Variable Bit-Precision Vector Extension for RISC-V Based Processors

Risikesh RK[1], Sharad Sinha[2], Nanditha Rao[1]
[1]International Institute of Information Technology Bangalore, Bangalore, India
[2]Indian Institute of Technology Goa, Goa, India
Email: risikesh.rk@iiitb.ac.in, sharad@iitgoa.ac.in, nanditha.rao@iiitb.ac.in

***Abstract*—Neural Network model execution is becoming an increasingly compute intensive task. With advances in optimisation techniques such as using lower-bit width precision, need for quantization and model compression, we need to find efficient ways of implementing these techniques. Most Instruction Set Architectures(ISA) do not support low bit-width vector instructions. In this work, we present an extension for the vector specification of the RISC-V ISA, which is targeted towards supporting the lower bit-widths or variable precision (1 to 16 bits) Multiply and Accumulate (MAC) operations. We demonstrate our proposed ISA extension by integrating it with a RISC-V processor named PicoRV32, which is considered as the baseline processor in the proposed work. We introduce the feature of bit-serial multiplication along with variable bit precision support to demonstrate the advantage over a 16 bit baseline processor model. We also build an assembler for the proposed instructions for easier integration into the testbench of the RTL model. We implement the processor on to a Xilinx Zynq based FPGA. We observe that, compared to the baseline RISC-V Vector processor which only supports 8, 16 and 32-bit vector instructions, our processor with variable precision support (1 to16 bits) performs 1.14x faster on an average on a matrix multiplication test program. The proposed processor architecture reduces the memory footprint by up to 1.88x as compared with a baseline 16-bit vector processor.**

***Index Terms*—vector architecture, neural networks, variable bit-width precision, FPGA**

I. Introduction

Deep learning models have gained importance in recent years and are being used in almost every application from cloud to edge computing. They are used due to their high accuracy in several applications such as image classification, natural language processing and speech processing. With the ability to execute tasks at such a high accuracy these models are heavily used in mobile applications for processing the information collected by the sensors and devices. In use cases such as audio recognition, image/video processing work loads Deep Neural Networks are used due to their high accuracy compared to traditional Machine learning models, these models typically have several layers (in the order of 10s) and thousands of neurons per layer with weights and biases defined for each neuron, thus making the DNN tasks both compute intensive and memory intensive, due to which the applications are often off-loaded to General Purpose GPU (GPGPU) or accelerators to take advantage of parallel processing capabilities. Further, to reduce the computational complexity and improve the efficiency, deep neural network (DNN) is optimised using techniques such as Network Quantisation, Pruning and Approximations [12] [13] [11]. These techniques help in compressing the DNNs, reduce the computation time and memory footprint.

Quantization techniques represent the weights and activations in the neural network with smaller bit-widths such as 4-bit, 8-bit and so on, while maintaining acceptable accuracy in the inference phase. Inspite of using these techniques, DNNs tend to take up more time to complete the task because of the lack of hardware architectural support inspite of the fact that the networks are quantized. One of the reasons for reduced efficiency is the lack of support for flexible quantization levels (Bit-Precision) and lack of Instruction Set Architecture (ISA) support in modern processors. For example a neural network model can be compressed to run at a 4-bit width or INT4 [14] but the hardware support is not available to run this model at this bit-width. Therefore, these models are generally mapped to the nearest supported precision, such as 8-bits or INT8. Therefore, a lot of time is wasted in extending the quantized data to match the nearest data type. Although some amount of memory space is saved, the benefits in terms of computation time is not achieved. Therefore, there is a need to design hardware or processor architecture such that it caters to the variable bit-precision data that it needs to operate on. The best way to support it is through instructions that support low bit-width.

Authors in [2] propose a set of ISA extensions called the XPulpNN, which are aimed at increasing the energy efficiency of low-bitwidth Quantized Neural Networks (QNNs). They implement low-bitwidth SIMD arithmetic instructions and domain-specific instructions in a RISC-V ISA based micro-controller system. However, their implementation is limited to 2, 4, 8,16,32 bit precisions. A unified neural processing unit (UNPU) is implemented in [1], in which the authors propose an energy-efficient deep neural network (DNN) accelerator. This is an ASIC design which supports variable weight bit precisions of 1-16 bit and is tested on multiply-and-accumulate (MAC) arrays. Authors in [4] present a vectorized bit-serial matrix multiplication overlay with variable-precision support. They argue that integer representations may be good enough for several applications and the precision requirements vary between for different applications can be different. Fewer bits for intermediate layers has been explored in [7] and demonstrate good performance. Further, the method of bit-

serial computations [8] provide a alternate possibility of using binary matrix multiplication to compute matrix multiplications of any precision and is also used in [4].

