type is used in many implementations of digital signal processing libraries and highly optimized linear algebra libraries.

Let's introduce a normalized indicator of reducing the time T of executing computational algorithms for different ways of optimizing program code $\left(T_{opt}/T_{base}\right)$. In this case, the average normalized index of the execution time of the algorithm K_i for objects of different sizes is calculated as follows:

$$K_i=\sum_{j=1}^{m}\frac{T_{opt}}{T_{base}}\Big/m, \quad (2)$$

where m is the number of objects under consideration; T_{opt} is the index of reducing the execution time of computing algorithms for different methods of optimizing program code in microseconds (μs); T_{base} is the execution time of the algorithm without optimization in microseconds (μs).

The averages of the algorithm execution time for different optimization methods are calculated as follows:

$$M_j=\sum_{i=1}^{n}K_{ij}\Big/n, \quad (3)$$

where n is the number of optimization considered algorithms.

The mathematical component of the algorithms under consideration

Consider the matrix multiplication algorithm with complexity $O\left(N^3\right)$, where N is the size of the input data in the units required to represent it. It is based on the classical formula for multiplying two matrices $C=A\cdot B$, whose elements are calculated by the formula:

$$C_{ij}=\sum_{k=1}^{m}A_{ij}\cdot B_{kj}, \quad (4)$$

where C is the result matrix of size $n\times p$; A is the result matrix of size $n\times m$; B is the result matrix of size $m\times p$.

Let's consider the Coppersmith-Winograd matrix multiplication algorithm, which is an improved modification of the classical matrix multiplication algorithm aimed at reducing the number of multiplication operations. Its key idea is to pre-calculate auxiliary values, which allows you to optimize and speed up the multiplication, especially when working with large square matrices.

For matrices of size $n\times m$ and $m\times p$, the following iterations of the algorithm are performed:

1) for each row of matrix A, the auxiliary matrix rowFactor is calculated

$$rowFactor[i]=\sum_{k=1}^{m/2}A[i][2k-2]\cdot A[i][2k-1]; \quad (5)$$

2) for each column of matrix B, the auxiliary matrix *colFactor* is calculated:

$$colFactor[j]=\sum_{k=1}^{m/2}B[2k-2][j]\cdot B[2k-1][j]; \quad (6)$$

3) the main multiplication is performed

$$C[i][j]=-rowFactor[i]-colFactor[j]=\sum_{k=1}^{m/2}\left(A[i][2k-2]+\cdots+B[2k-1][j]\right)\cdot\left(A[i][2k-1]+\cdots+B[2k-2][j]\right); \quad (7)$$

4) if the number of columns m is odd, an additional adjustment is performed

$$C[i][j]+=A[i][m-1]+B[m-1][j]. \quad (8)$$

Let's take a look at the Fourier transform. The Fourier transform is a mathematical operation that projects a function (signal) from the time domain to the frequency domain by decomposing it into a basis of complex exponents. It is based on the principle that

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2025. No. 3 (33)

any energy-limited function (signal) can be represented as a linear combination of harmonic functions of sine and cosine waves of corresponding frequencies and amplitudes.

In general, the complex Fourier transform is defined as

$$X(k)=\sum_{n=0}^{N-1}x(n)\cdot e^{-j\frac{2\pi kn}{N}},\quad \text{при}\quad k=0,1,2,\ldots,N-1,\quad (9)$$

where $X(k)$ is the result of the transformation in the frequency range; $x(n)$ is the input complex sequence; N is the length of the transformation; $e^{-j\frac{2\pi kn}{N}}$ is the complex exponential basis (twiddle factor).

The length of the Fourier transform determines the possible frequency resolution. That is, with a transform length of 512 and a sampling frequency of 48 kHz, according to the Nyquist-Shannon theorem, each frequency interval in the Fourier transform spectrum corresponds to a frequency bin with a step of 46.875 Hz.

Consider the wavelet transform. The Discrete Wavelet Transform (DWT) is a formalized implementation of the wavelet transform based on a discrete set of scaling and shifting coefficients that follow certain mathematical rules. This transformation allows the signal to be decomposed into a basis formed by mutually orthogonal wavelet functions. This approach is fundamentally different from the Continuous Wavelet Transform (CWT), as well as from its variations adapted to discrete time series, in particular the so-called Discrete-Time Continuous Wavelet Transform (DT-CWT).

The design of a wavelet function is based on the so-called scaling function, which defines the scaling properties of a wavelet. The requirement of orthogonality of the scaling function to its discrete translations imposes a number of mathematical conditions on it, among which the dilation equation plays a key role. This equation provides a recursive definition of the scaling function through its shifts and the corresponding filter coefficients, and is the basis for constructing orthogonal wavelet bases, such as Dobeshi wavelets.

The discrete wavelet transform is defined as

$$\Psi_x^{\Psi}\left[n,a^j\right]=\sum_{m=0}^{N-1}x[m]\cdot\Psi_j^*[m-n],\qquad (10)$$

where $\Psi_x^{\Psi}\left[n,a^j\right]$ is the wavelet coefficient for the signal $x[n]$ at scale $a(j)$ and offset n, i.e., the result of projecting the signal onto a wavelet; $x[m]$ is the input signal; $\Psi_j^*[m-n]$ is a complex-conjugate wavelet function shifted by n and scaled by j; $\sum_{m=0}^{N-1}$ is a discrete convolution of the signal with a wavelet function.

