차세대 반도체 디지털 혁신공유대학 사업단

POLARIS

Lecture 14: CNN accelerator IP Optimization

Xuan-Truong Nguyen

Road map

Sub-pixel layer

Review
System Integration

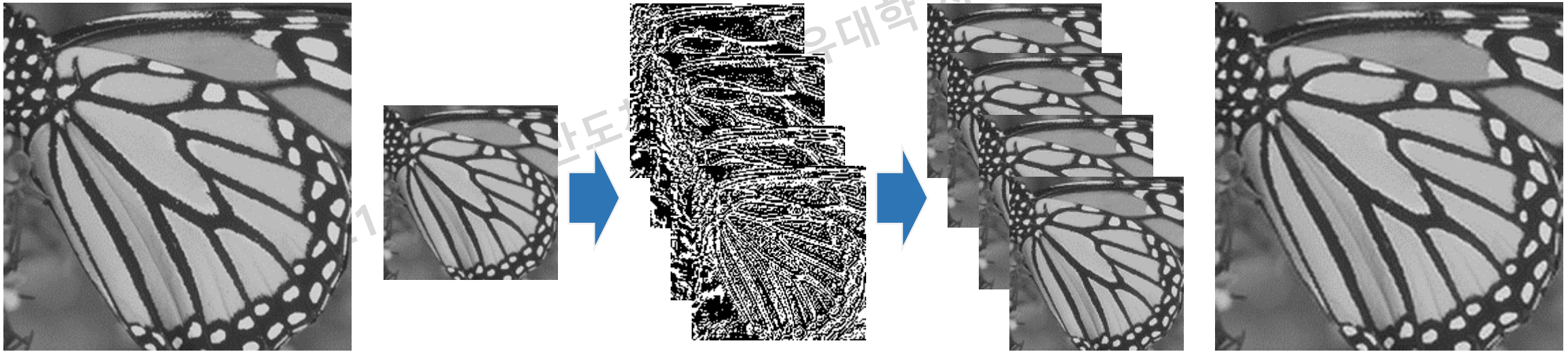
Optimization

Execution time
reduction

Buffer reduction

Review CNN-based Super-Resolution

- Generate a low-resolution image from a high-resolution image x
- Sub-pixel layer: CNN generates four residual images r_1, r_2, r_3, r_4
- Compute four reconstructed images $rec_1, rec_2, rec_3, rec_4$, where $rec_i = x + r_i, for\ i = 1, 2, 3, 4.$
- Flatten $rec_1, rec_2, rec_3, rec_4$ to obtain the high-resolution image



Ground-truth | Low-Resolution | Four residuals | Four recon. images | High-Resolution

Compute four reconstructed images

- Inputs: A low-resolution image $x \in Z^{H \times W}$, and four residual images $r_1, r_2, r_3, r_4 \in Z^{H \times W}$
- Output: four reconstructed images $rec_1, rec_2, rec_3, rec_4 \in Z^{H \times W}$
- Pseudocode:

For $(i = 0;\ i < H * W;\ i = i + 1)$

$px = \{1'b0, x[i]\}$ // Assume x[i] has eight bits.

$pr_k = r_k[i], for\ k = 1,2,3,4$ // $\{r_3[i], r_2[i], r_1[i], r_0[i]\}$ is a 32-bit output from Layer 3.

$$rec_k[i] = \begin{cases} 0 & if\ (px + pr_k) < 0 \\ 255 & if\ (px + pr_k) > 255 \\ px + pr_k \end{cases}$$ // Clipping or clamping.

End for

Compute the high-resolution image

- Inputs: four reconstructed images $rec_1, rec_2, rec_3, rec_4 \in Z^{H \times W}$
- Output: High-resolution image $y \in Z^{(2*H) \times (2*W)}$
- Pseudocode:

For $(row = 0; row < H; row = row + 1)$

For $(col = 0; col < W; col = col + 1)$

$$y[(2 * row\ \ ) \times (2 * W) + (2 * col\ \ )] = rec_1[row \times W + col]$$

$$y[(2 * row\ \ ) \times (2 * W) + (2 * col + 1)] = rec_2[row \times W + col]$$

$$y[(2 * row + 1) \times (2 * W) + (2 * col\ \ )] = rec_3[row \times W + col]$$

$$y[(2 * row + 1) \times (2 * W) + (2 * col + 1)] = rec_3[row \times W + col]$$

End for

End for

Compute the high-resolution image

- Inputs: four reconstructed images $rec_1, rec_2, rec_3, rec_4 \in Z^{H \times W}$
- Output: High-resolution image $y \in Z^{(2*H) \times (2*W)}$
- Pseudocode:

For $(row = 0; row < H; row = row + 1)$

For $(col = 0; col < W; col = col + 1)$

$$y[(2 * row\ \ ) \times (2 * W) + (2 * col\ \ )] = rec_1[row \times W + col]$$

$$y[(2 * row\ \ ) \times (2 * W) + (2 * col + 1)] = rec_2[row \times W + col]$$

$$y[(2 * row + 1) \times (2 * W) + (2 * col\ \ )] = rec_3[row \times W + col]$$

$$y[(2 * row + 1) \times (2 * W) + (2 * col + 1)] = rec_3[row \times W + col]$$

End for

End for

TODO: BMP image writer (bmp_image_writer_2x.v)

- Ports:
  - Inputs: din, vld
  - Output: frame_done
- Incoming pixels are stored in a buffer
  - out_img [0 : FRAME_SIZE-1]).
  - The internal counters (row, col) are updated

```
module bmp_image_writer_2x
#(parameter WI = 32,
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
);

// Image parameters
localparam FRAME_SIZE = (2*WIDTH)*(2*HEIGHT);
localparam FRAME_SIZE_W = $clog2(FRAME_SIZE);
reg [WI-1:0] out_img[0:FRAME_SIZE-1];   // Output feature map
reg [FRAME_SIZE_W-1:0] pixel_count;
```

To do ...

```
always@(posedge clk, negedge rstn) begin
    if(!rstn) begin
        for(k=0;k<FRAME_SIZE;k=k+1) begin
            out_img[k] <= 0;
        end
        row <= 0;
        col <= 0;
        pixel_count     <= 0;
        frame_done      <= 1'b0;
    end
    else begin
        if(vld) begin
            if(col == WIDTH-1) begin    // End of line
                col <= 0;
                if(row == HEIGHT-1)     // End of frame
                    row <= 0;
                else
                    row <= row + 1;     // Line Increment
            end
            else
                col <= col + 1;         // Pixel index increment

            if(pixel_count == FRAME_SIZE-4) begin   // End of frame
                pixel_count <= 0;
                frame_done <= 1'b1;
            end
            else begin
                pixel_count <= pixel_count + 4;     // Increase by 4
            end
            // Insert your code
            //{{{
            //out_img[/*Your code*/] <= din[/*Your code*/];
            //out_img[/*Your code*/] <= din[/*Your code*/];
            //out_img[/*Your code*/] <= din[/*Your code*/];
            //out_img[/*Your code*/] <= din[/*Your code*/];
            //}}}
        end
    end
end
```

BMP image writer (bmp_image_writer_new.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);
```

TODO: BMP image writer (bmp_image_writer_2x.v)

- When all pixels are stored in a buffer,
  - frame_done == 1
  - Write a 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 < 2*HEIGHT; h = h + 1) begin
            for(w = 0; w < 2*WIDTH; w = w + 1) begin
                $fwrite(fd, "%c", out_img[(2*HEIGHT-1-h)*(2*WIDTH) + w][7:0]);
                $fwrite(fd, "%c", out_img[(2*HEIGHT-1-h)*(2*WIDTH) + w][7:0]);
                $fwrite(fd, "%c", out_img[(2*HEIGHT-1-h)*(2*WIDTH) + w][7:0]);
            end
        end
        $fclose(fd);

    end
end
```

Open a file

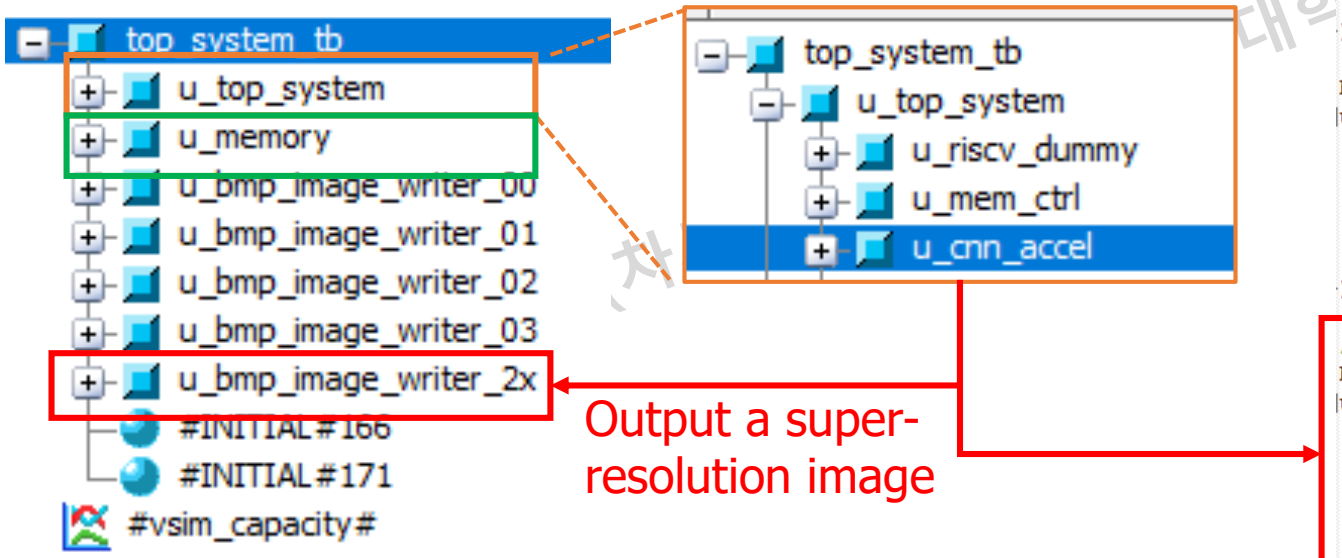
Write Header

Write data

```
//------------------------------------------------
// Save the output image
//------------------------------------------------
initial begin
    IW = 2*WIDTH;
    IH = 2*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);
```

Test bench (top_system_tb.v)

- Test the system
  - The CNN accelerator loads an image from memory and up-scales the low-resolution image
  - The high-resolution output image is displayed by the bmp writer.




```
bmp_image_writer#(.WIDTH(WIDTH),.HEIGHT(HEIGHT),.OUTFILE(OUTFILE00))
u_bmp_image_writer_00(
./*input              */clk        (HCLK                        ),
./*input              */rstn       (HRESETn                     ),
./*input [WI-1:0]     */din        (out_pixel[7:0]              ),
./*input              */vld        (out_valid & 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        (HCLK                        ),
./*input              */rstn       (HRESETn                     ),
./*input [WI-1:0]     */din        (out_pixel[15:8]             ),
./*input              */vld        (out_valid & 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        (HCLK                        ),
./*input              */rstn       (HRESETn                     ),
./*input [WI-1:0]     */din        (out_pixel[23:16]            ),
./*input              */vld        (out_valid & 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        (HCLK                        ),
./*input              */rstn       (HRESETn                     ),
./*input [WI-1:0]     */din        (out_pixel[31:24]            ),
./*input              */vld        (out_valid & q_is_last_layer),
./*output reg         */frame_done(frame_done[3]               )
);

// Super-Resolution Output
bmp_image_writer_2x #(.WIDTH(WIDTH),.HEIGHT(HEIGHT),.OUTFILE(OUTFILE2X))
u_bmp_image_writer_2x(
./*input              */clk        (HCLK                        ),
./*input              */rstn       (HRESETn                     ),
./*input [WI-1:0]     */din        (out_pixel                   ),
./*input              */vld        (out_valid & q_is_last_layer),
./*output reg         */frame_done(/* OPEN*/                    )
);
```

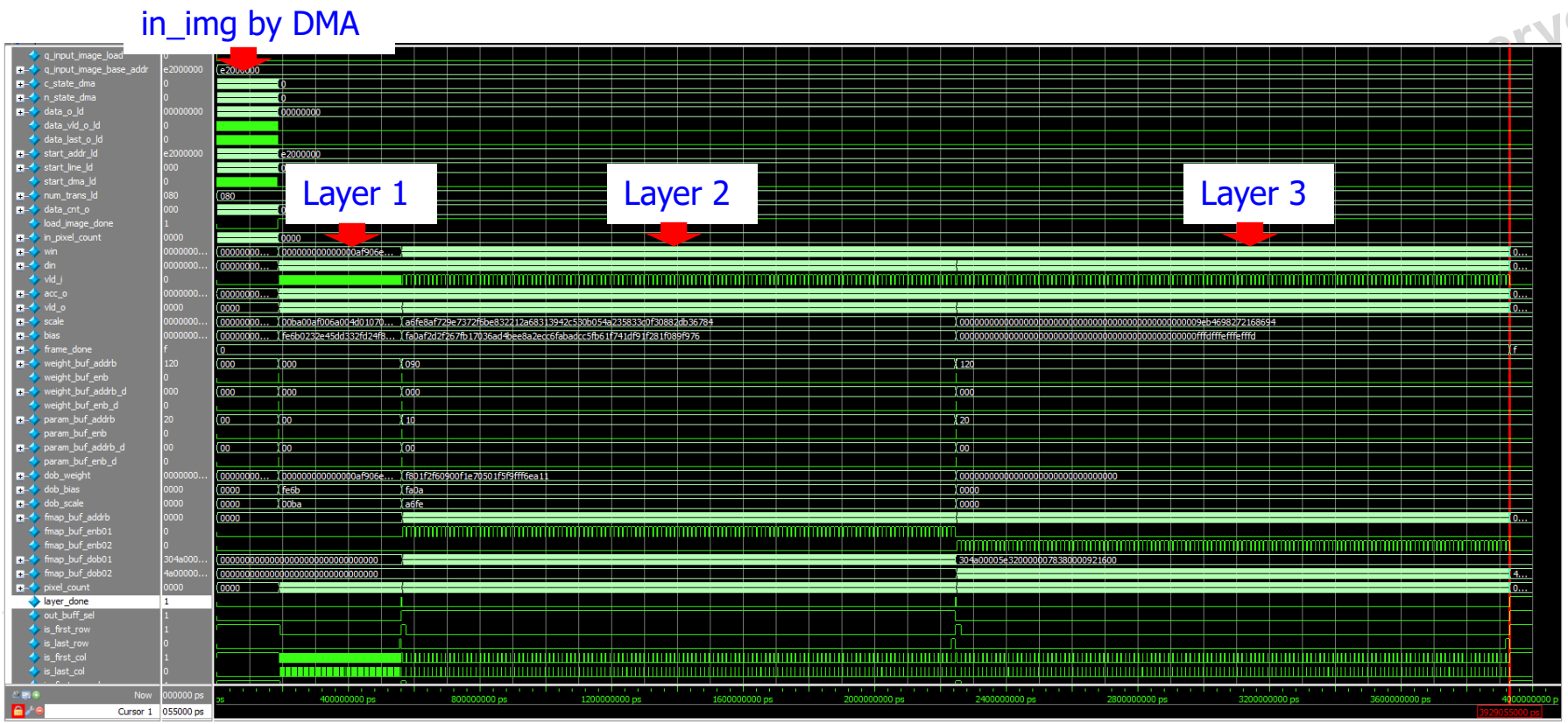
To do ...

- Modify the code (cnn_accel.v, bmp_image_writer_2x.v) to implement a sub-pixel layer
- Do a simulation with time = 4,000 microseconds
- Show the final high-resolution image.

How to **speed up** the simulation?

- Motivation: *Simulating a multiple-network is slow*

⇒ ***Can we speed up the simulation when optimizing Layer 3 only?***



Simulate a single layer (1/3): cnn_accel.v

- Example: Test Layer 3 only

⇒ Assume that Layer 2 "already stored" feature maps into the dual buffers

⇒ ***Initialize the dual buffer with convout_L2.hex***

```
//*************************************************************************************************
// Configure a directory for your OWN targeting layer
//*************************************************************************************************
dpram #(.FILENAME("out_sw/convout_L2.hex"),.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 #(.FILENAME("out_sw/convout_L2.hex"),.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                 )
);
```

Simulate a single layer (2/3): top_system_tb.v

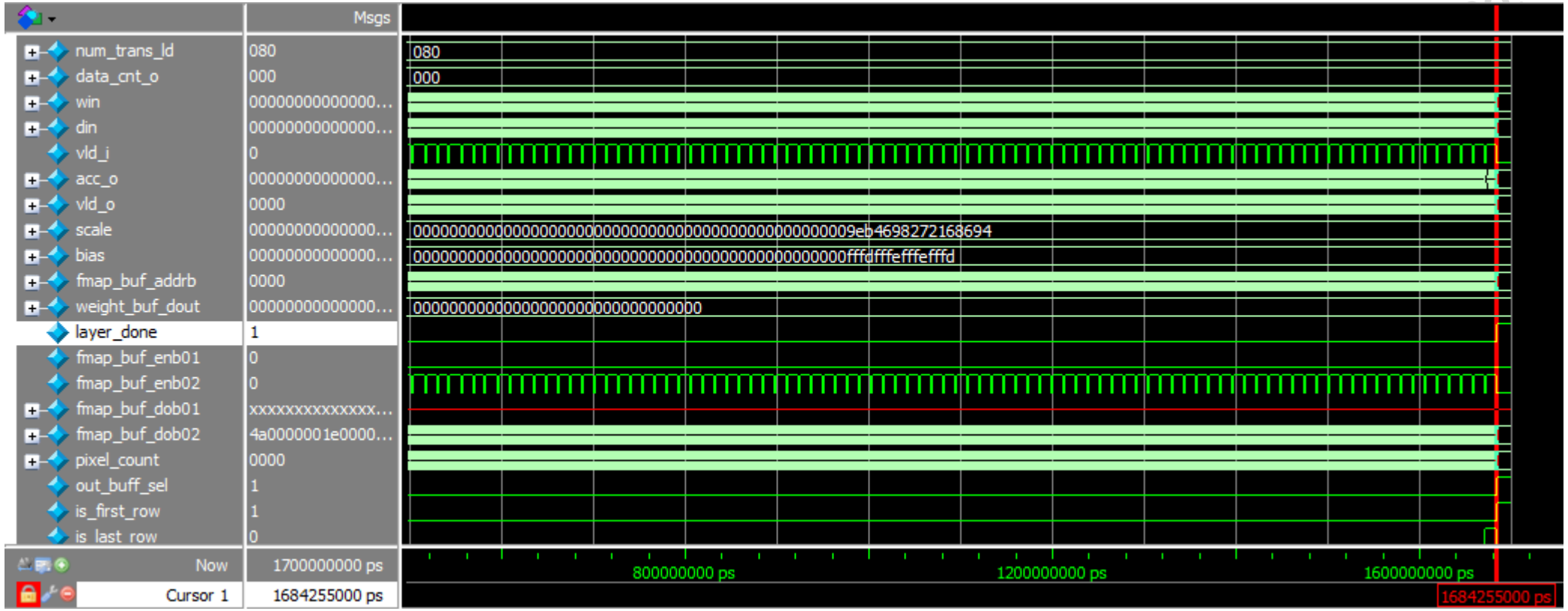
- Example: Test Layer 3 only

1. Set parameters for Layer 3
2. CPU configures the CNN accelerator to execute Layer 3
3. Polling: wait until the CNN engine completes its work (q_layer_done == 1)

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

//*****************************************************************************************************************
// Configure parameters for your OWN targeting layer
//*****************************************************************************************************************
//{{{
//idx = 0;  // Layer 1
idx = 2;    // LAYER 3
base_addr_weight = 160;
base_addr_param  = 32;
//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_top_system.u_riscv_dummy.task_AHBwrite(`CNN_ACCEL_BASE_ADDRESS, {base_addr_param&12'hFFF,base_addr_weight&20'hFFFFF});
    #(4*p) @(posedge HCLK)  u_top_system.u_riscv_dummy.task_AHBwrite(`CNN_ACCEL_LAYER_CONFIG, q_layer_config);
    // Start a frame
    #(4*p) @(posedge HCLK)  u_top_system.u_riscv_dummy.task_AHBwrite(`CNN_ACCEL_LAYER_START , 1'b1 );
    #(4*p) @(posedge HCLK)  u_top_system.u_riscv_dummy.task_AHBwrite(`CNN_ACCEL_LAYER_START , 1'b0 );

