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

Design and Implementation of CNN Custom Processor Based on RISC-V Architecture

Zhenhao Li[a,b], Wei Hu[a,b], and Shuang Chen[a,b]

[a]College of Computer Science and Technology, Wuhan University of Science and Technology, China
[b]Hubei Province Key Laboratory of Intelligent Information Processing and Real-time Industrial System, China

***Abstract*—With the rapid development of CNN(convolutional neural networks), the traditional CPU platform can not make full use of the parallelism of CNN. We decide to adopt a new and popular processor architecture: the risc-v architecture for experimental design. In this paper, a new convolutional neural network processor is designed based on risc-v architecture. The processor can take advantage of the parallelism of CNN and is more flexible. This paper completely designed a CNN processor based on the risc-v architecture. The processor uses a classic five-stage pipeline structure, and implements instruction buffer memory and data buffer memory, and adds peripherals such as FLASH, SRAM, and SDRAM. And, this paper designed custom instructions. Given the convolution operation frequently occurring in CNN, vector store instruction, vector load instruction, vector addition instruction, and convolution operation instruction are designed to accelerate the execution of the convolution process. The design has passed the simulation experiment. It can not only complete the general instructions but also run the custom instructions. The final simulation test verified the correctness of the design.**

***Index Terms*—RISC-V processor; CNN; custom instruction; SoC technology;**

I. Introduction

RISC-V is a new instruction set architecture, which was invented by Krste Asanovic, Andrew Waterman and Yunsup Lee of the University of California at Berkeley in 2010 [1]. ”RISC” means a reduced instruction set, and ”V” means the fifth generation instruction set designed by Berkeley from RISC I. RISC-V is an open, free ISA that opens up a new era of processor innovation because of open standards collaboration. Since 2010, Krste et al. of UCB began to study the RISC-V instruction set. By 2015, a new complete instruction set had been developed, including the corresponding compiler, simulator and tool chain [2]. As a new processor architecture, the risc-v architecture will have a profound impact on existing processor architectures.

As a widely used algorithm in deep learning, Convolutional Neural Networks(CNN) are implemented in a large number of applications, such as face recognition [3], lane detection [4], stereo vision matching [5], and speech recognition [6]. CNN is improved by Multilayer Perceptron (MLP) [7], which can learn features directly from the original data, thus avoiding complex feature extraction and data reconstruction. Therefore it is highly adaptable, and it can mine the local features of data so that it can achieve good results in various fields. But most of the calculations in CNN are convolution operations, and the convolutional layer contains a large number of multiply-and-accumulate operations, which greatly slows down the computational efficiency. FPGA can be used to speed up convolution operations because of its highly parallel computing features [8].

Corresponding author: huwei@wust.edu.cn

In the next section, we will introduce the architecture of risc-v and then learn about CNN's various acceleration solutions at home and abroad. Then, in the third part, we propose a new design, which is to design a processor based on the risc-v architecture to optimise and accelerate the instruction level convolution. In the scheme, we developed a new CNN processor architecture, customised CNN instructions, and streamlined the data path for higher efficiency.

II. Background

A. RISC-V ISA

RISC-V is a typical three-operand, load-store RISC architecture. The RISC-V instruction set supports the extension of variable length instructions. In the base ISA, there are four core instruction formats (R/I/S/U), and a further two variants of the instruction formats (SB/UJ) based on the handling of immediates, all are a xed 32 bits in length and must be aligned on a four-byte boundary in memory. The instruction format is shown in Figure 1.

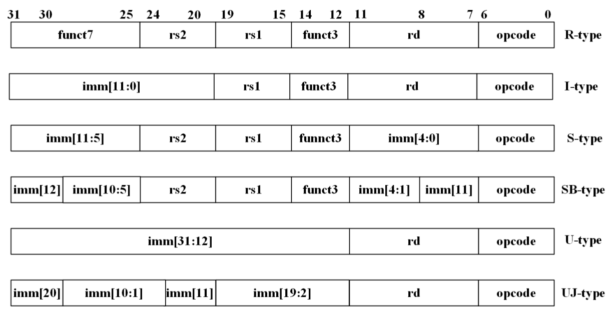

Fig. 1. RISC-V base instruction formats showing immediate variants.

