RI-MAC: Optimising MAC Operation using Custom RISC-V Instruction Set for Neural Network Inference

Imlijungla Longchar*
Department of CSE
IIT Guwahati, Guwahati, India
ilongchar@iitg.ac.in

Kaustubh Khutal*$
Department of IT
RSCOE, Savitribai Phule Pune University
kaustubh2e@gmail.com

Ashwath C. Reddy*$
Department of EE
IIT Tirupati, Tirupati, India
ee22b009@iittp.ac.in

Amey Varhade
Department of CSE
IIT Guwahati, Guwahati, India
varhade@alumni.iitg.ac.in

Hemangee K. Kapoor
Department of CSE
IIT Guwahati, Guwahati, India
hemangee@iitg.ac.in

2025 5th International Conference on AI-ML-Systems (AIMLSystems) | 979-8-3315-7318-8/25/$31.00 ©2025 IEEE | DOI: 10.1109/AIMLSYSTEMS67835.2025.11330892

***Abstract*—In Convolutional Neural Networks (CNNs), which are widely used in Computer Vision, most of the computation happens in the convolution layers, which rely heavily on repeated multiply-and-accumulate (MAC) operations. Optimizing these operations can significantly accelerate performance. Previous works have shown that reducing the precision of weights often has a minimal impact on accuracy. Having varying precisions for weights and activations for different layers allows us to reduce the computational overhead with a minor trade-off on the accuracy of these models. These optimisations can be done at the hardware, software, or both levels. In this regard, we consider hardware optimizations using RISC-V. The RISC-V architecture can be adapted and customised for domain-specific applications. Particularly, the integration of the custom instructions with an acceleration module can enhance the convolution operations.**

In this paper, we propose a reconfigurable multiplier unit, RUnit, which is capable of handling multiplications of operands having different bit-widths. We also provide a custom RISC-V instruction set for using our RUnit. Our reconfigurable multiplier is capable of handling multiple precision levels to enhance inference efficiency as per the custom instruction. When the unit is not used in full precision, it is capable of performing multiple low-bit operations in parallel. We achieve a maximum speedup of around 8× over baselines on execution times for CNN inference.

***Index Terms*—DNN, CNN, Reconfigurable Arithmetic, Neural Networks, Low Precision, RISC-V, custom instruction sets, accelerators**

I. Introduction

The demand for AI-specific hardware arises from the limited customization offered by traditional CPUs and GPUs, which struggle to support optimizations common in AI algorithms. Standard computing systems often fall short, prompting the development of efficient, high-performance AI accelerators. These accelerators are optimised for operations like matrix and vector computations, aiming to meet AI workloads while minimizing power consumption and cost.

* The authors have contributed equally to this research.
$ This work was done while the author was visiting IIT Guwahati

Convolutional Neural Networks (CNNs), a class of Deep Neural Networks (DNNs) used in image processing, consist of convolutional (CONV) layers, pooling layers, and fully connected layers. CONV layers compute dot products between input activations and weights, pooling layers reduce spatial dimensions, and fully connected layers perform classification. Each layer's output serves as the input for the next, with successive CONV layers extracting increasingly abstract features. Inference concludes with a final prediction. Both training and inference involve millions of dot-product operations, primarily implemented as multiply-and-accumulate (MAC) operations.

Convolution requires fetching data, performing computations, and storing results—operations that consume memory bandwidth and introduce latency. Hardware accelerators aim to optimise these factors for faster inference. This paper focuses on the MAC units within the architecture. MAC units can be implemented either bit-serially or bit-parallel, with latency depending on the operand bit-width or the number of MAC units. For hardware scalability across different networks, MACs are typically designed for the maximum required precision. Optimizing the number and latency of multiplications can significantly improve inference efficiency and performance. This work specifically targets the multiplication component of the MAC unit.

II. Background and Related Work

Various designs covering varying aspects have been proposed earlier to achieve efficient inference of Neural Networks. Before moving on to our work, we will set up some preliminaries here. This section is further divided into four parts. In the first part, we discuss the previous work and its limitations and place our work in the existing literature landscape. In the next few subsections, we introduce the preliminaries required for our work. The second part explains the RISC-V ISA, the

next one explains our multiplier, and the last section explains CNNs.

A. Related Work

Convolution operation in neural network inference is a computationally heavy task. Especially since it involves a significant number of multiplying operations followed by accumulating operations. Researchers have addressed optimising and speeding up the inference by either optimising the MAC units or lowering the bit-precision.

Lower and variable bit-precisions:
Previous works have been performed on the use of CNNs with reduced precision. Eyeriss [1] is an accelerator specifically designed for CNNs, featuring 168 processing elements (PEs) with a precision of 16 bits. It prioritises the energy efficiency of the entire system in various CNN configurations by dynamically reconfiguring its architecture. The work [2] studies the effect of quantization on different layers within a model and identifies a configuration that achieves minimal drop in accuracy. CNNs with fixed 8-bit precision weights have been studied in [3]. Another work [4] stretches quantization to its limits by having 1-bit weights and 2-bit activations.

Optimising MAC operations:
Exploration has been made to accelerate neural network workloads in FPGA as well as ASIC platforms. Bit-Fusion [5] introduces the concept of fused PEs that dynamically fuse during runtime to support various combinations of bit-width multiplier with the overhead of a new design of ISA. Bit-Beading [6] and Mini-Mac [7] adopt booth encoding for smaller multiplier units, where the former directly composes larger multiplier using smaller units and the latter adopts a hierarchical approach. Stripes [8] adopts a serial-wise processing where the execution cycle is proportional to the precision of the data. Compression techniques are also adapted to reduce memory access and optimise the computation with some overhead on the controller and design logic [9] [10].

