SEOUL NATIONAL UNIVERSITY
SYSTEM SEMICONDUCTOR ENGINEERING FOR AI
서울대학교 연합전공 인공지능반도체공학

2024 Deep Learning Hardware Design Competition: Final Presentation

Dongmin Han, Changwoo Lee, Donghwee Son

ChipChamps, Seoul National University

Contents

1. Outline

- Objective
  - Appropriately quantize the model so that all computations can be done in integer, without significant accuracy loss.
  - Design CNN inference accelerator considering accuracy, time, power, and resource utilization.
  - Implement the designed modules on actual HW, like FPGA.

1. Outline

- Final Result
  - Quantization (mAP) : 81.10%

```
class_id = 58, name = coca_cola_glass_bottle,    ap = 80.08 %
class_id = 59, name = twix,      ap = 87.91 %
 for thresh = 0.24, precision = 0.72, recall = 0.68, F1-score =
 for thresh = 0.24, TP = 2020, FP = 783, FN = 939, average IoU

 mean average precision (mAP) = 0.810961, or 81.10 %
Total Detection Time: 23.000000 Seconds
(base) handongmin@handongmin-ui-MacBookAir bin %
```

  - Original : 81.76%

```
float weight_quant_multiplier[TOTAL_CALIB_LAYER] = {
  16,     //conv 0
  128,     //conv 2
  128,     //conv 4
  128,     //conv 6
  64,     //conv 8
  2048,     //conv 10
  256,     //conv 12
  1024,     //conv 13
  256,     //conv 14
  128,     //conv 17
  512};    //conv 20
```

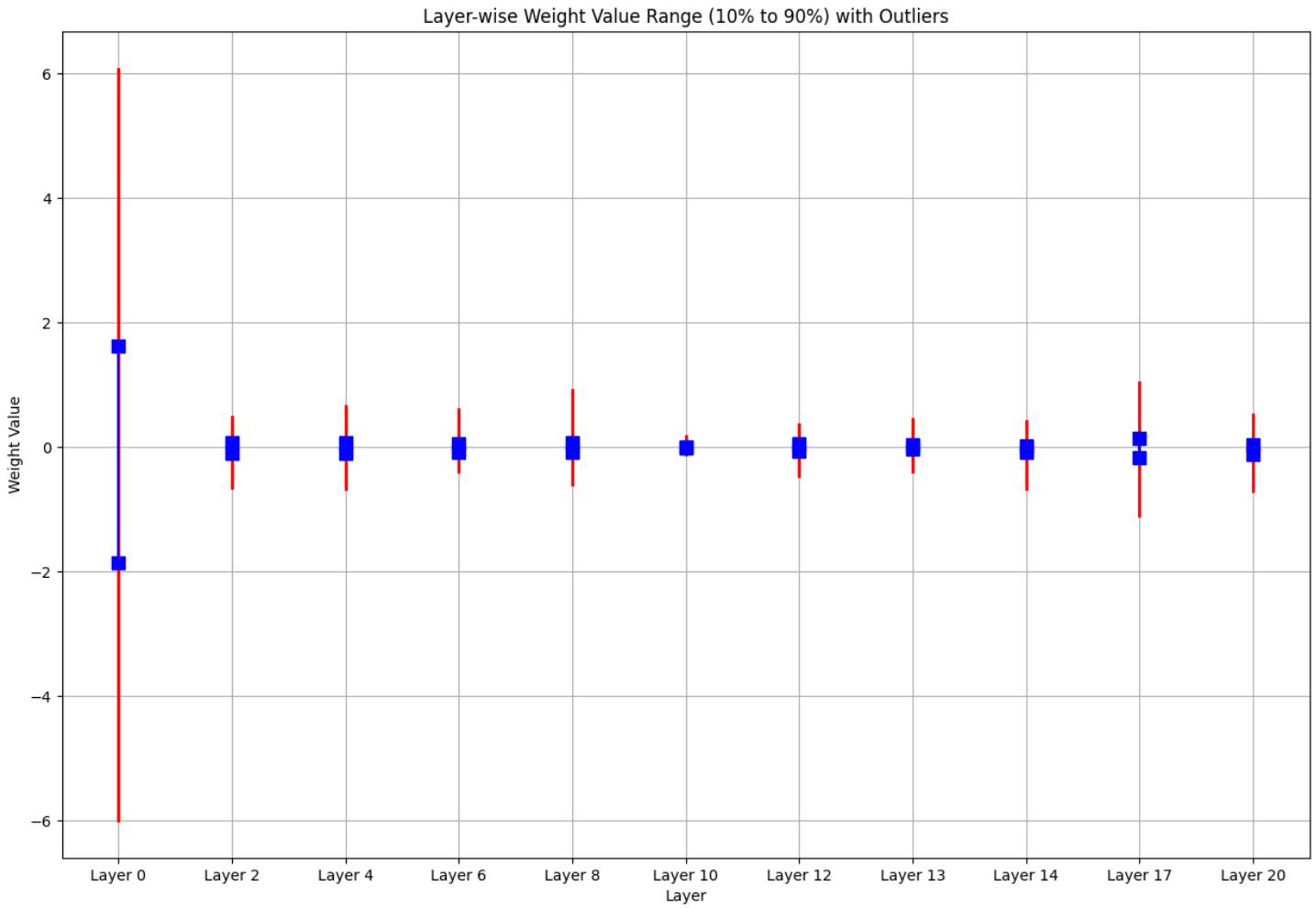
```
float input_quant_multiplier[TOTAL_CALIB_LAYER] = {
  64,     //conv 0
  8,     //conv 2
  8,     //conv 4
  8,     //conv 6
  8,     //conv 8
  8,     //conv 10
  8,     //conv 12
  8,     //conv 13
  8,     //conv 14
  8,     //conv 17
  8};    //conv 20
```

1. Outline

- Final Result
  - Simulation completed for all layers (conv0 ~ conv20)
    - This was implemented only with regs, and assuming all ifmaps/weights for each layer can fit into the regs. (Impractical on HW)
    - Therefore, ifmap tiling was implemented to put practical sizes of data can fit into the limited reg/bram of FPGA.

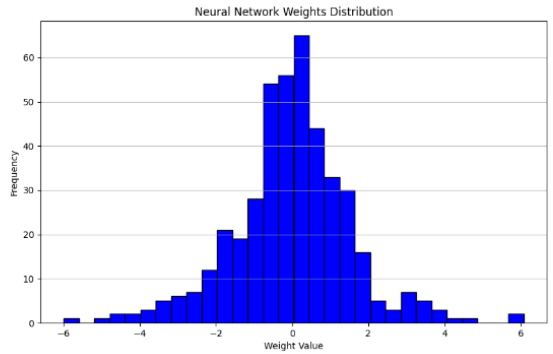
2. Idea and Novelty: Quantization Method

- Weight Distribution
  - Overall layer-wise view
  - Min/Max ranges are quite limited!

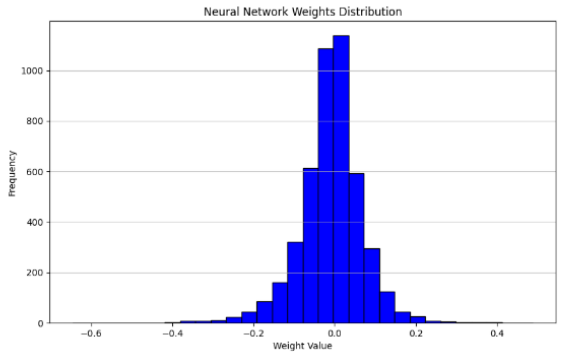


2. Idea and Novelty: Quantization Method

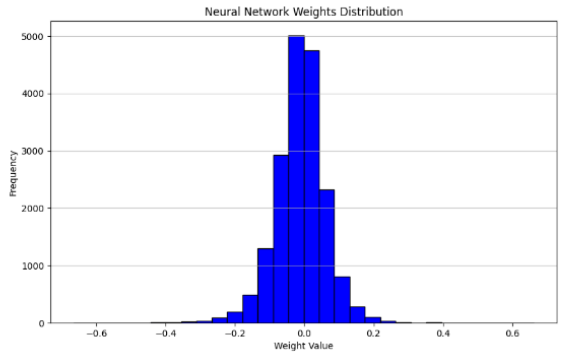
- Weight Distribution
  - Extracted weight values to get the best scaling factor empirically



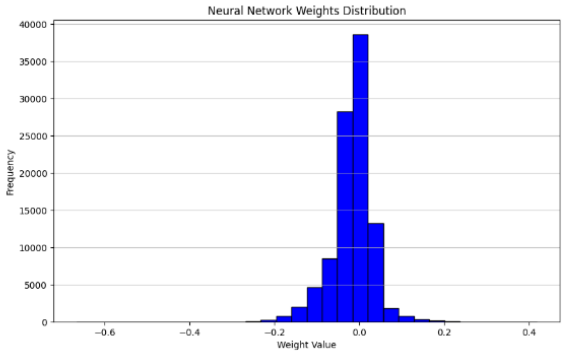
Layer 0



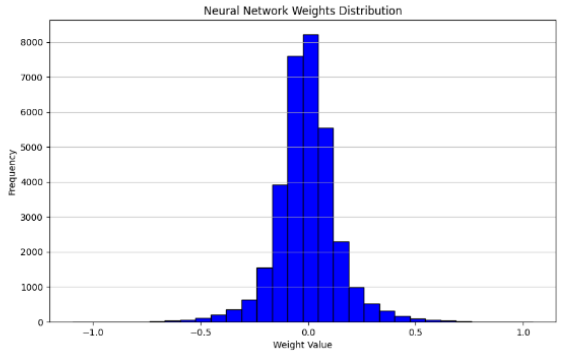
Layer 2



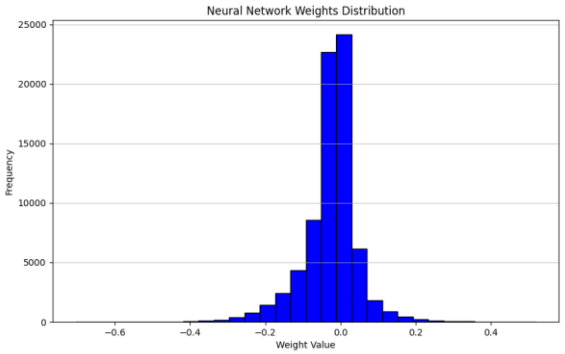
Layer 4



Layer 14



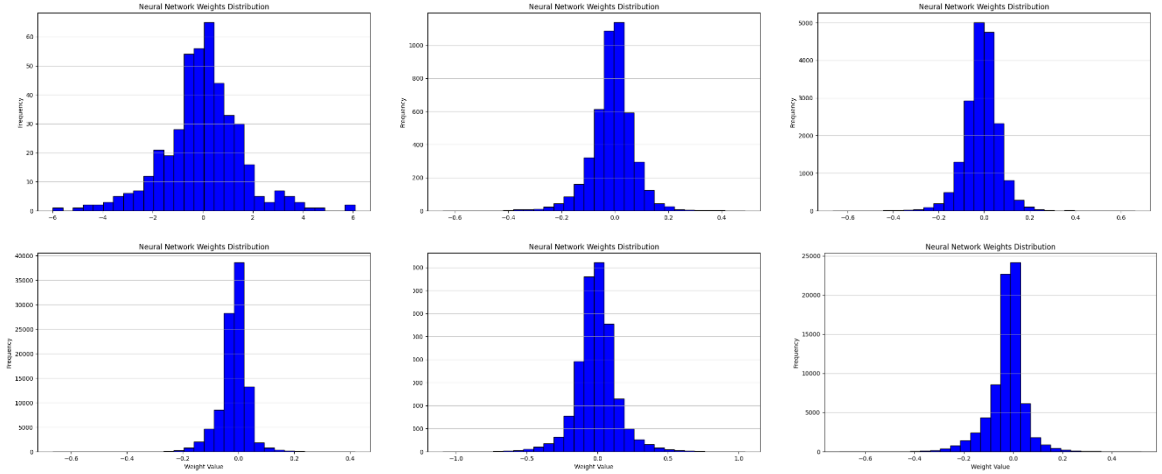
Layer 17



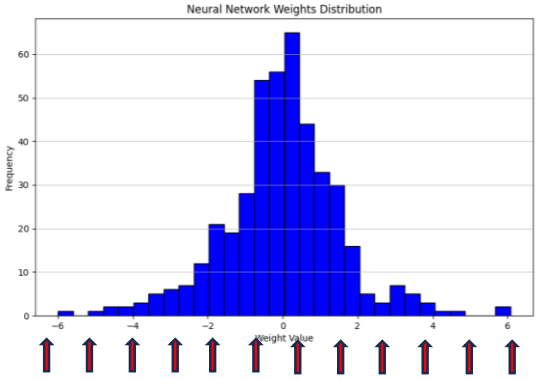
Layer 20

2. Idea and Novelty: Quantization Method

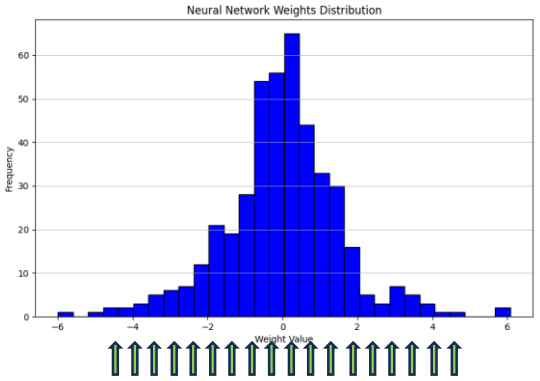
- Weight Distribution
  - From this result, get best scaling factor s which minimizes our metric: Mean Squared Error (MSE)[1]
  - Also, we observed distribution mean ≈ 0 and accepted symmetric scheme



s gives quantization standpoints



$s = 2^7$
MSE=0.919



$s = 2^4$
MSE=0.00031

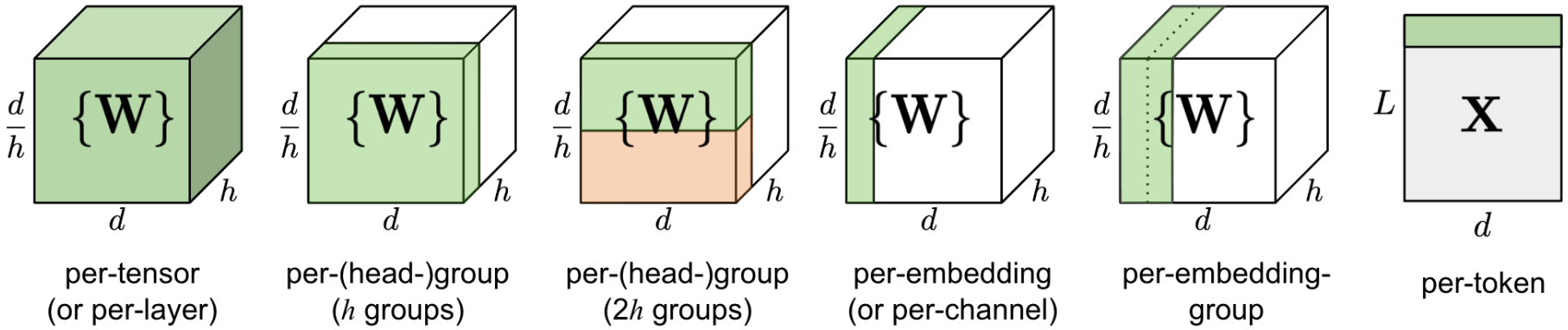
[1] Minsik Kim, Kyoungseok Oh, Youngmock Cho, Hojin Seo, Xuan Truong Nguyen, and Hyuk-Jae Lee. 2024. A Low-Latency FPGA Accelerator for YOLOv3-Tiny With Flexible Layerwise Mapping and Dataflow. IEEE Transactions on Circuits and Systems I: Regular Papers 71, 3 (2024), 1158-1171. DOI:https://doi.org/10.1109/tcsi.2023.3335949

2. Idea and Novelty: Quantization Method

- Power-of-two quantizer
  - In the previous setting, we measure MSE of s only by *power-of-two*
  - This choice can bring hardware efficiencies as scaling with s corresponds to *simple bit-shift*
  - The restricted expressiveness of the scale factor s can complicate the trade-off between the rounding and clipping error

2. Idea and Novelty: Quantization Method

- Quantization granularity
  - Quantization parameters can be set per layer, per channel, … this is called quantization granularity
  - Our parameters are shared **per layer**, for the sake of simplicity and hardware efficiency



2. Idea and Novelty: CONV Layers

- Three distinct computation methods each for CONV00, 3x3, and 1x1 conv.
  - CONV00
    - The only layer with ifmap channel of 3.
    - Needs separate dataflow compared to ordinary 3x3 convolution.

```
layer     filters    size              input                output
    0 conv     16  3 x 3 / 1   256 x 256 x   3   ->   256 x 256 x  16 0.057 BF
    1 max          2 x 2 / 2   256 x 256 x  16   ->   128 x 128 x  16
    2 conv     32  3 x 3 / 1   128 x 128 x  16   ->   128 x 128 x  32 0.151 BF
    3 max          2 x 2 / 2   128 x 128 x  32   ->    64 x  64 x  32
    4 conv     64  3 x 3 / 1    64 x  64 x  32   ->    64 x  64 x  64 0.151 BF
    5 max          2 x 2 / 2    64 x  64 x  64   ->    32 x  32 x  64
    6 conv    128  3 x 3 / 1    32 x  32 x  64   ->    32 x  32 x 128 0.151 BF
    7 max          2 x 2 / 2    32 x  32 x 128   ->    16 x  16 x 128
    8 conv    256  3 x 3 / 1    16 x  16 x 128   ->    16 x  16 x 256 0.151 BF
```

2. Idea and Novelty: CONV Layers

- Three distinct computation methods each for CONV00, 3x3, and 1x1 conv.
  - 3x3 convolution (except CONV00)
    - All ifmap and ofmap channels are multiple of 16.
    - All layers except CONV13 are followed by ReLU and maxpool.