Overall, a constant-precision implementation could be quite ineffective across different implementations or applications, for different type of inputs. Thus, there is a need to support variable bit-precision architecture in hardware to get the best performance benefit. It would be further useful if it is supported as part of an ISA extension on the processor, instead of supporting it as an accelerator as done in UNPU [1] . In this work, we propose custom ISA extensions to the the RISC-V Vector extension to support fully variable bit-precision from 1 bit to 16 bit. It is aimed at the compressed DNN models inference tasks in which the precision of the weights are known to be variable bit-width. The ISA extensions are proposed for Load/Store and arithmetic instructions with fully variable bit-precision (VBP) support from 1 bit to 16 bit for the vector data.

We integrate the proposed ISA extensions on the register transfer level (RTL) description of a RISC-V processor picoRV32 [9] and combine it with the benefits offered by the bit-serial implementation for the multiplier. We choose the bit serial multiplier in our compute unit due the ease with which the variable bit precision can be integrated. We implement a complete assembler which helps in assembling the vector and RV32I base instructions to be integrated with our testbench. We test the processor and toolchain by implementing a program running matrix multiplication with fully variable bit-width precision. We implement the processor on a Xilinx Zynq based FPGA Zedboard. We observe that the proposed architecture reduces the memory footprint by a factor of 1.88 as tested on the matrix multiplication test-program. The processor gives an overall performance speedup of 1.14 and reduces the computation delay by a factor of 1.25. The proposed processor with the ISA extensions uses 28% LUTs and 2.42% Flipflops of the total resources and runs at 100 MHz. This implementation incurs an overhead of 72% LUTs and 16% Flipflops for supporting Variable Bit Precision arithmetic. Most of the flip-flops or registers are being used up by the packing of the variable bit-precision arithmetic.

The rest of this paper is organized as follows. In section II we discuss the related work in a little more detail with respect to variable/multi bit precision arithmetic support. In section III, we present the proposed architecture for variable bit precision support. In section IV, we present the the tool flow or the execution stack which was built to test the processor. In section V, we present the results and we conclude in Section VI.

II. Related work

The growing use of Machine Learning (ML) or DNN models in wide range of applications has increased the demand to run the DNN/ML models more efficiently. There is, thus the need to improve the model's efficiency from the hardware and software point of view. As discussed earlier, the efficiency of the DNN models can be improved by Quantizing, Pruning and Approximation techniques. These methods compress the models and reduce the memory footprint and computation time. Traditionally the ML/DNN models are run in GP-GPU (General Purpose Graphical Processing Unit) because of its parallelism and easy programmability. But GP-GPUs are inefficient in running these compressed models as efficiently as compared to hardware built to support these compressed models. In this section, we will discuss briefly the previous work which support variable bit precision in their hardware to support machine learning/Deep Neural Network tasks such as Dedicated Accelerators(ASIC), FPGA Based Accelerators and ISA Extensions.

A. Dedicated and FPGA based Accelerators

Designs in [1], [3], [4] are ASIC and FPGA implementations of processors supporting variable bit precision arithmetic through an accelerator. In [1], the authors propose a Unified Neural processing unit, which is a combination of DNN processing core, Aggregation Core and 1-D SIMD processing core, where the data path is managed by a network on chip. In the DNN core which processes the GEMM (General matrix multiplication) where they supports fully variable weight bit precision of 1-16 bit, they use a combination of lookup tables and bit serial multipliers to compute the partial sum which is being aggregated and later on processed by the SIMD cores for the further layers. The bit-serial approach is being widely used in different accelerator implementations. In [3], the authors present an accelerator called Stripes which relies on bit-serial computing and parallelism existing with the neural network to improve performance. They propose a 256 bit broadcasting interconnect connected to the processing tiles and neural memory. In their design, the neural memory is designed to pack the variable bit data efficiently using the memory dispatcher to reduce the memory footprint. The processing tiles are designed to compute the variable precision arithmetic efficiently by using a bit-serial multiplier.

The work on Bit-Serial Matrix Multiplication Overlay for Reconfigurable Computing (BISMO) [4], presents an FPGA Based accelerator. It is a design which is motivated by the fact that matrix multiplication applications can work with reduced precision elements to increase their performance. The processing elements, which mainly comprises of the dot product unit in is designed using the Bit-Serial multiplier.

