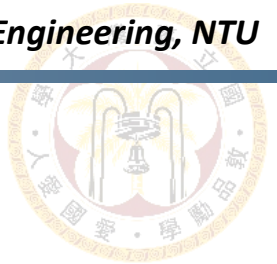
Advanced Computer-Aided VLSI System Design

Midterm Project: Single-Layer Convolution Engine with Quantization

Graduate Institute of Electronics Engineering, National Taiwan University

NTU GIEE

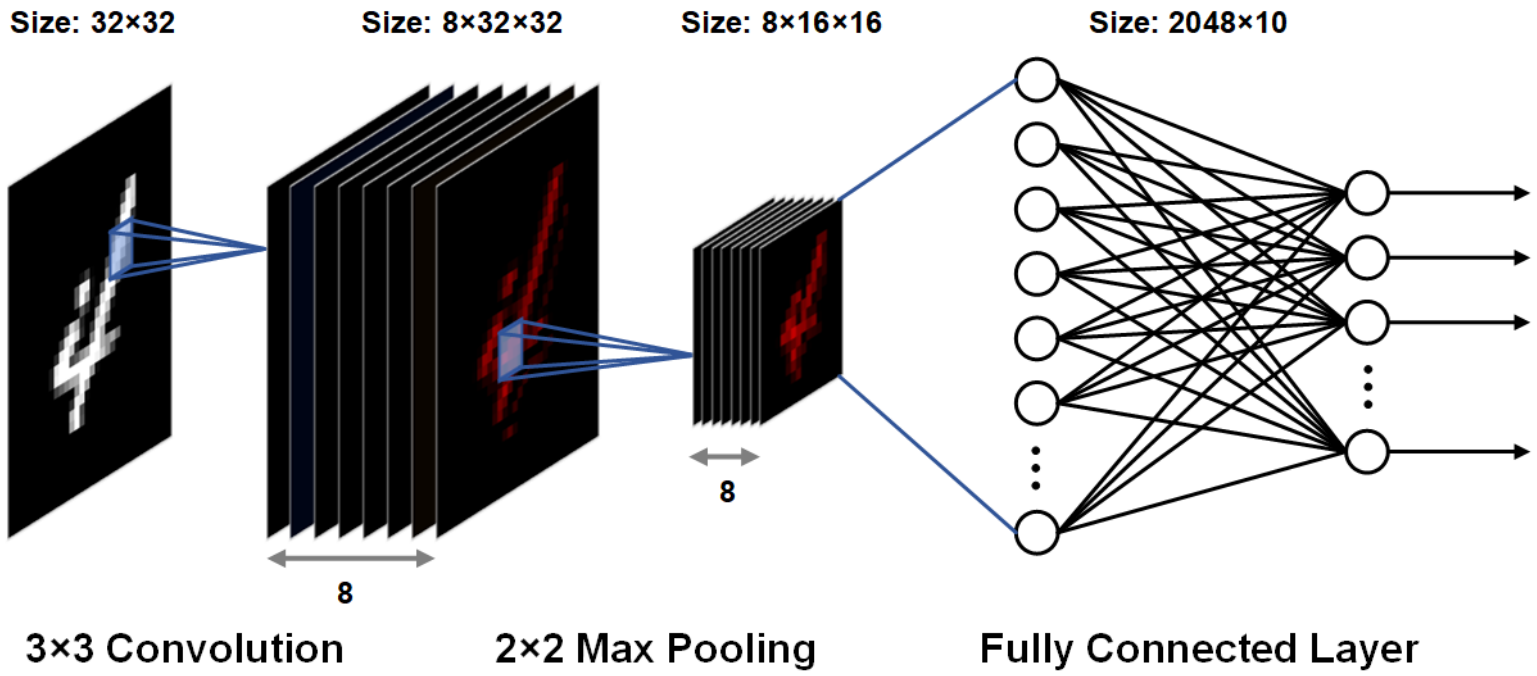


Goals

- In this homework, you will learn
  - AI accelerator with sparsity codec and quantization
  - Multi-bit Clock Domain Crossing technique
  - Communication with external memory via AXI bus

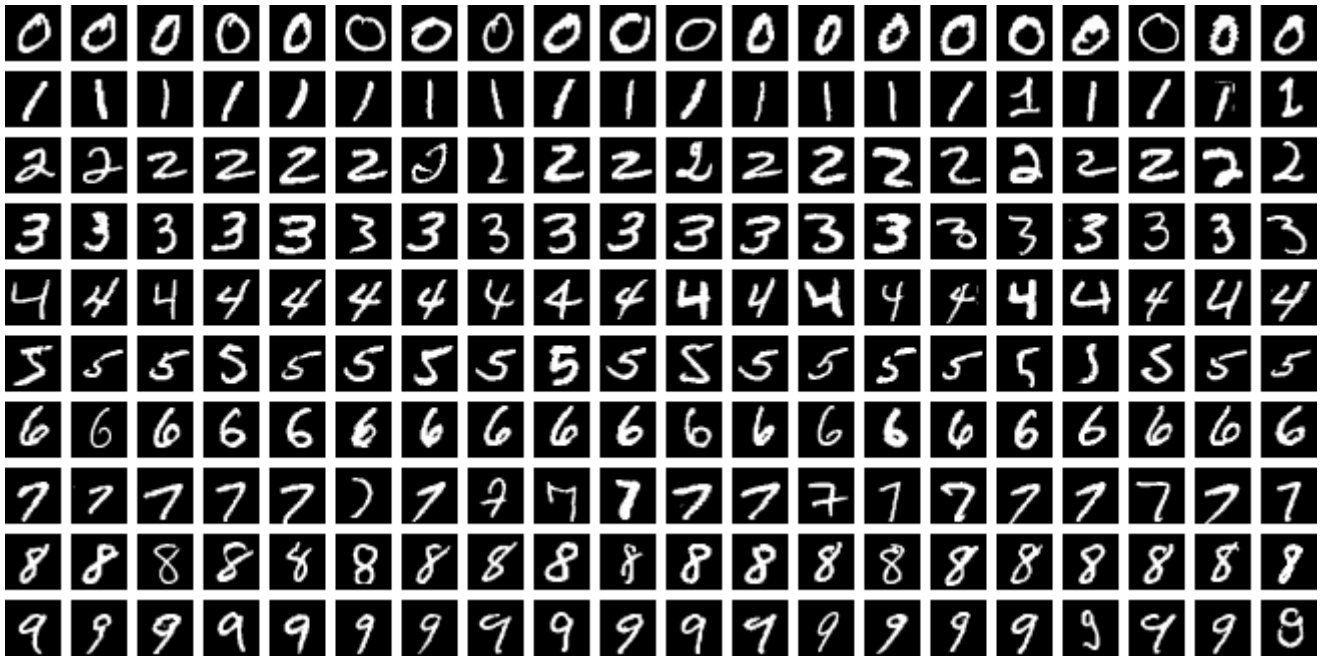
Introduction

- You are going to implement a convolution engine to accelerate convolution, ReLU, max pooling operations for the MNIST classifier below
- Sparsity codec and quantization are considered in the engine

