# Fast Square Root Calculation without Division for High Performance Control Systems of Power Electronics

Anton Dianov, *Senior Member, IEEE*, Alecksey Anuchin, *Senior Member, IEEE,* Alexey Bodrov, *Member, IEEE*

***Abstract*—The calculation of square roots is a frequently used operation in control systems of power electronics for different applications: motor drives, power converters, etc. At the same time, the execution of this procedure significantly loads microcontrollers and uses its power, which can be utilized for performing other important tasks. Therefore, it restricts the size of code, which can be processed by the microcontroller and compels developers to limit the number of functions, or to decrease execution frequency of a program. Thus, the calculation of square roots is a bottle-neck in implementation of high-performance control systems, thus effective optimization of this task is extremely important in modern and efficient devices. In respect that many applications do not need precise calculation of square roots, the optimization of execution time can be achieved by decreasing of precision of the result. The proposed technique is based on the approximation of parabola with hyperbola, which allows you to rapidly find the approximate value of a square root. Taking into account that many digital signal processors (DSP) are not equipped with an effective divider, the developed algorithm does not use divisions, so it can be executed faster. The payback for this optimization is approximation error with a maximum of 0.5%, however, it is acceptable for the overwhelming majority of control systems.**

***Index Terms*— Approximate computing, Approximation algorithms, Newton method, Numerical methods.**

## I. Introduction

HIGH levels of competition on modern markets of power electronics devices compels manufacturers to decrease prices and enhance functionality of their devices. In order to do this, developers use cheaper electronic parts and permanently increase the functionality of their products. As a result, there is great demand in the development of high-performance control systems that are capable of operating in weaker processors. Taking into account that increased functionality demands execution of large amounts of code, code optimization has become a task of utmost importance.

Analysis of the general control software of power electronic devices revealed several bottlenecks and one of them was the calculation of square roots. This function is called several times per modulation period and significantly loads a microcontroller unit (MCU). It is used in general mathematics and some specific functions for power electronics such as the calculation of Total Harmonic Distortion (THD) [1]; Root Mean Squares (RMS); observers [2] and models of control objects [3]; coordinate transformation and normalization. Especially intensively square roots are computed in control systems of electrical drives, where they are used in Maximum Torque Per Ampere (MTPA) control [4]-[6], field weakening [7], [8]; Maximum Torque Per Voltage (MTPV) control, Power Factor Correctors (PFC), speed and position estimators [9]-[11]; motor models [12], [13]; observers [14] and adaptive filters [15]. Therefore, time saving implementation of square root calculation can significantly decrease the load to processor and make execution of larger amount of code possible.

The authors of this work were faced with the above mentioned problem, whilst developing new motor drives with enhanced functionality [16], [17]. The basic software was taken from mass productive (MP) motor drives described in [18], [19], which had successfully been produced several years prior. Then, the control system of the motor drive was enhanced with additional functions including monitoring, communication, etc. At the same time, 80 MHz MCU was substituted with 64 MHz microcontroller (iHart Cortex-M3 core) for cost optimization. However, we found that a new MCU is incapable of executing the desired amount of code and performing control at 10 kHz, due to excessive load. Therefore, one of the most important tasks in the development of a new high-performance motor drive, was the optimization of the software (S/W), especially the implementation of square root computation.

A variety of modern MCUs includes fixed-point DSPs as an arithmetic core. These processors are typically designed for fast multiplication with addition, but have no hardware (H/W) units to accelerate division. As a result, division is implemented as a sequence of subtractions and takes as many processor cycles as numbers of bits of the result. For example, division of 32-bits number with 16-bits divider, which produces 16-bits results, takes about 16 computational cycles, significantly increasing the execution time of mathematical algorithms. Therefore, it is preferable to avoid divisions, especially when composing code for processors without H/W acceleration of this operation.

The next key point in the development of a new algorithm is desired precision, which impacts on the execution time. If control software does not need precise calculation of square

Manuscript received April 07, 2021; revised June 30, 2021; accepted October 08, 2021. date of publication June 25, 2022; date of current version June 18, 2022.

Anton Dianov is with the Daeyoung R&D center, 16954, Yongin, Korea (e-mail: anton.dianov@gmail.com).

Aleksey Anuchin is a Head of "Electrical Drives Department" with National Research University "Moscow Power Engineering Institute", 111250, Moscow, Russia (e-mail: anuchinas@mpei.ru).

Alexey Bodrov is with School of Engineering, the University of Manchester, Manchester, UK (e-mail: alexey.bodrov@manchester.ac.uk)

*(Corresponding Author: Anton Dianov)*

Digital Object Identifier 10.30941/CESTEMS.2022.00020

roots, it can significantly accelerate computations. Taking into account that the overwhelming majority of control systems of power electronics apply square root calculations to measured signals, it can be stated that the acceptable tolerance of an algorithm is the same as the precision of sensing circuits. Therefore, the maximum error of 0.5 – 3% is acceptable for computational algorithms, and it was a target of our work.

During the early stages of our work, we examined the existing solutions in order to define their applicability in our system. The authors of [20] proposed classification and reported detailed reviews of the most popular algorithms for the computing of square roots. This review paper may be extended with [21]-[24], which only pays specific attention to fixed-point algorithms.

The simplest solution for the implementation of square root calculation route is the usage of look-up tables. However, precision increase causes enlargement of the table size and needs additional memory. Moreover, calculation of square root near zero is not an easy task, because the square root derivative is high in that region and causes higher calculation errors. In order to solve this problem, special scaling procedures can be involved, however they eliminate main advantages: speed and simplicity [25].

The most popular algorithm for computing of square roots at fixed point DSPs, is the digit-by-digit technique, which is simple for understanding and implementation. As a result, many programmers use this method, despite its slow convergence and long execution time. Excellent examples of this method are given in [25], where it is implemented in hardware and [26], which uses software techniques. However, we do not recommend this method for practical implementation in high performance systems and the experimental part of our paper demonstrates why.