B. Processor ISA Extensions

Authors of the XpulpNN implementation [2], propose Single instruction multiple data (SIMD) ISA extension to support multi-bit precision arithmetic on a low-power microcontroller called RISCY. The extensions proposed by this work supports nibble(4bit) and crumb(2bit) instructions along with the regular 8 and 16bit arithmetic instructions. The design mainly aims at improving the efficiency of heavily Quantized Neural Networks(QNN) on a microcontroller class processor. They propose 15 new arithmetic instructions to support the computation in nibble and crumb data widths. This is supported

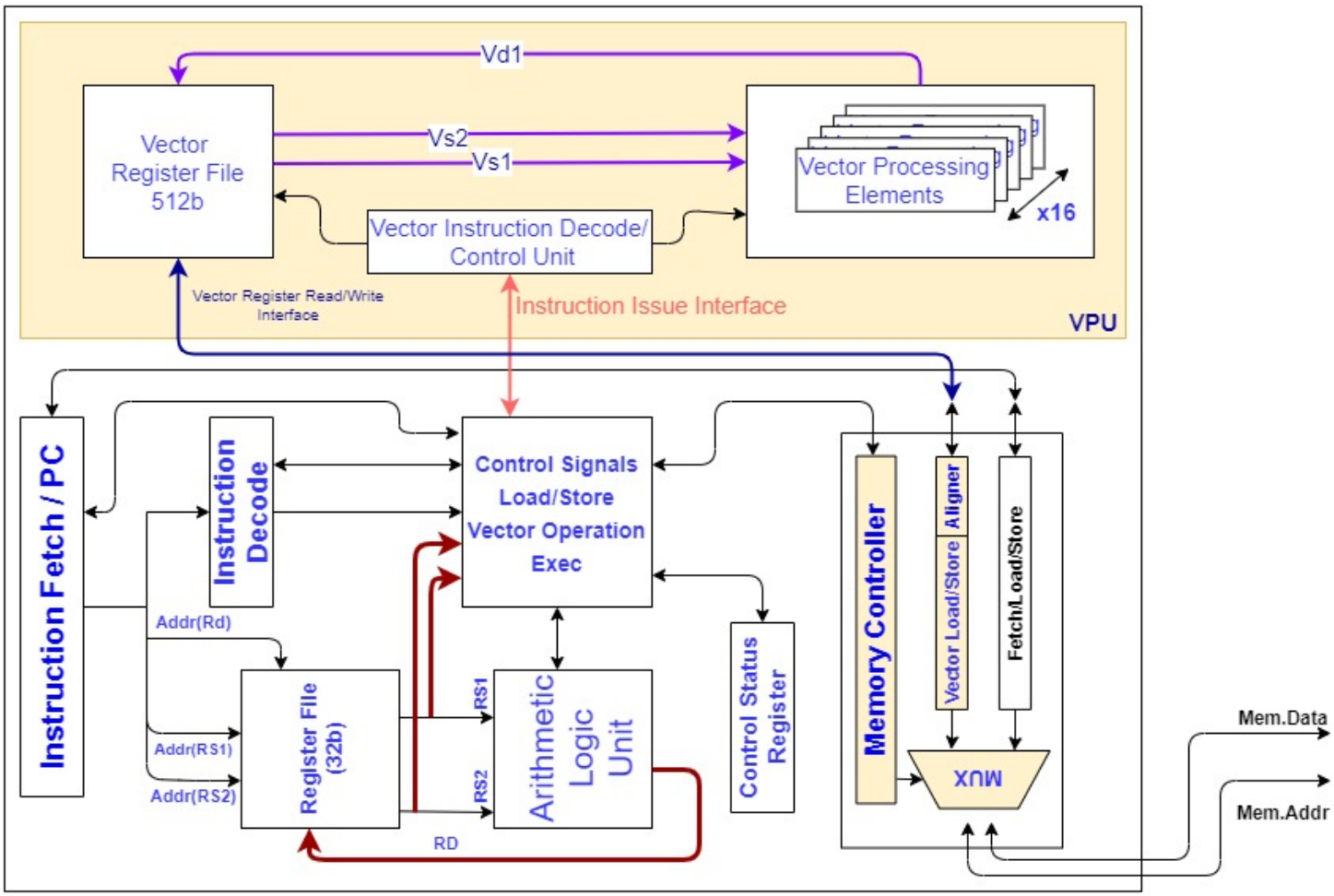

Fig. 1. Microarchitecture

microarchitecturally with the help of a Neural network register file (NN-RF) and a dot product unit.

III. Proposed Architecture and RISC-V ISA Extension

In this section, we discuss the microarchitecture of our proposed processor. We integrated the proposed vector ISA extensions on to an existing baseline processor PicoRV32 [9]. PicoRV32 is a Verilog-based in-order processor core that implements the RISC-V RV32IMC Instruction Set. Integration with an existing core enables us to study the performance and area overhead of the proposed extensions. We now discuss in detail, the following: Proposed ISA, PicoRV32 Microarchitecture, Vector Processing unit, Vector Register File, Vector Processing Element, Variable Bit Width Arithmetic using Bit serial Multiplication, Vector Load Store Unit and the Variable Bit Precision -Instruction Extension (Load/Store, Configuration and Arithmetic instructions)

A. Proposed Instructions

In Table I , we describe the proposed Vector ISA extensions to support variable bit-precision arithmetic. We have created two new control registers: *Variable Arithmetic Precision and Element Offset*, to assist the variable arithmetic precision arithmetic and load/store operations. Variable Arithmetic Precision is a 4 bit register, which stores the arithmetic precision at

Proposed instruction	Description
vset_precision , rs1, rs2	Vector set precision
vmul_varp vd, vs1, vs2	vector multiplication on variable precision
vadd_varp vd, vs1, vs2	vector addition on variable precision
vsub_varp vd, vs1, vs2	vector subtract on variable precision
vleu_varp vd, rs1, vm	vector load element unit-stride on variable precision
vles_varp vd, rs1, rs2, vm	vector load element strided on variable precision
vseu_varp vs3, rs1 vm	vector store element unit stride on variable precision
vses_varp vs3, rs1, rs2, vm	vector store element strided on variable precision

TABLE I
Proposed Instructions

which the variable bit arithmetic processing elements and vector Load/Store will be operating on. Element Offset is a 32 bit register which supports the Vector Load store unit to start loading the Nth element of the sequential data. This is being done since the variable bit data stored in the memory will not be aligned with the byte addressable format of the memory. The Element Offset control register computes the base address of the intended element. In Table I we have implemented Load/Store

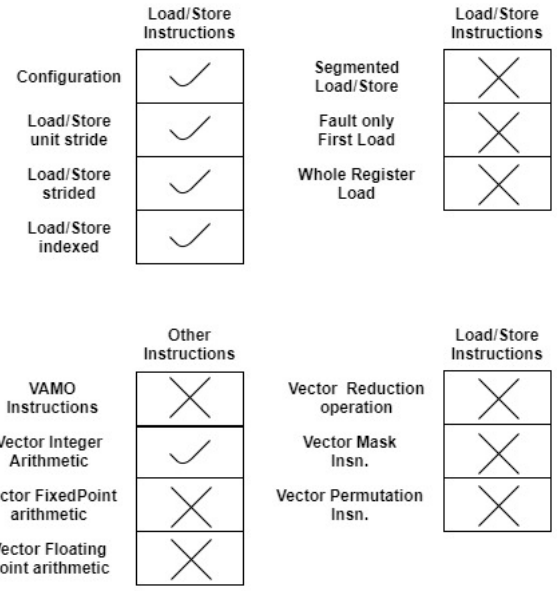

Fig. 2. Implemented Instructions

As mentioned earlier, the ISA extensions proposed in Table I have been implemented and integrated with an RTL model of PicoRV32 [9]. The microarchitecture of the PicoRV32 processor is shown in Figure 1 . The non-highlighted blocks refer to the existing blocks in the baseline processor. We also highlight the new blocks implemented by us, to indicate modifications made to the baseline processor. We highlight the Vector Processing Unit which has a Vector Register File, the Vector Processing Element and Vector Load Store Unit. Since PicoRV32 implements a basic instruction set RV32IMC, the first step was to integrate the required Vector instructions as per the RISC-V Vector Specification Draft [10], with a 512 bit wide vector register configuration. We then implemented custom instructions which support Variable Precision arithmetic in the vector processing unit which are shown in Table I. The set of vector instructions which are implemented in our design is summarised in Figure 2. These include: Vector load/stores, strided load/stores, indexed load/stores and integer arithmetic(vector-vector). The other vector instructions have not been implemented.

B. PicoRV32 Microarchitecture

PicoRV32 is a non-pipelined in-order RV32IMC processor, with dual ported register file configuration. PicoRV32 takes multiple cycles to execute an instruction as shown in the Figure 3. It implements the following seven processor states as a state machine.

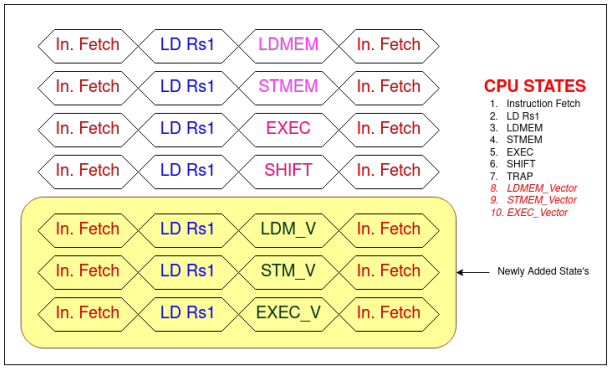

Fig. 3. CPU STATES

1) **Instruction Fetch**: This stage fetches an instruction from the instruction memory and aligns the instructions as per their bit-width in the vector register file.
2) **Decode/Register read, Load Rs1:** This stage performs a register read of two source registers: Rs1 and Rs2 and decodes the fetched instruction.
3) **Execute** This stage executes all the Arithmetic Logic Unit (ALU) operations on the core, along with the Control Status Register (CSR) operations. We modified the execute stage to also include the CSR of the RISC-V Vector ISA [10].
4) **Load from memory:** This stage loads data from data memory to the register file. The data can be of variable bit-widths. The vector register file is 512 bit wide. For example, if the data is 16-bit wide, 32 such data can be packed in the vector register file. Since we support bit-widths varying from 1 bit to 16 bits, packing the data in the vector register is extremely important to save space as well as to increase the compute time in the ALU stage.
5) **Store to memory:** This stage stores data to the data memory from the register file.
6) **Shift:** This stage is taken when the processor encounters shift instructions like SLL,SRR,SRA, etc
7) **Trap:** This stage is taken whenever CPU Trap event occurs.

In addition to these states, we implement the following three processor states to support the proposed ISA extensions.

1) **Vector Load:**This stage forwards the decoded Vector Load instruction to the the Vector Load Store Unit(VLSU) instead of directing it to the non-vector Load/Store unit. The memory access also takes place through a separate memory interface to this unit which is multiplexed with the main memory access interface. The VLSU fetches the data needed from the memory and stores it in the Vector register file through the register write interface of the VPU.

2) **Vector Store:** This stage forwards the decoded Vector Store instruction to the the Vector Load Store Unit(VLSU). The memory access also takes place through a separate memory interface to this unit which is multiplexed with the main memory access interface. VLSU fetches the data needed from the vector register file in the VPU using register read interface in the VPU and stores it in the memory.
3) **Vector Execute:** This stage forwards the Vector arithmetic logic instructions with the required control signals to the Vector ALU or Vector Processing Unit (VPU) to be executed. It also updates the Vector control status registers and General purpose registers.

C. Proposed Vector Processing Unit

The vector processing unit (VPU) is responsible for executing the instruction issued by the processor and updates the register file. The vector processing unit has an instruction issue interface and vector register file read/write interface through which instructions are issued to the VPU from the Vector Exec stage. Data can also be loaded from the Vector Load Store Unit to the vector register file. The vector register file contains 32 vector registers which are 512 bit wide and is dual ported. The vector processing element has 16 lanes or parallel elements and is capable of processing two 32 bit input operands.

1) Vector Processing Element: The Vector Processing element shown in Figure 4 has a combination of 16 parallel processing elements or lanes which support vector operations with variable precision arithmetic from 1 to 16 bits or in other words, INT1-INT16 variable arithmetic.

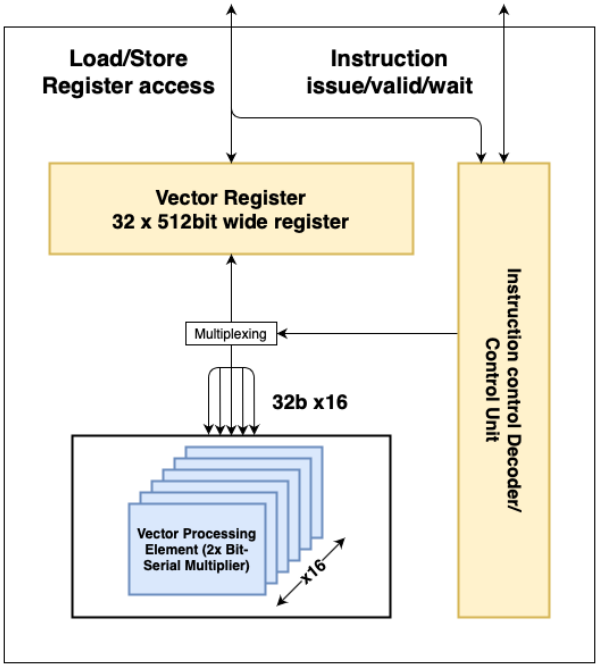

Fig. 4. Vector Processing Unit

D. Proposed Variable Bit Precision Multiplier using Bit Serial Arithmetic

To support variable bit precision multiplication, we implement a Bit Serial Multiplier which consumes lesser resources and also provides a flexibility to operate on variable bit width. The bit serial method is demonstrated in Figure 5. We show two operands A and B where Operand A is 16 bit wide and operand B is N bit wide(Where N can be INT1-INT16). In the context of a neural network, operand 'A' corresponds to the input activation and operand 'B' corresponds to variable precision weights. As shown in Figure 5, operand A is added to the accumulator if the most significant bit (MSB) of operand B is high. In the next cycle we right shift the accumulator and operand B by 1 bit and add operand A to the accumulator based on the MSB of operand B. We repeat this process of shift and ADD N times where N is the bit width of operand B. Overall it takes N cycles for a computation to take place, (that is, till all the N bits of Operand B have reached the MSB) .

We take advantage of the variable number of cycles that it takes to multiply for different bit-widths of weights. Deep Learning algorithms can operate with nearly the same accuracy with lower bit-width weights and hence this proposed structire is expected to save space and provide improved performance.

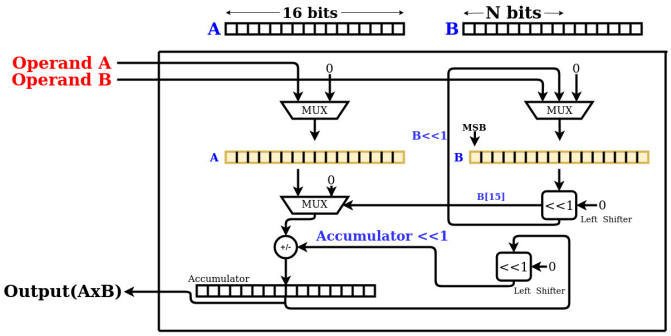

Fig. 5. Bit Serial Multiplier

IV. Assembler for Custom Instructions

We built a basic assembler which reads in the assembly code written in RV32IMC, the RISCV-Vector instructions and the proposed custom instructions. Our assembler parses through the assembly language code and converts the assembly into machine code, using the instruction format database which has the instructions formats and their encoding. The assembly code which is translated to the machine code is loaded into the memory of the processor to be executed. To evaluate the performance of our work, we have used the assembler to translate the assembly code written for a matrix multiplication of dimensions 3x3 , where one of the matrix is set with bitwidth of 16 (representing input activation) and the other matrix (representing weights) with variable bit precision of Int1- Int16. The pseudo-code for 3x3 matrix multiplication with variable bit precision support is shown in Figure 6.

Bit width	Number of CPU CYCLES			Overall Speedup		Memory Space (Bytes)	
	Design 1	Design 2	Design 3	(Design 2)/(Design 1)	(Design 3)/(Design 1)	Design 1	Design 2&3
1	439	580	TBC	1.32	TBC	20	36
2	448	580	TBC	1.29	TBC	21	36
3	457	580	TBC	1.27	TBC	22	36
4	468	580	TBC	1.24	TBC	23	36
5	479	580	TBC	1.21	TBC	24	36
6	486	580	TBC	1.19	TBC	25	36
7	497	580	TBC	1.17	TBC	26	36
8	506	580	TBC	1.15	TBC	27	36
9	517	580	TBC	1.12	TBC	29	36
10	530	580	TBC	1.09	TBC	30	36
11	535	580	TBC	1.08	TBC	31	36
12	544	580	TBC	1.07	TBC	32	36
13	555	580	TBC	1.05	TBC	33	36
14	566	580	TBC	1.02	TBC	34	36
15	575	580	TBC	1.01	TBC	35	36
16	580	580	TBC	1	TBC	36	36

TABLE II
3X3 MATRIX MULTIPLICATION MULTIPLICATION WITH MATRIX A BEING 16BIT AND MATRIX B BEING NBIT

SNO	Configuration	% LUT Used		%FlipFlop Used	
		PICORV32	VPU	PICORV32	VPU
Design 1	PICORV32 + VPU (With Variable Bit Precision arithmetic(Bitserial Multiplier))	28.28		2.42	
Design 2	PICORV32 + VPU (Without Variable Bit Precision arithmetic (Bitserial Multiplier))	16.41		2.07	
Design 3	PICORV32	1.7		1.359	

TABLE III
FPGA SYNTHESIS RESULTS (MAXIMUM CLOCK SPEED 100MHZ)

```
li r1, 0x003 # setting 3 bit precision
li r2, 0x003 # setting 3 as vector length
li r3,0 # offset to load from
li r7 0x800     #variable bitwidth matrix A
li r8 0x900     #matrix B
li r9 0xa00     #matrix C=AxB
vset_precision r1,r0 #where r0 is element offset
vlse_varp v1, (r7),r2
addi r3,r3,0x001
vset_precision r1,r3 #where r3 is element offset
vlse_varp v2, (r7),r2
addi r3,r3,0x001
vset_precision r1,r3 #where r3 is element offset
vlse_varp v3, (r7),r2
vset_precision r1,r0

for( int i =0 ;i<3;i++)
        vsetvli r6,r2,E16
        vsubvv v0,v0,v0
        vlse.v v4, (r8), r0
        addi r8,r8,0x002
        vlse.v v5, (r8), r0
        addi r8,r8,0x002
        vlse.v v6, (r8), r0
        addi r8,r8,0x002
        vmul_varp v7, v4,v1
        vadd_varp v0,v0, v7
        vmul_varp v7, v5,v2
        vadd_varp v0,v0, v7
        vmul_varp v7, v6,v3
        vadd_varp v0,v0, v7
        vse.v v0,(r9)
        addi r9,r9,0x006
```

Fig. 6. Pseudo code for 3x3 matrix multiplication with variable precision support

V. EXPERIMENTS AND RESULTS

We use a 3x3 matrix multiplication test program to validate the proposed vector processor architecture. Matrix A had elements with 16 bit width and matrix B contains elements with variable bit width. We ran the matrix multiplication example on three processor designs:

1) Design 1: PICORV32 processor with Vector processor without variable bit-precision support.
2) Design 2: PICORV32 processor with Vector processor with variable bit-precision support.
3) Design 3: Baseline PICORV32 processor

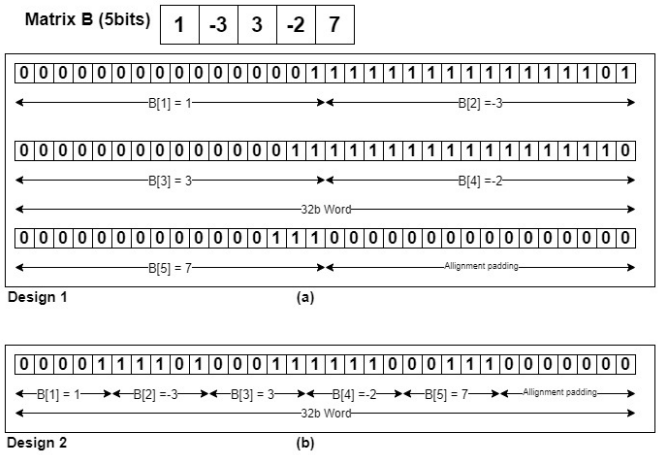

Fig. 7. Memory Usage and Data alignment

Both these designs were compared for speedup, memory footprint and ALU execution time. These three components are tabulated in Table II. We find that there is an average speedup of 1.14x in Design 2 while running the test program. The memory footprint for elements with 1-bit to 16-bit support is certainly lesser compared to the 16-bit supported by Design 1. The data elements with Variable Bit Precision support are stored in a different manner in the memory. For example, consider five data elements each with 5-bit width as shown in Figure 7. In this figure, we compare how a 5 bit-width data is stored in the memory for the 2 designs. The byte addressable memory for Design 1 stores data elements as shown in Figure 7 (a). This design represents the standard RISC-V Vector ISA which only supports a minimum of 16-bit representation for the data. Each location is assumed to store 32 bits of data. Thus, we can store a maximum of two elements per memory location. On the other hand, our proposed architecture which supports a different type pf memory alignment is represented by Design 2 as shown in Figure 7 (b). Since we support a variable bit-width arithmetic, we also modify the memory to store only the required 5 bits. Thus, the five data elements can be stored or "packed" within a single memory location as shown in Figure 7 (b). Therefore, Design 2 also enables reduction in memory footprint.

The reduction in memory footprint further reduces the load store times for a matrix with variable bit precision to be loaded into the vector register file. Since we do the load and store of the elements in the compressed or packed format, the number of clock cycles taken to execute the arithmetic operation also reduces. This time is further reduced with the use of a bit-serial multiplier. This multiplier takes N cycles for an N bit computation to finish. Further, since the VPU supports lanes with variable precision arithmetic, the total ALU execution time also reduces, for lower bit-width arithmetic, as indicated by the last column of Table II.

A. FPGA Synthesis results

The FPGA resource utilization of the designs synthesised for a Xilinx Zedboard using Vivado 2019.1 with CLK constraint of 100MHz is reported in Table III. The maximum available Look Up Tables (LUT) and Flip-Flops (FF) available on Xilinx Zedboard are: 53200 and 106400 respectively.

B. Multi-Core RISC-V Implementation on FPGA

Given that the VPU along with the PicoRV32 needs only LUTs and FFs in the FPGA device on Zedboard (xc7z020clg484) which has a capacity of 53200 LUTs and 106400 FFs, it is possible to implement 3 PicoRV32 with VPU as proposed in this paper. If we take a margin of 20% smaller multi-core RISC-V implementation to compensate for routing resource limitation, we still have 2 number of processors that can be implemented. In this way, the proposed enhanced PicoRV32 is small and lightweight (in terms of resource utilization) enough to form a sufficiently complex multi-core architecture for embedded artificial intelligence.

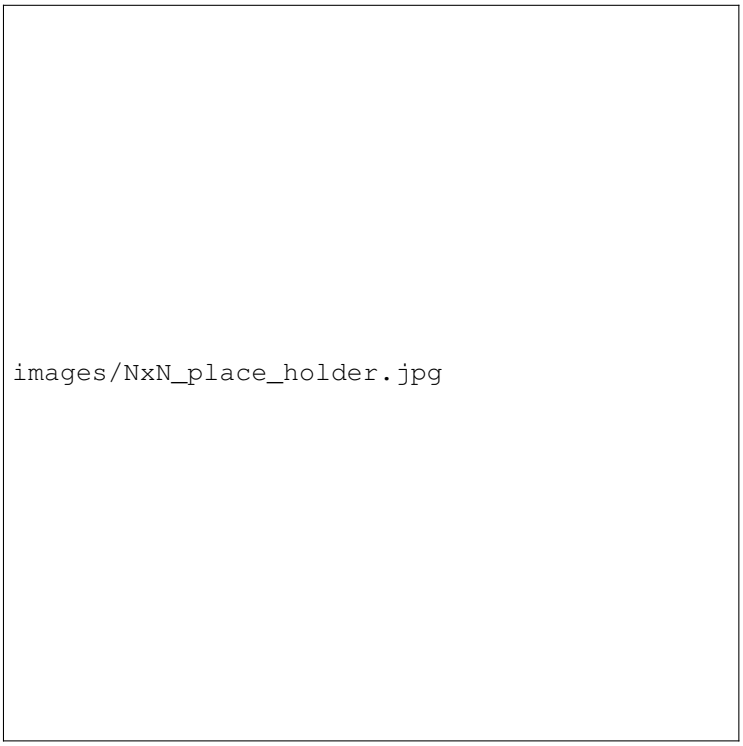

Fig. 8. Speedup of Matrix Multiplication of NxN matrix (Design 1 vs Design 2)

Further evaluation of such a multi-core architecture is left as a future exercise.

VI. Conclusion and future work

The weight precision needed by different Neural Network models can be different. They can continue to provide the same accuracy with reduced bit-precisions. This motivated us to design an ISA extension to a RISC-V vector architecture which can support variable bit precision arithmetic. As a test program, we run the most commonly used matrix multiplication example in which the weights can vary between 1 bit to 16 bits. We observe that our proposed architecture achieved 1.14x performance speedup, 1.88x reduction in memory footprint over a design that did not support variable bit-precision, these are preliminary results, and will be further extended as part of future work.. This work could be further extended by adding more instructions which natively support more DNN tasks like 2D-Convolution and 3D-convolution. We currently support the convolution and other tasks using the integer arithmetic operations and variable precision arithmetic operations. Further, these custom instruction could be added to the RISC-V-GCC compiler stack and the execution of the DNN models could be supported through a software library. The architecture needs to be tested on mode complex models and benchmarks to evaluate the complete efficacy.

References

[1] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim and H. Yoo, "UNPU: An Energy-Efficient Deep Neural Network Accelerator With Fully Variable

Weight Bit Precision," in IEEE Journal of Solid-State Circuits, vol. 54, no. 1, pp. 173-185, Jan. 2019, doi: 10.1109/JSSC.2018.2865489.

[2] A. Garofalo, G. Tagliavini, F. Conti, D. Rossi and L. Benini, "XpulpNN: Accelerating Quantized Neural Networks on RISC-V Processors Through ISA Extensions," 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2020, pp. 186-191, doi: 10.23919/DATE48585.2020.9116529.

[3] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt and A. Moshovos, "Stripes: Bit-serial deep neural network computing," 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2016, pp. 1-12, doi: 10.1109/MICRO.2016.7783722.

[4] Y. Umuroglu, L. Rasnayake and M. Själander, "BISMO: A Scalable Bit-Serial Matrix Multiplication Overlay for Reconfigurable Computing," 2018 28th International Conference on Field Programmable Logic and Applications (FPL), 2018, pp. 307-3077, doi: 10.1109/FPL.2018.00059.

[5] Wang, Kuan, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. "Haq: Hardware-aware automated quantization with mixed precision." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 8612-8620. 2019

[6] Sharma, Hardik, Jongse Park, Naveen Suda, Liangzhen Lai, Benson Chau, Vikas Chandra, and Hadi 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), pp. 764-775. IEEE, 2018.

[7] Park, Eunhyeok, Junwhan Ahn, and Sungjoo Yoo. "Weighted-entropy-based quantization for deep neural networks." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 5456-5464. 2017.

[8] Umuroglu, Yaman, and Magnus Jahre. "Streamlined deployment for quantized neural networks." arXiv preprint arXiv:1709.04060 (2017).

[9] Clifford Wolf. "PicoRV32"., github.com/cliffordwolf/picorv32

[10] RiscV Foundation, "RiscV-V-spec", github.com/riscv/riscv-v-spec/releases/download/0.8/riscv-v-spec-0.8.pdf

[11] Tulloch, Andrew, and Yangqing Jia. "High performance ultra-low-precision convolutions on mobile devices." arXiv preprint arXiv:1712.02427 (2017).

[12] Molchanov, Pavlo, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. "Pruning convolutional neural networks for resource efficient transfer learning." arXiv preprint arXiv:1611.06440 3 (2016).

[13] Wu, Jiaxiang, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. "Quantized convolutional neural networks for mobile devices." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4820-4828. 2016.

[14] Choukroun, Yoni, Eli Kravchik, Fan Yang, and Pavel Kisilev. "Low-bit Quantization of Neural Networks for Efficient Inference." In ICCV Workshops, pp. 3009-3018. 2019.