NYCU-EE IC LAB – Spring 2023

Lab08 Exercise

Design: Siamese Neural Network

Data Preparation

1. Extract files from TA's directory:
   % tar xvf ~iclabta01/Lab08.tar
2. The extracted LAB directory contains:
   a. **EXERCISE/**
   b. **EXERCISE_wocg/**
   c. **PRACTICE/**
   d. **JG/**

Design Description

The *Siamese Neural Network* is a type of neural network architecture that is designed to recognize the similarity between two different inputs. It was first introduced in the 1990s for the purpose of signature verification. The key characteristic of a *Siamese Neural Network* is that consists of two identical sub-networks that share the same architecture and weights. Each sub-network takes in one input, and the outputs from both sub-network are compared to determine their similarity. This comparison is usually done using a distance metric such as Euclidean distance. If the distance is close, it means that the two inputs are similar.

In this lab, you are asked to design a *Siamese Neural Network* accelerator. You have to take two Convolution Neural Networks (CNN) with identical structures and weights as sub-networks and concatenate them together to form a *Siamese Neural Network*. The CNN sub-network includes operations such as **convolution, quantization, max-pooling**, and **fully connected**. The image encodings computed by the two sub-networks were then compared using the **L1 distance** and the **activation function** to calculate the similarity score.

■ **Siamese Neural Network**

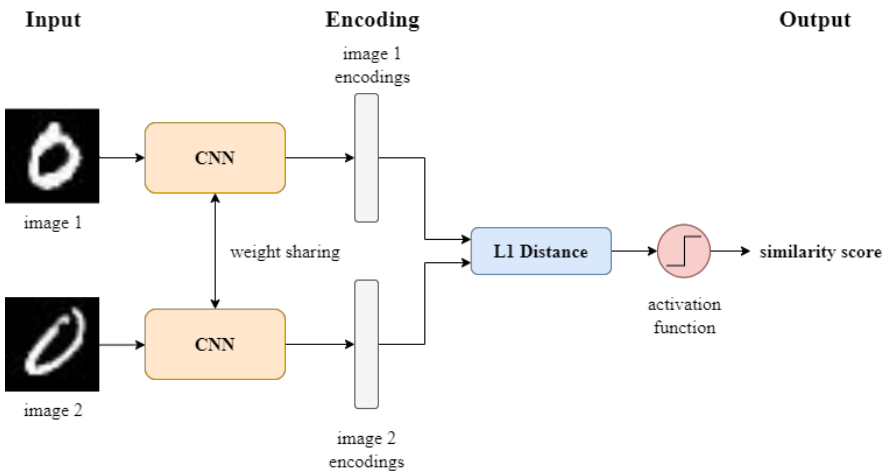


Fig1. Siamese Neural Network architecture

Description of input signals

When **in_valid** is high, the 8-bit unsigned **img** signal will receive **2*6*6 = 72 cycles** continuously to represent 2 input 6x6 images. The pixel values of image 1 will be received during the first 36 cycles (0 ~ 35 cycle), whereas the pixel values of image 2 will be received during the subsequent 36 cycles (36 ~ 71 cycle). The 8-bit unsigned **ker** signal will receive **9 cycles** continuously to represent the 3x3 kernel. The 8-bit unsigned **weight** signal will also receive **4 cycles** continuously to represent the 2x2 matrix for the weight of the fully connected layer. When the 2 images have been provided, i.e., after 72 cycles, **in_valid** will be pulled low. Note that the input signals **img**, **ker**, and **weight** are all sent in raster scan order.

■ **CNN sub-network**

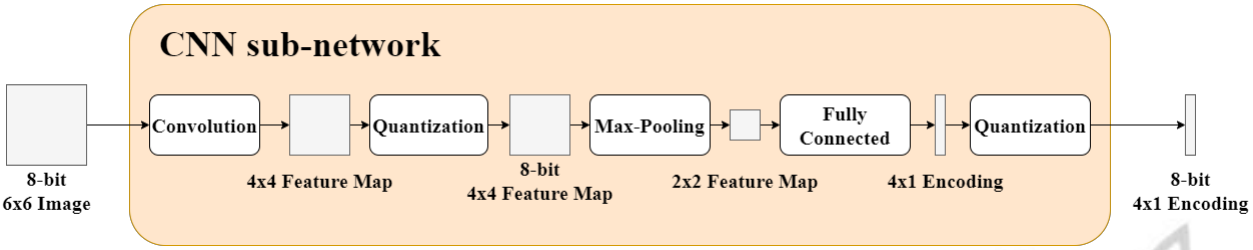


Fig2. CNN sub-network

Description of CNN sub-networks

The input of the CNN sub-network is an 8-bit 6x6 image. Image 1 will be fed into the upper sub-network, while image 2 will be fed into the lower sub-network. First, computing the convolution of the image with the kernel will result in a 4x4 feature map. Next, the 4x4 feature map will be quantized to an 8-bit 4x4 feature map. Then, the 8-bit 4x4 feature map will do the max-pooling operation, resulting in a 2x2 feature map. The next step is to input the 2x2 feature map to a fully connected layer, resulting in a 4x1 encoding vector. In this step, the 2x2 feature map will be multiplied by a 2x2 weight matrix, where **weight** represents the matrix elements and will be continuously provided for 4 cycles during the input stage. The resulting matrix will be flattened into a 4x1 encoding vector. Then quantizing the 4x1 encoding vector will result in an 8-bit encoding vector.

■ **Connection of two sub-networks**

After computing the encodings of the two sub-networks, the **L1 distance** between the encodings is then calculated. The result of the L1 distance is then passed through an **activation function** to obtain the output signal **10-bit data_out** (similarity score).

