# Computer-Aided VLSI System Design
# Homework 3: Single-Channel Convolution with Barcode Decoding Engine

TA: 吳璨霖 r13943017@ntu.edu.tw **Due Tuesday, Nov. 4, 13:59**

TA: 陳柏任 d13943013@ntu.edu.tw

## Data Preparation

1. Decompress 1141_hw3_vk.tar with following command

```
tar -xvf 1141_hw3_vk.tar
```

| Folder | File | Description |
|---|---|---|
| 00_TESTBED | testbench.v | Testbench |
| 00_TESTBED/ PATTERNS | Image*.png | Input image |
| | Image*.dat | Input image data (**unsigned**) |
| | weight*.dat | Weight data (**signed**) |
| | golden*.dat | Golden data of output |
| 01_RTL | core.v | Your design |
| | rtl_01.f | File list for RTL simulation |
| | 01_run | VCS command |
| | 02_lint | SpyGlass linting command |
| | lint.tcl | Script for linting |
| | 99_clean_up | Command to clean temporary data |
| 02_SYN | syn.tcl | Script for synthesis |
| | core_dc.sdc | Constraint file for synthesis |
| | 02_run.dc | Command for DC |
| | flist.sv | File list for synthesis |
| 03_GATE | rtl_03.f | File list for gate-level simulation |
| | 03_run | VCS command for gate-level simulation |
| | 99_clean_up | Command to clean temporary data |
| sram_****x8 | sram_****x8.v | SRAM design file |
| | sram_****x8_slow_syn.db | Synthesis model |
| | sram_****x8_slow_syn.lib | Timing and power model |
| | sram_****x8.pdf | Datasheet for SRAM |
| top | report.txt | Design report form |

# Introduction

***"...Prove yourself more Hardware Designer than RTL Complier..."***

— *Adapted from "**Hollow Knight: SilkSong**" (Team Cherry, 2025)*

Your client used to be a convenience store clerk. He asked you to design a convolution engine, but the operation parameters are hidden within barcodes — a format he is most familiar with — so that others would not easily understand his method.

In this homework, you are going to implement a single-channel convolution with barcode decoding engine. A 64×64 grayscale image (8-bit per pixel) will be loaded as input, in which a **Code 128-C barcode** is embedded within its **least significant bits (LSBs)**. The convolution operation should be performed under different configurations, as specified by the hyperparameters encoded in the barcode. If you are not familiar with convolution operation or barcode, we will have a simple explanation in the next section or you can refer to [1] and [2].