    // Polling
    while(!q_layer_done) begin
        #(128*p) @(posedge HCLK)  u_top_system.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
```

Simulate a single layer (2/3): top_system_tb.v

- Uncomment "TEST_ONE_LAYER_ONLY" in debug.v
- Do a simulation with time = 1.7 ms (instead of 4 ms)



Road map

Sub-pixel layer

Review
System integration

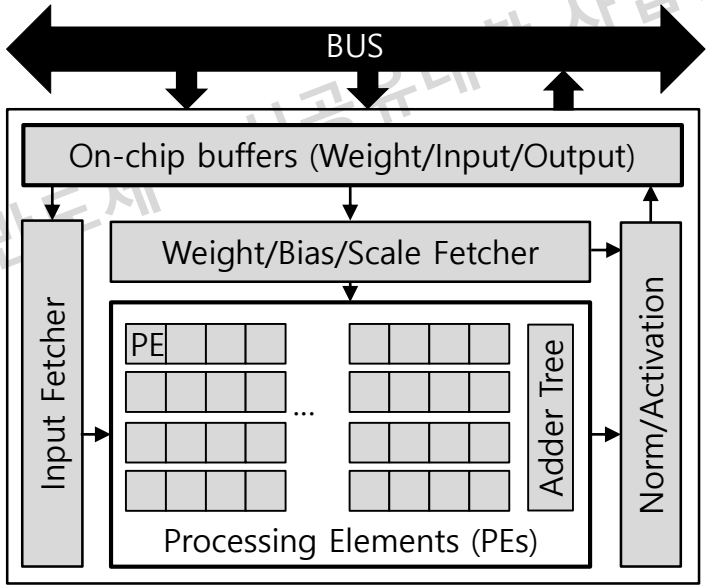
Optimization

Execution time
reduction

Buffer reduction

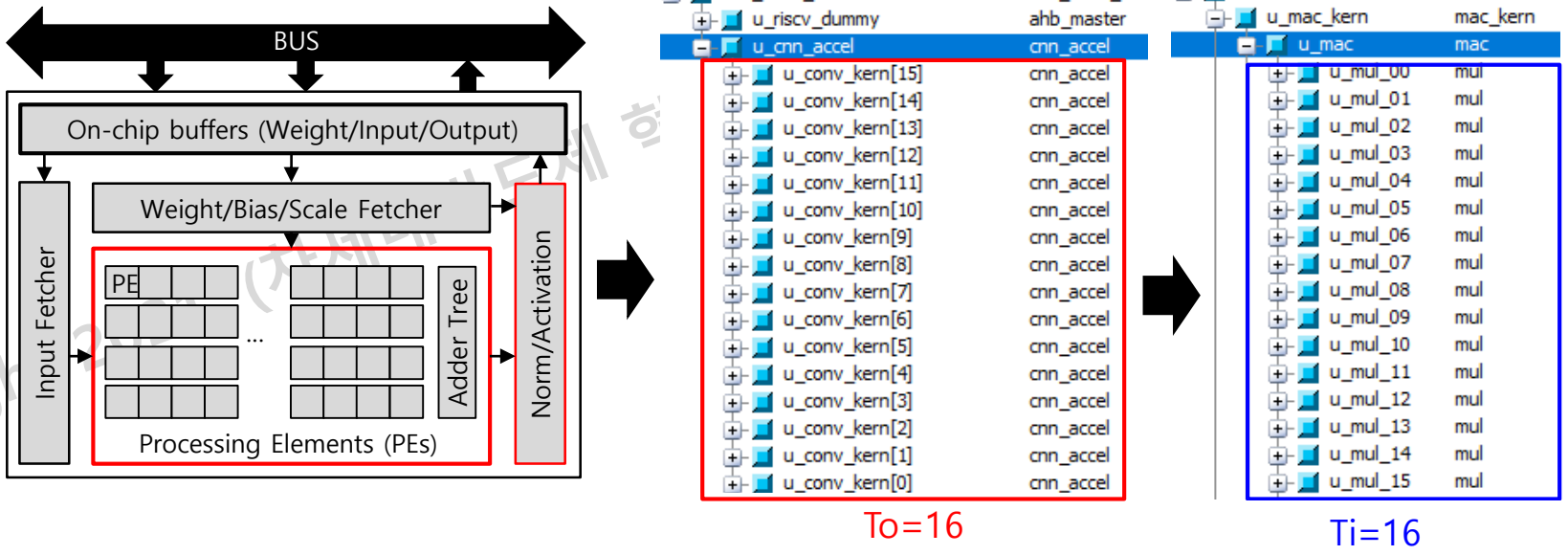
CNN accelerator

- 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



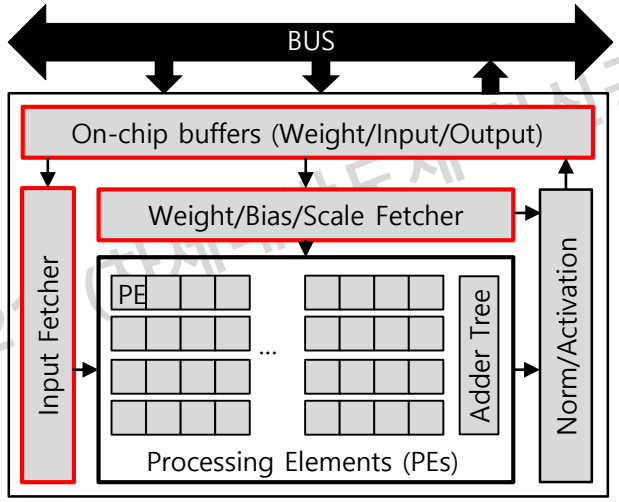
Processing Elements: ~ALU

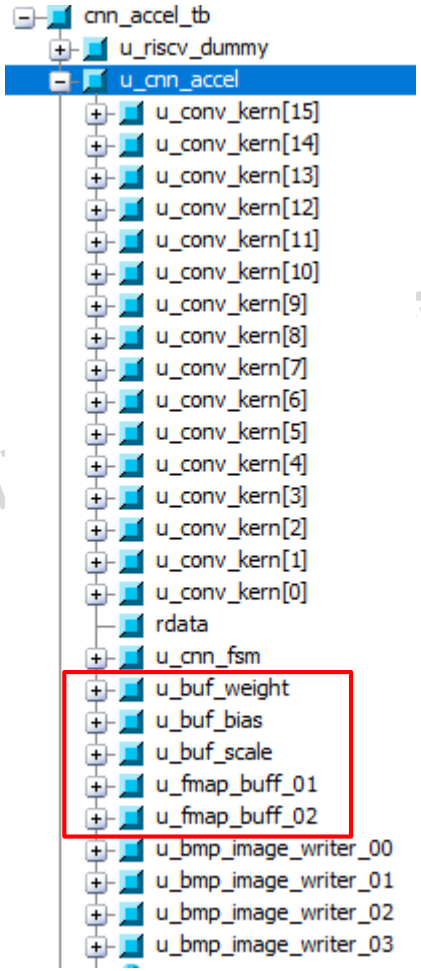
- Processing Element (PE) Array (conv_kern.v, mac_kern.v)
  - Perform convolution/activation/quantization operations.
  - To: The number of convolutional kernels (output feature maps)
  - Ti: The number of multipliers in a CONV kernel (conv_kern.v)
- The number of multipliers is : To× Ti
  - 256 (=16×16)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



Buffers: ~Register File

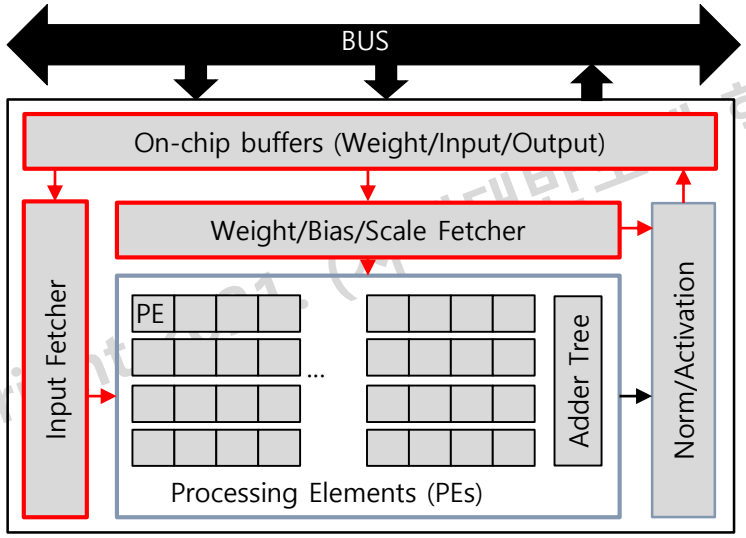
- Buffers:
  - Input buffer: in_img
  - Weight/bias/scale buffers
  - Dual (output) buffers of feature maps





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

- 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

- Some control signals and parameters
  - From the configuration registers or RISC-V CPU
  - An AHB Master configures 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

- Weight/bias/scale: We must access the global buffers which store weights, biases, and scales for all three layers to parameters for one layer

⇒ Use the base addresses to locate a layer's parameters in the global buffers.

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

- 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

- 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

Weight/Bias/Scale buffers

- Three **global** buffers (spram.v) for Weights, Biases, and scales.
  - u_weight_buffer, u_bias_buffer, u_scale_buffer
  - Store weights, biases, and scales of **all layers**, i.e. 3 layers.
  - Weight buffer
    - Each word stores Ti weights, i.e. Ti=16; a weight is stored in 8 bits

    ⇒ Each word has 128 bits
    - Number of words:
      - In the **worst case**, a layer has (3*3*To) weights, i.e., 144.
  - Bias/scale buffers
    - A bias word or a scale word is stored in 16 bits.
    - In the worst case, a layer has To biases and To scales.

Buffer	Buffer size (WI=8, PARAM_BITS=16, Ti = 16, To = 16)		
	Word (bit)	No. of words	Size (bit)
Weight	WI*Ti	3*(3*3*To)	3*3*3*To*WI*Ti
Bias	PARAM_BITS	3*To	3*To*PARAM_BITS
Scale	PARAM_BITS	3*To	3*To*PARAM_BITS

Weight/Bias/Scale buffers

- Three **local** buffers, i.e. registers
  - win_buf, scale and bias: Store weights, biases, and scales of **one layers**
- The base addresses are used to load weights, scales and biases of a layer from the **global** buffers to the local buffers or registers.

```
// 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
            if(ctrl_vsync_cnt < To*9) begin
                weight_buf_en   = 1'b1;
                weight_buf_
                weight_buf_
            end
        end
    end
end
```

Access a global buffer (u_buf_weight)

```
// 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)
);
```

Access the local buffer (win_buf)

```
// 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
        if(param_buf_en_d) begin
            bias [(param_buf_addr_d*PARAM_BITS)+:PARAM_BITS] <= param_buf_dout_bias;
            scale[(param_buf_addr_d*PARAM_BITS)+:PARAM_BITS] <= param_buf_dout_scale;
        end
    end
end
```

Weight/Bias/Scale buffers

- The minimum sizes of the weight/bias/scale buffers
  - Layer 1: 3x3x1 is reordered to 1x16 → Use To words instead of 3*3*To
  - Layer 3: There are only 4 output feature maps instead of 16.

Buffer	The size of buffers (Ti = 16, To = 16)			Number of words		
	Word (bit)	No. of words	Size (bit)	**Layer 1**	**Layer 2**	**Layer 3**
Weight	8*Ti	3*(3*3*To)	3*3*3*To*8*Ti	**To**	**3*3*To**	**3*3*To**
Bias	16	3*To	3*To*16	**To**	**To**	**no=4**
Scale	16	3*To	3*To*16	**To**	**To**	**no=4**

- The minimum numbers of **the buffer requests**
  - Corresponds to the numbers of words.
- The minimum number of cycles required for **weight/bias/scale preloading**
  - # of cycles for buffer requests
  - # of cycles for transferring data (delay) → one cycle
  - # of cycles for packing data → one cycle

⇒"win_buf", "bias" and "scale" are ready for use.

Input and feature map buffers

- Accessing the input buffer and the fmap buffers: Prepare "din"

Ports	In/Out	Definition	Source/Destination		
			Layer #1	Layer #2	Layer #3
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

- q_is_first_layer and out_buff_sel are used to define the source of din
  - For the first layer (q_is_first_layer == 1): "din" comes from in_img
  - Otherwise, it comes from either buffer 1 (out_buff_sel == 1) or buffer 2

```
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;
```

Input and feature map buffers

- Input buffer (in_img)
  - Each word has one 8-bit pixel
  - No. of words = WIDTH * HEIGHT

  ⇒ Buffer size = 128 Kilobits
- Dual buffers
  - Each word has To 8-bit pixels (=128 bits).
  - No. of words = WIDTH * HEIGHT

  ⇒ All feature maps are stored in the buffer

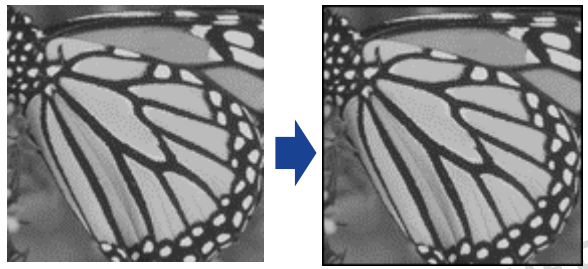
  ⇒ Buffer size = 2 Megabits (Mb) per each buffer

Buffer	Buffer size (Ti=To=16, WIDTH=HEIGHT=128)		
	Word (bit)	No. of words	Size (bit)
Input (in_img)	8	WIDTH*HEIGHT	WIDTH*HEIGHT*8*1 (=128 K)
fmap 1	8*To	WIDTH*HEIGHT	WIDTH*HEIGHT*8*To (= 2 M)
fmap 2	8*To	WIDTH*HEIGHT	WIDTH*HEIGHT*8*To (= 2 M)

Computation utilization: Layer 1

- Layer 1: num_ops=9 (=3×3×1) < 16
  - Layer 1: 3×3×1
  - din: 1×1×16

  ➔ din[9] = din[10] = …= din[15] = 0

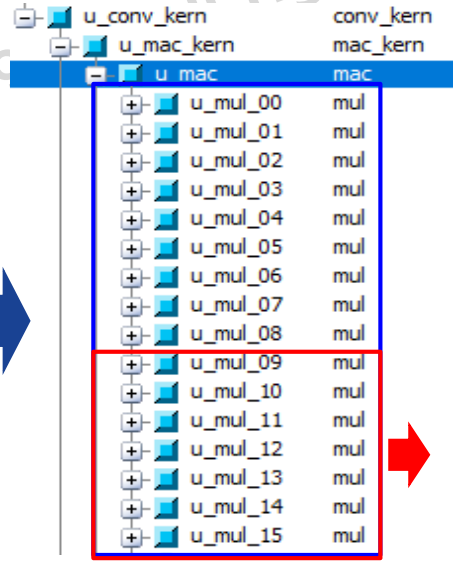


0	0	0	0	0	0	0	0
0	42	69	91	99	106	108	111
0	105	42	56	84	106	113	112
0	72	43	42	68	109	112	104

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

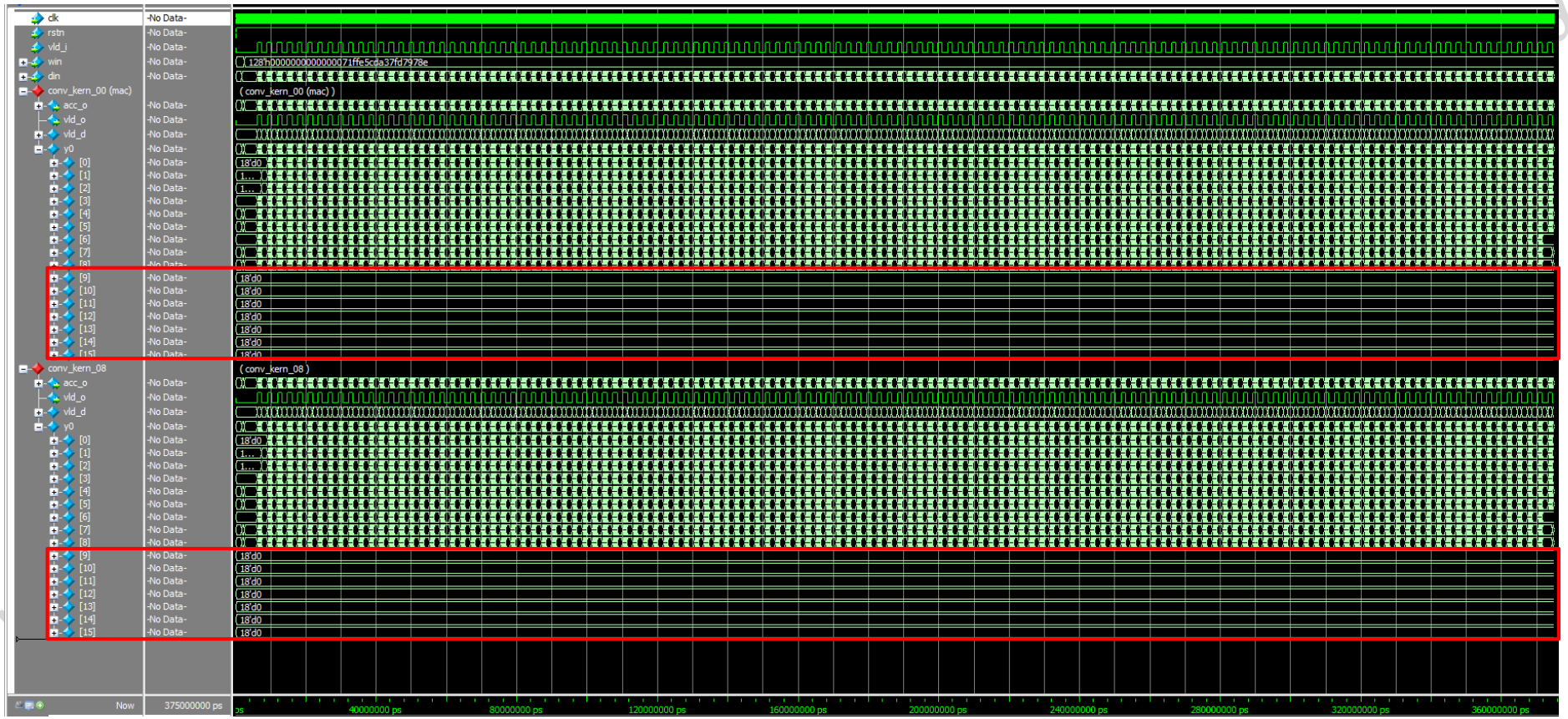


No impact on output

conv_kern: Ti=16

Computation utilization: Layer 1

- Layer 1: num_ops=9 (=3×3×1) < 16
  - din: 1×1×16 ➔ din[9] = din[10] = …= din[15] = 0

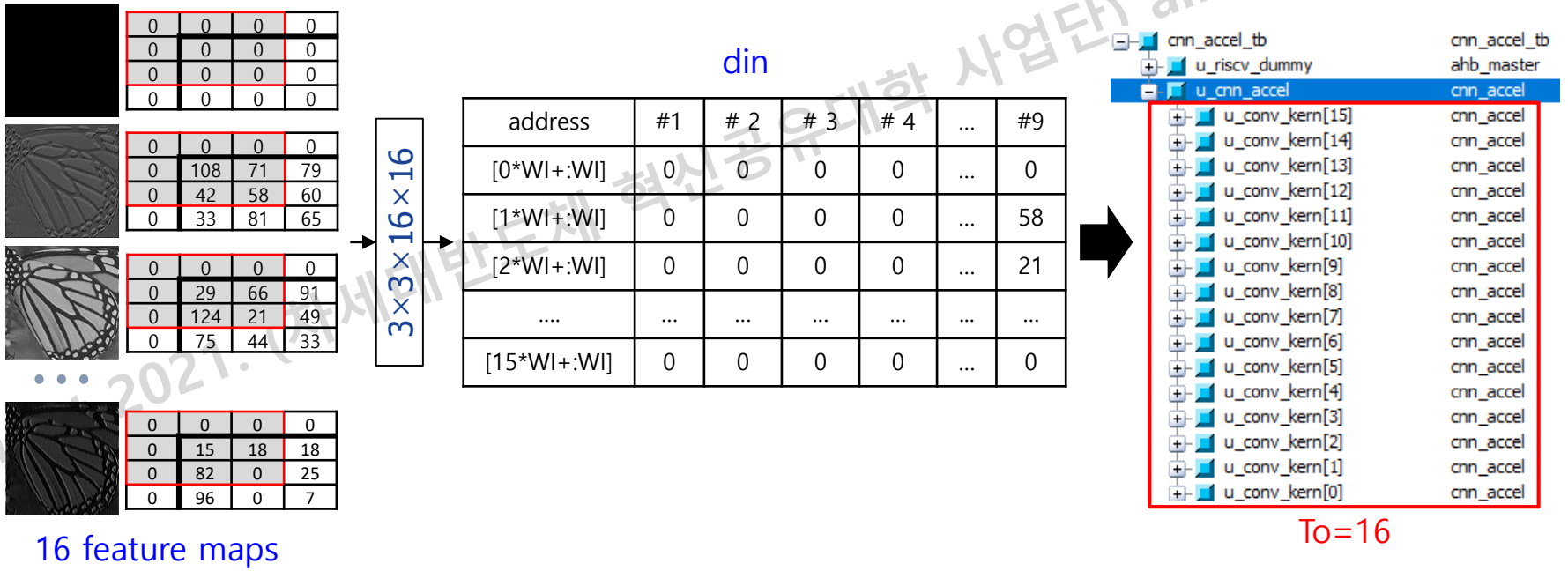


Computation utilization: Layer 2

- 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 output feature maps

  ⇒ All PEs are fully utilized

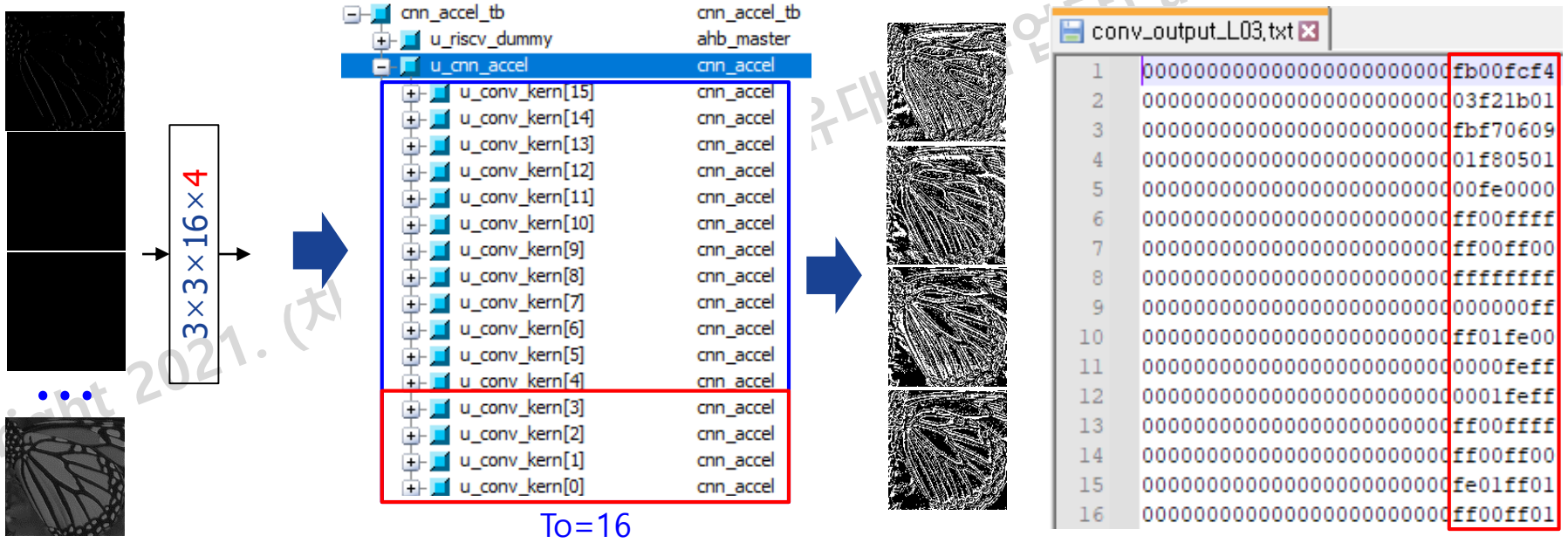


Computation utilization: Layer 3

- Layer 3: 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

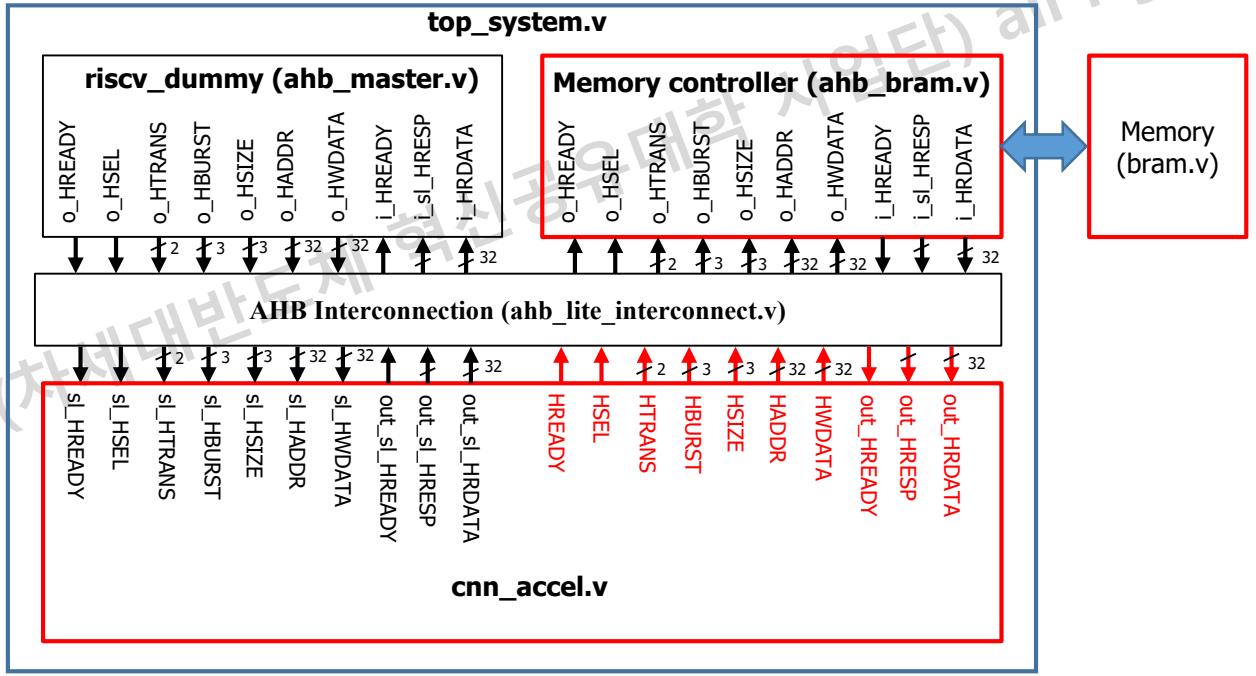
  ⇒ 4 output feature maps

  ⇒ Only 4 convolution kernels actually contributes to output generation.



Top system

- 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
- 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
```

