National Tsing Hua University
Department of Electrical Engineering
AI Accelerator Design for Intelligent Image Processing

Workshop II: Regular ring-based algebraic sparsity

Dec 23, 2021

Objective

In this workshop, you will learn:

1. Ring-based algebraic sparsity and corresponding hardware implementation.
2. Directional ReLU and the on-the-fly processing.

Demo checklist:

- ☐ Show simulation results (Simulation and image output).
- ☐ Compare the performance (timing, area, power…) of the dense convolution design and the ring-sparse convolution design.
- ☐ Questions:
  1. What is the difference between natural sparsity and structural sparsity?
  2. Why is directional ReLU needed for ring-based algebraic sparsity?

Environment Setup

Decompress workshopII.zip in *ICLAB2021_workshop/*. Check the file list in appendix

```
$ unzip workshopII.zip
$ cd workshopII/ring
```

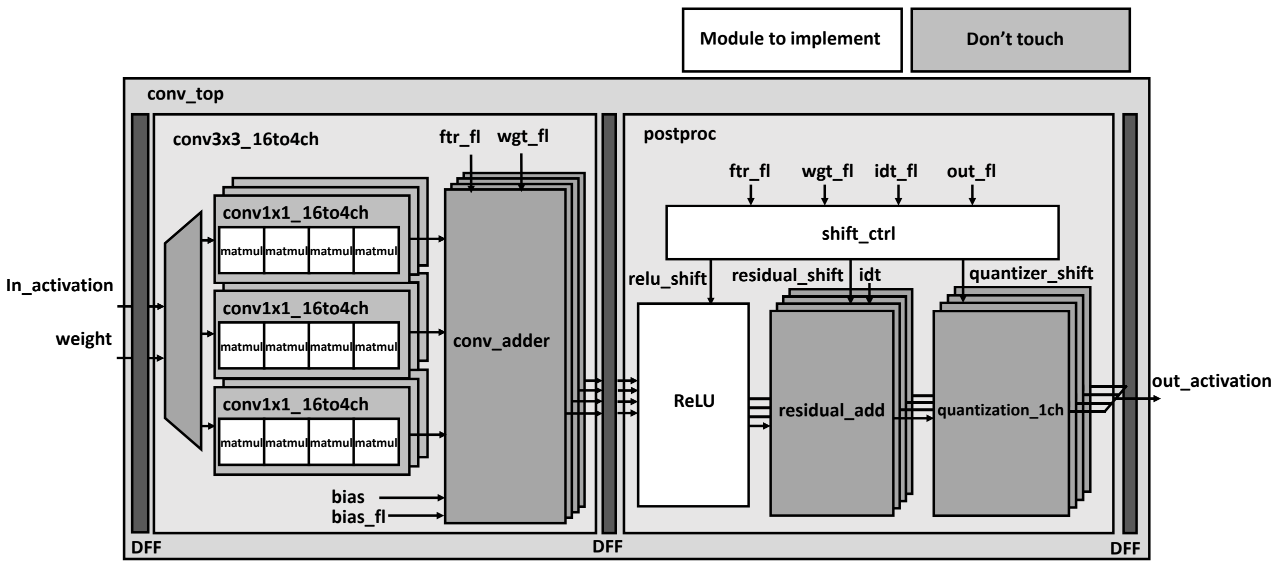


Figure 1. Overview of RingZebraSRNet accelerator.

Table 1. Signal description.

Signal	Description
in_activation	Input feature map vector of a 16×3×3 tensor (channel×h×w).
weight	Weight vector of a 4×16×3×3 tensor ($\text{channel}_{\text{output}}\times\text{channel}_{\text{input}}\times h\times w$).
bias	Bias vector of 4 output channels
idt	Identity branch of the residual connections.
out_activation	Output feature map vector of a 4×1×1 tensor (channel×h×w).
ftr_fl	Fractional length of input feature map.
wgt_fl	Fractional length of weight.
bias_fl	Fractional length of bias.
idt_fl	Fractional length of identity.
out_fl	Fractional length of output feature map.
relu_shift	Shift amount to align directional ReLU (set to 0 for workshop I).
residual_shift	Shift amount to align residual connection.
quantizer_shift	Shift amount for quantization.

