차세대 반도체 디지털 혁신공유대학 사업단

POLARIS

System Integration II

DMA, Top system

2022.02.28 (Mon)

Road map

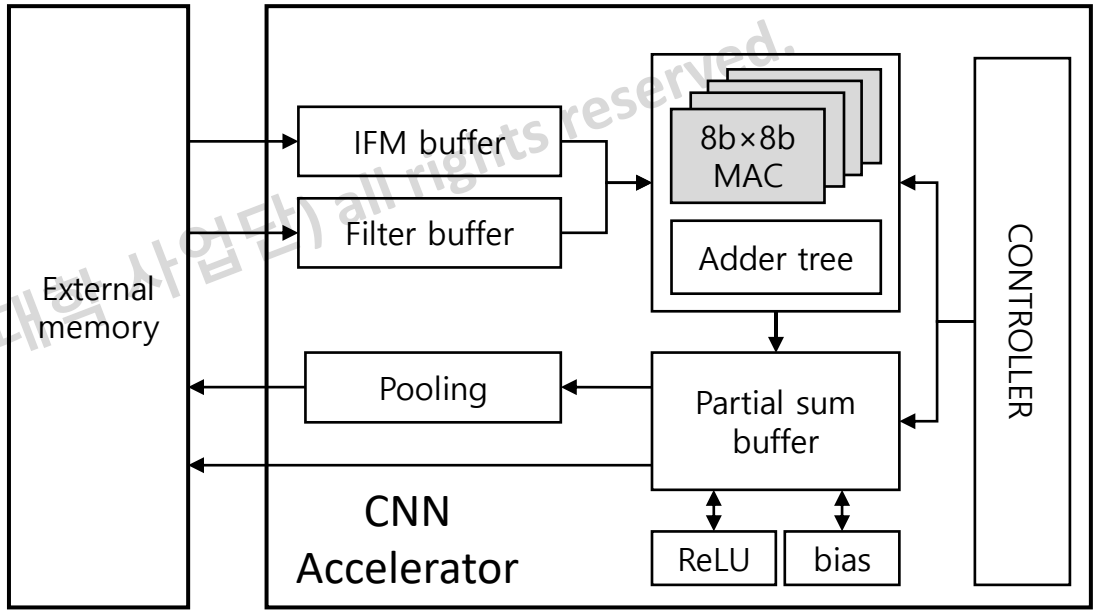
Review

DMA

Top system

Inference engine

- Processing Element (PE): MAC, adder tree
- Buffers
  - Input feature map (IFM)
  - Filters
  - Intermediate results or partial sum
- Pooling
- Controller



Scope

- The baseline code
  - Are designed provide the usage of components via examples or test benches
    - Processing elements: DSP, mul.v, mac.v
    - On-chip buffers: spram, dpram
    - Controller: cnn_ctrl.v
    - Load data from memory (axi_dma_rd.v) or store data to memory (axi_dma_wr.v)

- AIX2022
  - Understand how to use modules
  - Modify/optimize parameters
  - Implementation

layer	type	filter	input	output	Remark
1	conv	3x3x1x16	128x128x1	128x128x16	Tutorial
0	conv	3x3x3x16	320x320x3	320x320x16	AIX2022 Mid-term
1	max		320x320x16	160x160x16	
2	conv	3x3x16x32	160x160x16	160x160x32	
3	max		160x160x32	80x80x32	
4	conv	3x3x32x64	80x80x32	80x80x64	
5	max		80x80x64	40x40x64	

IO files

- How to prepare an input file, do processing and verify the output
- Simulation, verification, and debugging

Input file

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
13 62
14 61
15 5c
16 58
```

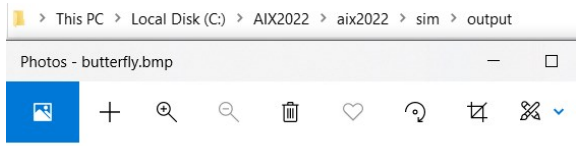
8-bit

butterfly_32bit.hex

```
1  0000002a
2  00000045
3  0000005b
4  00000063
5  0000006a
6  0000006c
7  0000006f
8  0000006d
9  0000006d
10 0000006a
11 00000067
12 00000064
13 00000062
14 00000061
15 0000005c
16 00000058
```

32-bit

Output file (e.g., bmp image writer)



Bmp writer (bmp_image_writer.v)

- When all pixels are stored in a buffer
  - frame_done == 1

  ⇒ write an BMP image file
- $open(OUTFILE, "wb+"):
  - Open a binary file for writing
  - The integer fd is the file identifier
- $fwrite(fd, "%c", ...)
  - Write a character to the file
- Debug: Open a txt file and write data in a hex file

```
116  initial begin
117      // Open file
118      fd = $fopen(OUTFILE, "wb+");
119      h = 0;
120      w = 0;
121  end
122
123  always@(frame_done) begin
124      if(frame_done == 1'b1) begin
125      // Write header
126          for(i=0; i<BMP_HEADER_NUM; i=i+1) begin
127              $fwrite(fd, "%o", BMP_header[i][7:0]);
128          end
129
130          // Write data
131          for(h = 0; h < HEIGHT; h = h + 1) begin
132              for(w = 0; w < WIDTH; w = w + 1) begin
133                  $fwrite(fd, "%o", out_img[(HEIGHT-1-h)*WIDTH + w][7:0]);
134                  $fwrite(fd, "%o", out_img[(HEIGHT-1-h)*WIDTH + w][7:0]);
135                  $fwrite(fd, "%o", out_img[(HEIGHT-1-h)*WIDTH + w][7:0]);
136              end
137          end
138          $fclose(fd);
139
140      end
141  end
```

Open a binary file

Write Header

Write data

CNN controller (cnn_ctrl.v)

- Generate control signals: Loop generator
- Inputs
  - Clock, reset
  - q_width, q_height, q_frame_size
  - Synchronization delay
    - Frame/layer synchronization (vsync_delay)
    - Row/line synchronization (hsync_delay)
  - Trigger (q_start)
- Outputs
  - Synchronization signals (o_ctrl_vsync_run, o_ctrl_hsync_run, o_ctrl_data_run)
  - Row, column, and pixel index (o_row, o_col, o_data_count)
  - Frame/layer done (o_end_frame)

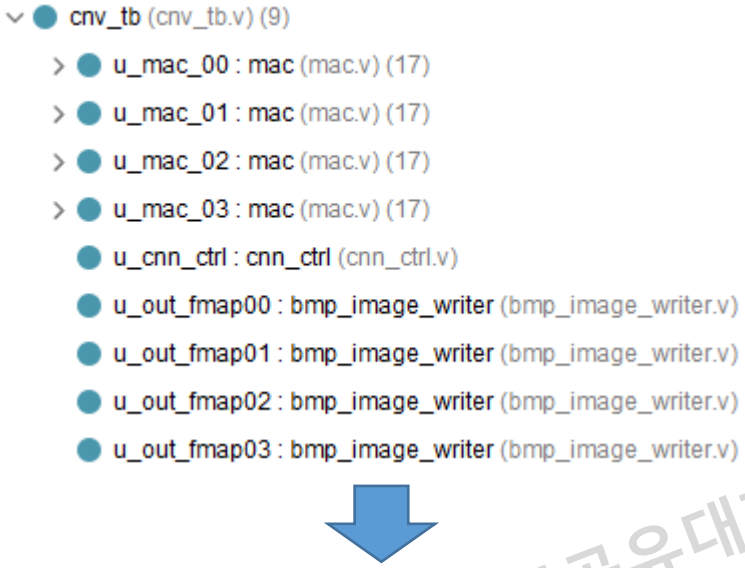
```
1   `timescale 1ns / 1ps
2
3   module cnn_ctrl(
4   clk,
5   rstn,
6   // Inputs
7   q_width,
8   q_height,
9   q_vsync_delay,
10  q_hsync_delay,
11  q_frame_size,
12  q_start,
13  //output
14  o_ctrl_vsync_run,
15  o_ctrl_vsync_cnt,
16  o_ctrl_hsync_run,
17  o_ctrl_hsync_cnt,
18  o_ctrl_data_run,
19  o_row,
20  o_col,
21  o_data_count,
22  o_end_frame
23  );
```

Convolutional kernels (cnv_tb.v)

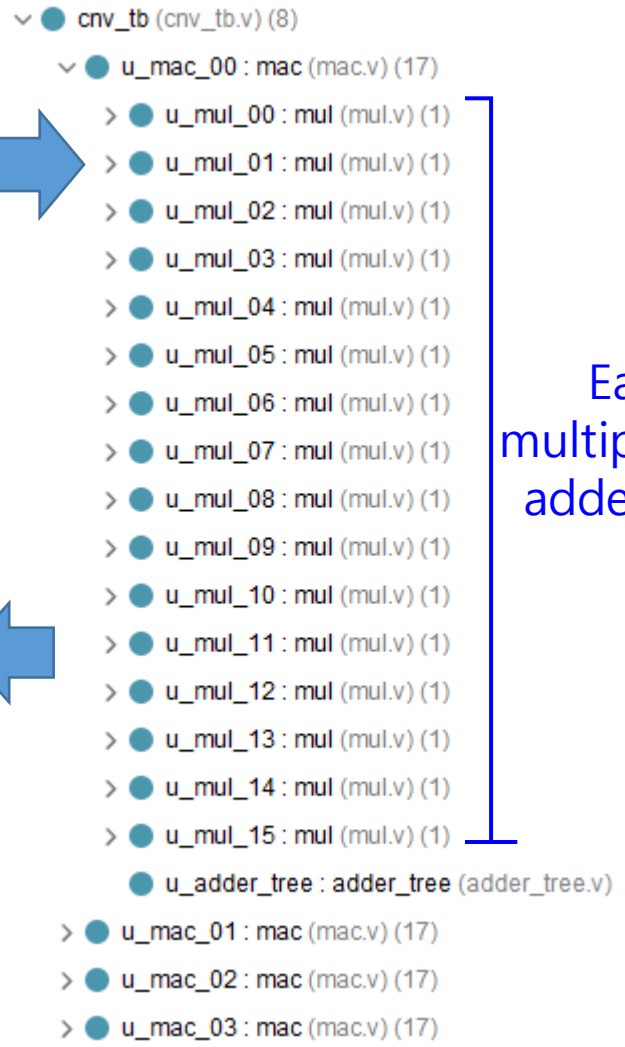
Four convolutiona kernels (mac.v)

CNN controller

Four output writer modules



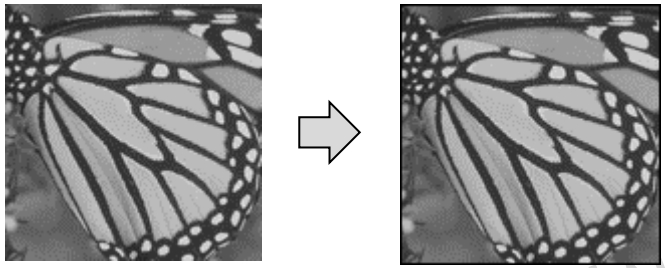
Execute 64 multiplication operations in parallel



Each kernel has 16 multipliers (mul.v) and one adder tree (adder_tree.v)

Sliding windows and mapping

- Generate din
  - A window of a filter is sliding
    - Left to right, top to bottom
  - In this case, only 9 multipliers are actually used.



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

out(0,0,k)[1]

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

out(0,1,k)

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

Generate din

Boundary checking

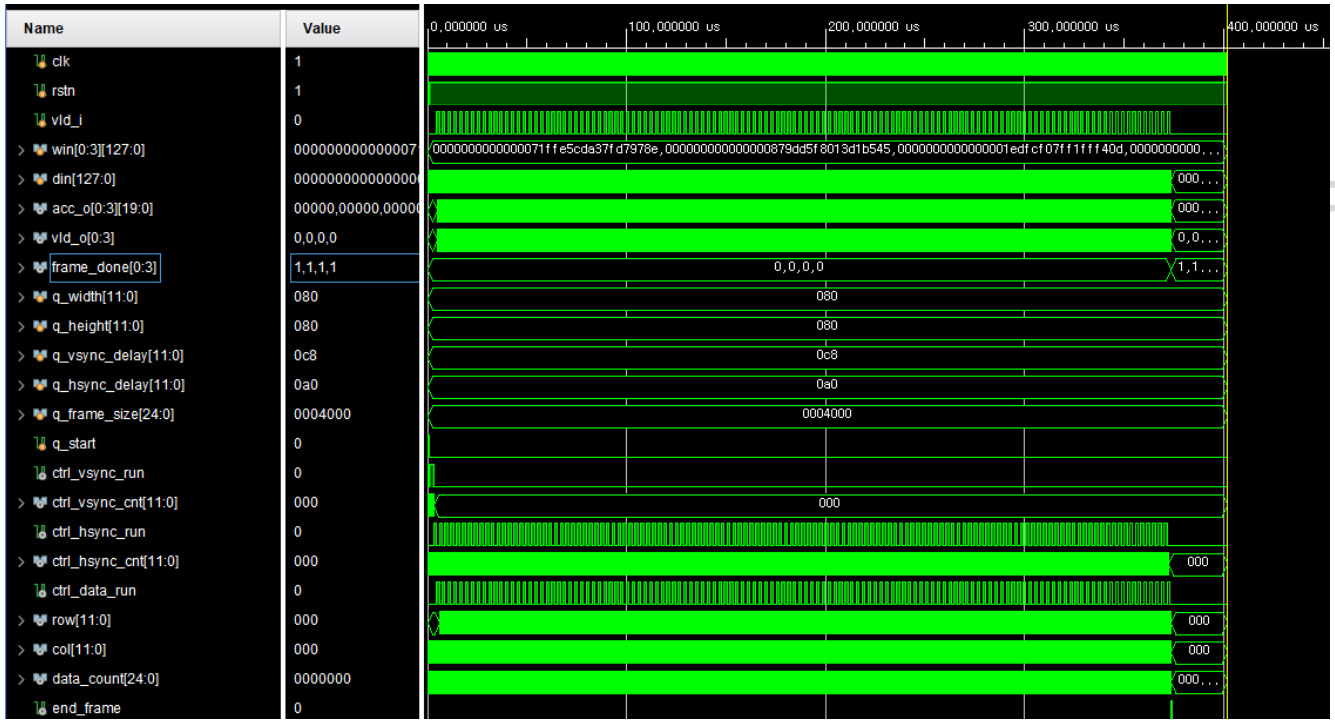
Generate din, vld_i

