Acceleration System for Single-Precision Floating-Point Arithmetic Based on IEEE754 Standard

Junxuan Yang *

Schools of Electronic and Optical Engineering, Nanjing University of Science & Technology, 210094, Nanjing, China

* Corresponding Author Email: yangjunxuan@njust.edu.cn

Abstract. In recent literature, there has been a heightened interest in the algorithms underpinning artificial intelligence. The crux of this research is the integration of these algorithms into embedded systems. Fundamental to these algorithms are arithmetic operations such as addition and multiplication. This study introduces an implementation of single-precision floating-point arithmetic in compliance with the IEEE754 standard. Adopting a modular approach, the system is deconstructed into distinct modules for addition, subtraction, multiplication, and division, each with its own unique implementation. Within an individual module, components such as sign, exponent, and mantissa are distinctly treated. Delving into the mathematical intricacies of the exponent and mantissa allows for the formulation of the subsequent programming. This study proposes a rounding operation to facilitate diverse rounding modes for the data. Advanced techniques such as the over-advancing adder and Zero judgement are employed to enhance the adder's speed, and constructs like Overflow and Error are introduced to evaluate the output data's status in adherence to the IEEE754 standard. Functional tests of the modules were conducted on a designated platform. A timing simulation was performed using ModelSim, corroborating the robust performance of the proposed system.

Keywords: IEEE754 standard; adder; exponent; mantissa.

1. Introduction

In the contemporary academic landscape, there has been a burgeoning interest in the advancements of deep learning technologies, given their transformative implications for artificial intelligence, the internet, and ancillary domains. Pivotal algorithms such as Convolutional Neural Networks (CNNs), Recursive Neural Networks (RNNs), and Recurrent Neural Networks have found extensive applicability in diverse sectors, notably computer vision, speech recognition, and natural language processing [1].

Field Programmable Gate Arrays (FPGAs), renowned for their inherent parallelism and adaptability, have been harnessed to bolster the performance of the aforementioned algorithms. It is salient to note that accumulative multiplication and summation undergird these algorithms. As such, a pressing academic inquiry has emerged: how can one holistically implement and refine the core arithmetic operations—addition, subtraction, multiplication, and division—on a programmable platform.

In this scholarly investigation, this paper introduces a meticulously designed module tailored for arithmetic operations on single-precision floating-point numbers. To align with foundational computational standards, this module adheres to the IEEE754 protocol—an internationally recognized specification for floating-point formats. Floating-point numbers can be broadly categorized into single-precision, double-precision, and extended double-precision types, with bit widths of 32, 64, and over 80 bits respectively. Although a larger bit width theoretically promises enhanced arithmetic precision, it also necessitates greater storage and computational resources, potentially exacerbating data error probabilities [2]. Given these considerations, for small to medium-scale FPGA designs, single-precision floating-point numbers predominantly suffice. This paper proposed module, aside from offering diverse rounding methodologies, astutely discerns overflow types and ensures outputs are consistent with IEEE754 standards. Employing the Verilog

programming language as its foundation, the module's performance has been evaluated through timing simulations in Modelsim.

2. Module Theory Analysis

2.1. IEEE Standard for Single-Precision Floating Point Numbers

The composition of single-precision floating point number is divided into three parts: the sign S, the exponent E and the unsigned specification mantissa M, the sign S occupies 1 bit, the exponent E occupies 8 bits, and the mantissa M occupies 23 bits, the mathematical expression is as follows.

$$N = (-1)^S \times 2^{E-127} \times 1.M \quad (1)$$

In the above equation: $(-1)^S$ represents the positive or negative of a floating-point number, when S = 1, then N is negative; when S = 0. N Is positive. E Represents an exponent with an offset B (bias) = 127 for the index, the actual index of the number is$(E - 127)$. M Denotes the fractional part of the tail of a specified floating-point number, where the 1 or 0 of the integer part of the specified floating-point number is omitted in arithmetic storage to save space. Due to the offset$B$, the exponent E can range from the original 1 to 254 to -126 to 127, so the maximum number over 0 that can be represented by a floating-point number is theMax, Max= 3.402823*10^{38} (≈2*2^{127}), the Minimum number over 0 is theMin, Min =1.175494*10^{-38} (≈1*2^{-126}), this paper can get the range of single-precision floating-point numbers is $(-Max, -Min] \cup [Min, Max)$ [3].

