ISSN 2959-6157

Design and Implementation of Digital Integrated Circuit Based CPU

Ruikai Zhang

University of Electronic Science and Technology of China, Glasgow College, Hainan, China
Corresponding author: 2839689Z@student.gla.ac.uk

Abstract:

This paper presents a simplified 16-bit CPU design and its implementation using digital integrated circuits developed with Verilog hardware description language, while emphasizing simulation through the EDA tool. This design puts more emphasis on the critical roles played by the data path and the control unit, executing key instructions including arithmetic operations such as addition, subtraction, bitwise logic, and conditional jumps. This structurally modular CPU is both functioning and stable in nature, allowing support for future enhancements and enhancements as well, thereby serving as an effective model towards an understanding of the principles of CPU design. The execution of these basic instructions puts the core of the CPU under test and reaffirms its proper operation in respect of the detailed instruction set, while its extensibility design confirms further improvement possibilities for future complex instructions as well as finite-state machine optimization for the control unit. The present work not only proves the possibility of creating a fully working CPU, using readily available bare minimum resources to implement a prototype, but is also a breakthrough in future research with CPU architecture. The obtained results will shed light on CPU functionality optimization and extension, being the ground for building more powerful and effective processors in the future.

Keywords: 16-bit CPU Design, Digital Integrated Circuits, Verilog Hardware Description Language, CPU Simulation and Verification

1. Introduction

With the rapid development of information technology, the design and optimization of the central processing unit (CPU), as the core component of a computer system, occupies an important position in the field of electronic information engineering.The performance of the CPU directly determines the overall processing capability of the computer system. Therefore, the study and design of efficient CPU architectures are of great significance to enhance the performance of computing devices. In particular, the demand for high-performance CPUs is becoming more and more urgent, driven by emerging technologies such as big data, artificial intelligence and the Internet of Things. Understanding and mastering the working principle and design methodology of CPUs, and designing them based on digital integrated circuit platforms has become an important task for electronic and information engineering students and related researchers [1].

Traditional CPU design usually involves complex architectures and advanced processes, which may be difficult for beginners and engineering education. By designing and implementing a simplified CPU model, however, students can intuitively understand the internal working mechanism of the CPU and master the whole process from instruction decoding to execution to result storage. Especially through the application of digital integrated circuit technology, it enables designers to deeply understand the various modules of the CPU at the hardware level, such as data paths and control units [2]. This enables students and researchers not only to consolidate the basic knowledge of digital circuits, but also to lay a solid foundation for more complex CPU design and research in the future.

This paper details the design and implementation of a 16-bit simplified CPU based on digital integrated circuits. By using Verilog Hardware Description Language (VHDL) and EDA tools (Vivado) [3], the various modules of the CPU have been designed, simulated and verified to ensure that it is able to correctly execute a series of basic arithmetic and logical instructions [4]. The main objective of this experiment is to verify the functionality and scalability of the CPU by designing and implementing its various functional modules such as data path and control unit.

The paper is organized as follows: Chapter 1 provides an introduction, outlining the research background, significance, and the main focus of the paper. Chapter 2 delves into the CPU design methodology, detailing the working principles of each module, with a particular emphasis on the design of the data path and control unit, along with the results obtained from the implementation. Chapter 3 pres-

ents and analyzes the results of the CPU design. Finally, Chapter 4 concludes the paper, summarizing the successes and challenges encountered during the experiments and suggesting future research directions.

2. Methodology

2.1 Overall CPU Architecture Overview

In this experiment, we designed and implemented a simplified 16-bit CPU model. The overall architecture of this CPU can be divided into two main parts: the data path and the control unit [2,5], as shown in Fig. 1. The data path is responsible for performing all arithmetic and logical operations, and reading instructions and performing write operations. The control unit is responsible for parsing the instructions and generating the corresponding control signals, and coordinating the operation between the modules [6]. The two work closely together through signaling lines to enable the CPU to execute the intended instruction set correctly.

