Cheng YH, Huang LB, Cui YJ *et al.* RV16: An ultra-low-cost embedded RISC-V processor core. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 37(6): 1307–1319 Nov. 2022. DOI 10.1007/s11390-022-0910-x

RV16: An Ultra-Low-Cost Embedded RISC-V Processor Core

Yuan-Hu Cheng (成元虎), Li-Bo Huang* (黄立波), *Senior Member*, *CCF*, Yi-Jun Cui (崔益俊)
Sheng Ma (马　胜), *Senior Member*, *CCF*, Yong-Wen Wang (王永文), *Senior Member*, *CCF*, and
Bing-Cai Sui (隋兵才), *Senior Member*, *CCF*

College of Computer Science and Technology, National University of Defense Technology, Changsha 410073, China

E-mail: {chengyuanhu, libohuang, cuiyijun18, masheng, wyw, bingcaisui}@nudt.edu.cn

Received August 15, 2020; accepted May 7, 2022.

Abstract Embedded and Internet of Things (IoT) devices have extremely strict requirements on the area and power consumption of the processor because of the limitation on its working environment. To reduce the overhead of the embedded processor as much as possible, this paper designs and implements a configurable 32-bit in-order RISC-V processor core based on the 16-bit data path and units, named RV16. The evaluation results show that, compared with the traditional 32-bit RISC-V processor with similar features, RV16 consumes fewer hardware resources and less power consumption. The maximum performance of RV16 running Dhrystone and CoreMark benchmarks is 0.92 DMIPS/MHz and 1.51 CoreMark/MHz, respectively, reaching 75% and 71% of traditional 32-bit processors, respectively. Moreover, a properly configured RV16 running program also consumes less energy than a traditional 32-bit processor.

Keywords embedded processor, RISC-V, architecture, Internet of Things (IoT)

1 Introduction

The Internet of Things (IoT), as an emerging information industry, is widely used in various fields, including smart homes, smart cities, industry, and transportation. With the development of the 5th generation mobile networks (5G), the IoT industry will generate a huge market that includes a large number of low-power devices [1]. The processor is an indispensable part of modern information equipment, and the processor design technology must be suitable for the development of IoT.

Limited by factors such as size and cost, the edge nodes of IoT need to work for several years with limited energy supply, which puts extremely stringent area and energy consumption requirements on the processor design [2]. Therefore, the current development trend of the IoT processor is smaller, cheaper, and lower power-consuming [3]. On the other hand, although 8-bit or 16-bit processors have obvious advantages in terms of area and power consumption, IoT devices are more inclined to use 32-bit processors because some IoT applications still have a higher demand for performance [4].

In order to reduce the area and power consumption of the embedded processor, this paper proposes RV16 from the aspect of optimizing the processor microarchitecture without significant performance degradation. RV16 is an ultra-low-cost in-order scalar 32-bit RISC-V processor core with two pipeline stages and supports 64 KiB instruction memory and 64 KiB data memory. Compared with traditional open-source in-order 32-bit and 64-bit RISC-V processors (such as Zero-riscy [5] and Rocket [6]), the significant difference of RV16 is that it is based on the 16-bit data path and processes 32-bit operand by reusing 16-bit function units.

Processors must be configurable because of vastly different demands of runtime resources for different IoT applications [1]. Therefore, RV16 supports three different configurations: RV16-32EC, RV16-32IC, and RV16-32IMC, which support RISC-V instruction set architecture (ISA) standard extensions RV32EC, RV32IC, and RV32IMC, respectively. In addition,

Regular Paper

This work was supported by the National Key Research and Development Project of China under Grant No. 2021YFB0300300, the National Natural Science Foundation of China under Grant Nos. 62090023, 61872374, 61672526 and 62172430, and the Natural Science Foundation of Hunan Province of China under Grant No. 2021JJ10052.

*Corresponding Author

RV16 provides a fast multiplier and a slow multiplier to support RISC-V "M" extension in different scenarios.

By optimizing the implementation of specific RISC-V instructions on the 16-bit data path, the maximum performance of RV16 running Dhrystone and CoreMark benchmarks is 0.92 DMIPS/MHz and 1.51 CoreMark/MHz respectively, reaching 75% and 71% of traditional 32-bit RISC-V processors, respectively. And the average performance of running the Embench benchmark suite can reach 76% of Cortex-M0. Moreover, the experimental results show that the hardware resources and the power consumption of RV16 are lower than those of traditional open-source 32-bit RISC-V processor cores.

In a nutshell, the contributions of this paper are as follows.

• A configurable 32-bit RISC-V processor core is designed and implemented based on the 16-bit data path.

• Some optimization methods for specific RISC-V instructions are proposed to improve the performance of the processor.

• The area, performance, power and energy consumption of the processor are evaluated.

The following sections are organized as follows. Some work related to this paper is introduced in Section 2. Section 3 introduces the details of the microarchitecture of RV16. In Section 4, the area, performance, power and energy consumption of RV16 are evaluated. Finally, Section 5 concludes our work.

2 Related Work

ARM Cortex-M series processors are currently the most widely-used embedded low-cost processors, in which Cortex-M0 is the most typical processor core. Cortex-M0 is a single-issue in-order core supporting ARMv6-M ISA with two pipeline stages. It is designed to be implemented in a small silicon area and low power environment. Since most of the instructions in the ARMv6-M are 16 bits, it also has a relatively high code density [7].

RISC-V ISA has developed rapidly with its advantages of open source. The biggest difference between RISC-V and traditional ISA is extensibility. To implement a RISC-V processor, only a simple basic instruction set must be implemented, and most standard and custom extensions are optionally implemented according to the requirements of the scenario [8]. With the extensibility, RISC-V can be applied to all application scenarios from high energy efficiency to high performance.

Many developers have developed many RISC-V processors for different embedded scenarios. Targeting higher-performance fields, BOOM [9–11] is a superscalar out-of-order processor core supporting RV64GC ISA with a TAGE (tagged geometric history length) branch predictor. And RSD [12] is an out-of-order RISC-V soft processor optimized for FPGA (Field Programmable Gate Array), which supports RV32IM ISA. For scenarios with limited area and power, Rocket [6] and Ariane [13] balance the performance and area. They use a 5-stage or 6-stage in-order scalar pipeline to support RV32/64GC ISA with a simple predictor and cache. However, most existing RISC-V processors are aimed at IoT devices with extremely demanding on the area and power consumption, such as mRISC [14], Hummingbird E203 [15], and Zero-riscy [5], and they all contain a simple 2-stage or 3-stage pipeline without branch predictor and cache. Moreover, Micro-riscy [5] is based on the RV32E instruction set containing fewer general-purpose registers, further reducing the overhead of the processor.

3 RV16 Microarchitecture

RV16 can be configured to support RISC-V "E", "I", "M", and "C" standard extension. The difference between RV32E and RV32I is that RV32E contains only 16 general-purpose registers, while RV32I contains 32. "M" extension requires to additionally support hardware multiplication and division based on RV32E or RV32I. And the length of instruction in the "C" standard extension is 16-bit, which can reduce the storage space required by the program.

The microarchitecture of RV16 is shown in Fig. 1, in which two pipeline stages are named IFA (Instruction Fetch and Align) and IDE (Instruction Decode and Execution). The rest of this section will introduce the pipeline structure of RV16 in detail.