```
201     // Generate din
202     wire is_first_row = (row == 0) ? 1'b1: 1'b0;
203     wire is_last_row  = (row == HEIGHT-1) ? 1'b1: 1'b0;
204     wire is_first_col = (col == 0) ? 1'b1: 1'b0;
205     wire is_last_col  = (col == WIDTH-1) ? 1'b1 : 1'b0;
206
207     always@(*) begin
208         din = 128'd0;
209         vld_i = 1'b0;
210         if(ctrl_data_run) begin
211             vld_i = 1'b1;
212             din[ 7: 0] = (is_first_row | is_first_col) ? 8'd0 : in_img[(row-1) * WIDTH + (col-1)];
213             din[15: 8] = (is_first_row              ) ? 8'd0 : in_img[(row-1) * WIDTH +  col   ];
214             din[23:16] = (is_first_row | is_last_col ) ? 8'd0 : in_img[(row-1) * WIDTH + (col+1)];
215             din[31:24] = (              is_first_col) ? 8'd0 : in_img[ row    * WIDTH + (col-1)];
216             din[39:32] =                               in_img[ row    * WIDTH +  col   ];
217             din[47:40] = (              is_last_col ) ? 8'd0 : in_img[ row    * WIDTH + (col+1)];
218             din[55:48] = (is_last_row | is_first_col ) ? 8'd0 : in_img[(row+1) * WIDTH + (col-1)];
219             din[63:56] = (is_last_row | is_first_col ) ? 8'd0 : in_img[(row+1) * WIDTH + (col-1)];
220             din[71:64] = (is_last_row | is_first_col ) ? 8'd0 : in_img[(row+1) * WIDTH + (col+1)];
221         end
222     end
```

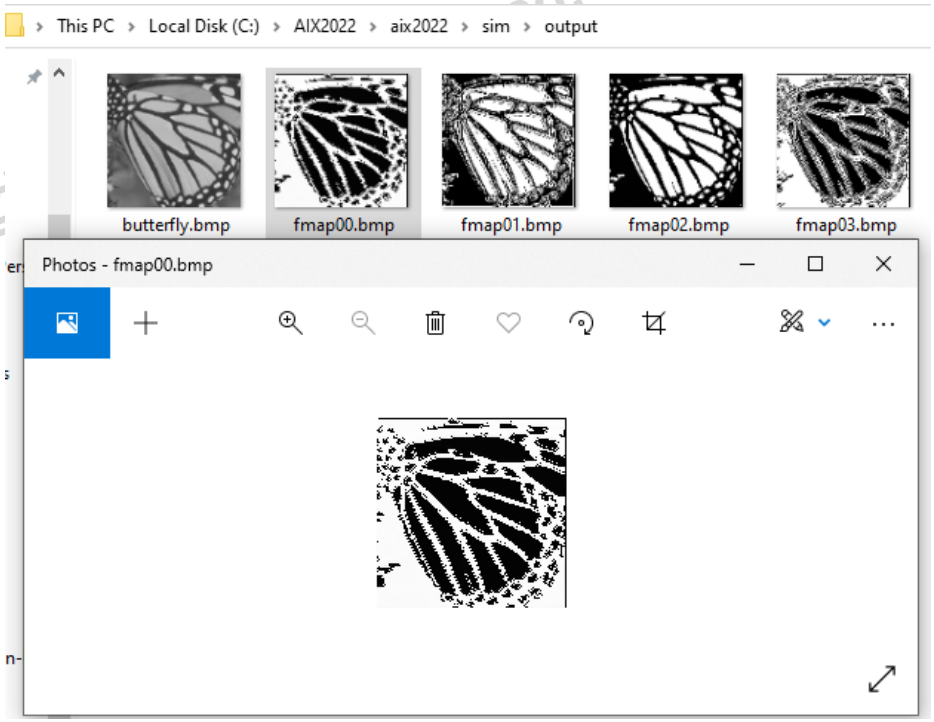
Do computation when ctrl_data_run = 1

Verification

- Do simulation with time = 400us
- The output images are stored at the predefined directories
  - Update directories based on your environment



Waveform



Feature maps/images

Objective

- AXI, DMA
  - Briefly understand the concept of bus
  - Understand the usage of dma_axi_rd, dma_axi_wr

- Top system
  - Input loader
  - On-chip buffers
  - MACs
  - Outputs

Road map

Review

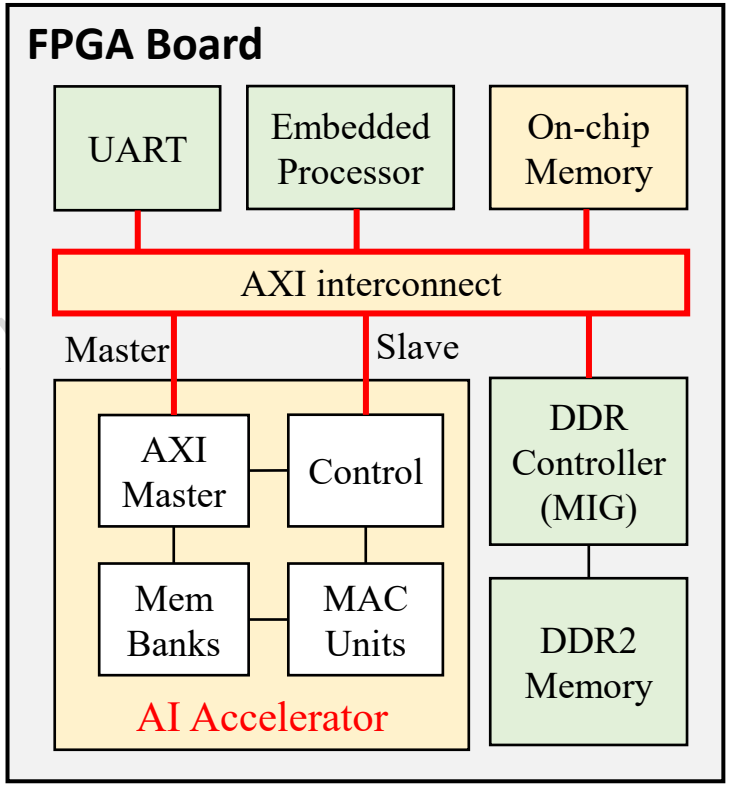
DMA

Top system

Bus

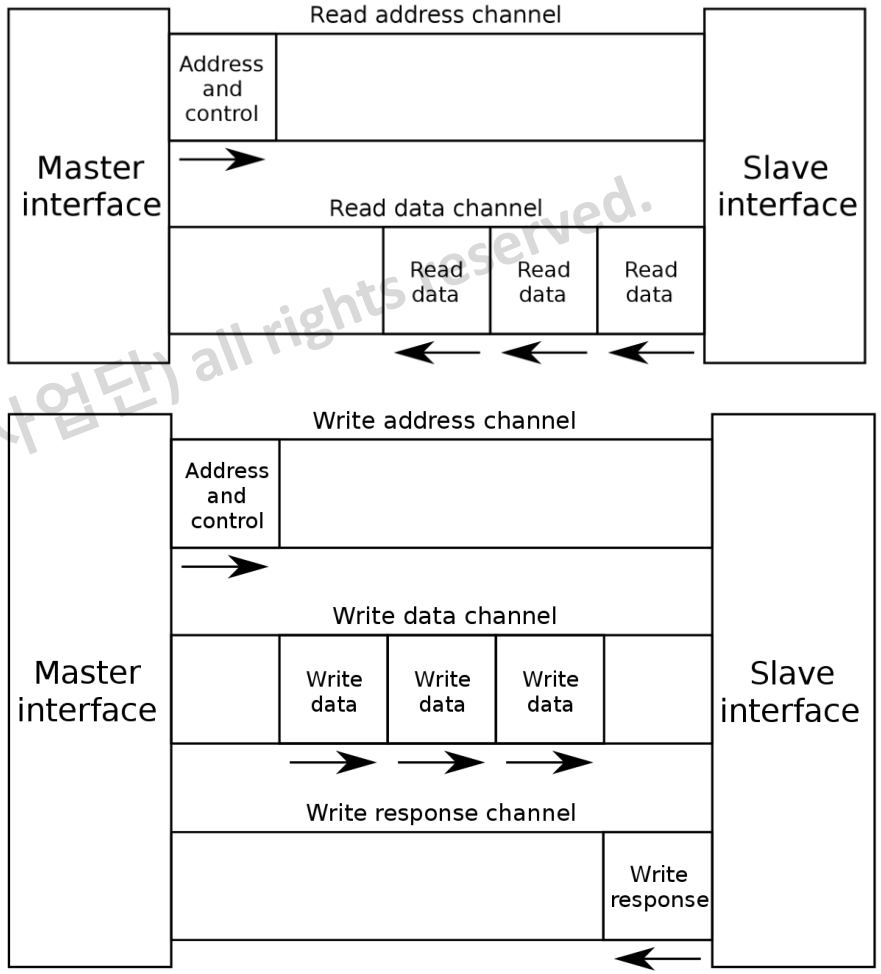
- An SoC system has various components (e.g., CPU, memory, IO, and user-defined IPs) connected via a bus

- The **Advanced eXtensible Interface** (**AXI**),
  - Part of the ARM Advanced Microcontroller Bus Architecture 3 (AXI3) and 4 (AXI4) specifications
  - A parallel high-performance, synchronous, high-frequency, multi-master, multi-slave communication interface, mainly designed for on-chip communication.



Read/Write channels

- In the AXI specification, five channels are
  - Read Address channel (AR)
  - Read Data channel (R)
  - Write Address channel (AW)
  - Write Data channel (W)
  - Write Response channel (B)
- Each channel is independent from each other and has its own couple of xVALID/xREADY signals

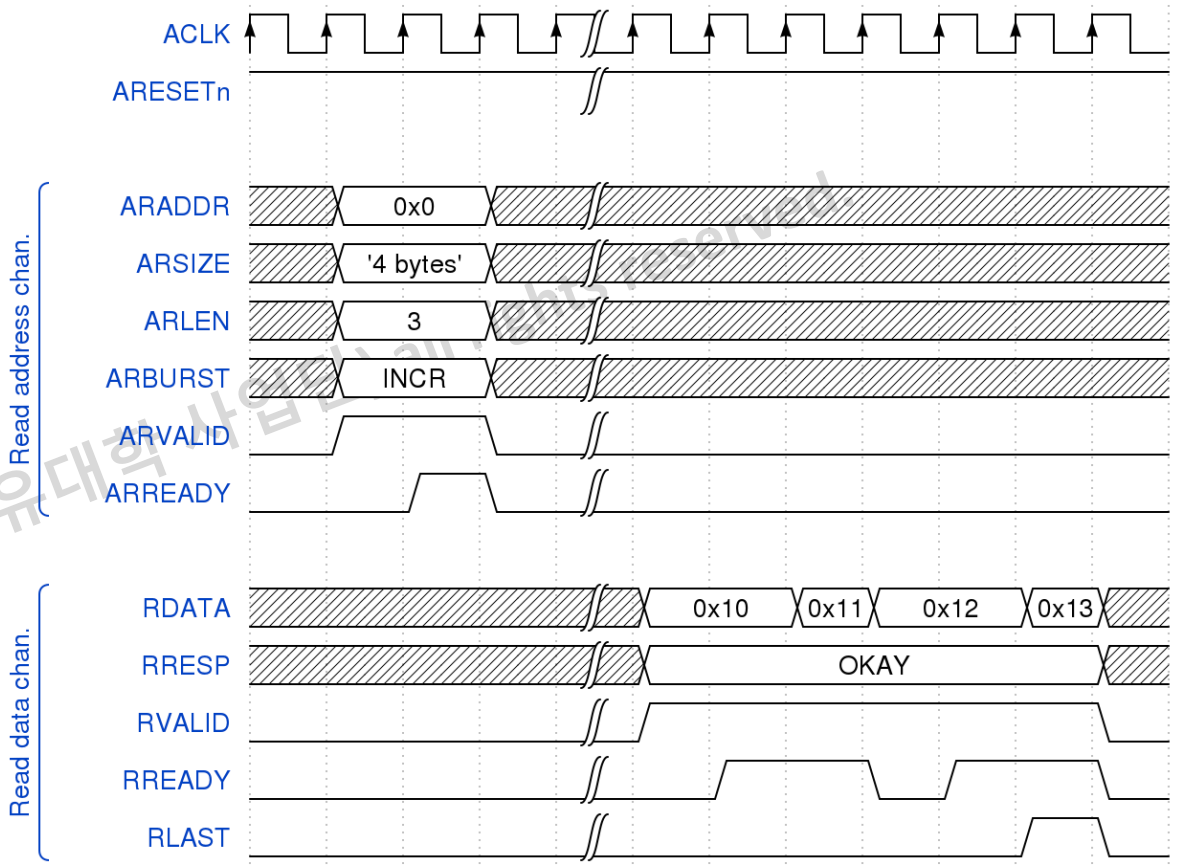


https://en.wikipedia.org/wiki/Advanced_eXtensible_Interface

Read transaction: Burst mode

- To start a read transaction, the master has to provide on the Read address channel:
  - the start address on ARADDR
  - the burst type, either FIXED, INCR or WRAP, on ARBURST (if present)
  - the burst length on ARLEN (if present).

- After the usual ARVALID/ARREADY handshake, the slave has to provide on the Read data channel:
  - the data corresponding to the specified address(es) on RDATA
  - the status of each beat on RRESP

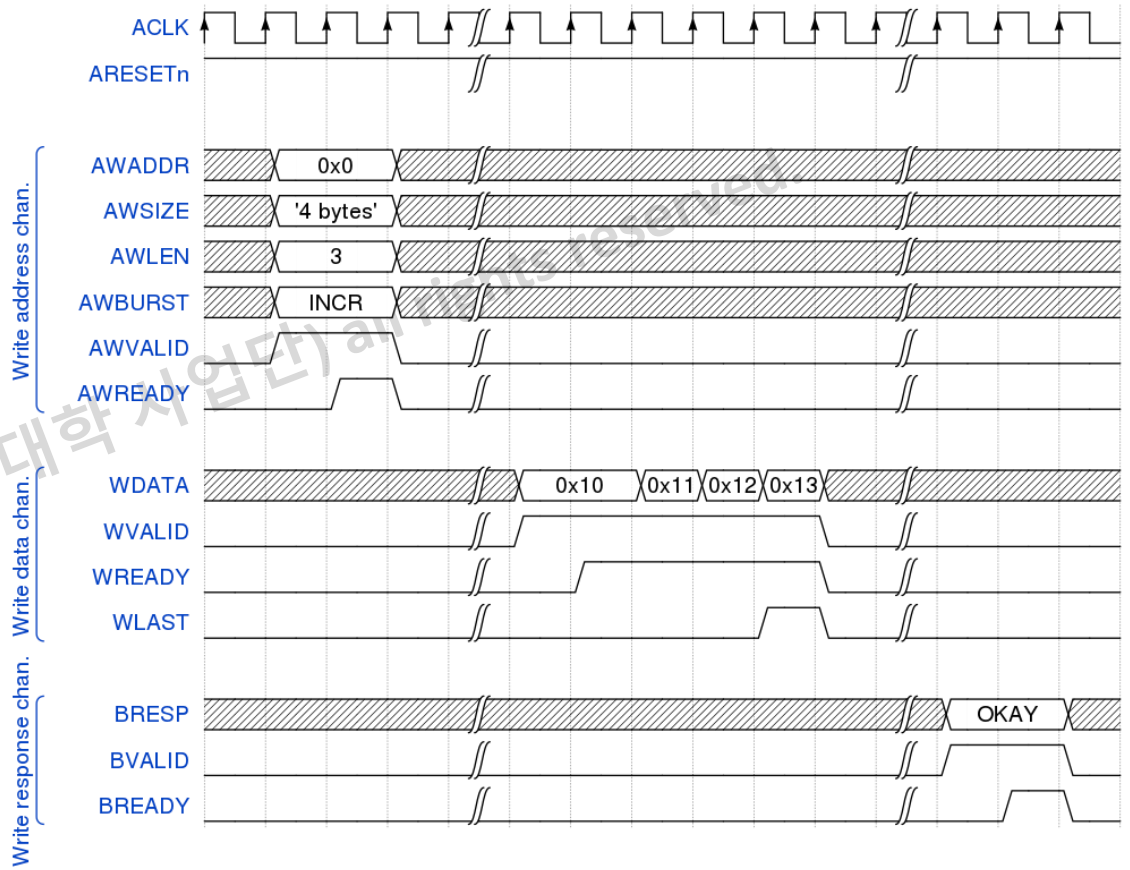


https://en.wikipedia.org/wiki/Advanced_eXtensible_Interface

Write transaction: Burst mode

- The address information is provided over the Write address channel:
  - the start address has to be provided on AWADDR
  - the burst type, either FIXED, INCR or WRAP, on AWBURST (if present)
  - the burst length on AWLEN (if present)

- A master has also to provide the data related to the specified address(es) on the Write data channel:
  - the data on WDATA
  - the "strobe" bits on WSTRB (if present), which conditionally mark the individual WDATA bytes as "valid" or "invalid"



https://en.wikipedia.org/wiki/Advanced_eXtensible_Interface

line_loader.v - Structure

Use DMA to load an input image (e.g., butterfly_08bit.hex) to a buffer

- line_loader
  - u_dma_rd ——— *// Module for DMA read operation*
  - u_dma_wr ——— *// Module for DMA write operation*
  - input_buffer ——— *// BRAM buffer; not used*

line_loader.v - u_dma_rd