Description

In the previous workshop, we implemented an 8-bit dynamic fixed-point ZebraSRNet accelerator. Now, we want to utilize Ring-based structural sparsity for further optimization.

I. Hardware-Efficient Ring

A ***ring*** is a set $\mathbb{R}$ equipped with binary operations + (addition) and · (multiplication). In conventional convolutional neural networks (CNNs), the convolution layer is

$$Y = \text{weight} \cdot X + \text{bias},$$

where the ring addition (+) is component-wise addition and ring multiplication (·) is matrix multiplication. Here we define the ***hardware-efficient ring*** as a set of 4-tuples with ring multiplication (·) being component-wise multiplication. Such ring-based sparse convolution provides up to 75% sparsity, and its great regularity ensures high efficiency for hardware accelerators. However, the lack of information mixing across tuples results in drastic quality degradation. Therefore, we seek for opportunities of exchanging information in ***directional ReLU***.

$$Y = weight \cdot X + bias$$

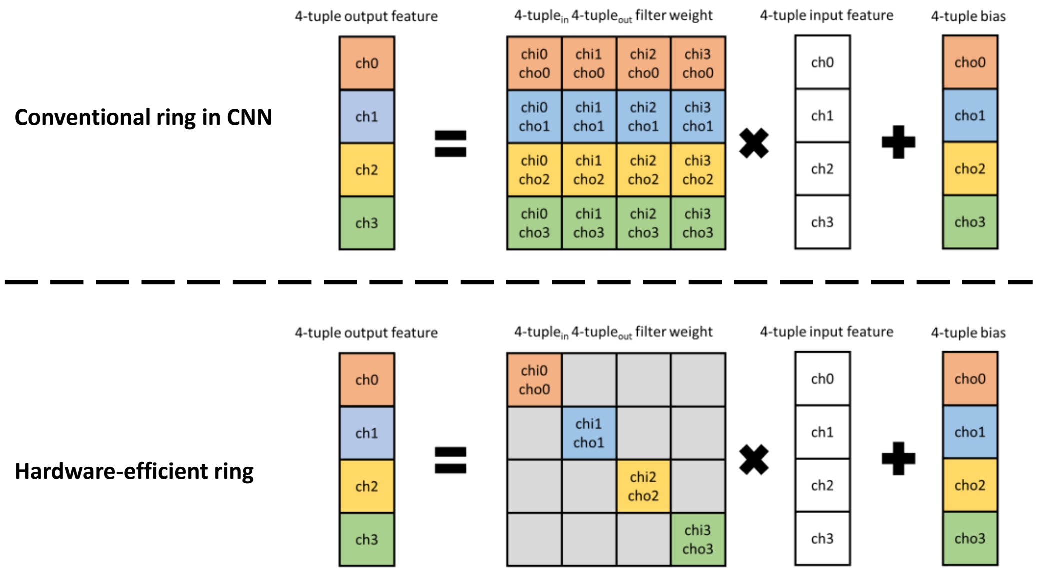


Figure 2. Ring algebra in neural networks.

$$\hat{z} = f_H(z)$$

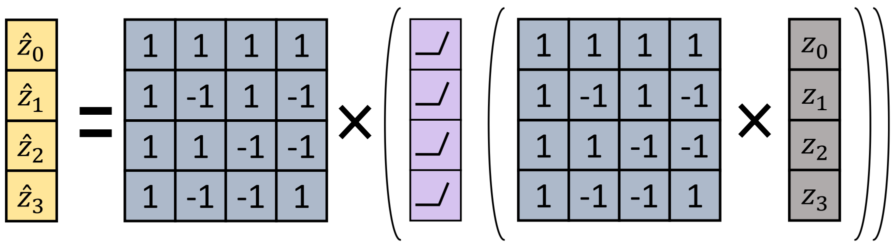


Figure 3. Directional ReLU with 4x4 Hadamard transform.

II. Directional ReLU

In conventional CNNs, the rectified linear unit (ReLU) layer provides non-linearity with component-wise ReLU. To mix information by non-linear layers, we apply Hadamard transform before and after the component-wise ReLU. The new *directional ReLU* is accordingly

$$f_H(y) \triangleq \mathrm{H} f_{cw}(\mathrm{H}y),$$

where H denotes the non-normalized 4x4 Hadamard matrix shown in figure 3.

