ISSN 2959-6157

An Advanced Hardware Implementation of the Trial Division Algorithm for Dividers Utilizing RISC-V ISA

Runpu Wang

The Sino-British College, University of Shanghai for Science and Technology, Shanghai, 200000, China
SBC-22-1024@sbc.usst.edu.cn

Abstract:

In this paper, we try to design a divider based on RISC-V instruction set architecture (ISA). Since the hardware design of the divider in the ALU is very important for the CPU to implement the arithmetic function. In view of the advantages of the trial division method in the division algorithm and the popularity of the RISC-V ISA, this paper attempts to design a trial division divider utilizing the RISC-V ISA. In this design, the principle of trial division, flow chart, state diagram, RISC-V instructions of division, whole divider module and logic will be designed in detail. To test the function and feasibility, the hardware description is carried out in the Verilog hardware description language (HDL). In this description, four division RISC-V instructions are integrated. Modelsim is used to verify and test the functionality of the divider. The test waveform is used as a result to verify the feasibility of the divider.

Keywords: Divider, Trail division, RISC-V ISA

1. Introduction

In the development of modern computers and automated machines, MPU and CPU play an important role as core control units. From the bottom layer of the hardware, the transistor circuit constitutes the logic gate circuit, the gate-level circuit of a certain scale constitutes the logic unit with different functions, and the core device is a variety of logic gates. There are two main directions for the logic gate-based design method: combinatorial logic circuits and sequential logic circuits. The combination of these two design and implementation methods can be used to design more complex and diverse hardware units. One of the more well-known products is the CPU. Nowadays, with the advancement of CPUs, the development of computers and intelligent machinery such as mobile phones, intelligent equipment and new energy vehicles has reached an unprecedented stage. As the brain of a computer, the CPU Is responsible for complex control and operation tasks, such as performing arithmetic or logical operations, storing and invoking instructions, handling interruptions, and so on. As a core and central part, the arithmetic logic unit (ALU) is responsible for the actual computing tasks of the computer. [1] ALU is indispensable in a variety of processor units such as CPU, MPU, GPU, etc. The function of ALU is to be able to implement basic mathematical operations, such as addition, subtraction, multiplication, and division, and the combination of basic operations can be used to achieve more complex calculations. The premise of the implementation of these operations is the corresponding hardware circuit, which is called the ALU module as a whole. [2] From the ALU module, there are addition module, subtraction module, multiplication module and division module. Since binary operations can correspond to hardware circuits, the gate-level circuit design of the most basic adders and subtractors is relatively simple, while the hardware implementations of multipliers and dividers are relatively complex. The performance of the ALU is one of the criteria to measure the performance of the CPU, and it is important to design the computing module in the ALU with excellent performance. There are relatively many algorithms and hardware implementations of the current adder, subtractor, and multiplier modules. The operation principle of the divider is more complex, the algorithm and the hardware design are also very diverse. Division algorithms can be divided into five classes: digit recurrence, functional iteration, very high radix, table look up, and variable latency. [3]

This article uses the digit recurrence algorithm, the trial division method. The essence of the trial division method is to simulate the manual calculation of division, but it is also adapted to binary. Its calculation principle is simple, each iteration produces one-bit result, it is very efficient, and the hardware design is simple and easy to integrate.

From the point of view of practicality, the hardware of the divider needs to be designed based on a predetermined

instruction set architecture (ISA). The instruction set is an important bridge between software and hardware. [4, 5, 6] It is precisely because of the existence of the instruction set that the upper-layer software can send the instruction or instruction combination of the command set to the underlying hardware, and manipulate the hardware circuit to run to achieve the desired operation result.

A hardware architecture based on an instruction set design is called an instruction set architecture. Currently, it is divided into two groups, CISC (Complex Instruction Set Computing) and RISC (Reduced Instruction-Set Computer). [7] With the development of ISAs over the years, CISC was the first to appear, and complex combined instructions in this instruction set accounted for the majority. Many hardware circuits are specifically designed to implement complex functions with excellent performance, which is excellent. However, with the development of the disadvantages, only 20% of the instructions are used in 80% of the scenarios, and the utilization efficiency is low. Compatibility is considered in every generation of processor architecture upgrades, which leads to a further large instruction set, as exemplified by Intel's x86 architecture for CPUs. [8] Due to the increasing disadvantages of CISC, in the 1980s, the new RISC was born, which made the instruction set architecture develop in the direction of simplification, such as the familiar ARM architecture, and the emerging RISC-V instruction set architecture. Due to the business model, the mainstream X86 architecture and ARM architecture require commercial authorization to be designed and developed, and RISC-V, as an open-source and free architecture, is naturally welcomed and is also an important research and development object. Therefore, with the trend of RISC-V, it is of practical significance to study hardware based on RISC-V instruction set architecture. [9]

This paper aims at designing and implementing a divider utilizing the trial division. At the same time, to make it has practical reference significance, the RISC-V ISA is used in the Verilog HDL.

2. Methodology

2.1 Trial division algorithm

Trail division algorithm is a method derived from the digit recurrence. This method describes the paper-and-pencil calculation process, which are the most common and intuitional algorithms for people to design the hardware circuit. The process of division is given two numbers, dividend and divisor, to obtain two results, the quotient and the remain. In the mathematic, decimal arithmetic is used to implement the division process. However, a computer or divider in an ALU can only carry out the binary arithmetic through the transistor circuits which correspond to two states, open circuit and closed circuit. The design of a divider needs to depend on binary arithmetic division. In particular, the radix of binary is the number 2, and the radix of decimal is 10.

Table 1. Trial division for decimal in one iteration

$\overline{P_{m-1}\dots P_0}$ ⟌ $\overline{D_{n-1}\dots D_{n-m}D_{n-m-1}\dots D_1D_0}$ $\overline{Q_{l-1}\dots\dots Q_1Q_0}$ $(D, P, Q are decimal numbers)$

Trail of Q_{l-1}	Trail result	When	Result of Q_{l-1}
0	0	$Maximum of trail result \le D_{n-1}\dots\bar{D}_{n-m}D_{n-m-1}$	$Q_{l-1}=i$
1	$1\times P_{m-1}\bar{P}_{m-2}\dots P_1P_0$		
2	$2\times P_{m-1}\bar{P}_{m-2}\dots P_1P_0$		
…	……		
$i(i\le 9)$	$i\times P_{m-1}\bar{P}_{m-2}\dots P_1P_0$		

Table 1 shows the trail operation in one iteration. When conducting the integer decimal division, for m bits divisor and n bits dividend, the most significant bit (MSB) of the quotient starts from n-m-1 bit of dividend. In one iteration, the bit of quotient can be determined by the trial result, which means to try the number of this bit that the product of divisor and one bit of quotient can satisfy the condition, the trial result is less than the first n-m bits of dividend. Every iteration can determine one bit of quotient. This process is called trial division.

Table 2. Trial division for binary in one iteration

$\overline{P_{m-1}\ldots P_0}\ \overline{)\ D_{n-1}\ldots D_{n-m}D_{n-m-1}\ldots D_1D_0}\ \overline{Q_{l-1}\ldots\ldots Q_1Q_0}$ $(D,P,Qarebinarynumbers)$			
Trail of Q_{l-1}	Trail result	When	Result of Q_{l-1}
0	0	$Maximumoftrailresult \le D_{n-1}\ldots\bar{D}_{n-m}D_{n-m-1}$	$Q_{l-1}=i(i=0or1)$
1	$1\times P_{m-1}\bar{P}_{m-2}\ldots P_1P_0$		

After analyzing the decimal quotient principle, the process of binary quotient trial is essentially a degradation of the decimal quotient principle. As Table 2 shows, the decimal test quotient results are as high as ten, while the binary test results only require two trial results. Since the binary quotient process only involves two numbers, 0 and 1, it is equivalent to the comparison process, and only two results are obtained: larger than or less than.

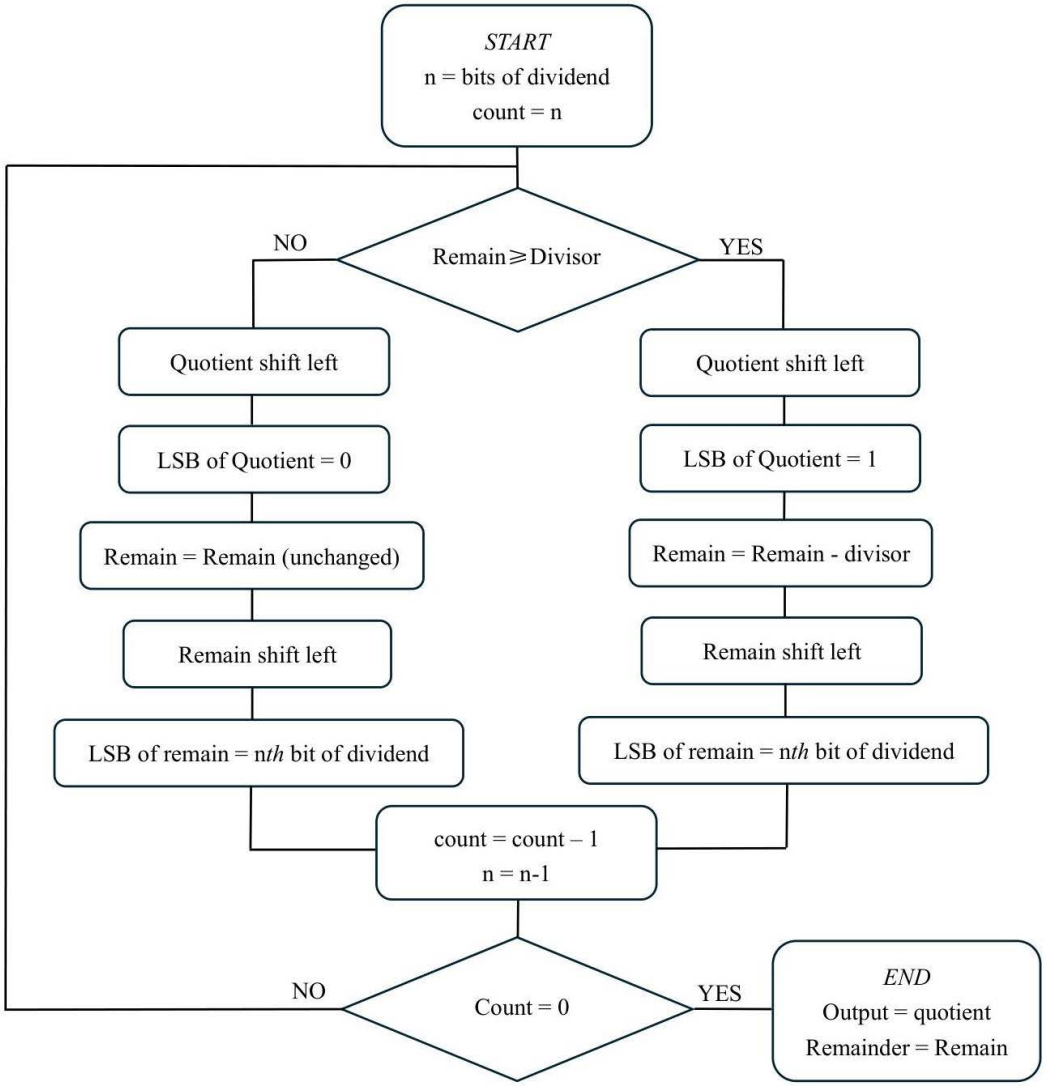

Fig. 1 Trial division flow chart

The whole process of the trial division method is as follows: Fig. 1 shows that the whole process involves com-

paring, data shifting, making differences, and assigning the new remains.

Step 1. After the dividend and divisor are loaded, the initial value of remain is 0.

Step 2. Compare the remain and divisor and get the two results which produce two branches.

Step 3. If remain is greater than or equal to the divisor, the quotient and the dividend shift left for one bit, the LSB of the quotient is 1, the remain's value is assigned to the difference between the remain and the divisor and shifts left, the LSB of the remain is replaced by the MSB of the dividend, if remain is less than the divisor, the quotient and the dividend shift left, the quotient's LSB is 0, the remain is shifted to the left and the LSB is replaced by the MSB of the dividend.

Step 4. Subtract the counter by one and determine whether the counter is zero.

Step 5. If the counter is not zero, repeat Step 2 to 4, if the counter is zero, the calculation is finished, and the quotient and remainder are obtained.

2.2 Division Operations in RISC-V ISA

Table 3. Instruction assembly format of RISC-V ISA

Instruction assembly format			
div rd, rsl, rs2	divu rd, rsl, rs2	rem rd, rsl, rs2	remu rd, rsl, rs2

RISC-V ISA defines the optional integer multiplication and division, as the "M" Standard Extension. There are four division operation instructions, DIVU/REMU and DIV/REM. DIV and DIVU perform an XLEN bits by XLEN bits signed and unsigned integer division of rs1 by rs2, rounding towards zero. REM and REMU provide the remainder of the corresponding division operation. For REM, the sign of the result equals the sign of the dividend. [10]

• The div instruction divides the 32-bit integers in the operand registers rsl and rs2, where the values in rsl and rs2 are both treated as signed numbers, and the quotient obtained by the division is written back to the register rd.

• The divu instruction divides the 32-bit integers in the operand registers rsl and rs2, where the values in rsl and rs2 are both treated as unsigned numbers, and the quotient obtained by the division is written back to the register rd.

• The rem instruction divides the 32-bit integers in the operand registers rsl and rs2, where the values in sl and rs2 are both treated as signed numbers, and the remainder obtained by the division is written back to the register rd.

• The remu instruction divides the 32-bit integers in the operand registers rsl and rs2, where the values in rs and rs2 are both treated as unsigned numbers, and the remainder obtained by the division is written back to the register rd.

2.3 Hardware implementation

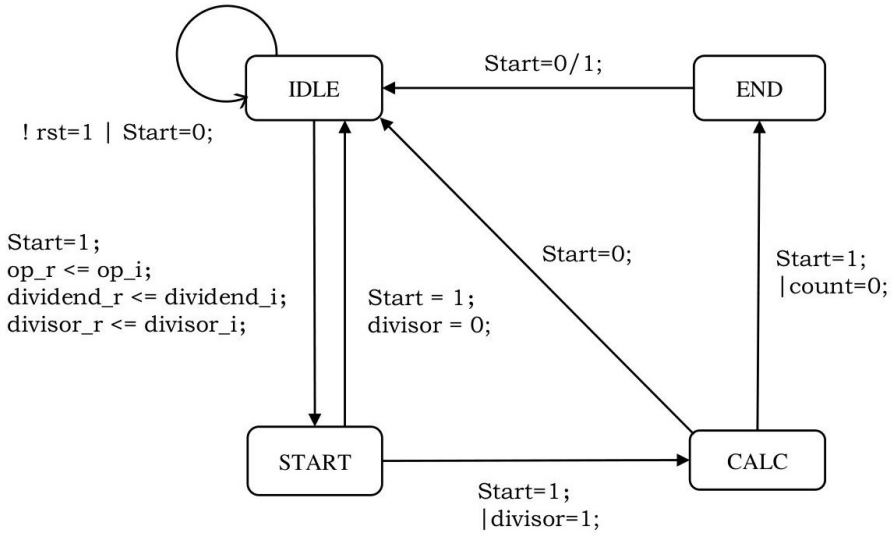

Fig. 2 State convert diagram

Four states can be defined:

1. *IDLE state:* free mode, the divider doesn't work, only loads the input of dividend and divisor. Other signals don't change until skip to another state.

2. *START state:* the divider starts working, after loading the dividend and divisor in IDLE state. The divisor is checked if it is zero. When it is zero, the divider skips to the IDLE state. If the divisor is not zero, then the sign is checked if it is negative. The divider will take the 2's complement of the dividend and divisor when they are negative.

3. *CALC state:* when the preparation of the start state has been done, the actual calculation process starts. This process executes the whole calculation process.

4. *END state:* when the calculation step ends, the state skips to the end state. If the result is in the form of 2's complement, then change it into the origin form. All are done, skip to IDLE state.

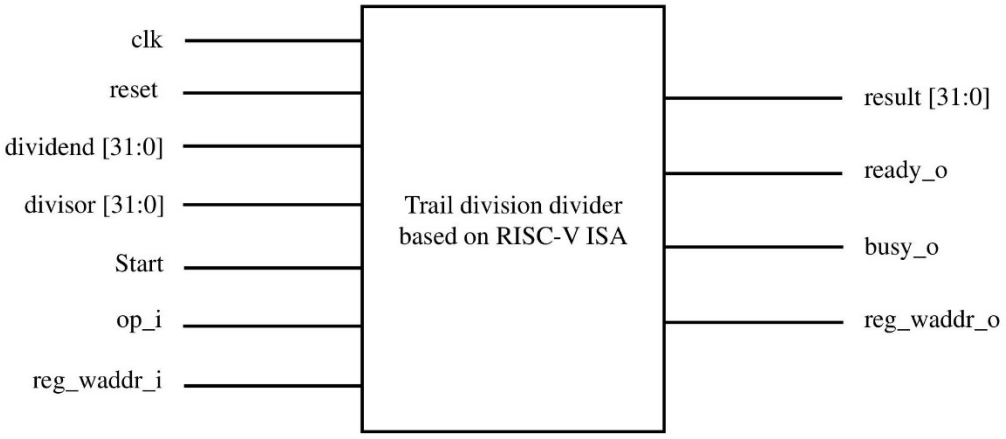

Fig. 3 divider module

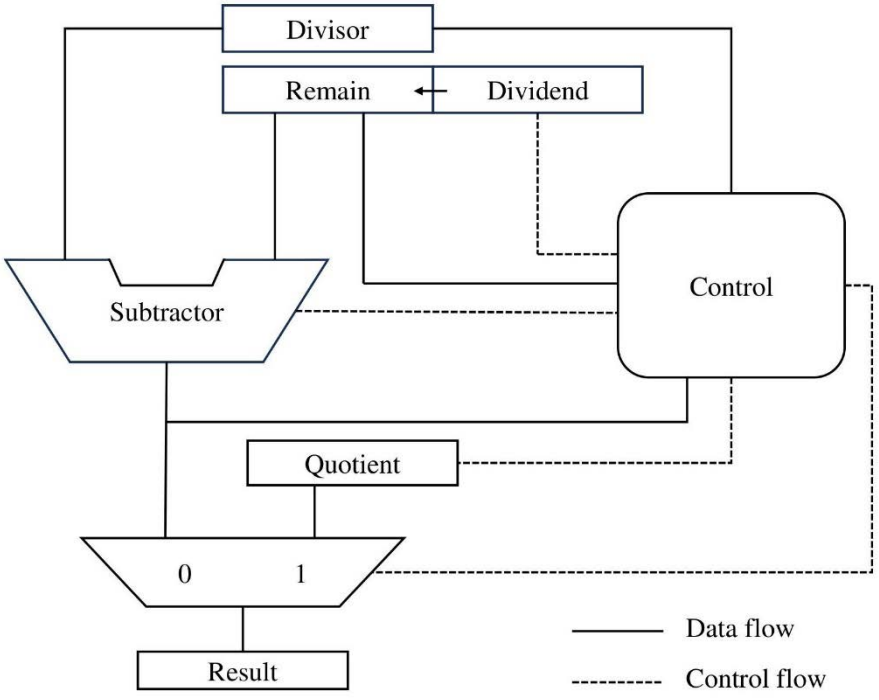

Fig. 4 Trail division logic

The divider is designed to be a 32-bit divider. The remain register is a 32-bit register whose initial value is zero. The dividend register can shift left into the remain register. The control unit controls when the subtractor works, when shifts the registers, when inputs new values, etc. The result can be quotient or reminder, which depend on the operation the divider executes.

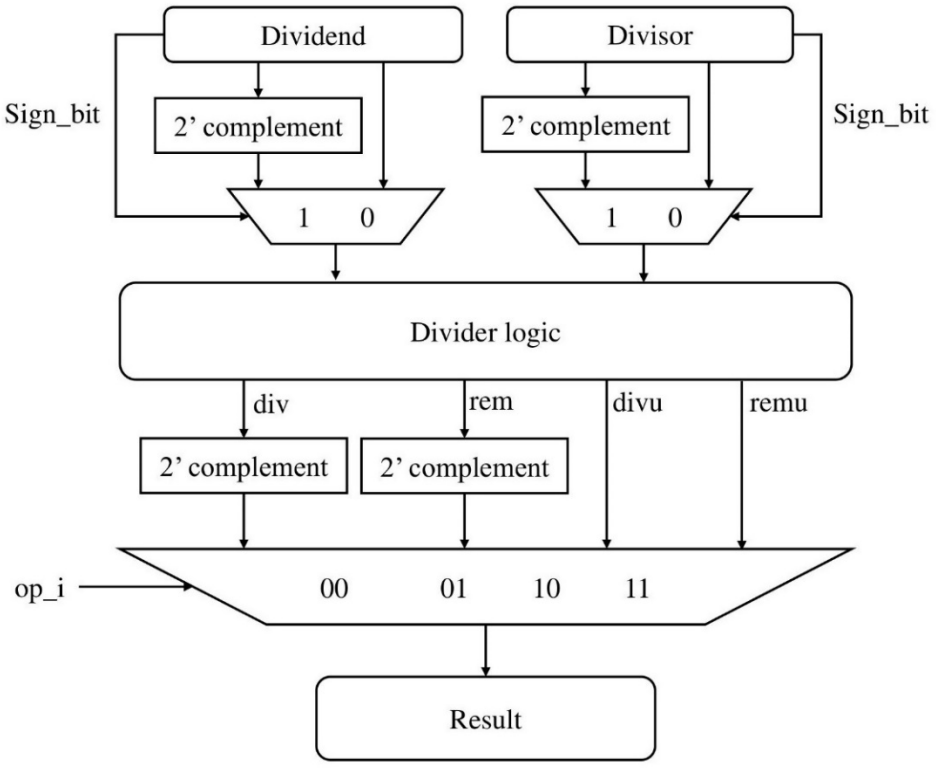

Fig. 5 Trail division structure chart

3. Test and model evaluation

3.1 Initial settings of Testbench

The entire architecture is modeled using VerilogHDL and the simulation is running on the Modelsim SE-64. To test the function of the divider, is to check if the four operations can work properly. Testbench environment setting: define the time-scale be 1ns, and clock cycle time be 10 time scales. Define the input and output. Run the instructions in order, first to be 'divu', set the dividend integer be 8 and divisor be 3. Second to be 'div', set the dividend integer to be negative 8 and divisor to be 3. Third to be 'rem', set the dividend integer to be negative 8 and divisor to be 3. Fourth to be 'remu', set the dividend integer be 8 and divisor be 3.

3.2 Instruction code of four operations

(1) op_div = (op_r == 3'b100); //*op_r == 100*
(2) op_divu = (op_r == 3'b101); //*op_r == 101*
(3) op_rem = (op_r == 3'b110); //*op_r == 110*
(4) op_remu = (op_r == 3'b111); //*op_r == 111*

3.3 Wave diagram

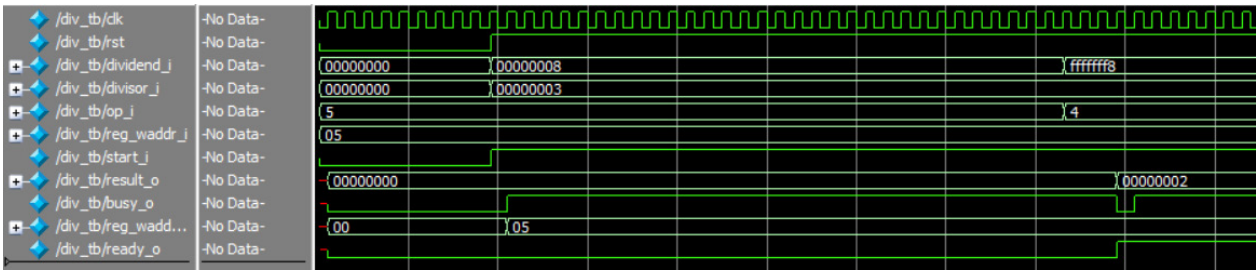

Fig. 6 Result of 'divu' instruction

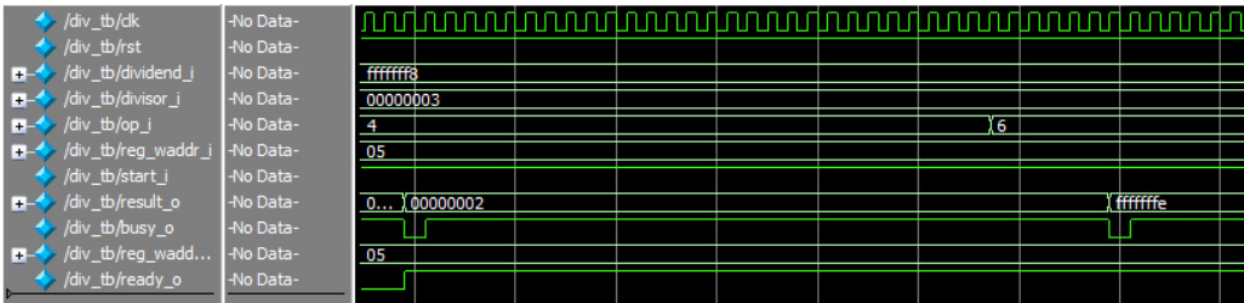

Fig. 7 Result of 'div' instruction

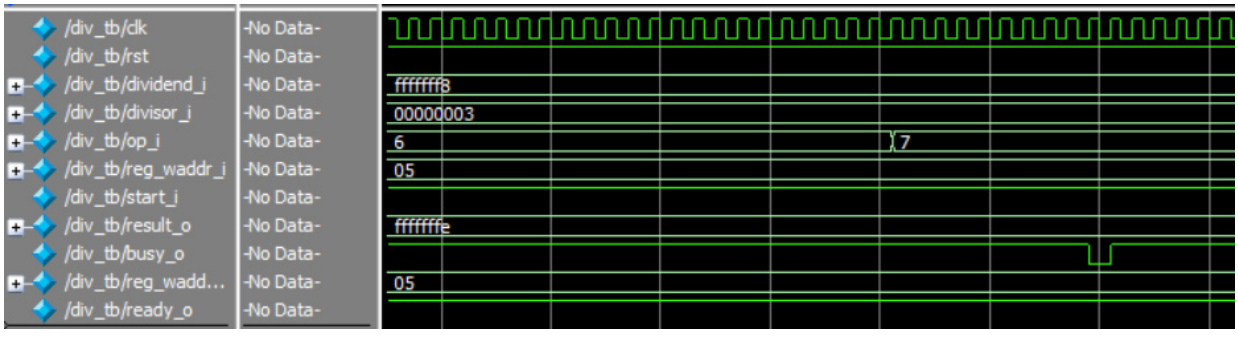

Fig. 8 Result of 'rem' instruction

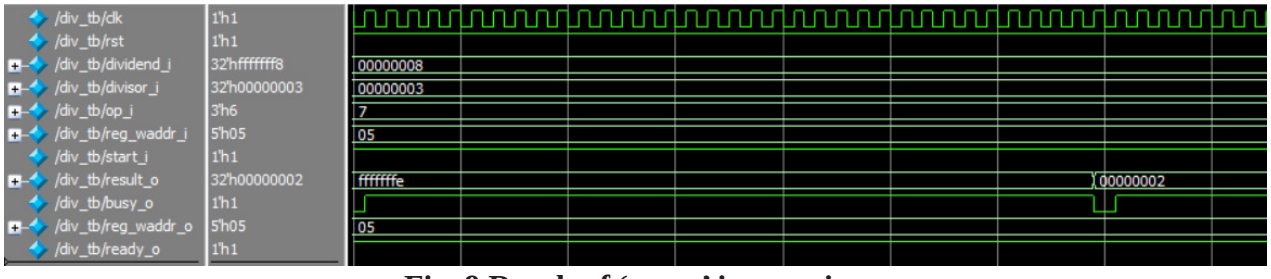

Fig. 9 Result of 'remu' instruction

3.3 Interpretation of results

• When the rst (clear) signal and the Start signal jump high, the divider starts working.

• As Fig. 6, when the op_i is serial data 101, the quotient operation of the unsigned number is performed. The dividend is assigned a value of 8 and the divisor is 3. The quotient is 2.

• When the op_i is 100 for serial data, the quotient operation of the signed number is performed.

• As Fig. 7, The dividend is assigned as ffffff8 (i.e. minus 8) and the divisor is still 3. After the operation, the quotient is fffffffe (i.e. minus 2).

• When the op_i is serial data 110, perform the remainder operation of the signed number.

• As Fig. 8, The dividend is assigned as ffffff8 and the divisor is 3. After the calculation, the remainder is fffffffe.

• When the op_i is serial data 111, perform the remainder operation of the unsigned number.

• As Fig. 9, The dividend is assigned as 8 and the divisor is 3. After the calculation, the remainder is 2.

3.4 Model evaluation

The divider module implements the division operation and remainder operation in the basic arithmetic operation. Through the operation of the testbench program and the waveform simulation results of Modelsim, it can be seen that the trial division divider based on the RISC-V instruction set architecture successfully realizes the four division functions defined in the RISC-V instruction set. With a 32-bit wide input, the division process takes a lot of clock cycles, which consumes a large number of clock cycles, which is undoubtedly a disadvantage of this divider. However, compared with the dividers constructed by other division algorithms, the divider of the trial quotient method has the advantage of a simple principle. At present, the divider can only do integer division, and floating-point division is not implemented, which requires an advanced algorithm design.

4. Conclusion

This paper attempts to implement a divider utilizing the algorithm of trail division and combines the four integer division instructions involved in the existing emerging RISC-V instruction set architecture. From the results, it can be seen that the Verilog hardware language description of the test quotient divider is successful, and it also proves that the test quotient division based on the RISC-V instruction set can be implemented in hardware.

5. Reference

[1] L. Gopal, N. S. Mohd Mahayadin, A. K. Chowdhury, A. A. Gopalai and A. K. Singh, "*Design and synthesis of reversible arithmetic and Logic Unit (ALU),*" 2014 International Conference on Computer, Communications, and Control Technology (I4CT), Langkawi, Malaysia, 2014, pp. 289-293, doi: 10.1109/I4CT.2014.6914191.

[2] S. Purohit, P. Laddha and R. Parekh, "*Implementation and Physical Design of 8/4-Bit Signed Divider,*" *2021* 8th International Conference on Signal Processing and Integrated Networks (SPIN), Noida, India, 2021, pp. 829-834, doi: 10.1109/SPIN52536.2021.9566020.

[3] S. F. Obermann and M. J. Flynn, "*Division algorithms and implementations,*" in IEEE Transactions on Computers, vol. 46, no. 8, pp. 833-854, Aug. 1997, doi: 10.1109/12.609274.

[4] Patterson D A, Hennessy J L. Computer organization and design: the hardware/software interface 5th ed[J]. 2014.

[5] A. Akram and L. Sawalha, "*A Study of Performance and Power Consumption Differences Among Different ISAs,*" 2019 22nd Euromicro Conference on Digital System Design (DSD), Kallithea, Greece, 2019, pp. 628-632, doi: 10.1109/DSD.2019.00098.

[6] M. Ling, X. Xu, Y. Gu and Z. Pan, "*Does the ISA Really Matter? A Simulation Based Investigation,*" 2019 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM), Victoria, BC, Canada, 2019, pp. 1-6, doi: 10.1109/PACRIM47961.2019.8985059.

[7] A. D. George, "*An overview of RISC vs. CISC,*" [1990] Proceedings. The Twenty-Second Southeastern Symposium on System Theory, Cookeville, TN, USA, 1990, pp. 436-438, doi: 10.1109/SSST.1990.138185.

[8] Y. He and X. Chen, "*Survey and Comparison of Pipeline of Some RISC and CISC System Architectures,*" 2023 8th International Conference on Computer and Communication Systems (ICCCS), Guangzhou, China, 2023, pp. 785-790, doi: 10.1109/ICCCS57501.2023.10150975.

[9] D. -T. Nguyen-Hoang, K. -M. Ma, D. -L. Le, H. -H. Thai, T. -B. -T. Cao and D. -H. Le, "*Implementation of a 32-Bit RISC-V Processor with Cryptography Accelerators on FPGA and ASIC,*" 2022 IEEE Ninth International Conference on Communications and Electronics (ICCE), Nha Trang, Vietnam, 2022, pp. 219-224, doi: 10.1109/ICCE55644.2022.9852060.

[10] Waterman A, Asanović K. The RISC-V Instruction Set Manual, Volume I: Unprivileged ISA, Document Version 20191213. SiFive Inc., University of California, Berkeley, 2019.