In turn, the wavelet function $\Psi_j[n]$ is defined as.

Description of the experimental part

To study the impact of different methods of optimizing computationally intensive algorithms, we selected objects of different dimensions in order to compare the execution time and hardware costs for each of the proposed implementation platforms. When measuring the execution time of algorithms on each platform, the high-resolution timer available on the platform was taken into account.

To obtain statistics on the algorithm execution time, a template function was developed that uses the timer and executes the algorithm for a specified number of iterations, and then calculates the statistics on the execution time of the selected algorithm.

The analysis of hardware costs for the PL part was performed by analyzing the post-synthesis report in Vivado after exporting the IP core from Vitis HLS to Vivado.

The set of matrices was analyzed on all platforms for square matrices of (16×16), (24×24) and (32×32) elements. The implementation of the Fourier transform was compared at the transform lengths of 512, 4096, and 8192, which is determined by the number of N-analyzing components of the FFT series, according to (9). The wavelet transform was analyzed using the db4 wave function, decomposition level 5, and signal length 256 on all platforms.

The methods of code optimization used in the study are denoted as follows:

C – basic implementation of the algorithm for the float type without optimization;

C1 – loop unrolling (loop unrolling, float type);

C2 – a basic implementation of using fixed-point arithmetic instead of the float type;

C3 – complex optimization with loop unrolling and the use of fixed-point arithmetic instead of the float type.

It should be taken into account that when using the basic implementation of the algorithm, the result of measuring its execution time is equal to T_{basic}, so for the method of optimizing the code C, the average time K_i is not calculated.

Сучасний стан наукових досліджень та технологій в промисловості. 2025. № 3 (33)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

The following algorithms will be used as algorithms for further analysis:

- Matrix $n(3)$ is a classical matrix multiplication algorithm;
- Winograd $n(2,5)$ – an optimized matrix multiplication algorithm that reduces the number of multiplication operations due to preliminary calculations;
- FFT is an algorithm based on the Fast Fourier Transform;
- Wavelet – an algorithm based on the wavelet transform using the db4 wave function and decomposition 5.

Table 1 shows the results of measuring the execution time of the considered algorithms in microseconds (μs) and the corresponding averaged K1/K2/K3 for different methods of optimizing C1/C2/C3 when implemented on a PC platform with an Apple M1 processor, with a matrix size of 32×32 elements and a Fourier transform length of 512.

Table 1. *Results of measuring the execution time of algorithms on a PC*

Algorithms	Optimization methods and their average algorithm execution time						
	C	C1	K(1)	C2	K(2)	C3	K(3)
Matrix n3	4 μs	4 μs	1	4 μs	1	3 μs	0.75
Grapes n2,5	3 μs	2 μs	0.67	3 μs	1	2 μs	0.67
FFT	16 μs	14 μs	0,86	3 μs	0,19	3 μs	0,19
Wavelet	7 μs	5 μs	0,71	6 μs	0,86	5 μs	0,71

The results shown in Table 1 demonstrate a reduction in execution time for some algorithms when applying different optimization methods: in particular, for the FFT transform, a gradual decrease in time is observed ($0.19 \le 0.19 < 0.86$, i.e., K((3) <) (K) ((2) () <) (K) (1)), and for the classical matrix multiplication algorithm (Matrix n3), there is an improvement when moving to fixed-comma (C2) and hybrid optimization (C3). Instead, for the Wavelet transform and the Winograd n2,5matrix multiplication algorithm, the optimizations did not significantly reduce the execution time on the used hardware platform.

Table 2 shows the results of measuring the execution time of the considered algorithms in microseconds (μs) and the corresponding averaged K1/K2/K(3) for different C1/C2/C3 optimization methods when implemented on the Raspberry P_i platform with an ARM Cortex A53 core for matrix sizes of 32×32 elements, FFT of 512 elements.

Table 2. *Results of measuring the execution time of algorithms on the Raspberry Pi platform*

Algorithms	Optimization methods and their average algorithm execution time						
	C	C1	K(1)	C2	K(2)	C3	K(3)
Matrix n3	406 μs	395 μs	0,97	310 μs	0,76	290 μs	0,71
Grapes n2,5	303 μs	303 μs	1	263 μs	0,87	243 μs	0,81
FFT	719 μs	718 μs	1	51 μs	0,07	48 μs	0,07
Wavelet	61 μs	61 μs	1	73 μs	1,2	73 μs	1,2

The results shown in Table 2 demonstrate a significant improvement in the performance of some algorithms on the Raspberry Pi platform due to the use of various optimization methods.

The most pronounced effect is observed for the FFT transform: the algorithm execution time decreases from 719 μs to 48 μs (respectively, K1= 1, K(3) = 0.07).

For the classical Matrix n3matrix multiplication, an improvement was also recorded – 406 μs versus 290 μs (respectively, K1= 0.97, K3= 0.71), as well as for the Winograd n2.5algorithm - 303 μs versus 243 μs (K1= 1, K3= 0.81).

Instead, for the Wavelet transform, the execution time increased from 61 μs to 73 μs (K1= 1, K3= 1.2), which indicates the inefficiency of optimization in this case.

Table 3 shows the results of measuring the execution time of the considered algorithms in microseconds (μs) and the corresponding averaged execution times K1/K2/K(3) for different methods of optimizing C1/C2/C3 when implemented on the Zynq-7000 platform, namely on its PS part, with a matrix size of 128×128 elements and a Fourier transform length of 8192.

