UDC - 004.855.5

DEVELOPMENT OF SPECIALIZED SYSTEM WHICH INCREASES THE ACCURACY OF THE ARITMETHIC CALCULATIONS

Gagik T. Kirakossian
National Polytechnical University of Armenia
105 Teryan St., Yerevan 0009, RA
gkirakos@seua.am
ORCID iD: 0000-0001-8086-9782
Republic of Armenia

Antranig J. Momjian
National Polytechnical University of Armenia
105 Teryan St., Yerevan 0009, RA
antranigmom@hotmail.com
ORCID iD: 0000-0002-8592-6878
Syrian Arab Republic

Abstract

The need to develop a new format to represent real numbers in order to increase the accuracy of arithmetic calculations in computer systems is proved. A new fractional format to represent number in computer system is developed, and the differences between the suggested format and floating-point format have been presented. A specialized RISC processor that includes the Arithmetic Logic Unit (ALU) intended for processing the numbers represented in the fractional format has been developed, and a system that increases the accuracy of the calculations using the developed processor has been implemented.

Key words: data type, ALU, processor, RTL scheme, Verilog HDL, hardware realization.

Introduction

In computer systems the most common format to represent real numbers is the floating-point format [1]:

- Advantages: big range, simplicity of arithmetic operations execution and hardware realization.
- Disadvantages: impossibility of accurately representing the repeating decimals. For example: 1/3=0, (3), 1/7=0, (14 257), because of which the comparison operations are not performed correctly.

Binary floating-point numbers are represented in computer hardware as base 2 (binary) fractions. For example, the decimal fraction 0.125 has value 1/10 + 2/100 + 5/1000, and in the same way the binary fraction. 0.001 has value 0/2 + 0/4 + 1/8. These two fractions have identical values, the only real difference being that the first is written in base 10 fractional notation, and the second in base 2.

Unfortunately, most decimal fractions cannot be represented exactly as binary fractions. A consequence is that, in general, the decimal point numbers you enter are only approximated by the binary floating-point numbers actually stored in the machine [2-3].

The problem is easier to understand at first in base 10. Consider the fraction 1/3. You can approximate that as a base 10 fraction: 0.3 or, better, 0.33 or, better, 0.333 and so on. No matter how many digits you're willing to write down, the result will never be exactly 1/3, but will be an increasingly better approximation of 1/3.

In the same way, no matter how many base 2 digits you're willing to use, the decimal value 0.1 cannot be represented exactly as a base 2 fraction. In base 2, 1/10 is the infinitely repeating fraction: 0.0001100110011001100110011001100110011001100110011.. since 0.1 is not exactly 1/10, summing three values of 0.1 may not yield exactly 0.3, either. So the command ".1 + .1 + .1 == .3" in computer languages will return Flase.

High floating point inaccuracies may lead to serious software failures and even disastrous results. Known examples include the Vancouver Stock Exchange event [4] and the sinking of the Sleipner A offshore platform [5].

To solve the mentioned inaccuracies, IEEE has developed the decimal floating point format. Decimal floating-point (DFP) arithmetic refers to both a representation and operations on decimal floating-point numbers. Working directly with decimal (base-10) fractions can avoid the rounding errors that otherwise typically occur when converting between decimal fractions (common in human-entered data, such as measurements or financial information) and binary (base-2) fractions [6].

The Decimal floating point format provides several advantages over binary floating point format. Decimal is based on a floating-point model which was designed with people in mind, and necessarily has a paramount guiding principle – computers must provide an arithmetic that works in the same way as the arithmetic that people learn at school." – except from the decimal arithmetic specification.

Decimal numbers can be represented exactly. In contrast, numbers like 1.1 and 2.2 do not have exact representations in binary floating point. End users typically would not expect 1.1 + 2.2 to display as 3.3000000000000003 as it does with binary floating point.