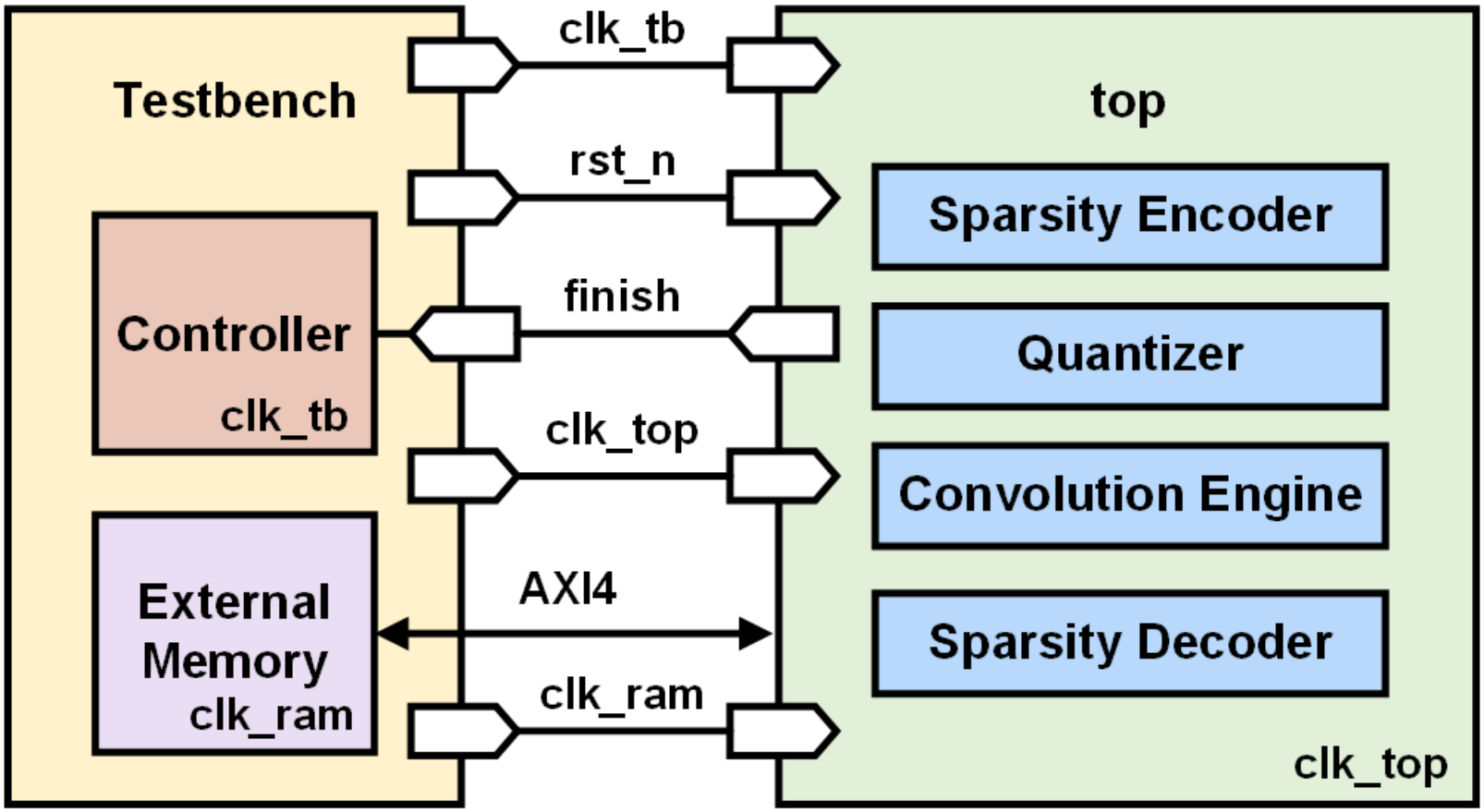


MNIST Dataset

- MNIST dataset includes 70,000 handwritten digits for training an image processor
- It is commonly-used in machine learning for computer vision



Block Diagram



Input/Output

Signal Name	I/O	Width	Simple Description
clk_tb	I	1	Clock for control signal (positive edge trigger). Inputs have a **half-cycle delay**. Outputs should be synchronized at clock **rising** edge.
clk_top	I	1	Clock for your design (positive edge trigger).
clk_ram	I	1	Clock for AXI channel (positive edge trigger). Inputs are synchronized with the positive edge clock. Outputs should be synchronized at clock **rising** edge.
rst_n	I	1	Active **low** synchronous reset.
finish	O	1	Output finish signal. The signal should be asserted **high for three cycles** to indicate the end of computation.

Input/Output

Signal Name	I/O	Width	Simple Description
awaddr araddr	O	15	AXI Write / Read request channel. Address of the first transfer in a transaction.
awlen arlen	O	8	AXI Write / Read request channel. Total number of transfers in a transaction.
awsize arsize	O	3	AXI Write / Read request channel. Maximum number of bytes in a transfer within a transaction.
awburst arburst	O	2	AXI Write / Read request channel. Mode of the address increment.
awvalid arvalid	O	1	AXI Write / Read request channel. Write / Read request valid indicator.
awready arready	I	1	AXI Write / Read request channel. Write / Read request ready indicator.

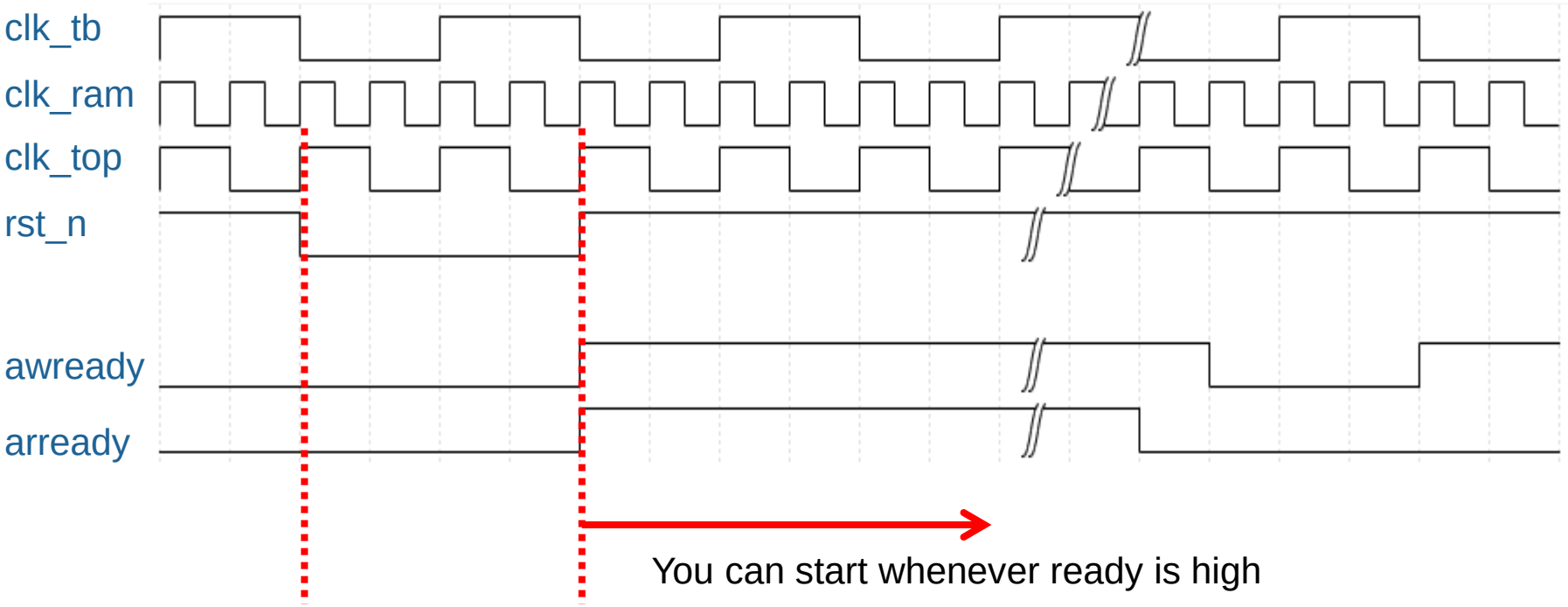
Input/Output

Signal Name	I/O	Width	Simple Description
wdata **rdata**	O I	8	AXI Write / Read data channel. Write / Read data.
wlast **rlast**	O I	1	AXI Write / Read data channel. Indicator of the last Write / Read transfer in a transaction.
wvalid **rvalid**	O I	1	AXI Write / Read data channel. Write / Read data valid indicator.
wready **rready**	I O	1	AXI Write / Read data channel. Write / Read data ready indicator.
wstrb	O	1	AXI Write data channel. Indicator of which byte lanes of write data contain valid data.
bresp **rresp**	I	2	AXI Write response / Read data channel. Response for transactions in Write response / Read data channel.
bvalid **bready**	I O	1	AXI Write response channel. Write response valid / ready indicator.

