Design and Verification of Three-stage Pipeline CPU Based on RISC-V Architecture

Wendi Zhang
Hainan University
College of Information and Communication Engineering
Haikou, China
wdzhang@hainanu.edu.cn

Yonghui Zhang
Hainan University
College of Information and Communication Engineering
Haikou, China
yhzhang@hainanu.edu.cn

Kun Zhao
Hainan University
College of Information and Communication Engineering
Haikou, China
kunzhao@hainanu.edu.cn

2021 5th Asian Conference on Artificial Intelligence Technology (ACAIT) | 978-1-6654-2630-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/ACAIT53529.2021.9731161

***Abstract*—RISC-V is a new open-source instruction set architecture, which has received extensive attention from the industry, and it is explored and designed in this context. Based on the RISC-V instruction set architecture, this paper designs a processor that supports a subset of the RV32IM instruction. It uses a three-stage pipeline technology, namely, value, decoding, and execution modules, with static branch prediction and Harvard storage structure. The main modules are pipeline module, control module, interrupt exception and storage module. Iverilog was used for simulation testing and passed the RISC-V test case. The final result showed that the processor can run normally at a clock frequency of 50MHz.**

***Index Terms*—RISC-V architecture, three-stage pipeline, processor**

I. Introduction

With the evolution of processor design technology and the development of large-scale integrated circuit design technology, now we have entered the era of processors, and the CPU is the core, integrated circuits and memories complete the main functions of information processing in the system. The CPU is the core component of the processor, and how to design and implement an effective processor has become a key technology [1].

At the same time, the demand for microelectronics technology in the embedded field has increased year by year, which has promoted the development of the RISC-V instruction set. It has the advantages of free and open source, minimalism, modularity and customizable expansion, which is undoubtedly a good chance for the processor industry. A rare good opportunity [2].The integrated circuit industry is a national strategic industry, and it is the source and power to promote the development of the information industry.The processor design has to consider the assembly line design. The classic assembly line has five stages, and the more the number of pipeline stages, the better is the throughput but the overhead will be greater, so this article uses the most common three-stage pipeline depth in ARM to achieve the goal of processor design.

Based on the RISC-V architecture, this design researched a processor core that supports a subset of RV32IM instructions, including 47 basic integer instructions and 8 extended integer multiplication and division instructions, using three-stage pipeline technology, and finally realized a RISC-V instruction set sequence, single launch, single-core 32-bit processor, and the processor was simulated and verified to achieve the predetermined goal.

∗ The corresponding author is Y. H. Zhang.

II. RISC-V ARCHITECTURE AND PIPELINE TECHNOLOGY

A. RISC-V instruction set

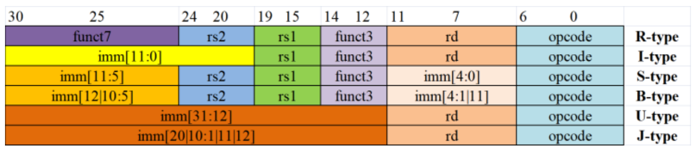
RISC-V is an open-source instruction set architecture (ISA) based on the reduced instruction set computer (RISC) principle. The fixed 47-instruction RV32I is used as the core. It is also the only module that RISC-V requires the processor to support. Only the RV32I instruction subset module can run a complete software stack. The other instruction subsets are all available selected modules. The representative modules include M, A, F, D, C [3]. RISC-V architecture instruction set modularization chooses different configurations of RISC-V instruction sets to achieve, as shown in Table 1.

B. RISC-V instruction format

The instruction length of RISC-V is 32 bits. An instruction is composed of several parts of separate numbers that is all composed of 0 and 1. Each part has its specific function. In the RISC-V instruction set, these individual numbers are fixed in the same position in different instruction types, and the digital composition of related instructions also has similarities, which greatly reduces the complexity of processor design [4]. The instruction set format used in this design is shown in Figure 1.

TABLE I
MODULARIZATION OF RISC-V INSTRUCTION SET ARCHITECTURE

Basic instruction set	Number of instructions	Description
RV32I	**47**	**The most basic instruction set, supports 32-bit addressing space and 32-bit integer registers**
RV32E	**47**	**A subset of RV32I, the number of integer registers is reduced by 16**
RV64I	**59**	**Supports 64-bit address space and 64-bit integer registers**
RV128I	**71**	**Support 128-bit address space and 128-bit integer register**
M	**8**	**Support integer multiplication and division instructions**
A	**11**	**Support storage atomic operation (Atomic) and load-reserved/store-conditional instructions**
F	**26**	**Support single-precision floating -point instructions**
D	**26**	**Support double-precision floating -point instructions (provided that F extended instructions are supported)**
C	**46**	**Support compressed instructions with a length of 16 bits**



30 25	24 20	19 15	14 12	11 7	6 0	
funct7	rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]		rs1	funct3	rd	opcode	I-type
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]	rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]				rd	opcode	U-type
imm[20|10:1|11|12]				rd	opcode	J-type

Fig. 1. RISC-V regular instruction encoding format.