The exactness carries over into arithmetic. In decimal floating point, 0.1 + 0.1 + 0.1 - 0.3 is exactly equal to zero. In binary floating point, the result is 5.5511151231257827e-017. While near to zero, the differences prevent reliable equality testing and differences can accumulate. For this reason, decimal is preferred in accounting applications which have strict equality invariants.

The decimal module incorporates a notion of significant places so that 1.30 + 1.20 is 2.50. The trailing zero is kept to indicate significance. This is the customary presentation for monetary applications. For multiplication, the "schoolbook" approach uses all the figures in the multiplicands. For instance, 1.3 * 1.2 gives 1.56 while 1.30 * 1.20 gives 1.5600.

Some computer languages have implementations of decimal floating-point arithmetic, including PL/I, C#, Java with big decimal, emacs with calc, and Python's decimal module. In 1987, the IEEE released IEEE 854, a standard for computing with decimal floating point, which lacked a specification for how floating-point data should be encoded for interchange with other systems. This was subsequently addressed in IEEE 754-2008, which standardized the encoding of decimal floating-point data, albeit with two different alternative methods.

IBM POWER6 and newer POWER processors include DFP in hardware, as does the IBM System z9 (and later zSeries machines). SilMinds offers SilAx, a configurable vector DFP coprocessor. IEEE 754-2008 defines this in more detail. Fujitsu also has 64-bit Sparc processors with DFP in hardware.

Applications involve financial computations: banking, telephone billing, tax calculation, currency conversion, insurance and accounting in general.

However, the decimal floating point still does not solve the issue completely as it can't represent fractions which base is not lower than 10. For example: in decimal floating point 1 - 1/3 - 1/3 - 1/3 still is not equal to zero.

Conflict setting

Based on what is presented above we suggest to develop a new format to represent real numbers in computers in their fractional form, and implement a system that increases the accuracy of the calculations using the developed format.

Software approach

This problem can be solved using software approach by defining a new structure or class to store the numbers in fractional format and implementing the arithmetic operation for that format. In addition, by using a variable of the new defined type we can implement a program that handles the calculations accurately. However, that approach can't be a final solution to the problem because the arithmetic operations on the numbers represented in that format should be performed by complex software functions which calls recursive functions and its execution time is so big, and makes the application of written program slow.

Development of the fractional format

As we know, any rational number can be expressed as the quotient or fraction x=m/n of two integers, a numerator *m* and a non-zero denominator *n.* For that reason, we suggest to represent the numbers in their fractional form, using the format presented in Fig. 1.

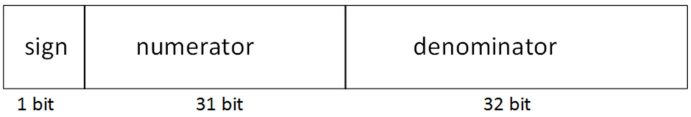

Fig. 1 Fractional format

Comparison between suggested format and floating-point format

To understand the differences (advantages and disadvantages) between the suggested format and the floating-point format, let's compare the potentials of the numbers represented in the formats mentioned above using the same number of bits. For simplicity, we will perform the comparison for 16 bit numbers.

The floating-point formats allows to accurately represent only small part of numbers. For the rest of numbers the inaccuracy of the representation can be calculated by the equation R=A-A', where A' is the value of the representation of A number.

For Example: Let's calculate the inaccuracy of 0.4 representation in half-precision format. For that we should convert 0.4 to binary using the multiplication by two method:

M = 1,1001100110; P = - 2; E = 15 – 2 = 13;

A' = 1 + 307/512.

R= 4/10 – (307 + 512) / 512 = ~ 0,0001.

When representing the same number in fractional format the inaccuracy is zero.

In 16 bit floating-point format the numerator can be stored using 10 bits. However, the denominator should be power of 2, otherwise the approximated value of the number is stored.