Another approach was proposed by engineers from Analog Devices [27], who used fifth order polynomial approximation at the interval of [0.5..1]. Unfortunately, this method uses scaling, which is performed by built-in H/W of DSP from Analog Devices, therefore its implementation in the processors of other makers may not be efficient.

The algorithms proposed in [28]-[34] suggested H/W implementation of non-iterative methods designed for 8-bit controllers. These algorithms operate extremely fast, but need specific hardware and cannot be used in general computations.

An unusual approach to solving the mentioned problem was proposed by [35]. The authors suggested iterative running of non-linear Infinite Impulse Response (IIR) filter in order to obtain the square root of a given number. However, this method does not use divisions, it intensively involves multiplications with additions, therefore, MCU load remains significant. As a result, this method is not recommended for usage in systems which prioritize optimization of code execution time.

The authors of [36] proposed to use approximation of their function containing square root with Maclaurin's series at a predefined interval. They reported that results were perfect, however this method is not applicable to general calculation of square roots. Some interesting tricks for DSP algorithms, including square root calculations were demonstrated in [37], where the authors suggested to use two variables method. This method was considered in detail in [38], which reported poorer performance, compared to other algorithms. An interesting approach was proposed in [39], where the authors suggested to use sixteen intervals of approximation for 16-bit numbers. In every interval, they approximated the square root with second order polynomial, and reported excellent results. Unfortunately, their experimental results demonstrate unbalanced errors at the interval of approximation, where positive maximum error, negative maximum error and errors at the ends of intervals are not equal. Furthermore, this method needs a lot of memory for storing gains of each approximating polynomial, therefore it is not applicable in high performance systems.

The most perspective algorithms for the calculation of square roots were implemented in code, analyzed and compared in [38], which reported that the Newton-Raphson algorithm was one of the fastest, despite usage of divisions. Therefore, we included the Newton-Raphson algorithm in our research as the main competitive technique.

## II. Newton-Raphson Method

The Newton-Raphson algorithm is one of the most popular methods for the approximation of functions. It is quite simple, speedy and has quadratic convergence, which makes it widely used for approximate calculations. An acceptable example of this technique, adjusted for the optimization of division, is reported in [40], where the authors calculated inversed DC-link voltage without division, accelerating code execution.

Let's use Newton-Raphson for the square root calculation, which is similar to the solution of:

$$x^2 - S = 0 \tag{1}$$

where $S$ is the number, which the square root has to be found.

According to this algorithm, the initial approximation $x_0$ of the square root has to be proposed, and after that, the next step approximation $x_{i+1}$ can be found as:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}, \quad i = 0,1\ldots \tag{2}$$

where $x_i$ is approximation at the current step and $f(x)$ is a function, which root is necessary to find, i.e., (1). This iterative process is completed, when the desired accuracy $\varepsilon$ has been reached:

$$\left|x_{i+1} - x_i\right| \le \varepsilon \tag{3}$$

However, for the purpose of simplification, the calculations can be limited to predefined number of iterations.

Applying (2) to (1) results:

$$x_{i+1} = x_i - \frac{x_i^2 - S}{2x_i} = \frac{1}{2}\left(x_i + \frac{S}{x_i}\right). \tag{4}$$

This equation is also known as the Babylonian method [41], which has been used for ages. Since this technique is quite popular, many researches adopted it for their convenience and proposed several algorithms based on this idea. A detailed explanation of some similar algorithms, which are based on the Newton-Raphson technique is given in [42], where the authors explain their features and compare these algorithms.

Basic formulae of the Newton-Raphson algorithm (4)

calculates square roots with low numbers of arithmetic operations, however, each iteration contains division, which is undesired for DSPs without H/W acceleration. Therefore, it is preferable to eliminate divisions or decrease their number as much as possible. For this purpose, the first operation of division may be changed with an arithmetical shift. Let's denote solution of (1) with $X$ and locate it at the interval:

$$2^n \le X < 2^{n+1}. \quad (5)$$

Then, the initial value of approximation $x_0$ can be selected at this interval, say using middle point:

$$x_0 = 3 \cdot 2^{n-1}. \quad (6)$$

Using (6) as an initial value with combination of (4) results:

$$x_1 = \frac{1}{2}\left(3 \cdot 2^{n-1} + \frac{S}{3 \cdot 2^{n-1}}\right) = 3 \cdot 2^{n-2} + \frac{S}{3 \cdot 2^n}. \quad (7)$$

where division is substituted by the bit shift. However, the next steps after the initial one have to be performed according to (4) and need division. The geometrical interpretation of this method is given in Fig. 1.

In order to evaluate the number of iterations needed to reach the desired tolerance of 0.5 – 3%, the maximum error for every step must be calculated. It is clear that the maximum relative error occurs, when $X$ lies at the lower border of the interval of location: $X=2^n$. Relative error of initial approximation is:

$$\delta_0 = \frac{x_0}{X} - 1 = \frac{3 \cdot 2^{n-1}}{2^n} - 1 = \frac{1}{2} = 50\%. \quad (8)$$

Then, approximate value, calculated at the first step:

$$x_1 = 3 \cdot 2^{n-2} + \frac{2^{2n}}{3 \cdot 2^n} = \frac{9 \cdot 2^{n-2} + 2^n}{3} = \frac{13 \cdot 2^n}{12}. \quad (9)$$

Relative error of the first step approximation:

$$\delta_1 = \frac{x_1}{X} - 1 = \frac{13 \cdot 2^n}{12 \cdot 2^n} - 1 = \frac{1}{12} \approx 8.3\%. \quad (10)$$

Approximate value, calculated at the second step:

$$x_2 = \frac{1}{2}\left(\frac{13 \cdot 2^n}{12} + \frac{12 \cdot 2^{2n}}{13 \cdot 2^n}\right) = \frac{313 \cdot 2^n}{312}. \quad (11)$$