Among them, opcode is the operation code, which is used to indicate the instruction operation and distinguish the field of the instruction format. The funct3 and funct7 parts serve as additional opcode fields, and opcode jointly determine the specific operation of the instruction . Rs1 and rs2 are used as register numbers. According to the instruction format, it can be seen that the rs1, rs2 and rd fields in all instructions are kept in the same position. At the same time, the opcode and funtc3 fields keep the same size and are in the same position. RISC-V is extremely large simplify hardware design.

C. Pipeline technology

The execution process of RISC-V instructions generally needs to go through five stages, which are fetching, decoding, executing, fetching, and writing back. The five stages of instruction execution are shown in Figure 2. Assuming that these five steps can be completed in time T, if pipeline technology is used to execute all stages of the three instructions, it will take 7T. If these three instructions are executed in a single cycle and non-pipelined situation, it will take 15T. It can be seen that pipeline technology is greatly increased the number of instructions executed, thereby improving work efficiency. Usually, digital system clock frequency and performance improvement can be achieved through pipeline technology, and pipeline technology is also the core technology of processor design [5].

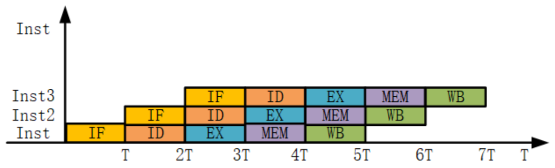


Fig. 2. Pipeline instruction execution process.

III. RISC-V PROCESSOR DESIGN

The processor structure of this design is shown in Figure 3. Based on the RISC-V instruction set, a three-stage pipeline SoC is designed, which is mainly composed of RISC-V Core (CPU), debugging unit, bus, and peripherals. The processor structure is Harvard architecture, which refers to the separate processing of instruction memory and data memory. Instructions and data are stored in different memories. Compared with the traditional von Neumann architecture, the processor accesses to instructions and data from each other independent and can run in parallel, which greatly improves system performance [6]. The schematic diagram of the RISC-V core is shown in Figure 4.

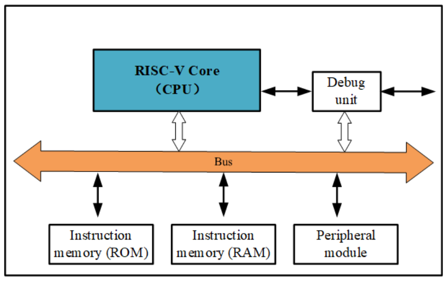


Fig. 3. RISC-V SoC structure diagram.

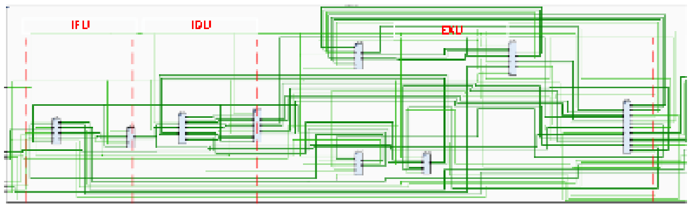


Fig. 4. CPU schematic.

The CPU uses the verilog hardware description language to implement, supports the RV32IM instruction set, and uses a three-stage pipeline design, that is, value, decoding, and execution. The following is a brief introduction to each module of the pipeline.

pc_reg: PC register module, used to generate the value of the PC register, the value will be used as the address signal of the instruction memory.

if_id: Fetch the instruction to the module between the decoding, used to beat the instruction output from the instruction memory and send it to the decoding module.

id: Decoding module, pure combinational logic circuit, decodes according to the instructions sent by the if_id module. When a specific instruction (such as an add instruction) is decoded, a signal of whether to write a register or not to read a register is generated. Finally, the data will be sent to the id_ex module together with the write register signal.

id_ex: The module between decoding and execution, which is used to send the signal of whether to write the register and the register data to the execution module.

ex: Execution module, perform corresponding operations according to specific instructions, such as add instruction to perform addition operations, etc.