      → Integrated!

```
 2 conv     32  3 x 3 / 1   128 x 128 x  16   ->   128 x 128 x  32 0.151 BF
 3 max          2 x 2 / 2   128 x 128 x  32   ->    64 x  64 x  32
 4 conv     64  3 x 3 / 1    64 x  64 x  32   ->    64 x  64 x  64 0.151 BF
 5 max          2 x 2 / 2    64 x  64 x  64   ->    32 x  32 x  64
 6 conv    128  3 x 3 / 1    32 x  32 x  64   ->    32 x  32 x 128 0.151 BF
 7 max          2 x 2 / 2    32 x  32 x 128   ->    16 x  16 x 128
 8 conv    256  3 x 3 / 1    16 x  16 x 128   ->    16 x  16 x 256 0.151 BF
 9 max          2 x 2 / 2    16 x  16 x 256   ->     8 x   8 x 256
10 conv    512  3 x 3 / 1     8 x   8 x 256   ->     8 x   8 x 512 0.151 BF
11 max          2 x 2 / 1     8 x   8 x 512   ->     8 x   8 x 512
12 conv    256  1 x 1 / 1     8 x   8 x 512   ->     8 x   8 x 256 0.017 BF
13 conv    512  3 x 3 / 1     8 x   8 x 256   ->     8 x   8 x 512 0.151 BF
14 conv    195  1 x 1 / 1     8 x   8 x 512   ->     8 x   8 x 195 0.013 BF
```

2. Idea and Novelty: CONV Layers

- Three distinct computation methods each for CONV00, 3x3, and 1x1 conv.
  - 1x1 convolution
    - All layers except CONV14 and CONV20 have ofmap channels with multiple of 16.
    - All layers except CONV14 and CONV20 are followed with ReLU.
    - No maxpool!

```
12 conv    256  1 x 1 / 1     8 x   8 x 512   ->     8 x   8 x 256 0.017 BF
13 conv    512  3 x 3 / 1     8 x   8 x 256   ->     8 x   8 x 512 0.151 BF
14 conv    195  1 x 1 / 1     8 x   8 x 512   ->     8 x   8 x 195 0.013 BF
15 yolo
16 route  12
17 conv    128  1 x 1 / 1     8 x   8 x 256   ->     8 x   8 x 128 0.004 BF
18 upsample            2x     8 x   8 x 128   ->    16 x  16 x 128
19 route  18 8
20 conv    195  1 x 1 / 1    16 x  16 x 384   ->    16 x  16 x 195 0.038 BF
21 yolo
```

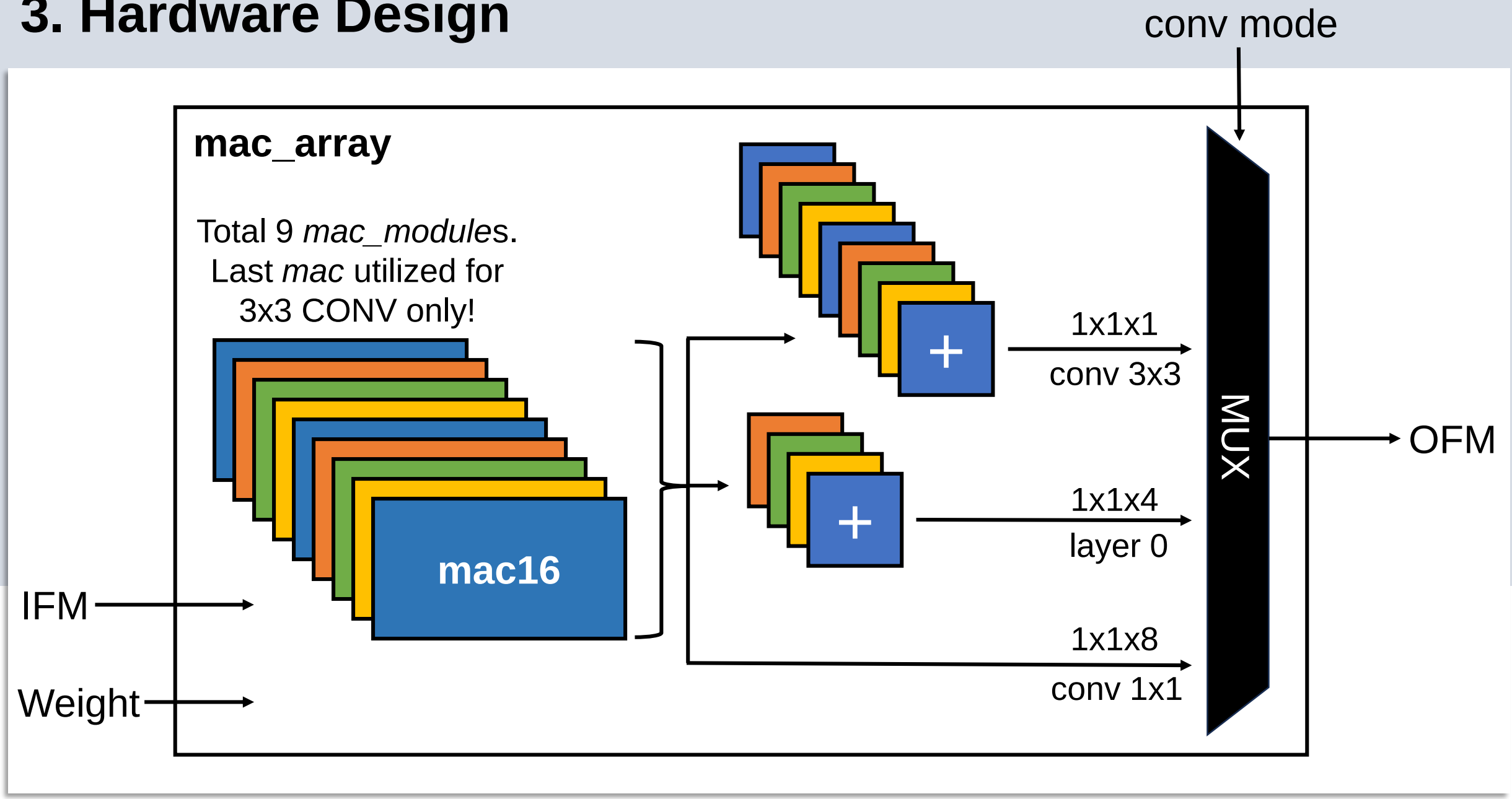
3. Hardware Design

- Terminology
  - **Ni**: ifmap channel dimension
  - **No**: ofmap channel dimension
  - **Fx**, **Fy**: filter dimension
  - **Tr**: row-wise factor, always set to 2
  - **Tc**: column-wise factor, always set to 2
  - **Ti**: ifmap channel-wise factor
  - **To**: ofmap channel-wise factor
  - CONV module consumes **(Tr x Tc) x (Fx x Fy x Ti)** ifmap pixels and produces **(Tr x Tc x To)** ofmap pixels every cycle!

3. Hardware Design

- PE utilization
  - Total 4 *mac_arrays*
    - Each *mac_array* contains 9 *mac_modules*
      - Each *mac_module* contains 16 **multipliers** and an *adder_tree*
  - Total 4 x 9 x 16 = 576 **multipliers**!
    - Nexys A7-100T FPGA board contains 240 DSPs which can map to 480 **multipliers**.
    - Remaining 96 **multipliers** are mapped to LUTs.

3. Hardware Design



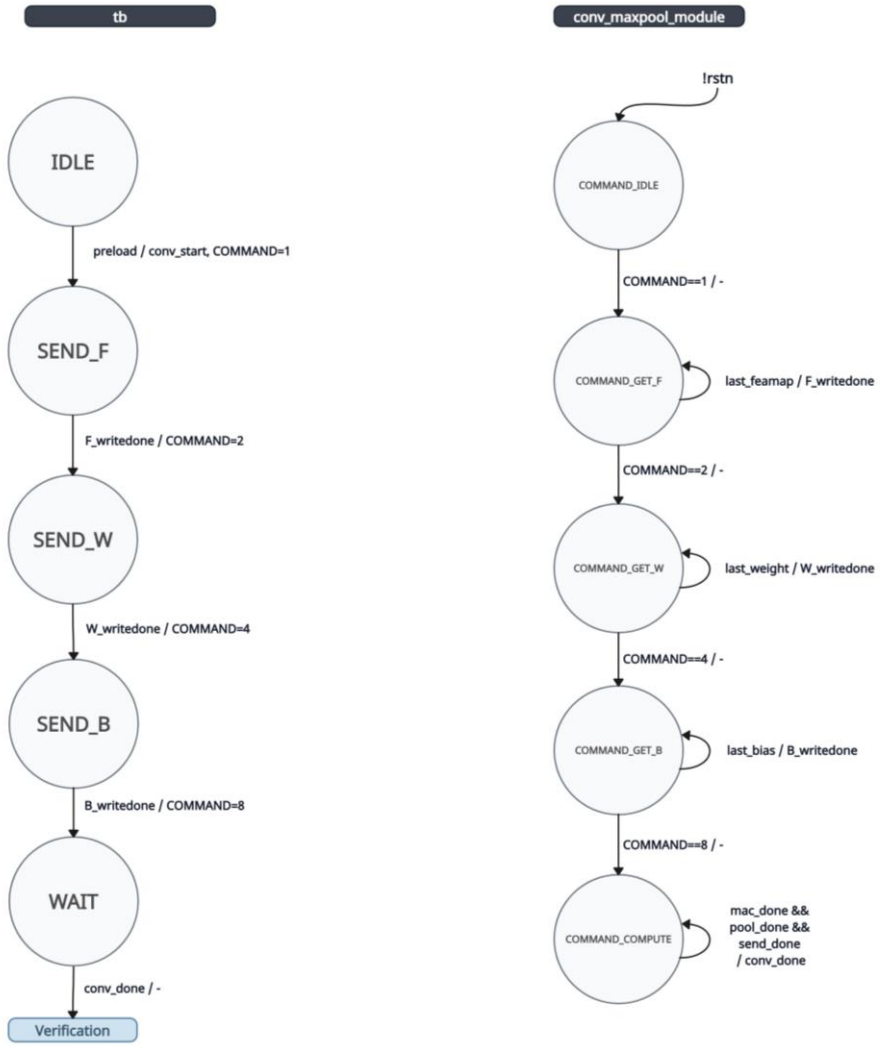
3. Hardware Design

- **Module FSM**
  - Interaction with the testbench

- Get input feature map, weights, and biases, respectively.
  - Get **32-bit data every cycle**
  - Considering the AXI granularity

- Compute and send output to *tb*

- *tb* validates the module output



3. Hardware Design: CONV00

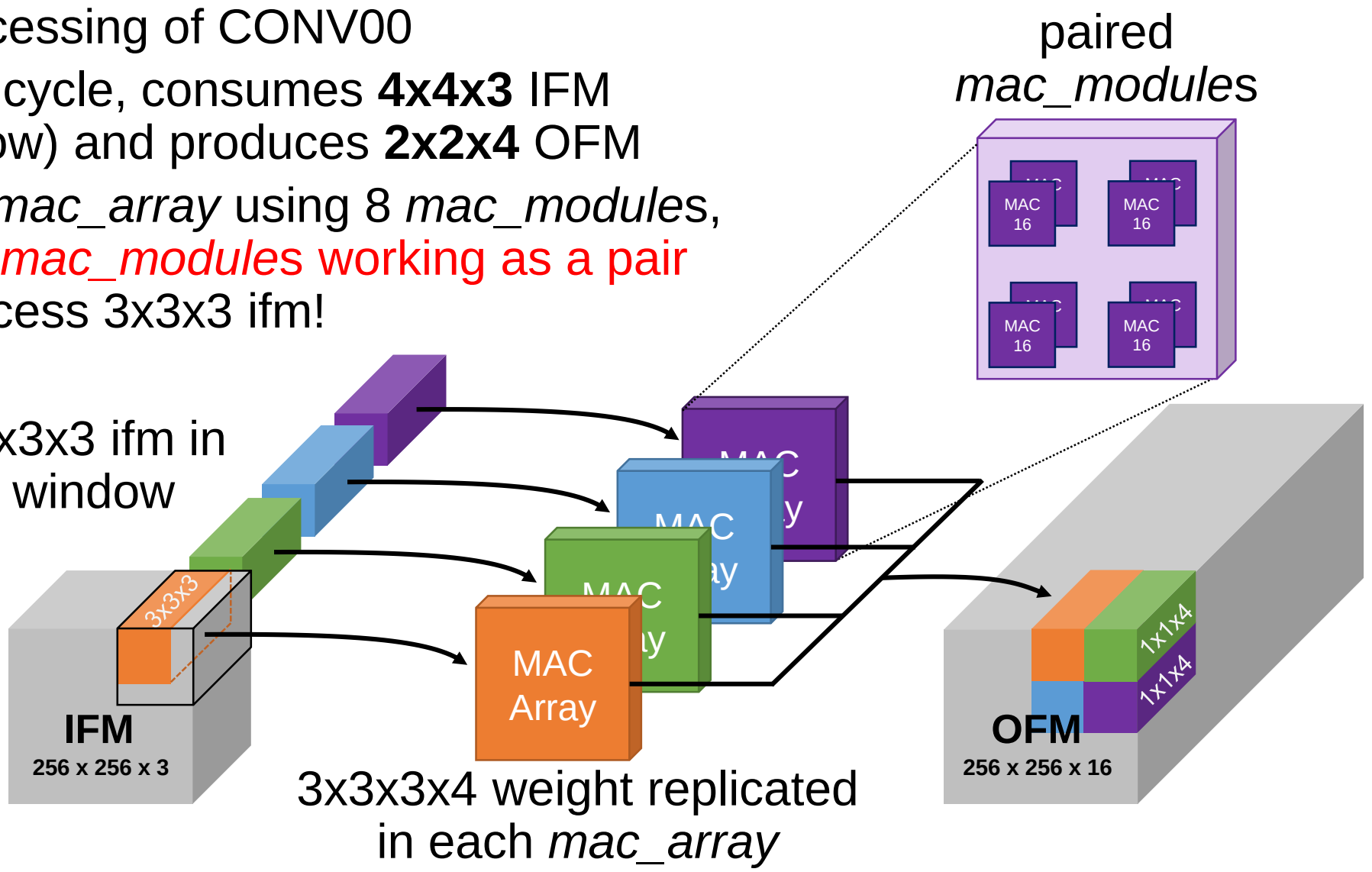
- CONV00 parameters:
  - **Ni** = 4 (actually 3)
  - **No** = 16
  - **Fx, Fy** = 3
  - **Tr, Tc** = 2
  - **Ti, To** = 4

```
/* ################################### CONV 00 ###
parameter is_CONV00 = 1;
parameter is_1x1 = 0;
parameter is_relu = 1;
parameter Tr = 2, Tc = 2;
parameter Ti = 4, To = 4;
parameter SCALE_FACTOR = 10;
parameter NEXT_LAYER_INPUT_M = 3;

// Weight
parameter Fx = 3, Fy = 3;
parameter Ni = 4, No = 16;
parameter WGT_DATA_SIZE    = Fx*Fy*Ni*No;
parameter WGT_WORD_SIZE    = 32;
```