Specification (1/3)

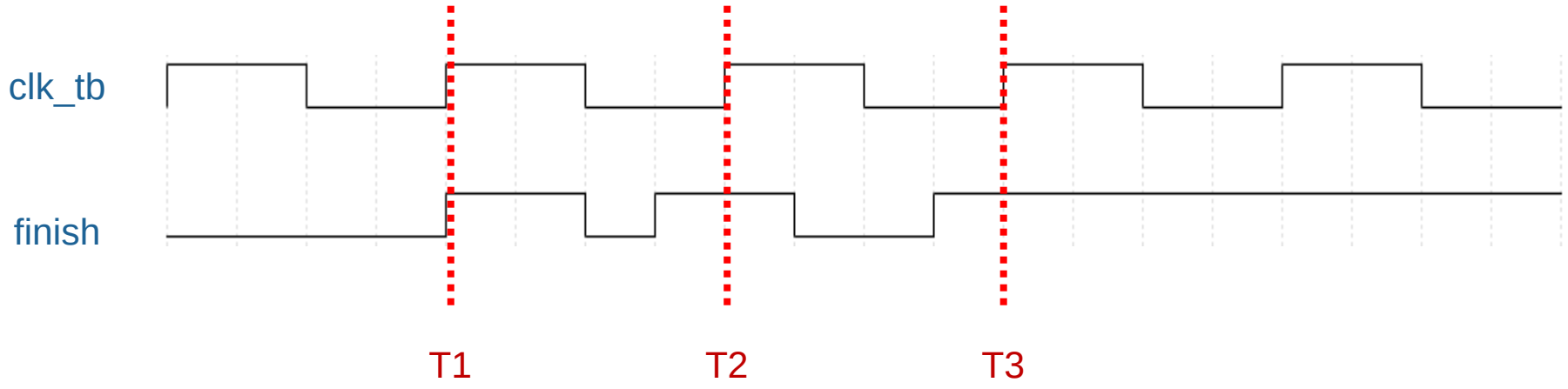
- Active low synchronous reset is synchronized with the **positive** clock edge of **clk_top** and asserted only once
- AXI inputs are synchronized with the **positive** clock edge of **clk_ram**
- AXl outputs should be synchronized with **clk_ram**
- The finish signal should be synchronized with **clk_tb**
- Data is loaded into the external memory once the reset signal is de-asserted
- No start signal in this project, and you can start operations whenever the external memory is ready

Specification (2/3)



Specification (3/3)

- You should finish all computation and store the result back to the external memory before finish is high for the first time (T1)
- The finish signal should be set to high for three times with clk_tb



Convolution

- Convolution spec
  - kernel size = 3×3, stride = 1, padding = 1
- Toy example
  - input image size = 4×4, with binary elements

1	1	0	1
0	0	0	1
1	0	1	0
0	1	1	1

input image

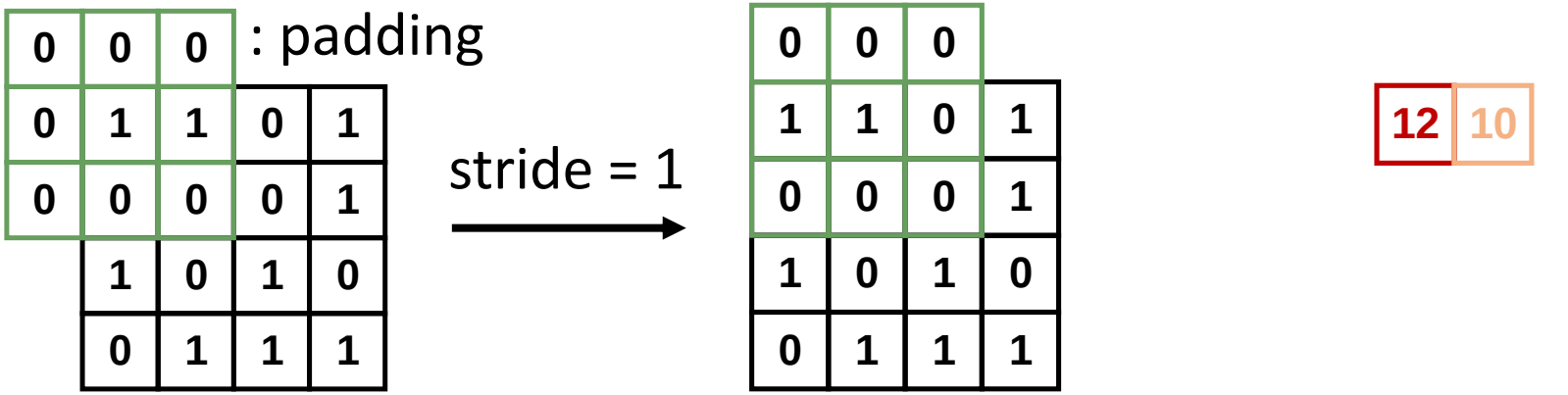
1	2	3
4	5	6
7	8	9

kernel (filter)

1

bias

Convolution

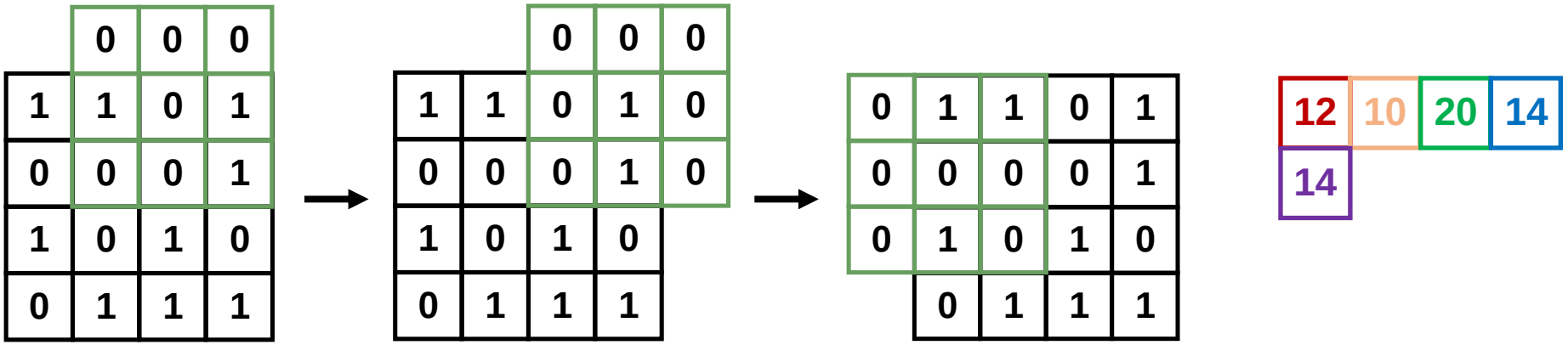


= 1×5 + 1×6 + 1 = 12
(multiply and sum, then add bias)