We trained and fine-tuned an 8-bit RingZebraNet (see figure 4) pretrained model with hardware-efficient Ring and directional ReLU. In this workshop, we are going to implement the hardware accelerator.

Note: Do not implement sequential circuits, or you might disturb the pipeline stages. The control and data movement are taken care by the testbench.

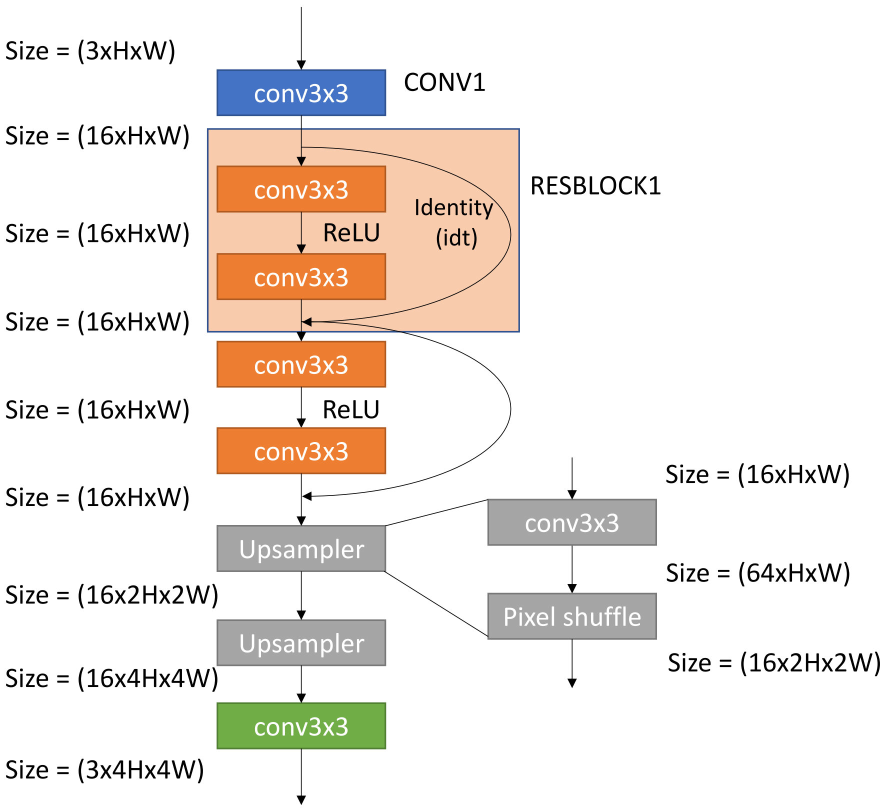


Figure 4. Network architecture of ZebraSRNet.