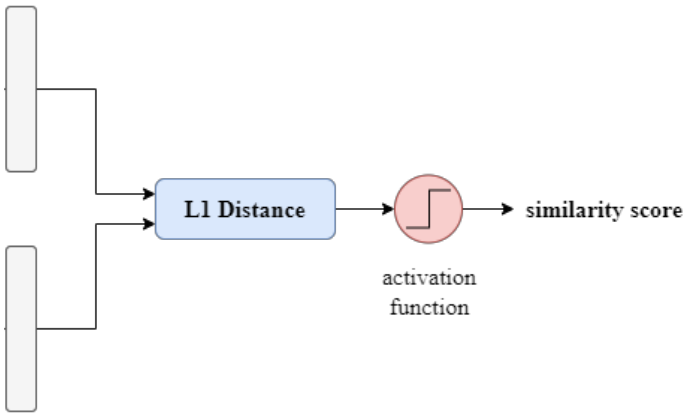


Fig3. Connection of two sub-network

■ Convolution

Formula of convolution

$$FeatureMap[\mathrm{m,n}] = \sum_{j}\sum_{i} Image[m,n] \cdot Kernel[m-i, n-j]$$

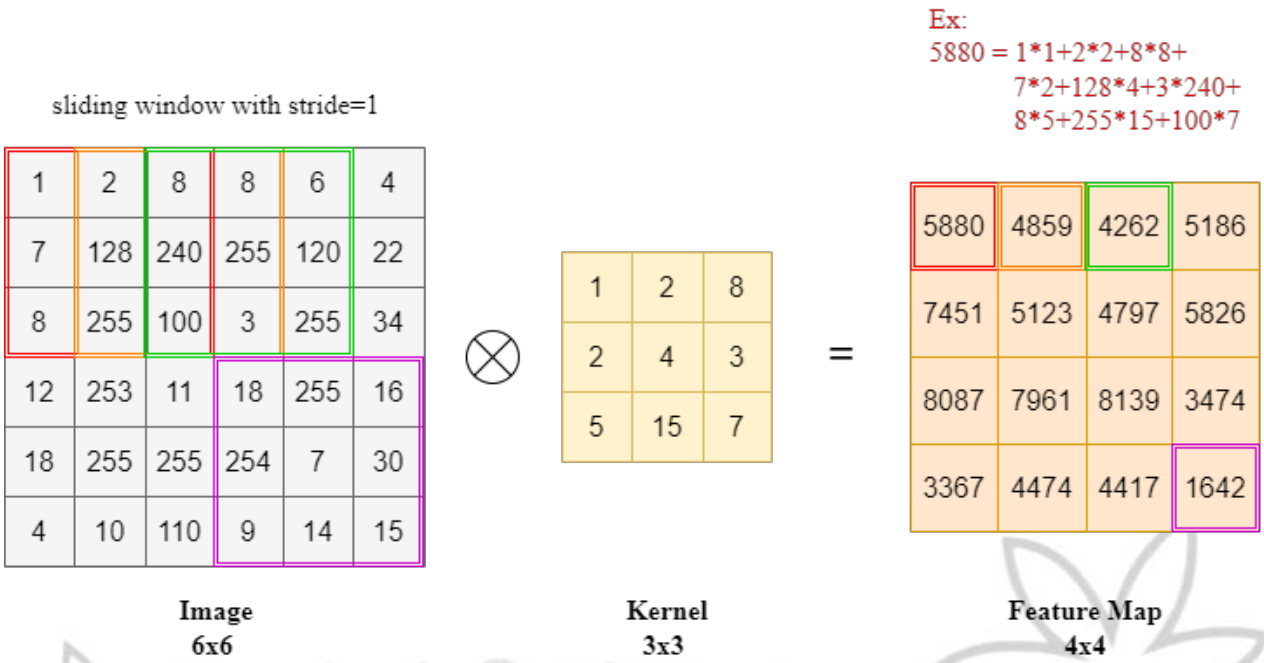


Fig4. Example of convolution operation

■ Quantization

Formula of quantization

$$scale = \frac{V_{max} - V_{min}}{q_{max} - q_{min}}$$

$$q = floor(\frac{V}{scale})$$

The *scale* represents the scaling parameter of quantization. The V_{max} and V_{min} represent the maximum and minimum value of elements (feature map or encoding vector). The q_{max} and q_{min} represent the maximum and minimum value in the quantized range. In this Lab, $q_{max} = 255$ and $q_{min} = 0$. The q represents the quantization value.

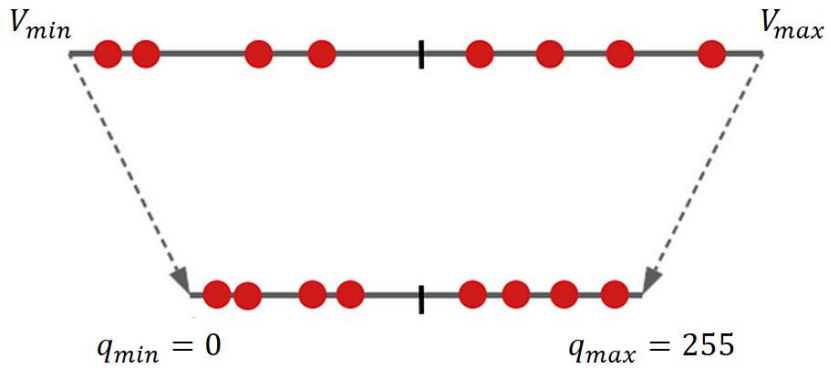


Fig5. Quantization

Exampe1 - Quantization of 4x4 feature map

The maximum value of **4x4 feature map element is 585225**.