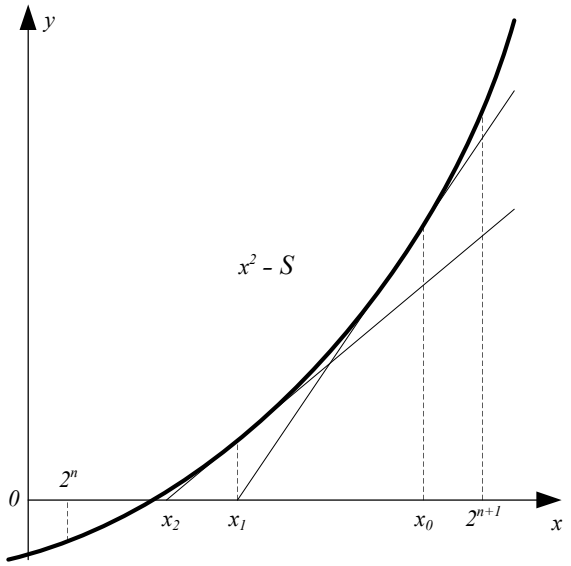



Fig. 1. Geometrical interpretation of the Newton's method.

Relative error of the second step approximation:

$$\delta_2 = \frac{x_2}{X} - 1 = \frac{313 \cdot 2^n}{312 \cdot 2^n} - 1 = \frac{1}{312} \approx 0.32\%. \quad (12)$$

As can be seen, precision of the calculated square root is acceptable after two iterations, thus only one division is needed. Therefore, this result was used in our analysis as a reference value, which we tried to improve in our solution.

## III. Proposed Method

The key idea of the proposed algorithm is to approximate square root parabola with another function, whose root can be easily found. After careful analysis of many functions, we concluded that the best choice is approximation with hyperbola, with general equation:

$$y(x) = \frac{-A}{x - X_0} - Y_0, \quad (13)$$

where $A$, $X_0$, $Y_0$ are hyperbola coefficients. The root of this hyperbola is:

$$x = X_0 - \frac{A}{Y_0}, \quad (14)$$

which can be computed easily.

Therefore, the target parabola is approximated with hyperbola (13) at the interval $[2^n, 2^{n+1}]$ to which the desired square root belongs. It can be seen that equation (13) contains three coefficients, which have to be found. In order to find three unknowns, three criteria have to be used. For this purpose, we can demand hyperbola to be equal to parabola at the borders of approximating interval, which results in zero errors there, as illustrated in Fig. 2. It means that error function is continuously differentiating, which is important for some calculations.

Demanding of equality of the target and approximating functions at the borders of approximating interval results:

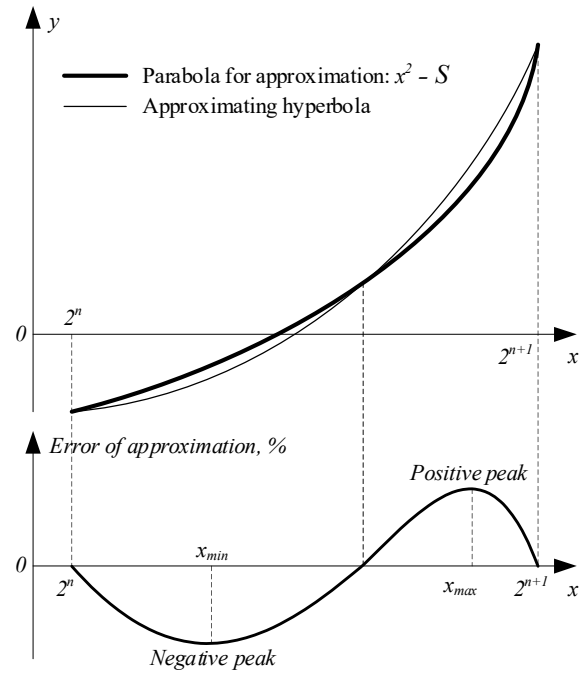



Fig. 2. Approximation of parabola with hyperbola with zero errors at borders of interval of approximation.

$$\begin{cases} 2^{2n} - S + \dfrac{A}{2^n - X_0} - Y_0 = 0 \\ 2^{2n+2} - S + \dfrac{A}{2^{n+1} - X_0} - Y_0 = 0 \end{cases}, \quad (15)$$

which solution is:

$$\begin{cases} X_0 = \dfrac{1}{3}\left(7 \cdot 2^n + \dfrac{Y_0 - S}{2^n}\right) \\ A = \dfrac{Y_0^2 + S^2}{3 \cdot 2^n} + \dfrac{4 \cdot 2^{3n}}{3} - \dfrac{Y_0 S}{3 \cdot 2^{n-1}} + \dfrac{5 \cdot 2^n \cdot Y_0}{3} - \dfrac{5 \cdot 2^n \cdot S}{3} \end{cases}. \quad (16)$$

Therefore (14) transforms into:

$$x = -\frac{S^2}{3 \cdot 2^n \cdot Y_0} + S\left(\frac{5 \cdot 2^n}{3 \cdot Y_0} + \frac{1}{3 \cdot 2^n}\right) + \frac{2^{n+1}}{3} - \frac{4 \cdot 2^{3n}}{3 \cdot Y_0}, \quad (17)$$

which depends only on the single parameter $Y_0$. The authors of [43] demonstrated, that the best approximation can be obtained, when $Y_0$ depends on $S$, however this solution needs one division, which is undesirable. In order to exclude divisions, $Y_0$ is demanded to be constant expressing as:

$$Y_0 = k \cdot 2^{2n}, \quad (18)$$

where $k$ is a balancing coefficient. Thus, (17) transforms into:

$$x = -\frac{1}{3 \cdot k} \cdot \frac{S^2}{2^{3n}} + \frac{S}{2^n}\left(\frac{5+k}{3 \cdot k}\right) + \left(\frac{2k-4}{3k}\right)2^n. \quad (19)$$

The balancing coefficient can be found using criterion of equality of moduli of positive and minimum absolute errors of approximation:

$$\Delta x(x) = x_a - x = X_0 - \frac{A}{Y_0} - x, \quad (20)$$

For this purpose, we used numerical calculations and prepared a simple program in C, which varied $k$ until maximum and minimum of the error function were equal. These calculations resulted:

$$k = 8.8332192$$

$$x(S) = -0.037736\frac{S^2}{2^{3n}} + 0.522015\frac{S}{2^n} + 0.515721 \cdot 2^n. \quad (21)$$

Absolute error of the square root calculated according to (21) can be found as:

$$\Delta x(x) = x_a - x = \\ = -0.037736\frac{x^4}{2^{3n}} + 0.522015\frac{x^2}{2^n} + 0.515721 \cdot 2^n - x, \quad (22)$$

and this expression is illustrated by Fig. 3, which witnesses that positive and negative peaks are equal.

