International Journal of
Science and Engineering Investigations vol. 11, issue 121, February 2022
Received on February 21, 2022 ISSN: 2251-8843

Design of 32-bit ISA RISC-V Processor with CNN Instruction Integration

An Nguyen[1], Kim Anh Phan Vo[2], Hieu Nguyen[3], Hung Nguyen[4], Linh Tran[5]
[1,2,3,4,5]Department of Electronics, Ho Chi Minh City University of Technology, Ho Chi Minh City, Vietnam
([1]1652039@hcmut.edu.vn, [2]pvkanh@hcmut.edu.vn, [3]hieunt@hcmut.edu.vn, [4]ngthung@hcmut.edu.vn, [5]linhtran@hcmut.edu.vn)

Abstract-This paper presents the hardware design of a 32-bit ISA RISC-V based on Verilog HDL language, under 5-stage pipeline model. We created two new instructions to support CNN computation. The design is functional simulate with Modelsim and synthesized with Intel Quartus. The proposed design can operate with the test code, interact with the memory and perform the CNN computation on the grayscale image.

Keywords*- RISC-V, CNN, FPGA, New Instruction, CPU*

I. Introduction

The reduced instruction set computer (RISC), which give a cost-efficient processor solution as opposed to complex instruction set computer (CISC), is important to embedded computers and computers for general purposes [1]. The prevalence of RISC in the industry is astounding, and it derives from the simplicity of its instruction set, which allows for simpler instruction decoding inside the hardware. As a result, smaller hardware with equivalent performance to CISC is produced.

The RISC V instruction set architecture (ISA) is designed to be a versatile and user-friendly solution for RISC designs. It enables a modest and reliable base of instruction set that could be modify for particular purpose [2]. Furthermore, researches on RISC-V can benefit from flexible hardware such as field-programmable gate array (FPGA) because of their capacity to quickly reconfigure. Recent researches focus on applying RISC-V on convolutional neural network (CNN) application such as machine learning, image processing. Lee et al. [3] apply RISC-V CNN coprocessor for real-time epilepsy detection. Li et al. [4] implement four new instructions for RISC-V CNN operation. RV-CNN [5], [6] modify micro-architecture of the RISC-V ISA to configure it for CNN. Research [7] uses a popcount instruction to accelerate CNN. Research [8] shows a CNN RISC-V coprocessor on FPGA Kit DE2-115 as a soft-core processor. Research [9, 10] focus on energy efficiency when design their RISC-V core on CNN operation. Liu et al. [11] implement an CNN accelerator for their RISC-V separately. Recent related works also choose to modify or add new instructions into the RISC ISA [12, 13, 14]. In this paper, we will integrate two new CNN instructions in the proposed RISC-V architecture for CNN processing. The proposed contributions of this paper include:

• Hardware design of a 32-bit ISA RISC-V Processor with 5-stage pipeline configuration.

• Added new two instructions to support in CNN computation.

• Perform CNN testing of the proposed design on input grayscale images.

The rest of the paper is organized as follows: Section 2 presents the RISC-V instruction set and theoretical pipeline and hazard, the CNN integration we intended to apply as the two new instructions. Section 3 shows the proposed design of RISC-V in parts with CNN instructions integrated. In section 4, we present the synthesis result of our design and simulate on the design a Sobel filter test. Finally, we conclude the paper in section 5.

II. Theory design of rics-v and CNN instructions

A. *RISC-V*

RISC-V is an open-source hardware ISA of central processing unit (CPU) based on established simplified instruction set computing (RISC) principles.

The fundamental 32-bit integer ISA is RISC-V 32-bit (RV32I). The 32-registers on the RV32I are each 32 bits wide. Register x0 is hardwired to have all bits set to 0. General purpose registers x1–x31 store values that are interpreted by different instructions as a collection of Boolean values, two’s complement signed binary integers, or unsigned binary integers. There is one more unprivileged register: the program counter (pc), which contains the location of the currently executed instruction.

There are 4 base components included in the CPU: PC, Register Files, Arithmetic and Logical Unit (ALU) and Logic Controller (Controller). PC takes responsibility for storing the address of next executing instruction, Register Files store the temporal variable for calculating, ALU computes data based on the signal from controller.

Based on those components, the RISC-V processor is divided into 5 basic phases in every instruction executing cycle:

1. Instruction Fetch

2. Decode – Register Read

3. Execute

4. Memory Access

5. Register Write

From decoding the instruction, the processor will choose the needed components through a logic controller. In some special cases, one of these phases will be ignored.