How to send data from mem to in_img?

- Method 1: Using CPU
- 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!!!
#
```

How to send data from mem to in_img?

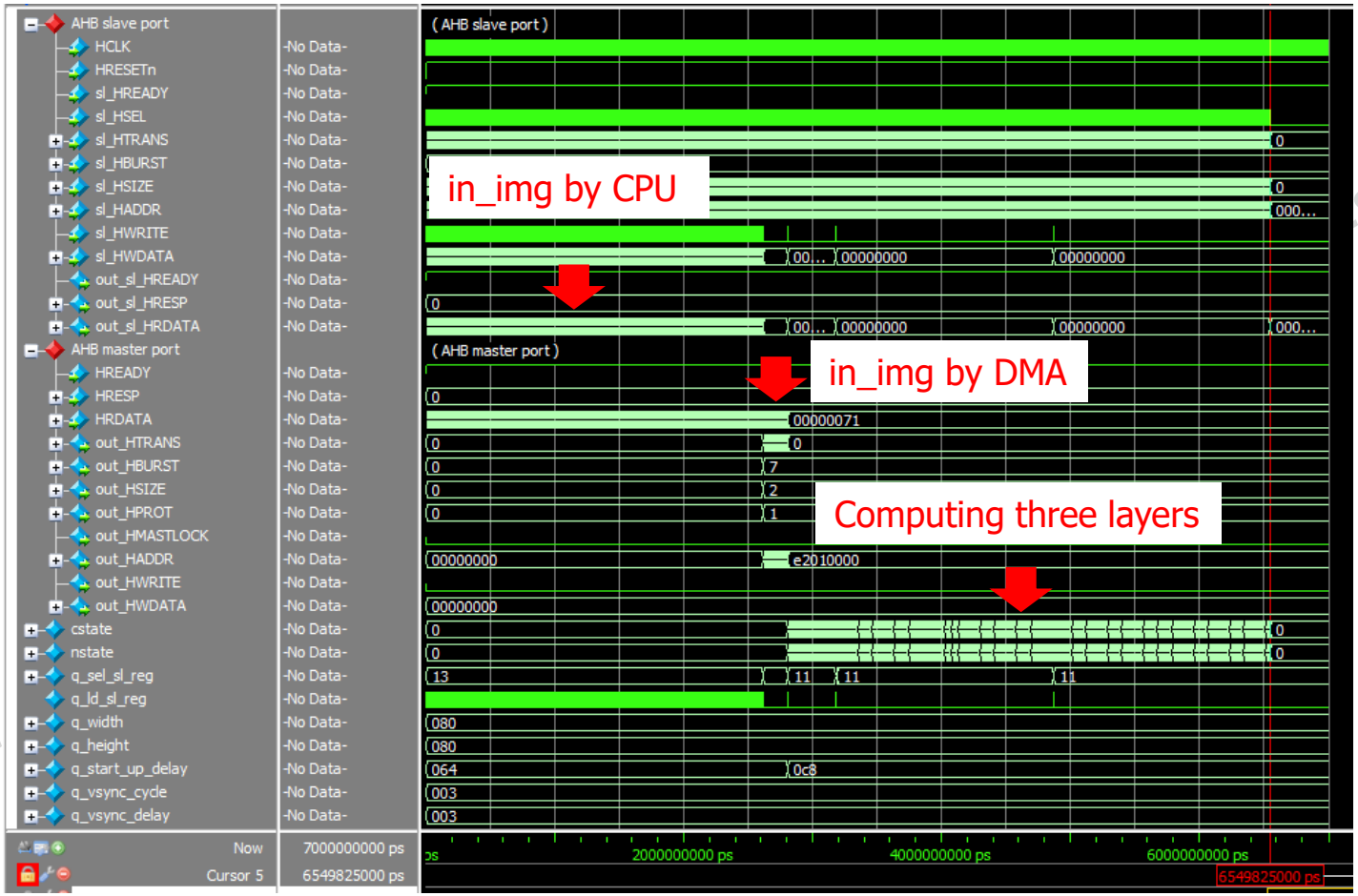
- Method 2: Using DMA
  - In the testbench (top_system_tb.v), allow CNN accelerator to directly access memory
    - Set a flag
    - Polling: wait until all pixel data are read by DMA

- 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);
```

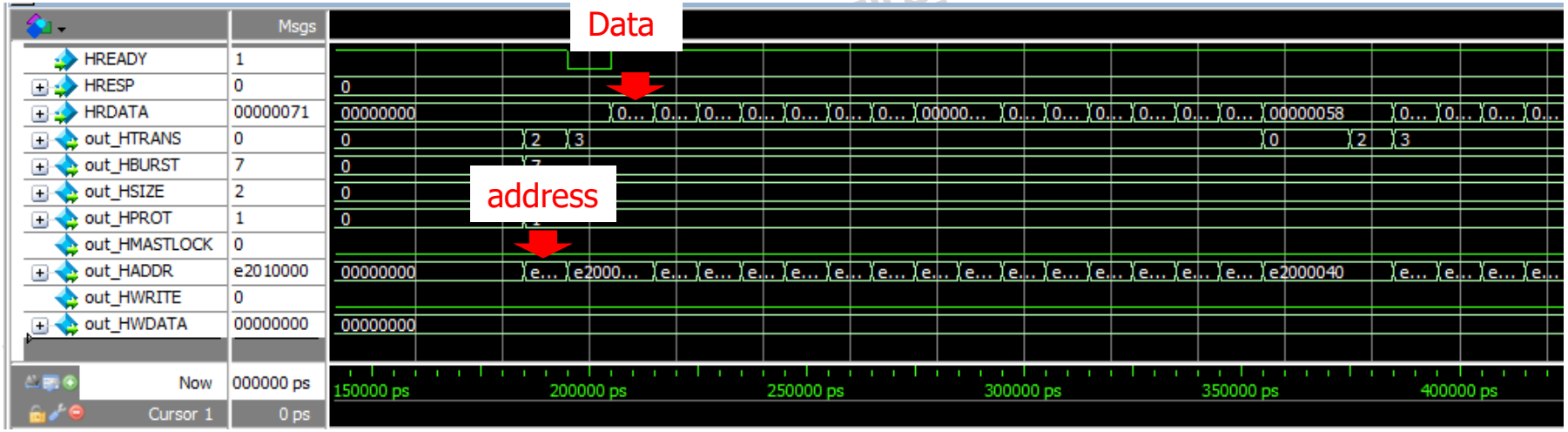
Waveform

- Do full simulation with time = 7ms



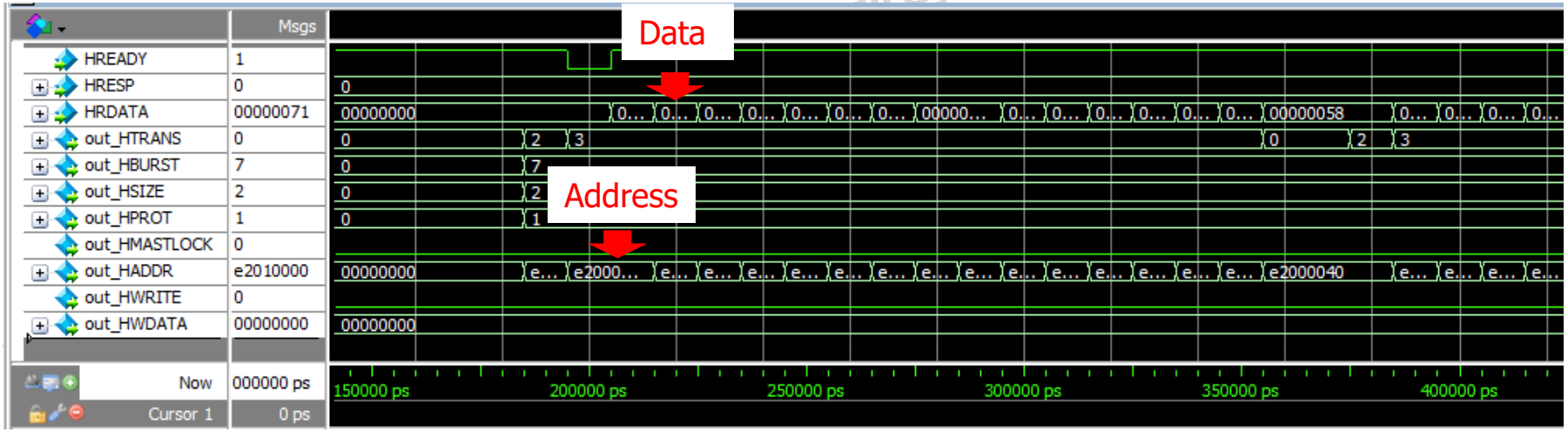
Why DMA is faster than CPU?

- The number of requests:
  - CPU: each pixel requires one LOAD and one WRITE
  - DMA: one LOAD
- Burst mode:
  - CPU: A request has address and data phases (SINGLE)
  - DMA: The address phase and the data phase are pipelined (16)



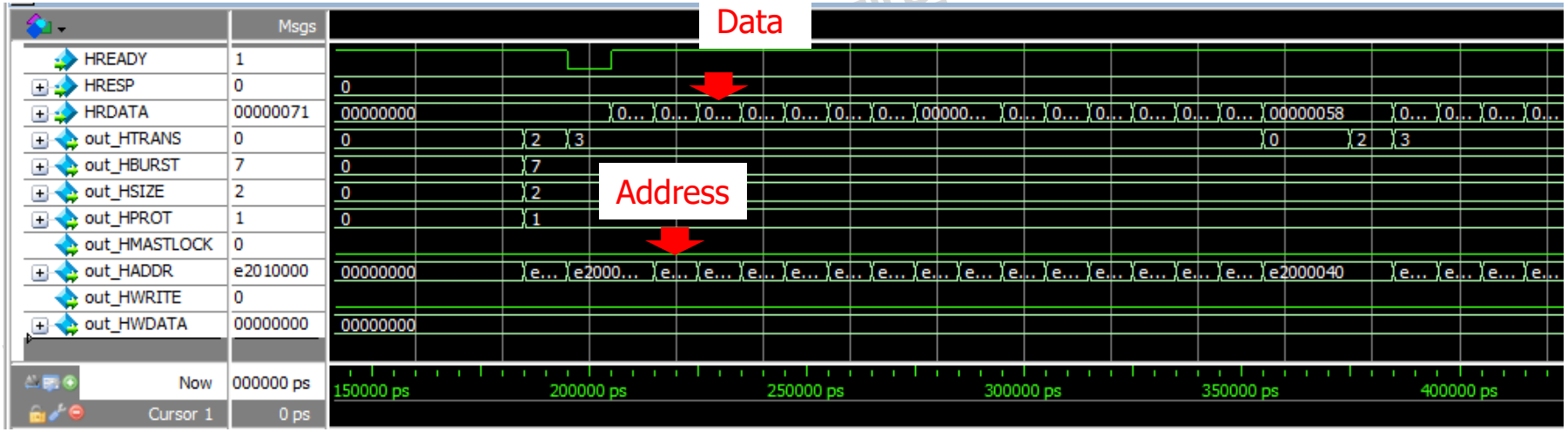
Why DMA is faster than CPU?

- The number of requests:
  - CPU: each pixel requires one LOAD and one WRITE
  - DMA: one LOAD
- Burst mode:
  - CPU: A request has address and data phases (SINGLE)
  - DMA: The address phase and the data phase are pipelined (16)



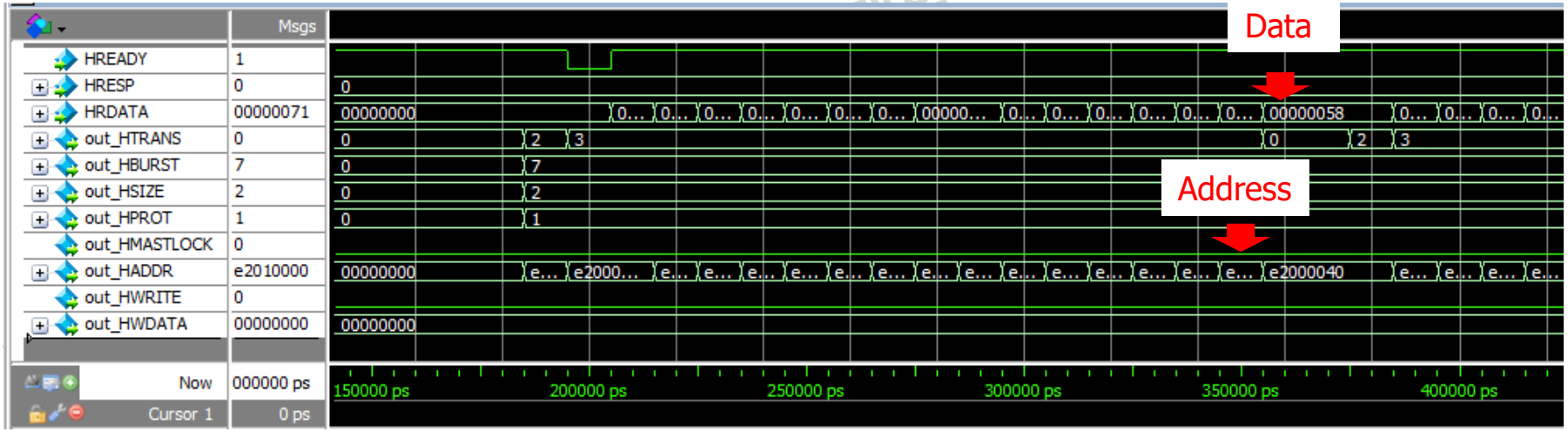
Why DMA is faster than CPU?