3.1 Instruction Fetch and Align

The IFA stage generates an instruction address and sends it to instruction memory every cycle. Considering the length of instruction in the RV32I base instruction set, RV16 still has a 32-bit instruction path to fetch 32-bit instruction data in each cycle. It makes possible for RV16 to complete a 32-bit instruction that processes 32-bit data in one cycle.

However, the support for the "C" extension may make the length of the instruction processed by RV16

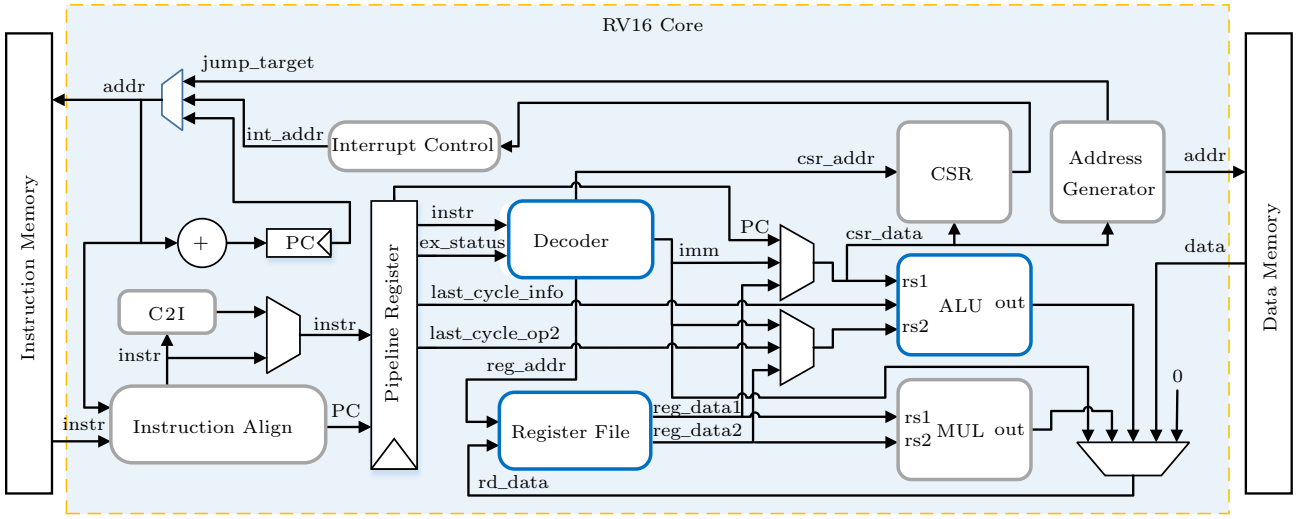


Fig.1. Microarchitecture of RV16.

be 16-bit or 32-bit. An instruction from the instruction memory needs to be aligned firstly by an instruction aligner to identify the compressed instruction. Before sending the instruction to the decoder, the compressed instruction will be translated into an uncompressed instruction in the compressed instruction decode unit (named C2I in Fig.1).

In order to minimize the hardware resource consumption, in RV16, only a 16-bit register is used to store 16-bit instruction data that may not be used in one cycle. An instruction is generated by combining the data in the register and the data from the instruction memory according to whether the data in the register is valid or not.

3.2 Instruction Decode and Execution

A RISC-V instruction always processes 32-bit data by default. As a result, the decode and execution units in the IDE stage of RV16 are different from traditional RISC-V processors because of the 16-bit data path that can only process a 16-bit operand per cycle. RV16 solves this problem by increasing the execution cycle of an instruction to reuse the 16-bit data path and units. Understandably, for most instructions in RISC-V, RV16 will take two execution cycles to complete the response operation: one cycle processes the lower 16-bit in 32-bit data, and the other cycle processes the upper 16-bit.

3.2.1 Decoder

In two execution cycles of one instruction, the RV16 decoder is required to generate different control signals at different execution cycles to select the correct data and operation. As shown in Fig.2, when the decoder decodes an instruction, it depends not only on the input instruction but also on the execution status of the instruction (it is represented by a one-bit signal, named ex_status in Fig.2, which is reset to 0 in the first execution cycle of one instruction). In practice, the execution status is used to indicate whether the upper 16 bits or lower 16 bits of the 32-bit data should be processed in the current execution cycle. Therefore, the signals related to the operand will be affected by the execution status (such as reg_addr and imm in Fig.2).

3.2.2 Register File

There are two ways to implement the register file based on the 16-bit internal data path. The first way keeps each register 32 bits and selects the lower or upper 16 bits of a register through a control signal. The second way is to divide each 32-bit register into two 16-bit registers and index them by address. Our experiments prove that the latter consumes fewer hardware resources because its control logic is simpler. As a result, 16 32-bit registers in RV32E and 32 32-bit registers in RV32I are implemented by a register file containing 32 and 64 16-bit registers, respectively.

The read and write logic of the register file in RV16 is shown in Fig.3. Different from traditional RISC-V processors, the register address of RV16 consists of the register number in the instruction and an additional least significant bit. The least significant bit of the register address is determined by the type of instructions and its execution status.

In most cases, when an instruction enters the IDE stage from the IFA stage, the lower 16-bit data is pro-

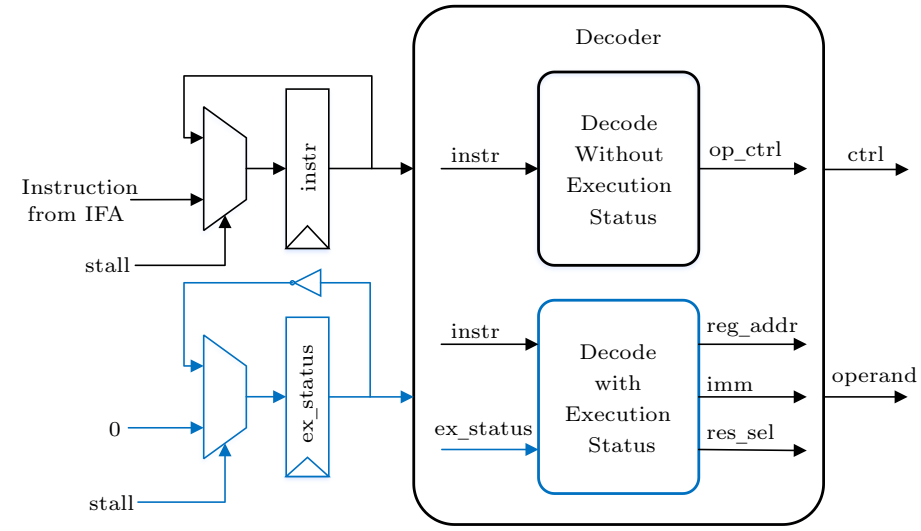


Fig.2. Decoder logic in RV16.

