DESIGN OF COMMUNICATION ORIENTED KERNEL COMPONENTS

Haifei Zhou
Changzhou College of Information Technology, Changzhou, China
E-mail: zhou_haifeihf@163.com

Abstract: In order to develop a new algorithm used in core component design field, a research of processor core components based on communication algorithm is designed. The effectiveness of this kernel component is tested by three research methods. With the rapid growth of user requirements and application complexity, the mobile communications industry has undergone enormous changes in the base-band processor for communication algorithms applications. The performance of processor core functional components is an important factor that restricts the development of communications. The delay of functional components also affects the communication time of the base-band processor. Thus, the design and implementation of components have important theoretical and practical significance. The experimental results show that the floating-point unit based on communication algorithm is relatively low. Meanwhile, the design of processor oriented floating-point unit for communication algorithms is relatively simple, and the implementation conditions of these instructions are optimized from varying degrees. Based on the above finding, it is concluded that the communication algorithm is of great significance to the design of the internal core components of various systems.

Keywords: Communication Algorithm; Floating-Point Arithmetic Unit; Core Functional Components.

1. Introduction

The field of information and communication is not only a wide range of high-tech areas, but also the infrastructure of a country. However, in practice, these technologies are critical to every task that needs to be accomplished by an increasingly modern society. Therefore, it has been identified as a national key technology, and the information and communication technologies are also of universal significance [1]. In the United States, because of the breadth of these technologies, the strong international technical competition that US manufacturers face in this technical field will prevail throughout the entire economy. Therefore, there is no doubt that maintaining the level of Technological Development in information technology determines the economic efficiency of the US manufacturing and service sectors which are increasing substantially. Although the proportion of China's ICT industry is not large, the contribution rate of GDP in China is far lower than that of developed countries. However, the growth rate of China's ICT industry shows that it is one of the fastest speed countries and regions in the world.

According to research reports released by the Ministry of Industry and Information Technology, nowadays, about 0.7% to 0.9% of China's GDP growth is driven by the information and communications industry each year [2]. Based on this environment, in view of the demand of communication algorithm and the characteristics of ARMv7 instruction set, an isomorphic multi-core processor for communication algorithms is proposed. The main task is to design and implement the functional components of the processor core (floating point components). According to the application of the communication algorithm, the required communication instructions are classified and processed, and then the pipelining of the data path of the instruction is divided.

2. Literature review

In 1951, Booth A D first proposed the Booth coding method and Wallace's tree compression method. In 2001, Intel implemented an arithmetic unit with a delay of only 482 ps through the adoption of the 0.18 um CMOS process.

The unit not only uses the technology of insulator silicon, but also realizes the advanced structure. The unit also reduces power consumption as it improves performance, marking a new milestone in the study of multipliers.

In 2008, Godson No.1 CPU IP core was made by 0.18-micron CMO S process. It has a 64-bit floating point unit, and the floating-point component is fully compatible with the MIP S floating-point instruction set.

The floating-point components and their associated system software are fully in line with the ANSI/IEEE754-1985 binary floating- point standard.

From the development of microprocessor at home and abroad, it is shown that there are still some differences between our microprocessor development and foreign countries. Although the design of floating point units has been applied to a great number of algorithms and has been developing toward more and more complex designs, the floating-point unit designed for communication algorithms is relatively low.

The processor oriented floating-point unit design for communication algorithms is relatively simple [3]. According to the IEEE-754 standard, the IEEE-754 binary floating- point standard is the most widely used floating-point number standard at present. It specifies the floating-point operand format, rounding mode, precision representation, and exception handling required to conform to the standard.

The implementation of these floating- point operations mainly consists of the following three phases: pre-operation normalization, mantissa processing, and post processing. For the 32-bit floating-point operand format of this processor, the highest bit symbol is thirty-first bits. The order segment is twenty-third to thirtieth bits, and the mantissa segment is zeroth bit to twenty-second bit. Then, the order and mantissa of the two operands are compared respectively, and the pre-processing is done for the floating- point addition.

This stage is completed in the first access section of the pipeline. In the mantissa processing stage, the corresponding operations are carried out according to the corresponding floating-point operations, and the normalization of symbol bits and the determination of the order codes are performed. When a floating-point integer is converted, the symbol bit remains unchanged.

There is no need to consider symbol bits in floating-point addition [4]. In the floating-point subtraction process, if the absolute value of the subtraction is large, the sign bit is negative.