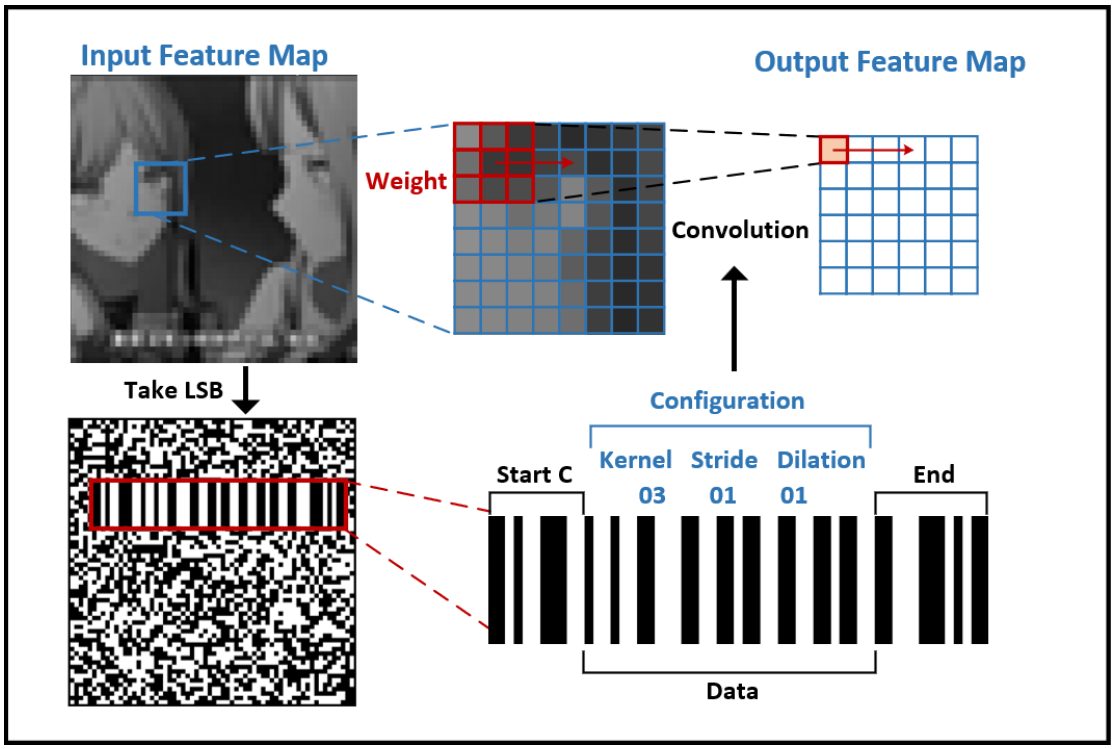



***Fig.1*** Workflow Overview of the homework

## Block Diagram

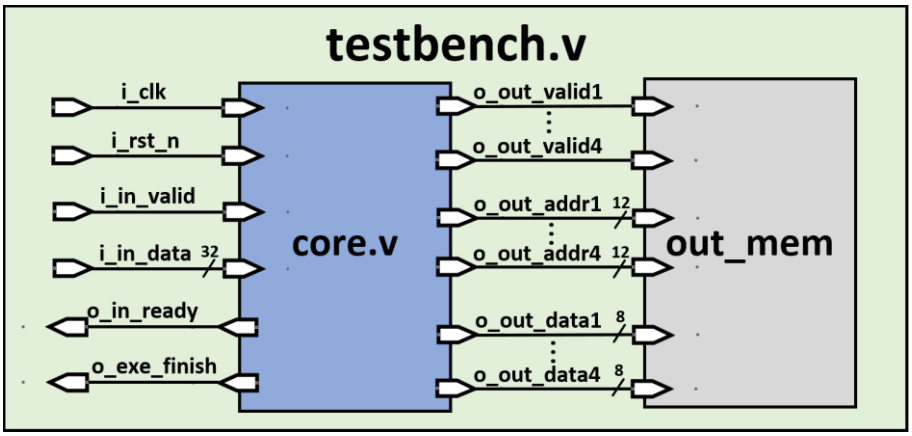



***Fig.2*** Block Diagram

## Specifications

1. Top module name: **core**
2. Input/output description:

| Signal Name | I/O | Width | Simple Description |
|---|---|---|---|
| i_clk | I | 1 | Clock signal in the system. |
| i_rst_n | I | 1 | Active **low** asynchronous reset. |
| i_in_valid | I | 1 | This signal is **high** if input pixel data is valid |
| i_in_data | I | 32 | Input image/weight data |
| o_in_ready | O | 1 | Set to **high** if ready to get **next 4** input data |
| o_out_valid1 | O | 1 | Set to **high** with valid output data |
| o_out_valid2 | O | 1 | Set to **high** with valid output data |
| o_out_valid3 | O | 1 | Set to **high** with valid output data |
| o_out_valid4 | O | 1 | Set to **high** with valid output data |
| o_out_addr1 | O | 12 | Address of output data |
| o_out_addr2 | O | 12 | Address of output data |
| o_out_addr3 | O | 12 | Address of output data |
| o_out_addr4 | O | 12 | Address of output data |
| o_out_data1 | O | 8 | Output data / Kernel size |
| o_out_data2 | O | 8 | Output data / Stride size |
| o_out_data3 | O | 8 | Output data / Dilation size |
| o_out_data4 | O | 8 | Output data |
| o_exe_finish | O | 1 | Set to **high** if the execution is finished |

3. All inputs are synchronized with the **negative** edge clock.
4. All outputs should be synchronized at clock **rising** edge.
5. You should reset all your outputs when i_rst_n is **low**. Active low asynchronous reset is used and only once.
6. i_in_valid and o_out_valid can't be **high** at the same time.
7. In covolution stage, any two outputs **can't** be written to the **same address at the same time**.
8. o_out_valid should be **high** for valid output results.
9. Pull **o_exe_finish** to **high after finish all the operation.**
10. **At least one SRAM** is implemented in your design.
11. Only worst-case library is used for synthesis.
12. The synthesis result of data type should **NOT** include any **Latch**.
13. The slack for setup time should be **non-negative**.
14. **No timing violations or glitches** in the gate-level simulation **after reset**.