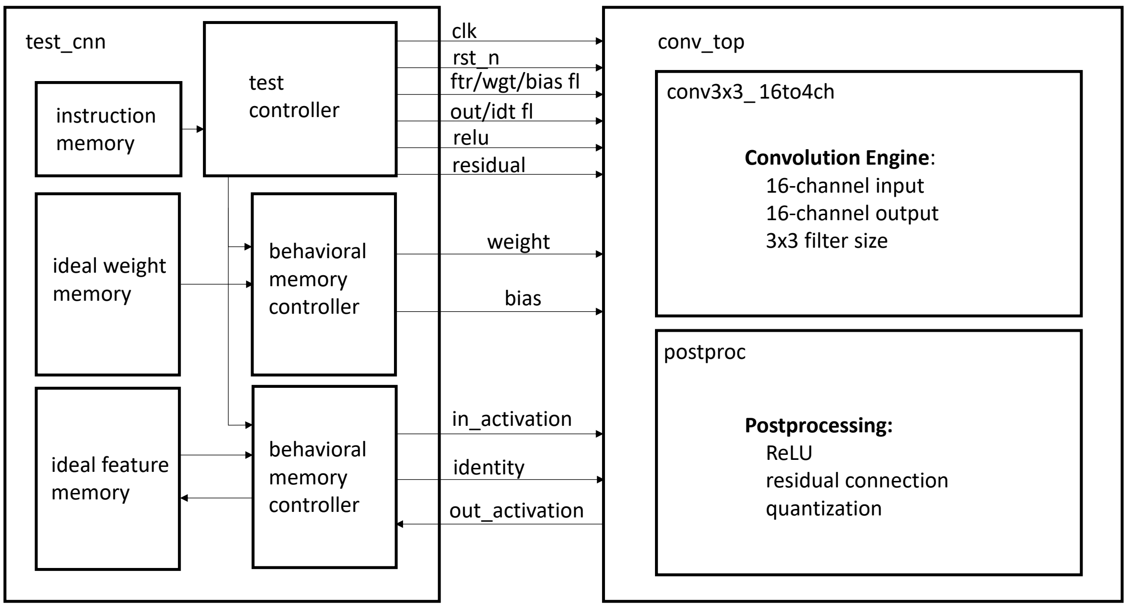


Figure 5. The simulation setup. The testbench include memories with ideal bandwidth and behavioral controllers.

Action Items

I. Ring-based Sparse Convolution Engine.

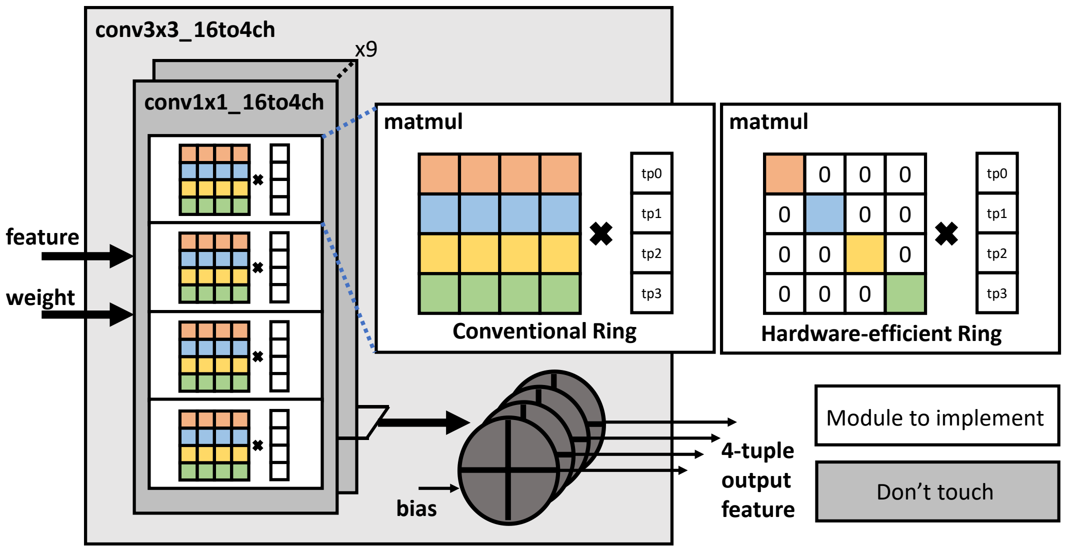


Figure 6. Conv3x3 engine and 4-tuple hardware-efficient ring multiplication.

Refer to workshopI/dyn_8/hdl/*matmul.v* to understand the conventional ring multiplication in the previous workshop. As shown in figure 6, the hardware-efficient ring multiplication is equal to 75 % structural sparse weight. Aside from the computation saving, we don't need to load or store 0s too. Implement the hardware-efficient ring multiplication in *workshopII/ring/hdl/matmul.v*, and verify your design by simulate the computation of first CONV layer. Open *ring/sim/test_run.f*, switch to CONV1 mode, and type the following command:

```
$ cd sim/
$ ncverilog -f test_run.f
```

If you pass the CONV1 simulation, continue to the next part.