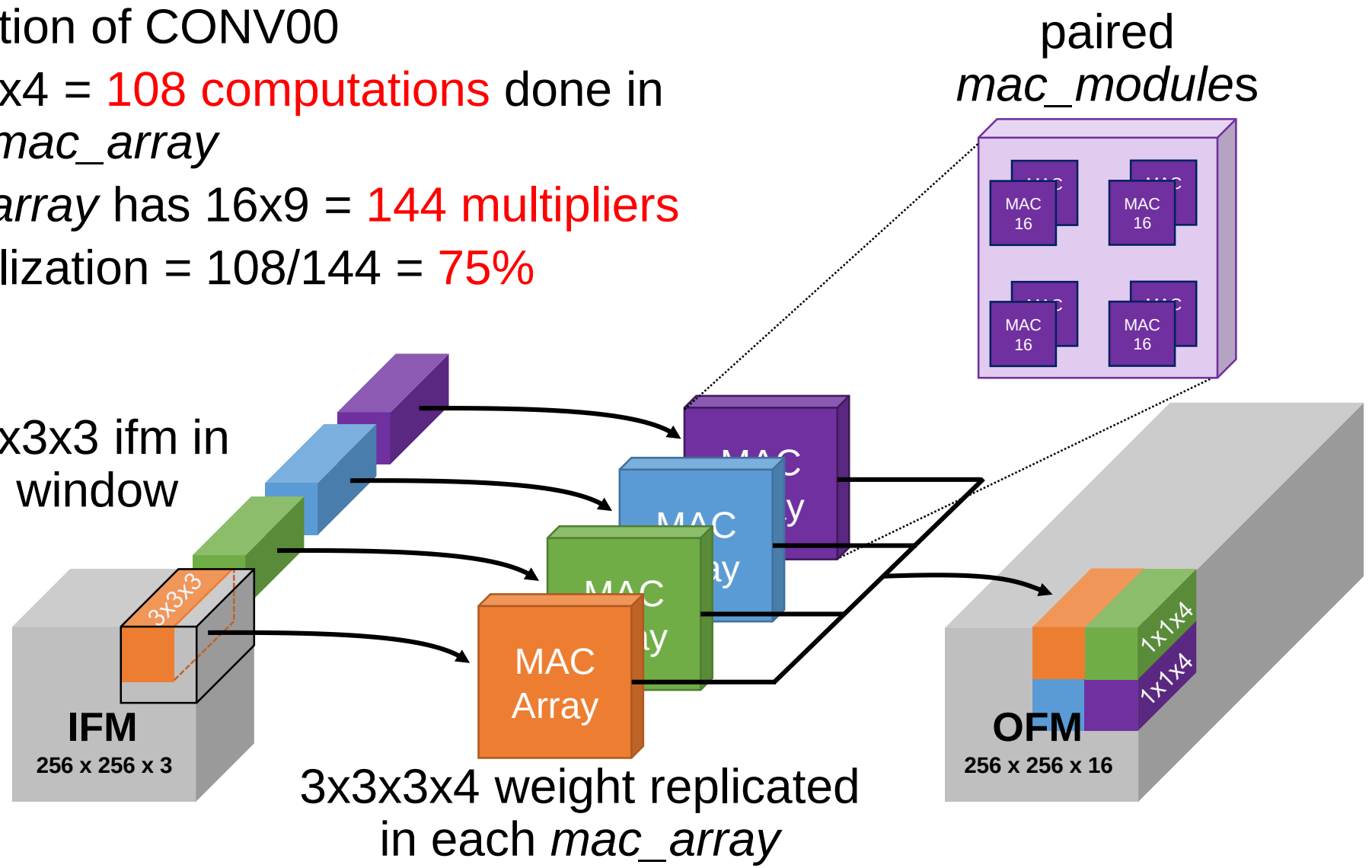
3. Hardware Design: CONV00

- Data Processing of CONV00
  - Every cycle, consumes **4x4x3** IFM (window) and produces **2x2x4** OFM
  - Each *mac_array* using 8 *mac_modules*, and 2 *mac_modules* working as a pair to process 3x3x3 ifm!

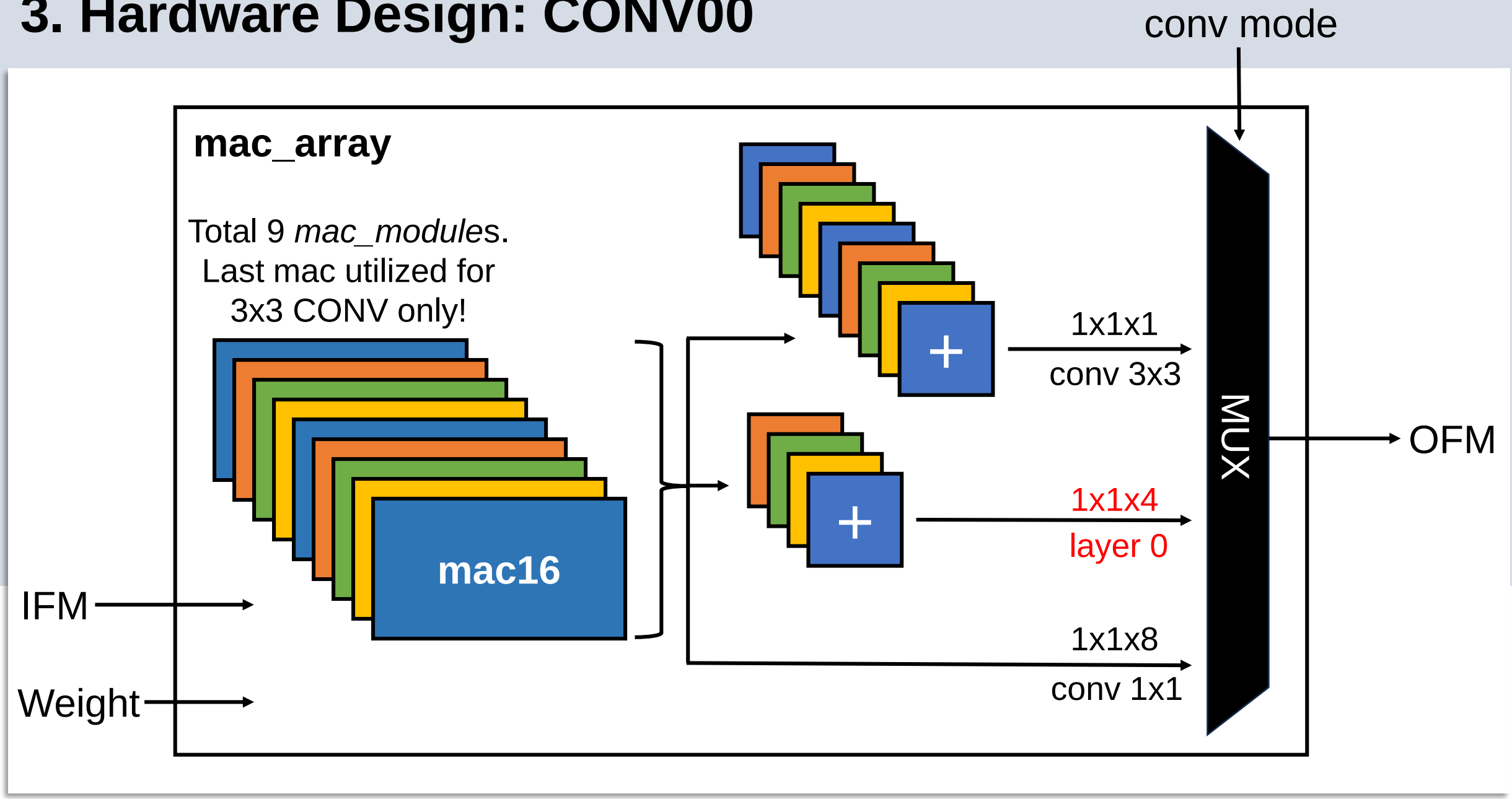


3. Hardware Design: CONV00

- PE utilization of CONV00
  - 3x3x3x4 = 108 computations done in each *mac_array*
  - *mac_array* has 16x9 = 144 multipliers
  - PE utilization = 108/144 = 75%



3. Hardware Design: CONV00



3. Hardware Design: CONV00 - Validation

```
// Validation
$display ("\n=== Validation ===\n");
$display ("Validation for layer %0d", LAYER_NUM);
$display ("Loading answers from file: %s", ANSWER_FILE);
$readmemh(ANSWER_FILE, answer);

if (!is_last) verify;
else          verify_last;


if (compare_flag) begin
    $display("\nResult is correct!\n");
end

$display("Validation done.\n");
$finish;
```

```
task verify;
begin :verifyBase
    for (i = 0; i < OFM_DATA_SIZE / 4; i = i + 1) begin
        if (OFM[i] != answer[i]) begin
            $display("\nResult is different at %0d th line!", i+1);
            $display("Expected value: %h", answer[i]);
            $display("Output value: %h\n", OFM[i]);

            compare_flag = 1'b0;
            wrong_cnt = wrong_cnt + 1;
            if (wrong_cnt == THRES) begin
                $display("Too many errors, only first %0d errors are printed.\n", THRES);
                i = OFM_DATA_SIZE / 4; // break the loop;
            end
        end
    end
end
endtask
```

```
Conv module done
Total cycle: 196710


=== Validation ===

Validation for layer 0
Loading answers from file: ../../inout_data_sw/log_feamap/CONV02_input_32b.hex

Result is different at 510 th line!
Expected value: 02830016
Output value: 027e0016

Validation done.
```

- Only one output is wrong, however, **the expected value is weird**
- Output after ReLU should be non-negative, but **0x83** is **-125**.
- Note that 2's complement of 0x83 is 0x7e

3. Hardware Design: 3x3 conv

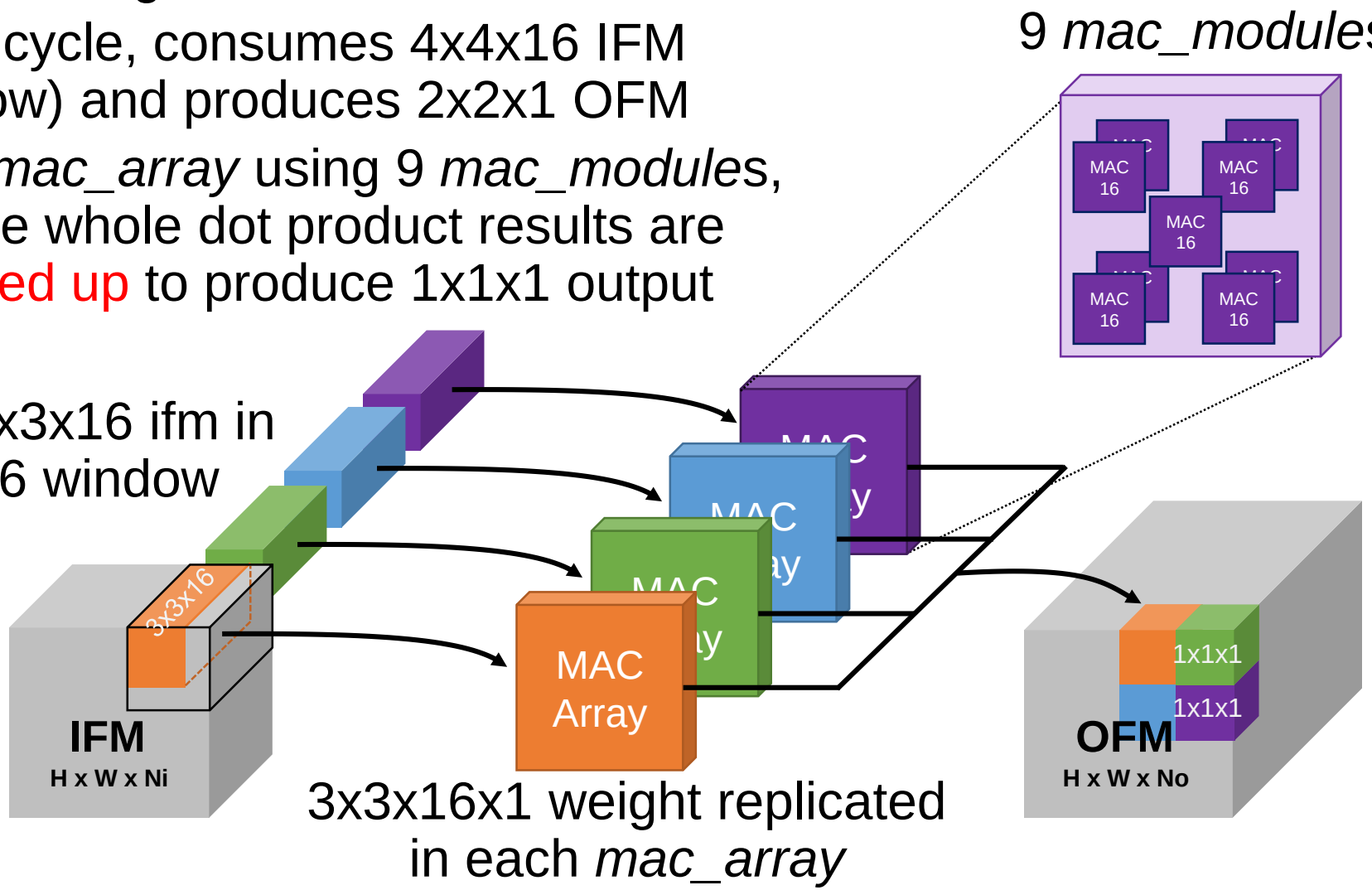
- 3x3 conv parameters:
  - **Fx, Fy** = 3
  - **Tr, Tc** = 2
  - **Ti** = 16
  - **To** = 1

```
/* ################################### CONV 02 ###
parameter is_CONV00 = 0;
parameter is_1x1 = 0;
parameter is_relu = 1;
parameter Tr = 2,  Tc = 2;
parameter Ti = 16, To = 1;
parameter SCALE_FACTOR = 10;
parameter NEXT_LAYER_INPUT_M = 3;

// Weight
parameter Fx = 3, Fy = 3;
parameter Ni = 16, No = 32;
parameter WGT_DATA_SIZE   = Fx*Fy*Ni*No;
parameter WGT_WORD_SIZE   = 32;
```