```
axi_dma_rd #(
    .BITS_TRANS(18),
    .OUT_BITS_TRANS(13),
    .AXI_WIDTH_USER(1),
    .AXI_WIDTH_ID(AXI_WIDTH_ID),
    .AXI_WIDTH_AD(AXI_WIDTH_AD),
    .AXI_WIDTH_DA(AXI_WIDTH_DA),
    .AXI_WIDTH_DS(AXI_WIDTH_DS)
)
u_dma_rd (
    .M_ARVALID(m_axi_memory_bus_ARVALID),    // address/control valid handshake
    .M_ARREADY(m_axi_memory_bus_ARREADY),    // Read addr ready
    .M_ARADDR(m_axi_memory_bus_ARADDR),     // Address Read
    .M_ARID(m_axi_memory_bus_ARID),       // Read addr ID
    .M_ARLEN(m_axi_memory_bus_ARLEN),      // Transfer length
    .M_ARSIZE(m_axi_memory_bus_ARSIZE),     // Transfer width
    .M_ARBURST(m_axi_memory_bus_ARBURST),    // Burst type
    .M_ARLOCK(m_axi_memory_bus_ARLOCK),     // Atomic access information
    .M_ARCACHE(m_axi_memory_bus_ARCACHE),    // Cachable/bufferable infor
    .M_ARPROT(m_axi_memory_bus_ARPROT),     // Protection info
    .M_ARQOS(m_axi_memory_bus_ARQOS),      // Quality of Service
    .M_ARREGION(m_axi_memory_bus_ARREGION),   // Region signaling
    .M_ARUSER(m_axi_memory_bus_ARUSER),     // User defined signal

    //Read data channel
    .M_RVALID(m_axi_memory_bus_RVALID),     // Read data valid
    .M_RREADY(m_axi_memory_bus_RREADY),     // Read data ready (to Slave)
    .M_RDATA(m_axi_memory_bus_RDATA),      // Read data bus
    .M_RLAST(m_axi_memory_bus_RLAST),      // Last beat of a burst transfer
    .M_RID(m_axi_memory_bus_RID),        // Read ID
    .M_RUSER(m_axi_memory_bus_RUSER),      // User defined signal
    .M_RRESP(m_axi_memory_bus_RRESP),      // Read response

    //Functional Ports
    .start_dma(start_dma_input_read),
    .num_trans(num_trans), //Number of 32-bit words transferred
    .start_addr(start_addr),
    .data_o(data_o),
    .data_vld_o(data_vld_o),
    .data_cnt_o(data_cnt_o),
    .done_o(done_o),

    //Global signals
    .clk(ap_clk),
    .rstn(ap_rst_n)
);
```

AXI ports; *just connect them appropriately!*

Functional ports (user interface)

line_loader.v - u_dma_rd

- Functional Ports of axi_dma_rd
  - Input
    - start_dma *// start the DMA read operation*
    - num_trans *// number of transfer*
    - start_addr *// the address from which to start the read operation*
  - Output
    - data_o *// read data*
    - data_vld_o *// valid signal*
    - data_cnt_o *// data count*
    - done_o *// done*

line_loader.v - u_dma_wr

```
axi_dma_wr #(
    .BITS_TRANS(18),
    .OUT_BITS_TRANS(13),
    .AXI_WIDTH_USER(1),
    .AXI_WIDTH_ID(AXI_WIDTH_ID),
    .AXI_WIDTH_AD(AXI_WIDTH_AD),
    .AXI_WIDTH_DA(AXI_WIDTH_DA),
    .AXI_WIDTH_DS(AXI_WIDTH_DS)
)
u_dma_wr (
    .M_AWVALID(m_axi_memory_bus_AWVALID),     // address/control valid handshake
    .M_AWADDR(m_axi_memory_bus_AWADDR),      // Address Write
    .M_AWREADY(m_axi_memory_bus_AWREADY),
    .M_AWID(m_axi_memory_bus_AWID),         // Address ID
    .M_AWLEN(m_axi_memory_bus_AWLEN),       // Transfer length
    .M_AWSIZE(m_axi_memory_bus_AWSIZE),      // Transfer width
    .M_AWBURST(m_axi_memory_bus_AWBURST),     // Burst type
    .M_AWLOCK(m_axi_memory_bus_AWLOCK),      // Atomic access information
    .M_AWCACHE(m_axi_memory_bus_AWCACHE),     // Cachable/bufferable infor
    .M_AWPROT(m_axi_memory_bus_AWPROT),      // Protection info
    .M_AWQOS(m_axi_memory_bus_AWQOS),
    .M_AWREGION(m_axi_memory_bus_AWREGION),
    .M_AWUSER(m_axi_memory_bus_AWUSER),

    //Write data channel
    .M_WVALID(m_axi_memory_bus_WVALID),      // Write data valid
    .M_WREADY(m_axi_memory_bus_WREADY),      // Write data ready
    .M_WDATA(m_axi_memory_bus_WDATA),       // Write Data bus
    .M_WSTRB(m_axi_memory_bus_WSTRB),       // Write Data byte lane strobes
    .M_WLAST(m_axi_memory_bus_WLAST),       // Last beat of a burst transfer
    .M_WID(m_axi_memory_bus_WID),          // Write ID
    .M_WUSER(m_axi_memory_bus_WUSER),

    //Write response channel
    .M_BVALID(m_axi_memory_bus_BVALID),      // Response info valid
    .M_BREADY(m_axi_memory_bus_BREADY),      // Response info ready (to slave)
    .M_BRESP(m_axi_memory_bus_BRESP),       // Buffered write response
    .M_BID(m_axi_memory_bus_BID),          // buffered response ID
    .M_BUSER(m_axi_memory_bus_BUSER),

    //User interface
    .ap_start(ap_start),
    .ap_done(ap_done),
    .num_trans(num_trans),
    .mem_start_addr(start_addr),
    //buff_start_addr,
    .indata(data_tbw),
    .indata_req_o(indata_req_o),
    .buff_valid(1'b1),
    .fail_check(fail_check),
    //User signals
    .clk(ap_clk),
    .rstn(ap_rst_n)
);
```

AXI ports; *just connect them appropriately!*

Functional ports (user interface)

line_loader.v - u_dma_wr

- Functional Ports of axi_dma_wr
  - Input
    - ap_start *// start the DMA write operation*
    - num_trans *// number of transfer*
    - mem_start_addr *// the address from which to start the write operation*
    - buff_valid *// buffer valid signal*
  - Output
    - ap_done *// done*
    - indata *// data to be written*
    - indata_req_o *// data request*
    - fail_check *// indicates write fail*

line_loader.v - FSM

