Project 2 – Matrix Multiplication Unit

EE361 – Digital Circuit Design and Language

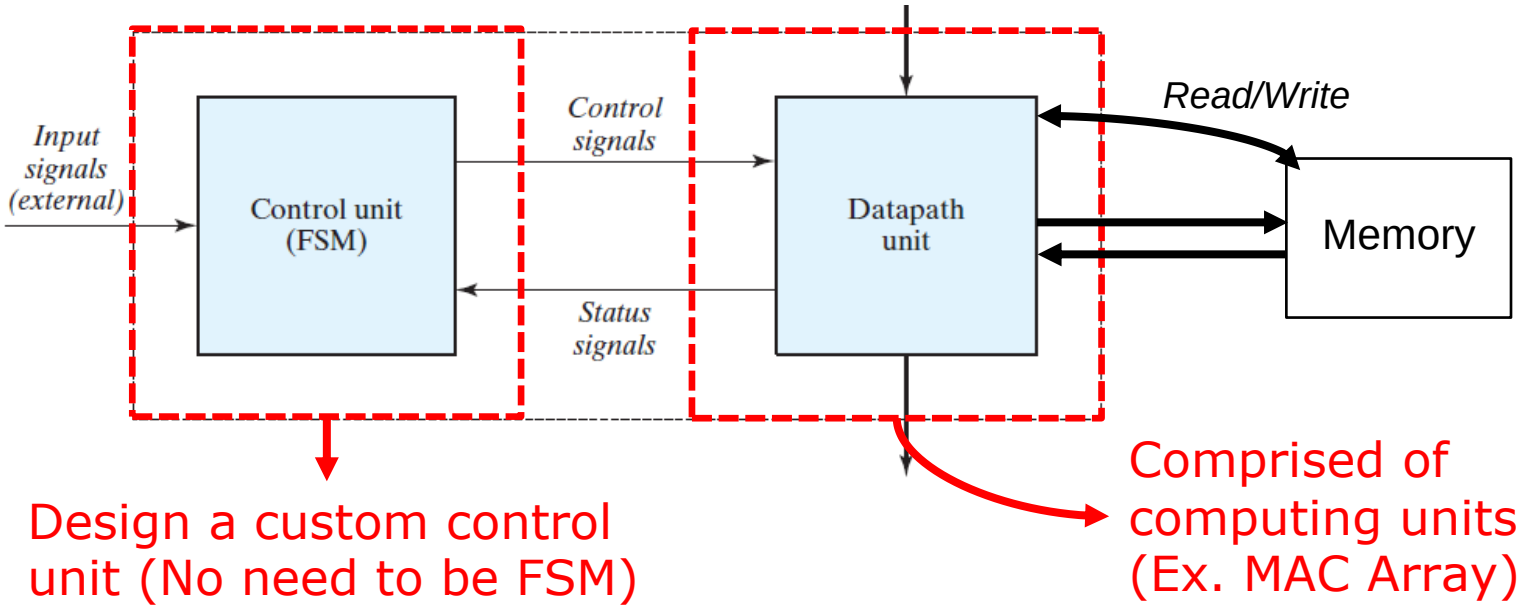
Kyung Hee University
Electrical Engineering

Spring 2025

Seungkyu Choi (seungkc@khu.ac.kr)