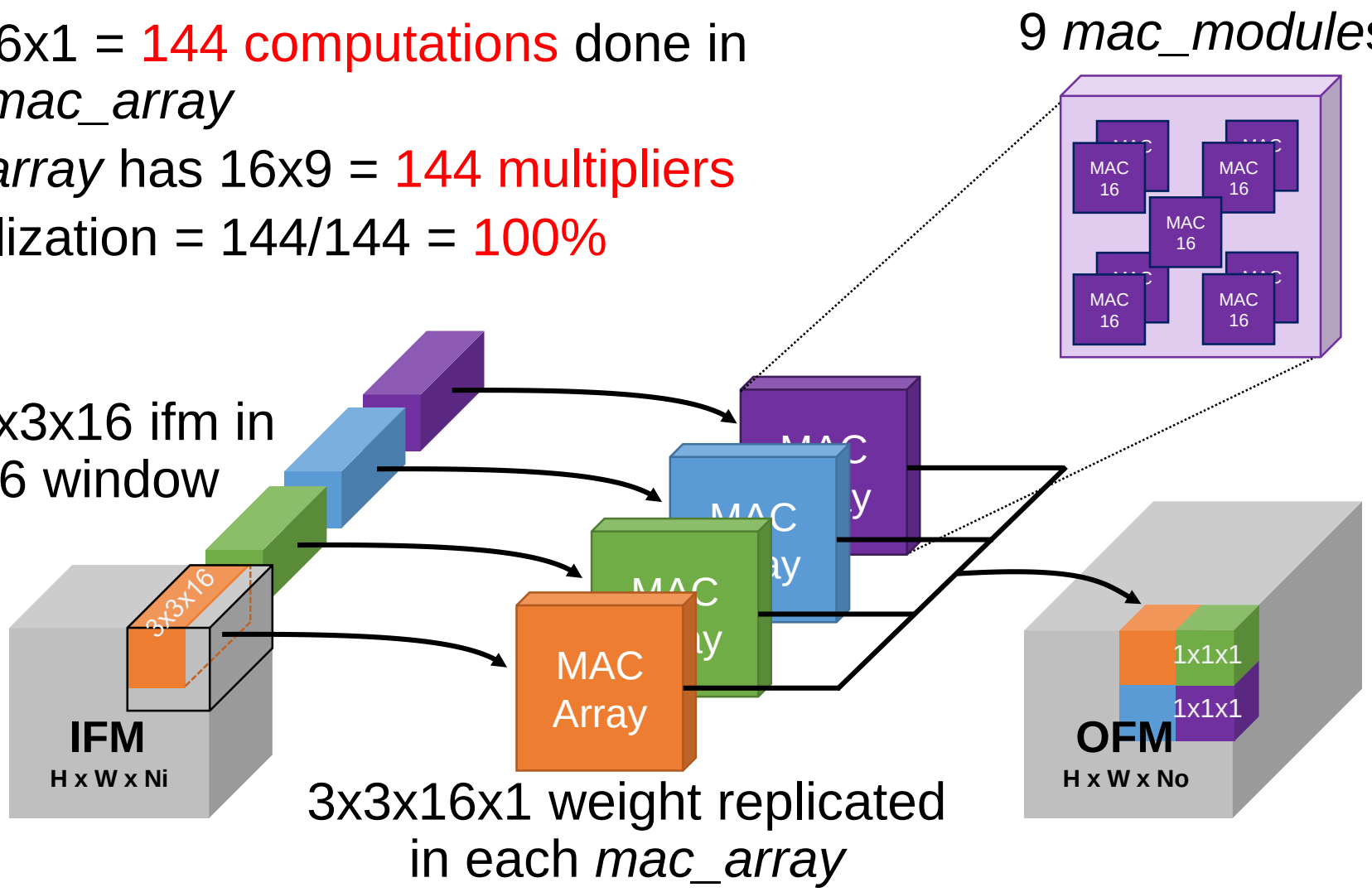
3. Hardware Design: 3x3 conv

- Data Processing of 3x3 conv
  - Every cycle, consumes 4x4x16 IFM (window) and produces 2x2x1 OFM
  - Each *mac_array* using 9 *mac_modules*, and the whole dot product results are summed up to produce 1x1x1 output

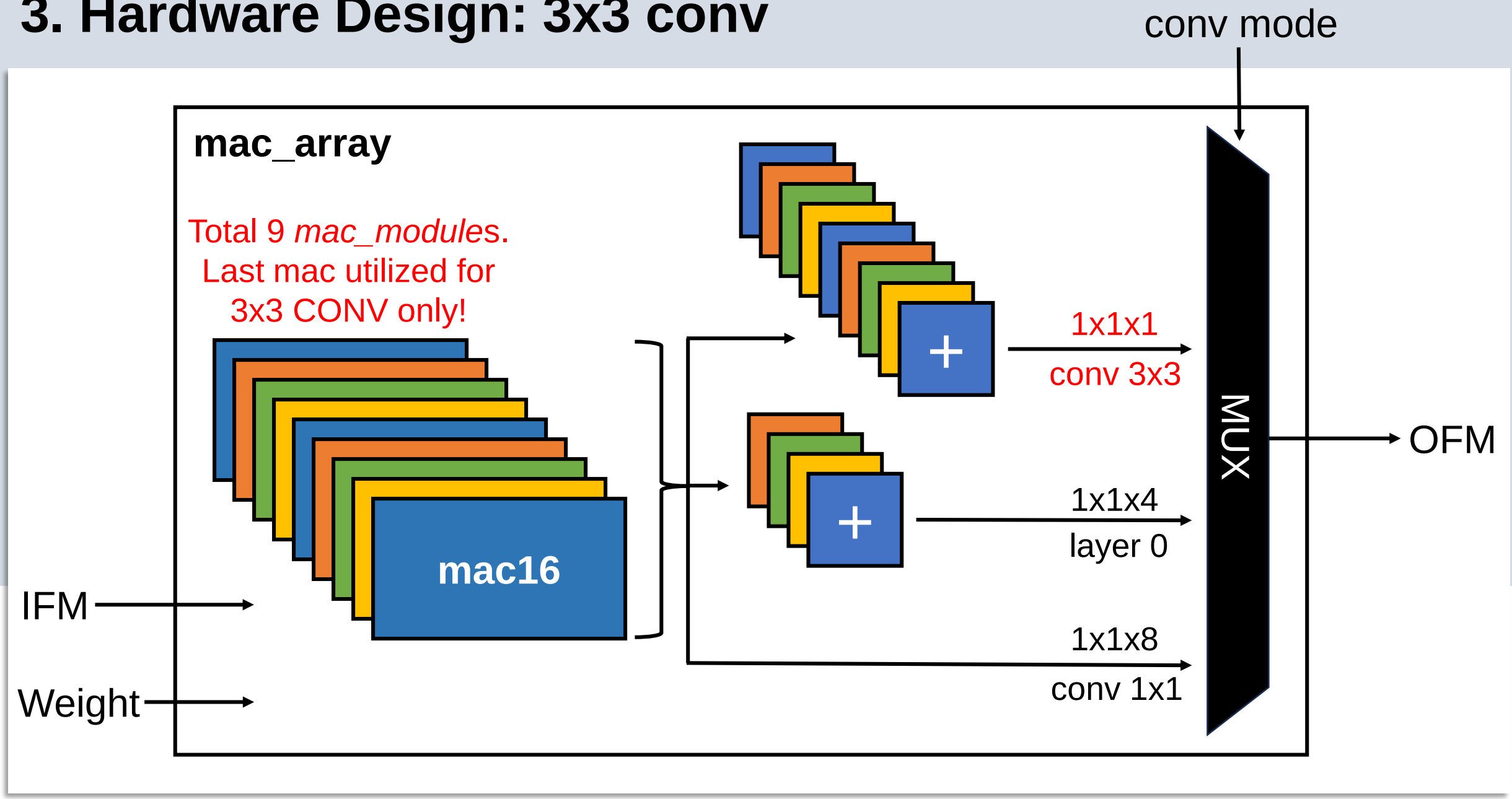


3. Hardware Design: 3x3 conv

- PE utilization of 3x3 conv
  - 3x3x16x1 = 144 computations done in each *mac_array*
  - *mac_array* has 16x9 = 144 multipliers
  - PE utilization = 144/144 = 100%



3. Hardware Design: 3x3 conv



3. Hardware Design: 3x3 conv

- For all 3x3 conv layers except 13 are followed by ReLU and maxpool, which is the most common case in the model.

- Bias addition, ReLU, quantization for the next layer, and maxpool operations are all integrated at the outputs of each *mac_array*s!
  - Note that the **integration** is not applied to CONV10, which is followed by maxpool with stride of 1.
  - Same for CONV13, where maxpool is not followed.

- No additional modules, since all operations can be pipelined.

3. Hardware Design: 3x3 conv - Validation

- Verify the result with the **next conv layer's input** hex file.
  - Note that CONV10 and CONV13 are verified with the current layer's output file.

- All verification results are archived as an image file.
  - See '1_Code/4_Captured_Results (Waveforms, Utilization)/ver_results/'

```
Conv module done
Total cycle: 264262


=== Validation ===

Validation for layer 2
Loading answers from file: ../../inout_data_sw/log_feamap/CONV04_input_32b.hex

Result is correct!

Validation done.
```

3. Hardware Design: 1x1 conv

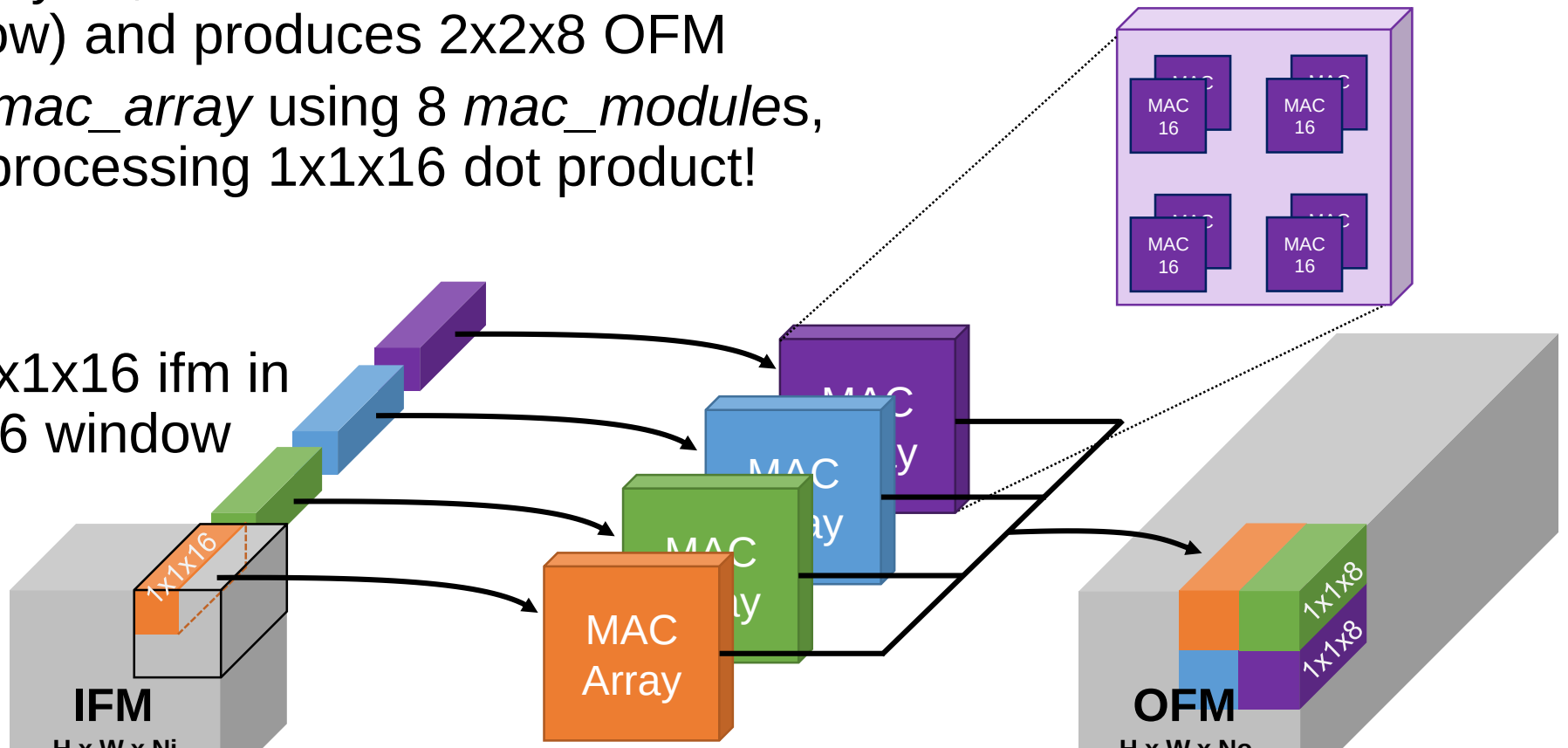
- 1x1 conv parameters:
  - **Fx, Fy** = 1
  - **Tr, Tc** = 2
  - **Ti** = 16
  - **To** = 8

```
/* ################################### CONV 12 ###
parameter is_CONV00 = 0;
parameter is_1x1 = 1;
parameter is_relu = 1;
parameter Tr = 2,  Tc = 2;
parameter Ti = 16, To = 8;
parameter SCALE_FACTOR = 11;
parameter NEXT_LAYER_INPUT_M = 3;

// Weight
parameter Fx = 1, Fy = 1;
parameter Ni = 512, No = 256;
parameter WGT_DATA_SIZE   = Fx*Fy*Ni*No;
parameter WGT_WORD_SIZE   = 32;
```

3. Hardware Design: 1x1 conv

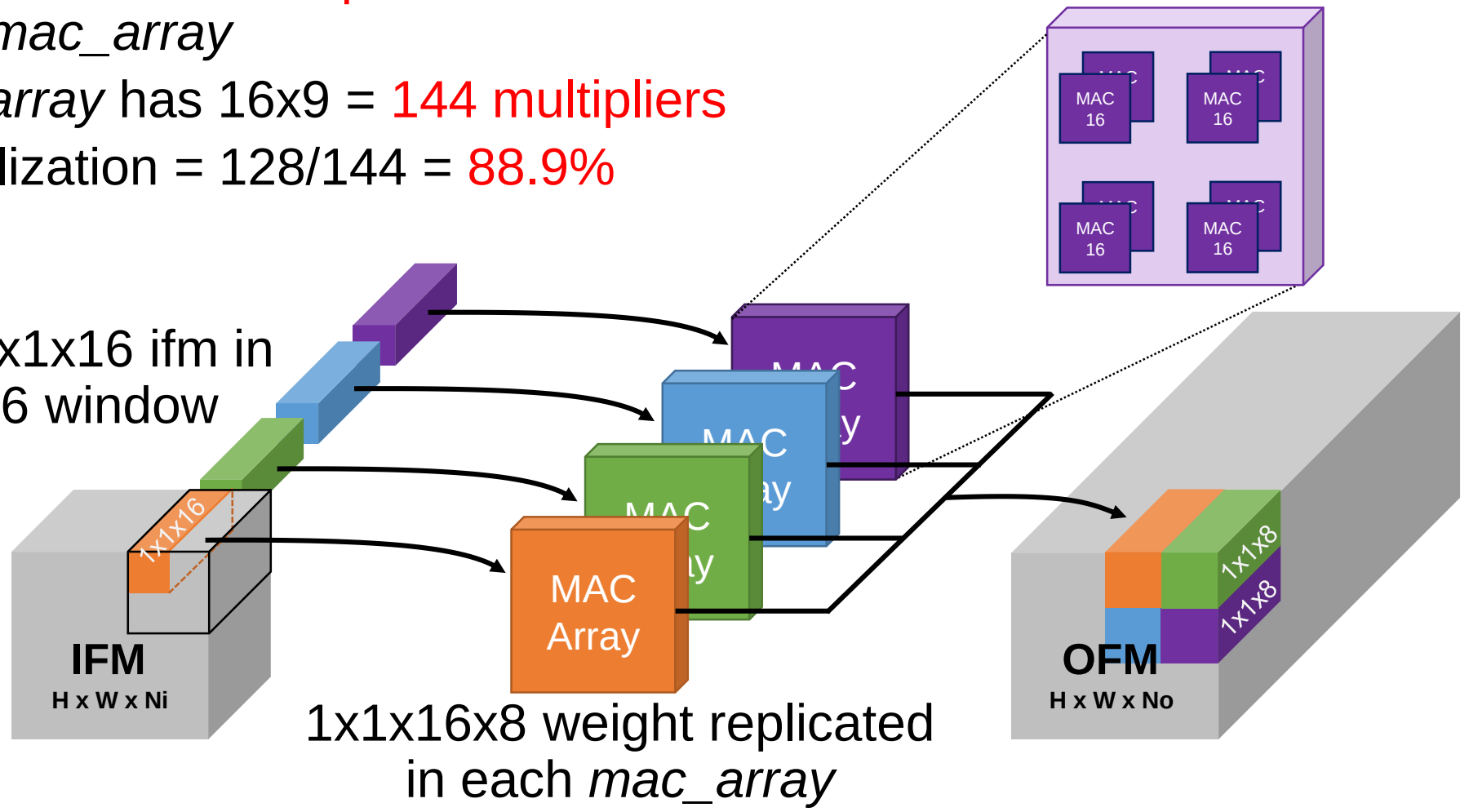
- Data Processing of 1x1 conv
  - Every cycle, consumes 2x2x16 IFM (window) and produces 2x2x8 OFM
  - Each *mac_array* using 8 *mac_modules*, each processing 1x1x16 dot product!