So, $V_{max} = 585225,\ V_{min} = 0.$

Assume $V = 5880,$

$$scale = \frac{V_{max} - V_{min}}{q_{max} - q_{min}} = \frac{585225 - 0}{255 - 0} = 2295$$ (16-bit) (12 bit)

$$q = floor\left(\frac{V}{scale}\right) = floor\left(\frac{5880}{2295}\right) = 2$$

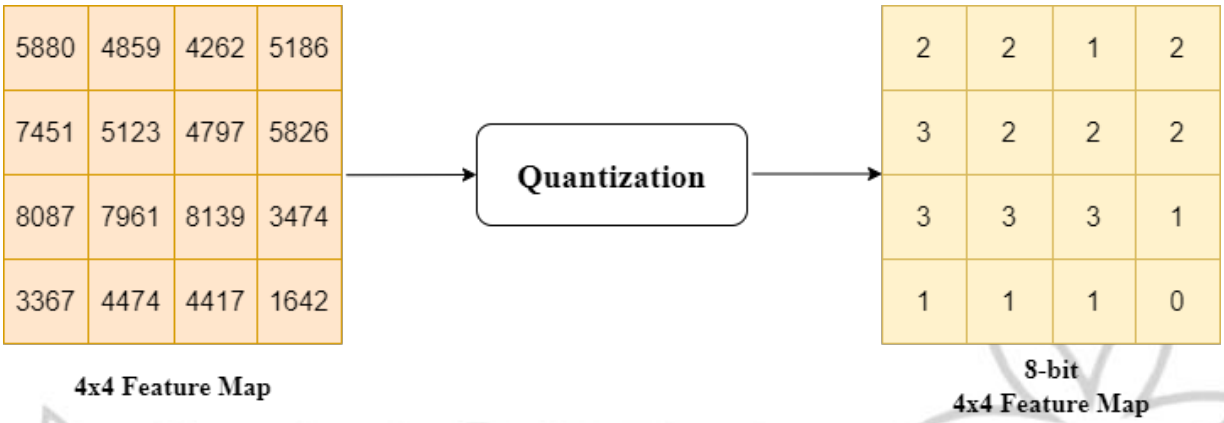


Fig6. Quantization of 4x4 feature map

Exampe2 - Quantization of 4x1 encoding vector

The maximum value of **4x1 encoding vector element is 130050.**

So, $V_{max} = 130050,\ V_{min} = 0.$

$$scale = \frac{V_{max} - V_{min}}{q_{max} - q_{min}} = \frac{130050 - 0}{255 - 0} = 510$$

$$q = floor\left(\frac{V}{510}\right)$$

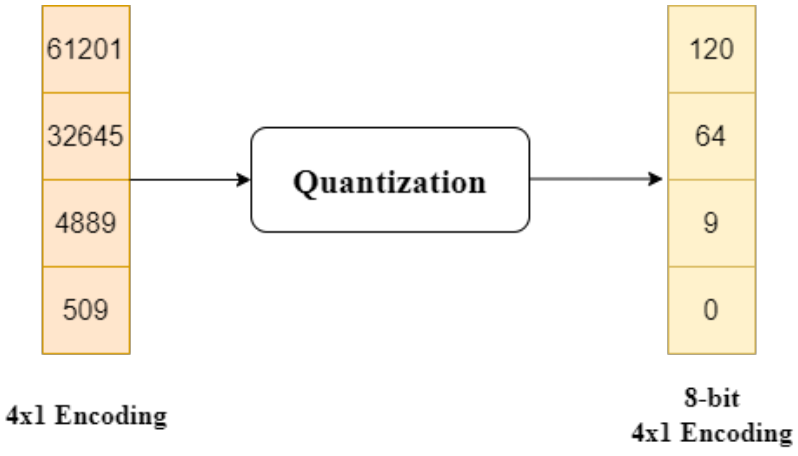


Fig7. Quantization of 4x1 encoding vector

■ **Max-Pooling**

The max-pooling operation works by sliding a 2x2 window, over the input feature map and taking the maximum value in each window as output.

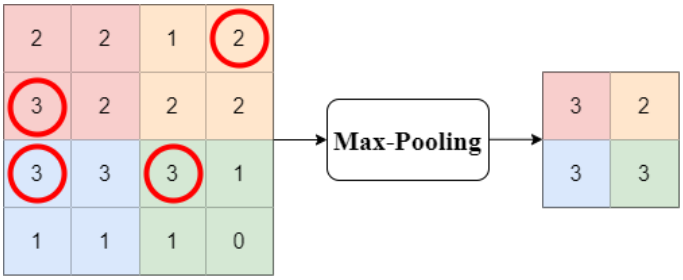


Fig8. Example of max-pooling operation

■ **Fully Connected**

A fully connected layer can be represented as a matrix multiplication operation between the input matrix and weight matrix. Flattening the output matrix in obtaining an encoding vector.

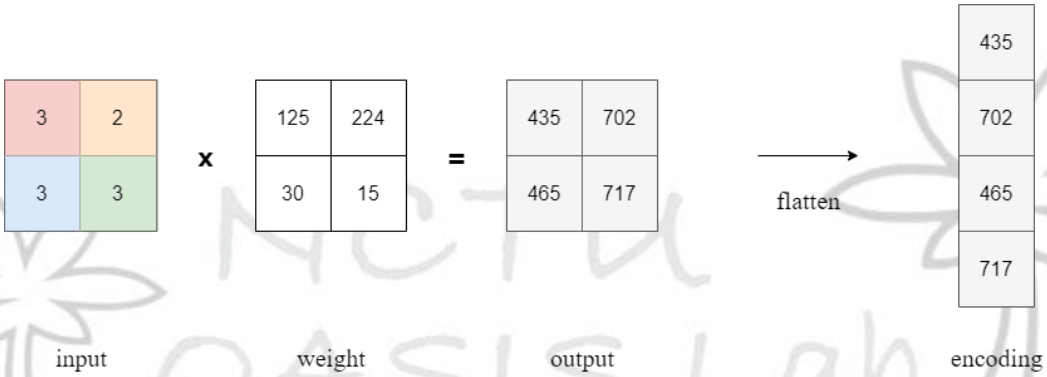


Fig9. Example of fully connected layer

■ **L1 distance**

L1 distance also known as Manhattan distance. It is defined as the sum of the absolute differences between the corresponding coordinates of the two points.