Although only 8 bit is allocated to store the numerator in the fractional format, its advantage is that 8 bit is allocated to store the denominator, which allows the denominator to be any number between 0 and 255. The potentials of the two formats to represent numbers is presented in Table 1.

Table 1

Comparison of the floating-point format and fractional format capabilities to represent the numbers as accurate as possible

	Half precision	16 bit fractional format
$n=\frac{x}{2^k}: k\in\mathbb{N}, k\le 8$	accurate	accurate
$n=\frac{x}{y}: y\in\mathbb{N}\cap[0,255]/\{2^k: k\in[0,8]\}$	Approximated. The inaccuracy of the representation is directly proportional to the number represented $\varepsilon = n\times 2^{-10}$	accurate
$n=\frac{x}{y}: y\in\mathbb{N}\cap[256,+\infty[/\{2^k: k\in[8,+\infty[\}$	Approximated. The inaccuracy of the representation is directly proportional to the number represented $\varepsilon = n\times 2^{-10}$	Approximated. The inaccuracy is $\frac{1}{2^8}$

Development of the fractional numbers arithmetic operations' modified algorithms

To develop the co-processor, development of ALU intended for manipulating numbers represented in fractional format is required. To develop the mentioned ALU, we need to modify the algorithms of fractional numbers arithmetic operations so their hardware implementation becomes possible.

Addition

Usually, to calculate the result of $\frac{a}{b}+\frac{c}{d}$ we perform the following steps:

- Calculate the Least Common Multiple (LCM) of b and d numbers. Let's label it with m.
- Calculate the numbers p1 and p2 as follows:

$$p1 = m \div b \quad (1)$$

$$p2 = m \div d \quad (2)$$

- Multiply the numerator and denominator of the first number with p1.
- Multiply the numerator and denominator of the second number with p2.
- With that, we get two fractions that have the same denominator, which summary's numerator is the sum of the two fraction's numerator and the dominator is the common denominator. In other words:

$$\frac{a}{b}+\frac{c}{d}=\frac{a\times p1}{b\times p1}+\frac{c\times p2}{d\times p2}=\frac{a\times p1}{m}+\frac{c\times p2}{m}=\frac{a\times p1+c\times p2}{m}=\frac{n}{m} \quad (3)$$

- Calculate the Great Common Divisor (GDC) of the number m and n. Let's label it with g.
- Divide the numerator and the denominator of the previously calculated fraction on g:

$$\frac{a}{b}+\frac{c}{d}=\frac{n}{m}=\frac{n\div g}{m\div g}=\frac{x}{y} \tag{4}$$

However, this algorithm is not suitable for Hardware realization. But we can bring it to the following form:

- Calculate the following expression

$$\frac{a}{b}+\frac{c}{d}=\frac{a\times d}{b\times d}+\frac{c\times b}{d\times b}=\frac{a\times d+c\times b}{b\times d}=\frac{n1}{m1} \tag{5}$$

- Calculate the GDC of the numbers n1 and m1. Let's label it with g1.
- Divide the numerator and the denominator of the previously calculated fraction on g1:

$$\frac{a}{b}+\frac{c}{d}=\frac{n1}{m1}=\frac{n1\div g1}{m1\div g1}=\frac{x1}{y1} \tag{6}$$

The latest algorithm is more suitable for hardware implementation, by which we get rid of two division blocks and LCM block, which increases the speed of the scheme and decrease the space that the scheme occupies on the chip.

