차세대 반도체 디지털 혁신공유대학 사업단

POLARIS

CNN accelerator IP
AHB interface, FSM, datapath

2023.1.16 (Mon)

Road map

Review

Sliding window

CNN accelerator IP
(AHB interface, FSM)

CNN accelerator IP

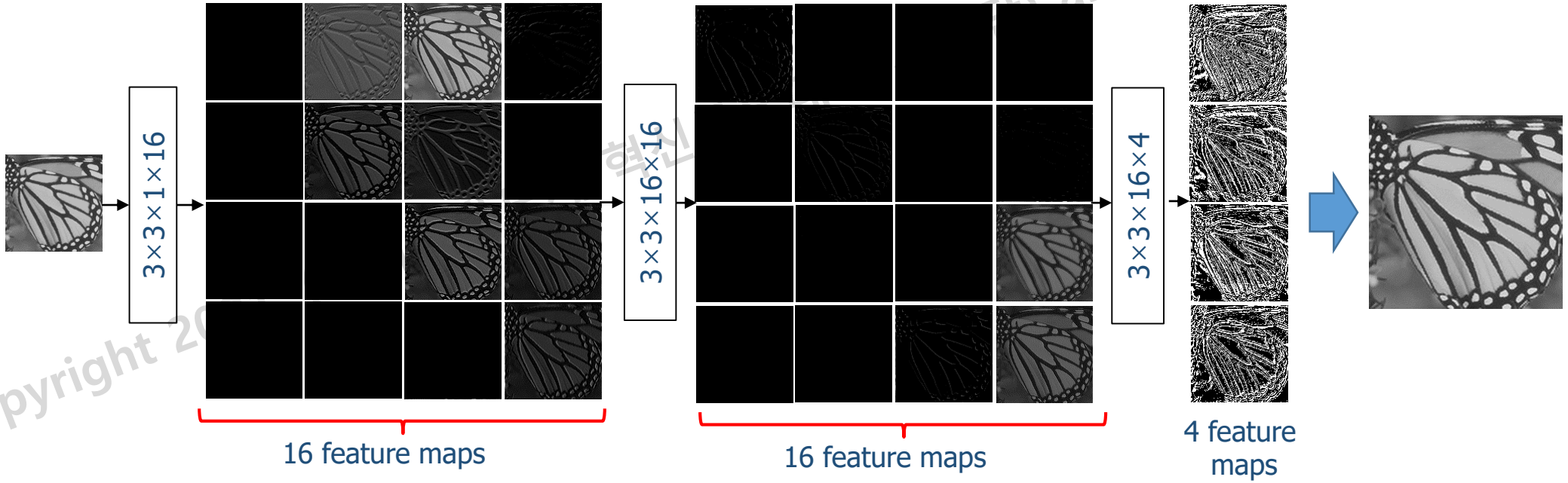
Last week

- Accelerator and motivation
- Deep neural network
  - Convolutional neural network: input/output neuron, weight, bias
  - Weight quantization
  - Activation quantization
    - RELU
    - Linear
- Hardware implementation
  - Multiplication and accumulation (MAC) (mac.v)
  - MAC kernel (mac_kern.v)
  - Normalization and activation (bnorm_quant_act.v)
  - Convolutional kernel (conv_kern.v)

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	num_ops
1	3×3	1	16	128×128×1	128×128×16	3×3×1
2	3×3	16	16	128×128×16	128×128×16	3×3×16
3	3×3	16	4	128×128×16	128×128×4	3×3×16



MAC (mac.v)

- Compute a sum of *N* products

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

- Pseudo code

$y_1^{(0)} = w_0 * x_0, \ldots, y_{15}^{(0)} = w_{15} * x_{15}$ // N multipliers

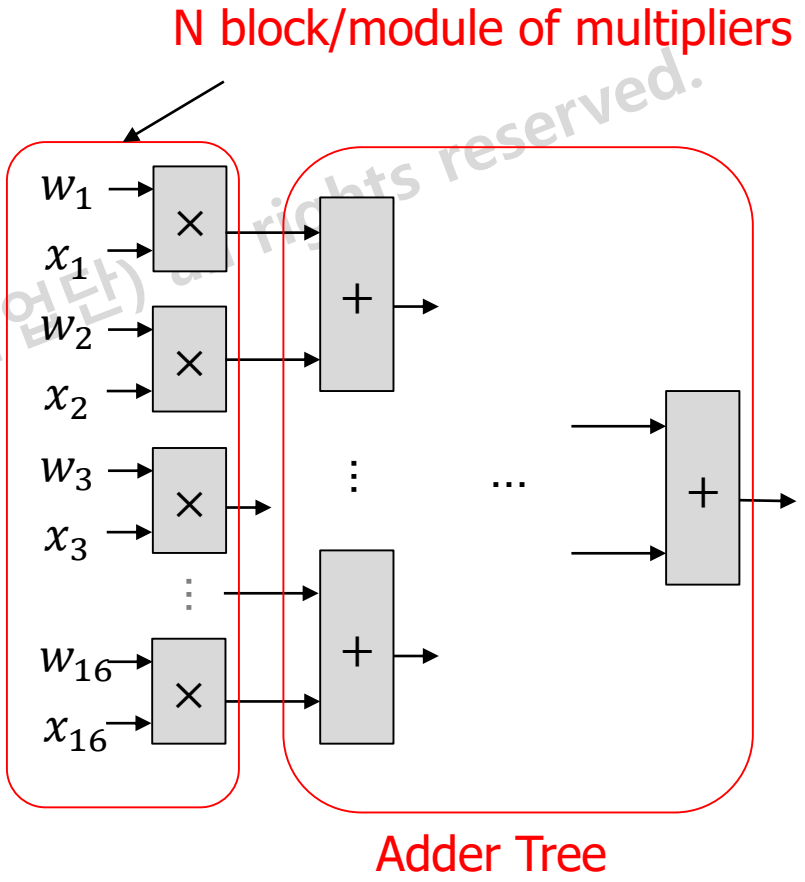
$y_1^{(1)} = y_0^{(0)} + y_1^{(0)}, \ldots, y_7^{(1)} = y_{14}^{(0)} + y_{15}^{(0)}$ // N/2 adders

$y_1^{(2)} = y_0^{(1)} + y_1^{(1)}, \ldots, y_3^{(2)} = y_6^{(1)} + y_7^{(1)}$ // N/4 adders

$y_1^{(3)} = y_0^{(2)} + y_1^{(2)}, \ldots, y_1^{(3)} = y_2^{(2)} + y_3^{(2)}$ // N/4 adders

$y_1^{(4)} = y_0^{(3)} + y_1^{(3)}$ // N/4 adders

$Y = y_1^{(4)}$ // Output

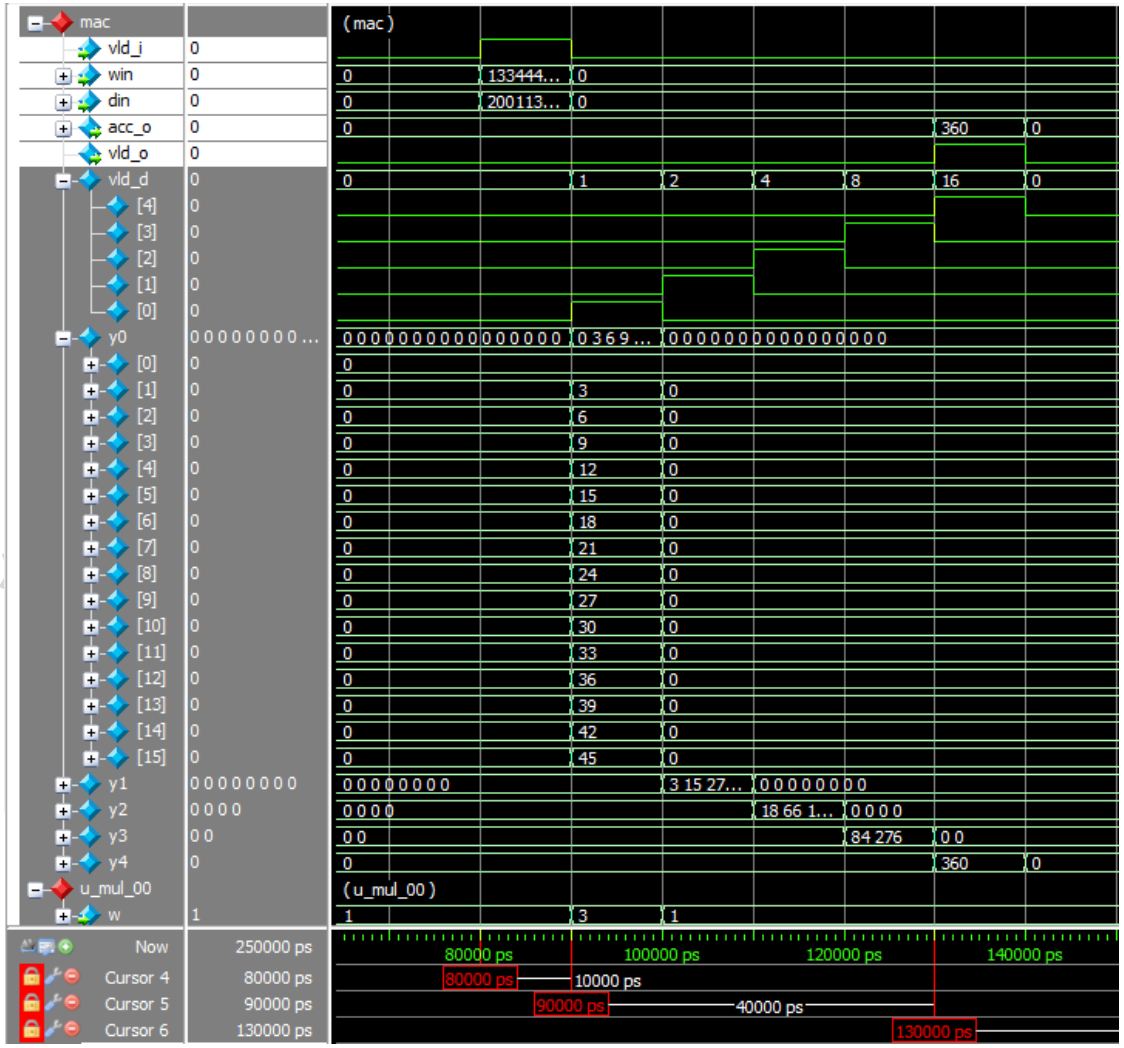


Waveform

- Delays
  - Input format + multiplier: one cycle
  - Adder tree: 4 cycles
    - $\log_2(16)$
- Results
  - Stored weight = 1

  ⇒ Weight = 1*2+1= 3
  - Output

$$\sum_{i=0}^{15}(3 \times i) = 3 \times \frac{15 \times 16}{2} = 360$$



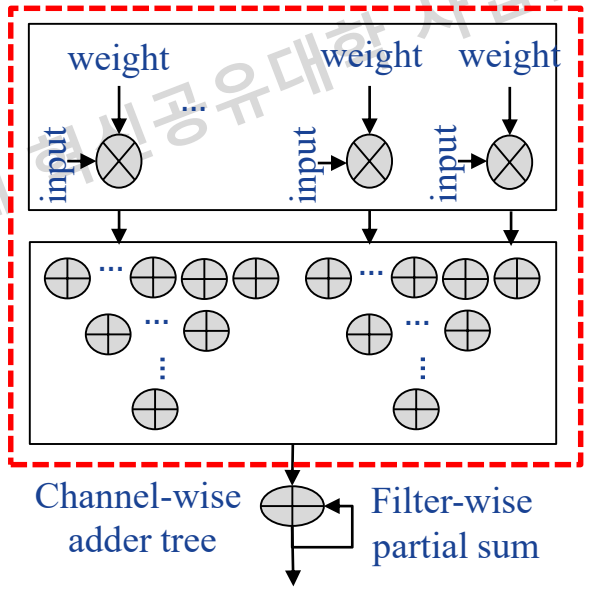
mac_tb.v

```
// Data preparation
initial begin
    for(i = 0; i <N; i = i +1) begin
        activation_block[i] = i;
        weight_block[i] = 1;
    end
end
```

Mapping

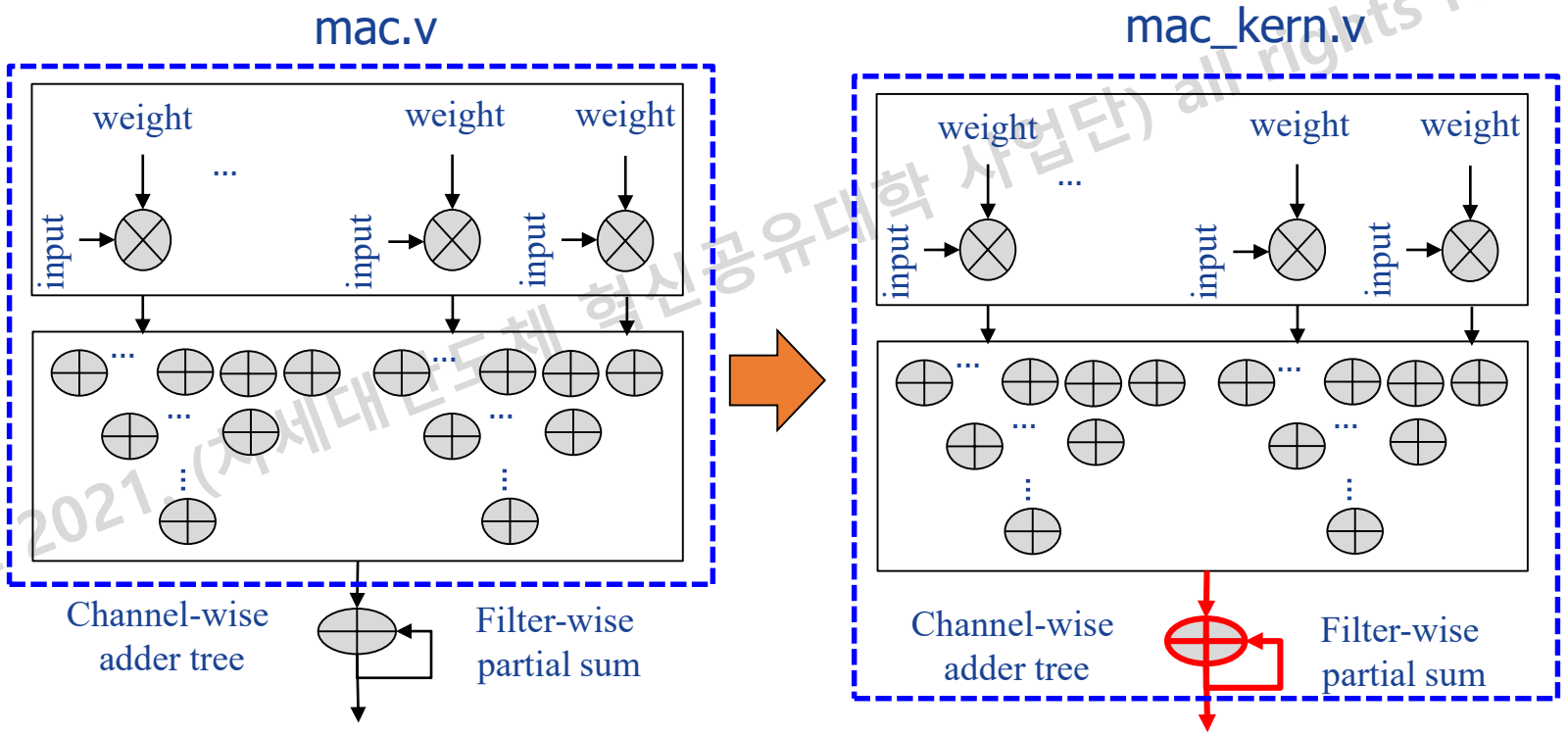
- The required number of multiplication operations per pixel (num_ops)
  - Layer 1: num_ops=9 (=3×3×1)
  - Layer 2 and 3: num_ops=144 (= 3×3×16).
- H/W module: N = 16
  - Layer 1: num_ops < N => calculating an output pixel needs 1 cycles
  - Layer 2, 3: num_ops > N => calculating an output pixel needs 9 cycles

16 input pixels
16 weights
16 multipliers



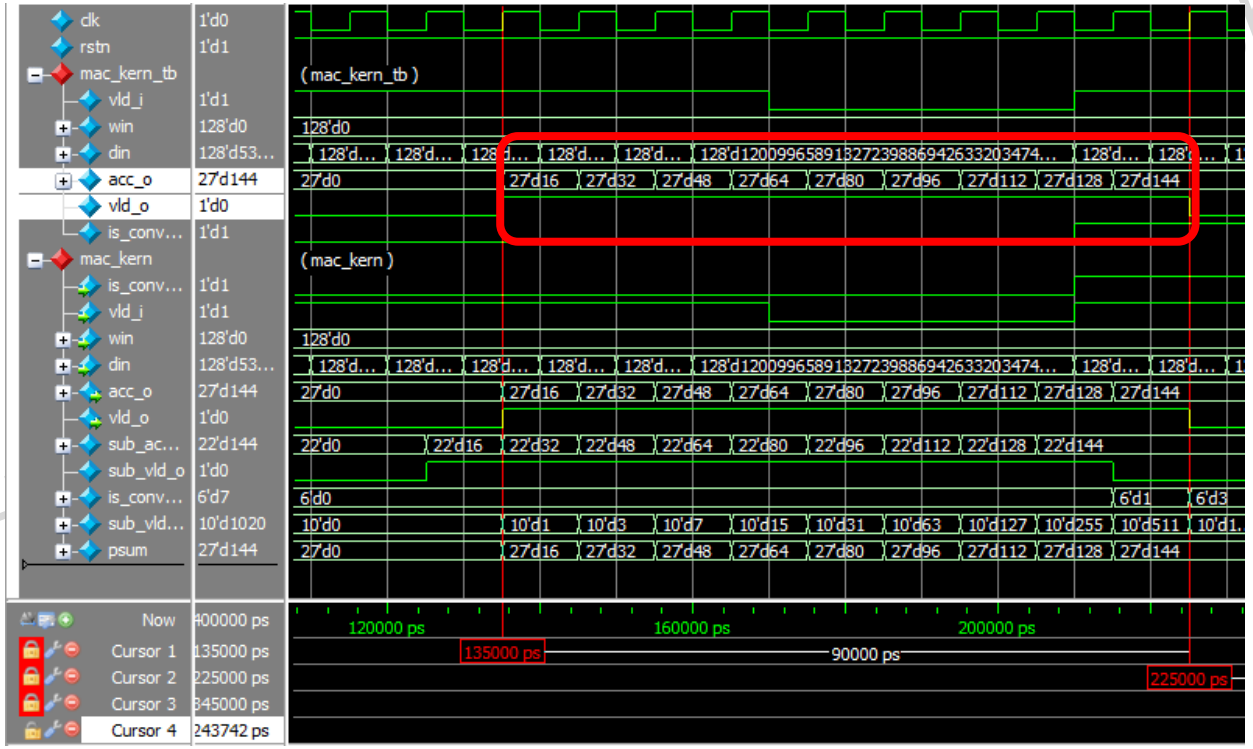
mac_kern.v vs mac.v

- MAC kernel
  - Include an instance of mac
  - Add an accumulated sum
- Mode: 0: 1x1 conv, 1: 3x3 conv.