The RISC-V ISA keeps the source (rs1 and rs2), and destination (rd) registers at the same position in all formats to simplify decoding. Immediates are packed towards the leftmost available bits in the instruction and have been allocated to reduce hardware complexity. In particular, the sign bit for all immediates is always in bit 31 of the instruction to speed sign-extension circuitry.

B. Hardware Implementation of CNN

Due to the nature of CNN instruction execution, general purpose processors are not suitable for mining the parallelism of CNN. To make full use of the parallelism of CNN, a solution accelerated by hardware is proposed. Designers expect optimised hardware design to accelerate CNN execution.

In 2009, Farabet et al. [9] proposed an FPGA-based CNN. The structure uses a convolution operation unit to process the data and control it with a general purpose processor. However, limited by the hardware resources of the FPGA, the platform only implements one convolution unit. If it calculates multiple convolution cores, it can only perform serial operations. In 2013, Peemen et al. [10] implemented a storage-centric CNN coprocessor, which utilised a large number of memory access characteristics of CNN itself, in which the storage used Scratchpad Memory, and the PE used SIMD structure. In 2015, Fang Rui et al. of Tsinghua University proposed a pipeline acceleration scheme based on the high pipelining characteristics of CNN layers. The CPU provides data and controls for the whole logic through the PCIE interface. In recent years, Tianshi Chen and others from the Institute of Computing Technology of the Chinese Academy of Sciences have proposed the DianNao series of CNN accelerators [11], which is the best chip in the field of neural network hardware implementation at present and can support algorithms such as MLP/CNN/DNN.

Pure hardware implementation of CNN accelerator has high specificity. Although configurable design can be carried out, its flexibility is far from the general processor. The design of this paper is based on the risc-v architecture and customises the specialised instructions for the CNN algorithm. According to the customised instructions, a microprocessor architecture is designed, so that the processor can run not only the general algorithm but also has a significant acceleration effect on the CNN algorithm.

III. RISC-V CNN PROCESSOR

The RISC-V CNN processor implements the RISC-V base instruction set and custom instructions. The length of this processor instruction is 32-bit, the same as the custom instruction.The risc-v CNN processor implements a classic 5-stage folded pipeline consisting of instruction fetch (IF), instruction decode (ID), execute (EX), memory access (MEM), and register write-back (WB).

In this chapter, we will first introduce the encoding format and function of custom instructions, then present the overall architecture of the processor, and finally, show how to optimise the data path.

A. Custom Instructions

According to the characteristics of CNN algorithm, a set of special instructions is defined. The length of the customised instructions is set to 32 bits for compatibility with 32-bit RISC-V processors.

According to the risc-v basic opcode map in the risc-v instruction set manual, four opcodes can be customised. The opcodes are 0001011, 0101011, 1011011, 1111011. We select 0001011 as the opcode for our custom instruction set.

The operating data of CNN is generally a matrix, it needs to transfer multiple data with external memory at one time, to avoid the delay caused by multiple access to memory. To meet this requirement, vector loading instruction (VLOAD) and vector storage instruction (VSTORE) are designed to transmit multiple data needed for the calculation. Convolution operation instruction(CONV) to achieve matrix multiplication. Vector addition instruction(VADD) is designed to perform vector addition. The encoding format of the custom instructions is shown in Figure 2.

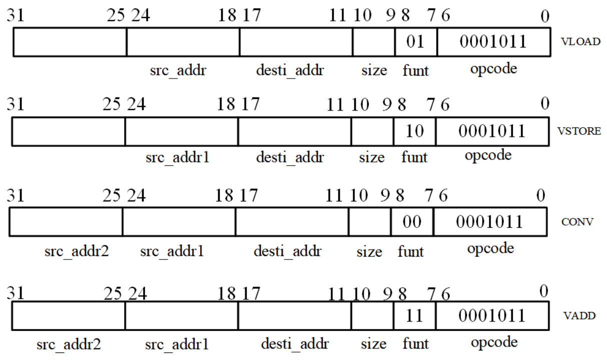

Fig. 2. Custom instructon encoding format.

Next, we will introduce the general encoding rules of custom instructions, and then explain the encoding format of each instruction in detail. We divide the instruction code of the custom instruction into six fixed areas, from the lower bits, they are defined as: Instruction Opcode(opcode), Function Code(funct), Size(size), Destination Address(desti_addr), Source Address 1(src_addr1), Source Address 2(src_addr2).