Now, let's prove that the two algorithms are equivalent. In other words: $\frac{x}{y}=\frac{x1}{y1}$

It's clear that any common multiple (CM) of two numbers is greater than or equal to their LCM and it is multiple to LCM. Thus, we can write:

$$CM=k\times LCM;k\in\mathbb{N}^* \tag{7}$$

Which allows us to write:

$$m1=b\times d=k\times m \tag{8}$$

From 8 we find:

$$\begin{aligned} &b\times d=k\times m\\ &\Rightarrow\frac{b\times d}{b}=\frac{k\times m}{b}\\ &\overset{1}{\Rightarrow}d=k\times p1 \end{aligned} \tag{9}$$

And:

$$\begin{aligned} &b\times d=k\times m\\ &\Rightarrow\frac{b\times d}{d}=\frac{k\times m}{d}\\ &\overset{2}{\Rightarrow}b=k\times p2 \end{aligned} \tag{10}$$

From 5 we find:

$$\begin{aligned}
&\frac{n1}{m1} = \frac{a \times d + c \times b}{b \times d} \\
&\overset{8,9,10}{\Rightarrow} \frac{n1}{m1} = \frac{a \times k \times p1 + c \times k \times p2}{k \times m} \\
&\Rightarrow \frac{n1}{m1} = \frac{k\left(a \times p1 + c \times p2\right)}{k \times m} \\
&\overset{3}{\Rightarrow} \frac{n1}{m1} = \frac{k \times n}{k \times m}
\end{aligned} \tag{11}$$

And:

$$\begin{aligned}
&g1 = GCD\left(n1, m1\right) \\
&\overset{11}{\Rightarrow} g1 = GCD\left(k \times n, k \times m\right) \\
&\Rightarrow g1 = k \times GCD\left(n, m\right) \\
&\Rightarrow g1 = k \times g
\end{aligned} \tag{12}$$

From 6 we find:

$$\begin{aligned}
&\frac{x1}{y1} = \frac{n1 \div g1}{m1 \div g1} \\
&\overset{11,12}{\Rightarrow} \frac{x1}{y1} = \frac{\left(k \times n\right) \div \left(k \times g\right)}{\left(k \times m\right) \div \left(k \times g\right)} \\
&\Rightarrow \frac{x1}{y1} = \frac{n \div g}{m \div g} \\
&\overset{4}{\Rightarrow} \frac{x1}{y1} = \frac{x}{y}
\end{aligned} \tag{13}$$

From what is presented above, we find that the two algorithms are equivalent.

Subtraction

By generalizing the addition operation on the signed numbers, we can consider the subtraction operation a private case of the addition operation. In other words:

$$\frac{a}{b} - \frac{c}{d} = \frac{a}{b} + \frac{-c}{d}$$

We will use the same addition algorithm

Multiplication

Usually, to calculate the result of $\frac{a}{b} \times \frac{c}{d}$ we perform the following steps:

- Multiply the numerators with each other and the denominators with each other

$$\frac{a}{b} \times \frac{c}{d} = \frac{a \times c}{b \times d} = \frac{n}{m} \tag{14}$$

- Calculate the GDC of the number m and n. Let's label it with g.
- Divide the numerator and the denominator of the previously calculated fraction on g:

$$\frac{a}{b}+\frac{c}{d}=\frac{n}{m}=\frac{n\div g}{m\div g}=\frac{x}{y} \tag{15}$$

Division

The division operation can be considered multiplication with the inverse of the divisor. And because we are dealing with fractional numbers, the number's inverse can be found by simply switching the places of its numerator and denominator. In other words:

$$\frac{a}{b}\div\frac{c}{d}=\frac{a}{b}\times\frac{d}{c} \tag{16}$$

We will calculate the same multiplication algorithm to calculate the result of the division operation.

Research results

Based on the modified algorithms the ALU intended for manipulating numbers represented in the fractional format. The structure of RTL scheme is represented in Fig. 2.