When floating point multiplication and floating-point division are performed, if the sign bits of the two operands are different, the result is negative. If the two are the same number, the result is positive. The input of the floating-point prescription must be positive and the result is positive.

3. Methods

3.1 Floating point addition unit

It is known that floating-point units need to experience the following operations in the treatment of floating-point addition. Order subtraction of operands: First, the absolute value of the larger order code is used to subtract the absolute value of the smaller order code to obtain the absolute value difference ΔE. The two order codes are then converted to the same value. If it is subtraction, the size of the two order codes needs to be compared [5]. After determining the symbol, the absolute value ΔE of the difference in the order code of the two operands is obtained. Step shift of order code: When mantissa is performed, the exponent of the index with relatively small operands is added to the 4E accordingly.

The mantissa is shifted right by ΔE bits so that the order of the two operands becomes the same. Addition and subtraction of operands: At this stage, the mantissa is added and removed after the shift of the step.

Result normalized shift: In order to make the results of the operation conform to the IEEE standard, the operation result should be changed to the standard format, and the mantissa needs to be normalized and rounded. Meanwhile, the order code should be adjusted [6].

This process is also known as normalization. Rounding results: The result of the mantissa is rounded to the fixed digit after normalization. Result accuracy judgment: Finally, it is necessary to check whether the 9 bits sequence of rounding operations overflow and judge if the result is valid. Flow chart of floating-point addition: As shown in figure 1, this is the concrete implementation structure of floating point addition. As a common operation, the realization of the main part of the floating-point adder is the pipelining of the single channel structure [7].

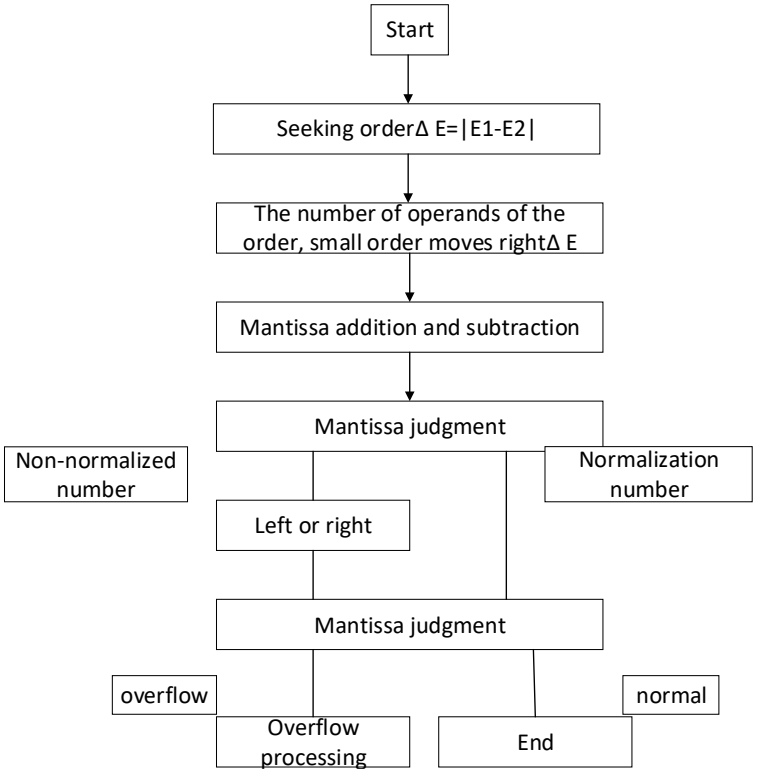

Figure 1 The hierarchical structure of floating-point addition and subtraction

3.2 Floating-point multiplication unit

The multiplication of floating-point numbers is simpler than the addition and subtraction of floating numbers [8]. It can be understood that the multiplication of floating-point numbers includes two algorithms, that is, the multiplication of floating point mantissa (man) and the addition of floating point order code (exp). The concrete calculation process is as follows: c=a*b=a(man)*b(man)=2 a (exp)＋b(exp)=c(man)c（exp. The calculation methods and the states of the order code and mantissa in floating point multiplication are explained.

The first step is to multiply the operands from the floating-point operands. The second step is to add the exponential part of the floating-point operand sent, and the result is c (exp, order code). The third step is to determine the mantissa. If the mantissa is 0, the seventh step is performed.