```
case (state)
    IDLE: begin
        start_dma_input_read <= 1'b0;
        ap_start <= 1'b0;
        num_trans <= 64;
        seq_state <= 1'b0;
        line_count <= 0;
        load_done <= 1'b0;
        if (load_start) state <= READ_P1;
    end
    READ_P1: begin
        start_dma_input_read <= 1'b1;
        line_count <= line_count + 1;
        start_addr <= start_addr + WIDTH;
        ena_reg <= 1'b1;
        wea_reg <= 1'b1;
        state <= READ_P2;
        if (line_count == HEIGHT) begin
            load_done <= 1'b1;
            state <= IDLE;
        end
    end
    READ_P2: begin
        start_dma_input_read <= 1'b0;
        if (data_vld_o) begin
            img_reg = (img_reg >> 16);
            img_reg[8*WIDTH*HEIGHT-1-:16] = data_o;
        end
        if (done_o) begin
            state <= READ_P1;
        end
    end
```

Wait until load_start signal is received
**num_trans is fixed to 64; thus, 16 bit*64 = 1024 bits are transferred for each DMA read*

Set line_count, start_addr and start the DMA read; Go to READ_P2 state

If all lines are loaded, set load_done signal to HIGH and go back to IDLE state

Update img_reg with the new pixel

Go back to READ_P1 state when the entire line is transferred

Road map

Review

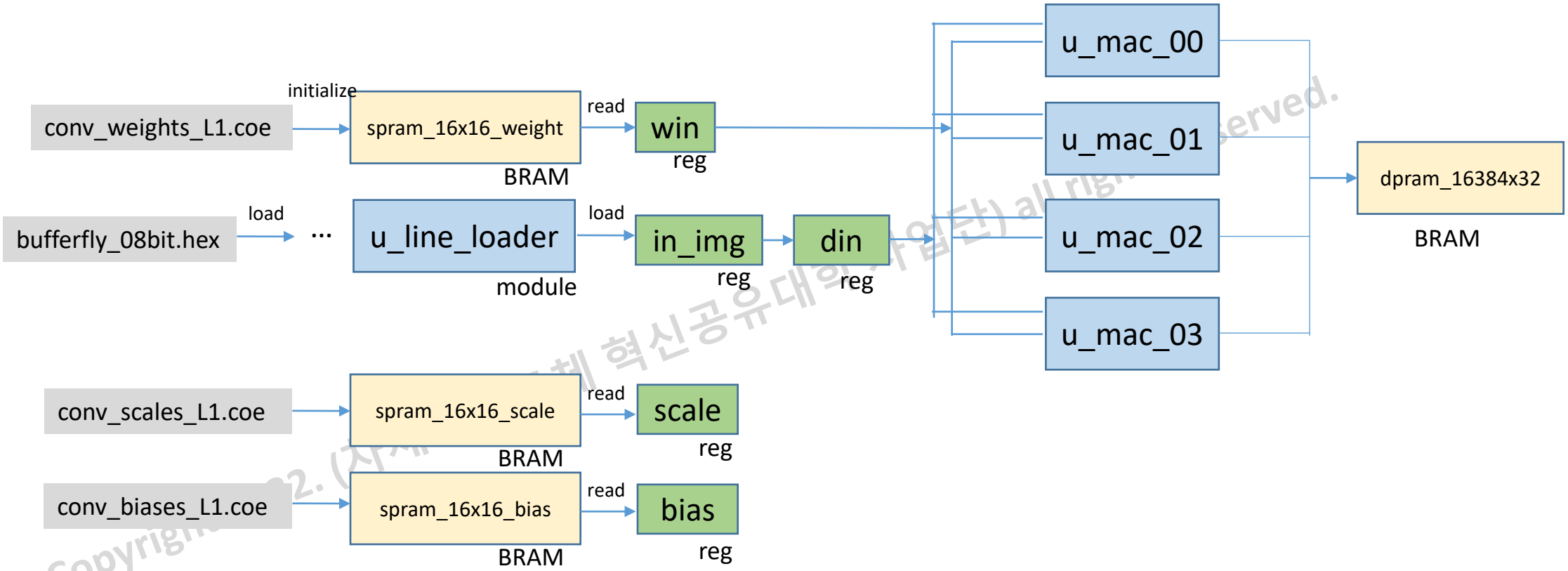
DMA

Top system

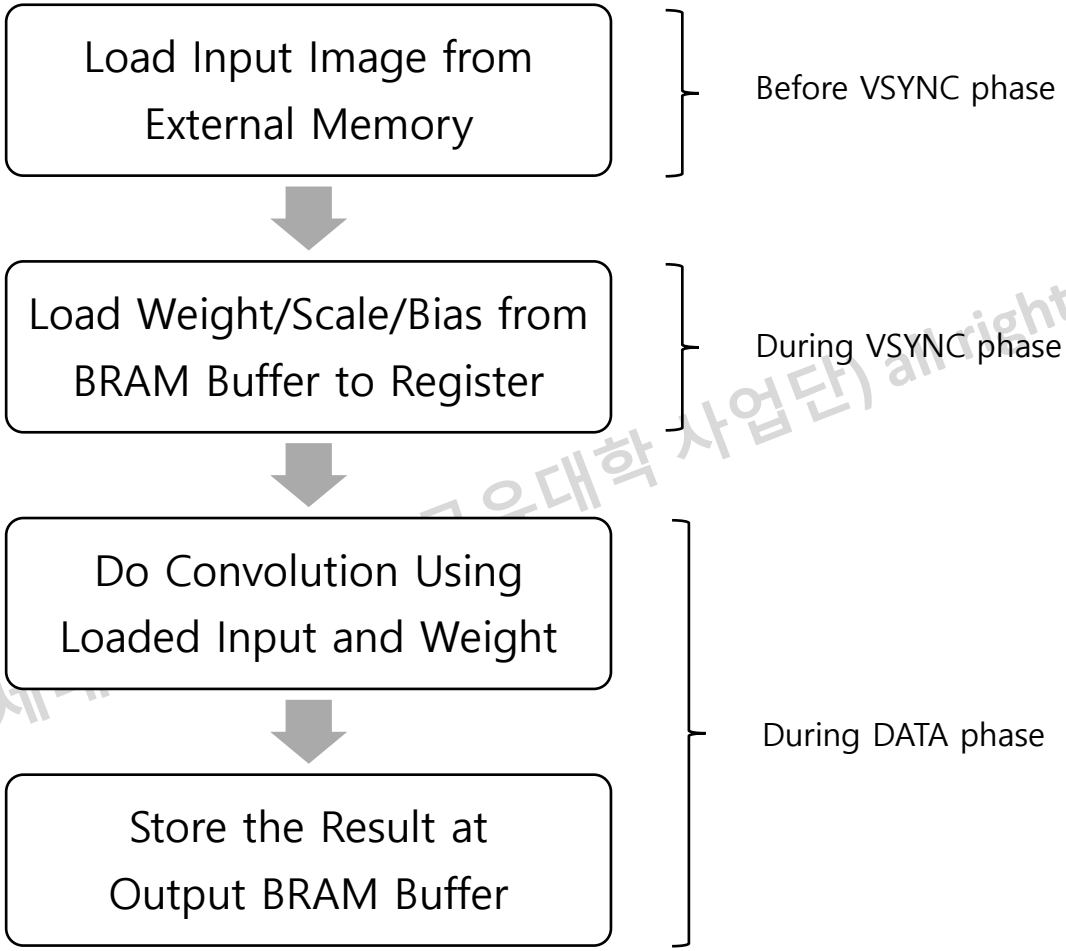
cnv_tb.v - Structure

- cnv_tb
  - u_axi_ext_mem_if_input — *// AXI Slave for External Memory (Input)*
  - u_ext_mem_input — *// SRAM (Input)*
  - u_axi_ext_mem_if_output — *// AXI Slave for External Memory (Output)*
  - u_ext_mem_output — *// SRAM (Output)*
  - u_line_loader — *// Image loader; loads the input image(line-by-line) via DMA*
  - u_cnn_ctrl — *// Controller*
  - u_buf_weight — *// BRAM buffer for weight*
  - u_buf_bias — *// BRAM buffer for bias*
  - u_buf_scale — *// BRAM buffer for scale*
  - u_buf_fmap — *// BRAM buffer for output*
  - u_mac_00, u_mac_01, u_mac_02, u_mac_03 — *// MAC modules for convolution operation*

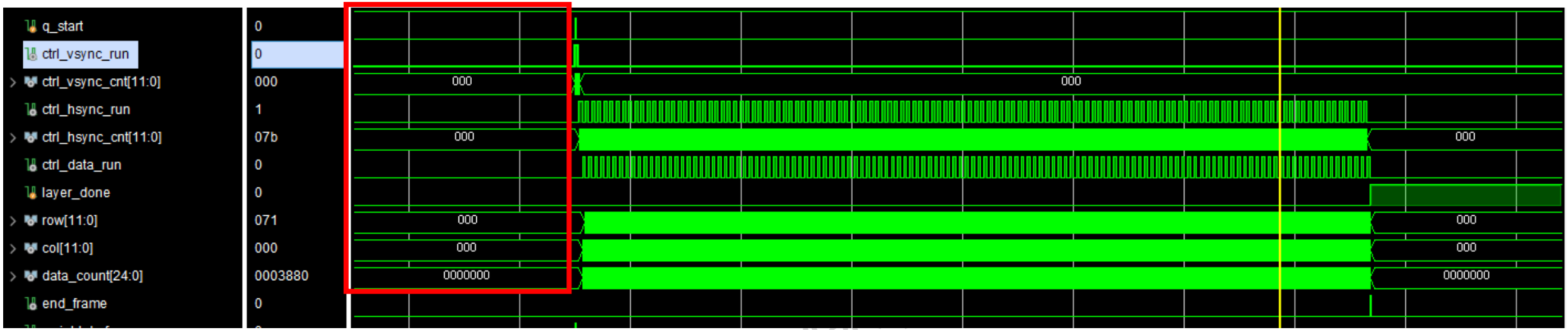
Overall data flow



Overall Flow



cnv_tb.v – overall flow



Load Input Image from External Memory

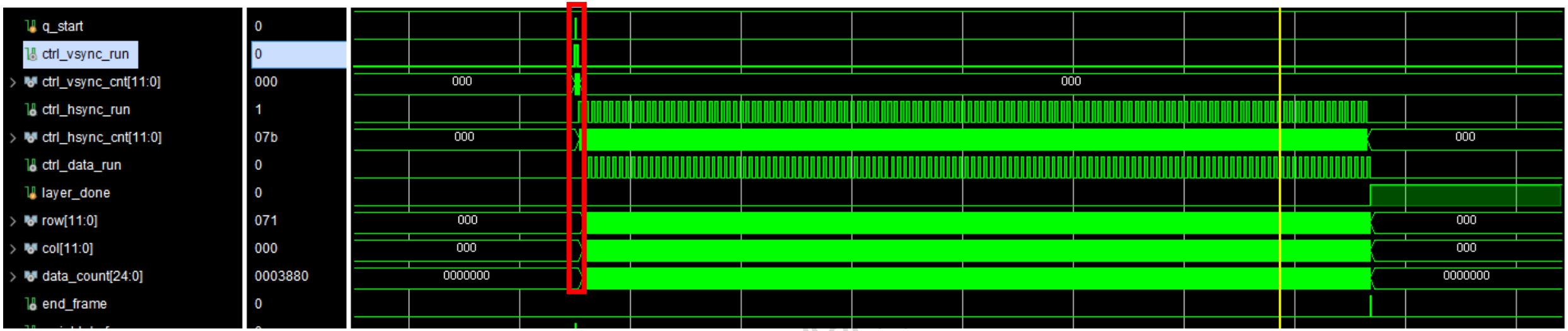
Do Convolution Using Loaded Input and Weight

Read the Output BRAM buffer for checking

Load Weight/Scale/Bias from BRAM Buffer to Register

Store the Result at Output BRAM Buffer

cnv_tb.v – overall flow



Load Input Image from External Memory

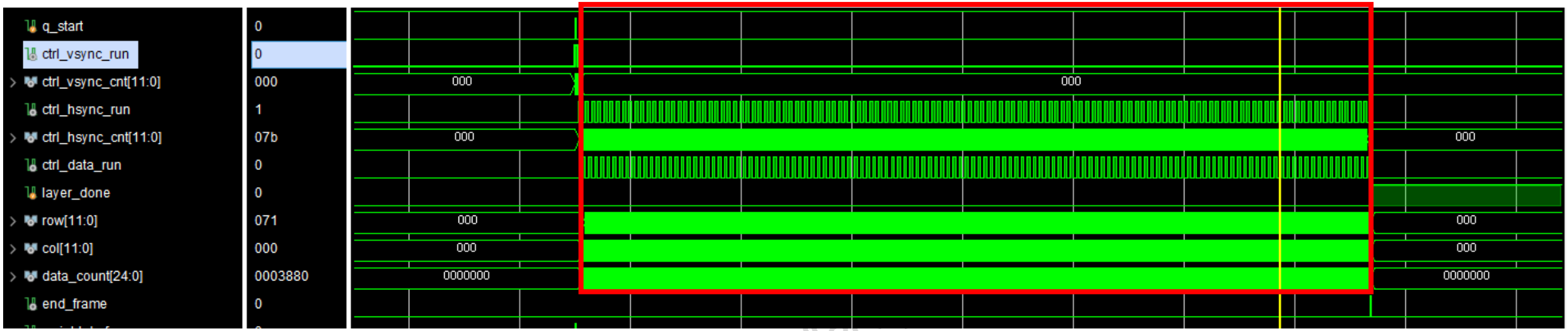
Do Convolution Using Loaded Input and Weight

Read the Output BRAM buffer for checking

Load Weight/Scale/Bias from BRAM Buffer to Register

Store the Result at Output BRAM Buffer

cnv_tb.v – overall flow



Load Input Image from External Memory

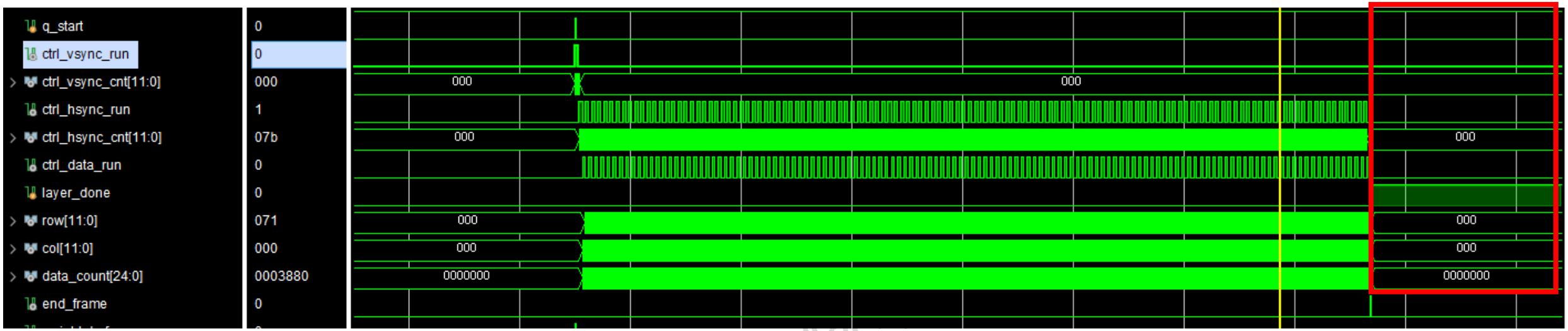
Do Convolution Using Loaded Input and Weight

Read the Output BRAM buffer for checking

Load Weight/Scale/Bias from BRAM Buffer to Register

Store the Result at Output BRAM Buffer

cnv_tb.v – overall flow



Load Input Image from External Memory

Do Convolution Using Loaded Input and Weight

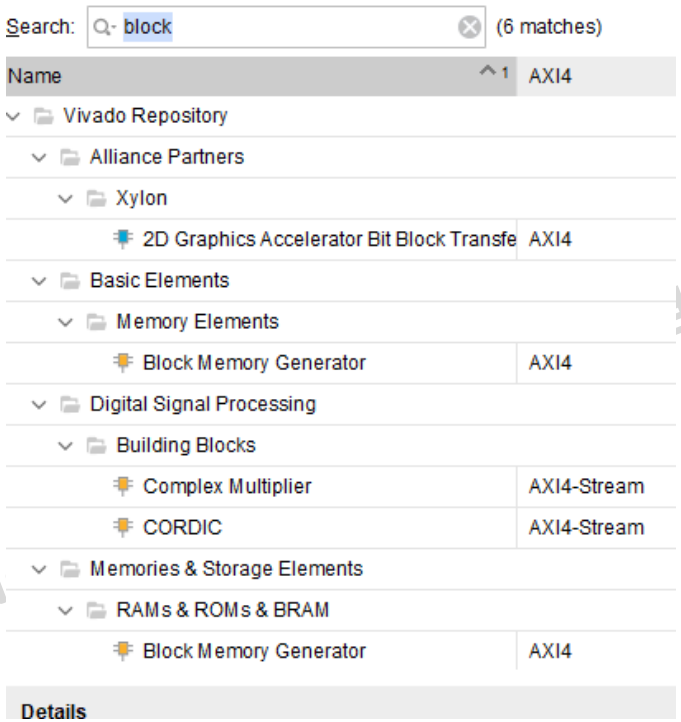
Read the Output BRAM buffer for checking

Load Weight/Scale/Bias from BRAM Buffer to Register

Store the Result at Output BRAM Buffer

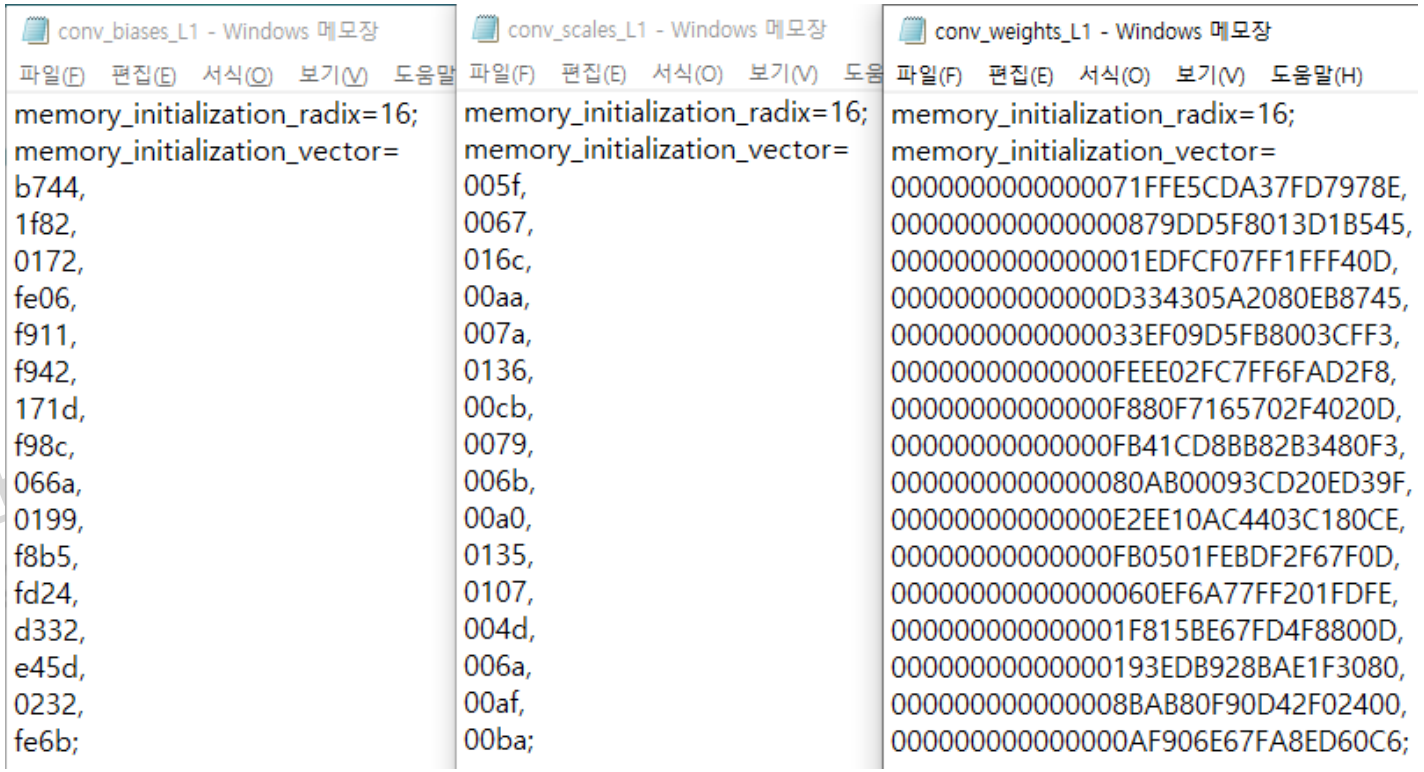
weight/bias/scale buffers - spram

- Three different spram_wrapper
- You have to add BRAM IPs using IP catalog



weight/bias/scale files

- Initialized files for weights, biases and scales
  - Word size : 16bits for “bias” and “scale”, 128bits for “weight”
  - Number of words : 16



conv_biases_L1 - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말

```
memory_initialization_radix=16;
memory_initialization_vector=
b744,
1f82,
0172,
fe06,
f911,
f942,
171d,
f98c,
066a,
0199,
f8b5,
fd24,
d332,
e45d,
0232,
fe6b;
```

conv_scales_L1 - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움

```
memory_initialization_radix=16;
memory_initialization_vector=
005f,
0067,
016c,
00aa,
007a,
0136,
00cb,
0079,
006b,
00a0,
0135,
0107,
004d,
006a,
00af,
00ba;
```

conv_weights_L1 - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
memory_initialization_radix=16;
memory_initialization_vector=
0000000000000071FFE5CDA37FD7978E,
0000000000000879DD5F8013D1B545,
000000000000001EDFCF07FF1FFF40D,
00000000000000D334305A2080EB8745,
000000000000033EF09D5FB8003CFF3,
00000000000000FEEE02FC7FF6FAD2F8,
00000000000000F880F7165702F4020D,
00000000000000FB41CD8BB82B3480F3,
000000000000080AB00093CD20ED39F,
00000000000000E2EE10AC4403C180CE,
00000000000000FB0501FEBDF2F67F0D,
000000000000060EF6A77FF201FDFE,
000000000000001F815BE67FD4F8800D,
0000000000000193EDB928BAE1F3080,
000000000000008BAB80F90D42F02400,
000000000000000AF906E67FA8ED60C6;
```

weight/bias/scale buffers

- Define three spram instances for weight, bias, and scale buffers
  - Parameters
    - DW, AW, DEPTH
  - Ports
    - clk, cs, addr, wdata, we, rdata

```
// Weight buffer
spram_wrapper_weight #(.DW(Ti*Wi),.AW(W_CELL),.DEPTH(N_CELL))
u_buf_weight(
    .clk (clk            ), // Clock input
    .cs  (weight_buf_en  ), // RAM enable (select)
    .addr(weight_buf_addr), // Address input(word addressing)
    .wdata (/*unused*/     ), // Data input
    .we  (weight_buf_we  ), // Write enable
    .rdata (weight_buf_dout)  // Data output
);

// Bias buffer
spram_wrapper_bias #(.DW(PARAM_BITS),.AW(W_CELL),.DEPTH(N_CELL))
u_buf_bias(
    .clk (clk            ), // Clock input
    .cs  (param_buf_en  ), // RAM enable (select)
    .addr(param_buf_addr), // Address input(word addressing)
    .wdata (/*unused*/     ), // Data input
    .we  (param_buf_we  ), // Write enable
    .rdata (param_buf_dout_bias)  // Data output
);

// Scale buffer
spram_wrapper_scale #(.DW(PARAM_BITS),.AW(W_CELL),.DEPTH(N_CELL))
u_buf_scale(
    .clk (clk            ), // Clock input
    .cs  (param_buf_en  ), // RAM enable (select)
    .addr(param_buf_addr), // Address input(word addressing)
    .wdata (/*unused*/     ), // Data input
    .we  (param_buf_we  ), // Write enable
    .rdata (param_buf_dout_scale)  // Data output
);
```

Internal singals for weight/bias/scale buffers

- Enable signals: *_buf_en, *_buf_en_d, *_buf_we
- Addresses: *_buf_addr, *_buf_addr_d, *_buf_addr_2d
- Buffer outputs: *_buf_dout_*

```
// Weight/bias/scale buffer's signals
// weight
reg                  weight_buf_en;        // primary enable
reg                  weight_buf_en_d;      // primary enable
reg                  weight_buf_we;        // primary synchronous write enable
reg [W_CELL-1:0]     weight_buf_addr;      // address for read/write
reg [W_CELL-1:0]     weight_buf_addr_d;    // 1-cycle delay address
reg [W_CELL-1:0]     weight_buf_addr_2d;   // 2-cycle delay address
wire[Ti*WI-1:0]      weight_buf_dout;      // Output for weights
// bias/scale
reg                    param_buf_en;       // primary enable
reg                    param_buf_en_d;     // primary enable
reg                    param_buf_we;       // primary synchronous write enable
reg [W_CELL_PARAM-1:0] param_buf_addr;     // address for read/write
reg [W_CELL_PARAM-1:0] param_buf_addr_d;   // 1-cycle delay address
reg [W_CELL_PARAM-1:0] param_buf_addr_2d;  // 2-cycle delay address
wire[PARAM_BITS-1:0]   param_buf_dout_bias;  // Output for biases
wire[PARAM_BITS-1:0]   param_buf_dout_scale; // Output for scales
```

Internal singals for weight/bias/scale buffers

- Assume that weight, scale and bias registers are updated during frame synchronization
  - VSYNC
  - Request address is updated from the VSYNC counter ctrl_vsync_cnt

```
// 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(ctrl_vsync_cnt < To + 1) begin // 2 cycle delay spram : To + 1, 1 cycle delay spram : To
            weight_buf_en   = 1'b1;
            weight_buf_we   = 1'b0;
            weight_buf_addr = ctrl_vsync_cnt[W_CELL-1:0];
        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 + 1) begin // 2 cycle delay spram : To + 1, 1 cycle delay spram : To
            param_buf_en   = 1'b1;
            param_buf_we   = 1'b0;
            param_buf_addr = ctrl_vsync_cnt[W_CELL-1:0];
        end
    end
end
```

Internal singals for weight/bias/scale buffers

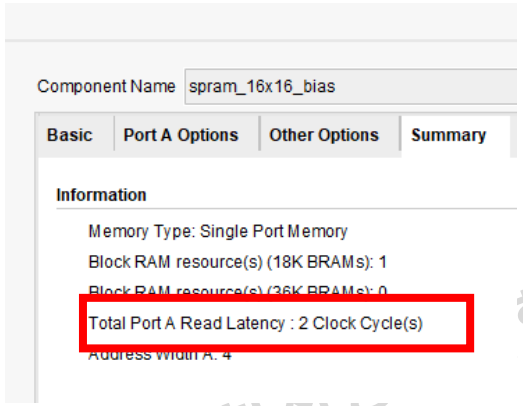
- Cycle delay signals
  - One cycle delay
    - Enable : *_buf_en_d
    - Address : *_buf_addr_d
  - Two cycle delay
    - Enable : *_buf_en_2d
    - Address : *_buf_addr_2d
- Win, scale, bias registers are updated when outputs from their buffers are read out.

```
// one-cycle, two_cycle delay
always@(posedge clk, negedge rstn)begin
    if(~rstn) begin
        weight_buf_en_d   <= 1'b0;
        weight_buf_en_2d   <= 1'b0;
        weight_buf_addr_d <= {W_CELL{1'b0}};
        weight_buf_addr_2d <= {W_CELL{1'b0}};
        param_buf_en_d    <= 1'b0;
        param_buf_en_2d       <= 1'b0;
        param_buf_addr_d  <= {W_CELL{1'b0}};
        param_buf_addr_2d  <= {W_CELL{1'b0}};
    end
    else begin
        weight_buf_en_d   <= weight_buf_en;
        weight_buf_en_2d   <= weight_buf_en_d;
        weight_buf_addr_d <= weight_buf_addr;
        weight_buf_addr_2d <= weight_buf_addr_d;
        param_buf_en_d    <= param_buf_en;
        param_buf_en_2d       <= param_buf_en_d;
        param_buf_addr_d  <= param_buf_addr;
        param_buf_addr_2d  <= param_buf_addr_d;
    end
end
```

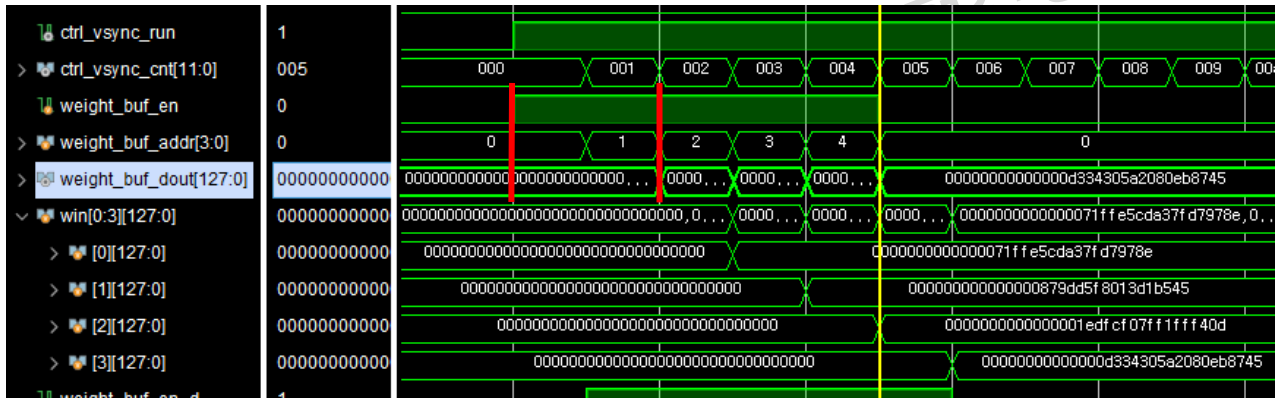
```
// two_cycle delay
always@(posedge clk, negedge rstn)begin
    if(~rstn) begin
        for(ch_idx = 0; ch_idx <To; ch_idx=ch_idx+1) begin
            win[ch_idx]  <= {(Ti*WI){1'b0}};
            scale[ch_idx] <= {PARAM_BITS{1'b0}};
            bias[ch_idx] <= {PARAM_BITS{1'b0}};
        end
    end
    else begin
        // Weight
        if(weight_buf_en_2d)
            win[weight_buf_addr_2d] <= weight_buf_dout;
        // Scale/bias
        if(param_buf_en_2d) begin
            bias[param_buf_addr_2d] <= param_buf_dout_bias;
            scale[param_buf_addr_2d] <= param_buf_dout_scale;
        end
    end
end
```

Internal singals for weight/bias/scale buffers

- Weight_buf_en, param_buf_en signals should be high until 'ctrl_vsync_run < To + 1' because of 2 cycle delay for read BRAM IP



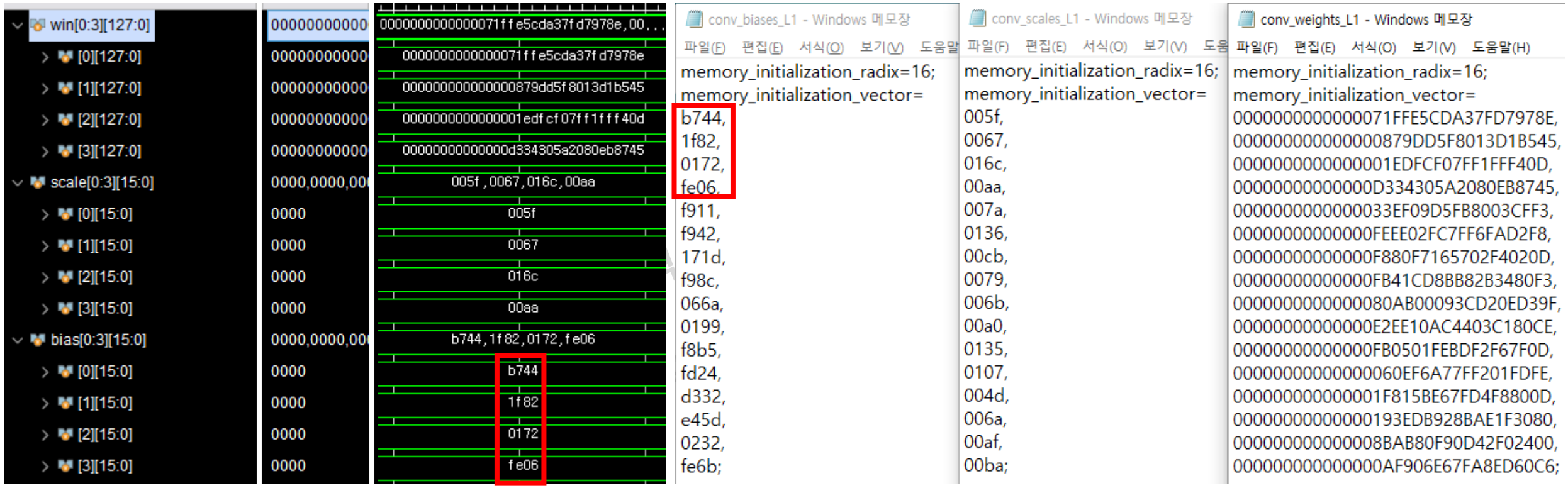
2 cycle delay for read




```
// 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(ctrl_vsync_cnt < To + 1) begin // 2 cycle delay spram : To + 1, 1 cycle delay spram : To
            weight_buf_en   = 1'b1;
            weight_buf_we   = 1'b0;
            weight_buf_addr = ctrl_vsync_cnt[W_CELL-1:0];
        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 + 1) begin // 2 cycle delay spram : To + 1, 1 cycle delay spram : To
            param_buf_en   = 1'b1;
            param_buf_we   = 1'b0;
            param_buf_addr = ctrl_vsync_cnt[W_CELL-1:0];
        end
    end
end
```

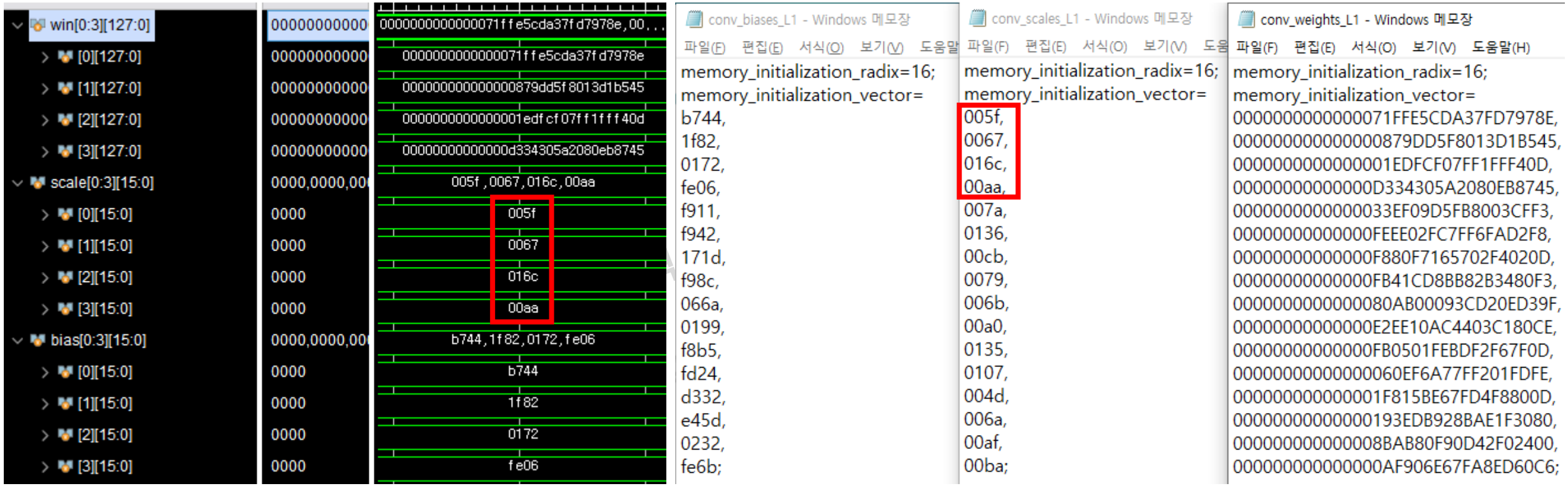
weight/bias/scale files

- win, scale, bias registers store the same values as the data in COE file.



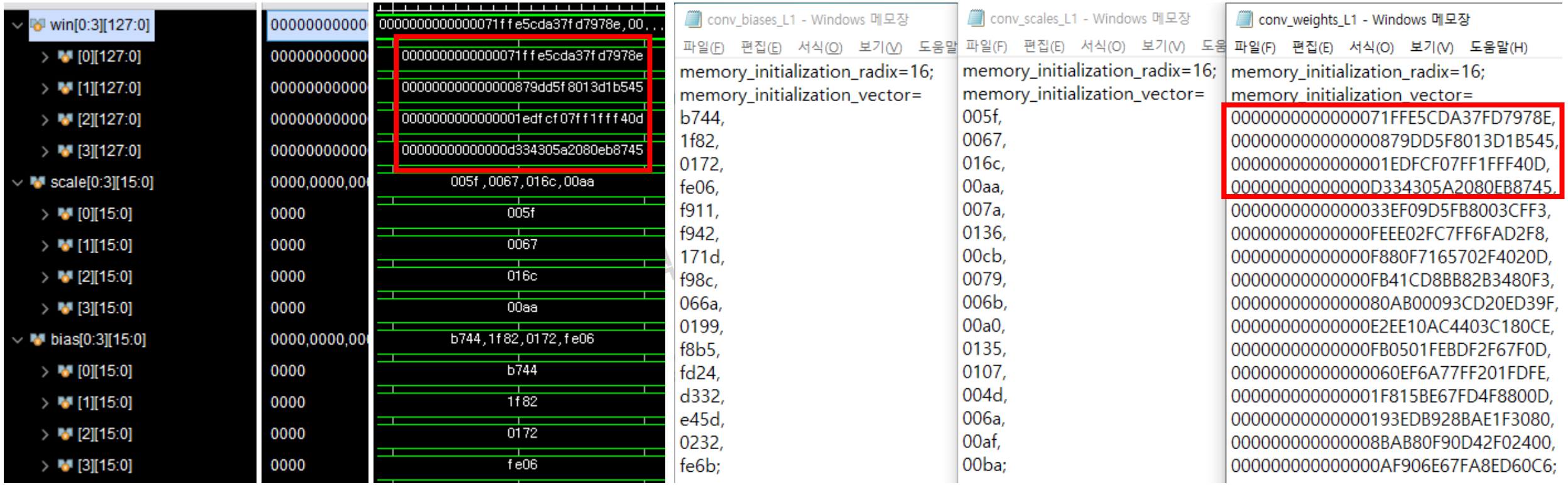
weight/bias/scale files

- win, scale, bias registers store the same values as the data in COE file.



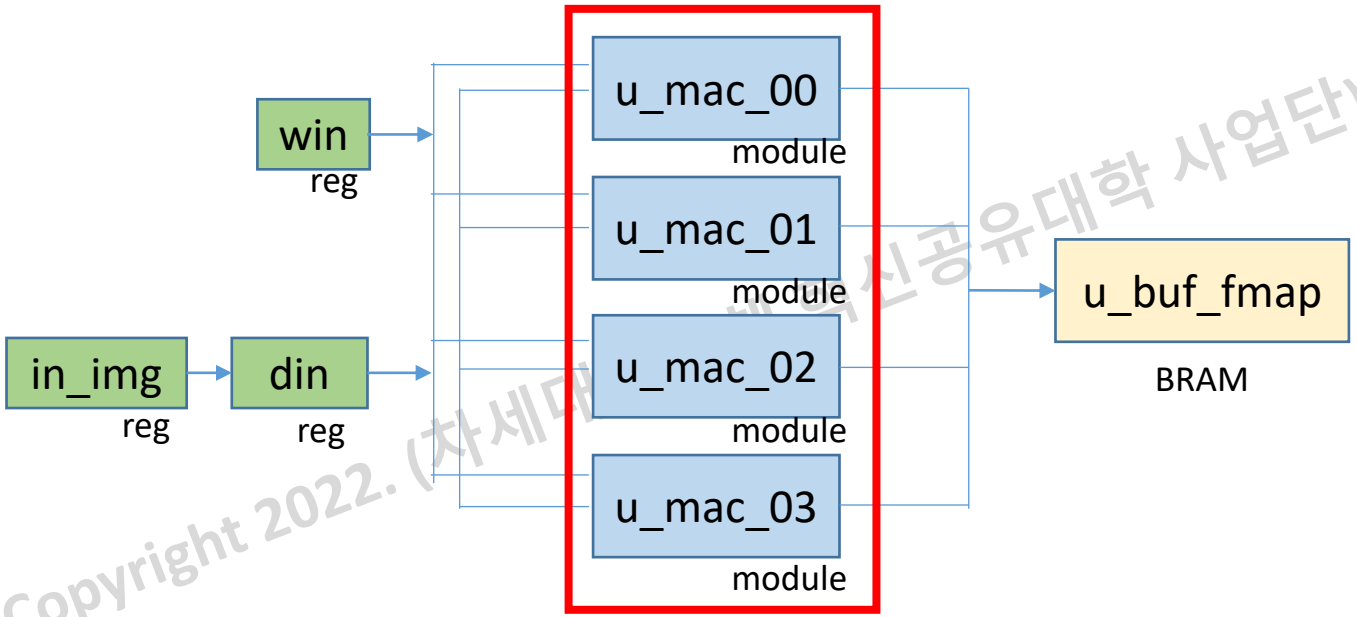
weight/bias/scale files

- win, scale, bias registers store the same values as the data in COE file.



mac.v

- Use only 4 macs
- Convolutions for only 4 filters (weights)




```
//------------------------------------------------------------------------------
// DUT: MACs
//------------------------------------------------------------------------------

mac u_mac_00(
.clk(clk),
.rstn(rstn),
.vld_i(vld_i),
.win(win[0]),
.din(din),
.acc_o(acc_o[0]),
.vld_o(vld_o[0])
);
mac u_mac_01(
.clk(clk),
.rstn(rstn),
.vld_i(vld_i),
.win(win[1]),
.din(din),
.acc_o(acc_o[1]),
.vld_o(vld_o[1])
);
mac u_mac_02(
.clk(clk),
.rstn(rstn),
.vld_i(vld_i),
.win(win[2]),
.din(din),
.acc_o(acc_o[2]),
.vld_o(vld_o[2])
);
mac u_mac_03(
.clk(clk),
.rstn(rstn),
.vld_i(vld_i),
.win(win[3]),
.din(din),
.acc_o(acc_o[3]),
.vld_o(vld_o[3])
);
```

cnv_tb.v - din

Row	Col 0	1	2	…	125	126	127
0	2a	45	5b	…	45	45	44
1	69	2a	38	…	41	41	44
	48	2b	2a	…	2c	2d	32
…	…	…	…	…	…	…	…
	41	34	4a	…	6f	6e	70
126	39	46	69	…	70	6f	71
127	3a	4c	4b	…	71	71	71

bufferfly_08bit.hex

padding

0	0	0	0	…	0	0	0	0
0	2a	45	5b	…	45	45	44	0
0	69	2a	38	…	41	41	44	0
0	48	2b	2a	…	2c	2d	32	0
…	…	…	…	…	…	…	…	…
0	41	34	4a	…	6f	6e	70	0
0	39	46	69	…	70	6f	71	0
0	3a	4c	4b	…	71	71	71	0
0	0	0	0	…	0	0	0	0

cnv_tb.v - din

row : 0 , col : 0

0	0	0	0	…	0	0	0	0
0	2a	45	5b	…	45	45	44	0
0	69	2a	38	…	41	41	44	0
0	48	2b	2a	…	2c	2d	32	0
…	…	…	…	…	…	…	…	…
0	41	34	4a	…	6f	6e	70	0
0	39	46	69	…	70	6f	71	0
0	3a	4c	4b	…	71	71	71	0
0	0	0	0	…	0	0	0	0

```
wire is_first_row = (row == 0) ? 1'b1 : 1'b0;
wire is_last_row = (row == HEIGHT-1) ? 1'b1 : 1'b0;
wire is_first_col = (col == 0) ? 1'b1 : 1'b0;
wire is_last_col = (col == WIDTH-1) ? 1'b1 : 1'b0;
```