Recently, the RISC-V ISA has attracted attention due to its flexible, customisable instruction set. These have been used to perform acceleration for edge computing platforms [11] and cryptographic computations [12]. The customisability of RISC-V is also increasing the interest of researchers for its use in CNN implementations [13], [14] [15].

Lower-bit representations are crucial in modern deep learning, particularly for convolutional neural networks (CNNs), as they significantly reduce memory bandwidth, storage requirements, and computational load. Using reduced precision—such as 8-bit, 4-bit, or even binary formats—instead of standard 32-bit floating-point not only compresses the model size but also accelerates inference by simplifying arithmetic operations. However, traditional hardware like general-purpose CPUs and even standard GPUs are not optimised for these ultra-low-precision operations. This drives the need for specialised hardware, including reduced-precision multipliers and custom-designed processing units that can efficiently execute quantised operations. Custom instructions that are tailored to handle packed low-bit data, perform parallelised operations,

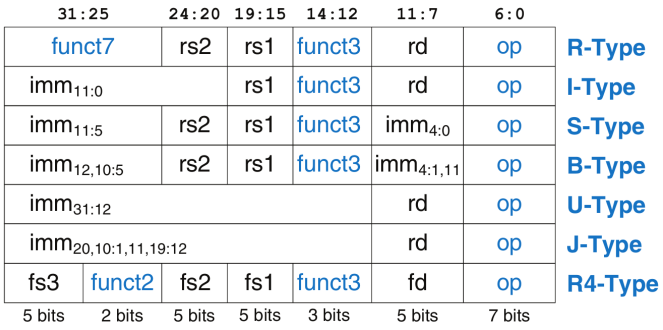

Fig. 1. Base Set of Instructions in RISC-V Formats

and exploit data reuse patterns in CNNs deliver both power efficiency and throughput unattainable with conventional architectures. Working on similar lines, in this paper, we propose a custom instruction set for a reconfigurable MAC unit.

B. RISC-V ISA

The RISC-V Instruction Set Architecture (ISA) is designed with a modular and extensible structure, beginning with a minimal "base" set of integer instructions. These instructions are organised into six fixed 32-bit formats—R, I, S, B, U, and J—as illustrated in Figure 1. The base subset, referred to as 'I', encompasses essential operations such as load, store, arithmetic, immediate value manipulation, and control flow (e.g., branch and jump). Additional optional subsets can be layered on top, such as 'M' for integer multiplication and division (which uses the R-type format), or 'F' and 'D' for floating-point operations (using R- and I-type formats). A 32-bit implementation including both the I and M subsets is conventionally labeled RV32IM. To support flexibility and future compatibility, RISC-V claims only the opcode patterns it requires, reserving the rest for custom or domain-specific instructions, thus enabling safe ISA extensions without conflict with future standard features.

Here, we briefly touch upon the instructions outlined in Figure 1. The figure depicts the six primary instruction formats used in the RISC-V RV32I instruction set: R-Type, I-Type, S-Type, B-Type, U-Type, J-Type, and the extended R4-Type format. Each format is exactly 32 bits wide but arranged differently to serve distinct instruction purposes.

1) The **R-Type** format, used for register-register arithmetic and logical instructions, includes `funct7` (7 bits), `rs2` and `rs1` (5 bits each for source registers), `funct3` (3 bits for sub-operation encoding), `rd` (5 bits destination register), and `opcode` (7 bits).
2) The **I-Type** format, used for immediate arithmetic and loads, replaces `funct7` and `rs2` with a 12-bit immediate field (`imm[11:0]`), keeping the remaining fields the same. In the **S-Type** format, used for store instructions, the 12-bit immediate is split across `imm[11:5]` and `imm[4:0]`, flanking the source registers and `funct3`.
3) The **B-Type** format, used for conditional branches, rearranges the immediate bits into a signed offset spread

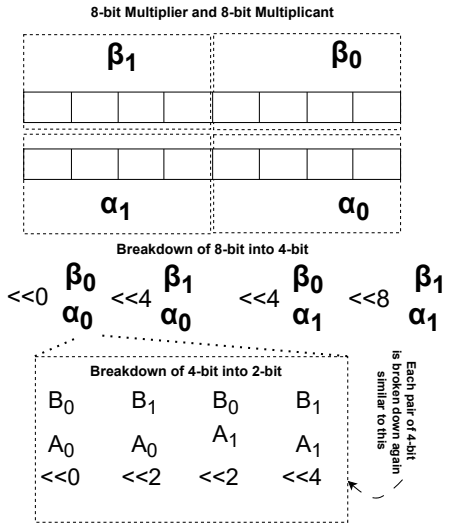

Fig. 2. Abstract diagram showing breakdown of 8x8 into 4x4

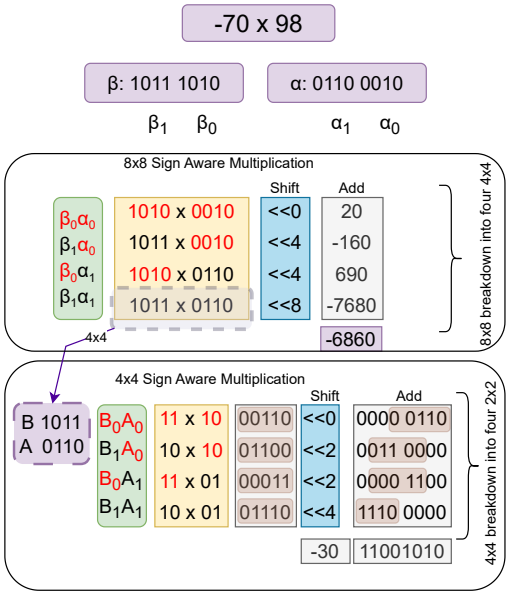

Fig. 3. Breakdown Illustration

over non-contiguous fields: `imm[12]`, `imm[10:5]`, `imm[4:1]`, and `imm[11]`.