## Design Description

1. Input feature map loading:
   - An 64×64 feature map is loaded for 1024 cycles in **raster-scan** order.
   - The size of each pixel is 8 bits (unsigned).
   - If o_in_ready is 0, stop input data until o_in_ready is 1.
   - The input feature map will be loaded only once at the beginning.

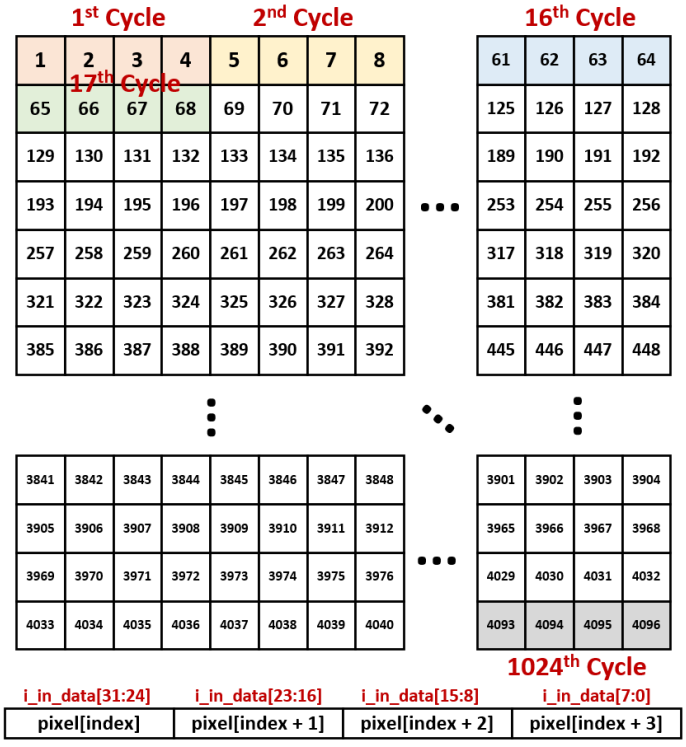



***Fig.3*** Mapping of input image and i_in_data

2. Code 128-C Barcode:

**CODE 128** is a high-density linear barcode symbology capable of encoding all 128 ASCII characters, including control codes such as *[ESC]* and *[CR]*.

- A typical CODE 128 barcode consists of five sections:
  - Quiet zone (Not used in this homework)
  - Start Symbol
  - Encoded data
  - Check digit (Not used in this homework)
  - Stop symbol
- In this homework, A Code 128-C barcode with a height of 10 pixels is embedded within the **least significant bits (LSBs)** of input image.
- The barcode sequentially encodes three hyperparameters in following order:
  - Kernel size (K) : fixed to 03.
  - Stride size (S): limited to {01, 02}.
  - Dilation size (D): limited to {01, 02}.
- The illustration below shows an example of the barcode in this homework.