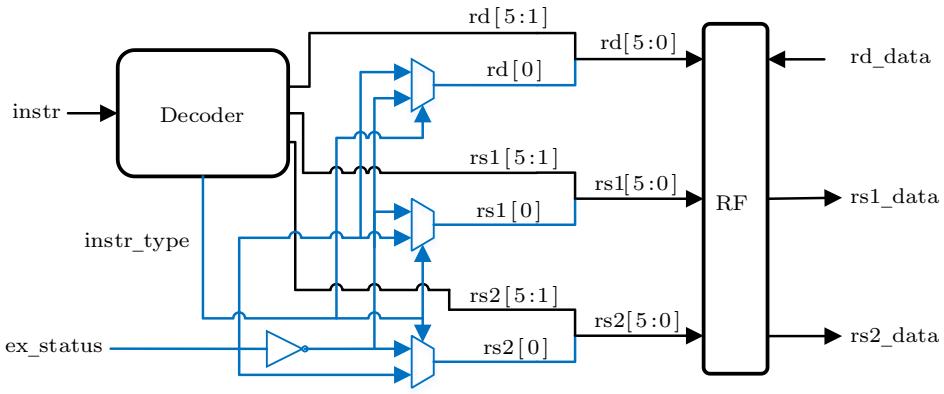


Fig.3. Read and write logic of the register file in RV16 with 64 registers.

cessed in the first execution cycle, and the upper 16-bit is processed in the next cycle. For example, to process an ADD instruction, RV16 reads and writes the lower 16-bit register in the first execution cycle, which sets the least significant bit of register file address to 0. In the second execution cycle, it will operand the upper register, and the least significant bit will be set to 1. Obviously, in these cases, the least significant bit is equal to ex_status in Fig.3.

However, there are some exceptions. Firstly, for shift right instructions, there is no way to handle the lower 16-bit data without the upper 16-bit, because the bits shifted out from the upper 16-bit data will appear in the lower 16-bit result. RV16 cannot read the upper 16-bit data while reading the lower data without adding the register file read port in one cycle. As a consequence, for these instructions, RV16 first processes the upper 16-bit data and keeps it in a special register for reuse in the next cycle. At this point, the least significant bit is equal to NOT ex_status in Fig.3. Secondly, for load and store instructions, the base address register is the lower 16-bit register and cannot change during the memory access because of the 16-bit address space; thus the least significant bit is always equal to 0.

3.2.3 *Arithmetic Logic Unit*

The arithmetic logic unit (ALU) in RV16 includes three parts: a shifter, an adder, and a logic unit, as shown in Fig.4(a). The compare unit after the adder generates signals to indicate the magnitude relationship between the two operands according to the result of the adder, which is used to achieve the compare and branch instructions. The logic unit of RV16 is a traditional 16-bit logic unit because bits of the logical result do not affect each other, while the shifter and the adder are different from the traditional RISC-V processor.

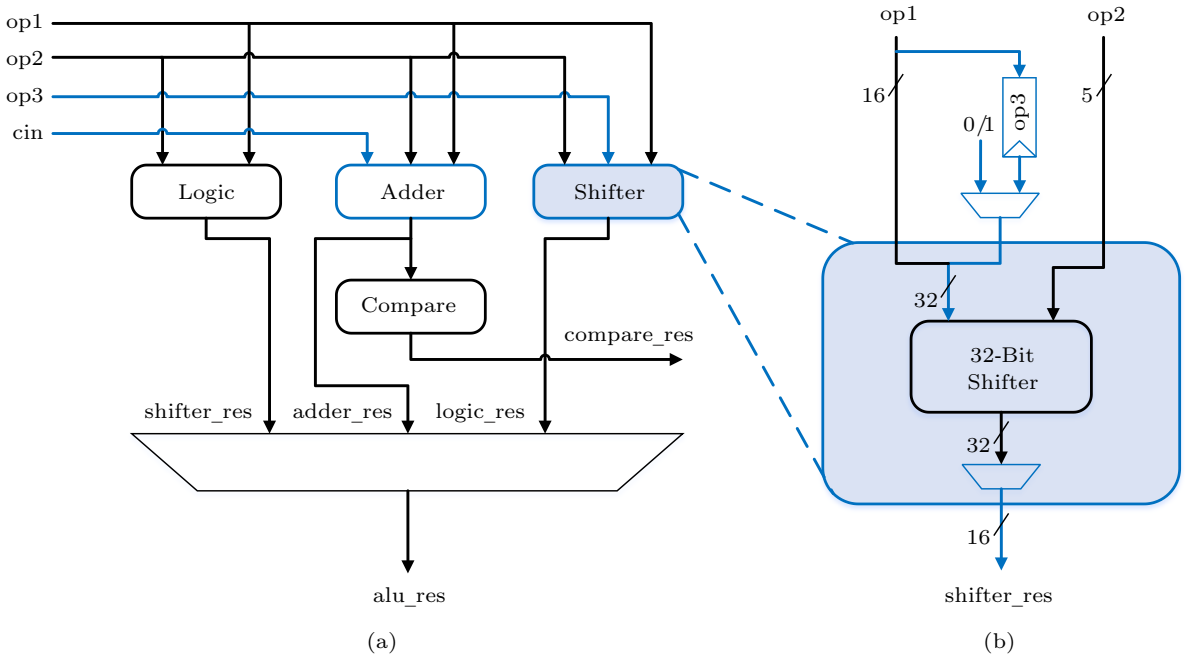


Fig.4. RV16 ALU and shifter structure. (a) ALU structure. (b) Shifter structure.

As mentioned earlier, the correct 32-bit result cannot be obtained if only 16-bit data is used per cycle for a shift operation. RV16 processes the upper 16-bit data first for a shift right operation and the lower first for a shift left operation. The advantage is that the 16-bit data processed first does not depend on other 16-bit data for a 32-bit operand.

The detailed logic of the RV16 shifter is shown in Fig.4(b), which is mainly composed of a 32-bit shifter. The operand of the 32-bit shifter is spliced by two 16-bit operands (op1 and op3 in Fig.4(b)). And op3 is set to 1 or 0 by bit according to the shift mode in the first execution cycle and is equal to the previous op1 in the second cycle. For example, an SRA (shift right arithmetic) instruction sets all bits of op3 to the sign bit of the shifted data and lets op1 equal the upper 16 bits of the shifted data in the first execution cycle. In the second execution cycle, op3 is equal to the first cycle op1, and op1 is equal to the lower 16 bits of the shifted data. The SRA instruction always generates a 32-bit shifter operand according to op3 in the upper 16 bits and op1 in the lower. Lastly, it selects the lower 16 bits of the 32-bit shift result to write to the destination register as the result of this cycle.

For the adder, it is obvious that the lower carry bit needs to be considered in the upper 16-bit operation. Therefore, unlike the traditional RISC-V processor, the adder in RV16 needs to process the carry signal. Understandably, the carry signal is set to 0 in the first execution cycle and is generated by the first cycle in the second execution cycle.

3.2.4 *Multiplier and Divider for RV32M*

Multiplier and divider exist only in RV16 that supports RISC-V “M” standard extension. The implementation of the divider is similar to that of the traditional RISC-V processor but the operands require two cycles to pass in. RV16 provides two implementations of multiplier for configurability: fast multiplier (F) and slow multiplier (S).

In the fast multiplier, the 32-bit multiplication operation is achieved by a 16-bit single-cycle multiplier. When the lower 32 bits of the multiplication result are needed (MUL instruction), three 16-bit multiplication operations are required. When the upper 32 bits are needed (MULH instruction), four operations are required. And the slow multiplier is implemented by accumulating operands in multiple cycles.

3.3 Optimization

Generally, it will take two cycles to process a 32-bit operation (except for multiplication and division) by the 16-bit data path, which has a significant impact on performance. As a result, RV16 has proposed some optimization methods for certain RV32I instructions to improve performance.