Formula of L1 distance

$$L1\ \ distance = \sum_{1}^{n} |p_i - q_i|$$

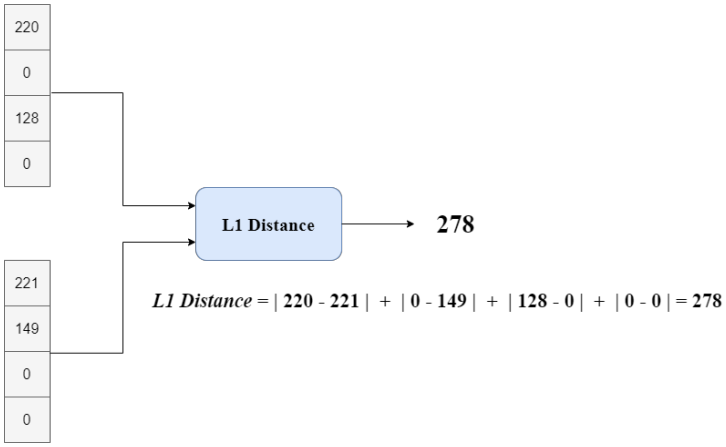


Fig10. Example of L1 distance

■ **Activation Function**

An activation applies a non-linear transformation to the result of the L1 distance.

Formula of activation function

$$A(x) = \begin{cases} 0, & x < 16 \\ x, & x \geq 16 \end{cases}$$

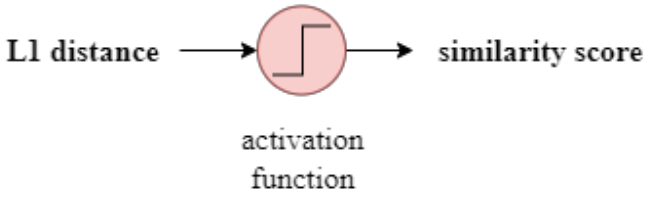


Fig11. Example of activation function

Note: All of the computations mentioned above must ensure that their process and results do not overflow.

Lab Hint

In this lab, you need to design two version of module.

■ **Stage 1** (w/o clock gating)

At EXERCISE_wocg/**SNN_wocg.v**, which means SNN without clock gating, you should design a SNN module as below figure.

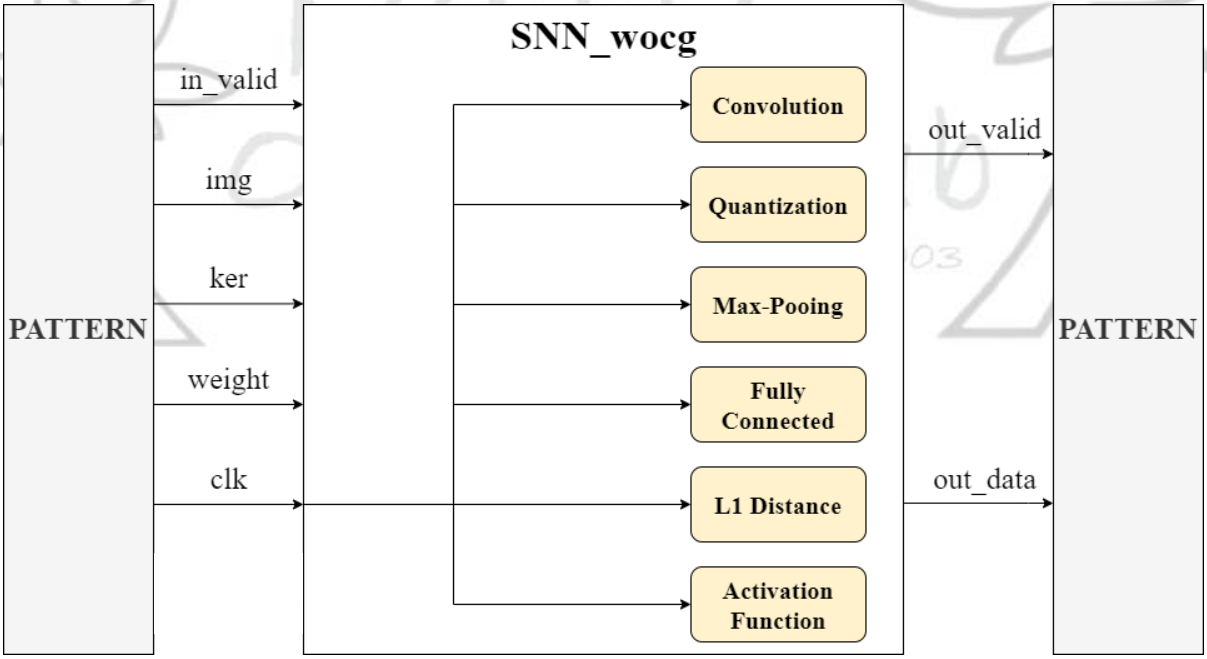


Fig12. Siamese Neural Network without clock gating

- **INPUT:** Receive **img, ker** and **weight** from PATTERN
  - → The [7:0] **img** will be given 2*6*6 = 72 cycles.
  - → The [7:0] **ker** will be given 3*3 = 9 cycles.
  - → The [7:0] **weight** will be given 2*2 = 4 cycles.
- **SNN_wocg:** Design module with some submodules performs series processing described as above.
- **OUTPUT:** Output resulting [9:0] **out_data**.

■ **Stage 2**

At EXERCISE_wocg/**SNN.v**, you should design a **SNN module with clock gating cell** as below figure. Main different part is clock gating cell and cg_en input signal.