- Instruction Opcode(opcode): The opcode is located in the last 7 bits of the instruction, and it is the identification code of the instruction.
- Function code (funct): The funct is located in the 8th to 7th bits of the instruction, which is used to determine the function of the instruction. 00 is the convolution operation instruction(CONV), 01 is the vector load instruction(VLOAD), and 10 is the vector store instruction(VSTORE), 11 is the vector addition instruction(VADD).
- Size(size): The size is located in the 10th to 9th bits of the instruction to identify the size of the data to be operated. 00 means the window size is 2,01 means3, 10 means 4, 11 means 5.
- Destination Address (desti_addr): The desti_addr is located between 17 and 11 bits of the instruction, indicating

the starting address of the location where the final result will be stored.

- Source Address 1 (src_addr1): The src_addr1 is located between 24 and 18 bits of the instruction, indicating the starting address of the storage location of the first operand.
- Source Address 2 (src_addr2): The src_addr2 is located between 31 and 25 bits of the instruction, indicating the starting address of the storage location of the first operand.

Below, we will explain the function of custom instructions.Next, we will explain the functionality of custom instructions. There are four customisation instructions: VLOAD, VSTORE, VADD and CONV.

1) VLOAD: The VLOAD instruction loads the size bytes of data from the off-ship memory space to the on-chip with src_addr1 as the starting address. The starting address stored on the chip is desti_addr. Expressed as : VLOAD #desti_addr, #src_addr1, size.

2) VSTORE: The VSTORE instruction stores the size bytes of data in the on-chip space into off-chip memory with src_addr1 as the starting address. The starting address of the off-chip memory is desti_addr. Expressed as : VSTORE #desti_addr, #src_addr1, size.

3) CONV: The CONV instruction implements matrix multiplication. Size is used to represent the size of a convolution window. The starting addresses of the two source operand arrays are src_addr1 and src_addr2, respectively. desti_addr is the starting address of the result of the operation. Expressed as : CONV #desti_addr, #src_addr1, #src_addr2, size.

4) VADD: The VADD instruction is a vector addition instruction that adds the two vector data. The addresses of these two source operands are src_addr1 and src_addr2, respectively, and the result of the addition is stored in the address desti_addr. The number of bytes in the operand is expressed in size. Expressed as : VADD #desti_addr, #src_addr1, #src_addr2, size.

B. Structure of CNN Processor

This processor supports 32-bit instructions in the RISC-V instruction set, including addition, subtraction, multiplication, division, shift, jump, load, storage, and custom instructions. The structure of CNN processor as shown in Figure.3.

The processor is a 5-level pipeline processor. Different from the conventional risc-v processor, this processor adds specific processing modules to implement customised instructions. Next, we will introduce the structure and design of each pipeline level.

- Instruction Fetch (IF): The Instruction Fetch unit loads a new instruction from the program memory. The address of the instruction to load is held by the Program Counter (PC). The Program Counter is 32-bits wide. There are three possible cases for PC: (1) When the program is normally executed, PC+4 is sent to Icache. (2) When a jump instruction occurs during program execution, the jump address is assigned to Icache. (3) When an interrupt occurs, the current PC value needs to be saved, and the interrupt entry PC is sent to the Icache.
- Instruction Decode (ID):The decoding stage is divided into general instruction decoding and customised instruction decoding. General instruction decoding separates the operation type of instruction, source operand 1 and source operand 2, and the address of the destination register to be written. The custom instruction decoding separates the signals of funct, size, src_addr1, src_addr2, desti_addr according to the operation code of the instruction, and determines the type of the custom instruction to be executed according to the funct. The structure of the decoding module is shown in Figure 4.

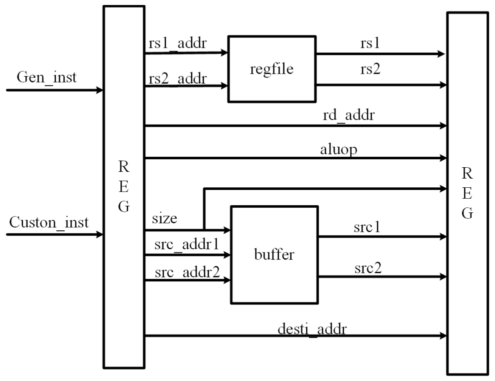

Fig. 4. The structure of the decoding module.