First of all, the branch instruction judges the magnitude relationship between the two operands and determines if the branch is taken or not based on the result. In RV16, whether a branch instruction handles the lower 16-bit data is determined by the result of the upper operation. Only when the upper 16 bits of two operands are equal, the lower bits will be processed. If they are not equal, RV16 can directly get the result of the relationship between them.

Additionally, LBU (load byte unsigned) and LHU (load half-word unsigned) instructions can hide one-cycle delay required to access memory by writing the upper 16-bit destination register first. As shown in Fig. 5, the first execution cycle of a load instruction generates and sends the address to data memory, and writes 0 to the upper destination register. In the second execution cycle, the first 16-bit load data is already valid and can be written into the lower destination register. At this point, LBU and LHU have been completed. If an LB (load byte) or LH (load half-word) instruction is being executed, whether the third execution cycle is needed depends on the sign bit of the load data. And for an LW (load word) instruction, RV16 will load the higher data from memory in the second execution cycle, and write the load data to the upper destination register in the third cycle.

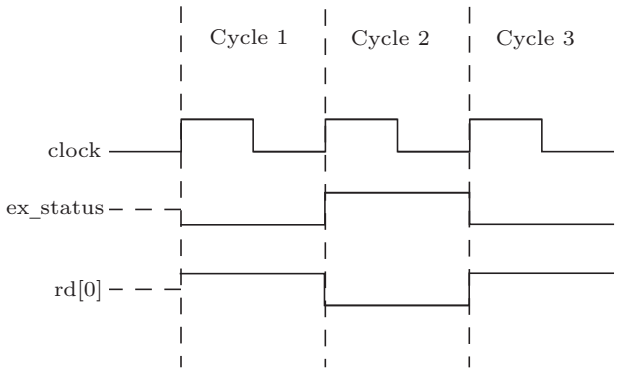


Fig. 5. Partial waveform of key signals when executing a load instruction.

Similarly, an SLT (set less than) instruction only needs to write 0 or 1 to the destination register, and thus its upper 16-bit destination register is always 0. Therefore, in the first execution cycle, RV16 writes 0 to the upper destination register because the result of the comparison between the two operands has not come out. Then, RV16 only requires to write 0 or 1 to the lower destination register according to the comparison result in the second execution cycle.

3.4 Theoretical Execution Cycles

Table 1 counts the number of theoretical execution cycles required by various RISC-V instructions in RV16. Because RV16 only supports the 16-bit address space, the Jump instruction calculates the target address and saves the return address in the first cycle, and jumps to the correct address to fetch the instruction in another cycle. The Branch instruction requires one or two cycles when the branch is not taken, and just like Jump, an extra cycle is required to jump to the target when the branch is taken. All ALU instructions require two cycles to process higher and lower 16 bits of the 32-bit operand separately. LBU/LHU and LW instructions can be completed in two cycles and three cycles respectively; LB and LH require two or three cycles depending on the sign bit. The SB (store byte) and SH (store half-word) instructions can be completed in one cycle, while SW (store word) requires two cycles. MUL and MULH require three cycles and five cycles in the fast multiplier, respectively, while they both require 35 cycles in the slow multiplier. Lastly, the division operation requires three cycles when the divisor is equal to zero, while 38 cycles are required in the other cases.

Table 1. Number of Theoretical Execution Cycles of Various Instructions in RV16

Instruction	Number of Execution Cycles
Jump	2
Branch	1, 2 or 3
ALU	2
Load	2 or 3
Store	1 or 2
Multiple (fast)	3 or 5
Multiple (slow)	35
Divide	3 or 38

4 Evaluation

In this section, the area, performance, power and energy consumption of RV16 are evaluated. We synthesize RV16 RTL (Register Translation Level) code on FPGA and custom CMOS (Complementary Metal Oxide Semiconductor) respectively to get its area cost information. The performance of RV16 is evaluated by running Dhrystone, CoreMark, and Embench benchmark suite [16]. Finally, the power and energy consumption of RV16 is evaluated by running Dhrystone and CoreMark on custom CMOS.

Zero-riscy/Micro-riscy [5] and E203 [15] are traditional open-source 32-bit RISC-V processor cores that

contain similar features to RV16, and thus their information is shown for comparison. In addition, ARM Cortex-M0 processor is also included in the comparison, because it is a widely-used commercial processor core in the low-cost field. It is worth noting that the Cortex-M0 processor evaluated in the experiment comes from the Cortex-M0 DesignStart (DS) provided by ARM for free, which contains a single-cycle fast multiplier.

4.1 Area

The FPGA resources consumed by each processor core are shown in the form of the number of LUTs (Look-Up Tables), FFs (Flip-Flops), and DSPs (Digital Signal Processors) in Table 2. The second column lists the specific ISA supported by the processor core. RV16 based on the RV32I instruction set consumes more LUTs and FFs than that based on RV32E because it contains more general-purpose registers. The single-cycle multiplier is implemented by DSP on FPGA, and thus RV16-32I/EMC(F) consumes fewer LUTs than RV16-32I/EMC(S).

When supporting the same RISC-V ISA, RV16 consumes fewer FPGA resources than other RISC-V processor cores. For example, to support RV32IMC, RV16 with a fast multiplier consumes 1 808 LUTs, 1 365 FFs, and 1 DSP, while Zero-riscy consumes 3 352 LUTs, 2 006 FFs, and 1 DSP. On the other hand, FPGA resources consumed by RV16 are also lower than Cortex-M0: RV16-32EMC(F) contains the features closest to Cortex-M0 and consumes 52% LUT and 92% FF of Cortex-M0.

Table 2. FPGA Resource Consumption of Processors

Processor	ISA	Number of LUTs	Number of FFs	Number of DSPs
RV16-32IMC(F)	IMC	1 808	1 365	1
RV16-32IMC(S)	IMC	1 867	1 365	0
RV16-32EMC(F)	EMC	1 580	851	1
RV16-32EMC(S)	EMC	1 617	851	0
RV16-32IC	IC	1 461	1 222	0
RV16-32EC	EC	1 088	742	0
Micro-riscy	EC	2 501	1 352	0
Zero-riscy	IMC	3 352	2 006	1
E203	IMC	3 772	1 814	0
Cortex-M0	v6-M	2 814	922	3

The synthesized result on custom CMOS at 100 MHz is shown in Fig.6. Similar to the FPGA results, RV16 occupies the smallest CMOS area in RISC-V processor cores that support the same ISA. For RV32IMC, the area of RV16-32IMC(F) is 20 757.91 μm^2, while that of E203 is 29 024.18 μm^2 and that of Zero-riscy is 32 802.67 μm^2. In addition, the area of RV16-32EMC(F) is 64% of that of Cortex-M0.

