

CNN accelerator IP (Reference S/W, Buffers)

Xuan-Truong Nguyen
2023.1.18 (Wed)



Road map

Review

Reference Software

weight/bias/scale
buffers

Output buffers

Control path
(CONV3x3)

Recap: mapping convol2.m

- Kernel operations
 - Element-wise multiplication (products)
 - 16 multipliers
 - Sum (sum of products)
 - Adder tree (mac.v)
 - Accumulation (mac_kern.v)

```
% Convolution
for k = 1:c_out % Number of output channels
    win = kernel(:,:,:,:,k); % Weight
    for i = 1:h_out % Row
        for j = 1:w_out % Column
            % Input feature map
            din = pad img(1+(i-1)*s:1+(i-1)*s+f-1, 1+(j-1)*s:1+(j-1)*s+f-1, :);
            % Element-wise Multiplication => Products
            scalar = win .* din;
            % Sum (Sum of Products): Adder tree
            out(i,j,k) = sum(scalar(:));
        end
    end
end
```

Instance	Design unit
conv_kern_tb	conv_kern_tb
u_conv_kern	conv_kern
u_mac_kern	mac_kern
u_mac	mac
u_mul_00	mul
u_mul_01	mul
u_mul_02	mul
u_mul_03	mul
u_mul_04	mul
u_mul_05	mul
u_mul_06	mul
u_mul_07	mul
u_mul_08	mul
u_mul_09	mul
u_mul_10	mul
u_mul_11	mul
u_mul_12	mul
u_mul_13	mul
u_mul_14	mul
u_mul_15	mul
#ALWAYS#29	mac
#ALWAYS#63	mac
#ALWAYS#105	mac
#ASSIGN#115	mac
#ASSIGN#116	mac
#ALWAYS#52	mac_kern
#ALWAYS#74	mac_kern
#ASSIGN#85	mac_kern
#ASSIGN#86	mac_kern
u_bnorm_quant_act	bnorm_quant_act

Recap: mapping convol2.m

- Sliding window: Loop and data generation
 - extract a tensor of an input map (din) to do convolution
 - To compute $\text{out}(i, j, k)$, in this case, a window of pixels is defined by
 - Row: $i-1$ to $i+1$
 - Column: $j-1$ to $j+1$
 - All input channels:

```
% Convolution
for k = 1:c_out % Number of output channels
    win = kernel(:,:,:,:,k); % Weight
    for i = 1:h_out % Row
        for j = 1:w_out % Column
            % Input feature map
            din = pad img(1+(i-1)*s:1+(i-1)*s+f-1, 1+(j-1)*s:1+(j-1)*s+f-1, :);
            % Element-wise Multiplication => Products
            scalar = win .* din;
            % Sum (Sum of Products): Adder tree
            out(i,j,k) = sum(scalar(:));
        end
    end
end
```

Copyright

Loop generation: Finite State Machine

- Update the current state (cstate) by the next state (nstate): Sequential logic
- Decide the next state based on the current state and other conditions: Combinational logic
- Pseudocode for FSM or loop generation

ST_VSYNC: frame synchronization,

E.g., preload filters or line data

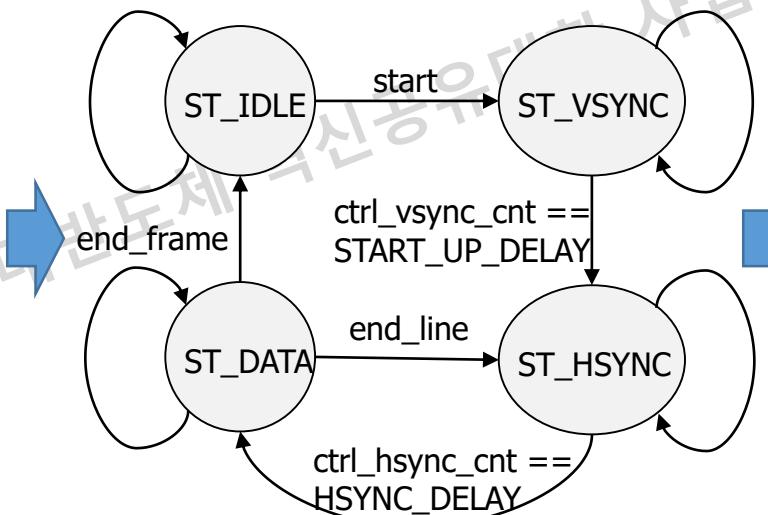
for row = 0 to height-1

ST_HSYNC: row synchronization, e.g.,

E.g., preload filters or line data

For col = 0 to width-1

ST_DATA: do calculation



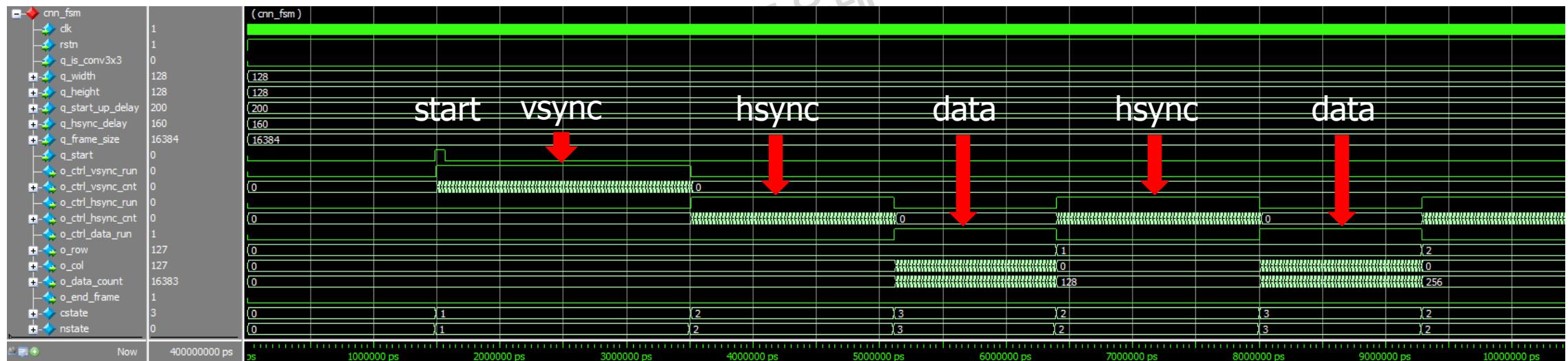
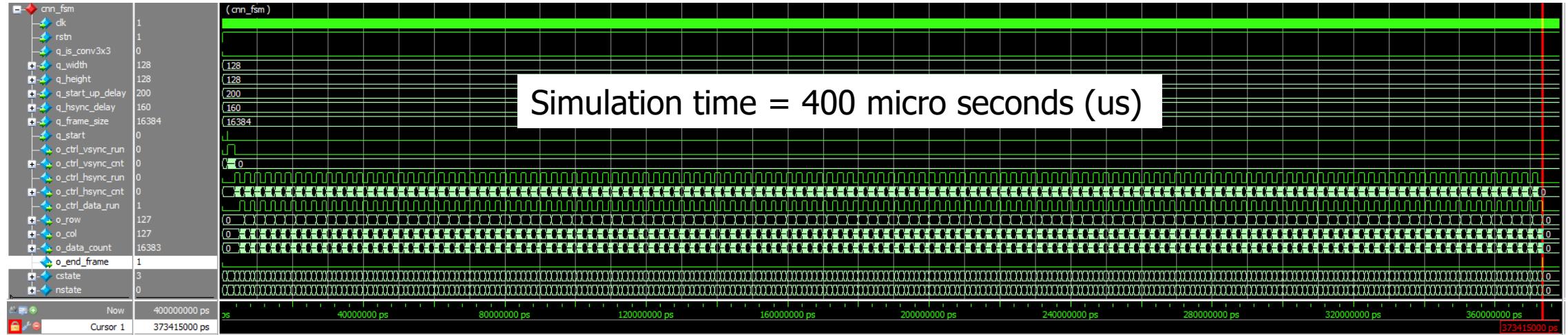
```
always @ (posedge clk, negedge rstn)
begin
    if(~rstn) begin
        cstate <= ST_IDLE;
    end
    else begin
        cstate <= nstate;
    end
end
```

```
always @ (*) begin
    case(cstate)
        ST_IDLE: begin
            if(start)
                nstate = ST_VSYNC;
            else
                nstate = ST_IDLE;
        end
        ...
        default: nstate = ST_IDLE;
    endcase
end
```

Update the current state (cstate) by the next state (nstate)

Decide the next state based on the current state and other conditions.

Waveform: Finite state machines



Recap: Data generation

- Assume that filters (e.g., win, scale, bias) are preloaded to registers.
- Using row and col indexes to generate din
 - Zero adding: assign zero for pixels outside of the image.

```
if (row == 0) begin
    if(col == 0) begin
        @ (posedge clk)      vld_i = 1'b1;
        /* Insert your code*/
    end
    else if (col == WIDTH-1) begin
        @ (posedge clk)      vld_i = 1'b1;
        /* Insert your code*/
    end
    else begin
        @ (posedge clk)      vld_i = 1'b1;
        din[0*WI+:WI] = 8'd0/*in_img[(row-1) * WIDTH + col-1]*/;
        din[1*WI+:WI] = 8'd0/*in_img[(row-1) * WIDTH + col ]*/;
        din[2*WI+:WI] = 8'd0/*in_img[(row-1) * WIDTH + col+1]*/;
        din[3*WI+:WI] = in_img[(row ) * WIDTH + col-1];
        din[4*WI+:WI] = in_img[(row ) * WIDTH + col ];
        din[5*WI+:WI] = in_img[(row ) * WIDTH + col+1];
        din[6*WI+:WI] = in_img[(row+1) * WIDTH + col-1];
        din[7*WI+:WI] = in_img[(row+1) * WIDTH + col ];
        din[8*WI+:WI] = in_img[(row+1) * WIDTH + col+1];
    end
end
end
```

Lab 1: first_layer_tb.v



```
// Boundary-checking flags
assign is_first_row = (row==0)?1'bl:1'b0;
assign is_last_row = (row==q_height-1)?1'bl:1'b0;
assign is_first_col = (col==0)?1'bl:1'b0;
assign is_last_col = (col==q_width-1)?1'bl:1'b0;
// Generate din
always@(*) begin
    vld_i = 0;
    din   = 0;
    // First layer
    if(q_is_first_layer) begin
        vld_i = ctrl_data_run;
        din[0*WI+:WI] = (is_first_row | is_first_col) ? 8'd0 : in_img[(row-1) * q_width+ (col-1)];
        din[1*WI+:WI] = (is_first_row           ) ? 8'd0 : in_img[(row-1) * q_width+ col ];
        din[2*WI+:WI] = (is_first_row | is_last_col) ? 8'd0 : in_img[(row-1) * q_width+ (col+1)];
        din[3*WI+:WI] = (           is_first_col) ? 8'd0 : in_img[ row * q_width+ (col-1)];
        din[4*WI+:WI] = (           is_first_col) ? 8'd0 : in_img[ row * q_width+ col ];
        din[5*WI+:WI] = (           is_last_col) ? 8'd0 : in_img[ row * q_width+ (col+1)];
        din[6*WI+:WI] = (is_last_row | is_first_col) ? 8'd0 : in_img[(row+1) * q_width+ (col-1)];
        din[7*WI+:WI] = (is_last_row           ) ? 8'd0 : in_img[(row+1) * q_width+ col ];
        din[8*WI+:WI] = (is_last_row | is_last_col) ? 8'd0 : in_img[(row+1) * q_width+ (col+1)];
    end
end
```

Boundary

Pixel indexing

Lab 3: cnn_accel.v

Recap: Data generation

- Assume that filters (e.g., win, scale, bias) are preloaded to registers.
- Using row and col indexes to generate din: $\text{data_count} = \text{row} * \text{q_width} + \text{col}$
 - Zero adding: assign zero for pixels outside of the image.

```
cnn_fsm u_cnn_fsm (
    .clk(clk),
    .rstn(rstn),
    // Inputs
    .q_is_conv3x3(q_is_conv3x3),
    .q_width(q_width),
    .q_height(q_height),
    .q_start_up_delay(q_start_up_delay),
    .q_hsync_delay(q_hsync_delay),
    .q_frame_size(q_frame_size),
    .q_start(q_start),
    //output
    .o_ctrl_vsync_run(ctrl_vsync_run),
    .o_ctrl_vsync_cnt(ctrl_vsync_cnt),
    .o_ctrl_hsync_run(ctrl_hsync_run),
    .o_ctrl_hsync_cnt(ctrl_hsync_cnt),
    .o_ctrl_data_run(ctrl_data_run),
    .o_row(row),
    .o_col(col),
    .o_data_count(data_count),
    .o_end_frame(layer_done)
);

    // Boundary-checking flags
    assign is_first_row = (row==0)?1'b1:1'b0;
    assign is_last_row = (row==q_height-1)?1'b1:1'b0;
    assign is_first_col = (col==0)?1'b1:1'b0;
    assign is_last_col = (col==q_width-1)?1'b1:1'b0;
    // Generate din
    always@(*) begin
        vld_i = 0;
        din = 0;
        // First layer
        if(q_is_first_layer) begin
            vld_i = ctrl_data_run;
            din[0*WI+:WI] = (is_first_row | is_first_col) ? 8'd0 : in_img[data_count - q_width - 1];
            din[1*WI+:WI] = (is_first_row ) ? 8'd0 : in_img[data_count - q_width ];
            din[2*WI+:WI] = (is_first_row | is_last_col ) ? 8'd0 : in_img[data_count - q_width + 1];
            din[3*WI+:WI] = ( is_first_col) ? 8'd0 : in_img[data_count - 1];
            din[4*WI+:WI] =
                in_img[data_count ];
            din[5*WI+:WI] = ( is_last_col ) ? 8'd0 : in_img[data_count + 1];
            din[6*WI+:WI] = (is_last_row | is_first_col) ? 8'd0 : in_img[data_count + q_width - 1];
            din[7*WI+:WI] = (is_last_row ) ? 8'd0 : in_img[data_count + q_width ];
            din[8*WI+:WI] = (is_last_row | is_last_col ) ? 8'd0 : in_img[data_count + q_width + 1];
        end
    end
)
```

Lecture plan

- Today we will answer the following questions:
 - Why/when/how to prepare the data, e.g., weight/scale/bias?
 - How to handle the output results?
 - Does FSM work correctly to handle CONV3x3?
- Via three labs:
 - Lab 1: Weight/bias/scale buffers
 - Lab 2: Output buffers
 - Lab 3: Controller for conv3x3

Road map

Review

Reference Software

weight/bias/scale
buffers

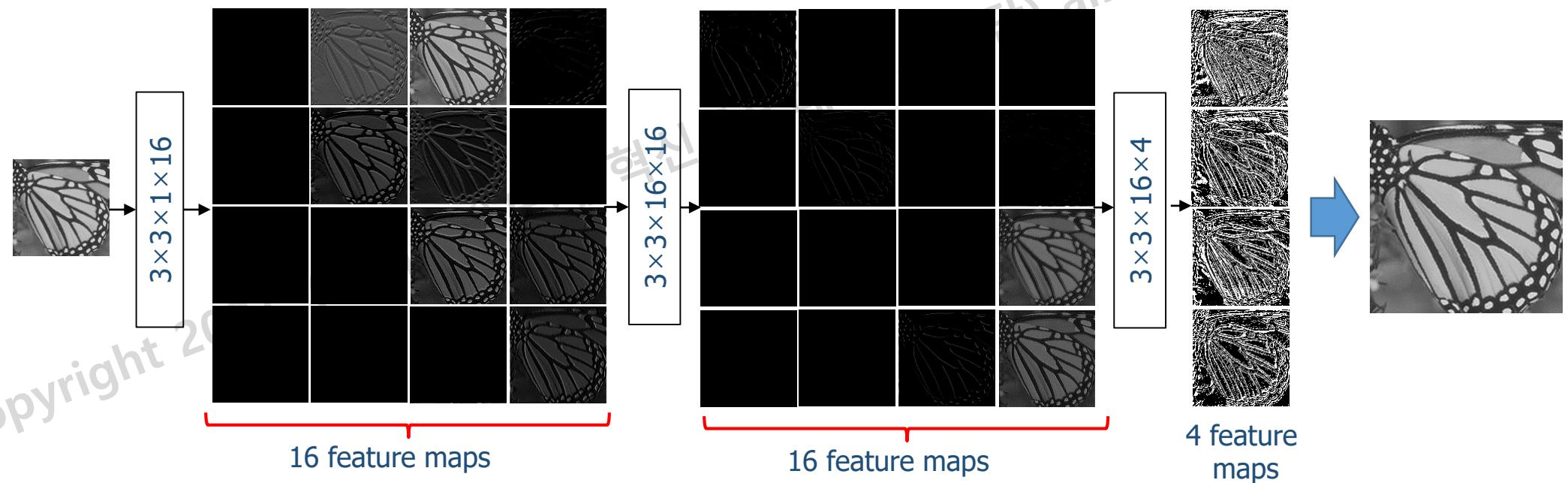
Output buffers

Control path
(CONV3x3)

Sim-ESPCN

- Sim-ESPCN has three CONV layers

Layer	Filter size	No. of input channels	No. of output channels	Input feature maps	Output feature maps	NP
1	3x3	1	16	128x128x1	128x128x16	3x3x1
2	3x3	16	16	128x128x16	128x128x16	3x3x16
3	3x3	16	4	128x128x16	128x128x4	3x3x16



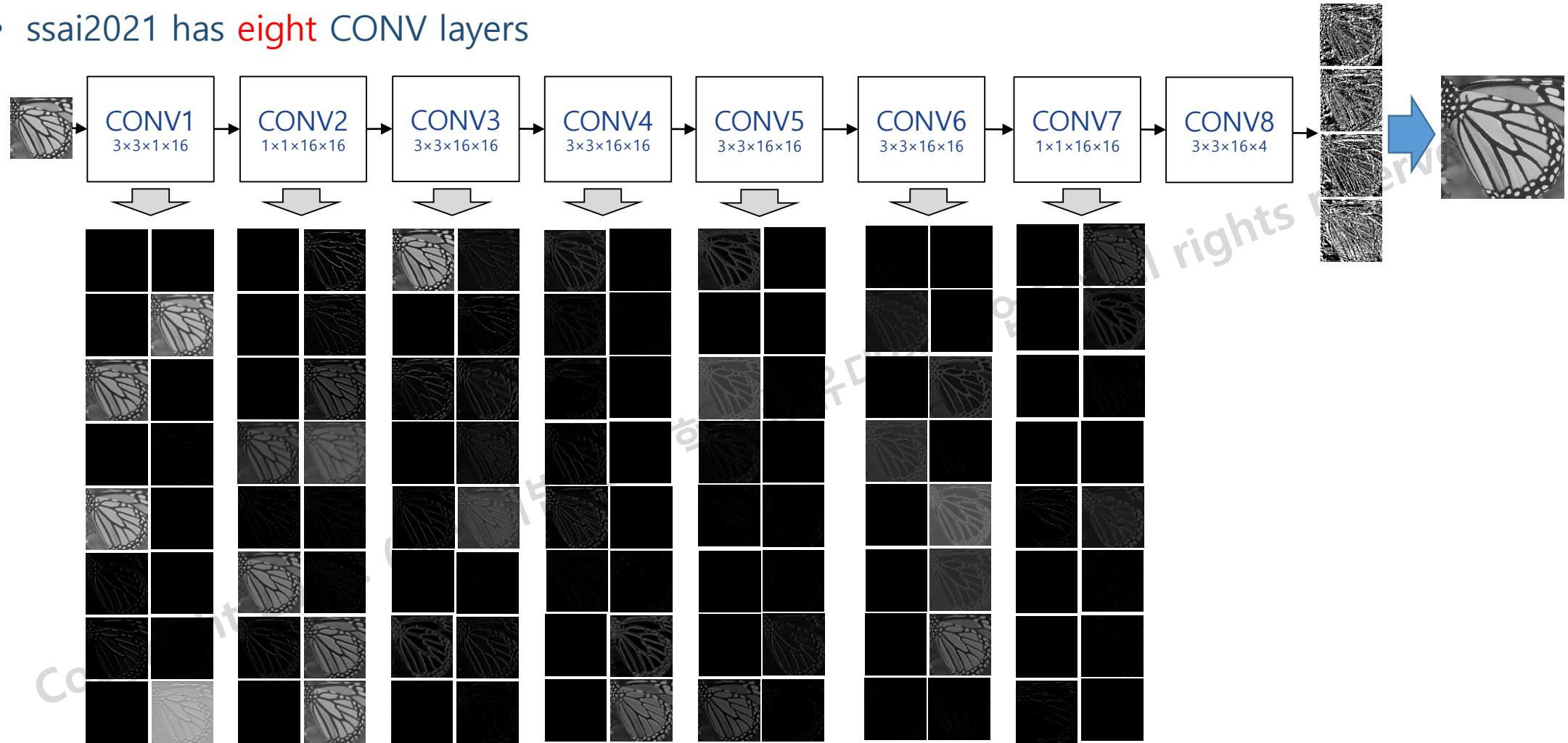
The final project

- In the final project, the CNN model (ssai2021) has **eight** CONV layers
- The given baseline code
 - Reference Software: Quantized model, all calculation steps.
 - Baseline H/W code
 - mul.v, mac.v, mac_kern, bnorm_act_quant.v, conv_kern.v
 - cnn_accel.v, block ram

Layer	Filter size	No. of input channels	No. of output channels	Input feature maps	Output feature maps
1	3×3	1	16	$128 \times 128 \times 1$	$128 \times 128 \times 16$
2	1×1	16	16	$128 \times 128 \times 16$	$128 \times 128 \times 16$
3	3×3	16	16	$128 \times 128 \times 16$	$128 \times 128 \times 16$
4	3×3	16	16	$128 \times 128 \times 16$	$128 \times 128 \times 16$
5	3×3	16	16	$128 \times 128 \times 16$	$128 \times 128 \times 16$
6	3×3	16	16	$128 \times 128 \times 16$	$128 \times 128 \times 16$
7	1×1	16	16	$128 \times 128 \times 16$	$128 \times 128 \times 16$
8	3×3	16	4	$128 \times 128 \times 16$	$128 \times 128 \times 4$

The final project: SSAI2021

- ssai2021 has **eight** CONV layers



The final project

- Due date: **January 31st (Tue).**
- Each team has **ONLY one person.**
- What you have to do:
 - Successfully run the network on the NPU (100p)
 - Prepare the input image
 - Prepare the quantized model
 - Do inference with the input image and the model on the NPU
 - Give the correct output feature maps.
 - Optimization (90p)
 - Time: Reduce the number of cycles for inference
 - H/W cost: reduce the on-chip buffer
 - Ranking: A: 90p, B: 60p, C: 30p
 - Analysis and optimality (10p)
 - Analyze your design and check its optimality
 - i.e., MAC utilization

Comparison

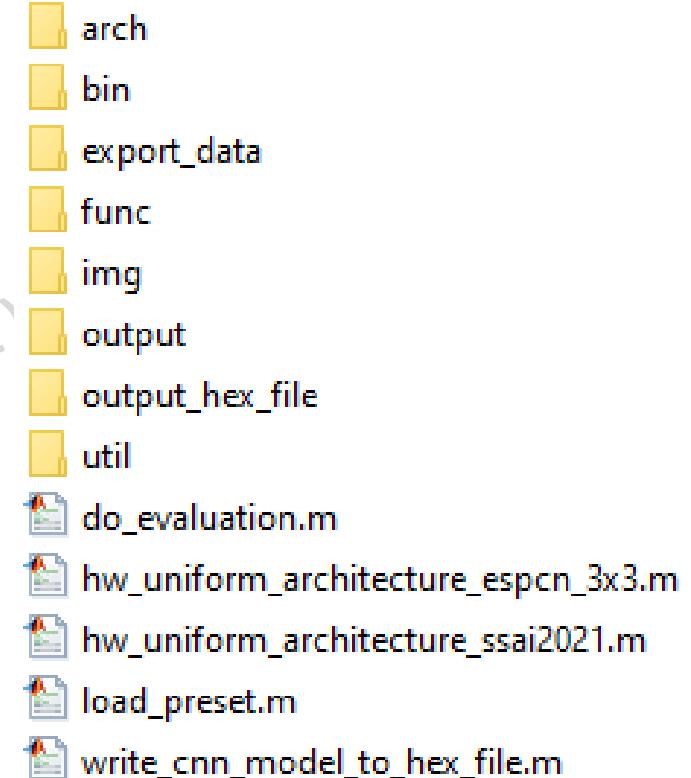
- Two metrics
 - Reconstruction quality: Peak-signal to noise ratio (PSNR) in dB
 - Model size: the number of weights in a network work
- Compared to the state-of-the-art networks, SSAI2021 gives a competitive reconstruction quality
 - Model size: 18.27% of SRCNN [1], 83.83% of FSRCNN [2]

# of parameters Set 14 (dB)		
Bicubic	N/A	30.23
SRCNN [1]	57,184	32.18
FSRCNN [2]	12,464	32.63
SSAI2021	10,448	32.45

[1] Dong, C., Loy, C.C., He, K., Tang, X.: Learning a deep convolutional network for image super-resolution. In *Proceedings of European Conference on Computer Vision (ECCV) 2014*, pp. 184–199.