4) The **U-Type** format (used in instructions like LUI and AUIPC) reserves the upper 20 bits `imm[31:12]` for the immediate value, followed by `rd` and `opcode`. The **J-Type** format (used for JAL) arranges the jump offset across multiple bit slices: `imm[20]`, `imm[10:1]`, `imm[11]`, and `imm[19:12]`, enabling far jumps.
5) Lastly, the **R4-Type** format, typically used in floating-point instructions with three sources, contains fields `fs3`, `funct2`, `fs2`, `fs1`, `funct3`, `fd`, and `opcode`. Further, we will define our own custom instructions in the extended R4 format in the next section on our proposed work.

C. Convolution Neural Networks

Convolutional Neural Networks (CNNs) [16], [17], [18]are a class of deep learning models particularly effective for processing data with a grid-like structure, such as images. Convolutional Neural Networks (CNNs) rely on the convolution operation, where a small matrix of learnable weights—called a kernel or filter— moves over the input for the detection of patterns. At each position, the filter and the input perform element-wise multiplication and summation, and form a feature map that highlights certain features such as edges, textures, or shapes. This process, known as a multiply-and-accumulate (MAC) operation, is repeated across the entire input and constitutes the core computational workload in CNNs.

Key parameters like stride, which controls the sliding space of the filter, affect the output. Pooling layers are used to reduce the spatial dimensions, while the ReLU activation function introduces non-linearity. CNNs build and extract features from the data by having multiple layers of these operations, thereby making them highly effective for tasks such as image classification and object detection.

There are several convolution acceleration algorithms in use today, namely im2col [19], Winograd [20], and FFT (Fast Fourier Transform) [21]. FTT is particularly useful when we have larger kernels; however, it involves complex numbers, and the fundamental operations change.

III. Proposed Work

We propose the design of a multiplier that can work on different bit-widths. In particular, we compose bigger multipliers using smaller units. We call our designed reconfigurable multiplier as *'RUnit'*. In the cases when we use the RUnit for smaller bit-width operations, we can perform multiple operations in parallel in the same cycle. We integrate RUnit inside the RISC-V processor and design a custom instruction set for it. We support our design with simulation and synthesis results.

In this section, we first discuss the design of a reconfigurable multiplier, followed by an example. RISC-V ISA is discussed, and our proposed custom instructions are described in detail.

A. Design of Reconfigurable MAC

When two n-bit operands A and B (where n is a power of two) are split into high and low halves:

$$A = A_H \cdot 2^{\frac{n}{2}} + A_L, \quad B = B_H \cdot 2^{\frac{n}{2}} + B_L \tag{1}$$

their product expands to four smaller multiplications plus fixed shifts:

$$A \times B = (A_H B_H)\, 2^n + (A_H B_L)\, 2^{\frac{n}{2}} + (A_L B_H)\, 2^{\frac{n}{2}} + A_L B_L \tag{2}$$

TABLE I
OPCODE CLASSIFICATION BASED ON INST[6:2] IN RISC-V BASE INSTRUCTION FORMAT.

Inst[6:5]	Inst[4:2]							
	000	**001**	**010**	**011**	**100**	**101**	**110**	**111**
00	LOAD	LOAD-FP	*custom-0*	MISC-MEM	OP-IMM	AUIPC	OP-IMM-32	48b
01	STORE	STORE-FP	*custom-1*	AMO	OP	LUI	OP-32	64b
10	MADD	MSUB	NMSUB	NMADD	OP-FP	reserved	*custom-2/rv128*	48b
11	BRANCH	JALR	reserved	JAL	SYSTEM	reserved	*custom-3/rv128*	$\geq$ 80b

Here, $A_H B_H$ produces the most significant bits of the result (shifted by n positions), $A_H B_L$ and $A_L B_H$ produce mid-range partial products (each shifted by $n/2$), and $A_L B_L$ produces the least significant bits without shifting.

By performing these four $\frac{n}{2} \times \frac{n}{2}$ multiplications in parallel and adding their appropriately shifted outputs, we reconstruct the full $2n$-bit product. If $\frac{n}{2} > 2$, the same splitting process applies recursively: each $\frac{n}{2}$-bit multiplication is further broken down until only 2-bit × 2-bit multipliers remain, each producing a 4-bit result directly. This systematic breakdown ensures that for any $n = 4, 8, 16, \ldots$, the same 2-bit multiplier block combined with a uniform shift-and-add stage produces the correct wide-precision product.

Illustration: In Equation 2 $\{2^n, 2^{\frac{n}{2}}, 2^{\frac{n}{2}}, 2^0\}$ shows shifting preferences, while composing 4x4 multipliers for 8x8 multiplication, this shifting preferences will become {8, 4, 4, 0}.