In the IEEE754 standard, floating-point numbers can be divided into three categories: Normal Number, Subnormal Number and Non-number. Normal Number is used to represent the most common values, e.g. 1.2, 1000, -7.99, 0.003. However, Normal Number cannot represent 0 or numbers very close to 0. In general, most of the data are specification numbers. Subnormal Number is used to represent 0, and numbers smaller thanMin, e.g. 1E-38. The introduction of Subnormal Number is to allow the loss of precision bit by bit when floating-point numbers are underflowed, thus expressing very small numbers near 0 as accurately as possible. Special Number is used to represent "infinity" and "not a number (NAN)". If the number is larger thanMax, the number will be storage as infinity. If a number is not in the IEEE 754 mathematical standard, such as$\sqrt{-1}$, this number will be defined as NAN.

IEEE754 specifies that if all the exponent bits E are filled with 0, the number is denoted as a subnormal number. Similar to subnormal number, if all the exponent bits E are filled with 1, the number is denoted as a Special Number, and based on this, when the mantissa M is filled with 0, the number is denoted as an infinity and when the mantissa number M is not all 0, then the data is saved as NAN [4].

It is worth noting that, under the IEEE754 standard, each move an index bit adjacent to the two digits of the interval is not equal, the greater the index bit, the greater the interval between the two adjacent, after consulting the corresponding interval table, it can be known that the precision of the 32-bit floating-point number is probably in the decimal 7 bits or so. Such an inaccuracy is basically negligible, and can meet the application of most of the projects [5].

2.2. Simulation Software

In this scholarly investigation, this paper has elected to utilize ModelSim for our simulation endeavors. ModelSim, a distinguished simulation tool developed by Mentor Graphics, facilitates the simulation and debugging of digital circuit architectures. The salient features of ModelSim encompass: Extensive compatibility with hardware description languages, including but not limited to Verilog and VHDL. This comprehensive support facilitates a robust foundation for both singular and multi-language design verification frameworks. An environment characterized by stability and user-friendliness, enabling FPGA designers to leverage advanced functionalities requisite for effective debugging. Such a conducive environment fosters seamless project simulation completion.

ModelSim has three simulation methods: Event-based simulation, Time-based simulation, and mixed simulation. Among them, event-based simulation means that the simulation will be carried out only when the signal changes, which is suitable for large-scale digital circuit design; time-based simulation means that the simulator carries out the simulation according to a certain time step, which is suitable for simulating continuous-time systems; mixed simulation is a combination of event-based simulation and time-based simulation, which can simulate digital circuits and analogue circuits at the same time. And analogue circuits at the same time [6].

ModelSim provides a high-performance, full-featured waveform window. This feature can effectively help us complete the timing simulation of waveforms. The Waveform Window provides cursors to mark and measure the time distance between cursors. The waveform window can trace the simulation timing of each intermediate signal. Waveform comparisons can be easily made between two simulation results. Timing differences between RTL and gate-level simulation results can be easily handled with user-specified timing filters.

3. Algorithm Principles and Simulation Analysis

3.1. Adder and Subtractor

3.1.1. Theory

To construct an adder, this paper use a finite state machine to divide the addition into an initialization phase, special number handling (SPECIAl), zero judgement (ZERO), exponents equaling (EQUALEXP) and rounding handling (ROUND), Normalizing mantissa (NORMALIZE) , and overflow judgement (OVERFLOW). The following is a description of the functions of the six parts.