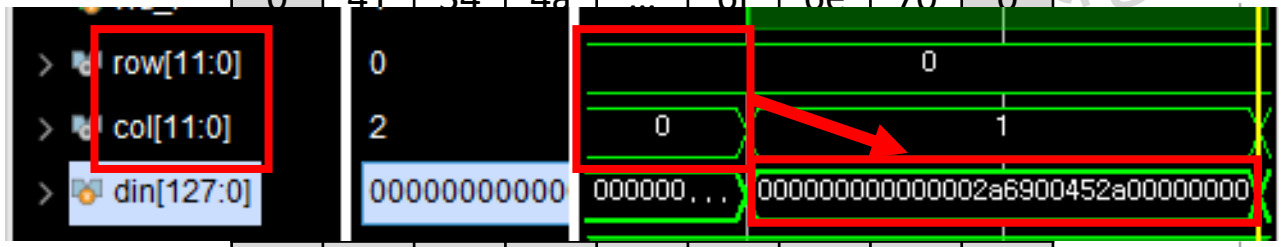
```
always @(posedge clk) begin
    if (load_done == 1'b1) begin
        q_start <= 1'b1;
        #(4*CLK_PERIOD)
            @(posedge clk)
                q_start <= 1'b0;
    end

    vld_i = 1'b0;
    if(ctrl_data_run) begin
        vld_i = 1'b1;
        din[ 7: 0] = (is_first_row | is_first_col) ? 8'd0 : in_img[(row-1) * WIDTH + (col-1)];
        din[15: 8] = (is_first_row              ) ? 8'd0 : in_img[(row-1) * WIDTH +  col   ];
        din[23:16] = (is_first_row | is_last_col ) ? 8'd0 : in_img[(row-1) * WIDTH + (col+1)];
        din[31:24] = (               is_first_col) ? 8'd0 : in_img[ row    * WIDTH + (col-1)];
        din[39:32] =                                        in_img[ row    * WIDTH +  col   ];
        din[47:40] = (               is_last_col ) ? 8'd0 : in_img[ row    * WIDTH + (col+1)];
        din[55:48] = (is_last_row | is_first_col ) ? 8'd0 : in_img[(row+1) * WIDTH + (col-1)];
        din[63:56] = (is_last_row               ) ? 8'd0 : in_img[(row+1) * WIDTH +  col   ];
        din[71:64] = (is_last_row | is_last_col ) ? 8'd0 : in_img[(row+1) * WIDTH + (col+1)];
    end
end
```

cnv_tb.v - din

row : 0 , col : 0

din

0	0	0	0	…	0	0	0	0
0	2a	45	5b	…	45	45	44	0
0	69	2a	38	…	41	41	44	0
0	48	2b	2a	…	2c	2d	32	0
…	…	…	…	…	…	…	…	…
0	41	34	4a	…	6f	6e	70	0
0	0	0	0	…	0	0	0	0




```
wire is_first_row = (row == 0) ? 1'b1 : 1'b0;
wire is_last_row = (row == HEIGHT-1) ? 1'b1 : 1'b0;
wire is_first_col = (col == 0) ? 1'b1 : 1'b0;
wire is_last_col = (col == WIDTH-1) ? 1'b1 : 1'b0;
```

```
always @(posedge clk) begin
    if (load_done == 1'b1) begin
        q_start <= 1'b1;
        #(4*CLK_PERIOD)
            @(posedge clk)
                q_start <= 1'b0;
    end

    vld_i = 1'b0;
    if(ctrl_data_run) begin
        vld_i = 1'b1;
        din[ 7: 0] = (is_first_row | is_first_col) ? 8'd0 : in_img[(row-1) * WIDTH + (col-1)];
        din[15: 8] = (is_first_row              ) ? 8'd0 : in_img[(row-1) * WIDTH +  col   ];
        din[23:16] = (is_first_row | is_last_col ) ? 8'd0 : in_img[(row-1) * WIDTH + (col+1)];
        din[31:24] = (               is_first_col) ? 8'd0 : in_img[ row    * WIDTH + (col-1)];
        din[39:32] =                                        in_img[ row    * WIDTH +  col   ];
        din[47:40] = (               is_last_col ) ? 8'd0 : in_img[ row    * WIDTH + (col+1)];
        din[55:48] = (is_last_row | is_first_col ) ? 8'd0 : in_img[(row+1) * WIDTH + (col-1)];
        din[63:56] = (is_last_row               ) ? 8'd0 : in_img[(row+1) * WIDTH +  col   ];
        din[71:64] = (is_last_row | is_last_col ) ? 8'd0 : in_img[(row+1) * WIDTH + (col+1)];
    end
end
```

cnv_tb.v - din

row : 0 , col : 1

0	0	0	0	…	0	0	0	0
0	2a	45	5b	…	45	45	44	0
0	69	2a	38	…	41	41	44	0
0	48	2b	2a	…	2c	2d	32	0
…	…	…	…	…	…	…	…	…
0	41	34	4a	…	6f	6e	70	0
0	39	46	69	…	70	6f	71	0
0	3a	4c	4b	…	71	71	71	0
0	0	0	0	…	0	0	0	0

```
wire is_first_row = (row == 0) ? 1'b1 : 1'b0;
wire is_last_row = (row == HEIGHT-1) ? 1'b1 : 1'b0;
wire is_first_col = (col == 0) ? 1'b1 : 1'b0;
wire is_last_col = (col == WIDTH-1) ? 1'b1 : 1'b0;
```

```
always @(posedge clk) begin
    if (load_done == 1'b1) begin
        q_start <= 1'b1;
        #(4*CLK_PERIOD)
            @(posedge clk)
                q_start <= 1'b0;
    end

    vld_i = 1'b0;
    if(ctrl_data_run) begin
        vld_i = 1'b1;
        din[ 7: 0] = (is_first_row | is_first_col) ? 8'd0 : in_img[(row-1) * WIDTH + (col-1)];
        din[15: 8] = (is_first_row              ) ? 8'd0 : in_img[(row-1) * WIDTH +  col   ];
        din[23:16] = (is_first_row | is_last_col ) ? 8'd0 : in_img[(row-1) * WIDTH + (col+1)];
        din[31:24] = (             is_first_col) ? 8'd0 : in_img[ row    * WIDTH + (col-1)];
        din[39:32] =                                      in_img[ row    * WIDTH +  col   ];
        din[47:40] = (             is_last_col ) ? 8'd0 : in_img[ row    * WIDTH + (col+1)];
        din[55:48] = (is_last_row | is_first_col ) ? 8'd0 : in_img[(row+1) * WIDTH + (col-1)];
        din[63:56] = (is_last_row               ) ? 8'd0 : in_img[(row+1) * WIDTH +  col   ];
        din[71:64] = (is_last_row | is_last_col ) ? 8'd0 : in_img[(row+1) * WIDTH + (col+1)];
    end
end
```

cnv_tb.v - din

row : 0 , col : 127

0	0	0	0	…	0	0	0	0
0	2a	45	5b	…	45	45	44	0
0	69	2a	38	…	41	41	44	0
0	48	2b	2a	…	2c	2d	32	0
…	…	…	…	…	…	…	…	…
0	41	34	4a	…	6f	6e	70	0
0	39	46	69	…	70	6f	71	0
0	3a	4c	4b	…	71	71	71	0
0	0	0	0	…	0	0	0	0

```
wire is_first_row = (row == 0) ? 1'b1 : 1'b0;
wire is_last_row = (row == HEIGHT-1) ? 1'b1 : 1'b0;
wire is_first_col = (col == 0) ? 1'b1 : 1'b0;
wire is_last_col = (col == WIDTH-1) ? 1'b1 : 1'b0;
```

```
always @(posedge clk) begin
    if (load_done == 1'b1) begin
        q_start <= 1'b1;
        #(4*CLK_PERIOD)
            @(posedge clk)
                q_start <= 1'b0;
    end

    vld_i = 1'b0;
    if(ctrl_data_run) begin
        vld_i = 1'b1;
        din[ 7: 0] = (is_first_row | is_first_col) ? 8'd0 : in_img[(row-1) * WIDTH + (col-1)];
        din[15: 8] = (is_first_row              ) ? 8'd0 : in_img[(row-1) * WIDTH +  col   ];
        din[23:16] = (is_first_row | is_last_col ) ? 8'd0 : in_img[(row-1) * WIDTH + (col+1)];
        din[31:24] = (               is_first_col) ? 8'd0 : in_img[ row    * WIDTH + (col-1)];
        din[39:32] =                                        in_img[ row    * WIDTH +  col   ];
        din[47:40] = (               is_last_col ) ? 8'd0 : in_img[ row    * WIDTH + (col+1)];
        din[55:48] = (is_last_row | is_first_col ) ? 8'd0 : in_img[(row+1) * WIDTH + (col-1)];
        din[63:56] = (is_last_row               ) ? 8'd0 : in_img[(row+1) * WIDTH +  col   ];
        din[71:64] = (is_last_row | is_last_col ) ? 8'd0 : in_img[(row+1) * WIDTH + (col+1)];
    end
end
```

cnv_tb.v - din

row : 1 , col : 0

0	0	0	0	…	0	0	0	0
0	2a	45	5b	…	45	45	44	0
0	69	2a	38	…	41	41	44	0
0	48	2b	2a	…	2c	2d	32	0
…	…	…	…	…	…	…	…	…
0	41	34	4a	…	6f	6e	70	0
0	39	46	69	…	70	6f	71	0
0	3a	4c	4b	…	71	71	71	0
0	0	0	0	…	0	0	0	0

```
wire is_first_row = (row == 0) ? 1'b1 : 1'b0;
wire is_last_row = (row == HEIGHT-1) ? 1'b1 : 1'b0;
wire is_first_col = (col == 0) ? 1'b1 : 1'b0;
wire is_last_col = (col == WIDTH-1) ? 1'b1 : 1'b0;
```

```
always @(posedge clk) begin
    if (load_done == 1'b1) begin
        q_start <= 1'b1;
        #(4*CLK_PERIOD)
            @(posedge clk)
                q_start <= 1'b0;
    end

    vld_i = 1'b0;
    if(ctrl_data_run) begin
        vld_i = 1'b1;
        din[ 7: 0] = (is_first_row | is_first_col) ? 8'd0 : in_img[(row-1) * WIDTH + (col-1)];
        din[15: 8] = (is_first_row              ) ? 8'd0 : in_img[(row-1) * WIDTH +  col   ];
        din[23:16] = (is_first_row | is_last_col ) ? 8'd0 : in_img[(row-1) * WIDTH + (col+1)];
        din[31:24] = (               is_first_col) ? 8'd0 : in_img[ row    * WIDTH + (col-1)];
        din[39:32] =                                        in_img[ row    * WIDTH +  col   ];
        din[47:40] = (               is_last_col ) ? 8'd0 : in_img[ row    * WIDTH + (col+1)];
        din[55:48] = (is_last_row | is_first_col ) ? 8'd0 : in_img[(row+1) * WIDTH + (col-1)];
        din[63:56] = (is_last_row               ) ? 8'd0 : in_img[(row+1) * WIDTH +  col   ];
        din[71:64] = (is_last_row | is_last_col ) ? 8'd0 : in_img[(row+1) * WIDTH + (col+1)];
    end
end
```

cnv_tb.v - din

row : 127 , col : 0

0	0	0	0	…	0	0	0	0
0	2a	45	5b	…	45	45	44	0
0	69	2a	38	…	41	41	44	0
0	48	2b	2a	…	2c	2d	32	0
…	…	…	…	…	…	…	…	…
0	41	34	4a	…	6f	6e	70	0
0	39	46	69	…	70	6f	71	0
0	3a	4c	4b	…	71	71	71	0
0	0	0	0	…	0	0	0	0

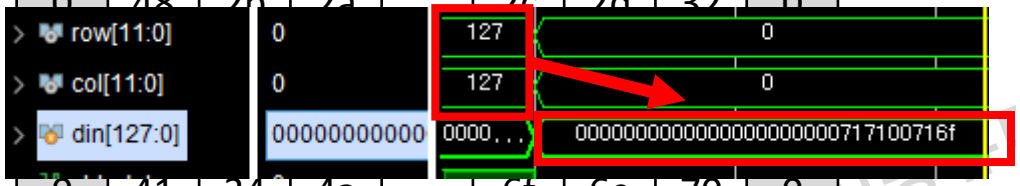
```
wire is_first_row = (row == 0) ? 1'b1 : 1'b0;
wire is_last_row = (row == HEIGHT-1) ? 1'b1 : 1'b0;
wire is_first_col = (col == 0) ? 1'b1 : 1'b0;
wire is_last_col = (col == WIDTH-1) ? 1'b1 : 1'b0;
```

```
always @(posedge clk) begin
    if (load_done == 1'b1) begin
        q_start <= 1'b1;
        #(4*CLK_PERIOD)
            @(posedge clk)
                q_start <= 1'b0;
    end

    vld_i = 1'b0;
    if(ctrl_data_run) begin
        vld_i = 1'b1;
        din[ 7: 0] = (is_first_row | is_first_col) ? 8'd0 : in_img[(row-1) * WIDTH + (col-1)];
        din[15: 8] = (is_first_row              ) ? 8'd0 : in_img[(row-1) * WIDTH +  col   ];
        din[23:16] = (is_first_row | is_last_col ) ? 8'd0 : in_img[(row-1) * WIDTH + (col+1)];
        din[31:24] = (               is_first_col) ? 8'd0 : in_img[ row    * WIDTH + (col-1)];
        din[39:32] =                                        in_img[ row    * WIDTH +  col   ];
        din[47:40] = (               is_last_col ) ? 8'd0 : in_img[ row    * WIDTH + (col+1)];
        din[55:48] = (is_last_row | is_first_col ) ? 8'd0 : in_img[(row+1) * WIDTH + (col-1)];
        din[63:56] = (is_last_row               ) ? 8'd0 : in_img[(row+1) * WIDTH +  col   ];
        din[71:64] = (is_last_row | is_last_col ) ? 8'd0 : in_img[(row+1) * WIDTH + (col+1)];
    end
end
```

cnv_tb.v - din

row : 127 , col : 127

0	0	0	0	…	0	0	0	0
0	2a	45	5b	…	45	45	44	0
0	69	2a	38	…	41	41	44	0
0	41	34	4a	…	6f	6e	70	0
0	39	46	69	…	70	6f	71	0
0	3a	4c	4b	…	71	71	71	0
0	0	0	0	…	0	0	0	0