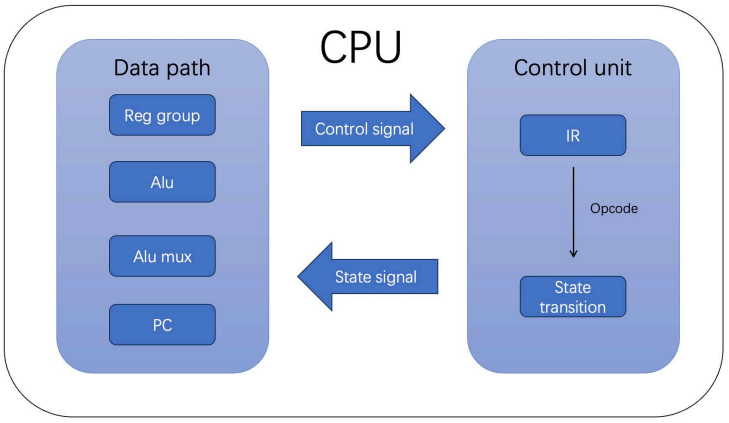

Fig.1 CPU architecture diagram

2.2 Datapath design

The datapath is the core computing part of the CPU [7], which handles the transfer of data, arithmetic operations, logical operations, and execution of instructions. The key modules in the datapath include the Register Group, the Arithmetic Logic Unit (ALU), the Program Counter (PC), and the Multiplexer (MUX), as shown in Figure 2.