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2025. No. 3 (33)

Table 3. *Results of measuring the execution time of algorithms on the PS part of ZYNQ*

Algorithms	Optimization methods and their average algorithm execution time						
	C	C1	K(1)	C2	K(2)	C3	K(3)
Matrix n3	35380 μs	34845 μs	0,98	28786 μs	0,81	25496 μs	0,72
Grapes n2,5	36666 μs	35818 μs	0,97	31129 μs	0,85	23466 μs	0,64
FFT	12075.6 μs	12076.1 μs	1	1550 μs	0,13	1550 μs	0,13
Wavelet	754.1 μs	755.1 μs	1	1867.5 μs	2,5	1867.4 μs	2,5

The results shown in Table 3 show the effectiveness of various ways to optimize the code of computational algorithms on the PS part of ZYNQ. In particular, for the FFT transformation, we observe a significant reduction in execution time from 12076.1 μs to 1550 μs (respectively, K1=1, K3=0.13), and for the classical matrix multiplication algorithm Matrix n(3),we see a moderate improvement (K1= 0.98, K(3) = 0.72) when switching to fixed-point calculations and hybrid optimization. The Winograd n2.5algorithm also demonstrates a slight decrease in time from 35818 μs to 23466 μs (respectively K1= 0.97, K3= 0.64), while for Wavelet transforms, optimizations proved to be ineffective – the execution time even increased (K1= 1, K3= 2.5).

Table 4 shows the results of measuring the execution time of the considered algorithms in microseconds (μs) and the corresponding averaged K1/K2/K(3) for different methods of optimizing C1/C2/C3 when implemented on the Zynq-7000 platform, namely on the PL part, with a matrix size of 24×24 elements and an FFT size of 512 elements.

Table 4. *Results of measuring the execution time of algorithms on the PL part of ZYNQ*

Algorithms	Optimization methods and their average algorithm execution time						
	C	C1	K(1)	C2	K(2)	C3	K(3)
Matrix n3	41.192 μs	40.128 μs	0,97	39.14 μs	0,95	39.54 μs	0,96
Grapes n2,5	41.118 μs	41.118 μs	1	39.54 μs	0,96	39.54 μs	0,96
FFT	1500.97 μs	1500.7 μs	1	1325.8 μs	0,88	1288.8 μs	0,86
Wavelet	324.066 μs	339.465 μs	1,05	225.97 μs	1	319.15 μs	0,98

The results presented in Table 4 show that for the FFT transformation, there is a moderate decrease in the algorithm execution time from 1500.9 μs to 1288.8 μs (respectively, K1= 1, K(3) = 0.86), which indicates a limited effect of different code optimization methods. For the classical Matrix n3matrix multiplication and the Winograd n2.5algorithm, the changes are insignificant – the time is reduced by only a few microseconds, and the improvement factors remain close to one (for example, for Matrix n^3 K1= 0.97 and K(3) = 0.96). The most noticeable improvement is observed for the wavelet transform – the algorithm execution time decreases from 324.07 μs to 225.97 μs (C2), which corresponds to a decrease in the coefficient from K1= 1.05 to K(2) = 1. However, in general, all optimization methods have a less pronounced effect in the PL part compared to the PS part.

Table 5 summarizes the time characteristics of different C-code optimization methods M1/M2/M(3) for different algorithms on the corresponding hardware implementation platforms, namely, on a PC with an Apple M1 processor, on a Raspberry P_i single-board computer with an ARM Cortex A53 core, and on the PS and PL parts of the Zynq-7000 SoC.

Table 5 *Summary of time characteristics of different code optimization methods*

Hardware platform	Matrix n3			FFT			Wavelet		
	M(1)	M(2)	M(3)	M(1)	M(2)	M(3)	M(1)	M(2)	M(3)
PC	0,88	0,81	0,63	0,93	0,3	0,3	0.64	0,85	0,69
Raspberry Pi	1	0,715	0,68	0,99	0,078	0,078	1	1,15	1,15
PS part of Zynq	0,99	0,87	0,68	1	0,083	0,083	1	2,55	2,55
PL part of Zynq	0,98	0,96	0,95	1	0,86	0,85	1,05	1	0,98

To visualize the results shown in Table 5, let's draw graphs.

Figure 1 shows the dependence of the normalized execution time of the algorithms on different ways of

Сучасний стан наукових досліджень та технологій в промисловості. 2025. № 3 (33)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

optimizing the C++ code for matrix multiplication and Fourier transform implemented on the PS part of the ZYNQ SoC, Figure 2 – on a PC with an Apple M1 processor, and Figure 3 – on the PL part of the ZYNQ SoC.

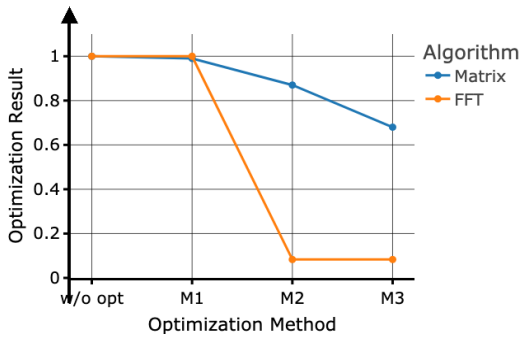

Fig. 1. Normalized execution time of algorithms for different ways of optimizing C/C++ code for the PS part of Zynq

Figure 1 shows that the optimization with the transition to fixed-point calculations on the Zynq PS platform significantly reduces the execution time of the FFT transformation (orange graph), while matrix multiplication remains almost unchanged (blue graph).