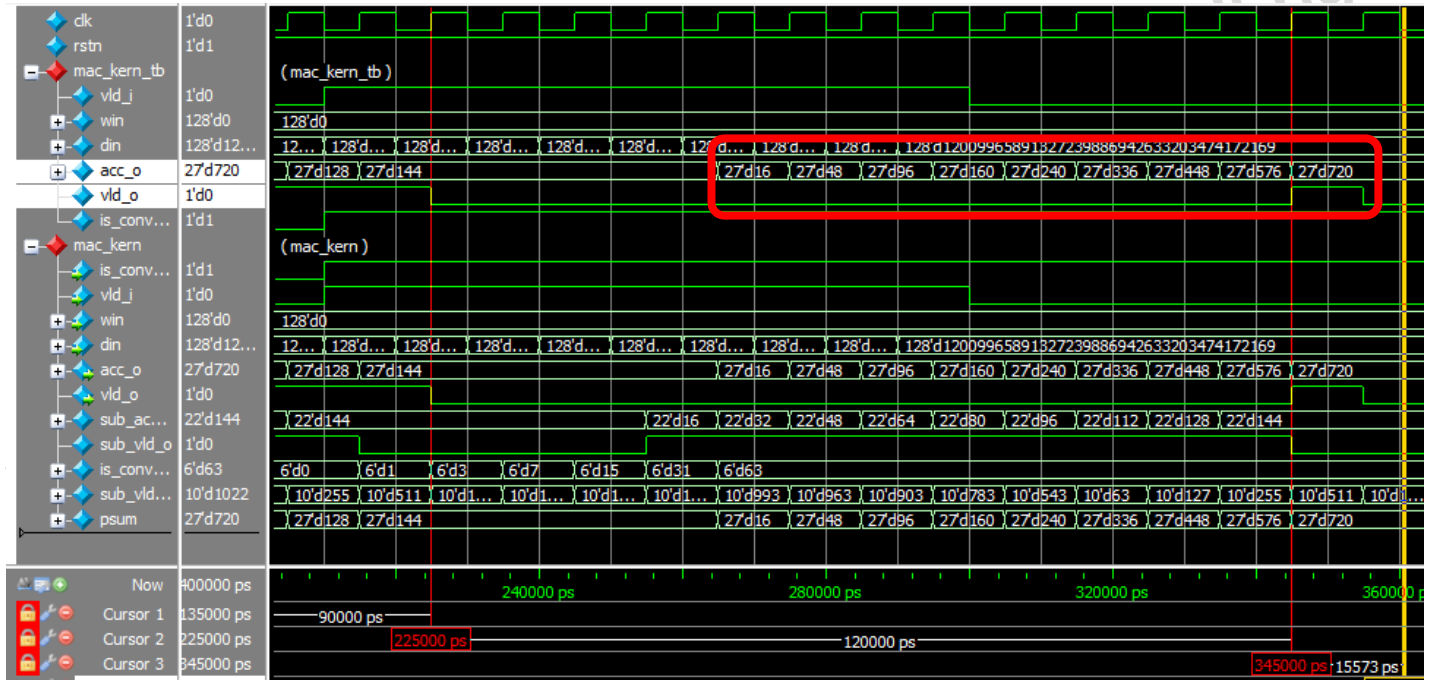
Waveform: conv1x1

- Conv1x1: we can output a pixel at every cycle
- Delay between input and output valid signals is 6 cycles
  - 5 cycle delay caused by mac.v
  - 1 cycle delay caused by the accumulation step at mac_kern.v



Waveform: conv3x3

- Conv3x3: we can output a pixel at every 9 cycles
  - It takes 9 cycles to give an output.
- Delay between input and output valid signals is 14 cycles
  - 5 cycle delay caused by mac.v
  - 9 cycle delay caused by the accumulation step at mac_kern.v



Waveform

- Stored weights = 0 → weights = 0*2+1 = 1.
- Conv1x1: **is_conv3x3=0**, vld_i=1
  - Cycle 1: acc_o = mac_o = $\sum_{i=0}^{15} 1 \times 1 = 16$
  - Cycle 2: acc_o = mac_o = $\sum_{i=0}^{15} 1 \times 2 = 32$

  …
  - Cycle 9: acc_o = mac_o = $\sum_{i=0}^{15} 1 \times 9 = 144$

- Conv3x3: **is_conv3x3=1**, vld_i=1
  - Cycle 1: mac_o = $\sum_{i=0}^{15} 1 \times 1 = 16$, acc_o ← mac_o = 16
  - Cycle 2: mac_o = $\sum_{i=0}^{15} 1 \times 2 = 32$, acc_o ← mac_o + acc_o

  …
  - Cycle 9: mac_o = $\sum_{i=0}^{15} 1 \times 9 = 144$, acc_o ← mac_o + acc_o

acc_o = 16 + 32 + … + 144 = 16 × (1 + 2 + … + 9)= 720.

Convolutional kernel (conv_kern.v)

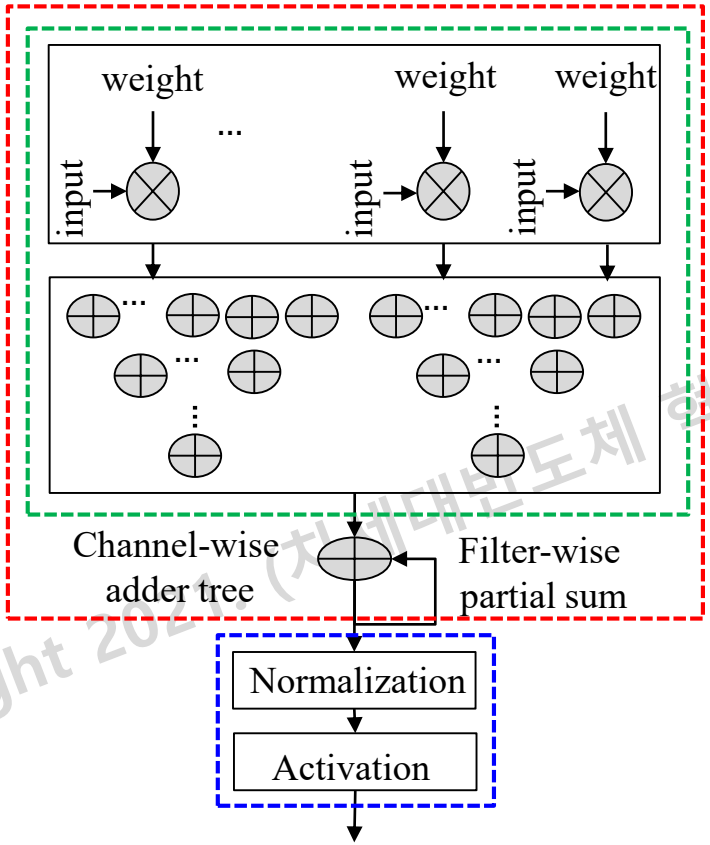
- Two modules:
  - mac_kern.v
    - Do convolution
  - bnorm_quant_act.v
    - Do batch normalization.
    - Adding a bias.
    - Do activation quantization
  - An accumulated result of mac_kern is an input of bnorm_quant_act.v

```
//------------------------------------------------
// Component: MAC
//------------------------------------------------
// DUT
mac_kern u_mac_kern(
./*input              */clk(clk),
./*input              */rstn(rstn),
./*input              */is_conv3x3(is_conv3x3),
./*input              */vld_i(vld_i),
./*input [N*WI-1:0]   */win(win),
./*input [N*WI-1:0]   */din(din),
./*output[WO-1:0]     */acc_o(mac_kern_acc_o),
./*output reg         */vld_o(mac_kern_acc_vld_o)
);
//------------------------------------------------
// Component: Batch-normalization, Activation quantization
//------------------------------------------------
bnorm_quant_act #(.DATA_BITS(DATA_BITS))
u_bnorm_quant_act
(
./*input                  */clk(clk),
./*input                  */resetn(rstn),
./*input                  */is_last_layer(is_last_layer),
./*input [PARAM_BITS-1:0]*/scale(scale),
./*input [PARAM_BITS-1:0]*/bias(bias),
./*input [2:0]            */act_shift(act_shift),
./*input [4:0]            */bias_shift(bias_shift),
./*input [DATA_BITS-1:0]  */accum_in(mac_kern_acc_o),
./*input                  */accum_vld_in(mac_kern_acc_vld_o),
./*output [ACT_BITS-1:0]  */accum_out(acc_o),
./*output                 */accum_vld_out(vld_o)
);
```

Convolution kernel

- Convolutional kernel (conv_kern.v)

conv_kern.v



mac_kern.v

```
//------------------------------------------------
// Component: MAC
//------------------------------------------------
// DUT
mac_kern u_mac_kern(
.*input              */clk(clk),
.*input              */rstn(rstn),
.*input              */is_conv3x3(is_conv3x3),
.*input              */vld_i(vld_i),
.*input [N*WI-1:0]   */win(win),
.*input [N*WI-1:0]   */din(din),
.*output[WO-1:0]     */acc_o(mac_kern_acc_o),
.*output reg         */vld_o(mac_kern_acc_vld_o)
);
```

bnorm_quant_act.v

```
//------------------------------------------------
// Component: Batch-normalization, Activation quantization
//------------------------------------------------
bnorm_quant_act #(.DATA_BITS(DATA_BITS))
u_bnorm_quant_act
(
.*input                    */clk(clk),
.*input                    */resetn(rstn),
.*input                    */is_last_layer(is_last_layer),
.*input [PARAM_BITS-1:0]*/scale(scale),
.*input [PARAM_BITS-1:0]*/bias(bias),
.*input [2:0]              */act_shift(act_shift),
.*input [4:0]              */bias_shift(bias_shift),
.*input [DATA_BITS-1:0]  */accum_in(mac_kern_acc_o),
.*input                    */accum_vld_in(mac_kern_acc_vld_o),
.*output [ACT_BITS-1:0]  */accum_out(acc_o),
.*output                   */accum_vld_out(vld_o)
);
```

Test bench (see reference S/W)

- Input pixels in the top-left corner.

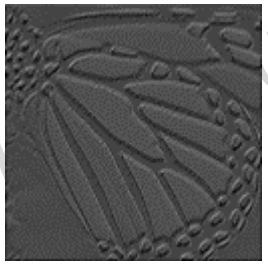
42	69	91	99	106	108	111
105	42	56	84	106	113	112
72	43	42	68	109	112	104

- Filter of channel 2

Ch#02		
139	-149	-93
39	-255	191
-69	243	17

Ch#02 (stored)		
69	181	209
19	128	95
221	121	8

- Output feature map in the top-left corner

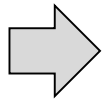


108	71	79	89	93	94
42	58	60	70	77	73
33	81	65	69	66	58

Test case

- Test the filter channel 2 of the first layer
  - bias 8066, scale 103
  - bias_shift = 9
  - act_shift = 7

Ch#02 (stored)		
69	181	209
19	128	95
221	121	8



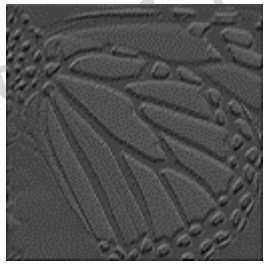
```
// Test cases
initial begin
    rstn = 1'b0;                // Reset, low active
    vld_i = 1'b0;
    win = 0;
    din = 0;
    is_conv3x3 = 1'b0;
    is_last_layer = 1'b0;
    scale = 16'd103;
    bias = 16'd8066;
    bias_shift = 9;
    act_shift = 7;
    #(4*CLK_PERIOD) rstn = 1'b1;

    // First layer, channel 2:
    win[0*WI+:WI] = 8'd69;
    win[1*WI+:WI] = 8'd181;
    win[2*WI+:WI] = 8'd209;
    win[3*WI+:WI] = 8'd19;
    win[4*WI+:WI] = 8'd128;
    win[5*WI+:WI] = 8'd95;
    win[6*WI+:WI] = 8'd221;
    win[7*WI+:WI] = 8'd121;
    win[8*WI+:WI] = 8'd8;
    // Test case 1: test conv1x1
    is_conv3x3 = 1'b0;
```

Test case

- A 3x3 window

42	69	91	99	106	108	111
105	42	56	84	106	113	112
72	43	42	68	109	112	104

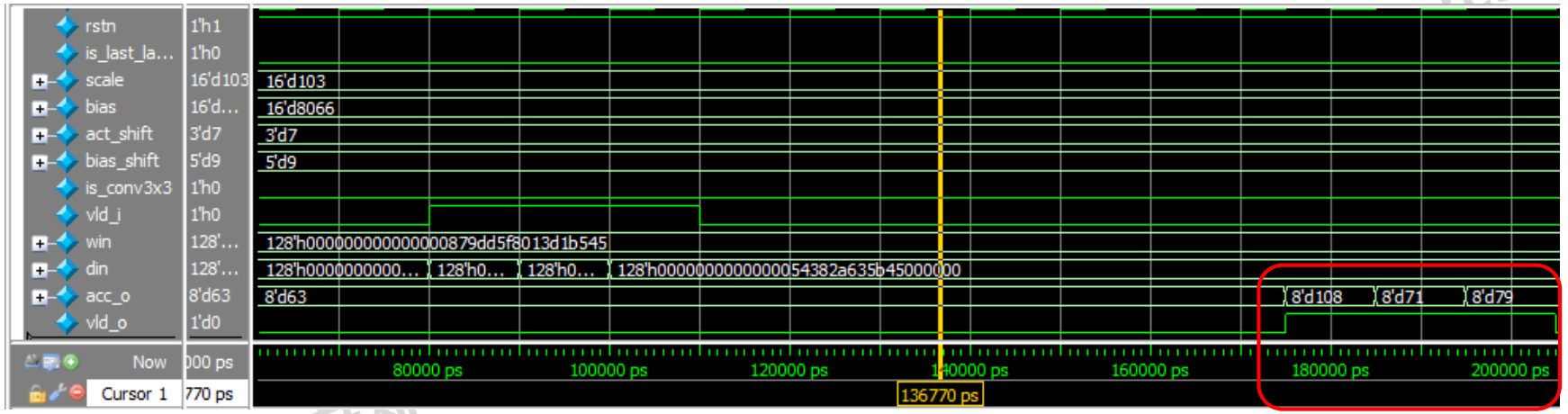
```
#(4*CLK_PERIOD) vld_i = 1'b1;
                din[0*WI+:WI] = 8'd0;
                din[1*WI+:WI] = 8'd0;
                din[2*WI+:WI] = 8'd0;
                din[3*WI+:WI] = 8'd0;
                din[4*WI+:WI] = 8'd42;
                din[5*WI+:WI] = 8'd69;
                din[6*WI+:WI] = 8'd0;
                din[7*WI+:WI] = 8'd105;
                din[8*WI+:WI] = 8'd42;
```

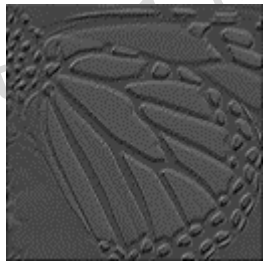


108	71	79	89	93	94
42	58	60	70	77	73
33	81	65	69	66	58

Waveform

- Do simulation with time = 250ns
- Show the waveform





108	71	79	89	93	94
42	58	60	70	77	73
33	81	65	69	66	58

Road map

Review

Sliding window

CNN accelerator IP
(AHB interface, FSM)

CNN accelerator IP

Recall: Convolution kernel function

- Before building a H/W, we need to build a reference S/W
- Why?
  - S/W: abstract, high-level representation of a function, an algorithm
  - Generate test-bench for H/W verification

- Convolution function (convol.2)

```
% Function: Convolution Only
% Input
%   - img: Input feature map
%   - kernel: Weight
%   - s: stride
%   - p: padding
% Output
%   - out
function out = convol2(img,kernel,s,p)
```

Recall: Convolution kernel function

- **Input Buffers**
  - Padded input image for handling boundary pixels
    - Zero padding

```
function out = convol2(img,kernel,s,p)
% Input Feature Maps
h = size(img,1);     % height
w = size(img,2);     % width
c = size(img,3);     % number of input features

% Padded image: zero padding
pad_img = zeros(h+p,w+p,c);
st_p = ceil(p/2);
pad_img(1+st_p:st_p+h,1+st_p:st_p+w,:) = img;

% Filter kernel size
f = size(kernel,1);

% Output feature maps
c_out = size(kernel,4);                 % Number of
w_out = floor((w - f + p) / s + 1); % width
h_out = floor((h - f + p) / s + 1); % height

out = zeros(h_out,w_out,c_out);      % buffer of output feature maps
```

img — 128x128 single

	1	2	3	4
1	42	69	91	99
2	105	42	56	84
3	72	43	42	68
4	43	45	41	60

pad_img — 130x130 double

	1	2	3	4	5
1	0	0	0	0	0
2	0	42	69	91	99
3	0	105	42	56	84
4	0	72	43	42	68
5	0	43	45	41	60