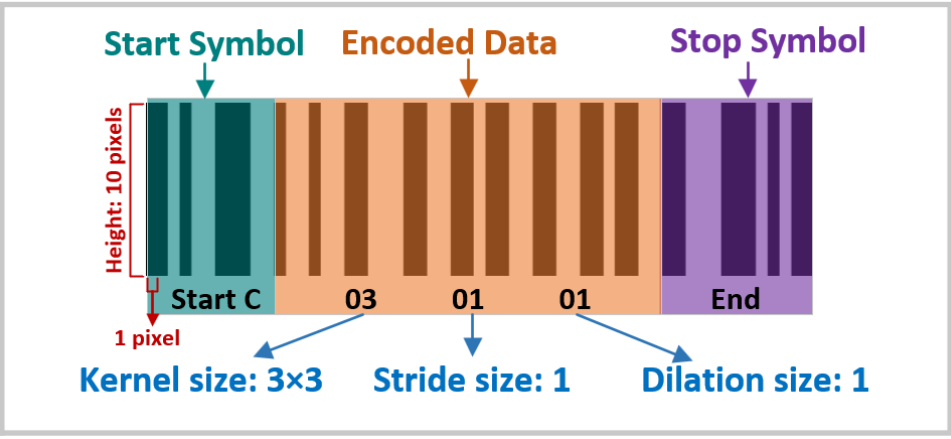



***Fig.4*** An example of barcode.

- Barcode in this homework always start with **Start C** and end with **Stop Symbol**
- **After loading the image**, your design must output the **3** hyperparameters simultaneously to verify the correctness by testbench.
- Use **o_out_data1**, **o_out_data2**, and **o_out_data3** to output the decoded hyperparameters:
  - o_out_data1: *Kernel size*.
  - o_out_data2: *Stride size*.
  - o_out_data3: *Dilation size*.
- If the decoded hyperparameters do not match any of the valid combinations specified above (**K = 3**, **S ∈ {1, 2}**, **D ∈ {1, 2}**), the configuration is considered **invalid**, all three outputs should be set to **0**.

- The barcode can appear **anywhere in the image** but will always **stay fully inside the image boundaries**, and your design must correctly handle all possible barcode positions.
- For convenience, a partial Code 128-C table is provided below for reference:

| Value | 128C | Pattern | Widths |
|---|---|---|---|
| 1 | 01 | 11001101100 | 222122 |
| 2 | 02 | 11001100110 | 222221 |
| 3 | 03 | 10010011000 | 121223 |
| 105 | Start Code C | 11010011100 | 211232 |
| — | Stop symbol | 1100011101011 | 2331112 |

3. Convolution:

Convolution is widely used in image processing and computer vision applications. We will introduce the 2D convolution with some simple examples.

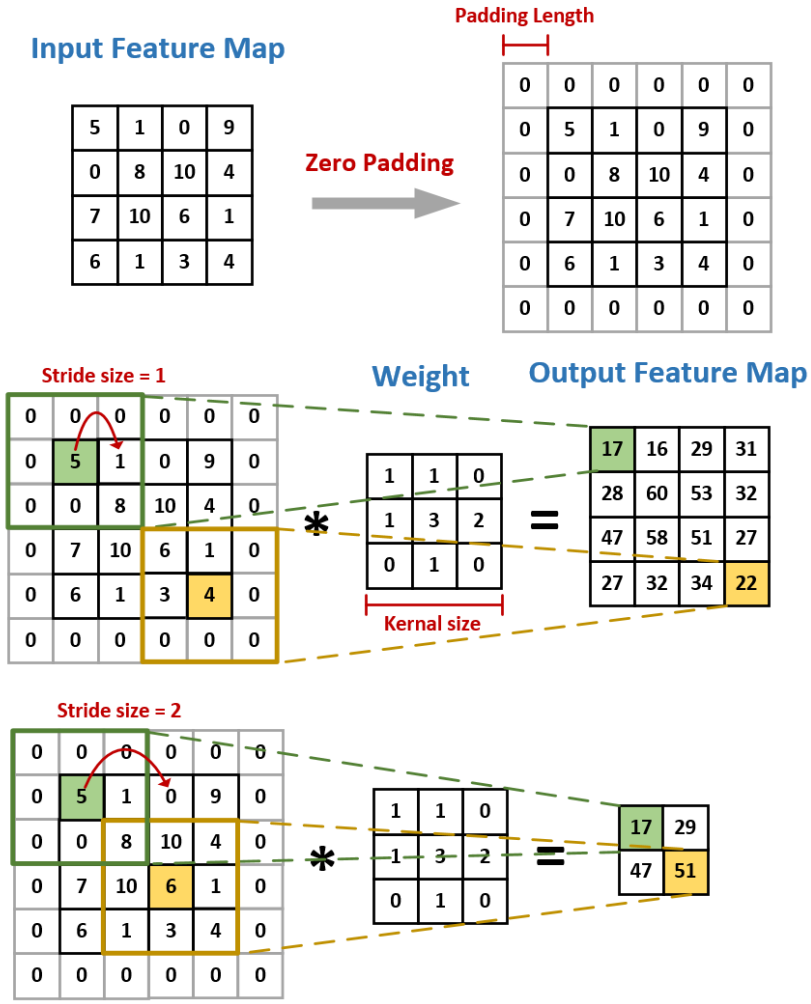



***Fig.5*** Example of convolving 4×4 feature map with 3×3 kernel using zero padding (top: stride = 1, bottom: stride = 2).