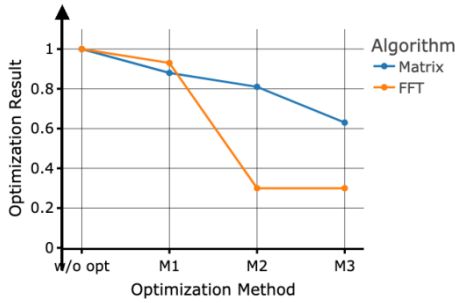

Fig. 2. Normalized execution time of algorithms for different ways of optimizing C/C++ code for PC

Figure 2 demonstrates similar behavior when implemented on a PC with an ARM M1 processor, where the transition to fixed point significantly reduces the execution time of the algorithms (orange graph).

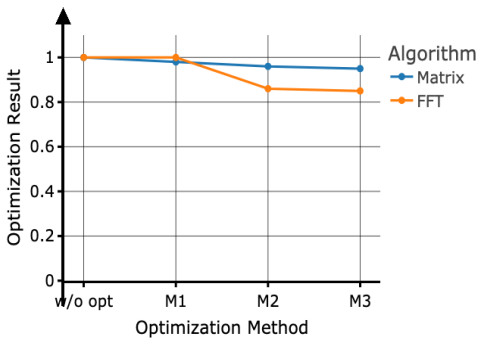

Fig. 3. Normalized algorithm execution time for different ways of optimizing C/C++ code for the PL part of ZYNQ

Figure 3 shows that when implementing algorithms on the PL part of Zynq, code optimization almost did not affect the execution time of the algorithms. A slight reduction in execution time is present for the FFT transform (orange graph).

Let's analyze the hardware costs of implementing various algorithms on the PL part of the Zynq-7000 SoC using different ways of optimizing the code for two input data options:

– option 1: matrix size 24×24 and Fourier transform length 512;

– option 2: matrix size 32×32 and Fourier transform length 4096.

The data in the cells is given as a percentage (%) of the total number of corresponding primitives available on the ZYNQ 7000, namely

– number of FPGA basic logic units (Look-Up-Table, LUT): 53200;

– number of synchronous memory elements (Flip-Flop, FF): 106400;

– number of digital signal processing units (Digital Signal Processing DSP): 220;

– number of block memory (Block Random Access Memory, BRAM): 140.

Table 6 shows the absolute (Abs column) and normalized (Norm column) hardware costs in % when implementing algorithms on the PL part of the Zynq-7000 SoC using different ways of optimizing the C1/C2/C3 code. The size of the matrix is 24×24 elements, the length of the Fourier transform is 512.

Table 7 shows the absolute (Abs column) and normalized (Norm column) hardware costs in % when implementing the algorithms on the PL part of the Zynq-7000 SoC using different ways to optimize the C1/C2/C3 code. The size of the matrix is 32×32, the length of the Fourier transform is 4096.

Figures 4 – 5 show the normalized results of hardware costs for the implementation of the Fourier transform of 512 samples and 4096 samples, respectively, on the PL part of the ZYNQ for all types of available ZYNQ resources, namely LUTs, FFs, DSP blocks and built-in BRAM.

Figures 4 and 5 show that for the implementation of the Fourier transform, the transition to fixed-point calculations significantly reduces hardware costs.

ISSN 2522-9818 (print)
ISSN 2524-2296 (online)
Innovative technologies and scientific solutions for industries. 2025. No. 3 (33)

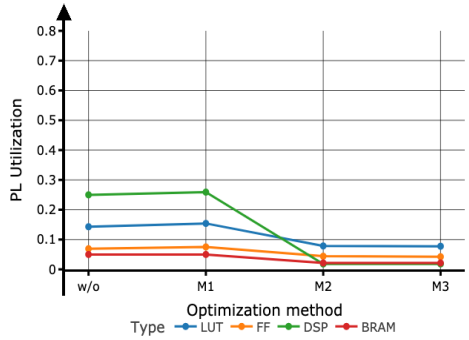

Fig. 4. Normalized results of hardware costs for the implementation of the Fourier transform with a length of 512 samples

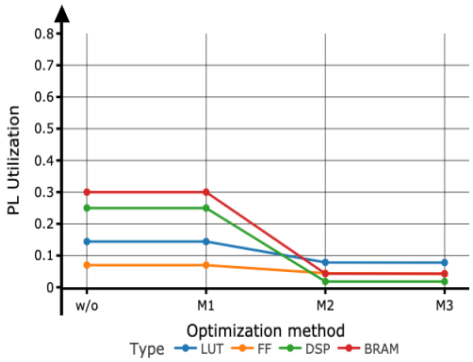

Fig. 5. Normalized results of hardware costs for implementing the Fourier transform with a length of 4096 samples

Table 6. *Hardware costs for different optimization methods on the ZYNQ PL part (option 1)*