Recall: Convolution kernel function

- **Output Buffers**
  - Calculate the sizes
  - Allocate a buffer/variable for output feature maps

```
function out = convol2(img,kernel,s,p)
% Input Feature Maps
h = size(img,1);     % height
w = size(img,2);     % width
c = size(img,3);     % number of input features

% Padded image: zero padding
pad_img = zeros(h+p,w+p,c);
st_p = ceil(p/2);
pad_img(1+st_p:st_p+h,1+st_p:st_p+w,:) = img;

% Filter kernel size
f = size(kernel,1);

% Output feature maps
c_out = size(kernel,4);                % Number of output features
w_out = floor((w - f + p) / s + 1); % width
h_out = floor((h - f + p) / s + 1); % height

out = zeros(h_out,w_out,c_out);      % buffer of output feature maps
```

Recall: Convolution kernel function

- Three loops
  - Number of output channels
  - Row: line by line
  - Column: from left to right
- Kernel operations
  - Element-wise multiplication (products)
  - Sum (sum of products)

```
% Convolution
for k = 1:c_out                 % Number of output channels
    for i = 1:h_out             % Row
        for j = 1:w_out         % Column
            % Element-wise Muliplication => Products
            scalar = kernel(:,:,:,k).* ...
                     pad_img(1+(i-1)*s:1+(i-1)*s+f-1, 1+(j-1)*s:1+(j-1)*s+f-1, :);
            % Sum (Sum of Products)
            out(i,j,k) = sum(scalar(:));
        end
    end
end
```

Mapping: Reorganize convol2.m

- Choose a weight set (win).
- Kernel operations
  - Element-wise multiplication (products)
  - Sum (sum of products)

```
% 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-
            % Element-wise Muliplication => Products
            scalar = win .* din;
            % Sum (Sum of Products): Adder tree
            out(i,j,k) = sum(scalar(:));
        end
    end
end
```

Ch#02 (stored)		
69	181	209
19	128	95
221	121	8

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

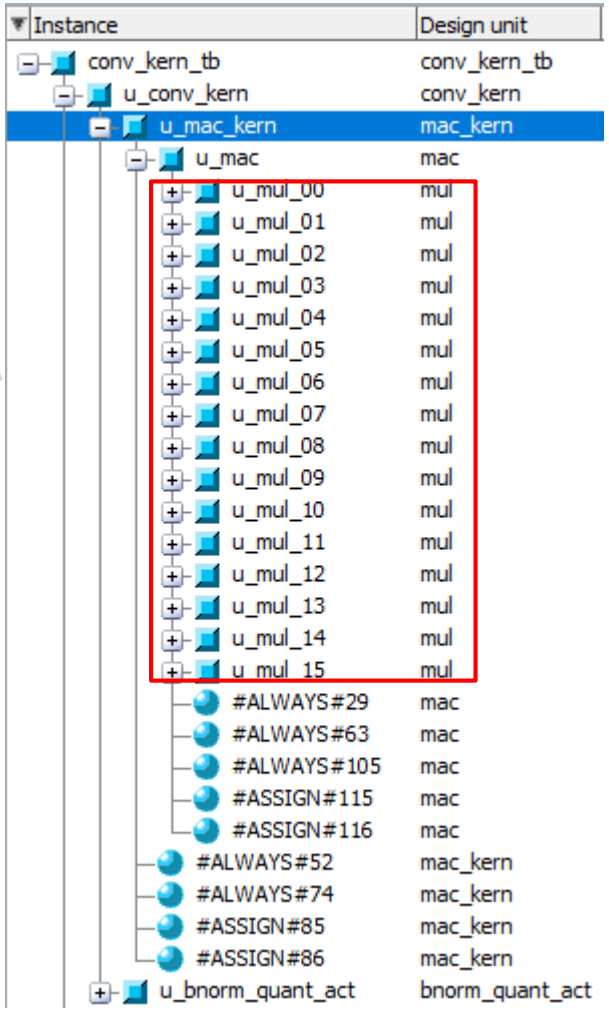
Ch#02		
139	-149	-93
39	-255	191
-69	243	17

Each weight is in 8 bits

Mapping: Reorganize convol2.m

- Choose a weight set (win).
- 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 Muliplication => Products
            scalar = win .* din;
            % Sum (Sum of Products): Adder tree
            out(i,j,k) = sum(scalar(:));
        end
    end
end
```



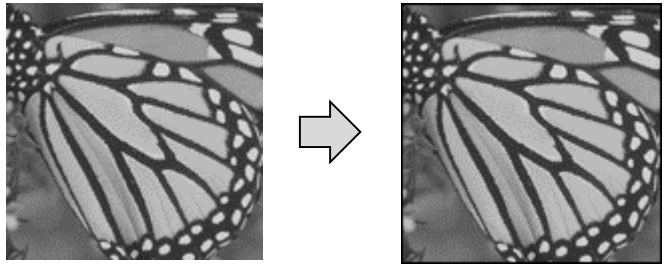
Mapping: Reorganize convol2.m

- Sliding window: extract a tensor of an input map (din) to do convolution
  - To compute 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 Muliplication => Products
            scalar = win .* din;
            % Sum (Sum of Products): Adder tree
            out(i,j,k) = sum(scalar(:));
        end
    end
end
```

Example: Sliding window

- First layer: one channel
  - Zero padding



0	0	0	0	0	0	0	0
0	42	69	91	99	106	108	111
0	105	42	56	84	106	113	112
0	72	43	42	68	109	112	104

- Sliding window

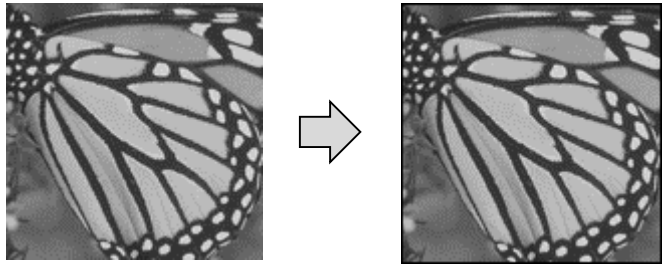
0	0	0	0	0	0	0	0
0	42	69	91	99	106	108	111
0	105	42	56	84	106	113	112
0	72	43	42	68	109	112	104

out(0,0,k)[1]

[1] In C or Verilog, loop indexes are started from 0. In Matlab, they start from 1.

Example: Sliding window

- First layer: one channel
  - Zero padding



0	0	0	0	0	0	0	0
0	42	69	91	99	106	108	111
0	105	42	56	84	106	113	112
0	72	43	42	68	109	112	104

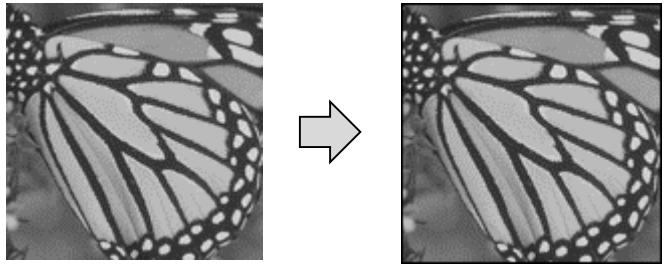
- Sliding window

0	0	0	0	0	0	0	0
0	42	69	91	99	106	108	111
0	105	42	56	84	106	113	112
0	72	43	42	68	109	112	104

out(0,1,k)

Example: Sliding window

- First layer: one channel
  - Zero padding



0	0	0	0	0	0	0	0
0	42	69	91	99	106	108	111
0	105	42	56	84	106	113	112
0	72	43	42	68	109	112	104

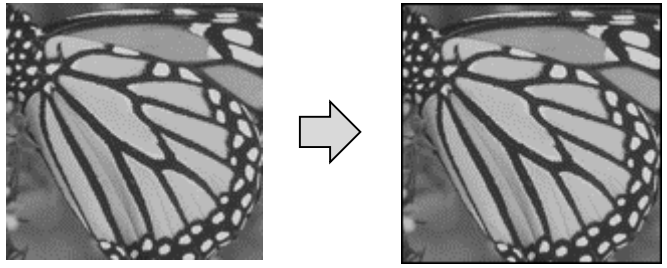
- Sliding window

0	0	0	0	0	0	0	0
0	42	69	91	99	106	108	111
0	105	42	56	84	106	113	112
0	72	43	42	68	109	112	104

out(0,2,k)

Example: Sliding window

- First layer: one channel
  - Zero padding



0	0	0	0	0	0	0	0
0	42	69	91	99	106	108	111
0	105	42	56	84	106	113	112
0	72	43	42	68	109	112	104

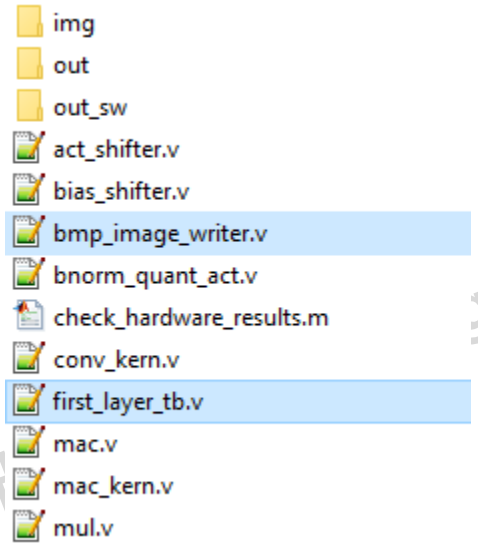
- Sliding window

0	0	0	0	0	0	0	0
0	42	69	91	99	106	108	111
0	105	42	56	84	106	113	112
0	72	43	42	68	109	112	104

out(0,3,k)

Lab 1: Sliding windows

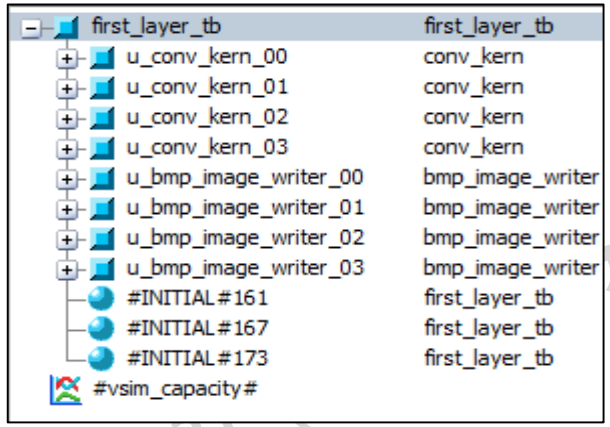
- **Lab 1: First layer**
  - Implement first_layer_tb.v
  - Four convolution kernels (conv_kern.v)
  - Each kernel is connected to an image writer (bmp_image_writer.v)

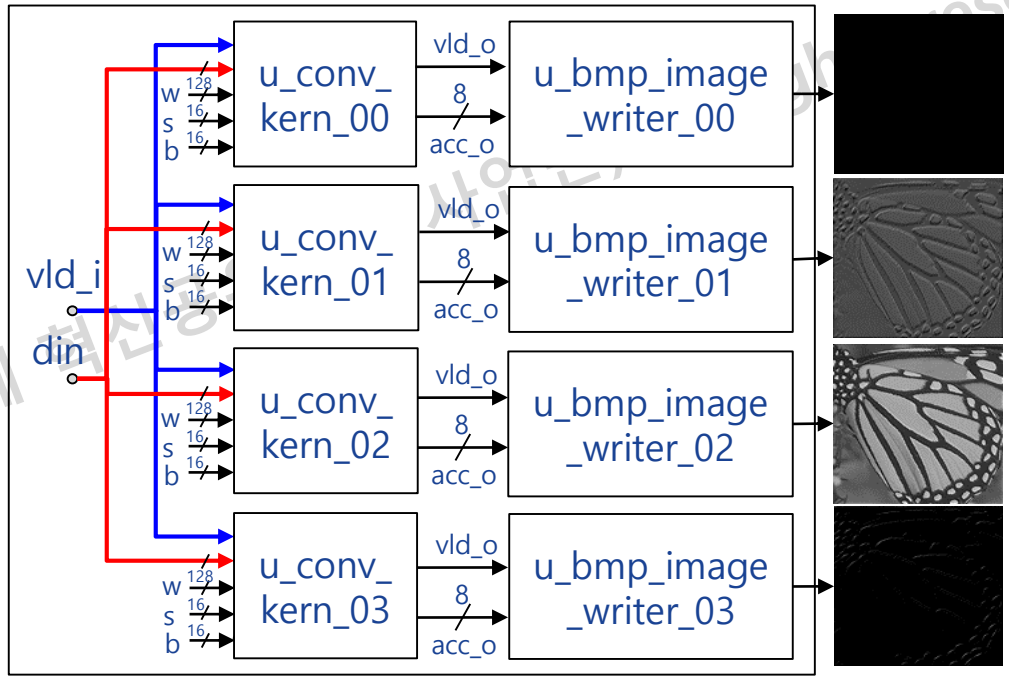



```
module first_layer_tb;

// Image parameters
parameter BMP_HEADER_NUM = 54;
parameter WIDTH      = 128;
parameter HEIGHT     = 128;
parameter INFILE     = "./img/butterfly_08bit.hex";
parameter OUTFILE00    = "./out/convout_layer01_ch01.bmp";
parameter OUTFILE01    = "./out/convout_layer01_ch02.bmp";
parameter OUTFILE02    = "./out/convout_layer01_ch03.bmp";
parameter OUTFILE03    = "./out/convout_layer01_ch04.bmp";
```

First layer (first_layer_tb.v)

- Four convolution kernels (conv_kern.v)
  - Share an input valid (vld_i) and data inputs (din)
- Each kernel is connected to an image writer





Note: w: weights (win), s: scale, b: bias

Convolutional kernels and image writers

```
//-----------------------------------------------
// DUT: Convolution Kernels
//-----------------------------------------------
// Channel 00
conv_kern u_conv_kern_00(
.*input                  */clk(c
.*input                  */rstn(
.*input                  */is_last_layer(is_last_layer),
.*input [PARAM_BITS-1:0]*/scale(scale[0]),
.*input [PARAM_BITS-1:0]*/bias(bias[0]),
.*input [2:0]            */act_shift(act_shift),
.*input [4:0]            */bias_shift(bias_shift),
.*input                  */is_conv3x3(is_conv3x3),
.*input                  */vld_i(vld_i),
.*input [N*WI-1:0]       */win(win[0]),
.*input [N*WI-1:0]       */din(din),
.*output [ACT_BITS-1:0] */acc_o(acc_o[0]),
.*output                 */vld_o(vld_o[0])
);
```

conv_kern_00

```
conv_kern u_conv_kern_01(
.*input                  */clk(c
.*input                  */rstn(
.*input                  */is_last_layer(is_last_layer),
.*input [PARAM_BITS-1:0]*/scale(scale[1]),
.*input [PARAM_BITS-1:0]*/bias(bias[1]),
.*input [2:0]            */act_shift(act_shift),
.*input [4:0]            */bias_shift(bias_shift),
.*input                  */is_conv3x3(is_conv3x3),
.*input                  */vld_i(vld_i),
.*input [N*WI-1:0]       */win(win[1]),
.*input [N*WI-1:0]       */din(din),
.*output [ACT_BITS-1:0] */acc_o(acc_o[1]),
.*output                 */vld_o(vld_o[1])
);
```

conv_kern_01

```
//-----------------------------------------------
// Image Writer
//-----------------------------------------------
bmp_image_writer#(.WIDTH(WIDTH),.HEIGHT(HEIGHT),.OUTFILE(OUTFILE00))
u_bmp_image_writer_00(
.*input              */clk(clk),
.*input              */rstn(rstn),
.*input [WI-1:0]     */din(acc_o[0]),
.*input              */vld(vld_o[0]),
.*output reg         */frame_done(frame_done[0])
);
```

```
bmp_image_writer#(.WIDTH(WIDTH),.HEIGHT(HEIGHT),.OUTFILE(OUTFILE01))
u_bmp_image_writer_01(
.*input              */clk(clk),
.*input              */rstn(rstn),
.*input [WI-1:0]     */din(acc_o[1]),
.*input              */vld(vld_o[1]),
.*output reg         */frame_done(frame_done[1])
);
```

Input image buffer (first_layer_tb.v)

- Similar to the LCD driver example, an input image is stored in a hex file.
  - 42 = $2A_{Hex}$
  - INFILE: img/butterfly_08bit.hex
- Read a file to a buffer (first_layer_tb.v)
  - All input pixels are stored in a buffer
    - in_image[0:HEIGHT*WIDTH-1]

```
clc
clear all
close all

load('test_vector_sim_espcn_3x3.m
input = test_vector{1,1};
%% Write a test image
buf = input';
buf  = buf(:);
% 8-bit format
fid = fopen('output/butterfly_08bit.hex','wt');
fprintf(fid,'%x\n',buf);
fclose(fid);
% 32-bit format
fid = fopen('output/butterfly_32bit.hex','wt');
fprintf(fid,'%08x\n',buf);
fclose(fid);
```

input

128x128 single

	1	2	3	4
1	42	69	91	99
2	105	42	56	84
3	72	43	42	68
4	43	45	41	60

butterfly_08bit.hex

1	2a
2	45
3	5b
4	63
5	6a
6	6c
7	6f
8	6d

```
//----------------------------------------------------
// Test cases
//----------------------------------------------------
// Read the input file to memory
initial begin
    $readmemh(INFILE, in_img ,0,FRAME_SIZE-1);
end

// Clock
parameter CLK_PERIOD = 10;  //100MHz
initial begin
    clk = 1'b0;
    forever #(CLK_PERIOD/2) clk = ~clk;
end

// Test cases
initial begin
    rstn = 1'b0;              // Reset, low active
    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;
    #(4*CLK_PERIOD) rstn = 1'b1;
```