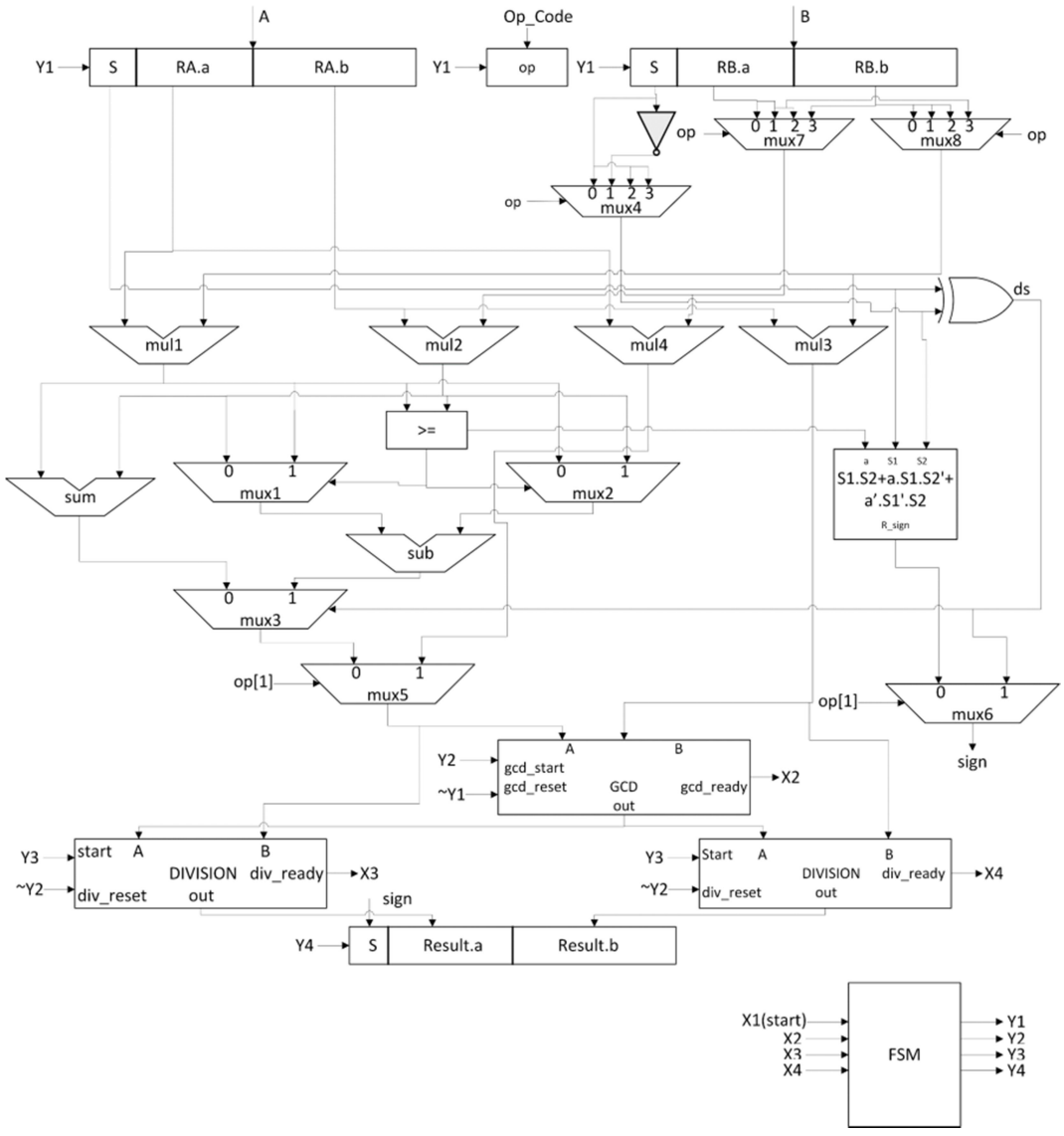

Fig. 2 The RTL scheme structure of the ALU intended for manipulating numbers represented in the fractional format

The scheme consist of six multiplexers, four multipliers, one Grand Common Divisor and two Division blocks, a control unit (FSM), etc. The Grand Common Divisor and two Division blocks are sequential circuits that are controlled by the FSM.

Development and implementation of the specialized RISC processor

A specialized RISC processor has been developed to use the developed ALU to process data. RTL scheme structure is presented in Fig. 3.

