NYCU-EE IC LAB – Spring 2024

Lab05 Exercise

Design: Matrix convolution, max pooling and transposed convolution

Data Preparation

1. Extract files from TA's directory:
   % tar xvf ~iclabTA01/Lab05.tar
2. The extracted LAB directory contains:
   a. Practice/
   b. Exercise/

Design Description

Convolution, max pooling and transposed convolution are three commonly used operations in deep learning, applied to tasks such as image processing, computer vision, and convolutional neural networks (CNNs). They play significant roles in image processing, feature extraction, and generation.

In this lab, you need to create a calculator that can calculate the convolution, max pooling and transposed convolution of multiple matrices.

Convolution:

Convolution is a mathematical operation used for image processing and feature extraction. In a convolutional layer of a neural network, a small matrix known as a convolutional kernel (or filter) is slid over an input image, performing element-wise multiplication and summation. This process helps capture local features in an image, such as edges and textures. Convolutions are highly useful in image processing as they reduce the number of parameters and retain important spatial information.

Example:

The sliding window operates sequentially and directionally from left to right and top to bottom. The output involves performing accumulation operations based on the corresponding location coordinates, resulting in a final output feature map.

kernel

K_{11}	K_{12}
K_{21}	K_{22}

Image

I_{11}	I_{12}	I_{13}
I_{21}	I_{22}	I_{23}
I_{31}	I_{32}	I_{33}

step 1:

I_{11}	I_{12}	I_{13}
I_{21}	I_{22}	I_{23}
I_{31}	I_{32}	I_{33}

Output

O_{11}	

$O_{11} = I_{11}*K_{11} + I_{12}*K_{12} + I_{21}*K_{21} + I_{22}*K_{22}$

step 2:

I_{11}	I_{12}	I_{13}
I_{21}	I_{22}	I_{23}
I_{31}	I_{32}	I_{33}

Output

O_{11}	O_{12}

$O_{12} = I_{12}*K_{11} + I_{13}*K_{12} + I_{22}*K_{21} + I_{23}*K_{22}$

step 3:

I_{11}	I_{12}	I_{13}
I_{21}	I_{22}	I_{23}
I_{31}	I_{32}	I_{33}

Output

O_{11}	O_{12}
O_{21}	

$O_{21} = I_{21}*K_{11} + I_{22}*K_{12} + I_{31}*K_{21} + I_{32}*K_{22}$

step 4:

I_{11}	I_{12}	I_{13}
I_{21}	I_{22}	I_{23}
I_{31}	I_{32}	I_{33}

Output

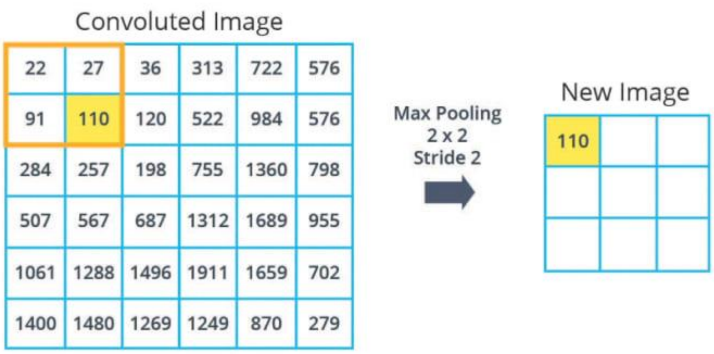
O_{11}	O_{12}
O_{21}	O_{22}

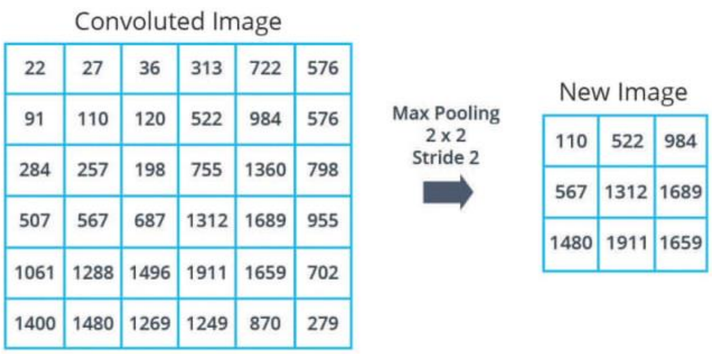
$O_{22} = I_{22}*K_{11} + I_{23}*K_{12} + I_{32}*K_{21} + I_{33}*K_{22}$

Max Pooling:

Max pooling is a pooling operation that calculates the maximum value for patches of a feature map, and uses it to create a down-sampled (pooled) feature map. It is usually used after a convolutional layer. It adds a small amount of translation invariance - meaning translating the image by a small amount does not significantly affect the values of most pooled outputs.