A. *Finger fetching module design*

Fetching refers to reading the instruction coded by 0 and 1 in the instruction memory for the processor core to execute [7]. The instruction fetch unit (IFU) of this design is mainly composed of a program counter PC, an instruction memory ROM and a static branch prediction unit. To avoid taking out the branch jump instruction to pause the pipeline until the execution stage judges whether to jump, which causes a serious speed drop of the processor. This design adds static branch prediction to predict branch jump instructions. It always predicts that branch jump instructions will not jump. As long as the prediction is wrong, that is, branch jump instructions jump, the pipeline will stop, so only stop one clock cycle, at which time the PC is updated to the jump address sent from the execution stage. Compared with most RISC architecture processors, when running a program with a branch jump function, you need to implement this function through two instructions. The first is the comparison instruction, and then the jump instruction. The RISC-V architecture only need one instruction to complete the comparison and jump functions, which simplifies the hardware design.

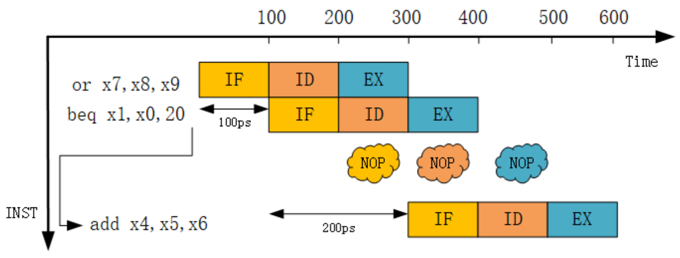


Fig. 5. The pipeline when the conditional branch jumps.

As shown in Figure 5, assuming that the conditional branch in the figure jumps, and the branch target address of the jump is an add instruction, a pause or bubble will be inserted after the branch jump instruction. In fact, the inserted bubble is actually a NOP instruction , that is, a null instruction, the instruction that has been fetched is invalid and becomes a NOP instruction.

B. *Decoding module design*

The decoding stage is mainly to translate the instructions fetched from the instruction fetch unit and send the relevant decoding information to the execution unit. The decoding information generally includes the registration number to be read and written back, also known as the register index (Index) [8]. The decoding module is a pure combinational logic circuit, which decodes according to the instructions sent by the value module. When decoding the specific instruction (such as add instruction), generated whether to write register signal, read register signal, etc. The register adopts the asynchronous read mode, so as long as the read register signal is sent, the corresponding register data will be immediately obtained, and this data will be sent to the execution module together with the write register signal.

The integer general register bank module is mainly used to implement the integer general register bank defined by the RISC-V architecture. The general register bank designed in this article is shown in Figure 6 below. The instruction used in this design has at most two operands, so the general register group only needs to support at most two read registers, and only one data is written back at a time, so this module only needs one write register. The input end of the general register group needs three register numbers and one data to be written back, and the output is the data of the corresponding register output according to the input register number.

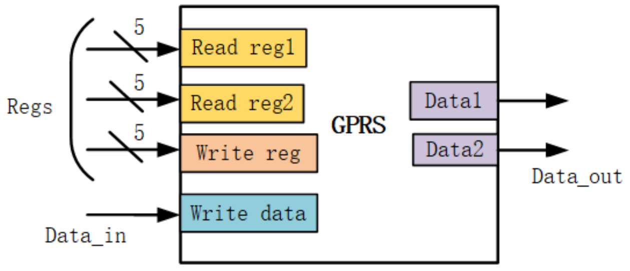


Fig. 6. General Register Group.

C. *Execution module design*

In the execution stage, it performs corresponding operations according to the specific instruction types and related operations sent from the decoding stage [9]. The processor of this design is a three-stage pipeline, that is, the memory access and write-back phases are completed in the execution module. The main functions of the execution module include arithmetic logical unit (ALU), load and store instruction memory access address calculation, integer multiplication and division operations, data-related solutions, CRS read-write control and static branch prediction analysis. The framework of the execution module is shown in Figure 7.

The ALU operation does not involve memory access, so after the result is calculated in the operation data path, it is written back to the destination register. This process is relatively simple. There are 8 memory access instructions in the RISC-V instruction set including 5 load instructions, and

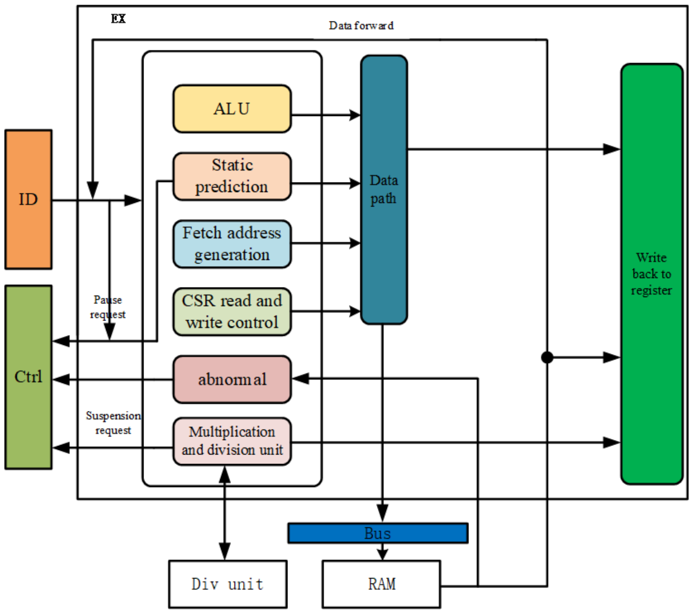


Fig. 7. Block diagram of execution module.