= 1×4 + 1×5 + 1= 10

output

Convolution



= 4 + 6 + 9 + 1
= 20

= 5 + 8 + 1
= 14

= 2 + 3 + 8 + 1
= 14

output

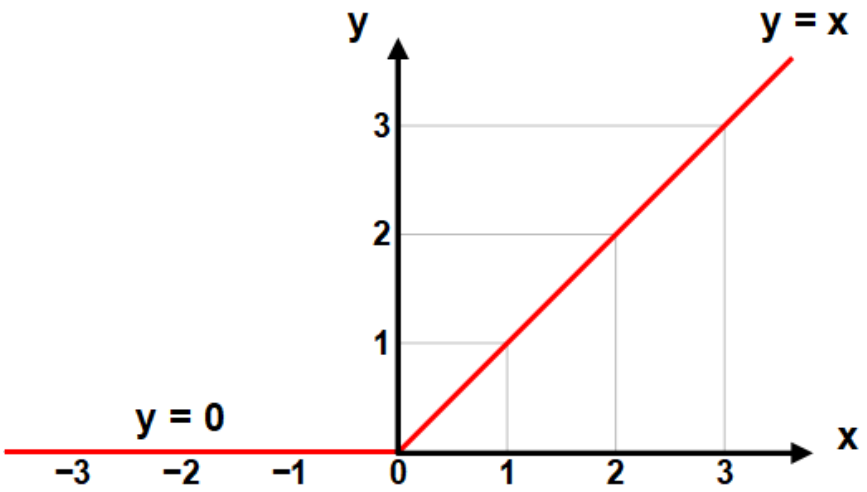
Convolution

- Output has the same size as the input image
- Position of each sum is determined by the center of the kernel
- Multiple kernels can be applied, and there are eight kernels in this project

12	10	20	14
14	20	19	15
15	28	33	22
9	16	18	10

convolution result

ReLU



$$y = \mathrm{ReLU}(x) = \begin{cases} x, x \geq 0 \\ 0, x < 0 \end{cases}$$

5	-3	2	4
-6	-5	-4	0
1	0	-2	-5
6	5	0	-1

convolution result

ReLU →

5	0	2	4
0	0	0	0
1	0	0	0
6	5	0	0

ReLU result

Max Pooling

- 2×2 max pooling divides the convolution result into groups with size 2×2, and output the maximum in each group

12	10	20	14
14	20	19	15
15	28	33	22
9	16	18	10

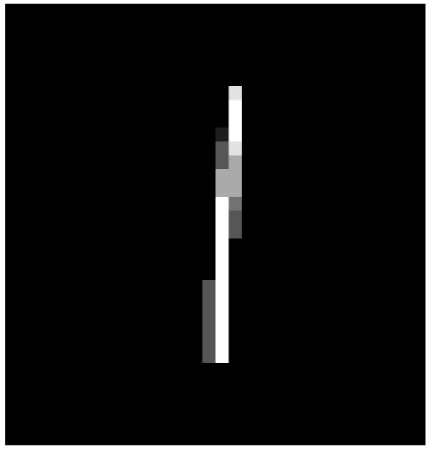
convolution result

20	20
28	33

final output

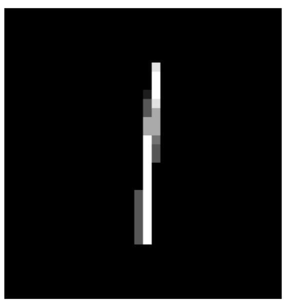
Bitmap Encoding

- Assume zero is the element with highest frequency of occurrence
- Bitmap encoding encodes an image into its all nonzero elements and a bitmap denoting the positions of nonzeros

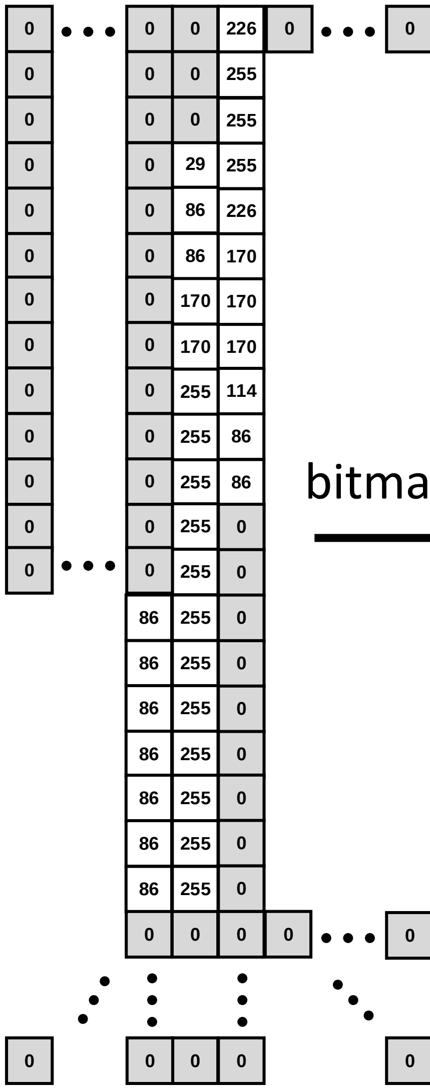


pixel values = 0 ~ 255
black = 0
# of nonzeros = 34
(# of pixels = 1024)

Bitmap Encoding



=



bitmap encoding →

```
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000100000000000000
00000000000000000100000000000000
00000000000000000100000000000000
00000000000000001100000000000000
00000000000000001100000000000000
00000000000000001100000000000000
00000000000000001100000000000000
00000000000000001100000000000000
00000000000000001100000000000000
00000000000000001100000000000000
00000000000000001100000000000000
00000000000000001000000000000000
00000000000000001000000000000000
00000000000000001000000000000000
00000000000000011000000000000000
00000000000000011000000000000000
00000000000000011000000000000000
00000000000000011000000000000000
00000000000000011000000000000000
00000000000000011000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
```

bitmap

```
226
255
255
29
255
86
226
86
170
170
170
170
170
255
114
255
86
255
86
255
255
255
86
255
86
255
86
255
86
255
86
255
86
255
```

nonzeros (raster-scan)

Bitmap Encoding

- Zero may not have the highest frequency, and then the "skipped element" should also be recorded
  - In MNIST dataset, zero must have the highest frequency
  - However, the case may not hold in the output image of convolution

```
1  10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10
2  10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10
3  10,10,10,10,10,10,10,10,60,92,10,10,10,10,10,10
4  10,10,10,10,10,10,10,10,50,188,10,10,10,10,10,10
5  10,10,10,10,10,10,10,10,0,155,10,10,10,10,10,10
6  10,10,10,10,10,10,10,10,0,113,10,10,10,10,10,10
7  10,10,10,10,10,10,10,10,63,107,10,10,10,10,10,10
8  10,10,10,10,10,10,10,10,109,55,10,10,10,10,10,10
9  10,10,10,10,10,10,10,10,111,27,10,10,10,10,10,10
10 10,10,10,10,10,10,10,10,155,10,10,10,10,10,10,10
11 10,10,10,10,10,10,10,0,155,10,10,10,10,10,10,10
12 10,10,10,10,10,10,10,0,155,10,10,10,10,10,10,10
13 10,10,10,10,10,10,10,0,155,10,10,10,10,10,10,10
14 10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10
15 10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10
16 10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10
```

result of convolution #1

=

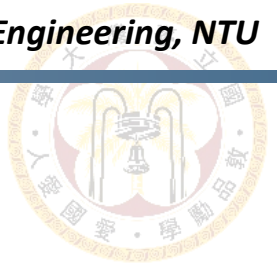
```
0000000000000000
0000000000000000
0000000011000000
0000000011000000
0000000011000000
0000000011000000
0000000011000000
0000000011000000
0000000011000000
0000000010000000
0000000110000000
0000000110000000
0000000110000000
0000000000000000
0000000000000000
0000000000000000
```

bitmap

```
10
60
92
50
188
0
155
0
113
60
107
109
55
111
27
155
0
155
0
155
0
155
```

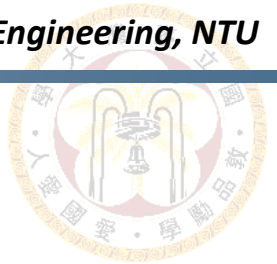
skipped

not skipped



Bitmap Encoding

- Zero may not have the highest frequency, and then the "skipped element" should also be recorded
  - In MNIST dataset, zero must have the highest frequency
  - However, the case may not hold in the output image of convolution
- For simplicity, the "skipped element" is set as the leftmost and uppermost element

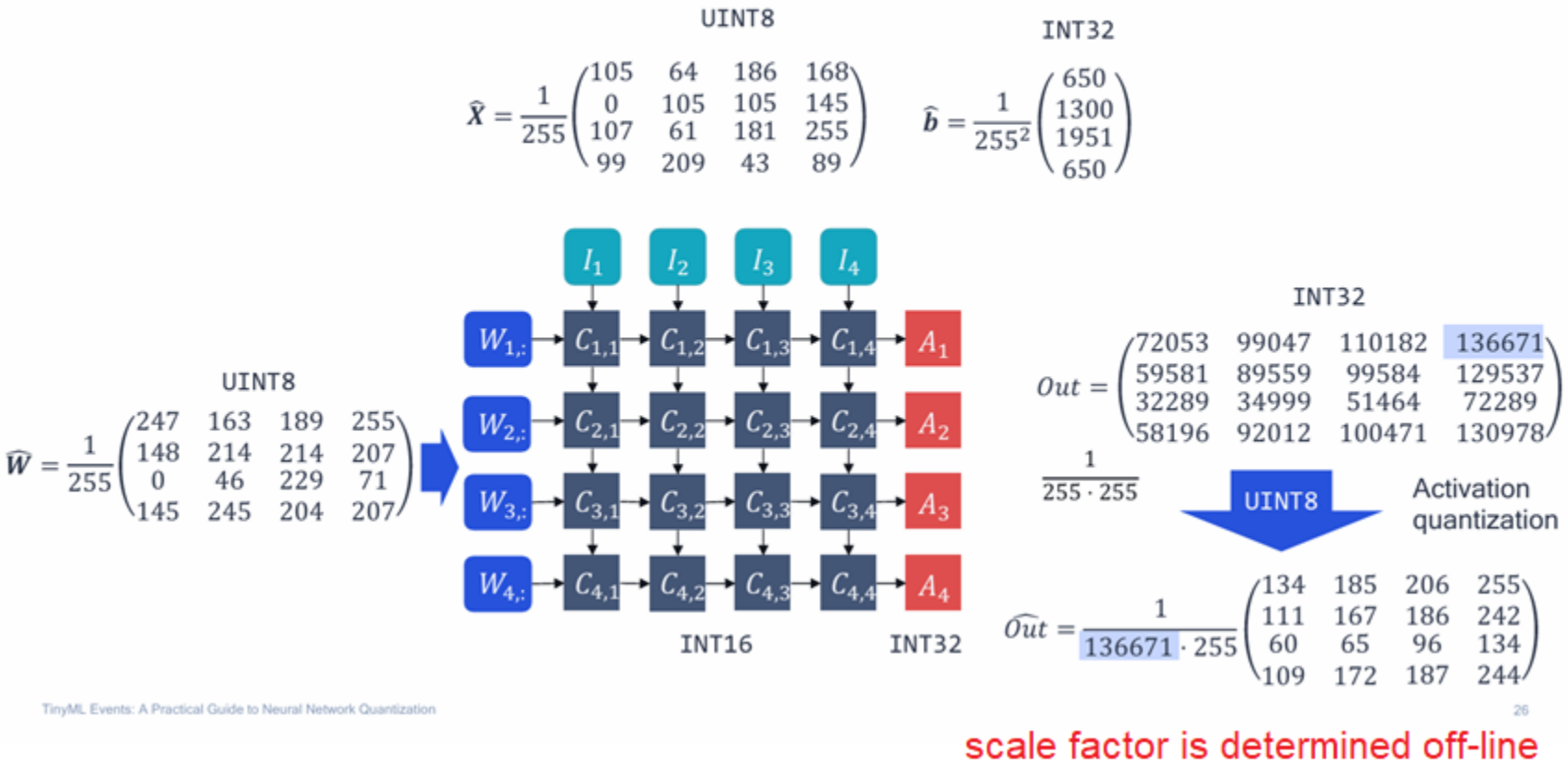


Quantization

- Number of bits affect hardware complexity and power consumption
  - 8-bit integer (INT8) operations have higher energy efficiency than 32-bit floating-point (FP32) operations
- One well-trained model suffers little performance loss when FP32 operations are replaced by INT8 operations

Quantization

Quantized inference using symmetric quantization



Source:
https://cms.tinyml.org/wp-content/uploads/industry-news/tinyML_Talks-_Marios_Fournarakis_210929.pdf

Quantization

- Input data and kernel weights are in INT8 format
- Bias and convolution result are in INT32 format
- Scale factor is applied on the convolution result to make the result in INT8 format
  - Scale factor is determined off-line and selected that the quantization result must fall in the INT8 range
    - INT8 range may be 0 ~ 255 or -128 ~ 127
  - Rounding strategy is **rounding to the nearest integer**

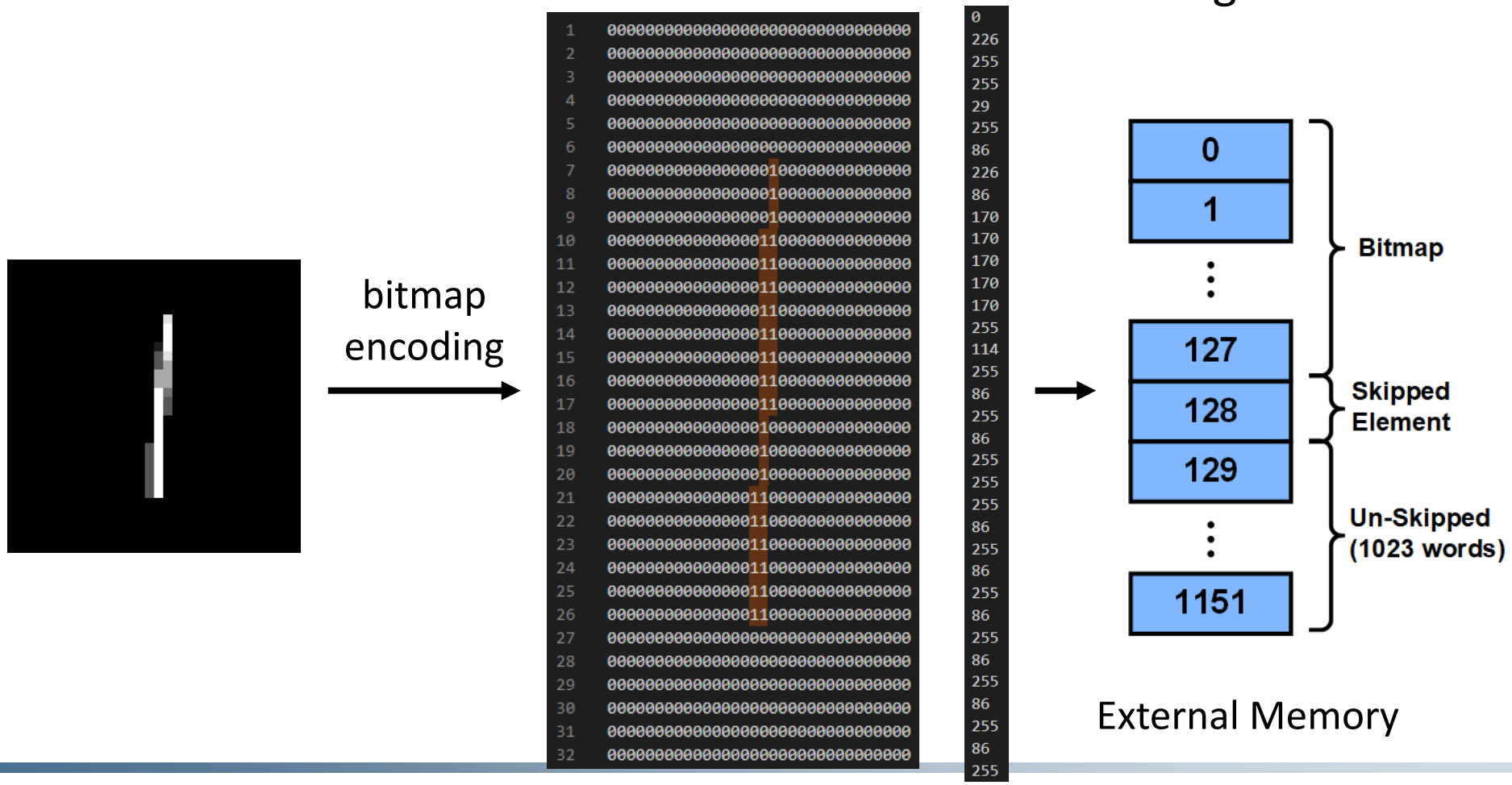
$$x_{\text{INT8}} = \text{round}(\text{scale factor} \cdot x_{\text{INT32}})$$

Design Description

- The image has a fixed size of 32×32 pixels, and each pixel is **unsigned** INT8 (ranging from 0 to 255)
- The image is encoded by bitmap encoding and stored in the external memory
- Eight sets of kernel weights, bias, scale factor are stored in the external memory
- The results after convolution → ReLU → max pooling → quantization → bitmap encoding should be stored back to the external memory
- The word size of the external memory is 8-bit. The memory can store 32768 words.

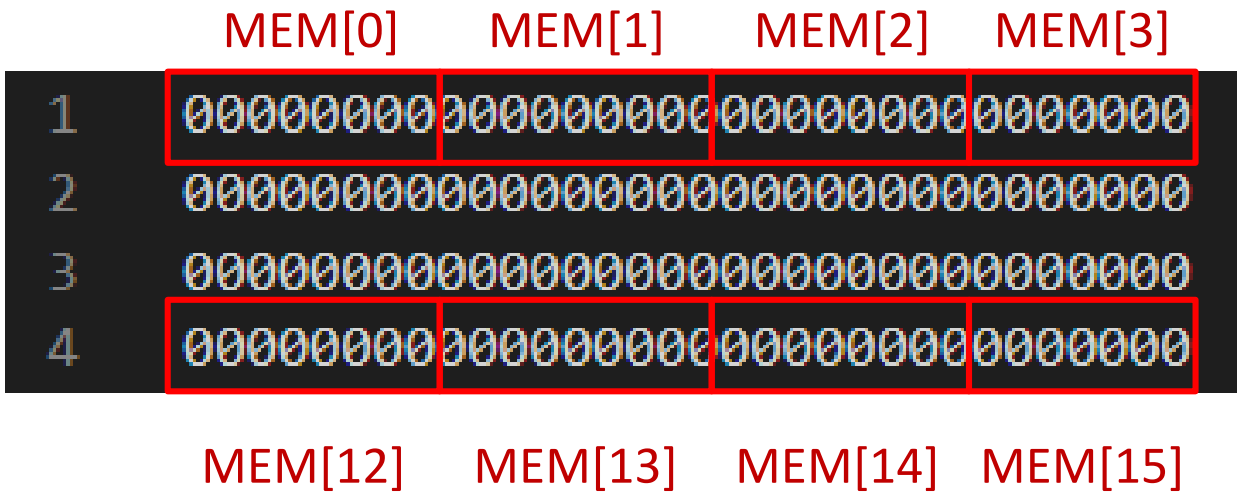
Encoded Input Image

- Bitmap is stored in raster-scan ordering, and un-skipped elements are also determined in raster-scan ordering



Encoded Input Image

- Bitmap is stored in raster-scan ordering, and un-skipped elements are also determined in raster-scan ordering



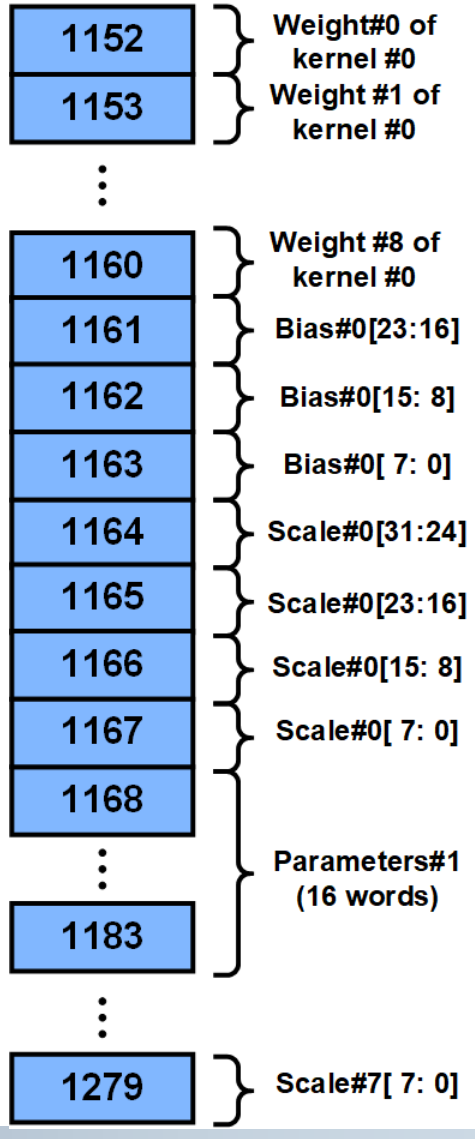
Convolution Parameters

- Kernel weights are **signed** INT8, ranging from -128 to 127
  - Stored in raster-scan ordering
- Biases are **signed** 24-bit integers and stored in the memory from the most significant bit (MSB).
- Scale factors are stored in **unsigned** 32-bit fixed-point format
  - The 32-bit FP has a binary scaling of 2^{-32}; that is, all 32 bits are fraction bits
  - Ex: $0.75 = 0.5 + 0.25 = (0.11)_2$ is stored as 1100…0.

Convolution Parameters

#0	#1	#2
#3	#4	#5
#6	#7	#8

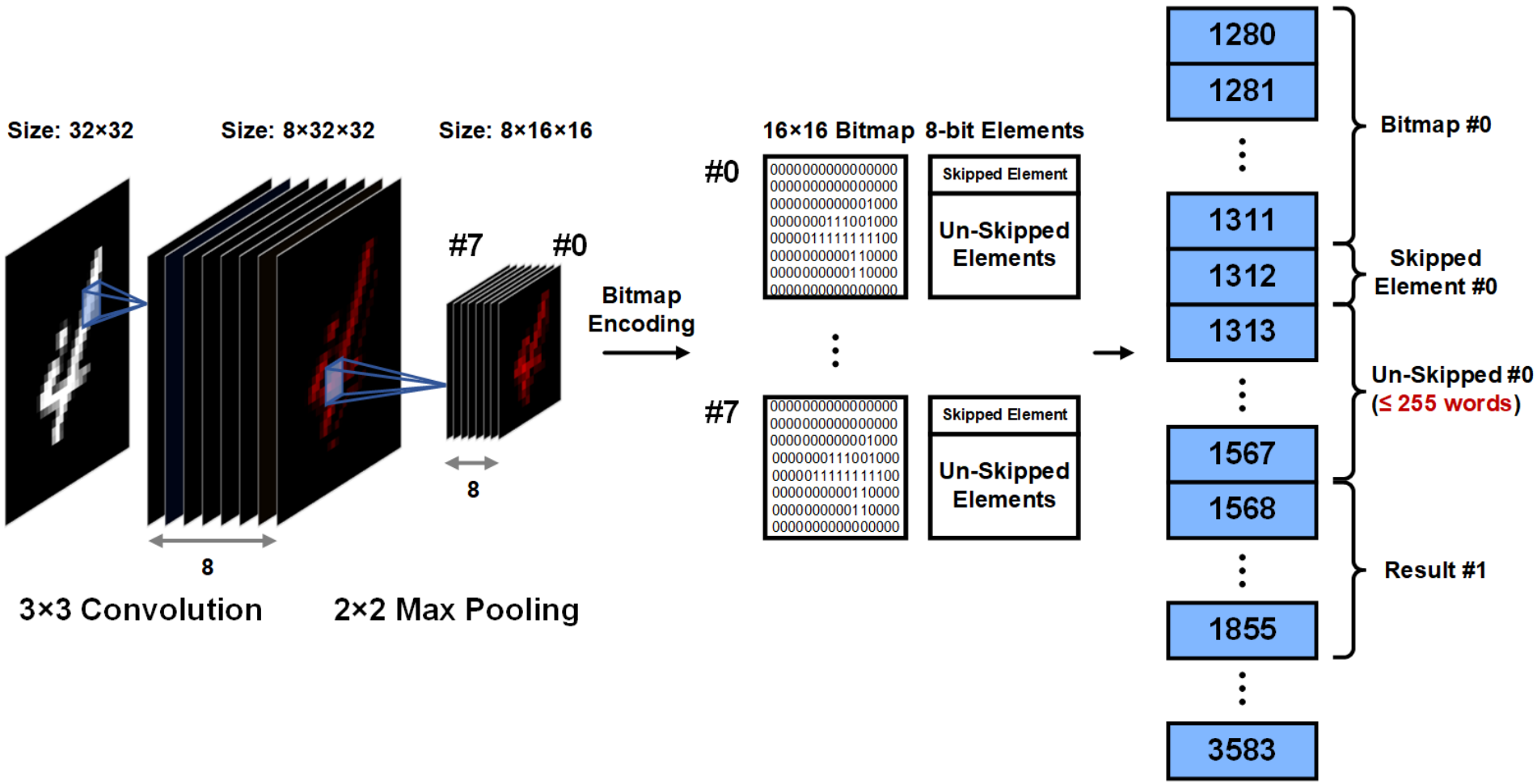
Weight Indexing



Results

- The result of quantization is in **unsigned** INT8 format, ranging from 0 to 255
  - ReLU makes the convolution output must be non-negative
- You should encode the eight output images with bitmap encoding and store them back to the memory
- Store the results from MEM[1280] to MEM[3583], and some memory may be empty due to the sparsity
  - You can store anything in the place where the testbench will not check

Results


Size: 32×32
Size: 8×32×32
Size: 8×16×16
#7
#0
8
8
3×3 Convolution
2×2 Max Pooling
Bitmap Encoding
16×16 Bitmap
8-bit Elements
#0
#7
Skipped Element
Un-Skipped Elements
1280
1281
1311
1312
1313
1567
1568
1855
3583
Bitmap #0
Skipped Element #0
Un-Skipped #0
(≤ 255 words)
Result #1

Submission

- Create a folder named **studentID_midterm (in lowercase)** and follow the hierarchy below

```
r12943008_midterm
        ├─ 01_RTL
        │     ├─ rtl_01.f
        │     ├─ xxx.v  (other Verilog files)
        │     ├─ xxx.sv (other SystemVerilog files)
        │     └─ top.v
        ├─02_SYN
        │     ├─ top_syn.tcl
        │     ├─ top_syn.sdc
        │     ├─ top_syn.timing_min
        │     ├─ top_syn.timing_max
        │     ├─ top_syn.area
        │     └─ top_syn.ddc
        ├─03_GATE
        │     ├─ rtl_03.f
        │     ├─ top_syn.v
        │     └─ top_syn.sdf
        ├─04_UPF (optional)
        │     ├─ top.rtl.upf
        │     └─ top.syn.upf
        ├─05_POWER
        │     └─ p0_2.power
        └─ report.txt
```

Submission - Workstation

- Pack the folder **studentID_midterm** into a **tar.gz file** named **acvsdxxx_midterm.tar.gz**
  - `tar -zcvf acvsdxxx_midterm.tar.gz studentID_midterm`
  - All letters are in lowercase. (e.g. acvsd000_midterm.tar.gz)
  - Pack the folder on ADFP server to avoid OS-related problems
  - Place the tar.gz file at the root of your ADFP account

- TA will only check the latest version

Submission - NTU COOL

- Pack the folder **studentID_midterm** into a **tar.gz file** named **acvsdxxx_midterm_v*k*.tar.gz** (*k* is the version, *k* =1,2,…)
  - **`tar -zcvf acvsdxxx_midterm_vk.tar.gz studentID_midterm`**
  - All letters are in **lowercase**. (e.g. acvsd000_midterm_v1.tar)
  - Pack the folder on IC Design LAB server
- Submit NTU COOL

```
r12943008_midterm
        ├─ 01_RTL
        │     ├─ rtl_01.f
        │     ├─ xxx.v  (other Verilog files)
        │     ├─ xxx.sv (other SystemVerilog files)
        │     └─ top.v
        ├─02_SYN
        │     ├─ top_syn.tcl
        │     └─ top_syn.sdc
        ├─04_UPF (optional)
        │     ├─ top.rtl.upf
        │     └─ top.syn.upf
        └─ report.txt
```

Grading Policy (1/4)

- Simulation:

	Score
RTL simulation	30%
Gate-level simulation	30%
Hidden patterns (gate-level)	10%

- Performance: $\text{Score} = (\sum_{i=0}^{2} \text{power}_i \times \text{time}_i) \times \text{area}$

Unit: power (mW), time (ns), area (um^2)

Baseline: 10^{10}

	Score
Baseline (need passing hidden patterns)	10%
Ranking (need passing baseline)	20%

Grading Policy (2/4)

- Grading command for RTL
  - `vcs -full64 -R -f rtl_01.f +v2k -sverilog -v2005 -debug_access+all +notimingcheck +define+pi`
  - *i* = 0, 1, 2, 3, 4, 5. *i* = 0, 1, 2 are public patterns and *i* = 3, 4, 5 are hidden patterns
- Simulation: **70%**
  - RTL simulation for public patterns: 30%
  - Gate-level simulation for public patterns: 30%
  - Gate-level simulation for hidden patterns: 10%

	Score
RTL simulation	30%
Gate-level simulation	30%
Hidden patterns (gate-level)	10%

Grading Policy (3/4)

- Grading command for gate-level
  - `vcs -full64 -R -f rtl_03.f +v2k -sverilog -v2005 -debug_access+all +maxdelays -negdelay +neg_tchk +define+SDF+pi`
- Performance: **30%**
  - You need to pass all hidden patterns to compare performance
  - $\text{Score} = (\sum_{i=0}^{2} \text{power}_i \times \text{time}_i) \times \text{area}$
    - Unit: power (mW), time (ns), area (um^2)
  - Baseline = 10^{10}

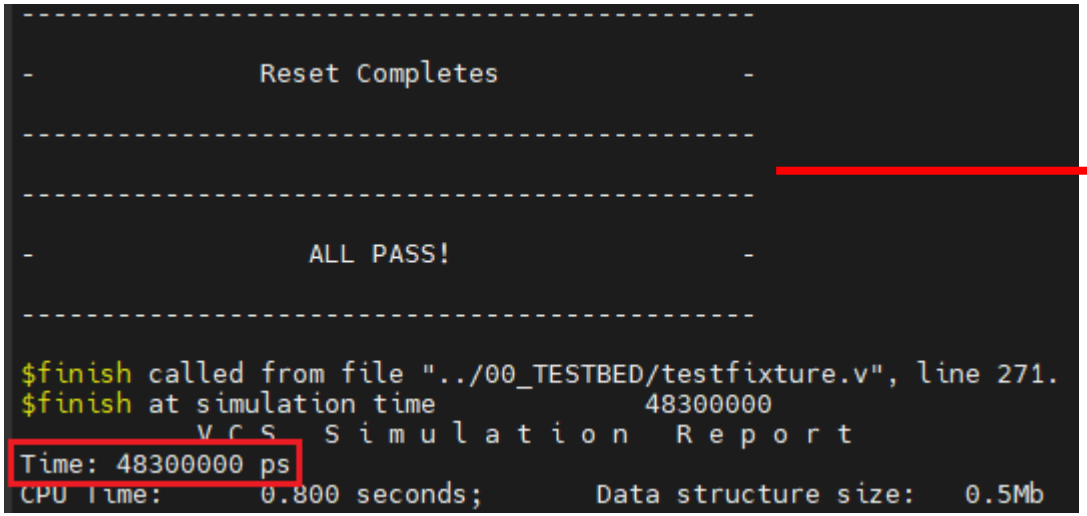
	Score
Baseline (need passing hidden patterns)	10%
Ranking (need passing baseline)	20%