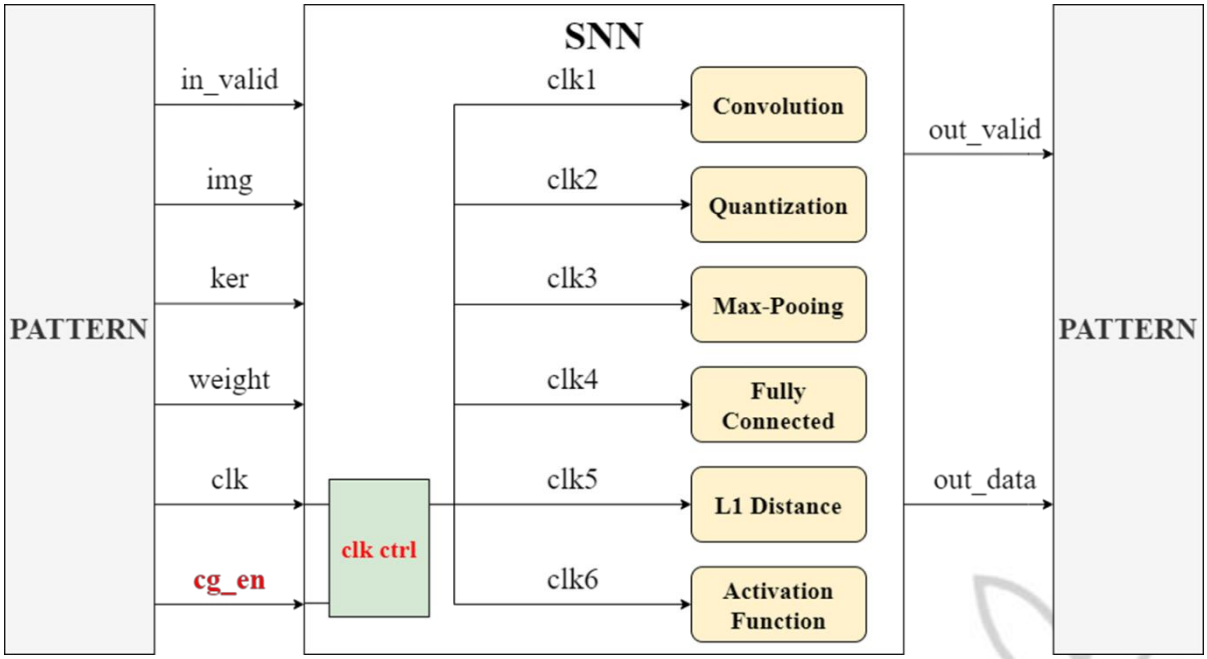


Fig13. Siamese Neural Network with clock gating

- **INPUT:** Receive **img, ker, weight and cg_en** from PATTERN
  - → The [7:0] **img** will be given 2*6*6 = 72 cycles.
  - → The [7:0] **ker** will be given 3*3 = 9 cycles.
  - → The [7:0] **weight** will be given 2*2 = 4 cycles.
- **SNN:** Design module. You should add clock gating cell in the design module.
  If **cg_en** is high, 6 processing blocks can perform clock gating; otherwise, if **cg_en** is low, 6 processing blocks follow **clk**.
- **OUTPUT:** Output resulting [9:0] **out_data**.

Inputs and Outputs

I/O	Signal Name	Bit Width	Description
Input	**clk**	1	Clock
Input	**rst_n**	1	Asynchronous active low reset
Input	**cg_en**	1	If **cg_en** is high, the series processing blocks should execute clock gating. Otherwise, if **cg_en** is low, the processing blocks follow **clk**.
Input	**in_valid**	1	High when input signals are valid.
Input	**img**	8	**img** is valid when **in_valid** is high. The **img** will be given in 2*6*6 = 72 cycles continuously in raster scan order. image 1: 0 ~ 35 cycle image 2: 36 ~ 71 cycle The value are unsigned integers. (0~255)
Input	**ker**	8	**ker** is valid when **in_valid** is high. The **ker** will be given in 3*3 = 9 cycles continuously in raster scan order. The value are unsigned integers. (0~255)
Input	**weight**	8	**weight** is valid when **in_valid** is high. The weight will be given in 2*2 = 4 cycles continuously in raster scan order. The value are unsigned integers. (0~255)
Output	**out_valid**	1	Should set to high when your **out_data** is ready.
Output	**out_data**	10	Output the resulting data. **out_data** should be given in 1 cycle. The value are unsigned integers. (0~255)

Specifications

1. Top module name: SNN (File name: SNN.v)
2. Input pins: clk, rst_n, cg_en, in_valid, [7:0] img, [7:0] ker, [7:0] weight
   Output pins: **out_valid, [9:0] out_data**
3. Use **asynchronous** reset active low architecture.
4. All your output register should be set zero after reset.
5. Changing clock period is prohibited **(fixed at 15ns).**
6. The instance name of the clock gating cell (GATED_OR) should be GATED_XXX (e.g., GATED_cnt).

```
GATED_OR GATED_out (
    .CLOCK(clk), .SLEEP_CTRL(G_sleep_out),
    .RST_N(rst_n), .CLOCK_GATED(G_clock_out)
);
```