[2] Dong, C., Loy, C.C., He, K., Tang, X.: Image super-resolution using deep convolutional networks. In *Proceedings of European Conference on Computer Vision (ECCV) 2016*, pp. 295–307.

SRNPU H/W auto format: Code structure

- Code
 - hw_uniform_architecture_espcn_3x3.m: sim-espcn
 - hw_uniform_architecture_ssai2021.m: ssai2021
 - write_cnn_model_to_hex_file.m: export hex files.
 - arch/: network configurations (sim_espcn_3x3.m, ssai2021.m)
 - bin/: full-precision parameters of networks
 - func/: functions (convol2.m, hwu_relu_quantize.m, ...)
 - util/: utilities for export hex files, compute PSNR
 - Data
 - export_data/: exported feature maps
 - output/: final output image
 - output_hex_file/: hex files
 - weights, scales, biases, convout
- 
- arch
 - bin
 - export_data
 - func
 - img
 - output
 - output_hex_file
 - util
 - do_evaluation.m
 - hw_uniform_architecture_espcn_3x3.m
 - hw_uniform_architecture_ssai2021.m
 - load_preset.m
 - write_cnn_model_to_hex_file.m

SRNPU H/W auto format

- Top file (hw_uniform_architecture.m)
 1. Define a network architecture (arch/)
 2. Load weights from files
 - ❑ Do quantization: 8-bit weights, 16-bit biases, 16-bit scales
 3. Load a test image
 - ❑ Convert an RGB image to an YCbCr one
 - ❑ Make an input image which is the Y-channel image
 4. Do inference
 - ❑ Do convolution
 - ❑ Add biases
 - ❑ Activation and quantization
 5. Evaluation
 - ❑ Compare the reconstructed image and the ground truth

Network architecture

- The network architecture is defined at the folder arch/
 - Each line defines the configurations of one layer including
 - Convolution settings
 - Activation quantization parameters
 - Weight quantization parameters

Copy

```
architecture = { ...  
    {'conv', 0, ps.conv_f3_p2_s1, 16, ps.act_relu_8_8_0, ps.wts_scale_linear_8, ps.scales_16_1_1, ps.biases_16_10_1};  
    {'conv', 0, ps.conv_f3_p2_s1, 16, ps.act_relu_8_8_0, ps.wts_scale_linear_8, ps.scales_16_1_1, ps.biases_16_10_1};  
    {'conv', 0, ps.conv_f3_p2_s1, 4, ps.act_lineq_8_8_1, ps.wts_scale_linear_8, ps.scales_16_1_1, ps.biases_16_10_1};  
    {'sr_flat'};  
    {'lp_sres'};  
};  
architecture = { ...  
    {'conv', 0, ps.conv_f3_p2_s1, 16, ps.act_relu_8_7_0, ps.wts_scale_linear_8, ps.scales_16_4_1, ps.biases_16_8_1};  
    {'conv', 0, ps.conv_f1_p0_s1, 16, ps.act_relu_8_7_0, ps.wts_scale_linear_8, ps.scales_16_4_1, ps.biases_16_8_1};  
    {'conv', 0, ps.conv_f3_p2_s1, 16, ps.act_relu_8_7_0, ps.wts_scale_linear_8, ps.scales_16_4_1, ps.biases_16_8_1};  
    {'conv', 0, ps.conv_f1_p0_s1, 16, ps.act_relu_8_8_0, ps.wts_scale_linear_8, ps.scales_16_4_1, ps.biases_16_8_1};  
    {'conv', 0, ps.conv_f3_p2_s1, 4, ps.act_lineq_8_8_1, ps.wts_scale_linear_8, ps.scales_16_4_1, ps.biases_16_8_1};  
    {'sr_flat'};  
    {'lp_sres'};  
};
```

Network architecture

- The network architecture is defined in the file arch/sim_espcn_3x3.m
- Convolution (conv)
 - Filter size (f)Padding (p) and stride (s), the number of filters.
 - Example:
 - Layer 1:
 - $f=3, p=2, s=1 \Rightarrow$ Filter 3x3
 - 16 output channels.

1

```
architecture = { ...  
    {'conv', 0, ps.conv_f3_p2_s1, 16, ps.act_relu_8_8_0 , ps.wts_scale_linear_8, ps.scales_16_1_1, ps.biases_16_10_1};  
    {'conv', 0, ps.conv_f3_p2_s1, 16, ps.act_relu_8_8_0 , ps.wts_scale_linear_8, ps.scales_16_1_1, ps.biases_16_10_1};  
    {'conv', 0, ps.conv_f3_p2_s1, 4, ps.act_lineq_8_8_1, ps.wts_scale_linear_8, ps.scales_16_1_1, ps.biases_16_10_1};  
    {'sr_flat'};  
    {'lp_sres'};  
};
```

Copy...

Network architecture

- The network architecture is defined in the file arch/sim_espcn_3x3.m
- Activation quantization: [nbit fbit sign]
 - The number of bit (nbit)
 - The number of fractional bits (fbit)
 - The signed bit
 - ReLU: 0
 - Linear: 1

2

```
architecture = { ...  
    {'conv', 0, ps.conv_f3_p2_s1, 16, ps.act_relu_8_8_0, ps.wts_scale_linear_8, ps.scales_16_1_1, ps.biases_16_10_1};  
    {'conv', 0, ps.conv_f3_p2_s1, 16, ps.act_relu_8_8_0, ps.wts_scale_linear_8, ps.scales_16_1_1, ps.biases_16_10_1};  
    {'conv', 0, ps.conv_f3_p2_s1, 4, ps.act_lineq_8_8_1, ps.wts_scale_linear_8, ps.scales_16_1_1, ps.biases_16_10_1};  
    {'sr_flat'};  
    {'lp_sres'};  
};
```

Copy...

Network architecture

- The network architecture is defined in the file arch/sim_espncn_3x3.m
- Weight quantization: [nbit fbit sign]
 - The number of bit (nbit)
 - The number of fractional bits (fbit)
 - The signed bit
 - ReLU: 0
 - Linear: 1

3

```
architecture = { ...  
    {'conv', 0, ps.conv_f3_p2_s1, 16, ps.act_relu_8_8_0 , ps.wts_scale_linear_8, ps.scales_16_1_1, ps.biases_16_10_1};  
    {'conv', 0, ps.conv_f3_p2_s1, 16, ps.act_relu_8_8_0 , ps.wts_scale_linear_8, ps.scales_16_1_1, ps.biases_16_10_1};  
    {'conv', 0, ps.conv_f3_p2_s1, 4, ps.act_lineq_8_8_1, ps.wts_scale_linear_8, ps.scales_16_1_1, ps.biases_16_10_1};  
    {'sr_flat'};  
    {'lp_sres'};  
};
```

Copy...

Load weights

- Load weights from files
- Do quantization on weights, biases and scales.

```
case 'conv'
    [batchnorm, convol_settings, output_channels, ~, weights_settings] = architecture{i}{2:6};
    [kernel_size, ~, ~] = convol_settings{::};
    wts_scheme = weights_settings{1};

    input_channels = all_input_channels(i);
    param_file = [model_prefix num2str(i-1)];

    % Load weights, biases and scales from files
    [weightx, bias, scale, rolling mean, rolling var] = read conv param module(param file, kernel size, input channels, output channels);

    % Quantization
    if strcmp(wts_scheme, 'none')
        weight = weightx;
        w_bonus_scale_factor = ones(output_channels,1);
    elseif strcmp(wts_scheme, 'uniform')
        [wts_nbit, wts_fbit, ~] = weights_settings{2:end};
        wts_step = 2^-wts_fbit;
        [weight, ~] = uniform_quantize(weightx, wts_step, wts_nbit);
        w_bonus_scale_factor = ones(output_channels,1);
    elseif strcmp(wts_scheme, 'scale_linear')
        wts_nlevel = 2^weights_settings{2};
        [weight, w_bonus_scale_factor] = scale_linear_quantize(weightx, output_channels, wts_nlevel/2);
    elseif strcmp(wts_scheme, 'scale_linear_float')
        wts_nlevel = 2^weights_settings{2};
        [weight, w_bonus_scale_factor] = scale_linear_quantize_float(weightx, output_channels, wts_nlevel/2);
    end

    weights{i} = permute(reshape(weight, kernel_size, kernel_size, input_channels, output_channels), [2, 1, 3, 4]);
```

Input preparation

- Load a test image from a file (imread)
- Resize an image to generate a Low-resolution image (imresize)
- Convert an RGB to YCbCr
 - Only work with the Y-channel image

```
%> work on illuminance only
im = imread(image_name);
input_img = modcrop(im, up_scale);
input_img = single(input_img)/255;
input_img = imresize(input_img, 1/up_scale, 'bicubic');
if size(im,3) > 1
    im_ycbcr = rgb2ycbcr(im);
    im = im_ycbcr(:,:,1);
end
im_gnd = modcrop(im, up_scale);
im_gnd = single(im_gnd)/255;
im_l = imresize(im_gnd, 1/up_scale, 'bicubic');

input = floor(im_l(:,:,1) * 255);
```



High-resolution



Low-resolution

Do convolution

- Load quantized weights
- Do convolution (convol2)
- Do activation quantization (hwu relu quantize/hwu linear quantize)

```
weight = weights{i};

conv_out = convol2(input, weight, stride, pad);
for j = 1:size(conv_out, 3)
    conv_out(:,:,j) = conv_out(:,:,j) .* scales{i}(j);
    conv_out(:,:,j) = floor(conv_out(:,:,j) / 2^bit_shift(i)); %if floating point is
    conv_out(:,:,j) = conv_out(:,:,j) + biases{i}(j);
end

if strcmp(activation, 'float_relu')
    output = hwu_float_relu_activate(conv_out);
elseif strcmp(activation, 'relu')
    [act_nbit, act_fbit] = act_settings{2:3};
    act_step = 2^-act_fbit;
    [output, ~] = hwu_relu_quantize(conv_out, act_step, act_nbit, biases_fbit);
elseif strcmp(activation, 'line_q')
    [act_nbit, act_fbit] = act_settings{2:3};
    act_step = 2^-act_fbit;
    [output, ~] = hwu_linear_quantize(conv_out, act_step, act_nbit, biases_fbit);
else
    output = conv_out;
end
```

Copyright © 2022, reserved.

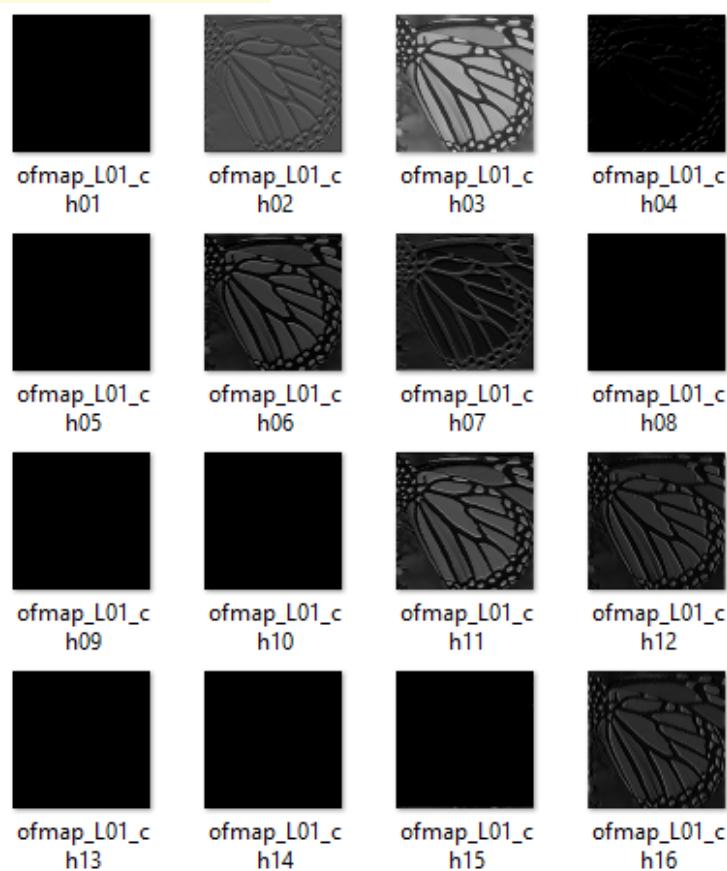
Test vector preparation

```
elseif strcmp(activation, 'relu')
    [act_nbit, act_fbit] = act_settings{2:3};
    act_step = 2^-act_fbit;
    [output, ~] = hwu_relu_quantize(conv_out, act_step, act_nbit, biases_fbit);
    for j = 1:size(output, 3)
        fmap = uint8(output(:,:,j));
        imwrite(fmap,sprintf('%s/%s/ofmap_L%02d_ch%02d.bmp',outdir,model_name,i,j));
    end
elseif strcmp(activation, 'line_q')
    [act_nbit, act_fbit] = act_settings{2:3};
    act_step = 2^-act_fbit;
    [output, ~] = hwu_linear_quantize(conv_out, act_step, act_nbit, biases_fbit);
    for j = 1:size(output, 3)
        fmap = output(:,:,j);
        fmap(fmap<0) = 255-fmap(fmap<0);
        fmap = uint8(fmap);
        imwrite(fmap,sprintf('%s/%s/ofmap_L%02d_ch%02d.bmp',outdir,model_name,i,j));
    end
else
    output = conv_out;
end
weight_store = (weight-1)/2;
weight_store(weight_store<0) = weight_store(weight_store<0) + 256;
test_vector{i,1} = input;
test_vector{i,2} = weight_store;
test_vector{i,3} = biases{i};
test_vector{i,4} = scales{i};
test_vector{i,5} = bit_shift(i);
test_vector{i,6} = biases_fbit-act_fbit;
test_vector{i,7} = output;
```

Test vector preparation

- Store the output feature maps of all layers
 - Layer indexes and channel indexes

```
elseif strcmp(activation, 'relu')
    [act_nbit, act_fbit] = act_settings{2:3};
    act_step = 2^act_fbit;
    [output, ~] = hwu_relu_quantize(conv_out, act_step, act_nbit
for j = 1:size(output, 3)
    fmap = uint8(output(:,:,j));
    imwrite(fmap,sprintf('%s/%s/ofmap_L%02d_ch%02d.bmp',outd
end
elseif strcmp(activation, 'line_q')
    [act_nbit, act_fbit] = act_settings{2:3};
    act_step = 2^act_fbit;
    [output, ~] = hwu_linear_quantize(conv_out, act_step, act_nb
for j = 1:size(output, 3)
    fmap = output(:,:,j);
    fmap(fmap<0) = 255-fmap(fmap<0);
    fmap = uint8(fmap);
    imwrite(fmap,sprintf('%s/%s/ofmap_L%02d_ch%02d.bmp',outd
end
else
    output = conv_out;
end
weight_store = (weight-1)/2;
weight_store(weight_store<0) = weight_store(weight_store<0) + 25
test_vector{i,1} = input;
test_vector{i,2} = weight_store;
test_vector{i,3} = biases(i);
test_vector{i,4} = scales(i);
test_vector{i,5} = bit_shift(i);
test_vector{i,6} = biases_fbit-act_fbit;
test_vector{i,7} = output;
```



Test vector preparation

- Store test vectors of all layers in a 7-column format
 - Input, weights, biases, scales, bias_shift, act_shift and output.
- Example: the first layer of ESPCN
 - Input: 128x128, output: 128x128x16
 - Weights: 3x3x1x16, biases: 16x1, scales: 16x1
 - bias_shift = 9, act_shift = 7

The screenshot shows the MATLAB workspace with a cell array named 'test_vector'. A blue arrow points from the list above to the workspace. A red dashed arrow points from the bottom code block to the third row of the 'test_vector' table.

test_vector

	1	2	3	4	5	6	7
1	128x128 single	4-D double	16x1 double	16x1 double	9	7	128x128x16 double
2	128x128x16 double	4-D double	16x1 double	16x1 double	17	7	128x128x16 double
3	128x128x16 double	4-D double	[-3;-2;-2;-3]	[34452;29206;27010;40628]	17	7	128x128x4 double

```
else
    output = conv_out;
end

weight_store = (weight-1)/2;
weight_store(weight_store<0) = weight_store(weight_store<0) + 256;
test_vector{i,1} = input;
test_vector{i,2} = weight_store;
test_vector{i,3} = biases{i};
test_vector{i,4} = scales{i};
test_vector{i,5} = bit_shift(i);
test_vector{i,6} = biases_fbit-act_fbit;
test_vector{i,7} = output;
```

Evaluation

- How to evaluation? Compare an output image with the original image.
 - Peak-signal-to-noise (PSNR) in dB: Higher -> better
 - Mean Absolute Error (MAE): Smaller -> better
- Comparison
 - Bicubic: PSNR = 27.43 dB, MAE = 6.40
 - ESPCN: PSNR = 30.97 dB, MAE = 4.58
 - SSAI2021: PSNR = 32.68 dB, MAE = 3.64



Ground truth

Low-resolution

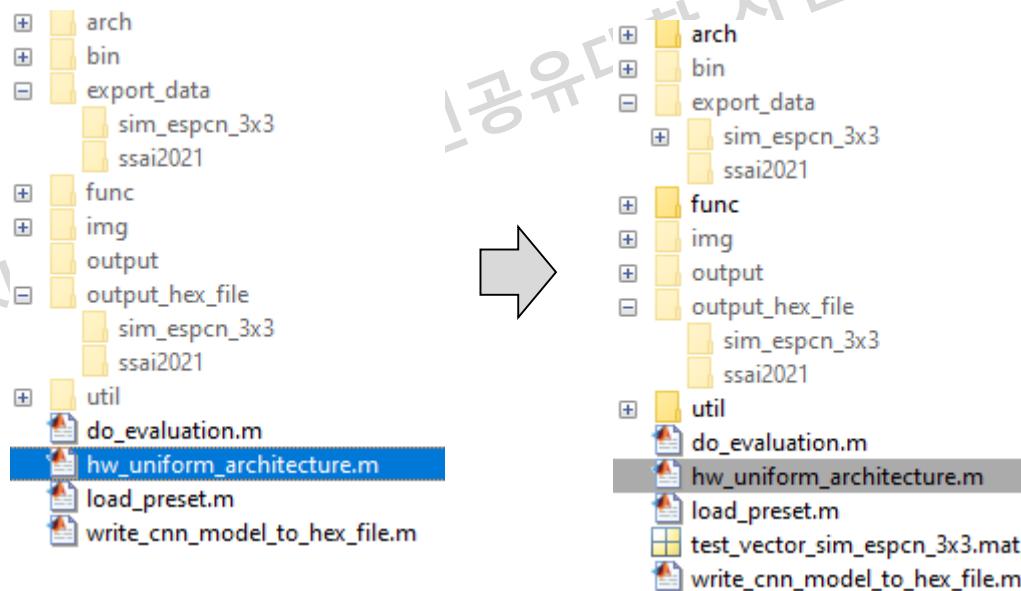
Bicubic Interpolation

ESPCN (30.97 dB)

SSAI (32.68 dB)

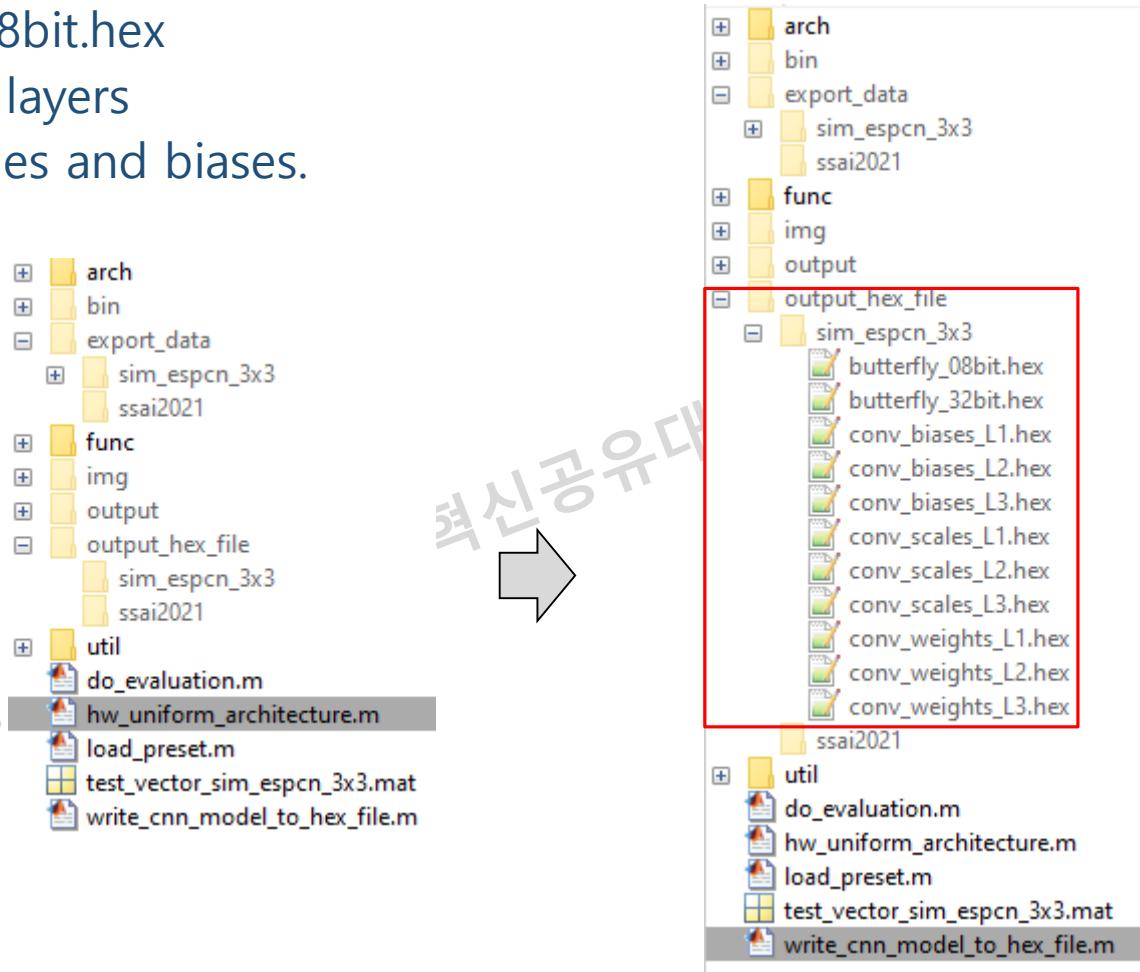
To do ...