In Fig.6, the area advantage of RV16-32EC (compared with Micro-riscy) is more significant than that of

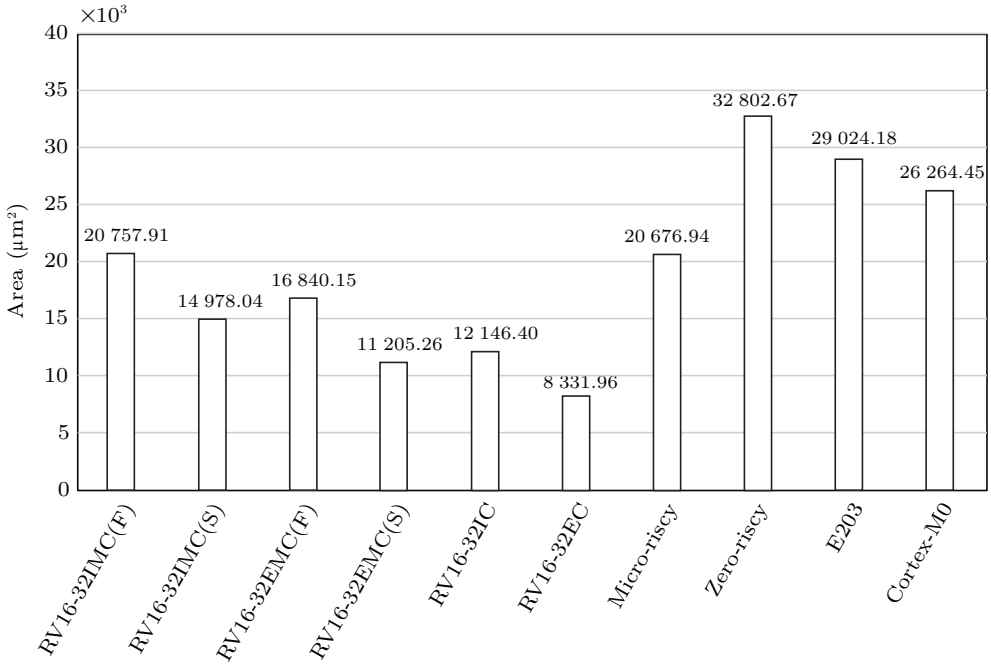


Fig.6. Custom CMOS area of processors at 100 MHz.

RV16-32IMC(F) (compared with Zero-riscy and E203). In addition to the factor of the register file, another reason is that the area overhead of the multiplier is significant in the low-cost processor core: a 16-bit multiplier for supporting RV32M increases the area of RV16-32EC by 8 508.19 μm^2.

4.2 Performance

The performance of RV16 is evaluated in two ways. The first takes two SRAMs (Static Random Access Memories) as instruction and data memory respectively to run benchmarks to get the performance. Another way runs benchmarks in Cortex-M0 DS SoC prototype simulation environment.

4.2.1 *Comparison With RISC-V Processors*

The performance of RV16 and other low-cost RISC-V processor cores running Dhrystone and CoreMark is shown in Table 3. The maximum performance of RV16-32IMC(F) running Dhrystone binary code compiled based on RV32IM can reach 0.92 DMIPS/MHz, which is comparable to the traditional 32-bit processor. Moreover, the smallest RV16-32EC can also provide competitive performance, reaching 0.81 DMIPS/MHz.

Table 3. Performance of Running Dhrystone and CoreMark

Processor	Maximum Performance	
	Dhrystone (DMIPS/MHz)	CoreMark (CoreMark/MHz)
RV16-32IMC(F)	0.92	1.51
RV16-32IMC(S)	0.87	1.04
RV16-32EMC(F)	0.84	1.49
RV16-32EMC(S)	0.81	1.03
RV16-32IC	0.89	0.62
RV16-32EC	0.81	0.58
Micro-riscy [5]	1.19	0.91
Zero-riscy	1.29	2.20
E203 [15]	1.23	2.14

The 16-bit data path has a more significant impact on CoreMark. The maximum performance of RV16-32IMC(F) is 1.51 CoreMark/MHz, about 70% of the traditional 32-bit RISC-V processor. However, unlike the running Dhrystone, the performance of RV16 without supporting hardware multiplication and division (RV16-32I/EC) running CoreMark is much lower than that with supporting hardware multiplication and division (RV16-32I/EMC). To analyze the reasons for this situation, we count the instruction distribution in the main loop of two benchmarks compiled for RV32IM, and the results are shown in Fig.7.

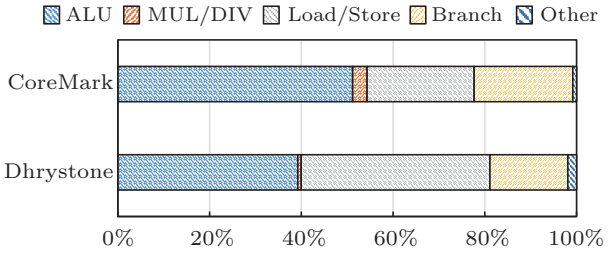


Fig.7. Instruction distribution of CoreMark and Dhrystone compiled for RV32IM.

The ratio of multiplication and division instructions (MUL/DIV in Fig.7) in CoreMark is higher than that in Dhrystone. As a consequence, software multiplication and division will significantly increase the number of instructions in CoreMark. As shown in Table 4, compiled with GCC, the number of RV32I instructions in CoreMark's main loop is 2.45 times the number of RV32IM instructions, while Dhrystone is just 1.11 times. A larger number of instructions lead to a lower performance for running CoreMark on the processor that does not support hardware multiplication and division.

Table 4. Number of Instructions Executed in the Main Loop of Dhrystone and CoreMark

ISA	Instruction Count	
	Dhrystone (500 Loops)	CoreMark (1 Loop)
RV32E	160 677	776 542
RV32I	147 245	732 505
RV32IM	132 745	298 825

We calculate the IPC (Instructions-Per-Cycle) of RV16 running Dhrystone and CoreMark, which is shown in Table 5. Overall, when hardware multiplication and division are not supported, the IPC of RV16 is about 0.46. It drops slightly after supporting a fast multiplier because a multiplication or division instruction requires more execution cycles. The absence of division instructions in CoreMark's main loop results in a smaller drop in IPC. Finally, Table 6 shows the actual average execution cycles of various RISC-V instructions in RV16-32IMC(F).

Table 5. IPC of RV16 Running Dhrystone and CoreMark

Processor	IPC	
	Dhrystone	CoreMark
RV16-32EC	0.46	0.45
RV16-32IC	0.46	0.46
RV16-32IMC(F)	0.43	0.45

Table 6. Number of Actual Execution Cycles of Various Instructions in RV16-32IMC(F)

Instruction	Number of Execution Cycles	
	Dhrystone	CoreMark
Jump	2.00	2.00
Branch	2.12	2.50
ALU	2.00	2.00
Load	2.87	2.52
Store	1.92	1.87
Multiple	3.00	3.00
Divide	38.00	-

4.2.2 Compared With Cortex-M0

Different from taking two SRAMs as instruction and data memory, the simulation environment of Cortex-M0 DS is based on the AHB-Lite (Advanced High-performance Bus) where instruction and data share the same memory, which means that instruction and data cannot be accessed at the same cycle [17]. Since most of the instructions supported by Cortex-M0 are 16-bit, we employ the RV16 that supports "C" extensions to compare with it.

In the Cortex-M0 DS simulation environment, the performance of RV16 and Cortex-M0 running Dhrystone and CoreMark is shown in Fig.8, which has been normalized by the performance of Cortex-M0. The performance of RV16 running Dhrystone is only slightly lower than that of Cortex-M0. The performance of RV16-32IMC(F) reaches 91% of Cortex-M0 and RV16-32EMC(F) reaches 82%. Combined with the area overhead, this is a competitive result.