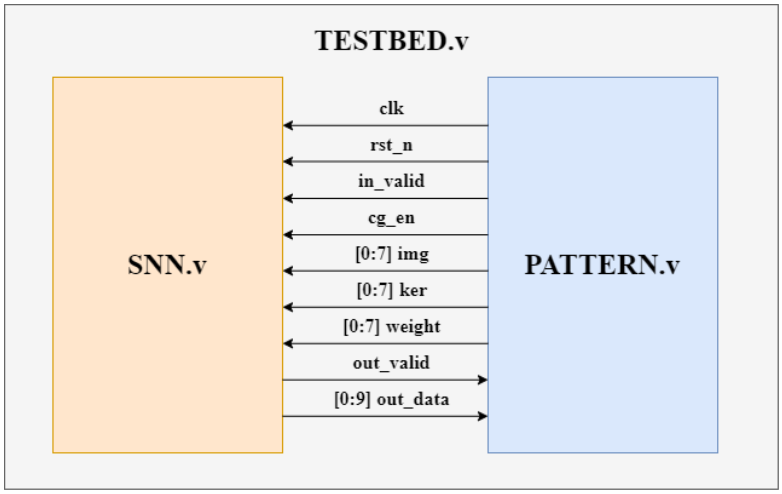
7. After synthesis, check the "SNN.area" and "SNN.timing" in the folder "Report". The area report is valid only when the slack in the end of "SNN.timming" is **non-negative** and the result should be **MET**.
8. The next input will come in **2~5** cycles after your **out_valid** is pulled down.
9. The synthesis result **cannot** contain any **LATCH except for clock gating latch.**
10. The syntheses result **cannot** contain any error.
11. The output loading is set to 0.05.
12. Input delay and output delay are 0.5*Clock Period.
13. You can't have timing violation in gate-level simulation.
14. You **can't use memory** in this lab.
15. You **can't use DesignWare IP** in this lab.
16. The latency is limited in **1000 cycles**.
    latency = 1 + execution latency
17. The **out_valid** cannot overlap with **in_valid**.
18. The **out_valid** should be set to high for 1 cycle when **out_data** is valid.
19. The **out_valid** should set to low when your **out_data** is invalid.
20. Your design should have **at least 25% power reduction** from **cg_en-off** to **cg_en-on**, otherwise it will be regarded as failed. $\frac{P_{cg\_enoff} - P_{cg\_enon}}{P_{cg\_enoff}} \geq 25\%$ (use Prime Time 來算)
21. **Power report position: 04_PTPX/Report/SNN_POWER** or **SNN_CG_POWER**
    Report Total Power Example:

SNN_POWER (w/o clock gating)			SNN_CG_POWER (with clock gating)		
Net Switching Power	= 1.160e-03	(9.51%)	Net Switching Power	= 6.579e-04	(14.66%)
Cell Internal Power	= 0.0110	(90.30%)	Cell Internal Power	= 3.806e-03	(84.82%)
Cell Leakage Power	= 2.317e-05	(0.19%)	Cell Leakage Power	= 2.317e-05	(0.52%)
Intrinsic Leakage	= 2.317e-05		Intrinsic Leakage	= 2.317e-05	
Gate Leakage	= 0.0000		Gate Leakage	= 0.0000	
Total Power	= 0.0122	(100.00%)	Total Power	= 4.487e-03	(100.00%)
X Transition Power	= 1.193e-06		X Transition Power	= 1.193e-06	
Glitching Power	= 0.0000		Glitching Power	= 0.0000	
Peak Power	= 0.1198		Peak Power	= 0.1098	
Peak Time	= 246127		Peak Time	= 97	

$$\frac{P_{cg\_enoff} - P_{cg\_enon}}{P_{cg\_enoff}} = \frac{0.0122 - 4.487 \times 10^{-3}}{0.0122} = 63.2\% \geq 25\%$$

22. The gate level simulation cannot include any timing violation without the *notimingcheck* command.
23. 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.
24. Don't write Chinese comments or other language comment in the file you turned in.
25. **Any error message during synthesis and simulation, regardless of the result will lead to failure in this lab.**
26. **Synthesis time should not exceed 2 hours.**