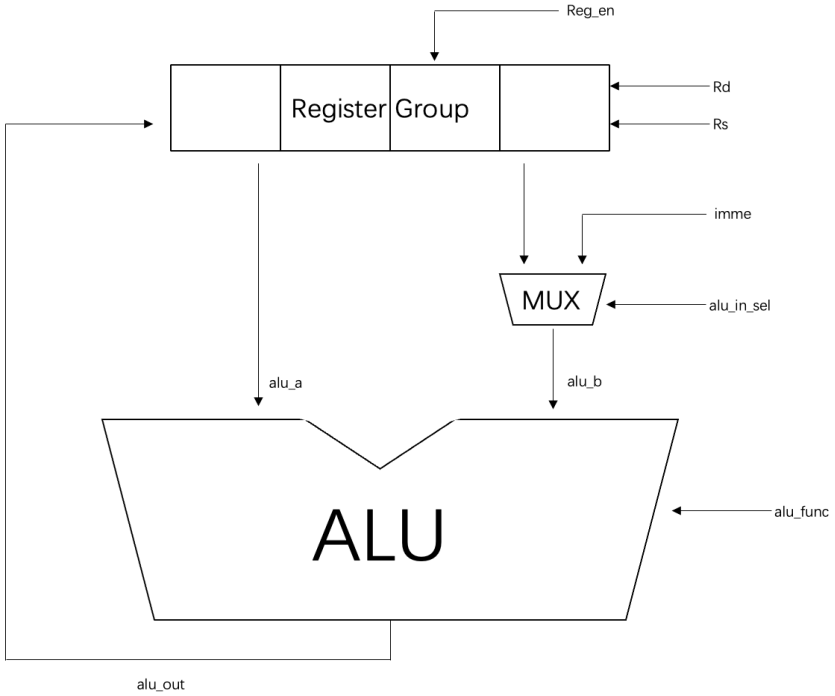

Fig. 2 Datapath schematic diagram

- Register Group:
The register group is the core module for storing and reading operands in the CPU. We designed four registers (R0, R1, R2, R3) for temporary storage of operands and intermediate operation results. The registers control the reading and writing of data through address selection signals (Rd, Rs) and enable signals (reg_en). In short, the register group selects the source operands through Rs in the read stage, and writes the operation results of the ALU to the specified registers in the write stage according to the reg_en signal from the control unit.
- ALU:
The ALU is the arithmetic core of the datapath and is responsible for performing all arithmetic and logical operations in the CPU instructions. The ALU is designed to support a variety of operation modes including addition, subtraction, bit-wise and, bit-wise or, etc [8]. Depending on the Opcode, the ALU reads operands from a register bank or uses an immediate number (imme) from an instruction as an input. The function selection of the ALU is determined by a control signal (alu_func), and different opcodes correspond to different modes of operation. For example, when the opcode instructs to perform addition, the ALU adds the two input operands and passes the result to a register bank or program counter.
- Program Counter:
The program counter is used to keep track of the address of the instruction currently being executed by the CPU. After each instruction execution, PC is incremented to point to the address of the next instruction. However, when a jump instruction is executed, the value of PC is updated according to the jump address. The update operation of PC is controlled by the pc_ctrl signal of the control unit, which supports the functions of maintaining the current address, incrementing the address, and jumping to a new address.
- Multiplexer:
The multiplexer in the datapath is used to select the input data source for the ALU. The MUX selects the immediate number from the instruction register as input when an immediate number operation needs to be performed, and the MUX selects the output of a register bank as input when a register operation is to be performed. The selection operation of the MUX, which is determined by the alu_in_sel signal of the control unit, ensures that the datapath can correctly handle different types of operation instructions.

2.3 Control unit design

The control unit is the command center of the CPU, which is responsible for parsing instructions and generating control signals to guide the modules in the data path to perform operations in sequence. The design of the control unit is based on a finite state machine (FSM), which includes functional modules such as instruction decoding, state transfer and control signal generation.
- Instruction Register:
The instruction register is used to store the instruction currently being executed. At the beginning of each instruction cycle, the CPU reads the instruction from the memory and loads it into the IR. the output of the IR is passed to the control unit for decoding and generating the appropriate control signals.
- Instruction Decoder:
The control unit first decodes the instruction in the current instruction register (IR). The instruction consists of an opcode (Opcode), a destination register (Rd), a source register (Rs) and an immediate number (imme). The instruction decoder maps the opcode to specific control signals that are used to activate the corresponding operation in the data path. For example, when the opcode instructs to perform an addition operation, the decoder generates the corresponding alu_func signal that instructs the ALU to perform the addition operation.
- Finite State Machine:
The state machine of the control unit is responsible for managing the operation flow of the CPU. The FSM [8] consists of multiple states, initial, finger fetch, decode, execute, and write back, etc.[3,9], as shown in Fig. 3. Each state represents a different stage of the CPU in executing an instruction. The FSM decides the next state based on the current state and input signals (e.g., opcodes, conditional flags, etc.) and generates the corresponding control signals [10]. For example, when the CPU is in the finger fetch state, the FSM generates a signal to control the PC increment and prepares to load the next decoding instruction.

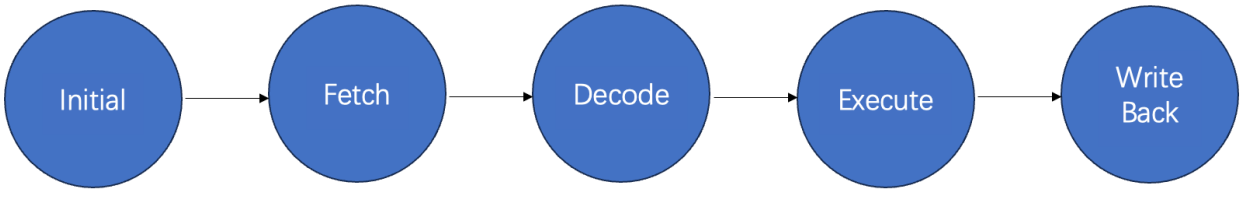

Fig. 3 State transfer diagram

Through the above design, the control unit is able to effectively parse and execute various instructions while working closely with the data path to ensure that the CPU maintains high efficiency and stability during execution. This design approach lays a solid foundation for further functional expansion and performance optimization.