During the next step, we used the same algorithm and found balancing coefficient for (19), which provides equality of positive and negative peaks of relative error:

$$\delta x(x) = \frac{x_a}{x} - 1 = \frac{X_0}{x} - \frac{A}{Y_0 x} - 1. \quad (23)$$

These calculations resulted:

$$k = 8.4934584$$

$$x(S) = -0.039245\frac{S^2}{2^{3n}} + 0.529563\frac{S}{2^n} + 0.509683 \cdot 2^n. \quad (24)$$

The relative error of the square root calculated according to (24) can be found as:

$$\delta x(x) = \frac{x_a}{x} - 1 = \\ = -0.039245\frac{x^3}{2^{3n}} + 0.529563\frac{x}{2^n} + 0.509683 \cdot \frac{2^n}{x} - 1, \quad (25)$$

and this expression is illustrated by Fig. 4, which demonstrates that positive and negative peaks are equal.

Despite low absolute (22) and relative (25) errors of approximation with hyperbolas coinciding with target parabola at borders of interval of approximation, these errors can be decreased more, if zero errors at the end of this interval are not needed (the error function may not be continuously differentiable). In this case, we can change the criteria for definition of hyperbola coefficients. We may demand border errors to be equal by modulus to absolute values of positive and negative peaks, as illustrated in Fig. 6.

Taking into account that resulting approximation is a second order polynomial, we can write the solution as:

$$\Delta x(S) = a_0 \cdot 2^n + \frac{a_1}{2^n} \cdot S + \frac{a_2}{2^{3n}} \cdot S^2, \quad (26)$$

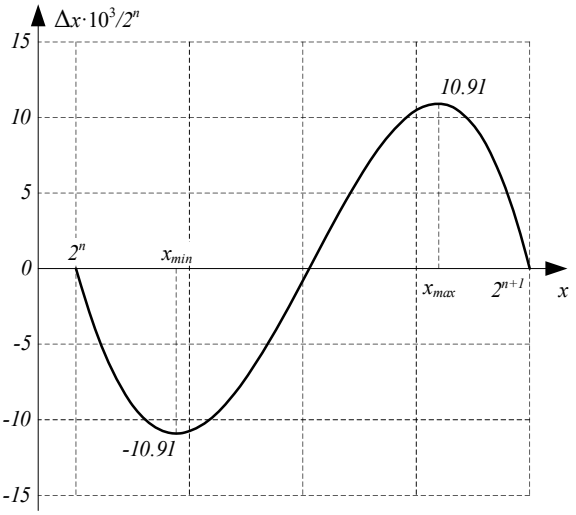



Fig. 3. Normalized error for approximation with balanced $\Delta x$.

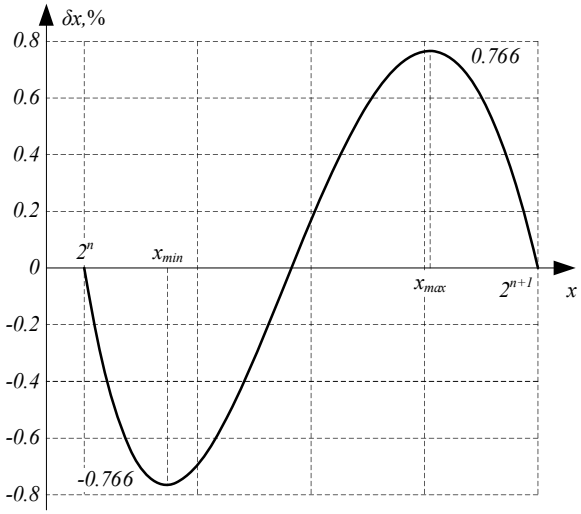



Fig. 4. Relative error for approximation with balanced $\delta x$ hyperbola.

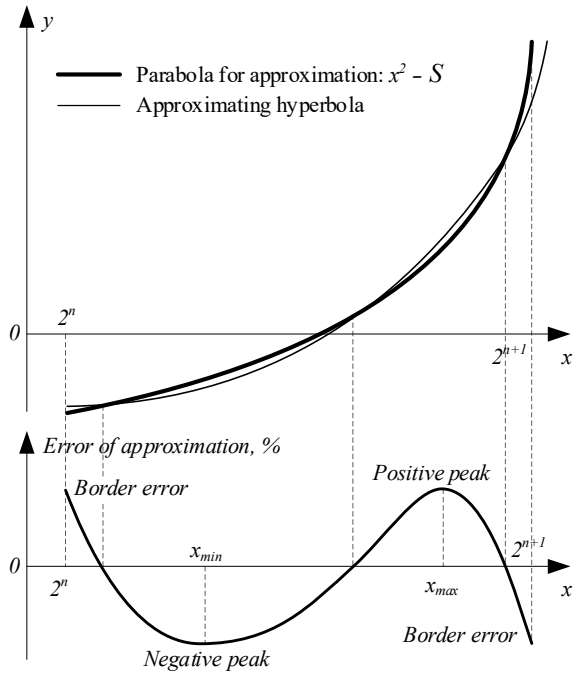



Fig. 6. Approximation of parabola with hyperbola with zero errors at borders of interval of approximation.