3 store instructions, while other instructions cannot access the memory. The assembly format of load and store instructions is shown in Figure 8. Load and store instructions add the value of the base address register rs1 and the 12-bit signed immediate data contained in the instruction to obtain the memory address. After the execution module receives the load and store instruction information sent from the decoding stage, it will be sent to the load and store. The instruction memory access address generation unit, which is mainly responsible for the calculation of the memory access address, request access to the bus and read and write data memory RAM addresses.

```
lw   rd, offset(rs1)
lh   rd, offset(rs1)
lhu  rd, offset(rs1)
lb   rd, offset(rs1)
lbu  rd, offset(rs1)
sw   rd2, offset(rs1)
sh   rd2, offset(rs1)
sb   rd2, offset(rs1)
```

Fig. 8. Assembly format of Load and Store instructions.

The integer multiplication and division unit use a single-cycle multiplier and an independent multi-cycle division module DIV (Divider). The division module DIV uses a trial quotient method. A division operation requires dozens of clock cycles to complete because the division instruction is multi-cycle Instruction, so when encountering the division instruction information, the execution module sends a pipeline pause signal to the pipeline control module and then cancels the pause signal after the calculation is completed.

One problem that the execution module needs to solve is the data hazard problem [10]. The data hazard involved in this article mainly occurs in the program related to the load instruction. Because the load instruction fetches the data during the memory fetching stage, it also needs to stop the pipeline, also called insert the bubble, as shown in Figure 9. It is a path diagram of the pre-loading instruction data, where it is assumed that the load instruction and the next add instruction have data correlation.

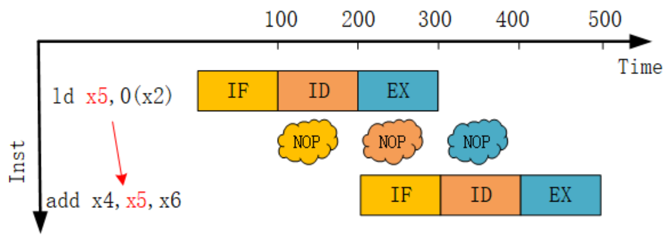


Fig. 9. Data forwarding path diagram of Load instruction.

D. Jump mechanism

Jump is to change the value of the PC register. The jump or not will be known at the execution stage. When a jump is needed, the pipeline will be flushed. Its structure is shown in Figure 10.

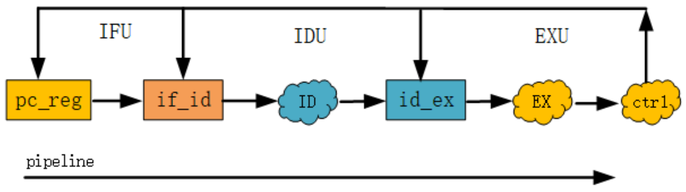


Fig. 10. Jump structure diagram.

In the execution stage, when it is judged that a jump needs to occur, the jump signal and jump address are sent to the ctrl module. After the ctrl module judges that the jump signal is valid, it will send a pipeline pause signal to the pc_reg, if_id and id_ex modules, and also send a jump address to the pc_reg module. When the rising edge of the clock arrives, if_id and id_ex modules detect that the pipeline pause signal is valid, they will send NOP instructions, so that the entire pipeline (decoding stage, execution stage) flows through NOP instructions, and the instructions that have been fetched will be invalid. This is the flushing mechanism of the assembly line.

E. Interrupt exception

The exception signal of this design mainly comes from the instructions of the execution and write-back unit. The instructions in the write-back phase are mainly for the load instruction. If no exception occurs, the instruction stream runs normally in the pipeline. If an exception occurs, a pipeline stall request will be sent to the pipeline control module, and the PC value updated in the mepc register is the PC where the abnormal instruction occurred [11]. The interrupt signal of this design mainly comes from software interrupts, the timer interrupts and interrupt sources managed in PLIC. If an interrupt occurs, the pipeline pause request will be sent to the pipeline control module. The PC value updated in the mepc register is also the PC that generated the interrupt instruction, so that after exiting the interrupt exception, the program that was previously aborted can continue to be executed. The interrupt exception structure is shown in Figure 11.