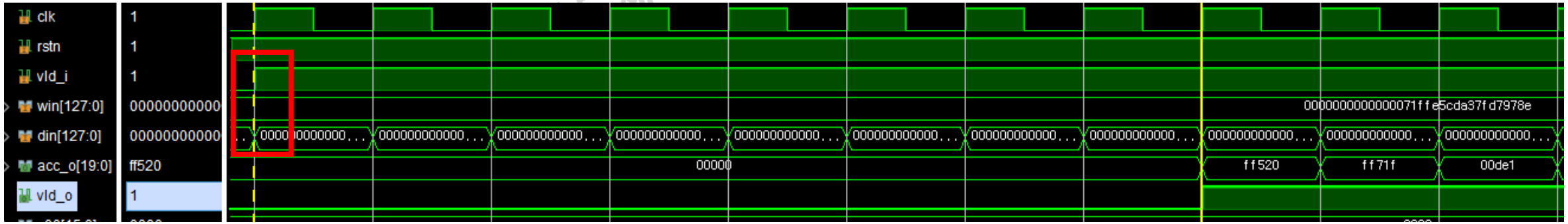

```
wire is_first_row = (row == 0) ? 1'b1 : 1'b0;
wire is_last_row = (row == HEIGHT-1) ? 1'b1 : 1'b0;
wire is_first_col = (col == 0) ? 1'b1 : 1'b0;
wire is_last_col = (col == WIDTH-1) ? 1'b1 : 1'b0;
```

```
always @(posedge clk) begin
    if (load_done == 1'b1) begin
        q_start <= 1'b1;
        #(4*CLK_PERIOD)
            @(posedge clk)
                q_start <= 1'b0;
    end

    vld_i = 1'b0;
    if(ctrl_data_run) begin
        vld_i = 1'b1;
        din[ 7: 0] = (is_first_row | is_first_col) ? 8'd0 : in_img[(row-1) * WIDTH + (col-1)];
        din[15: 8] = (is_first_row              ) ? 8'd0 : in_img[(row-1) * WIDTH +  col   ];
        din[23:16] = (is_first_row | is_last_col ) ? 8'd0 : in_img[(row-1) * WIDTH + (col+1)];
        din[31:24] = (               is_first_col) ? 8'd0 : in_img[ row    * WIDTH + (col-1)];
        din[39:32] =                                        in_img[ row    * WIDTH +  col   ];
        din[47:40] = (               is_last_col ) ? 8'd0 : in_img[ row    * WIDTH + (col+1)];
        din[55:48] = (is_last_row | is_first_col ) ? 8'd0 : in_img[(row+1) * WIDTH + (col-1)];
        din[63:56] = (is_last_row               ) ? 8'd0 : in_img[(row+1) * WIDTH +  col   ];
        din[71:64] = (is_last_row | is_last_col ) ? 8'd0 : in_img[(row+1) * WIDTH + (col+1)];
    end
end
```

mac.v

- vld_i = 1 -> start mac operation

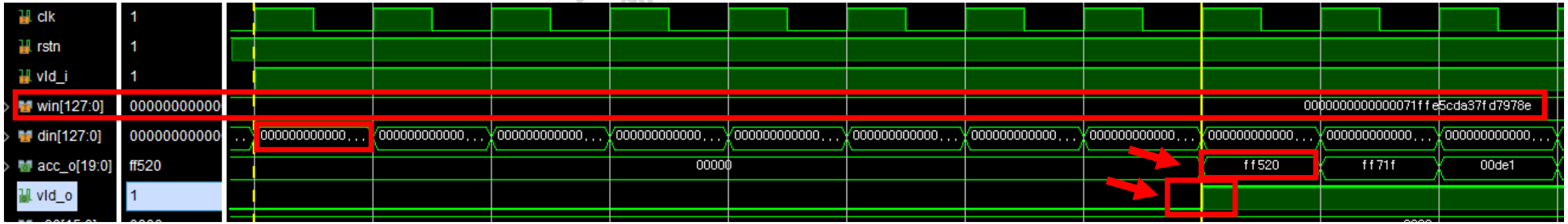
```
module mac(
input clk,
input rstn,
input vld_i,
input [127:0] win,
input [127:0] din,
output[ 19:0] acc_o,
output        vld_o
);
```



mac.v

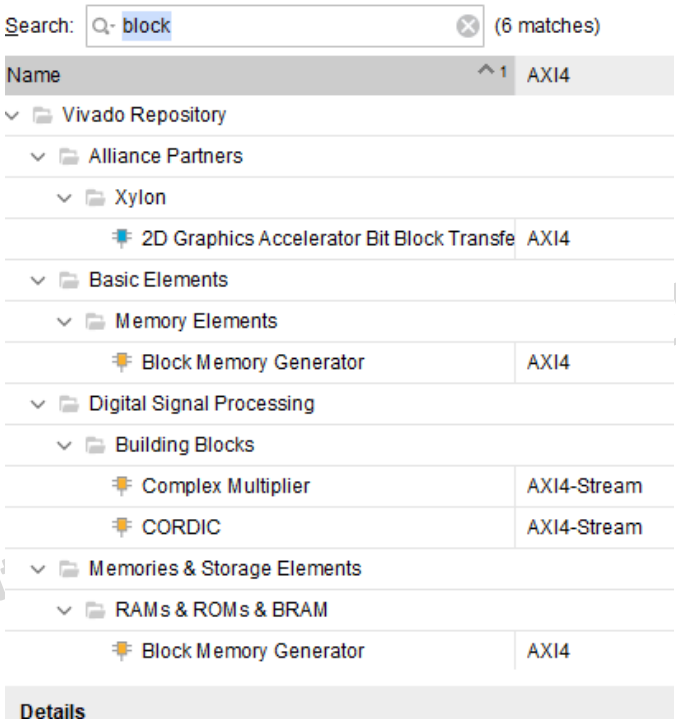
- vld_i = 1 -> start mac operation
- After 8 cycles, the result is coming out as acc_o
- And also vld_o becomes high

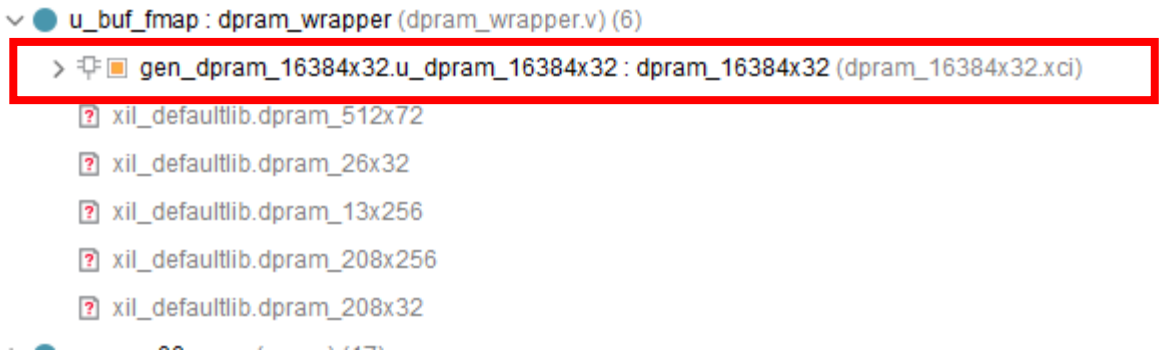
```
module mac(
input clk,
input rstn,
input vld_i,
input [127:0] win,
input [127:0] din,
output[ 19:0] acc_o,
output        vld_o
);
```



outputs buffer - dpram

- dpram_wrapper for output buffer
- You have to add BRAM IPs using IP catalog





outputs buffer

```
wire [ACT_BITS*To-1:0] all_acc_o = {
    acc_o[3][19:12], acc_o[2][19:12], acc_o[1][19:12], acc_o[0][19:12]
};
```

quantize the results of mac to 8 bits

```
// store the mac results & read and check
dpram_wrapper #(.DW(To*ACT_BITS), .AW(FRAME_SIZE_W),.DEPTH(FRAME_SIZE))
u_buf_fmap(
    .clk   (clk   ),
    .ena   (vld_o[0]),
    .wea   (vld_o[0]),
    .addra (pixel_count),
    .enb   (layer_done), // for read
    .addrb (addrb ),  // for read
    .dia   (all_acc_o),
    .dob   (dob )   // for read
);
```

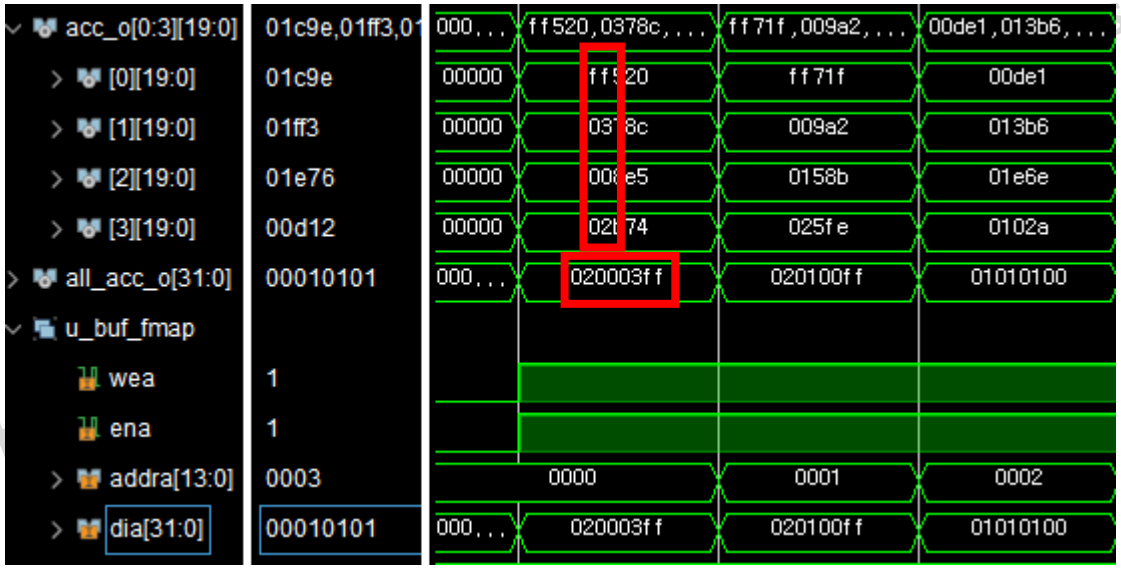
make u_buf_fmap

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

When vld_o is high (mac done) count the pixel_count until buffer store the whole results

outputs buffer

- quantize the acc_o[0], acc_o[1], acc_o[2], acc_o[3] to all_acc_o
- Write the quantized value to u_buf_fmap

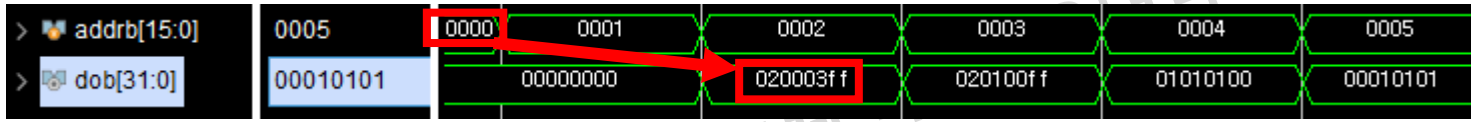


outputs buffer

- Read the stored values
- Check the stored values are same as the all_acc_o

```
////////////////////////////////////////////////
// for checking : read fmap and check
reg [15:0] addrb;
wire [31:0] dob;
always@(posedge clk, negedge rstn) begin
    if(!rstn) addrb <= 0;
    else if(layer_done) begin
        if(addrb == 16'd16384) layer_done <= 0;
        else addrb <= addrb + 1;
    end
end
////////////////////////////////////////////////
```

2 cycle delay for read

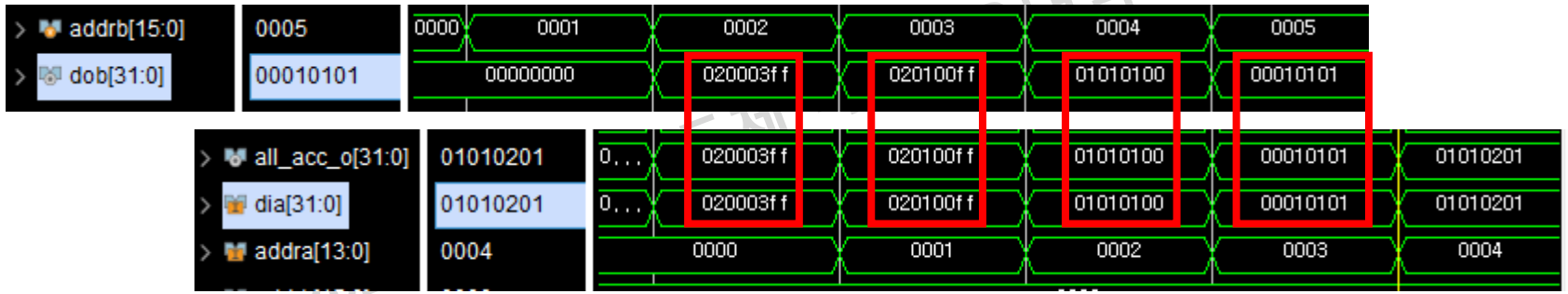


outputs buffer

- Read the stored values
- Check the stored values are same as the all_acc_o

```
//////////////////////////////////////////////
// for checking : read fmap and check
reg [15:0] addrb;
wire [31:0] dob;
always@(posedge clk, negedge rstn) begin
    if(!rstn) addrb <= 0;
    else if(layer_done) begin
        if(addrb == 16'd16384) layer_done <= 0;
        else addrb <= addrb + 1;
    end
end
//////////////////////////////////////////////
```

Same!



Incoming lectures

- System integration
  - User-defined IP
  - PC-Host communication