3. Results

In this experiment, the designed 16-bit CPU successfully fulfills all the expected functions and passes the simulation verification with a variety of instructions [11]. The data path, as in Fig. 4, and the control unit, as in Fig. 5, of the various modules, including register sets, ALUs, program counters, and state machines, show good functionality and stability. The simulation results show that the CPU is able to correctly execute instructions such as addition, subtraction, bitwise-and, bitwise-or, and conditional jumps, and all the arithmetic results are consistent with expectations, as in Fig. 6.

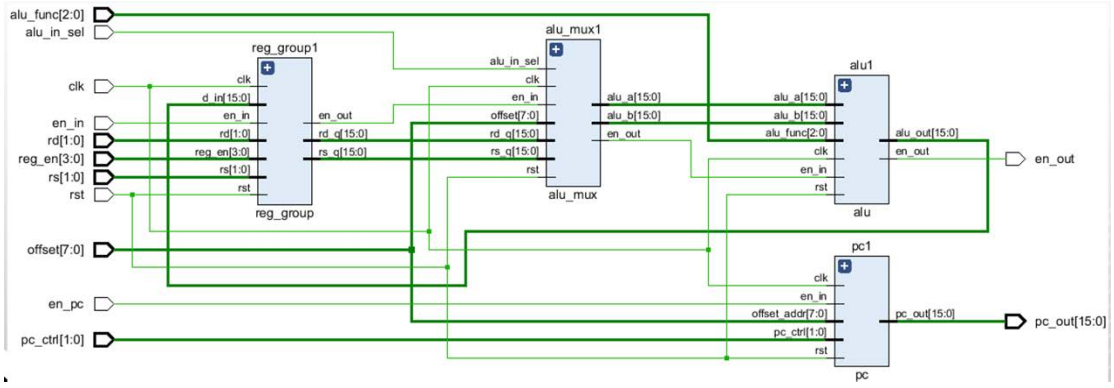

Fig. 4 RTL Data path

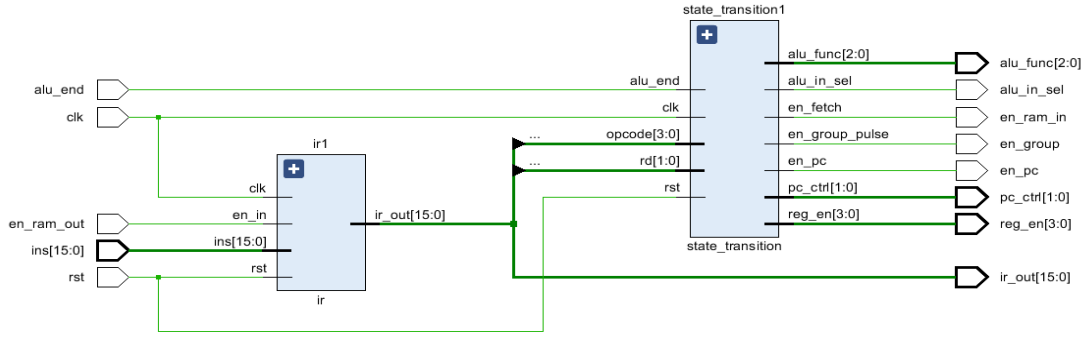

Fig. 5 RTL Control unit

In the comprehensive analysis, the hardware resource utilization of the CPU is more reasonable, and the timing simulation results show that the clock cycle of the system meets the design requirements, and the delay of the critical path is within the acceptable range. In addition, the scalability of the instruction set is preliminarily verified, and by adding new instructions, the CPU still maintains good operational performance and functional consistency. The simulation also shows that when processing complex instructions, the data path and control unit are well coordinated and can effectively complete the parsing and execution of instructions.