The index of the result, that is, the order code, is set to -128. The fourth step and the fifth step normalize the results obtained. If the result is shifted to 1 bits and normalized, then skip the eighth step. The mantissa is shifted to the right, and 1 is added directly. If the result is shifted to two bits and normalized, then skip the ninth step. The mantissa is shifted to the right two bits, and the index is added two. The tenth step is to extend the mantissa result.

The sixth step to the eleventh step is to determine the situation of order code. If the rank code overflows, then it will skip to the fourteenth step. If the number of floating point mantissa is greater than 0, then the order code is changed to the maximum positive number. If the order code is less than 0, the order code is set to the minimum negative number. If the order code overflows, the fifteenth step is executed. Meanwhile, the order code is changed to -28, and the mantissa is 0. If the rank code is within range, the sixteenth step is performed to obtain the final result of floating point multiplication.

Figure 2 is a flowchart of floating point arithmetic, which shows the flow of floating-point multiplication components. The floating-point multiplication unit is a relatively important part of the floating-point arithmetic unit. In floating point multiplication, the symbol bit judgment is handled by XOR, and the calculation of the code segment is realized by addition [9]. Mantissa processing is more complex, and Wallace tree compression is a widely used program.

In the initialization phase of the floating-point component operand initialization, the floating multiplication operation starts with the simultaneous determination of whether the two floating-point operands are 0. If the operand is 0, the floating-point multiplication is completed at this point.

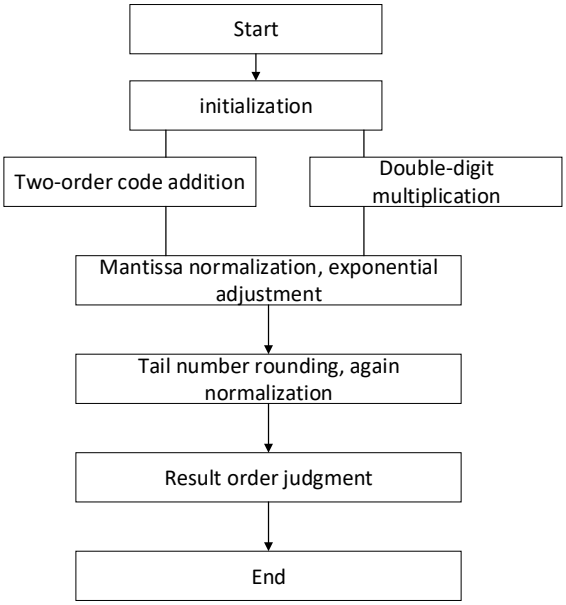

Figure 2 Arithmetic flow of floating point numbers

3.3 Floating division principle

Floating point division takes 28 cycles in the processor core for communication algorithms, while the floating-point evolution requires 27 cycles. When the instruction is executed, the pipeline is suspended and the pipelining continues to run after the divide command is completed. The implementation of these long period division operations is described below.

Compared with floating-point multiplication and addition, floating division and square roots do not have much use. For the communication oriented processor, the floating-point division unit may become the bottleneck restricting the improvement of system performance [10].

The implementation of floating point division is shown in figure 3, The 23-bit mantissa division needs a total of 23 cycles, accounted for 91% of the total time of floating-point division. In figure 3, it needs to be determined that whether the two floating-point operands are within the representation range, that is, whether it is 0 or infinity. If it is 0 or infinity, the result is set to 0, and the entire floating-point division is completed in the initialization phase. If this is not the case, the two operand is within the scope of the part calculation. The order numbers and mantissa of two floating point operands are separated, and the order code and mantissa are processed separately.