Initialization phase initializes the incoming data. Based on practical engineering considerations, the maximum number of non-statute is about 1.18×10-38, much lower than the general embedded system signals, so the Subnormal Numbers, Infinity and NaN do not enter the operation process, but directly turning to the special number of the processing state. For the different special numbers, this paper set the error status judgement flag ERROR to represent different situations. For the normal numbers, this paper needs to separate the exponent, the mantissa, and the sign bit, and they will be stored separately in the register [7].

Zero judgement determines whether the augend or the addend exists 0. If there is 0, direct non-zero numbers as results and skip the following calculations. Meanwhile the whole process will direct into the overflow judgement in order to save the operation process and time.

Exponents equaling will equal the exponents of augends and addends. Then, this paper will add the mantissas together. Exponents equaling is similar to the number of decimal digits. Only in the case of the same exponent, the addition of the two numbers is meaningful. This paper will make the small exponent to the large exponent of the alignment, for each bit added to the minor order, the whole mantissa will be shifted to the right by 1 digit, in order to ensure the accuracy of the data [8].

After aligning, the mantissa will be added. In order to increase the speed of arithmetic, this paper use the Carry-lookahead adder to complete the operation. The Boolean expression for the adder of one bit is shown below.

$$S = A \oplus B \oplus C_i = A\overline{B}\,\overline{C_i} + \overline{A}B\overline{C_i} + \overline{A}\,\overline{B}C_i + ABC_i \tag{2}$$

$$C_o = AB + BC_i + AC_i \tag{3}$$

Replace the middle part of the parameter with G, P and D.

$$G = AB \tag{4}$$

$$D = \overline{A}\,\overline{B} \tag{5}$$

$$P = A \oplus B \tag{6}$$

$$C_o(G,P) = G + PC_i \tag{7}$$

$$S(G,P) = P \oplus C_i \tag{8}$$

The following relationship exists at the position of each bit in the N-bit adder

$$C_{o,k} = f\left(A_k, B_k, C_{o,k-1}\right) = G_k + P_k) \tag{9}$$

$$C_{o,k-1} = G_k + \mathrm{P}_k(G_{k-1} + P_{k-1}C_{o,k-2}) \tag{10}$$

$$C_{o,k} = G_k + \mathrm{P}_k(G_{k-1} + \mathrm{P}_{k-1}(\ldots + \mathrm{P}_1(G_0 + P_0C_{i,0}) \tag{11}$$

In practice, it is not reasonable to implement a 23-bit super-prevolution because the circuit is too complex and the inputs to the gate cells are too large, which greatly reduces the performance of the circuit. Therefore, our solution is to combine four adders into one unit, and then connect these small units in series [9].

$$P_{cn} = C_{i,0}\prod_{i=0}^{n} P_n \tag{12}$$

$$C_{o,0} = G_0 + P_0C_{i,0} \tag{13}$$

$$C_{o,1} = G_1 + P_1G_0 + P_1P_0C_{i,0} \tag{14}$$

$$C_{o,2} = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_{i,0} \tag{15}$$

$$C_{o,3} = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_{i,0} \tag{16}$$

And the combined Carry-lookahead adder is shown in figure 1.

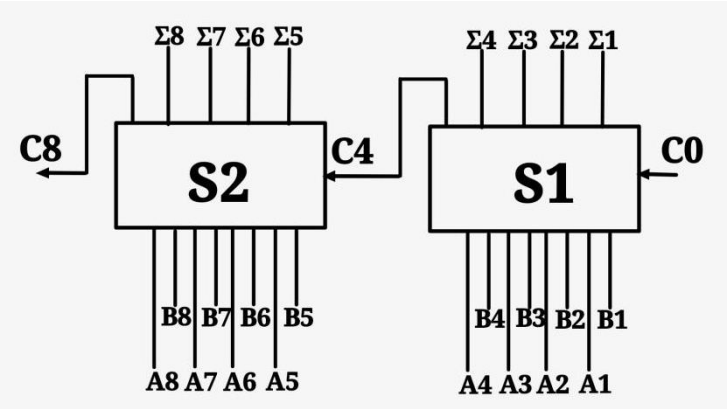

Fig 1. Combined Carry-lookahead adder (Photo/Picture credit: Original).