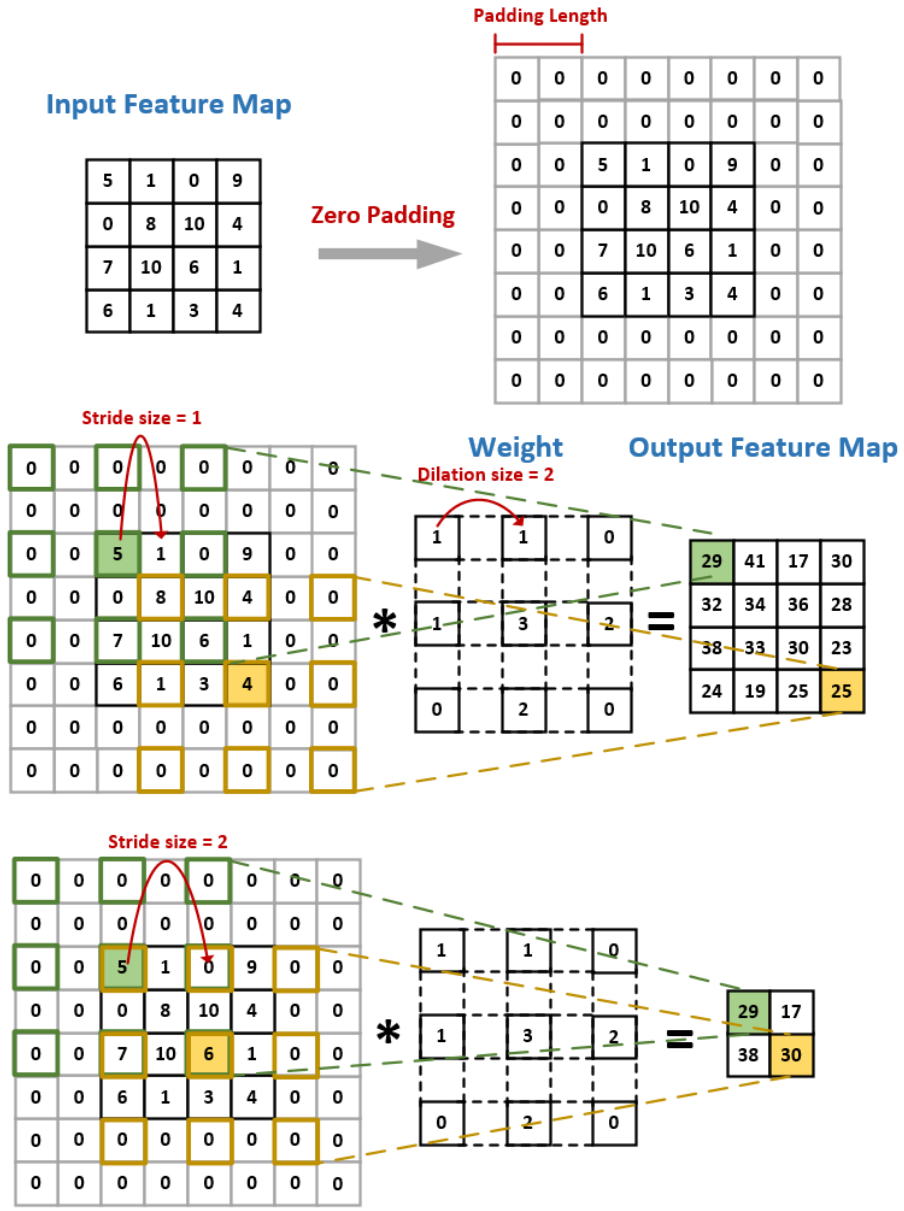



***Fig.6*** Example of convolving 4×4 feature map with 3×3 kernel using zero padding (dilation size = 2, top: stride = 1, bottom: stride = 2).

- For this homework, you have to perform convolution **over the entire image**.
- The feature map needs to be zero-padded for convolution.
  - ***Padding size = ((Kernel size - 1) * Dilation size)//2***
    - Ex: Padding size = 2 for Kernel size = 3, Dilation size = 2.
- The accumulation results should be **rounded to the nearest integer** [3].
  - Do not truncate temporary results during computation.
- After rounding, the final results should be **clamped to [0, 255]**
- The clamped final results of convolution should be written to the output

memory, and the output address is determined by **o_out_addr**.

- The size of the kernel is 3×3, and it will be loaded in **3 cycles** after the testbench verifying the correct hyperparameters. All weights are stored in 8bits signed fixed point number represented by 2's complement. Detail information of the kernel and weight format are illustrated in the figure below.

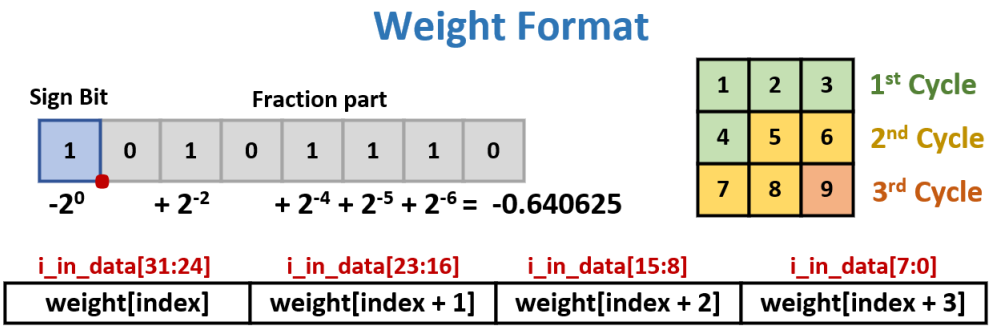
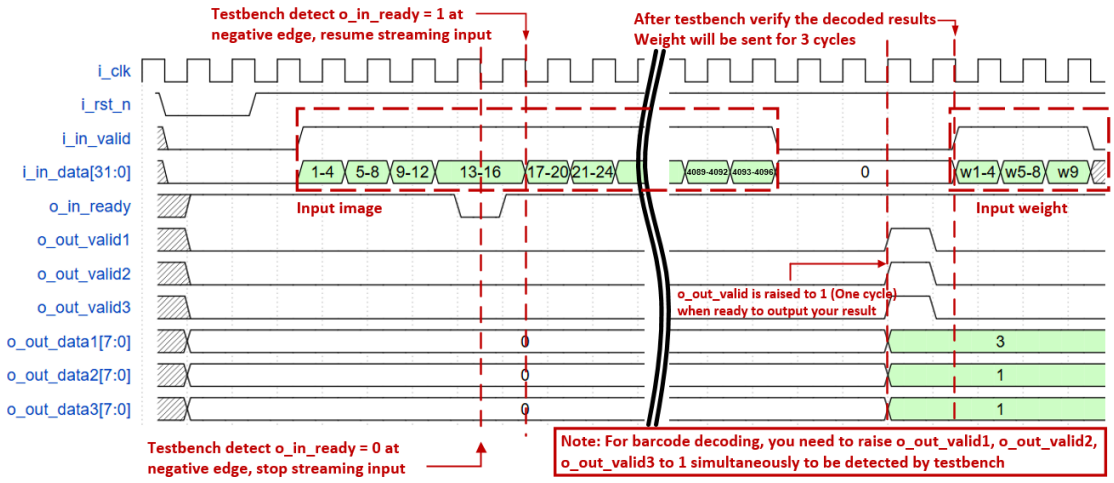



***Fig.7*** Detail information of the kernel and weight format

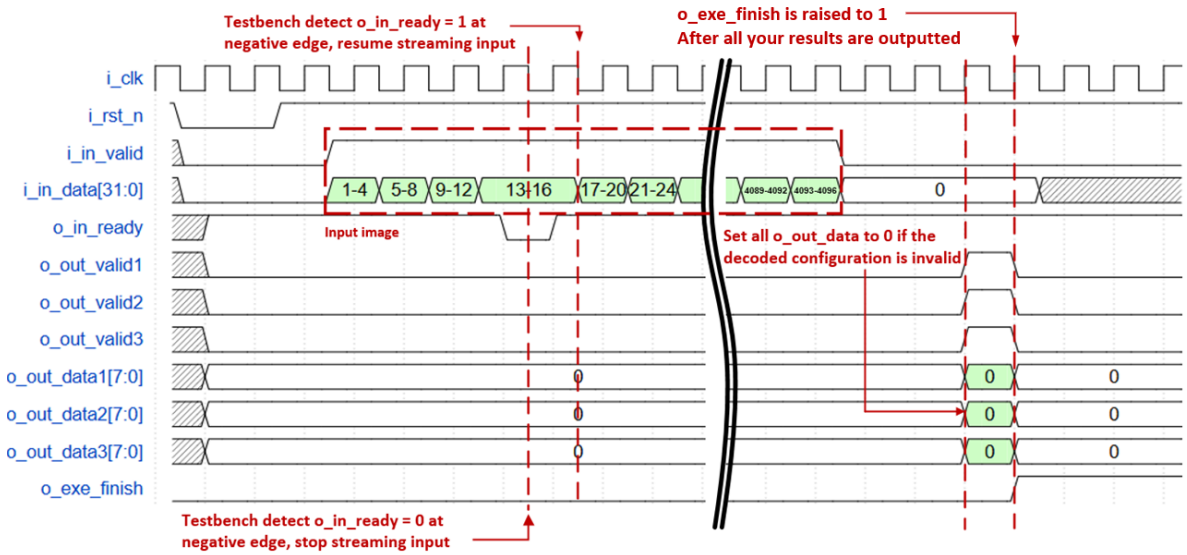
4. About convolution outputs:
   - For stride = 1, the output feature map size would be 64×64 = 4096.
   - For stride = 2, the output feature map size would be 32×32 = 1024.
   - The output address is defined in **raster-scan** sequence for both stride sizes.
   - The actual write order can be arbitrary as long as **no two outputs are written to the same address at the same time**.
   - After all your results are outputted, you should pull **o_exe_finish** to **high** so the testbench will then check all your results.

## Sample Waveform

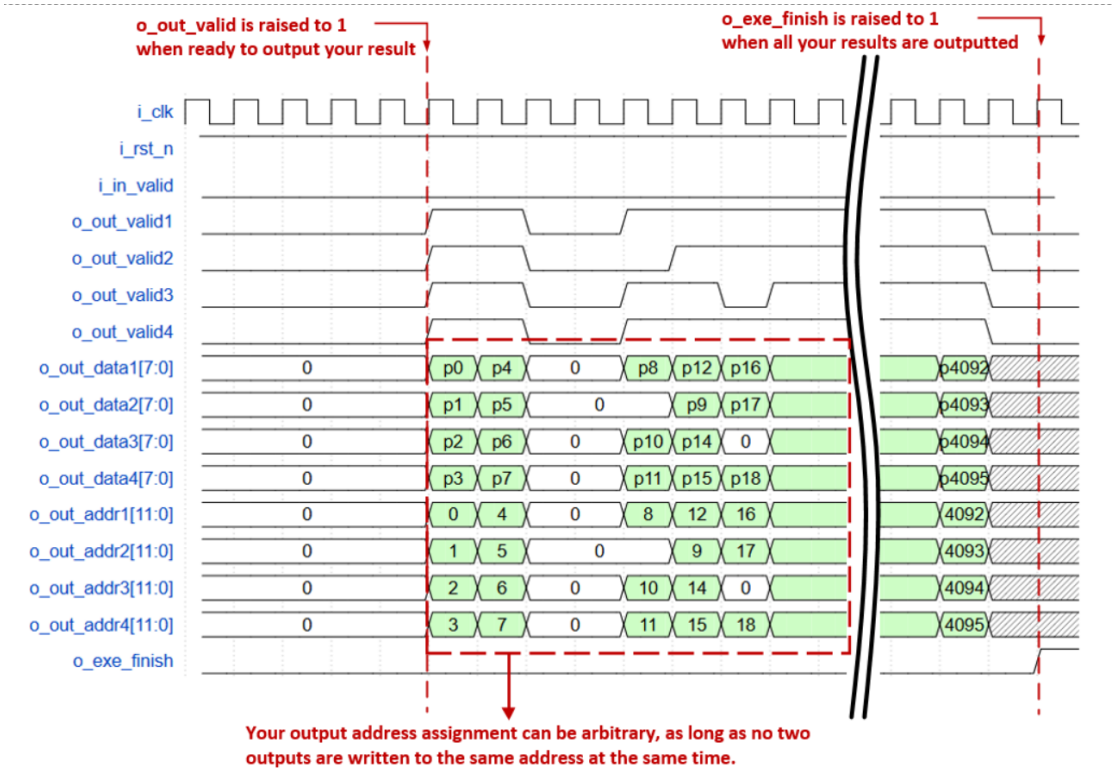
Load image data and weight



Load image (invalid configuration)




Output waveform for convolution



## Submission

1. Create a folder named **studentID_hw3** and follow the hierarchy below

```
r13943017_hw3/
    |— 01_RTL/
    |       |— core.v (and other Verilog files)
    |       |__ rtl_01.f
    |— 02_SYN/
    |       |— syn.tcl
    |       |— flist.sv
    |       |— core_dc.sdc
    |       |— Netlist/
    |       |      |— core_syn.v
    |       |      |— core_syn.sdf
    |       |      |__ core_syn.ddc
    |       |__ Report/
    |             |— core_syn.area
    |             |__ core_syn.timing
    |— 03_GATE/
    |       |__rtl_03.f
    |
    |__ report.txt
```

Note: Use **lower case** for the letter in your student ID. (Ex. r13943017_hw3)

2. Compress the folder **studentID_hw3** in a **tar file** named **studentID_hw3_v*k*.tar** (***k* is the number of version, *k* =1,2,…)**

```
tar -cvf studentID_hw3_vk.tar studentID_hw3
```

TA will only check the last version of your homework.

Note: Use **lower case** for the letter in your student ID.
(Ex. r13943017_hw3_v1.tar)

3. Submit to NTU COOL

## Grading Policy

1. TA will run your code with following format of commands.
   a. RTL simulation (under **01_RTL**)

```
vcs -f rtl_01.f -full64 -R -debug_access+all +v2k +notimingcheck
-sverilog +define+tb0
```

   b. Gate-level simulation (under **03_GATE**)

```
vcs -f rtl_03.f -full64 -R -debug_access+all +v2k +maxdelays -negdelay
+neg_tchk +define+SDF+tb0
```

2. Correctness of simulation: **70%** (follow our spec)

| Pattern | Description | RTL simulation | Gate-level simulation |
|---|---|---|---|
| **tb0** | Barcode decoding (invalid configuration) | 5% | 5% |
| **tb1** | Barcode decoding + convolution (S=1,D=1) | 5% | 10% |
| **tb2** | Barcode decoding + convolution (S=2,D=1) | 5% | 5% |
| **tb3** | Barcode decoding + convolution (S=1,D=2) | 5% | 10% |
| **tb4** | Barcode decoding + convolution (S=2,D=2) | 5% | 5% |
| **tbh** | Hidden patterns | x | 10% |

*S means stride size, D means dilation size

3. Performance: **30%**
   - Performance = **Area * Time ($\mu m^2$ * ns)**
     - **Time = total simulation time of tb1 + tb2 + tb3 + tb4**
     - The lower the value, the better the performance.
   - Performance score only counts if your design passes **all** the test patterns.
4. **No late submission is allowed**
   - Any submissions after the deadline will receive 0 point
5. Lose **5 points** for any wrong naming rule or format for submission.
   - Ensure that the submitted files can be decompressed and executed without issues
6. **No plagiarism**
   - Plagiarism in any form is strictly prohibited, including copying from online sources or past assignments
7. **Violations of any spec (p.3, p.4) incur point penalties**
   - Negative slack
     - 0 point for gate-level simulations and performance
   - Design without SRAM
     - 0 point for gate-level simulations and performance
   - Violate other rules but pass all simulations
     - Performance score * 0.7

8. **You shouldn't use improper methods** to finish the homework, we have the right to deduct your score. If we find some weird thing in your code, you must explain your purpose, otherwise, you may lose your score.

## References

[1] A guide to convolution arithmetic for deep learning
https://arxiv.org/abs/1603.07285
[2] Code 128 wiki
https://en.wikipedia.org/wiki/Code_128
[3] Rounding to the nearest
https://www.mathworks.com/help/fixedpoint/ug/rounding.html