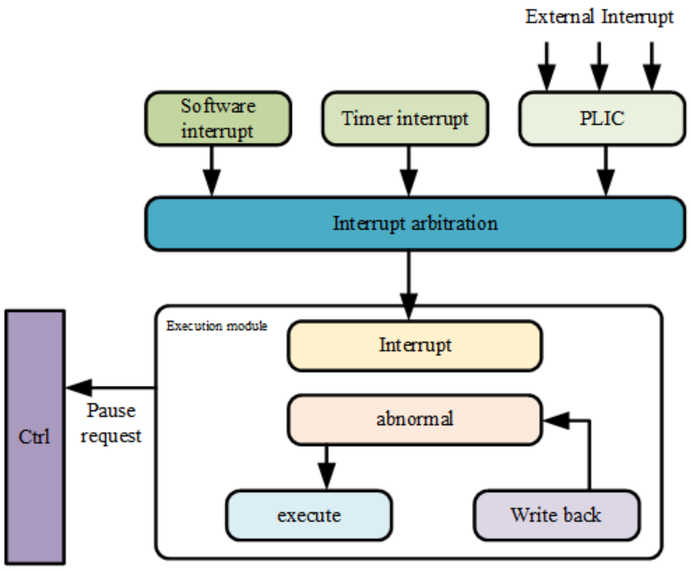


Fig. 11. interrupt exception structure.

F. Debug module

The debugging method officially supported by RISC-V is JTAG (Joint Test Action Group). The standard JTAG adopts a four-wire method, which are TCK (test clock input), TMS (test mode selection), TDI (test data input) and TDO (test data input). Data output), there is an optional TRST pin. The CMSIS-DAP debugger is used in this design [12].

RISC-V system debugging framework debug host, debug hardware and debugging module. Inside the debugging module, the DTM module directly interacts with the debugging tool, and the DTM module interacts with the DM module through the DMI interface. In this design, the debugger host computer openocd officially supported by RISC-V is used to update the application program. When openocd is started, a cfg file needs to be formulated through the -f parameter to start openocd, realize debugging, and carry out embedded application RISC-V The realization procedure of JTAG is shown as in Figure 12.

IV. VERIFICATION

The processor is simulated and verified, and the RISC-V SoC is stimulated through a TestBench test platform. Generally, the RISC-V SoC design itself is called DUT (Design Under Test). The test platform TestBench is usually written by

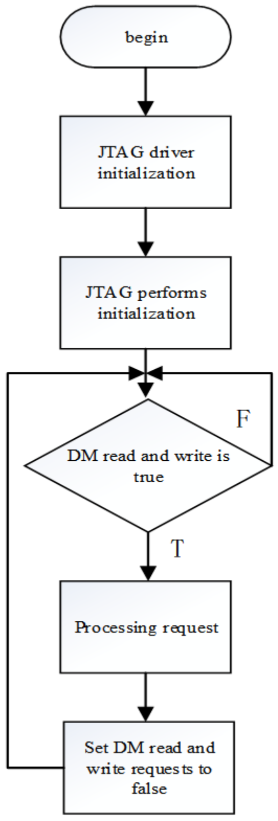

begin
JTAG driver initialization
JTAG performs initialization
DM read and write is true
F
T
Processing request
Set DM read and write requests to false

Fig. 12. JTAG implementation process.

hardware description language verilog and surrounds the DUT. Its purpose is to simulate and verify the designed circuit, that is, the DUT, to accelerate the understanding of circuit design. The TestBench test platform is shown in Figure 13.

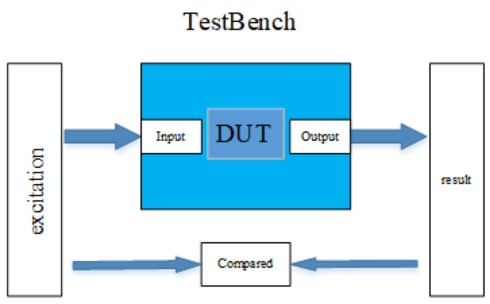


Fig. 13. TestBench test platform.

A. FPGA logic resource occupation

The occupancy of FPGA wiring and logic resources synthesized using Vivado tools is shown in Figures 14 and 15. From the FPGA resource occupancy rate, the occupied resources are good.

B. Run CoreMark

Download the compiled coremark running program to the processor through the OpenOCD debugging software and

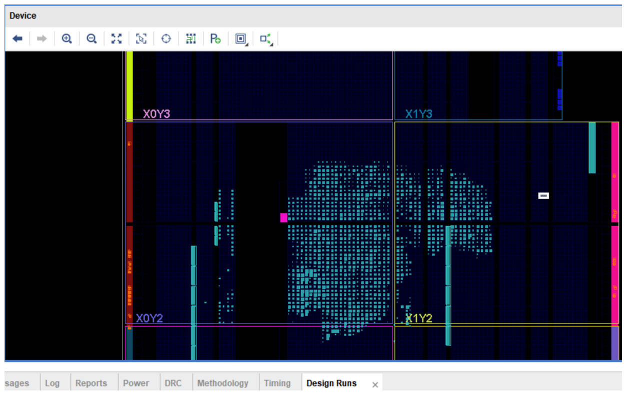
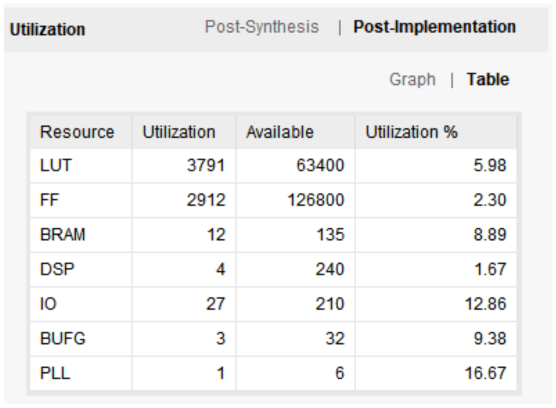


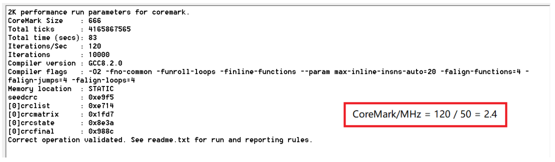
Fig. 14. Resource distribution


Utilization
Post-Synthesis
Post-Implementation
Graph
Table

Resource	Utilization	Available	Utilization %
LUT	3791	63400	5.98
FF	2912	126800	2.30
BRAM	12	135	8.89
DSP	4	240	1.67
IO	27	210	12.86
BUFG	3	32	9.38
PLL	1	6	16.67

Fig. 15. Status of resources occupied

JTAG debugger. After running the coremark running program, the corresponding information will be displayed on the host serial terminal. The serial port printing information is shown in Figure 16, which can be seen the processor's running score is 2.4.



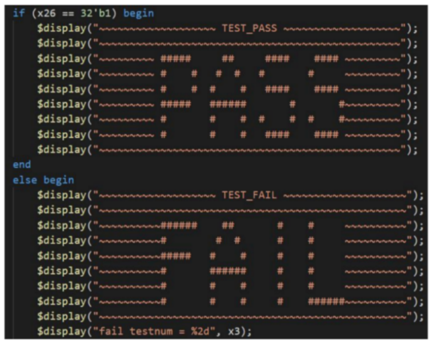
```
2K performance run parameters for coremark.
CoreMark Size    : 666
Total ticks      : 4165867565
Total time (secs): 83
Iterations/Sec   : 120
Iterations       : 10000
Compiler version : GCC8.2.0
Compiler flags   : -O2 -fno-common -funroll-loops -finline-functions --param max-inline-insns-auto=20 -falign-functions=4 -
falign-jumps=4 -falign-loops=4
Memory location  : STATIC
seedcrc          : 0xe9f5
[0]crclist       : 0xe714
[0]crcmatrix     : 0x1fd7
[0]crcstate      : 0x8e3a
[0]crcfinal      : 0x988c
Correct operation validated. See readme.txt for run and reporting rules.
```

CoreMark/MHz = 120 / 50 = 2.4

Fig. 16. Run CoreMatk

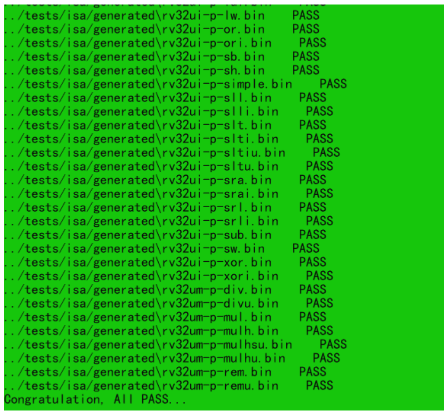
C. TestBench runs test cases

The RISC-V test case (Self-Check-Testcase) is a test program that can self-check whether it runs successfully or fails. riscv-test is an open-source project maintained by the developers of the RISC-V architecture. It contains some test programs that test whether the processor conforms to the instruction set architecture definition. These test programs are written in assembly language [13]. We created a simple TestBench test platform written by Verilog, used the platform to run test cases, parsed the name of the test case according to the test command, and analyzed whether the test case was executed successfully after the run in the source file of TestBench judge the value of the x27 register. If the value of x27 is 1, it means pass. Print the word PASS on the terminal, otherwise, it will print the word FAIL. The result of the test case printed in TestBench is shown in Figure 17. The result of running the test set As shown in Figure 18. In the process of running test cases, the simulator tool iverilog is used.



```
if (x26 == 32'b1) begin
    $display("~~~~~~~~~~~~~~~~~~~~ TEST_PASS ~~~~~~~~~~~~~~~~~~~~");
    $display("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
    $display("~~~~~~~~~ #####     ##     ####    #### ~~~~~~~~~");
    $display("~~~~~~~~~ #    #   #  #   #        #     ~~~~~~~~~");
    $display("~~~~~~~~~ #    #  #    #   ####    #### ~~~~~~~~~");
    $display("~~~~~~~~~ #####   ######       #       #~~~~~~~~~");
    $display("~~~~~~~~~ #       #    #  #    #  #    #~~~~~~~~~");
    $display("~~~~~~~~~ #       #    #   ####    #### ~~~~~~~~~");
    $display("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
end
else begin
    $display("~~~~~~~~~~~~~~~~~~~~ TEST_FAIL ~~~~~~~~~~~~~~~~~~~~~");
    $display("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
    $display("~~~~~~~~~~######    ##      #    #      ~~~~~~~~~~");
    $display("~~~~~~~~~~#        #  #     #    #      ~~~~~~~~~~");
    $display("~~~~~~~~~~#####   #    #    #    #      ~~~~~~~~~~");
    $display("~~~~~~~~~~#       ######    #    #      ~~~~~~~~~~");
    $display("~~~~~~~~~~#       #    #    #    #      ~~~~~~~~~~");
    $display("~~~~~~~~~~#       #    #    #    ######~~~~~~~~~~");
    $display("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
    $display("fail testnum = %2d", x3);
```

Fig. 17. TestBench test case.

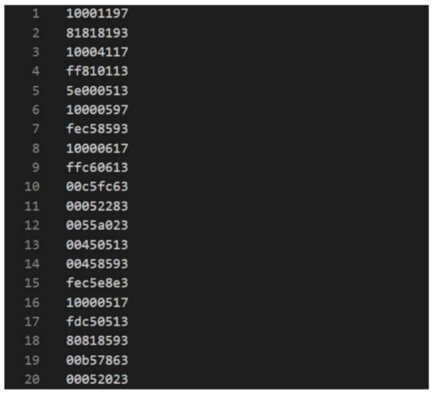


```
../tests/isa/generated\rv32ui-p-lw.bin    PASS
../tests/isa/generated\rv32ui-p-or.bin    PASS
../tests/isa/generated\rv32ui-p-ori.bin    PASS
../tests/isa/generated\rv32ui-p-sb.bin    PASS
../tests/isa/generated\rv32ui-p-sh.bin    PASS
../tests/isa/generated\rv32ui-p-simple.bin    PASS
../tests/isa/generated\rv32ui-p-sll.bin    PASS
../tests/isa/generated\rv32ui-p-slli.bin    PASS
../tests/isa/generated\rv32ui-p-slt.bin    PASS
../tests/isa/generated\rv32ui-p-slti.bin    PASS
../tests/isa/generated\rv32ui-p-sltiu.bin    PASS
../tests/isa/generated\rv32ui-p-sltu.bin    PASS
../tests/isa/generated\rv32ui-p-sra.bin    PASS
../tests/isa/generated\rv32ui-p-srai.bin    PASS
../tests/isa/generated\rv32ui-p-srl.bin    PASS
../tests/isa/generated\rv32ui-p-srli.bin    PASS
../tests/isa/generated\rv32ui-p-sub.bin    PASS
../tests/isa/generated\rv32ui-p-sw.bin    PASS
../tests/isa/generated\rv32ui-p-xor.bin    PASS
../tests/isa/generated\rv32ui-p-xori.bin    PASS
../tests/isa/generated\rv32um-p-div.bin    PASS
../tests/isa/generated\rv32um-p-divu.bin    PASS
../tests/isa/generated\rv32um-p-mul.bin    PASS
../tests/isa/generated\rv32um-p-mulh.bin    PASS
../tests/isa/generated\rv32um-p-mulhsu.bin    PASS
../tests/isa/generated\rv32um-p-mulhu.bin    PASS
../tests/isa/generated\rv32um-p-rem.bin    PASS
../tests/isa/generated\rv32um-p-remu.bin    PASS
Congratulation, All PASS...
```

Fig. 18. Test set.