Methods of of optimization	Resources PL ZYNQ	Algorithms							
		Matrix n3		Grapes n2,5		FFT		Wavelet	
		Abs	*Norm*	*Abs*	*Norm*	*Abs*	*Norm*	*Abs*	*Norm*
C	LUT	7933	0.149	14477	0.0272	7604	0.143	18927	0.356
	FF	10063	0.095	7133	0.161	7362	0.069	20712	0.195
	DSP	120	0.545	108	0.491	55	0.25	89	0.405
	BRAM	52	0.371	54	0.386	7	0.05	31	0.221
C1	LUT	7984	0.15	14478	0.0272	8194	0.154	26442	0.497
	FF	10149	0.095	17133	0.161	7996	0.075	29158	0.274
	DSP	120	0.545	108	0.491	57	0.259	120	0.545
	BRAM	52	0.371	54	0.386	7	0.05	45	0.321
C2	LUT	2438	0.046	3629	0.068	4178	0.079	15307	0.288
	FF	2977	0.028	3353	0.032	4684	0.044	9368	0.088
	DSP	72	0.327	108	0.491	4	0.018	30	0.136
	BRAM	52	0.371	54	0.386	3	0.021	19	0.136
C3	LUT	2438	0.046	3630	0.068	4122	0.077	15785	0.297
	FF	2977	0.028	3353	0.032	4504	0.042	9753	0.092
	DSP	72	0.327	108	0.491	4	0.018	30	0.136
	BRAM	52	0.371	54	0.386	3	0.021	19	0.136

Table 7. *Hardware costs for different optimization methods on the ZYNQ PL part (option 2)*

Methods of of optimization	Resources PL ZYNQ	Algorithms					
		Matrix n3		Grapes n2,5		FFT	
		Abs	*Normal*	*Abs*	*Norm*	*Abs*	*Norm*
C	LUT	10057	0.189	18758	0.353	7666	0.144
	FF	12630	0.119	22068	0.207	7454	0.07
	DSP	160	0.727	144	0.655	55	0.25
	BRAM	68	0.486	70	0.5	42	0.3
C1	LUT	10062	0.189	18762	0.353	7666	0.144
	FF	12713	0.119	22068	0.207	7454	0.07
	DSP	160	0.727	144	0.655	55	0.25
	BRAM	68	0.486	70	0.5	42	0.3
C2	LUT	2734	0.051	4366	0.082	4190	0.079
	FF	3277	0.031	3659	0.034	4690	0.044
	DSP	96	0.436	144	0.655	4	0.018
	BRAM	68	0.486	70	0.5	6	0.043
C3	LUT	2734	0.051	4366	0.082	4153	0.078
	FF	3277	0.031	3659	0.034	4510	0.042
	DSP	96	0.436	144	0.655	4	0.018
	BRAM	68	0.486	70	0.5	6	0.021

Сучасний стан наукових досліджень та технологій в промисловості. 2025. № 3 (33)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

Figures 6 and 7 show the normalized results of hardware costs for the implementation of the Winograd algorithm for 24×24 and 32×32 matrices, respectively, on the PL part of ZYNQ for all types of available ZYNQ resources, namely LUTs, FFs, DSP blocks, and on-chip BRAM.

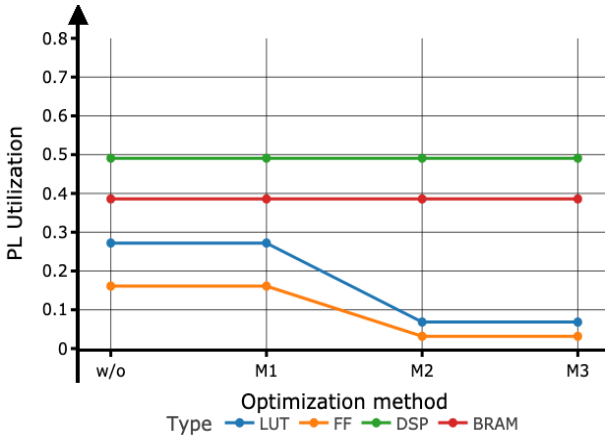

Fig. 6. Normalized results of hardware costs for implementing the Winograd algorithm for 24x24 matrices

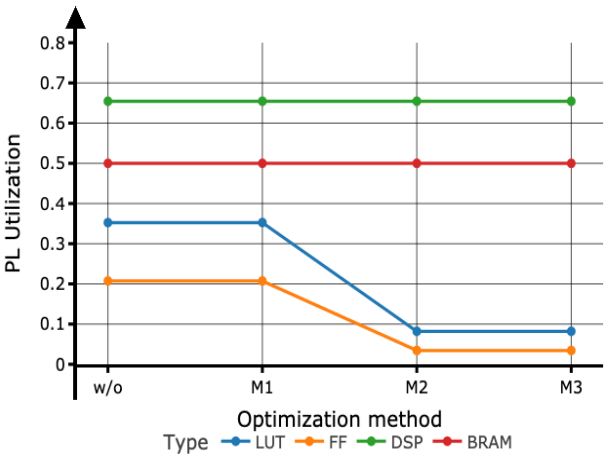

Fig. 7. Normalized results of hardware costs for implementing the Winograd algorithm for 32x32 matrices

When analyzing Figures 6 and 7, it should be noted that in the case of implementing matrix multiplication by the Winograd algorithm, a significant reduction occurs only in terms of the costs of such PL primitives as LUT and FF, while optimization does not affect DSP blocks and the use of internal BRAM memory.