Figure 1 present the ISA 32-bit RISC-V CPU single cycle data-path, which includes these components:

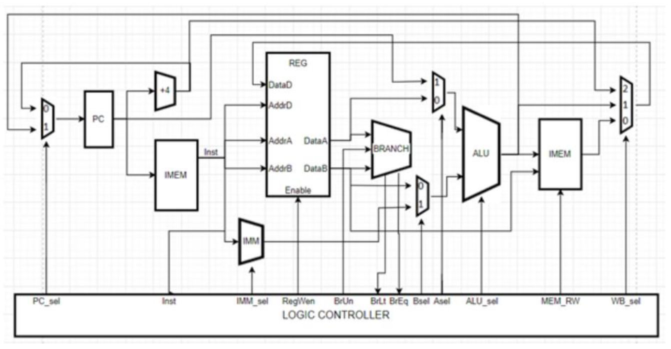

Figure 1. ISA 32-bit RISC-V data-path without pipeline

• Program counter register (PC): store the address of executed instructions

• Add-4 (+4): update the PC

• Instruction memory (IMEM): store all the instructions • Register files (REG): contains temporal data

• Immediate generator (IMM): decode generate immediate for each instruction

• Branch comparator (BRANCH): compare two operands and export the results

• A multiplexer (MUXA): choose between Registers and PC for the input of ALU

• B multiplexer (MUXB): choose between Registers and Immediate for the input of ALU

• Arithmetic and logical unit (ALU): do arithmetic or logical calculation

• Data memory (DMEM): the main memory that store data

• Write back mux (MUXWB): choose between Memory, ALU and PC+4 to be the write back data of Registers.

• PC multiplexer (MUXPC): choose the next address of instruction from PC or ALU

B. Pipeline and hazards

1) Pipeline implementation

Pipeline allows storing and executing instructions in an orderly process. The CPU is split into stages, which are linked together to form a pipe-like structure. The use of pipeline improves total instruction throughput.

Base on the single cycle structure, the CPU is divided into 5 stages: Instruction Fetch (IF), Instruction Decode (ID), Execution (EX), Memory Access (MA), Write Back (WB). Each stage takes the responsibility similar to each phase in the single cycle model. A fundamental RISC-V pipeline model is presented in figure 2.

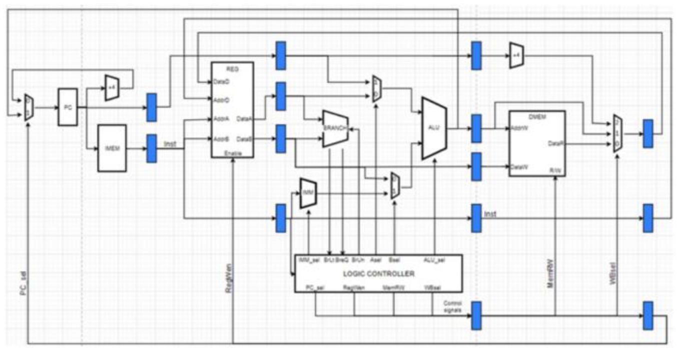

Figure 2. ISA 32-bit RISC-V data-path pipeline model

2) Hazards in Pipeline technique and sotuion

There are four types of hazards need to be solved while design a pipelined RISC-V CPU:

• Structural hazard

• Data hazard

• Control hazard

• Timing hazard

For structural hazard, it is caused by two or more instructions in the pipeline compete for access to a single physical resource (registers, memory). Figure 3 shows an example for structural hazard. The solutions for structural hazard are:

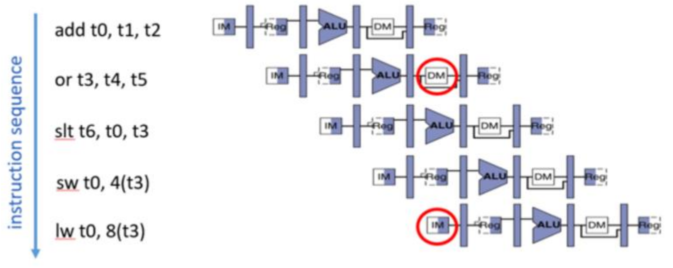

Figure 3. Example for structural hazard in RISC-V

• For registers: Design a register file that has 2 independent read ports and 1 independent write port.

• For memory: Add 2 separate caches for instruction and data if there is only one memory or use 2 memories for IMEM and DMEM. In this thesis, we use 2 separate memories.