Rounding handling: I. The simplest rounding mechanism is truncation or rounding to zero. II. Rounding to the nearest number (rounding to an even number): the floating-point number closest to the number is selected as the result. When the number to be rounded is located in the middle of two consecutive floating-point numbers, the IEEE rounding mechanism selects the point with the lowest bit of 0 (i.e., rounding to an even number). III. Rounding to Positive Infinity: Selects the nearest valid floating-point number in the direction of positive infinity as the result. IV. Rounding to Negative Infinity: Selects the nearest valid floating-point number in the direction of negative infinity as the result [10].

Normalizing mantissa: in the process of adding the tails, there may be in or out of place, for the case of in, this paper will sum the tails of the whole right one and sum the tails of the first position of 0, at the same time, and the index of the number of digits plus 1; for the case of out of place, this paper will sum the tails of the whole left one and sum the tails of the end of the number of positions 0.

Overflow judgement judges the overflow situation by detecting the exponent and the mantissa of the sum. If the exponent of the sum is all 1, then it is overflow, output overflow=2'b01; If the exponent of the sum is all 0, and the mantissa of the sum is not all 0, then the result is a subnormal number,

output overflow=2'b10; When the sum's exponent and mantissa are both in the normal number range, output overflow=2'b00 [11].

To construct a subtractor, according to the IEEE754 floating point standard, the positive and negative of a floating-point number is determined directly by the sign bit. Such a property is similar to the original code, so this paper directly negates the sign bit, other operations are handled with the adder is exactly the same, so do not go into details.

3.1.2. Simulation analysis

According to the theoretical description, this section uses modelsim to perform timing simulation tests and the results are shown by the following images. And the first part of the adder simulation is shown in figure 2.

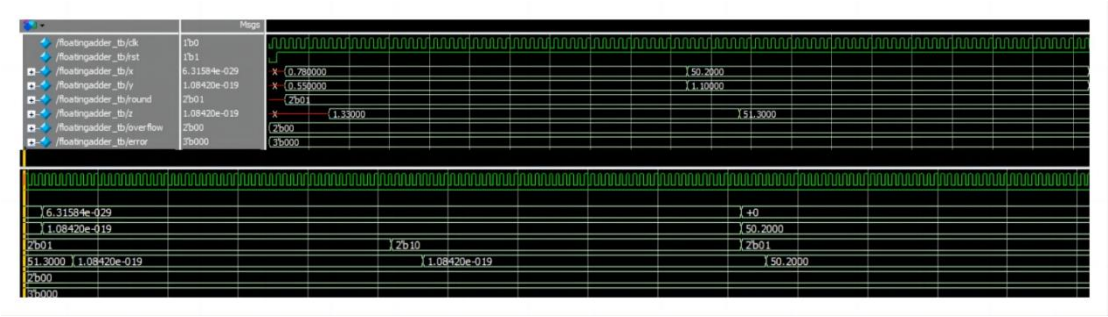

Fig 2. Adder Simulation 1 (Photo/Picture credit: Original).

When rst goes from 0 to 1, the clear state is over and the system starts. The paper first tests the Type II rounding: rounding to the nearest (round=2'b01). Some normal numbers are added into the system and the result is correct. Delay is 8 clock cycles.

Then the paper verifies the Zero judgement and case that the difference between exponents of two numbers is too large ($\Delta E > 24$). Delay is 6 clock cycles. This shows that Zero judgement can reduce latency. And the second part of the adder simulation is shown in figure 3.

Fig 3. Adder Simulation 2 (Photo/Picture credit: Original).

Then paper tests some normal numbers including positive and negative numbers. The same paper verifies that operations between decimals. Delay is 8 clock cycles. Then, paper changes the Type II Rounding to the Type III Rounding. (Round=2'b10). Due to the fact that the exponent difference is too large and rounds to infinity, the result of the Type III Rounding is larger than the result of the Type II Rounding. And the third part of the adder simulation is shown in figure 4.

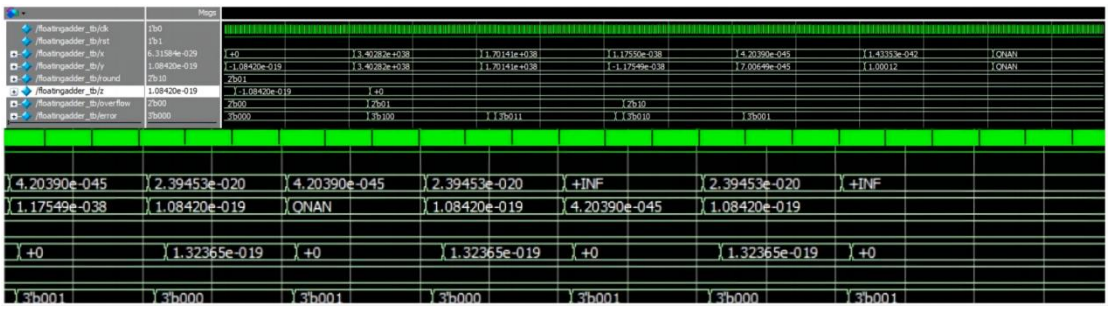

Fig 4. Adder Simulation 3 (Photo/Picture credit: Original).

The paper finally tests some special circumstances. When not a number exists in the number entered into the system, the error will turn to 3'b001. Similarly, the paper places some more extreme numbers to guide the production of some subnormal numbers and infinity. When extreme results are produced, overflow likewise gives the corresponding data. The following table 1 gives the types of error and table 2 gives the type of overflow. The simulation appears to be consistent with the design.

Table 1. Type of error.

Error	Meaning
3'b000	Normal numbers
3'b001	Input non-numbers
3'b010	Output subnormal numbers
3'b011	Output infinity
3'b100	Output NAN

Table 2. Type of overflow.

Overflow	Meaning
2'b00	Normal
2'b01	Overflow
2'b10	Underflow

3.2. Multiplier and Divider

3.2.1. Theory

The initial stage of the multiplier is the same as that of the adder, where the sign bit, exponent, and mantissa of the two multipliers are separated after the arrival of the clock signal. After separating the sign bit, exponent, and mantissa, this paper process three parts as follows.

The sign of the result is the sign of the inputs to do an exclusive or:

$$S_o = S_a \oplus S_b \tag{17}$$

Since Exponent is the result of adding 127 to the original floating-point number, the calculation should be done by subtracting 127 from A, B respectively to get the true order code, and then adding 127 to get the Exponent of output.

$$expo = (expa - 127) + (expb - 127) + 127 + n \tag{18}$$

The mantissa directly invokes the multiplication IP in the program to do the multiplication operation. In the IEEE standard, A, B two numbers of the highest bit 1 is hidden, so here the Mantissa only represents the last 23 bits. In the calculation, this paper needs to bring the highest back, assuming that A is 1.m, B is 1.n [12].

$$A * B = 1 + 0.m + 0.n + 0.m * 0.n \tag{19}$$

Given the complexity inherent in the multiplication operations of the mantissa and exponent, this research predominantly focuses on two rounding methods: zero rounding and rounding to the nearest number. For the zero-rounding method, defined when round_cfg = 0: When the highest bit of the mantissa is 0, the 47th bit is discarded, the 46th bit is hidden, and the 45th-23rd bits are taken as the resultant number, necessitating no shift. Conversely, if the highest bit is 1, the 47th bit is hidden, the 46th-24th bits become the resultant number, indicating a shift. The exponent is then incremented by 1. For rounding to the nearest number, defined when round_cfg = 1: If the highest bit of the mantissa is 0 and the 22nd bit is 1, the 47th bit is discarded, the 46th bit is hidden, and 1 is added to the 45th-23rd bits for the resultant mantissa. If the 22nd bit is 0, the 47th bit is discarded, the 46th bit is hidden, and the 45th-23rd bit is taken as the resultant mantissa. When the highest bit is 1 and the 23rd bit is 1, the 47th bit is hidden, the 46th-24th bits are taken, 1 is added for the resultant mantissa, and the exponent is incremented by 1. When the 23rd bit is 0, the 47th bit is hidden, the 46th-24th bits become the resultant mantissa, and the exponent is incremented by 1.

Regarding overflow judgment, it is observed that the multiplication operation result of the mantissa is not prone to overflow; hence, the result's overflow is dictated by the exponent. The methodology for determining overflow is consistent with that employed in floating-point addition and subtraction operations. Given that underflow is not feasible in multiplication operations, an additional bit in the exponent serves as the determining factor. An overflow is signified when the highest bit of the exponent is 1; conversely, a 0 denotes the absence of overflow.

The subsequent stage involves the consolidation of the sign, exponent, and mantissa into a 32-bit number compliant with IEEE standards. It's imperative to note that the operational principles for both the divisor and multiplier are fundamentally similar. The discernible discrepancy with the multiplier resides in the following facets: The divisor should never equate to zero [13]. If a zero divisor is input, the resultant exponent is directly designated as 8'H11, and the mantissa assumes a zero value, symbolizing infinity. In instances where the divisor is infinite, both the ordinal and mantissa bits are unequivocally assigned a value of 0. During the execution of the division operation, the exponents undergo subtraction, and the mantissas are divided via the invocation of the dividing IP.

$$expo = expa - expb + 127 + n \tag{20}$$

3.2.2. Simulation analysis

According to the theoretical description, this section uses modelsim to perform timing simulation tests and the results are shown by the following figure 5.

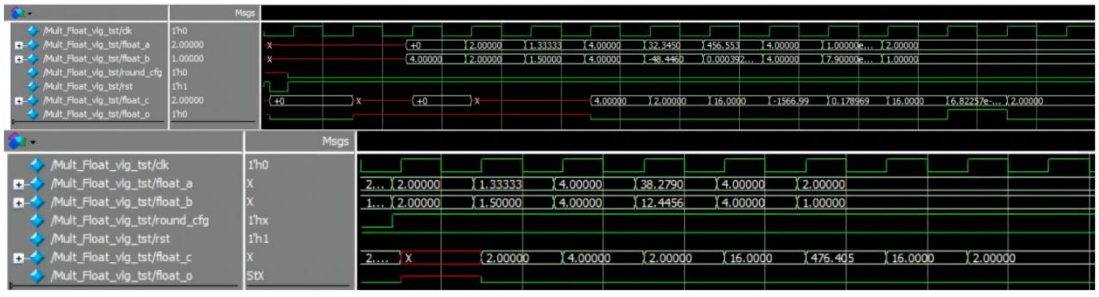

Fig 5. Multiplier Simulation (Photo/Picture credit: Original).

When rst goes from 0 to 1, the clear state is over and the system starts. The paper first tests the Type I rounding and the overflow. Since this paper directly invokes the IP core to handle the multiplication of mantissa, using zero judgement nearly makes no delay. Then, paper gives some integers multiplied by integers including positive and negative numbers and some decimals multiplied by decimals. Also, when the result is out of range, the overflow turns to 1'b1. Finally, the paper tests the Type II rounding and the overflow. All the simulation results are in accordance with the theoretical.