- Run the top file (hw_uniform_architecture.m)
 - Add break points to understand the flow in page 11
- You should see the results
 - The feature maps at export_data/sim_espcn_3x3
 - The test vector (test_vector_sim_espcn_3x3.mat)
 - Evaluation images at output/sim_espcn_3x3



To do ...

- Generate hex files for H/W simulation (write_cnn_model_to_hex_file.m)
 - Input: butterfly_08bit.hex
 - Parameters of all layers
 - Weights, scales and biases.



To do ...

- Modify the code that is used to generate the input image in a hex file
 - Original code: `buf = transposed input (input')`
 - Modified code: `buf = input`
- Use the newly generated `butterfly_08bit.hex` for exercise 1.
 - How do the output feature maps change?

```
%% Load the test vector
load(sprintf('test_vector_%s.mat', model_name));

%% Write a test image
input = test_vector{1,1};
buf = input';      % Transpose
%buf = input;      % |
buf  = buf(:);
% 8-bit format
fid = fopen(sprintf('%s/%s/butterfly_08bit.hex',outdir, model_name), 'wt');
fprintf(fid, '%x\n',buf);
fclose(fid);
% 32-bit format
fid = fopen(sprintf('%s/%s/butterfly_32bit.hex',outdir, model_name), 'wt');
fprintf(fid, '%08x\n',buf);
fclose(fid);
```

Copyright 2021

The final project

- Run the top file (hw_uniform_architecture_espcn_3x3.m)
- Generate hex files for H/W simulation (write_cnn_model_to_hex_file.m)
 - Input: butterfly_08bit.hex
 - Parameters of all layers
 - Weights, scales and biases.

Copyright 2021. (차세대반도체 혁신공유대학 사업단) all rights reserved.

Road map

Review

Reference Software

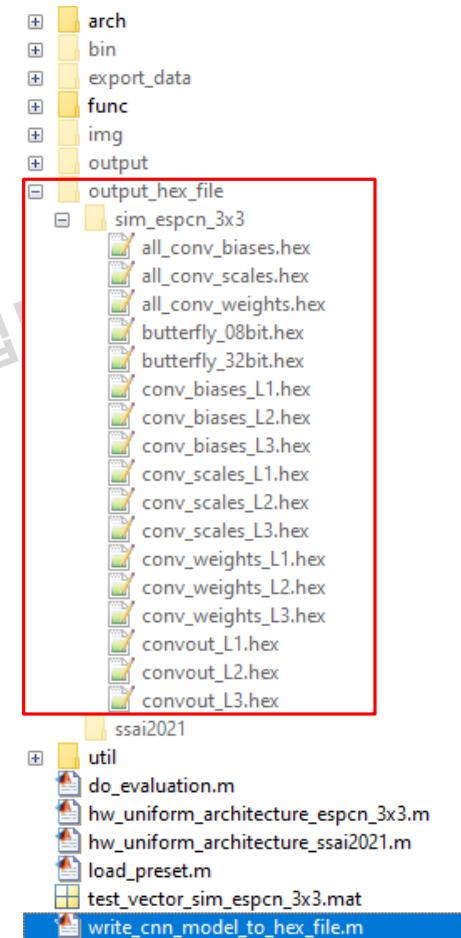
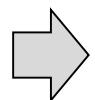
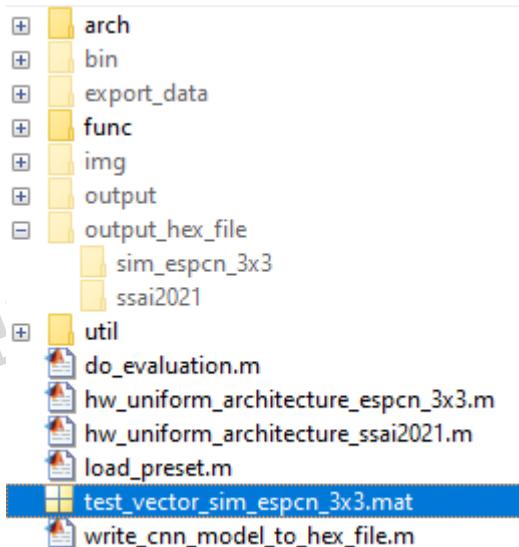
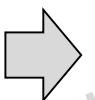
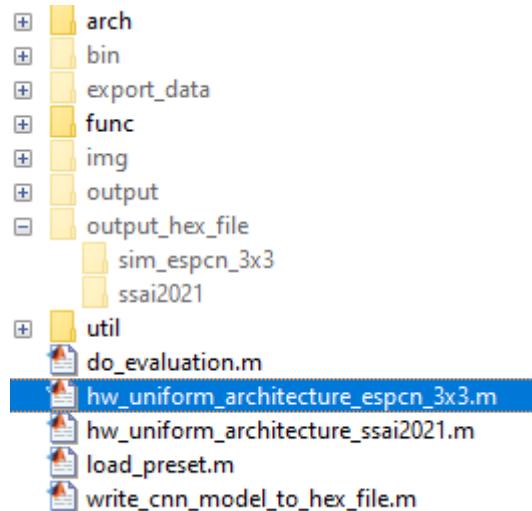
weight/bias/scale
buffers

Output buffers

Control path
(CONV3x3)

Motivation

- Generate hex files for H/W simulation (write_cnn_model_to_hex_file.m)
 - Input: butterfly_08bit.hex
 - Parameters of all layers
 - Weights, scales and biases.



How to set scales and biases?

- Four biases and four scales are preloaded to a buffer
- Four kernels use the same bias_shift and the same act_shift

```
initial begin
    rstn = 1'b0;                      // Reset,
    vld_i = 1'b0;
    win[0] = 0;
    win[1] = 0;
    win[2] = 0;
    win[3] = 0;
    din = 0;
    is_conv3x3 = 1'b0;
    is_last_layer = 1'b0;
    scale[0] = 16'd95;
    scale[1] = 16'd103;
    scale[2] = 16'd364;
    scale[3] = 16'd170;
    bias[0] = 16'd46916;      // -18620
    bias[1] = 16'd8066;        // 8060
    bias[2] = 16'd370;         // 370
    bias[3] = 16'd65030;       // -506
    bias_shift = 9;
    act_shift = 7;
```

Biases

	1
1	-18620
2	8066
3	370
4	-506
5	-1775
6	-1726
7	5917
8	-1652
9	1642
10	409
11	-1867
12	-732
13	-11470
14	-7075
15	562
16	-405

Scales

	1
1	95
2	103
3	364
4	170
5	122
6	310
7	203
8	121
9	107
10	160
11	309
12	263
13	77
14	106
15	175
16	186

How to set weights?

- Weights are preloaded to a buffer (win)
 - Weights of four channels in first_layer_tb.v

```
// First layer, channel 00:  
win[0][0*WI+:WI] = 8'd142;  
win[0][1*WI+:WI] = 8'd151;  
win[0][2*WI+:WI] = 8'd215;  
win[0][3*WI+:WI] = 8'd127;  
win[0][4*WI+:WI] = 8'd163;  
win[0][5*WI+:WI] = 8'd205;  
win[0][6*WI+:WI] = 8'd229;  
win[0][7*WI+:WI] = 8'd255;  
win[0][8*WI+:WI] = 8'd113;
```

```
// First layer, channel 02:  
win[2][0*WI+:WI] = 8'd13;  
win[2][1*WI+:WI] = 8'd244;  
win[2][2*WI+:WI] = 8'd255;  
win[2][3*WI+:WI] = 8'd241;  
win[2][4*WI+:WI] = 8'd127;  
win[2][5*WI+:WI] = 8'd240;  
win[2][6*WI+:WI] = 8'd252;  
win[2][7*WI+:WI] = 8'd237;  
win[2][8*WI+:WI] = 8'd1;
```

```
// First layer, channel 01:  
win[1][0*WI+:WI] = 8'd69;  
win[1][1*WI+:WI] = 8'd181;  
win[1][2*WI+:WI] = 8'd209;  
win[1][3*WI+:WI] = 8'd19;  
win[1][4*WI+:WI] = 8'd128;  
win[1][5*WI+:WI] = 8'd95;  
win[1][6*WI+:WI] = 8'd221;  
win[1][7*WI+:WI] = 8'd121;  
win[1][8*WI+:WI] = 8'd8;
```

```
// First layer, channel 03:  
win[3][0*WI+:WI] = 8'd69;  
win[3][1*WI+:WI] = 8'd135;  
win[3][2*WI+:WI] = 8'd235;  
win[3][3*WI+:WI] = 8'd128;  
win[3][4*WI+:WI] = 8'd32;  
win[3][5*WI+:WI] = 8'd90;  
win[3][6*WI+:WI] = 8'd48;  
win[3][7*WI+:WI] = 8'd52;  
win[3][8*WI+:WI] = 8'd211;
```

Software

```
val(:,:,:,1,1) =  
  
142 151 215  
127 163 205  
229 255 113
```

```
val(:,:,:,1,2) =  
  
69 181 209  
19 128 95  
221 121 8
```

```
val(:,:,:,1,3) =  
  
13 244 255  
241 127 240  
252 237 1
```

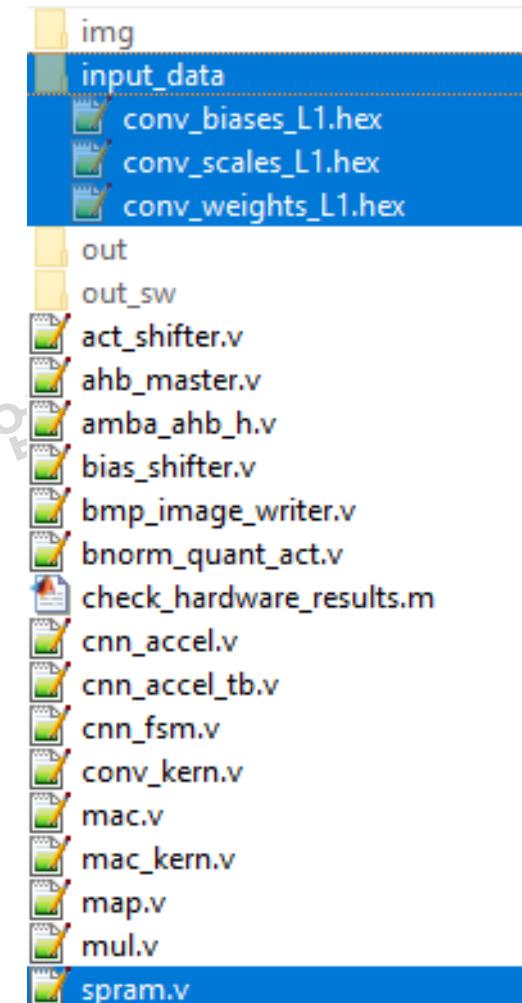
```
val(:,:,:,1,4) =  
  
69 135 235  
128 32 90  
48 52 211
```

Lab 1: weight, bias, and scale buffers

- Lab 1:
 - Reuse cnn_fsm.v
 - cnn_accel.v
 - Reuse code for setting configuration registers
 - Reuse code to generate din, vld_i
 - Add buffers for weights, biases and scales.
 - Implement related logics
 - Do simulation with time = 400 us
 - Show/capture the output results

Code structure

- RTL file
 - Convolution kernel (conv_kern.v)
 - MAC kernel: mac_kern.v, mac.v, mul.v
 - Batch normalization and activation
 - bnorm_quant_act.v
 - bias_shifter.v, act_shifter.v
 - CNN accelerator IP
 - cnn_accel.v, cnn_fsm.v
 - AHB interface
 - amba_ahb_h.v, ahb_master.v, map.v
 - BMP image writer (bmp_image_writer.v)
 - Test bench: cnn_accel_tb.v
 - Single port block ram (spram.v)
- img/: input image in hex file
- out/: output results by H/W simulation
- out_sw/: outputs by S/W simulation
- input_data/: weights, scales, biases files for the first layer



Single-port block RAM (spram.v)

- Single-port block ram (spram.v)
 - Ports
 - Read/Write Enable (en)
 - Address (addr)
 - Write data (din)
 - Write enable (we)
 - Read data (dout).
 - Memory cell: q_mem
- Parameters
 - W_DATA: word size
 - N_WORD: # of words
 - W_WORD: address size
 - INIT_FILE: initialized file
 - EN_LOAD_INIT_FILE: flag

```
module spram(
    clk,      // Clock input
    en,       // RAM enable (select)
    addr,     // Address input(word addressing)
    din,      // Data input
    we,       // Write enable
    dout);   // Data output

parameter W_DATA = 32;
//RAM cell size(# of words)
parameter N_WORD = 16;
parameter W_WORD = 4;
parameter EN_LOAD_INIT_FILE = 1'b0;
parameter INIT_FILE = "img/kodim03_32bit.hex";

input          clk;    // Clock input
input          en;     // RAM enable (select)
input [W_WORD-1:0] addr;  // Address input(word addressing) prior to read
input [W_DATA-1:0] din;   // Data input
input          we;     // Write enable
output [W_DATA-1:0] dout;  // Data output

reg [W_DATA-1:0] q_mem[N_WORD-1:0] /* synthesis syn_ramstyle="block_ram" */;
reg [W_DATA-1:0] dout = 32'h0000_0000;
```

Single-port block RAM (spram.v)

- Initialization: q_mem can be initialized from a file

- EN_LOAD_INIT_FILE == 1

- Read operation

- If (en)
 - dout \leftarrow q_mem[addr]

- Write operation

- If (en & we)
 - q_mem[addr] \leftarrow din

- Single-port

- Read/write operations share "addr"
 - Only read or write occurs at a time

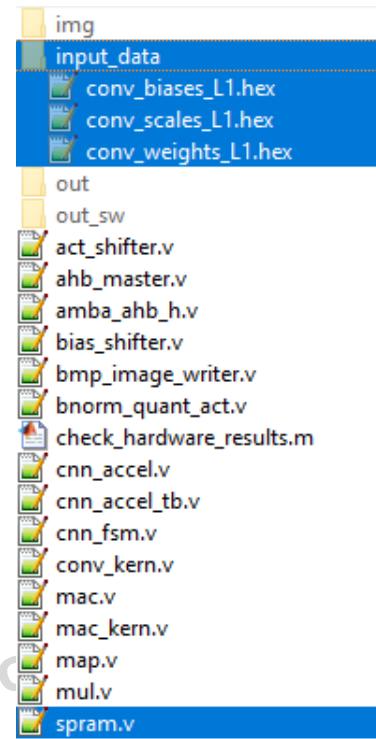
```
reg [W_DATA-1:0] q_mem[N_WORD-1:0] /* synthesis syn_ramstyle;
reg [W_DATA-1:0] dout = 32'h0000_0000;

always @ (posedge clk)
begin
    if(en & we)
        q_mem[addr] <= din;
    else if(en)
        dout <= q_mem[addr];
end

// synopsys translate_off
initial
begin
    if(EN_LOAD_INIT_FILE)
        $readmemh(INIT_FILE, q_mem);
end
// synopsys translate_on
```

Weight/bias/scale files

- Initialized files for weights, biases and scales are stored in input_data/
 - Word size: 128 bits for "weight", 16 bits for "bias" and "scale".
 - Number of words: 16



conv_weight_L1.hex

1	0000000000000071FFE5CDA37FD7978E
2	00000000000000879DD5F8013D1B545
3	000000000000001EDFCF07FF1FFF40D
4	00000000000000D334305A2080EB8745
5	0000000000000033EF09D5FB8003CFF3
6	00000000000000FEEE02FC7FF6FAD2F8
7	00000000000000F880F7165702F4020D
8	00000000000000FB41CD8BB82B3480F3
9	0000000000000080AB00093CD20ED39F
10	00000000000000E2EE10AC4403C180CE
11	00000000000000FB0501FEBDF2F67F0D
12	0000000000000060EF6A77FF201FDFE
13	000000000000001F815BE67FD4F8800D
14	00000000000000193EDB928BAE1F3080
15	000000000000008BAB80F90D42F02400
16	000000000000AF906E67FA8ED60C6

conv_bias_L1.hex

1	b744
2	1f82
3	0172
4	fe06
5	f911
6	f942
7	171d
8	f98c
9	066a
10	0199
11	f8b5
12	fd24
13	d332
14	e45d
15	0232
16	fe6b

conv_scale_L1.hex

1	005f
2	0067
3	016c
4	00aa
5	007a
6	0136
7	00cb
8	0079
9	006b
10	00a0
11	0135
12	0107
13	004d
14	006a
15	00af
16	00ba

Weight/bias/scale buffers

- Define three spram instances for weight, bias and scale buffer
 - Parameters
 - INIT_FILE
 - EN_LOAD_INIT_FILE (1)
 - W_DATA, N_WORD, W_WORD
 - Ports
 - en, addr, din, we, dout
 - To do ...
 - Complete the missing codes for bias and scale buffers

```
// Weight buffer
spram #(.INIT_FILE("input_data/conv_weights_L1.hex"),
    .EN_LOAD_INIT_FILE(EN_LOAD_INIT_FILE),
    .W_DATA(Ti*WI), .W_WORD(W_CELL), .N_WORD(N_CELL))
|u_buf_weight(
    .clk (clk), // Clock input
    .en (weight_buf_en), // RAM enable (select)
    .addr(weight_buf_addr), // Address input(word addressing)
    .din /*unused*/), // Data input
    .we (weight_buf_we), // Write enable
    .dout(weight_buf_dout) // Data output
);
// Bias buffer
spram #(.INIT_FILE(/*Insert your code*/),
    .EN_LOAD_INIT_FILE(EN_LOAD_INIT_FILE),
    .W_DATA/*Insert your code*/, .W_WORD(W_CELL_PARAM), .N_WORD(N_CELL_PARAM))
|u_buf_bias(
    .clk (clk), // Clock input
    .en (param_buf_en), // RAM enable (select)
    .addr(param_buf_addr), // Address input(word addressing)
    .din /*unused*/), // Data input
    .we (param_buf_we), // Write enable
    .dout(param_buf_dout_bias) // Data output
);
// Scale buffer
spram #(.INIT_FILE(/*Insert your code*/),
    .EN_LOAD_INIT_FILE(EN_LOAD_INIT_FILE),
    .W_DATA/*Insert your code*/, .W_WORD(W_CELL_PARAM), .N_WORD(N_CELL_PARAM))
|u_buf_scale(
    .clk (clk), // Clock input
    .en /*Insert your code*/, // RAM enable (select)
    .addr/*Insert your code*/, // Address input(word addressing)
    .din /*unused*/), // Data input
    .we /*Insert your code*/, // Write enable
    .dout/*Insert your code*/ // Data output
);
```

Internal signals for weight/bias/scale buffers

- Enable signals: *_buf_en, *_buf_en_d, *_buf_we
- Addresses: *_buf_addr, *_buf_addr_d
- Buffer outputs: *_buf_dout_*

```
// Weight/bias/scale buffer's signals
// weight
reg          weight_buf_en;           // primary enable
reg          weight_buf_en_d;         // primary enable
reg          weight_buf_we;          // primary synchronous write enable
reg [W_CELL-1:0] weight_buf_addr;    // address for read/write
reg [W_CELL-1:0] weight_buf_addr_d;   // 1-cycle delay address
wire[Ti*WI-1:0] weight_buf_dout;     // Output for weights

// bias/scale
reg          param_buf_en;          // primary enable
reg          param_buf_en_d;         // primary enable
reg          param_buf_we;          // primary synchronous write enable
reg [W_CELL_PARAM-1:0] param_buf_addr; // address for read/write
reg [W_CELL_PARAM-1:0] param_buf_addr_d; // 1-cycle delay address
wire[PARAM_BITS-1:0] param_buf_dout_bias; // Output for biases
wire[PARAM_BITS-1:0] param_buf_dout_scale; // Output for scales
integer ch_idx;
```