For data hazard, it is caused by an instruction depends on completion of data access by a previous instruction. For example, register s0 used in next instructions has not been updated at the current cycle. Figure 4 shows an example for data hazard. The solutions for data hazard are:

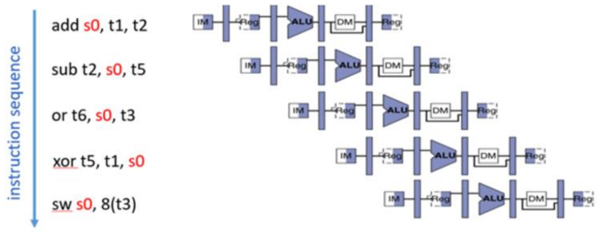

Figure 4. Example for data hazard in RISC-V

• Stall the next instruction for 2 cycles until the Registers can be updated. This solution leads to reduce performance.

• Forwarding (Bypassing): grab the operand from pipeline stage, rather than Registers. In this project, we create a module Forwarding to apply this solution.

• Rearrange code to avoid data hazard (hardware unrelated).

For control hazard, it is caused by fetching next instruction mismatches with branch outcome that result in wrong flow of control. For example, the program will go wrong if the branch BEQ taken. Figure 5 shows an example for control hazard. The solutions for control hazard are:

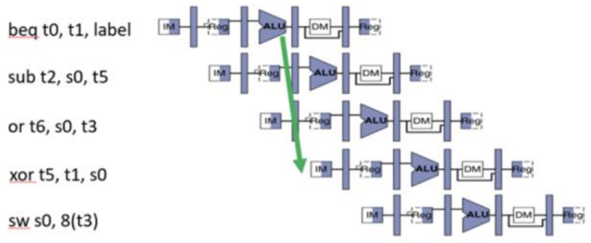

Figure 5. Example for control hazard in RISC-V

• If branch not taken, then instructions fetched sequentially after branch are correct.

• If branch or jump taken, then need to flush incorrect instructions from pipeline by converting to NOPs (2 cycles after) and fetch the correct instructions.

For timing hazard, it is caused by the loading process takes too much time for accessing the memory, so in the case of forwarding, there is not enough time for data processing. For example, AND instruction use the same register with LOAD instruction and it need forwarding. However, the memory exports at the late of the cycle, which causes error. Figure 6 shows an example for timing hazard. The solution for timing hazard is stalling the processor for one cycle to wait for memory loading.

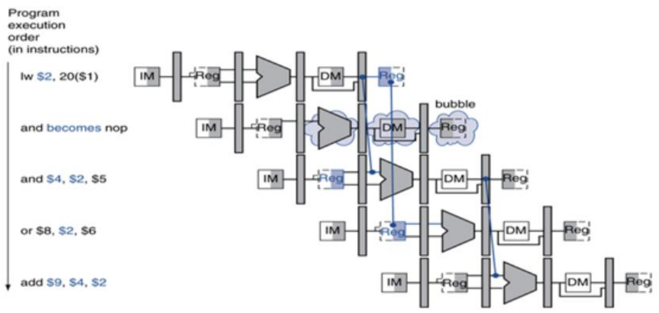

Figure 6. Example for timing hazard in RISC-V

C. *CNN Integration*

In this section, we present two new CNN instructions: Convolution and maxpooling. These are the two CNN instructions integrated in our 32-bit ISA RISC-V design.

1) Convolution theory

Convolution is one of the main building blocks of a CNN. The term convolution refers to the mathematical combination of two functions to produce a third function. It merges two sets of information.

In the case of a CNN, the convolution is performed on the input data with the use of a filter or kernel (these terms are used interchangeably) to then produce a feature map. We execute a convolution by sliding the filter over the input. At every location, a matrix multiplication is performed and sums the result onto the feature map.

Figure 7 shows the example of convolution on image.

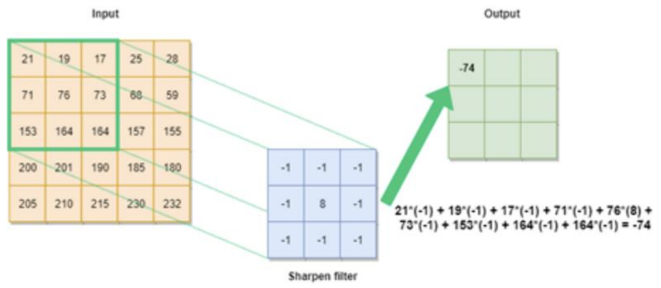

Figure 7. Example of convolution on image

We perform numerous convolutions on our input, where each operation uses a different filter. This results in different feature maps. In the end, we take all of these feature maps and put them together as the final output of the convolution layer.

2) Convolution instruction theory

After creating the instruction to load feature map from the memory and store the values of pixels from R1 to R9, we attempt to create the new instruction to take the convolution results between the feature map and the filter. According to the filter value, again, we fit the functionality of store 9 pixels of filters from R10 to R18 in RISC-V Register. Table 1 present both of these operations.

TABLE I. DESIGN CONVENTION FROM R1 TO R18

R1	Pixel 1 of input image
R2	Pixel 2 of input image
R3	Pixel 3 of input image
….	….
R9	Pixel 9 of input image
R10	Pixel 1 of filter
R11	Pixel 2 of filter
R12	Pixel 3 of filter
….	….
R10	Pixel 9 of filter

By fixing the function of Register, now it has the output port is "Convolution" and the output result Convolution [31:0] is calculated by:

$$\frac{(R1\times R10)+(R2\times R11)+(R3\times R12)}{(R10+R11+R12+R13+R14+R15+R16+R17+R18)}+\frac{(R4\times R13)+(R5\times R14)}{(R10+R11+R12+R13+R14+R15+R16+R17+R18)}+\frac{(R6\times R15)+(R7\times R16)}{(R10+R11+R12+R13+R14+R15+R16+R17+R18)}+\frac{(R8\times R17)+(R9\times R18)}{(R10+R11+R12+R13+R14+R15+R16+R17+R18)}$$

3) Convolution instruction in design

Table 2 presents the convolution instruction structure. The custom convolution instruction has 32 bit with:

TABLE II. CONVOLUTION INSTRUCTION IN MACHINE LANGUAGE

31 25	24 20	19 15	14 12	11 7	6 0
Imm[11:5]	RS2	RS1	000	Imm[4:0]	0000101

• Bit 6 to 0: Opcode of the instruction (opcode = 7'b0000101)

• Bit 11 to 7: The offset Imm[4:0]

• Bit 14 to 12: Do not have any function, so I set it to 3'b000

• Bit 19 to 15: Do not have any function.

• Bit 24 to 20: The register holds the base address to be stored in memory

• Bit 31 to 25: The offset Imm [11:5]

Our new convolution instruction will have the function:

Convolution Out => MEM[RS2+Imm[11 : 0]]

4) Maxpooling instruction

Maximum pooling, or max pooling, is a pooling operation that calculates the maximum, or largest, value in each patch of each feature map. Figure 8 shows an example of maxpooling.

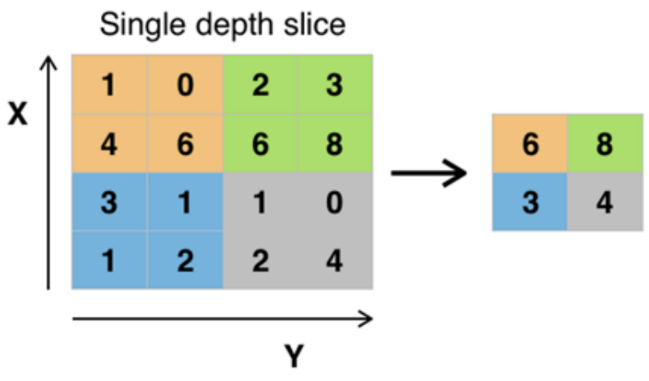

Figure 8. Maxpooling example

The results are down sampled or pooled feature maps that highlight the most present feature in the patch, not the average presence of the feature in the case of average pooling. We can make the max pooling operation concrete by again applying it to the output feature map of the line detector convolutional operation and manually calculate the first row of the pooled feature map.

5) Maxpooling instruction in design

We introduce the second custom instruction: Maxpooling. The instruction takes the largest value from R1 to R7, which is feature map as an input, and store that value at the Address RS2 + Imm[11:0] in the memory. Table 3 presents the maxpooling instruction structure. The custom maxpooling instruction has 32 bit:

• Bit 6 to 0: Opcode of the instruction (opcode = 7'b0001001)

• Bit 11 to 7: The offset Imm[4:0]

• Bit 14 to 12: Do not have any function, so I set it to 3'b000

TABLE III. CONVOLUTION INSTRUCTION IN MACHINE LANGUAGE

31 25	24 20	19 15	14 12	11 7	6 0
Imm[11:5]	RS2	RS1	000	Imm[4:0]	0001001

• Bit 19 to 15: Do not have any function.

• Bit 24 to 20: The register holds the base address to be stored in memory

• Bit 31 to 25: The offset Imm [11:5]

Our new maxpooling instruction will have the function:

Maxpooling => MEM[RS2+Imm[11 : 0]]

We present an example of the maxpooling instruction in table 4. In this case, the maxpooling instruction get the largest value from R1 to R9, which is 244, and the store the value in Imm[31:7] address in the memory

TABLE IV. MAXPOOLING EXAMPLE BY REAL DATA

Registers	Datat Registers hold
R1	134
R2	244
R3	156
R4	44
R5	0
R6	244
R7	79
R8	25
R9	144

III. HARDWARE DESIGN OF PIPELINE RISC-V WITH CNN INTERGRATION

The RISC-V pipelined processor consists of 5 stages: Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory Access (MA), Write Back (WB). The proposed processor design is shown in figure 9.

• Program Counter (PC)

– Function: The PC is a module that increases the PC to point to the next instruction.

– Behavior: Using to the signal provide from Controller, the output will stall the PC, or choose PC = PC +4

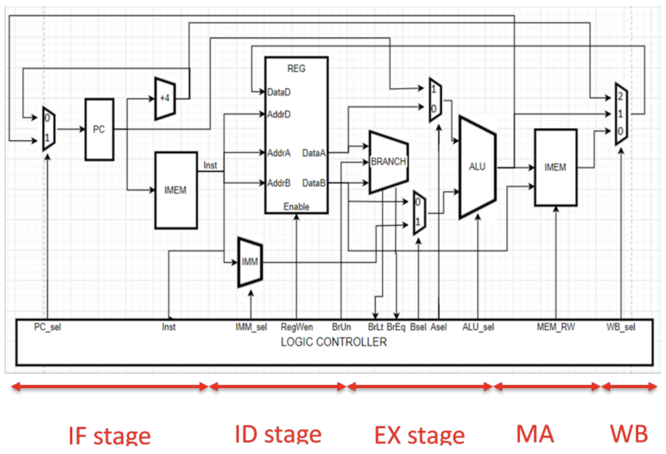

Figure 9. The proposed processor design with all main components

• Instruction memory (IMEM)

– Function: This module is read-only-memory (rom) that read the instruction stored inside this block.

– Behavior: At each positive edge memory clock (clkmem), IMEM reads the content at the provided address. • Register Files

– Function: Registers keep temporal variables for calculation.

– Behavior: The registers take two clock sources which are base clock and memory clock. At each positive edge of memory clock, the module outputs the contents of which Address A and Address B point to. Also at this edge, if the Write Enable is set, the data will be written into the registers.

• Arithmetic and Logical Unit (ALU)

– Function: This unit responsible for the arithmetic and logical computation for the processor.

TABLE V. CONVOLUTION INSTRUCTION IN MACHINE LANGUAGE

INST1	INST2	INST3	Rs_inst1 = rd_inst2	Rs_inst1 = rd_inst3	DATA	Selection Signal	Stall
R_inst	NM_Inst	NM_Inst	x	x	inst1	00	0
R_inst	NM_Inst	MR_Inst	x	0	inst1	00	0
R_inst	NM_Inst	MR_Inst	x	1	inst3	01	0
R_inst	MR_Inst	NM_Inst	0	x	inst1	00	0
R_inst	MR_Inst	NM_Inst	1	x	inst2	10	0
R_inst	MR_Inst	MR_Inst	0	0	inst1	00	0
R_inst	MR_Inst	MR_Inst	0	1	inst3	01	0
R_inst	MR_Inst	MR_Inst	1	0	inst2	10	0
R_inst	MR_Inst	MR_Inst	1	1	inst2	10	0
R_inst	L-Type	NM_Inst	0	x	inst1	00	0
R_inst	L-Type	NM_Inst	1	x	inst2	10	1
R_inst	L-Type	MR_Inst	0	0	inst1	00	0
R_inst	L-Type	MR_Inst	0	1	inst3	01	0
R_inst	L-Type	MR_Inst	1	x	inst2	10	1
N_inst	x	x	x	x	inst0	00	0

– Behavior: ALU takes the ALU selection signals (alusel) from the Controllers to decide the operational mode between two input operands and export the result. There are ten working mode of this unit which are Addition, Subtraction, Bitwise-And, Bitwise Or, Bitwise-Xor, Logical Shift Left, Logical Shift Right, Arithmetic Shift Right, Signed Comparison and Unsigned Comparison.

• Branch Comparison (BRANCH)

– Function: BRANCH does the comparison and give the result for processing branch instruction.

– Behavior: Because the BRANCH only serves branch instructions so it only takes two data input from registers (dataA and dataB) and one decoded signal from Controller which indicates signed or unsigned comparison. The outputs determine whether dataA is equal (breq = 1), less than (brlt = 1) or greater than (breq = 0 and brlt = 0).

• Immediate Generator (IMM)

– Function: IMM create immediate for each type of instruction

– Behavior: the module is controlled by IMM mode selection signals (immsel) from the Controller. The IMM then reorders the input signals and adds sign-extend bits to the immediate to create full 32 bits operand in the output (immout).

• Logic Controller (CONTROLLER)

– Function: Logic Controller's task is that identifies the instruction and sends control signal to the components of each stage.

– Behavior: the module takes instruction and starts to decode. After that, bases on the result, the CONTROLLER sends active signals to the needed components for this instruction. Especially, when encounters branch instructions, the controller sends control signals then waits for the result and drive the control flow.

• Memory selection (MEMSEL)

– Function: The MEMSEL modules only supports for Load instructions. This module will take the output data from the DMEM, choose which bits need to be loaded and sign-extend these bits.

– Behavior: The modules take bits 12 to 14 from the instruction which indicates the type of load instruction, to be the selection mode. If the instruction is LB or LBU, the last byte will be taken and sign-extend. Else, in case of LH or LHU, last two bytes will be chosen and sign-extend. Finally, if the instruction is LW, the whole bits will be taken.

• Data memory (DMEM)

– Function: DMEM loads and stores the data for all the programs.

– Behavior: DMEM contains two parts which are memory controller and the main memory. The controller uses instruction at MA stage, address, write data and memory clock to be inputs. When the instruction is S-Type, the controller will drive one, two or four bytes depended on which type of store instruction, to one, two or four consecutive addresses of memory. If the instruction is L-Type, four consecutive addresses will be fetched out to load the data.

We also use another module called forwarding control. The module takes three instructions from EX, MA and WB stage for checking all the cases that there are interferences in register contents between two or more consecutive instructions. It will determine which register needs to be grabbed or when the load instruction stalls for ensure the data are updated timely.

We compare the instruction in Execution stage (inst1) versus two previous instructions which are in Memory Access stage (inst2) and Write Back stage (inst3). Because the data, when inference happens, needed to be driven from the destination register of previous instructions to the source registers of current instruction, so we will compare the source register of inst1 (rs inst1) with destination register of inst2 and inst3 (rd inst2 and rd inst3). Additionally, the picked data must be the latest data so the comparison with inst2 is set to higher priority.

We divide the instruction in EX stage into two types: one source register used instruction (R Inst) and no source registers used instruction (N-Inst). Besides, the instruction in MA stage and WB stage is also classified into two types: register modified instruction (MR Inst) and non-register modified instruction (NR Inst). However, there is one exception that due to the time delay in loading data from memory, the processor must stall one cycle when grabbing data from Load instruction to ensure the timing requirement.

The outputs of the module are Afsel, Bfsel and stall signal. The forwarding control truth table is shown in table 5.

TABLE VI. SYTHESIS RESULT OF PROPOSED DESIGN AND COMPONENTS

	Area				Speed[MHz]
	ALM	Registers	BRAM	DSP	
Proposed design	3715	2091	78	18	43
Baranch & Forwarding Control	40	70	3	0	-
Memory & Control	5.2	2	75	0	-
ALU	237	0	0	0	-
Register & Controllers	226	992	0	0	-
CNN New Instructions	2348	631	0	18	-

IV. Synthesis result and functional simulation

A. *Synthesis on Quartus*

We achieve the overall speed of 43 MHz for the proposed design of RISC-V with CNN new instructions when synthesized with Quartus. The proposed design and its components synthesis result is shown in table 6.

We present some of RISC-V main entities to show the portion of resources that CNN New Instructions accounts for. The CNN New Instructions takes 63% of the ALM and all the DSPs used in the proposed RISC-V design. Because we do not utilize resource re-use, the logic mathematical functions of Maxpooling and Convolution instructions uses many ALM resources. In future research, it is important that we integrate the computation heavy tasks in these instructions to the ALU.

The overall speed of the design could also be improved by optimize the path between the original RISC-V CPU and the new CNN instructions core, which could bring the design speed back to 140 MHz, similar to the original RISC-V design.

B. *Simulation on processing application*

In this section, we used the proposed RISC-V with CNN instructionsß design to simulate a CNN functions called Sobel filter. The Sobel operator, sometimes called the Sobel–Feldman operator or Sobel filter, is used in image processing and computer vision, particularly within edge detection algorithms where it creates an image emphasizing edges [15].

Sobel and Feldman presented the idea of an ”Isotropic 3 × 3 Image Gradient Operator” at a talk at SAIL in 1968. Technically, it is a discrete differentiation operator, computing an approximation of the gradient of the image intensity function. At each point in the image, the result of the Sobel–Feldman operator is either the corresponding gradient vector or the norm of this vector.

The Sobel–Feldman operator is based on convolving the image with a small, separable, and integer-valued filter in the horizontal and vertical directions and is therefore relatively inexpensive in terms of computations. On the other hand, the gradient approximation that it produces is relatively crude, in particular for high-frequency variations in the image. Figure 10 presents the Sobel filer used in detecting the edge.

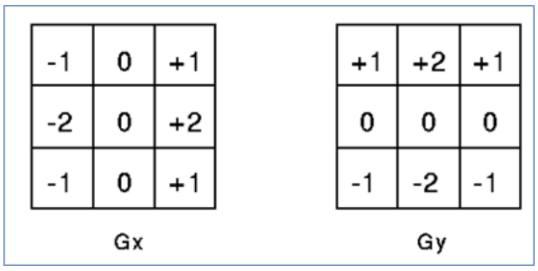

Figure 10. Sobel filter used in detecting the edge [16]

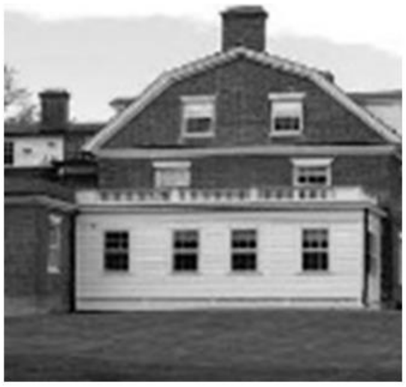

Figure 11. Input data 1, grayscale image 128x128 resolution

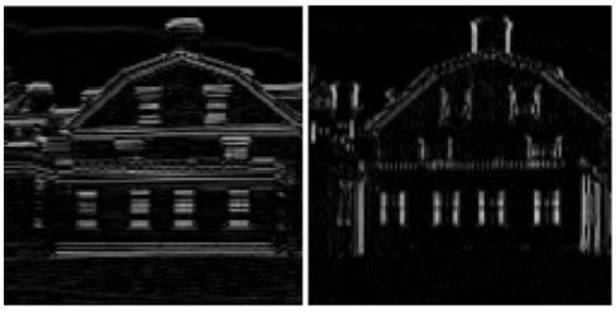

Figure 12. Output data1, detect horizontal and vertical edge

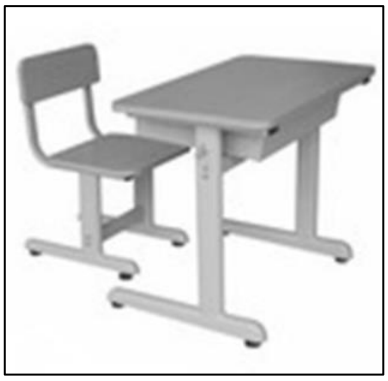

Figure 13. Input data 2, grayscale image 128x128 resolution

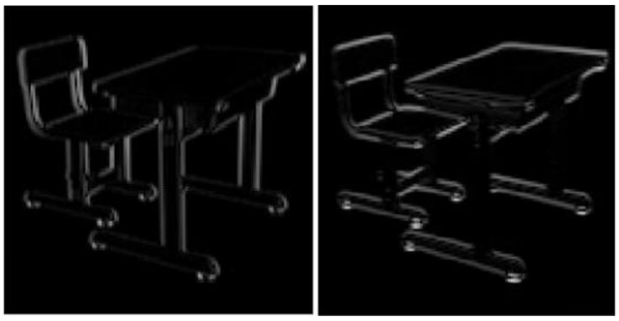

Figure 14. Input data 2, grayscale image 128x128 resolution

The simulation applies this Gx and Gy filter, using the original and custom instruction to perform CNN computation. Our target is checking if it is possible to detect the edge of the image.

The processor processed the image and return the result to the memory correctly. We show the first example image and result in figure 11 and figure 12 and the second example image and result in figure 13 and figure 14, respectively. This proves the two CNN instructions worked successfully according to our theory and design.

V. CONCLUSION

The 32-bit ISA RISC-V with CNN instruction integration can perform CNN computation by applying Sobel kernel to detect the vertical and horizontal edge. The design is virtually simulated on the Modelsim software with all the signals, the logical functions and memory behave correctly.

In future research, we will create cache memory module and use external memories to speed up the processor. In addition, to improve the performance of RISC-V, new technologies could be integrated such as multiprocessor, multi-scalar, etc. Furthermore, we need to implement extra data bus interface to connect with the peripherals. The proposed processor could be embedded to do tasks such as machine learning.

ACKNOWLEDGMENT

This research is funded by Ho Chi Minh City University of Technology (HCMUT), VNU-HCM under grant number To-DDT-2020-09. We acknowledge the support of time and facilities from Ho Chi Minh City University of Technology (HCMUT), VNU-HCM for this study.

REFERENCES

[1] G. Kane, *mips RISC Architecture*. Prentice-Hall, Inc., 1988.

[2] A. Raveendran, V. B. Patil, D. Selvakumar, and V. Desalphine, "A risc-v instruction set processor-micro-architecture design and analysis," in *2016 International Conference on VLSI Systems, Architectures, Technology and Applications (VLSI-SATA)*, pp. 1–7, IEEE, 2016.

[3] S.-Y. Lee, Y.-W. Hung, Y.-T. Chang, C.-C. Lin, and G.-S. Shieh, "Risc-v cnn coprocessor for real-time epilepsy detection in wearable application," *IEEE transactions on biomedical circuits and systems*, vol. 15, no. 4, pp. 679–691, 2021.

[4] Z. Li, W. Hu, and S. Chen, "Design and implementation of cnn custom processor based on risc-v architecture," in 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), pp. 1945–1950, IEEE, 2019.

[5] W. Lou, C. Wang, L. Gong, and X. Zhou, "Rv-cnn: flexible and efficient instruction set for cnns based on risc-v processors," in *International Symposium on Advanced Parallel Processing Technologies*, pp. 3–14, Springer, 2019.

[6] Q. Jiao, W. Hu, Y. Wen, Y. Dong, Z. Li, and Y. Gan, "Design of a convolutional neural network instruction set based on risc-v and its microarchitecture implementation," in *International Conference on Al gorithms and Architectures for Parallel Processing*, pp. 82–96, Springer, 2020.

[7] O. Myllynen, "Latch-based risc-v core with popcount instruction for cnn acceleration," *Energy*, 2021.

[8] R. Porter, S. Morgan, and M. Biglari-Abhari, "Extending a soft-core risc-v processor to accelerate cnn inference," in *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*, pp. 694–697, IEEE, 2019.

[9] Y.-W. Hung, Y.-T. Chang, S.-Y. Lee, C.-C. Lin, and G.-S. Shieh, "An energy-efficient and programmable risc-v cnn coprocessor for real-time epilepsy detection and identification on wearable devices," in *2021 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW)*, pp. 1–2, IEEE, 2021.

[10] A. Garofalo, M. Rusci, F. Conti, D. Rossi, and L. Benini, "Pulp- nn: accelerating quantized neural networks on parallel ultra-low-power risc-v processors," *Philosophical Transactions of the Royal Society A*, vol. 378, no. 2164, p. 20190155, 2020.

[11] Z. Liu, J. Jiang, G. Lei, K. Chen, B. Qin, and X. Zhao, "A heterogeneous processor design for cnn-based ai applications on iot devices," *Procedia Computer Science*, vol. 174, pp. 2–8, 2020.

[12] S. Wang, J. Zhu, Q. Wang, C. He, and T. T. Ye, "Customized instruction on risc-v for winograd-based convolution acceleration," in *2021 IEEE 32nd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pp. 65–68, IEEE, 2021.

[13] X. Qin, X. Liu, and J. Han, "A cnn hardware accelerator designed for yolo algorithm based on risc-v soc," in *2021 IEEE 14th International Conference on ASIC (ASICON)*, pp. 1–4, IEEE, 2021.

[14] E. Flamand, D. Rossi, F. Conti, I. Loi, A. Pullini, F. Rotenberg, and L. Benini, "Gap-8: A risc-v soc for ai at the edge of the iot," in *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pp. 1–4, IEEE, 2018. [15] I. Sobel, R. Duda, P. Hart, and J. Wiley, "Sobel-feldman operator,"

[15] I. Sobel, R. Duda, P. Hart, and J. Wiley, "Sobel-feldman operator,"

[16] "Sobel edge detector." https://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm.

How to Cite this Article:

Nguyen, A., Vo, K. A. P., Nguyen, H., Nguyen, H. & Tran, L. (2022). Design of 32-bit ISA RISC-V Processor with CNN Instruction Integration. International Journal of Science and Engineering Investigations (IJSEI), 11(121), 14-21. http://www.ijsei.com/papers/ijsei-1112122-03.pdf