Implementation of 14 bits floating point numbers of calculating units for neural network hardware development

I V Zoev, A P Beresnev, E A Mytsko and A N Malchukov

Tomsk Polytechnic University, 30, Lenina Ave., Tomsk, 634050, Russia

E-mail: zoev.ivan@yandex.ru

Abstract An important aspect of modern automation is machine learning. Specifically, neural networks are used for environment analysis and decision making based on available data. This article covers the most frequently performed operations on floating-point numbers in artificial neural networks. Also, a selection of the optimum value of the bit to 14-bit floating-point numbers for implementation on FPGAs was submitted based on the modern architecture of integrated circuits. The description of the floating-point multiplication (multiplier) algorithm was presented. In addition, features of the addition (adder) and subtraction (subtractor) operations were described in the article. Furthermore, operations for such variety of neural networks as a convolution network - mathematical comparison of a floating point ('less than' and 'greater than or equal') were presented. In conclusion, the comparison with calculating units of Atlera was made.

1. Introduction

Neural networks are of great importance in the modern world. They are used in problems of prediction, control and classification and in process automation. The scope may be vast, industrial, agricultural equipment, and automatic control of various air and ground equipment. For example, automation of warehouse loaders, agricultural harvester, autopilot in cars.

Current researches are carried out in the direction of increasing the size of neural networks in order to improve accuracy of their work [1]. However increasing the size of network leads to an increase of the perform time. One of the ways to reduce the operating time of the neural network is the use of optimization algorithms. The other way is to experiment with computing architecture of NN. Today, most of the studies are primarily focused at the software level [2-5].

However, if the optimization of calculations is made at the hardware level, it will be possible to achieve better performance in a system of small size. Also, hardware systems require less power than others do. But this task is not as simple as building a network in hardware as this is only one part of the work. For normal operation, the network must be trained. But hardware implementation of learning is not trivial. Since there are many program implementation of learning algorithms, it is simpler to train the network at the program level than to transfer the parameters trained network to hardware.

2. Numbers representation

To transfer the parameters (weights) of the network, firstly, it is necessary to determine in which form they should be stored. Most of the neural networks are working with single precision floating point numbers, rarely with double-precision ones. However, in a hardware implementation, there is a problem the number of logical cells required to implement. And the smaller the bitness of the number, the less logic cells are required for the operation with it. Table 1 shows the number of cells required, depending on the bitness of a number of different accuracies.

Table 1. Dependence of logic cells' numbers on representation of floating point bitness.

Bitness of	Multiplication			Addition		Comparison	
floating point	LUT	REG	Multiplier	LUT	REG	LUT	REG
Fp16	75	27	1 (18x18)	240	29	44	35
Fp32	127	32	1 (27x27)	524	45	82	67
Fp64	307	38	4 (27x27)	1121	107	167	133

The smallest bitness, according to the IEEE 754 standard, is represented with a half floating point [6], which is also supported by Nvidia graphic cards manufacturers, and it can accelerate the process of training neural networks [7].

If we consider the multiplication operation of floating point numbers and compare them with the field-programmable gate array (FPGA) architecture, in which hardware implementation is made, then we can note an interesting feature. For example, Cyclone V has special units for multiplying (DSP – a digital signal processor), which contain the hard multipliers size and number: 3 - 9x9, 2 - 18x18, 1-27x27[8]. In order to save DSP data blocks, it is logical to use variation with the largest number of multipliers. For example, there is only one configuration in Spartan 6 by Xilinx — 18x18, but it can operate at a higher clock frequency [9].

If we go back to the representation of numbers and look at the part of the number that is multiplied (10-bit mantissa + 1 implicit bit), we will find that it is necessary to use an 11x11 multiplier. Therefore, DSP multipliers of 18x18 size will be used to implement this multiplication block. The loss of one unit of the multiplier due to a 2-bit mantissa for hardware implementation of neural networks is not equivalent. Therefore, for the hardware implementation, it has been decided to use a format of a 14-bit floating point from half precision with two bits of the mantissa truncated. This allows using FPGA multipliers optimally, and is compatible with the format of 16 bits.

This transformation affects the calculation accuracy in the neural network, but in the classification task, this problem does not affect the result [10].

3. Multiplication

According to the rules of the IEEE 754 standard operation with floating point numbers, multiplication takes place as follows. The sign of the resulting number occurs from an XOR operation on the signs operands of multiplication. Exponents of operands are added together and the resulting exponent is chosen from the recent multiplication mantissa (simple sum or increment sum). The mantissa is obtained by multiplying, and the result is written into the result number with the truncation up to 9 bits. Representation of the circuit is shown in figure 1.

This scheme does not describe the whole IEEE 754 standard, which may affect the operation of the scheme. Namely, processing of plus/minus infinity cases is lacking. But the neural network almost does not work with these values.

Table 2 shows the value of occupied cells on the FPGA implementation multiplier for the full and non-full IEEE 754 standard.

Table 2. Dependence logic cells of the multiply unit with and without the case of infinity.

	LUT	Different LUT	Multiplier
Fp14 multiplier with infinity	35	2	1 (9x9)
Fp14 multiplier without infinity	33		

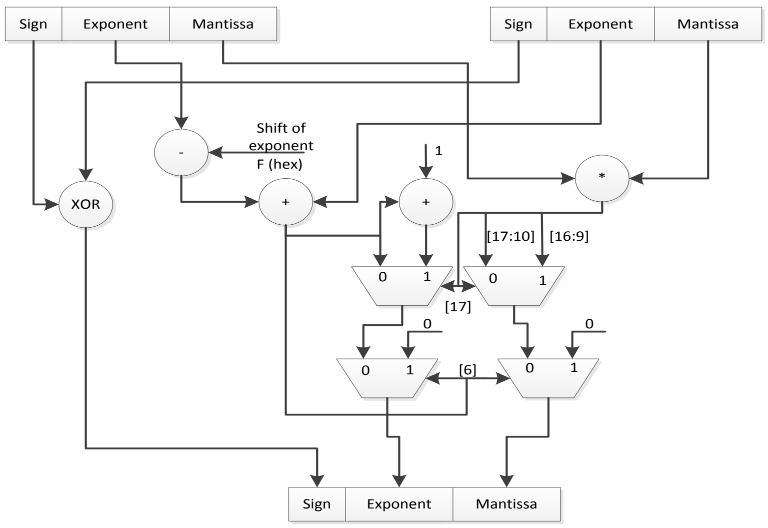

Figure 1. The scheme of the hardware floating point multiplier.

4. Addition and subtraction

Hardware implementation of the addition for floating point numbers is more difficult than multiplication. According to the rules, the first action brings mantissa values to one order.

This requires bringing of the two numbers to a fixed point format. The result of the conversion will be the 40-bits representation. After, addition occurs. The result will be represented in a 41-bit fixed point. Then it is necessary to perform the reverse conversion to the floating point format. The functional diagram is shown in figure 3.

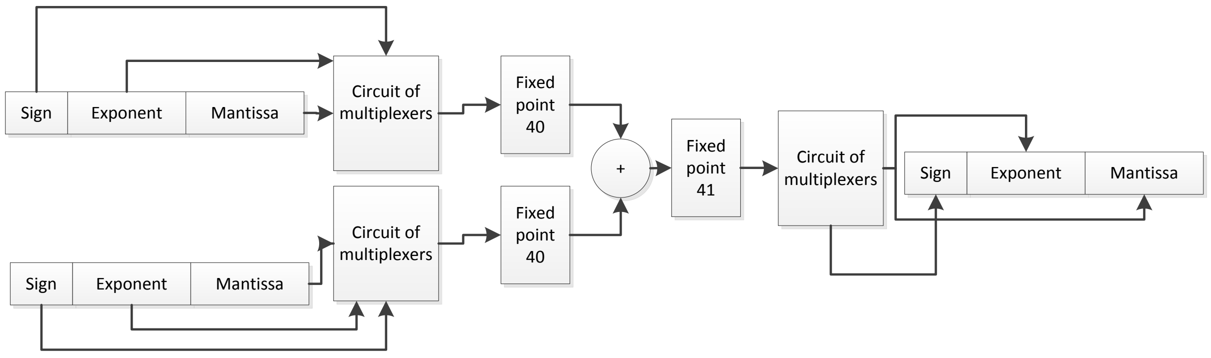

Figure 3. The scheme of the hardware floating point adder.

At the stage of conversion to a fixed point, a sign of operands must be considered and a conversion into two's complement representation for negative numbers has to be made. To implement subtraction, simply change the sign of the second operand.

5. Comparison

Also, other operations are used in neural networks besides the addition and multiplication. For example, a selection of the maximum element is used in convolutional networks. To implement the search of the maximum number, it is necessary to use the adder performing the subtraction. If the result is negative, then the subtrahend is larger than the minuend. If the result is positive, the subtrahend is greater than or equal to the minuend. The scheme is shown in figure 4.

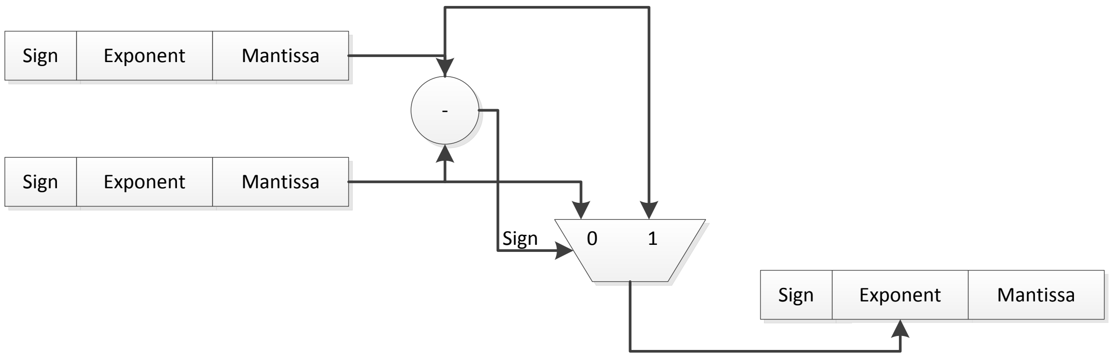

Figure 4. The scheme of the hardware floating point comparator.

6. Result & Conclusion

Implementations of calculating blocks were created using hardware description languages. Altera has a free IP core performing the same functions. However, their modules cannot be configured for the desired parameters in a 14-bit floating point (16-bit representation minimum).

The presented implementations have a form of combinational circuits; therefore, the sync impulse is not required. The delay speed depends on the logical cells. The comparative characteristics tables (tables 3-5) show the difference of the characteristics of the two implementations.

Table 3. Comparison of the multiplier of Altera with logic elements, registers, and performance.

	LUT	REG	Multiplier	F_{max} (MHz)	Latency
Fp16	75	27	1 (18x18)	58	2
Fp14 **(Fp14 inf)**	35 (33)	0	1 (9x9)	123 (137)	1

Table 4. Comparison of the adder of Altera with logic elements, registers, and performance.

	LUT	REG	F_{max} (MHz)	Latency
Fp16	240	29	44	1
Fp14	497	0	35	1

Table 5. Comparison of the comparator of Altera with logic elements, registers, and performance.

	LUT	REG	F_{max} (MHz)	Latency
Fp16	44	35	225	1
Fp14	279	0	60	1

Based on table 3, we can conclude that there are better results of performance and less occupied logic cells compared to the multiplier of Altera. However, Altera wins in the search of the maximum element and the adder by the numbers of occupied logic cells. But if the register is considered as a classical D flip-flop, which contains 6 logic cells, the difference between the modules of Altera becomes smaller.

The optimization problem of these two calculators and the reduction of occupied LUT and/or the increase of the performance are still relevant and give grounds to continue the research of these functional units.

Despite drawbacks, the presented implementation provides the basis for the creation of hardware development of neural networks with software learning. Classic, recurrent and convolution neural networks can be created with these implementations. And the implementation on FPGA provides parallel work of neurons [9].

References

[1] Simonyan K , Zisserman A 2015 *Int. Conf. on Learning Representations* (San Diego: Corenell University Library) p 1
[2] Jia Y, Shelhamer E, Donahue J, Karayev S, Long J, Girshick R, Guadarrama S, Darrell T 2014 *ACM Conf. on Multimedia* (Orlando: Association for Computing Machiner) p 675
[3] Cireşan D, Meier U, Masci J, Schmidhuber J 2012 *J. Neural Networks* **32** 333-338
[4] Dong Y, Eversole A, Seltzer M L, Kaisheng Y 2015 *An Introduction to Computational Networks and the Computational Network Toolkit* Tech. Rep. MSR-TR-2014-112
[5] Gu J, Liu Y, Gao Y, Zhu M 2016 *4th Int. Workshop on OpenCL* (Vienna: Association for Computing Machinery) pp 1-12
[6] IEEE Computer Society 2008 *IEEE Standard for Floating-Point Arithmetic* (New York: IEEE)
[7] Dixon P R, Oonishi T, Furui S 2009 *IEEE Int. Conf. on Acoustics, Speech and Signal Proc.* (Taipei: IEEE) p 4321
[8] Pozniak K T, Czarski T, Romaniuk R S 2004 *The Int. Society for Optical Eng.* (Warsaw) p 1
[9] Omondi A R, Rajapakse J C 2006 *FPGA Implementations of Neural Networks* (Netherlands: Springer)
[10] Gupta S, Agrawal A, Gopalakrishnan K, Narayanan P 2015 *32nd Int. Conf. on Machine Learning* (Lille: International Machine Learning Society) p 1