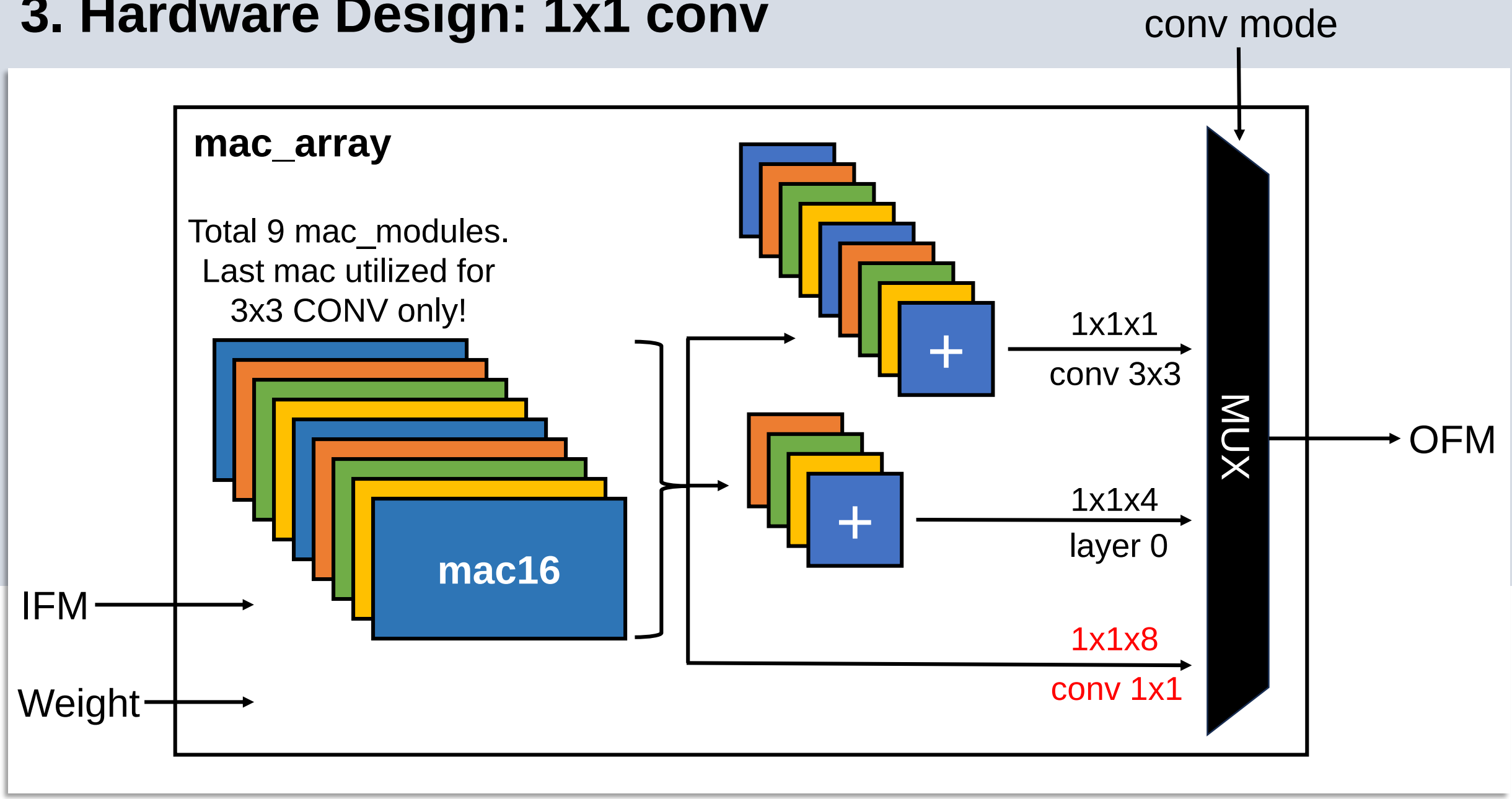
1x1x16x8 weight replicated in each *mac_array*

3. Hardware Design: 1x1 conv

- PE utilization of 1x1 conv
  - 1x1x16x8 = 128 computations done in each *mac_array*
  - *mac_array* has 16x9 = 144 multipliers
  - PE utilization = 128/144 = 88.9%



3. Hardware Design: 1x1 conv



3. Hardware Design: 1x1 conv

- Verify the result with the layer output hex file.

- All verification results are archived as an image file.
  - See ‘1_Code/4_Captured_Results (Waveforms, Utilization)/ver_results/’

```
12 conv    256  1 x 1 / 1     8 x   8 x 512   ->     8 x   8 x 256 0.017 BF
13 conv    512  3 x 3 / 1     8 x   8 x 256   ->     8 x   8 x 512 0.151 BF
14 conv    195  1 x 1 / 1     8 x   8 x 512   ->     8 x   8 x 195 0.013 BF
15 yolo
16 route  12
17 conv    128  1 x 1 / 1     8 x   8 x 256   ->     8 x   8 x 128 0.004 BF
18 upsample            2x     8 x   8 x 128   ->    16 x  16 x 128
19 route  18 8
20 conv    195  1 x 1 / 1    16 x  16 x 384   ->    16 x  16 x 195 0.038 BF
21 yolo
```

3. Hardware Design: 1x1 conv - Validation

```
task verify_last;
begin : verifyLast
    reg [7:0] answer_tol [3:0];
    reg [3:0] tol_flag;
    for (i = 0; i < OFM_DATA_SIZE / 4; i = i + 1) begin
        tol_flag = 1'b0;
        answer_tol[3] = answer[i][24+:8]+1;
        answer_tol[2] = answer[i][16+:8]+1;
        answer_tol[1] = answer[i][ 8+:8]+1;
        answer_tol[0] = answer[i][ 0+:8]+1;
        if (OFM[i] != answer[i]) begin
            // Tolerate the difference of 1 when the output is negative
            tol_flag[3] = answer[i][31] ? (OFM[i][24+:8] == answer[i][24+:8] || OFM[i][24+:8] == answer_tol[3]) : (OFM[i][24+:8] == answer[i][24+:8]);
            tol_flag[2] = answer[i][23] ? (OFM[i][16+:8] == answer[i][16+:8] || OFM[i][16+:8] == answer_tol[2]) : (OFM[i][16+:8] == answer[i][16+:8]);
            tol_flag[1] = answer[i][15] ? (OFM[i][ 8+:8] == answer[i][ 8+:8] || OFM[i][ 8+:8] == answer_tol[1]) : (OFM[i][ 8+:8] == answer[i][ 8+:8]);
            tol_flag[0] = answer[i][ 7] ? (OFM[i][ 0+:8] == answer[i][ 0+:8] || OFM[i][ 0+:8] == answer_tol[0]) : (OFM[i][ 0+:8] == answer[i][ 0+:8]);
            if (&tol_flag) begin
                compare_flag = compare_flag;
            end else begin
                $display("\nResult is different at %0d th line!", i+1);
                $display("Expected value: %h", answer[i]);
                $display("Output value: %h\n", OFM[i]);

                compare_flag = 1'b0;
                wrong_cnt = wrong_cnt + 1;
                if (wrong_cnt == THRES) begin
                    $display("Too many errors, only first %0d errors are printed.\n", THRES);
                    i = OFM_DATA_SIZE / 4; // break the loop;
                end
            end
        end
    end
end
endtask
```

Validation code for layer 14 and layer 20

```
Conv module done
Total cycle: 80496
    READ  | f: 16384, b: 100, w: 51200
 COMPUTE | 12812


=== Validation ===

Validation for layer 14
Loading answers from file: ../../inout_data_sw/log_feamap/CONV14_output_32b.hex

Result is correct!

Validation done.
```

- Note that the layer 14 and layer 20 has **no following ReLU**
  - Tolerate the difference of 1 when the output is negative
  - Considering the quantization policy

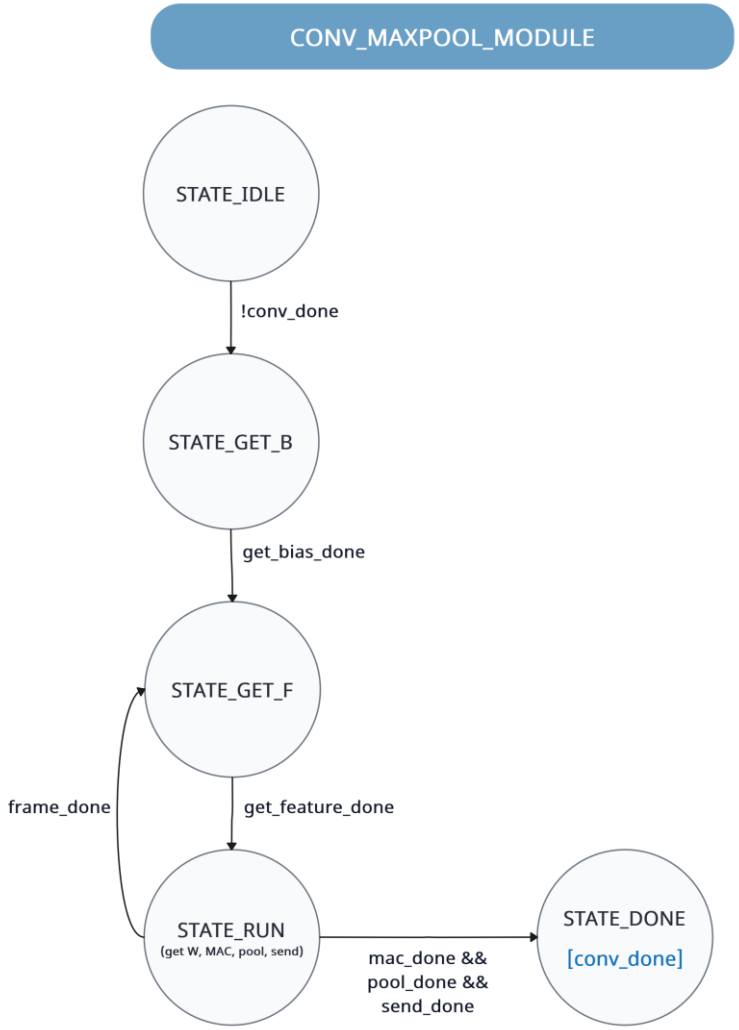
3. Hardware Design: 3x3 conv with IFM tile

- Having all IFMs and weights of the layer within the module is **NOT practical**.

- Leverage **IFM tile** and **streaming weights**.
  - Do not have to store incoming weights.
  - Since our convolution model is input stationary.

3. Hardware Design: 3x3 conv with IFM tile

- **Module FSM**
- IFM buffer contains **4x4xNi** data
- Computing with **streaming weights** performs better than store-then-load style



3. Hardware Design: 3x3 conv with IFM tile – Validation

- Verify **layer 06's result** with the **layer08's input** hex file.

- Vivado project and the validation results are archived.
  - '1_Code/2_RTL_Simulation/tile_tb'
  - '1_Code/4_Captured_Results (Waveforms, Utilization)/tiling_results/'

```
=== Validation <IFM TILING> ===

Validation for layer 6
Loading answers from file: ../../inout_data_sw/log_feamap/CONV08_input_32b.hex

Result is correct!

Validation done.
```

4. Team Plan

Category	Item	February				March					April				May			
		w1	w2	w3	w4	w1	w2	w3	w4	w5	w1	w2	w3	w4	w1	w2	w3	w4
Study	가속기 관련 논문 Study																	
Algorithm 구현 및 평가	Quantization방법 결정																	
	quantization 구현																	
	Data Reordering																	
Test Bench 설계 및 검증	single/multi layer TB																	
	TB update w/ fixed C-model																	
H/W 설계 및 FPGA 구현	Block Diagram 작성																	
	Shared global buffer feasibility 확인																	
	RTL 설계																	
	RTL 검증 및 FPGA 최적화																	

5. More about Dataflow

BRAM Resources

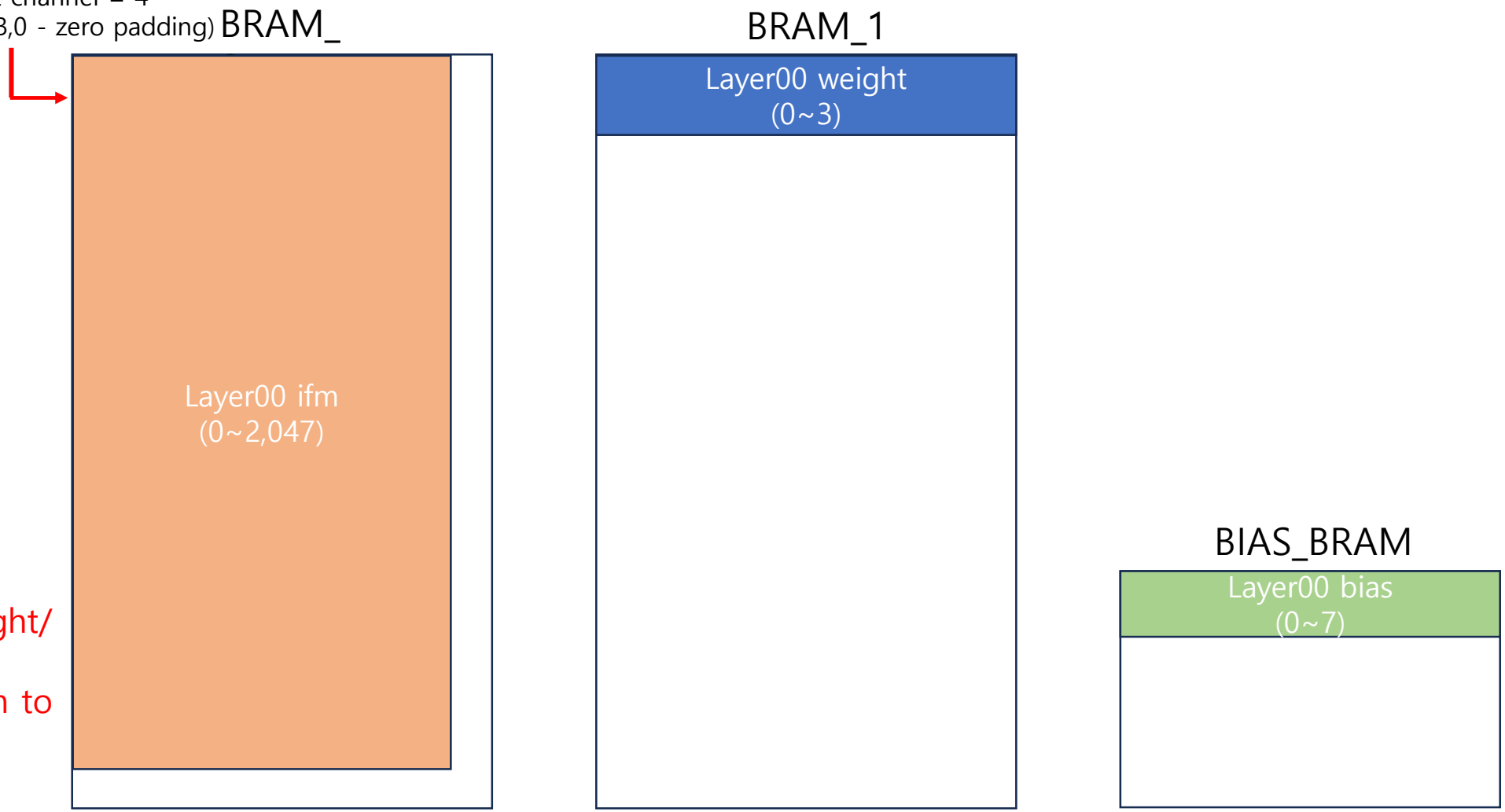
- 4,860 Kbits in total
- 8.192 Kbits for yolo_engine.v
- Can utilize approximately 4,850 Kbits for BRAM_top.v

- BRAM0, BRAM1 : 1152*2080 each -> 2,396,160 bits *2
- BIAS_BRAM : 32*384 -> 8,192 bits

- 4,800.512 Kbits used in BRAM_top.v

Layer00

Input channel = 4
(R,G,B,0 - zero padding)