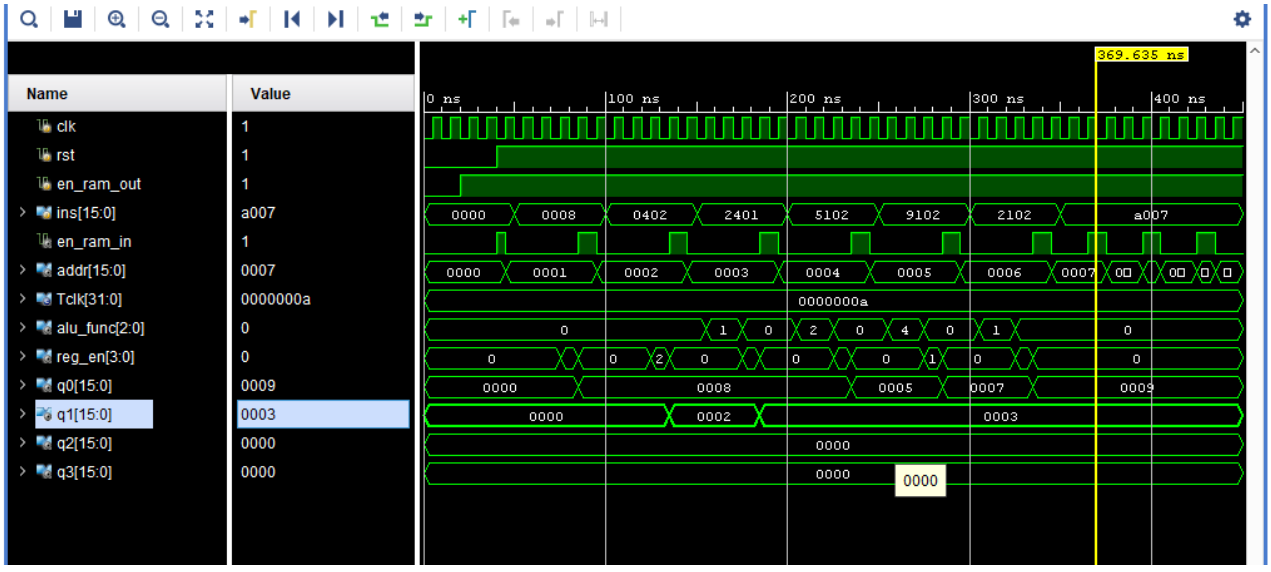

Fig. 6 CPU timing simulation results

Overall, the CPU designed in this experiment achieves the expected design goals, verifies the feasibility and stability of the architecture, and provides strong support for subsequent optimization and expansion.

4. Conclusion

Through this experiment, we have successfully designed and implemented a 16-bit simplified CPU based on digital integrated circuits, and carried out a comprehensive simulation to verify it. The experimental results show that the CPU can correctly execute basic instructions including addition, subtraction, bitwise and, bitwise or, conditional jump, etc., which fully verifies the effectiveness and stability of its design. The design of the data path and control unit is the core part of the whole CPU architecture, where the data path is responsible for arithmetic and data transmission, and the control unit ensures the sequential execution of instructions by generating precise control signals. Through programming in Verilog Hardware Description Language [3,8] and simulation with EDA tools, we verified the rationality of the design and ensured the coordination and compatibility among the modules.

The successes in this design are mainly reflected in the following aspects: firstly, the data path is reasonably designed, and the key modules such as ALU, register group and multiplexer can effectively work together to ensure the smooth execution of instructions. Second, the control unit adopts the finite state machine (FSM) design, which can accurately decode the instructions and generate the corresponding control signals, thus realizing the correct processing of complex instructions. In addition, in terms of the utilization of hardware resources, we ensure that the CPU achieves the expected functions while occupying fewer resources by optimizing the design.

However, some challenges and limitations were revealed during the experiments. Although the designed CPU is capable of handling basic arithmetic and logic instructions, the existing ALU and control unit may need to be further optimized when dealing with more complex operations (e.g., multiplication, division, etc.). In addition, simulation results show that the state machine design of the control unit will face higher complexity as the instruction set expands and complexity increases, which may lead to a more difficult design and verification process.