Access weight/bias/scale buffers

- Assume that weight, scale and bias registers are updated during frame synchronization
 - VSYNC
 - Request address is updated from the VSYNC counter `ctrl_vsync_cnt`
 - CONV1x1
 - To requests are needed
 - To do ...**
 - Complete the missing codes for bias and scale requests**

```
// Weight
always@(*) begin
    weight_buf_en  = 1'b0;
    weight_buf_we = 1'b0;
    weight_buf_addr = {W_CELL{1'b0}};
    if(ctrl_vsync_run) begin
        if(!q_is_conv3x3) begin // Conv1x1
            if(ctrl_vsync_cnt < To) begin
                weight_buf_en  = 1'b1;
                weight_buf_we = 1'b0;
                weight_buf_addr = ctrl_vsync_cnt[W_CELL-1:0];
            end
        end
    end
end

// Scale/bias
always@(*) begin
    param_buf_en  = 1'b0;
    param_buf_we = 1'b0;
    param_buf_addr = {W_CELL{1'b0}};
    if(ctrl_vsync_run) begin
        if(ctrl_vsync_cnt < To) begin
            //param_buf_en  = /*Insert your code*/;
            //param_buf_we = /*Insert your code*/;
            //param_buf_addr = /*Insert your code*/;
        end
    end
end
```

Access weight/bias/scale buffers

- One-cycle delay signals
 - Enable: *_buf_en_d
 - Address: *_buf_addr_d
- Win, scale, bias registers are updated when outputs from their buffers are read out.
- Example
 - $\text{win}[\text{weight_buf_addr_d}] \leftarrow \text{weight_buf_dout}$

```
// one-cycle delay
always@(posedge clk, negedge rstn)begin
  if(~rstn) begin
    weight_buf_en_d  <= 1'b0;
    weight_buf_addr_d <= {W_CELL{1'b0}};
    param_buf_en_d   <= 1'b0;
    param_buf_addr_d <= {W_CELL{1'b0}};
  end
  else begin
    weight_buf_en_d  <= weight_buf_en;
    weight_buf_addr_d <= weight_buf_addr;
    param_buf_en_d   <= param_buf_en;
    param_buf_addr_d <= param_buf_addr;
  end
end

always@(posedge clk, negedge rstn)begin
  if(~rstn) begin
    for(ch_idx = 0; ch_idx < To; ch_idx=ch_idx+1) begin
      win[ch_idx]  <= {(Ti*WI){1'b0}};
      bias[ch_idx] <= {PARAM_BITS{1'b0}};
      scale[ch_idx] <= {PARAM_BITS{1'b0}};
    end
  end
  else begin
    // Weight
    if(weight_buf_en_d)
      win[weight_buf_addr_d] <= weight_buf_dout;
    // Scale/bias
    /*Insert your code*/
  end
end
```

Test bench (cnn_accel_tb.v)

- Test bench
 - Master: RISC-V (ahb_master.v)
 - Slave: A CNN accelerator IP (cnn_accel.v)

```
'timescale 1ns / 100ps
`include "amba_ahb_h.v"
`include "map.v"

module cnn_accel_tb;
parameter W_ADDR=32;
parameter W_DATA=32;
parameter IMG_PIX_W = 8;

//parameter WIDTH    = 768,
//      HEIGHT   = 512,
parameter   WIDTH    = 128,
            HEIGHT   = 128,
            START_UP_DELAY = 200,
            HSYNC_DELAY = 160,
            FRAME_SIZE = WIDTH * HEIGHT;
localparam W_SIZE    = 12;                      // Max 4K QHD (3840x1920).
localparam W_FRAME_SIZE = 2 * W_SIZE + 1; // Max 4K QHD (3840x1920).
localparam W_DELAY   = 12;

parameter N_LAYER = 3;
// Inputs
reg HCLK;
reg HRESETn;
```

Master

```
//-----
// Master
//-----

ahb_master u_riscv_dummy(
    .HRESETn      (HRESETn),
    .HCLK         (HCLK),
    ..i_HRDATA   (w_RISC2AHB_mst_HRDATA),
    ..i_HRESP    (w_RISC2AHB_mst_HRESP),
    ..i_HREADY   (w_RISC2AHB_mst_HREADY),
    ..o_HADDR    (w_RISC2AHB_mst_HADDR),
    ..o_HWDATA   (w_RISC2AHB_mst_HWDATA),
    ..o_HWRITE   (w_RISC2AHB_mst_HWRITE),
    ..o_HSIZE    (w_RISC2AHB_mst_HSIZE),
    ..o_HBURST   (w_RISC2AHB_mst_HBURST),
    ..o_HTRANS   (w_RISC2AHB_mst_HTRANS)
);

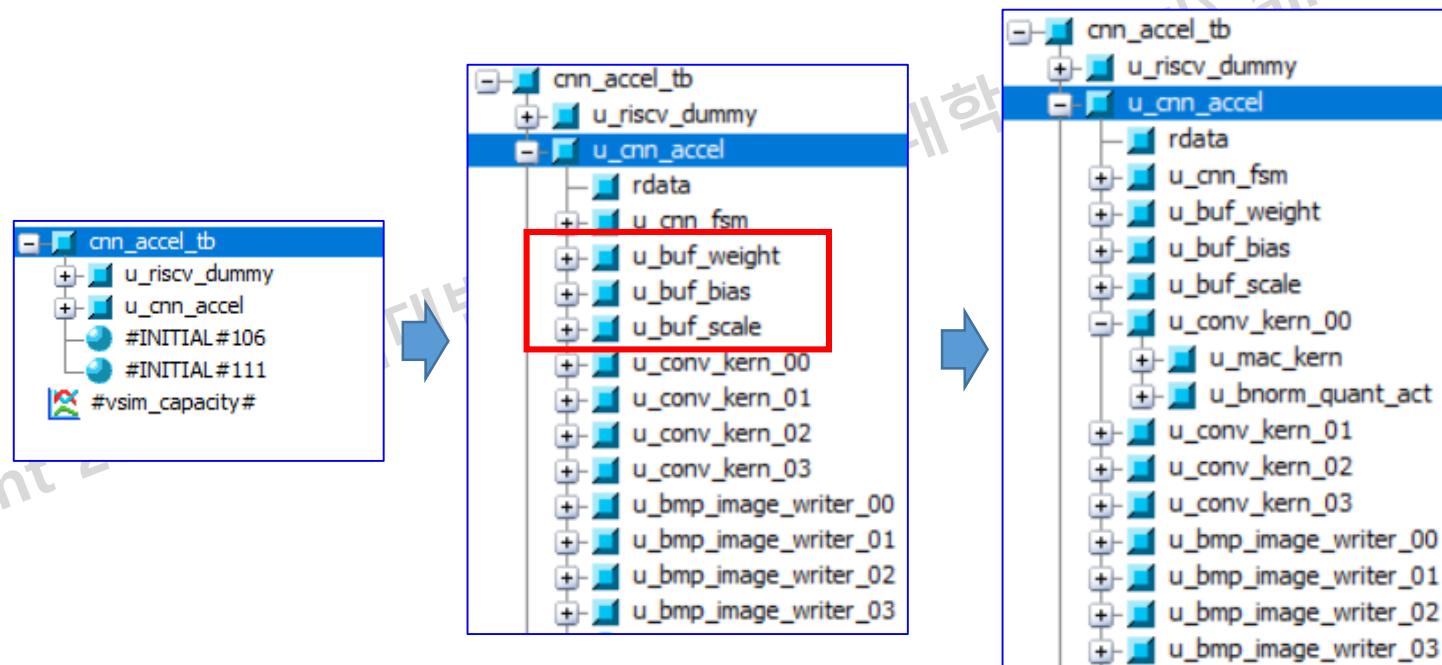
//-----
// Slave
//-----

cnn_accel u_cnn_accel (
    .HCLK(HCLK),
    .HRESETn(HRESETn),
    .sl_HREADY(1'b1),
    .sl_HSEL(1'b1),
    .sl_HTRANS(w_RISC2AHB_mst_HTRANS),
    .sl_HBURST(w_RISC2AHB_mst_HBURST),
    .sl_HSIZE(w_RISC2AHB_mst_HSIZE),
    .sl_HADDR(w_RISC2AHB_mst_HADDR),
    .sl_HWRITE(w_RISC2AHB_mst_HWRITE),
    .sl_HWDATA(w_RISC2AHB_mst_HWDATA),
    .out_sl_HREADY(w_RISC2AHB_mst_HREADY),
    .out_sl_HRESP(w_RISC2AHB_mst_HRESP),
    .out_sl_HRDATA(w_RISC2AHB_mst_HRDATA)
);
```

Slave

Test bench top view (cnn_accel_tb.v)

- cnn_accel_tb: a master (riscv_dummy) and a slave (cnn_accel)
- cnn_accel: A controller (cnn_fsm), **weight/bias/scale buffers**, four convolutional kernels (conv_kern), and four output writers
- cnn_kern: an MAC kernel (mac_kern) and a batch normalization and quantization (bnorm_quant_act)
- mac_kern: MAC having 16 multipliers and an adder tree



Signals and test vector (cnn_accel_tb.v)

- Define internal register for a three-layer network
- Generate clock, reset signals
- Prepare a test vector

```
reg [W_SIZE-1 :0]          q_width;
reg [W_SIZE-1 :0]          q_height;
reg [W_DELAY-1:0]          q_start_up_delay;
reg [W_DELAY-1:0]          q_hsync_delay;
reg [W_FRAME_SIZE-1:0]      q_frame_size;
reg                      q_layer_start;
reg [3:0]                  q_layer_index;
reg                      q_layer_done;
reg [31:0]     q_layer_config;
reg [2:0]      q_act_shift [0:N_LAYER-1];
reg [4:0]      q_bias_shift [0:N_LAYER-1];
reg          q_is_conv3x3 [0:N_LAYER-1];
reg [7:0]      q_in_channels [0:N_LAYER-1];
reg [7:0]      q_out_channels[0:N_LAYER-1];
reg          q_is_first_layer;
reg          q_is_last_layer;
reg          q_conv_type;

reg [19:0]    base_addr_weight;
reg [11:0]    base_addr_param;

reg [W_DATA-1:0] rdata;
reg [W_ADDR-1:0] i;
reg image_load_done;
integer idx;
```



```
//-----
// Test vectors
//-----
// Clock
parameter p = 10; //100MHz
initial begin
  HCLK = 1'b0;
  forever #(p/2) HCLK = ~HCLK;
end

initial begin
  // Initialize Inputs
  HRESETn = 0;
  q_width      = WIDTH;
  q_height     = HEIGHT;
  q_start_up_delay = START_UP_DELAY;
  q_hsync_delay = HSYNC_DELAY;
  q_frame_size = FRAME_SIZE;
  q_layer_index = 4'd0;
  q_layer_done  = 1'b0;
  q_is_first_layer = 1'b0;
  q_is_last_layer = 1'b0;
  q_layer_config = 32'h0;

  // Define Network's parameters
  q_bias_shift[0] = 9 ; q_act_shift[0] = 7; q_is_conv3x3[0] = 0;
  q_bias_shift[1] = 17; q_act_shift[1] = 7; q_is_conv3x3[1] = 1;
  q_bias_shift[2] = 17; q_act_shift[2] = 7; q_is_conv3x3[2] = 1;
  // Loop/Layer index
  idx = 0;
```

Test vector (cnn_accel_tb.v)

```
// Weight/bias/Scale base addresses
base_addr_weight = 0;
base_addr_param = 0;

// Initialize RISCV dummy core
u_riscv_dummy.task_AHBinit();

// Memory
rdata = 0;
i = 0;
image_load_done = 0;

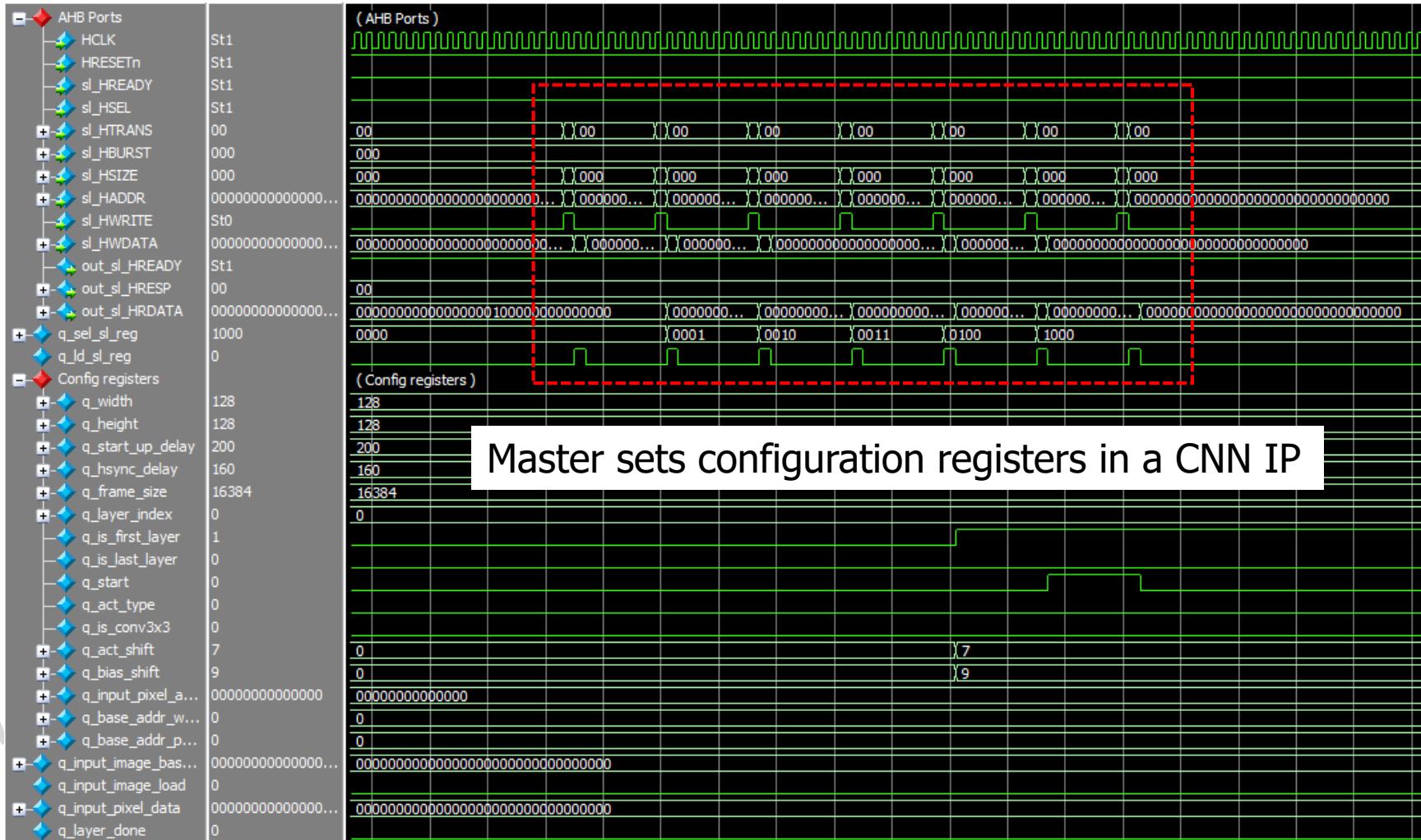
#(p/2) HRESETn = 1;
//*****
// CNN Accelerator configuration
//*****
#(100*p)
#(4*p) @ (posedge HCLK) u_riscv_dummy.task_AHBwrite(`CNN_ACCEL_FRAME_SIZE , q_frame_size );
#(4*p) @ (posedge HCLK) u_riscv_dummy.task_AHBwrite(`CNN_ACCEL_WIDTH_HEIGHT , {q_height&16'hFFF, q_width&16'hFFF});
#(4*p) @ (posedge HCLK) u_riscv_dummy.task_AHBwrite(`CNN_ACCEL_DELAY_PARAMS , {q_hsync_delay, q_start_up_delay});
...
//*****
// Loop
//*****
idx = 0; // Layer 1
q_layer_index = idx;
q_is_last_layer = (idx == N_LAYER-1)?1'b1:1'b0;
q_is_first_layer = (idx == 0) ? 1'b1: 1'b0;
q_layer_config = {q_act_shift[idx], q_bias_shift[idx], q_layer_index, q_is_last_layer, q_is_conv3x3[idx], q_is_last_layer, q_is_first_layer};
#(4*p) @ (posedge HCLK) u_riscv_dummy.task_AHBwrite(`CNN_ACCEL_BASE_ADDRESS, {base_addr_param&12'hFFF, base_addr_weight&20'hFFFF});
#(4*p) @ (posedge HCLK) u_riscv_dummy.task_AHBwrite(`CNN_ACCEL_LAYER_CONFIG, q_layer_config);
// Start a frame
#(4*p) @ (posedge HCLK) u_riscv_dummy.task_AHBwrite(`CNN_ACCEL_LAYER_START , 1'b1 );
#(4*p) @ (posedge HCLK) u_riscv_dummy.task_AHBwrite(`CNN_ACCEL_LAYER_START , 1'b0 );
```

RISC-V configures a layer's configuration registers

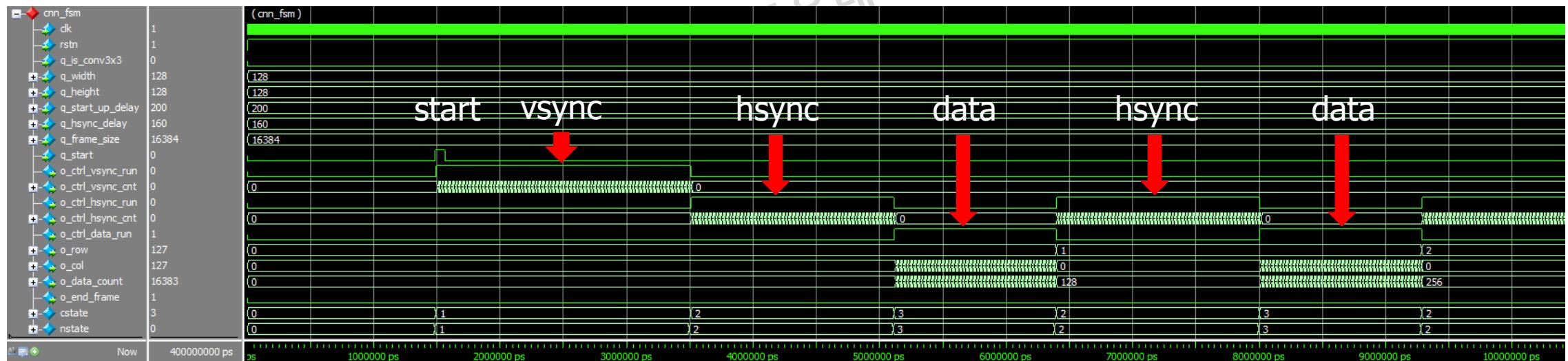
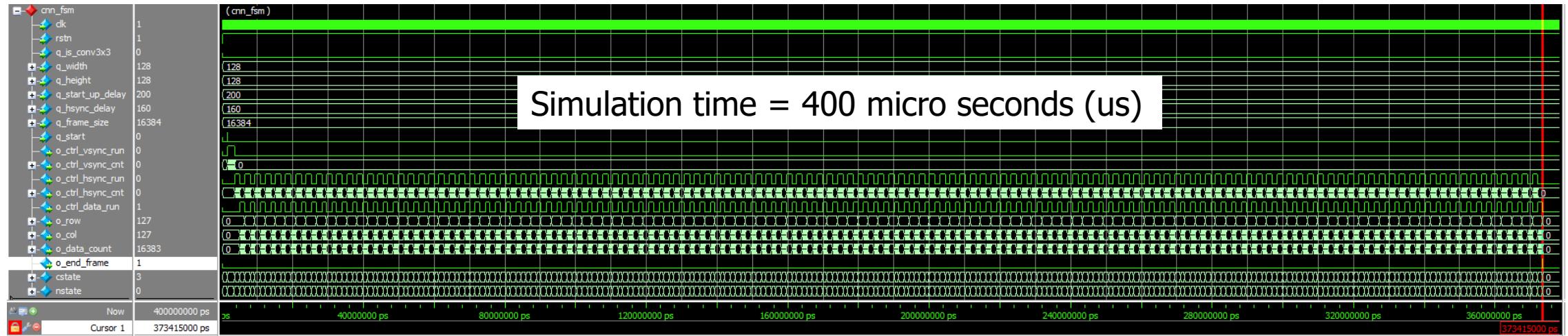
Configures a layer's configuration registers

Send a start signal

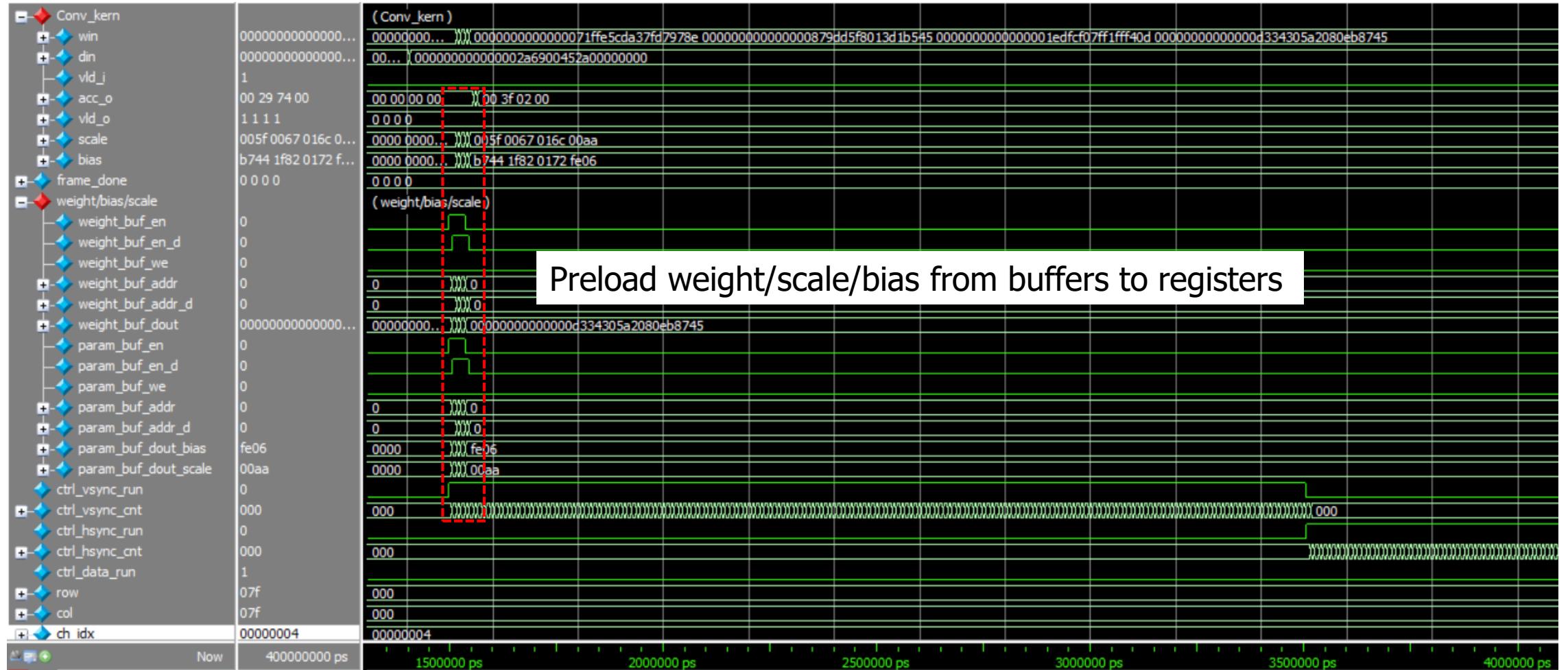
Waveform



Waveform: Finite state machines

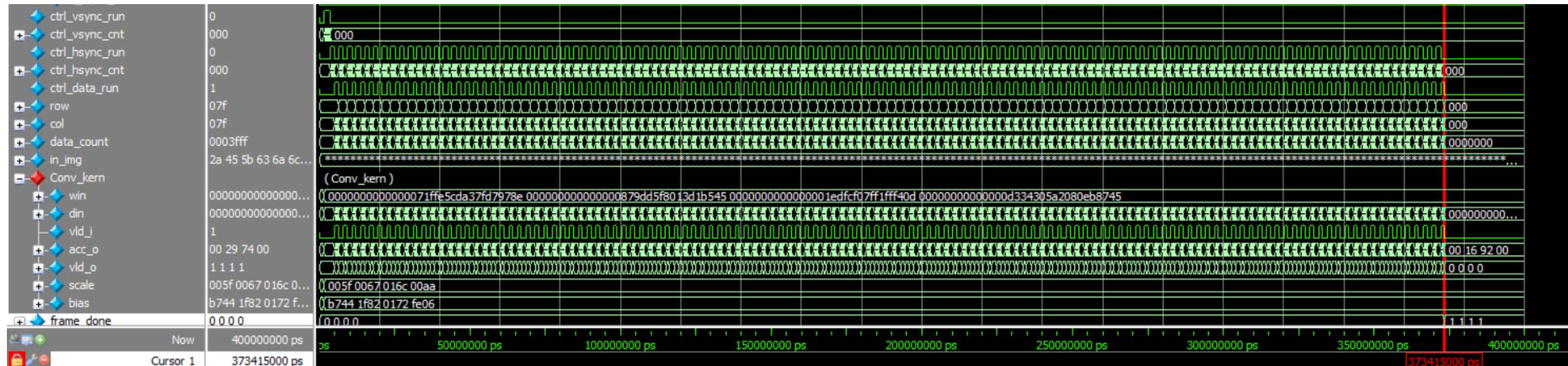


Weight/scale

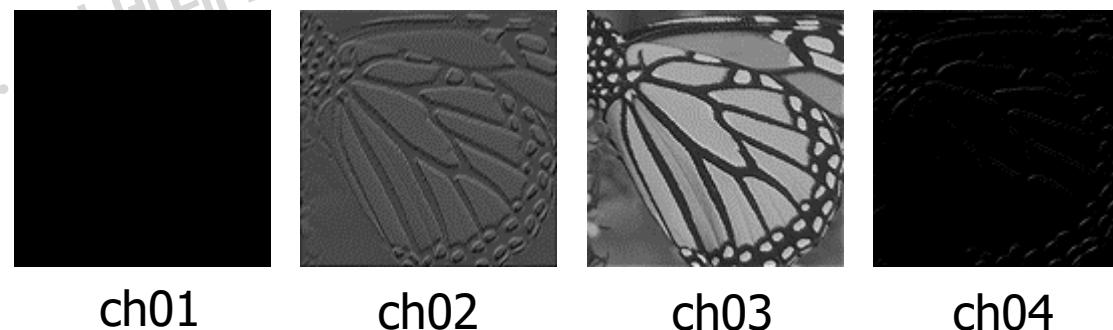


Simulation results

- Do simulation with time = 400 microseconds (us).



- Four image writer modules write the output results at the folder out/

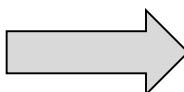


Verification

- Compare the S/W and H/W simulation results (check.hardware.results.m)
 - Load images at the folders out_sw/ and out/
 - Calculate the difference between two images.

```
for ch = 1:4
    % Output from the reference S/W
    im_sw = imread(sprintf('out_sw/ofmap_L01_ch%02d.bmp',ch));
    % Output from the H/W simulation
    im_hw = imread(sprintf('out/convout_layer01_ch%02d.bmp',ch));
    im_hw = im_hw(:,:,1);      % Gray image

    % Calculate the difference between S/W and H/W outputs
    img_diff = abs(single(im_hw) - single(im_sw));
    max_diff = max(img_diff(:));
    if(max_diff == 0)
        fprintf('Results of the channel %02d are same!\n', ch);
    else
        fprintf('ERROR: Results of the channel %02d are different!\n', ch);
        disp(max_diff);
        figure(ch)
        imshow(uint8(img_diff));
    end
end
Results of the channel 01 are same!
Results of the channel 02 are same!
Results of the channel 03 are same!
Results of the channel 04 are same!
>>
```



To do ...

- Complete the missing codes
 - Reuse cnn_fsm.v
 - cnn_accel.v
 - Reuse code for setting configuration registers
 - Reuse code to generate din, vld_i
 - Add codes for weight/bias/scale buffers
- Do a simulation with time = 400 us
- Show/capture the output results

Road map

Review

Reference Software

weight/bias/scale
buffers

Output buffers

Control path
(CONV3x3)

CNN Accelerator

- Why do we call `cnn_accel.v` as an CNN accelerator?
 - RISC-V: one ALU, one multiplier (optional).
- CNN accelerator
 - To: The number of convolutional kernels
 - T_i : The number of multipliers in a CONV kernel (`conv_kern.v`)

```
parameter Ti = 16;          // Each CONV kernel do 16 multipliers at the same time
parameter To = 4;           // Run 4 CONV kernels at the same time
//parameter To = 16;         // Run 16 CONV kernels at the same time
```

- The number of multipliers?

CNN Accelerator

- Use “generate” to create To instances of conv. kernels
 - To: The number of conv. kernels
 - Index “i” is used to select a specific output channel

```
//-----
// Computing units
//-----
generate
    genvar i;
    for (i=0; i<To; i=i+1) begin: u_conv_kern
        conv_kern u_conv_kern(
            ./*input          */clk(clk),
            ./*input          */rstn(rstn),
            ./*input          */is_last_layer(q_is_last_layer),
            ./*input [PARAM_BITS-1:0]*/scale(scale[i]),
            ./*input [PARAM_BITS-1:0]*/bias(bias[i]),
            ./*input [2:0]       */act_shift(q_act_shift),
            ./*input [4:0]       */bias_shift(q_bias_shift),
            ./*input          */is_conv3x3(q_is_conv3x3),           //0: 1x1, 1:3x3
            ./*input          */vld_i(vld_i),
            ./*input [N*WI-1:0] */win(win[i]),
            ./*input [N*WI-1:0] */din(din),
            ./*output [ACT_BITS-1:0] */acc_o(acc_o[i]),
            ./*output          */vld_o(vld_o[i])
        );
    end
endgenerate
```

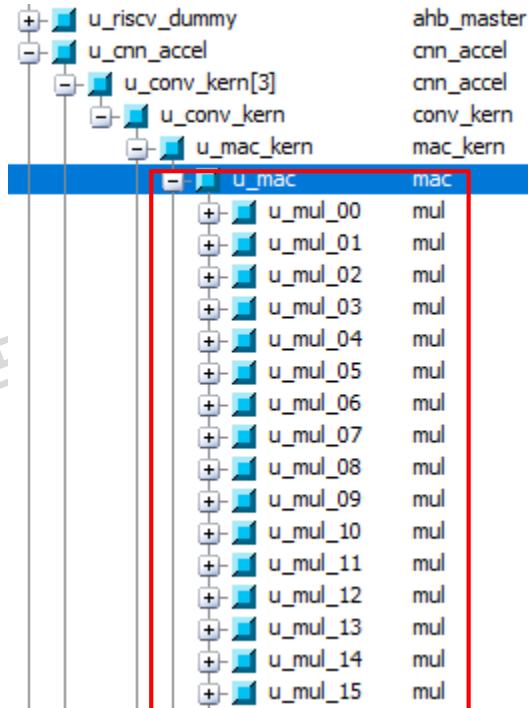
CNN Accelerator

- CNN accelerator
 - To: The number of convolutional kernels (output feature maps)
 - Last time, we generate 4 feature maps simultaneously.
 - Ti: The number of multipliers in a CONV kernel (conv_kern.v)



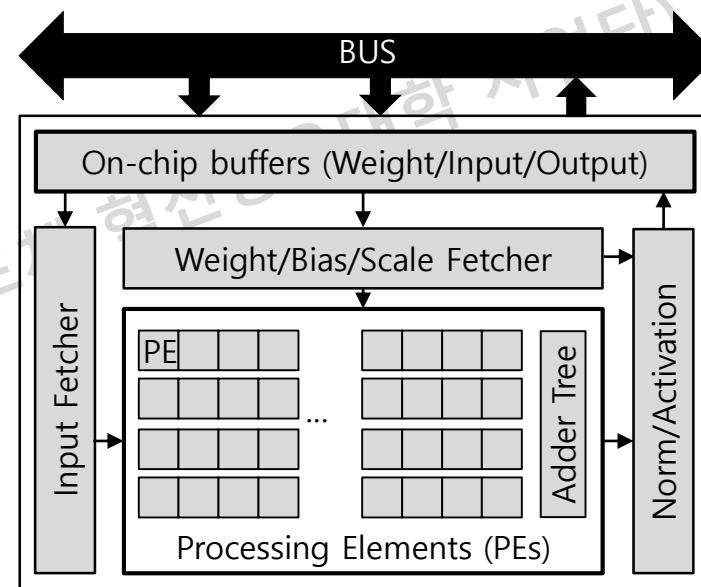
Convolution kernels

- CNN accelerator
 - To: The number of convolutional kernels
 - T_i : The number of multipliers in a CONV kernel (conv_kern.v)
 - The number of MACs in a MAC kernel, i.e. $T_i=N=16$



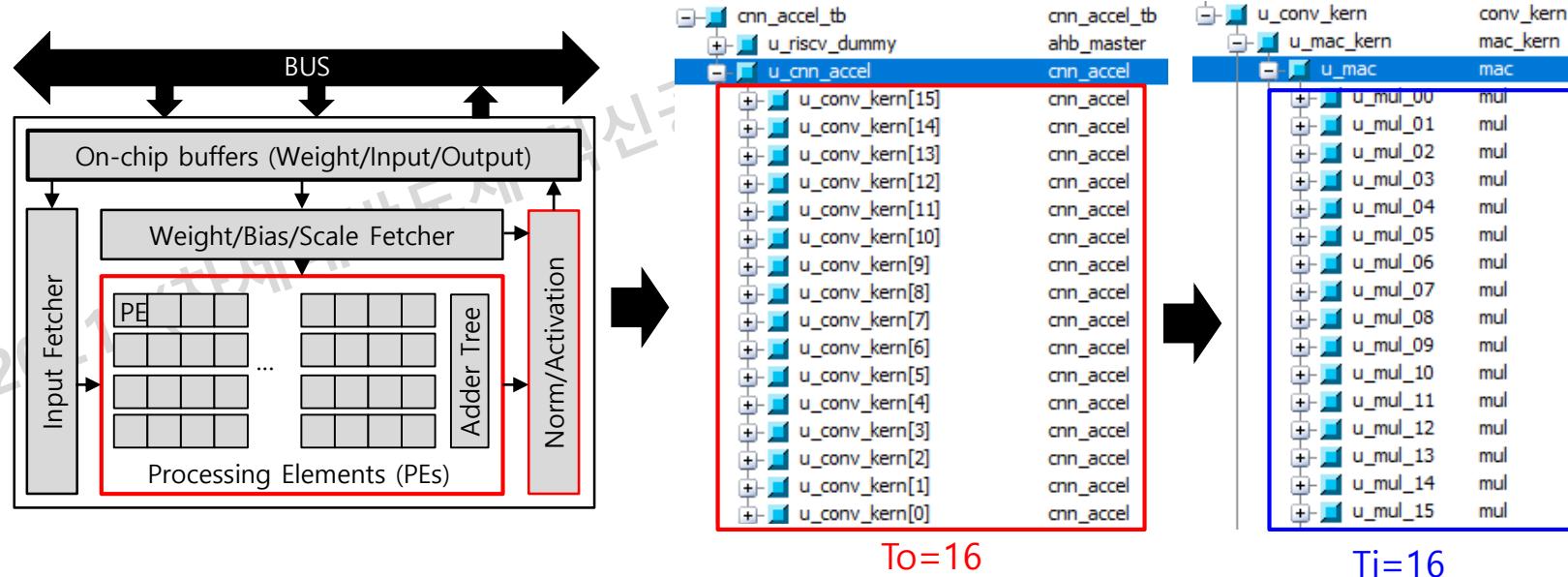
CNN accelerator

- Processing Element (PE) Array (conv_kern.v, mac_kern.v)
 - An array of PEs, a.k.a. MACs (multiplication and accumulation).
 - Perform convolution/activation/quantization operations.
- Buffers:
 - Input/output feature maps
 - Weight/bias/scale



Processing Elements

- Processing Element (PE) Array (conv_kern.v, mac_kern.v)
 - Perform convolution/activation/quantization operations.
 - To: The number of convolutional kernels (output feature maps)
 - Ti: The number of multipliers in a CONV kernel (conv_kern.v)
- The number of multipliers is : To \times Ti
 - 256 ($=16 \times 16$) multipliers run in parallel



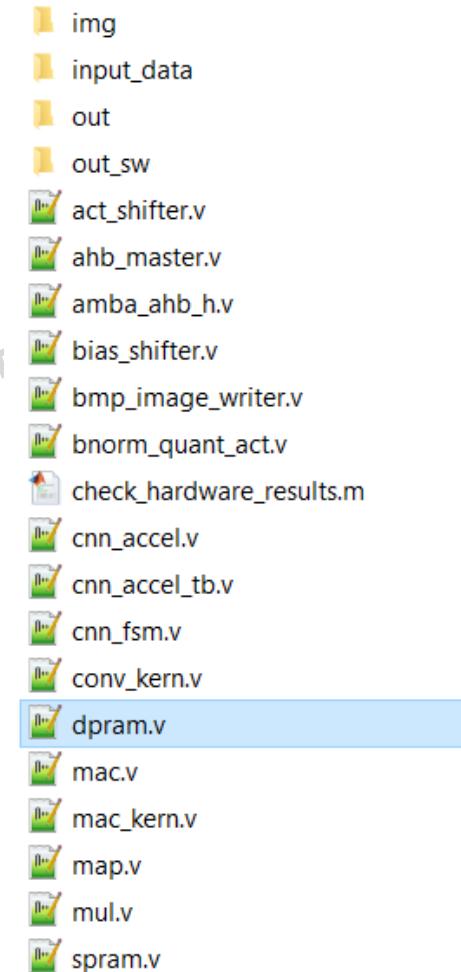
Lab 2: feature map buffers

- Lab 2:
 - Reuse cnn_fsm.v
 - cnn_accel.v
 - Reuse code for setting configuration registers
 - Reuse code to generate din, vld_i
 - Add buffers for weights, biases and scales.
 - Implement related logics
 - Do simulation with time = 1,200 us
 - Show/capture the output results

Copyright 2021. (차세대반도체 혁신공유대학 사업단) all rights reserved.

Code structure

- RTL file
 - Convolution kernel (conv_kern.v)
 - MAC kernel: mac_kern.v, mac.v, mul.v
 - Batch normalization and activation
 - bnorm_quant_act.v
 - bias_shifter.v, act_shifter.v
 - CNN accelerator IP
 - cnn_accel.v, cnn_fsm.v
 - AHB interface
 - amba_ahb_h.v, ahb_master.v, map.v
 - BMP image writer (bmp_image_writer.v)
 - Test bench: cnn_accel_tb.v
 - Single port block ram (spram.v)
 - Dual port block ram (dpram.v)
- img/: input image in hex file
- out/: output results by H/W simulation
- out_sw/: outputs by S/W simulation
- input_data/: weights, scales, biases files



Dual port block ram (dpram.v)

- Dual-port block ram (dpram.v)
 - Port A: Write only
 - Read/write request (ena)
 - Write enable (wea)
 - Address (addrA)
 - Write data (dina)
 - Port B: Read only
 - Read/write request (enb)
 - Address (addrB)
 - Read data (dob)
 - Memory cell: ram
- Parameters
 - W_DATA: word size
 - N_WORD: # of words
 - W_WORD: address size
 - FILENAME: initialized file
 - N_DELAY

```
module dpram(  
    clk ,  
    ena , // portA: enable  
    wea , // portA: primary synchronous write enable  
    addrA , // portA: address for read/write  
    enb , // portB: enable  
    addrB , // portB: address for read  
    dia , // portA: primary data input  
    dob // portB: primary data output  
);  
parameter W_DATA = 8;  
parameter N_WORD = 512;  
parameter W_WORD = $clog2(N_WORD);  
parameter FILENAME = "";  
parameter N_DELAY = 1;  
  
input clk; // clock input  
input ena; // primary enable  
input wea; // primary synchronous write enable  
input [W_WORD-1:0] addrA; // address for read/write  
input enb; // read port enable  
input [W_WORD-1:0] addrB; // address for read  
input [W_DATA-1:0] dia; // primary data input  
output [W_DATA-1:0] dob; // primary data output  
// share memory  
reg [W_DATA-1:0] ram[N_WORD-1:0];
```

Dual port block ram (dpram.v)

- Write operation
 - If (ena & wea)
 - $\text{ram}[\text{addr}] \leftarrow \text{dia}$
- Read operation ($N_DELAY=1$)
 - If (enb)
 - $\text{rdata} \leftarrow \text{ram}[\text{addrb}]$
 - $\text{dob} \leftarrow \text{rdata}$
- Dual-port
 - Allow read/write operations at the same time

```
// write port |  
always @ (posedge clk)  
begin: write  
    if(ena)  
        begin  
            if (wea)  
                ram[addr] <= dia;  
        end  
    end  
  
generate  
    if(N_DELAY == 1) begin: delay_1  
        reg [W_DATA-1:0] rdata; // primary data output  
        // read port  
        always @ (posedge clk)  
        begin: read  
            if(enb)  
                rdata <= ram[addrb];  
        end  
        assign dob = rdata;  
    end  
    else begin: delay_n  
        reg [N_DELAY*W_DATA-1:0] rdata_r;  
        always @ (posedge clk)  
        begin: read  
            if(enb)  
                rdata_r[0+:W_DATA] <= ram[addrb];  
        end  
  
        always @ (posedge clk) begin: delay  
            integer i;  
            for(i = 0; i < N_DELAY-1; i = i+1)  
                if(enb)  
                    rdata_r[(i+1)*W_DATA+:W_DATA] <= rdata_r[i*W_DATA+:W_DATA];  
        end  
        assign dob = rdata_r[(N_DELAY-1)*W_DATA+:W_DATA];  
    end  
endgenerate
```

Feature map buffers

- Dual buffers are used to store feature maps
 - Two buffers are identical.
 - Assume Port B's signals are OPEN.

```
//-----
// Update the output buffers.
//-----
always@(posedge clk, negedge rstn) begin
    if(!rstn) begin
        pixel_count <= 0;
        layer_done <= 0;
        out_buff_sel <= 1'b0;
    end else begin
        if(q_start) begin
            pixel_count <= 0;
            layer_done <= 0;
        end
        else begin
            if(vld_o[0]) begin
                if(pixel_count == q_frame_size-1) begin
                    pixel_count <= 0;
                    layer_done <= 1'b1;
                    out_buff_sel <= !out_buff_sel;
                end
                else begin
                    pixel_count <= pixel_count + 1;
                end
            end
        end
    end
end
```

```
//-----
// output buffers.
//-----
wire [ACT_BITS*To-1:0] all_acc_o = {
    acc_o[15], acc_o[14], acc_o[13], acc_o[12],
    acc_o[11], acc_o[10], acc_o[ 9], acc_o[ 8],
    acc_o[ 7], acc_o[ 6], acc_o[ 5], acc_o[ 4],
    acc_o[ 3], acc_o[ 2], acc_o[ 1], acc_o[ 0]
};
dpram #(W_DATA(To*ACT_BITS), .W_WORD(FRAME_SIZE_W), .N_WORD(FRAME_SIZE))
u_fmap_buf_01(
    .clk  (clk  ),
    .ena  ((!out_buff_sel) & vld_o[0]),
    .wea  ((!out_buff_sel) & vld_o[0]),
    .addr_a (pixel_count ),
    .enb  (*OPEN*),
    .addr_b (*OPEN*),
    .dia  (all_acc_o),
    .dob  (*OPEN*)
);

dpram #(W_DATA(To*ACT_BITS), .W_WORD(FRAME_SIZE_W), .N_WORD(FRAME_SIZE))
u_fmap_buf_02(
    .clk  (clk  ),
    .ena  (out_buff_sel & vld_o[0]),
    .wea  (out_buff_sel & vld_o[0]),
    .addr_a (pixel_count ),
    .enb  (*OPEN*),
    .addr_b (*OPEN*),
    .dia  (all_acc_o),
    .dob  (*OPEN*)
);
```