Figure 8 shows the normalized results of hardware costs when implementing the Wavelet transform using the wave function db4, decomposition level 5 and input sequence length 256 on the ZYNQ PLU for all types of available ZYNQ resources, namely LUTs, FFs, DSP blocks and internal BRAM memory.

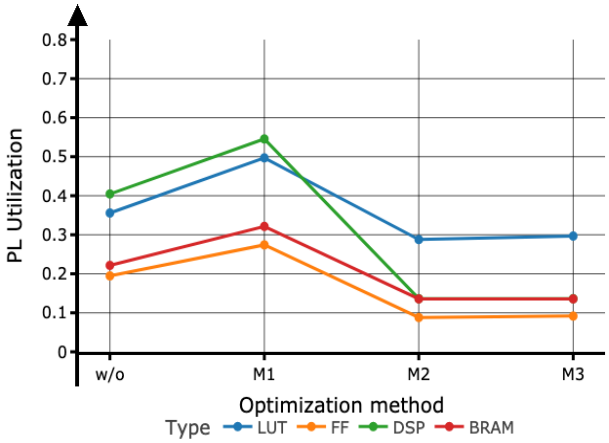

Fig. 8. Normalized results of hardware costs for Wavelet transform implementation

Figure 8 shows that optimization with loop deployment alone leads to an increase in hardware costs for implementing the Wavelet transform. However, the use of a fixed-comma switch also leads to a reduction in hardware costs.

Research results and discussion

After analyzing the research results from the point of view of the hardware platform, we can draw the following conclusions.

1. When using an Apple PC with an ARM M1 processor, all optimization methods for all algorithms work with a reduction in execution time from 30% to 70%.

2. When using a single-board computer Raspberry Pi and the PS part of the Zynq-7000 SoC, the time reduction is approximately the same (this is due to the Cortex processor cores underlying both platforms). For the Fourier transform, it is 90%, for matrices – 30%, for the wavelet transform, there is no time reduction.

3. When using the PL part of the Zynq-7000 SoC, the time reduction is observed only for the Fourier transform and not more than 15%.

From the point of view of the most highly loaded algorithms, the following can be noted.

1. For the group of matrix multiplication algorithms that use only arithmetic operations of addition, subtraction, and multiplication, all optimization methods work approximately the same and provide a time reduction of 5% to 35%.

2. The group of algorithms related to series decomposition is divided into 2 parts: the Fourier transform and the wavelet transform, which are

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2025. No. 3 (33)

differently affected by the code optimization methods under study. Optimization of FFT algorithms for processor devices provides 70 to 90% time reduction. For the wavelet transform, the stated optimization methods do not work, except for the PC implementation (30% on average).

3. The hardware costs for all algorithms implemented on the PL part of the SoC increase linearly with the size of the implementation objects.

We summarize the general results of the analyzed code optimization methods as follows.

1. For the wavelet transform, the stated optimization methods do not work, except for the variant with the PC implementation (on average, 30% reduction in algorithm execution time).

2. The optimization method C1 (loop deployment + float type) does not have a significant effect for all algorithms of embedded systems (except for PC). It also does not affect the hardware costs of the PL part of the Zynq SoC.

3. Optimization methods C2 (basic implementation + fixed-point arithmetic instead of float type) and C3 (loop deployment + fixed-point arithmetic instead of float type) have a high effect (up to 90% reduction in execution time) for Fourier transform algorithms (FFT) and up to 35% for matrix multiplication (for embedded systems). For wavelet transform, the stated optimization methods do not work.

4. For programmable logic, the time reduction for C2 and C3 is observed only for FFT and gives no more than 15% reduction in execution time. At the same time, there is a significant reduction in hardware costs up to 80% for matrix algorithms and up to 40% for FFT algorithms.

Conclusions and prospects for further development

As part of the study, we analyzed the impact of typical C/C++ code optimization techniques to determine the impact of each on algorithm execution time and hardware costs on several target platforms and made the following general conclusions

- for algorithms based on arithmetic operations, code optimization gives the effect of reducing the execution time by up to 30% on ARM processor devices;
- for algorithms based on the Fourier transform, comprehensive code optimization gives the effect of reducing the execution time by up to 90% on processor devices;
- for programmable logic devices, code optimization (for selected methods) does not significantly reduce the time;
- the use of fixed-point arithmetic for programmable logic devices gives a significant reduction in hardware costs (from 40 to 80%) for all types of algorithms.

Thus, we have identified code optimization options that are effective in terms of reducing hardware costs when replacing the float data type with an equivalent fixed-point type (fixed_16_16) used for calculations during high-level synthesis.

It was found that in terms of performance, when using the PL part of the Zynq SoC, different ways of optimizing the code have almost no effect on the result. This may be due to the fact that the developed implementation of the algorithm should be optimized for the specific properties of the SoC hardware.

That is why one of the directions for further research may be to analyze the properties of standard HLS-IP cores of the PL-device for use in high-level synthesis.

References

1. Tratt, L. (2025), "Four kinds of optimisation", URL: https://tratt.net/laurie/blog/2023/ four_kinds_of_optimisation.html (last accessed: 10.02.2025).
2. Xu, K., Zhang, G. L., Yin, X., Zhuo, C., Schlichtmann U., Li B. (2024), "Automated C/C++ program repair for high-level synthesis via large language models", *ACM/IEEE international symposium on machine learning for CAD (MLCAD '24)*, 09–11 September 2024, Salt Lake City, USA, P. 1–9. DOI: https://doi.org/10.1109/mlcad62225.2024.10740262
3. Licht, J. de Fine, Besta, M., Meierhans, S., Hoefler, T. (2021), "Transformations of high-level synthesis codes for high-performance computing", *IEEE Transactions on Parallel and Distributed Systems*, 2021,Vol. 32 (No. 5), P. 1014–1029. DOI: https://doi.org/10.1109/tpds.2020.3039409
4. Ahmad A., Du L., Zhang W. (2024), "Fast and practical Strassen's matrix multiplication using FPGAs", *34th International Conference on Field-Programmable Logic and Applications (FPL'24)*, 2-6 September 2024, Torino, Italy, P. 311–317. DOI: https://doi.org/10.1109/fpl64840.2024.00050
5. Khan G. N., Iniewski K. (2017), Embedded systems code optimization and power consumption: chapter in *Embedded and Networking System*, 2017, P. 97–116. DOI: https://doi.org/10.1201/b15497-8

ISSN 2522-9818 (print)
Сучасний стан наукових досліджень та технологій в промисловості. 2025. № 3 (33) *ISSN 2524-2296 (online)*

6. Almorin, H., Gal, B. L., Crenne, J., Jego, C., Kissel, V. (2022), "High-throughput FFT architectures using HLS tools", *29th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 24–26 October 2022, Glasgow, United Kingdom, P. 1–4. DOI: https://doi.org/10.1109/ icecs202256217.2022.9970886
7. Gayathri, S. (2022), "Improved FIR filter using Schonhage-Strassen algorithm based multipliers", *International Journal of Science and Research (IJSR)*, 2022, Vol. 11 (No. 11), P. 1103–1106. DOI: https://doi.org/10.21275/sr221120132156
8. Juang, W.-H., Wu, M.-C., Sheu, Y.-H., Shieh, J.-Y., Hsieh, T.-H. (2023), "A cost-efficient hardware accelerator design for 2D sliding discrete fourier transform", *International Conference on Consumer Electronics – Taiwan (ICCE-Taiwan)*, 17-19 July 2023, PingTung, Taiwan, P. 595–596. DOI: https://doi.org/10.1109/icce-taiwan58799.2023.10227037
9. Kumar, A. (2015), "New trends and challenges in source code optimization", *International Journal of Computer Applications*, 2015, Vol. 131(No. 16), P. 27–32. DOI: https://doi.org/10.5120/ijca2015907609
10. Lee, Y., Youn, J., Nam, K., Oh, H., Paek, Y. (2023), "Optimizing hardware resource utilization for accelerating the NTRU-KEM algorithm", *Computers*, 2023, Vol. 12 (No. 259), P. 1–14. DOI: https://doi.org/10.3390/computers12120259
11. Zhao, R., Cheng, J., Luk, W., Constantinides, G. A. (2022), "POLSCA: polyhedral high-level synthesis with compiler transformations", *32nd International Conference on Field-Programmable Logic and Applications (FPL)*, 29 August – 2 September 2022, Belfast, United Kingdom, P. 235–242. DOI: https://doi.org/10.1109/fpl57034.2022.00044
12. Qian, X., Shi, J., Shi, L., Zhang, H., Bian, L., Qian, W. (2022), "Scheduling information-guided efficient high-level synthesis design space exploration", *IEEE 40th International Conference on Computer Design (ICCD)*, 23-26 October 2022, Olympic Valley, USA, P. 203–206. DOI: https://doi.org/10.1109/iccd56317.2022.00038
13. Hong, H., Xiao, C., Wang, S. (2024), "Rethinking high-level synthesis design space exploration from a contrastive perspective", *42nd International Conference on Computer Design (ICCD)*, 18–20 November 2024, Milan, Italy, P. 179–182. DOI: https://doi.org/10.1109/iccd63220.2024.00035
14. Ferikoglou, A., Kakolyris, A., Kypriotis, V., Masouros, D., Soudris, D., Xydis, S. (2023), "Data-driven HLS optimization for reconfigurable accelerators", *61st ACM/IEEE Design Automation Conference*, 23-27 June 2023, San Francisco, USA, P. 1–16. DOI: https://doi.org/10.1145/3649329.3658471
15. Basalama, S., Cong, J. (25), "Stream-HLS: towards automatic dataflow acceleration", *ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '25)*, 27 February 2025 – 1 March 2025, Monterey, USA, P. 103–114. DOI: https://doi.org/10.1145/3706628.3708878
16. Si, Q., Schaefer, C. B. (2023), "ADVICE: automatic design and optimization of behavioral application specific processors", *Great Lakes Symposium on VLSI (GLSVLSI'23)*, 5-7 June 2023, Knoxville, USA, P. 327–332. DOI: https://doi.org/10.1145/3583781.3590214

Надійшла (Received) 22.05.2025

Відомості про авторів / About the Authors

Shkil Oleksandr − PhD, associated professor, Kharkiv National University of Radio Electronics, associated professor of design automation department, Kharkiv, Ukraine; e-mail: oleksandr.shkil@nure.ua; ORCID ID: https://orcid.org/0000-0003-1071-3445
Filippenko Oleh − PhD, associated professor, Kharkiv National University of Radio Electronics, associate professor of infocommunication engineering department named by V.V. Popovsky, Kharkiv, Ukraine; e-mail: oleh.filippenko@nure.ua; ORCID ID: https://orcid.org/0000-0003-4616-250X
Rakhlis Dariia − PhD, associated professor, Kharkiv National University of Radio Electronics, associated professor of design automation department, Kharkiv, Ukraine; e-mail: dariia.rakhlis@nure.ua; ORCID ID: https://orcid.org/0000-0002-6652-1840
Filippenko Inna − PhD, associated professor, Kharkiv National University of Radio Electronics, associated professor of design automation department, Kharkiv, Ukraine; e-mail: inna.filippenko@nure.ua; ORCID ID: https://orcid.org/0000-0002-3584-2107
Korniienko Valentyn – PhD student, Kharkiv National University of Radio Electronics, design automation department, Kharkiv, Ukraine; e-mail: valentyn.korniienko1@nure.ua; ORCID ID: https://orcid.org/0000-0001-7070-5127

Шкіль Олександр Сергійович – кандидат технічних наук, доцент, Харківський національний університет радіоелектроніки, доцент кафедри автоматизації проектування обчислювальної техніки, Харків, Україна.
Філіпенко Олег Ігорович – кандидат технічних наук, Харківський національний університет радіоелектроніки, доцент кафедри інфокомунікаційної інженерії ім. В.В. Поповського, Харків, Україна.
Рахліс Дарія Юхимівна – кандидат технічних наук, доцент, Харківський національний університет радіоелектроніки, доцент кафедри автоматизації проектування обчислювальної техніки, Харків, Україна.
Філіпенко Інна Вікторівна – кандидат технічних наук, Харківський національний університет радіоелектроніки, доцент кафедри автоматизації проектування обчислювальної техніки, Харків, Україна.
Корнієнко Валентин Русланович – аспірант, Харківський національний університет радіоелектроніки, кафедра автоматизації проектування обчислювальної техніки, Харків, Україна.

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2025. No. 3 (33)

ОПТИМІЗАЦІЯ ПРОГРАМНОГО КОДУ ДЛЯ ВИСОКОРІВНЕВОГО СИНТЕЗУ ПРИ АПАРАТНІЙ РЕАЛІЗАЦІЇ ОБЧИСЛЮВАЛЬНО-НАВАНТАЖЕНИХ АЛГОРИТМІВ

Предметом дослідження є вплив методів оптимізації коду високонавантажених алгоритмів, що застосовуються у цифровій обробці сигналів, на апаратні витрати та швидкодію при реалізації на різних платформах. **Мета.** Порівняльний аналіз впливу трьох підходів до оптимізації С-коду, а саме розгортання циклів, перехід до арифметики з фіксованою комою та їх комбінації, на ефективність реалізації алгоритмів множення матриць, швидкого перетворення Фур'є та вейвлет-перетворення за допомогою засобів високорівневого синтезу (HLS) на платформі SoC, персональних комп'ютерах (ПК) та одноплатних комп'ютерів. У статті вирішуються такі **завдання**: реалізація високонавантажених алгоритмів на базі обраних апаратних платформ та з використанням HLS; порівняння часу виконання алгоритмів із застосуваннями трьох підходів до оптимізації та без; порівняння апаратних витрат для реалізацій алгоритмів з різними варіантами оптимізації коду та без; сформулювати висновки про вплив різних способів оптимізації С-коду на швидкодію та апаратні витрати на різних цільових платформах. Використовуються такі **методи**: методи оптимізації C/C++ коду, діагностичний експеримент за допомогою інструментарію високорівневого синтезу для реалізації алгоритмів цифрової обробки сигналів на обраній апаратній платформі та збору статистичних даних з використанням Python. **Результати.** Для алгоритмів на основі арифметичних операцій оптимізація коду забезпечила до 30% зменшення часу виконання на ARM-платформах. Для алгоритмів на основі перетворення Фур'є комплексна оптимізація дозволила скоротити час виконання до 90% на процесорних пристроях. Для програмованої логіки (FPGA) жоден з методів оптимізації не забезпечив значного прискорення виконання, однак перехід до фіксованої арифметики зумовив зменшення апаратних витрат на 40–80% незалежно від типу алгоритму. **Висновки.** Вибір стратегії оптимізації С-коду має суттєвий вплив на ефективність реалізації алгоритмів на процесорних архітектурах, тоді як для FPGA ключову роль відіграє оптимізація використаних типів даних. Отримані результати можуть слугувати практичними рекомендаціями для проєктування вбудованих систем із застосуванням HLS з метою прискорення алгоритмів.

Ключові слова: вбудовані системи; високорівневий синтез; оптимізація коду С; система на кристалі.

Бібліографічні описи / Bibliographic descriptions

Шкіль О.С., Філіпенко О.І., Рахліс Д.Ю., Філіпенко І.В., Корнієнко В.Р. Оптимізація програмного коду для високорівневого синтезу при апаратній реалізації обчислювально-навантажених алгоритмів. *Сучасний стан наукових досліджень та технологій в промисловості*. 2025. № 3 (33). С. 189–202. DOI: https://doi.org/10.30837/2522-9818.2025.3.189

Shkil, O., Filippenko, O., Rakhlis, D. Filippenko, I., Korniienko, V. (2025), "Optimization of software code for high-level synthesis during hardware implementation of the computationally-loaded algorithms", *Innovative Technologies and Scientific Solutions for Industries*, No. 3 (33), P. 189–202. DOI: https://doi.org/10.30837/2522-9818.2025.3.189