Input image buffer (first_layer_tb.v)

- Clock (clk): 100MHz, reset (rstn)
- Parameters
  - Scales
  - Biases
  - Layer configuration
    - is_conv3x3 = 1'b0
      - num_ops=9 < N=16
    - is_last_layer = 1'b0
    - bias_shift = 9
    - act_shift = 7

```
//------------------------------------------------------
// Test cases
//------------------------------------------------------
// Read the input file to memory
initial begin
    $readmemh(INFILE, in_img ,0,FRAME_SIZE-1);
end

// Clock
parameter CLK_PERIOD = 10;  //100MHz
initial begin
    clk = 1'b0;
    forever #(CLK_PERIOD/2) clk = ~clk;
end

// Test cases
initial begin
    rstn = 1'b0;            // Reset, low active
    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;
    #(4*CLK_PERIOD) rstn = 1'b1;
```

Scales and biases

- Four biases and four scales are preloaded to buffers (e.g., scale, bias)
- 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

test_vector{1, 3}

	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

test_vector{1, 4}

	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

Weights

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

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 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 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 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
```

Generate din

- Extract a tensor of an input map (din) to do convolution
  - Loop: Row and column counters
  - Check boundary conditions

```
for(row = 0; row < HEIGHT; row = row + 1) begin
   for(col = 0; col < WIDTH; col = col + 1) begin
      if(row == 0 ) begin                    // First row
         if (col == 0)                       // First column
         else if (col == WIDTH-1)            // Last column
         else                                // Middle column
      end
      else if (row == HEIGHT-1)begin         // Last row
         if (col == 0)                       // First column
         else if (col == WIDTH-1)            // Last column
         else                                // Middle column
      end
      else begin                             // Middle row
         if (col == 0)                       // First column
         else if (col == WIDTH-1)            // Last column
         else                                // Middle column
      end
   end
end
```

Generate din

- Extract a tensor of an input map (din) to do convolution
  - Loop: Row and column counters
  - Check boundary conditions

```
for(row = 0; row < HEIGHT; row = row + 1) begin
    for(col = 0; col < WIDTH; col = col + 1) begin
            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
```

Generate din: To do …

- Zero padding: Pixels at invalid indexes are assigned to zeros.
  - col == 0
    - col-1 is an invalid index
  - col == WIDTH-1
    - col+1 is an invalid index

First row

```
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
```

Last row

```
else if (row == HEIGHT-1) 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] = in_img[(row-1) * WIDTH + col-1]        ;
        din[1*WI+:WI] = in_img[(row-1) * WIDTH + col  ]        ;
        din[2*WI+:WI] = 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] = 8'd0/*in_img[(row+1) * WIDTH + col-1]*/;
        din[7*WI+:WI] = 8'd0/*in_img[(row+1) * WIDTH + col  ]*/;
        din[8*WI+:WI] = 8'd0/*in_img[(row+1) * WIDTH + col+1]*/;
    end
end
```

Generate din: To do …

- Zero padding: Pixels at invalid indexes are assigned to zeros.
  - col == 0
    - col-1 is an invalid index
  - col == WIDTH-1
    - col+1 is an invalid index

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

BMP image writer (bmp_image_writer.v)

- Write an BMP file from incoming pixels
- Ports:
  - Inputs: din, vld
  - Output: frame_done
- Incoming pixels are stored in a buffer
  - out_img[0:FRAME_SIZE-1]).
  - Internal counter (pixel_counter) is updated

```
module bmp_image_writer
#(parameter WI = 8,
parameter BMP_HEADER_NUM = 54,
parameter WIDTH      = 128,
parameter HEIGHT     = 128,
parameter OUTFILE    = "./out/convout.bmp")(
    input clk,
    input rstn,
    input [WI-1:0] din,
    input vld,
    output reg frame_done
);
```

```
//------------------------------------------------
// Update the internal buffers.
//------------------------------------------------
always@(posedge clk, negedge rstn) begin
    if(!rstn) begin
        for(k=0;k<FRAME_SIZE;k=k+1) begin
            out_img[k] <= 0;
        end
        pixel_count <= 0;
        frame_done <= 1'b0;
    end else begin
        if(vld) begin
            if(pixel_count == FRAME_SIZE-1) begin
                pixel_count <= 0;
                frame_done <= 1'b1;
            end
            else begin
                pixel_count <= pixel_count + 1;
            end
            out_img[pixel_count] <= din;
        end
    end
end
```

BMP image writer (bmp_image_writer.v)

- When all pixels are stored in a buffer,
  - frame_done == 1
  - write an BMP image file

```
// Write the output file
//{{{
initial begin
    // Open file
    fd = $fopen(OUTFILE, "wb+");
    h = 0;
    w = 0;
end

always@(frame_done) begin
    if(frame_done == 1'b1) begin
    // Write header
        for(i=0; i<BMP_HEADER_NUM; i=i+1) begin
            $fwrite(fd, "%c", BMP_header[i][7:0]);
        end

        // Write data
        for(h = 0; h < HEIGHT; h = h + 1) begin
            for(w = 0; w < WIDTH; w = w + 1) begin
                $fwrite(fd, "%c", out_img[(HEIGHT-1-h)*WIDTH + w][7:0]);
                $fwrite(fd, "%c", out_img[(HEIGHT-1-h)*WIDTH + w][7:0]);
                $fwrite(fd, "%c", out_img[(HEIGHT-1-h)*WIDTH + w][7:0]);
            end
        end
        $fclose(fd);

    end
end
//}}}
```

Open a file

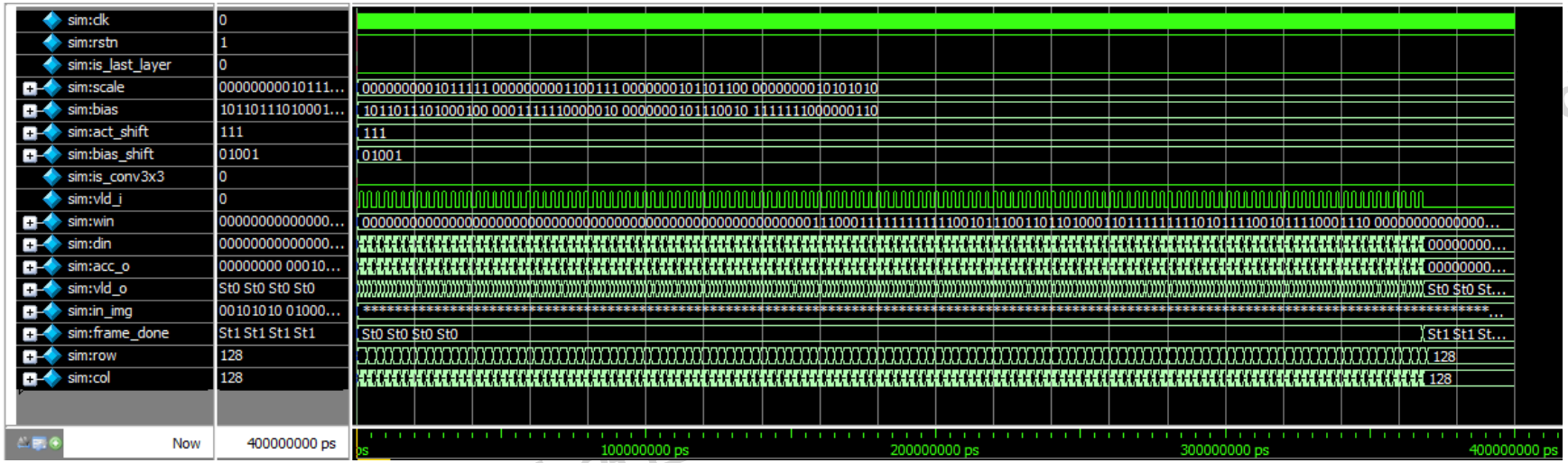
Write Header

Write data

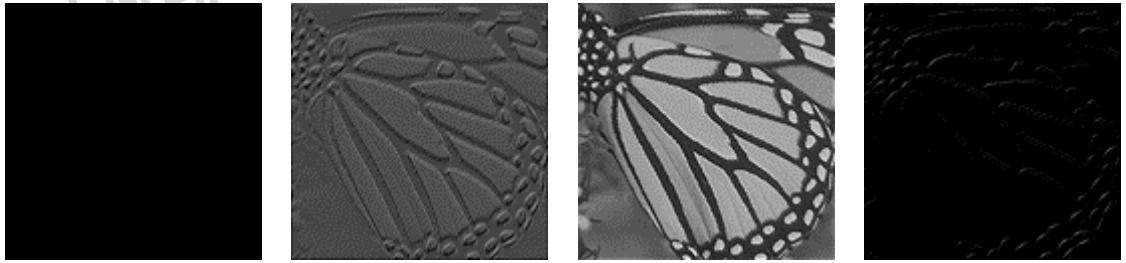
```
//------------------------------------------------
// Save the output image
//------------------------------------------------
initial begin
    IW = WIDTH;
    IH = HEIGHT;
    SZ = FRAME_SIZE + BMP_HEADER_NUM;
    BMP_header[ 0] = 66;
    BMP_header[ 1] = 77;
    BMP_header[ 2] = ((SZ & 32'h000000ff) >>  0);
    BMP_header[ 3] = ((SZ & 32'h0000ff00) >>  8);
    BMP_header[ 4] = ((SZ & 32'h00ff0000) >> 16);
    BMP_header[ 5] = ((SZ & 32'hff000000) >> 24);
    BMP_header[ 6] =  0;
    BMP_header[ 7] =  0;
    BMP_header[ 8] =  0;
    BMP_header[ 9] =  0;
    BMP_header[10] = 54;
    BMP_header[11] =  0;
    BMP_header[12] =  0;
    BMP_header[13] =  0;
    BMP_header[14] = 40;
    BMP_header[15] =  0;
    BMP_header[16] =  0;
    BMP_header[17] =  0;
    BMP_header[18] =  ((IW & 32'h000000ff) >>  0);
    BMP_header[19] =  ((IW & 32'h0000ff00) >>  8);
    BMP_header[20] =  ((IW & 32'h00ff0000) >> 16);
    BMP_header[21] =  ((IW & 32'hff000000) >> 24);
    BMP_header[22] =  ((IH & 32'h000000ff) >>  0);
    BMP_header[23] =  ((IH & 32'h0000ff00) >>  8);
    BMP_header[24] =  ((IH & 32'h00ff0000) >> 16);
    BMP_header[25] =  ((IH & 32'hff000000) >> 24);
```

Simulation results

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



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



ch01 ch02 ch03 ch04

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!
>>
```

Road map

Review

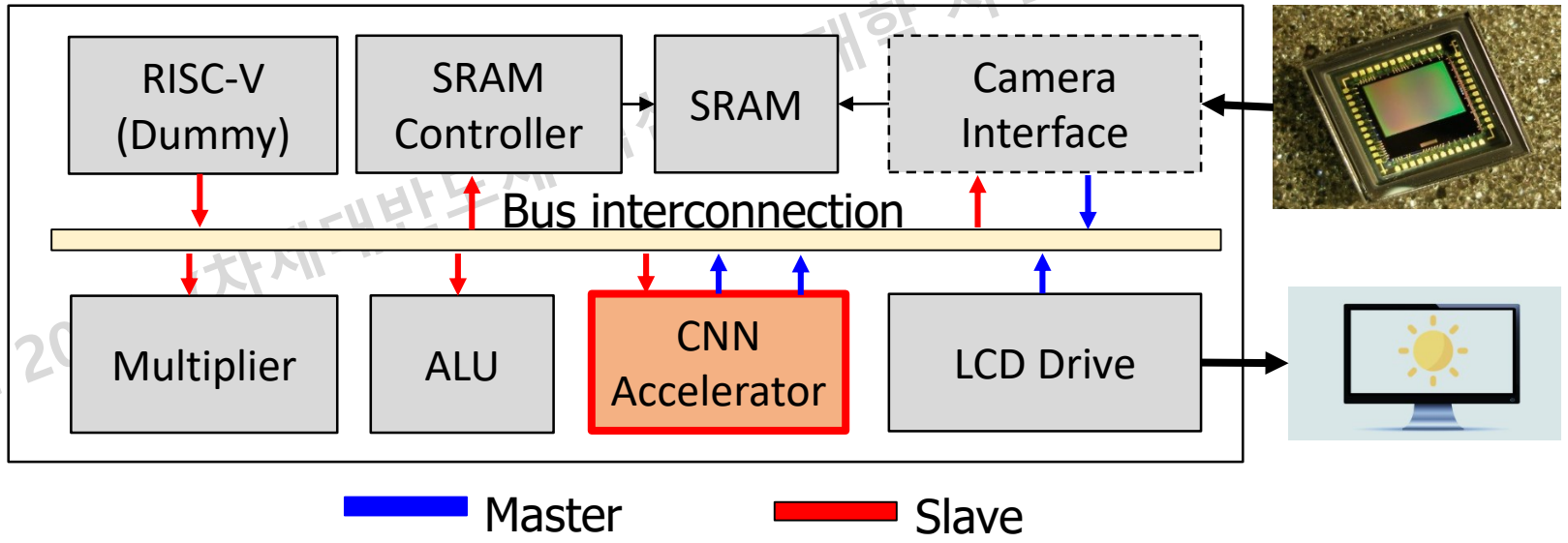
Sliding window

CNN accelerator IP
(AHB interface, FSM)

CNN accelerator IP

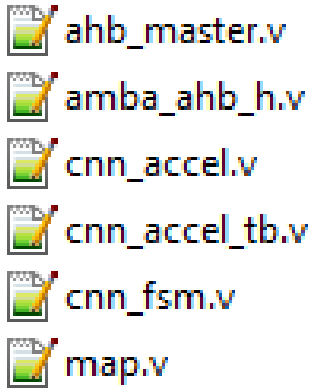
System block diagram

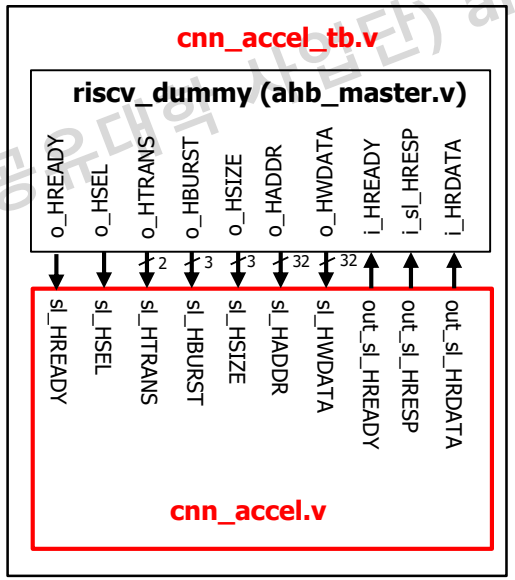
- System with a bus architecture:
  - Master (RISC-V), Slave Memory: SRAM
  - Custom IPs: Multiplier, ALU (single transfer)
  - Input: Camera Interface → Assume that an image is stored in memory
  - Output: LCD Drive
  - CNN accelerator



Lab 2: A simple system (cnn_accel_tb.v)

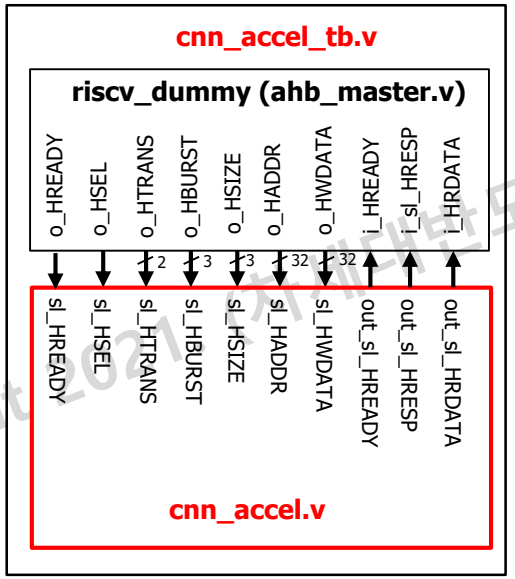
- Lab 2: Build and test a CNN accelerator (cnn_accel_tb.v)
  - A simple system:
    - Master: RISC-V (ahb_master.v)
    - Slave: A CNN accelerator IP (cnn_accel.v)
      - Finite state machine (cnn_fsm.v)





CNN accelerator IP (cnn_accel.v)

- AHB slave ports
  - AHB slave interface
  - An AHB master can read/write configuration registers in a slave via this port
- Questions: What are configuration registers?




```
module cnn_accel #(
    parameter W_ADDR = 32,
    parameter W_DATA = 32,
    parameter WB_DATA = 4,
    parameter W_WB_DATA = 2,
    parameter DEF_HPROT = {`PROT_NOTCACHE, `PROT_UNBUF, `PROT_USER, `PROT_DATA},
    parameter WIDTH      = 128,
    parameter HEIGHT     = 128,
    parameter START_UP_DELAY = 200,
    parameter HSYNC_DELAY = 160,
    parameter INFILE     = "./img/butterfly_08bit.hex",
    parameter OUTFILE00    = "./out/convout_ch01.bmp",
    parameter OUTFILE01    = "./out/convout_ch02.bmp",
    parameter OUTFILE02    = "./out/convout_ch03.bmp",
    parameter OUTFILE03    = "./out/convout_ch04.bmp")
(
    //CLOCK
    HCLK,
    HRESETn,
    //input signals of control port(slave)
    sl_HREADY,
    sl_HSEL,
    sl_HTRANS,
    sl_HBURST,
    sl_HSIZE,
    sl_HADDR,
    sl_HWRITE,
    sl_HWDATA,
    //output signals of control port(slave)
    out_sl_HREADY,
    out_sl_HRESP,
    out_sl_HRDATA
);
```