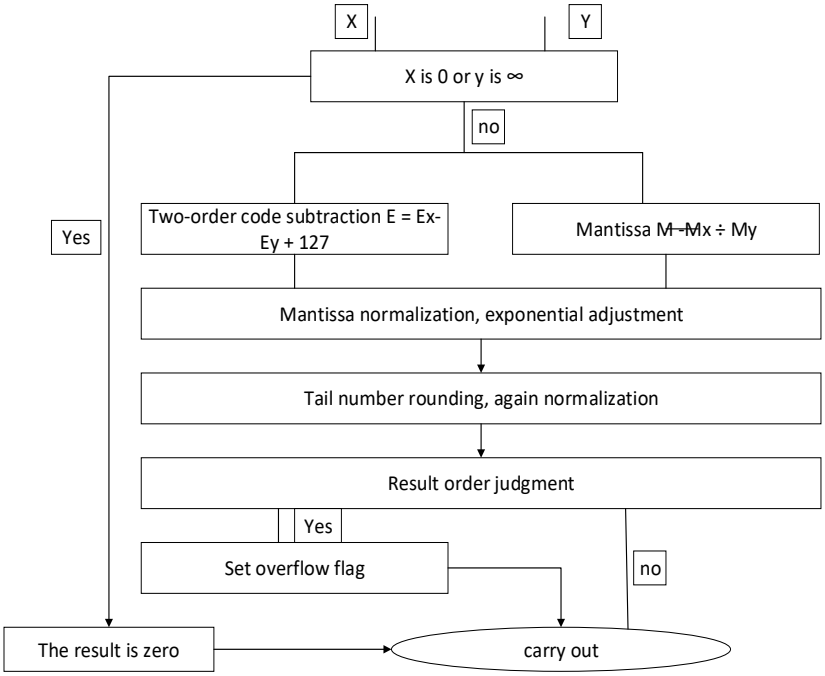

Figure 3 Floating-point division implementation process

3.4 Float evolution principle

In several operations of floating, float evolution algorithm is relatively simple. The number of operation of floating-point evolution is relatively small, and it is only one. The result of calculation tends to 1. The calculation of this paper is mainly for the operation of the mantissa. The operation of the floating-point order is straightforward, and the data of the order code is directly divided by 2 in the order code.

The mantissa involves a true squares operation. The main process is to determine whether the floating-point operand is 0, and the floating-point start is completed when the floating-point operand is initialized. At this time, if the radicand order number is odd, then the order code only need to add 1 and divided by 2, then the offset 63 is added. At this time, the evolution for the mantissa calculated. If radicand order number is even, then the process of the exponent is to divide the order code by 2 and add 65.

At this point, the mantissa is shifted to the left with one unit and then the mantissa operation is continued. Finally, the rounding of the mantissa result is carried out, and the normalized sequence of code segments is judged over and over. In addition to dealing with earlier parity check operations on operands, there is also a correlation of the subsequent mantissa. In addition to the operation process, floating evolution unit is similar to the other addition, subtraction, multiplication process.

4. Results and discussion

After the addition and subtraction method is completed, it is necessary to judge whether the result is overflow or not. If it overflows, then the result is shifted to the right with 1 unit add 1 to the code. If this is non-normalized, a leading zero logic decision is required. Then, the number of zeros is used to move the mantissa to the left, while the order number is subtracted from the number of zeros. For the last 28-bit mantissa, the nearest rounding is selected. After the completion of the mantissa processing, the sequence code may overflow.

Therefore, the overflow of the order code is processed and rounded by the nearest round, so that the overall result conforms to the normalization standard.

Floating point addition and subtraction parts are common components. Therefore, the optimization of its main structure and the flow of its internal structure can improve the performance of floating-point addition and subtraction components. The floating-point adder implements three level pipelining, each of which has the pipeline register of the segment. First of all, the temporary data sent by the superior is stored and sent to the next level of the device.

The multiplication normalization module normalizes the 48-bit mantissa of 1-bit, 8-bit and partial. The result of partial summation is used as the result of the multiplication of the mantissa of the floating-point number. The results need to be normalized. In the floating-point multiplication pipeline, the final operation is mainly to normalize the results of the mantissa multiplication, and adjust the resulting order code, where the floating-point multiplication operation is over.

Division determines the positive and negative values of our calculations based on whether the two signals sign 1 and sign 2 are the same. Finally, the quotient obtained by the mantissa division is normalized. According to the processing result, the rank code segment is adjusted, so that the data format of the merchant conforms to the normalization number of the IEEE-754 standard.

Then the rounding process is carried out, and then the rounding merchant is judged to see if it is necessary to do the normalization process again. If the specification is carried out again, it is also necessary to determine whether the step code segment obtained by the normalized process has an overflow. If an overflow occurs, the it is placed at the bit zone. If there is no overflow, the operation ends here.

The normalization of the floating-point module is mainly the processing of the mantissa. There will be no overflow of the emergence of the prescription. It is mainly an adjust on the mantissa rounding and order code after the mantissa normalized process. Using the start-up item. Synopsys_dc.setup, the DC synthesis process of each sub module of the floating-point arithmetic unit is computed, and the result is shown in table 1.

The comprehensive contrast is shown in figure 4:

Table 1 The Timing, Area and power consumption results after the integration of each module of the floating-point unit

	Timing (ns)	Area (um^2)	Power (uW)
FADD	14.63	5663.145682	1413.0
FMUL	12	5530.492884	955.8981
FDIV	11	3307.147268	389.9674
FSQRT	9.85	2032.128028	49.2886

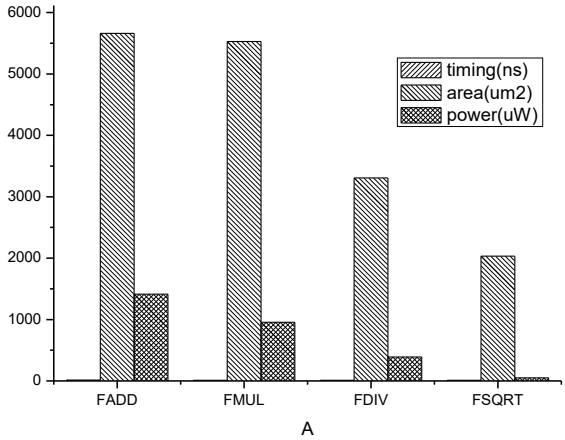

Figure 4 Comparison of Timing, Area and Power after synthesis of different modules

5. Conclusions

In the aspect of floating point unit, all the hardware implemented by floating point unit are discussed in detail. Although more components are implemented, floating point data paths are relatively complex. However, it has little impact on the processing performance of floating-point instructions according to instructions. It is able to execute instructions sent in sequence and quickly handles floating-point instructions in parallel.

The hardware implementations of these instructions are optimized from varying degrees. At the same time, the related work of the project is still going on. On the basis of the existing achievements, it is the inevitable development direction to continue to carry out project engineering. The focus of the future work is mainly reflected in the following aspects: The interface between the functional components of this topic and the other components inside the microprocessor is being matched. At the same time, the functional realization of the microprocessor also requires a good match between the various functional components. Multi-core interconnection also needs to solve the routing and other issues.

References

[1] Wu, Zhenyong, et al. "Nuclear product design knowledge system based on FMEA method in new product development." Arabian Journal for Science and Engineering 39.3 (2014): 2191-2203. https://doi.org/10.1007/s13369-013-0726-7

[2] Booth, A D, A Signed Binary Multiplication Technique [J]. Quarterly Journal of and Applied Mathematics} 1951, 4 (2):236-240 https://doi.org/10.1093/qjmam/4.2.236

[3] Bedrij O. Carry Select Adder. IRE Trans. on Electronic Computers, 1962, 11(6): 340-346. https://doi.org/10.1109/iretelc.1962.5407919

[4] Nieße, Astrid, Martin Tröschel, and Michael Sonnenschein. "Designing dependable and sustainable Smart Grids–How to apply Algorithm Engineering to distributed control in power systems." Environmental Modelling & Software 56 (2014): 37-51. https://doi.org/10.1016/j.envsoft.2013.12.003

[5] Reed, Daniel A, and Jack Dongarra. "Exascale computing and big data. “Communications of the ACM 58.7 (2015): 56-68. https://doi.org/10.1145/2699414

[6] Bui, Duong Minh, et al. "Investigate dynamic and transient characteristics of microgrid operation and develop a fast-scalable-adaptable algorithm for fault protection system." Electric Power Systems Research 120 (2015): 214-233.

[7] Khajavirad, Aida, Jeremy J. Michalek, and Timothy W. Simpson. "Solving the joint product platform selection and product family design problem: An efficient decomposed multi-objective genetic algorithm with generalized commonality." Advances in Product Family and Product Platform Design. Springer New York, 2014. 271-294.

[8] Wang, J., Lu, K., Zhang, S., Fan, J., Zhu, Y., & Cheng, B. (2015). An efficient communication relay placement algorithm for content-centric wireless mesh networks. International Journal of Communication Systems, 28(2), 262-280.

[9] Li, G., Wimalajeewa, T., & Varshney, P. K. (2015). Decentralized and collaborative subspace pursuit: a communication-efficient algorithm for joint sparsity pattern recovery with sensor networks. IEEE Transactions on Signal Processing, 64(3), 556-566. https://doi.org/10.1109/tsp.2015.2483482.

[10] Fadlullah, Z. M., Takaishi, D., Nishiyama, H., & Kato, N. (2016). A dynamic trajectory control algorithm for improving the communication throughput and delay in uav-aided networks. IEEE Network, 30(1), 100-105. https://doi.org/10.1109/mnet.2016.7389838.