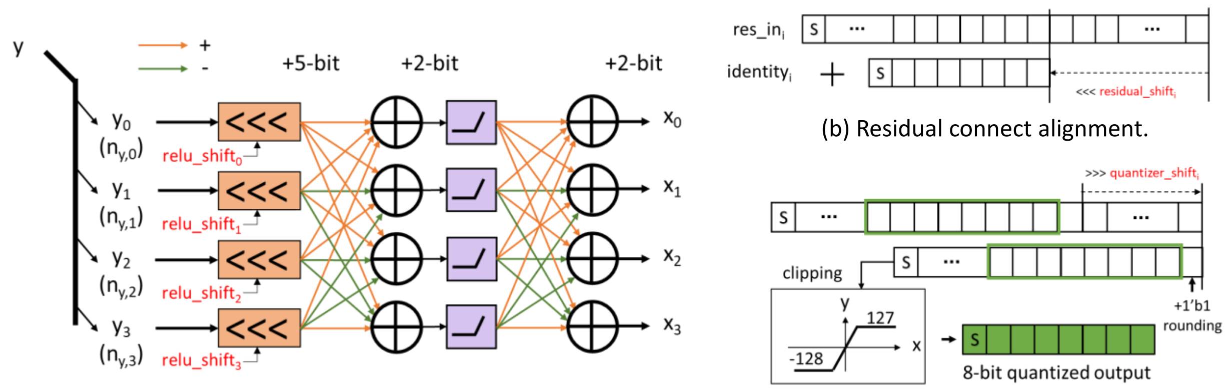
II. Directional ReLU and fractional length alignment

1. Fractional length alignment

Since we assign different fractional length to different tuples, they need to be aligned before Hadamard transform, and the shift amount of other postprocessing operations also need to be adjusted. Calculate the shift amount in figure 7 in *hdl/shift_ctrl.v*.

2. Directional ReLU

Implement on-the-fly directional ReLU in figure 7 (a). Note that non-linearity is not always taken, but we still need to output shifted y_i if (relu==0).



(a) Directional ReLU and tuple alignment. (b) Residual connect alignment. (c) Output quantization.

Figure 7. On-the-fly directional ReLU and postprocessing alignment.

3. Simulation

Switch to RESBLOCK1 and RESBLOCK2 mode in *test_run.f*, then type

```
$ ncverilog -f test_run.f
```

III. Simulation and Synthesis

1. Simulation

If you pass both CONV1 and RESBLOCK1,2 modes in the testbench, switch to IMAGE_OUT mode in *test_run.f*, and run the simulation of whole model (this takes a few minutes). Once the result turns out no error, you can check out the output image and compare with bicubic super resolution result.

```
$ cd result/
```

```
$ python2.7 out2img.py
```

PSNR of ring sparsity model output:________dB,

PSNR of dense model output:________dB (from previous workshop).

Note: You can modify *sim/test_run.f* to simulate with hidden images. Note that the toy model has limited parameter size and dataset, and so is its output image quality.

2. Synthesis

Synthesize the design with following command (this takes about 10 minutes), and compare to the dense ZebraNet accelerator in previous workshop.

```
$ cd ../../syn/

$ sh run_dc.sh
```

Area of ring sparse CNN accelerator:______________,

Area of dense CNN accelerator:______________

Appendix

Here is the file structure of the lab package. Please check if anything is missing!

Directory	**Filename**	**Description**
golden	*.png	Test image and bicubic output
workshopII/hdl	conv_define.v	parameter define
workshopII/hdl	conv_top.v	Top module
workshopII/hdl	conv3x3_16to4ch.v	CONV3x3 engine
workshopII/hdl	conv1x1_16to4ch.v	1x1 convolution
workshopII/hdl	matmul.v	Matix multiplication
workshopII/hdl	conv_adder.v	Sum 9 pixel and bias
workshopII/hdl	postproc.v	postprocessing
workshopII/hdl	shift_ctrl.v	Shift amount controller
workshopII/hdl	ReLU.v	ReLU/directional ReLU
workshopII/hdl	residual_add.v	Residual connection
workshopII/hdl	quantization_1ch.v	Output quantization
workshopII/sim	data/	Test pattern
workshopII/sim	result/out2img.py	Input image pattern
workshopII/sim	run.sh	Shell script
workshopII/sim	test_run.f	Run for testbench
workshopII/sim	test_cnn.v	testbench
workshopII/sim	task.v	Task definition
workshopII/syn	*.tcl, .synopsys_dc_setup	Synthesis scripts