D. ROM and RAM simulation test

ROM and RAM are responsible for storing programs and data [14]. In order to verify the functions of ROM and RAM, some test programs need to be used, then compiled into executable .bin files, and finally converted into inst_rom which can be read by the system function readmenh. data file, as shown in Figure 19, it is a partial example of the test program that prints out the words "Hello RISC-V" from the serial port UART into the instructions in the ins_rom.data file. The simulation waveform diagram of ROM and RAM memory and bus interface is shown in Figure 20. The signal data_o is the output of ROM and RAM memory respectively.The ROM and RAM memories are respectively connected to the slave device interfaces s1 and s2 of the bus. It can be seen from the simulation waveform that the value of data_o is equal to the value of the corresponding slave device interface, that is, the ROM and RAM memories can work normally.



```
10001197
81818193
10004117
ff810113
5e000513
10000597
fec58593
10000617
ffc60613
00c5fc63
00052283
0055a023
00450513
00458593
fec5e8e3
10000517
fdc50513
80818593
00b57863
00052023
```

Fig. 19. Example of instruction in inst_rom.data file.

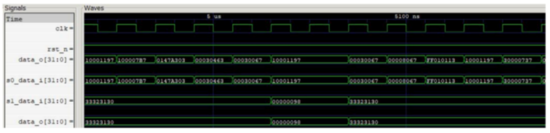


Fig. 20. Simulation waveforms of ROM and RAM.

V. CONCLUDING REMARKS

In the future, RISC-V is likely to develop into one of the world mainstream CPUs. This paper designs a three-stage pipelined RISC-V processor based on RV32IM. Compared with the traditional single-design pipelined CPU, it has higher practicability and is convenient for developers carry out simulation test and download debugging. The result shows that the designed processor functions correctly and achieves the predetermined goal.

The next step is to add a cache to increase the memory speed, and embed development in the FPGA hardware board.

ACKNOWLEDGMENT

This research was supported by the Innovative Research Projects for Graduate Students in Ordinary Colleges and Universities in Hainan Province (Hys2020-218).

REFERENCES

[1] Wang Shaokun based on FPGA five-stage pipeline CPU. Computer System Applications, 2015.
[2] Ni Guangnan.Meet the new trend of open source chips[J].Information Security and Communication Confidentiality,2019(02):11-13.
[3] Andrew Waterman, David Patterson. RISC-V Manual- a guide to the open source instruction set. https://university.imgtec.com/resources/books/, November 2018.
[4] Hu Zhenbo. Teach you how to design a CPU-RISC-V processor[M]. Beijing: People's Posts and Telecommunications Press, 2018.25-26.
[5] Lei Silei.Summary of RISC-V architecture open source processor and SoC research[J].Single Chip Microcomputer and Embedded System Applications,2017,17(02):56-60.
[6] Zhang Yonghui, Shen Zhong, Chen Baodan, etc. Principle and Application of ARM Cortex-M3 Microcontroller[M]. Beijing: Publishing House of Electronics Industry, 2013, 114-117.
[7] Translated by Yi Jiangfang, Liu Xianhua, etc. Computer Composition and Design: Hardware/Software Interface (Fifth Edition of the Original Book RISC-V Edition) [M]. Beijing: Mechanical Industry Press, 2020.191-192.
[8] Lei Silei. Handwriting CPU by yourself[M]. Beijing: Publishing House of Electronics Industry, 2014.192-198.
[9] Li Z , Hu W , Chen S. Design and Implementation of CNN Custom Processor Based on RISC-V Architecture[C]// 2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS). IEEE, 2019.
[10] Y. Lee, A. Ou, and A. Magyar, "Z-scale: Tiny 32-bit risc-v systems," in Open-RISC Conf., Geneva, Switzerland, 2015.
[11] Aslesa Singh, Neil Franklin*, Nidhi Gaur,Paursuh Bhulania, Processor Core using Virtex-7 and VirtexUltraScale, 2020 IEEE 5th International Conference on Computing Communication and Automation (ICCCA) Galgotias University, Greater Noida, UP, India. Oct 30-31, 2020.
[12] D. K. Dennis et al., "Single cycle RISC-V micro architecture processor and its FPGA prototype," 2017 7th International Symposium on Embedded Computing and System Design (ISED), Durgapur, 2017, pp. 1-5.
[13] Huang Zichen, Li Dehua. Embedded cross-development environment based on OpenOCD and JTAG[J]. User of Instrumentation, 2012, 19(01): 73-75.
[14] Liu Zhongyu. Design of Embedded CPU Bus Interface Unit. Microprocessor. 2014.04.004.