Future research can further extend and optimize the functionality and performance of the CPU based on the existing design. First, the instruction set can be enriched by adding support for complex instructions (e.g., multiplication, division, displacement operations, etc.), and the state machine design of the control unit can be optimized to adapt to more complex instruction processing requirements. In addition, the introduction of pipelined design can significantly improve the processing efficiency of instructions and enable CPUs to perform better in multitasking environments, although this will also increase the complexity of design and verification.

In addition, in order to enhance the utilization efficiency of hardware resources, optimization of circuit layout and timing control can be considered to reduce power consumption and increase computation speed. The exploration of multi-core architectures is also a direction worthy of in-depth study, which will help cope with the growing demand for parallel computing. With these improvements, CPU designs can not only meet current performance requirements, but also support more complex and efficient computing tasks in the future.

References

[1]. J. H. Lee, S. E. Lee, H. C. Yu and T. Suh, "Pipelined CPU Design With FPGA in Teaching Computer Architecture," in IEEE Transactions on Education, vol. 55, no. 3, pp. 341-348, Aug. 2012, doi: 10.1109/TE.2011.2175227.

[2]. A. Yıldız, H. F. Ugurdag, B. Aktemur, D. İskender and S. Gören, "CPU design simplified," 2018 3rd International Conference on Computer Science and Engineering (UBMK), Sarajevo, Bosnia and Herzegovina, 2018, pp. 630-632, doi: 10.1109/UBMK.2018.8566475.

[3]. Lai, Xianzhi. Simple CPU design based on FPGA[J]. Journal of Chongqing Vocational and Technical College, 2005, 14(1): 117-119.

[4]. Korobilis C. Design of a CPU using VHDL[D]. Aristotle University of Thessaloniki, 2020.

[5]. WANG Ben-You, SU Shou-Bao, WANG De-Ru. An FPGA-based CPU design[J]. Computer Technology and Development, 2008, 18(6): 221-224.

[6]. X. Liu, L. Fu, W. Rao, X. Lin, M. Liao and B. Shi, "Multi-Cycle CPU Design With FPGA for Teaching of Computer Organization Principle," 2019 14th International Conference on Computer Science & Education (ICCSE), Toronto, ON, Canada, 2019, pp. 472-477, doi: 10.1109/ICCSE.2019.8845494.

[7]. J. B. Lazaro, M. D. Pabiania, L. J. S. Bitangcor, J. C. B. De Torres, J. M. A. Dumapit and J. N. Mapote, "Design of a 32-bit Datapath for a Reduced Instruction Set Computers (RISC) Implementation using the DE0-nano FPGA," 2024 16th International Conference on Computer and Automation Engineering (ICCAE), Melbourne, Australia, 2024, pp. 88-92, doi: 10.1109/ICCAE59995.2024.10569847.

[8]. T. Cao, "Implementation and Research of Vivado HLS's Function on FPGA," 2022 3rd International Conference on Computer Science and Management Technology (ICCSMT), Shanghai, China, 2022, pp. 240-243, doi: 10.1109/ICCSMT58129.2022.00057.

[9]. M. Becker, "Faster Verilog simulations using a cycle based programming methodology," Proceedings. IEEE International Verilog HDL Conference, Santa Clara, CA, USA, 1996, pp. 24-31, doi: 10.1109/IVC.1996.496014.

[10]. S. Alsubaei, S. M. Qaisar and W. Alhalabi, "A VHDL based Moore and Mealy FSM example for education," 2017 IEEE 2nd International Conference on Signal and Image Processing (ICSIP), Singapore, 2017, pp. 456-459, doi: 10.1109/SIPROCESS.2017.8124583.

[11]. G. Prasad, "A Tutorial on Design of Datapath and Controller of an ALU using Verilog and Verification using Open Source EDA Tools," 2021 2nd International Conference on Communication, Computing and Industry 4.0 (C2I4), Bangalore, India, 2021, pp. 1-4, doi: 10.1109/C2I454156.2021.9689339.