CNN accelerator IP (cnn_accel.v)

- Construct registers and register map based on parameters
  - Width, height, delays, frame size (data count), layer index and start.

```
// AHB signals
localparam N_REGS = 10;
localparam W_REGS = $clog2(N_REGS);
localparam CNN_ACCEL_FRAME_SIZE         = 0;    // WIDTH * HEIGHT
localparam CNN_ACCEL_WIDTH_HEIGHT       = 1;    // 0:15 -> WIDTH, 16:31: HEIGHT
localparam CNN_ACCEL_DELAY_PARAMS       = 2;    // 0~11: start_up, 12~23: hsync,
localparam CNN_ACCEL_BASE_ADDRESS       = 3;    // 0~19: weight; 20~31: param (sc
localparam CNN_ACCEL_LAYER_CONFIG       = 4;    // 0: is_first_layer, 1: q_is_las
                                                // 4~7: layer_index, 8~12: bias_s
                                                // 16~31: Reserved
localparam CNN_ACCEL_INPUT_IMAGE        = 5;
localparam CNN_ACCEL_INPUT_IMAGE_BASE   = 6;    // DMA: Base address for the inpu
localparam CNN_ACCEL_INPUT_IMAGE_LOAD   = 7;    // DMA: Start
localparam CNN_ACCEL_LAYER_START        = 8;    // Start
localparam CNN_ACCEL_LAYER_DONE         = 9;    // Done
```

```
// Configuration registers
reg [W_REGS-1:0]        q_sel_sl_reg;
reg                     q_ld_sl_reg;
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 [3:0]    q_layer_index;
reg          q_is_first_layer;
reg          q_is_last_layer;
reg          q_start;
reg          q_act_type;            // 0: RELU, 1: Linear
reg          q_is_conv3x3;          // 1: 3x3 conv, 0: 1x1
reg [ 2:0]   q_act_shift;           // Activation shift
reg [ 4:0]   q_bias_shift;          // Bias Shift (before
reg [19:0]   q_base_addr_weight;
reg [11:0]   q_base_addr_param;
reg          q_layer_done;
```

Map (map.v)

- Add the mapping addresses to map.v
  - Register maps of Slaves are visible to Masters

```
//------------------------------------------------------------
// Base Address
//------------------------------------------------------------
`define RISCV_ALU_BASE_ADDR             32'hE000_0000
`define RISCV_MULTIPLIER_BASE_ADDR      32'hE000_1000
`define RISCV_LCD_DRIVE_BASE_ADDR       32'hE100_0000
`define RISCV_LCD_DRIVE_IMG_OFFSET      32'h0010_0000
`define RISCV_MEMORY_BASE_ADDR          32'hE200_0000
`define RISCV_CNN_ACCEL_BASE_ADDR       32'hE300_0000

//------------------------------------------------------------
// CNN Accelerator                  map.v
//------------------------------------------------------------
`define CNN_ACCEL_FRAME_SIZE        (`RISCV_CNN_ACCEL_BASE_ADDR + 32'h00)
`define CNN_ACCEL_WIDTH_HEIGHT      (`RISCV_CNN_ACCEL_BASE_ADDR + 32'h04)
`define CNN_ACCEL_DELAY_PARAMS      (`RISCV_CNN_ACCEL_BASE_ADDR + 32'h08)
`define CNN_ACCEL_BASE_ADDRESS      (`RISCV_CNN_ACCEL_BASE_ADDR + 32'h0C)
`define CNN_ACCEL_LAYER_CONFIG      (`RISCV_CNN_ACCEL_BASE_ADDR + 32'h10)
`define CNN_ACCEL_INPUT_IMAGE       (`RISCV_CNN_ACCEL_BASE_ADDR + 32'h14)
`define CNN_ACCEL_INPUT_IMAGE_BASE  (`RISCV_CNN_ACCEL_BASE_ADDR + 32'h18)
`define CNN_ACCEL_INPUT_IMAGE_LOAD  (`RISCV_CNN_ACCEL_BASE_ADDR + 32'h1C)
`define CNN_ACCEL_LAYER_START       (`RISCV_CNN_ACCEL_BASE_ADDR + 32'h20)
`define CNN_ACCEL_LAYER_DONE        (`RISCV_CNN_ACCEL_BASE_ADDR + 32'h24)
```

```
// Configuration registers
reg [W_REGS-1:0]         q_sel_sl_reg;
reg                      q_ld_sl_reg;
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 [3:0]    q_layer_index;
reg          q_is_first_layer;
reg          q_is_last_layer;
reg          q_start;
reg          q_act_type;              // 0: RELU, 1: Linear
reg          q_is_conv3x3;            // 1: 3x3 conv, 0: 1x1
reg [ 2:0]   q_act_shift;             // Activation shift
reg [ 4:0]   q_bias_shift;            // Bias Shift (before
reg [19:0]   q_base_addr_weight;
reg [11:0]   q_base_addr_param;
reg          q_layer_done;
```

Address decoder (cnn_accel.v)

- At each AHB slave IP, use an OFFSET to access a register
  - Offset is 4 bytes, so it ignores W_WB_DATA LSB bits
  - Access all registers, so it needs W_REGS bits.
    - Example: W_REGS = 4 to access 10 configuration registers

```
//------------------------------------------------------------
// Decode Stage: Address Phase
//------------------------------------------------------------
always @(posedge HCLK or negedge HRESETn)
begin
    if(~HRESETn)
    begin
        //control
        q_sel_sl_reg <= 0;
        q_ld_sl_reg <= 1'b0;
        q_input_pixel_addr <= 0;
    end
    else begin
        if(sl_HSEL && sl_HREADY && ((sl_HTRANS == `TRANS_NONSEQ) || (sl_HTRANS == `TRANS_SEQ)))
        begin
            q_sel_sl_reg        <= sl_HADDR[W_REGS+W_WB_DATA-1:W_WB_DATA];
            q_ld_sl_reg         <= sl_HWRITE;
            q_input_pixel_addr  <= sl_HADDR[(2+FRAME_SIZE_W+W_REGS+W_WB_DATA-1):(2+W_REGS+W_WB_DATA)];
        end
        else begin
            q_ld_sl_reg <= 1'b0;
        end
    end
end
```

```
// AHB signals
localparam N_REGS = 10;
localparam W_REGS = $clog2(N_REGS);
localparam CNN_ACCEL_FRAME_SIZE         = 0;
localparam CNN_ACCEL_WIDTH_HEIGHT       = 1;
localparam CNN_ACCEL_DELAY_PARAMS       = 2;
localparam CNN_ACCEL_BASE_ADDRESS       = 3;
localparam CNN_ACCEL_LAYER_CONFIG       = 4;


localparam CNN_ACCEL_INPUT_IMAGE        = 5;
localparam CNN_ACCEL_INPUT_IMAGE_BASE   = 6;
localparam CNN_ACCEL_INPUT_IMAGE_LOAD   = 7;
localparam CNN_ACCEL_LAYER_START        = 8;
localparam CNN_ACCEL_LAYER_DONE         = 9;
```

Data phase: Write operation

- Write operation
  - q_ld_sl_reg = 1 (WRITE)
  - Select a target register for update by its address (q_sel_sl_reg)
- For example
  - If q_sel_sl_reg == CNN_ACCEL_WIDTH_HEIGHT
  - Copy an input data (sl_HWDATA) into the register q_width, q_height

```
//data-transfer state(data phase)
if(q_ld_sl_reg)
begin
    case(q_sel_sl_reg)
        CNN_ACCEL_FRAME_SIZE:   q_frame_size <= sl_HWDATA[W_FRAME_SIZE-1 :0];
        CNN_ACCEL_WIDTH_HEIGHT: begin
            q_width     <= sl_HWDATA[W_SIZE-1 :0];
            q_height    <= sl_HWDATA[(W_SIZE+16-1):16];
        end
        CNN_ACCEL_DELAY_PARAMS: begin
            q_start_up_delay    <= sl_HWDATA[W_DELAY-1 :0];
            q_hsync_delay       <= sl_HWDATA[2*W_DELAY-1:W_DELAY];
        end
        CNN_ACCEL_BASE_ADDRESS: begin
            q_base_addr_weight  <= sl_HWDATA[19: 0];
            q_base_addr_param   <= sl_HWDATA[31:20];
        end
        CNN_ACCEL_LAYER_CONFIG: begin
            //q_is_first_layer  <= /*Insert your code*/;
            //q_is_last_layer   <= /*Insert your code*/;
            //q_is_conv3x3      <= /*Insert your code*/;
            //q_act_type        <= /*Insert your code*/;
            //q_layer_index     <= /*Insert your code*/;
            //q_bias_shift      <= /*Insert your code*/;
            //q_act_shift       <= /*Insert your code*/;
        end
        CNN_ACCEL_INPUT_IMAGE:      q_input_pixel_data <= sl_HWDATA;
        CNN_ACCEL_INPUT_IMAGE_BASE: q_input_image_base_addr <= sl_HWDATA;
        CNN_ACCEL_INPUT_IMAGE_LOAD: q_input_image_load <= sl_HWDATA[0];
        CNN_ACCEL_LAYER_START:  q_start <= sl_HWDATA[0];
        CNN_ACCEL_LAYER_DONE:   q_layer_done <= sl_HWDATA[0];
    endcase
end
```

Data phase: Read operation

- Read operation
  - Select a target register for update by its address (q_sel_sl_reg)
- For example
  - If q_sel_sl_reg == LCD_DRIVE_WIDTH
    - Copy q_width to out_sl_HRDATA

```
assign out_sl_HREADY = 1'b1;
assign out_sl_HRESP = `RESP_OKAY;
always @*
begin:rdata
    case(q_sel_sl_reg)
        CNN_ACCEL_FRAME_SIZE:       out_sl_HRDATA = q_frame_size;
        CNN_ACCEL_WIDTH_HEIGHT:     out_sl_HRDATA = {q_height &16'hFFFF,q_width &16'hFFFF};
        CNN_ACCEL_DELAY_PARAMS:     out_sl_HRDATA = {q_hsync_delay,q_start_up_delay};
        CNN_ACCEL_BASE_ADDRESS:     out_sl_HRDATA = {q_base_addr_param & 12'hFFF, q_base_addr_weight & 20'hFFFFF};
        CNN_ACCEL_LAYER_CONFIG:     out_sl_HRDATA = {q_act_shift, q_bias_shift, q_layer_index, q_act_type, q_is_conv3x3, q_is_last_layer, q_is_first_layer};
        CNN_ACCEL_INPUT_IMAGE:      out_sl_HRDATA = q_input_pixel_data;
        CNN_ACCEL_INPUT_IMAGE_BASE: out_sl_HRDATA = q_input_image_base_addr;
        CNN_ACCEL_INPUT_IMAGE_LOAD: out_sl_HRDATA = load_image_done;
        CNN_ACCEL_LAYER_START:      out_sl_HRDATA = q_start;
        CNN_ACCEL_LAYER_DONE:       out_sl_HRDATA = layer_done;
        default: out_sl_HRDATA = 32'h0;
    endcase
end
```

CNN controller: Finite state machine (FSM)

- CNN controller (cnn_fsm.v)
  - Inputs
    - Clock, reset
    - Layer type: q_is_conv3x3
    - Size: q_width, q_height, q_frame_size
    - Frame delay: q_start_up_delay
    - Line delay: q_hsync_delay
    - Start signals: q_start
  - Output
    - o_ctrl_vsync_run, o_ctrl_vsync_cnt
    - o_ctrl_hsync_run, o_ctrl_hsync_cnt
    - o_ctrl_data_run, o_row, o_col, o_data_count
    - o_end_frame

```
module cnn_fsm (
clk,
rstn,
// Inputs
q_is_conv3x3,
q_width,
q_height,
q_start_up_delay,
q_hsync_delay,
q_frame_size,
q_start,
//output
o_ctrl_vsync_run,
o_ctrl_vsync_cnt,
o_ctrl_hsync_run,
o_ctrl_hsync_cnt,
o_ctrl_data_run,
o_row,
o_col,
o_data_count,
o_end_frame
);
```

```
parameter W_SIZE  = 12;                  //
parameter W_FRAME_SIZE  = 2 * W_SIZE + 1;
parameter W_DELAY = 12;
input clk, rstn;
input q_is_conv3x3;
input [W_SIZE-1 :0] q_width;
input [W_SIZE-1 :0] q_height;
input [W_DELAY-1:0] q_start_up_delay;
input [W_DELAY-1:0] q_hsync_delay;
input [W_FRAME_SIZE-1:0] q_frame_size;
input q_start;

output                      o_ctrl_vsync_run;
output [W_DELAY-1:0]        o_ctrl_vsync_cnt;
output                      o_ctrl_hsync_run;
output [W_DELAY-1:0]        o_ctrl_hsync_cnt;
output                      o_ctrl_data_run;
output [W_SIZE-1:0]         o_row;
output [W_SIZE-1:0]         o_col;
output [W_FRAME_SIZE-1:0]o_data_count;
output o_end_frame;
```

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

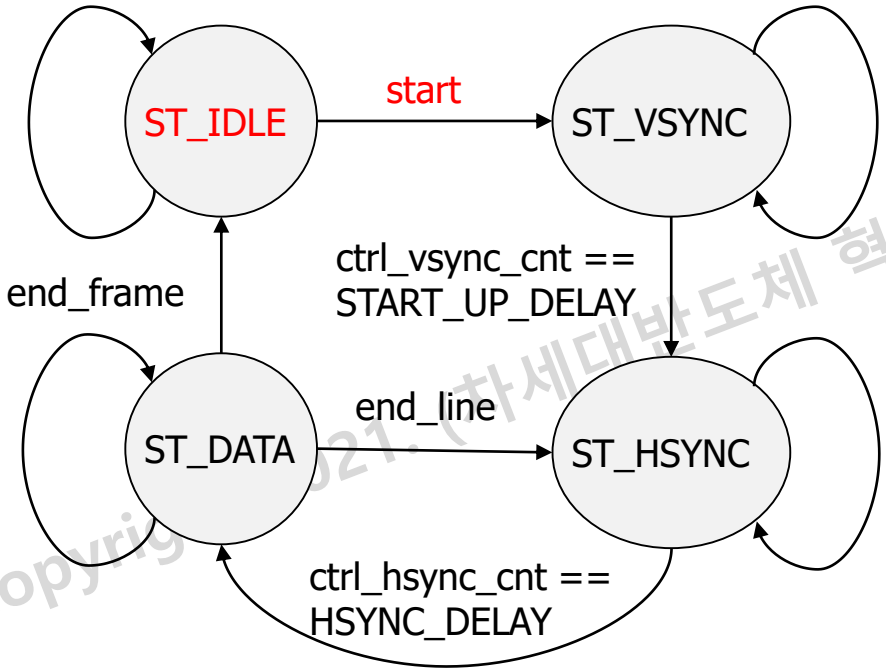
```
always @ (posedge clk, negedge rstn)
begin
    if(~rstn) begin
        cstate <= ST_IDLE;
    end
    else begin
        cstate <= nstate;
    end
end

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.

Finite State Machine

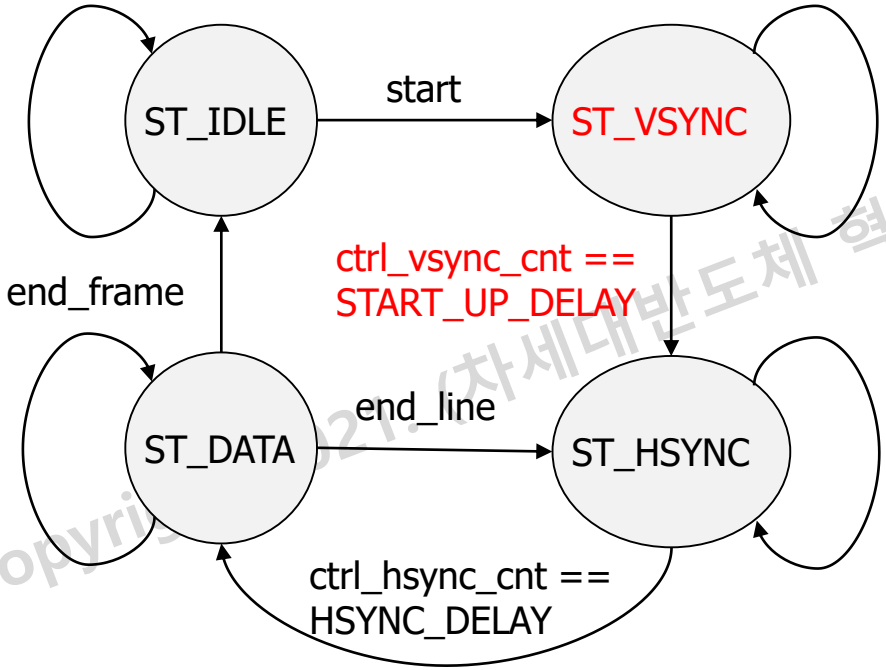
- ST_IDLE:
  - FSM is initialized at ST_IDLE
  - When "start" goes HIGH, FSM moves to ST_VSYNC.