Weight/bias/scale buffers

- Define three spram instances for weight, bias and scale buffer
 - Parameters
 - INIT_FILE
 - EN_LOAD_INIT_FILE (1)
 - W_DATA, N_WORD, W_WORD
 - Ports
 - en, addr, din, we, dout
 - To do ...
 - Complete the missing codes for bias and scale buffers

```
// Weight buffer
spram #(.INIT_FILE("input_data/all_conv_weights.hex"),
        .EN_LOAD_INIT_FILE(EN_LOAD_INIT_FILE),
        .W_DATA(Ti*WI), .W_WORD(W_CELL), .N_WORD(N_CELL))
u_buf_weight(
    .clk (clk)           , // Clock input
    .en  (weight_buf_en ), // RAM enable (select)
    .addr(weight_buf_addr), // Address input(word addressing)
    .din /*unused*/      , // Data input
    .we   (weight_buf_we ), // Write enable
    .dout(weight_buf_dout) // Data output
);
// Bias buffer
spram #(.INIT_FILE(/*Insert your code*/),
        .EN_LOAD_INIT_FILE(EN_LOAD_INIT_FILE),
        .W_DATA(/*Insert your code*/), .W_WORD(W_CELL_PARAM), .N_WORD(N_CELL_PARAM))
u_buf_bias(
    .clk (clk)           , // Clock input
    .en  (param_buf_en  ), // RAM enable (select)
    .addr(param_buf_addr), // Address input(word addressing)
    .din /*unused*/      , // Data input
    .we   (param_buf_we ), // Write enable
    .dout(param_buf_dout_bias) // Data output
);
// Scale buffer
spram #(.INIT_FILE(/*Insert your code*/),
        .EN_LOAD_INIT_FILE(EN_LOAD_INIT_FILE),
        .W_DATA(/*Insert your code*/), .W_WORD(W_CELL_PARAM), .N_WORD(N_CELL_PARAM))
u_buf_scale(
    .clk (clk)           , // Clock input
    .en  /*Insert your code*/, // RAM enable (select)
    .addr/*Insert your code*/, // Address input(word addressing)
    .din /*unused*/      , // Data input
    .we   /*Insert your code*/, // Write enable
    .dout/*Insert your code*/ // Data output
);
```

Access weight/bias/scale buffers

- Assume that weight, scale and bias registers are updated during frame synchronization
 - VSYNC
 - Request address is updated from the VSYNC counter `ctrl_vsync_cnt`
 - CONV1x1
 - To requests are needed
 - **To do ...**
 - **Complete the missing codes for bias and scale requests**

```
//-----  
// Weights, biases, scales  
//-----  
// Weight  
always@(*) begin  
    weight_buf_en = 1'b0;  
    weight_buf_we = 1'b0;  
    weight_buf_addr = {W_CELL{1'b0}};  
    if(ctrl_vsync_run) begin  
        if(!q_is_conv3x3) begin // Conv1x1  
            if(ctrl_vsync_cnt < To) begin  
                weight_buf_en = 1'b1;  
                weight_buf_we = 1'b0;  
                weight_buf_addr = ctrl_vsync_cnt[W_CELL-1:0];  
            end  
        end  
        else begin // Conv3x3  
            // Insert your code  
        end  
    end  
end  
  
// Scale/bias  
always@(*) begin  
    param_buf_en = 1'b0;  
    param_buf_we = 1'b0;  
    param_buf_addr = {W_CELL{1'b0}};  
    if(ctrl_vsync_run) begin  
        if(ctrl_vsync_cnt < To) begin  
            //param_buf_en = /*Insert your code*/;  
            //param_buf_we = /*Insert your code*/;  
            //param_buf_addr = /*Insert your code*/;  
        end  
    end  
end
```

Access weight/bias/scale buffers

- One-cycle delay signals
 - Enable: *_buf_en_d
 - Address: *_buf_addr_d
- Win, scale, bias registers are updated when outputs from their buffers are read out.
- Example
 - $\text{win}[\text{weight_buf_addr_d}] \leftarrow \text{weight_buf_dout}$

```
// one-cycle delay
always@(posedge clk, negedge rstn)begin
  if(~rstn) begin
    weight_buf_en_d  <= 1'b0;
    weight_buf_addr_d <= {W_CELL{1'b0}};
    param_buf_en_d   <= 1'b0;
    param_buf_addr_d <= {W_CELL{1'b0}};
  end
  else begin
    weight_buf_en_d  <= weight_buf_en;
    weight_buf_addr_d <= weight_buf_addr;
    param_buf_en_d   <= param_buf_en;
    param_buf_addr_d <= param_buf_addr;
  end
end

always@(posedge clk, negedge rstn)begin
  if(~rstn) begin
    for(ch_idx = 0; ch_idx < To; ch_idx=ch_idx+1) begin
      win[ch_idx]  <= {(Ti*WI){1'b0}};
      bias[ch_idx] <= {PARAM_BITS{1'b0}};
      scale[ch_idx] <= {PARAM_BITS{1'b0}};
    end
  end
  else begin
    // Weight
    if(weight_buf_en_d)
      win[weight_buf_addr_d] <= weight_buf_dout;
    // Scale/bias
    /*Insert your code*/
  end
end
```

Test bench (cnn_accel_tb.v)

- Test bench
 - Master: RISC-V (ahb_master.v)
 - Slave: A CNN accelerator IP (cnn_accel.v)

```
'timescale 1ns / 100ps
`include "amba_ahb_h.v"
`include "map.v"

module cnn_accel_tb;
parameter W_ADDR=32;
parameter W_DATA=32;
parameter IMG_PIX_W = 8;

//parameter WIDTH    = 768,
//      HEIGHT   = 512,
parameter WIDTH    = 128,
      HEIGHT   = 128,
      START_UP_DELAY = 200,
      HSYNC_DELAY = 160,
      FRAME_SIZE = WIDTH * HEIGHT;
localparam W_SIZE   = 12;           // Max 4K QHD (3840x1920).
localparam W_FRAME_SIZE = 2 * W_SIZE + 1; // Max 4K QHD (3840x1920).
localparam W_DELAY   = 12;

parameter N_LAYER = 3;
parameter Ti = 16; // Each CONV kernel do 16 multipliers at the same time
//parameter To = 4; // Run 4 CONV kernels at the same time
parameter To = 16; // Run 16 CONV kernels at the same time
parameter N = 16;
// Inputs
reg HCLK;
reg HRESETn;
```

Master

```
//-----
// Master
//-----

ahb_master u_riscv_dummy(
    .HRESETn      (HRESETn),
    .HCLK         (HCLK),
    ..i_HRDATA   (w_RISC2AHB_mst_HRDATA),
    ..i_HRESP    (w_RISC2AHB_mst_HRESP),
    ..i_HREADY   (w_RISC2AHB_mst_HREADY),
    ..o_HADDR    (w_RISC2AHB_mst_HADDR),
    ..o_HWDATA   (w_RISC2AHB_mst_HWDATA),
    ..o_HWRITE   (w_RISC2AHB_mst_HWRITE),
    ..o_HSIZE    (w_RISC2AHB_mst_HSIZE),
    ..o_HBURST   (w_RISC2AHB_mst_HBURST),
    ..o_HTRANS   (w_RISC2AHB_mst_HTRANS)
);
```

```
//-----
// Slave
//-----

cnn_accel u_cnn_accel (
    .HCLK(HCLK),
    .HRESETn(HRESETn),
    .sl_HREADY(1'b1),
    .sl_HSEL(1'b1),
    .sl_HTRANS(w_RISC2AHB_mst_HTRANS),
    .sl_HBURST(w_RISC2AHB_mst_HBURST),
    .sl_HSIZE(w_RISC2AHB_mst_HSIZE),
    .sl_HADDR(w_RISC2AHB_mst_HADDR),
    .sl_HWRITE(w_RISC2AHB_mst_HWRITE),
    .sl_HWDATA(w_RISC2AHB_mst_HWDATA),
    .out_sl_HREADY(w_RISC2AHB_mst_HREADY),
    .out_sl_HRESP(w_RISC2AHB_mst_HRESP),
    .out_sl_HRDATA(w_RISC2AHB_mst_HRDATA)
);
```

Slave

Signals and test vector (cnn_accel_tb.v)

- Define internal register for a three-layer network
- Generate clock, reset signals
- Prepare a test vector

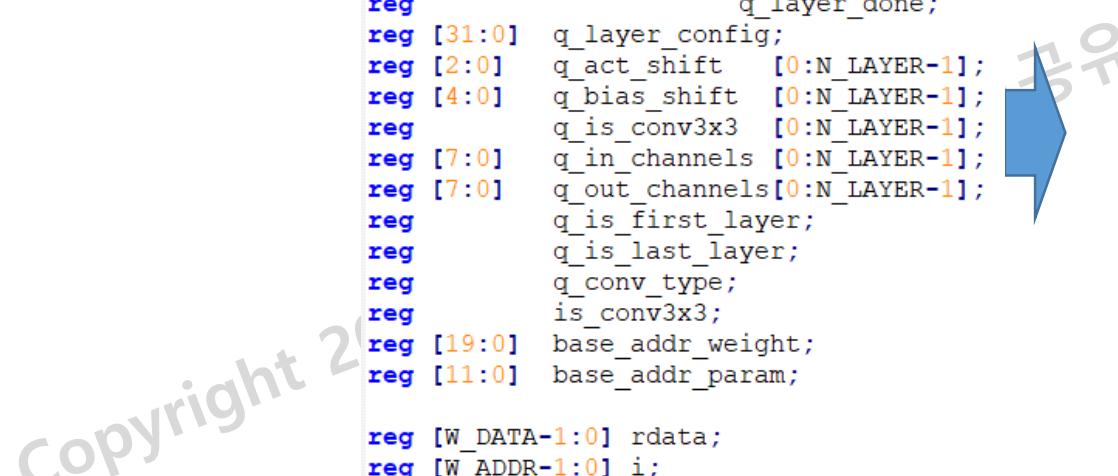
```
reg [W_SIZE-1 :0]          q_width;
reg [W_SIZE-1 :0]          q_height;
reg [W_DELAY-1:0]          q_start_up_delay;
reg [W_DELAY-1:0]          q_hsync_delay;
reg [W_FRAME_SIZE-1:0]      q_frame_size;
reg                      q_layer_start;
reg [3:0]                  q_layer_index;
reg                      q_layer_done;
reg [31:0]     q_layer_config;
reg [2:0]      q_act_shift [0:N_LAYER-1];
reg [4:0]      q_bias_shift [0:N_LAYER-1];
reg          q_is_conv3x3 [0:N_LAYER-1];
reg [7:0]      q_in_channels [0:N_LAYER-1];
reg [7:0]      q_out_channels[0:N_LAYER-1];
reg          q_is_first_layer;
reg          q_is_last_layer;
reg          q_conv_type;
reg          is_conv3x3;
reg [19:0]    base_addr_weight;
reg [11:0]    base_addr_param;

reg [W_DATA-1:0] rdata;
reg [W_ADDR-1:0] i;
reg image_load_done;
integer idx;
```

```
//-----
// Test vectors
//-----
// Clock
parameter p = 10; //100MHz
initial begin
  HCLK = 1'b0;
  forever #(p/2) HCLK = ~HCLK;
end

initial begin
  // Initialize Inputs
  HRESETn = 0;
  q_width           = WIDTH;
  q_height          = HEIGHT;
  q_start_up_delay = START_UP_DELAY;
  q_hsync_delay     = HSYNC_DELAY;
  q_frame_size      = FRAME_SIZE;
  q_layer_index     = 4'd0;
  q_layer_done       = 1'b0;
  q_is_first_layer   = 1'b0;
  q_is_last_layer    = 1'b0;
  q_layer_config     = 32'h0;

  // Define Network's parameters
  q_bias_shift[0] = 9 ; q_act_shift[0] = 7; q_is_conv3x3[0] = 0;
  q_bias_shift[1] = 17; q_act_shift[1] = 7; q_is_conv3x3[1] = 1;
  q_bias_shift[2] = 17; q_act_shift[2] = 7; q_is_conv3x3[2] = 1;
  // Loop/Layer index
  idx = 0;
```



Test case: Common parameters

```
// Loop/Layer index  
idx = 0;  
  
// Weight/bias/Scale base addresses  
base_addr_weight = 0;  
base_addr_param = 0;  
  
// Initialize RISCV dummy core  
u_riscv_dummy.task_AHBinit();  
  
// Memory  
rdata = 0;  
i = 0;  
image_load_done = 0;
```

RISC-V configures a layer's configuration registers

```
#(p/2) HRESETn = 1;  
*****  
// CNN Accelerator configuration  
*****  
#(100*p)  
#(4*p) @ (posedge HCLK) u_riscv_dummy.task_AHBwrite(`CNN_ACCEL_FRAME_SIZE , q_frame_size );  
#(4*p) @ (posedge HCLK) u_riscv_dummy.task_AHBwrite(`CNN_ACCEL_WIDTH_HEIGHT , {q_height&16'hFFFF,q_width&16'hFFFF});  
#(4*p) @ (posedge HCLK) u_riscv_dummy.task_AHBwrite(`CNN_ACCEL_DELAY_PARAMS , {q_hsync_delay,q_start_up_delay});  
....
```

Test case: Layer-specific parameters

```
for(idx = 0; idx < N_LAYER; idx=idx+1) begin
    q_layer_index      = idx;
    q_is_last_layer   = (idx == N_LAYER-1)?1'b1:1'b0;
    q_is_first_layer  = (idx == 0) ? 1'b1: 1'b0;
    is_conv3x3         = 0 /*q_is_conv3x3[idx]*//; // Dummy code: Assume all layers are conv1x1
    q_layer_config     = {q_act_shift[idx], q_bias_shift[idx], q_layer_index, q_is_last_layer, is_conv3x3, q_is_last_layer, q_is_first_layer};
    #(4*p) @ (posedge HCLK) u_riscv_dummy.task_AHBwrite(`CNN_ACCEL_BASE_ADDRESS, {base_addr_param&12'hFFF,base_addr_weight&20'hFFFF});
    #(4*p) @ (posedge HCLK) u_riscv_dummy.task_AHBwrite(`CNN_ACCEL_LAYER_CONFIG, q_layer_config);
    // Start a frame
    #(4*p) @ (posedge HCLK) u_riscv_dummy.task_AHBwrite(`CNN_ACCEL_LAYER_START , 1'b1 );
    #(4*p) @ (posedge HCLK) u_riscv_dummy.task_AHBwrite(`CNN_ACCEL_LAYER_START , 1'b0 );

    // Polling
    while(!q_layer_done) begin
        #(128*p) @ (posedge HCLK) u_riscv_dummy.task_AHBread(`CNN_ACCEL_LAYER_DONE,q_layer_done);
    end
    #(128*p) @ (posedge HCLK) $display("T=%03t ns: Layer %0d done!!!\n", $realtime/1000, idx+1);
    // Reset q_layer_done
    q_layer_done = 0;

    // Update the base addresses
    if(q_is_conv3x3[idx]) begin
        base_addr_weight  = base_addr_weight + (Ti*To*9)/N;
        base_addr_param   = base_addr_param + To;
    end
    else begin
        base_addr_weight  = base_addr_weight + To;
        base_addr_param   = base_addr_param + To;
    end
end
```

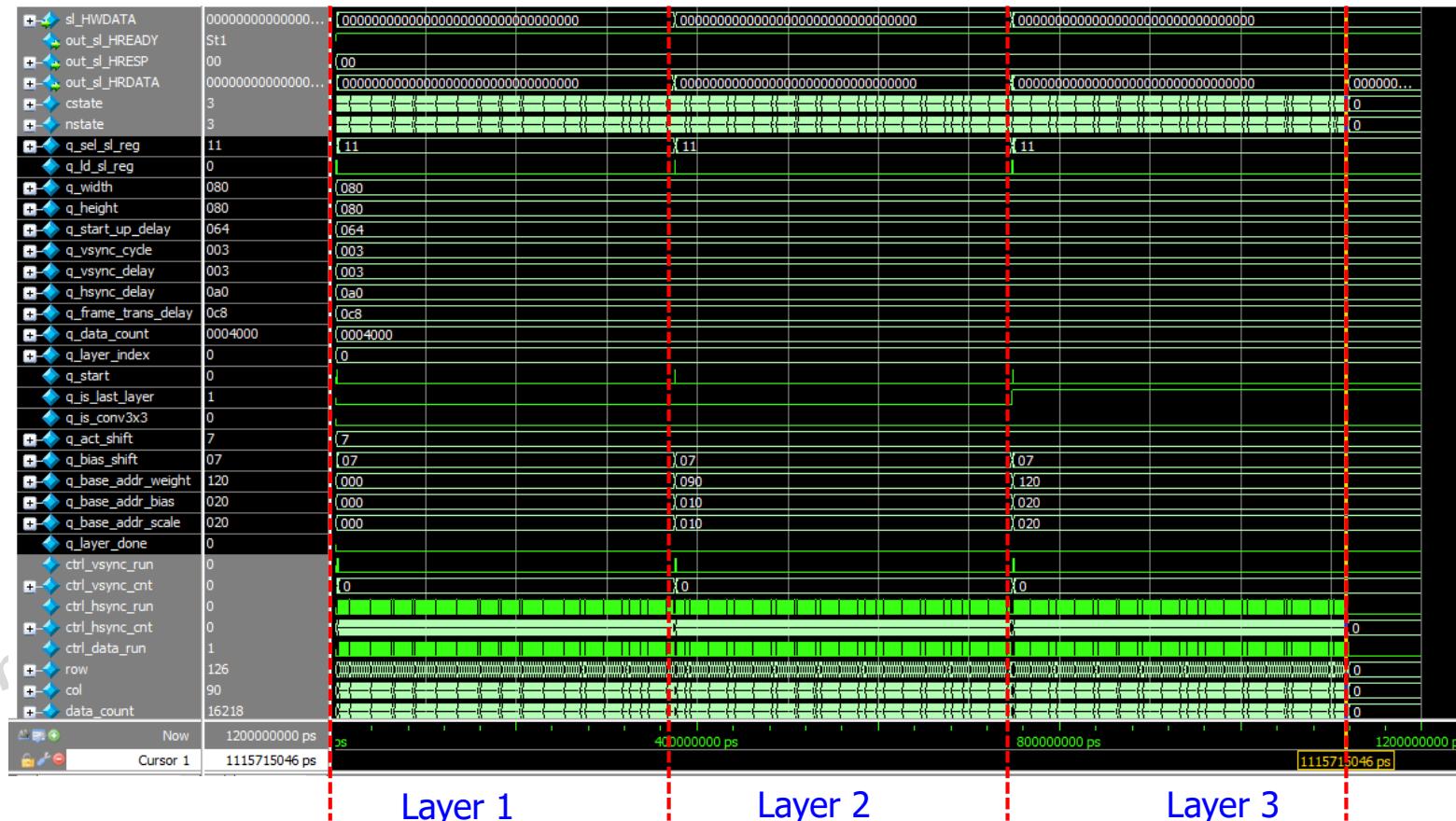
Configures a layer's configuration registers

Wait until the layer completes operations

Change the offset addresses

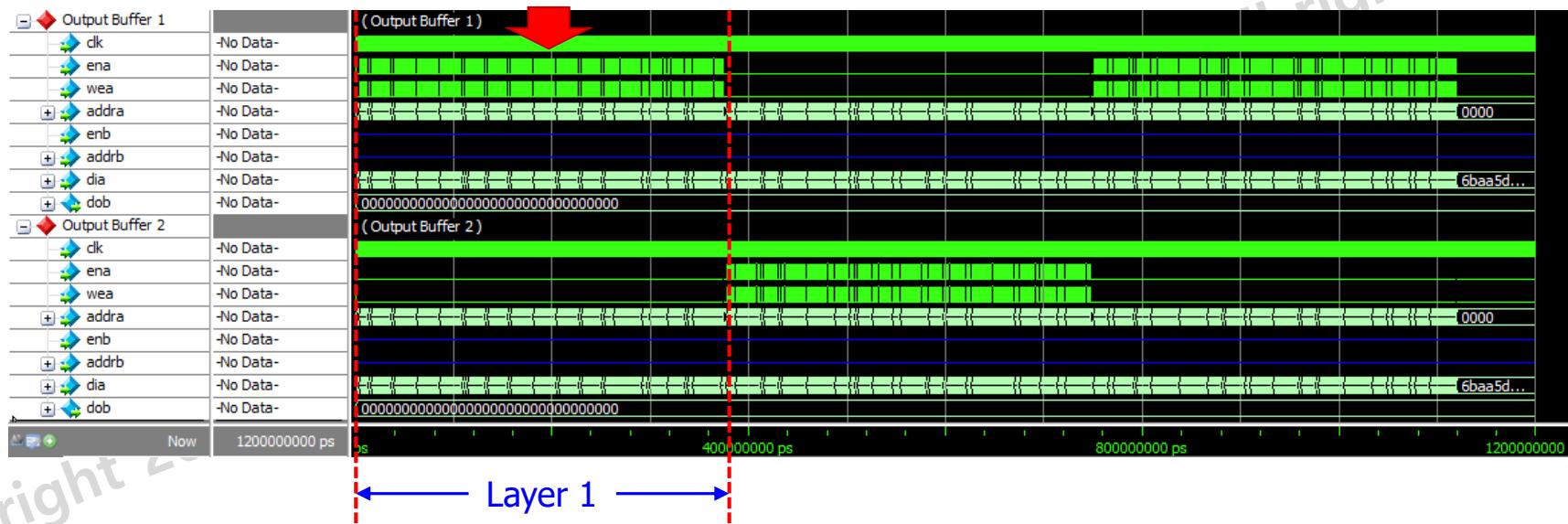
Waveform

- Do a simulation with time = 1200us
 - Note that: all layers are configured as conv1x1



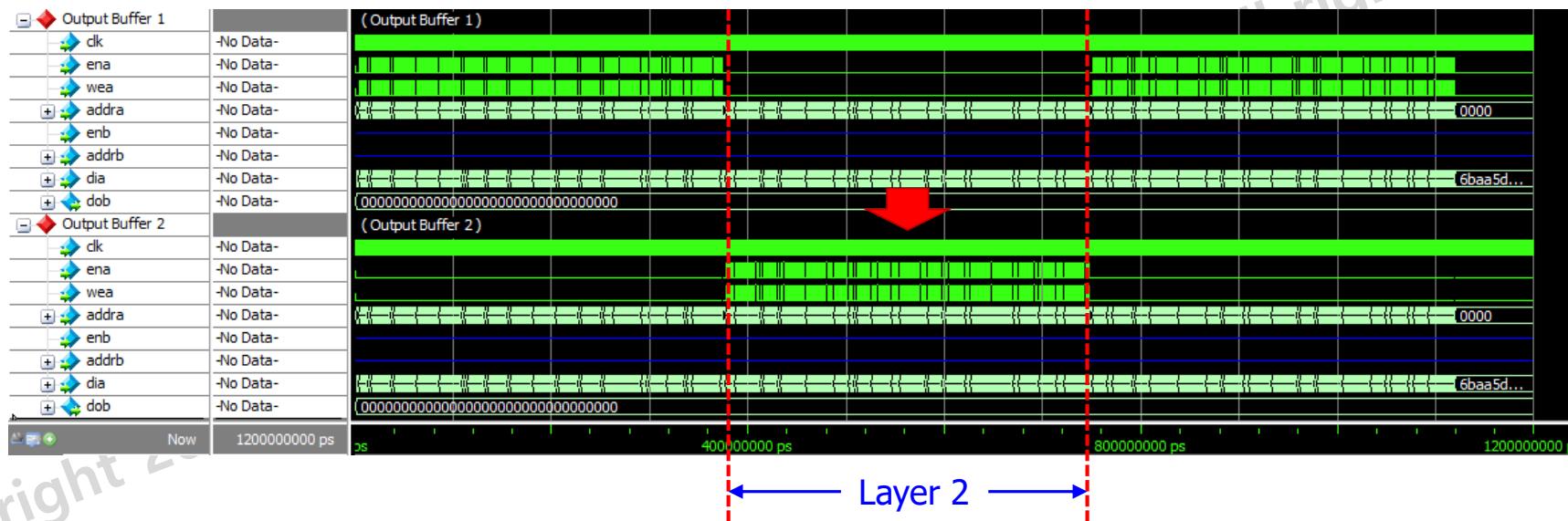
Waveform

- In Layer 1, buffer 1 is selected for storing the output feature map
 - Enable and Write signals (ena and wea) are ONE
 - Assume Port B's signals are OPEN.