<layer signals>
weight_i_start_addr = 2,048
bias_i_start_addr = 8
Ifm_i_start_addr = 4
weight_i_which_bram = 0
Ifm_i_which_bram = 1

weight_o_start_addr = 0
bias_o_start_addr = 0
Ifm_o_start_addr = 0
weight_o_which_bram = 1
Ifm_o_which_bram = 1

DMA works only for weight/ bias
= No need to offload ifm to DRAM

Address size : 2,080
Word size : 1152(144*8bits)

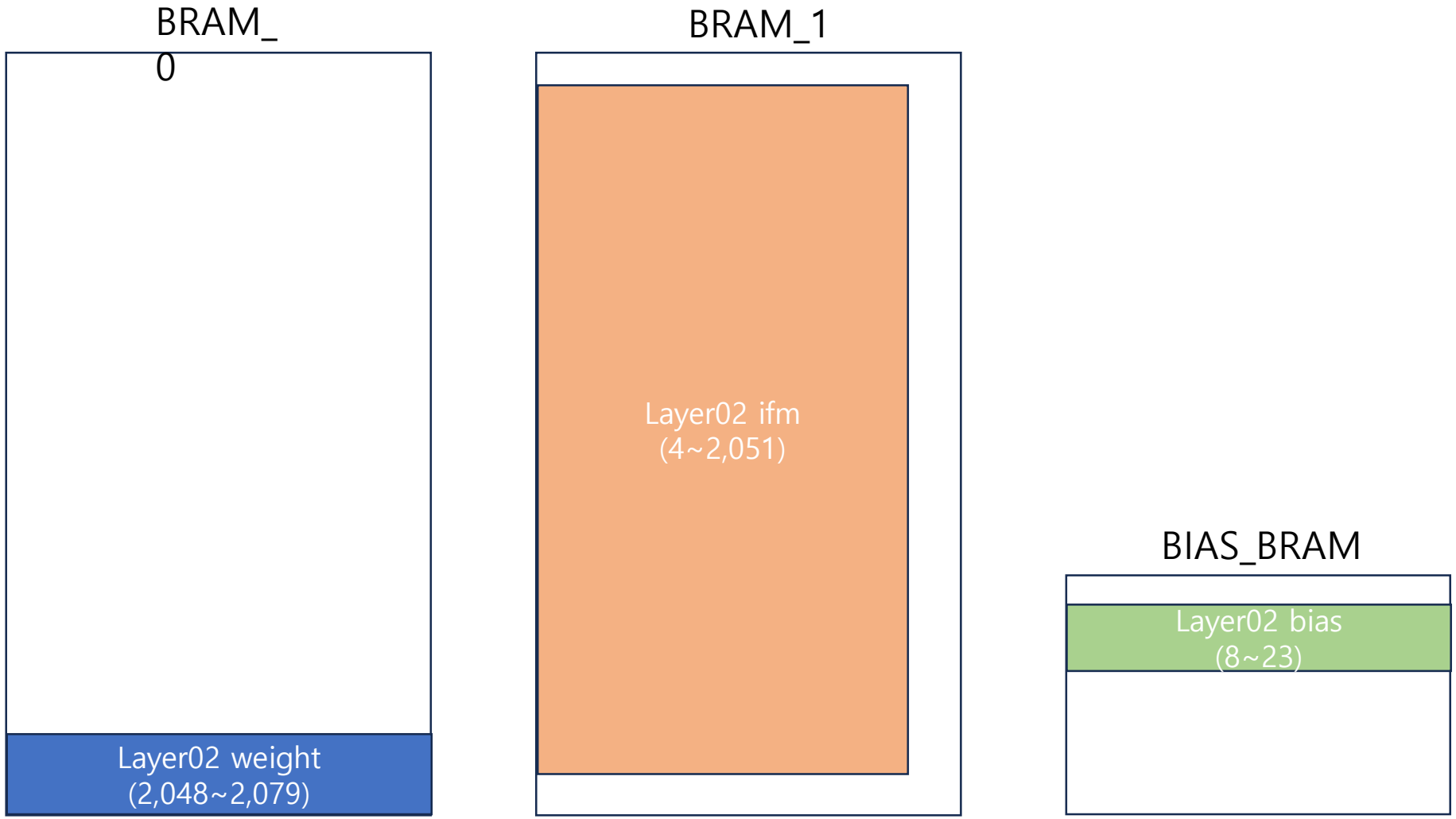
Address size : 384
Word size : 1152(144*8bits

Layer02

<layer signals>
weight_i_start_addr = 1,024
bias_i_start_addr = 24
Ifm_i_start_addr = 0
weight_i_which_bram = 0
Ifm_i_which_bram = 0

weight_o_start_addr = 2,048
bias_o_start_addr = 8
Ifm_o_start_addr = 4
weight_o_which_bram = 0
Ifm_o_which_bram = 1

Can not overlap weight read , weight write and ofm write (have only 2 ports in BRAM 0)



Address size : 2,080
Word size : 1152(144*8bits)

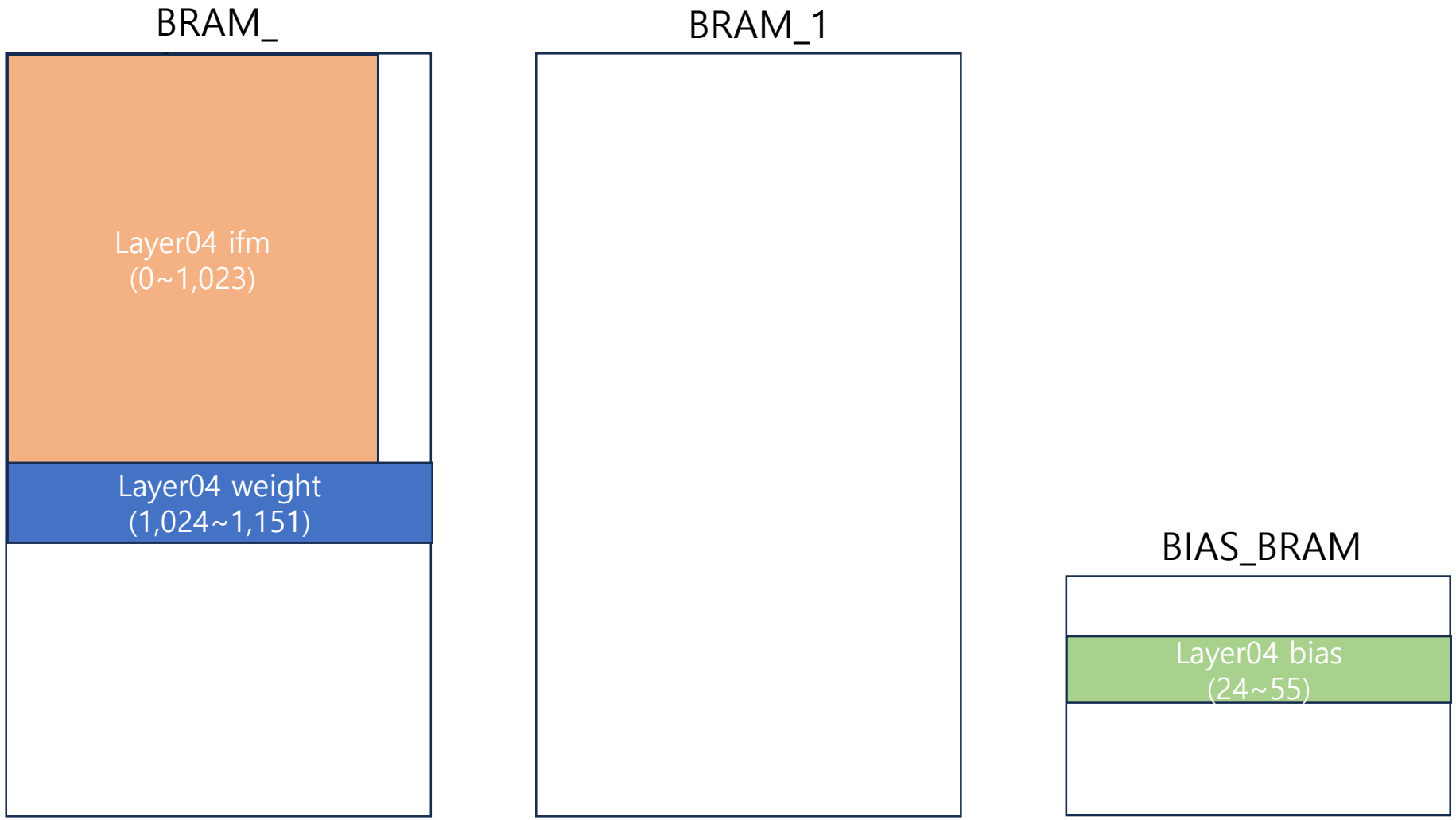
Address size : 384
Word size : 1152(144*8bits

Layer04

<layer signals>
weight_i_start_addr = 1,568
bias_i_start_addr = 56
Ifm_i_start_addr = 0
weight_i_which_bram = 0
Ifm_i_which_bram = 1

weight_o_start_addr = 1,024
bias_o_start_addr = 24
Ifm_o_start_addr = 0
weight_o_which_bram = 0
Ifm_o_which_bram = 0

Can not overlap weight read , ifm read, weight load (have only 2 ports in BRAM1)



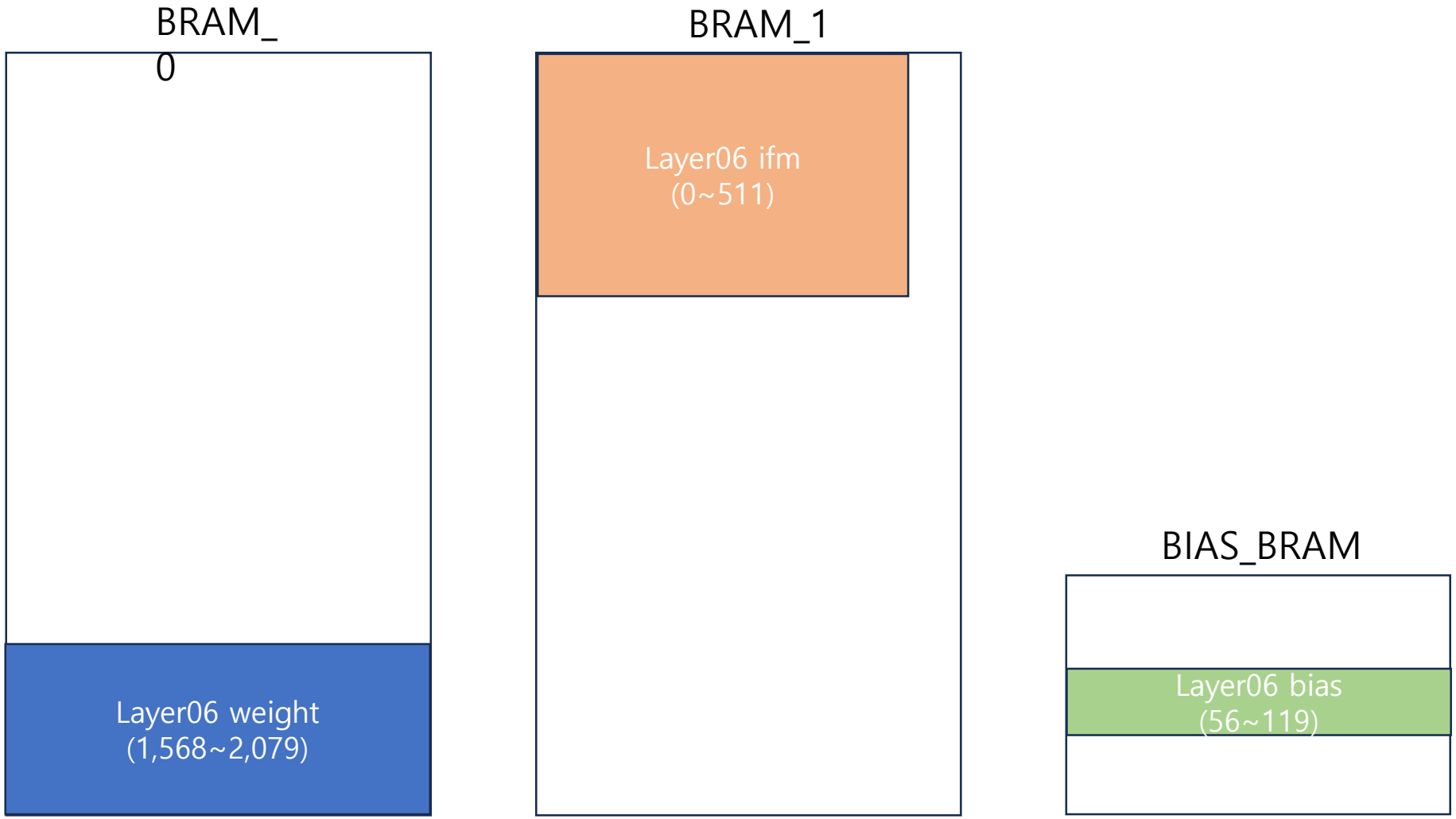
Address size : 2,080
Word size : 1152(144*8bits)

Address size : 384
Word size : 1152(144*8bits

Layer06

<layer signals>
weight_i_start_addr = 0
bias_i_start_addr = 120
Ifm_i_start_addr = 1,825
weight_i_which_bram = 0
Ifm_i_which_bram = 1

weight_o_start_addr = 1,568
bias_o_start_addr = 56
Ifm_o_start_addr = 0
weight_o_which_bram = 0
Ifm_o_which_bram = 1



Address size : 2,080
Word size : 1152(144*8bits)

Address size : 384
Word size : 1152(144*8bits

Layer08

<layer signals>
weight_i_start_addr = 0(after co
nv)
bias_i_start_addr = 0
Ifm_i_start_addr = 0
weight_i_which_bram = 0(after c
onv)
Ifm_i_which_bram = 1

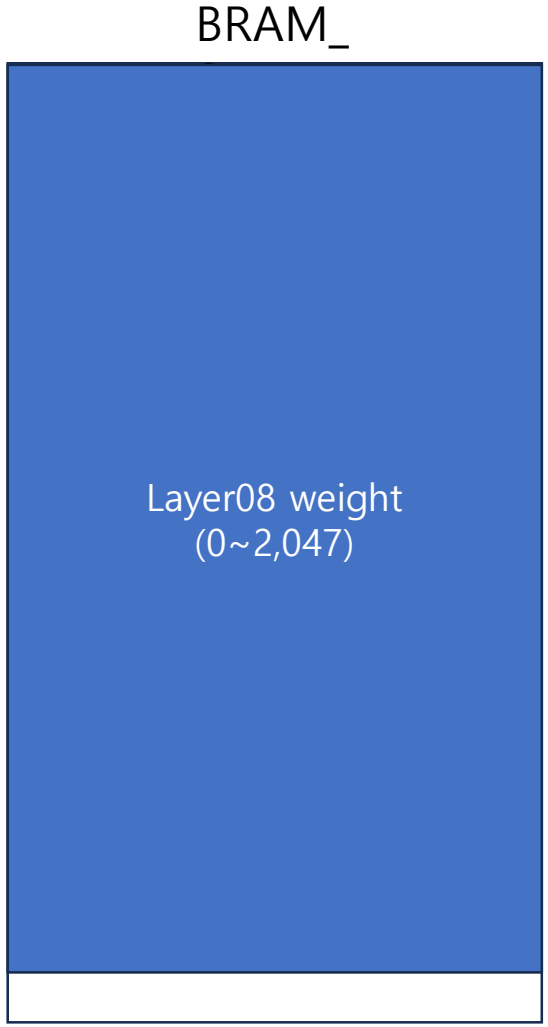
weight_o_start_addr = 0
Bias_o_start_addr = 120
Ifm_o_start_addr = 1,825
weight_o_which_bram = 0
Ifm_o_which_bram = 1

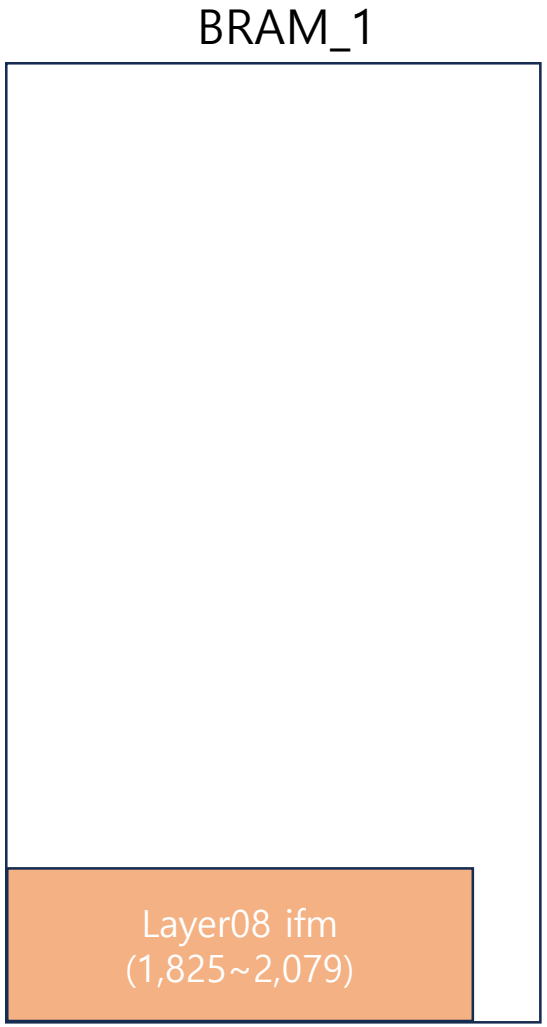
Before start, need to wait fo
r weight 08

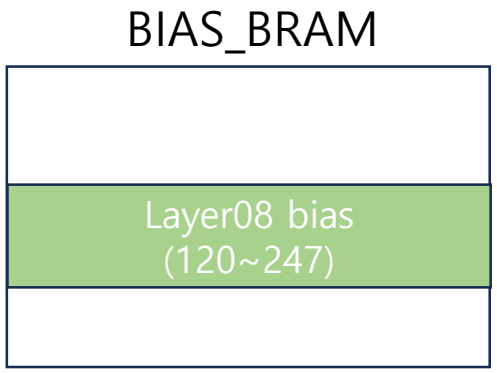
Store layer08 ofm for route

After conv, load weight 10 (
partial weight of layer 10, it'
s size is too large)

Address size : 2,080
Word size : 1152(144*8bits)







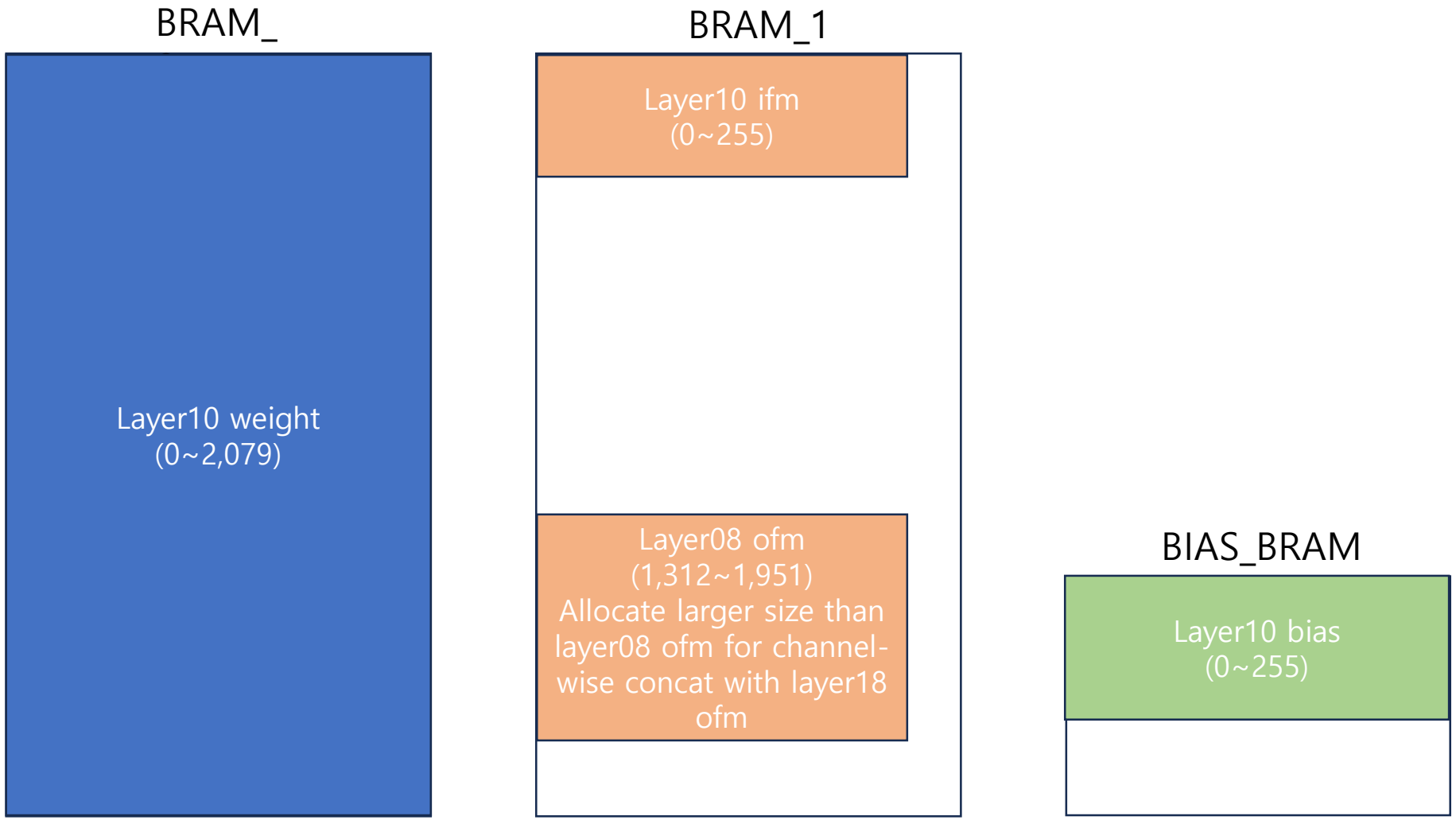
Address size : 384
Word size : 1152(144*8bits

Layer10

<layer signals>
weight_i_start_addr = 0 (after 4iters)
bias_i_start_addr = 256
Ifm_i_start_addr = 256
weight_i_which_bram = 0 (after 4 iters)
Ifm_i_which_bram = 1

weight_o_start_addr = 0
Bias_o_start_addr = 0
Ifm_o_start_addr = 0
weight_o_which_bram = 0
Ifm_o_which_bram = 1

Iterate 4 times
2,080 * 4 > 8,192



Address size : 2,080
Word size : 1152(144*8bits)

Address size : 384
Word size : 1152(144*8bits

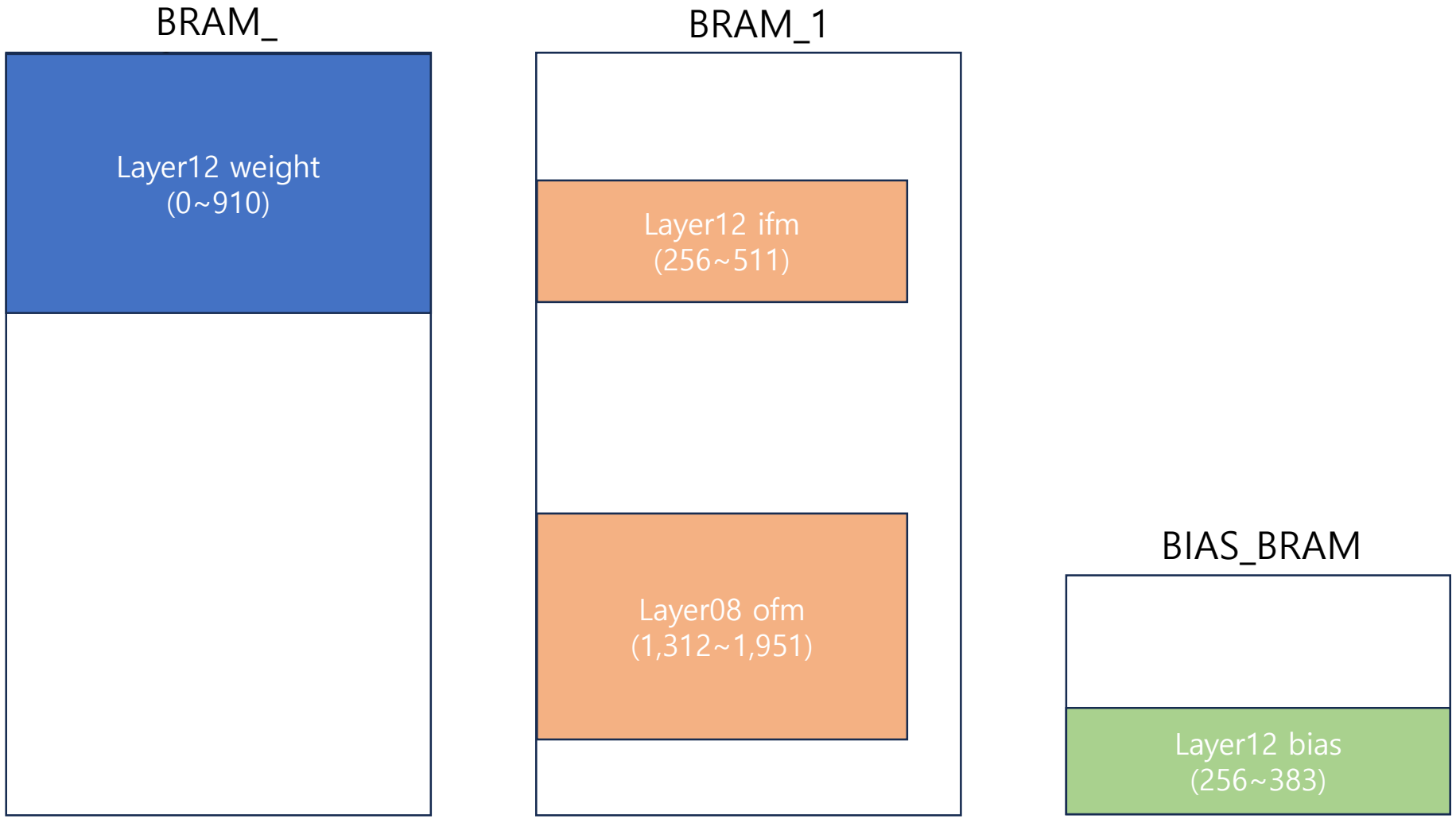
Layer12

<layer signals>
weight_i_start_addr = 0 bias_i_st
art_addr = 0
Ifm_i_start_addr = 512
weight_i_which_bram = 0
Ifm_i_which_bram = 1

weight_o_start_addr = 0
Bias_o_start_addr = 256
Ifm_o_start_addr = 256
weight_o_which_bram = 0
Ifm_o_which_bram = 1

Load layer13 weight after la
yer12 done

Store layer12 ofm for route



Address size : 2,080
Word size : 1152(144*8bits)

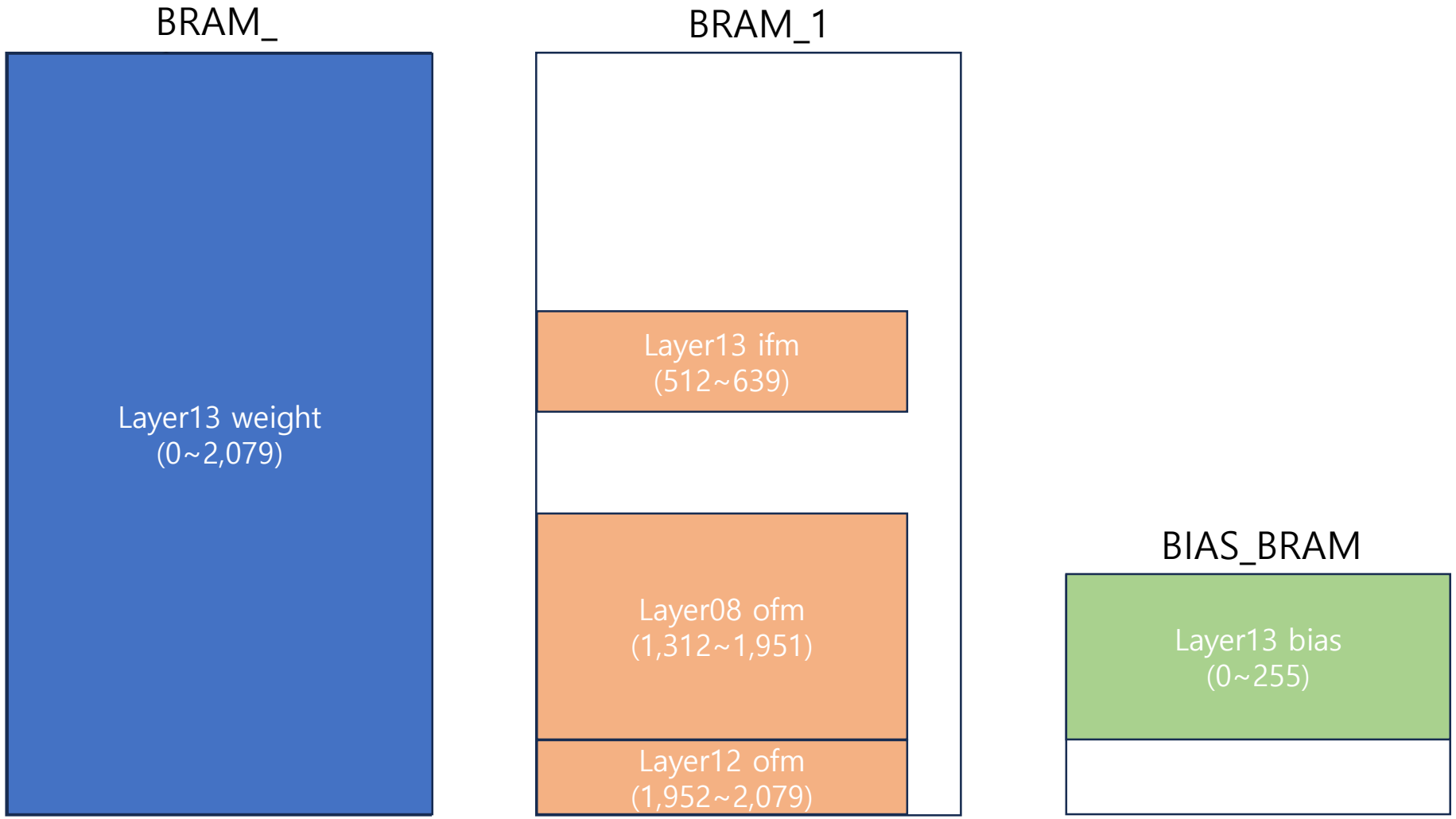
Address size : 384
Word size : 1152(144*8bits

Layer13

<layer signals>
weight_i_start_addr = 0 (4 iters)
bias_i_start_addr = 256
Ifm_i_start_addr = 0
weight_i_which_bram = 0 (4 iters)
Ifm_i_which_bram = 1

weight_o_start_addr = 0
Bias_o_start_addr = 0
Ifm_o_start_addr = 512
weight_o_which_bram = 0
Ifm_o_which_bram = 1

iterate 4 times



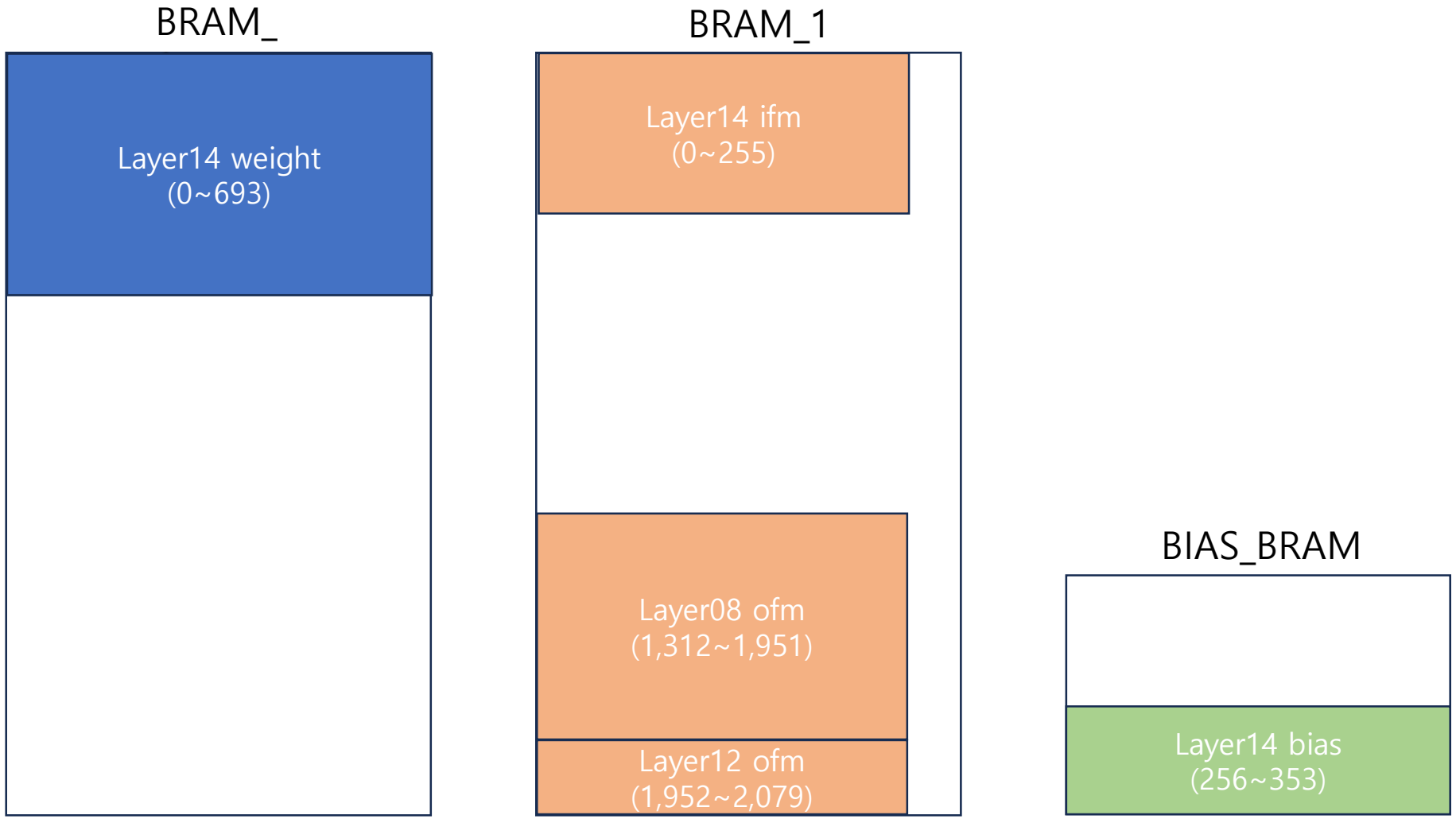
Address size : 2,080
Word size : 1152(144*8bits)

Address size : 384
Word size : 1152(144*8bits

Layer14

<layer signals>
weight_i_start_addr = 694 bias_i
_start_addr = 0
Ifm_i_start_addr = 256
weight_i_which_bram = 0
Ifm_i_which_bram = 1

weight_o_start_addr = 0
Bias_o_start_addr = 256
Ifm_o_start_addr = 0
weight_o_which_bram = 0
Ifm_o_which_bram = 1



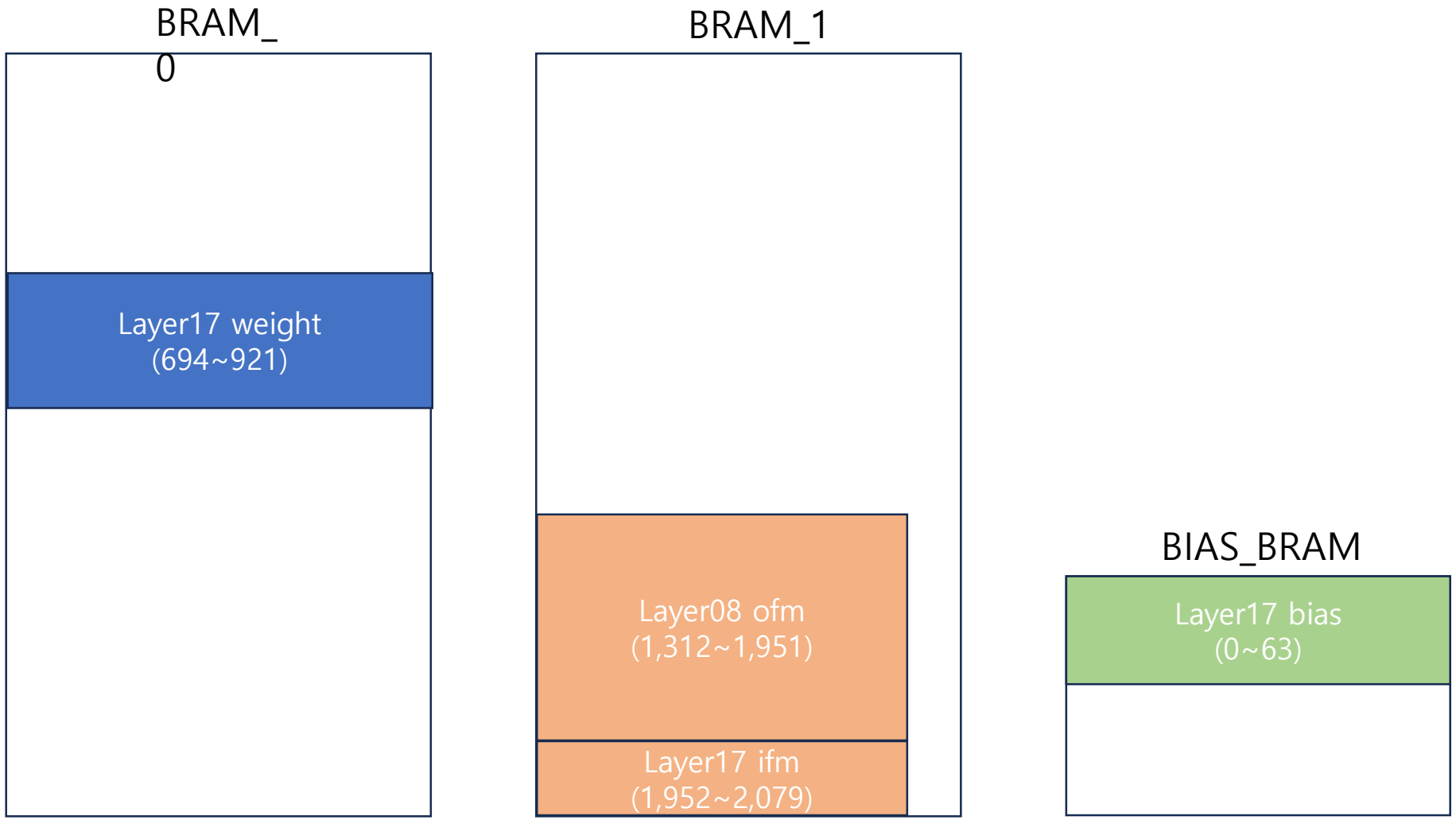
Address size : 2,080
Word size : 1152(144*8bits)

Address size : 384
Word size : 1152(144*8bits

Layer17

<layer signals>
weight_i_start_addr = 922 bias_i
_start_addr = 64
Ifm_i_start_addr = 0
weight_i_which_bram = 0
Ifm_i_which_bram = 1

weight_o_start_addr = 0
Bias_o_start_addr = 0
Ifm_o_start_addr = 1,952
weight_o_which_bram = 0
Ifm_o_which_bram = 1



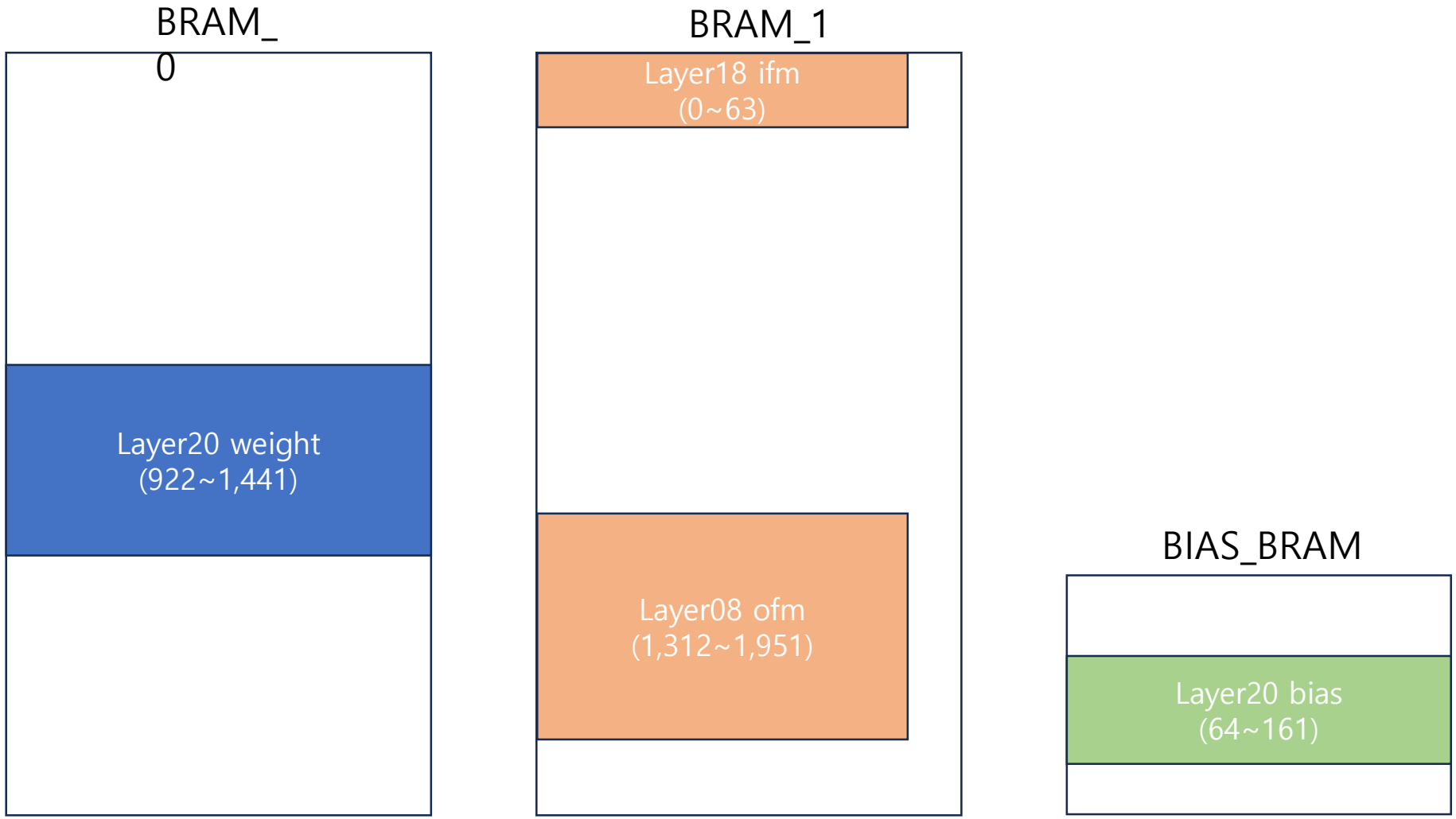
Address size : 2,080
Word size : 1152(144*8bits)

Address size : 384
Word size : 1152(144*8bits

Layer18 - Upsampling

<layer signals>
weight_i_start_addr = x bias_i_st
art_addr = x
Ifm_i_start_addr = 1,312
weight_i_which_bram = x
Ifm_i_which_bram = 1

weight_o_start_addr = x
Bias_o_start_addr = x
Ifm_o_start_addr = 0
weight_o_which_bram = x
Ifm_o_which_bram = 1



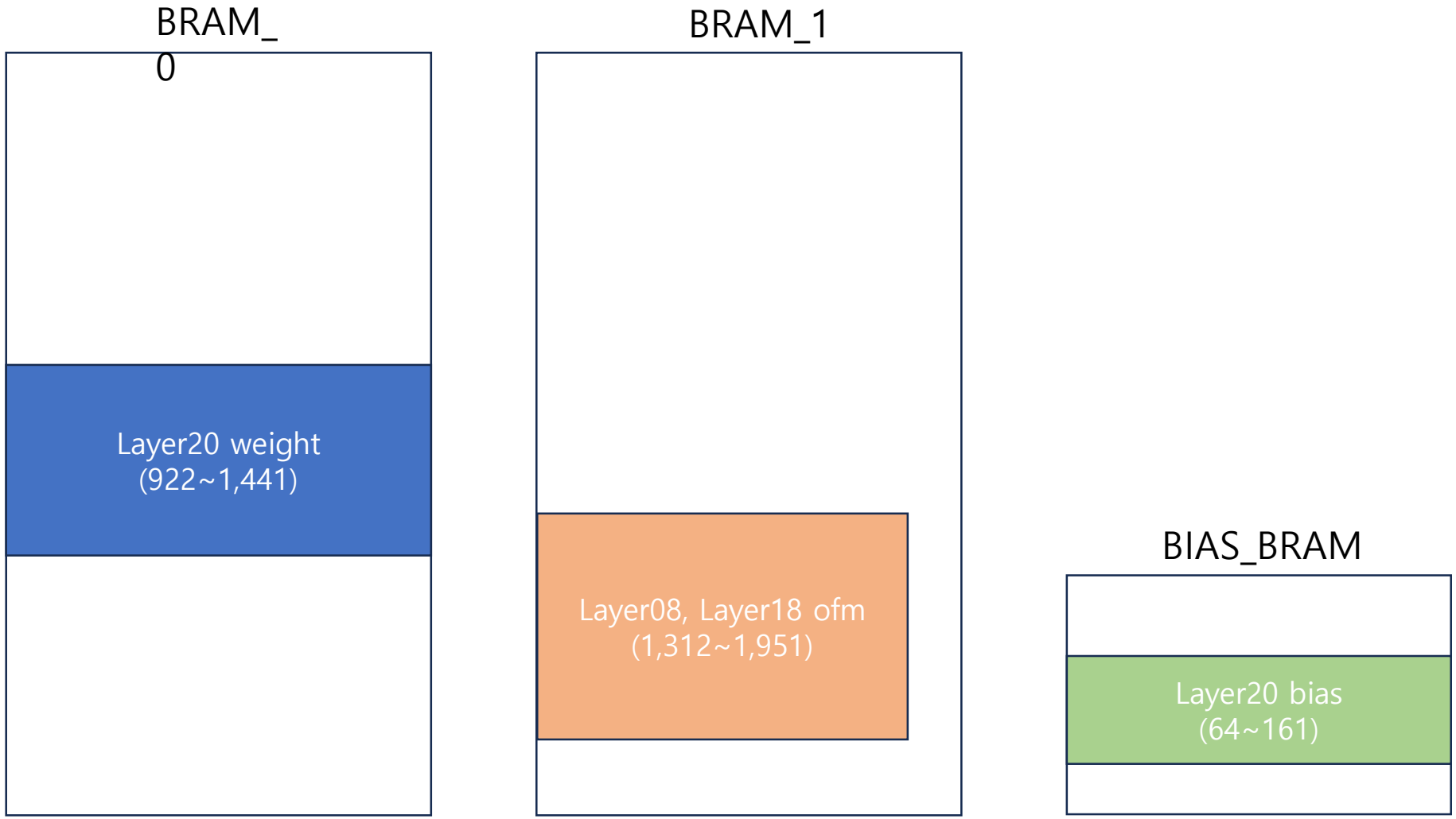
Address size : 2,080
Word size : 1152(144*8bits)

Address size : 384
Word size : 1152(144*8bits

Layer20

<layer signals>
weight_i_start_addr = x bias_i_st
art_addr = x
Ifm_i_start_addr = 1
weight_i_which_bram = x
Ifm_i_which_bram = 1

weight_o_start_addr = 922
Bias_o_start_addr = 64
Ifm_o_start_addr = 1,312
weight_o_which_bram = 0
Ifm_o_which_bram = 1



Address size : 2,080
Word size : 1152(144*8bits)

Address size : 384
Word size : 1152(144*8bits

Thank You!