4. Conclusion

This paper uses the Verilog language to complete the addition, subtraction, multiplication, and division module under the IEEE754 standard. Paper uses modelsim software to successfully complete the timing simulation of the module functions. The module has a comparatively high precision, small overall volume, and a wide range of operations, which can meet the needs of most FPGAs for computing. The speed and stability of the module is improved by using over-advancing adders and Zero judgement. This paper makes a more accurate judgement of the overflow and can satisfy the project for the result of the state of the data judgement. The setting of the error flag can also help the user to check if the system is running normally. For special cases, such as subnormal numbers, the result data gives the correct processing. The disadvantage of the module is that the multiplier relies on the internal IP core for the operations, and if the multiplication operation provided by the IP core has a high latency, the latency of the whole multiplication procedure is uncontrollable. In the future, the structure can be further improved, for example, using the booth algorithm, SRT algorithm to improve the multiplier and divider. This paper can also incorporate pipeline techniques to optimise

the structure of the whole program and improve operational efficiency. But this paper still need to maintain the balance between the space complexity of the FPGA line and the speed of operation. The module of this paper can be used in the project with smaller area and less demanding latency requirements. In digital circuit applications, modules can also be transformed in the time and frequency domains by means of Fourier transformations, allowing for more accurate and faster digital signal processing. Based on this module, the project can implement more advanced operations such as convolutional operations, integral and differential operations. Using this module in embedded systems, due to the instantaneous nature of FPGAs, this paper can use it for applications such as image detection, noise filtering, sensitive data monitoring, etc.

References

[1] Li Z, Liu F, Yang W, et al. A survey of convolutional neural networks: analysis, applications, and prospects. IEEE transactions on neural networks and learning systems, 2021.

[2] Daoud L, Zydek D, Selvaraj H. A survey on design and implementation of floating-point adder in FPGA. Progress in Systems Engineering: Proceedings of the Twenty-Third International Conference on Systems Engineering. Springer International Publishing, 2015: 885-892.

[3] Kahan W. IEEE standard 754 for binary floating-point arithmetic. Lecture Notes on the Status of IEEE, 1996, 754(94720-1776): 11.

[4] Brain M, Tinelli C, Rümmer P, et al. An automatable formal semantics for IEEE-754 floating-point arithmetic. 2015 IEEE 22nd Symposium on Computer Arithmetic. IEEE, 2015: 160-167.

[5] Benz F, Hildebrandt A, Hack S. A dynamic program analysis to find floating-point accuracy problems. ACM SIGPLAN Notices, 2012, 47(6): 453-462.

[6] Wiśniewski R, Bukowiec A, Węgrzyn M. Benefits of hardware accelerated simulation. Proceedings of the International Workshop Discrete-Event System Design (DESDes), Poland. 2001: 229-234.

[7] Louca. Implementation of IEEE single precision floating point addition and multiplication on FPGAs. 1996 Proceedings IEEE Symposium on FPGAs for Custom Computing Machines. IEEE, 1996: 107-116.

[8] Nachtigal M, Thapliyal H, Ranganathan N. Design of a reversible floating-point adder architecture. 2011 11th IEEE International Conference on Nanotechnology. IEEE, 2011: 451-456.

[9] Hasan M, Islam M S, Ahmed M R. Performance improvement of 4-bit static CMOS carry look-ahead adder using modified circuits for carry propagate and generate terms. Science Journal of Circuits, Systems and Signal Processing, 2019, 8(2): 76-81.

[10] Kornerup P, Lefevre V, Louvet N, et al. On the computation of correctly rounded sums. IEEE Transactions on Computers, 2011, 61(3): 289-298.

[11] Daoud L, Zydek D, Selvaraj H. A survey on design and implementation of floating-point adder in FPGA. Progress in Systems Engineering: Proceedings of the Twenty-Third International Conference on Systems Engineering. Springer International Publishing, 2015: 885-892.

[12] Jain A, Dash B, Panda a K, et al. FPGA design of a fast 32-bit floating point multiplier unit. 2012 International Conference on Devices, Circuits and Systems (ICDCS). IEEE, 2012: 545-547.

[13] Even G, Paul W. On the design of IEEE compliant floating-point units. Proceedings 13th IEEE Sympsoium on Computer Arithmetic. IEEE, 1997: 54-63.