Block Diagram



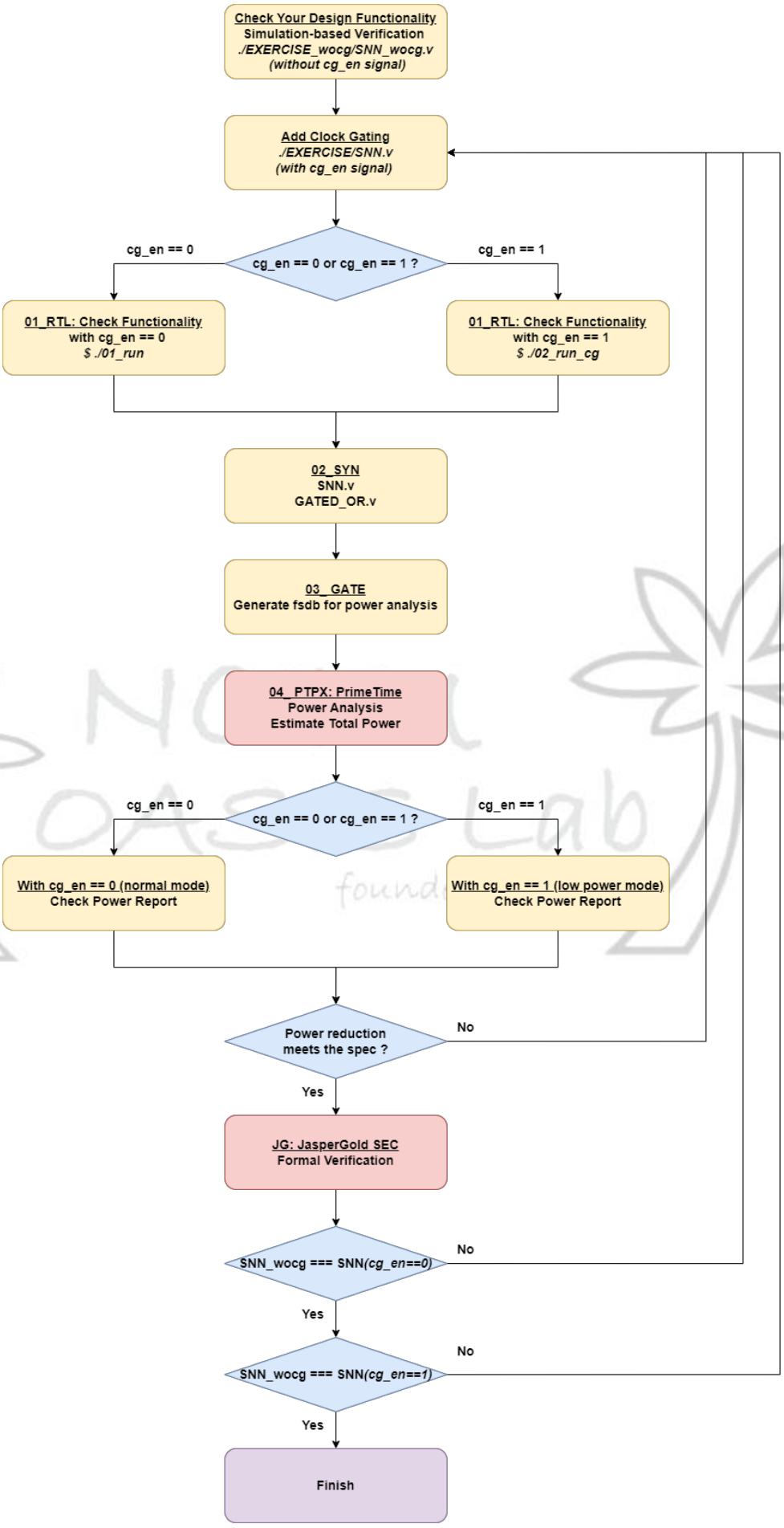
Grading Policy

- Functionality (70%)
  - SNN_wocg and SNN: 60%
  - JasperGold SEC check (10%): Run1 (5%), Run2 (5%)
    (**JasperGold No 2nd demo chance**)
- Performance (30%)
  - (**Total Latency** * **Total Power** (*gated with CG*)) * **Area**
  - Total Latency = 1 + Execution Latency

Note

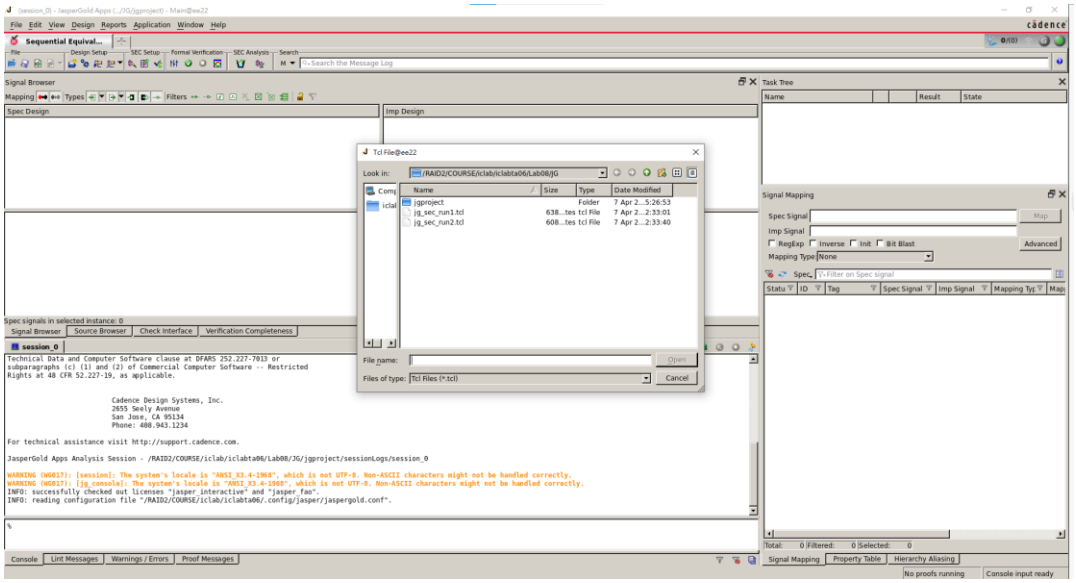
1. Please submit your design **SNN_wocg.v** and **SNN.v** in
   Lab08/EXERCISE/09_SUBMIT
   a. 1st_demo deadline: 2023/04/24 (Mon.) 12:00:00
   b. 2nd_demo deadline: 2023/04/26 (Wed.) 12:00:00
2. Template folders and reference commands:
   01_RTL/ (RTL simulation)
   ./01_run **./02_run_cg** *(only in EXERCISE folder)*
   02_SYN/ (Synthesis)
   ./01_run_CG_dc **./clk_run_dc** *(only in EXERCISE folder)*
   (Check if there is any **latch** or **error** in your design in **syn.log**)
   (Check the timing of the design in /Report/**SNN.timing**)
   03_GATE_SIM/ (Gate-level simulation)
   ./01_run **./02_run_cg** *(only in EXERCISE folder)*
   04_PTPX/ (PrimeTime power analysis)
   ./01_run_ptpx **./02_run_cg_ptpx**
   (Get the power of your design)
   09_SUBMIT/
   ./01_submit **./02_check**
   You can key in **./09_clean_up** to clear all log files and dump files in each folder.