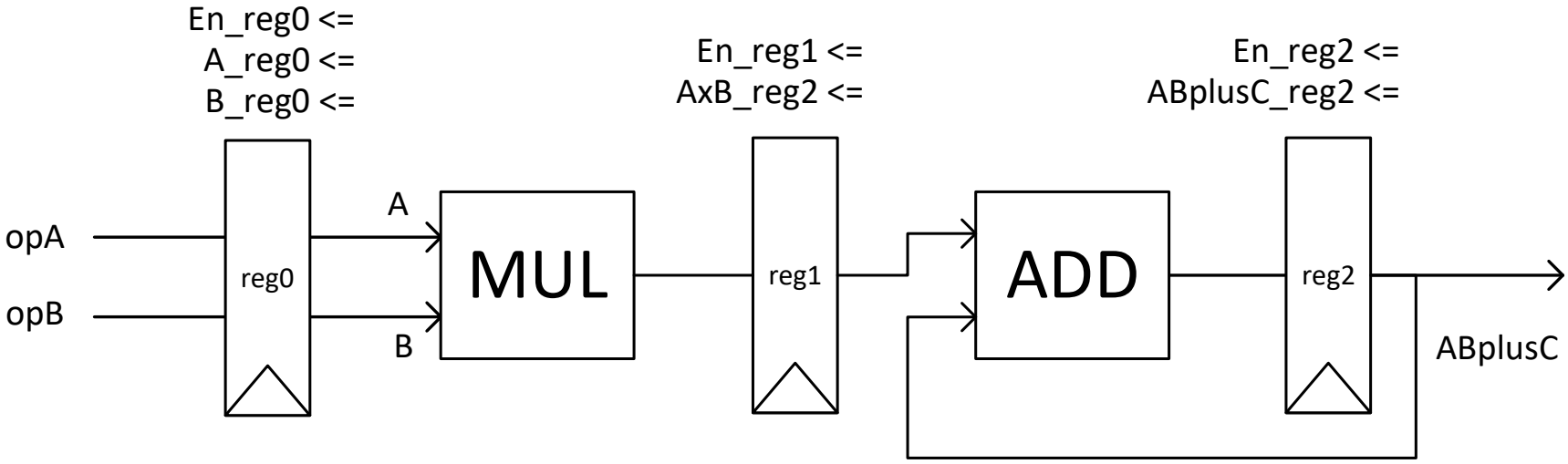
System Design

- Control unit + datapath + memory structure
  - ✓ The datapath will be the processing part, which is comprised of multiple computing units.
  - ✓ The control unit can be implemented in any forms.
  - ✓ Input memory for read, output memory for write



Utilizing a MAC Unit Design

- A Pipelined Multiply-and-Accumulate (MAC) Unit
  - ✓ The MAC unit must be implemented with a pipelined system. (+MUL and ADD must be pipelined)
  - ✓ How to integrate the control system?

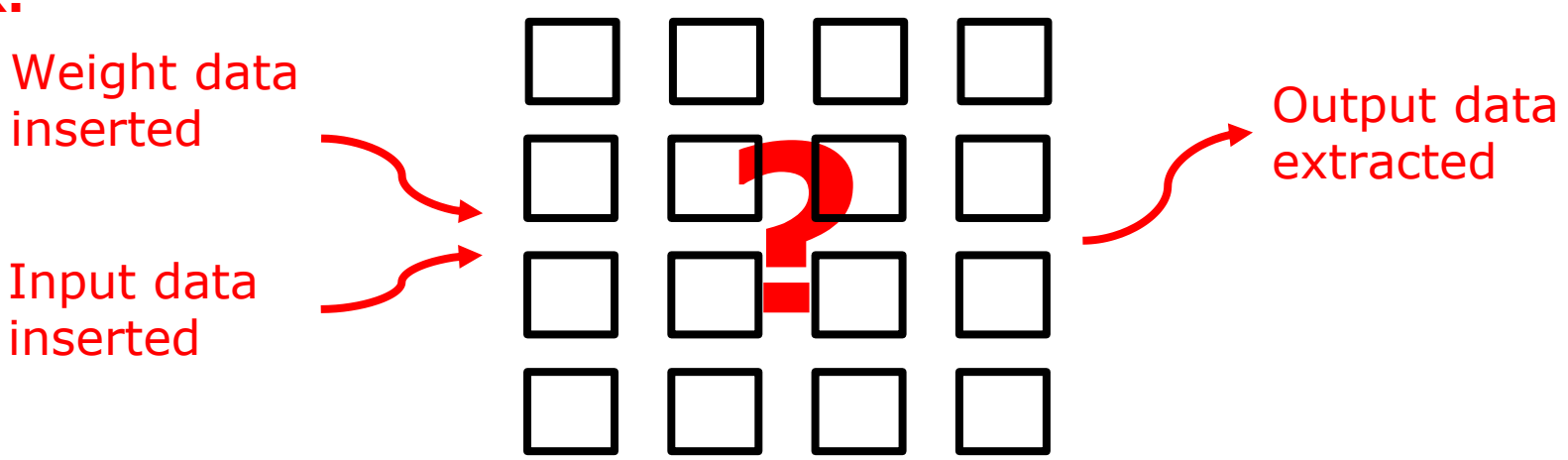


Prj 2. 4x4 MAC Array Accelerator

- **I (TxN) X W (NxM) = O (TxM) Matrix Multiplier**

(M, N, T = 1~8)

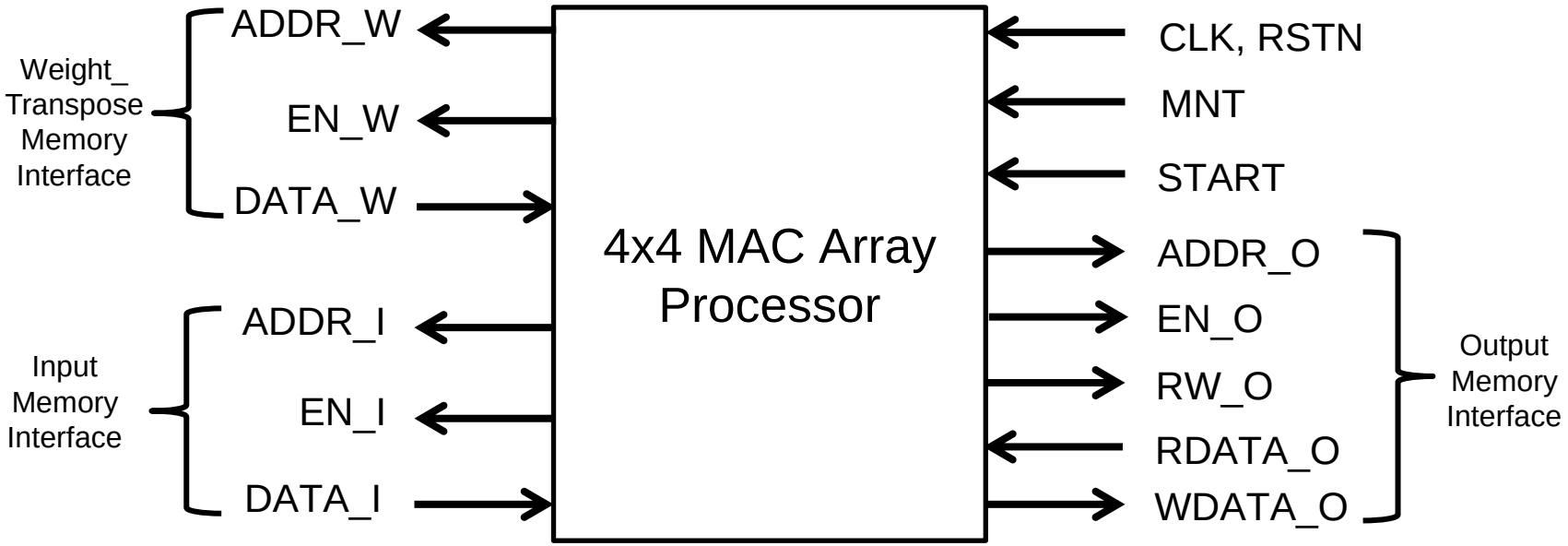
Ex.



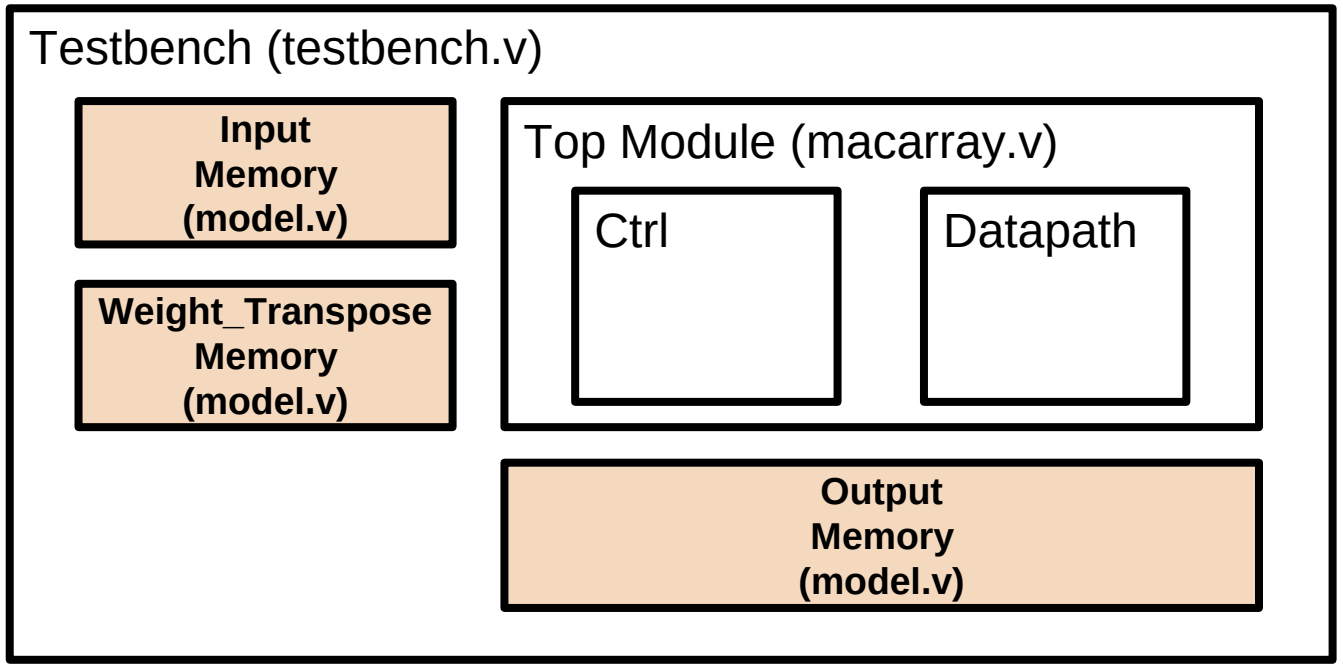
- Prepared (TxN) input matrix and (NxM) weight matrix are loaded from the input and weight memory, respectively.
- Output should be stored into the output memory in groups of 4: 4 output data are stored in the same address.
- Detailed load and store processes of the data will be explained.
- M, N, T is a variable that can be any value from 1 to 8.

Prj 2. Top Module Interface

In CLK, RSTN	Clock	**Out EN_W**	Weight_Tr Mem. Enable
In [11:0] MNT	4-bit M,N,T values	**In [31:0] RDATA_W**	Weight_Tr Data Bus
In START	Start Signal	**Out [3:0] ADDR_O**	Out Mem. Address
Out [3:0] ADDR_I	Input Mem. Address	**Out EN_O**	Out Mem. Enable
Out EN_I	Input Mem. Enable	**Out RW_O**	Out Mem. Read(0)/Write(1)
In [31:0] RDATA_I	Input Data Bus	**In [63:0] RDATA_O**	Out Mem. Read Data Bus
Out [3:0] ADDR_W	Weight_Tr Mem. Address	**Out [63:0] WDATA_O**	Out Mem. Write Data Bus



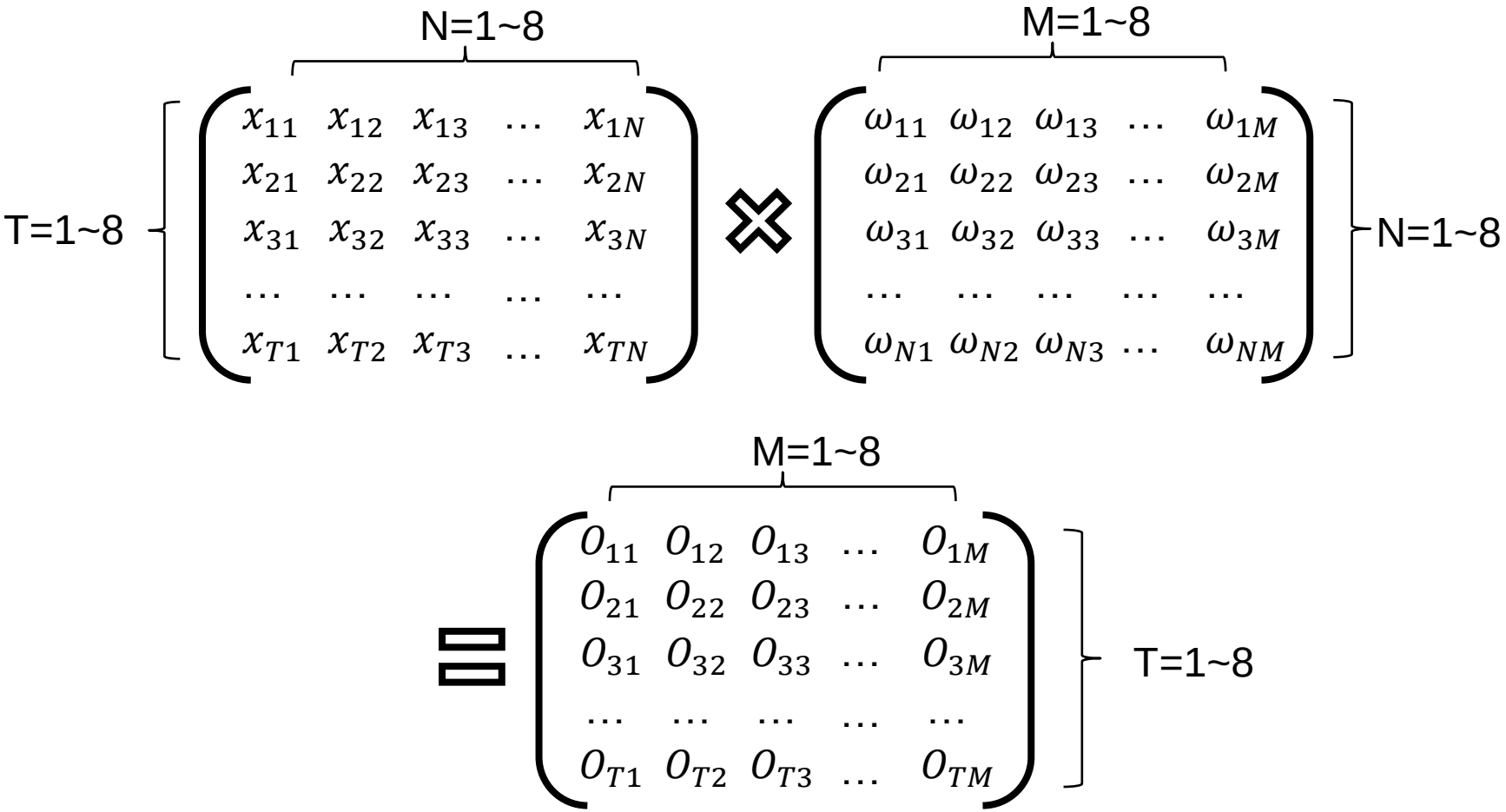
Prj 2. Testbench I/O



Type	Width	Name	
Input	1	CLK	Clock
Input	1	RSTN	Asynchronous reset negative
Input	1	Start	Start signal
Input	12	MNT	4-bit concatenated M, N and T values ranging from 1~8 (Ex. 12'h568 means M=5, N=6, T=8)

Prj 2. Memory Allocation Rule

- Read and Write data should be Input, Weight, Output participating the matrix multiplication process shown below.



$$\begin{pmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1N} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2N} \\ x_{31} & x_{32} & x_{33} & \dots & x_{3N} \\ \dots & \dots & \dots & \dots & \dots \\ x_{T1} & x_{T2} & x_{T3} & \dots & x_{TN} \end{pmatrix} \times \begin{pmatrix} \omega_{11} & \omega_{12} & \omega_{13} & \dots & \omega_{1M} \\ \omega_{21} & \omega_{22} & \omega_{23} & \dots & \omega_{2M} \\ \omega_{31} & \omega_{32} & \omega_{33} & \dots & \omega_{3M} \\ \dots & \dots & \dots & \dots & \dots \\ \omega_{N1} & \omega_{N2} & \omega_{N3} & \dots & \omega_{NM} \end{pmatrix} = \begin{pmatrix} O_{11} & O_{12} & O_{13} & \dots & O_{1M} \\ O_{21} & O_{22} & O_{23} & \dots & O_{2M} \\ O_{31} & O_{32} & O_{33} & \dots & O_{3M} \\ \dots & \dots & \dots & \dots & \dots \\ O_{T1} & O_{T2} & O_{T3} & \dots & O_{TM} \end{pmatrix}$$

Prj 2. Input and Weight Utilization

- Input and weight should be prepared with hex files as 8x8 matrices. The file names 'input.hex' and 'weight_transpose.hex' should not be changed.
  (Only M,N,T signals can control the sizes of input and weight.)
- A single data is 8-bit 2's complement data (2 hex num).
- Be careful of the weight since it should be stored and inserted in a transposed version.

```
// ------------------------------------------
// Input (T X N) and Weight (N X M) test matrices should be stored as.
// ------------------------------------------
// Caution : Assumption : input files have hex data like below.
//           Input      : (1,1) (1,2) ... (1,N)
//          (T x N)       (2,1) (2,2) ... (2,N)
//                        (3,1) (3,2) ... (3,N)
//                         ...   ...  ...  ...
//                        (T,1) (T,2) ... (T,N)
//
//    Weight_transpose   : (1,1) (1,2) ... (1,N)
//          (M x N)       (2,1) (2,2) ... (2,N)
//                        (3,1) (3,2) ... (3,N)
//                         ...   ...  ...  ...
//                        (M,1) (M,2) ... (M,N)
//
//  hex files will be  -> line1: (1,1)(1,2)(1,3)(1,4)(1,5)(1,6)(1,7)(1,8)
//                     -> line2: (2,1)(2,2)(2,3)(2,4)(2,5)(2,6)(2,7)(2,8)
//                     -> line3: (3,1)(3,2)(3,3)(3,4)(3,5)(3,6)(3,7)(3,8)
//                         ...    ...  ...  ...  ...  ...  ...  ...  ...
//                     -> line8: (8,1)(8,2)(8,3)(8,4)(8,5)(8,6)(8,7)(8,8)
```

Prj 2. Input and Weight Utilization

- Example (MNT=12'h568)

input.hex

```
1  01020304050607 08
2  090A0B0C0D0E0F01
3  0102030405060708
4  090A0B0C0D0E0F01
5  0102030405060708
6  090A0B0C0D0E0F01
7  0102030405060708
8  090A0B0C0D0E0F01
```

$=$

N=6

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 & 13 & 14 & 15 & 1 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 & 13 & 14 & 15 & 1 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 & 13 & 14 & 15 & 1 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 & 13 & 14 & 15 & 1 \end{bmatrix}$$

T=8

→

[macarray]

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 9 & 10 & 11 & 12 & 13 & 14 \\ 1 & 2 & 3 & 4 & 5 & 6 \\ 9 & 10 & 11 & 12 & 13 & 14 \\ 1 & 2 & 3 & 4 & 5 & 6 \\ 9 & 10 & 11 & 12 & 13 & 14 \\ 1 & 2 & 3 & 4 & 5 & 6 \\ 9 & 10 & 11 & 12 & 13 & 14 \end{bmatrix}$$

weight_transpose.hex

```
1  01020304050607 08
2  0102030405060708
3  0102030405060708
4  0102030405060708
5  0102030405060708
6  0102030405060708
7  0102030405060708
8  0102030405060708
```

$=$

N=6

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{bmatrix}$$

M=5

transpose →

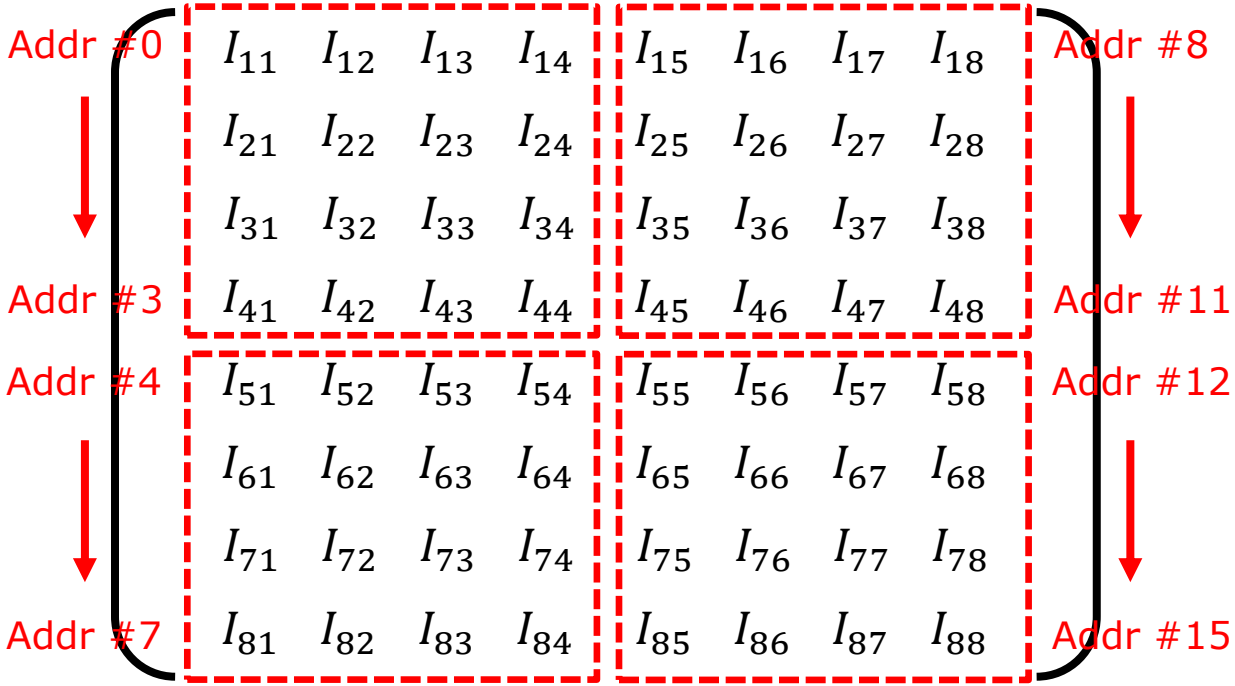
×

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 \\ 5 & 5 & 5 & 5 & 5 \\ 6 & 6 & 6 & 6 & 6 \end{bmatrix}$$

Prj 2. In/W Memory Allocation Rule

- The data from hex file is automatically stored as below, having a 32-bit width per address.
- Input and weight have a fixed read bandwidth of 32 bits per cycle.

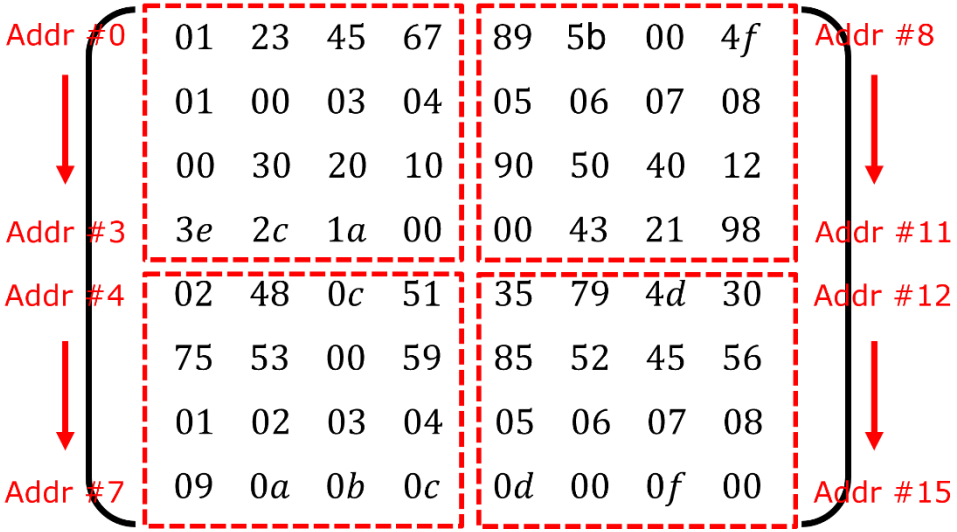
Example) *Input*

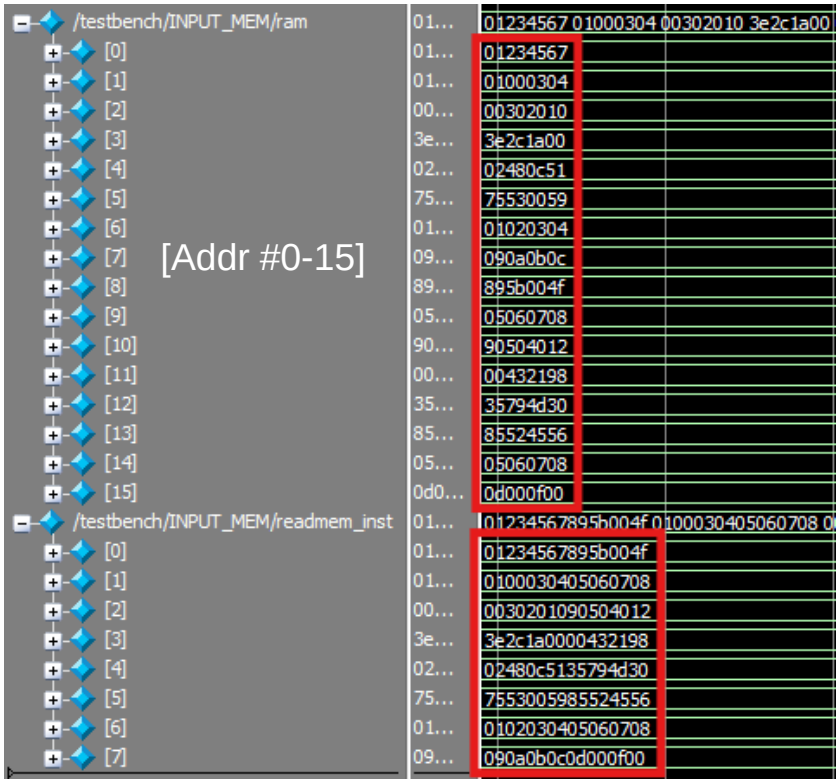


[Same for *Weight*]

Prj 2. In/W Memory Allocation Rule

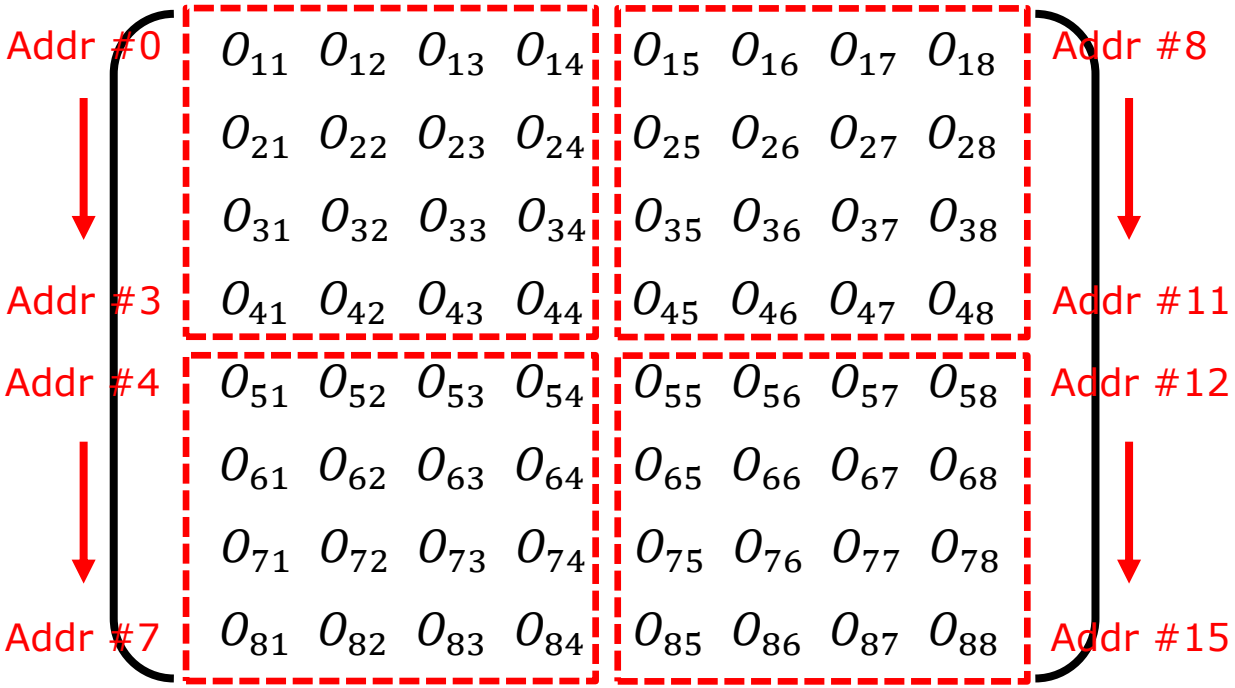
- Example





Prj 2. Output Memory Allocation Rule

- Output data is stored in the memory following the rule below, having a 64-bit width per address (16-bit width 4 data).
- Addressing is fixed, and unused row/column data should be set to zero.



Prj 2. Output Monitoring

- OUT should be stored as the same manner stated in the memory allocation rule.
- A single data is 16-bit 2's complement data (4 hex num).
- You do not need to consider overflow cases.
- You should not modify any of the testbench file and model file to access output directly when submitting your source code.

Prj 2. SDC table

Frequency (MHz)	Input Delay(ns)	Output Delay(ns)
~ 50	4.0 ~ 6.0	4.0 ~ 6.0
50 ~ 100	2.0 ~ 4.0	2.0 ~ 4.0
100 ~ 150	1.5 ~ 2.5	1.5 ~ 2.5
150 ~ 200	1.0 ~ 2.0	1.0 ~ 2.0

Submission Rule (Due:6/16 11:59 PM)

- Submission file: Zip File - Verilog codes and reports

File Name: prj2_xx.zip (xx: team number 01~24)

- Verilog Codes: **every files**
  (Do not change the name of the skeleton files.)
- Do not modify any of the testbench and model files when submitting your code.
- You can test by changing the values of the weight and input hex files. Be careful to follow the same rule indicated in the testbench code when making the weight and input matrix. (!!! Weight should be stored in transposed version.)
- Two reports should be prepared: An **8 page-limit word/hwp file** and a **6 min long PPT**
  - Reports should include any details of your design.
  - Screen captures of your Quartus results and analysis are mandatory.
    (Device: CycloneⅢ EP3C120F484C7)
  - Indicate the overview of the control system and key strengths of your design in the PPT file. Submitted version cannot be modified and will be the final version that will be used for the presentation.