In Fig.8(b), the performance of RV16-I/EC running CoreMark is far lower than that of Cortex-M0. After implementing hardware division and fast multiplier, the performance of RV16 has been greatly improved, reaching 86% of Cortex-M0. A slow multiplier also rises the performance of RV16-32E/IMC(S) to 61% of Cortex-M0. On the other hand, it can be speculated that the single-cycle multiplier in Cortex-M0 is one of the reasons why it runs CoreMark better.

As shown in Fig. 9, the average performance of RV16-32IMC(F) running the Embench benchmark suite is 84% of that of Cortex-M0, and that of RV16-32EMC(F) is 76%. The performance of RV16-32IMC(F) running the minver and aha-mont64 benchmark is faster than that of Cortex-M0. However, the multiplication operation is still a major factor that affects the performance of RV16. Because of including many multiplication operations, matmult-int is the worst benchmark for RV16, and the performance of RV16-32E/IC is much lower than that of RV16-32E/IMC.

The dotted line in Fig.9 shows the impact of the multiplier implementation on the performance of RV16 running different benchmark programs, and the larger the number, the smaller the impact. When the ratio is equal to 1, it means that the multiplier implementation has no impact on the performance of the benchmark program. The multiplier has less impact on the performance of the RV16 based on RV32E than on RV32I (the ratio of RV16-32EMC(S)/RV16-32EMC(F) in the figure is always greater than or equal to that of RV16-32IMC(S)/RV16-32IMC(F)). The reason is that RV32E contains fewer registers than RV32I and thus the number of instructions generated by the compiler based on RV32E is often greater than that on RV32I, which reduces the proportion of multiplication instructions in all instructions.

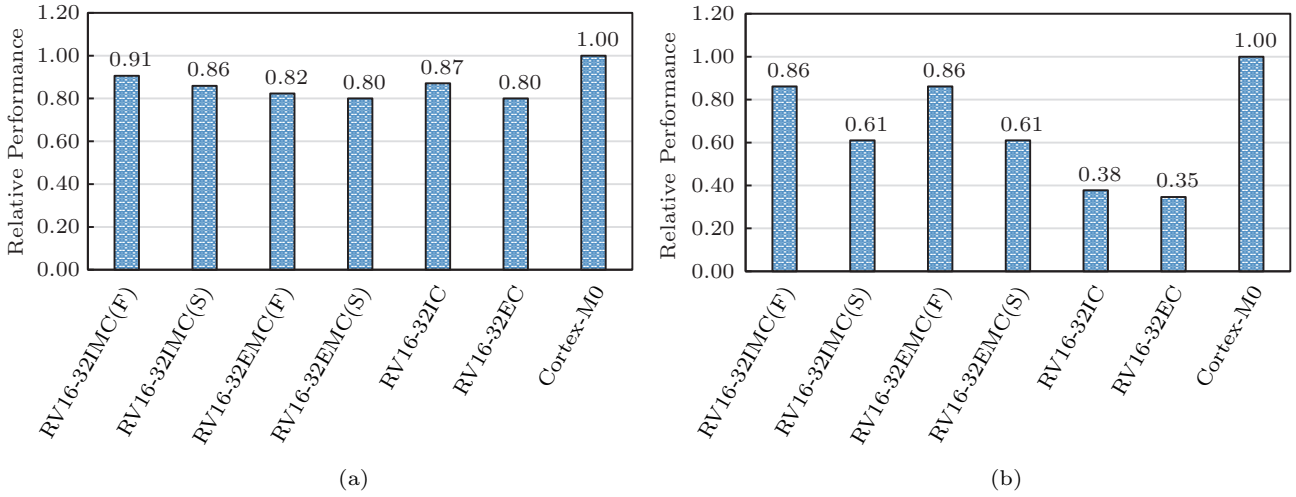


Fig.8. Relative performance of RV16 and Cortex-M0 running (a) Dhrystone and (b) CoreMark.

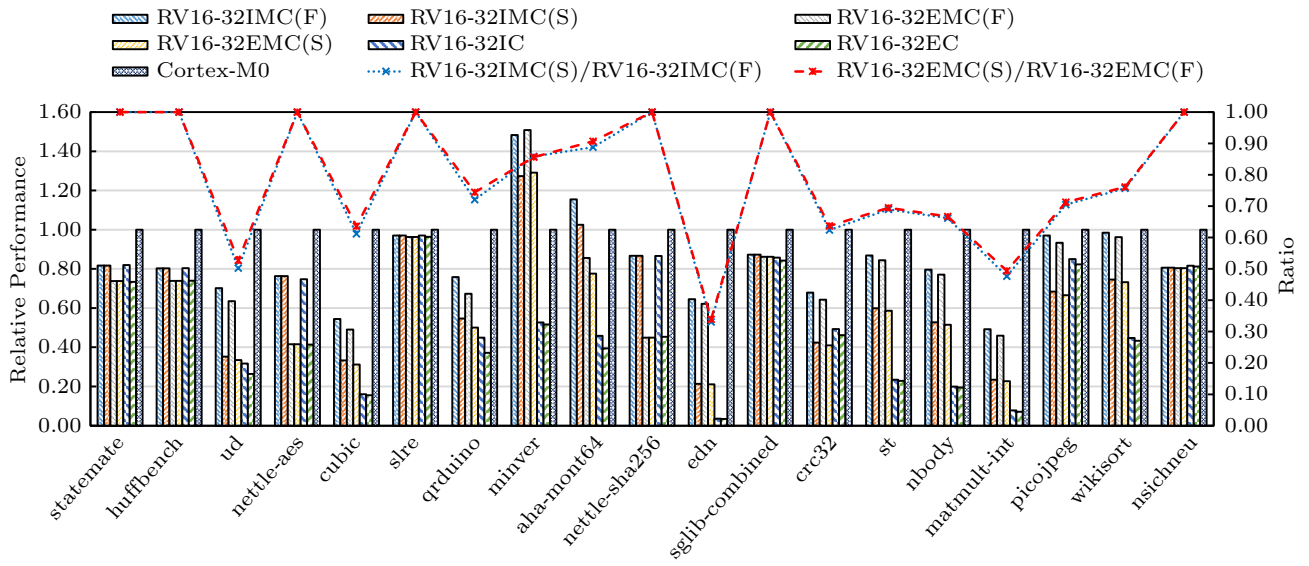


Fig.9. Relative performance of RV16 and Cortex-M0 running the Embench benchmark suite.

4.3 Power and Energy Consumption

Based on the Switching Activity Interchange Format (SAIF) file extracted by the simulation and the netlist file generated after synthesis, we evaluate the power and energy consumption of RV16 running benchmark programs on custom CMOS. The SAIF file used for evaluation only contains the switching activity information in the main loop of the benchmark program. The partial information of the SAIF file generated by RV16-32IMC(F) and Cortex-M0 running Dhrystone is shown in Table 7, and Table 8 is the explanation of the parameters.

Fig.10 shows the static and dynamic power of RV16 running Dhrystone and CoreMark at 0.81 V and 100 MHz. With fewer logic gates, both the static and the dynamic power of RV16 are lower than those of Zero-riscy and Cortex-M0. In a similar configuration, the static power of RV16-32EMC(F) is about 52% of Cortex-M0, and that of RV16-32IMC(F) is about 60% of Zero-riscy. We can also get similar conclusions in terms of dynamic power. The dynamic power running CoreMark is slightly more than that of running Dhrystone, which means that the switching activity running CoreMark is higher than that of running Dhrystone.

Using the power and performance information obtained earlier, we can calculate the energy consumption of processors. The final energy consumption information of running Dhrystone (500 loops) and CoreMark (one loop) is shown in Fig.11, in which all values are normalized based on Cortex-M0.

In Fig.11(a), since the power of RV16 is lower and the performance gap is small, the energy consumption of running Dhrystone on RV16 is lower than that of Zero-riscy and Cortex-M0. The energy consumption of RV16-32IMC(F) and RV16-32IMC(S) is 73% and 59% of Cortex-M0, respectively, lower than that of Zero-riscy which supports the same ISA. The implementation of multiplication instruction has little effect on the performance of running Dhrystone, and thus RV16-32EMC(S) has the lowest energy consumption.

Table 7. Partial Information of the SAIF File

Signal	RV16-32IMC(F)					Cortex-M0				
	T0 (ps)	T1 (ps)	TX (ps)	TC	IG	T0 (ps)	T1 (ps)	TX (ps)	TC	IG
HADDR[0]	3 697 990 000	20 020 000	0	4 004	0	3 314 160 000	20 030 000	0	4 006	0
HADDR[1]	3 162 430 000	555 580 000	0	111 116	0	2 672 370 000	661 820 000	0	128 362	0
HADDR[2]	1 951 420 000	1 766 590 000	0	164 130	0	1 664 460 000	1 669 730 000	0	183 497	0
HADDR[3]	1 961 410 000	1 756 600 000	0	106 084	0	1 679 630 000	1 654 560 000	0	134 330	0
HADDR[4]	1 866 810 000	1 851 200 000	0	85 087	0	1 654 740 000	1 679 450 000	0	101 250	0
HADDR[5]	2 081 620 000	1 636 390 000	0	71 066	0	1 569 500 000	1 764 690 000	0	88 200	0
HADDR[6]	2 011 160 000	1 706 850 000	0	68 070	0	1 714 830 000	1 619 360 000	0	62 185	0
HADDR[7]	1 776 430 000	1 941 580 000	0	61 062	0	2 054 630 000	1 279 560 000	0	80 156	0

Table 8. Explanation of Parameters in the SAIF File

Name	Explanation
T0	Duration of the time found in logic 0 state
T1	Duration of the time found in logic 1 state
TX	Duration of the time found in unknown X state
TC	Sum of the rise and fall transitions that are captured during monitoring
IG	Number of 0–X and 1–X glitches captured during monitoring

The energy consumption result of running CoreMark is different from that of running Dhrystone. Two RV16 cores that do not support hardware multiplication and division consume the most energy because their poor performance leads to longer running time. Among the compared processors, RV16-32EMC(S) is still the processor with the lowest energy consumption, about 67% of Cortex-M0. The energy consumption of RV16-32IMC(F) running CoreMark is 79% of Cortex-M0, which is also lower than that of Zero-riscy.

5 Conclusions

This paper designs a configurable ultra-low-cost embedded 32-bit RISC-V processor core, which is implemented by the 16-bit internal data path. Since more than one cycle is required to process 32-bit data by multiplexing the 16-bit data path and units, the paper studies the optimization methods for some instructions to reduce the number of cycles required. Experimental results showed that RV16 has significant advantages in area, power and energy consumption while its perfor-

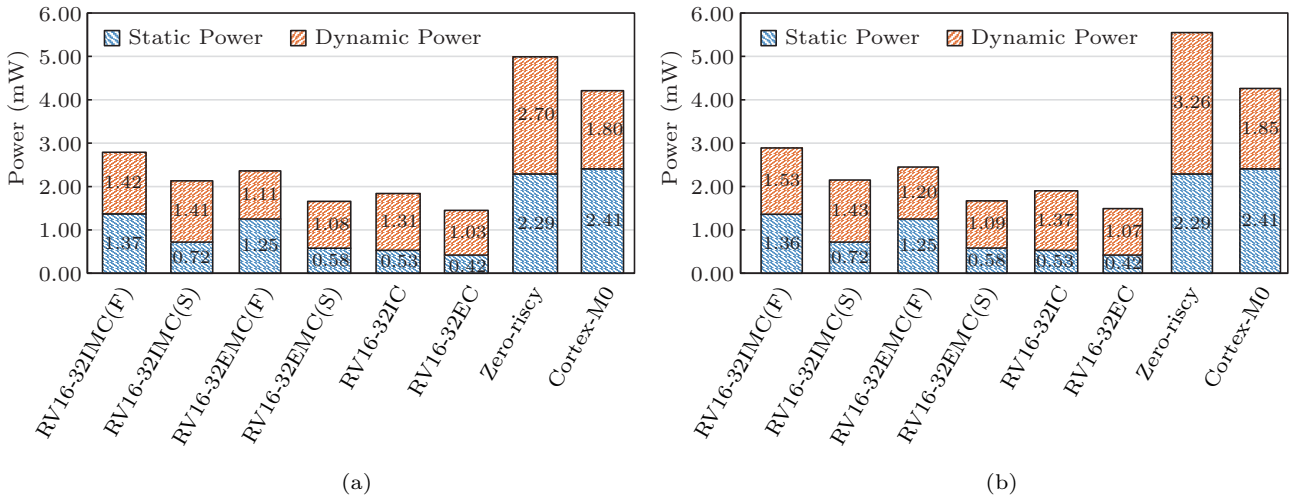


Fig.10. Power of running (a) Dhrystone and (b) CoreMark at 0.81 V and 100 MHz.

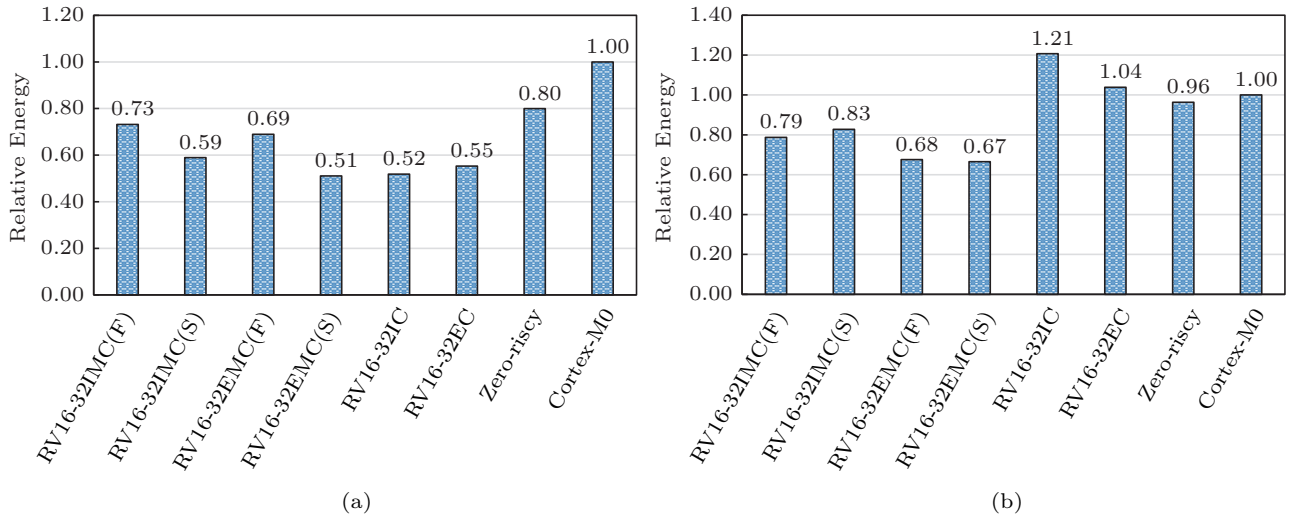


Fig.11. Energy consumption of running (a) Dhrystone and (b) CoreMark at 0.81 V and 100 MHz.

mance is only slightly lower than that of traditional 32-bit processors. Running representative benchmarks in the embedded field showed that, among various configurations of RV16, RV16E/I(C) can provide competitive performance for applications with fewer multiplication instructions. However, RV16E/IM(C) is more suitable for applications with more multiplication instructions.