■ Flow

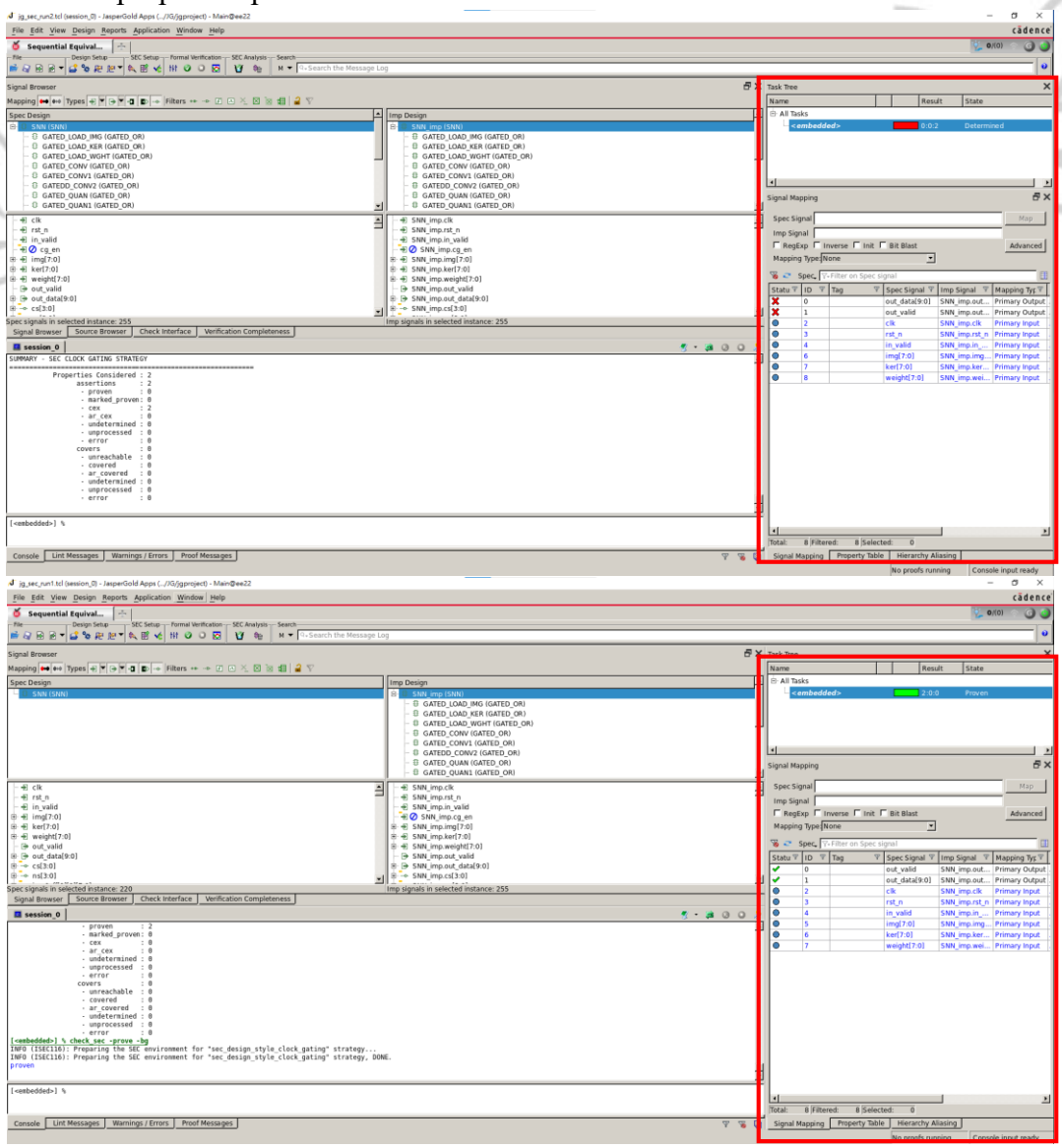

Check Your Design Functionality
Simulation-based Verification
./EXERCISE_wocg/SNN_wocg.v
(without cg_en signal)
Add Clock Gating
./EXERCISE/SNN.v
(with cg_en signal)
cg_en == 0 or cg_en == 1 ?
cg_en == 0
cg_en == 1
01_RTL: Check Functionality
with cg_en == 0
$ ./01_run
01_RTL: Check Functionality
with cg_en == 1
$ ./02_run_cg
02_SYN
SNN.v
GATED_OR.v
03_GATE
Generate fsdb for power analysis
04_PTPX: PrimeTime
Power Analysis
Estimate Total Power
cg_en == 0 or cg_en == 1 ?
cg_en == 0
cg_en == 1
With cg_en == 0 (normal mode)
Check Power Report
With cg_en == 1 (low power mode)
Check Power Report
Power reduction
meets the spec ?
No
Yes
JG: JasperGold SEC
Formal Verification
SNN_wocg === SNN(cg_en==0)
No
Yes
SNN_wocg === SNN(cg_en==1)
No
Yes
Finish

3. JasperGold SEC execution steps: In folder JG/

(1) **./01_run**, **./02_run** or **jg –sec &** and click the **File/Tcl_script/source**.



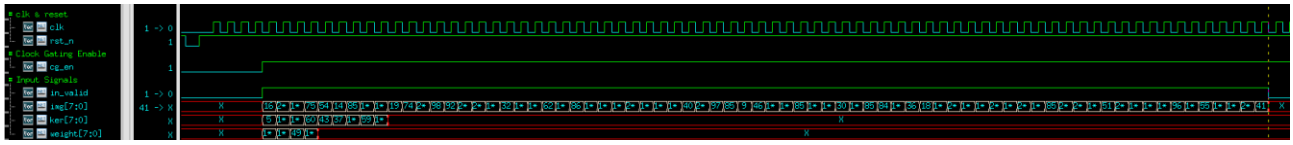
(2) Check all properties pass.



You should pass all tasks in both **./01_run** & **./02_run**.

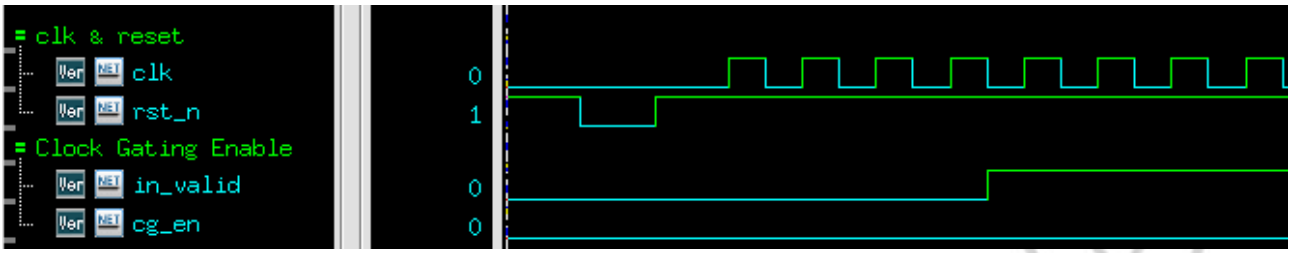
Waveform Example

1. Input Signals



2. Output Signals

3. For ./EXERCISE/PATTERN.v , you need to set cg_en = 0:



4. For ./EXERCISE/PATTERN_CG.v , you need to set cg_en = 1: