

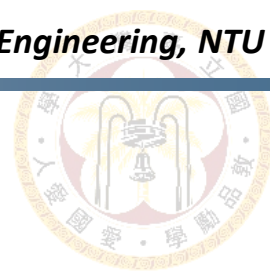
Computer-Aided VLSI System Design

Homework 3: Single-Channel Convolution with Barcode Decoding Engine

Graduate Institute of Electronics Engineering, National Taiwan University

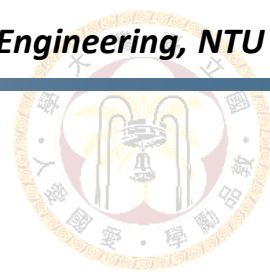


NTU GIEE



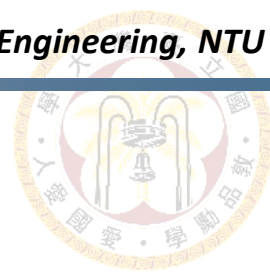
Goal

- In this homework, you will learn
 - How to synthesis your design
 - How to run gate-level simulation
 - How to integrate SRAM to your design
 - How to trade of between area and timing
 - How to generate test-pattern



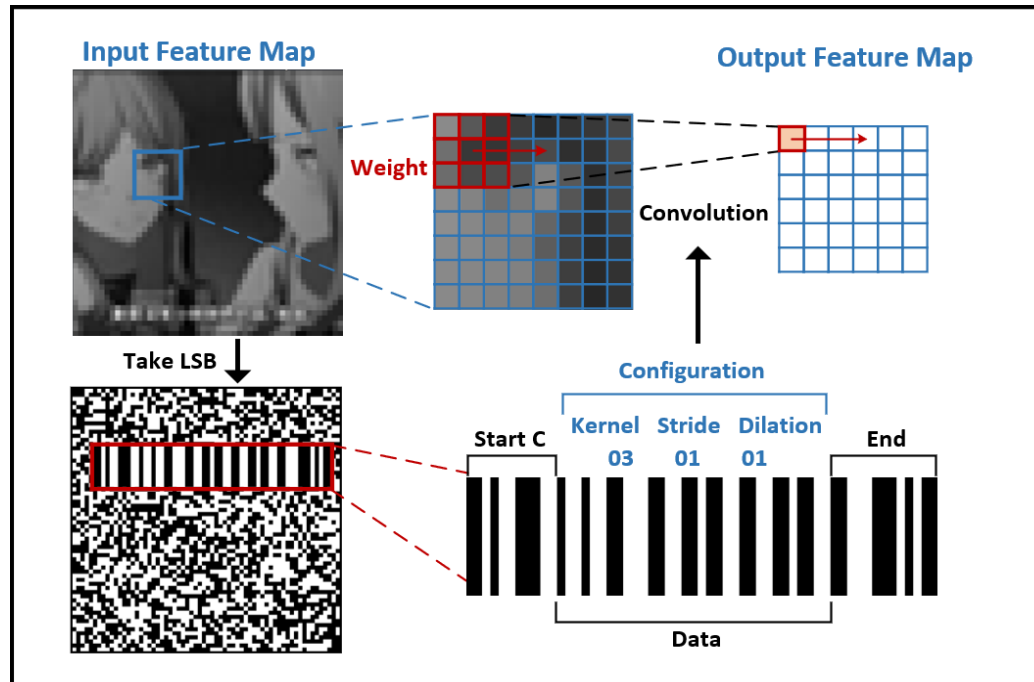
Introduction

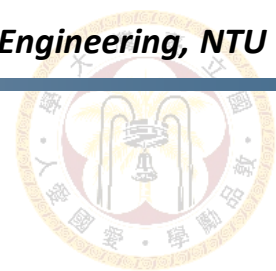
- Your client used to be a convenience store clerk. He asked you to design a convolution engine
 - Operation parameters are hidden within barcodes.
- In this homework, you are going to implement a single-channel convolution with Code 128-C barcode decoding engine.



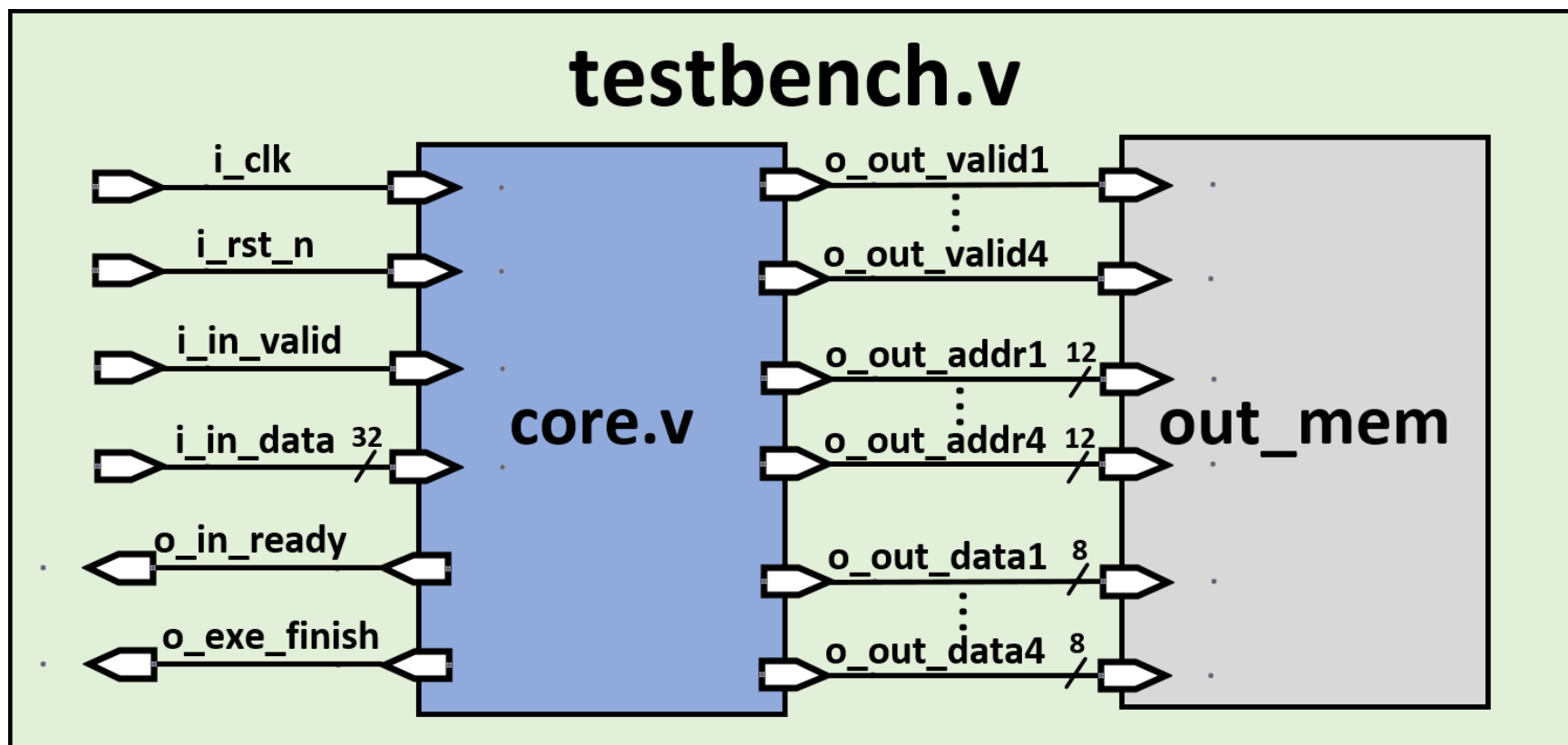
Workflow Overview

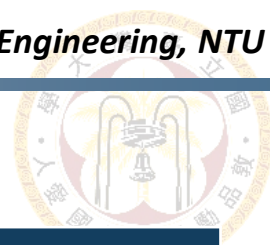
- A 64×64 grayscale image (8-bit per pixel) will be loaded as input.
- A **Code 128-C barcode** is embedded in its **least significant bits**.
- The convolution operation should be performed under different configurations, as specified by the hyperparameters encoded in the barcode.





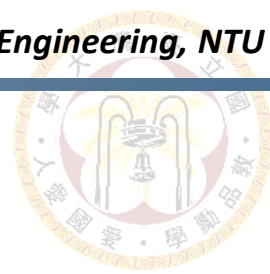
Block Diagram





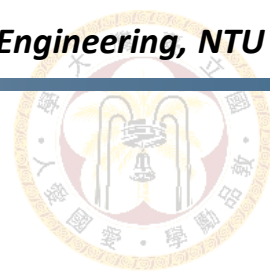
Input/Output

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 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	
o_out_valid3	O	1	
o_out_valid4	O	1	
o_out_addr1	O	12	Address of output data
o_out_addr2	O	12	
o_out_addr3	O	12	
o_out_addr4	O	12	
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



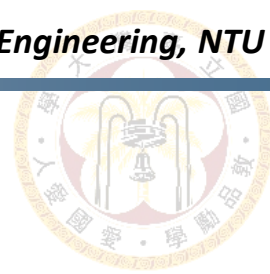
Specification (1)

- All inputs are synchronized with the **negative** edge clock
- All outputs should be synchronized at clock **rising** edge
- You should reset all your outputs when i_rst_n is **low**
 - Active low asynchronous reset is used and only once



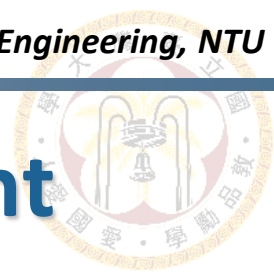
Specification (2)

- i_in_valid and o_out_valid can't be **high** at the same time
- In convolution stage, any two outputs **can't** be written to the **same address at the same time**
- o_out_valid should be **high** for valid output results
- Pull **o_exe_finish** to **high** after finish all the operation
- **At least one SRAM** is implemented in your design

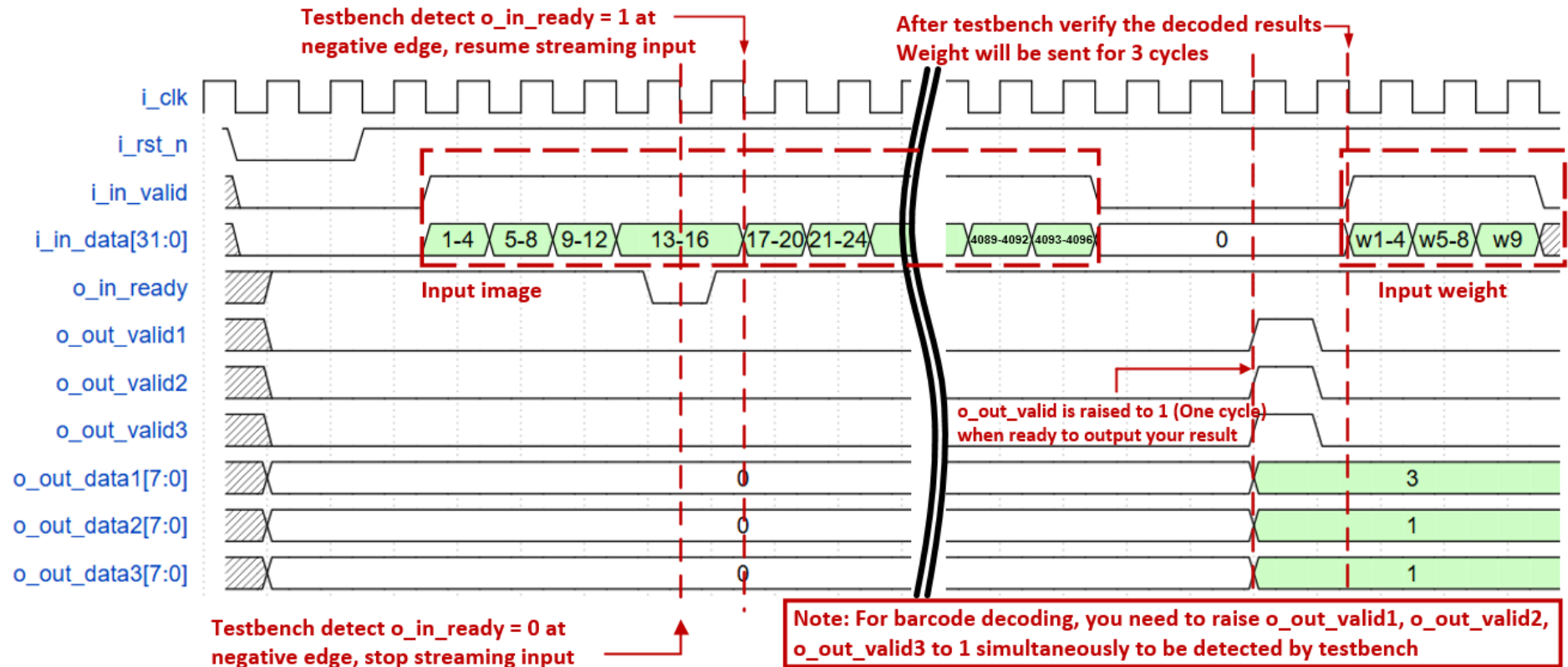


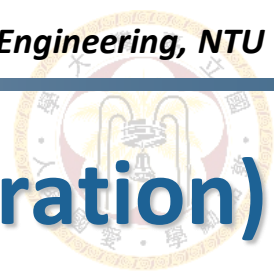
Specification (3)

- Only worst-case library is used for synthesis
- The synthesis result of data type should **NOT** include any **Latch**
- The slack for setup time should be **non-negative**
- **No timing violations or glitches** in the gate-level simulation after reset

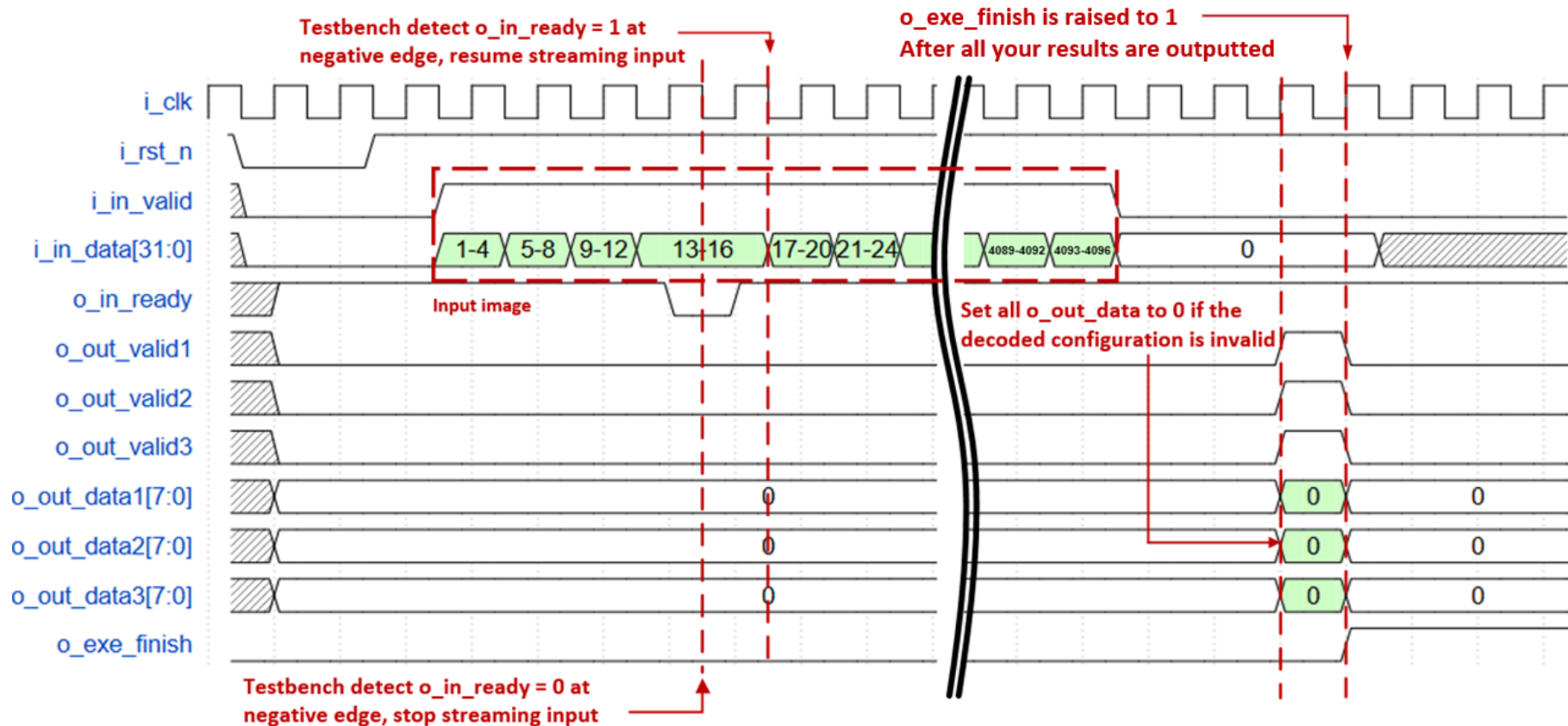


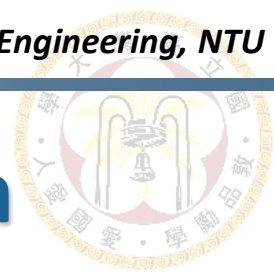
Waveform: Loading Image & Weight



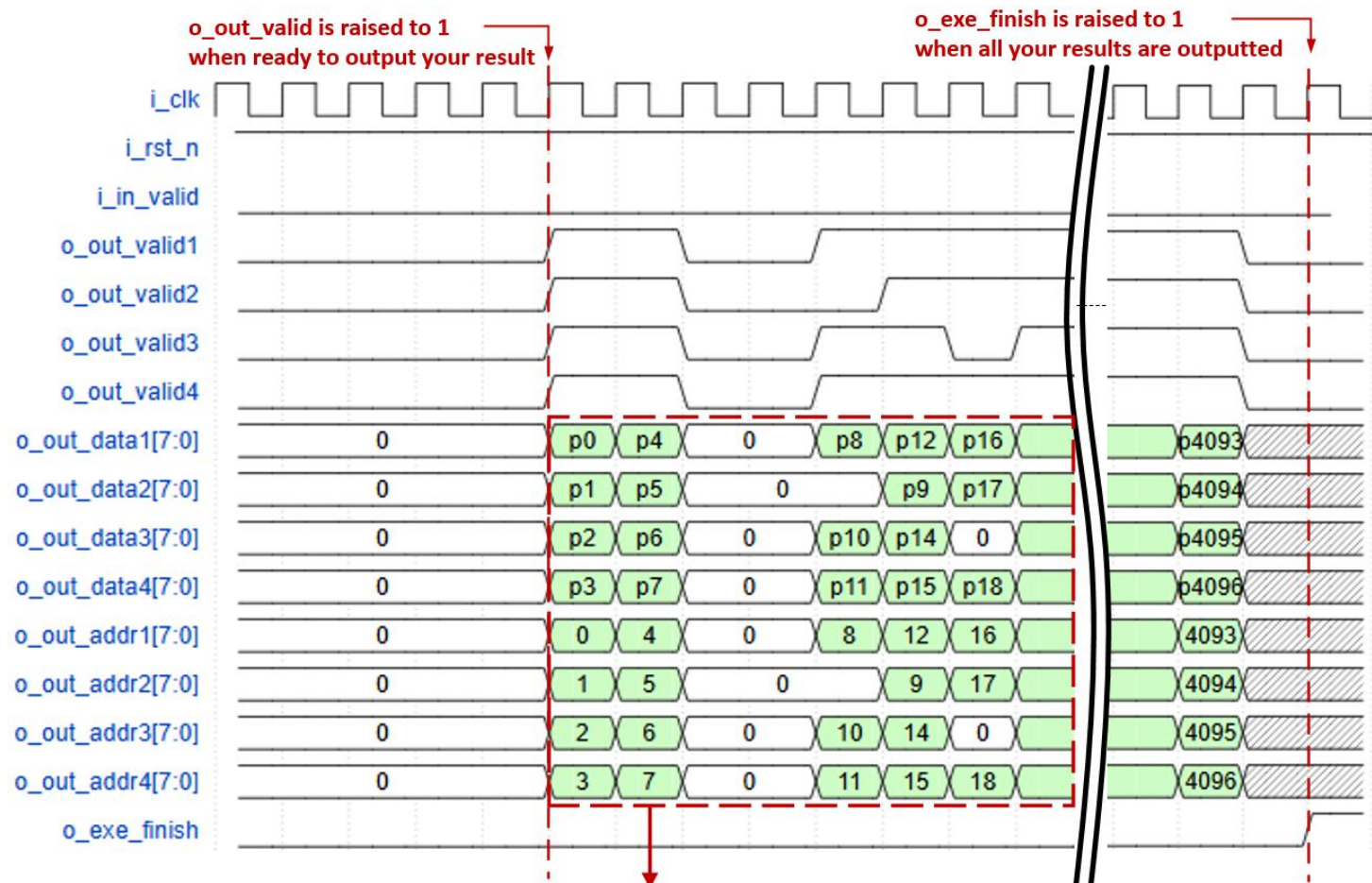


Waveform: Load Image (Invalid Configuration)

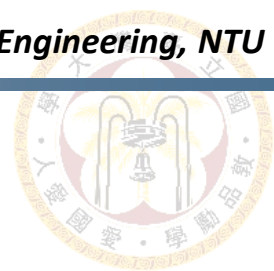




Waveform: Output of Convolution

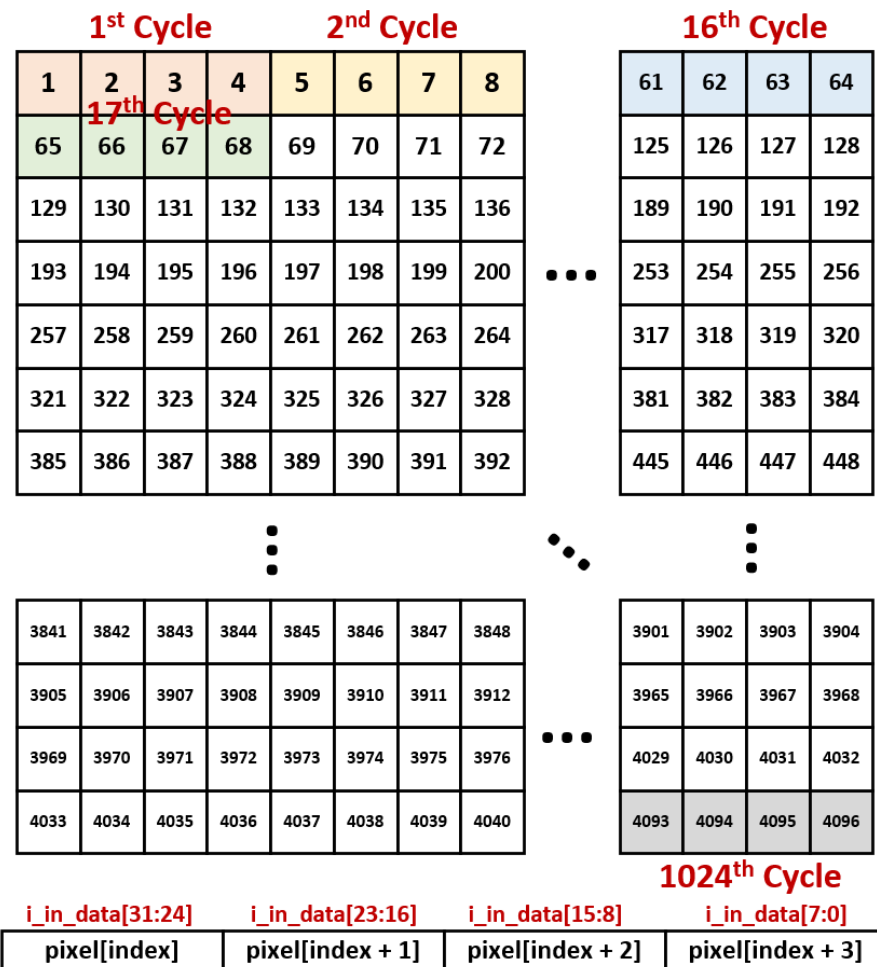


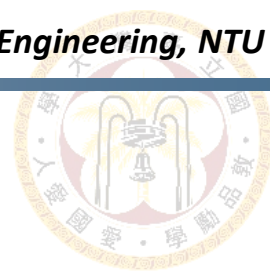
Your output address assignment can be arbitrary, as long as no two outputs are written to the same address at the same time.



Input Image

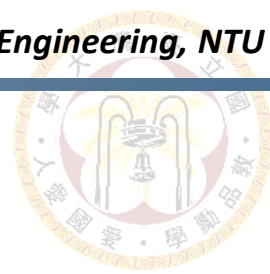
- The input image is given in **raster-scan** order





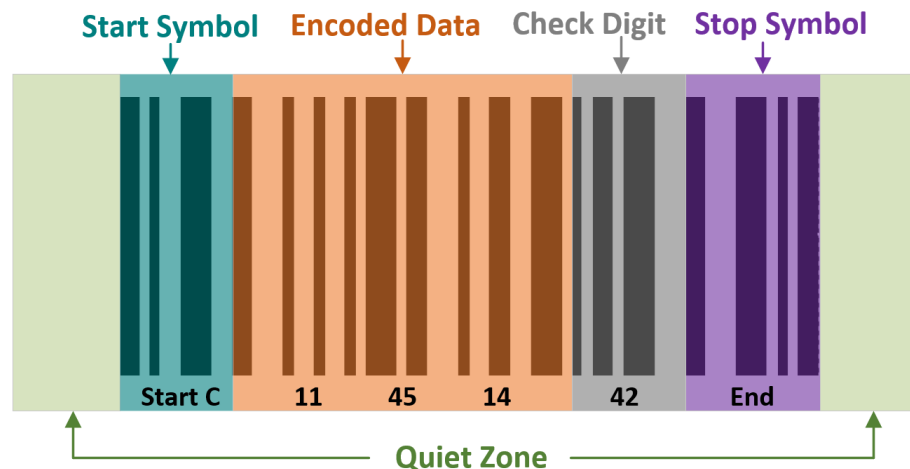
Input Image Loading

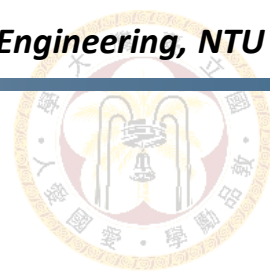
- An 64x64 image is loaded for 1024 cycles in **raster-scan** order
- The size of each pixel is 8 bits (**unsigned**)
- If **o_in_ready** is 0, stop input data until **o_in_ready** is 1
- The input feature map will be loaded only once at the beginning



Code 128 Barcode

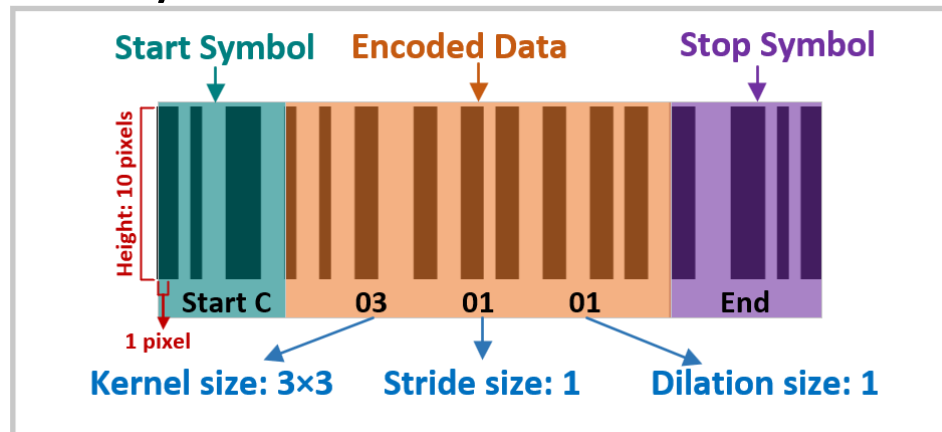
- **CODE 128** is a high-density linear barcode symbology capable of encoding all 128 ASCII characters, including control codes such as *[ESC]* and *[CR]*.
- A typical CODE 128 barcode consists of five sections:
 - Quiet zone (Not used in this homework)
 - Start Symbol
 - Encoded data
 - Check digit (Not used in this homework)
 - Stop symbol

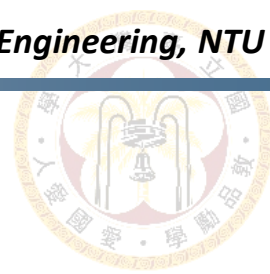




Code 128-C Barcode

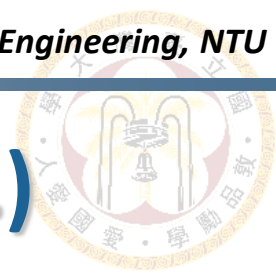
- A **Code 128-C barcode** with a height of 10 pixels is embedded within the **least significant bits (LSBs)** of input image.
- The barcode sequentially encodes three hyperparameters in following order:
 - Kernel size (K): fixed to 03
 - Stride size (S): limited to {01, 02}
 - Dilation size (D): limited to {01, 02}
- The barcode always start with **Start C** and end with **Stop Symbol**





Details of Barcode Decoding

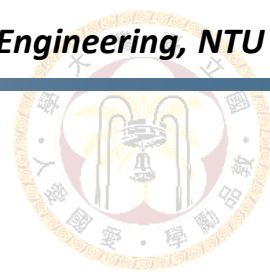
- After loading the image, your design must output the 3 hyperparameters **simultaneously**
- Use **o_out_data1**, **o_out_data2**, and **o_out_data3** to output the decoded hyperparameters:
 - o_out_data1: *Kernel size*
 - o_out_data2: *Stride size*
 - o_out_data3: *Dilation size*
- If the decoded hyperparameters do not match any of the valid combinations specified before ($K = 3, S \in \{1, 2\}, D \in \{1, 2\}$)
 - Configuration is considered **invalid**, all outputs should be set to 0.



Details of Barcode Decoding (cont.)

- The barcode can appear **anywhere in the image** but will always **stay fully inside the image boundaries**
- For convenience, a partial Code 128-C table is provided below for reference:

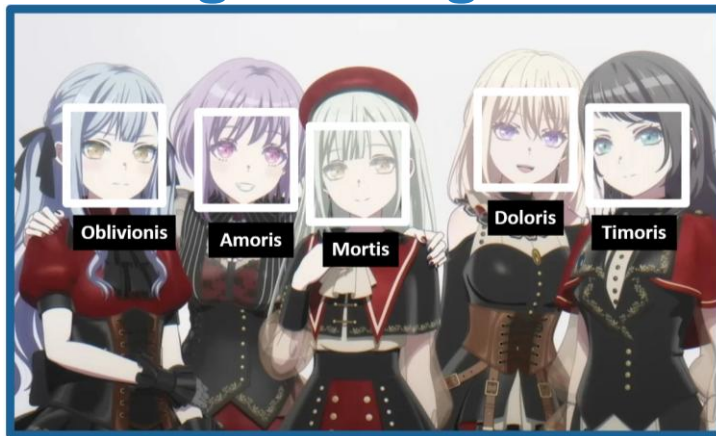
Value	128-C	Pattern	Widths
1	01	11001101100	222122
2	02	11001100110	222221
3	03	10010011000	121223
105	Start Code C	11010011100	211232
—	Stop symbol	1100011101011	2331112



Introduction to Convolution

- Convolution is widely used in image processing and computer vision applications
- Applications:

Image Recognition

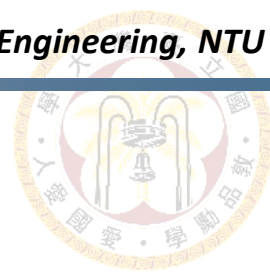


Source: <https://ani.gamer.com.tw/>

Image Segmentation

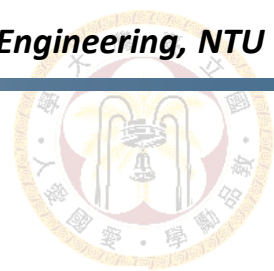


- For this homework, you have to implement a single channel convolution processor with 3×3 kernel size, supporting different stride size and dilation size



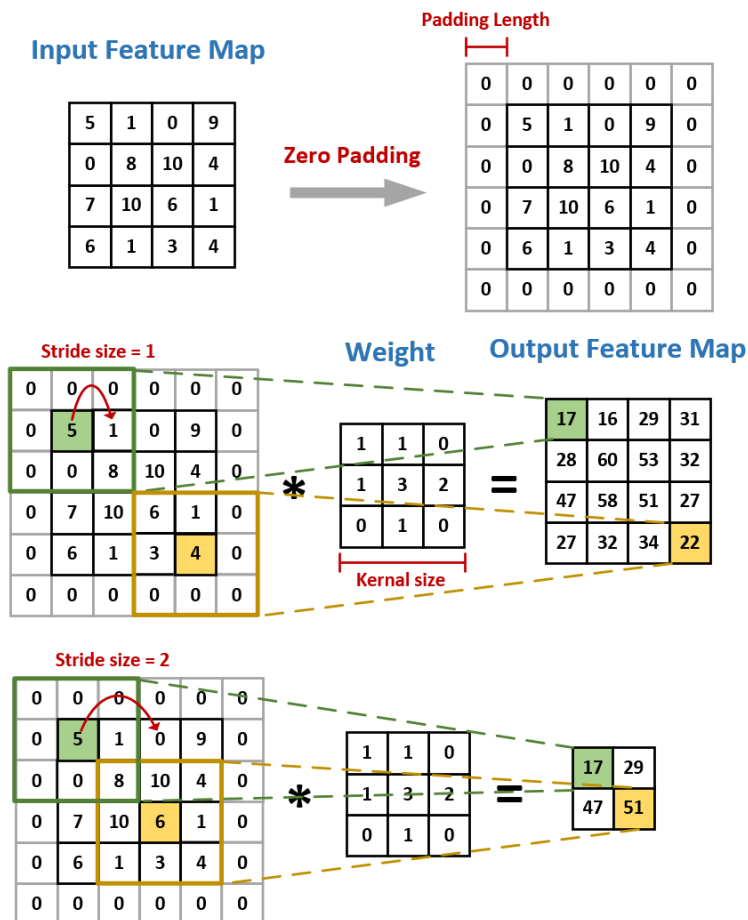
Convolution Operation

- 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** [1]
 - 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
 - Output address is determined by **o_out_addr**.

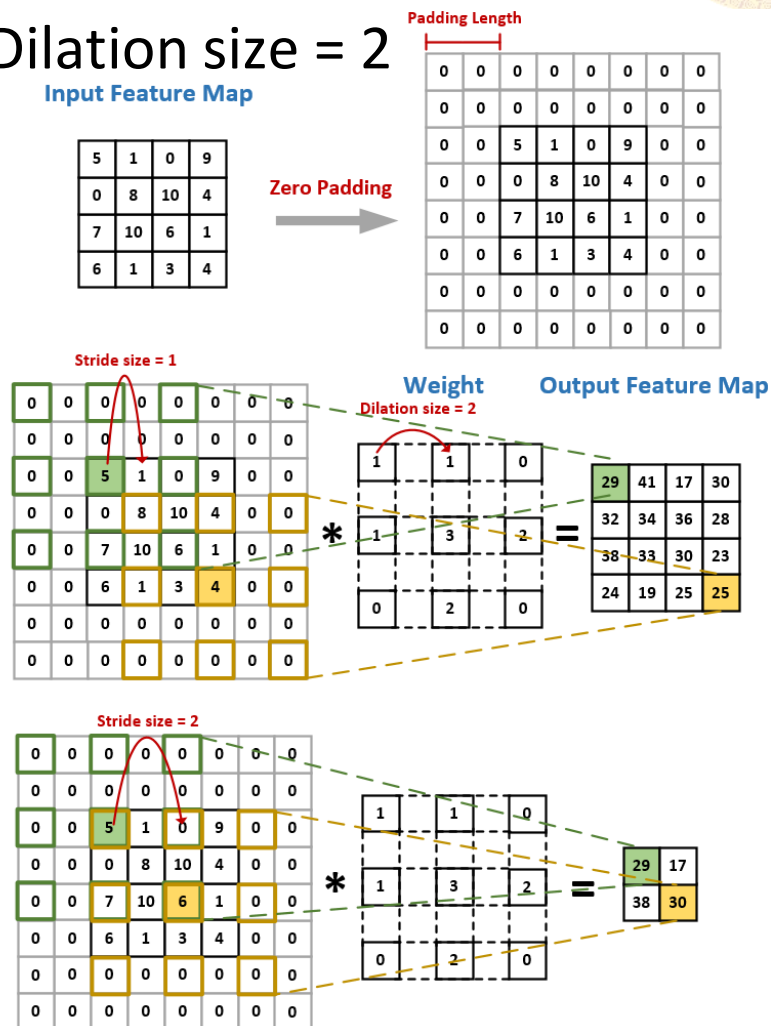


Example of Convolution

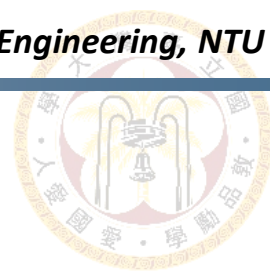
■ Dilation size = 1



■ Dilation size = 2



■ Or you can refer to [2](with animation)



Weight Format

- The size of the kernel is 3×3
- 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

Sign Bit

Fraction part

1	0	1	0	1	1	1	0
---	---	---	---	---	---	---	---

 -2^0 $+2^{-2}$ $+2^{-4} + 2^{-5} + 2^{-6} = -0.640625$

1	2	3	1 st Cycle
4	5	6	2 nd Cycle
7	8	9	3 rd Cycle

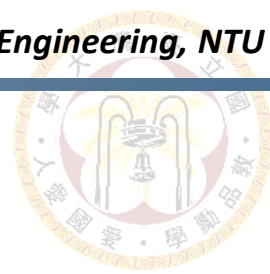
 $i_in_data[31:24]$ $i_in_data[23:16]$ $i_in_data[15:8]$ $i_in_data[7:0]$

weight[index]

weight[index + 1]

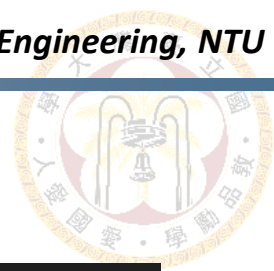
weight[index + 2]

weight[index + 3]



About Convolution Outputs

- For stride = 1, the output feature map size would be 64×64
- For stride = 2, the output feature map size would be 32×32
- 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.



Testbench

```

`timescale 1ns/1ps
`define CYCLE      5.0    // CLK period.
`define HCYCLE     (`CYCLE/2)
`define MAX_CYCLE  20000
`define RST_DELAY  2

`ifdef tb1
  `define INFILE  "../00_TESTBED/PATTERN/img1_030101_00.dat"
  `define WFILE   "../00_TESTBED/PATTERN/weight_img1_030101_00.dat"
  `define GOLDEN  "../00_TESTBED/PATTERN/golden_img1_030101_00.dat"
  `define K_SIZE  3
  `define S_SIZE  1
  `define D_SIZE  1
  `define VALID_OP 1
  `define OUTPUTSIZE 4096
`elseif tb2
  `define INFILE  "../00_TESTBED/PATTERN/img1_030102_053.dat"
  `define WFILE   "../00_TESTBED/PATTERN/weight_img1_030102_053.dat"
  `define GOLDEN  "../00_TESTBED/PATTERN/golden_img1_030102_053.dat"
  `define K_SIZE  3
  `define S_SIZE  1
  `define D_SIZE  2
  `define VALID_OP 1
  `define OUTPUTSIZE 4096
`elseif tb3
  `define INFILE  "../00_TESTBED/PATTERN/img1_030201_70.dat"
  `define WFILE   "../00_TESTBED/PATTERN/weight_img1_030201_70.dat"
  `define GOLDEN  "../00_TESTBED/PATTERN/golden_img1_030201_70.dat"
  `define K_SIZE  3
  `define S_SIZE  2
  `define D_SIZE  1
  `define VALID_OP 1
  `define OUTPUTSIZE 1024

```

```

`elseif tb4
  `define INFILE  "../00_TESTBED/PATTERN/img1_030202_753.dat"
  `define WFILE   "../00_TESTBED/PATTERN/weight_img1_030202_753.dat"
  `define GOLDEN  "../00_TESTBED/PATTERN/golden_img1_030202_753.dat"
  `define K_SIZE  3
  `define S_SIZE  2
  `define D_SIZE  2
  `define VALID_OP 1
  `define OUTPUTSIZE 1024
// `elseif tbh
// `define INFILE  "../00_TESTBED/PATTERN/.dat"
// `define WFILE   "../00_TESTBED/PATTERN/.dat"
// `define GOLDEN  "../00_TESTBED/PATTERN/.dat"
// `define K_SIZE
// `define S_SIZE
// `define D_SIZE
// `define VALID_OP
// `define OUTPUTSIZE
`else
  `define INFILE  "../00_TESTBED/PATTERN/img1_050102_514.dat"
  `define WFILE   "../00_TESTBED/PATTERN/weight_img1_050102_514.dat"
  `define GOLDEN  "../00_TESTBED/PATTERN/golden_img1_050102_514.dat"
  `define K_SIZE  0
  `define S_SIZE  0
  `define D_SIZE  0
  `define VALID_OP 0
  `define OUTPUTSIZE 0
`endif

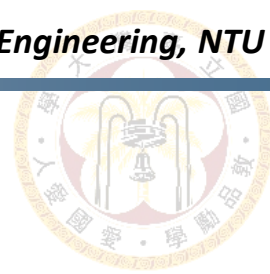
// Modify your sdf file name
`define SDDFILE "../02_SYN/Netlist/core_syn.sdf"

```

```

// For gate-level simulation only
`ifdef SDF
  initial $sdf_annotate(`SDDFILE, u_core);
  initial #1 $display("SDF File %s were used for this simulation.", `SDDFILE);
`endif

```

Pattern

- Total **5** images of test pattern are provided for debugging!
 - For public test case, only **img1** is used
 - Does not include any hidden pattern

{k_size, s_size, d_size}

img1_030101_00.dat

Barcode data

```

1  0F
2  0F
3  0E
4  0F
5  0E
6  0E
7  0F
8  0F
9  0F
10 0E
11 0E
12 0F
13 0E
14 0E
15 09
16 02
17 02
18 03
19 03
20 02
21 02
22 04
23 03
24 03
25 02
26 04
27 05
28 05
29 04
  
```

{col, row}

weight_img1_030101_00.dat

Barcode position (top left corner)

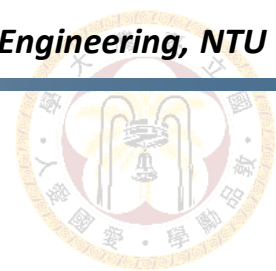
```

1  C6
2  77
3  E8
4  7D
5  E1
6  49
7  E8
8  C0
9  2E
10 
  
```

golden_img1_030101_00.dat

```

1  03
2  0D
3  0F
4  0C
5  0E
6  0E
7  0D
8  0E
9  0D
10 0E
11 0E
12 0C
13 0D
14 07
15 09
16 08
17 02
18 01
19 01
20 02
21 02
22 02
23 03
24 01
25 04
26 03
27 03
28 03
29 0E
  
```



01_RTL

■ core.v

```
module core (                                     //Don't modify interface
    input      i_clk,
    input      i_rst_n,
    input      i_in_valid,
    input [31: 0] i_in_data,

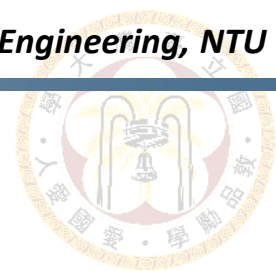
    output      o_in_ready,

    output [ 7: 0] o_out_data1,
    output [ 7: 0] o_out_data2,
    output [ 7: 0] o_out_data3,
    output [ 7: 0] o_out_data4,

    output [11: 0] o_out_addr1,
    output [11: 0] o_out_addr2,
    output [11: 0] o_out_addr3,
    output [11: 0] o_out_addr4,

    output      o_out_valid1,
    output      o_out_valid2,
    output      o_out_valid3,
    output      o_out_valid4,

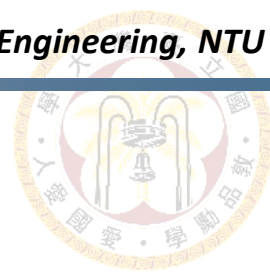
    output      o_exe_finish
);
```



01_RTL

■ rtl_01.f

```
// -----  
// Simulation: HW3  
// -----  
  
// testbench  
// -----  
../00_TESTBED/testbench.v  
  
// memory file  
// -----  
../sram_256x8/sram_256x8.v  
../sram_512x8/sram_512x8.v  
../sram_4096x8/sram_4096x8.v  
  
// design files  
// -----  
./core.v
```



01_RTL

- Run the RTL simulation under 01_RTL folder

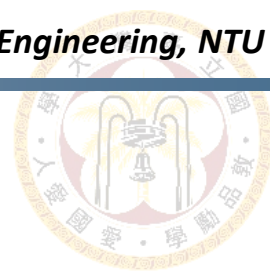
```
vcs -f rtl_01.f -full64 -R -debug_access+all +v2k  
+notimingcheck -sverilog +define+tb0
```

tb0, tb1, tb2, tb3, tb4

or

```
./01_run tb0 5.0
```

clock period



01_RTL

- SpyGlass linting

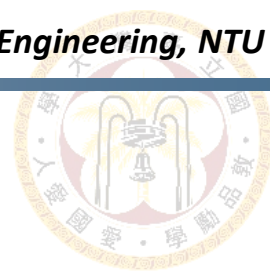
```
./02_lint
```

- Command for cleaning temporary files

```
./99_clean_up
```

- Note that before executing the shell script, change the file permissions by

```
chmod +x ./01_run ./02_lint ./99_clean_up
```



02_SYN

- **core_dc.sdc**

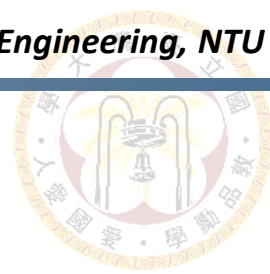
```
# operating conditions and boundary conditions #  
set cycle 5.0; # modify your clock cycle here #
```

- **flist.sv**

```
1 // list all paths to your design files  
2 `include "../01_RTL/core.v"
```

- Run the command to do synthesis

```
dc_shell-t -f syn.tcl | tee syn.log
```



03_GATE

- Run gate-level simulation under 03_GATE folder

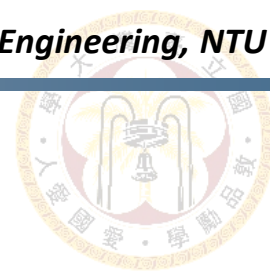
```
vcs -f rtl_03.f -full64 -R -debug_access+all +v2k  
+maxdelays -negdelay +neg_tchk +define+SDF+tb0
```

tb0, tb1, tb2, tb3, tb4

or

```
./03_run tb0 5.0
```

clock period



sram_256x8

Pin Description

Pin	Description
A[7:0]	Addresses (A[0] = LSB)
D[7:0]	Data Inputs (D[0] = LSB)
CLK	Clock Input
CEN	Chip Enable
WEN	Write Enable
Q[7:0]	Data Outputs (Q[0] = LSB)

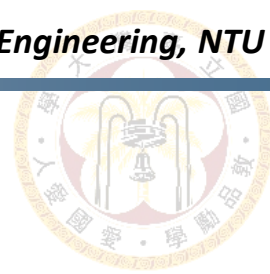
Area

Area Type	Width (mm)	Height (mm)	Area (mm ²)
Core	0.143	0.125	0.018
Footprint	0.153	0.136	0.021

The footprint area includes the core area and user-defined power ring and pin spacing areas.

SRAM Logic Table

CEN	WEN	Data Out	Mode	Function
H	X	Last Data	Standby	Address inputs are disabled; data stored in the memory is retained, but the memory cannot be accessed for new reads or writes. Data outputs remain stable.
L	L	Data In	Write	Data on the data input bus D[n-1:0] is written to the memory location specified on the address bus A[m-1:0], and driven through to the data output bus Q[n-1:0].
L	H	SRAM Data	Read	Data on the data output bus Q[n-1:0] is read from the memory location specified on the address bus A[m-1:0].

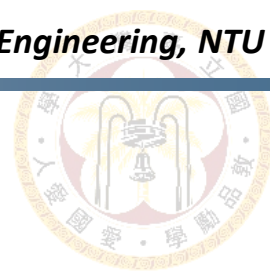


Submission

- Create a folder named **studentID_hw3** and follow the hierarchy below

```
r11943006_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
```

- Compress the folder **studentID_hw3** in a tar file named **studentID_hw3_vk.tar** (k is the number of version, $k = 1, 2, \dots$)
 - Use lower case for the letter in your student ID
(Ex. r13943017_hw3_v1.tar)



Grading Policy

- 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%

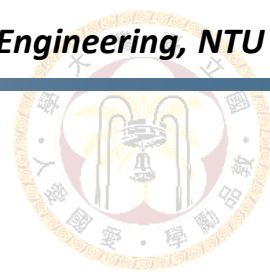
*S means stride size, D means dilation size

- Performance: **30%**

– Performance = **Area * Time** ($\mu\text{m}^2 * \text{ns}$)

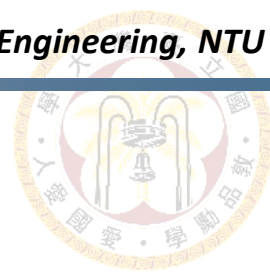
- **Time = total simulation time of tb1 + tb2 + tb3 + tb4**
- The lower the value, the better the performance

– Performance score only counts if your design passes all the test patterns



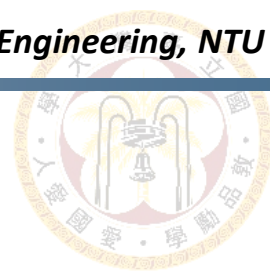
Grading Policy

- **No late submission is allowed**
 - Any submissions after the deadline will receive 0 point
- **Lose 5 points** for any wrong naming rule or format for submission
 - Ensure that the submitted files can be decompressed and executed without issues
- **No plagiarism**
 - Plagiarism in any form is strictly prohibited, including copying from online sources or past assignments



Grading Policy

- **Violations of any spec (p.7 – p.9) 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
- You should NOT use the **improper method** to finish the homework, we have the right to deduct your score
 - Try to copy and paste the golden pattern to your code
 - Make the synthesis circuit and RTL code mismatch deliberately
- If we find some weird thing in your code, you must explain your purpose, otherwise, you may lose your score



Area

■ core_syn.area

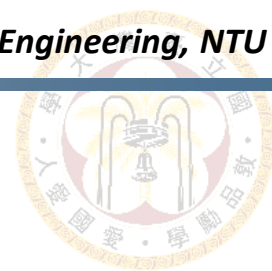
```
Number of ports:          1058
Number of nets:           30734
Number of cells:          27064
Number of combinational cells: 23297
Number of sequential cells: 3721
Number of macros/black boxes: 8
Number of buf/inv:        3032
Number of references:      7

Combinational area:       241592.639554
Buf/Inv area:             17281.229314
Noncombinational area:    128951.475231
Macro/Black Box area:     247583.093750
Net Interconnect area:    3140029.522614

Total cell area:          618127.208535
Total area:               3758156.731149
```

Number of macros/black boxes
should not be 0

618127.208535 μm^2



Time

■ Simulation Result

```
-----
STAGE 1:  BARCODE DECODING
-----
All Configurations Correct! Permission Granted to Enter STAGE 2
-----
STAGE 2:  CONVOLUTION
-----

*****
*   OVERALL COMPARISON RESULTS   *
*****

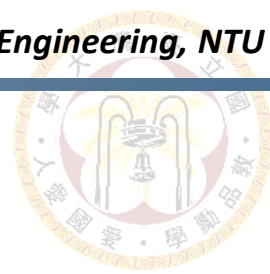
#   #####
#   #
#   #   CORRECT   #
#   #   #   #   #
#   #   #####

-----
CONGRATULATION! ALL DATA PASS!
-----

-----
Total SPEC Error:      0
-----
$finish called from file "../00_TESTBED/testbench.v", line 374.
$finish at simulation time      7535500
VCS Simulation Report
Time: 7535500 ps
```

7535.5 ns

Number of SPEC Error should be 0



Report

- Your design will be simulated using the clock period in report.txt
- **report.txt**

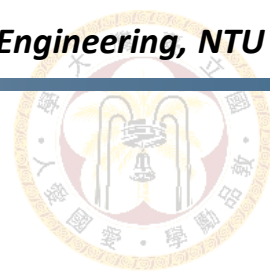
```
student ID:  r13943017

Clock Period: 3.5 (ns)

simulation time for tb1: 7535.5 (ns)
simulation time for tb2: ...
simulation time for tb3: ...
simulation time for tb4: ...

Area: 618127.208535 (um^2)
```

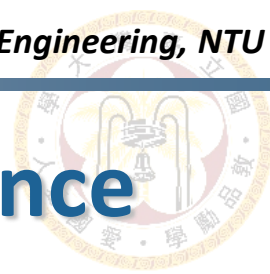
The clock period that can pass all gate-level simulations without any timing violations



Notice

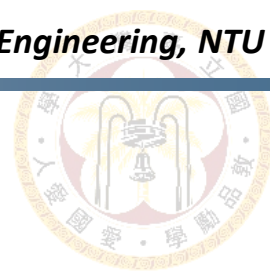
- **This homework is challenging**
 - You also have to prepare for **Midterm**
 - Make sure to start your RTL design earlier
- For those who doesn't have enough time
 - Notice that all stride=2 outputs is the same to 2x down-sampling of those of stride=1 (refer to p.21) 😊





Some Guidelines for Better Performance

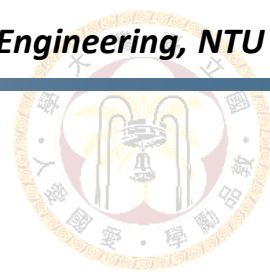
- Performance can be decomposed to $\text{Area} * \text{Cycle time} * \# \text{ of cycles}$
 - Hardware (multipliers) reuse
 - **Area↓**
 - Increasing **hardware utilization rate**, that is, make the idle cycles of multipliers as less as possible
 - **# of cycles↓ , Area ↓**
 - Do more pipelining, avoid long path
 - SRAM, Multipliers, In/output, etc.
 - **Cycle time ↓**
 - A more efficient memory mapping, data flow
 - **# of cycles↓, Area of control logic↓**



Discussion

- **NTU Cool Discussion Forum**
 - For any questions not related to assignment answers or privacy concerns, please use the NTU Cool discussion forum
 - **TAs will prioritize answering questions on the NTU Cool discussion forum**

- **Email: r13943017@ntu.edu.tw**
 - Title should start with **[CVSD 2025 Fall HW3]**
 - Email with wrong title will be moved to trash automatically



References

- [1] Rounding to the nearest
 - [Rounding - MATLAB & Simulink \(mathworks.com\)](https://www.mathworks.com/help/matlab/matlab_prog/rounding-to-the-nearest.html)
- [2] A guide to convolution arithmetic for deep learning
 - <https://arxiv.org/abs/1603.07285>
- [3] Code 128 wiki
 - https://en.wikipedia.org/wiki/Code_128