차세대 반도체 디지털 혁신공유대학 사업단

POLARIS

CNN accelerator IP System integration, DMA

2023.1.19 (Thu)

Road map

Review

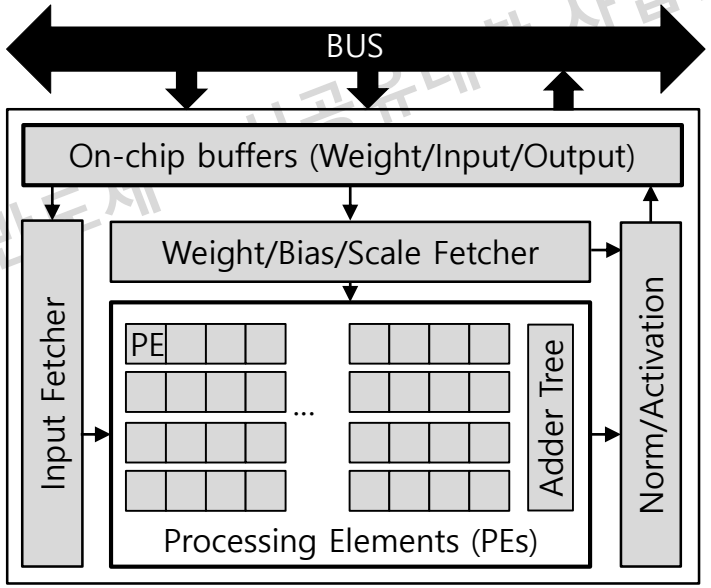
Control path, data path

System integration

Direct memory access (DMA)

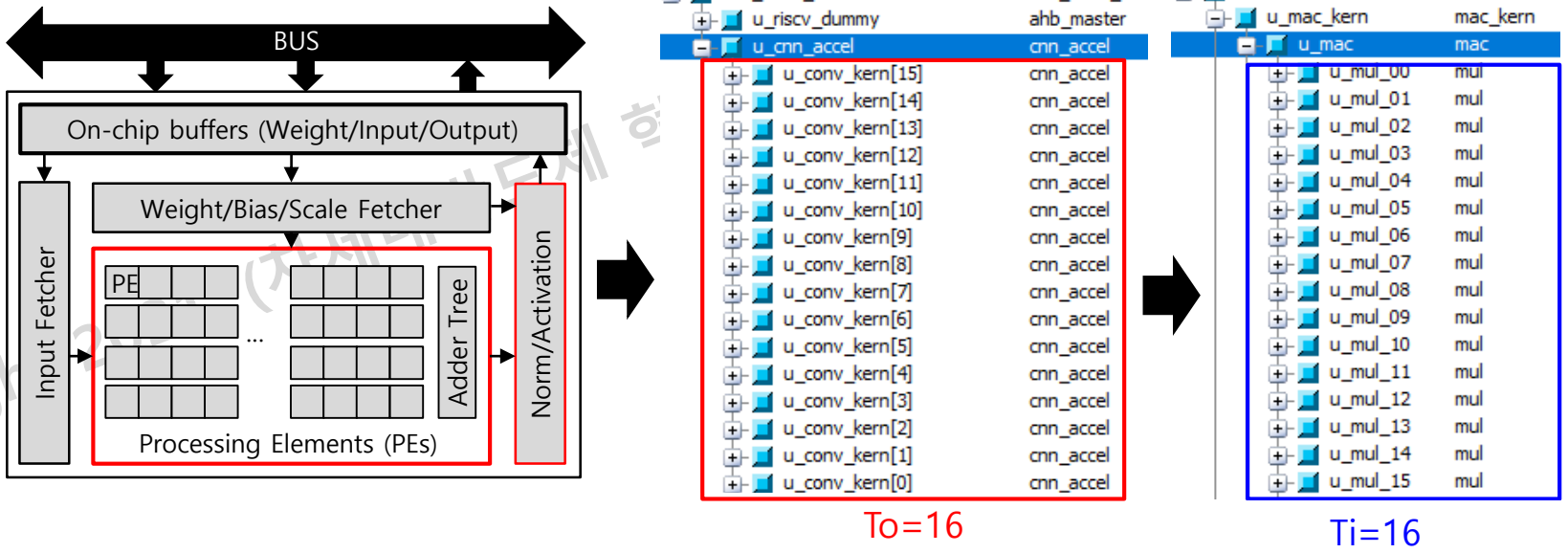
CNN accelerator

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



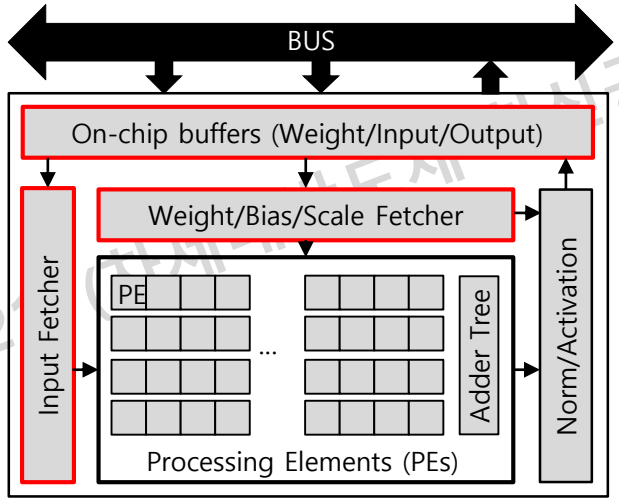
Processing Elements: ~ALU

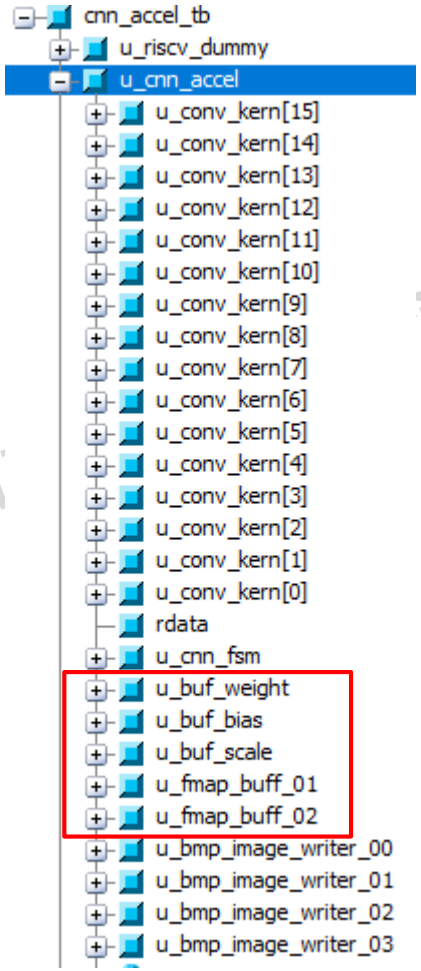
- Processing Element (PE) Array (conv_kern.v, mac_kern.v)
  - Perform convolution/activation/quantization operations.
  - To: The number of convolutional kernels (output feature maps)
  - Ti: The number of multipliers in a CONV kernel (conv_kern.v)
- The number of multipliers is : To× Ti
  - 256 (=16×16)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



Buffers: ~Register File

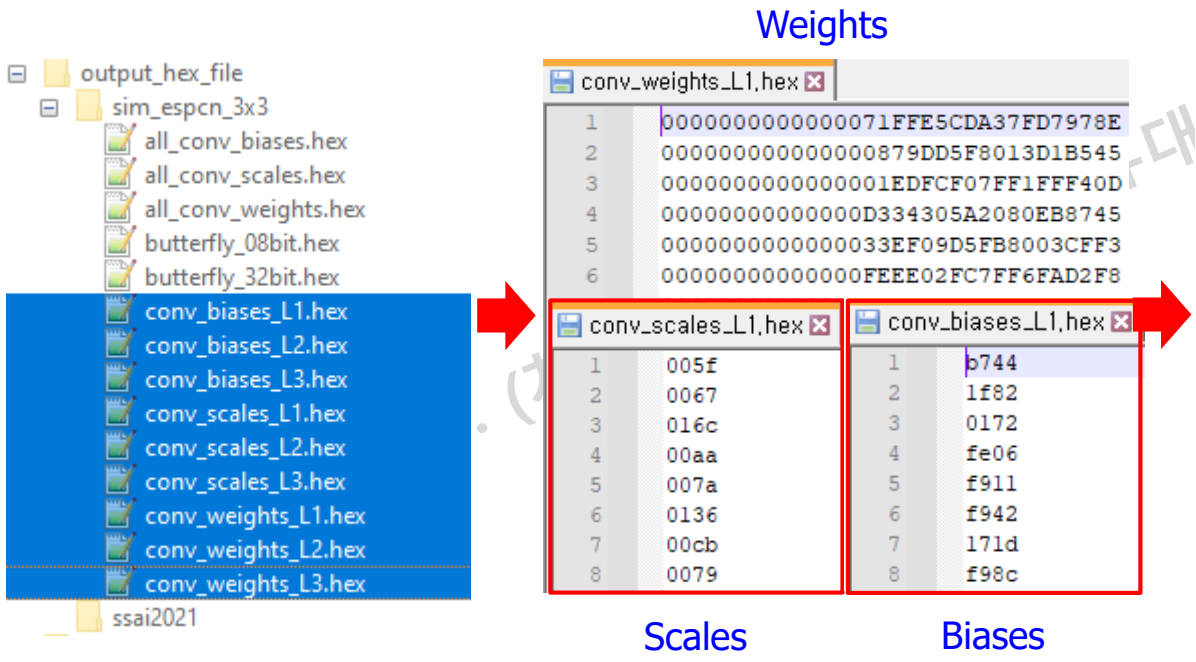
- Buffers:
  - Input buffer: in_img
  - Weight/bias/scale buffers
  - Dual (output) buffers of feature maps





Weight/Bias/Scale buffers

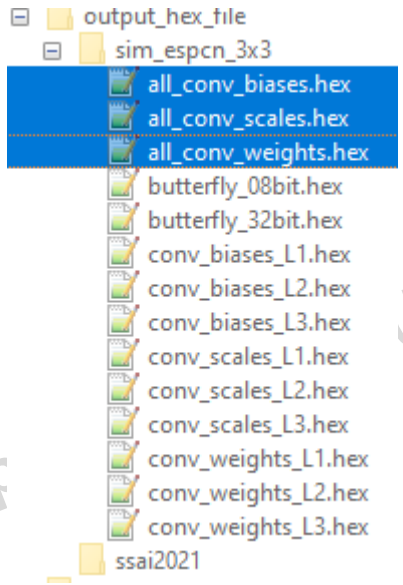
- Reference software
  - Generate weight/bias/scale hex files
  - E.g., write_cnn_model_to_hex_file.hex
- Hardware
  - Weight/bias/scale buffers are initialized by them.




```
// Weight buffer
spram #(.INIT_FILE("input_data/all_conv_weights.hex"),
        .EN_LOAD_INIT_FILE(EN_LOAD_INIT_FILE),
        .W_DATA(Ti*WI),.W_WORD(W_CELL),.N_WORD(N_CELL))
u_buf_weight(
    .clk (clk             ),
    .en  (weight_buf_en   ),
    .addr(weight_buf_addr),
    .din (/*unused*/      ),
    .we  (weight_buf_we   ),
    .dout(weight_buf_dout)
);
// Bias buffer
spram #(.INIT_FILE("input_data/all_conv_biases.hex"),
        .EN_LOAD_INIT_FILE(EN_LOAD_INIT_FILE),
        .W_DATA(PARAM_BITS),.W_WORD(W_CELL_PARAM),.N_WORD(N_CELL_PARAM))
u_buf_bias(
    .clk (clk                ),
    .en  (param_buf_en       ),
    .addr(param_buf_addr     ),
    .din (/*unused*/         ),
    .we  (param_buf_we       ),
    .dout(param_buf_dout_bias)
);
// Scale buffer
spram #(.INIT_FILE("input_data/all_conv_scales.hex"),
        .EN_LOAD_INIT_FILE(EN_LOAD_INIT_FILE),
        .W_DATA(PARAM_BITS),.W_WORD(W_CELL_PARAM),.N_WORD(N_CELL_PARAM))
u_buf_scale(
    .clk (clk                ),
    .en  (param_buf_en       ),
    .addr(param_buf_addr     ),
    .din (/*unused*/         ),
    .we  (param_buf_we       ),
    .dout(param_buf_dout_scale)
);
```

Weight/Bias/Scale buffers

- Generate weight/bias/scale hex files (write_cnn_model_to_hex_file.hex)
  - Merge weights of all layers in a file (all_conv_weights.hex)
  - Merge biases of all layers in a file (all_conv_biases.hex)
  - Merge scales of all layers in a file (all_conv_scales.hex)
- The worst case for Ti=16, To=16, conv3x3
  - 3×3×Ti×To = 2034 eight-bit weights = 144 words of 128 bits.



Layer	Filter size	Input channels	Output channels	NP	Address range(1)		
					Weights	Scales	Biases
1	3×3	1	16	3×3×1	0~15	0~15	0~15
2	3×3	16	16	3×3×16	16~159	16~31	16~31
3	3×3	16	4	3×3×16	160~303	32~47	32~47

(1) In Verilog, the starting address is zero (0).

Weight/Bias/Scale buffers

- Three **single-port** SRAMs for buffers
  - Weights
  - Scales
  - Biases.
- Here, we consider the worst case
  - Ti=16, To=16, conv3x3
    - 432 words (= 3×(3×3×To))
  - Bias/scale buffers
    - Allocate 432 words

```
parameter N_DELAY       = 1;
parameter N_LAYER       = 3;
parameter N_CELL        = N_LAYER * (Ti*To*9)/N;
parameter N_CELL_PARAM  = N_LAYER * (To);
parameter W_CELL        = $clog2(N_CELL);
parameter W_CELL_PARAM  = $clog2(N_CELL_PARAM);
parameter EN_LOAD_INIT_FILE = 1'b1; // Initialize
```

```
// Weight buffer
spram #(.INIT_FILE("input_data/all_conv_weights.hex"),
        .EN_LOAD_INIT_FILE(EN_LOAD_INIT_FILE),
        .W_DATA(Ti*WI),.W_WORD(W_CELL),.N_WORD(N_CELL))
u_buf_weight(
    .clk (clk            ),
    .en  (weight_buf_en  ),
    .addr(weight_buf_addr),
    .din (/*unused*/     ),
    .we  (weight_buf_we  ),
    .dout(weight_buf_dout)
);
// Bias buffer
spram #(.INIT_FILE("input_data/all_conv_biases.hex"),
        .EN_LOAD_INIT_FILE(EN_LOAD_INIT_FILE),
        .W_DATA(PARAM_BITS),.W_WORD(W_CELL_PARAM),.N_WORD(N_CELL_PARAM))
u_buf_bias(
    .clk (clk                ),
    .en  (param_buf_en       ),
    .addr(param_buf_addr     ),
    .din (/*unused*/         ),
    .we  (param_buf_we       ),
    .dout(param_buf_dout_bias)
);
// Scale buffer
spram #(.INIT_FILE("input_data/all_conv_scales.hex"),
        .EN_LOAD_INIT_FILE(EN_LOAD_INIT_FILE),
        .W_DATA(PARAM_BITS),.W_WORD(W_CELL_PARAM),.N_WORD(N_CELL_PARAM))
u_buf_scale(
    .clk (clk                ),
    .en  (param_buf_en       ),
    .addr(param_buf_addr     ),
    .din (/*unused*/         ),
    .we  (param_buf_we       ),
    .dout(param_buf_dout_scale)
);
```

Test vector preparation

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

test_vector

5x7 cell

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

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

Execution Flow of a layer

- Write the configuration registers for a layer, i.e. Layer 3
  - is_last_layer = 1
  - is_conv3x3 = 1
  - bias_shift = 17, act_shift = 7
  - base_addr_weight = 160, base_addr_scale = base_addr_bias= 32.
- Start the CNN accelerator (q_start = 1)
- Polling: Wait until the CNN layer computation is DONE.

```
q_layer_index       = idx;
q_is_last_layer     = (idx == N_LAYER-1)?1'b1:1'b0;
q_is_first_layer    = (idx == 0) ? 1'b1: 1'b0;
is_conv3x3          = q_is_conv3x3[idx];
q_layer_config      = {q_act_shift[idx], q_bias_shift[idx], q_layer_index, q_is_last_layer, is_conv3x3, q_is_last_layer, q_is_first_layer};
#(4*p) @(posedge HCLK)  u_riscv_dummy.task_AHBwrite(`CNN_ACCEL_BASE_ADDRESS, {base_addr_param&12'hFFF,base_addr_weight&20'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 );

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

Feature map buffers

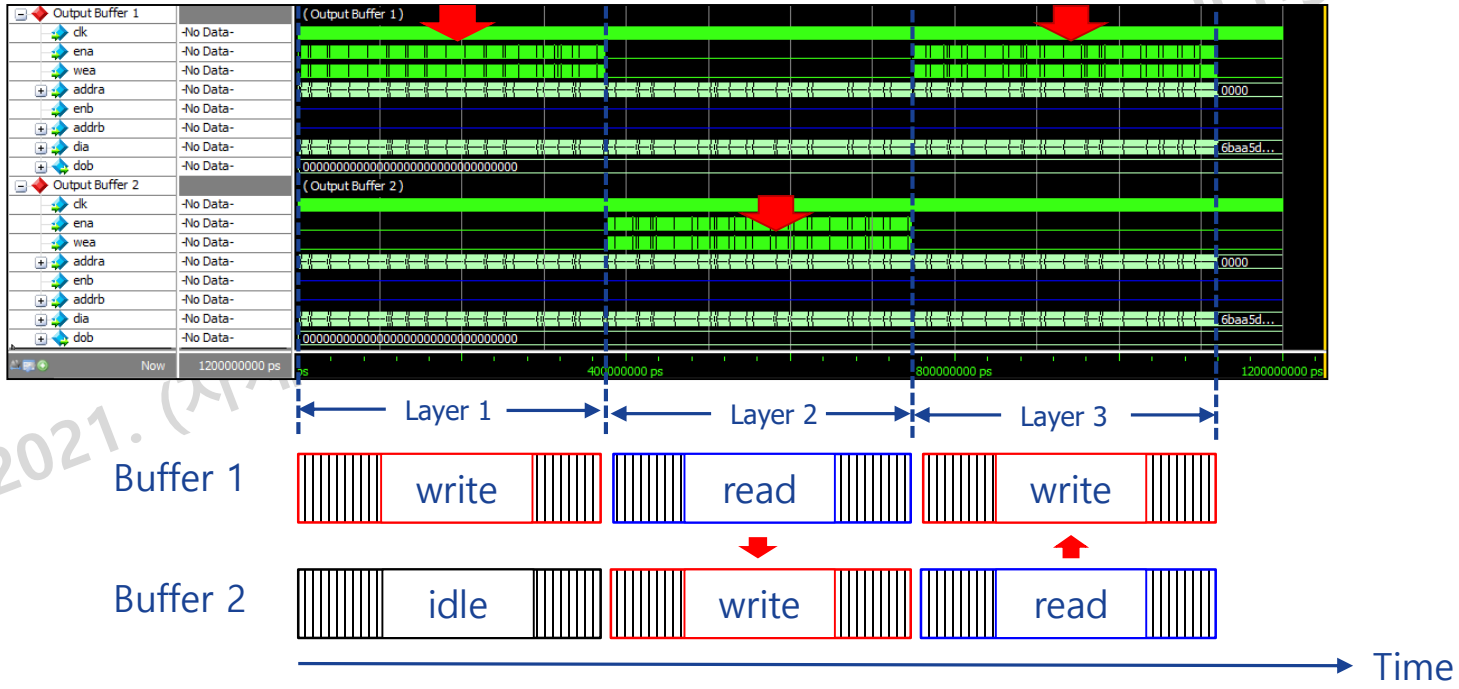
- Dual buffers are used to store feature maps
  - Two buffers are identical.
  - Assume Port B's signals are OPEN.
  - A one-bit flag (out_buff_sel) is used to select a specific buffer.

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

```
//------------------------------------------------
// Output buffers.
//------------------------------------------------
wire [ACT_BITS*To-1:0] all_acc_o = {
    acc_o[15], acc_o[14], acc_o[13], acc_o[12],
    acc_o[11], acc_o[10], acc_o[ 9], acc_o[ 8],
    acc_o[ 7], acc_o[ 6], acc_o[ 5], acc_o[ 4],
    acc_o[ 3], acc_o[ 2], acc_o[ 1], acc_o[ 0]
};
dpram #(.W_DATA(To*ACT_BITS), .W_WORD(FRAME_SIZE_W),.N_WORD(FRAME_SIZE))
u_fmap_buff_01(
    .clk    (clk   ),
    .ena    ((!out_buff_sel) & vld_o[0]),
    .wea    ((!out_buff_sel) & vld_o[0]),
    .addra  (pixel_count ),
    .enb    (/*OPEN*/ ),
    .addrb  (/*OPEN*/ ),
    .dia    (all_acc_o),
    .dob    (/*OPEN*/ )
);

dpram #(.W_DATA(To*ACT_BITS), .W_WORD(FRAME_SIZE_W),.N_WORD(FRAME_SIZE))
u_fmap_buff_02(
    .clk    (clk   ),
    .ena    (out_buff_sel & vld_o[0]),
    .wea    (out_buff_sel & vld_o[0]),
    .addra  (pixel_count ),
    .enb    (/*OPEN*/ ),
    .addrb  (/*OPEN*/ ),
    .dia    (all_acc_o),
    .dob    (/*OPEN*/ )
);
```

Dual buffers and the timing flow

- Layer 1: The output fmaps are stored to Buffer 1
- Layer 2: The output fmaps are stored to Buffer 2
  - The input fmaps are read from Buffer 1
- Layer 3: The output fmaps are written to Buffer 1
  - The input fmaps are read from Buffer 2



Road map

Review

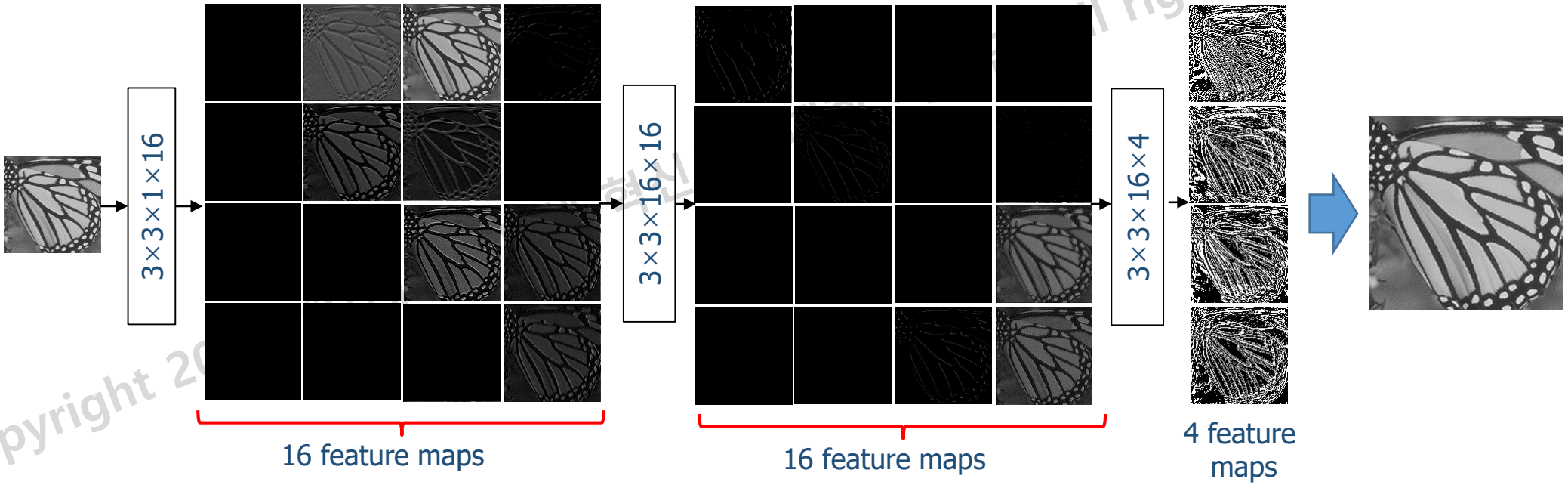
Control path, data path

System integration

Direct memory access (DMA)

Sim-ESPCN

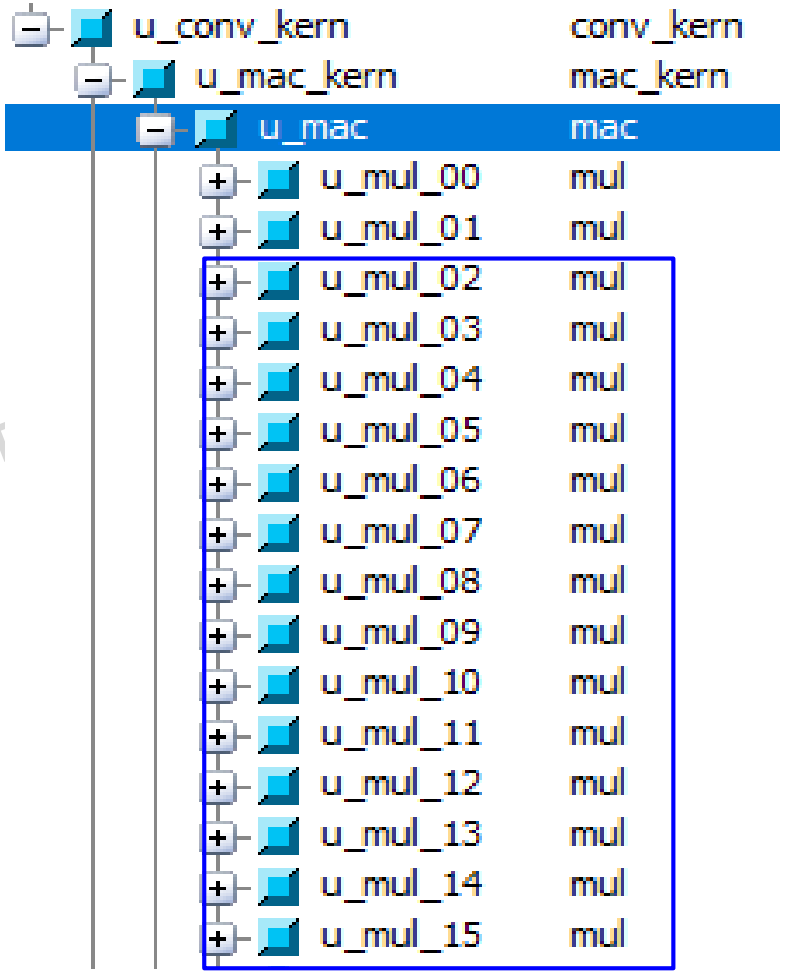
- Three CONV layers:
  - Layer 1: one input image → 16 output feature maps
  - Layer 2: 16 input feature maps → 16 output feature maps.
  - Layer 3: 16 input feature maps → 4 output feature maps.



Mapping

- Number of multipliers per an output pixel (num_ops)
  - Layer 1: 9 (=3×3×1) < 16
  - Layer 2 and 3: 144 (= 3×3×16).

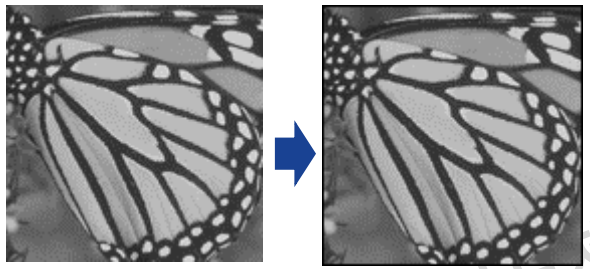
- Convolution types
  - CON1×1 (is_conv3×3 == 0)
    - Layer 1: 3×3×1
    - conv1×1 (the final project): 1×1×16
    - An output pixel is given every one cycle
  - CONV3×3 (is_conv3×3 == 1)
    - Layer 2, 3: input vector = 1×1×16
    - An output pixel is given every nine cycles



conv_kern: Ti=16

CONV1x1

- Layer 1: num_ops=9 (=3×3×1) < 16
  - Layer 1: 3×3×1
  - conv1×1 (the final project): 1×1×16
  - An output pixel is given every one cycle
- Cycle 1

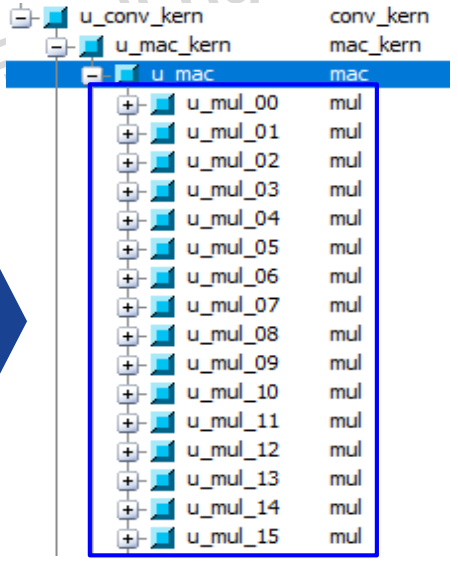


0	0	0	0	0	0	0	0
0	42	69	91	99	106	108	111
0	105	42	56	84	106	113	112
0	72	43	42	68	109	112	104

out(0,0,:)

address	Cycle1	Cycle 2
[0*WI+:WI]	0	0
[1*WI+:WI]	0	0
[2*WI+:WI]	0	0
[3*WI+:WI]	0	42
[4*WI+:WI]	42	69
[5*WI+:WI]	69	91
[6*WI+:WI]	0	105
[7*WI+:WI]	105	42
[8*WI+:WI]	42	56
....	0	0

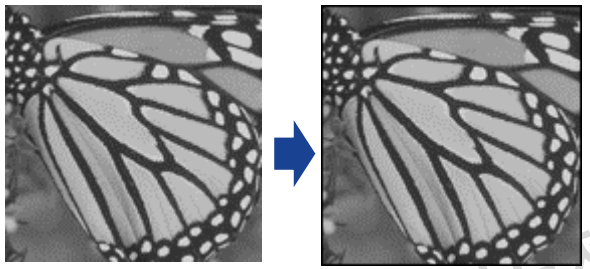
din



conv_kern: Ti=16

CONV1x1

- Layer 1: num_ops=9 (=3×3×1) < 16
  - Layer 1: 3×3×1
  - conv1×1 (the final project): 1×1×16
  - An output pixel is given every one cycle
- Cycle 2

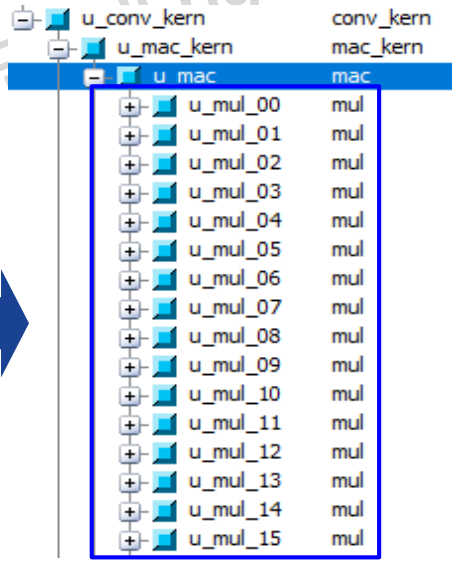


0	0	0	0	0	0	0	0
0	42	69	91	99	106	108	111
0	105	42	56	84	106	113	112
0	72	43	42	68	109	112	104

out(0, 1, :)

address	Cycle1	Cycle 2
[0*WI+:WI]	0	0
[1*WI+:WI]	0	0
[2*WI+:WI]	0	0
[3*WI+:WI]	0	42
[4*WI+:WI]	42	69
[5*WI+:WI]	69	91
[6*WI+:WI]	0	105
[7*WI+:WI]	105	42
[8*WI+:WI]	42	56
....	0	0

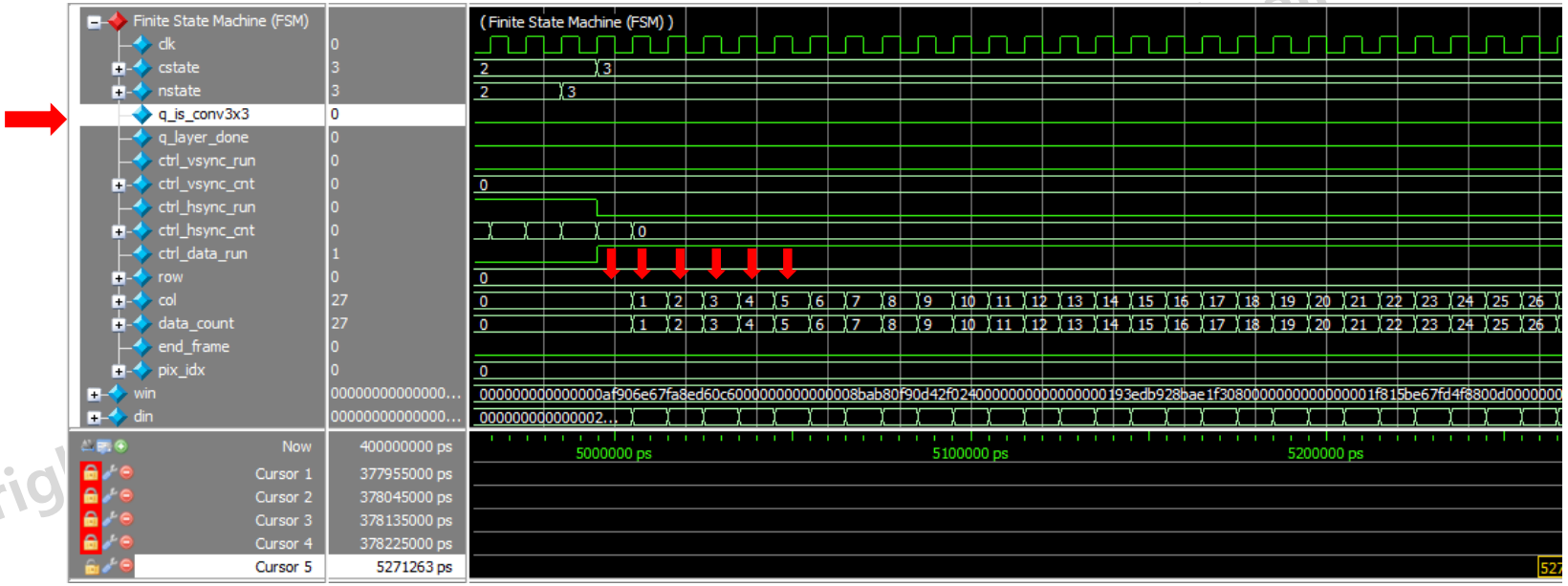
din



conv_kern: Ti=16

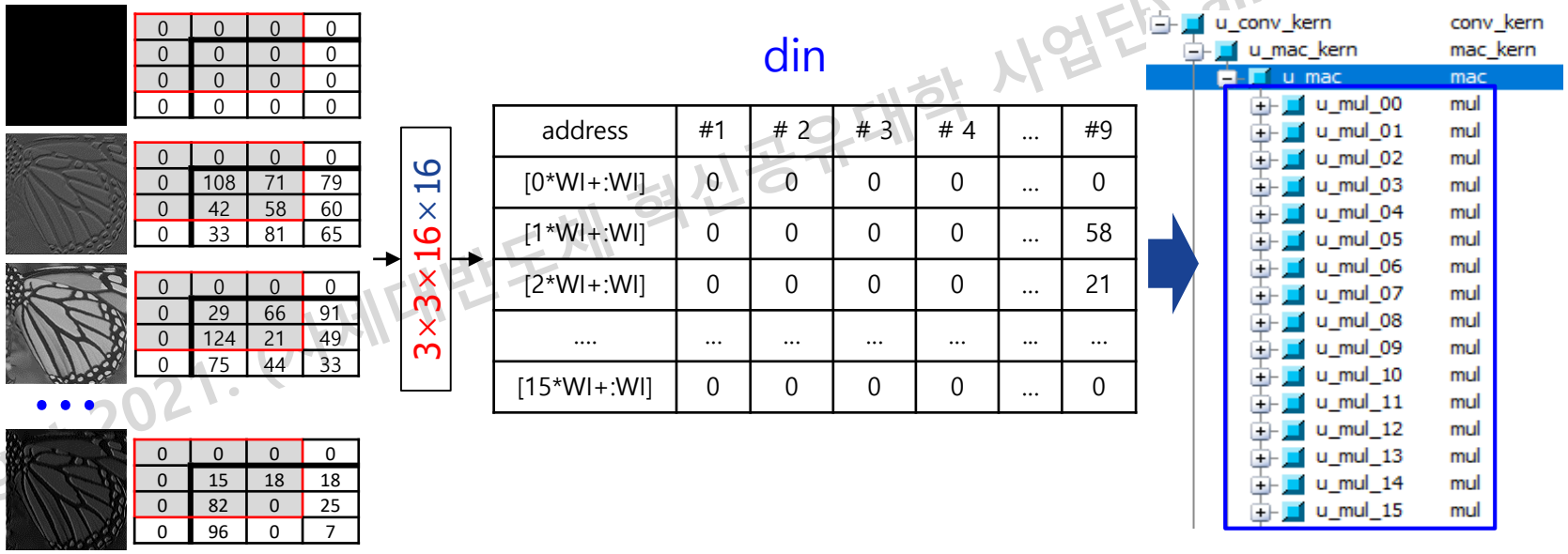
CONV1x1

- The baseline version supports CONV1x1 (q_is_conv3x3==0)
  - If ctrl_data_run == 1
    - row, col and data_count are updated.



CONV3x3

- Layer 2: num_ops=144 (=3×3×16)
  - CONV3×3 (is_conv3×3 == 1)
  - Input vector (din) = 1×1×16
  - An output pixel is given every nine cycles

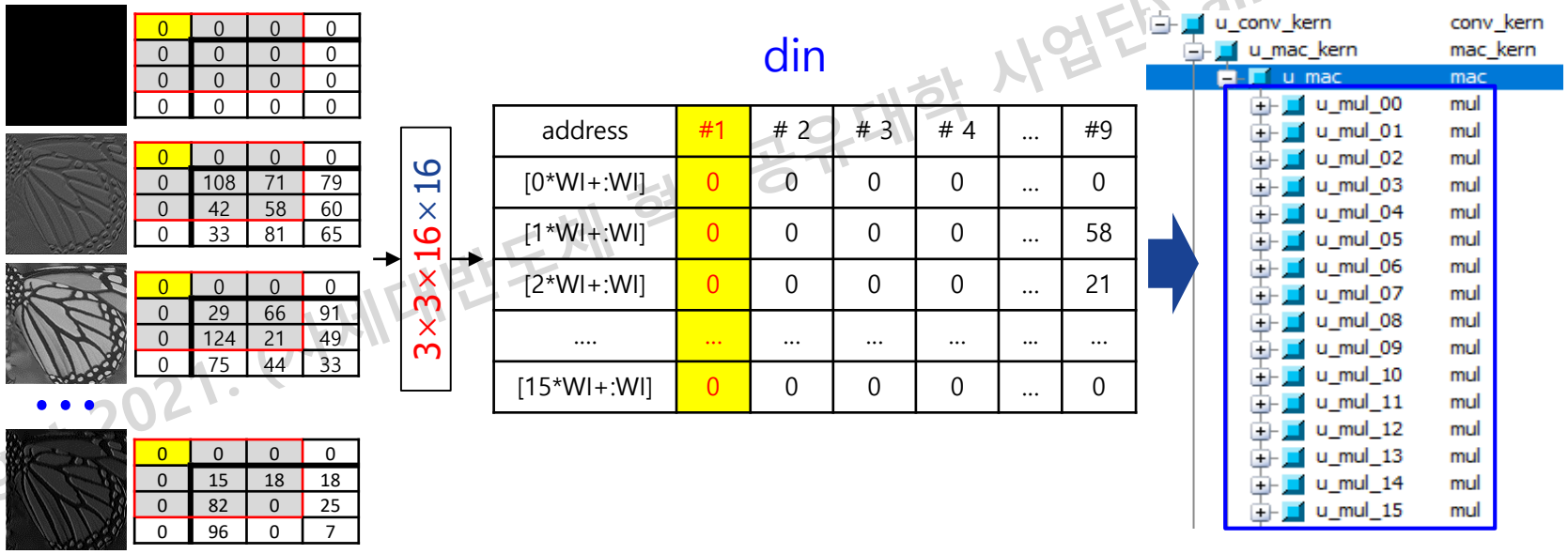


16 feature maps

conv_kern: Ti=16

CONV3x3

- Layer 2: num_ops=144 (=3×3×16)
  - CONV3×3 (is_conv3×3 == 1)
  - Input vector (din) = 1×1×16
  - An output pixel is given every nine cycles
- Cycle #1

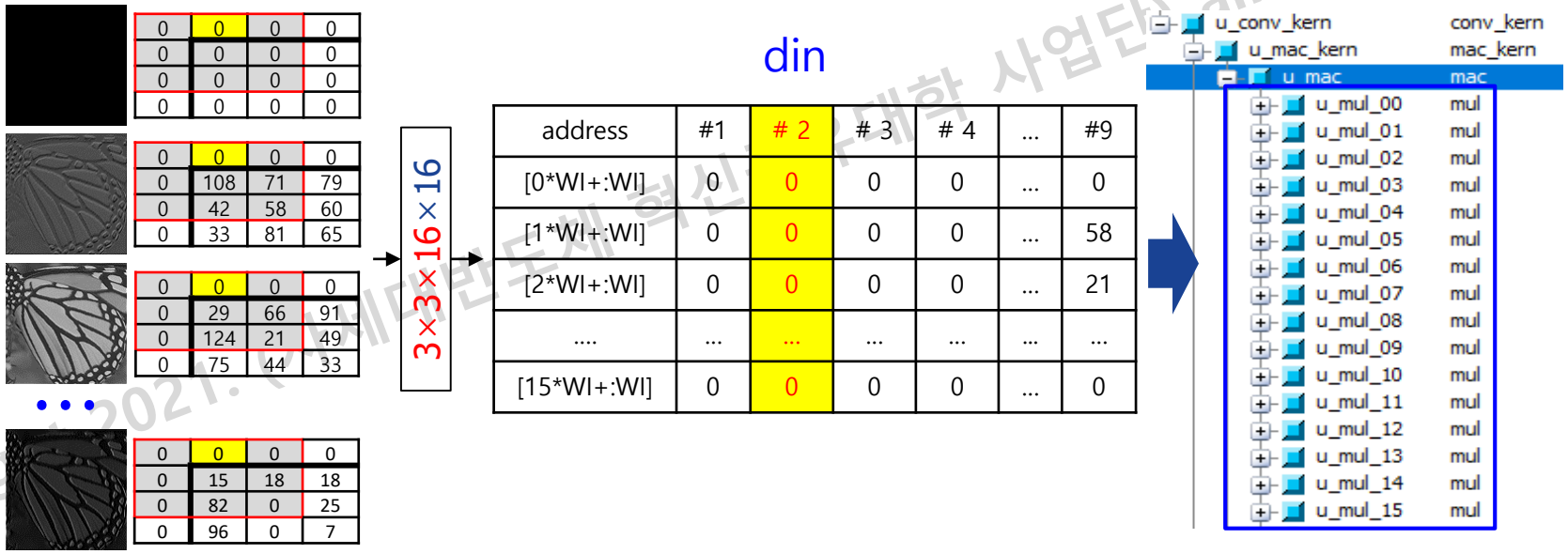


16 feature maps

conv_kern: Ti=16

CONV3x3

- Layer 2: num_ops=144 (=3×3×16)
  - CONV3×3 (is_conv3×3 == 1)
  - Input vector (din) = 1×1×16
  - An output pixel is given every nine cycles
- Cycle #2

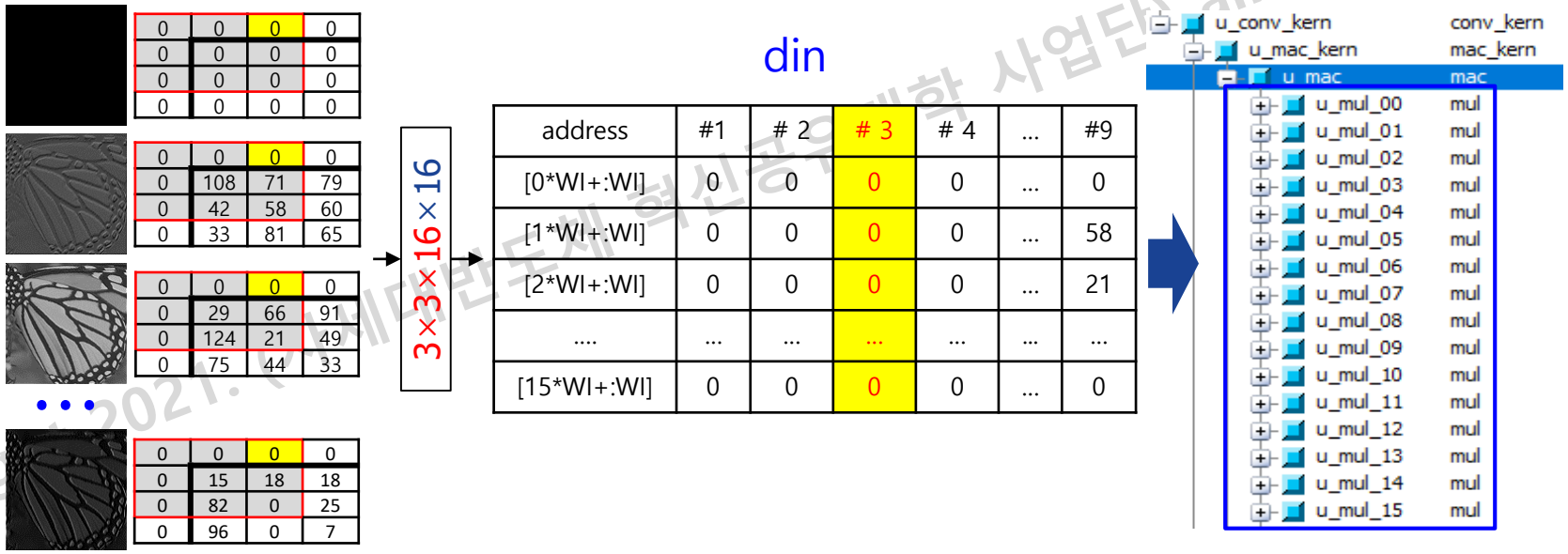


16 feature maps

conv_kern: Ti=16

CONV3x3

- Layer 2: num_ops=144 (=3×3×16)
  - CONV3×3 (is_conv3×3 == 1)
  - Input vector (din) = 1×1×16
  - An output pixel is given every nine cycles
- Cycle #3

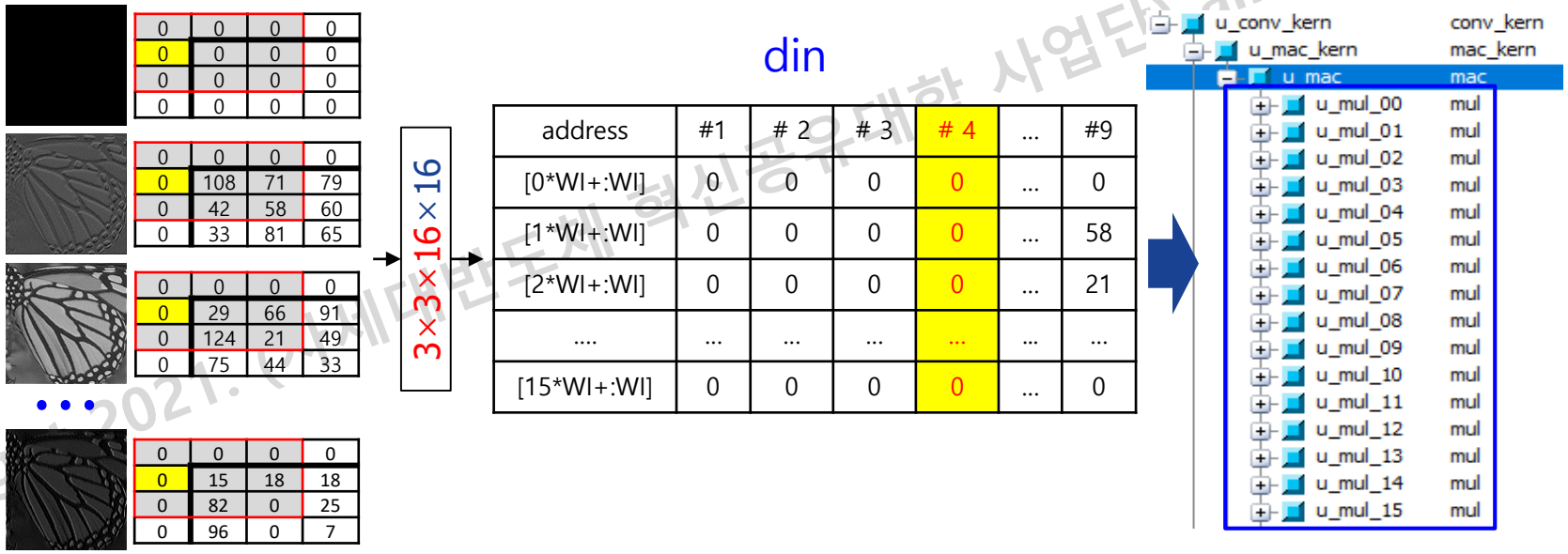


16 feature maps

conv_kern: Ti=16

CONV3x3

- Layer 2: num_ops=144 (=3×3×16)
  - CONV3×3 (is_conv3×3 == 1)
  - Input vector (din) = 1×1×16
  - An output pixel is given every nine cycles
- Cycle #4

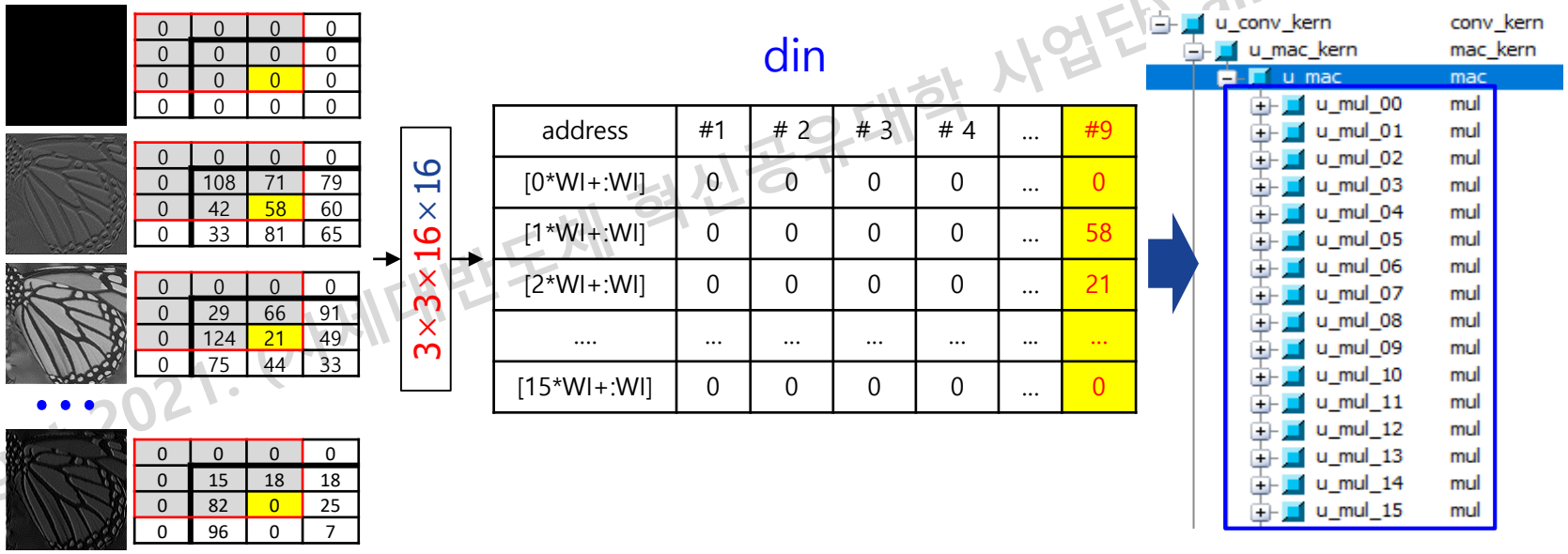


16 feature maps

conv_kern: Ti=16

CONV3x3

- Layer 2: num_ops=144 (=3×3×16)
  - CONV3×3 (is_conv3×3 == 1)
  - Input vector (din) = 1×1×16
  - An output pixel is given every nine cycles
- Cycle #9

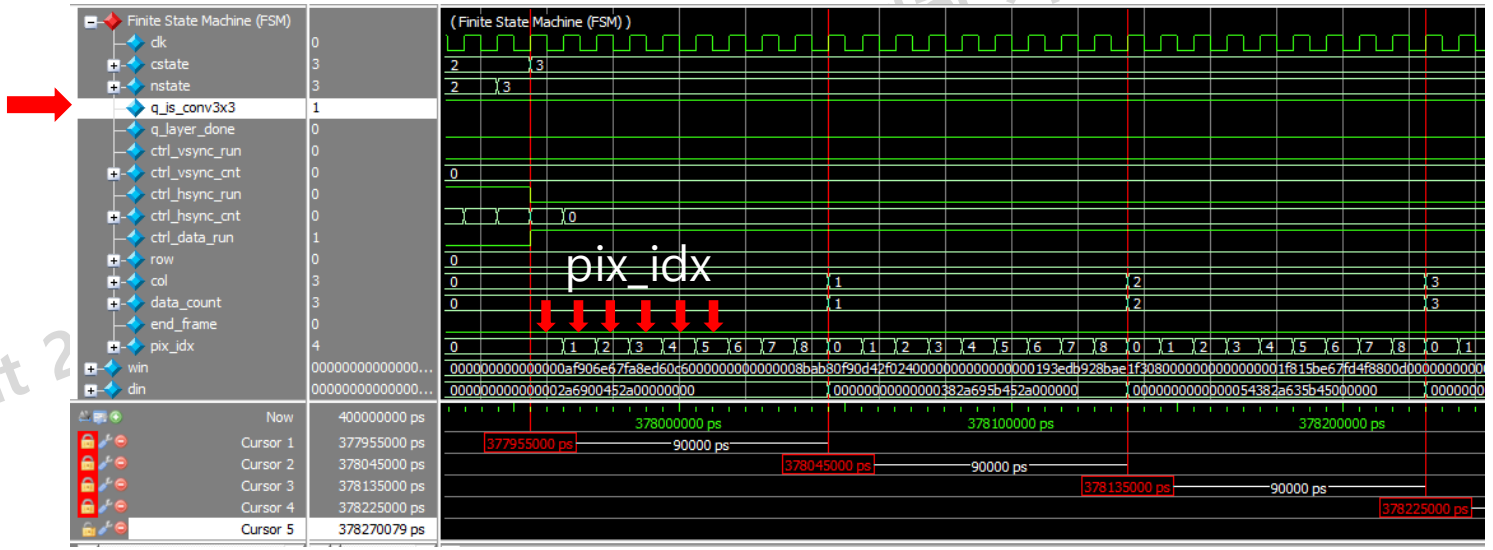


16 feature maps

conv_kern: Ti=16

CONV 3x3

- Use a counter for updating a pixel index (pix_idx)
  - Count up to 8 and reset to 0.
- CONV3x3 (q_is_conv3x3==1)
  - If ctrl_data_run == 1
    - Need to check the pixel counter to update row, col, data_count.
    - The update occurs every 9 cycles (pix_idx == 8)



Control path

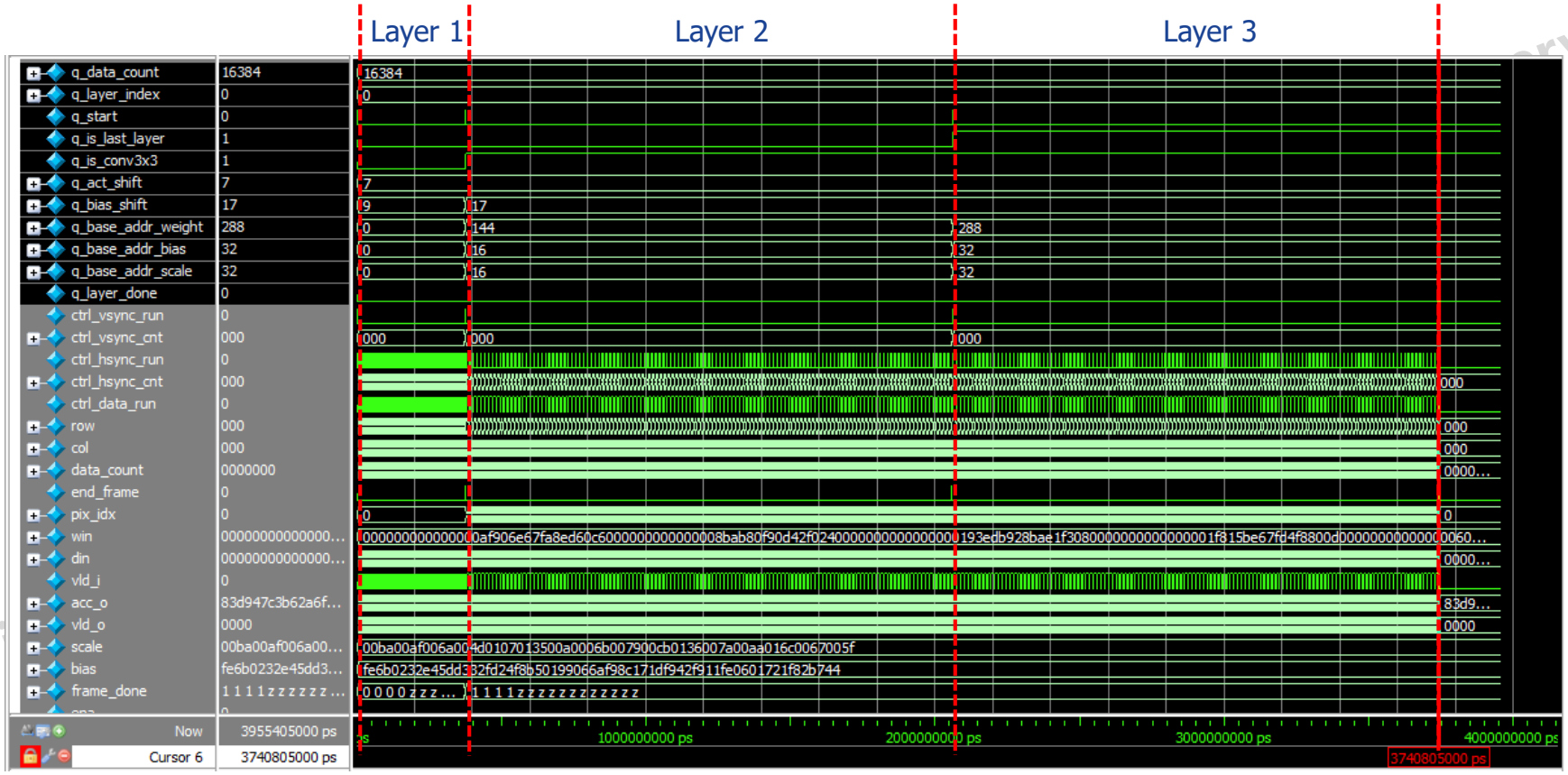
- Row, column, data count and FSM's ST_DATA
  - CONV1×1: !q_is_conv3x3
  - CONV3×3: q_is_conv3x3 && (pix_idx == 8)
- Add the condition: (!q_is_conv3x3) || (q_is_conv3x3 && (pix_idx == 8))
  - (!q_is_conv3x3) || (pix_idx == 8)

```
always@(posedge HCLK, negedge HRESETn)
begin
    if(~HRESETn) begin
        row <= 0;
        col <= 0;
    end
    else begin
        if(ctrl_data_run /*Insert your code*/) begin
            if(col == q_width - 1) begin
                if(end_frame)
                    row <= 0;
                else
                    row <= row + 1;
            end
            if(col == q_width - 1)
                col <= 0;
            else
                col <= col + 1;
        end
    end
end
always@(posedge HCLK, negedge HRESETn)
begin
    if(~HRESETn) begin
        data_count <= 0;
    end
    else begin
        if(ctrl_data_run /*Insert your code*/) begin
            if(!end_frame)
                data_count <= data_count + 1;
            else
                data_count <= 0;
        end
    end
end
```

```
always @(*) begin
    case(cstate)
    ST_IDLE: begin
                if(q_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 == q_hsync_delay)
                    nstate = ST_DATA;
                else
                    nstate = ST_HSYNC;
        end
        ST_DATA: begin
                if(end_frame /*Insert your code*/)    //end of frame
                    nstate = ST_IDLE;
                else begin
                    if((col == q_width-1) /*Insert your code*/)    //end of line
                        nstate = ST_HSYNC;
                    else
                        nstate = ST_DATA;
                    end
        end
        default: nstate = ST_IDLE;
    endcase
end
```

Waveform

- Do simulation with time = 4 ms
  - Sim-ESPCN completes all computations in 3,750 us.



Road map

Review

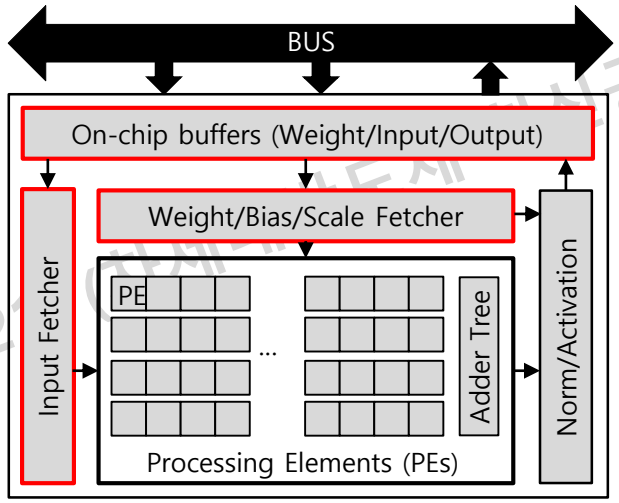
Control path, data path

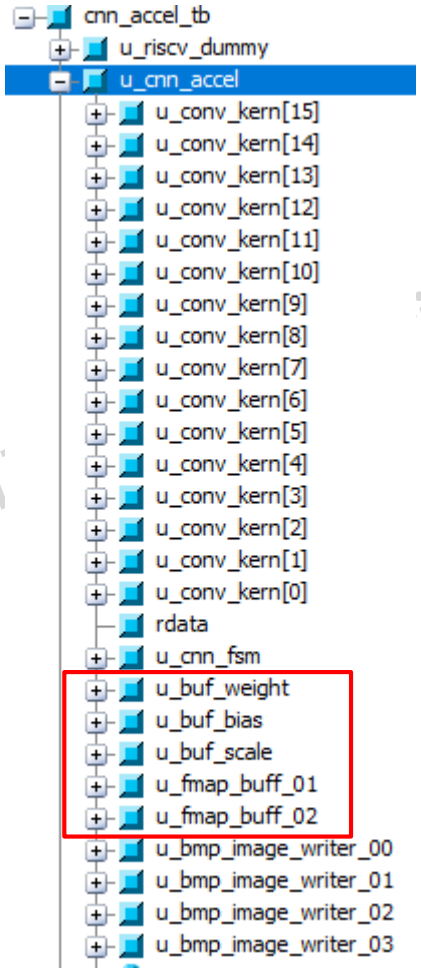
System integration (Dataflow)

Direct memory access (DMA)

Buffers: ~Register File

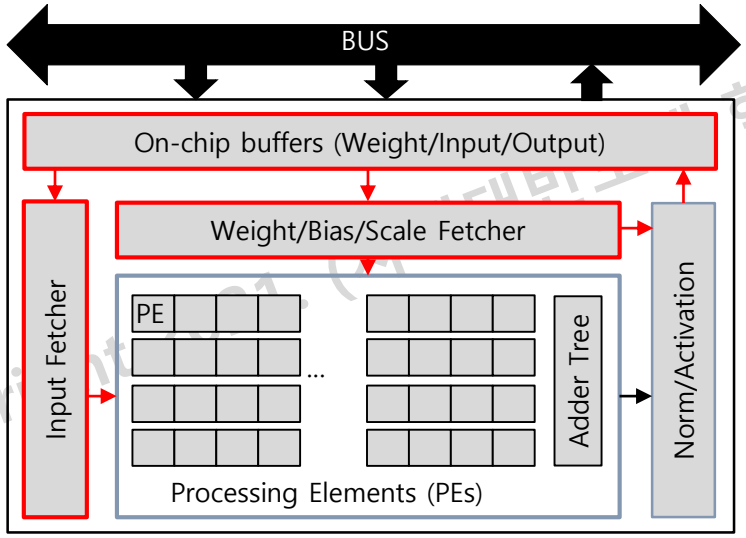
- Buffers:
  - Input buffer: in_img
  - Weight/bias/scale buffers
  - Dual (output) buffers of feature maps
- How to access buffers?





Motivation: Access buffers

- To fetch the inputs to the computing units
  - Input buffer (in_img), dual (output) buffers of feature maps
  - Weight/bias/scale buffers
- To read the outputs from the computing units
  - Dual (output) buffers of feature maps
- The convolution kernels are as functions: Need to provide the inputs and get the return value




```
generate
    genvar i;
    for (i=0; i<To; i=i+1) begin: u_conv_kern
        conv_kern u_conv_kern(
        ./*input                    */clk(clk),
        ./*input                    */rstn(rstn),
        ./*input                    */is_last_layer(q_is_last_layer),
        ./*input [PARAM_BITS-1:0]*/scale(scale[i*PARAM_BITS+:PARAM_BITS]),
        ./*input [PARAM_BITS-1:0]*/bias(bias[  i*PARAM_BITS+:PARAM_BITS]),
        ./*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[i*Ti*WI+:Ti*WI]),
        ./*input [N*WI-1:0]         */din(din),
        ./*output [ACT_BITS-1:0] */acc_o(acc_o[i*ACT_BITS+:ACT_BITS]),
        ./*output                   */vld_o(vld_o[i])
        );
    end
endgenerate
```

Convolution kernels: ~ALU

- Complete the table for a "conv_kern" function call
  - Input/output ports of convolution kernels

Ports	In/Out	Definition	Source/Destination		
			Layer #1	Layer #2	Layer #3
is_last_layer	In	0: ReLU, 1: Linear			
is_conv3x3	In	0: conv1x1, 1: conv3x3			
bias_shift	In	Shift amount before adding a bias			
act_shift	In	Shift amount for activation quantization			
bias	In	Biases			
scale	In	Scales			
win	In	Weights			
vld_i	In	Input valid signal			
din	In	Input pixels			
vld_o	Out	Output valid signal			
acc_o	Out	Output pixels			

Convolution kernels: ~ALU

- Control signals and parameters
  - From the configuration registers
  - An AHB Master (e.g., RISC-V) sets those registers.

Ports	In/Out	Definition	Source/Destination		
			Layer #1	Layer #2	Layer #3
is_last_layer	In	0: ReLU, 1: Linear	**0**	**0**	**1**
is_conv3x3	In	0: conv1x1, 1: conv3x3	**0**	**1**	**1**
bias_shift	In	Shift amount before adding a bias	**9**	**17**	**17**
act_shift	In	Shift amount for activation quantization	**7**	**7**	**7**
bias	In	Biases			
scale	In	Scales			
win	In	Weights			
vld_i	In	Input valid signal			
din	In	Input pixels			
vld_o	Out	Output valid signal			
acc_o	Out	Output pixels			

Convolution kernels: ~ALU

- Weight/bias/scale
  - From weight/bias/scale buffers.
  - Note: their base addresses should be defined.

Ports	In/Out	Definition	Source/Destination		
			Layer #1	Layer #2	Layer #3
is_last_layer	In	0: ReLU, 1: Linear	0	0	1
is_conv3x3	In	0: conv1×1, 1: conv3×3	0	1	1
bias_shift	In	Shift amount before adding a bias	9	17	17
act_shift	In	Shift amount for activation quantization	7	7	7
bias	In	Biases ← bias_buffer	0	16	32
scale	In	Scales ← scale_buffer	0	16	32
win	In	Weights ← weight_buffer	0	16	160
vld_i	In	Input valid signal			
din	In	Input pixels			
vld_o	Out	Output valid signal			
acc_o	Out	Output pixels			

Convolution kernels: ~ALU

- Input valid signal
  - Generated from Finite State Machine (FSM), i.e., ctrl_data_run.
- The output valid signal is generated from vld_i and is_conv3x3.

Ports	In/Out	Definition	Source/Destination		
			Layer #1	Layer #2	Layer #3
is_last_layer	In	0: ReLU, 1: Linear	0	0	1
is_conv3x3	In	0: conv1x1, 1: conv3x3	0	1	1
bias_shift	In	Shift amount before adding a bias	9	17	17
act_shift	In	Shift amount for activation quantization	7	7	7
bias	In	Biases ← bias_buffer	0	16	32
scale	In	Scales ← scale_buffer	0	16	32
win	In	Weights ← weight_buffer	0	16	160
vld_i	In	Input valid signal ← FSM (ctrl_data_run)	FSM	FSM	FSM
din	In	Input pixels			
vld_o	Out	Output valid signal	vld_i	vld_i	vld_i
acc_o	Out	Output pixels			

Convolution kernels: ~ALU

- Input pixels: From in_img or feature map buffers
  - Controlled by the layer index (is_first_layer/is_last_layer) and out_buff_sel
- Output pixels: To dual buffers
  - Controlled by the buffer selection flag (out_buff_sel).

Ports	In/Out	Definition	Source/Destination		
			Layer #1	Layer #2	Layer #3
is_last_layer	In	0: ReLU, 1: Linear	0	0	1
is_conv3x3	In	0: conv1x1, 1: conv3x3	0	1	1
bias_shift	In	Shift amount before adding a bias	9	17	17
act_shift	In	Shift amount for activation quantization	7	7	7
bias	In	Biases ← bias_buffer	0	16	32
scale	In	Scales ← scale_buffer	0	16	32
win	In	Weights ← weight_buffer	0	16	160
vld_i	In	Input valid signal ← FSM (ctrl_data_run)	FSM	FSM	FSM
din	In	Input pixels	in_img	Buffer 1	Buffer 2
vld_o	Out	Output valid signal	vld_i	vld_i	vld_i
acc_o	Out	Output pixels	Buffer 1	Buffer 2	Buffer 1

Lab 1: CNN accelerator system integration

- Lab 1:
  - Complete the missing codes to access the buffers (cnn_accel.v)
    - Weight/bias/scale buffers
    - Feature map buffers
  - Do simulation
    - Time = 400us for debugging
    - Time = 4,000us for full simulation
  - Verification
    - Check the output hex files
    - Check the final output images

Weight/Bias/Scale buffers

- Three **single-port** SRAMs for buffers
- Read enable signals
  - weight_buf_en
  - param_buf_en
- Request addresses
  - weight_buf_addr
  - param_buf_addr
- Return/readout data
  - weight_buf_dout_weight
  - param_buf_dout_bias
  - param_buf_dout_scale

```
// Weight buffer
spram #(.INIT_FILE("input_data/all_conv_weights.hex"),
        .EN_LOAD_INIT_FILE(EN_LOAD_INIT_FILE),
        .W_DATA(Ti*WI),.W_WORD(W_CELL),.N_WORD(N_CELL))
u_buf_weight(
    .clk (clk             ),
    .en  (weight_buf_en   ),
    .addr(weight_buf_addr),
    .din (/*unused*/      ),
    .we  (weight_buf_we   ),
    .dout(weight_buf_dout)
);
// Bias buffer
spram #(.INIT_FILE("input_data/all_conv_biases.hex"),
        .EN_LOAD_INIT_FILE(EN_LOAD_INIT_FILE),
        .W_DATA(PARAM_BITS),.W_WORD(W_CELL_PARAM),.N_WORD(N_CELL_PARAM))
u_buf_bias(
    .clk (clk                 ),
    .en  (param_buf_en        ),
    .addr(param_buf_addr      ),
    .din (/*unused*/          ),
    .we  (param_buf_we        ),
    .dout(param_buf_dout_bias)
);
// Scale buffer
spram #(.INIT_FILE("input_data/all_conv_scales.hex"),
        .EN_LOAD_INIT_FILE(EN_LOAD_INIT_FILE),
        .W_DATA(PARAM_BITS),.W_WORD(W_CELL_PARAM),.N_WORD(N_CELL_PARAM))
u_buf_scale(
    .clk (clk                 ),
    .en  (param_buf_en        ),
    .addr(param_buf_addr      ),
    .din (/*unused*/          ),
    .we  (param_buf_we        ),
    .dout(param_buf_dout_scale)
);
```

To do …

- Complete the missing codes to access weight/scale/bias buffers
  - Weight: weight_buf_en, weight_buf_we, and weight_buf_addr for CONV3x3
  - Scale/Bias: param_buf_en, param_buf_we, param_buf_addr
- Note: The base addresses are used here.

```
// Weight
always@(*) begin
    weight_buf_en   = 1'b0;
    weight_buf_we   = 1'b0;
    weight_buf_addr = {W_CELL{1'b0}};
    if(ctrl_vsync_run) begin
        if(!q_is_conv3x3) begin // Conv1x1
            if(ctrl_vsync_cnt < To) begin
                weight_buf_en   = 1'b1;
                weight_buf_we   = 1'b0;
                weight_buf_addr = q_base_addr_weight + ctrl_vsync_cnt[W_CELL-1:0];
            end
        end
        else begin              // Conv3x3
            /*Insert your code*/
        end
    end
end

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

To do …

- Complete the missing codes for scale and bias inputs for CONV kernels
  - Generate bias and scale from outputs of their buffers
  - Use bit vector selection

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

// Update weight/bias/scale buffers
always@(posedge clk, negedge rstn)begin
    if(~rstn) begin
        win_buf <= {(9*To*Ti*WI){1'b0}};
        bias    <= {(To*PARAM_BITS){1'b0}};
        scale   <= {(To*PARAM_BITS){1'b0}};
    end
    else begin
        // Weight
        if(weight_buf_en_d)
            win_buf[(weight_buf_addr_d*Ti*WI)+:(Ti*WI)] <= weight_buf_dout;
        // Bias/scale
        /*Insert your code*/
    end
end
```

Test bench (cnn_accel_tb.v)

- Add some registers of all layers
  - act_shift, bias_shift, q_is_conv3x3
  - Base addresses of weight/bias/scale buffers for each layer.
  - Loop/layer index

Registers

```
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 [31:0] base_addr_weight;
reg [31:0] base_addr_bias;
reg [31:0] base_addr_scale;

integer idx;
```

Initialization (Reset)

```
// Define the shift amounts
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;

// Weight/bias/Scale base addresses
base_addr_weight    = 0;
base_addr_bias      = 0;
base_addr_scale     = 0;
```

Test bench (cnn_accel_tb.v)

- Use a loop execute all layers in a network.

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

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

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

end
```

(1) Configure registers: layer index, is_first_layer, is_last_layer, is_conv3x3, bias_shift, act_shift and addresses.
(2) Start a layer's execution
(3) Polling until a layer is done.

Test bench (cnn_accel_tb.v)

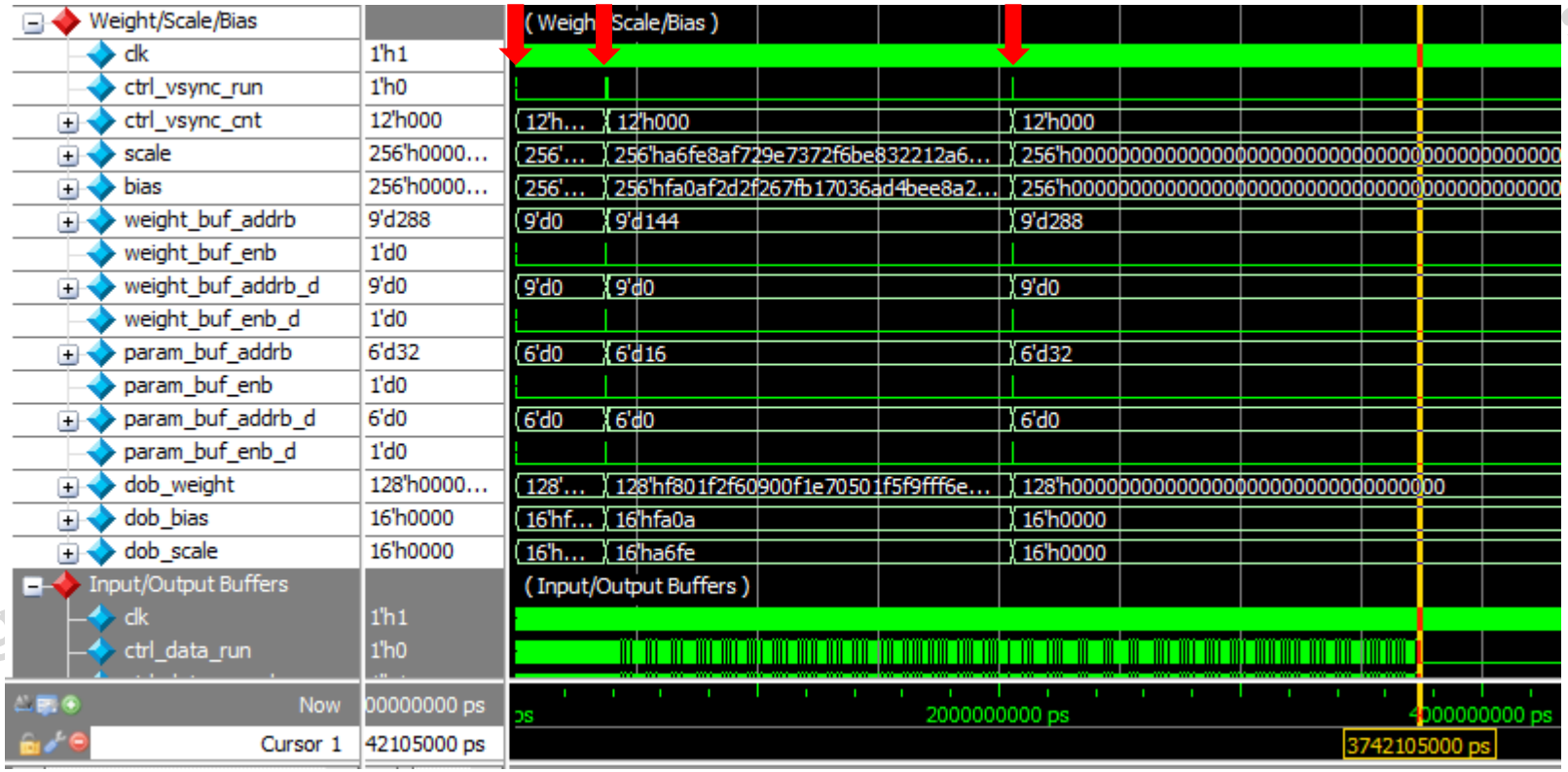
- Use a loop execute all layers in a network.

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

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

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

(4) Reset q_layer_done
(5) Update the base addresses

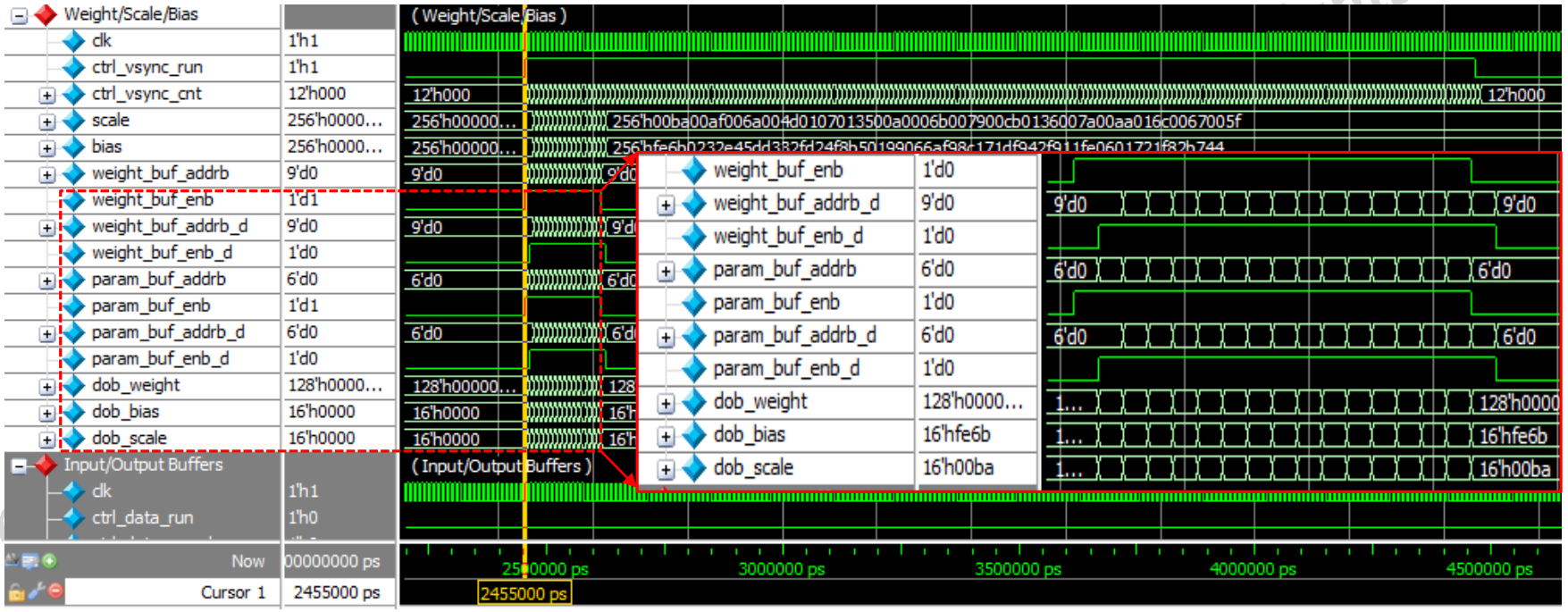
Waveform

- Do simulation with time = 4ms
  - Weight/bias/scale buffers are updated when a CNN layer is started
  - ctrl_vsync_run



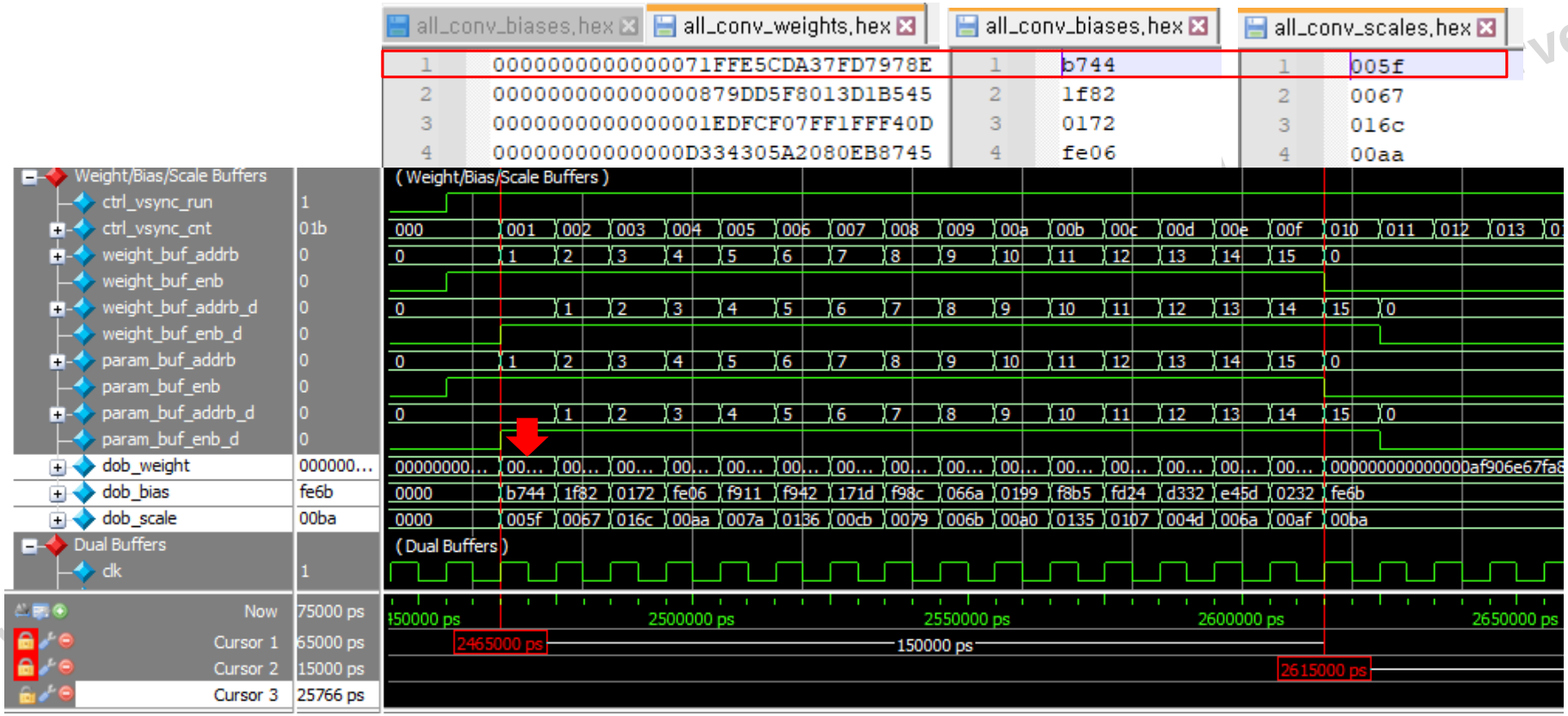
Waveform: Layer 1

- Unit test/Debugging:
  - Layer 1: do simulation with time = 4,500ns



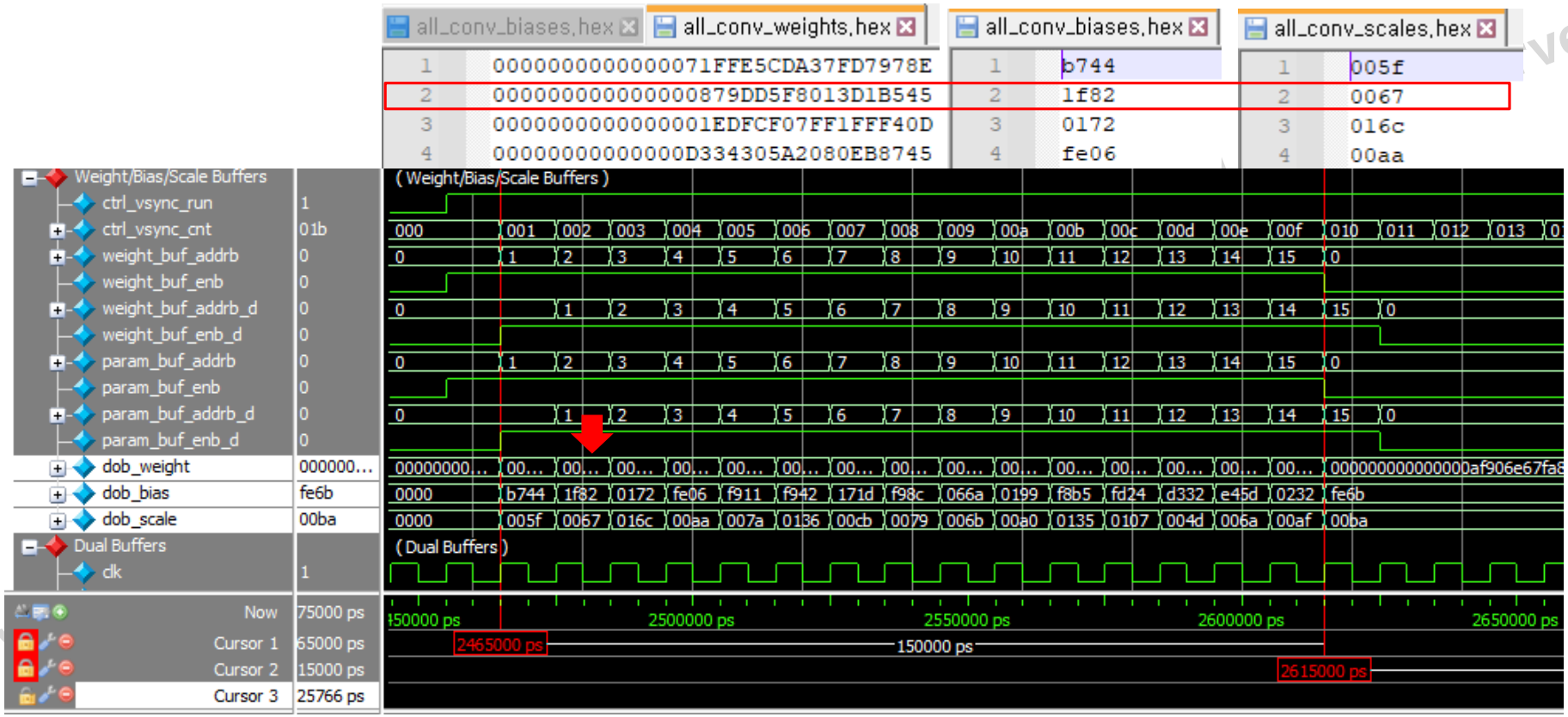
Waveform: Layer 1

- Unit test/Debugging:
  - Layer 1: do simulation with time = 4,500ns



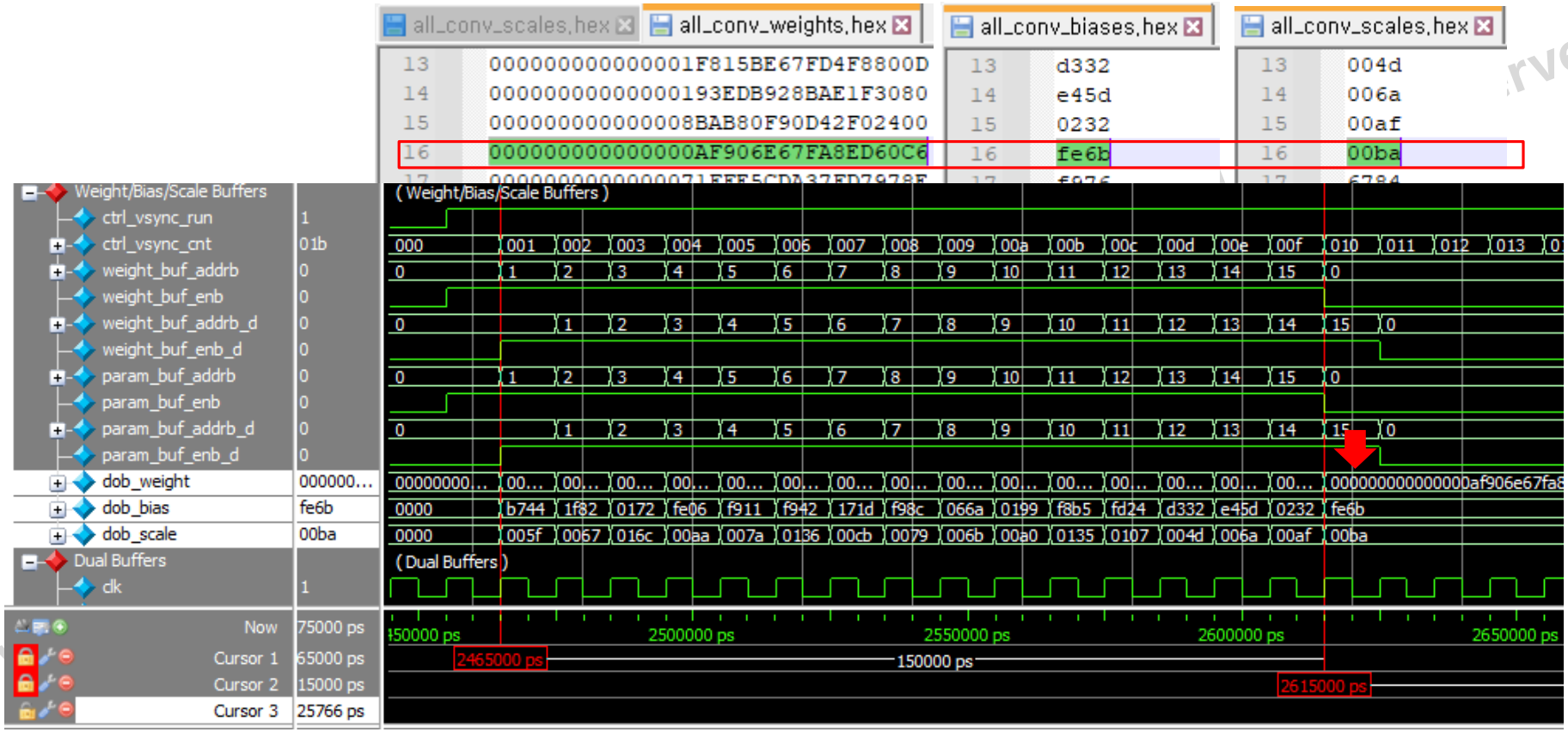
Waveform: Layer 1

- Unit test/Debugging:
  - Layer 1: do simulation with time = 4,500ns



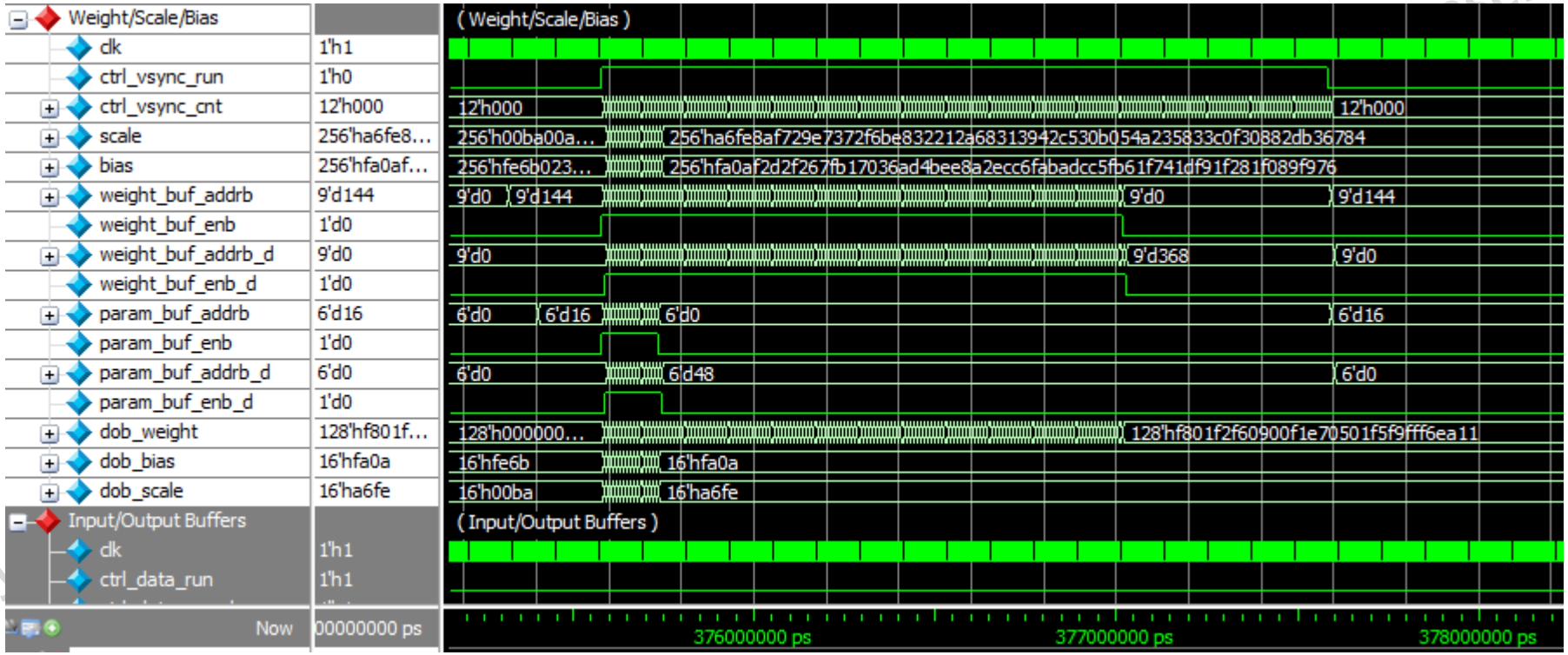
Waveform: Layer 1

- Unit test/Debugging:
  - Layer 1: do simulation with time = 4,500ns



Waveform: Layer 2

- Unit test/Debugging:
  - Layer 2: do simulation with time = 400us



Lab 1a: To do …

- Complete the missing codes to access the buffers (cnn_accel.v)
  - Weight/bias/scale buffers
- Do simulation
  - Time = 4500ns for Layer
  - Time = 400us for debugging
- How many cycles are required to preload weights, biases and scales?

Convolution kernels: Input/output buffers

- Input pixels: From in_img or feature map buffers
  - Controlled by the layer index (is_first_layer/is_last_layer) and out_buff_sel
- Output pixels: To dual buffers
  - Controlled by the buffer selection flag (out_buff_sel).

Ports	In/Out	Definition	Source/Destination		
			Layer #1	Layer #2	Layer #3
is_last_layer	In	0: ReLU, 1: Linear	0	0	1
is_conv3x3	In	0: conv1x1, 1: conv3x3	0	1	1
bias_shift	In	Shift amount before adding a bias	9	17	17
act_shift	In	Shift amount for activation quantization	7	7	7
bias	In	Biases ← bias_buffer	0	16	32
scale	In	Scales ← scale_buffer	0	16	32
win	In	Weights ← weight_buffer	0	16	160
vld_i	In	Input valid signal ← FSM (ctrl_data_run)	FSM	FSM	FSM
din	In	Input pixels	in_img	Buffer 1	Buffer 2
vld_o	Out	Output valid signal	vld_i	vld_i	vld_i
acc_o	Out	Output pixels	Buffer 1	Buffer 2	Buffer 1

File logging

- Write the output pixels in hex files
  - Writing the data in dual buffers .

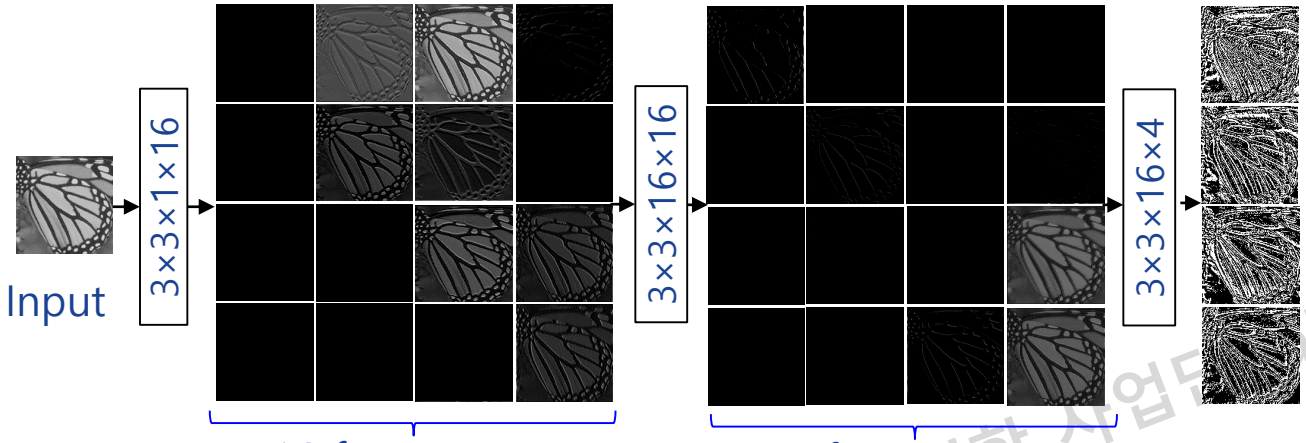
```
// Debugging
integer fp_output_L01;
integer fp_output_L02;
integer fp_output_L03;

integer idx;
always @(posedge clk or negedge rstn) begin
    if(~rstn) begin
        fp_output_L01= $fopen("out/conv_output_L01.txt", "w");
        fp_output_L02= $fopen("out/conv_output_L02.txt", "w");
        fp_output_L03= $fopen("out/conv_output_L03.txt", "w");
        idx <= 0;
    end
    else begin
        if(vld_o[0]) begin
            for(idx = To*ACT_BITS/4-1; idx >= 0; idx=idx-1) begin
                if(idx == 0) begin
                    case(q_layer_index)
                        3'd0: $fwrite(fp_output_L01,"%01h\n", acc_o[idx*4+:4]);
                        3'd1: $fwrite(fp_output_L02,"%01h\n", acc_o[idx*4+:4]);
                        3'd2: $fwrite(fp_output_L03,"%01h\n", acc_o[idx*4+:4]);
                    endcase
                end
                else begin
                    case(q_layer_index)
                        3'd0: $fwrite(fp_output_L01,"%01h", acc_o[idx*4+:4]);
                        3'd1: $fwrite(fp_output_L02,"%01h", acc_o[idx*4+:4]);
                        3'd2: $fwrite(fp_output_L03,"%01h", acc_o[idx*4+:4]);
                    endcase
                end
            end
        end
    end
end
```

Open a file identifier

Write acc_o line by line

Can you guess the data order?

Data order



butterfly_08bit.hex

```
2a
45
5b
63
6a
6c
6f
6d
6d
6a
67
64
62
61
5c
58
```

conv_output_L01.txt

```
0f0000000200000000001000361d6c00
1200000000000000003739002f424700
1200000011000000003c4d00125b4f00
0d0000001500000000002950000e5c5900
0e0000001b000000001a5400175f5d00
0c0000001900000000155500 1b5e5e00
0e0000001c000000001759001c625b00
0b0000001a000000001857001 95f5b00
0c0000001c000000001857001 8605a00
0a0000001b000000001754001 75d5a00
0a0000001a000000001951001 65a5800
0900000018000000001 94e0015575800
0900000017000000001a4d0016565800
0a0000001900000000 1b4c0014565600
0700000017000000001c470011515500
0700000016000000001d4400104e5400
```

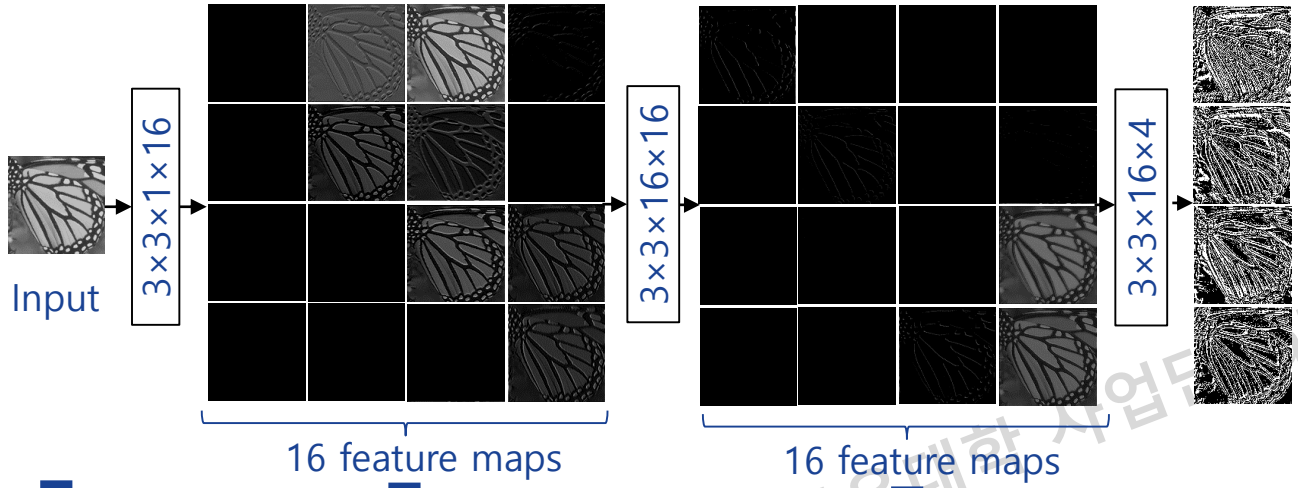
conv_output_L02.txt

```
000900001c000000000000000000000000
1c0000002000000000000000000000000
2a0800002c000000000000000000000000
2a0000002d000000000000000000000000
2d0000003000000000000000000000000
2c0000003300000000000000000000000
2d0000003400000000000000000000000
2c0000003500000000000000000000000
2d0000003300000000000000000000000
2c0000003300000000000000000000000
2a0000003100000000000000000000000
290000002f000000000000000000000
280000002f000000000000000000000
280000002d000000000000000000000
260000002c000000000000000000000
240000002900000000000000000000000
```

conv_output_L03.txt

```
0000000000000000000000000fb00fcf4
0000000000000000000000000003f21b01
0000000000000000000000000fbf70609
0000000000000000000000000001f80501
0000000000000000000000000fe0000
0000000000000000000000000ff00ffff
0000000000000000000000000ff00ff00
0000000000000000000000000ffffffff
0000000000000000000000000000000ff
0000000000000000000000000ff01fe00
0000000000000000000000000feff
0000000000000000000000000001feff
0000000000000000000000000ff00ffff
0000000000000000000000000ff00ff00
0000000000000000000000000fe01ff01
0000000000000000000000000ff00ff01
```

Data order



butterfly_08bit.hex

1	2a
2	45
3	5b
4	63
5	6a
6	6c
7	6f
8	6d
9	6d
10	6a
11	67
12	64

conv_output_L01.txt

1	0f0000000200000000001000361d6c00
2	12000000000000000003739002f424700
3	12000000110000000003c4d00125b4f00
4	0d000000150000000002950000e5c5900
5	0e0000001b000000001a5400175f5d00
6	0c0000001900000000155500 1b5e5e00
7	0e0000001c000000001759001c625b00
8	0b0000001a000000001857001 95f5b00
9	0c0000001c000000001857001 8605a00
10	0a0000001b000000001754001 75d5a00
11	0a0000001a000000001951001 65a5800
12	09000000180000000019 4e0015575800

conv_output_L02.txt

1	000900001c000000000000000000000
2	1c000000200000000000000000000000
3	2a0800002c000000000000000000000
4	2a0000002d000000000000000000000
5	2d000000300000000000000000000000
6	2c000000330000000000000000000000
7	2d000000340000000000000000000000
8	2c000000350000000000000000000000
9	2d000000330000000000000000000000
10	2c000000330000000000000000000000
11	2a000000310000000000000000000000
12	290000002f000000000000000000000

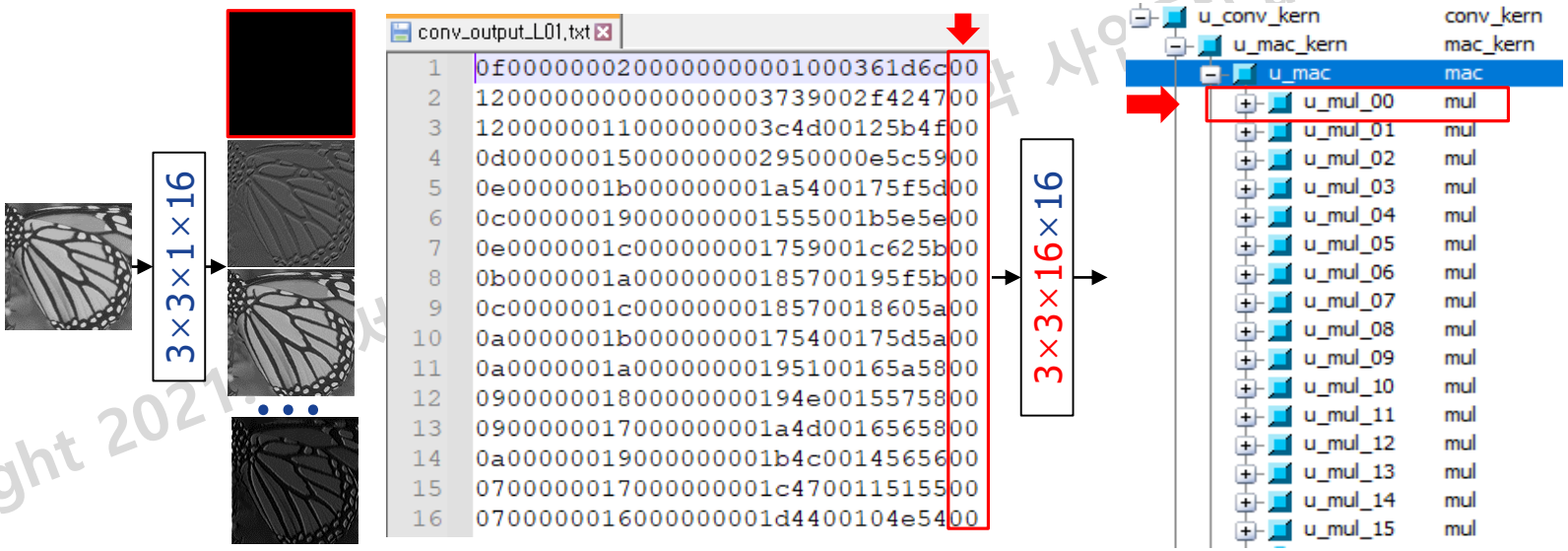
conv_output_L03.txt

1	0000000000000000000000000fb00fcf4
2	00000000000000000000000003f21b01
3	0000000000000000000000000fbf70609
4	000000000000000000000000001f80501
5	0000000000000000000000000fe0000
6	000000000000000000000000ff00ffff
7	000000000000000000000000ff00ff00
8	000000000000000000000000ffffffff
9	00000000000000000000000000000ff
10	000000000000000000000000ff01fe00
11	00000000000000000000000000feff
12	000000000000000000000000001feff

The input buffer and the dual buffers use the same indexing order.
Example: access the first word in any buffer, use data_count = 0

Data order

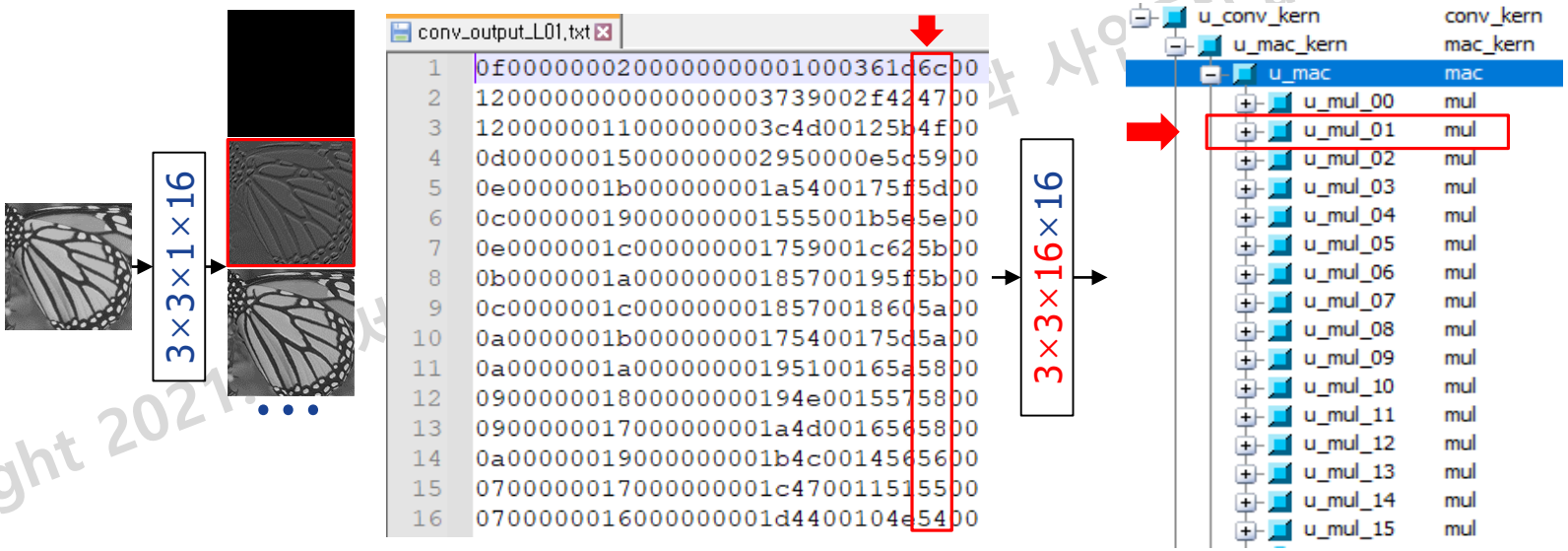
- Layer 2: num_ops=144 (=3×3×16)
  - CONV3×3 (is_conv3×3 == 1)
  - Input vector = 1×1×16
  - din[0*WI+:WI], win[0*WI+:WI]



16 feature maps conv_output_L01.txt conv_kern: Ti=16

Data order

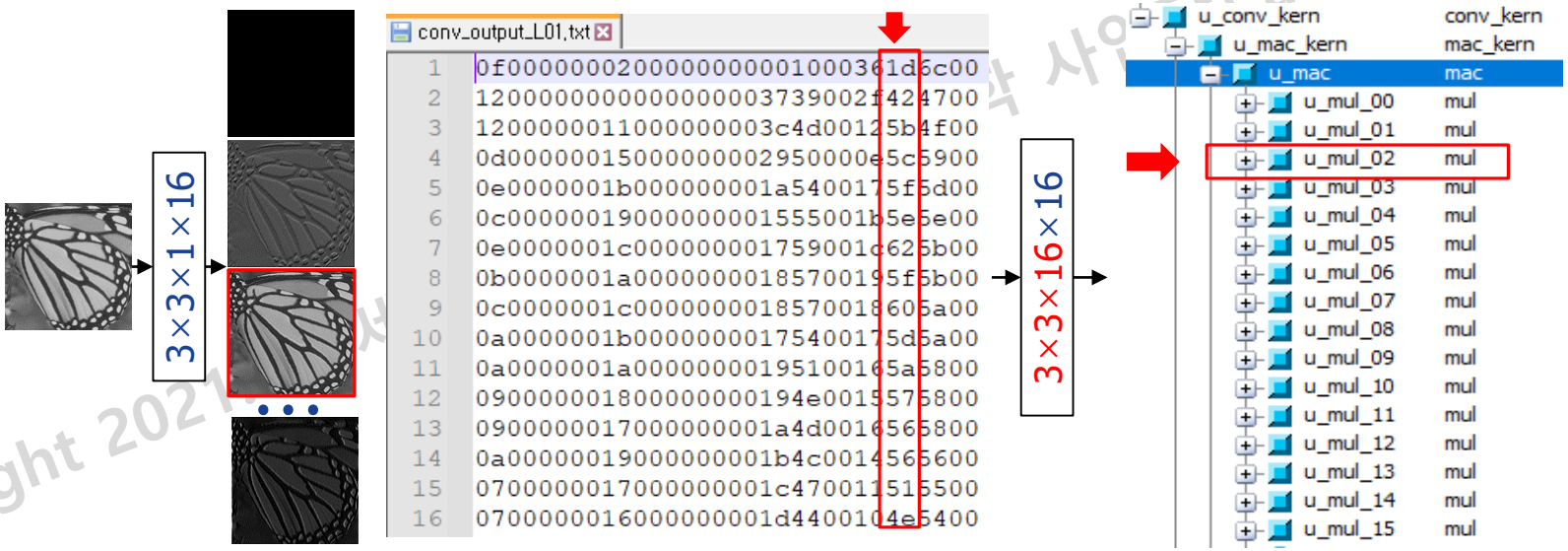
- Layer 2: num_ops=144 (=3×3×16)
  - CONV3×3 (is_conv3×3 == 1)
  - Input vector = 1×1×16
  - din[1*WI+:WI], win[1*WI+:WI]



16 feature maps conv_output_L01.txt conv_kern: Ti=16

Data order

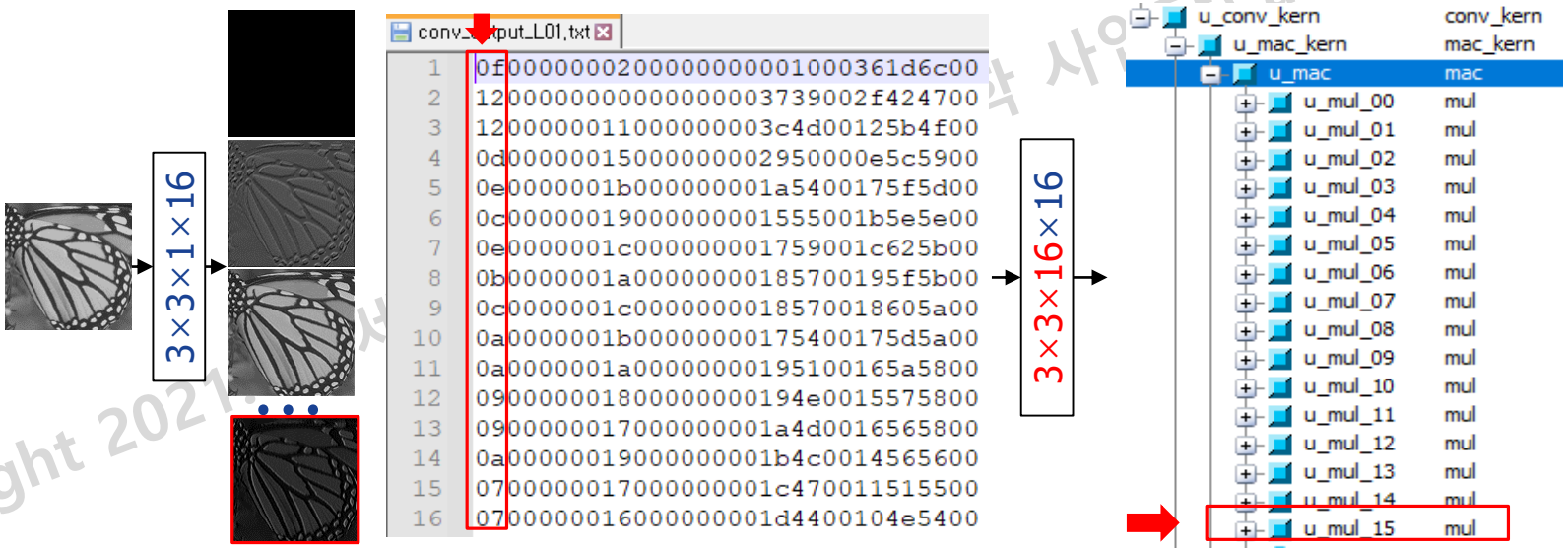
- Layer 2: num_ops=144 (=3×3×16)
  - CONV3×3 (is_conv3×3 == 1)
  - Input vector = 1×1×16
  - din[2*WI+:WI], win[2*WI+:WI]



16 feature maps conv_output_L01.txt conv_kern: Ti=16

Data order

- Layer 2: num_ops=144 (=3×3×16)
  - CONV3×3 (is_conv3×3 == 1)
  - Input vector = 1×1×16
  - din[15*WI+:WI], win[15*WI+:WI]



16 feature maps conv_output_L01.txt conv_kern: Ti=16

Feature map buffers

- Input/output signals of Port B:
  - Port B is a readout port.
  - Enable signal: fmap_buf_enb01, fmap_buf_enb02
  - Address: fmap_buf_addrb
  - Readout data: fmap_buf_dob01, fmap_buf_dob02 (to din).

```
dpram #(.W_DATA(To*ACT_BITS), .W_WORD(FRAME_SIZE_W),.N_WORD(FRAME_SIZE))
u_fmap_buff_01(
   .clk   (clk   ),
   .ena   ((!out_buff_sel) & vld_o[0]),
   .wea   ((!out_buff_sel) & vld_o[0]),
   .addra (pixel_count   ),
   .enb   (fmap_buf_enb01),
   .addrb (fmap_buf_addrb),
   .dia   (acc_o         ),
   .dob   (fmap_buf_dob01)
);

dpram #(.W_DATA(To*ACT_BITS), .W_WORD(FRAME_SIZE_W),.N_WORD(FRAME_SIZE))
u_fmap_buff_02(
   .clk   (clk   ),
   .ena   (out_buff_sel & vld_o[0]),
   .wea   (out_buff_sel & vld_o[0]),
   .addra (pixel_count   ),
   .enb   (fmap_buf_enb02),
   .addrb (fmap_buf_addrb),
   .dia   (acc_o         ),
   .dob   (fmap_buf_dob02)
);
```

To do …

- Complete the missing codes to access the dual buffers (cnn_accel.v)
  - Why? To read the dual buffer.
  - Enable signals: fmap_buf_enb01/ fmap_buf_enb02
  - Address: fmap_buf_addrb
- Why do we check the condition (!q_is_first_layer) here?

Enable signals

conv3x3

conv1x1

```
always@(*) begin
    fmap_buf_addrb = 0;
    fmap_buf_enb01 = 0;
    fmap_buf_enb02 = 0;
    if(!q_is_first_layer) begin
        fmap_buf_enb01 =  out_buff_sel & ctrl_data_run;
        fmap_buf_enb02 = !out_buff_sel & ctrl_data_run;
        if(q_is_conv3x3) begin  // Conv3x3
            case(pix_idx)
                4'd0: /*Insert your code*/;
                4'd1: /*Insert your code*/;
                4'd2: /*Insert your code*/;
                4'd3: /*Insert your code*/;
                4'd4: fmap_buf_addrb =  data_count;
                4'd5: /*Insert your code*/;
                4'd6: /*Insert your code*/;
                4'd7: /*Insert your code*/;
                4'd8: /*Insert your code*/;
                default: begin
                end
            endcase
        end
        else begin              // Conv1x1
            fmap_buf_addrb = data_count;
        end
    end
end
```

win and din

- Generate win, din and the valid sigal (vld_i) for the convolution kernels
  - vld_i: from FSM, i.e. ctrl_data_run/ctrl_data_run_d.
  - win : from win_buf
  - din : from either the input buffer (in_img) or the feature map buffers.
- Example: Layer 1

```
always@(*) begin
    din     = 0;
    win     = 0;
    vld_i   = 0;
    // First layer
    if(q_is_first_layer) begin
        vld_i = ctrl_data_run;
        din[0*WI+:WI] = (is_first_row | is_first_col)? 8'd0: in_img[data_count - q_width - 1];
        din[1*WI+:WI] = (is_first_row              )? 8'd0: in_img[data_count - q_width    ];
        din[2*WI+:WI] = (is_first_row | is_last_col )? 8'd0: in_img[data_count - q_width + 1];
        din[3*WI+:WI] = (             is_first_col)? 8'd0: in_img[data_count           - 1];
        din[4*WI+:WI] =                                   in_img[data_count             ];
        din[5*WI+:WI] = (             is_last_col )? 8'd0: in_img[data_count           + 1];
        din[6*WI+:WI] = (is_last_row  | is_first_col)? 8'd0: in_img[data_count + q_width - 1];
        din[7*WI+:WI] = (is_last_row               )? 8'd0: in_img[data_count + q_width    ];
        din[8*WI+:WI] = (is_last_row  | is_last_col )? 8'd0: in_img[data_count + q_width + 1];
        win   = win_buf[0+:To*Ti*WI];
    end
```

To do …

- Complete the missing codes to generate din
  - Conv3x3

- Hints:

➢ If a layer is not Layer 1, din is read from dual buffers.

➢ Why we use a pixel index here?

➢ Why **pix_idx_d** instead of pix_idx?

```
else begin
    vld_i = ctrl_data_run_d;
    if(q_is_conv3x3) begin      // CONV3x3
        if(out_buff_sel) begin
            case(pix_idx_d)
                4'd0: /*Insert your code*/;
                4'd1: /*Insert your code*/;
                4'd2: /*Insert your code*/;
                4'd3: /*Insert your code*/;
                4'd4: din = fmap_buf_dob01;
                4'd5: /*Insert your code*/;
                4'd6: /*Insert your code*/;
                4'd7: /*Insert your code*/;
                4'd8: /*Insert your code*/;
                default: begin
                end
            endcase
        end
        else begin
            case(pix_idx_d)
                4'd0: /*Insert your code*/;
                4'd1: /*Insert your code*/;
                4'd2: /*Insert your code*/;
                4'd3: /*Insert your code*/;
                4'd4: din = fmap_buf_dob02;
                4'd5: /*Insert your code*/;
                4'd6: /*Insert your code*/;
                4'd7: /*Insert your code*/;
                4'd8: /*Insert your code*/;
                default: begin
                end
            endcase
        end
        win = win_buf[pix_idx_d*(To*Ti*WI)+:(To*Ti*WI)];
    end
    else begin                  // CONV1x1
        din = out_buff_sel ? fmap_buf_dob01 : fmap_buf_dob02;
        win = win_buf[0+:To*Ti*WI];
    end
end
```

Buffer 1

Buffer 2

Weights (win)

conv1x1

Test bench (cnn_accel_tb.v)

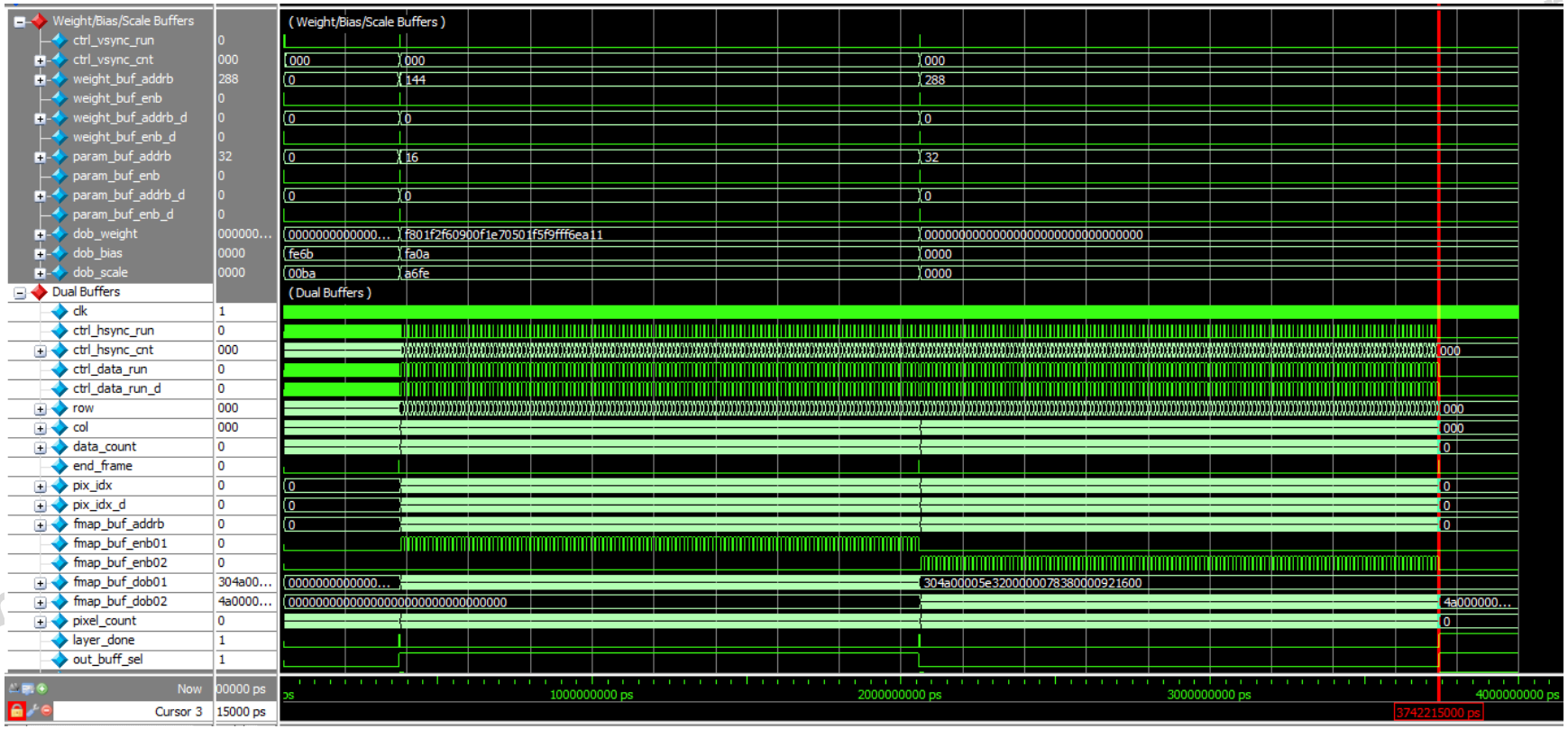
- Use a loop execute all layers in a network.

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

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

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

end
```

(1) Configure registers: layer index, is_first_layer, is_last_layer, is_conv3x3, bias_shift, act_shift and addresses.
(2) Start a layer's execution
(3) Polling until a layer is done.
(4) Reset q_layer_done
(5) Update the base addresses

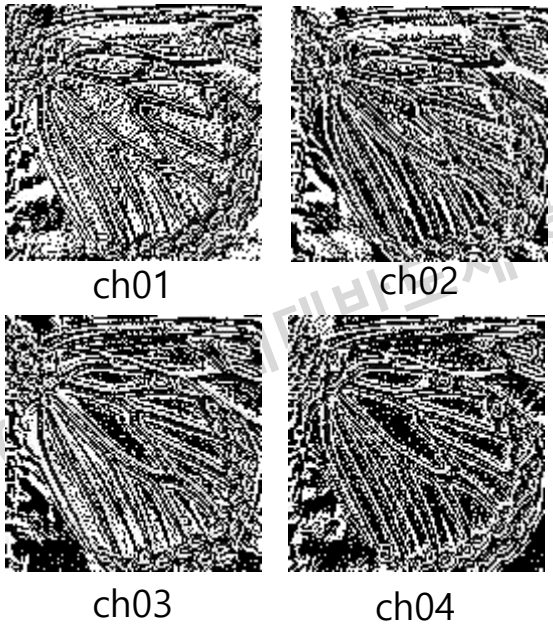
Waveform

- Do simulation with time = 4ms
  - Full simulation



Verification

- Do simulation with time = 4ms
  - Full simulation
- Four image writer modules write the output results at the folder out/




```
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*ACT_BITS+:ACT_BITS]),
./*input             */vld(vld_o[0] && q_is_last_layer),
./*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*ACT_BITS+:ACT_BITS]),
./*input             */vld(vld_o[1] && q_is_last_layer),
./*output reg        */frame_done(frame_done[1])
);

bmp_image_writer#(.WIDTH(WIDTH),.HEIGHT(HEIGHT),.OUTFILE(OUTFILE02))
u_bmp_image_writer_02(
./*input             */clk(clk),
./*input             */rstn(rstn),
./*input [WI-1:0]    */din(acc_o[2*ACT_BITS+:ACT_BITS]),
./*input             */vld(vld_o[2] && q_is_last_layer),
./*output reg        */frame_done(frame_done[2])
);

bmp_image_writer#(.WIDTH(WIDTH),.HEIGHT(HEIGHT),.OUTFILE(OUTFILE03))
u_bmp_image_writer_03(
./*input             */clk(clk),
./*input             */rstn(rstn),
./*input [WI-1:0]    */din(acc_o[3*ACT_BITS+:ACT_BITS]),
./*input             */vld(vld_o[3] && q_is_last_layer),
./*output reg        */frame_done(frame_done[3])
);
```

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.

Load the images after Layer 3

check_hardware_results.m

```
clc
clear all
close all

for ch = 1:4
    % Output from the reference S/W
    %im_sw = imread(sprintf('out_sw/ofmap_L01_ch%02d.bmp',ch));
    im_sw = imread(sprintf('out_sw/ofmap_L03_ch%02d.bmp',ch));
    % Output from the H/W simulation
    %im_hw = imread(sprintf('out/convout_layer01_ch%02d.bmp',ch));
    im_hw = imread(sprintf('out/convout_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
```

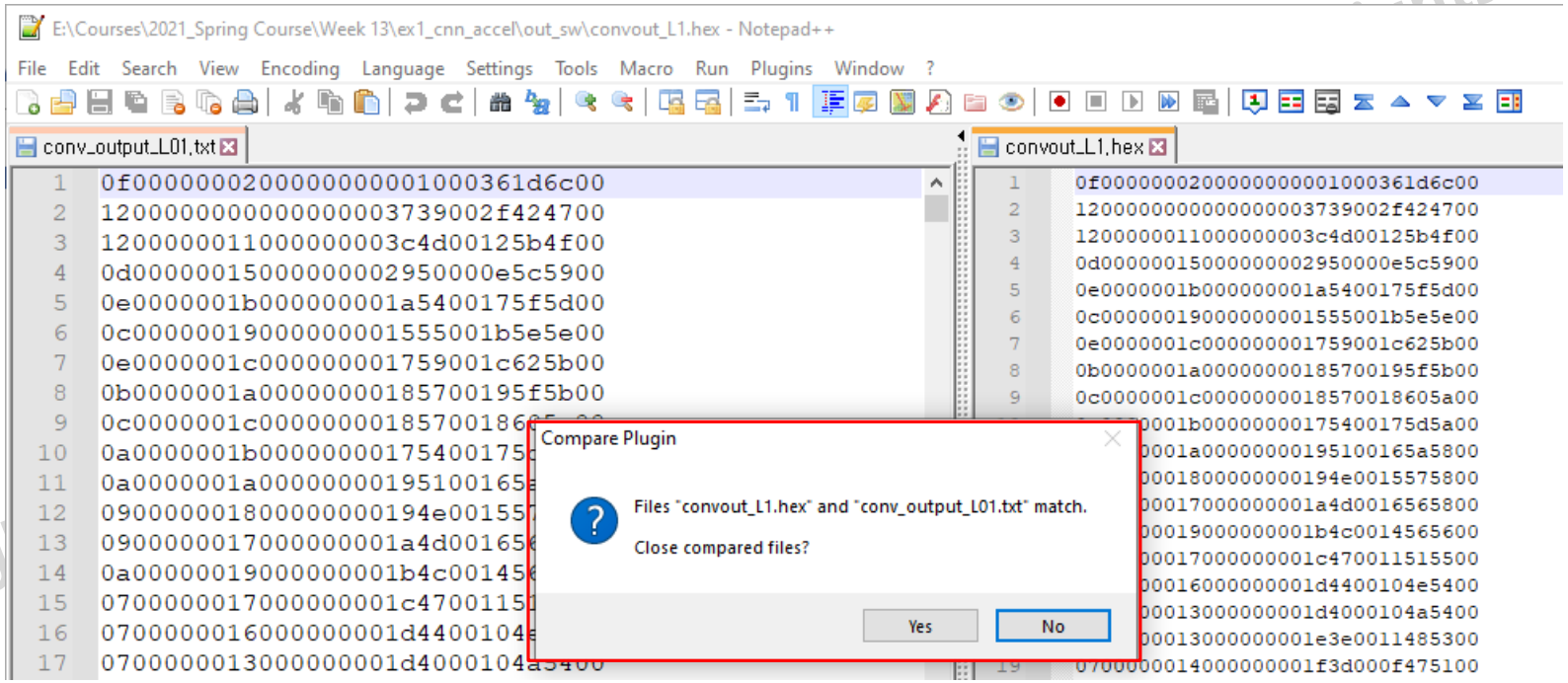
Command Window

New to MATLAB? See resources for Getting Started.

```
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!
>>
```

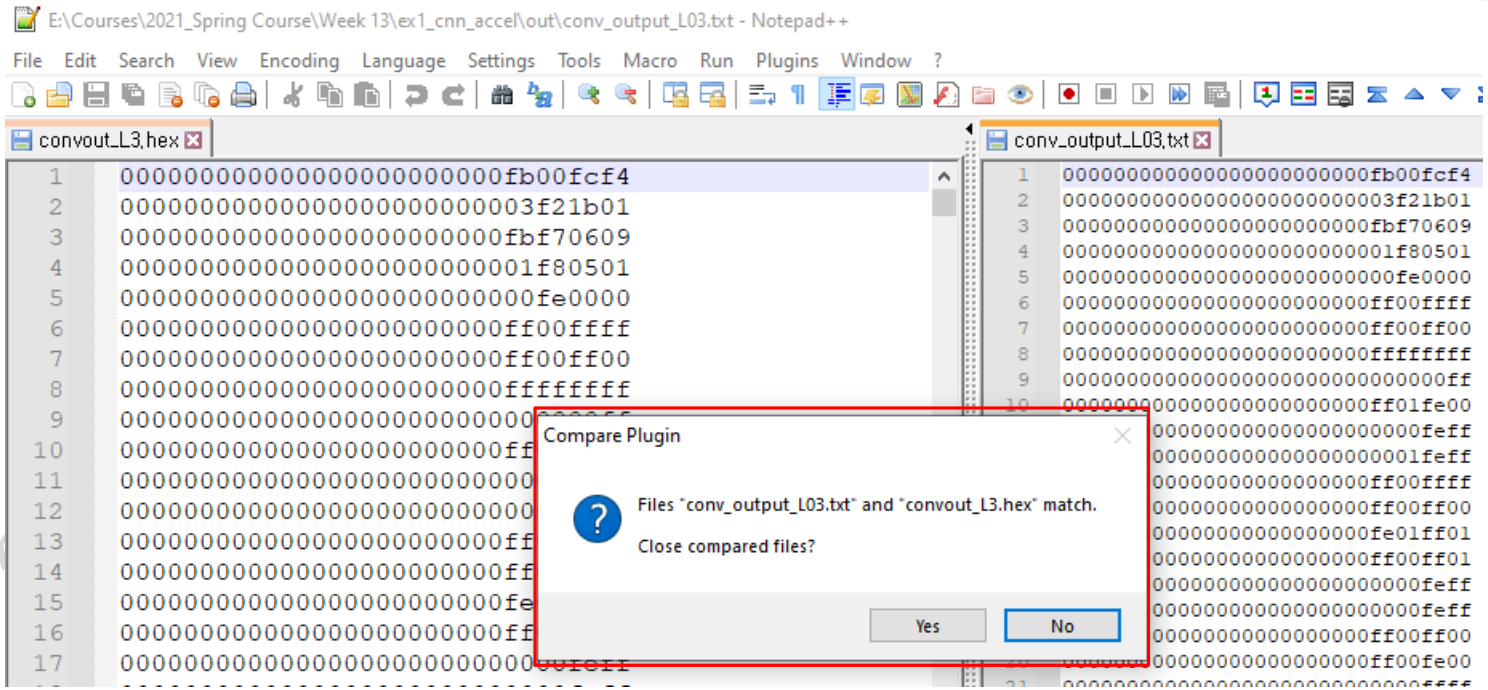
Verification

- Compare two hex files by using Notepad++
  - Plugins→ Compare → Compare (Ctrl + Alt + C)
- Example:
  - Compare the outputs of S/W and H/W for Layer 1



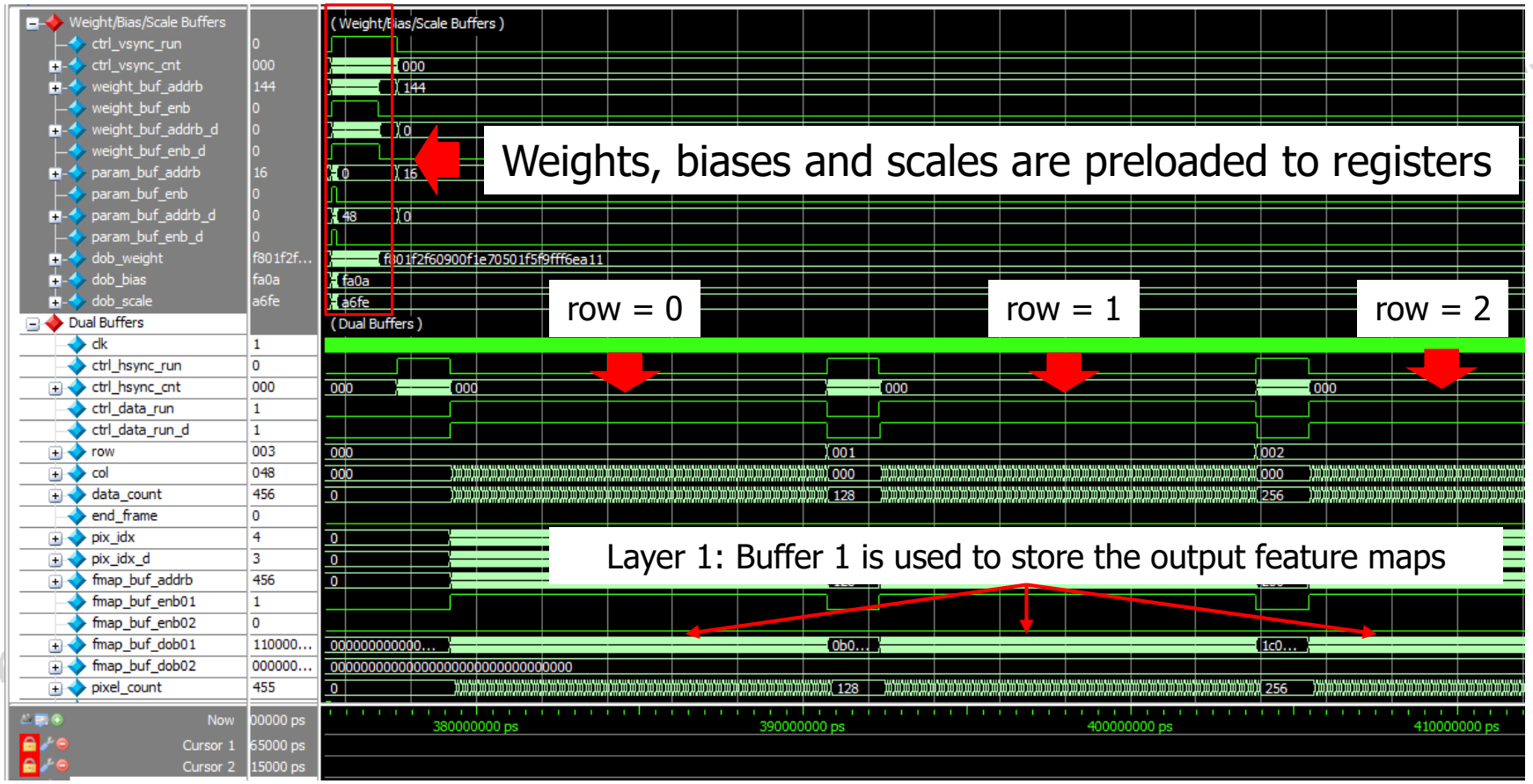
Verification

- Compare two hex files by using Notepad++
  - Plugins→ Compare → Compare (Ctrl + Alt + C)
- Example:
  - Compare the outputs of S/W and H/W for Layer 3 (The final result)



Waveform

- Unit test/debugging
  - Time = 400us

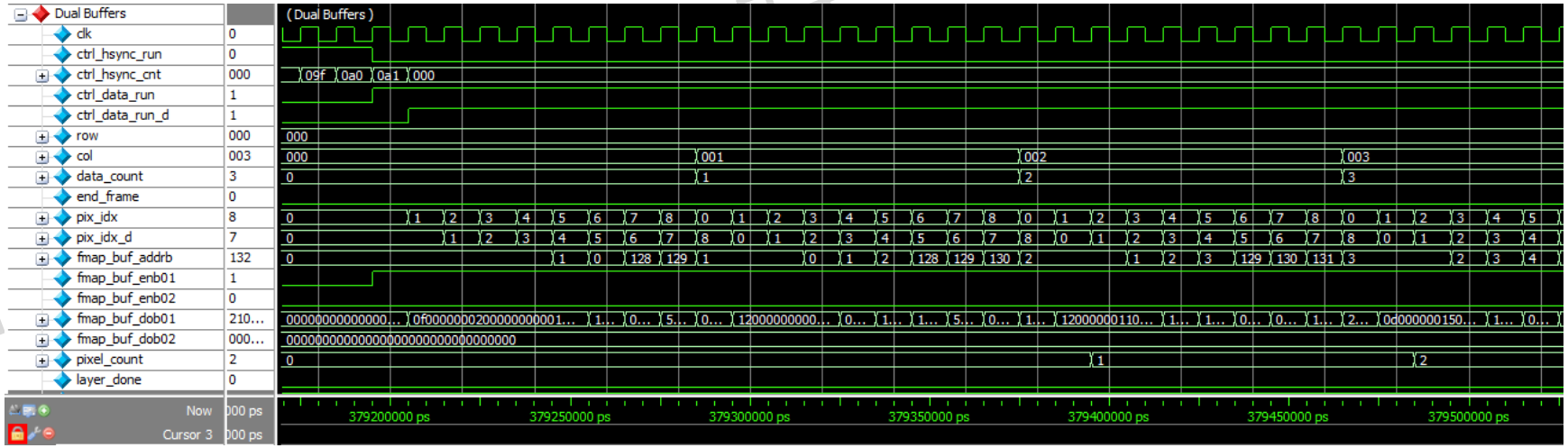


Unit test/debugging

- Verify if the output pixels, i.e. fmap_buf_dob01 is correct
- Example:
  - addr=0: 0f000000020000000001000361d6c00
  - addr=1: 12000000000000000003739002f424700
  - …

conv_output_L01.txt

```
1  0f000000020000000001000361d6c00
2  12000000000000000003739002f424700
3  12000000110000000003c4d00125b4f00
4  0d000000150000000002950000e5c5900
5  0e0000001b000000001a5400175f5d00
6  0c000000190000000001555001b5e5e00
7  0e0000001c000000001759001c625b00
8  0b0000001a000000001857001 95f5b00
9  0c0000001c000000001857001 8605a00
10 0a0000001b000000001754001 75d5a00
11 0a0000001a000000001951001 65a5800
12 090000001800000000194e0015575800
13 090000001700000000 1a4d0016565800
14 0a000000190000000 1b4c0014565600
15 070000001700000000 1c470011515500
16 070000001600000000 1d4400104e5400
```



Road map

Review

Control path, data path

System integration

Direct memory access (DMA)

Motivation: Input buffer

- The CNN accelerator assumes that the image exists (cnn_accel.v)
  - Initialized from a hex file (img/butterfly_08bit.hex)
  - Load all 16,384 pixels (=128×128) into in_img
  - When running the first layer, in_img is used to calculate din.
- **in_img must be loaded from Memory.**

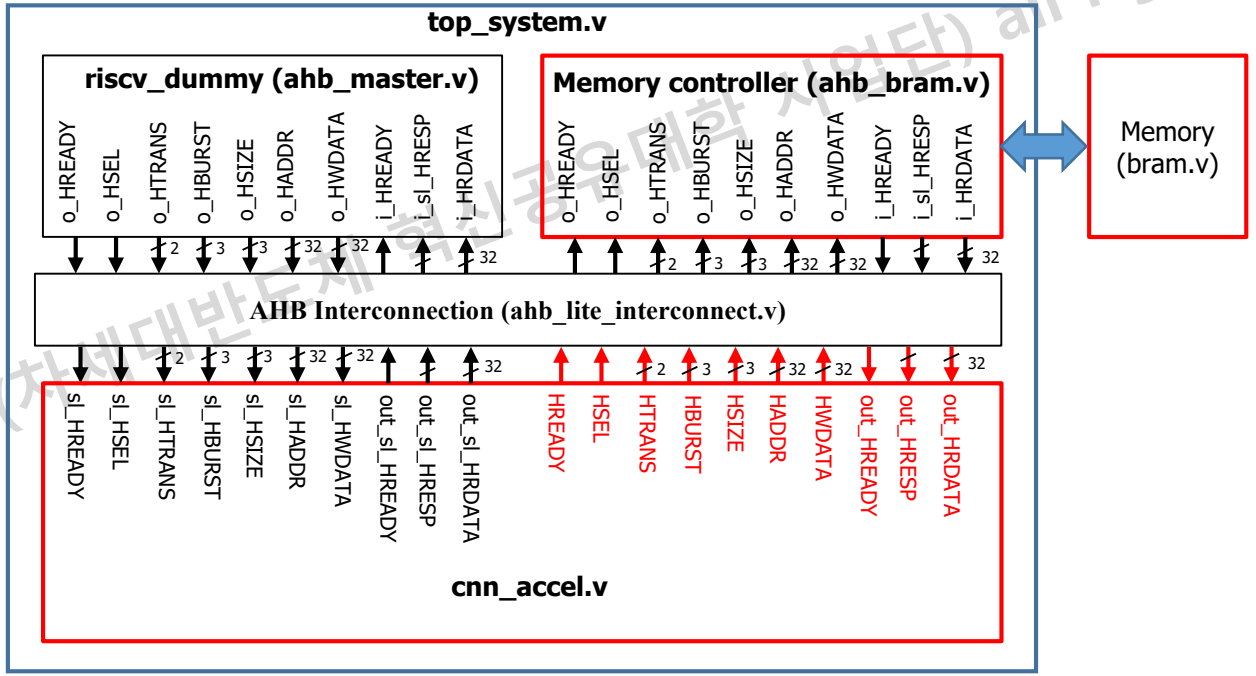
butterfly_08bit.hex

```
2a
45
5b
63
6a
6c
6f
6d
6d
6a
67
64
62
61
5c
58
```

```
//------------------------------------------------------------------------------
// Input feature buffer
//------------------------------------------------------------------------------
initial begin
    $readmemh(INFILE, in_img ,0,FRAME_SIZE-1);
end
// First layer
if(q_is_first_layer) begin
    vld_i = ctrl_data_run;
    din[0*WI+:WI] = (is_first_row | is_first_col)? 8'd0: in_img[data_count - q_width - 1];
    din[1*WI+:WI] = (is_first_row              )? 8'd0: in_img[data_count - q_width    ];
    din[2*WI+:WI] = (is_first_row | is_last_col )? 8'd0: in_img[data_count - q_width + 1];
    din[3*WI+:WI] = (             is_first_col)? 8'd0: in_img[data_count           - 1];
    din[4*WI+:WI] =                                   in_img[data_count             ];
    din[5*WI+:WI] = (             is_last_col )? 8'd0: in_img[data_count           + 1];
    din[6*WI+:WI] = (is_last_row  | is_first_col)? 8'd0: in_img[data_count + q_width - 1];
    din[7*WI+:WI] = (is_last_row              )? 8'd0: in_img[data_count + q_width    ];
    din[8*WI+:WI] = (is_last_row  | is_last_col )? 8'd0: in_img[data_count + q_width + 1];
    win   = win_buf[0+:To*Ti*WI];
end
```

Lab 2: Input buffer

- Lab 2:
  - Build a top system that consists of RISC-V, memory and CNN accelerator connected via AHB interconnect.
  - Update the input buffer by two methods
    - CNN accelerator has an AHB master port to directly access memory (DMA).



CNN accelerator (cnn_accel.v)

- The AHB Slave interface
  - Used to set the configuration registers
  - is_last_layer, is_conv3x3
  - bias_shift, act_shift
  - base_addr_weight/scale/bias.

- The AHB master interface
  - Read the input image from memory.

AHB slave

AHB master

```
//CLOCK
HCLK,
HRESETn,
// Slave port: Configuration
//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,
// Master port 1: Input image
//AHB transactor signals
HREADY,
HRESP,
HRDATA,
//output signals outgoing to the AHB lite
out_HTRANS,
out_HBURST,
out_HSIZE,
out_HPROT,
out_HMASTLOCK,
out_HADDR,
out_HWRITE,
out_HWDATA
```

Top system (top_system.v)

- A top system consists of RISC-V, memory and CNN accelerator
  - Two AHB masters (N_MASTER = 2): RISC-V and CNN Accelerator
  - Two AHB slaves (N_SLAVE = 2): Memory and CNN Accelerator

```
/*
    RISC model  : master 1
    CNN accel   : master 2 (Input image)
    MEM         : slave 1
    CNN         : slave 2
*/
//parameter N_MASTER    =   1;
parameter N_MASTER  =   2;
parameter W_MASTER  =   $clog2(N_MASTER); //GetBitWidth(N_MASTER);
parameter N_SLAVE   =   2;
parameter W_SLAVE   =   $clog2(N_SLAVE); //GetBitWidth(N_SLAVE);

parameter ADDR_START_MAP = {
        `RISCV_CNN_ACCEL_BASE_ADDR,     // 1
        `RISCV_MEMORY_BASE_ADDR         // 0
    };

parameter ADDR_END_MAP   = {
        `RISCV_CNN_ACCEL_BASE_ADDR,     // 1
        `RISCV_MEMORY_BASE_ADDR         // 0
    };


parameter ADDR_MASK = {
        `RISCV_MASK_CNN_ACCEL_BASE_ADDR,// 1
        `RISCV_MASK_MEMORY_BASE_ADDR    // 0
    };
```

Master, slave configuration

- Connect ports from masters and slaves to AHB interconnect.

Masters

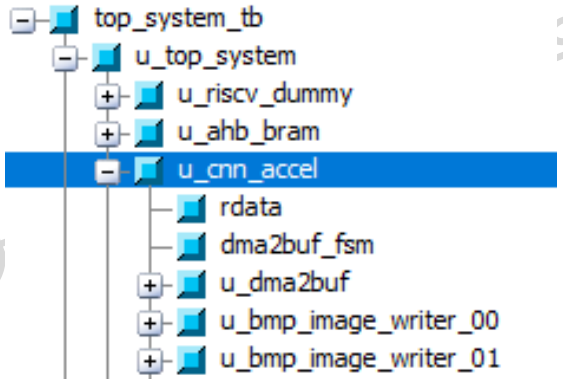
```
//------------------------------------------------------------------
//  AHB Masters
//------------------------------------------------------------------
// 0. RISCmodel
assign  w_AHB_IC_ma_HREADY    [0]                    =   1'b1                    ;
assign  w_AHB_IC_ma_HSEL      [0]                    =   |w_RISC2AHB_mst_HTRANS  ;
assign  w_AHB_IC_ma_HTRANS    [0*2+:2]               =   w_RISC2AHB_mst_HTRANS   ;
assign  w_AHB_IC_ma_HBURST    [0*`W_BURST+:`W_BURST] =   w_RISC2AHB_mst_HBURST   ;
assign  w_AHB_IC_ma_HSIZE     [0*3+:3]               =   w_RISC2AHB_mst_HSIZE    ;
assign  w_AHB_IC_ma_HPROT     [0*4+:4]               =   4'h0                    ;
assign  w_AHB_IC_ma_HMASTLOCK[0]                     =   |w_RISC2AHB_mst_HTRANS  ;
assign  w_AHB_IC_ma_HADDR     [0*32+:32]             =   w_RISC2AHB_mst_HADDR    ;
assign  w_AHB_IC_ma_HWRITE    [0]                    =   w_RISC2AHB_mst_HWRITE   ;
assign  w_AHB_IC_ma_HWDATA    [0*32+:32]             =   w_RISC2AHB_mst_HWDATA   ;
assign  w_RISC2AHB_mst_HREADY                        =   w_AHB_IC_out_ma_HREADY  [0];
assign  w_RISC2AHB_mst_HRESP                         =   w_AHB_IC_out_ma_HRESP   [0];
assign  w_RISC2AHB_mst_HRDATA                        =   w_AHB_IC_out_ma_HRDATA  [0*32+:32]  ;


// 1. CNN Accelerator: Input image
assign  w_AHB_IC_ma_HREADY    [1]                    =   1'b1                    ;
assign  w_AHB_IC_ma_HSEL      [1]                    =   |w_cnn_img_mst_HTRANS   ;
assign  w_AHB_IC_ma_HTRANS    [1*2+:2]               =   w_cnn_img_mst_HTRANS    ;
assign  w_AHB_IC_ma_HBURST    [1*`W_BURST+:`W_BURST] =   w_cnn_img_mst_HBURST    ;
assign  w_AHB_IC_ma_HSIZE     [1*3+:3]               =   w_cnn_img_mst_HSIZE ;
assign  w_AHB_IC_ma_HPROT     [1*4+:4]               =   4'h0                    ;
assign  w_AHB_IC_ma_HMASTLOCK[1]                     =   |w_cnn_img_mst_HTRANS   ;
assign  w_AHB_IC_ma_HADDR     [1*32+:32]             =   w_cnn_img_mst_HADDR ;
assign  w_AHB_IC_ma_HWRITE    [1]                    =   w_cnn_img_mst_HWRITE    ;
assign  w_AHB_IC_ma_HWDATA    [1*32+:32]             =   w_cnn_img_mst_HWDATA    ;
assign  w_cnn_img_mst_HREADY                         =   w_AHB_IC_out_ma_HREADY  [1];
assign  w_cnn_img_mst_HRESP                          =   w_AHB_IC_out_ma_HRESP   [1];
assign  w_cnn_img_mst_HRDATA                         =   w_AHB_IC_out_ma_HRDATA  [1*32+:32]  ;
```

Slaves

```
//------------------------------------------------------------------
//  AHB Slaves
//------------------------------------------------------------------
//  0. AHB2MEM
assign  mem_sl_HSEL                  =   w_AHB_IC_out_sl_HSEL    [0]     ;
assign  mem_sl_HADDR                 =   w_AHB_IC_out_sl_HADDR   [0*32+:32]  ;
assign  mem_sl_HTRANS                =   w_AHB_IC_out_sl_HTRANS  [0*2+:2]    ;
assign  mem_sl_HBURST                =   w_AHB_IC_out_sl_HBURST  [0*`W_BURST+:`W_BURST]  ;
assign  mem_sl_HSIZE                 =   w_AHB_IC_out_sl_HSIZE   [0*3+:3]    ;
assign  mem_sl_HPROT                 =   w_AHB_IC_out_sl_HPROT   [0*4+:4]    ;
assign  mem_sl_HWRITE                =   w_AHB_IC_out_sl_HWRITE  [0]     ;
assign  mem_sl_HWDATA                =   w_AHB_IC_out_sl_HWDATA  [0*32+:32]  ;
assign  mem_sl_HREADY                =   w_AHB_IC_out_sl_HREADY  [0]     ;
assign  w_AHB_IC_sl_HREADY   [0]     =   out_mem_sl_HREADY;
assign  w_AHB_IC_sl_HRESP    [0*2+:2]    =   out_mem_sl_HRESP;
assign  w_AHB_IC_sl_HRDATA   [0*32+:32]  =   out_mem_sl_HRDATA;

//  1. AHB2CNN
assign  cnn_sl_HSEL                  =   w_AHB_IC_out_sl_HSEL    [1]     ;
assign  cnn_sl_HADDR                 =   w_AHB_IC_out_sl_HADDR   [1*32+:32]  ;
assign  cnn_sl_HTRANS                =   w_AHB_IC_out_sl_HTRANS  [1*2+:2]    ;
assign  cnn_sl_HBURST                =   w_AHB_IC_out_sl_HBURST  [1*`W_BURST+:`W_BURST]  ;
assign  cnn_sl_HSIZE                 =   w_AHB_IC_out_sl_HSIZE   [1*3+:3]    ;
assign  cnn_sl_HPROT                 =   w_AHB_IC_out_sl_HPROT   [1*4+:4]    ;
assign  cnn_sl_HWRITE                =   w_AHB_IC_out_sl_HWRITE  [1]     ;
assign  cnn_sl_HWDATA                =   w_AHB_IC_out_sl_HWDATA  [1*32+:32]  ;
assign  cnn_sl_HREADY                =   w_AHB_IC_out_sl_HREADY  [1]     ;
assign  w_AHB_IC_sl_HREADY   [1]     =   out_cnn_sl_HREADY;
assign  w_AHB_IC_sl_HRESP    [1*2+:2]    =   out_cnn_sl_HRESP;
assign  w_AHB_IC_sl_HRDATA   [1*32+:32]  =   out_cnn_sl_HRDATA;
```

CNN accelerator (cnn_accel.v)

- CNN accelerator wants to generate in_img from memory.
  - DMA interface (dma2buf.v)
  - Two image writers
- We exclude many modules
  - Computing units (conv_kern.v, mac_kern.v, bnorm_quant_act.v)
  - Weight/bias/scale buffers
  - Output buffers




```
//------------------------------------------------
// Input feature buffer
//------------------------------------------------
//initial begin
//  $readmemh(INFILE, in_img ,0,FRAME_SIZE-1);
//end
integer i;
always @(posedge HCLK or negedge HRESETn)  begin
    if(!HRESETn)begin
        for(i = 0; i < FRAME_SIZE; i=i+1) begin
            in_img[i] <= 8'd0;
        end
    end
    else begin
        /* Insert your code*/
    end
end
```

How to send data from mem to in_img?

- Method 1: Similar to LCD drive, how about using CPU for this task?
- In the testbench (top_system_tb.v), use a loop
  - Read data in Memory at the address i, and store it to rdata.
    - Write rdata to in_img.

```
//*****************************************************************************************************************
// Slow loading
// 1. CPU reads data in Memory
// 2. CPU stores data to the input buffer of the CNN accelerator
//*****************************************************************************************************************
#(8*p)
// Load data to the frame buffer of LCD drive
for(i = 0; i < WIDTH * HEIGHT; i=i+1) begin
    #(4*p) @(posedge HCLK) u_top_system.u_riscv_dummy.task_AHBread( `RISCV_MEMORY_BASE_ADDR + 4*i, rdata );     // Read from SRAM
    #(4*p) @(posedge HCLK) u_top_system.u_riscv_dummy.task_AHBwrite(`CNN_ACCEL_INPUT_IMAGE  + (i << (WB_DATA + W_REGS)), rdata );
end
$display("T=%03t ns: SLOW loading the image by CPU is DONE!!!\n", $realtime/1000);
```

- Do simulation with time = 2.8ms

```
VSIM 60> run 2.8ms
# GetModuleFileName: The specified module could not be found.
#
#
# T=2621525 ns: SLOW loading the image by CPU is DONE!!!
#
```

Waveform

- cnn_accel is busy to get pixel data from RISC-V via its AHB slave port

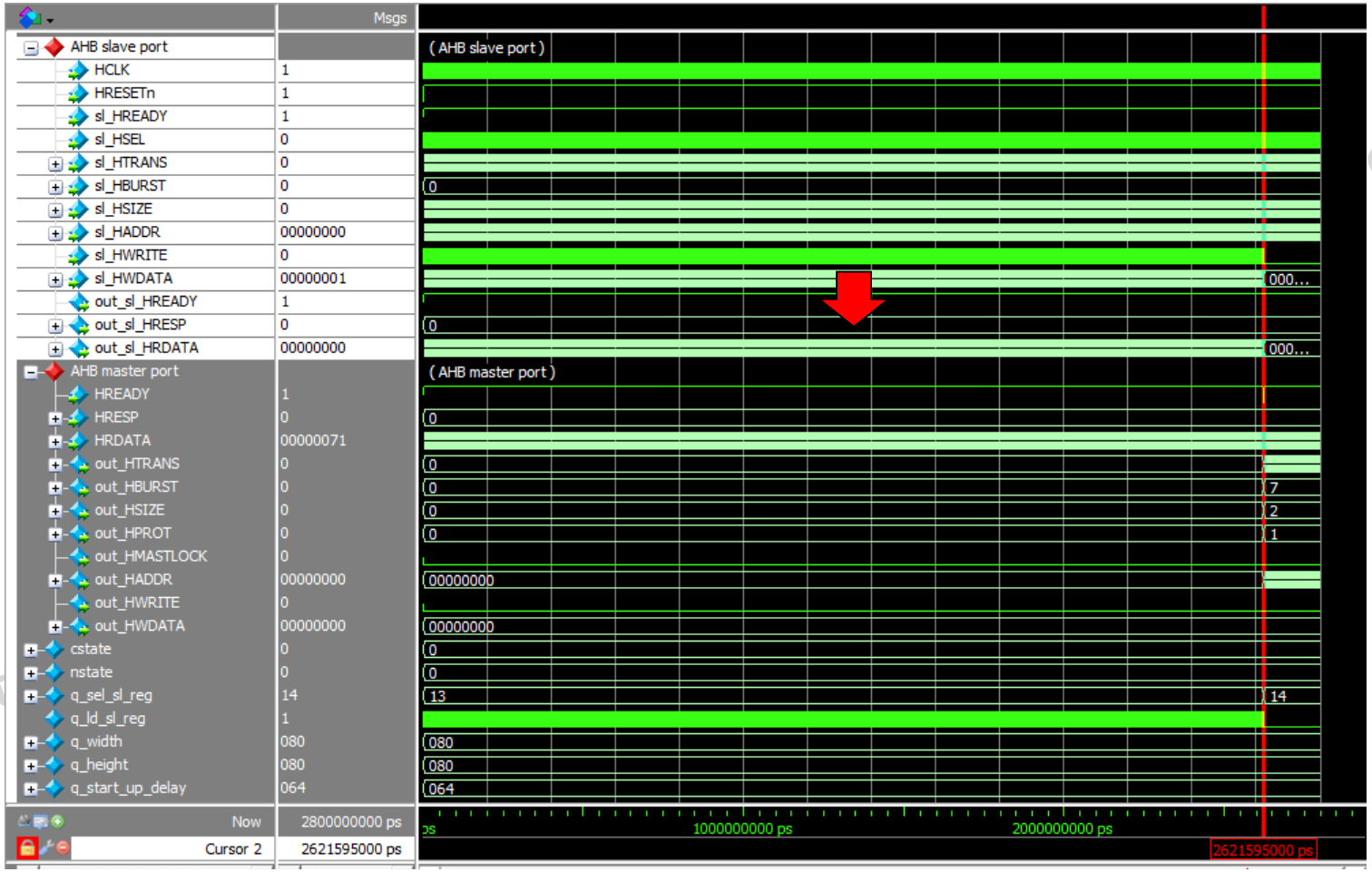


Image writers

- Two bmp image writers are used to store pixel data

```
//------------------------------------------------
// Image Writer
//------------------------------------------------
// synopsys translate_off
wire write_image_by_cpu_done;
wire write_image_by_dma_done;
bmp_image_writer#(.WIDTH(WIDTH),.HEIGHT(HEIGHT),.OUTFILE("out/input_image_by_cpu.bmp"))
u_bmp_image_writer_00(
./*input             */clk(clk),
./*input             */rstn(rstn),
./*input [WI-1:0]    */din(sl_HWDATA[7:0]),
./*input             */vld(q_ld_sl_reg && (q_sel_sl_reg == CNN_ACCEL_INPUT_IMAGE)),
./*output reg        */frame_done(write_image_by_cpu_done)
);

bmp_image_writer#(.WIDTH(WIDTH),.HEIGHT(HEIGHT),.OUTFILE("out/input_image_by_dma.bmp"))
u_bmp_image_writer_01(
./*input             */clk(clk),
./*input             */rstn(rstn),
./*input [WI-1:0]    */din(data_o_ld[7:0]),
./*input             */vld(data_vld_o_ld),
./*output reg        */frame_done(write_image_by_dma_done)
);
// synopsys translate_on
endmodule
```

Image result by CPU

- The pixel data by CPU are stored in out/input_image_by_cpu.bmp
  - It takes about 2.7 milliseconds.

```
//------------------------------------------------
// Image Writer
//------------------------------------------------
// synopsys translate_off
wire write_image_by_cpu_done;
wire write_image_by_dma_done;
bmp_image_writer#(.WIDTH(WIDTH),.HEIGHT(HEIGHT),.OUTFILE("out/input_image_by_cpu.bmp"))
u_bmp_image_writer_00(
./*input             */clk(clk),
./*input             */rstn(rstn),
./*input [WI-1:0]    */din(sl_HWDATA[7:0]),
./*input             */vld(q_ld_sl_reg && (q_sel_sl_reg == CNN_ACCEL_INPUT_IMAGE)),
./*output reg        */frame_done(write_image_by_cpu_done)
);

bmp_image_writer#(.WIDTH(WIDTH),.HEIGHT(HEIGHT),.OUTFILE("out/input_image_by_dma.bmp"))
u_bmp_image_writer_01(
./*input             */clk(clk),
./*input             */rstn(rstn),
./*input [WI-1:0]    */din(data_o_ld[7:0]),
./*input             */vld(data_vld_o_ld),
./*output reg        */frame_done(write_image_by_dma_done)
);
// synopsys translate_on
endmodule
```

How to send data from mem to in_img?

- Method 2: How about using DMA for this task?
  - In the testbench (top_system_tb.v), allow CNN accelerator to directly access memory
    - Set a flag
    - Polling: wait until all pixel data are read by DMA

- Do simulation with time = 3ms

```
//*************************************************************************************************************
// FAST loading: CPU enables CNN Accelerator to become a bus Master
//*************************************************************************************************************
#(4*p) @(posedge HCLK) u_top_system.u_riscv_dummy.task_AHBwrite(`CNN_ACCEL_INPUT_IMAGE_LOAD, 1);    // Start loading the input
#(4*p) @(posedge HCLK) u_top_system.u_riscv_dummy.task_AHBwrite(`CNN_ACCEL_INPUT_IMAGE_LOAD, 0);

while(!image_load_done) begin
    #(128*p) @(posedge HCLK)  u_top_system.u_riscv_dummy.task_AHBread(`CNN_ACCEL_INPUT_IMAGE_LOAD,image_load_done);
end
$display("T=%03t ns: FAST loading the image by DMA is DONE!!!\n", $realtime/1000);
```

dma2buf.v

- An AHB master interface between an on-chip buffer and an external (off-chip) memory.
  - Inputs
    - AHB master ports:
      - HREADY, HRESP, HRDATA
      - out_HTRANS, out_HBURST, out_ADDR, out_HWDATA
    - start_dma: an enable signal to activate DMA
    - num_trans: number of read transactions
    - start_addr: starting address for access
  - Outputs
    - data_o[31:0]: returned data
    - data_vld_o: valid signal
    - data_cnt_o: index of data
    - data_last_o: done beat

```
module dma2buf (
    //AHB transactor signals
    HREADY,
    HRESP,
    HRDATA,
    //output signals outgoing to the AHB lite
    out_HTRANS,
    out_HBURST,
    out_HSIZE,
    out_HPROT,
    out_HMASTLOCK,
    out_HADDR,
    out_HWRITE,
    out_HWDATA,
   //control signal
   start_dma,
   num_trans,
   start_addr,
   //Output
   data_o,
   data_vld_o,
   data_cnt_o,
   data_last_o,

   //Global signals
   clk,
   resetn
);
```

DMA's finite state machine

- Four states
  - ST_IMG_IDLE: idle state
    - If CPU sets q_input_image_load to ONE
      - Grant CNN IP to directly access memory
  - ST_IMG_LINE_DATA_LOAD
    - Activate signal
  - ST_IMG_LINE_DATA_WAIT
    - Polling while waiting data from memory
  - ST_IMG_DON
    - DMA completes loading a frame

```
always@(posedge clk, negedge rstn)begin
    if(~rstn)
        c_state_dma <= ST_IMG_IDLE;
    else begin
        c_state_dma <= n_state_dma;
    end
end

always @(*) begin: dma2buf_fsm
    case(c_state_dma)
        ST_IMG_IDLE: begin
            if(q_input_image_load)
                n_state_dma = ST_IMG_LINE_DATA_LOAD;
            else
                n_state_dma = ST_IMG_IDLE;
        end
        ST_IMG_LINE_DATA_LOAD: begin
            n_state_dma = ST_IMG_LINE_DATA_WAIT;
        end
        ST_IMG_LINE_DATA_WAIT: begin
            if(data_last_o_ld) begin
                if(start_line_ld == q_height-1)
                    n_state_dma = ST_IMG_DONE;
                else
                    n_state_dma = ST_IMG_LINE_DATA_LOAD;
            end
            else
                n_state_dma = ST_IMG_LINE_DATA_WAIT;
        end
        ST_IMG_DONE:
            n_state_dma = ST_IMG_IDLE;
        default:
            n_state_dma = ST_IMG_IDLE;
    endcase
end
```

DMA's FSM: start

- DMA is initialized at an IDLE state
- When CPU grants the right to access memory to DMA, DMA moves to active states

cnn_accel.v

```
always @(*) begin: dma2buf_fsm
    case(c_state_dma)
        ST_IMG_IDLE: begin
            if(q_input_image_load)
                n_state_dma = ST_IMG_LINE_DATA_LOAD;
            else
                n_state_dma = ST_IMG_IDLE;
        end
        ST_IMG_LINE_DATA_LOAD: begin
            n_state_dma = ST_IMG_LINE_DATA_WAIT;
        end
        ST_IMG_LINE_DATA_WAIT: begin
            if(data_last_o_ld) begin
                if(start_line_ld == q_height-1)
                    n_state_dma = ST_IMG_DONE;
                else
                    n_state_dma = ST_IMG_LINE_DATA_LOAD;
```

top_system_tb.v

```
//*************************************************************************************************
// FAST loading: CPU enables CNN Accelerator to become a bus Master
//*************************************************************************************************
#(4*p) @(posedge HCLK) u_top_system.u_riscv_dummy.task_AHBwrite(`CNN_ACCEL_INPUT_IMAGE_BASE, `RISCV_MEMORY_BASE_ADDR);
#(4*p) @(posedge HCLK) u_top_system.u_riscv_dummy.task_AHBwrite(`CNN_ACCEL_INPUT_IMAGE_LOAD, 1);    // Start loading the input
#(4*p) @(posedge HCLK) u_top_system.u_riscv_dummy.task_AHBwrite(`CNN_ACCEL_INPUT_IMAGE_LOAD, 0);

while(!image_load_done) begin
    #(128*p) @(posedge HCLK)  u_top_system.u_riscv_dummy.task_AHBread(`CNN_ACCEL_INPUT_IMAGE_LOAD,image_load_done);
end
$display("T=%03t ns: FAST loading the image by DMA is DONE!!!\n", $realtime/1000);
```

DMA's FSM: start a request

- ST_IMG_LINE_DATA_LOAD
  - DMA starts a request
  - start_dma_ld = 1'b1

```
// Sending a request
always @(*) begin
    n_state_dma = c_state_dma;
    start_dma_ld = 1'b0;
    num_trans_ld = q_width;
    if(c_state_dma == ST_IMG_LINE_DATA_LOAD) begin
        start_dma_ld = 1'b1;
    end
end
```

```
always @(*) begin: dma2buf_fsm
    case(c_state_dma)
        ST_IMG_IDLE: begin
            if(q_input_image_load)
                n_state_dma = ST_IMG_LINE_DATA_LOAD;
            else
                n_state_dma = ST_IMG_IDLE;
        end
        ST_IMG_LINE_DATA_LOAD: begin
            n_state_dma = ST_IMG_LINE_DATA_WAIT;
        end
        ST_IMG_LINE_DATA_WAIT: begin
            if(data_last_o_ld) begin
                if(start_line_ld == q_height-1)
                    n_state_dma = ST_IMG_DONE;
                else
                    n_state_dma = ST_IMG_LINE_DATA_LOAD;
            end
            else
                n_state_dma = ST_IMG_LINE_DATA_WAIT;
        end
        ST_IMG_DONE:
            n_state_dma = ST_IMG_IDLE;
        default:
            n_state_dma = ST_IMG_IDLE;
    endcase
end
```

DMA's FSM: wait

- ST_IMG_LINE_DATA_WAIT
  - Waiting data until the last one comes (data_last_o_ld == 1)
  - If (data_last_o_ld == 1)
    - If this is the last line
      - Go to ST_IMG_DONE
    - Else
      - Go to ST_IMG_LINE_DATA_LOAD
        - Load another line

```
always @(*) begin: dma2buf_fsm
    case(c_state_dma)
        ST_IMG_IDLE: begin
            if(q_input_image_load)
                n_state_dma = ST_IMG_LINE_DATA_LOAD;
            else
                n_state_dma = ST_IMG_IDLE;
        end
        ST_IMG_LINE_DATA_LOAD: begin
            n_state_dma = ST_IMG_LINE_DATA_WAIT;
        end
        ST_IMG_LINE_DATA_WAIT: begin
            if(data_last_o_ld) begin
                if(start_line_ld == q_height-1)
                    n_state_dma = ST_IMG_DONE;
                else
                    n_state_dma = ST_IMG_LINE_DATA_LOAD;
            end
            else
                n_state_dma = ST_IMG_LINE_DATA_WAIT;
        end
        ST_IMG_DONE:
            n_state_dma = ST_IMG_IDLE;
        default:
            n_state_dma = ST_IMG_IDLE;
    endcase
end
```

Line counter and starting address

```
// Update request address and line counter
always@(posedge clk, negedge rstn)begin
    if(~rstn) begin
        start_addr_ld   <= 0;
        start_line_ld   <= 0;
        load_image_done <= 0;
    end
    else begin
        if(q_ld_sl_reg && q_sel_sl_reg == CNN_ACCEL_INPUT_IMAGE_LOAD) begin
            start_line_ld   <= 0;
            start_addr_ld   <= q_input_image_base_addr;
            load_image_done <= 0;
        end
        else begin
            if(data_last_o_ld) begin    // End of loading a line
                if(start_line_ld == q_height-1) begin   // Last line
                    start_line_ld <= 0;
                    start_addr_ld <= q_input_image_base_addr;
                    load_image_done <= 1'b1;
                end
                else begin  // Normal line
                    //start_line_ld <= /*Insert your code*/
                    //start_addr_ld <= /*Insert your code*/
                end
            end
        end
    end
end
```

Both DMA's line counter and access address must be updated

Update the input buffer

- For an incoming data (data_vld_o_ld == 1)
  - Update the pixel counter (in_pixel_count)
  - Copy it to the input buffer

```
//-----------------------------------------------------
// Input feature buffer
//-----------------------------------------------------
//initial begin
//  $readmemh(INFILE, in_img ,0,FRAME_SIZE-1);
//end
integer k;
always@(posedge clk, negedge rstn)begin
    if(~rstn) begin
        in_pixel_count <= 0;
    end
    else begin
        if(data_vld_o_ld) begin
            // Insert your code
        end
    end
end

always@(posedge clk, negedge rstn)begin
    if(~rstn) begin
        for(k = 0; k < q_frame_size; k=k+1) begin
            in_img[k] <= 8'd0;
        end
    end
    else begin
        if(data_vld_o_ld) begin
            // Insert your code
        end
    end
end
```

Test case (top_system.v)

- In the testbench (top_system_tb.v), allow CNN accelerator to directly access memory
  - Set a flag (CNN_ACCEL_INPUT_IMAGE_LOAD)
  - Polling:
    - Wait until all pixel data are read by DMA

```
//*************************************************************************************************************
// FAST loading: CPU enables CNN Accelerator to become a bus Master
//*************************************************************************************************************
#(4*p) @(posedge HCLK) u_top_system.u_riscv_dummy.task_AHBwrite(`CNN_ACCEL_INPUT_IMAGE_LOAD, 1);    // Start loading the input
#(4*p) @(posedge HCLK) u_top_system.u_riscv_dummy.task_AHBwrite(`CNN_ACCEL_INPUT_IMAGE_LOAD, 0);

while(!image_load_done) begin
    #(128*p) @(posedge HCLK)  u_top_system.u_riscv_dummy.task_AHBread(`CNN_ACCEL_INPUT_IMAGE_LOAD,image_load_done);
end
$display("T=%03t ns: FAST loading the image by DMA is DONE!!!\n", $realtime/1000);
```

Waveform

- cnn_accel is busy to get pixel data from RISC-V via its AHB master port

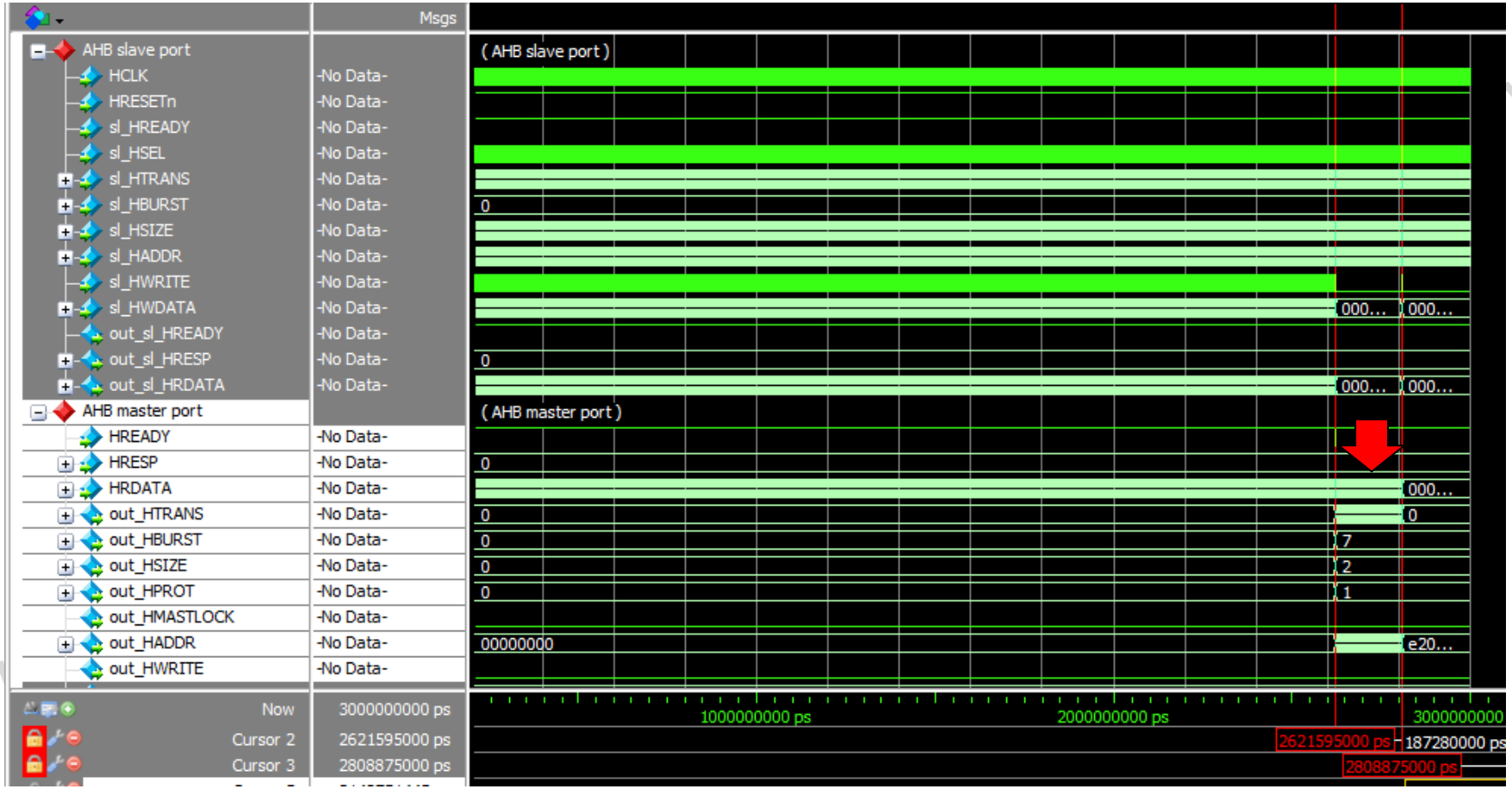
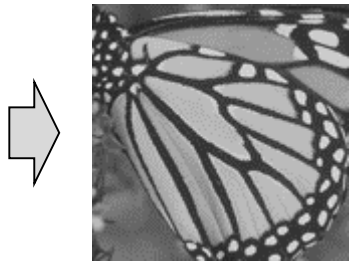


Image result by CPU

- The pixel data by CPU are stored in out/input_image_by_dma.bmp
  - It takes about 0.2 milliseconds.

```
//------------------------------------------------
// Image Writer
//------------------------------------------------
// synopsys translate_off
wire write_image_by_cpu_done;
wire write_image_by_dma_done;
bmp_image_writer#(.WIDTH(WIDTH),.HEIGHT(HEIGHT),.OUTFILE("out/input_image_by_cpu.bmp"))
u_bmp_image_writer_00(
./*input             */clk(clk),
./*input             */rstn(rstn),
./*input [WI-1:0]    */din(sl_HWDATA[7:0]),
./*input             */vld(q_ld_sl_reg && (q_sel_sl_reg == CNN_ACCEL_INPUT_IMAGE)),
./*output reg        */frame_done(write_image_by_cpu_done)
);

bmp_image_writer#(.WIDTH(WIDTH),.HEIGHT(HEIGHT),.OUTFILE("out/input_image_by_dma.bmp"))
u_bmp_image_writer_01(
./*input             */clk(clk),
./*input             */rstn(rstn),
./*input [WI-1:0]    */din(data_o_ld[7:0]),
./*input             */vld(data_vld_o_ld),
./*output reg        */frame_done(write_image_by_dma_done)
);
// synopsys translate_on
endmodule
```



Verification

- Compare the H/W simulation results (check_hardware_results.m)
  - Load images at the folders out/
  - Calculate the difference between two images.

```
4
5       %% Output from the H/W simulation (by CPU)
6 -     im_hw_cpu = imread('out/input_image_by_cpu.bmp');
7 -     im_hw_cpu = im_hw_cpu(:,:,1);    % Gray image
8
9       %% Output from the H/W simulation (by DMA)
10 -    im_hw_dma = imread('out/input_image_by_dma.bmp');
11 -    im_hw_dma = im_hw_dma(:,:,1);    % Gray image
12
13      %% Calculate the difference between S/W and H/W outputs
14 -    img_diff = abs(single(im_hw_cpu) - single(im_hw_dma));
15 -    max_diff = max(img_diff(:));
16
17      %% Evaluation
18 -    if(max_diff == 0)
19 -        fprintf('Two images are SAME!\n');
20 -    else
21 -        fprintf('Two images are different!\n');
22 -        disp(max_diff);
23 -        figure(1)
24 -        imshow(uint8(img_diff));
25 -    end
26
```

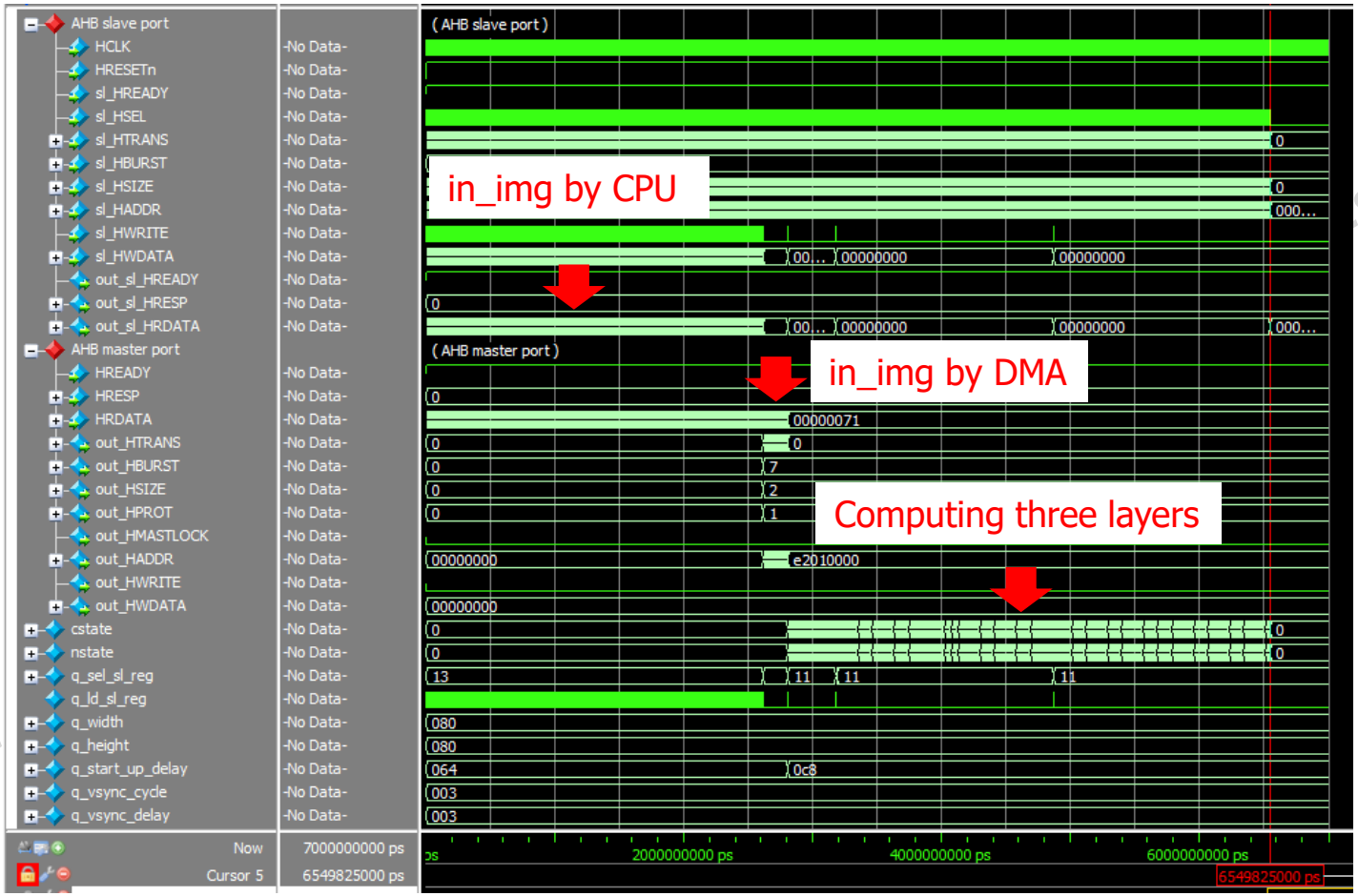
Command Window

New to MATLAB? See resources for Getting Started.

```
Two images are SAME!
>>
```

Waveform

- Do full simulation with time = 7ms



To do ...

- Completing the codes in cnn_accel.v
- Run the simulation to generate those waveform images.
  - Time = 2.7ms (CPU)
  - Time = 3ms (CPU, DMA)
  - Time = 7ms (CPU, DMA, CNN)