```
always @(*) begin
    case(cstate)
        ST_IDLE: begin
                if(start)
                    nstate = ST_VSYNC;
                else
                    nstate = ST_IDLE;
            end
        ST_VSYNC: begin
                if(ctrl_vsync_cnt == q_start_up_delay)
                    nstate = ST_HSYNC;
                else
                    nstate = ST_VSYNC;
            end
        ST_HSYNC: begin
                //if(ctrl_hsync_cnt == /* Insert your code here */)
                //    nstate = /* Insert your code here */;
                //else
                //    nstate = ST_HSYNC;
            end
        ST_DATA: begin
                if(ctrl_done) begin   //end of frame
                    //nstate = /* Insert your code here */;
                end
                else begin
                    if(col == /* Insert your code here */)    //end of line
                        nstate = ST_HSYNC;
                    else
                        nstate = ST_DATA;
                end
            end
        default: nstate = ST_IDLE;
    endcase
end
```

Finite State Machine

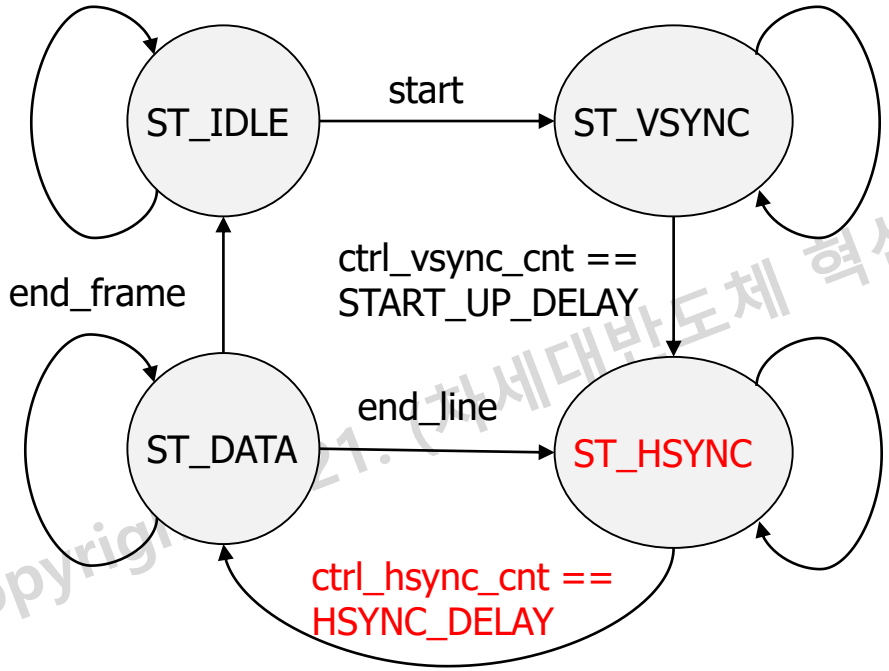
- ST_VSYNC: Frame synchronization
  - Start up for a frame
  - There is an VSYNC counter.
  - When the counter reaches to START_UP_DELAY, FSM moves to ST_HSYNC.




```
always @(*) begin
    case(cstate)
        ST_IDLE: begin
                if(start)
                    nstate = ST_VSYNC;
                else
                    nstate = ST_IDLE;
        end
          ST_VSYNC: begin
                    if(ctrl_vsync_cnt == q_start_up_delay)
                        nstate = ST_HSYNC;
                    else
                        nstate = ST_VSYNC;
          end
        ST_HSYNC: begin
                //if(ctrl_hsync_cnt == /* Insert your code here */)
                //    nstate = /* Insert your code here */;
                //else
                //    nstate = ST_HSYNC;
        end
        ST_DATA: begin
                if(ctrl_done) begin   //end of frame
                     //nstate = /* Insert your code here */;
                end
                else begin
                    if(col == /* Insert your code here */)    //end of line
                        nstate = ST_HSYNC;
                    else
                        nstate = ST_DATA;
                end
        end
        default: nstate = ST_IDLE;
    endcase
end
```

Finite State Machine

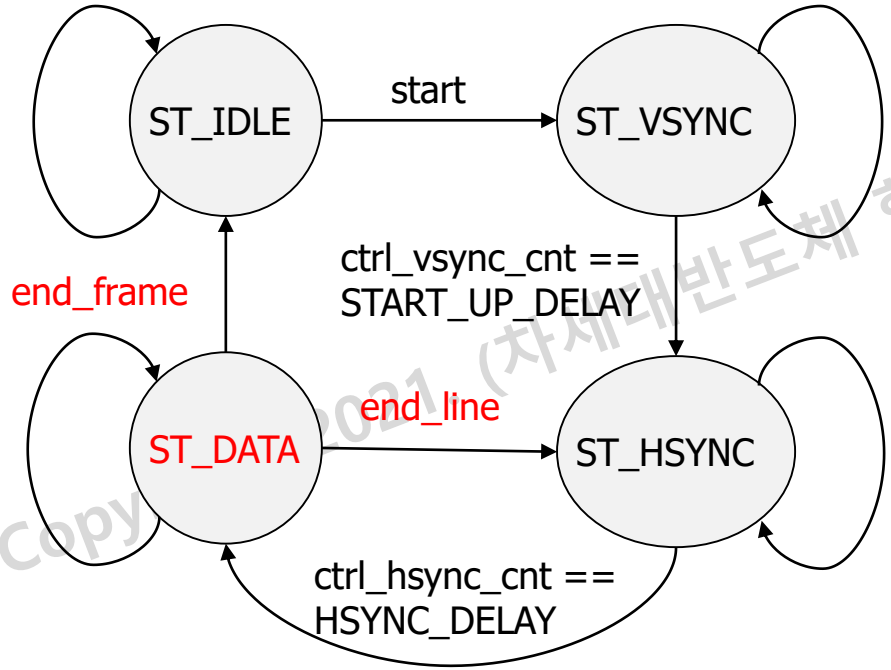
- ST_HSYNC: Line synchronization
  - There is an HSYNC counter.
  - When the counter reaches to HSYNC_DELAY, FSM moves to ST_DATA.




```
always @(*) begin
   case(cstate)
      ST_IDLE: begin
            if(start)
               nstate = ST_VSYNC;
            else
               nstate = ST_IDLE;
      end
      ST_VSYNC: begin
            if(ctrl_vsync_cnt == q_start_up_delay)
               nstate = ST_HSYNC;
            else
               nstate = ST_VSYNC;
      end
        ST_HSYNC: begin
                  //if(ctrl_hsync_cnt == /* Insert your code here */)
                  //    nstate = /* Insert your code here */;
                  //else
                  //    nstate = ST_HSYNC;
        end
      ST_DATA: begin
            if(ctrl_done) begin   //end of frame
                 //nstate = /* Insert your code here */;
            end
            else begin
               if(col == /* Insert your code here */)    //end of line
                  nstate = ST_HSYNC;
               else
                  nstate = ST_DATA;
            end
      end
      default: nstate = ST_IDLE;
   endcase
end
```

Finite State Machine

- ST_DATA: Sending pixel data
  - There are two counters
  - Line data counter: if it reaches to end of line, FSM moves to ST_HSYNC.
  - Frame data counter: if it reaches to end of frame, FSM moves to ST_IDLE.




```
always @(*) begin
    case(cstate)
        ST_IDLE: begin
                if(start)
                    nstate = ST_VSYNC;
                else
                    nstate = ST_IDLE;
        end
        ST_VSYNC: begin
                if(ctrl_vsync_cnt == q_start_up_delay)
                    nstate = ST_HSYNC;
                else
                    nstate = ST_VSYNC;
        end
        ST_HSYNC: begin
                //if(ctrl_hsync_cnt == /* Insert your code here */)
                //    nstate = /* Insert your code here */;
                //else
                //    nstate = ST_HSYNC;
        end
          ST_DATA: begin
                    if(ctrl_done) begin   //end of frame
                        //nstate = /* Insert your code here */;
                    end
                    else begin
                        //if(col == /* Insert your code here */;)    //end of line
                            nstate = ST_HSYNC;
                        else
                            nstate = ST_DATA;
                    end
          end
        default: nstate = ST_IDLE;
    endcase
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

Slave

```
//------------------------------------------------
// 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)
     );
```

Test bench (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 bench (cnn_accel_tb.v)

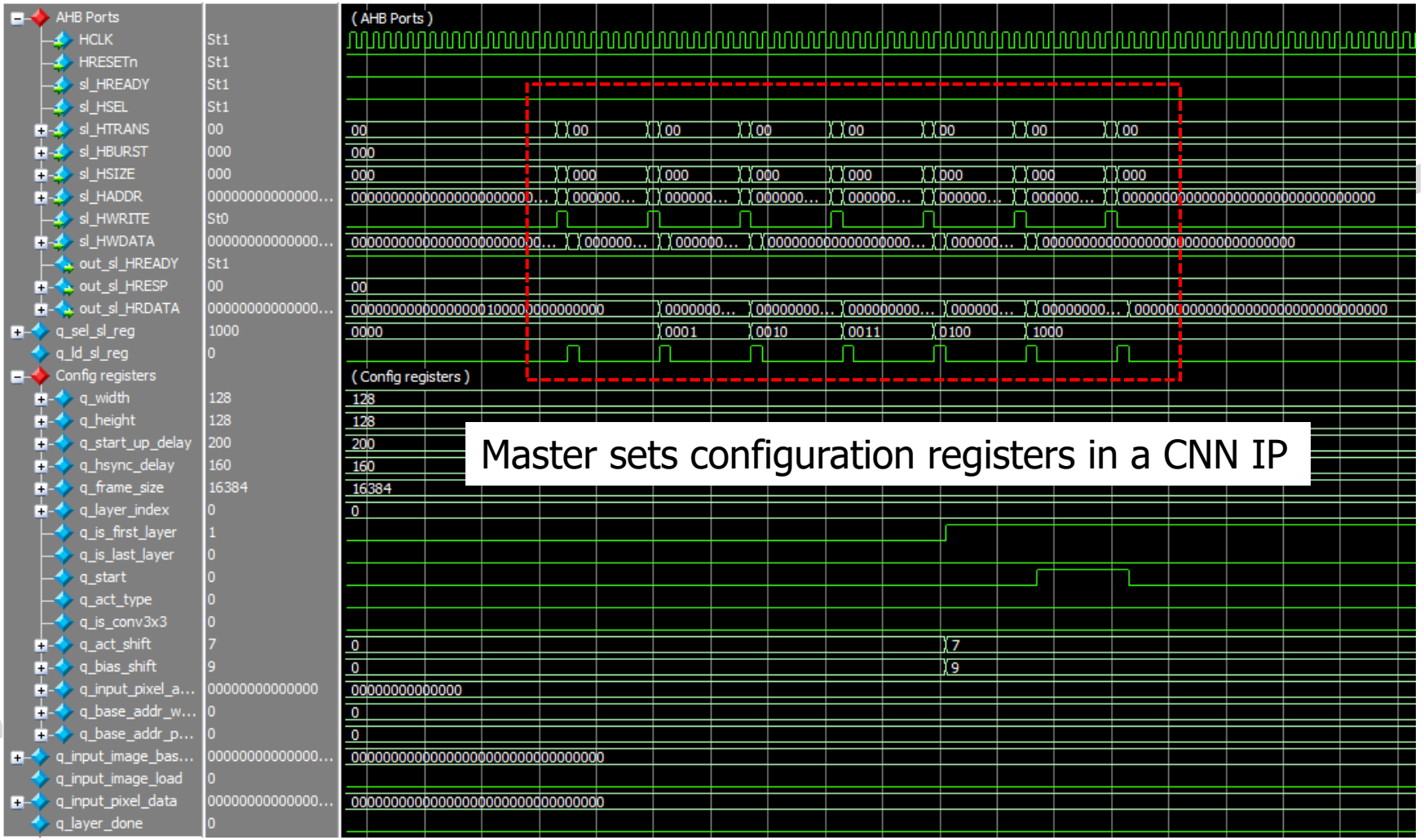
```
// Weight/bias/Scale base addresses
base_addr_weight    = 0;
base_addr_param     = 0;

// 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'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});

//*****************************************************************************************************
// 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_laye
#(4*p) @(posedge HCLK)  u_riscv_dummy.task_AHBwrite(`CNN_ACCEL_BASE_ADDRESS, {base_addr_param&12'hFFF,base_addr_weight&20'hFFFFF});
#(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 register

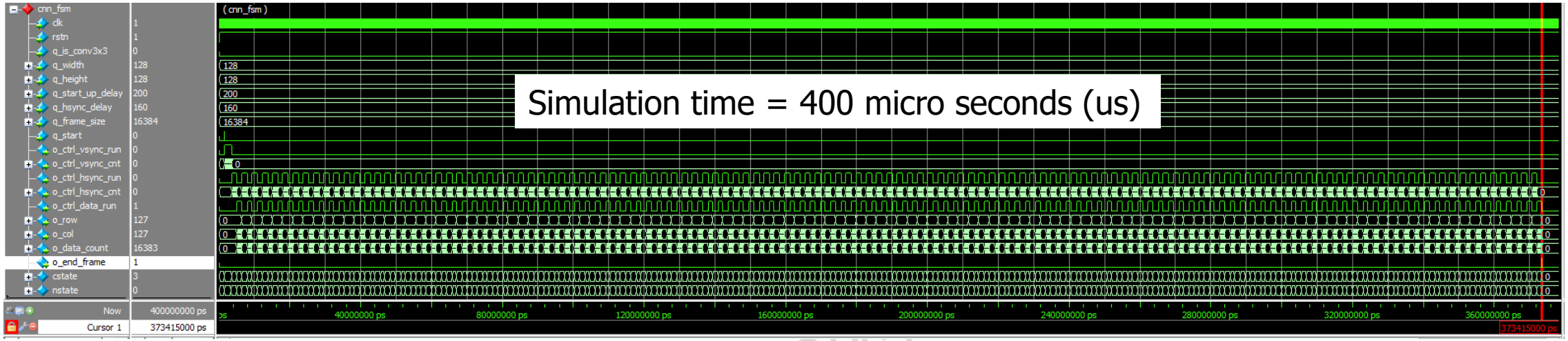
Configures a layer's configuration register

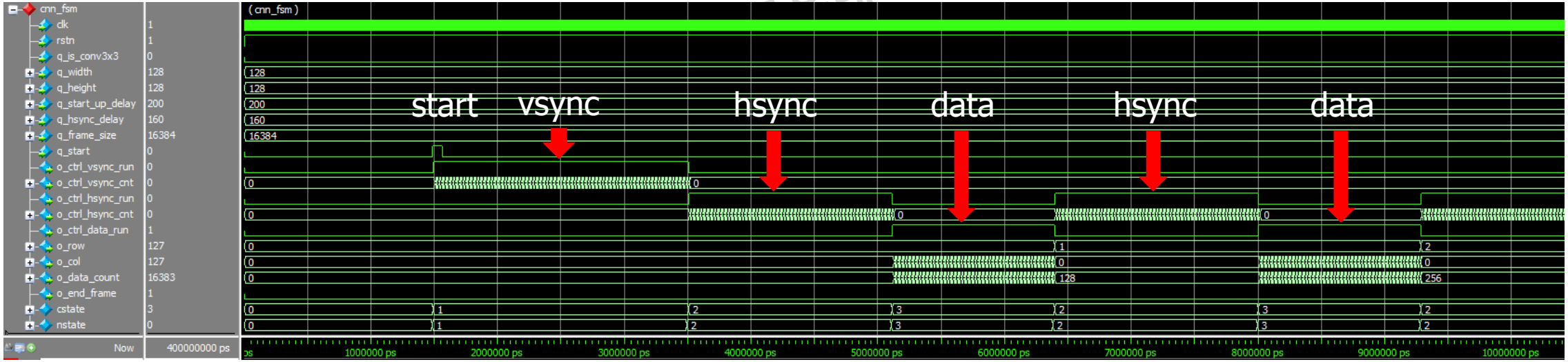
Send a start signal

Waveform



Waveform: Finite state machines





To do …

- Complete the missing codes
  - cnn_accel.v
  - cnn_fsm.v
- Do a simulation with time = 400 us
- Show/capture the waveform

```
                end
                //if(col == /*Insert your code*/)
                //  col <= 0;
                //else
                //  col <= col + 1;
            end
        end
    end
    always@(posedge clk, negedge rstn)
    begin
        if(!rstn) begin
            data_count <= 0;
        end
        else begin
            if(ctrl_data_run) begin
                if(!end_frame)
                    data_count <= data_count + 1;
                else
                    data_count <= 0;
            end
        end
    end
    //assign end_frame = (data_count == /*Insert your code*/)? 1'b1: 1'b0;
```

Output column index

End frame

FSM