- The number of requests:
  - CPU: each pixel requires one LOAD and one WRITE
  - DMA: one LOAD
- Burst mode:
  - CPU: A request has address and data phases (SINGLE)
  - DMA: The address phase and the data phase are pipelined (16)



Why DMA is faster than CPU?

- The number of requests:
  - CPU: each pixel requires one LOAD and one WRITE
  - DMA: one LOAD
- Burst mode:
  - CPU: A request has address and data phases (SINGLE)
  - DMA: The address phase and the data phase are pipelined (16)



Road map

Sub-pixel Layer

Review
System Integration

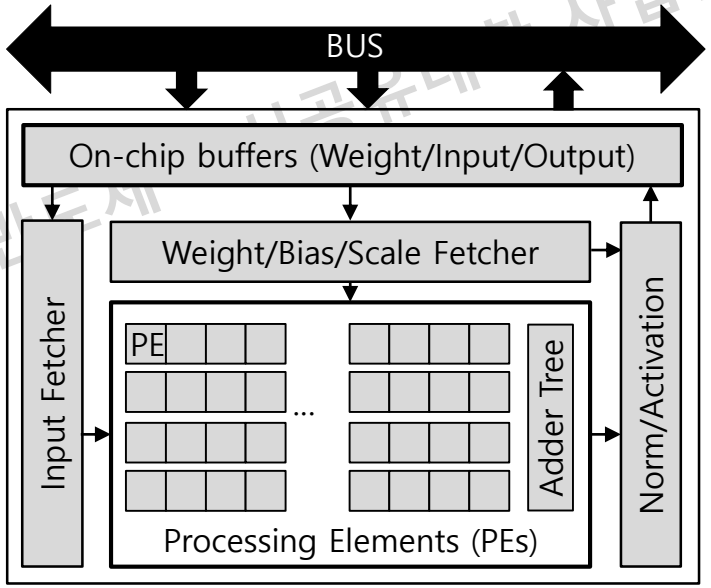
Optimization

Execution time
reduction

Buffer reduction

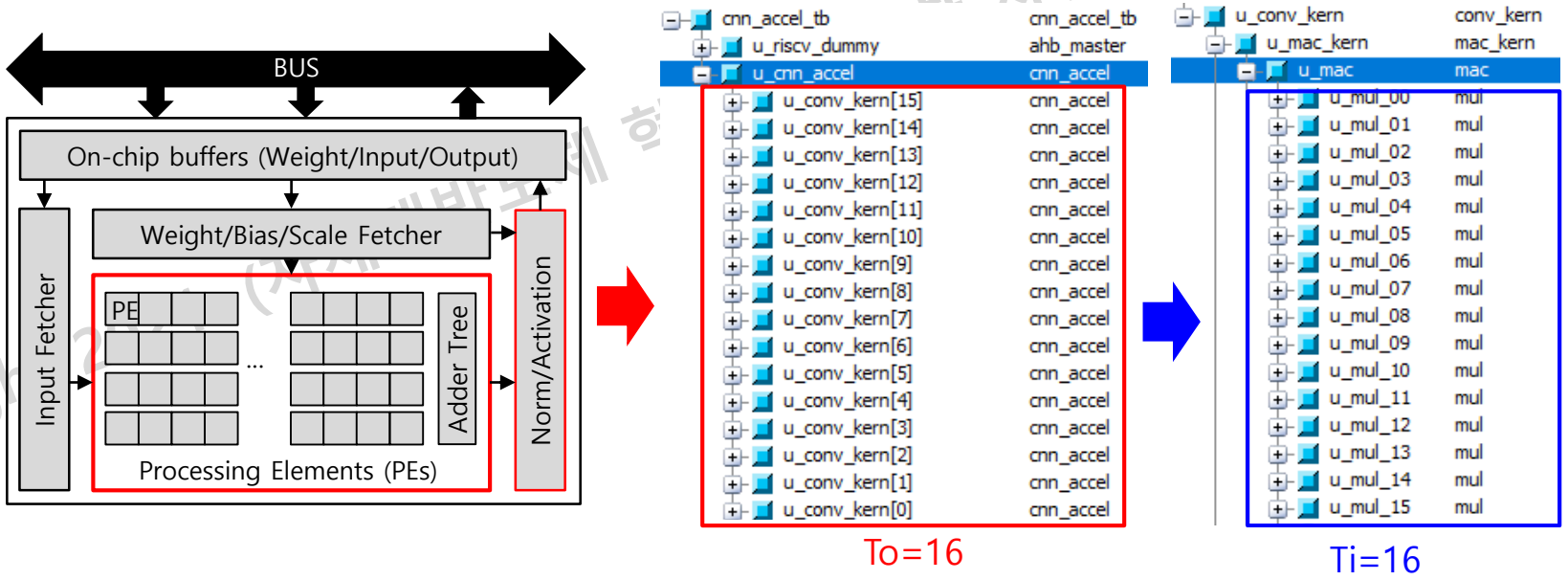
CNN accelerator

- 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

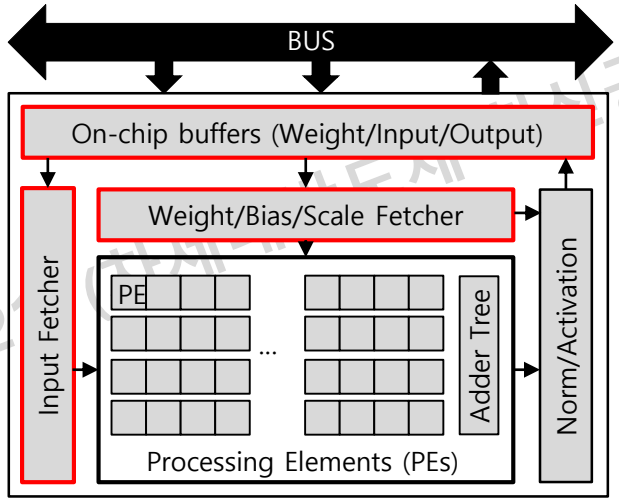
- Processing Element (PE) Array (conv_kern.v, mac_kern.v)
  - Perform convolution/activation/quantization operations.
  - To: The number of convolutional kernels (output feature maps)
  - Ti: The number of multipliers in a CONV kernel (conv_kern.v)
- The number of multipliers is : To× Ti
  - 256 (=16×16)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

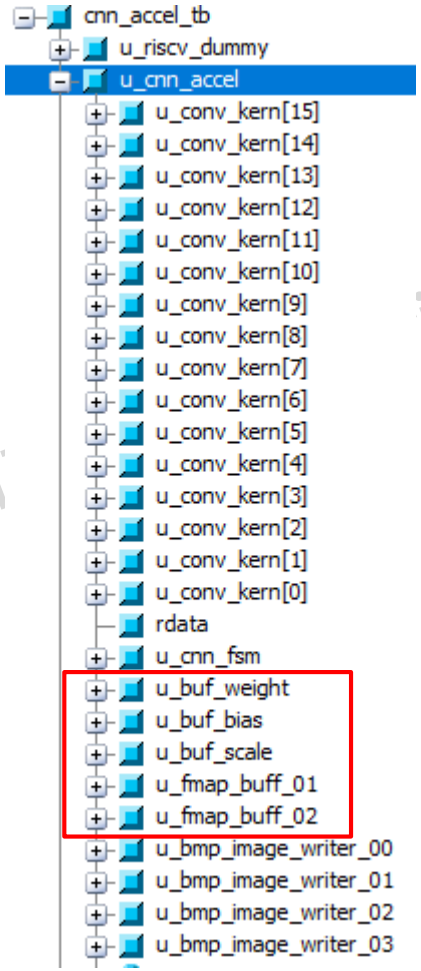


Buffers: ~Register File

- Buffers:
  - Input buffer: in_img (128 Kbits)
  - Weight/bias/scale buffers (55.5 Kbits)
  - Dual buffers of feature maps (4,096 Kbits)

  ⇒ The total buffer size is 4,279.5 Kbits.
- Can we do reduce the buffers?

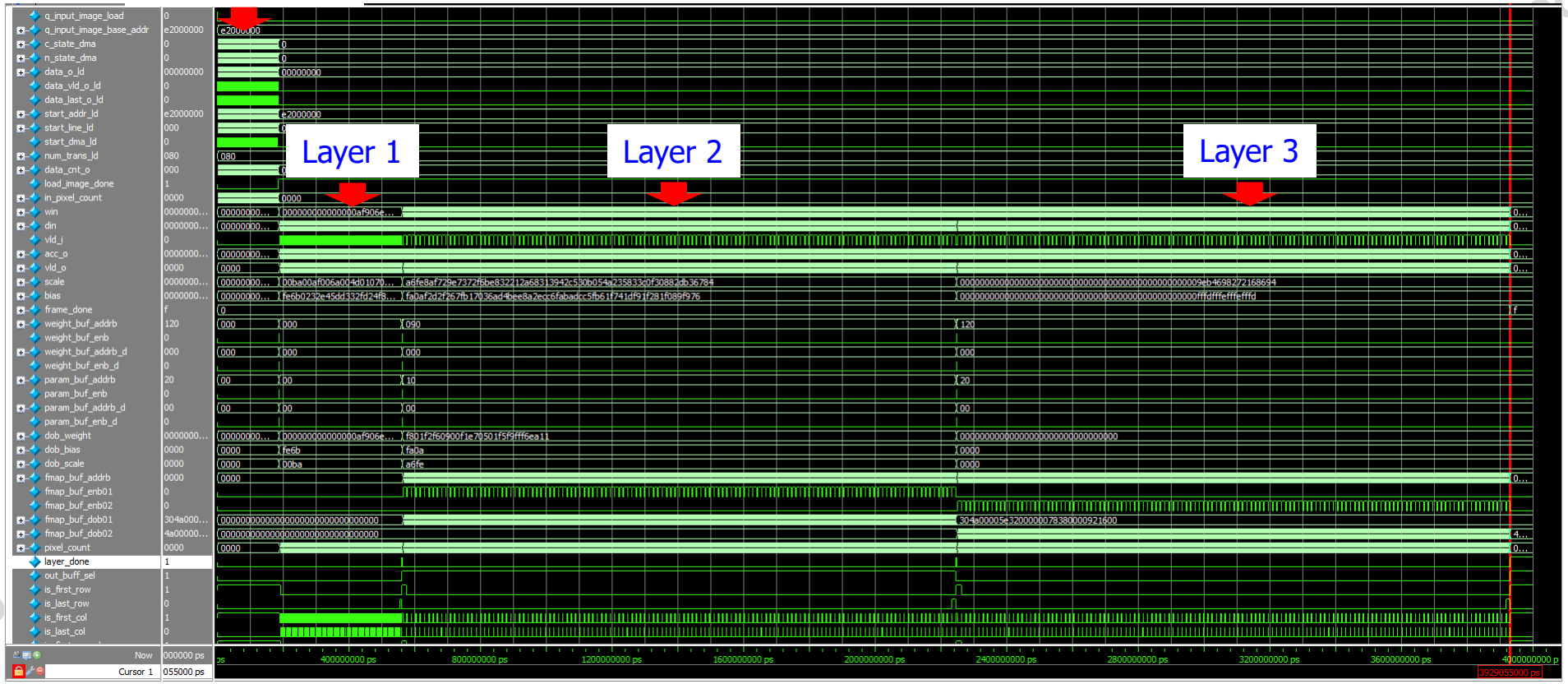




Waveform

- Do simulation with time = 4ms
- Can we do reduce the number of cycles?

in_img by DMA



Problem, constraints, and scopes

- CNN Accelerator Optimization problem
  - Reduce/minimize the number of cycles
  - Reduce/minimize the buffer size
  - Constraints
    - Ti = 16, To = 16.
    - WIDTH = 128, HEIGHT = 128.
    - 8-bit weights, 16-bit biases and 16-bit scales.
    - Fixed clock frequency 100MHz

- Scopes: What you can modify:
  - Hex files: reorder input or filters (weight, bias, scale).
  - Test bench: top_system_tb.v
  - Top module: cnn_accel.v

Road map

Sub-pixel Layer

Review
System Integration

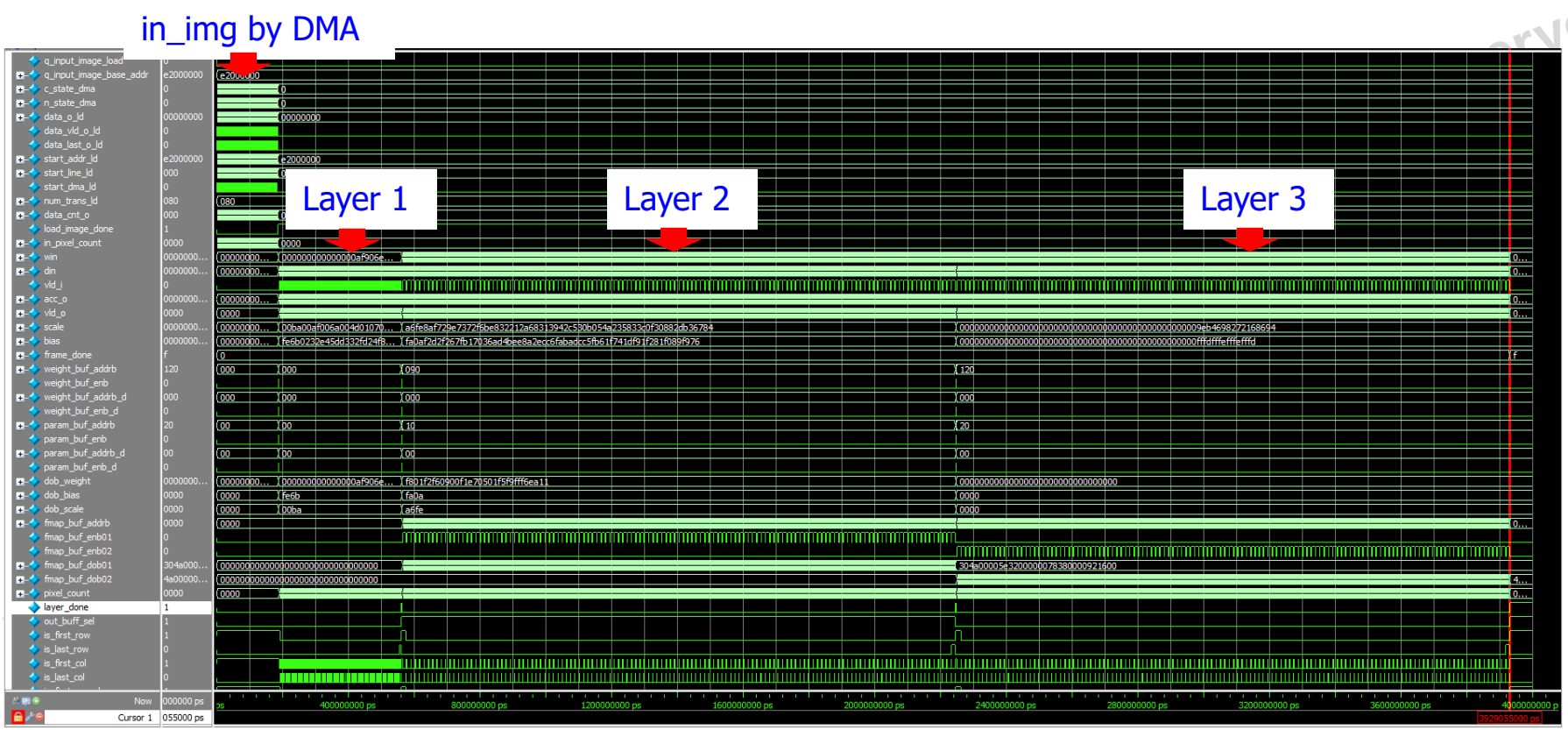
Optimization

Execution time
reduction

Buffer reduction

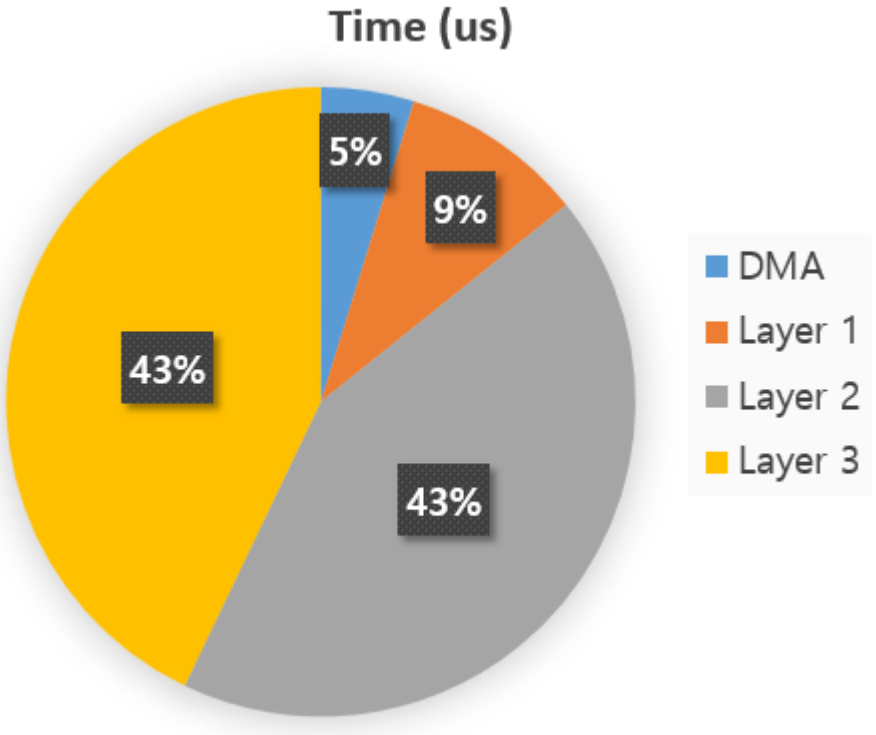
Running time profiling

- It takes approximately 4 ms (3,930us) to run the network.
  - DMA: 188us, Layer 1 (conv1x1): 372us, Layer 2, 3 (conv3x3): 1,685us



Target selection

- How to define a target for optimization?
  - Find a **“high-impact”** target
- Why?
  - **Amdahl’s law**



Amdahl's law

- Amdahl's law:

$$S_{latency}(s) = \frac{1}{(1-p) + \frac{p}{s}}$$

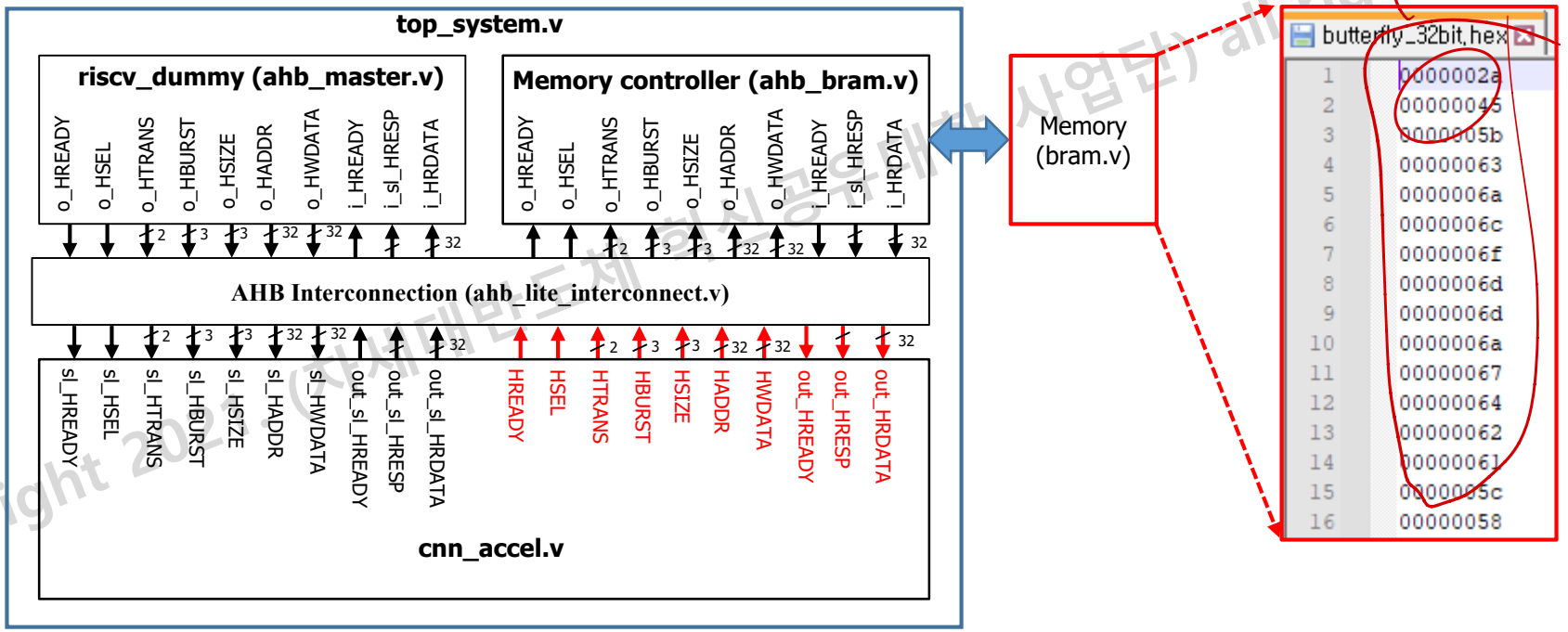
  - $S_{latency}$ is the theoretical speedup of the execution of the whole task
  - S is the speed up of the part of the task that benefits from improved system resources.
  - P the is proportion of execution time that the part benefiting from improved resources original occupied.