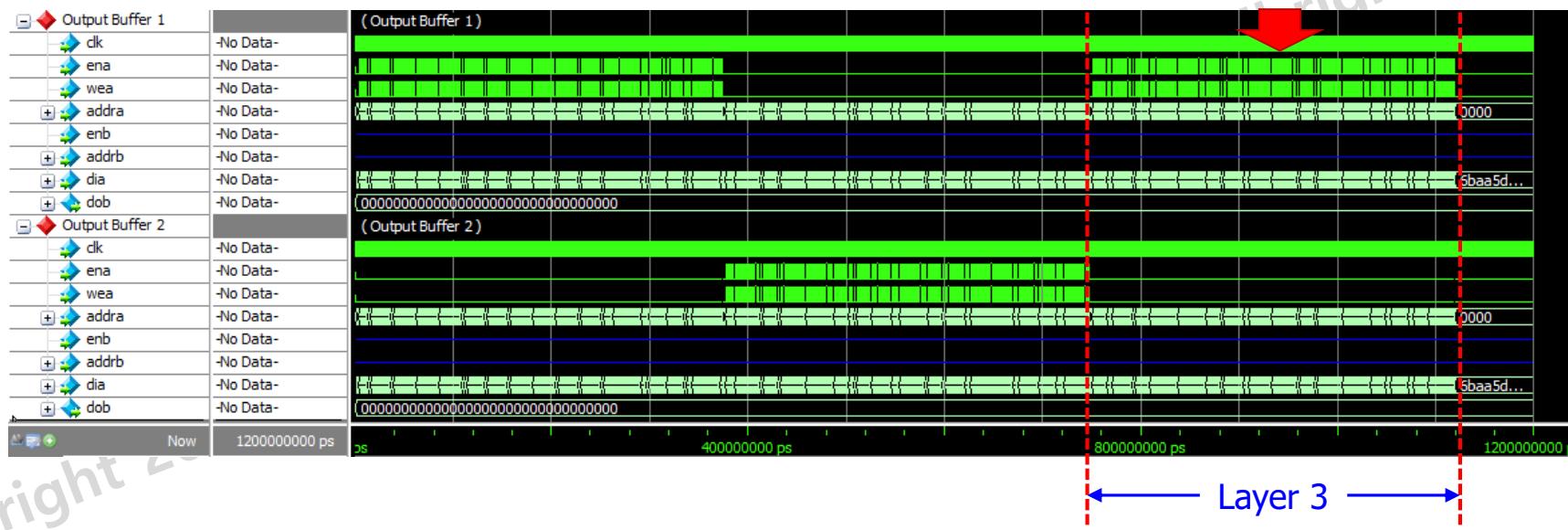
Waveform

- In Layer 2, buffer 2 is selected for storing the output feature map
 - Enable and Write signals (ena and wea) are ONE
 - Assume Port B's signals are OPEN.



Waveform

- In Layer 3, buffer 1 is selected for storing the output feature map
 - Enable and Write signals (ena and wea) are ONE
 - Assume Port B's signals are OPEN.



To do ...

- Complete the missing codes
 - Reuse cnn_fsm.v
 - cnn_accel.v
 - Reuse code for setting configuration registers
 - Reuse code to generate din, vld_i
 - Add codes for weight/bias/scale buffers
- Do simulation with time = 1,200 us
- Show/capture the output results
- Can you guess why we use dual buffers here?

Road map

Review

Reference Software

weight/bias/scale
buffers

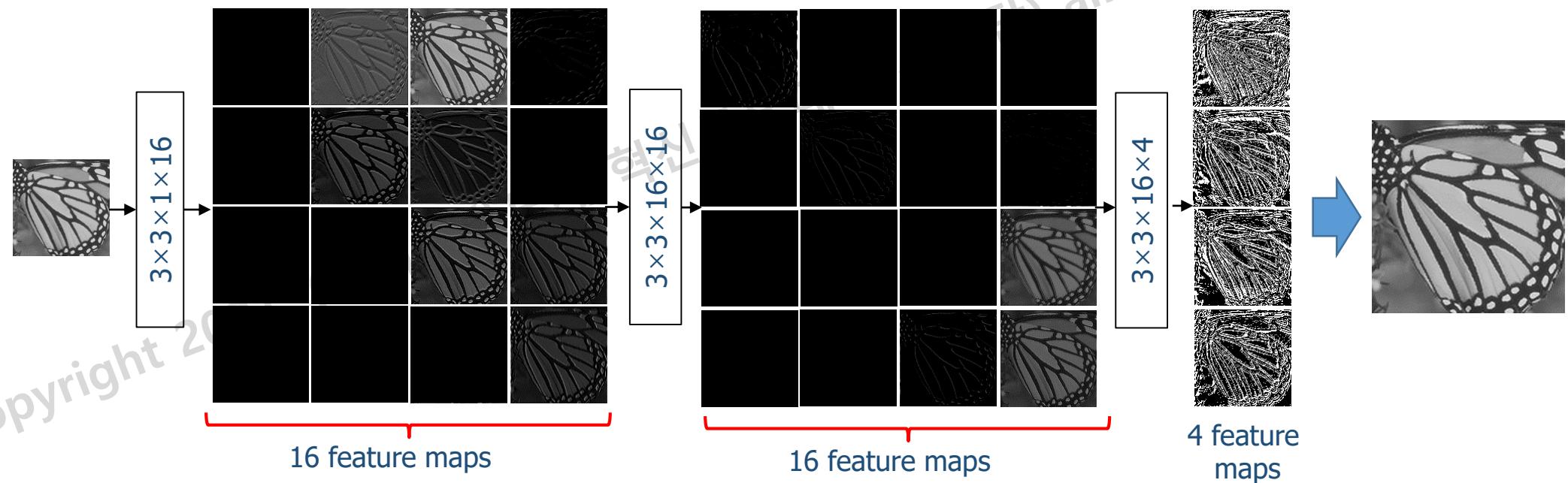
Output buffers

Control path
(CONV3x3)

Sim-ESPCN

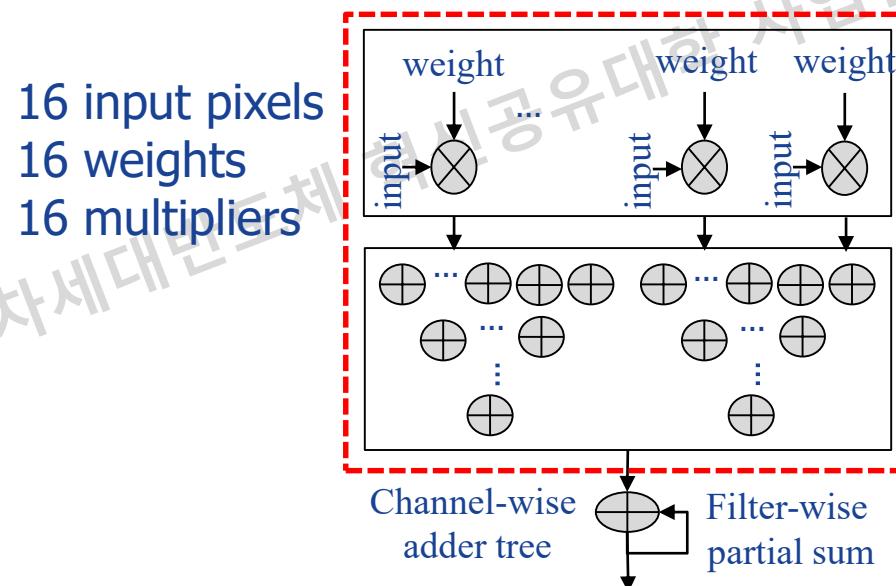
- Sim-ESPCN has three CONV layers

Layer	Filter size	No. of input channels	No. of output channels	Input feature maps	Output feature maps
1	3×3	1	16	$128 \times 128 \times 1$	$128 \times 128 \times 16$
2	3×3	16	16	$128 \times 128 \times 16$	$128 \times 128 \times 16$
3	3×3	16	4	$128 \times 128 \times 16$	$128 \times 128 \times 4$



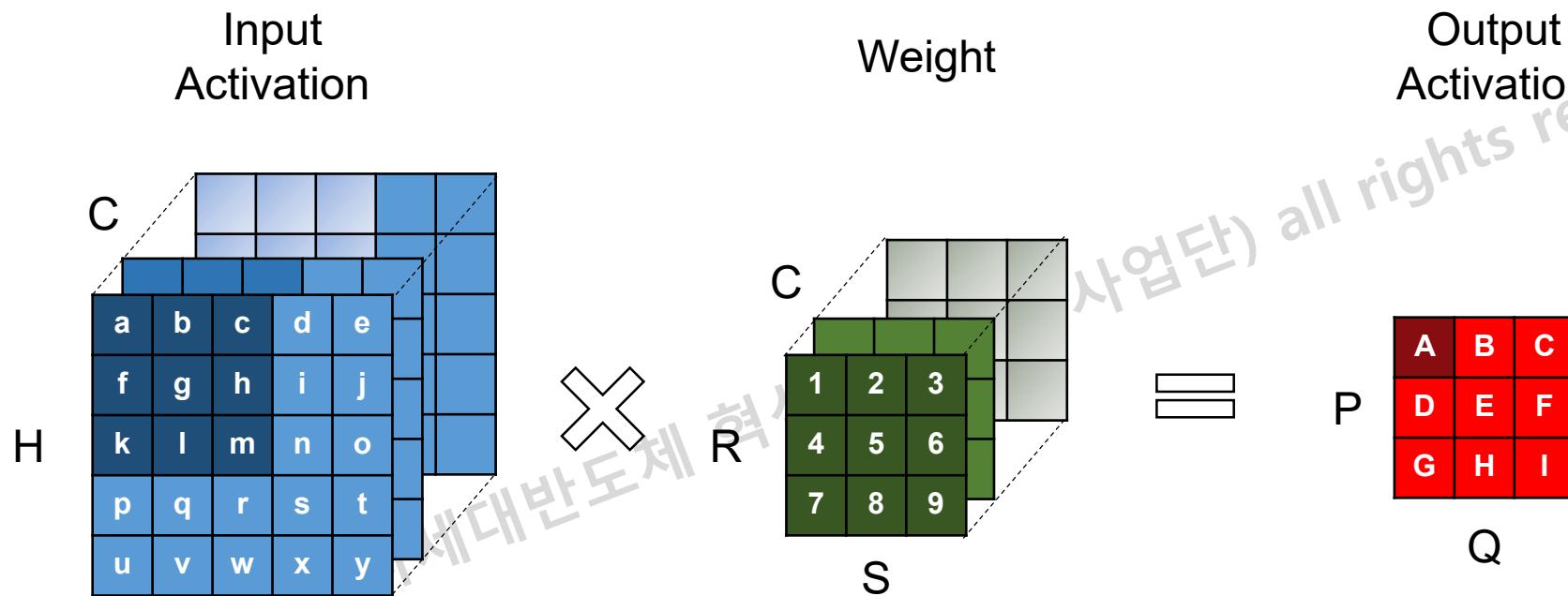
Mapping

- The required number of multiplication operations (num_ops) to calculate **ONE output pixel**
 - Layer 1: num_ops = 9 ($=3 \times 3 \times 1$)
 - Layer 2 and 3: num_ops = 144 ($= 3 \times 3 \times 16$).
- H/W module: N = 16
 - Layer 1: num_ops < N \Rightarrow calculating an output pixel needs 1 cycles
 - Layer 2, 3: num_ops > N \Rightarrow calculating an output pixel needs 9 cycles



Convolution: 3x3

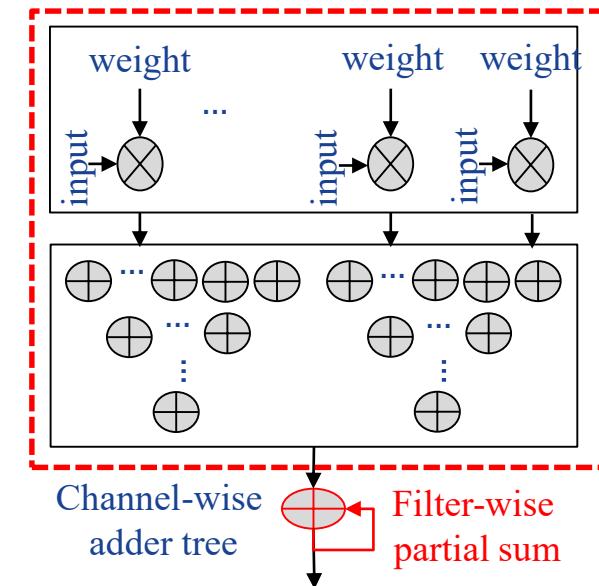
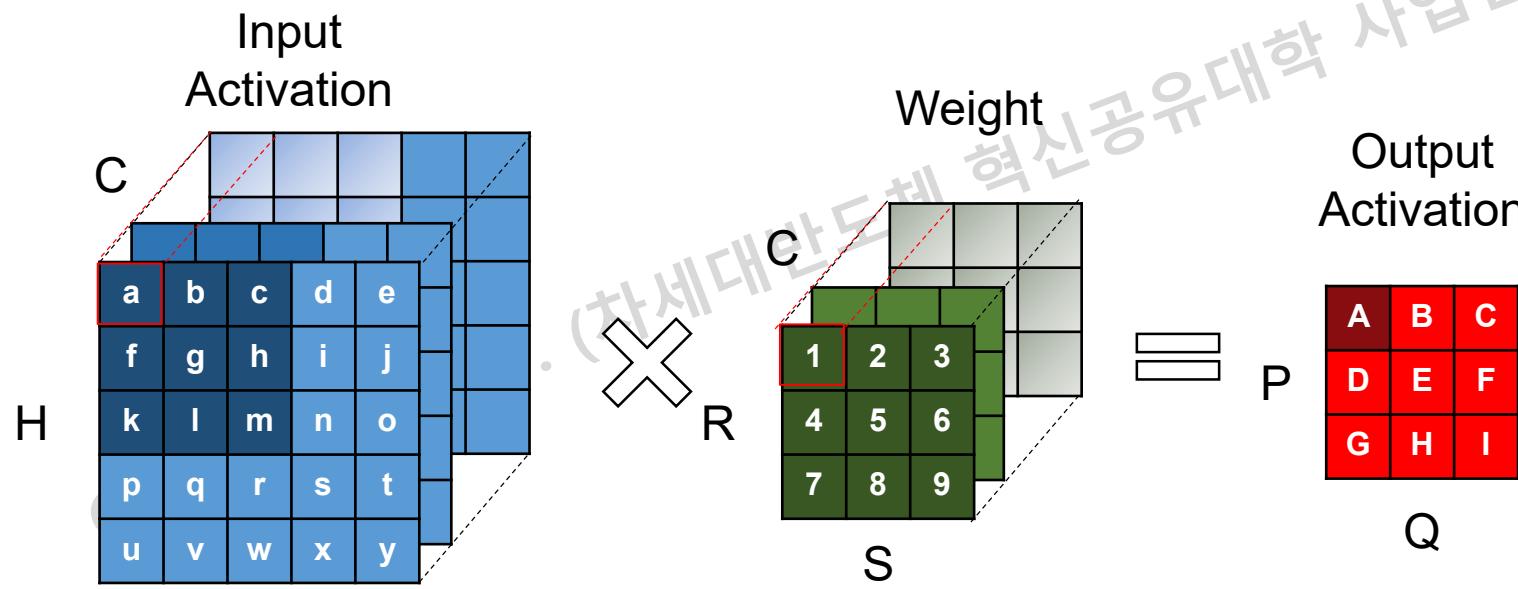
- **C:** # of Input Channels



Mapping: conv3x3

- Cycle 1:
 - Input activation "a": 1x1x16
 - Weight "1": 1x1x16
 - mac_kern.v:

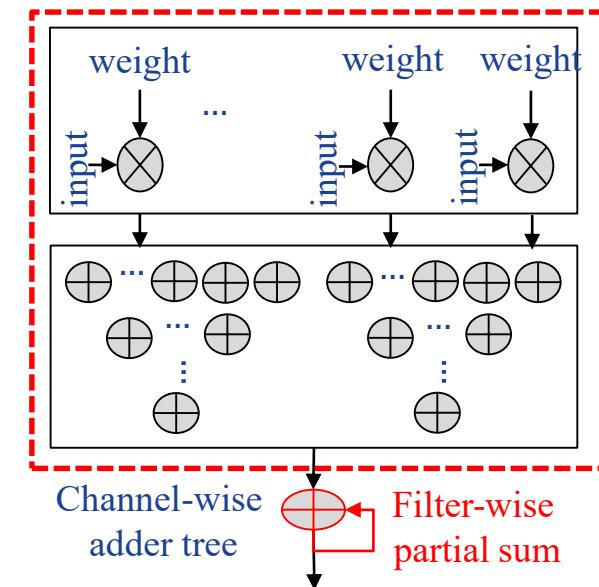
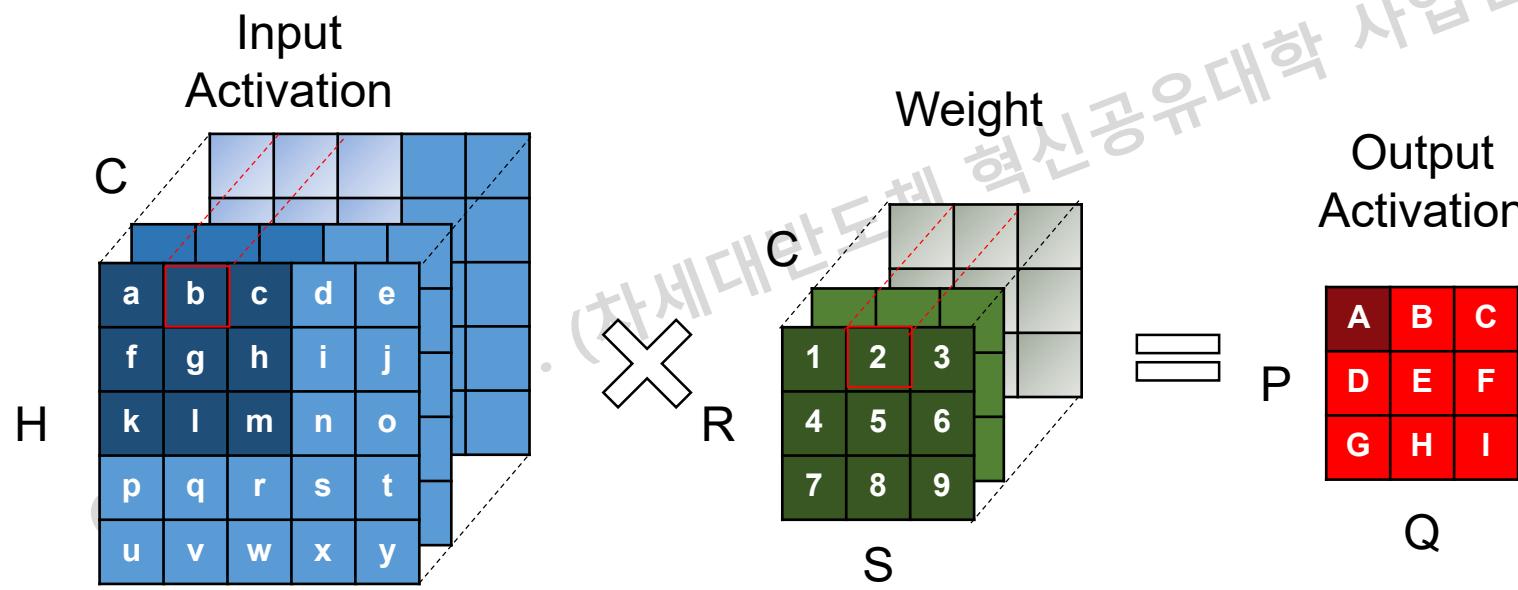
$$Y \leftarrow \sum_{i=0}^{15} w_i * x_i$$



Mapping: conv3x3

- Cycle 2:
 - Input activation "b": 1x1x16
 - Weight "2": 1x1x16
 - mac_kern.v:

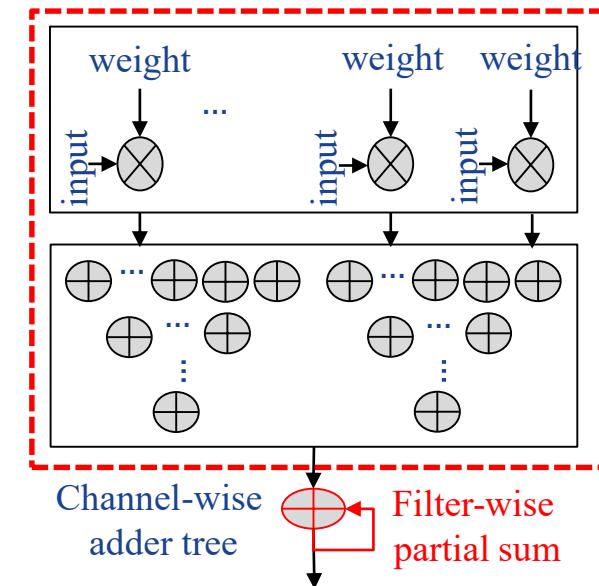
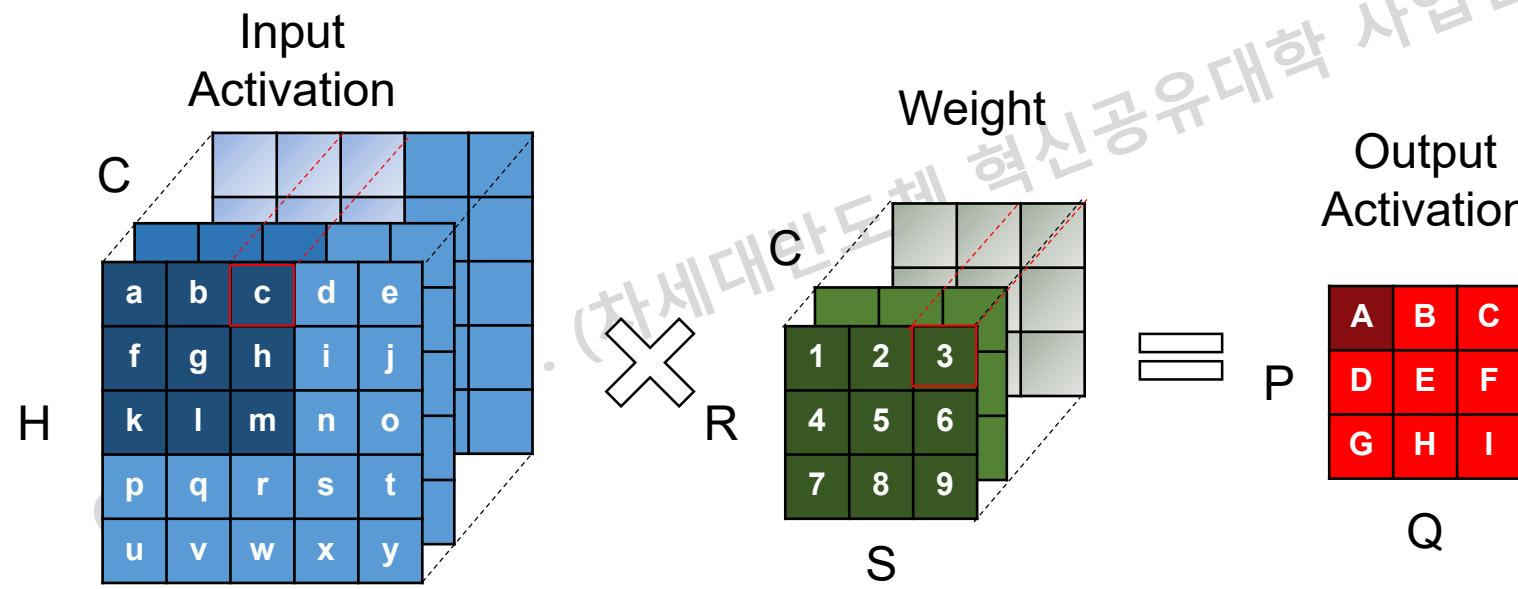
$$Y \leftarrow Y + \sum_{i=16}^{31} w_i * x_i$$



Mapping: conv3x3

- Cycle 3:
 - Input activation "c": 1x1x16
 - Weight "3": 1x1x16
 - mac_kern.v:

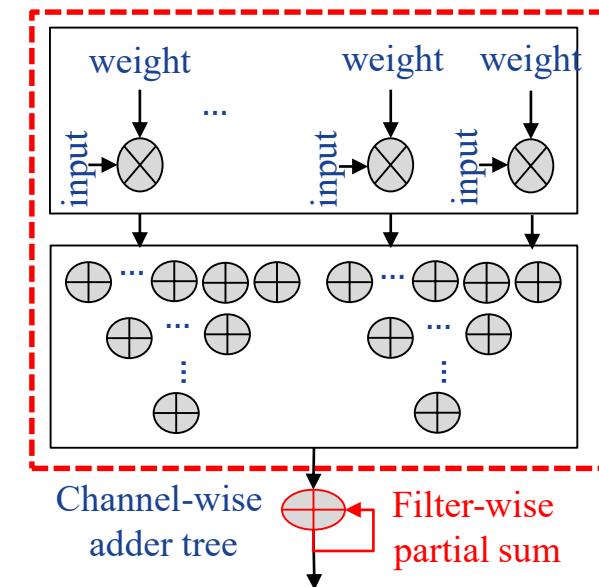
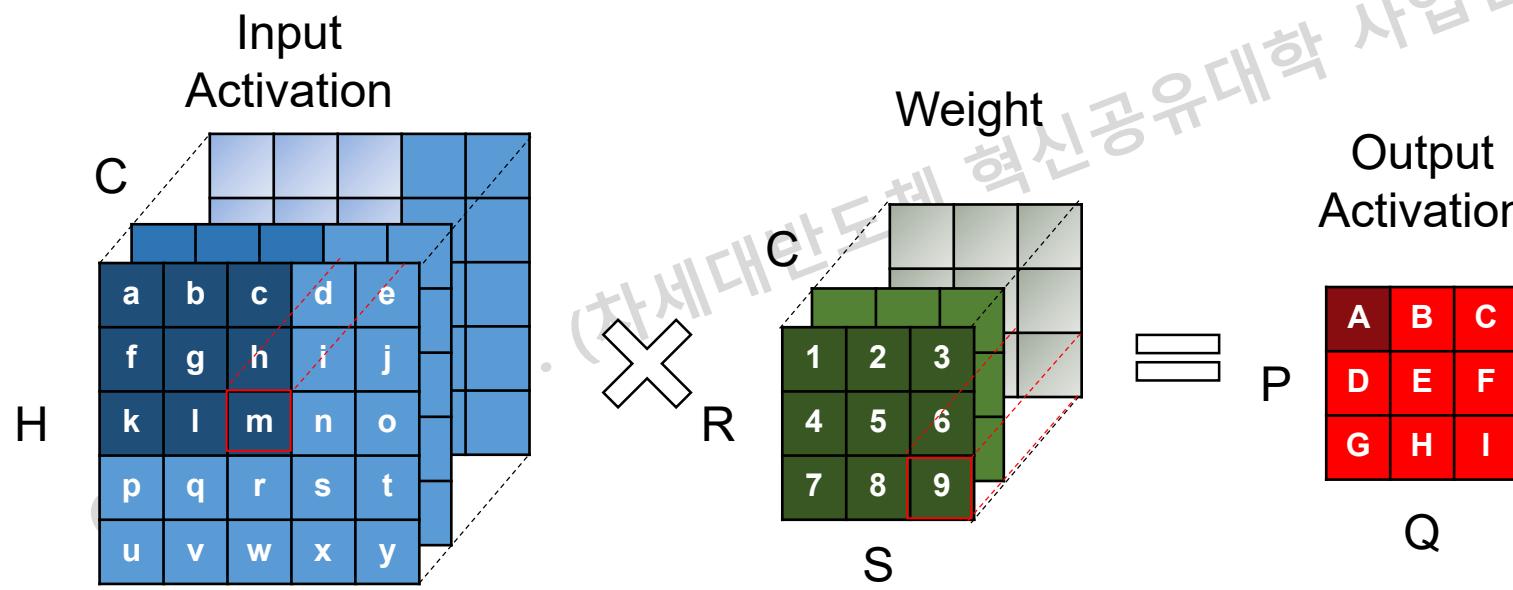
$$Y \leftarrow Y + \sum_{i=32}^{47} w_i * x_i$$



Mapping: conv3x3

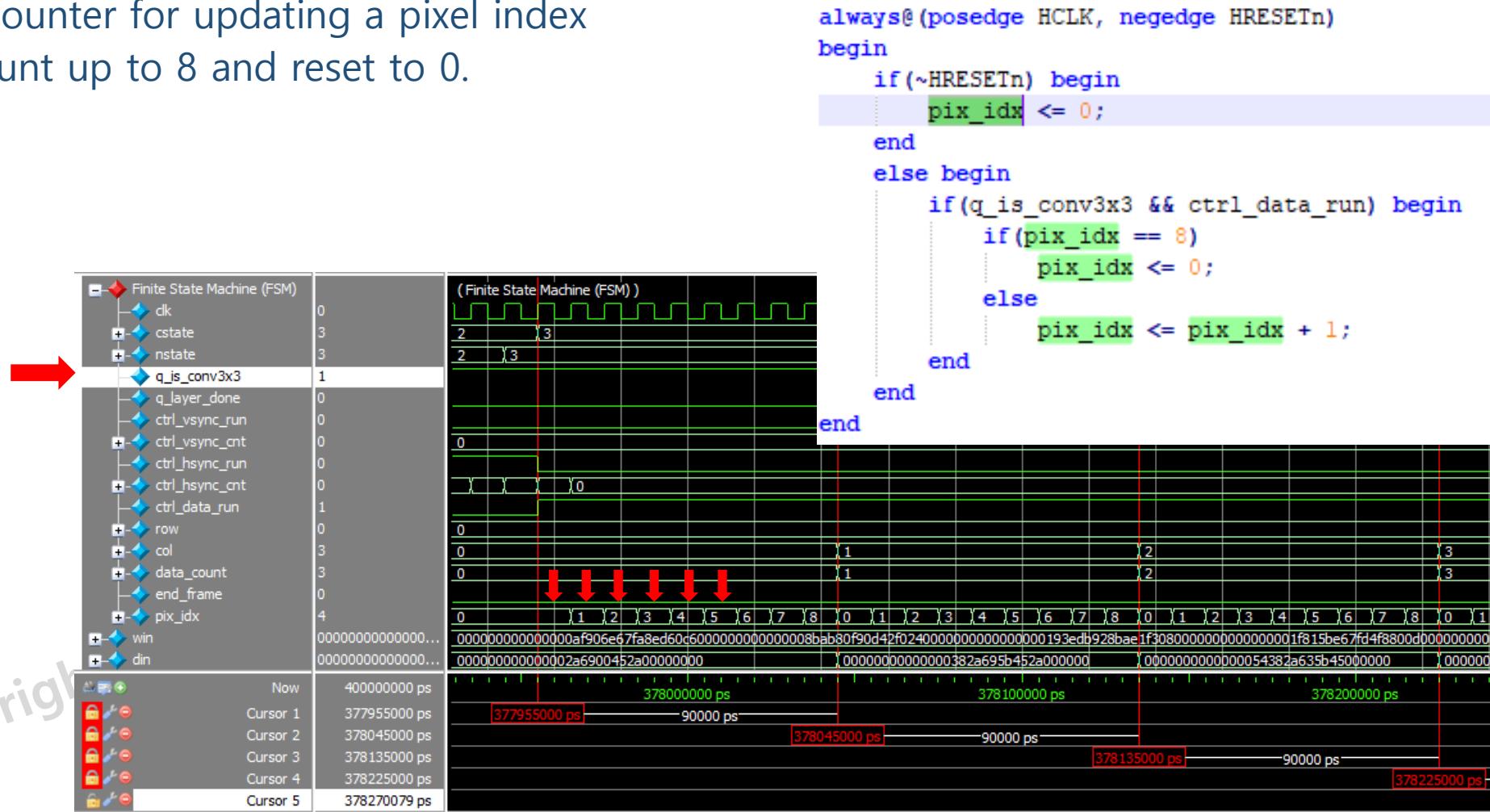
- Cycle 9:
 - Input activation "m": 1x1x16
 - Weight "9": 1x1x16
 - mac_kern.v:

$$Y \leftarrow Y + \sum_{i=128}^{143} w_i * x_i$$



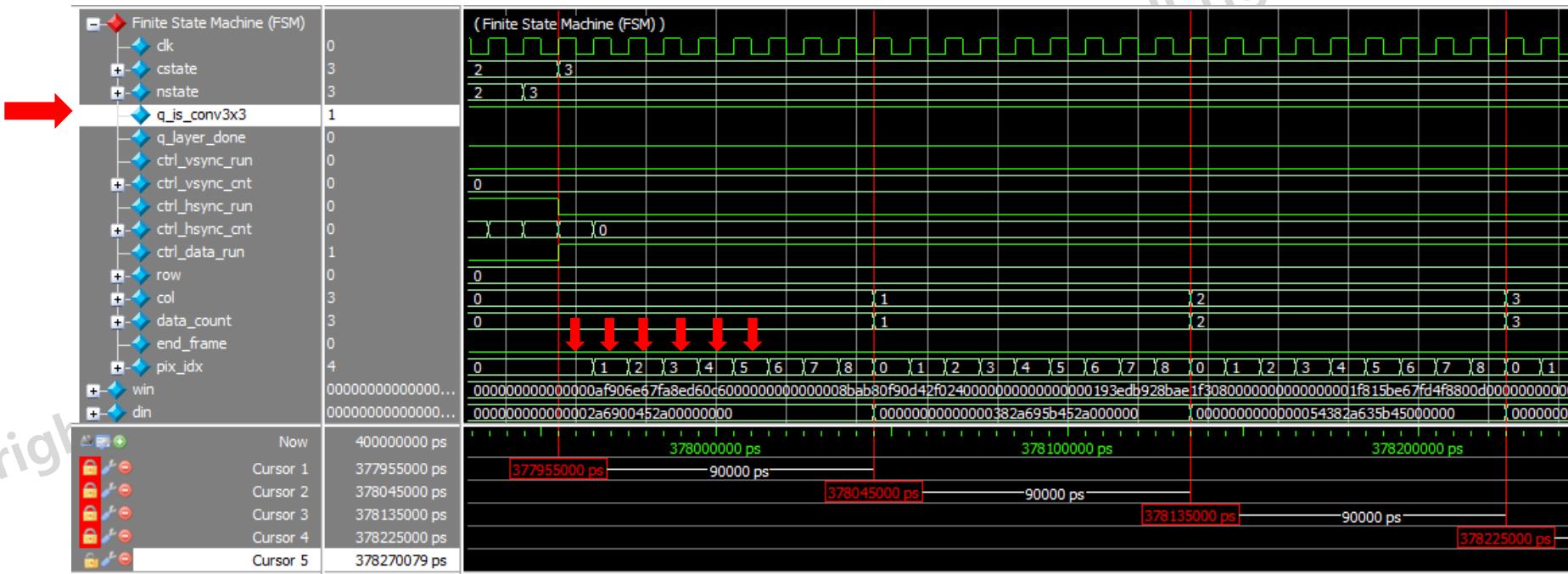
Pixel counter for CONV 3x3 (cnn_fsm.v)

- Use a counter for updating a pixel index
 - Count up to 8 and reset to 0.



CONV 3x3

- CONV3x3 (q_is_conv3x3==1)
 - If ctrl_data_run == 1
 - Need to check the pixel counter to update row, col, data_count.
 - The update happens every 9 cycles.



Test bench (cnn_accel_tb.v)

- Test bench
 - Master: RISC-V (ahb_master.v)
 - Slave: A CNN accelerator IP (cnn_accel.v)

```
'timescale 1ns / 100ps
`include "amba_ahb.h.v"
`include "map.v"

module cnn_accel_tb;
parameter W_ADDR=32;
parameter W_DATA=32;
parameter IMG_PIX_W = 8;

//parameter WIDTH    = 768,
//      HEIGHT   = 512,
parameter WIDTH    = 128,
      HEIGHT   = 128,
      START_UP_DELAY = 200,
      HSYNC_DELAY = 160,
      FRAME_SIZE = WIDTH * HEIGHT;
localparam W_SIZE   = 12;           // Max 4K QHD (3840x1920).
localparam W_FRAME_SIZE = 2 * W_SIZE + 1; // Max 4K QHD (3840x1920).
localparam W_DELAY   = 12;

parameter N_LAYER = 3;
parameter Ti = 16; // Each CONV kernel do 16 multipliers at the same time
//parameter To = 4; // Run 4 CONV kernels at the same time
parameter To = 16; // Run 16 CONV kernels at the same time
parameter N = 16;
// Inputs
reg HCLK;
reg HRESETn;
```

Master

```
//-----
// Master
//-----

ahb_master u_riscv_dummy(
    .HRESETn      (HRESETn),
    .HCLK         (HCLK),
    ..i_HRDATA   (w_RISC2AHB_mst_HRDATA),
    ..i_HRESP    (w_RISC2AHB_mst_HRESP),
    ..i_HREADY   (w_RISC2AHB_mst_HREADY),
    ..o_HADDR    (w_RISC2AHB_mst_HADDR),
    ..o_HWDATA   (w_RISC2AHB_mst_HWDATA),
    ..o_HWRITE   (w_RISC2AHB_mst_HWRITE),
    ..o_HSIZE    (w_RISC2AHB_mst_HSIZE),
    ..o_HBURST   (w_RISC2AHB_mst_HBURST),
    ..o_HTRANS   (w_RISC2AHB_mst_HTRANS)
);
```

```
//-----
// Slave
//-----

cnn_accel u_cnn_accel (
    .HCLK(HCLK),
    .HRESETn(HRESETn),
    .sl_HREADY(1'b1),
    .sl_HSEL(1'b1),
    .sl_HTRANS(w_RISC2AHB_mst_HTRANS),
    .sl_HBURST(w_RISC2AHB_mst_HBURST),
    .sl_HSIZE(w_RISC2AHB_mst_HSIZE),
    .sl_HADDR(w_RISC2AHB_mst_HADDR),
    .sl_HWRITE(w_RISC2AHB_mst_HWRITE),
    .sl_HWDATA(w_RISC2AHB_mst_HWDATA),
    .out_sl_HREADY(w_RISC2AHB_mst_HREADY),
    .out_sl_HRESP(w_RISC2AHB_mst_HRESP),
    .out_sl_HRDATA(w_RISC2AHB_mst_HRDATA)
);
```

Slave

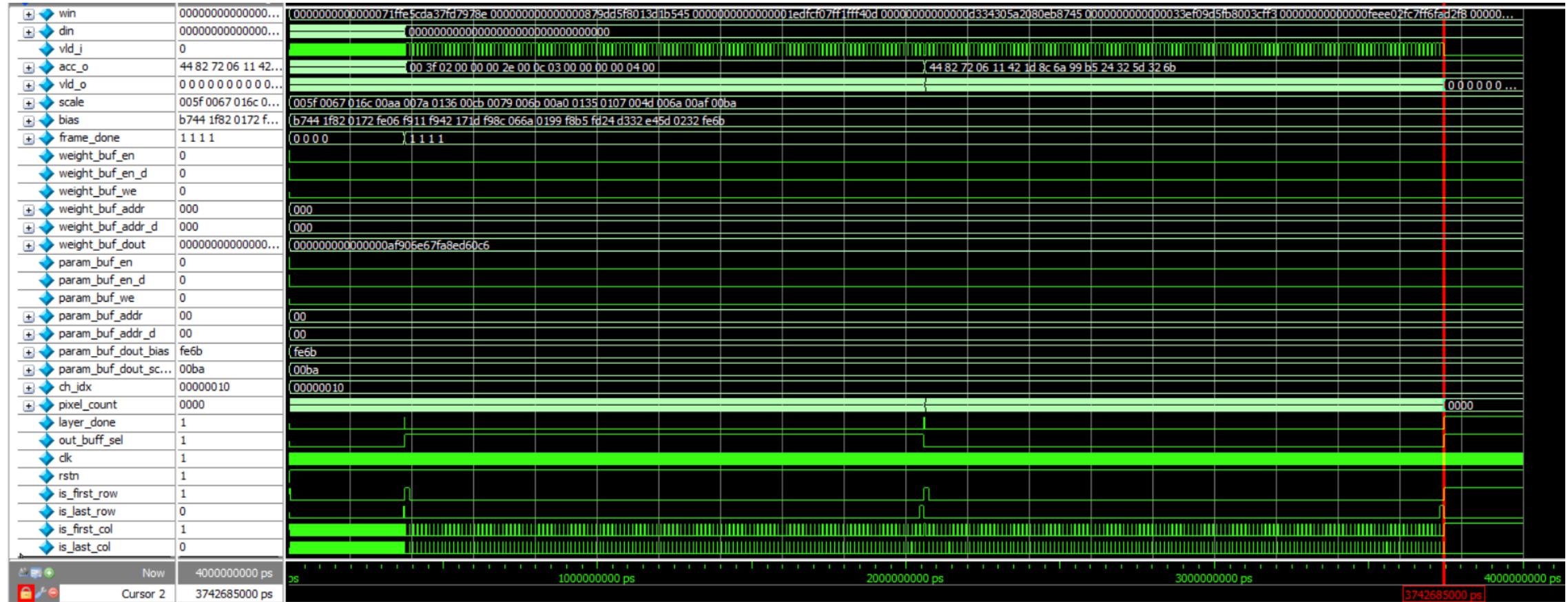
Test case

- Execute layer by layer
 - Configures a layer's configuration registers
 - Do not need to dummy "is_conv3x3" for Layers 2 and 3
 - Polling: Wait until the layer completes operations

```
//idx = 0; // Layer 1
for(idx = 0; idx <N_LAYER; idx=idx+1) begin
    q_layer_index      = idx;
    q_is_last_layer   = (idx == N_LAYER-1)?1'b1:1'b0;
    q_is_first_layer  = (idx == 0) ? 1'b1: 1'b0;
    is_conv3x3         = q_is_conv3x3[idx];
    q_layer_config     = {q_act_shift[idx], q_bias_shift[idx], q_layer_index, q_is_last_layer, is_conv3x3, q_is_last_layer, q_is_first_layer};
    #(4*p) @ (posedge HCLK) u_riscv_dummy.task_AHBwrite(`CNN_ACCEL_BASE_ADDRESS, {base_addr_param&12'hFFF,base_addr_weight&20'hFFFF});
    #(4*p) @ (posedge HCLK) u_riscv_dummy.task_AHBwrite(`CNN_ACCEL_LAYER_CONFIG, q_layer_config);
    // Start a frame
    #(4*p) @ (posedge HCLK) u_riscv_dummy.task_AHBwrite(`CNN_ACCEL_LAYER_START , 1'b1 );
    #(4*p) @ (posedge HCLK) u_riscv_dummy.task_AHBwrite(`CNN_ACCEL_LAYER_START , 1'b0 );

    // Polling
    while(!q_layer_done) begin
        #(128*p) @ (posedge HCLK) u_riscv_dummy.task_AHBread(`CNN_ACCEL_LAYER_DONE,q_layer_done);
    end
    #(128*p) @ (posedge HCLK) $display("T=%03t ns: Layer %0d done!!!\n", $realtime/1000, idx+1);
    // Reset q_layer_done
    q_layer_done = 0;
```

Waveform



Layer 1

Layer 2

Layer 3

To do ...

- Complete the missing codes
 - `cnn_fsm.v`
 - `cnn_accel.v`
 - Reuse code for setting configuration registers
 - Reuse code to generate din, vld_i
 - Add codes for weight/bias/scale buffers
- Do a simulation with time = 4,000 us
- Show/capture the output results

Copyright 2021. (차세대반도체 혁신공유대학 사업단) all rights reserved.