- Execute (ex): The Execute stage performs the required operation on the data provided by the Instruction Decode stage. The general instruction execution unit performs instructions such as logic, arithmetic, shift, multiplication and so on. The CONV instruction performs matrix multiplication transmitted from the decoding stage at the execution stage. The VADD instruction performs vector addition on the two vectors passed in the decoding stage. The structure of the execution module is shown in Figure 5.

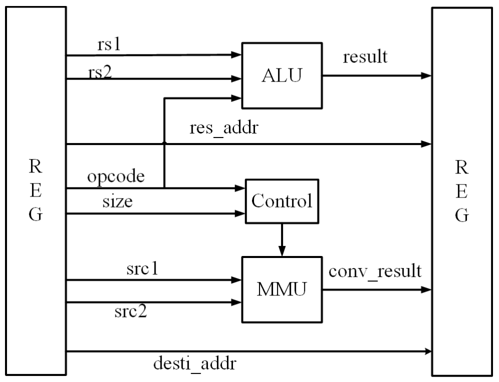

Fig. 5. The structure of the execution module.

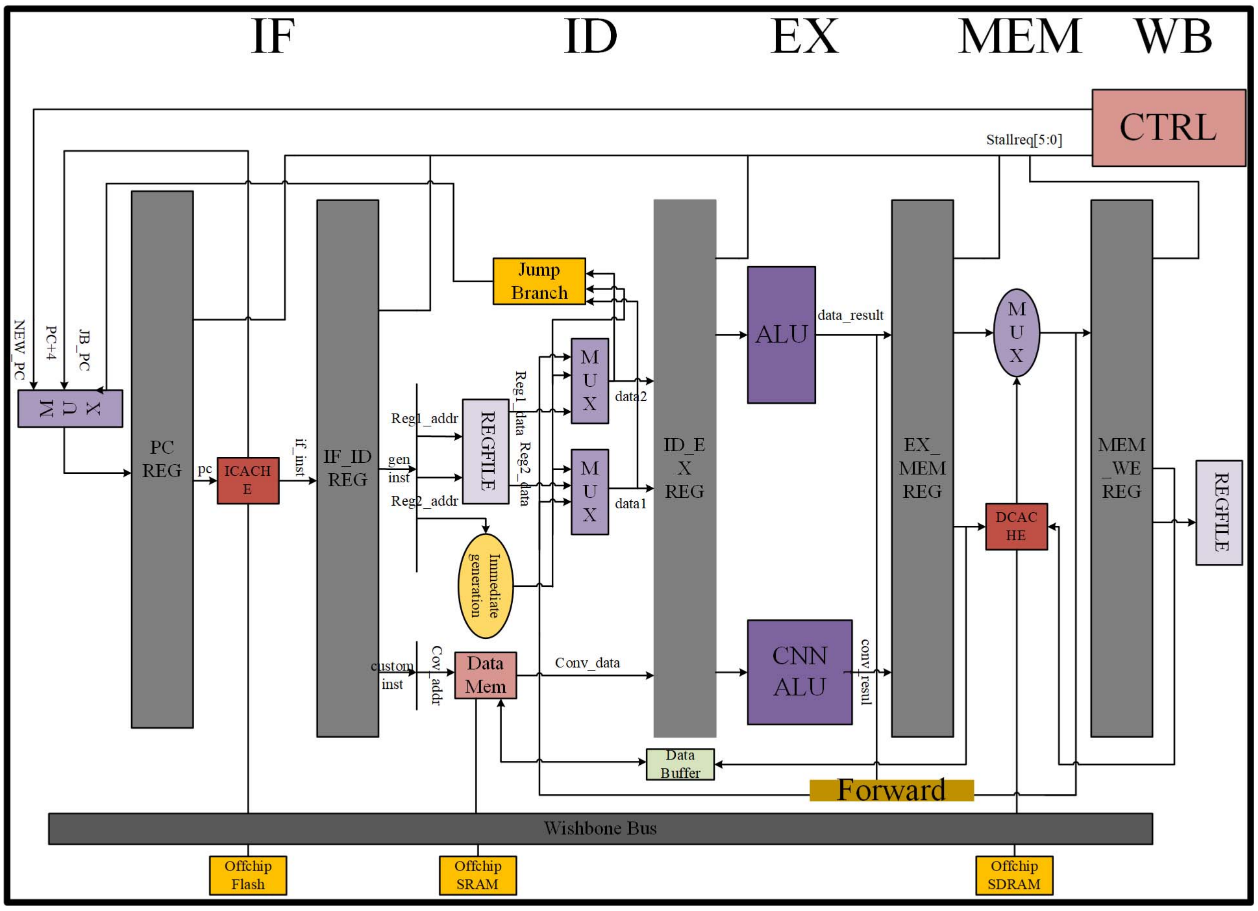

Fig. 3. Structure of CNN Processor.

In the execution phase, the opcode determines which actions will be performed. The MMU unit is a matrix operation unit for executing the VADD instruction and CONV instruction. The MMU unit consists of 25 single-precision floating-point multipliers and 24 single-precision floating-point adders, supporting up to 5×5 convolution operations.

- Memory-Access (MEM): The Memory Access stage waits for a memory read access to complete. When memory is accessed, address, data, and control signals are calculated during the Execute stage. The memory latches these signals and then performs the actual access. This means that read-data will not be available until one clock cycle later. This would be at the end of the Write-Back stage, and hence too late. Therefore the Memory-Access stage is added. The VLOAD instruction will read data from the off-chip memory to buffer during this stage.
- Write-Back (WB): The Write-Back stage writes the results from the Execution Units and memory-read operations into the Register File. The VSTORE instruction will wire data from the buffer to off-ship memory during this stage. The general instruction writes the result to the reg file register, and the VADD and CONV instructions write the result to the buffer.

C. Optimized design of data stream

Because reading data from external memory is usually a multi-cycle process, the general way for the processor to process memory access class instructions is to block the pipeline, that is, to stop the pipeline and then continue to run until the required data is obtained. However, for data-intensive CNN, there must be a large number of memory access operations. If we add several instructions that are not related to this memory instruction in the long process of fetching the number of memory instructions, or add a memory instruction that is not related to the current ones before we prepare to execute a large number of computational instructions, we can make great use of the latency of memory access and improve the efficiency of program execution. Because of the data independence between feature graphs in CNN layer, the operations can be executed in parallel, so the corresponding operation instructions of these feature graphs are also independent. Here we introduce a typical data stream processing method, table tennis operation, to efficiently develop the parallelism of intra-layer computing.

Ping-pong operation [12] is a technique for processing data streams. The ping-pong operation structure is shown in Figure 6.

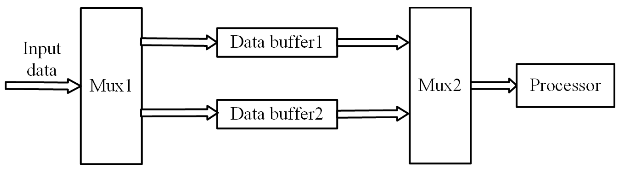

Fig. 6. The ping-pong operation structure.

The processing flow of the ping-pong operation is: the data selection unit Mux1 assigns the input data to the two data buffer modules in an equal manner. In the first processing cycle, the input data is sent to the Data Buffer1 module, and in the second processing cycle, the data selection unit Mux1 is switched to send the input data. The Data Buffer2 module also sends the data of the first processing cycle buffered in the Data Buffer1 module to the processing unit Processor for processing by the selection of the data selection unit Mux2. In the third processing cycle, Mux1 is switched again to put the input data into the Data Buffer1 module, and so on.

The processor designed in this paper needs to call multiple load instructions VLOAD to load the required data from the external memory before calculating the convolution operation. If the loading process stops the pipeline, the MMU unit will be idle if the data is not ready, which will waste a lot of time. So we make two on-chip shift registers, which use the double Buffer mode, named buf1 and buf2, which are used to buffer the data of different feature maps. When the buf1 module caches the first feature map data, the MMU can calculate the data of the second feature map previously buffered in buf2. When the data in the buf2 module is calculated, the data in the buf1 module is also cached. At this time, the MMU can continue to calculate the data in buf1 module, and the buf2 module starts to cache the data needed for the next calculation. If there is a situation where the buf1 module has not been cached after the MMU calculates the data in buf2, the wait signal is added. Through this ping-pong operation, the idle time of the MMU can be fully utilized, and the calculation efficiency is improved. We specify the data flow process as an instruction to specify the process of program execution. As shown in Figure 7.

In the double Buffer mode, the first line represents the instruction to process the A feature map, the second line represents the instruction to process the B feature map, the number identifies the order in which the instructions are called, and the box with the darkened colour indicates that the matrix operation unit is executing.

As can be seen from Figure 7, the matrix operation unit in the normal mode is idle in the process of loading data. In the double Buffer mode, if the time for loading data is less than the calculated time, the matrix operation unit is always in the calculation state. If the time to load the data is greater than the calculated time, the idle time of the matrix operation unit is equal to the time at which the data is loaded minus the time at which the data was calculated.

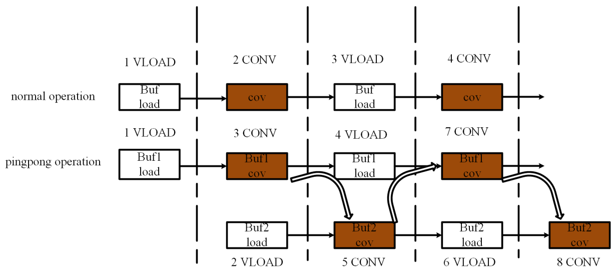

Fig. 7. Instruction flow diagram under Ping-Pong operation.

The pseudo code of the convolutional layer of the compiled CNN is as follows:

Algorithm 1 pseudo code of the convolutional

```
C( ){
    ...
    ...
    VLOAD buf1 //Load the feature map data into buf1.
    VLOAD buf2 //Load the feature map data into buf2.
    CONV(0,24) //Calculate the convolution of the data.
    VSTORE buff //Store convolution result.
    VLOAD buf1 //Load the feature map data into buf1.
    CONV(1,24) //Calculate the convolution of the data.
    VSTORE buff //Store convolution result.
    ...
    ...
    VADD(length)//Vector addition function.
    ...
    ...
  }
```

IV. Simulation and Experiment

In the previous section, we completed the processor's structural design, instruction customisation, and data path optimisation. In this section, we will simulate the processor functions. The simulation tool used in this design is Vivado 2016.4 of Xilinx Company. Vivado is powerful enough to perform circuit descriptions in a variety of ways, including VHDL, Verilog HDL, and schematics, and has a complete simulation tool. The resource occupancy rate of this processor is shown in Table 1.

We use the hardware description language Verilog to implement the design of the CNN processor. The CNN test program is written in C language. Test programs compatible with CNN processors are mainly implemented using custom instructions. The data of the input picture is stored in the off-chip SRAM, the weight is stored in the on-chip register, and the connection between the weight and the image is stored in the on-chip

TABLE I
RESOURCE OCCUPANCY RATE

Resource	Utilization	Available	Utilization%
FF	2521	202800	1.24
LUT	4304	101400	4.24
MemoryLUT	64	35000	0.18
I/O	83	400	20.75
DSP48	127	840	15.11
BUFG	11	32	34.38

register.After compiling the CNN link written in C using the RISC-V cross-compilation tool-chain, generate an executable ELF file and manually modify the machine code to implement a binary with custom instructions. The test flow chart is shown in Figure 8.

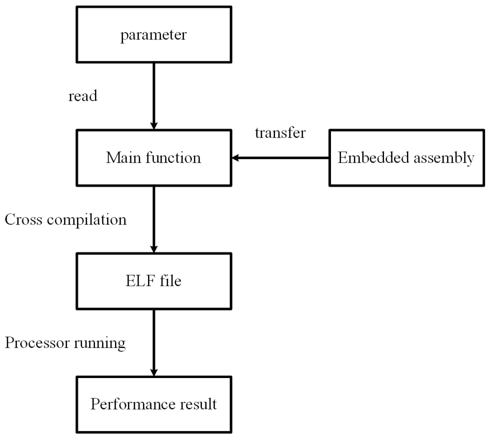

Fig. 8. Test flow.

Finally, we will simulate and verify the functionality of the processor. The input data size is 28×28 pixel data. The simulation result is shown in Figure 9.

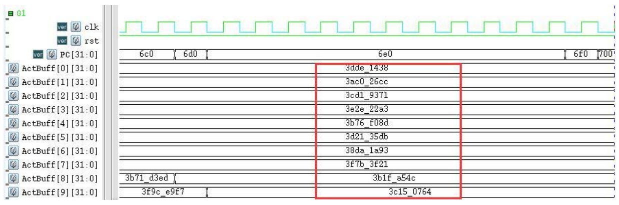

Fig. 9. Function simulation waveform.

The ten components of the classification result are converted into decimal fractions corresponding to 0.108437, 0.001466, 0.025583, 0.170054, 0.003768, 0.039358, 0.000104, 0.981432, 0.002436, 0.006096. It can be seen that the maximum value is the 7th digit 0.981432, that is, the classification result is number 7, which is consistent with the label value.

The classification results of each image are consistent with the classification results of the general software, and the correct rate is 98.25%. It indicates that the CNN processor fully supports the custom instructions and can execute the algorithm correctly.

V. CONCLUSIONS

As an advanced deep learning architecture, CNN is widely used in various fields. However, because the algorithm is data intensive and computationally intensive, the traditional CPU platform cannot fully exploit the parallelism of CNN. Pure hardware implementation CNN lacks some flexibility. Therefore, based on the characteristics of the CNN algorithm, this paper designs a new CNN processor that can balance parallelism and flexibility for the characteristics of CNN algorithm. It can not only support the execution of general algorithms but also significantly accelerate the CNN algorithm. This paper mainly completes the design of the processor based on the risc-v architecture, and successfully implements a five-stage pipeline processor. Aiming at the characteristics of CNN algorithm, four custom instructions are designed and implemented to speed up convolution operation, and optimized data transfer format to facilitate improvement of convolution efficiency. The next research plan is to reduce hardware overhead and improve accuracy.

ACKNOWLEDGEMENTS

This work was sponsored by Key Project of Hubei Provincial Department of Education under Granted No. D20181103.

REFERENCES

[1] Y. Lee, B. Zimmer, H. Cook, R. Avizienis, B. Richards, E. Alon, B. Nikolic, K. Asanovic, A. Waterman, and A. Puggelli, "Raven: A 28nm risc-v vector processor with integrated switched-capacitor dc-dc converters and adaptive clocking," 2015.
[2] Y. Lee, A. Ou, and A. Magyar, "Z-scale: Tiny 32-bit risc-v systems," in *Open-RISC Conf., Geneva, Switzerland*, 2015.
[3] S. Ji, W. Xu, M. Yang, and K. Yu, "3d convolutional neural networks for human action recognition," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, vol. 35, no. 1, pp. 221–231, 2013.
[4] A. R. Chowdhury, T. Y. Lin, S. Maji, and E. Learnedmiller, "One-to-many face recognition with bilinear cnns," 2015.
[5] X. C. Bao, R. Sahdev, and J. K. Tsotsos, "Integrating stereo vision with a cnn tracker for a person-following robot," in *International Conference on Computer Vision Systems*, 2017.
[6] T. Hori, S. Watanabe, Y. Zhang, and W. Chan, "Advances in joint ctc-attention based end-to-end speech recognition with a deep cnn encoder and rnn-lm," 2017.
[7] H. Taud and J. F. Mas, "Multilayer perceptron (mlp)," 2018.
[8] Z. Nagy and P. Szolgay, "Configurable multilayer cnn-um emulator on fpga," in *IEEE International Workshop on Cellular Neural Networks & Their Applications*, 2003.
[9] C. Farabet, C. Poulet, J. Y. Han, and Y. Lecun, "Cnp: An fpga-based processor for convolutional networks," in *International Conference on Field Programmable Logic & Applications*, 2009.
[10] M. Peemen, A. A. A. Setio, B. Mesman, and H. Corporaal, "Memory-centric accelerator design for convolutional neural networks," 2013.
[11] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: a small-footprint high-throughput accelerator for ubiquitous machine-learning," *Acm Sigplan Notices*, vol. 49, no. 4, pp. 269–284, 2014.
[12] A. J. Choudhury, H. Lim, and H. J. Lee, "Fpga implementation of pingpong-128 stream cipher for ubiquitous application," in *International Conference on Computer Sciences & Convergence Information Technology*, 2012.