- Speedup estimation:
  - OPT1: speed up DMA by 4x
  - OPT2: speed up Layer 3 by 4x

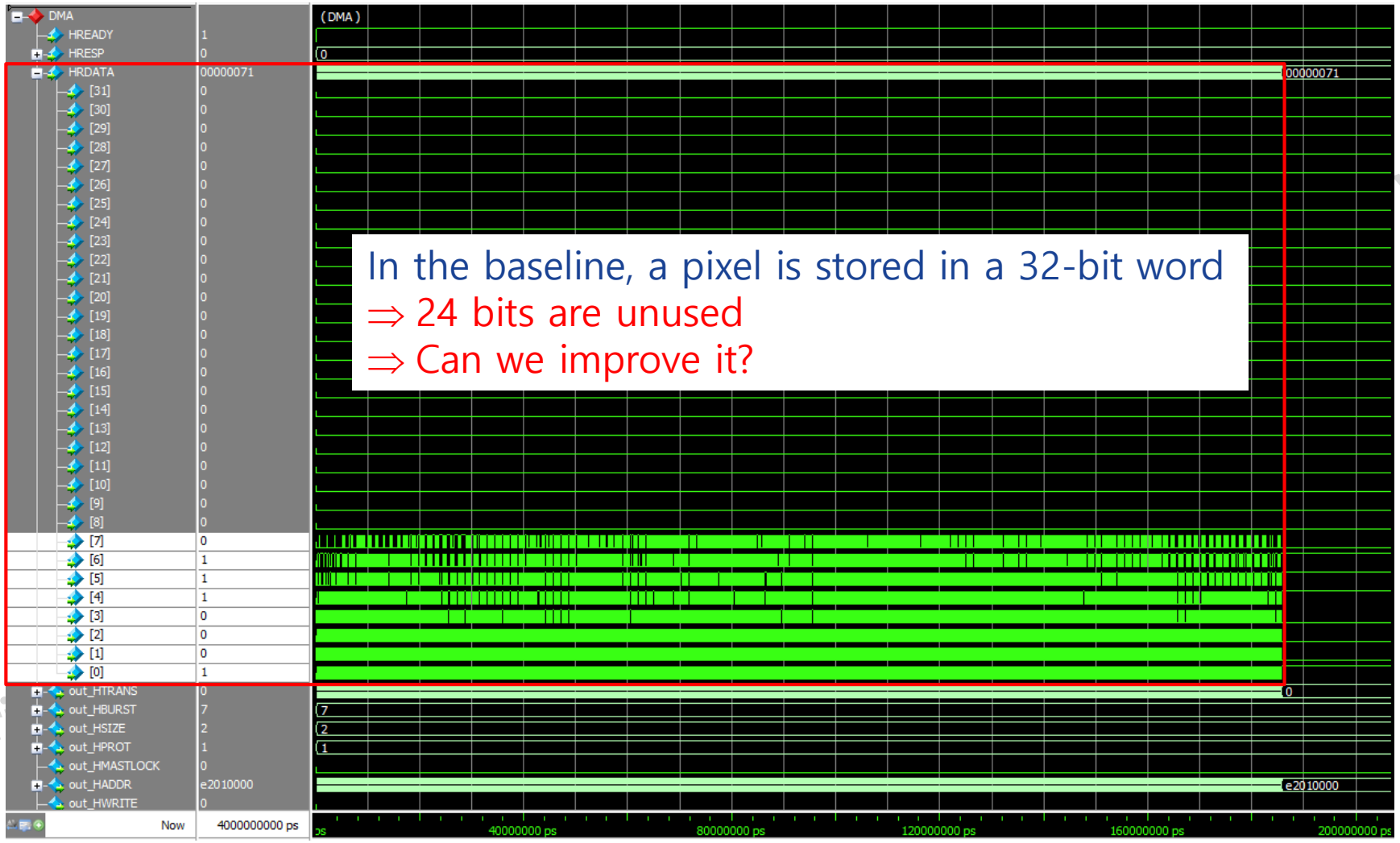
	Execution time (us)		
	Baseline	OPT1	OPT2
DMA	188 (5%)	47	188
Layer 1	372 (9%)	372	372
Layer 2	1,685 (43%)	1,685	1,685
Layer 3	1,685 (43%)	1,685	421
Total	3,930	3,789	2,666
Speedup	1.00	1.04	1.47

Top system

- 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
  - The input image (img/butterfly_32bit.hex) is stored in Memory

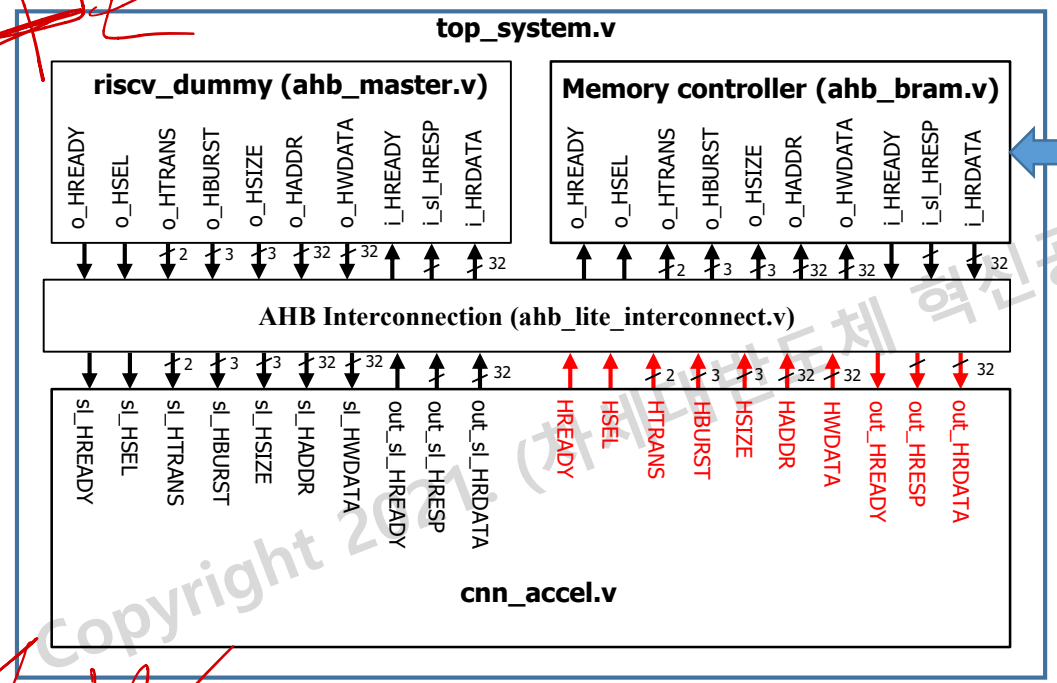


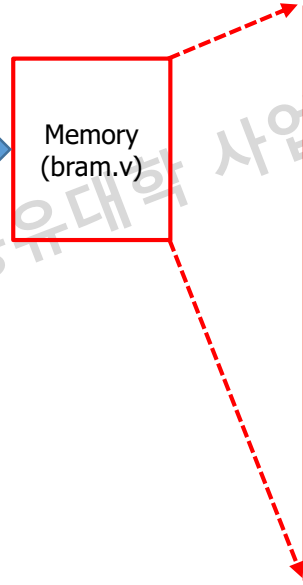
DMA: bandwidth

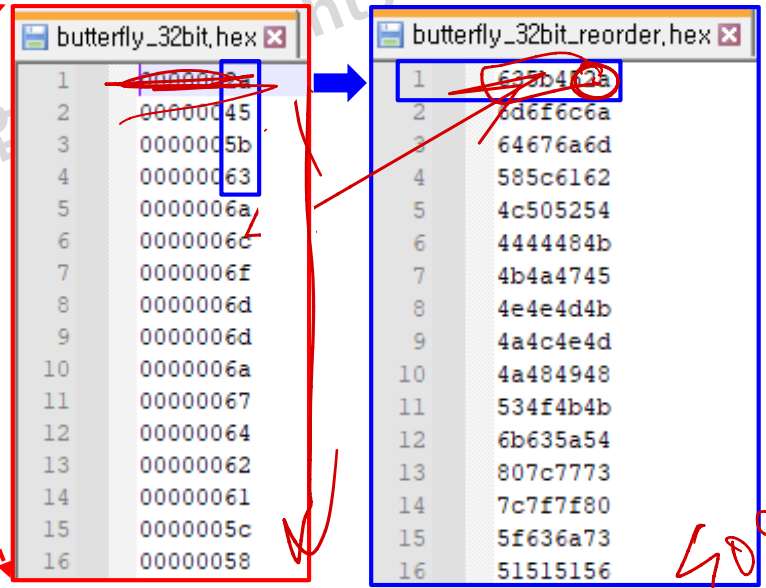


To do ...

- Reorganize the input file
  - The input image (img/butterfly_32bit_reorder.hex) is stored in Memory





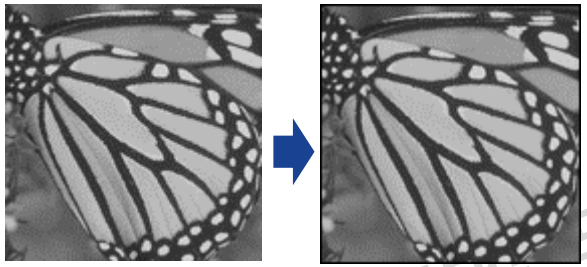


Computation utilization: Layer 1

- Layer 1: num_ops=9 (=3×3×1) < 16
  - Layer 1: 3×3×1
  - din: 1×1×16

➔ din[9] = din[10] = ...= din[15] = 0

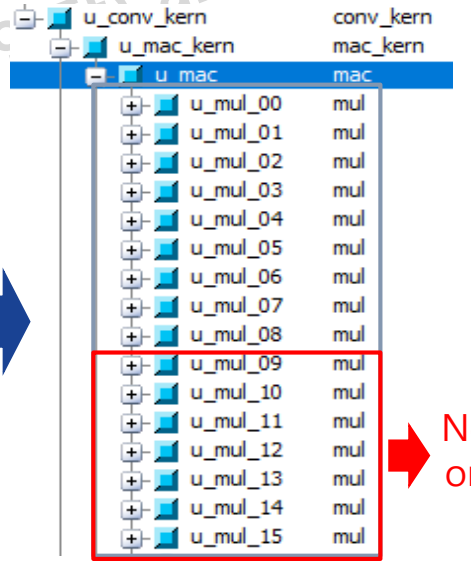
out(0,0,:)



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

din

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

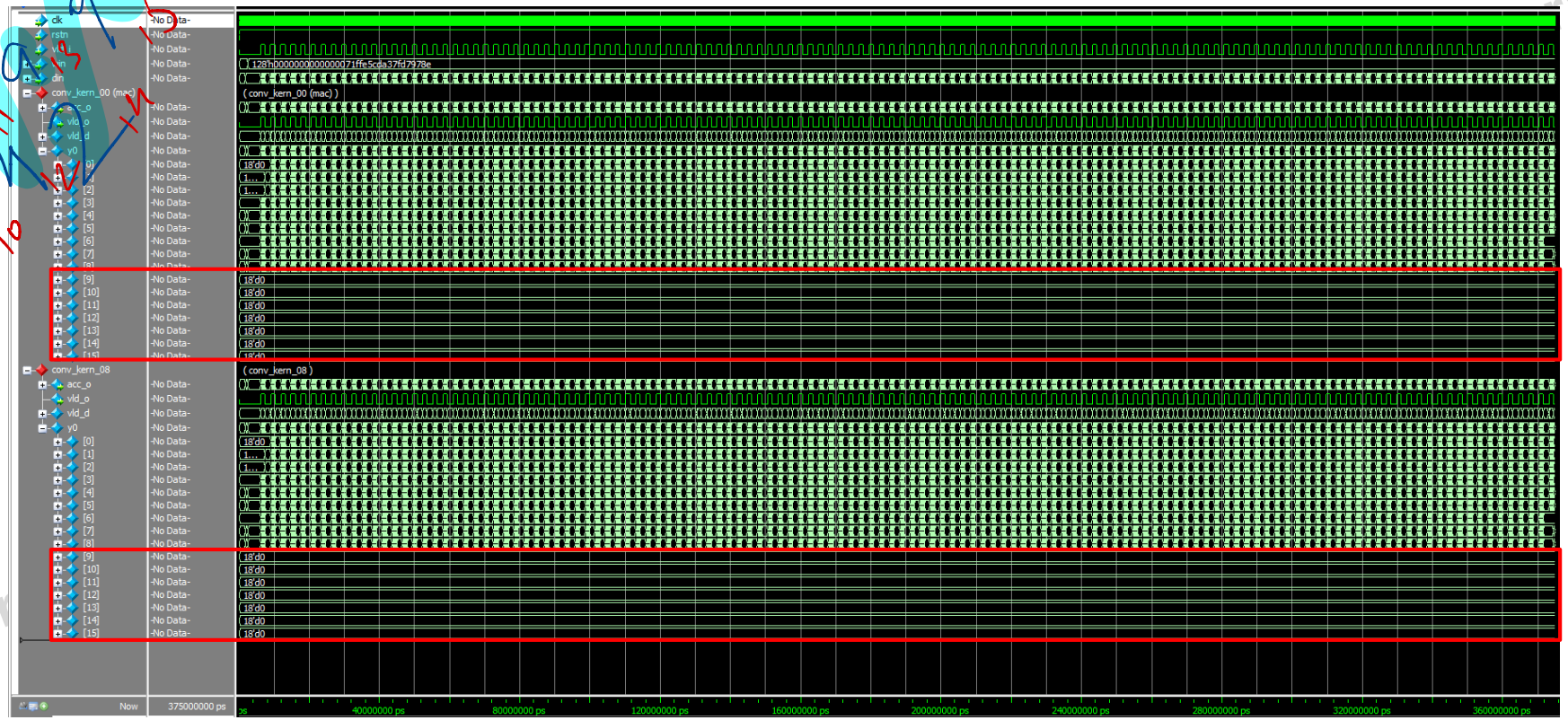


No impact on output

conv_kern: Ti=16

Computation utilization: Layer 1

- Do simulation Layer 1 (3×3×1)
  - din: 1×1×16 ➔ din[9] = din[10] = …= din[15] = 0
  - It seems that there is no much room for improvement

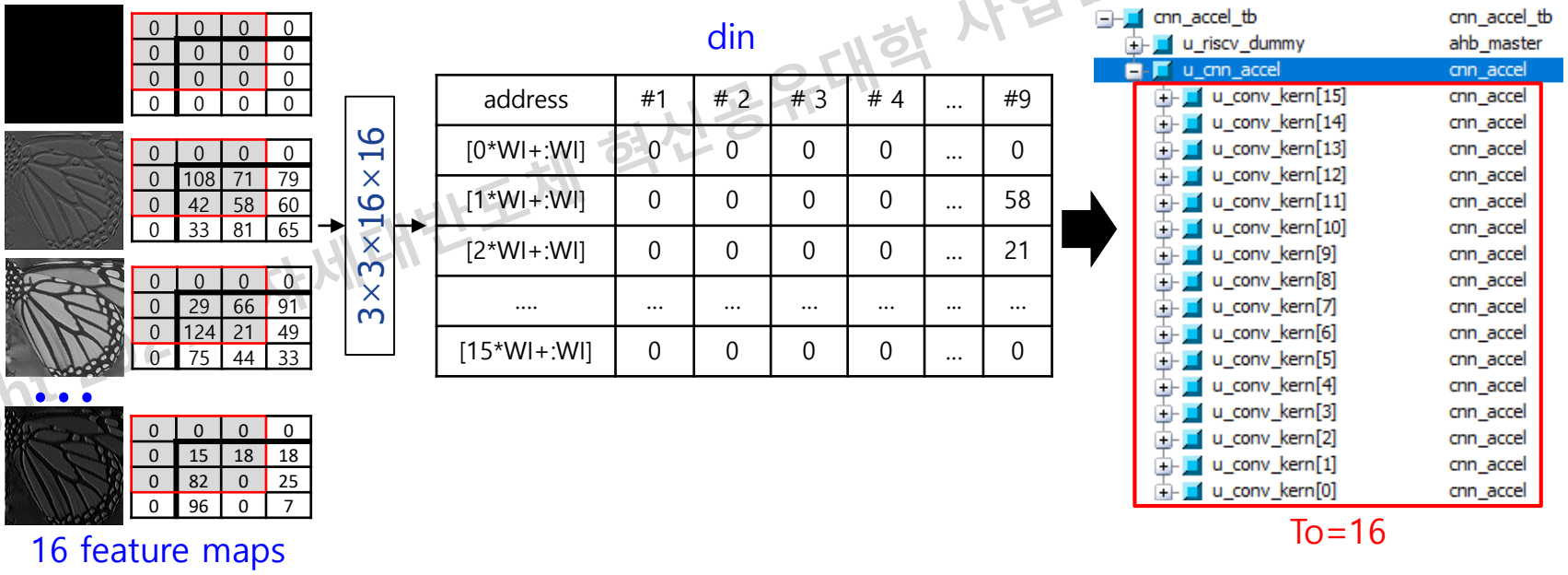


Computation utilization: Layer 2

- 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 output feature maps

  ⇒All Pes are fully utilized → No room for improvement



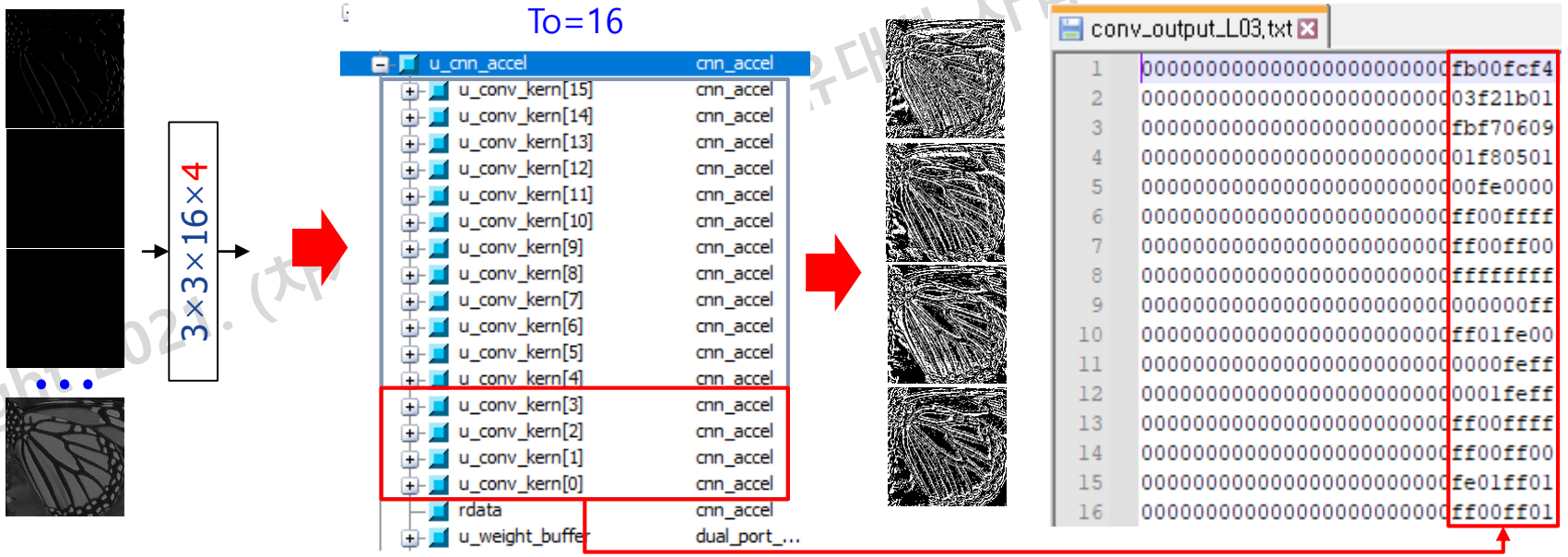
Computation utilization: Layer 3

- Layer 3: 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

⇒ 4 output feature maps

⇒ Only 4 convolution kernels actually contributes to output generation.