Example:





Transposed Convolution (or Deconvolution):

Transposed convolution is an operation used for image generation, up-sampling, and reconstruction. Despite its name, it's not the exact inverse of a convolution. Transposed convolutions operate inversely to convolutions and can transform low-dimensional feature maps into higher-dimensional ones, achieving up-sampling. They are often used for generative tasks like image synthesis, image segmentation, and are also employed in neural networks as deconvolution layers (up-sampling layers) to restore spatial resolution.

Example:

As the computation here involves a one-to-many relationship, during the accumulation process, instances of multiple values being summed up at the same location can occur.

kernel

K_{11}	K_{12}
K_{21}	K_{22}

Image

I_{11}	I_{12}	I_{13}
I_{21}	I_{22}	I_{23}
I_{31}	I_{32}	I_{33}

step 1:

Image

I_{11}	I_{12}	I_{13}
I_{21}	I_{22}	I_{23}
I_{31}	I_{32}	I_{33}

Output

$I_{11}*K_{11}$	$I_{11}*K_{12}$		
$I_{11}*K_{21}$	$I_{11}*K_{22}$		

step 2:

Image

I_{11}	I_{12}	I_{13}
I_{21}	I_{22}	I_{23}
I_{31}	I_{32}	I_{33}

Output

$I_{11}*K_{11}$	$I_{11}*K_{12}+$ $I_{12}*K_{11}$	$I_{12}*K_{12}$	
$I_{11}*K_{21}$	$I_{11}*K_{22}+$ $I_{12}*K_{21}$	$I_{12}*K_{22}$	

step 3:

Output

Image

I_{11}	I_{12}	I_{13}
I_{21}	I_{22}	I_{23}
I_{31}	I_{32}	I_{33}

$I_{11}*K_{11}$	$I_{11}*K_{12}+$ $I_{12}*K_{11}$	$I_{12}*K_{12}+$ $I_{13}*K_{11}$	$I_{13}*K_{12}$
$I_{11}*K_{21}$	$I_{11}*K_{22}+$ $I_{12}*K_{21}$	$I_{12}*K_{22}+$ $I_{13}*K_{21}$	$I_{13}*K_{22}$

step 4:

Image

I_{11}	I_{12}	I_{13}
I_{21}	I_{22}	I_{23}
I_{31}	I_{32}	I_{33}

Output

$I_{11}*K_{11}$	$I_{11}*K_{12}+$ $I_{12}*K_{11}$	$I_{12}*K_{12}+$ $I_{13}*K_{11}$	$I_{13}*K_{12}$
$I_{11}*K_{21}+$ $I_{21}*K_{11}$	$I_{11}*K_{22}+$ $I_{12}*K_{21}+$ $I_{21}*K_{12}$	$I_{12}*K_{22}+$ $I_{13}*K_{21}$	$I_{13}*K_{22}$
$I_{21}*K_{21}$	$I_{21}*K_{22}$		

•
•
•

step 9:

Image

I_{11}	I_{12}	I_{13}
I_{21}	I_{22}	I_{23}
I_{31}	I_{32}	I_{33}

Output

$I_{11}*K_{11}$	$I_{11}*K_{12}+$ $I_{12}*K_{11}$	$I_{12}*K_{12}+$ $I_{13}*K_{11}$	$I_{13}*K_{12}$
$I_{11}*K_{21}+$ $I_{21}*K_{11}$	$I_{11}*K_{22}+$ $I_{12}*K_{21}+$ $I_{21}*K_{12}+$ $I_{22}*K_{11}$	$I_{12}*K_{22}+$ $I_{13}*K_{21}+$ $I_{22}*K_{12}+$ $I_{23}*K_{11}$	$I_{13}*K_{22}+$ $I_{23}*K_{12}$
$I_{21}*K_{21}+$ $I_{31}*K_{11}$	$I_{21}*K_{22}+$ $I_{22}*K_{21}+$ $I_{31}*K_{12}+$ $I_{32}*K_{11}$	$I_{22}*K_{22}+$ $I_{23}*K_{21}+$ $I_{32}*K_{12}+$ $I_{33}*K_{11}$	$I_{23}*K_{22}+$ $I_{33}*K_{12}$
$I_{31}*K_{21}$	$I_{31}*K_{22}+$ $I_{32}*K_{21}$	$I_{32}*K_{22}+$ $I_{33}*K_{21}$	$I_{33}*K_{22}$

Because the sliding window has its property, we can apply pipeline strategies to observe if there are more implicit ways in doing convolution and transposed convolution.

■ **Size:**

The **input image matrix size** will be **8x8, 16x16 or 32x32**. The **input kernel matrix size** will be **5x5**. Also, you need to do the **2x2** max pooling after the convolution.

■ **Mode for operations:**

There are **two modes** for this lab. According to the given mode value, the mode indicates which operation needs to be executed.

mode = 1’b0 → Convolution + **2x2** Max Pooling mode = 1’b1 → Transposed Convolution

■ **Rules of input data:**

In this exercise, you will get **32 matrices continuously (16 image matrices and 16 kernel matrices)** at the beginning of each pattern. After that, you will get **the mode for the following action** and **two input matrix indices continuously**. **The order of the indices determines the image matrix and the kernel matrix**. The indices range from 0 to 15. After **16 sets** of indices (each set containing two indices), you will get the next pattern.

■ **Rules of output result:**

Because of the number limitation of the output pads in the afterwards lab, after finishing the action, you need to output the answer matrix in **raster scan order with serial out format**.

Inputs

Input	Bit Width	Definition and Description
clk	1	Clock.
rst_n	1	Asynchronous active-low reset.
in_valid	1	High when input signals are valid. It will be tied high for **16*size of image matrix (8x8, 16x16 or 32x32) cycles + 16*size of kernel matrix(5x5) cycles** continuously for **matrices**.
in_valid2	1	High when input signals are valid. It will be tied high for 2 cycles continuously for **matrix_idx**. And the first cycle will give **mode.**
matrix_size	2	Size of the input image matrix. It will be given in the first cycle when in_valid is high. **2’b0:** 8x8. **2’b1:** 16x16. **2’b2:** 32x32.
matrix	8	Elements of input matrix. It will be sent in **raster scan order** continuously when **in_valid** is high. The elements are signed integers, which are represented in 2’s complement format.

matrix_idx	4	Input the image matrix index and the kernel matrix index, this signal will be given when **in_valid2** is high. The first cycle of **in_valid2** will give the image matrix index, and the second cycle of **in_valid2** will give the kernel matrix index.
mode	1	The signal will determine the mode for the following action. It will be given in the first cycle when **in_valid2** is high. **1'b0:** Convolution + **2x2** Max Pooling. **1'b1:** Transposed Convolution.

Outputs

Output	Bit Width	Definition and Description
out_valid	1	High only when out_value is valid. It cannot be overlapped with **in_valid** and **in_valid2** signals.
out_value	1	It will output the final answer matrix in **raster scan order and serial out format from the LSB** after finishing matrix computing.

1. The **matrix** signal is delivered in **raster scan order** for **16*size of image matrix (8x8, 16x16 or 32x32) cycles + 16*size of kernel matrix(5x5) cycles continuously when in_valid is tied high**. When all matrices are delivered, it will be tied to an unknown state, and **in_valid** will also be tied low.

2. The **matrix_size** signal will be given in the **first cycle** when the **in_valid** signal is high.

3. Every time in_valid2 is triggered, it is tied high **for two cycles.**

4. The in_valid2 signal will be triggered **for a total 16 times (for two cycles)** after in_valid is tied low in a single pattern. After each time in_valid2 triggers, your design will do convolution or transposed convolution with specific matrices and then out_valid will be tied high for corresponding cycles.