Similarly, product output from 2x2 needs to be shifted by {4, 2, 2, 0}.

Figure 2 shows the abstract method to compose smaller multipliers to build larger ones by appropriate shifting and additions. Figure 3 demonstrates the breakdown of 8x8 into 4x4 and 4x4 into 2x2 with an example.

1) RUnit: Reconfigurable Variable Operand Width Multiplication: We design a reconfigurable MAC unit and integrate it with the execution unit of the RISC-V pipeline (cf. Fig. 4). This unit is capable of handling multiplication for operands of different bit-widths. The multiplier is built to support maximum bit width. In case the operand has a smaller bit precision requirement, our multiplier can handle multiple operations in parallel. Table II shows all possibilities. For example, the case C-MUL88 takes two 8-bit operands and produces "one" output. When used in the 8-bit by 4-bit mode, the same multiplier can perform "two" multiplications of the given precision and generate two outputs. We also provide the facility to add these outputs (when it is used towards a convolution operation). This way, our reconfigurable unit can compute in parallel when performing lower bit-width multiplications. Table II lists the multiplication modes followed by the bit-width of the two operands. The last column mentions the parallelism possible when the operands are smaller than 8-bits.

TABLE II
OPERATIONS THAT CAN BE PERFORMED ON RECONFIGURABLE MULTIPLIER

Case	Width of Operand a	Width of Operand b	Number of parallel Multiplications
C-MUL88	8	8	One
C-MUL84	8	4	Two
C-MUL82	8	2	Two
C-MUL44	4	4	Four
C-MUL42	4	2	Four
C-MUL22	2	2	Eight

B. Opcode map for RISC-V Custom Instructions

In order to use our reconfigurable unit in the RISC-V processor, we designed custom instructions for each case. The programmer/compiler can use these instructions to perform multiplications of a given precision. In this section, we describe our custom RISC-V instructions.

RISC-V ISA is very flexible in adding new instructions; the ISA reserves four opcodes for users to define additional custom instructions in the opcode map, as shown in Table I.

As outlined in the Figure 1 and Table I [22], the bits [6:2] (5 bits) pick one of the 32 "major" opcodes, and then bits [14:12] (3-bits called as funct3), along with bits [31:25] (7 bits called as funct7) allow to split each major opcode into a few more specific operations. Rather than occupying the entire opcode space, RISC-V uses only as many of these 32 slots as it actually needs, and deliberately leaves the rest empty or marked "reserved". While custom-0, 1, 2, and 3 are open to use for custom instructions without fear of dispute, some of those slots are reserved for official future extensions.

In the RISC-V instruction opcode map, some of the opcodes are kept for custom instructions to support extensibility and hardware-specific enhancements. These include custom-0 and custom-1, which occupy the opcode space typically used for load and store-like operations, allowing users to define their own memory-related instructions. Similarly, custom-2/rv128 and custom-3/rv128 are reserved in regions typically used for floating-point or control instructions, and are intended either for advanced custom instruction sets or for future use in the RV128 architecture extension. These custom opcode slots enable designers to integrate domain-specific accelerators, experimental instructions, or specialised hardware functions while maintaining compatibility with the standard RISC-V instruction set.

In this paper, we use the `0x0b` opcode for our proposed custom instructions. In order to use the reconfigurable MAC unit, we have given six custom instructions based on the RISC-V format. In order to get various reconfigurations for the MAC unit, we need to pass the bit width of the operands as part of

TABLE III
THE FORMAT OF THE PROPOSED CUSTOM INSTRUCTIONS FOR THE RECONFIGURABLE MULTIPLIER (USING UNUSED OPCODE 0X0B)

Instruction	**funct7** [31:25]	**rs2(src_addr)** [24:20]	**rs1(src_addr)** [19:15]	**funct3** [14:12]	**rd(dest_addr)** [11:7]	**Opcode** [6:0]
C-MUL88	000 0000	/	/	000	/	000 1011
C-MUL84	000 0001	/	/	000	/	000 1011
C-MUL82	000 0010	/	/	000	/	000 1011
C-MUL44	000 0001	/	/	001	/	000 1011
C-MUL42	000 0010	/	/	001	/	000 1011
C-MUL82	000 0010	/	/	010	/	000 1011
RELU	010 0000	/	/	010	/	000 1011

the instruction. Those bit-widths are encoded as:

- 8-bit → 00
- 4-bit → 01
- 2-bit → 10

We are performing operations on the values stored in the register, and hence we need to use the format of R-Type instruction given in the Figure 1. Table III shows how operand bitwidth information is encoded within the instructions.

- **opcode** (7 bits): available opcode that we choose for our custom instructions.
- **funct3** (3 bits): [1:0] encode the width of operand A.
- **funct7** (7 bits): [1:0] encode the width of operand B.
- **rs1**: (5 bits): Specifies the register holding the first operand/operands (operand A).
- **rs2**: (5 bits) Specifies the register holding the second operand/operands (operand B).
- **rd**: (5 bits) Specifies the destination register where the multiplication result is written.

Below we discuss each instruction from Table III:

- `C-MUL88`: The rs1 and rs2 are used to give the source address of the operands (multiplicands), and the rd is used to indicate the destination address. The operand widths are configured via control fields: funct3 = 000 for operand A and funct7 = 0000000 for operand B, both indicating 8-bit data. This performs a single 8-bit x 8-bit multiplication in a clock cycle.
- `C-MUL84`: The rs1 and rs2 are used to give the source address of the operands (multiplicands), and the rd is used to indicate the destination address. The operand widths are configured via control fields: funct3 = 000 for operand A (8-bit) and funct7 = 0000001 for operand B (4-bit). This performs two parallel 8-bit x 4-bit multiplications in a single clock cycle.
- `C-MUL82`: The rs1 and rs2 are used to give the source address of the operands, and rd indicates the destination. Operand A is 8-bit (funct3 = 000), and operand B is 2-bit (funct7 = 0000010). It executes two 8-bit x 2-bit multiplications simultaneously, enabling high throughput for low-bit precision data.
- `C-MUL44`: This instruction takes its operands from rs1 and rs2 and writes the result to rd. Operand A and B are both 4-bit wide, set by funct3 = 001 and funct7 = 0000001, respectively. It performs four parallel 4-bit x 4-bit multiplications in a single cycle, suited for vectorised low-bit workloads.
- `C-MUL42`: The rs1 and rs2 fields provide 4-bit and 2-bit operands respectively, indicated by funct3 = 001 (A) and funct7 = 0000010 (B). The instruction executes four 4-bit x 2-bit multiplications concurrently, with the packed result written to rd. Useful for compact quantised computing.
- `C-MUL22`: Registers rs1 and rs2 provide eight 2-bit operands each, and rd stores the result. The operand widths are encoded by funct3 = 010 and funct7 = 0000010, indicating 2 bits for both operands. It performs eight 2-bit x 2-bit multiplications in parallel, ideal for ultra-low precision computation.
- `RELU`: The ReLU activation function converts all negative values to 0 and is an identity function otherwise. It requires only one operand as it is a unary activation function. The RELU instruction uses the opcode 0x77 with funct3 set to 011 and the funct7 set to 0.

C. Integration with the processor architecture

The reconfigurable MAC unit, RUnit, is integrated in the 5-stage pipeline inside the EX stage. The custom instructions can be used to operate the unit. We can use normal LOAD instructions to load the input data into registers and STORE instructions to store the output to memory. Fig. 4 shows the abstract block diagram of the processor design along with our proposed unit (shown in yellow colour). This new unit is used when the new custom instructions are executed in the processor pipeline.

D. Convolutions using reconfigurable MAC

Convolution operations in neural networks and other machine learning workloads involve lots of multiplications. We can use our custom instruction to perform these multiplications, and the reconfigurable aspect will help in handling different bit precision. Neural network models working on different bit-widths at different layers can benefit from our design. In particular, more than one smaller bit-width operation can be done in parallel, giving better speedup.

The custom instructions can be used simply in place of the `MULT` instruction according to the programmers' requirements. With custom instructions, we can have less bit precision, which in turn helps our hardware achieve more parallelism.

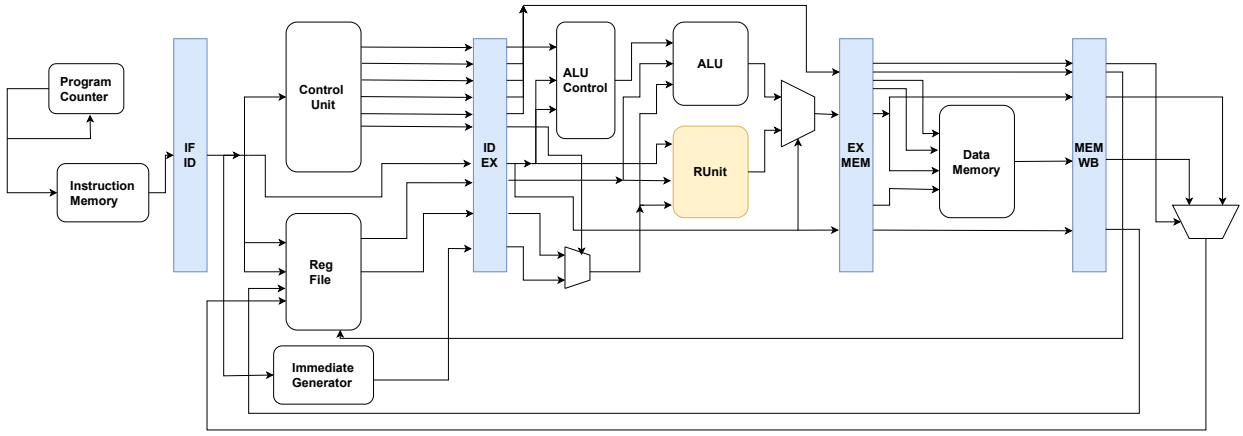

Fig. 4. Block diagram of pipeline processor architecture showing our reconfigurable unit integrated in the 'EX' stage.

TABLE IV
RESOURCE UTILISATION FOR BASELINE RISC-V PROCESSOR RV32I+MUL

Resource	Utilisation	Available	Utilisation %
LUT	5235	134600	3.89
FF	3530	269200	1.31
DSP	3	740	0.41
IO	2	400	0.50

TABLE V
RESOURCE UTILISATION FOR BASELINE RV32I+MUL INTEGRATED WITH RECONFIGURABLE MULTIPLIER UNIT

Resource	Utilisation	Available	Utilisation %
LUT	5649	134600	4.20
FF	3561	269200	1.32
DSP	3	740	0.41
IO	2	400	0.50

IV. EXPERIMENTS AND EVALUATION

We evaluate our designs by implementing them in Verilog. We use the Genus tool from Cadence to generate the area and power. We present our analysis on LeNet [23] and AlexNet [16] networks with an in-house simulator to obtain our results.

A. Hardware Design

The proposed reconfigurable multiplier unit was designed and synthesised using Xilinx Vivado in Verilog HDL. It was also integrated with the design of a complete 5-stage RISC-V pipeline processor architecture. In particular, we implemented RV32I – Base Integer Instruction Set and the 'MUL' instruction[1] The design was tested, and functional verification was carried out.

1) Xilinx synthesis: Fig. 5 shows the RTL schematic. As can be seen, an 8x8 multiplier unit is made up of smaller units. We also provide an adder at the end to add the partial products in cases when we perform smaller bit-width multiplications. We use this feature to add the partial products during the convolution operation.

The resource consumption of the full design is given in Table V. In addition, we also synthesised the normal RISC-V without our custom unit, and present the corresponding synthesis in the Table IV. It can be noted that our proposed RUnit adds $\sim 5\%$ extra area overhead to the baseline design.

2) Xilinx simulation: We also demonstrate the multiplication operation using simulation waveforms. Fig. 6 shows C-MUL88. As can be seen, C-MUL88 performs one multiplication. It take two inputs `0x83` and `0x37` and multiplies them to produce single output:
`0x83 x 0x37` = 6875_{10}.

Fig. 7 shows C-MUL44, which performs four multiplications in one instruction. It takes two 16-bit numbers and internally uses 4 bits each from these. Effectively, it performs the following multiplications in parallel.
We provide two inputs:
`input-1 = 0xe583` and `input-2 = 0xc937`.
Operations are performed on each 4-bit pair as shown below:
`output1 = 0x3 x 0x7` = 21_{10}
`output2 = 0x8 x 0x3` = -24_{10}
`output3 = 0x5 x 0x9` = -35_{10}
`output4 = 0xe x 0xc` = 8_{10}
Addition of all four outputs = `21 -24 -35 + 8 = -30`

At the end, it adds the outputs to save in the destination. We can use this parallelism in neural network inference, having low precision. As can be seen, we get a 4-times speedup due to the reconfigurability of our proposed design.

3) Cadence synthesis: The design was further synthesised using the Genus toolset from Cadence with 90nm Technology.

Table VI reports the synthesis result from Cadence using 90nm technology for our proposed work.

B. Evaluation over Neural Network models

To demonstrate the efficacy of custom RISC-V ISA extensions, we benchmark standard CNN models under varying quantization schemes. We present the results from our own

[1]Note that RISC-V has other extended instructions, but we implemented a subset to demonstrate our proposal. However, this does not limit the integration of RUnit in a full-fledged RISC-V processor.

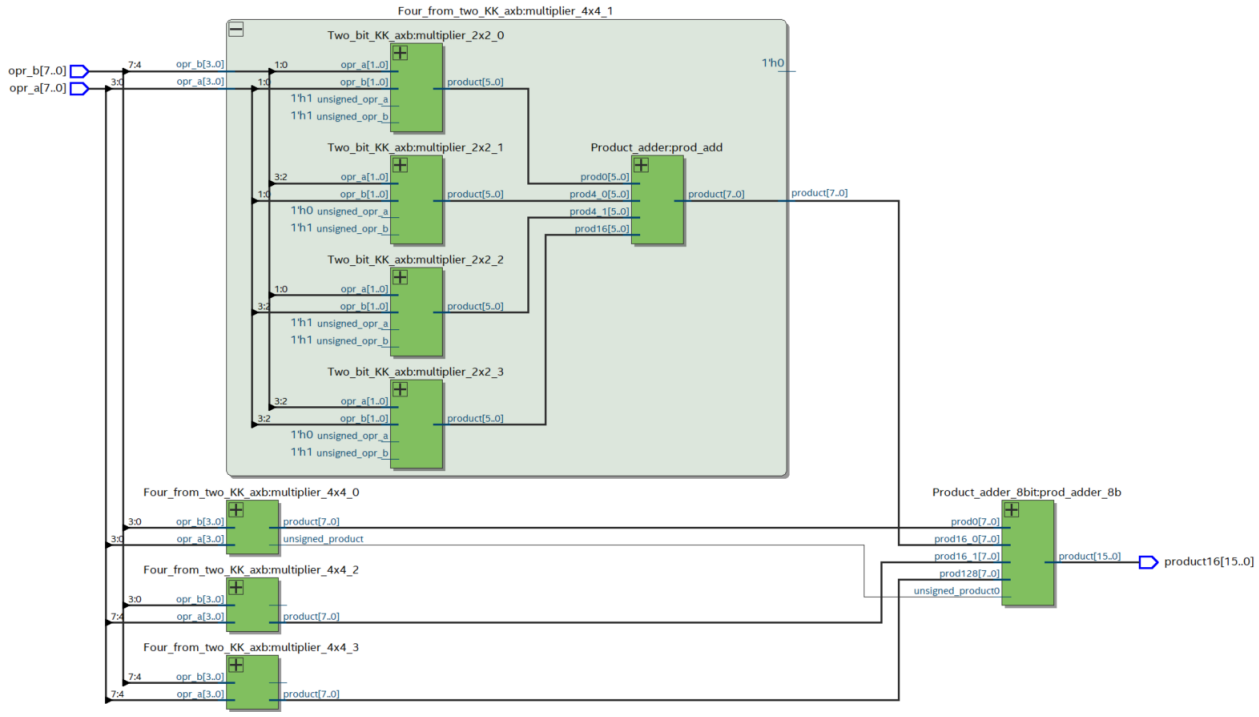

Fig. 5. RTL showing reconfigurable multiplier. Smaller bit multipliers compose to form larger bit-width multipliers.

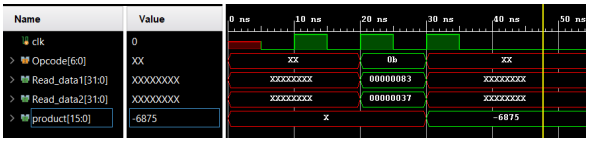

Fig. 6. C-MUL88: Simulation output demonstrating 8x8 multiplication

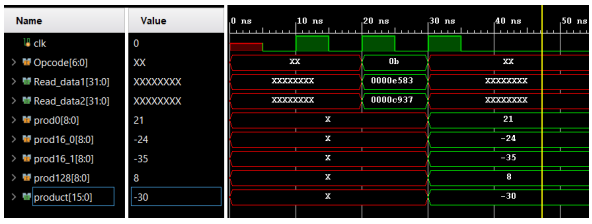

Fig. 7. C-MUL44: Simulation output demonstrating 4x4 multiplication. The two 16-bit input operands result in four 4x4 multiplications in parallel, followed by the addition at the end.

in-house simulator for execution time improvements enabled by low-bit quantization and our custom hardware multiplier. We evaluate on the standard CNNs, LeNet5 [23] and AlexNet [16], where we use the reduced bit width of the multiplication without much accuracy loss [4], [24].

The observed results across LeNet5 and AlexNet suggest a significant reduction in compute time when using 3-bit and 2-bit precision weights, compared to full-precision execution. For instance, in LeNet5, the execution times for 2-bit precision are reduced by a factor of eight across layers, as we can perform eight 2-bit x 2-bit multiplication operations in a single clock cycle. Similarly, in AlexNet, we observe a considerable drop in execution times from 101 million cycles to 12 million cycles for the CONV1 layer and in the CONV2 layer from approximately 449 million to 56 million cycles with 2-bit precision. The reduction in accuracy of the CNNs due to the usage of low-precision weights is all within 5%. The layerwise details for all models are shown in Figure 8 and Figure 9.

TABLE VI
SYNTHESIS REPORT FROM CADENCE

Architecture	Baseline: RV32I+MUL	Baseline+ RUnit
Clock Frequency	526 MHz	526 MHz
Cell Count	14046	16412
Total Area	96523.728	114481.248
Internal Power	9.75473e-03	1.12800e-02
Switching Power	7.65784e-03	9.21033e-03
Leakage Power	1.56988e-04	2.12574e-04
Total power	1.75696e-02	2.07029e-02

From a hardware perspective, the custom multiplier ISA has been designed to support eight parallel multiplications per cycle for 2-bit operands, and 4 per cycle for 3-bit or 4-bit operands. This simple design change allows for improved throughput without significant architectural overhead. The reduction in execution time from the RUnit — particularly for 2-bit quantization, where the observed timing reductions closely approach the ideal 8× factor is possible due to parallel execution of multiple low-precision multiplication operations.

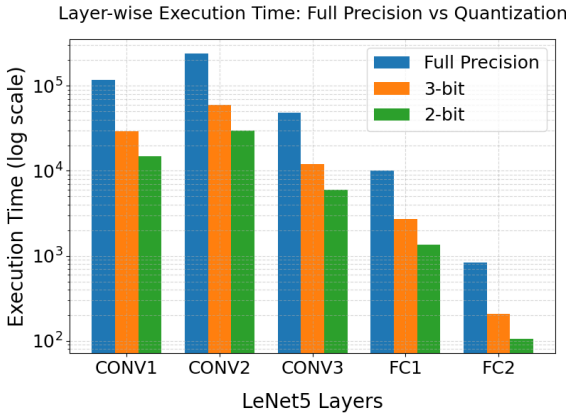

Fig. 8. Relative Execution Times for LeNet5

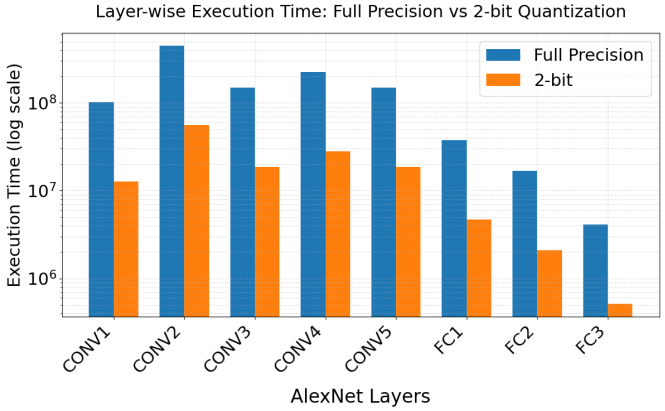

Fig. 9. Relative Execution Times for AlexNet

V. Conclusion

Deep learning applications use multiplication as the primary operation. In order to improve the inference latency, optimising the multiplication operations becomes imperative. Additionally, newer models work on variable bit-precisions. Quantisation also adds to having the operands of bit-width less than 8 bits. Researchers have addressed this by designing the MAC units to handle different precision by either performing bit-wise multiplications or having reconfigurable units.

In this paper, we propose to use a reconfigurable MAC unit, RUnit, which can work on different bit-widths (from 2-bit up to 8-bit inputs). In addition, we also integrated our design with the RISC-V processor pipeline and designed a custom instruction set to activate RUnit. The custom instructions are useful for specifying the operators' bit-width and thus exploiting our design's reconfigurability. We present the synthesis results of the custom instruction integrated with the RISC-V processor. Synthesis results show that RUnit has very little area overhead.

We demonstrate the utility of RUnit towards low-precision neural network workloads. When the bit-width is less than 8 bits, we are able to perform multiple multiplications in the same cycle, thus increasing parallelism and performance.

In this paper, we have experimented with two models with fixed precision, which can be extended to varying precision across layers. The work mainly involved improving inference for CNNs and can be extended to many other operations and architectures. Compiler support for the proposed instructions can be done as future work.

Results from the paper suggest that specialised low-bit support in design and instruction set could offer a practical pathway to speed up deep learning inference on resource-limited platforms.

References

[1] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.

[2] P. Judd, J. Albericio, T. Hetherington, T. Aamodt, N. E. Jerger, R. Urtasun, and A. Moshovos, "Reduced-precision strategies for bounded memory in deep neural nets," 2016. [Online]. Available: https://arxiv.org/abs/1511.05236

[3] B. Noune, P. Jones, D. Justus, D. Masters, and C. Luschi, "8-bit numerical formats for deep neural networks," 2022. [Online]. Available: https://arxiv.org/abs/2206.02915

[4] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: training neural networks with low precision weights and activations," *J. Mach. Learn. Res.*, vol. 18, no. 1, p. 6869–6898, Jan. 2017.

[5] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, V. Chandra, and H. Esmaeilzadeh, "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 2018, pp. 764–775.

[6] Z. Anwar, I. Longchar, and H. K. Kapoor, "Bit-beading: Stringing bit-level mac results for accelerating neural networks," in *2024 37th International Conference on VLSI Design and 2024 23rd International Conference on Embedded Systems (VLSID)*, 2024, pp. 216–221.

[7] I. Longchar and H. K. Kapoor, "Minimac: Design of heterogeneous computing units for matrix multiplication," in *2025 29th International Symposium on VLSI Design and Test (VDAT)*, 2025.

[8] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos, "Stripes: Bit-serial deep neural network computing," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016, pp. 1–12.

[9] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: Efficient inference engine on compressed deep neural network," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 243–254.

[10] M. Shi, V. Jain, A. Joseph, M. Meijer, and M. Verhelst, "Bitwave: Exploiting column-based bit-level sparsity for deep learning acceleration," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2024, pp. 732–746.

[11] E. Forno, A. Spitale, E. Macii, and G. Urgese, "Configuring an embedded neuromorphic coprocessor using a risc-v chip for enabling edge computing applications," in *2021 IEEE 14th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, 2021, pp. 328–332.

[12] T.-T. Hoang, C. Duran, A. Tsukamoto, K. Suzaki, and C.-K. Pham, "Cryptographic accelerators for trusted execution environment in risc-v processors," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020, pp. 1–4.

[13] A. Garofalo, G. Tagliavini, F. Conti, D. Rossi, and L. Benini, "Xpulpnn: Accelerating quantized neural networks on risc-v processors through isa extensions," in *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2020, pp. 186–191.

[14] X. Wang, C. Feng, X. Kang, Q. Wang, Y. Huang, and T. T. Ye, "Rv-scnn: A risc-v processor with customized instruction set for snn and cnn inference acceleration on edge platforms," *IEEE Transactions on*

Computer-Aided Design of Integrated Circuits and Systems, vol. 44, no. 4, pp. 1567–1580, 2025.

[15] L. Z. F. Z. Ning Wu, TaoJiang and F. G, "A reconfigurable convolutional neural network-accelerated coprocessor based on risc-v instruction set," *Electronics*, 2020.

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[17] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015. [Online]. Available: https://arxiv.org/abs/1409.1556

[18] Z. Li, W. Yang, S. Peng, and F. Liu, "A survey of convolutional neural networks: Analysis, applications, and prospects," 2020. [Online]. Available: https://arxiv.org/abs/2004.02806

[19] Y. Zhou, M. Yang, C. Guo, J. Leng, Y. Liang, Q. Chen, M. Guo, and Y. Zhu, "Characterizing and demystifying the implicit convolution algorithm on commercial matrix-multiplication accelerators," 2021. [Online]. Available: https://arxiv.org/abs/2110.03901

[20] A. Lavin and S. Gray, "Fast algorithms for convolutional neural networks," 2015. [Online]. Available: https://arxiv.org/abs/1509.09308

[21] K. Chitsaz, M. Hajabdollahi, N. Karimi, S. Samavi, and S. Shirani, "Acceleration of convolutional neural network using fft-based split convolutions," 2020. [Online]. Available: https://arxiv.org/abs/2003.12621

[22] S. Wang, X. Wang, Z. Xu, B. Chen, C. Feng, Q. Wang, and T. T. Ye, "Optimizing cnn computation using risc-v custom instruction sets for edge platforms," *IEEE Transactions on Computers*, vol. 73, no. 5, pp. 1371–1384, 2024.

[23] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[24] B. Liu, F. Li, X. Wang, B. Zhang, and J. Yan, "Ternary weight networks," in *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023, pp. 1–5.