where $a_0$, $a_1$, $a_2$ are coefficients to find. Absolute error of approximation with this function is:

$$\Delta x(x) = x_a - x = a_0 \cdot 2^n + \frac{a_1}{2^n} \cdot x^2 + \frac{a_2}{2^{3n}} \cdot x^4 - x, \qquad (27)$$

Border error equality criterion results:

$$\begin{cases} \Delta x\left(2^n\right) = \varepsilon \\ \Delta x\left(2^{n+1}\right) = -\varepsilon \end{cases}$$

$$\begin{cases} a_0 \cdot 2^n + \frac{a_1}{2^n} \cdot 2^{2n} + \frac{a_2}{2^{3n}} \cdot 2^{4n} - 2^n = \varepsilon \\ a_0 \cdot 2^n + \frac{a_1}{2^n} \cdot 2^{2n+2} + \frac{a_2}{2^{3n}} \cdot 2^{4n+4} - 2^{n+1} = -\varepsilon \end{cases}. \qquad (28)$$

This system is used to express $a_0$ via other coefficients:

$$a_0 = \frac{-17a_2 - 5a_1 + 3}{2} \qquad (29)$$

Then, two coefficients $a_1$ and $a_2$ have be found. They can be defined using two criteria: equality of positive and negative peaks by modulus and equality of these peaks to border errors. These coefficients were defined in numerical computations with the help of a previously prepared program, which was slightly modified, which resulted in:

$$a_1 = 0.515141, \qquad a_2 = -0.037296$$

$$x(S) = -0.037296 \cdot \frac{S^2}{2^{3n}} + 0.515141 \frac{S}{2^n} + 0.529159 \cdot 2^n. \qquad (30)$$

Absolute error of this approximation calculated according to (27) is illustrated by Fig. 5, which witnesses that moduli of positive peak, negative peaks and border errors are equal.

Using the same idea, we found the coefficients of (26), which provide the minimum relative error over the interval of approximation. Relative error of the approximation with (26) is:

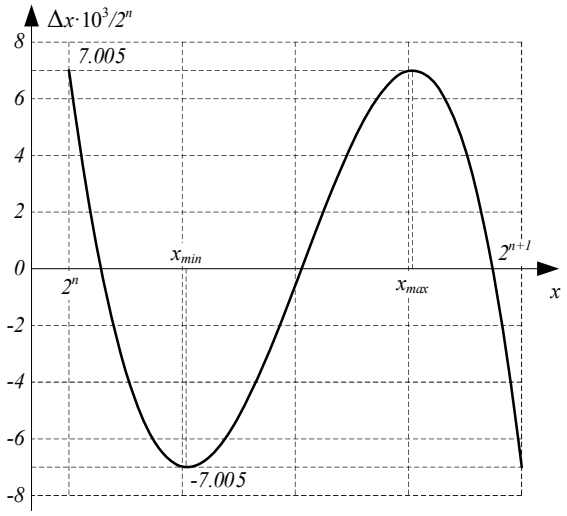



Fig. 5. Normalized error for approximation with minimal $\Delta x$ hyperbola.

$$\delta x(x) = \frac{x_a}{x} - 1 = \frac{a_0 \cdot 2^n}{x} + \frac{a_1}{2^n} \cdot x + \frac{a_2}{2^{3n}} \cdot x^3 - 1, \qquad (31)$$

Border error equality criterion results:

$$\begin{cases} \delta x\left(2^n\right) = \delta \\ \delta x\left(2^{n+1}\right) = -\delta \end{cases}$$

$$\begin{cases} \frac{a_0 \cdot 2^n}{2^n} + \frac{a_1}{2^n} \cdot 2^n + \frac{a_2}{2^{3n}} \cdot 2^{3n} - 1 = \delta \\ \frac{a_0 \cdot 2^n}{2^{n+1}} + \frac{a_1}{2^n} \cdot 2^{n+1} + \frac{a_2}{2^{3n}} \cdot 2^{3n+3} - 1 = -\delta \end{cases}. \qquad (32)$$

This system is used to express $a_0$ via other coefficients:

$$a_0 = -6a_2 - 2a_1 + \frac{4}{3}. \qquad (33)$$

Then, only two coefficients $a_1$ and $a_2$ have be found. They can be defined using two criteria: equality of positive and negative peaks by modulus and equality of peaks to border errors. These coefficients were defined in numerical computations with the help of a previously prepared program, which was slightly modified, which resulted in:

$$a_1 = 0.526010, \qquad a_2 = -0.039540$$

$$x(S) = -0.039540 \cdot \frac{S^2}{2^{3n}} + 0.526010 \frac{S}{2^n} + 0.518555 \cdot 2^n. \qquad (34)$$

Relative error of this approximation calculated according to (31) is illustrated by Fig. 8, which witnesses that moduli of positive peak, negative peaks and border errors are equal.

## IV. Experimental Results

In this part of our work we checked the performance of the proposed method and compared it with the performance of Newton-Raphson and digit-by-digit algorithms. For analysis of the code execution time, the iHart Cortex-M3 based MCU was used since this arithmetic core is widely used and does not have built-in units for hardware acceleration. Execution time of processor commands was taken from the technical reference [44]. When we calculated the execution time, we considered the worst-case scenarios, e.g. operation of division in this MCU

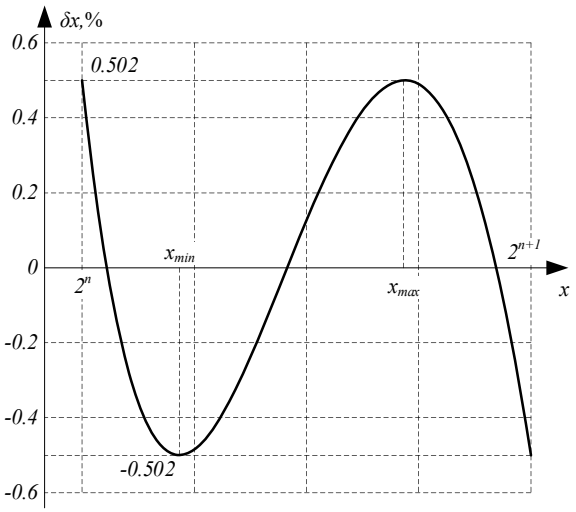



Fig. 8. Relative error for approximation with minimal $\delta x$ hyperbola.

```
CSB    ACC           ; Counts sign bits in Accumulator
LSLL   ACC, T        ; Shift Accumulator left by number of bits
MOV    @shift, T     ; Save number of bits into memory
```

Fig. 7. Example of normalization for C28 core.

takes 2-12 cycles, depending on the data, therefore we used 12 for the evaluation of the computational complexity of the algorithms. All the multiplications were considered as 32-bits operations, taking 1 cycle for execution and all these operations are supposed to involve one more shift operation for scaling of its result. The definition of the initial interval of approximation is similar to finding $n$ satisfying (5) and it was performed as shown in Fig. 9, taking 4 comparisons, 3 shifts and 4 additions, maximum. The flowchart was optimized to reduce penalization due to pipeline when branches occur. When using some microcontrollers such as C28 family, special instructions for normalizations can be utilized shown in Fig. 7.

We calculated the complexity of the proposed technique, efficient Newton-Raphson method, popular digit-by-digit algorithm and put the results into Table I. We counted the number of each operation involved into computations, its execution time and total execution time of each algorithm, in processor cycles. At the same time, we did not count the time needed for data initialization, data movement and branching, which strongly depends on the compiler optimization and conveyer mechanism of the microprocessor. Therefore, the real calculation time of each technique is typically 8-15 cycles the greater depending on the optimization settings.

In addition to theoretical calculations of number of execution cycles, this number was measured experimentally using IAR simulator. The code of routines for test was compiled using optimization of execution time to provide full utilization of the

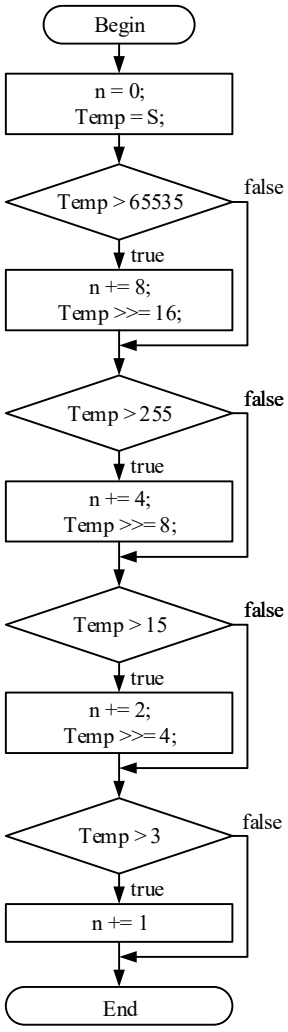



Fig. 9. Definition of the approximation interval.

data transferring. The input number was selected to use all 32-bits.

It can be clearly seen from Table I that the proposed algorithm has lower precision than the competing techniques, however it satisfies the desired conditions. Simultaneously, the execution time of the proposed method significantly decreased, which makes this technique an outstanding candidate for use in optimization of software. Therefore, this method is recommended for usage in high performance control systems of power electronics. The authors of this paper successfully used this algorithm in the control software of electrical drives [8] - [16] and put it into mass production.

## V. CONCLUSIONS

This paper discusses a novel algorithm for fast calculation of

TABLE I
COMPLEXITY OF CALCULATION METHODS AT FIXED POINT MCU

| Method \ Characteristics | Number of iterations | Maximum error | Operations | | | | | Number of cycles without data transfer | Number of cycles with data transfer |
|---|---|---|---|---|---|---|---|---|---|
| | | | +,-,&,\|,^ | <<, >> | Compare | × | ÷ | | |
| Digit-by-digit | 16 | $8 \cdot 10^{-6}$ % | 65 | 64 | 17 | 0 | 0 | 146 | 151 |
| Newton-Raphson | 2 | 0.32 % | 7 | 7 | 4 | 2 | 1 | 32 | 37 |
| Proposed method | 1 | 0.5 % | 6 | 5 | 4 | 2 | 0 | 17 | 23 |

square root without division, which is optimized on execution time. This method is designed for MCU without H/W acceleration of divisions and can significantly accelerate the execution of programs, especially those which involve intensive calculation of square roots. The proposed technique is based on the approximation of parabola with hyperbola in the interval of approximation, which allows one to find the approximate value of the square root rapidly. The payback for this acceleration is precision of calculation, where the maximum relative error is about 0.5%. However, this tolerance is adequate for the overwhelming majority of control systems of power electronics, where typical measurement errors are 1 – 5 %. This paper demonstrates the theoretical background for the proposed method and provides coefficients of the approximating curves for the minimized absolute and relative errors. The experimental part compares the performance of the proposed algorithm with the performance of two iterations of the Newton-Raphson method, which is the best among competitive techniques, and demonstrates a significant decrease of the code execution time, with an insignificant decrease of tolerance.

## REFERENCES


[1] L.M.A. Alsaqal, "Comparison of multiple modulation techniques for various topologies of multilevel converters for single phase AC motor drive," *Int. J. of Power Electron. and Drive Syst.*, vol. 10, no. 2, pp. 662-671, 2019.

[2] T. Sutikno, N.R.N. Idris, A. Jidin, et al, "An Improved FPGA Implementation of Direct Torque Control for Induction Machines," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 3, pp. 1280-1290,. 2013.

[3] T. Sutikno, N.R.N. Idris, A.Z. Jidin, et al, "FPGA based high precision torque and flux estimator of direct torque control drives," in *Proc. 2011 IEEE Applied Power Electronics Colloquium (IAPEC)*, pp. 122-127, 2011.

[4] A. Dianov, et al., "Robust self-tuning MTPA algorithm for IPMSM drives," in *34th Annu. Conf. of IEEE Ind. Electronics*, pp. 1355–1360, 2008.

[5] A. Dianov, A. Anuchin, "Adaptive Maximum Torque per Ampere Control of Sensorless Permanent Magnet Motor Drives," *Energies,* 13, 5071, pp. 1–13, 2020.

[6] A. Dianov, N.-S. Kim, S.-M. Lim, "Sensorless starting of horizontal axis washing machines with direct drive," in *2013 International Conference on Electrical Machines and Systems (ICEMS)*, pp. 1 – 6, 2013.

[7] A. Dianov, et al., "Substitution of the Universal Motor drives with electrolytic capacitorless PMSM drives in home appliances," *In Proc. 9th Int. Conf. on Power Electron. ECCE Asia*, 2015, pp. 1631 – 1637.

[8] A. Dianov, et al., "Future Drives of Home Appliances: Elimination of the Electrolytic DC-Link Capacitor in Electrical Drives for Home Appliances," *IEEE Ind. Electron. Mag.*, 2015, vol. 9, no 3, pp. 10–18.

[9] A. Dianov, et al., "Initial Rotor Position Detection Of PM Motors," in *Proc. EPE Power Electronics and Motion Control Conference,* pp. 1 – 6, 2004.

[10] A. Anuchin, et al., "Synchronous Constant Elapsed Time Speed Estimation Using Incremental Encoders," *IEEE/ASME Transactions on Mechatronics*, vol. 24, no 4, pp. 1893–1901, 2019.

[11] A. Anuchin, et al., "Speed estimation algorithm with specified bandwidth for incremental position encoder," in *Proc. 2016 17th International Conference on Mechatronics - Mechatronika (ME)*, pp. 1 – 6, 2016.

[12] A. Dianov, et al., "Sensorless Vector Controlled Drive for Reciprocating Compressor," in *IEEE Power Electron. Spec. Conf.*, pp. 580–586, 2007.

[13] A. Dianov, N.-S. Kim, S.-M. Kim, "Sensorless starting of direct drive horizontal axis washing machines," *Journal of International Conference on Electrical Machines and Systems*, 2014, vol. 3, issue 2, pp. 148 – 154.

[14] A. Anuchin, et al., "Adaptive Efficient Control for Switch-Reluctance Drives with DCDC-regulator for Inverter Supply," in *Proc. EPE Power Electronics and Motion Control Conference*, pp. 1 – 5, 2004.

[15] C. Moon, J.S. Han, Y.A. Kwon, "Square-Root Unscented Kalman Filter for State Estimation of Permanent Magnet Synchronous Motor," in *Proc. 55th Annual Conference of the Society of Instrument and Control Engineers of Japan*, pp. 460-464, 2016.

[16] A. Dianov, "Estimation of the mechanical position of reciprocating compressor for silent stoppage," *IEEE Open Journal of Power Electronics*, vol. 1, no. 1, pp. 64-73, 2020.

[17] A. Dianov, "Stoppage noise reduction of reciprocating compressors," in *XI Int. Conf. on Electrical Power Drive Systems*, pp. 1–6, 2020.

[18] A. Dianov, et al., "Sensorless IPMSM based drive for reciprocating compressor," in *Proc. 2008 13th International Power Electronics and Motion Control Conference*, pp. 1002 – 1008, 2008.

[19] A. Dianov, S.-T. Lee, "Novel IPMSM drive for compact washing machine," in *Proc. 31st Int. Telecommun. Energy Conf.*, pp. 1 – 7, 2009.

[20] P. Montuschi, P.M. Mezzalama, "Survey Of Square Rooting Algorithms", *IEEE Proc. – Comput. and Digit. Techn.*, vol. 137, no 1, pp. 31–40, 1990.

[21] A. Jena, S.K. Panda, "Revision of Various Square-Root Algorithms for Efficient VLSI Signal processing applications," *IOSR Journal of Electronics and Communication Eng. (IOSR-JECE)*, pp. 38-41, 2016.

[22] T. Bagala, et al., "Single Clock Square Root Algorithm Based on Binomial Series and Its FPGA Implementation," in *Proc. 7th Mediterranean Conference on Embedded Computing (MECO)*, pp. 1-4, 2018

[23] N. Takagi, K. Takagi, "A VLSI Algorithm for Integer Square-Rooting," in *Int. Symp. on Intell. Signal Process. and Commun.*, pp. 626-629, 2006.

[24] A. Seth, W.-S. Gan, "Fixed-Point Square Roots", *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, pp. 1725–1728, 2012.

[25] J. Chen, J.E. Stine, "Optimizations Of Bipartite Memory Systems For Multiplicative Divide And Square Root", in *Proc. 48th Midwest Symposium on Circuits and Systems*, vol. 2, pp. 1458–1461, 2005.

[26] M. D. Ercegovac, "On Digit-By-Digit Methods For Computing Certain Functions", in *Proc. Forty-First Asilomar Conference on Signals, Systems and Computers*, pp. 338–342, 2007.

[27] Edited by A. Mar, "Function Approximation," in "Digital signal processing applications using the ADSP-2100 family", Prentice Hall, NJ, USA: Englewood Cliffs, pp. 57–61, 1992.

[28] T. Sutikno, "A Simple Strategy To Solve Complicated Square Root Problem In DTC For FPGA Implementation", *IEEE Symposium on Industrial Electronics and Applications (ISIEA)*, pp. 691–695, 2010.

[29] M.H. Sheu, S.H. Lin, "Fast Compensative Design Approach For The Approximate Squaring Function", *IEEE Journal of Solid-State Circuits*, vol. 37, no 1, pp. 95–97, 2002.

[30] T.-J. Kwon, J. Sondeen, J. Draper, "Floating-Point Division and Square Root Using a Taylor-Series Expansion Algorithm," in *Proc. 50th Midwest Symposium on Circuits and Systems*, pp. 305-308, 2007.

[31] A. Vázquez, J.D. Bruguera, "Iterative Algorithm and Architecture for Exponential, Logarithm, Powering, And Root Extraction", *IEEE Transactions on Computers*, vol. 62, no 9, pp. 1721–1731, 2013.

[32] R. Vidya, W. Putra, "A Novel Fixed-Point Square Root Algorithm and Its Digital Hardware Design," in *Int. Conf. ICT for Smart Soc.*, pp. 1-4, 2013.

[33] S. Lachowicz, H. Pfleiderer, "Fast Evaluation of the Square Root and Other Nonlinear Functions in FPGA," in *Proc. 4th IEEE International Symposium on Electronic Design, Test and Applications*, pp. 474-477, 2008.

[34] J.M.P. Langlois, D. Al-Khalili, "Carry-Free Approximate Squaring Functions With O(N) Complexity And O(1) Delay", *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 53, issue 5, pp. 374–378, 2006.

[35] N. Mikami, et al., "A New DSP-Oriented Algorithm For Calculation Of The Square Root Using A Nonlinear Digital Filter", *IEEE Transactions on Signal Processing*, vol. 40, issue 7, pp. 1663–1669, 1992.

[36] C. Min, et al., "An Approximate Realization Method of the Square Root Signal Processing Algorithm of the Audio Directional Loudspeaker", in *Proc. Int. Conference on Mechatronics and Automation*, pp. 641-646, 2007.

[37] [1] F. Auger, Z. Lou, B. Feuvrie, et al, "Multiplier-Free Divide, Square Root, and Log Algorithms [DSP Tips and Tricks]," *IEEE Signal Processing Magazine* , vol. 28, issue 4, pp. 122-126, 2011.

[38] A. Dianov, A. Anuchin, "Review of Fast Square Root Calculation Methods for Fixed Point Microcontroller-based control systems of Power Electronics," *International Journal of Power Electronics and Drive System (IJPEDS)*, vol. 3, no. 11, pp. 1153-1164, 2020.

[39] C.-H. Chou, P.-C. Lin, J.-F. Wang, "Fixed-Point Acceleration of Square Root and Logarithm Using Quadratic Regression for HTK Kernel Modules," in *Proc. Fourth International Conference on Genetic and Evolutionary Computing*, pp. 602-605, 2010.

[40] A. Anuchin, et al., "Optimization Of The Division Operation For Real-Time Control Systems", in *International Siberian Conference on Control and Communications (SIBCON)*, pp. 1–4, 2015.

[41] O. Kosheleva, "Babylonian Method Of Computing The Square Root: Justifications Based On Fuzzy Techniques And On Computational Complexity", *Ann. Meeting of Inf. Processing Society*, pp. 1–6, 2009.

[42] C. Ramamoorthy, J. Goodman, and K. Kim, "Some Properties of Iterative Square Rooting Methods Using High-Speed Multiplication," *IEEE Transactions on Computers*, vol. 21, pp. 837-847, 1972.

[43] A. Dianov, A. Anuchin, "Fast Square Root Calculation for Control Systems of Power Electronics," in *Proc. 23rd Int. Conference on Electrical Machines and Systems (ICEMS)*, pp. 1–6, 2020.

[44] ARM Ltd., Cortex-M3 Technical Reference Manual, Rev. r2p0, http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0337h/index.html, Sep 6th 2019.

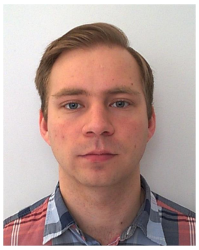
**ANTON DIANOV** (M'06–SM'18) received the B.Sc.(hns.), M.Sc.(hns.), and Ph.D.(hns.) degrees in electrical engineering from the National Research University "Moscow Power Engineering Institute", Moscow, Russia, in 2000, 2002, and 2005, respectively.

From 2005 to 2021 he was a Senior Engineer with Samsung Electronics, where he developed motor drives for home appliances. Since 2021 he has been a Senior Research Engineer with Daeyoung R&D center, Yongin, Korea and an Associate Professor with National Research University "Moscow Power Engineering Institute". He is the author of more than 40 publications in the referring journals and conferences on electrical drives and motor control. He is the author of several patents on control algorithms for electrical drives and power electronics. He is a member of editorial board of several journals on power electronics and electrical drives including IEEE Open Journal of the Industrial Electronics Society, International Journal of Power Electronics and Journal of Power Electronics. His research interests are electrical drives, sensorless and advanced control algorithms

Dr. Dianov was awarded with several personal scholarships including the scholarship form the President of Russian Federation.

**ALECKSEY ANUCHIN** (M'13–SM'19) received the B.Sc., M.Sc., Ph.D., and Dr. Eng. Sc. degrees from Moscow Power Engineering Institute, Moscow, Russia, in 1999, 2001, 2004, and 2018, respectively.

He has more than 20 years of experience covering control systems of electric drives, hybrid powertrains, and real-time communications. He is the author of three textbooks on the design of real-time software for the microcontroller of the C28 family and Cortex-M4F, and control system of electric drives (in Russian). He has authored or coauthored more than 100 conference and journal papers. He delivers lectures on "control systems of electric drives," "real-time software design," "electric drives," and "science research writing" in Moscow Power Engineering Institute. He is in a head position at the Electric Drives Department for the last nine years.



**ALEXEY BODROV** was born in Moscow, Russia in 1987. He received B.S. degree from Moscow Power Engineering Institute, M.S. degree from Seoul National University and Ph.D. degree from The University of Manchester all in electrical engineering in 2008, 2010 and 2019 respectively.

From 2010 to 2014 he was with Samsung Electronics Co. Ltd, working as a research engineer. From 2018 to 2021 he was a research associate in the University of Manchester. Since 2021 he is a control engineer in the CRRC TEIC UK, working on the EV propulsion and battery charging systems control design.