```
always @(*) begin
    case(cstate)
    ST_IDLE: begin
                //if(/*Insert your code*/)
                //    nstate = ST_VSYNC;
                //else
                //    nstate = ST_IDLE;
        end
        ST_VSYNC: begin
                if(ctrl_vsync_cnt == q_start_up_delay)
                    nstate = ST_HSYNC;
                else
                    nstate = ST_VSYNC;
        end
        ST_HSYNC: begin
                //if(ctrl_hsync_cnt == /*Insert your code*/)
                //    nstate = ST_DATA;
                //else
                //    nstate = ST_HSYNC;
        end
        ST_DATA: begin
                if(end_frame)     //end of frame
                    nstate = ST_IDLE;
                else begin
                    //if(col == /*Insert your code*/)//end of line
                    //    nstate = ST_HSYNC;
                    //else
                    //    nstate = ST_DATA;
                end
        end
        default: nstate = ST_IDLE;
    endcase
end
```

Road map

Review

Sliding window

CNN accelerator IP
(AHB interface, FSM)

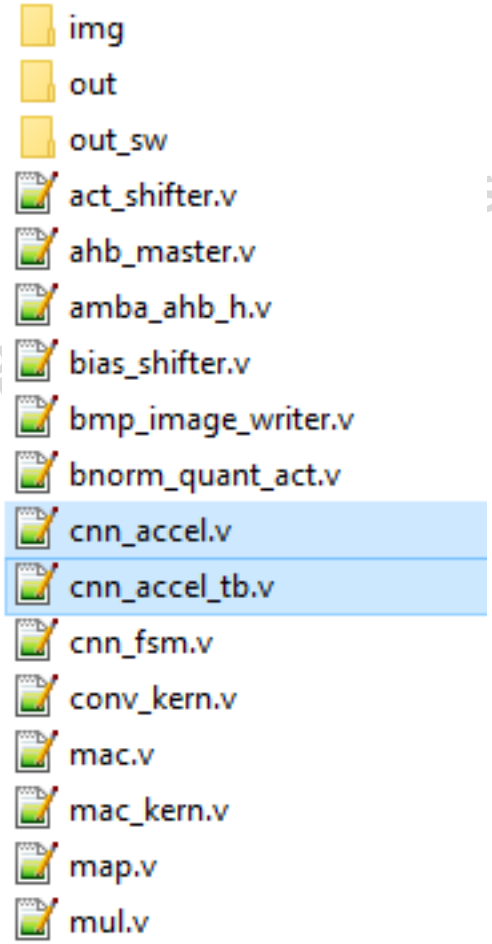
CNN accelerator IP

Lab 3: CNN Accelerator IP

- Lab 3:
  - Reuse code in Lab 2
    - Set configuration registers
    - cnn_fsm.v
  - Add data path for the CNN accelerator IP
  - Do simulation
  - Verify the outputs

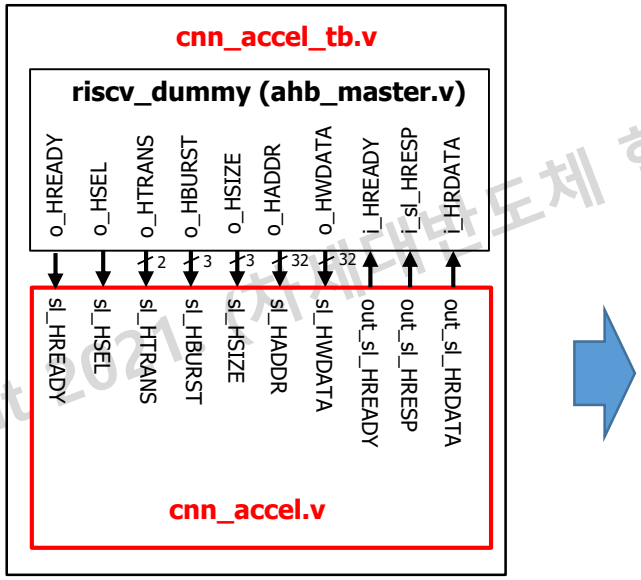
Lab 3: Code structure

- 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
- img/: input image in hex file
- out/: output results by H/W simulation
- out_sw/: outputs by S/W simulation



CNN accelerator IP (cnn_accel.v)

- AHB slave ports
  - AHB slave interface
  - An AHB master can read/write configuration registers in a slave via this port




```
module cnn_accel #(
    parameter W_ADDR = 32,
    parameter W_DATA = 32,
    parameter WB_DATA = 4,
    parameter W_WB_DATA = 2,
    parameter DEF_HPROT = {`PROT_NOTCACHE, `PROT_UNBUF, `PROT_USER, `PROT_DATA},
    parameter WIDTH      = 128,
    parameter HEIGHT     = 128,
    parameter START_UP_DELAY = 200,
    parameter HSYNC_DELAY = 160,
    parameter INFILE     = "./img/butterfly_08bit.hex",
    parameter OUTFILE00    = "./out/convout_ch01.bmp",
    parameter OUTFILE01    = "./out/convout_ch02.bmp",
    parameter OUTFILE02    = "./out/convout_ch03.bmp",
    parameter OUTFILE03    = "./out/convout_ch04.bmp")
(
    //CLOCK
    HCLK,
    HRESETn,
    //input signals of control port(slave)
    sl_HREADY,
    sl_HSEL,
    sl_HTRANS,
    sl_HBURST,
    sl_HSIZE,
    sl_HADDR,
    sl_HWRITE,
    sl_HWDATA,
    //output signals of control port(slave)
    out_sl_HREADY,
    out_sl_HRESP,
    out_sl_HRDATA
);
```

CNN accelerator IP (cnn_accel.v)

- Construct registers and register map based on parameters
  - Width, height, delays, frame size (data count), layer index and start.

```
// AHB signals
localparam N_REGS = 10;
localparam W_REGS = $clog2(N_REGS);
localparam CNN_ACCEL_FRAME_SIZE         = 0;    // WIDTH * HEIGHT
localparam CNN_ACCEL_WIDTH_HEIGHT       = 1;    // 0:15 -> WIDTH, 16:31: HEIGHT
localparam CNN_ACCEL_DELAY_PARAMS       = 2;    // 0~11: start_up, 12~23: hsync,
localparam CNN_ACCEL_BASE_ADDRESS       = 3;    // 0~19: weight; 20~31: param (sc
localparam CNN_ACCEL_LAYER_CONFIG       = 4;    // 0: is_first_layer, 1: q_is_las
                                                // 4~7: layer_index, 8~12: bias_s
                                                // 16~31: Reserved
localparam CNN_ACCEL_INPUT_IMAGE        = 5;
localparam CNN_ACCEL_INPUT_IMAGE_BASE   = 6;    // DMA: Base address for the inpu
localparam CNN_ACCEL_INPUT_IMAGE_LOAD   = 7;    // DMA: Start
localparam CNN_ACCEL_LAYER_START        = 8;    // Start
localparam CNN_ACCEL_LAYER_DONE         = 9;    // Done
```

```
// Configuration registers
reg [W_REGS-1:0]          q_sel_sl_reg;
reg                       q_ld_sl_reg;
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 [3:0]    q_layer_index;
reg          q_is_first_layer;
reg          q_is_last_layer;
reg          q_start;
reg          q_act_type;              // 0: RELU, 1: Linear
reg          q_is_conv3x3;            // 1: 3x3 conv, 0: 1x1
reg [ 2:0]   q_act_shift;             // Activation shift
reg [ 4:0]   q_bias_shift;            // Bias Shift (before
reg [19:0]   q_base_addr_weight;
reg [11:0]   q_base_addr_param;
reg          q_layer_done;
```

Map (map.v)

- Add the mapping addresses to map.v
  - Register maps of Slaves are visible to Masters

```
//------------------------------------------------------------
// Base Address
//------------------------------------------------------------
`define RISCV_ALU_BASE_ADDR             32'hE000_0000
`define RISCV_MULTIPLIER_BASE_ADDR      32'hE000_1000
`define RISCV_LCD_DRIVE_BASE_ADDR       32'hE100_0000
`define RISCV_LCD_DRIVE_IMG_OFFSET      32'h0010_0000
`define RISCV_MEMORY_BASE_ADDR          32'hE200_0000
`define RISCV_CNN_ACCEL_BASE_ADDR       32'hE300_0000

//------------------------------------------------------------
// CNN Accelerator                  map.v
//------------------------------------------------------------
`define CNN_ACCEL_FRAME_SIZE        (`RISCV_CNN_ACCEL_BASE_ADDR + 32'h00)
`define CNN_ACCEL_WIDTH_HEIGHT      (`RISCV_CNN_ACCEL_BASE_ADDR + 32'h04)
`define CNN_ACCEL_DELAY_PARAMS      (`RISCV_CNN_ACCEL_BASE_ADDR + 32'h08)
`define CNN_ACCEL_BASE_ADDRESS      (`RISCV_CNN_ACCEL_BASE_ADDR + 32'h0C)
`define CNN_ACCEL_LAYER_CONFIG      (`RISCV_CNN_ACCEL_BASE_ADDR + 32'h10)
`define CNN_ACCEL_INPUT_IMAGE       (`RISCV_CNN_ACCEL_BASE_ADDR + 32'h14)
`define CNN_ACCEL_INPUT_IMAGE_BASE  (`RISCV_CNN_ACCEL_BASE_ADDR + 32'h18)
`define CNN_ACCEL_INPUT_IMAGE_LOAD  (`RISCV_CNN_ACCEL_BASE_ADDR + 32'h1C)
`define CNN_ACCEL_LAYER_START       (`RISCV_CNN_ACCEL_BASE_ADDR + 32'h20)
`define CNN_ACCEL_LAYER_DONE        (`RISCV_CNN_ACCEL_BASE_ADDR + 32'h24)
```

```
// Configuration registers
reg [W_REGS-1:0]         q_sel_sl_reg;
reg                      q_ld_sl_reg;
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 [3:0]    q_layer_index;
reg          q_is_first_layer;
reg          q_is_last_layer;
reg          q_start;
reg          q_act_type;              // 0: RELU, 1: Linear
reg          q_is_conv3x3;            // 1: 3x3 conv, 0: 1x1
reg [ 2:0]   q_act_shift;             // Activation shift
reg [ 4:0]   q_bias_shift;            // Bias Shift (before
reg [19:0]   q_base_addr_weight;
reg [11:0]   q_base_addr_param;
reg          q_layer_done;
```

CNN controller: Finite state machine (FSM)

- CNN controller (cnn_fsm.v)
  - Inputs
    - Clock, reset
    - Layer type: q_is_conv3x3
    - Size: q_width, q_height, q_frame_size
    - Frame delay: q_start_up_delay
    - Line delay: q_hsync_delay
    - Start signals: q_start
  - Output
    - o_ctrl_vsync_run, o_ctrl_vsync_cnt
    - o_ctrl_hsync_run, o_ctrl_hsync_cnt
    - o_ctrl_data_run, o_row, o_col, o_data_count
    - o_end_frame

```
module cnn_fsm (
clk,
rstn,
// Inputs
q_is_conv3x3,
q_width,
q_height,
q_start_up_delay,
q_hsync_delay,
q_frame_size,
q_start,
//output
o_ctrl_vsync_run,
o_ctrl_vsync_cnt,
o_ctrl_hsync_run,
o_ctrl_hsync_cnt,
o_ctrl_data_run,
o_row,
o_col,
o_data_count,
o_end_frame
);
```

```
parameter W_SIZE  = 12;                  //
parameter W_FRAME_SIZE  = 2 * W_SIZE + 1;
parameter W_DELAY = 12;
input clk, rstn;
input q_is_conv3x3;
input [W_SIZE-1 :0] q_width;
input [W_SIZE-1 :0] q_height;
input [W_DELAY-1:0] q_start_up_delay;
input [W_DELAY-1:0] q_hsync_delay;
input [W_FRAME_SIZE-1:0] q_frame_size;
input q_start;

output                      o_ctrl_vsync_run;
output [W_DELAY-1:0]        o_ctrl_vsync_cnt;
output                      o_ctrl_hsync_run;
output [W_DELAY-1:0]        o_ctrl_hsync_cnt;
output                      o_ctrl_data_run;
output [W_SIZE-1:0]         o_row;
output [W_SIZE-1:0]         o_col;
output [W_FRAME_SIZE-1:0]o_data_count;
output o_end_frame;
```

CNN accelerator IP (cnn_accel.v)

- Add data path
  - in_img: a buffer to store an input image
  - win, scale, bias: weights, scales, biases
  - din: data input for convolution kernels
  - acc_o, vld_o: output pixels and valid signals
- To = 4
  - Four convolutional kernels

```
// Convolutional signals
reg [WI-1:0] in_img [0:FRAME_SIZE-1];   // Input image
reg [Ti*WI-1:0] win[0:To-1];            // Weight
reg [Ti*WI-1:0] din;                    // Input block data
reg vld_i;                              // Input valid signal
wire [ACT_BITS-1:0] acc_o[0:To-1];      // Output block data
wire vld_o[0:To-1];                     // Output valid signal
reg [PARAM_BITS-1:0] scale[0:To-1];     // Scales (Batch normalization)
reg [PARAM_BITS-1:0] bias[0:To-1];      // Biases
wire frame_done[0:To-1];
```

```
//------------------------------------------------------------
// Computing units
//------------------------------------------------------------
conv_kern u_conv_kern_00(
./*input                  */clk(clk),
./*input                  */rstn(rstn),
./*input                  */is_last_layer(q_is_last_layer),
./*input [PARAM_BITS-1:0]*/scale(scale[0]),
./*input [PARAM_BITS-1:0]*/bias(bias[0]),
./*input [2:0]            */act_shift(q_act_shift),
./*input [4:0]            */bias_shift(q_bias_shift),
./*input                  */is_conv3x3(q_is_conv3x3),
./*input                  */vld_i(vld_i),
./*input [N*WI-1:0]       */win(win[0]),
./*input [N*WI-1:0]       */din(din),
./*output [ACT_BITS-1:0] */acc_o(acc_o[0]),
./*output                 */vld_o(vld_o[0])
);

conv_kern u_conv_kern_01(
./*input                  */clk(clk),
./*input                  */rstn(rstn),
./*input                  */is_last_layer(q_is_last_layer),
./*input [PARAM_BITS-1:0]*/scale(scale[1]),
./*input [PARAM_BITS-1:0]*/bias(bias[1]),
./*input [2:0]            */act_shift(q_act_shift),
./*input [4:0]            */bias_shift(q_bias_shift),
./*input                  */is_conv3x3(q_is_conv3x3),
./*input                  */vld_i(vld_i),
./*input [N*WI-1:0]       */win(win[1]),
./*input [N*WI-1:0]       */din(din),
./*output [ACT_BITS-1:0] */acc_o(acc_o[1]),
./*output                 */vld_o(vld_o[1])
);
```

Convolutional kernels and image writers

```
//------------------------------------------------------------
// Computing units
//------------------------------------------------------------
conv_kern u_conv_kern_00(
./*input                  */clk(clk),
./*input                  */rstn(rstn),
./*input                  */is_last_layer(q_is_last_layer),
./*input [PARAM_BITS-1:0] */scale(scale[0]),
./*input [PARAM_BITS-1:0] */bias(bias[0]),
./*input [2:0]            */act_shift(q_act_shift),
./*input [4:0]            */bias_shift(q_bias_shift),
./*input                  */is_conv3x3(q_is_conv3x3),
./*input                  */vld_i(vld_i),
./*input [N*WI-1:0]       */win(win[0]),
./*input [N*WI-1:0]       */din(din),
./*output [ACT_BITS-1:0]  */acc_o(acc_o[0]),
./*output                 */vld_o(vld_o[0])
);
```

conv_kern_00

```
conv_kern u_conv_kern_01(
./*input                  */clk(clk),
./*input                  */rstn(rstn),
./*input                  */is_last_layer(q_is_last_layer),
./*input [PARAM_BITS-1:0] */scale(scale[1]),
./*input [PARAM_BITS-1:0] */bias(bias[1]),
./*input [2:0]            */act_shift(q_act_shift),
./*input [4:0]            */bias_shift(q_bias_shift),
./*input                  */is_conv3x3(q_is_conv3x3),
./*input                  */vld_i(vld_i),
./*input [N*WI-1:0]       */win(win[1]),
./*input [N*WI-1:0]       */din(din),
./*output [ACT_BITS-1:0]  */acc_o(acc_o[1]),
./*output                 */vld_o(vld_o[1])
);
```

conv_kern_01

```
//------------------------------------------------------------
// Image Writer
//------------------------------------------------------------
bmp_image_writer#(.WIDTH(WIDTH),.HEIGHT(HEIGHT),.OUTFILE(OUTFILE00))
u_bmp_image_writer_00(
./*input              */clk(clk),
./*input              */rstn(rstn),
./*input [WI-1:0]     */din(acc_o[0]),
./*input              */vld(vld_o[0]),
./*output reg         */frame_done(frame_done[0])
);
```

```
bmp_image_writer#(.WIDTH(WIDTH),.HEIGHT(HEIGHT),.OUTFILE(OUTFILE01))
u_bmp_image_writer_01(
./*input              */clk(clk),
./*input              */rstn(rstn),
./*input [WI-1:0]     */din(acc_o[1]),
./*input              */vld(vld_o[1]),
./*output reg         */frame_done(frame_done[1])
);
```

Input image buffer (cnn_accel.v)

- An input image is stored in a hex file.
  - INFILE: img/butterfly_08bit.hex
- Read a file to a buffer
  - All input pixels are stored in a buffer
    - in_img[0:HEIGHT*WIDTH-1]

```
//----------------------------------------------
// Input feature buffer
//----------------------------------------------
initial begin
    $readmemh(INFILE, in_img ,0,FRAME_SIZE-1);
end

// Generate singals
// Weights, biases, scales
always @(posedge clk or negedge rstn)begin
    if(!rstn) begin
        //
        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
```

Scales and biases

- Four biases and four scales are preloaded to a buffer

Biases

test_vector{1, 3}

	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

test_vector{1, 4}

	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

```
//------------------------------------------------
// Input feature buffer
//------------------------------------------------
initial begin
    $readmemh(INFILE, in_img ,0,FRAME_SIZE-1);
end

// Generate singals
// Weights, biases, scales
always @(posedge clk or negedge rstn)begin
    if(!rstn) begin
        //
        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
```

Weights

- Preload four weights or filters into a buffer (win)

Reference 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
```

```
// 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;
win[0][9*WI+:WI] <= 8'd0;
win[0][10*WI+:WI] <= 8'd0;
win[0][11*WI+:WI] <= 8'd0;
win[0][12*WI+:WI] <= 8'd0;
win[0][13*WI+:WI] <= 8'd0;
win[0][14*WI+:WI] <= 8'd0;
win[0][15*WI+:WI] <= 8'd0;
```

```
// 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;
win[1][9*WI+:WI] <= 8'd0;
win[1][10*WI+:WI] <= 8'd0;
win[1][11*WI+:WI] <= 8'd0;
win[1][12*WI+:WI] <= 8'd0;
win[1][13*WI+:WI] <= 8'd0;
win[1][14*WI+:WI] <= 8'd0;
win[1][15*WI+:WI] <= 8'd0;
```

```
// First layer, channel 02:
win[2][0*WI+:WI] <= 8'd0;/*Insert your code*/
win[2][1*WI+:WI] <= 8'd0;/*Insert your code*/
win[2][2*WI+:WI] <= 8'd0;/*Insert your code*/
win[2][3*WI+:WI] <= 8'd0;/*Insert your code*/
win[2][4*WI+:WI] <= 8'd0;/*Insert your code*/
win[2][5*WI+:WI] <= 8'd0;/*Insert your code*/
win[2][6*WI+:WI] <= 8'd0;/*Insert your code*/
win[2][7*WI+:WI] <= 8'd0;/*Insert your code*/
win[2][8*WI+:WI] <= 8'd0;/*Insert your code*/
win[2][9*WI+:WI] <= 8'd0;/*Insert your code*/
win[2][10*WI+:WI] <= 8'd0;
win[2][11*WI+:WI] <= 8'd0;
win[2][12*WI+:WI] <= 8'd0;
win[2][13*WI+:WI] <= 8'd0;
win[2][14*WI+:WI] <= 8'd0;
win[2][15*WI+:WI] <= 8'd0;
```

```
// 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;
win[3][9*WI+:WI] <= 8'd0;
win[3][10*WI+:WI] <= 8'd0;
win[3][11*WI+:WI] <= 8'd0;
win[3][12*WI+:WI] <= 8'd0;
win[3][13*WI+:WI] <= 8'd0;
win[3][14*WI+:WI] <= 8'd0;
win[3][15*WI+:WI] <= 8'd0;
```

To do …

- Generate din, vld_i
- Hint: Check the algorithm in Lab 1

```
// 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 = /*insert your code*/;
        //din[0*WI+:WI] = /*Insert your code*/;
        //din[1*WI+:WI] = /*Insert your code*/;
        //din[2*WI+:WI] = /*Insert your code*/;
        //din[3*WI+:WI] = /*Insert your code*/;
        //din[4*WI+:WI] = /*Insert your code*/;
        //din[5*WI+:WI] = /*Insert your code*/;
        //din[6*WI+:WI] = /*Insert your code*/;
        //din[7*WI+:WI] = /*Insert your code*/;
        //din[8*WI+:WI] = /*Insert your code*/;
    end
end
```

```
//------------------------------------------------------------
// Computing units
//------------------------------------------------------------
conv_kern u_conv_kern_00(
./*input                  */clk(clk),
./*input                  */rstn(rstn),
./*input                  */is_last_layer(q_is_last_layer),
./*input [PARAM_BITS-1:0]*/scale(scale[0]),
./*input [PARAM_BITS-1:0]*/bias(bias[0]),
./*input [2:0]            */act_shift(q_act_shift),
./*input [4:0]            */bias_shift(q_bias_shift),
./*input                  */is_conv3x3(q_is_conv3x3),
./*input                  */vld_i(vld_i),
./*input [N*WI-1:0]       */win(win[0]),
./*input [N*WI-1:0]       */din(din),
./*output [ACT_BITS-1:0] */acc_o(acc_o[0]),
./*output                 */vld_o(vld_o[0])
);
```

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;
```

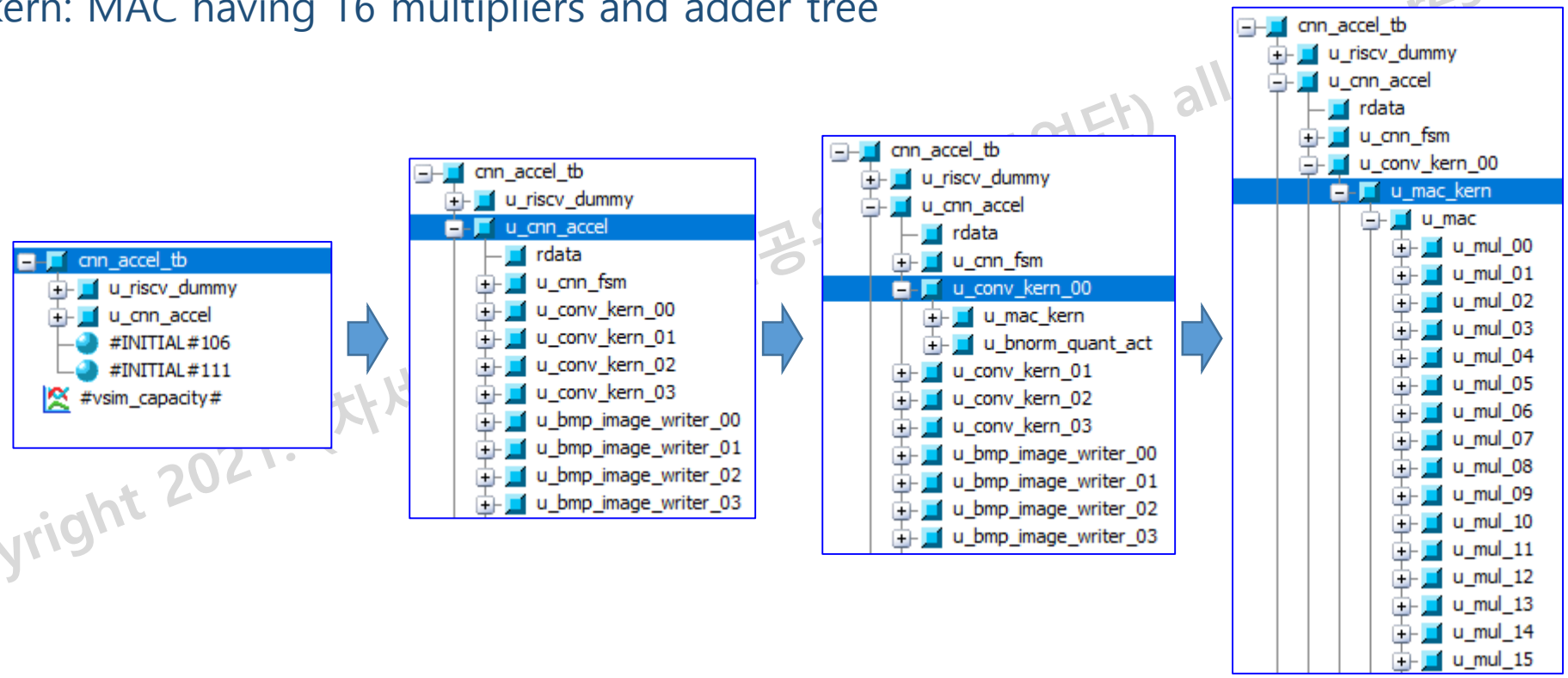
Master

Slave

```
//------------------------------------------------
// 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)
    );
```

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), four convolutional kernel (conv_kern), and four output writer
- cnn_kern: a MAC kernel (mac_kern) and a batch normalization and quantization (bnorm_quant_act)
- mac_kern: MAC having 16 multipliers and 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)

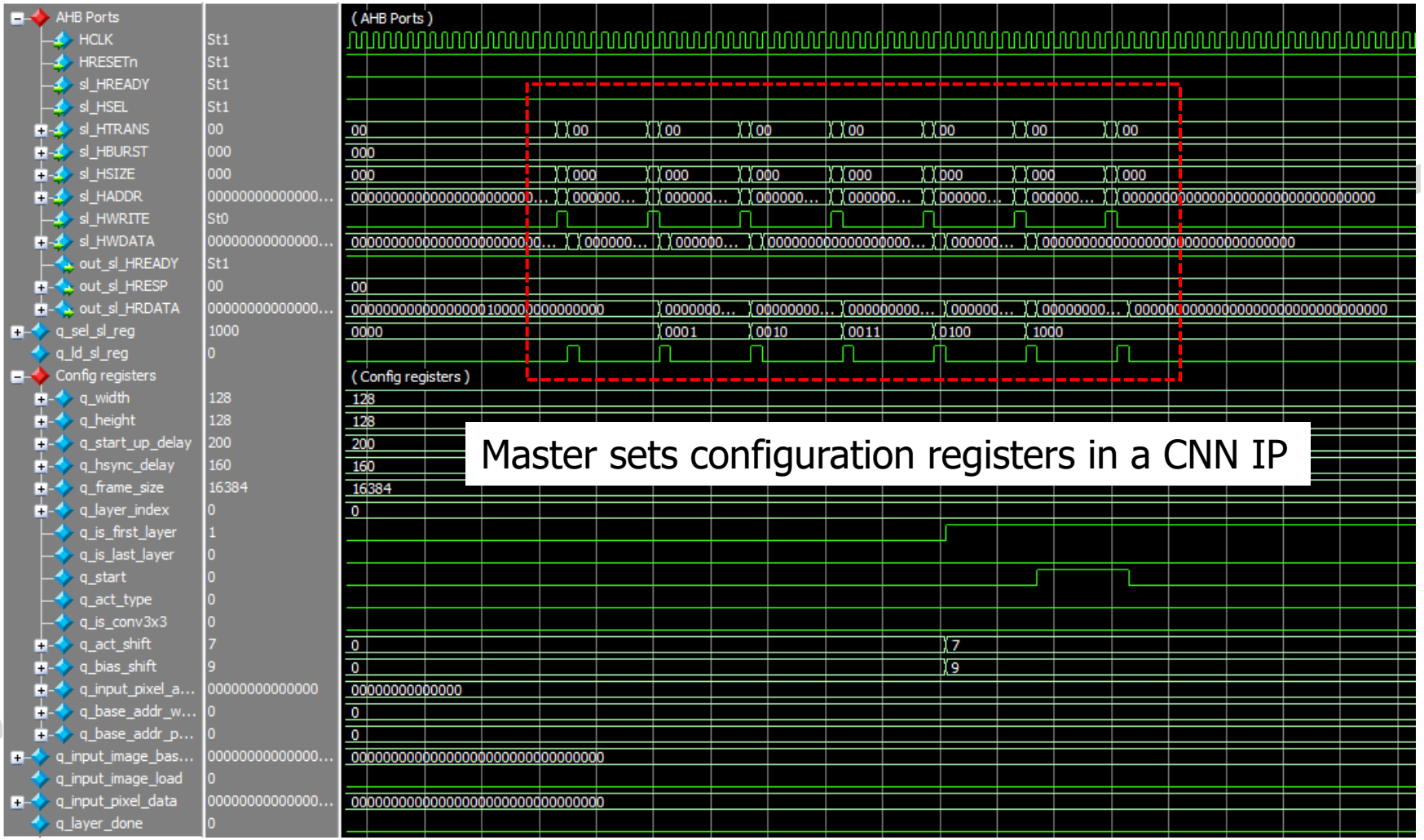
```
// Weight/bias/Scale base addresses
base_addr_weight    = 0;
base_addr_param     = 0;

// 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'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});

//*****************************************************************************************************
// 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_laye
#(4*p) @(posedge HCLK)  u_riscv_dummy.task_AHBwrite(`CNN_ACCEL_BASE_ADDRESS, {base_addr_param&12'hFFF,base_addr_weight&20'hFFFFF});
#(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 register

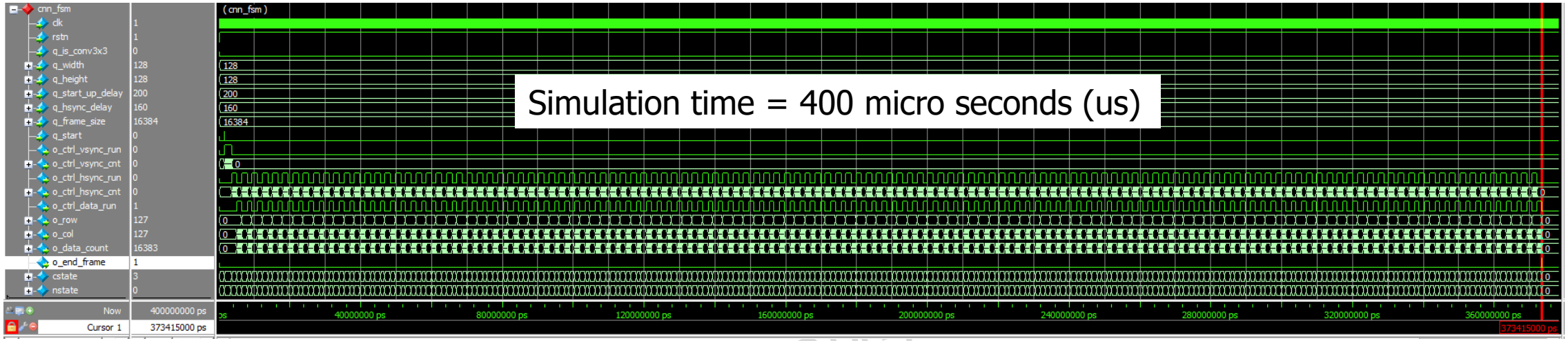
Configures a layer's configuration register

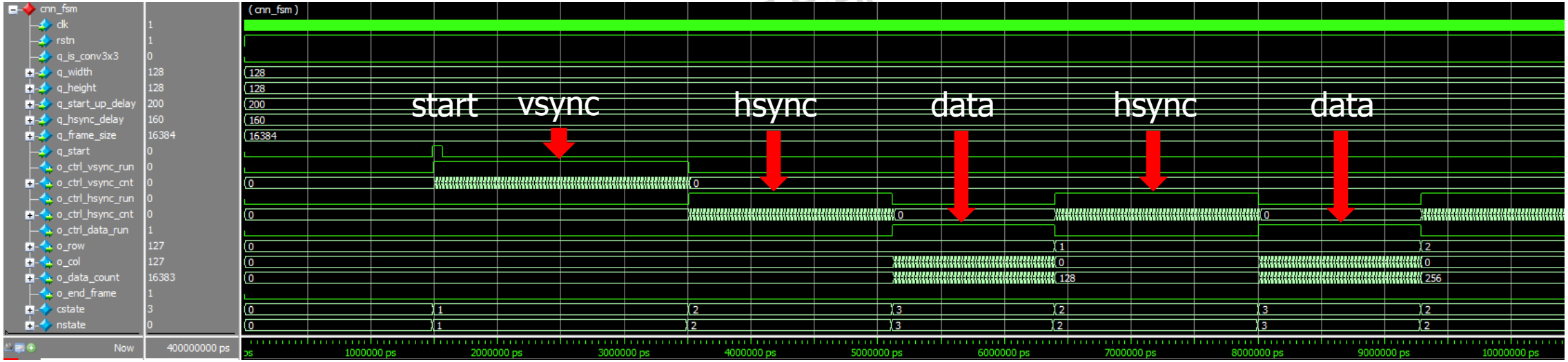
Send a start signal

Waveform



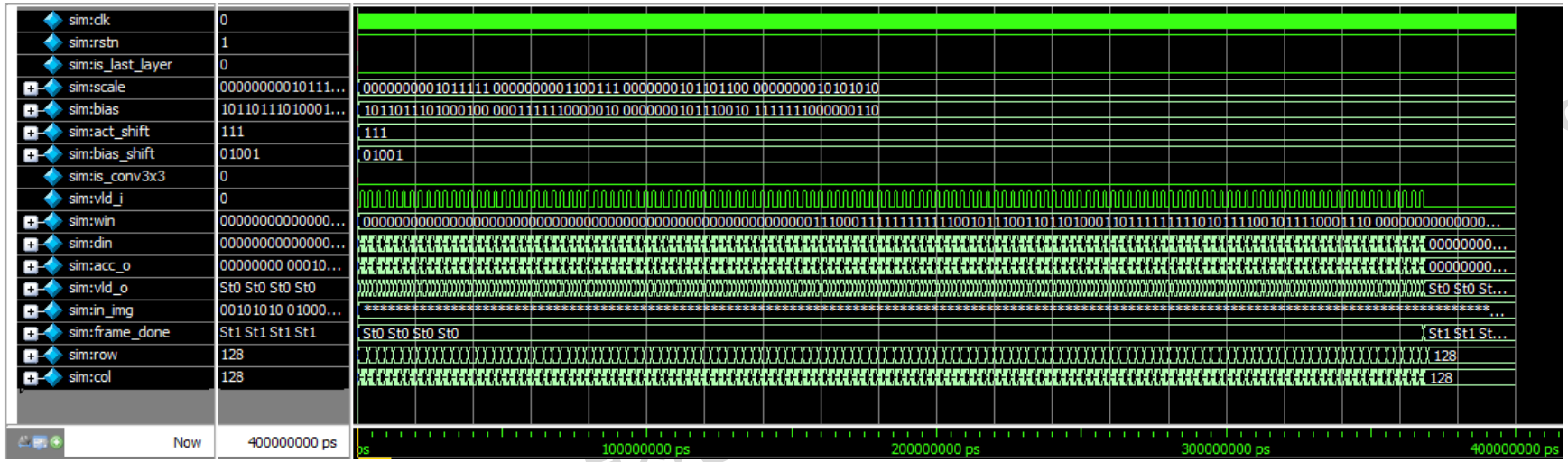
Waveform: Finite state machines



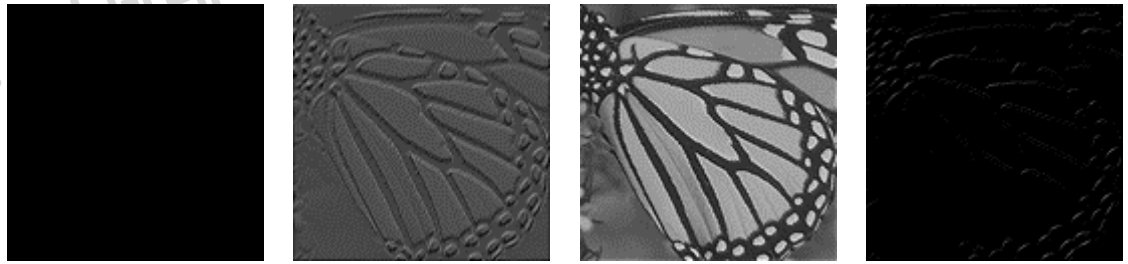


Simulation results

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



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

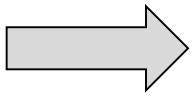


ch01 ch02 ch03 ch04

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 in Lab 2
  - cnn_accel.v
    - Reuse code for setting configuration registers in Lab 2
    - Add weights for the output channel 3
    - Generate din, vld_i
- Do a simulation with time = 400 us
- Show/capture the output results