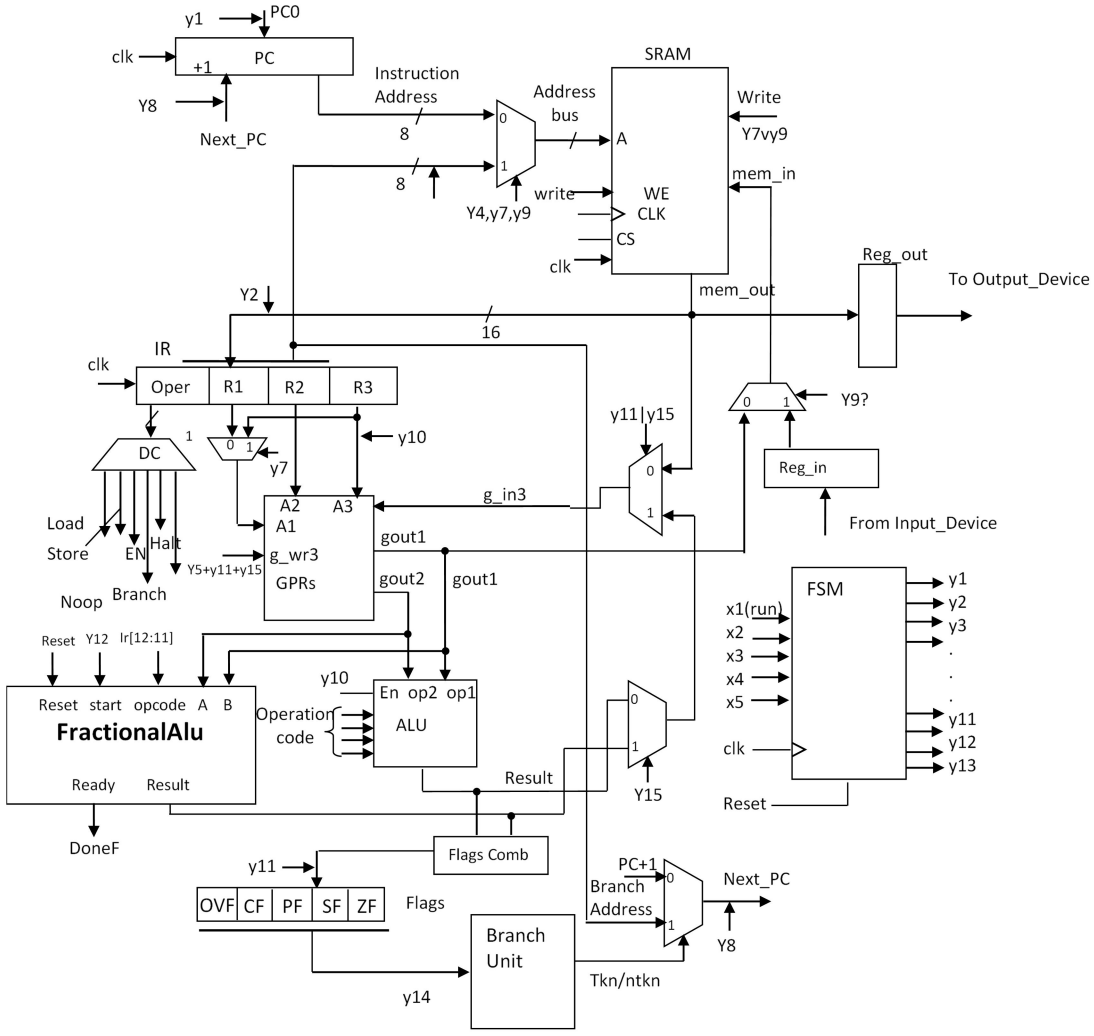

Fig. 3 RTL scheme of the specialized RISC processor that contains a co-processor to manipulate numbers represented in the fractional format

The specialized processor is described in Verilog HDL and simulated using Modelsim HDL Simulator where it showed a perfect behavior.

Implementation of the specialized system

The specialized system consists of a server on which Xilinx Alveo Accelerator cards is installed using PCIe slot. Xilinx Alveo is an acceleration card which includes a Xilinx Ultrascale+ FPGA on which our specialized RISC processor is implemented and run on the FPGA. The application needs to perform accurate calculations written in C/C++, will run on the server's CPU, and will send the commands of the operations that need to be performed

using fractional format to the specialized processor using the SDAccel platform. The Structure of the specialized system is presented in Fig. 4 [7-10].

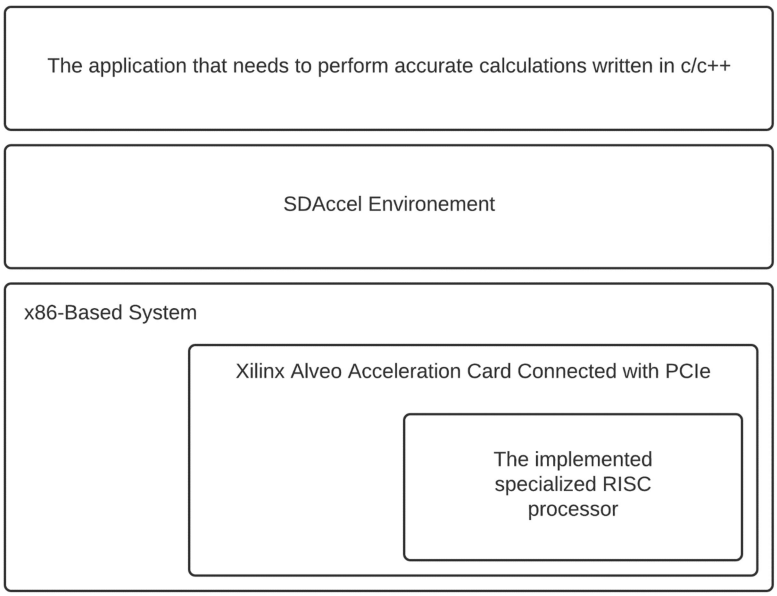

Fig. 4 The Structure of the specialized system that increases the accuracy of the arithmetic calculations

The SDAccel™ development environment is a heterogeneous system architecture platform to accelerate compute intensive tasks using Xilinx® FPGA devices. The SDAccel environment contains a Host x86 machine that is connected to one or more Xilinx FPGA devices through a PCIe® bus.

In the SDAccel framework, an application program is split between a host application and hardware accelerated kernels with a communication channel between them. The host application, written in C/C++ and using API abstractions like OpenCL, runs on an x86 server while hardware accelerated kernels run within the Xilinx FPGA. The API calls, managed by the Xilinx Runtime (XRT), are used to communicate with the hardware accelerators. Communication between the host x86 machine and the accelerator board, including control and data transfers, occurs across the PCIe bus.

Conclusion

From what is presented above we can conclude:

1. The need to develop a new format to represent real numbers in order to increase the accuracy of arithmetic calculations in computer systems is proved.
2. A new, fractional format is suggested to represent real numbers in computer systems, and the differences between the suggested format and floating-point format has been presented.
3. Addition, subtraction and multiplication of fractional numbers and modified variant of division operations have been developed which allow the hardware realization of that arithmetic operations.
4. The Arithmetic Logic Unit (ALU) intended for processing the numbers represented in the fractional format, and the specialized RISC processor that contains a co-processor to manipulate numbers represented in the fractional format have been developed.

5. The specialized system that increases the accuracy of the arithmetic calculations has been implemented

References

1. The Institute of Electrical and Electronics Engineers Inc., IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Std 754-1985, New York, August 12, 1985.
2. Yi X., Chen L., Mao X., Ji T. Efficient Global Search for Inputs Triggering High Floating-Point Inaccuracies (2017) //2017 24th Asia-Pacific Software Engineering Conference (APSEC), Nanjing, China, 4-8 Dec. 2017.
3. Goldberg D. What Every Computer Scientist Should Know About Floating-Point Arithmetic.(1991) //ACM Comput. Surv., vol. 23, no. 1.- p. 5–48.
4. Quinn K. Ever had problems rounding off figures? this stock exchange has (1983) //The Wall Street Journal.- p. 37.
5. Jakobsen B., Rosendahl F. The sleipner platform accident //Structural Engineering International, vol. 4, no. 3.- p. 19.
6. Institute of Electrical and Electronics Engineers, IEEE standard Floating-Point Arithmetic, IEEE Std 754-2008, New York, Aug 2008.- p. 1-58.
7. Xilinx, Alveo // https://www.xilinx.com/products/boards-and-kits/alveo.html
8. Xilinx, Alveo Data Center Accelerator Card Platforms User Guide, UG1120 (v1.7) December 10, 2021.- 47 p.
9. Xilinx, UltraScale Architecture and Product Data Sheet: Overview, DS890 (v4.1) January 7, 2022.- 50 p.
10. Xilinx, SDAccel Programmers Guide, UG1277 (v2019.1) May 22, 2019.- 55 p.

ՀԱՇՎԱՐԿՆԵՐԻ ՃՇՏՈՒԹՅՈՒՆԸ ԲԱՐՁՐԱՑՆՈՂ ՄԱՍՆԱԳԻՏԱՑՎԱԾ ՀԱՄԱԿԱՐԳԻ ՄՇԱԿՈՒՄԸ

Կիրակոսյան Գ.Տ., Մոմջյան Ա.Ժ.
Հայաստանի ազգային պոլիտեխնիկական համալսարան

Թվերի ներկայացման նոր ֆորմատի մշակման անհրաժեշտությունը հիմնավորվել է համակարգչում՝ հաշվարկների ճշգրտությունը բարձրացնելու նպատակով: Թվերի ներկայացման համար մշակվել է նոր կոտորակային ֆորմատ, և նրա հնարավորությունները համեմատվել են գոյություն ունեցող սահող ստորակետով ֆորմատի հետ: Ինչպես նաև մշակվել է մասնագիտացված RISC պրոցեսոր, որը կներառի կոտորակային ֆորմատով ներկայացված թվերի մշակման համար նախատեսված ԹՏՍ-ն, և իրականացվել է հաշվարկների ճշտությունը բարձրացնող մասնագիտացված համակարգ՝ օգտագործելով մշակված պրոցեսորը:

Բանալի բառեր. տվյալների ձևաչափ, ԹՏՍ, պրոցեսոր, RTL սխեմա, Verilog HDL, ապարատային իրագործում:

РАЗРАБОТКА СПЕЦИАЛИЗИРОВАННОЙ СИСТЕМЫ, ПОВЫШАЮЩЕЙ ТОЧНОСТЬ АРИФМЕТИЧЕСКИХ ВЫЧИСЛЕНИЙ

Киракосян Г.Т., Момджян А.Ж.
Национальный политехнический университет Армении

Обоснована необходимость разработки нового формата представления чисел с целью повышения точности расчетов в ЭВМ. Разработан новый дробный формат чисел, его возможности были сравнены с существующим форматом плавающей запятой. Разработан специализированный RISC-процессор, в состав которого входит арифметико-логическое устройство (АЛУ), предназначенное для обработки чисел, представленных в дробном формате, и реализована система, повышающая точность вычислений с использованием разработанного процессора.

Ключевые слова: формат данных, АЛУ, процессор, RTL схема, Verilog HDL, аппаратная реализация.

Submitted on 31.01.2022.
Sent for review on 31.01.2022.
Guaranteed for printing on 11.04.2022.