5. In each pattern, the in_valid2 signal will be triggered 1~3 cycles after in_valid is tied low, and the other 15 times in_valid2 will be triggered 1~3 cycles after out_valid is tied low.

6. The next input pattern will be triggered 1~5 cycles after **the sixteenth** out_valid of this pattern falls.

7. The **image matrix index** is given at **the first cycle** of in_valid2 tied high through the signal matrix_idx. The **kernel matrix index** is given at **the second cycle** of in_valid2 tied high through the signal matrix_idx.

8. All input signals are synchronized at the negative edge of the clock.

9. The **out_value** must be delivered for **corresponding cycles** and **out_valid** should be high simultaneously.

10. The out_valid cannot overlap with in_valid and in_valid2 at any time.

11. Each out_value contain 20 bits.

Specifications

1. Top module name: CAD (Design file name: CAD.v)
2. **It has asynchronous reset and active-low architecture. If you use synchronous reset (considering reset after clock starting) in your design, you may fail to reset signals should be reset after the reset signal is asserted.**
3. **The reset signal (rst_n) would be given only once at the beginning of simulation. All output signals should be reset after the reset signal is asserted.**
4. You can adjust your clock period by yourself, but the maximum period is **20 ns**.
5. The data type in the synthesis result **CAN NOT** include any **LATCH**.
6. After synthesis, the area report is valid only when the slack in the end of the timing report is **non-negative** and the result should be **MET**.
7. The next input pattern will come in **1~5** cycles after the sixteenth **out_valid** of this pattern is pulled down.
8. The **out_valid** cannot overlap with **in_valid** and **in_valid2**.
9. The execution latency is limited to 100000 **cycles.** The latency is the clock cycles between the falling edge of the **in_valid2** and the rising edge of the **out_valid**.
10. In this lab, you **must use the memory and generate it yourself. The number of words and the bits per each word is defined by yourself. The total number and kind of memory is unlimited. We will check it at CAD.area in 02_SYN/Report/ folder. The area of Macro/Black Box must not be 0. The example is shown in the following figure.**

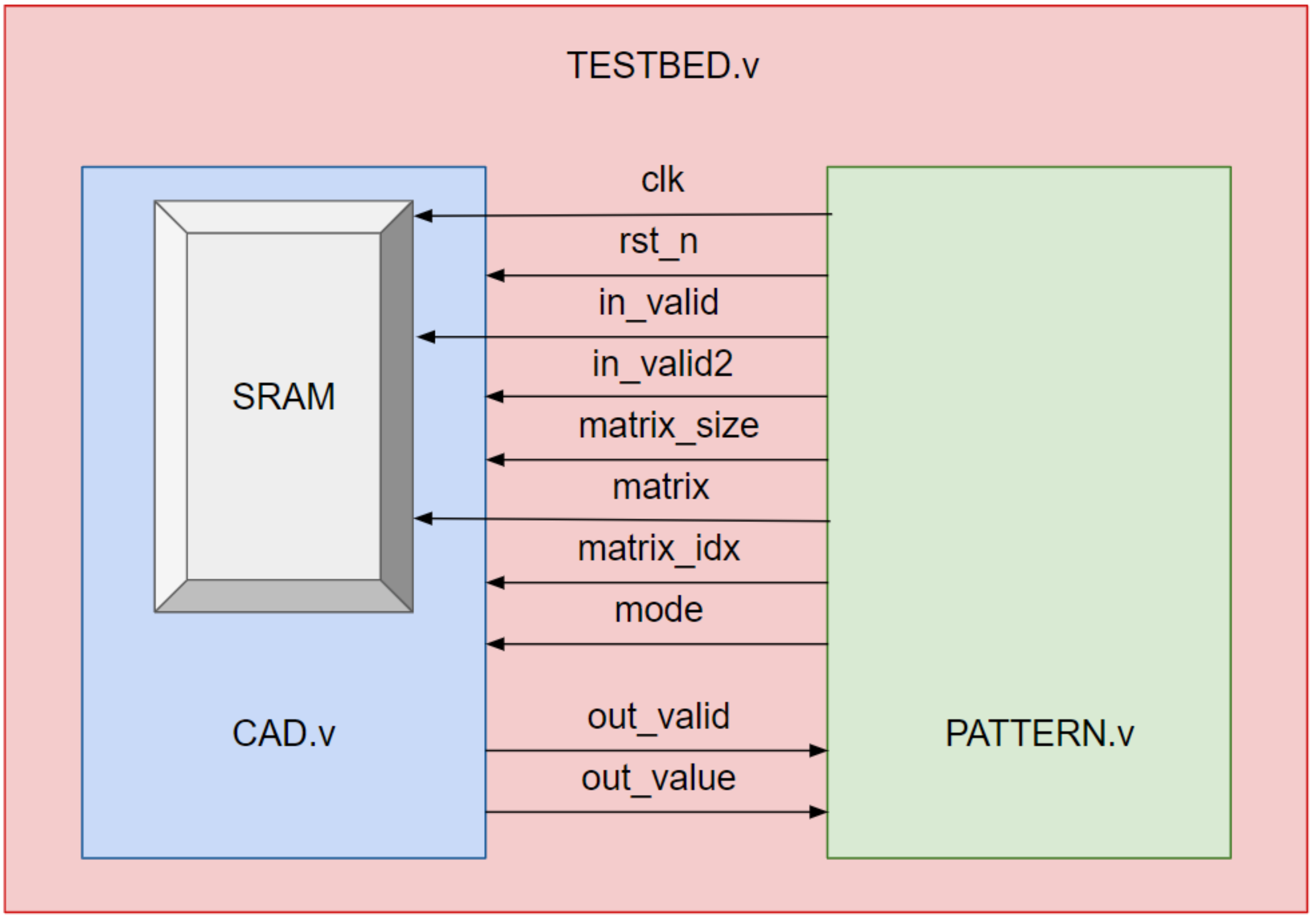
```
Combinational area:             1821995.696653
Buf/Inv area:                    111973.280126
Noncombinational area:           343750.185371
Macro/Black Box area:            214305.703125
Net Interconnect area:      undefined  (No wire load specified)

Total cell area:                2380051.585150
```

Fig 1. The area of your memory

11. The total cell area should not larger than **3,000,000** μm^2**.** Also, the synthesis time should be less than 2 **hours.**
12. **If any port of memory is connected with mismatched width, the memory will not be synthesized and you will get an error message. Even though the design may still pass gate level simulation, this situation will be regarded as a synthesis fail. In this case, the memory area will be 0 in CAD.area. We will check it at syn.log and CAD.area.**
13. **All numbers are signed integers and expressed in 2's complement format. Be sure the operations are done with signed operations.**
14. **Every** output signal should be correct when **out_valid** is high.
15. The input delay is set to **0.5*(clock period)**.
16. The output delay is set to **0.5*(clock period)**, and the output loading is set to **0.05**.
17. The gate level simulation cannot include any timing violations without the *notimingcheck* command.
18. Don't use any wire/reg/submodule/parameter name called *error*, *congratulation*, *latch* or *fail* otherwise you will fail the lab. Note: * means any char in front of or behind the word. e.g: error_note is forbidden.
19. Don' t write Chinese comments or other language comments in the file you turned in.
20. Verilog commands //synopsys dc_script_begin, //synopsys dc_script_end //synopsys translate_off, //synopsys translate_on are only allowed during the usage of including and setting designware IPs, other design compiler optimizations are forbidden.

21. Using the above commands are allowed, however any error messages during synthesis and simulation, regardless of the result will lead to failure in this lab.

Block Diagram



Grading Policy

The performance is determined by the area and latency of your design. The less cost your design has, the higher grade you get.

- Function Validity: 70%
- Performance: 30% **Area^2 * Total Latency**
- **Total latency=cycle time*(total cycle)**
- **Total cycle = $\sum$(cycle of each pattern + 1)**

Note

1. **Please submit your files under 09_SUBMIT before 12:00 at noon in Apr.1:**
   - If uploaded files violate the naming rule, you will get penalty (5 points).
   - In this lab, you can adjust your clock cycle time. Consequently, make sure to key in your clock cycle time after the command like the figure below. It's means that the TA will demo your design under this clock cycle time

```
/Exercise/09_SUBMIT]$ ./00_tar 15.0
```

After that, you should check the following files under **09_SUBMIT/Lab05_iclabXXX/**

RTL design: **CAD_iclabXXX.v** (**XXX** is your account no.)

clock_cycle_iclabXXX.txt

Memory file: **MEMORY_NAME_iclabXXX.v**

MEMORY_NAME_iclabXXX.db
file_list_iclabXXX.f

If you miss any files on the list, you will fail this lab.
Then use the command like the figure below to check if the files are uploaded or not.

```
[Exercise/09_SUBMIT]% ./02_check 1st_demo
```

- ***Example:***
  - Submit your design files:
    CAD_iclab999.v
    10.9_iclab999.txt
  - Given two memories in your design, RA1SH1 and RA1SH2
    A. Submit these memory files:
    RA1SH1_iclab999.v RA1SH1_iclab999.db
    RA1SH2_iclab999.v RA1SH2_iclab999.db
    B. Type following in file_list_iclab999.f and submit it:
    ../04_MEM/RA1SH1.v
    ../04_MEM/RA1SH2.v

2. Template folders and reference commands:

01_RTL/ (RTL simulation) **./01_run_vcs_rtl**
02_SYN/ (Synthesis) **./01_run_dc_shell**
(Check **latch** by searching the keyword "**Latch**" in 02_SYN/**syn.log**)
(Check the design's timing in /Report/CAD.timing)
(Check the design's area in /Report/CAD.area)
03_GATE/ (Gate-level simulation) **./01_run_vcs_gate**
04_MEM/ (Memory location)
(You should generate your own memory and put the required files here)
09_SUBMIT/ (submit your files) **./00_tar ./01_submit ./02_check**

□ You can key in **./09_clean_up** to clear all log files and dump files in each folder

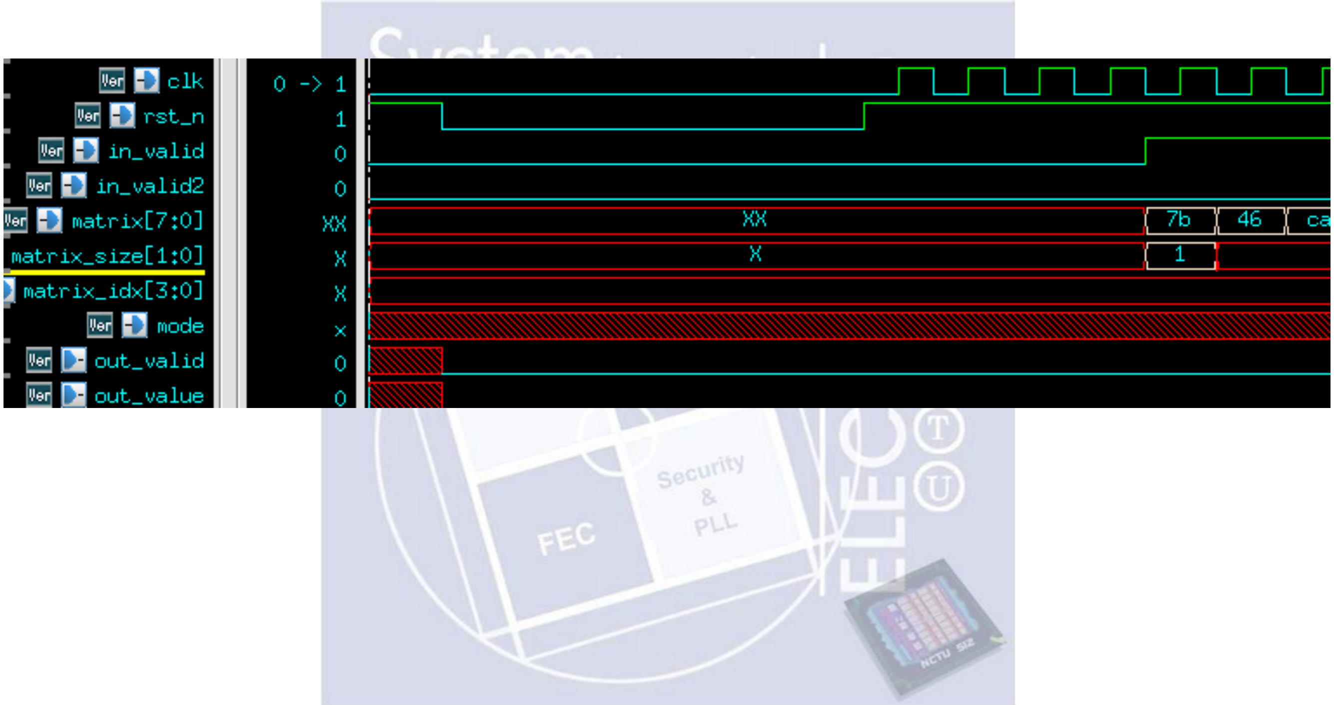
※You should make sure the **three clock period values identical** in 00_TESTBED/PATTERN.v and 02_SYN/syn.tcl

```
`ifdef RTL
    `timescale 1ns/10ps
    `include "CAD.v"
    `define CYCLE_TIME 20.0
`endif
`ifdef GATE
    `timescale 1ns/10ps
    `include "CAD_SYN.v"
    `define CYCLE_TIME 20.0
`endif
```

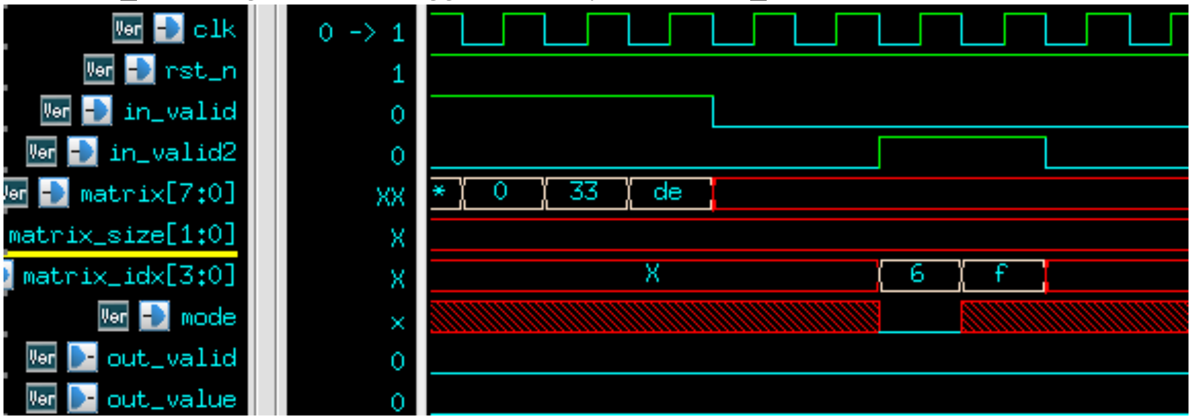
```
set DESIGN "CAD"
set CYCLE 20.0
```

Example Waveform

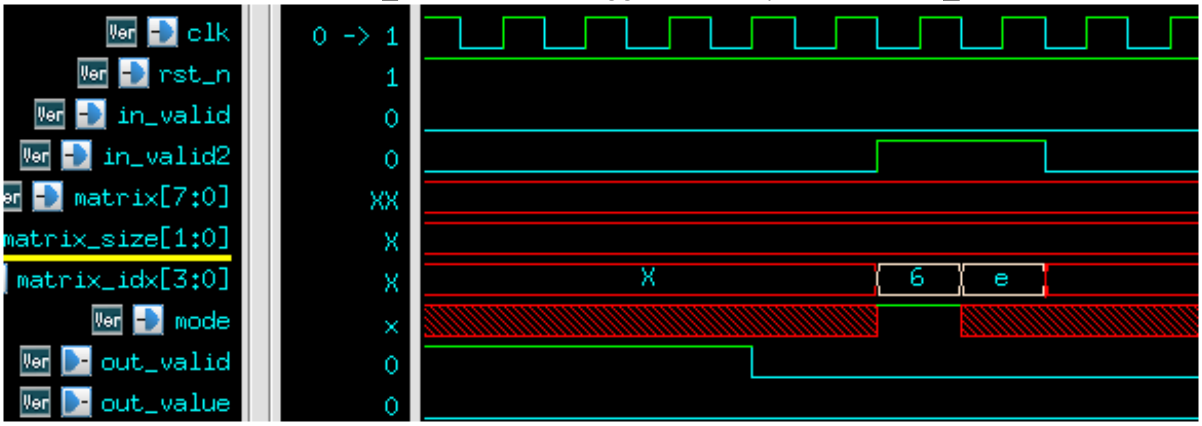
1. Asynchronous reset and active-low reset all output



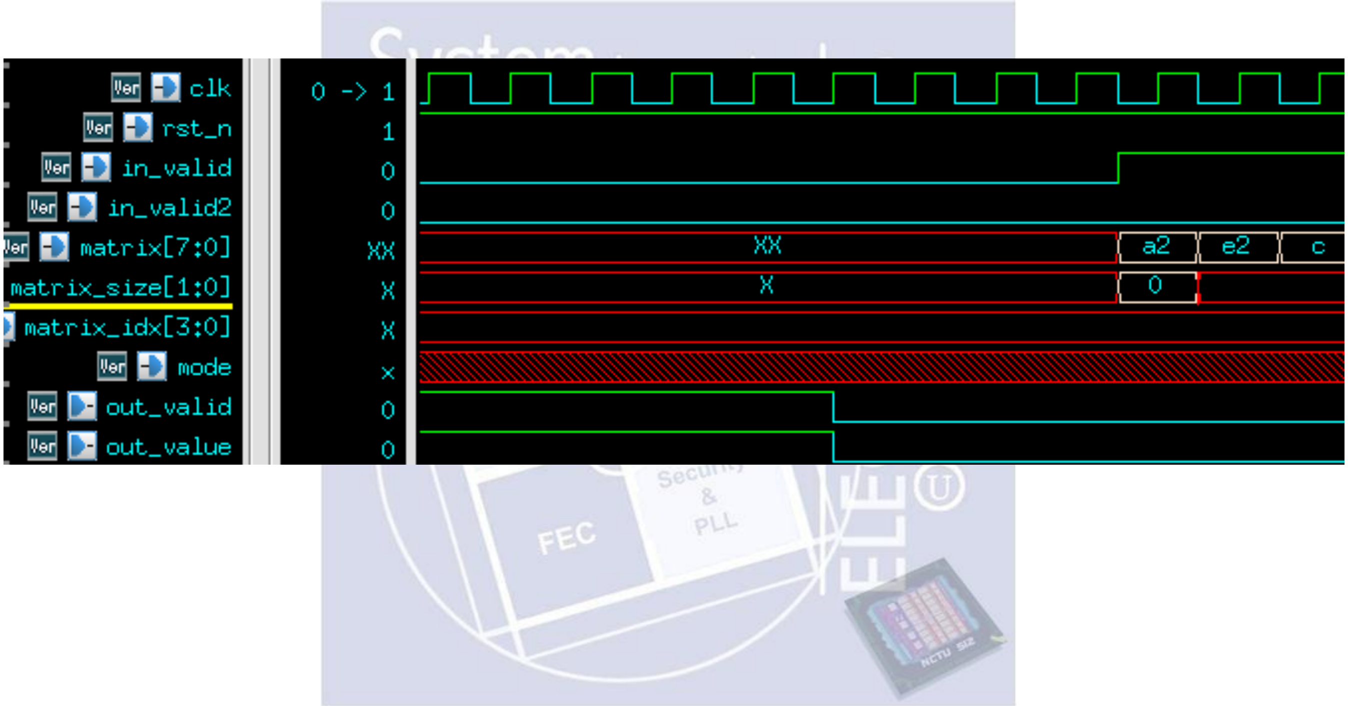
2. The in_valid2 signal will be triggered 1~3 cycles after in_valid is tied low



3. The other fifteen times in_valid2 will be triggered 1~3 cycles after out_valid is tied low.



4. The next input pattern will be triggered 1~5 cycles after **the sixteenth** out_valid of this pattern falls.


clk
rst_n
in_valid
in_valid2
matrix[7:0]
matrix_size[1:0]
matrix_idx[3:0]
mode
out_valid
out_value
0 -> 1
1
0
0
XX
X
X
x
0
0
XX
a2
e2
c
X
0
FEC