⇒ Can we utilize 12 convolution kernels? Estimated speed up for L3: x4



To do ...

- Modify cnn_accel.v to speed up Layer 3
  - Try to utilize all 16 convolution kernels instead of 4.
  - Expected speedup
    - Layer 3: ×4
    - Overall: ×1.47

Road map

Sub-pixel Layer

Review
System Integration

Optimization

Execution time
reduction

Buffer reduction

Buffer size profiling

- Our CNN accelerator
  - 256 PEs (=16×16), 8-bit fixed weight/input, 16-bit biases, scales.
  - Weight/scale/bias buffers: 55.5 Kbits
  - Input buffer: 128 Kbits
  - Dual buffers for feature maps: 4,096 Mbits.

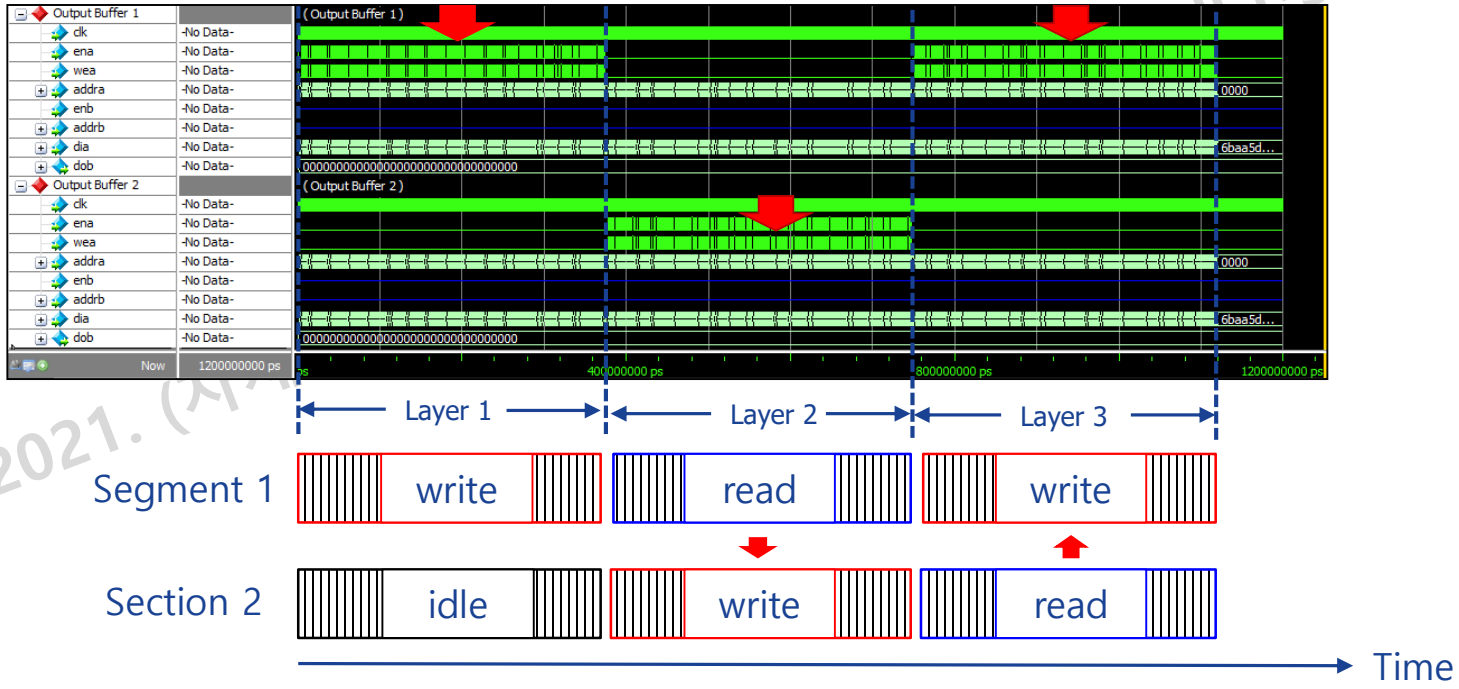
Buffer		Buffer size (Ti=16, To=16, WIDTH=128, HEIGHT=128)		
		Word (bit)	No. of words	Size (bit)
Filter	Weight	8*Ti	3*(3*3*To)	3*(3*3*To)*(8*Ti) (=54K)
	Bias	16	3*To	3*To*16 (=0.75K)
	Scale	16	3*To	3*To*16 (=0.75K)
Input	in_img	8	WIDTH*HEIGHT	WIDTH*HEIGHT*8 (=128K)
Output	Buffer 1	8*To	WIDTH*HEIGHT	WIDTH*HEIGHT*8*To (=2M)
	Buffer 2	8*To	WIDTH*HEIGHT	WIDTH*HEIGHT*8*To (=2M)

Target selection

- How to define a target for optimization?
  - Find a **“high-impact”** target
- Why?
  - Amdahl’s law for buffer reduction

Approach 1: using off-chip memory

- Do NOT store the feature map in the buffer in the CNN accelerator
- Write the feature maps to external memory
- Read/Load the feature maps when executing a layer, i.e. Layer 2, 3.



Approach 2: Line buffer

- The input buffer
  - 128 Kbits (=HEIGHT*WIDTH*8)
  - DMA takes 188us.

⇒ If it is possible to optimize it, why not?
  - **Is there any room for improvement?**

- **Observations**
  - We needs three lines of the input image to compute an output line

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

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

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

3 lines

Approach 2: Line buffer

- The input buffer
  - 128 Kbits (=HEIGHT*WIDTH*8)
  - DMA takes 188us.

⇒ If it is possible to optimize it, why not?

- **Is there any room for improvement?**

- **Observations**
  - We needs three lines of the input image to compute an output line
  - No need to store all images
  - Preloading a line while doing computation

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

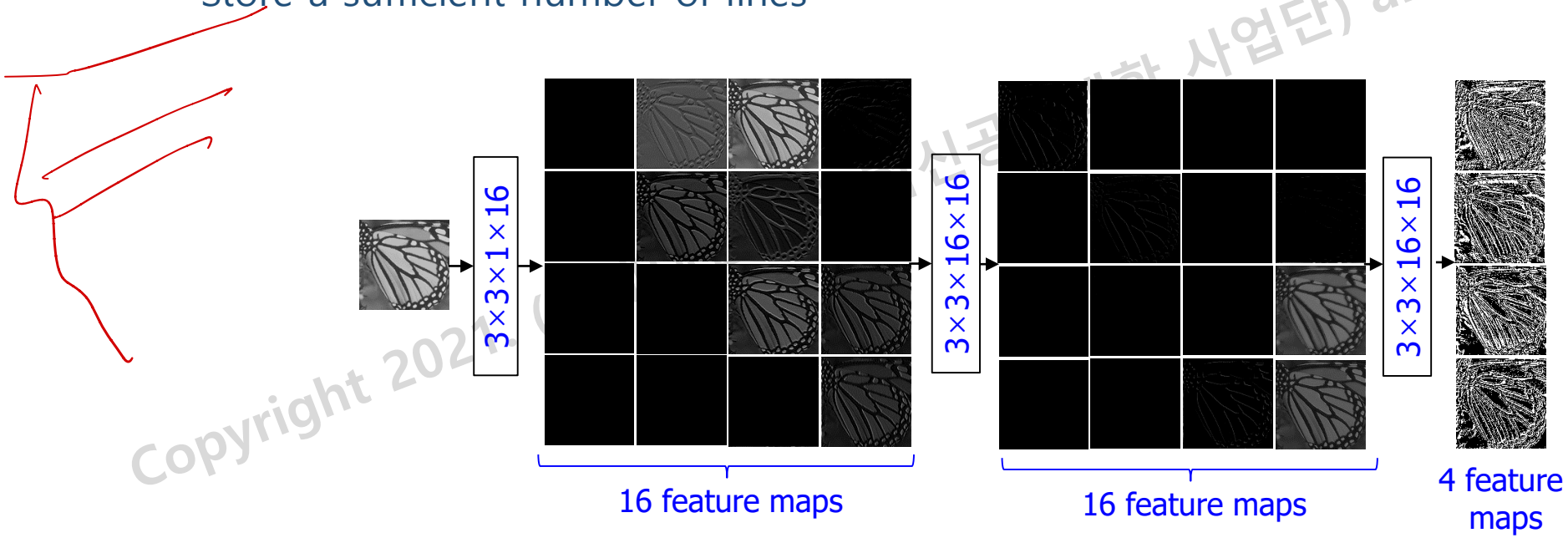
0	0	0	0	0	0	0	0
0	42	69	91	99	106	108	111
0	105	42	56	84	106	113	112
0	72	43	42	68	109	112	104

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

3 lines → 3 lines

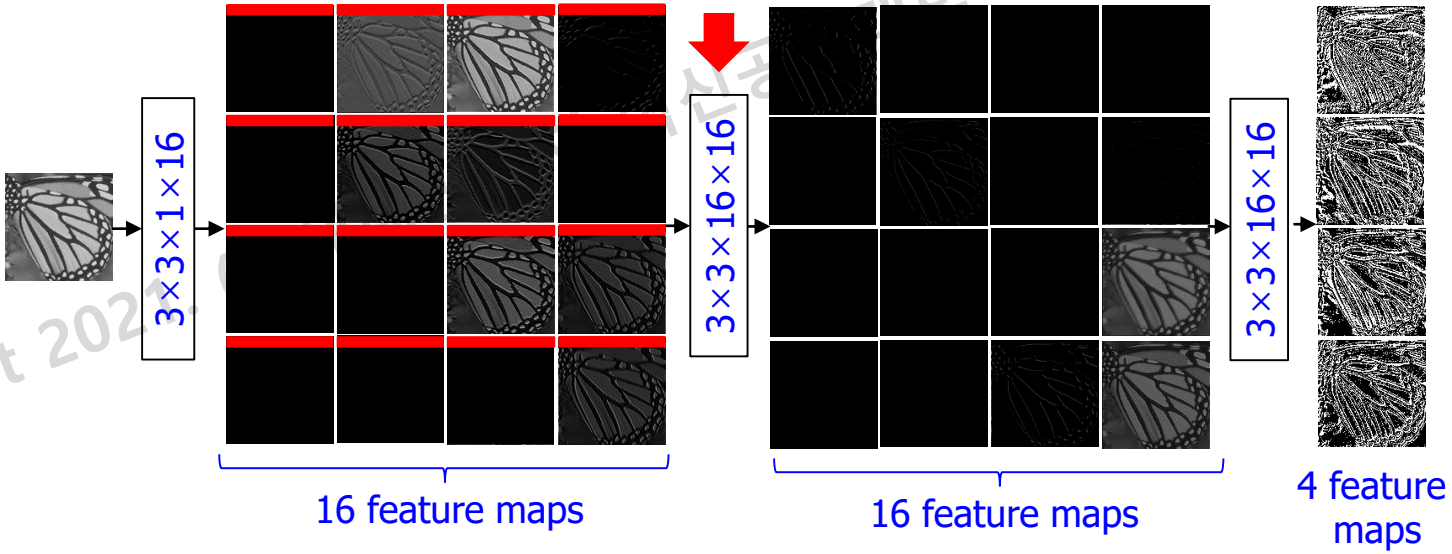
Approach 3: Layer fusion

- Baseline:
  - Store all feature maps → Need a huge buffer
  - Then start the next layer.
- Layer fusion
  - Start a layer "as early as possible"
  - Store a sufficient number of lines



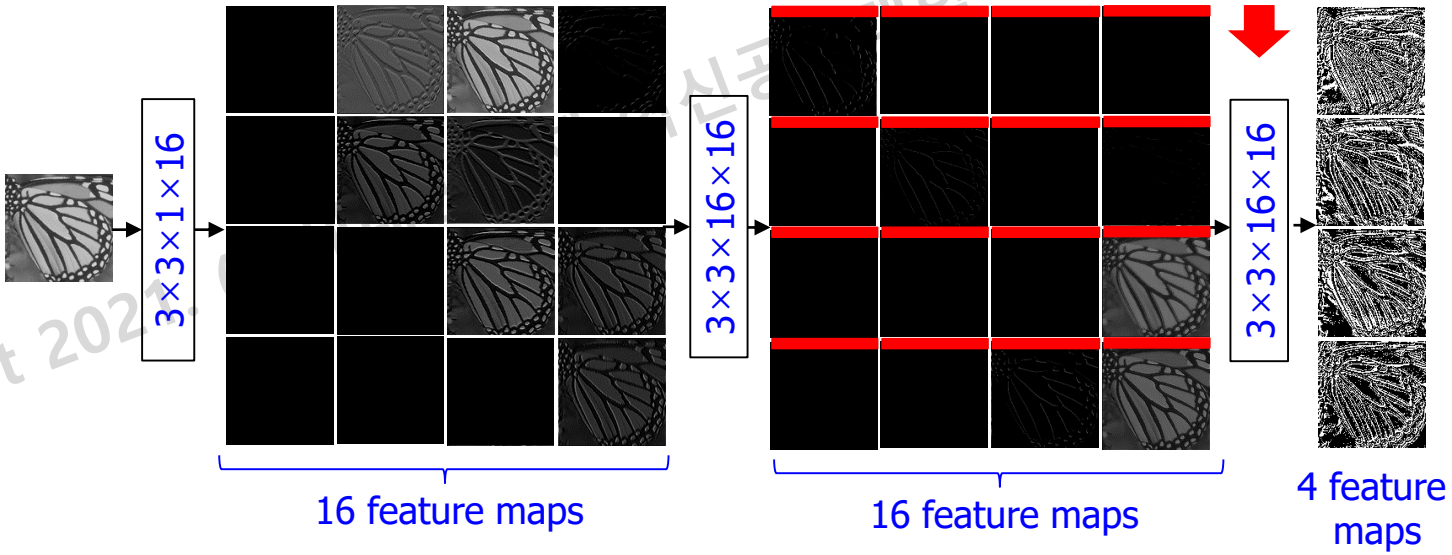
Layer fusion

- Layer fusion
  - Start a layer "as early as possible"
  - Store a sufficient number of lines
- Example
  - Compute a few lines of the output feature maps in Layer 1
  - Outputs are stored in a SMALL buffer
  - Then, start layer 2



Layer fusion

- Layer fusion
  - Start a layer "as early as possible"
  - Store a sufficient number of lines
- Example
  - Compute a few lines of the output feature maps in Layer 2
  - Outputs are stored in a SMALL buffer
  - Then, start layer 3



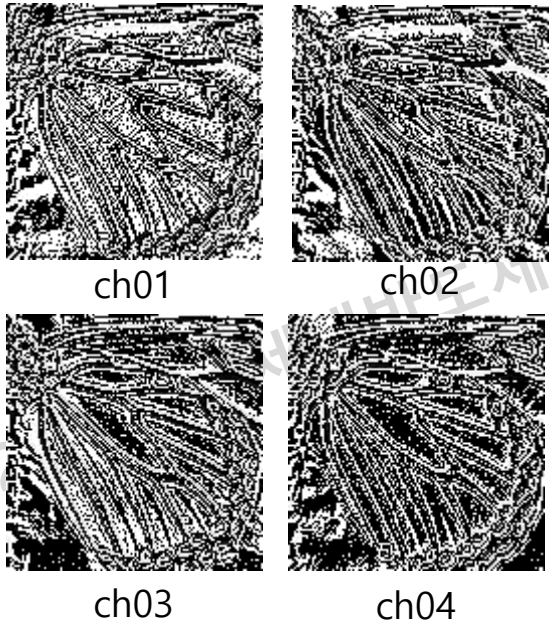
To do ...

- Modify cnn_accel.v to reduce the buffer size for the input image
  - Use a few lines of the image
  - The procedure
    - Preload a few lines
    - Start Layer 1 as the baseline
    - While doing computation, load some next image lines

Don’t forget the correctness!!!

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.

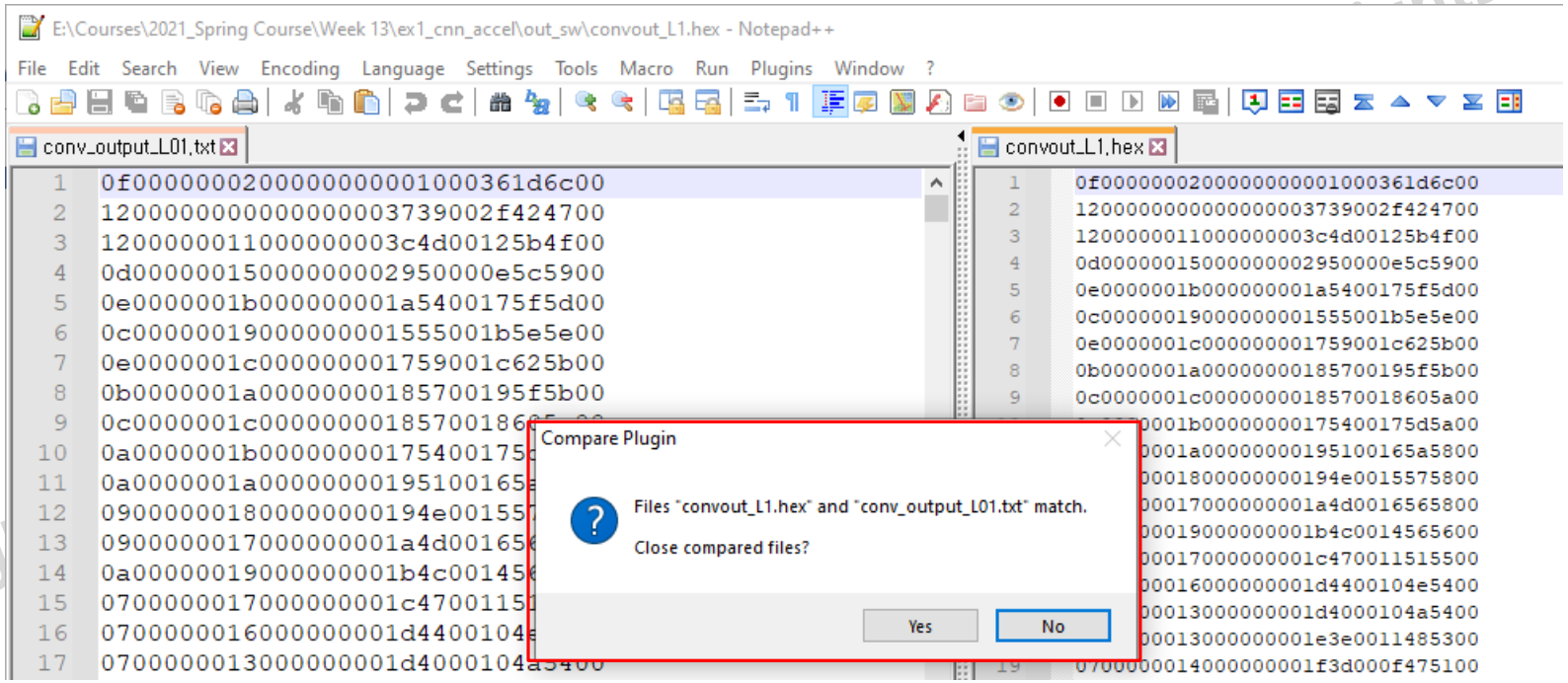
Load the images after Layer 3

```
check_hardware_results.m
 1 -   clc
 2 -   clear all
 3 -   close all
 4
 5 -   for ch = 1:4
 6         % Output from the reference S/W
 7         %im_sw = imread(sprintf('out_sw/ofmap_L01_ch%02d.bmp',ch));
 8 -       im_sw = imread(sprintf('out_sw/ofmap_L03_ch%02d.bmp',ch));
 9         % Output from the H/W simulation
10         %im_hw = imread(sprintf('out/convout_layer01_ch%02d.bmp',ch));
11 -       im_hw = imread(sprintf('out/convout_ch%02d.bmp',ch));
12 -       im_hw = im_hw(:,:,1);    % Gray image
13
14         % Calculate the difference between S/W and H/W outputs
15 -       img_diff = abs(single(im_hw) - single(im_sw));
16 -       max_diff = max(img_diff(:));
17 -       if(max_diff == 0)
18 -           fprintf('Results of the channel %02d are same!\n', ch);
19 -       else
20 -           fprintf('ERROR: Results of the channel %02d are different!\n', ch);
21 -           disp(max_diff);
22 -           figure(ch)
23 -           imshow(uint8(img_diff));
24 -       end
25 -   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!
fx >>
```

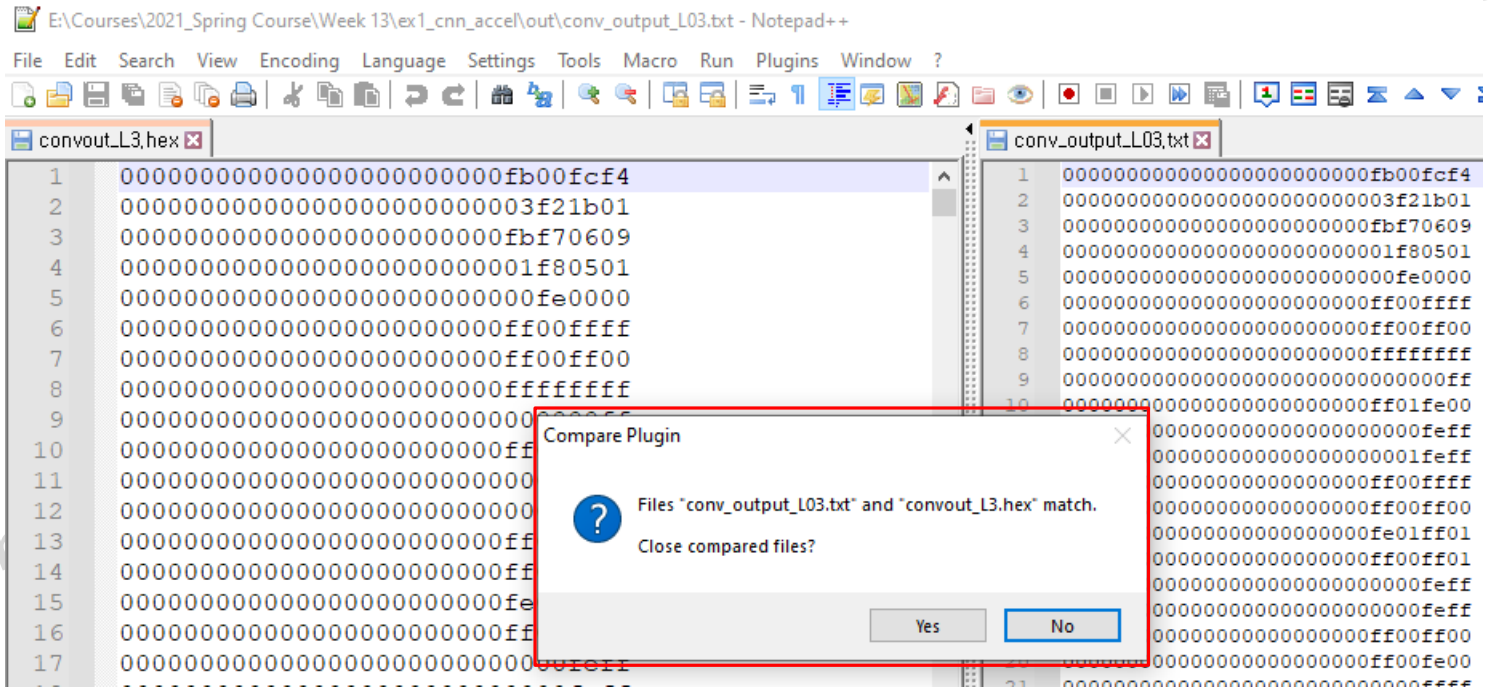
Verification

- 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



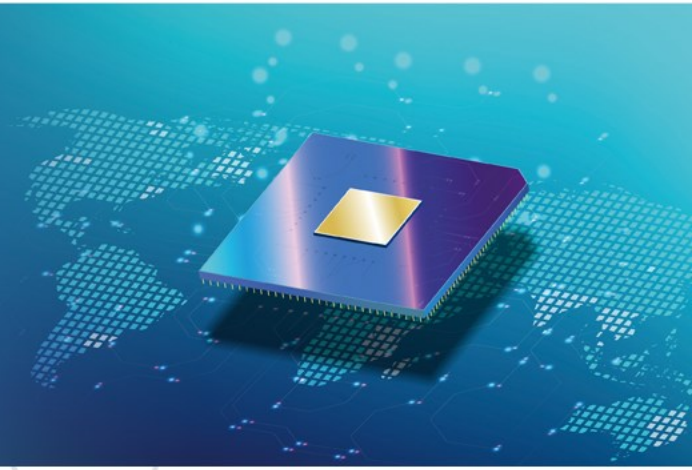
Verification

- 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)



Backup Slides

How to debug a program?



Dummy error

- Generate an error artificially.
  - Define GENERATE_OUTPUT_ERROR (debug.v).
  - Generate an error for the last layer (bnorm_quant_act.v)
    - Errors happen when quantizing the output of the last layer

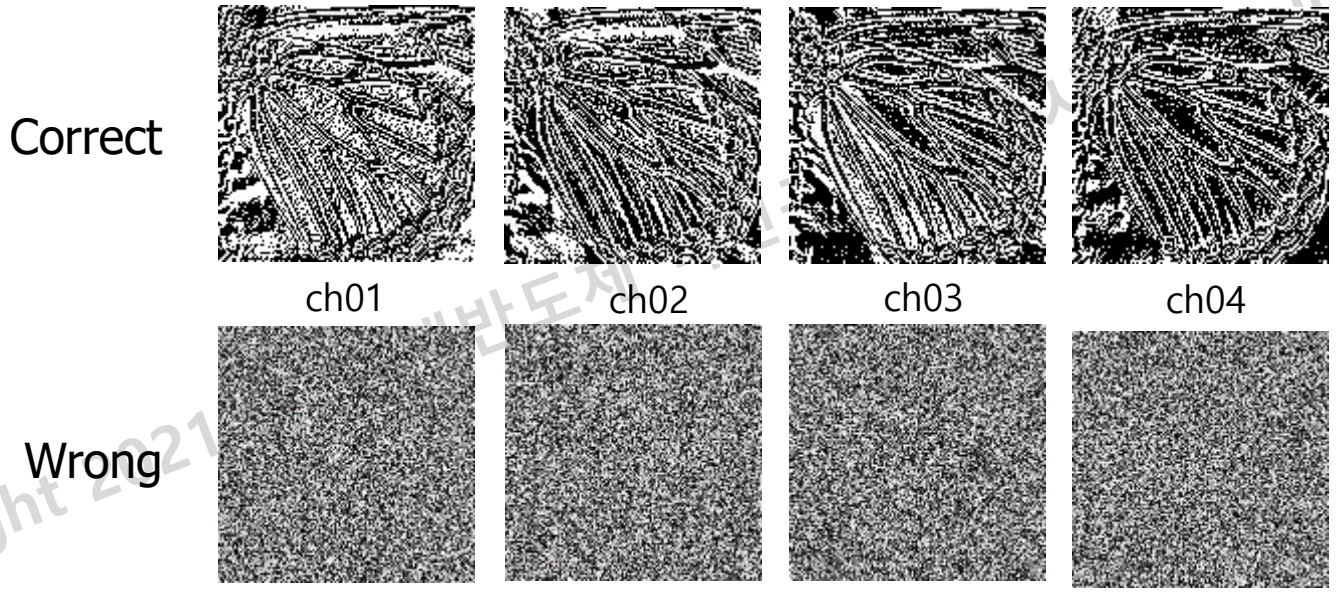
⇒ The challenging thing is that the error happens in the last layer

⇒ Error detection: Have to do full simulation

```
// Linear (Last Layer)
always @(posedge clk or negedge resetn)
    if(!resetn)
        accum_final <= 'h0;
`ifdef GENERATE_OUTPUT_ERRROR
    else
        accum_final <= acc_mean;        // Wrong
`else
    else
        accum_final <= acc_quant;       // Correct
`endif
```

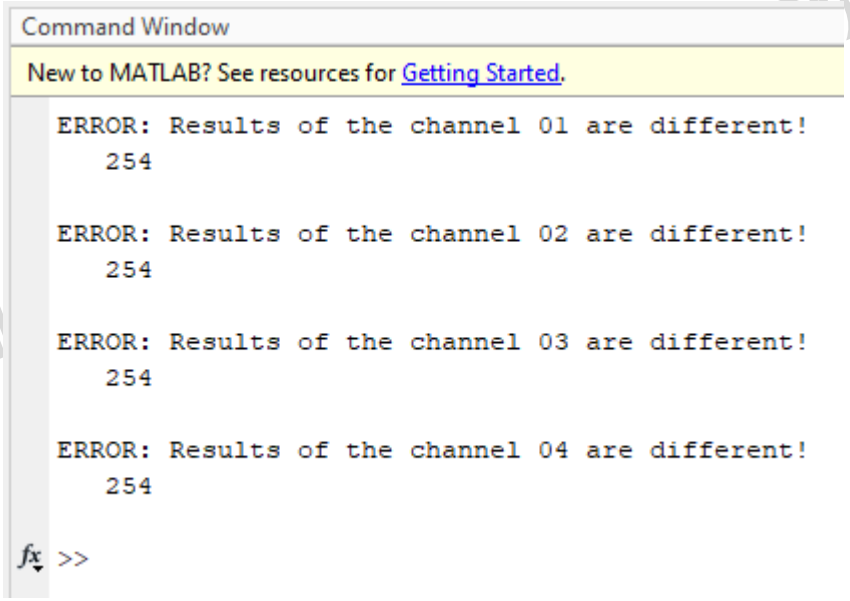
Visualize an error

- Do full simulation with time = 4ms
- 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.



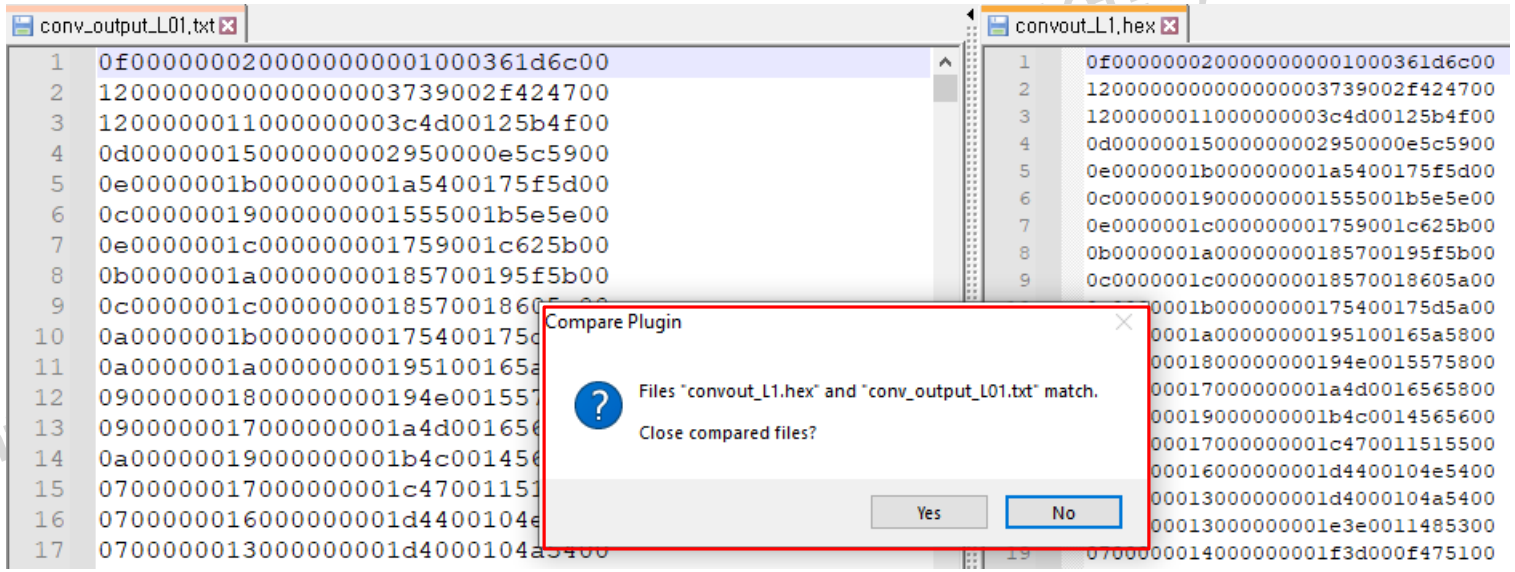
Visualize an error

- Do full simulation with time = 4ms
- 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.



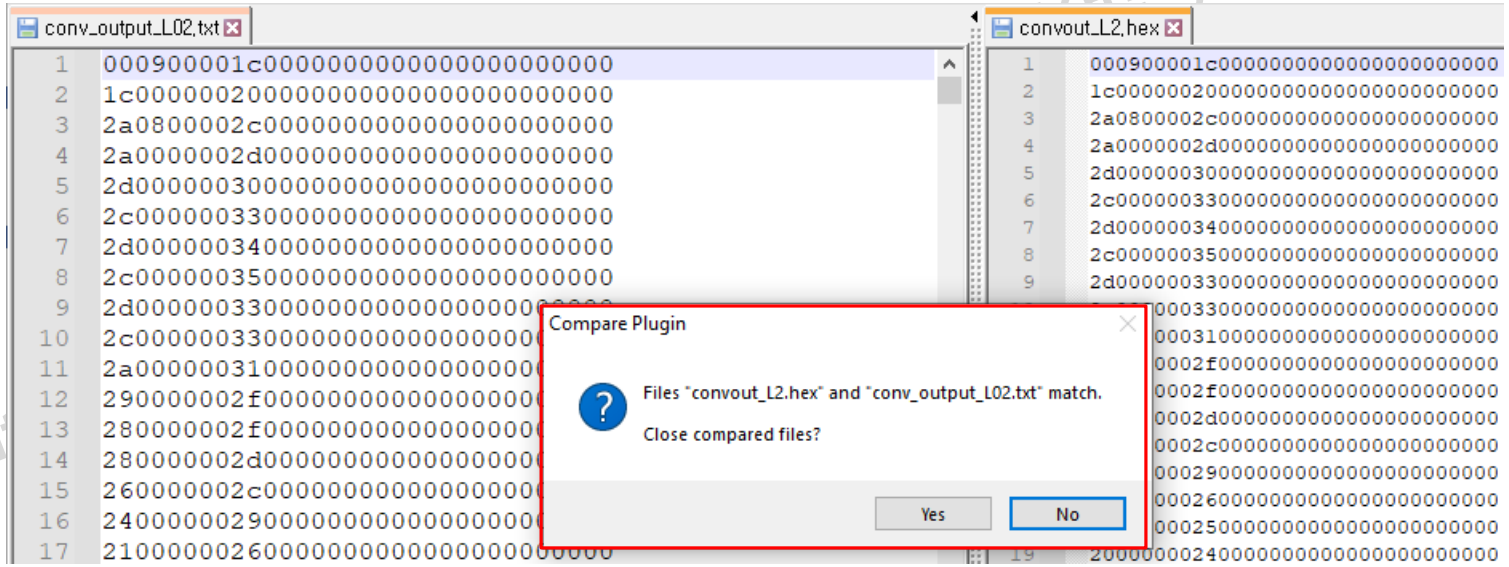
Debugging

- Detect the error POSITION.
  - When does the error occur?
- Compare two hex files by using Notepad++
  - Plugins→ Compare → Compare (Ctrl + Alt + C)
  - Compare the outputs of S/W and H/W for Layer 1



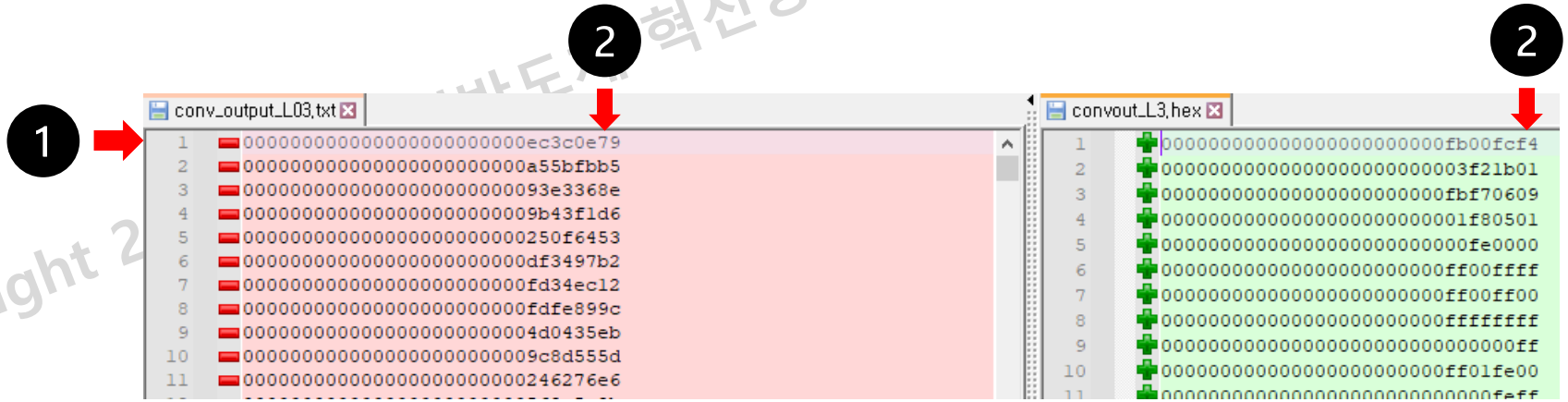
Debugging

- Detect the error POSITION.
  - When does the error occur?
- Compare two hex files by using Notepad++
  - Plugins→ Compare → Compare (Ctrl + Alt + C)
  - Compare the outputs of S/W and H/W for Layer 2



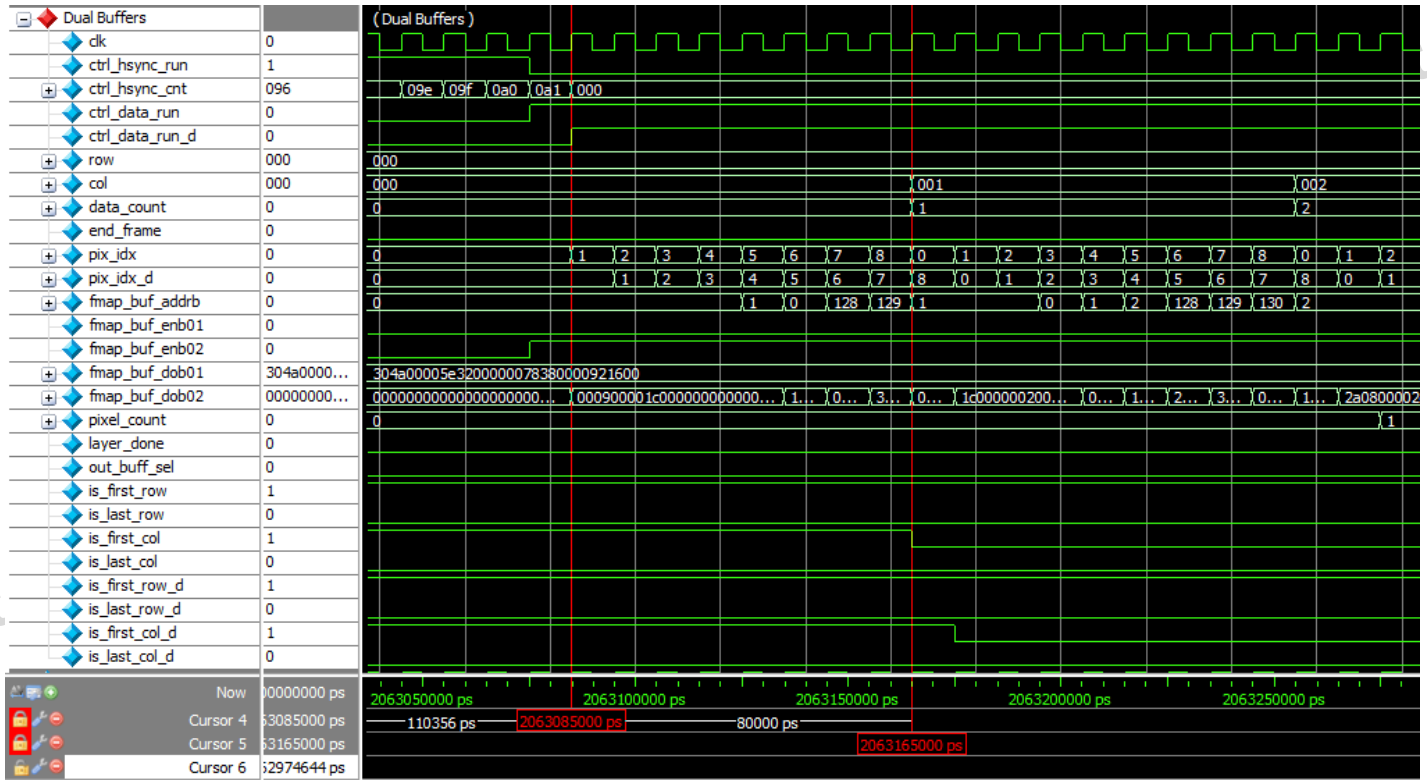
Debugging

- Detect the error POSITION.
  - When does the error occur?
- Compare two hex files by using Notepad++
  - Plugins→ Compare → Compare (Ctrl + Alt + C)
  - Compare the outputs of S/W and H/W for Layer 3
- Found it: Layer 3
  1. The first pixels (data_count = 1).
  2. All four convolution kernels give the wrong outputs



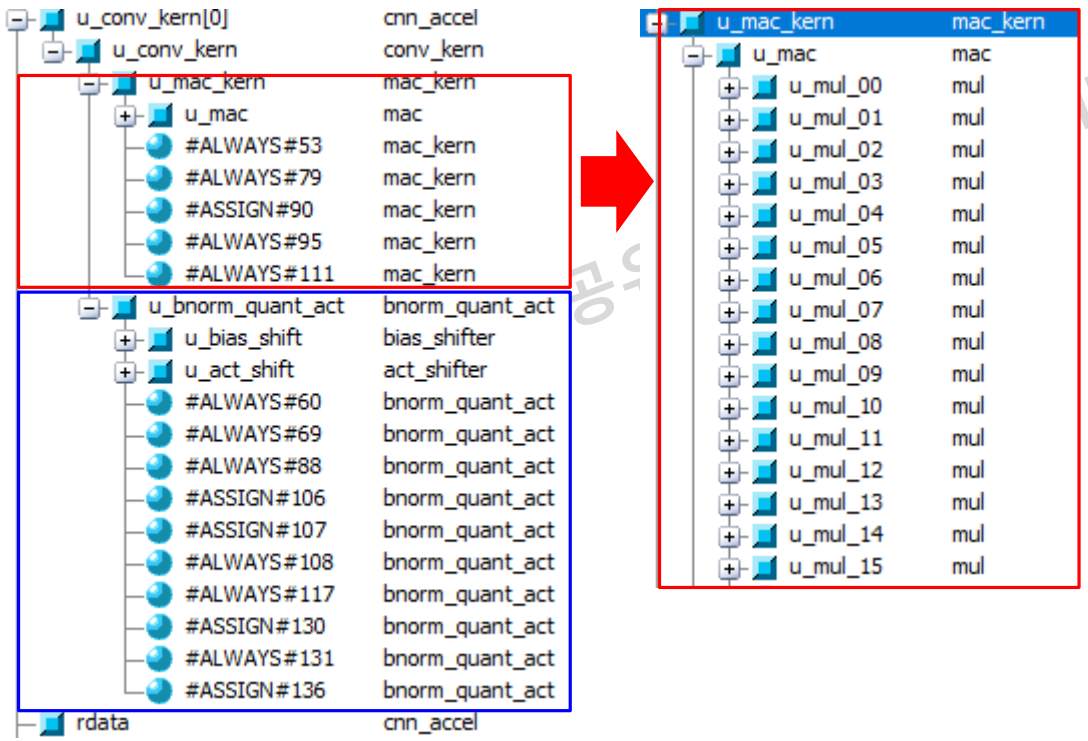
Debugging

- Check if the out-coming data from dual buffers are correct
  - Addresses vs data
  - Use cursors to log when the error occurs



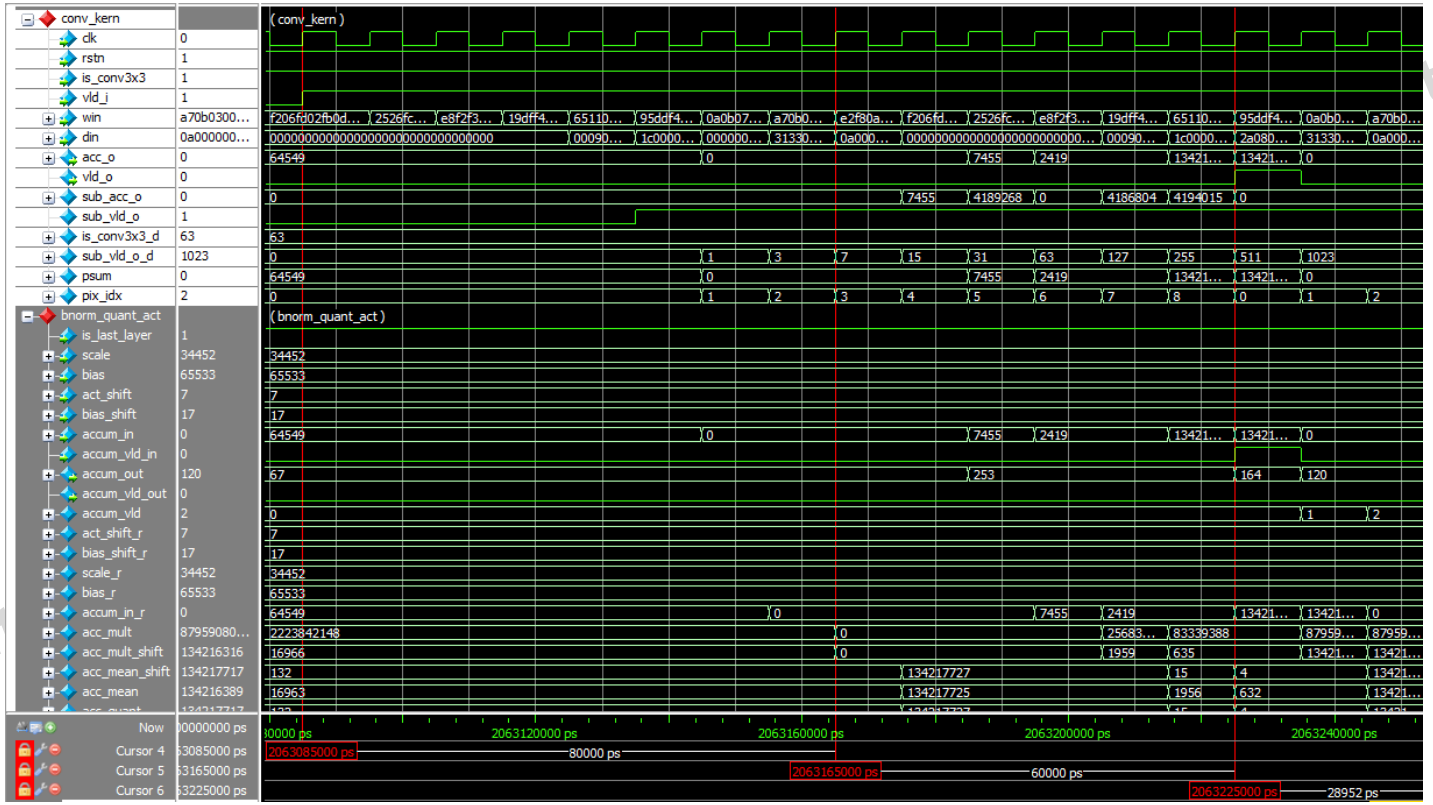
Debugging

- Convolution kernel (conv_kern.v)
  - MAC kernel (mac_kern.v)
  - Batch normalization/quantization/activation (bnorm_quant_act.v)



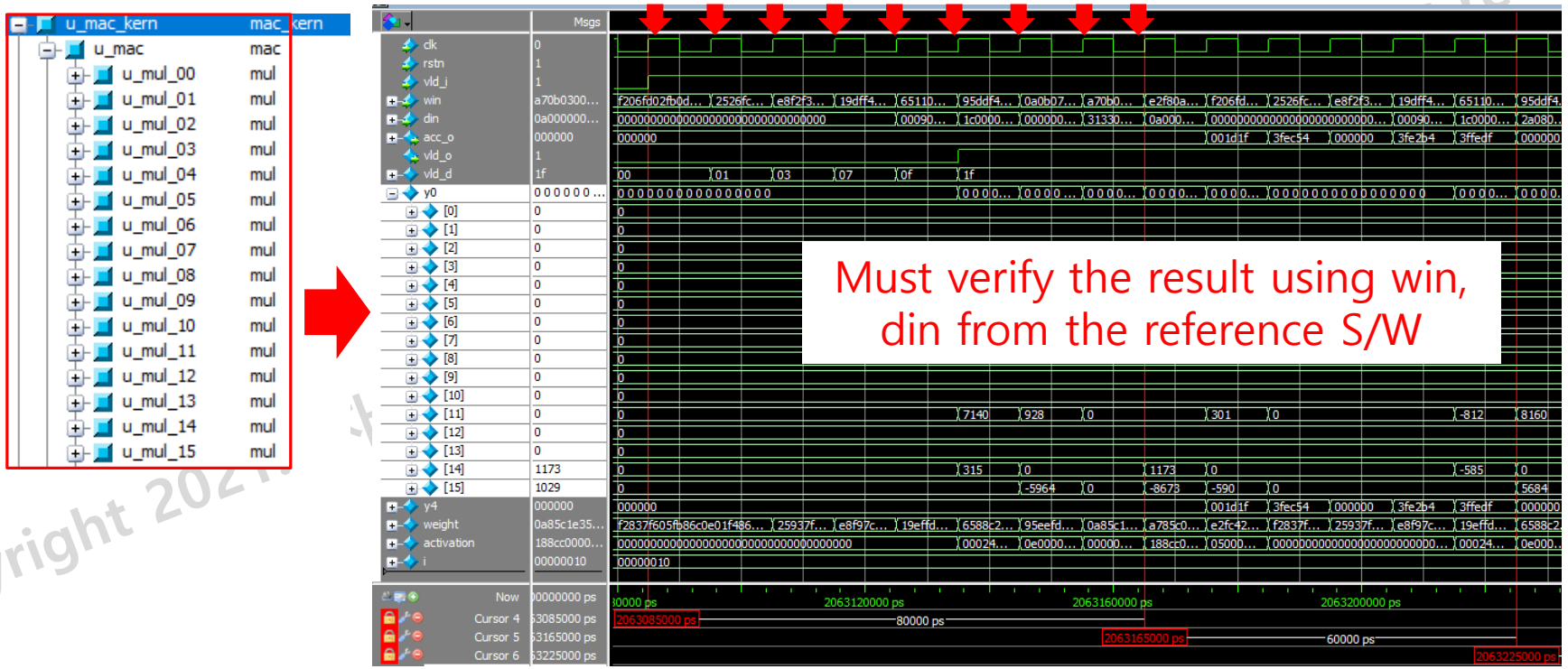
Debugging

- The output of bnorm_quant_act has an ERROR
- Find the error location: (acc_o)
  - If the output of mac_kern (the input of bnorm_quant_act) is OK?



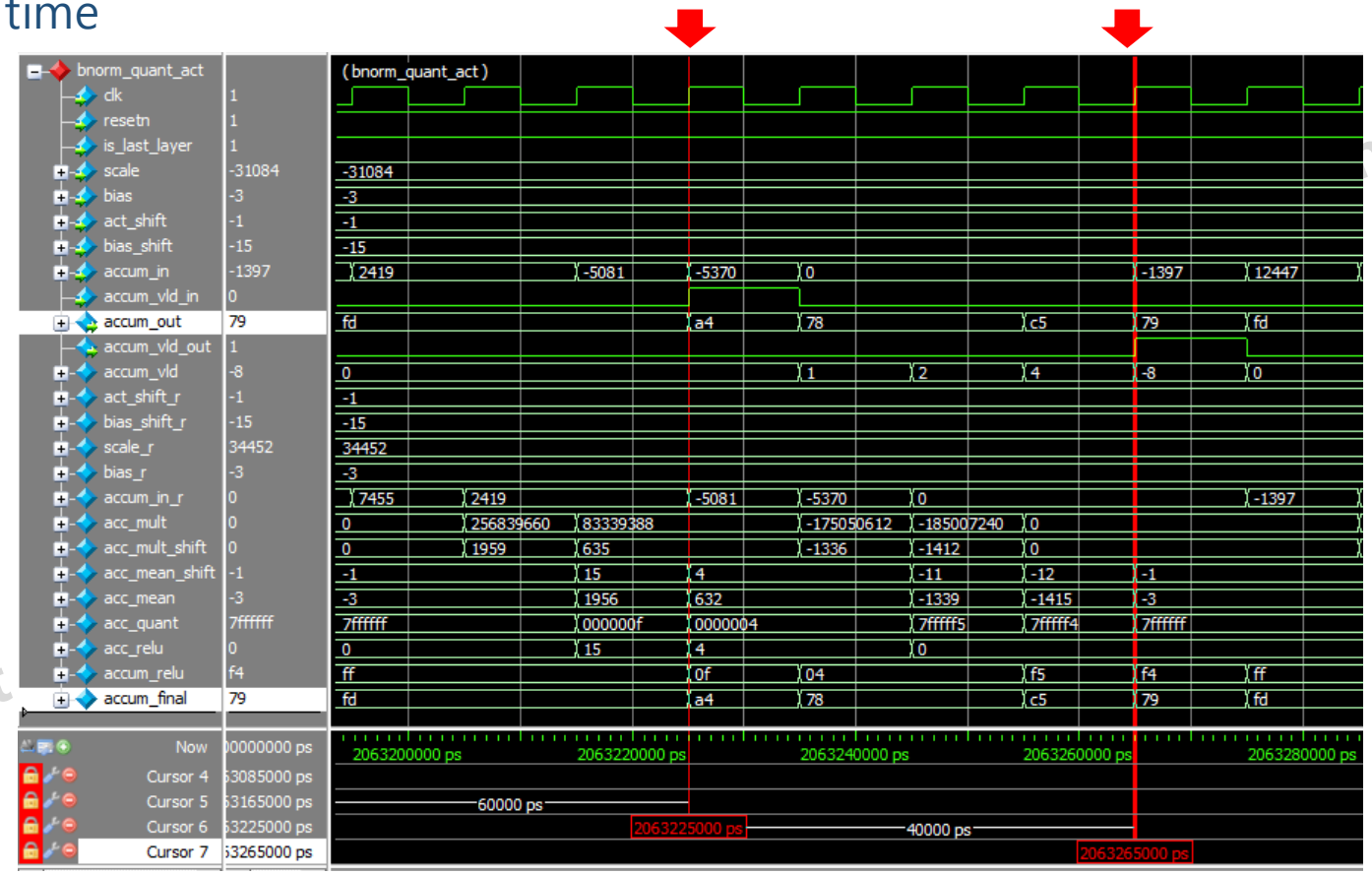
Case 1: NOT OK

- Find the error location: (acc_o)
  - If the output of mac_kern (the input of bnorm_quant_act) is OK?
- Case 1: NOT OK
  - The error occurs in "mac_kern"



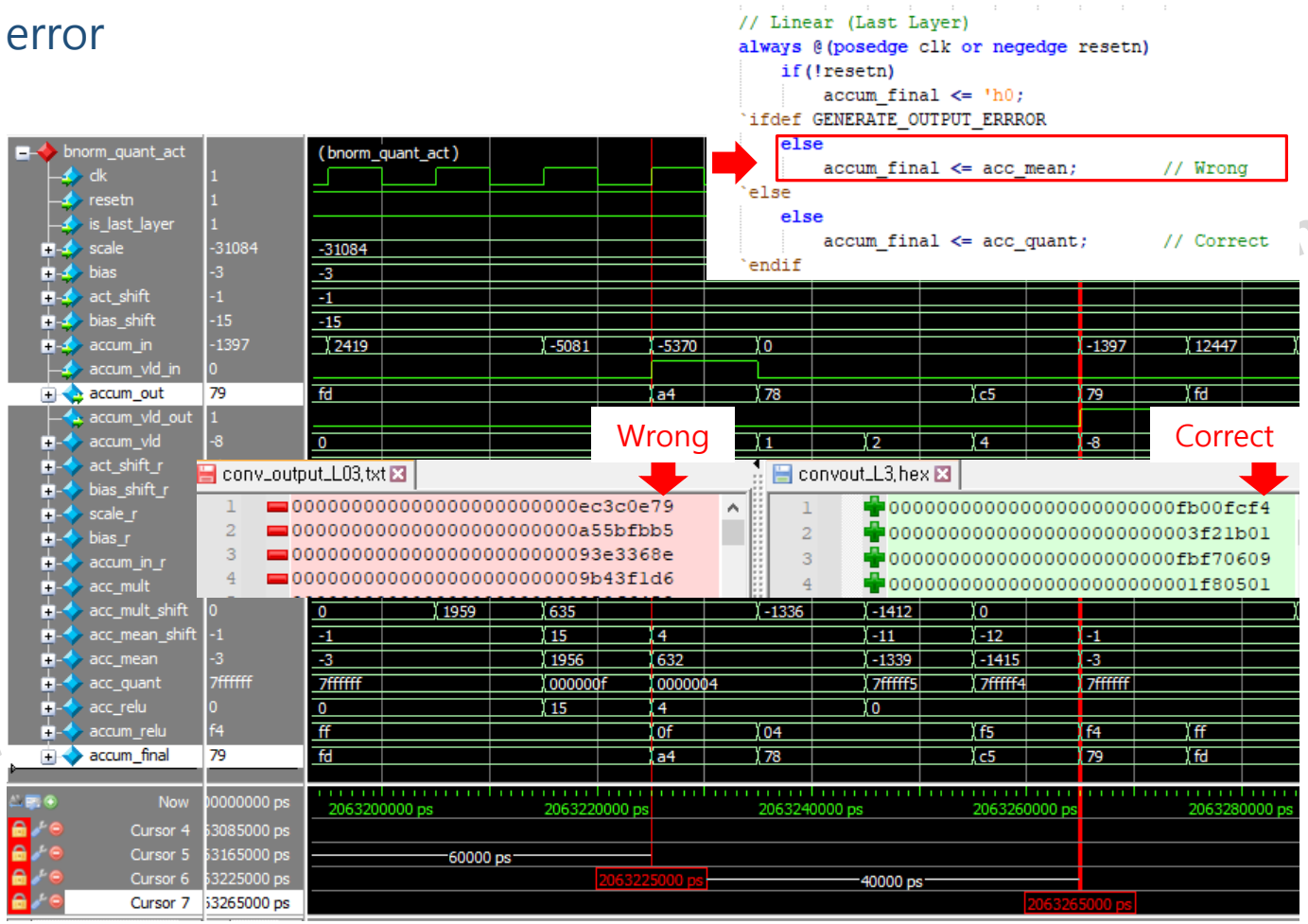
Case 2: OK

- The output of mac_kern is CORRECT
  - The error occurs at bnorm_quant_act.v
  - Log the time



Case 2: OK

- The output of mac_kern is CORRECT
  - Find the error
  - Fix it.



Debugging flow

- Detect the error location with some tips:
  - Compare the files to detect the error location
    - Understand the structure (pages 3-6).
    - Data scheduling (pages 16-24)
    - Data order (pages 51-55)
  - Use the cursor to localize and log the error location
- Fix the error
- Others
  - Make a unit test
  - For example, the function conv_kern has an error for given inputs

  ⇒ Make a test bench with the inputs

  ⇒ Find the error location and fix it.
- The most challenging task is to find where the error occurs.