References

[1] Adegbija T, Rogacs A, Patel C *et al.* Microprocessor optimizations for the Internet of Things: A survey. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018, 37(1): 7-20. DOI: 10.1109/TCAD.2017.2717782.

[2] Alioto M. Enabling the Internet of Things: From Integrated Circuits to Integrated Systems (1st edition). Springer, 2017.

[3] Banday M T. A study of current trends in the design of processors for the Internet of Things. In *Proc. the 2nd International Conference on Future Networks and Distributed Systems*, Jun. 2018, Article No. 21. DOI: 10.1145/3231053.3231074.

[4] Sultan I, Banday M T. A study of the design architectures of configurable processors for the Internet of Things. In *Proc. the 3rd International Conference on Contemporary Computing and Informatics*, Oct. 2018, pp.320-325. DOI: 10.1109/IC3I44769.2018.9007256.

[5] Schiavone P D, Conti F, Rossi D *et al.* Slow and steady wins the race? A comparison of ultra-low-power RISC-V cores for Internet-of-Things applications. In *Proc. the 27th International Symposium on Power and Timing Modeling, Optimization and Simulation*, Sept. 2017. DOI: 10.1109/PATMOS.2017.8106976.

[6] Asanović K, Avizienis R, Bachrach J *et al.* The rocket chip generator. Technical Report, Electrical Engineering and Computer Sciences Department, University of California at Berkeley, 2016. https://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.pdf, Aug. 2021.

[7] ARM. ARM Cortex-M0 technical reference manual. Technical Report, ARM, 2009. https://developer.arm.com/documentation/ddi0432/c/preface, Aug. 2021.

[8] Waterman A, Asanović K. The RISC-V instruction set manual–Volume I: Unprivileged ISA. Technical Report, Electrical Engineering and Computer Sciences Department, University of California at Berkeley, 2019. https://riscv.org/wp-content/uploads/2019/12/riscv-spec-20191213.pdf, Aug. 2021.

[9] Celio C, Patterson D A, Asanović K. The Berkeley out-of-order machine (BOOM): An industry-competitive, synthesizable, parameterized RISC-V processor. Technical Report, Electrical Engineering and Computer Sciences Department, University of California at Berkeley, 2015. https://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-167.pdf, Aug. 2021.

[10] Celio C, Chiu P F, Nikolic B *et al.* BOOM v2: An open-source out-of-order RISC-V core. Technical Report, Electrical Engineering and Computer Sciences Department, University of California at Berkeley, 2017. https://www2.eecs.berkeley.edu/Pubs/TechRpts/2017/EECS-2017-157.pdf, Aug. 2021.

[11] Zhao J, Korpan B, Gonzalez A *et al.* *Sonic*BOOM: The 3rd generation Berkeley out-of-order machine. In *Proc. the 4th Workshop on Computer Architecture Research with RISC-V*, May 2020.

[12] Mashimo S, Fujita A, Matsuo R *et al.* An open source FPGA-optimized out-of-order RISC-V soft processor. In *Proc. the 2019 International Conference on Field-Programmable Technology*, Dec. 2019, pp.63-71. DOI: 10.1109/ICFPT47387.2019.00016.

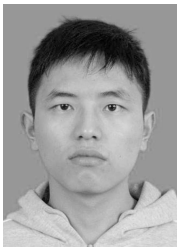
[13] Zaruba F, Benini L. The cost of application-class processing: Energy and performance analysis of a Linux-ready 1.7-GHz 64-bit RISC-V core in 22-nm FDSOI technology. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2019, 27(11): 2629-2640. DOI: 10.1109/TVLSI.2019.2926114.

[14] Duran C, Rueda D L, Castillo G *et al.* A 32-bit RISC-V AXI4-lite bus-based microcontroller with 10-bit SAR ADC. In *Proc. the 7th IEEE Latin American Symposium on Circuits & Systems*, Feb. 28-Mar. 2, 2016, pp.315-318. DOI: 10.1109/LASCAS.2016.7451073.

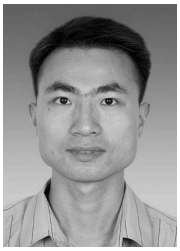
[15] Deng T, Hu Z. An ultra-low-power processor pipeline-structure. *Application of Electronic Technique*, 2019, 45(6): 50-53. DOI: 10.16157/j.issn.0258-7998.182563. (in Chinese)

[16] Bennett J, Dabbelt P, Garlati C *et al.* Embench™: An evolving benchmark suite for embedded IoT computers from an academic-industrial cooperative. https://riscv.org/wp-content/uploads/2019/06/9.25-Embench-RISC-V-Workshop-Patterson-v3.pdf, Apr. 2022.

[17] ARM. ARM Cortex-M0 DesignStart Eval user guide. Technical Report, ARM, 2017. https://developer.arm.com/documentation/dui0926/latest?_ga=2.192512938.1641451169.-1629894075-1043868800.1596886884, Aug. 2021.



Yuan-Hu Cheng received his B.S. degree in computer science and technology from Sichuan University, Chengdu, in 2018. He is a Master student at College of Computer Science and Technology, National University of Defense Technology, Changsha. His research interests include microprocessor architecture.

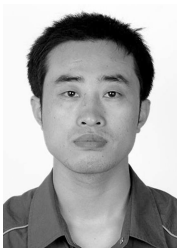


Li-Bo Huang received his B.S. and Ph.D. degrees in computer engineering from National University of Defense Technology, Changsha, in 2005 and 2010, respectively. He is an associated professor at College of Computer Science and Technology, National University of Defense Technology, Changsha. His research interests include computer architecture, hardware/software co-design, VLSI design, and on-chip communication.

Yi-Jun Cui received his B.S. degree in computer science and technology from Huazhong Agricultural University, Wuhan, in 2018. He is a Master student at College of Computer Science and Technology, National University of Defense Technology, Changsha. His research interests include microprocessor architecture.

Sheng Ma received his B.S. and Ph.D. degrees in computer science and technology from National University of Defense Technology, Changsha, in 2007 and 2012, respectively. He is an associated professor at College of Computer Science and Technology, National University of Defense Technology, Changsha. His research interests include on-chip networks, SIMD architecture and arithmetic unit design.

Yong-Wen Wang received his Ph.D. degree in computer science from National University of Defense Technology, Changsha, in 2004. He is a professor at College of Computer Science and Technology, National University of Defense Technology, Changsha. His research interests include computer architecture and high performance computing.



Bing-Cai Sui received his Ph.D. degree in electronic science and technology from National University of Defense Technology, Changsha, in 2010. He is an associated professor at College of Computer Science and Technology, National University of Defense Technology, Changsha. His research interests include microprocessor architecture, design and optimization.