Grading Policy (4/4)

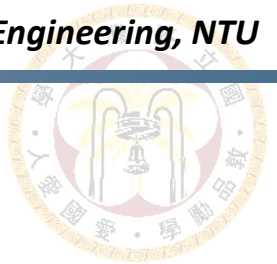
- 5 points minus for any incorrect naming or submission format
- DesignWare and SRAM are allowed
- Negative slack causes 0 point for gate-level simulation and performance
- Performance score $\times$ 0.7 if passing all simulations but violating other rules
- UPF files are optional, and TA will run simulation with UPF if submitted (you may perform better in ranking)
- No late submission, or **0 point** for this homework
- No plagiarism
  - Plagiarism in any form, including copying from online sources, is strictly prohibited

Time

- Time: Processing time from simulation (ex. 48300 ns below)



There should be no timing violations after reset is de-asserted



Area

- Area: total cell area (ex. 29206.72 um^2 below)

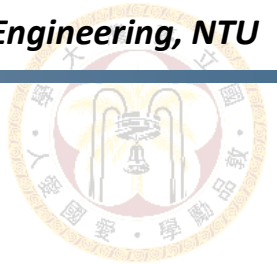
```
Number of ports:
Number of nets:
Number of cells:
Number of combinational cells:
Number of sequential cells:
Number of macros/black boxes:
Number of buf/inv:
Number of references:

Combinational area:              17263.135097
Buf/Inv area:                     3703.605144
Noncombinational area:            6991.660891
Macro/Black Box area:             4951.919189
Net Interconnect area:      undefined  (Wire load has zero net area)

Total cell area:                 29206.715178
Total area:                 undefined
```

Power

- Power: Use Primetime to calculate power (1.668 mW below)

Power Group	Internal Power	Switching Power	Leakage Power	Total Power	(%)	Attrs
clock_network	0.0000	0.0000	0.0000	0.0000	(0.00%)	i
register	0.0000	0.0000	0.0000	0.0000	(0.00%)	
combinational	9.278e-05	9.930e-05	5.234e-06	1.973e-04	(11.83%)	
sequential	8.143e-04	1.671e-05	4.455e-06	8.355e-04	(50.09%)	
memory	6.343e-04	4.868e-09	7.515e-07	6.350e-04	(38.08%)	
io_pad	0.0000	0.0000	0.0000	0.0000	(0.00%)	
black_box	0.0000	0.0000	0.0000	0.0000	(0.00%)	

Net Switching Power	= 1.160e-04	(6.96%)
Cell Internal Power	= 1.541e-03	(92.42%)
Cell Leakage Power	= 1.044e-05	(0.63%)
Total Power	= 1.668e-03	(100.00%)
X Transition Power	= 2.031e-06	
Glitching Power	= 0.0000	
Peak Power	= 0.6519	
Peak Time	= 60.055	

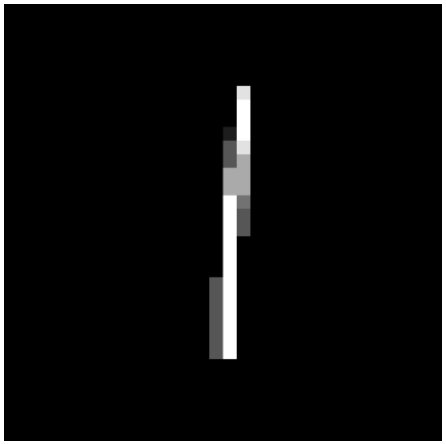


Report

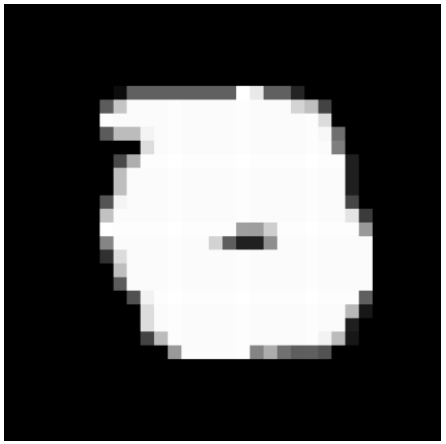
```
StudentID: r12943008
Clock period: 5.0 (ns)
Area: 50000.00 (um^2)
Is UPF used? No

-----------------------
p0 time: 50000 (ns)
p0 Power: 5 (mW)
-----------------------
p1 time: 50000 (ns)
p1 Power: 5 (mW)
-----------------------
p2 time: 50000 (ns)
p2 Power: 5 (mW)
```

Public Patterns

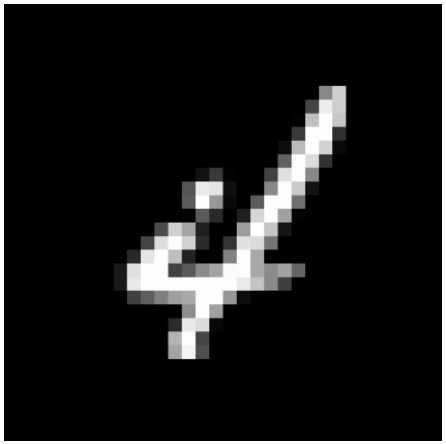
- Convolution parameters are the same in these three public patterns



Best Sparsity



Worst Sparsity



Random

Private Patterns

- Input images and convolution parameters are different from the public patterns
  - You cannot hard-coded the parameters in your design
- Input images also come from MNIST datasets
- Biases and scale factors require the same bits as the public one
  - Not all bits in bias[23:0] and scale[31:0] are useful

Hint

- Images in MNIST dataset are of size 28×28, and they are zero-padded to be 32×32
- The skipped element of the input image must be zero
- Input image must not have 1023 nonzeros, and hence the remaining memory without nonzeros stores 8'b0
- You do not really need to implement a 24-bit addition for adding biases and a 32-bit multiplication for scaling
- Perform Clock Domain Crossing techniques at the topmost module, and then sub-modules in top.v can be designed in single clock domain
- FIFO can be found in DesignWare, or you can design by yourself

Discussion

- **NTU COOL Discussion Forum**
  - For any questions without assignment answers or privacy concerns, please use the NTU COOL discussion forum.
  - **TAs will prioritize answering questions on the NTU COOL discussion forum**

- **Email:** r12943008@ntu.edu.tw
  - Title should start with **[ACVSD 2025 Spring Midterm]**
  - Email with wrong title will be moved to trash automatically

References

- [1] Reference for MNIST Datasets
  - https://git-disl.github.io/GTDLBench/datasets/mnist_datasets/
- [2] Reference for Convolution Layer
  - 【機器學習2021】卷積神經網路 (Convolutional Neural Networks, CNN)
  - CVSD HW3
- [3] Reference for Bitmap Encoding
  - C.-W. Chang *et al.*, "A 101mW, 280fps Scene Graph Generation Processor for Visual Context Understanding on Mobile Devices", *2024 IEEE Symposium on VLSI Technology and Circuits*, 2024.
- [4] Reference for Quantization
  - https://cms.tinyml.org/wp-content/uploads/industry-news/tinyML_Talks-_Marios_Fournarakis_210929.pdf
- [5] Reference for AXI Protocol
  - AMBA AHB Protocol Specification