Integrated Circuit Design
Final Project
Due: 14:20, June 9, 2025
*No late project is allowed.

1 Problem Description

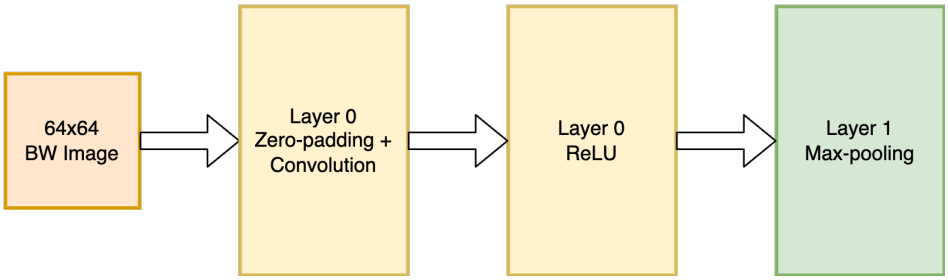

Figure 1: The flow for this project.

In this project, you need to implement a design to calculate the convolution, ReLU, and max-pooling for the image stored in the testbench. Convolution is a fundamental technique in image processing, often used to extract meaningful patterns from visual data. In the convolutional layer of a neural network, a small matrix called a *kernel* (or *filter*) moves across the image. It performs a combination of multiplication and addition at each position. This operation emphasizes important local features within the image, such as edges, corners, and textures. Using convolution, we can significantly reduce the number of learnable parameters while preserving essential spatial structures, making it efficient and effective for image-related tasks.

In this project, you are required to implement a hardware design for an Image Convolutional Circuit, referred to as the CONV circuit. The input to the CONV circuit is a 64 × 64 single-channel grayscale image. The design must include the following two processing layers, executed sequentially:

Layer 0 – Zero-padding + Convolution + ReLU

The input image is 64 (width) × 64 (height) pixels in size. Before performing convolution, zero-padding must be applied to ensure that the output size of the image remains the same. You will be provided with two different kernel sizes: 3×3 and 5×5 (dilated 3×3) as shown in Figure 3. For a 3×3 kernel, pad the input with 1 row (or 1 column) on each side. For a 5×5 kernel, pad the input with 2 rows (or 2 columns) on each side. After padding, convolution is performed using the given kernel. After convolution, you should add a bias to the result. After that, the output must go through a ReLU activation function. The resulting output image will still be of size 64×64. This output should be stored in pseudo memory and will be checked by the testbench.

The pixel values of the image range from 0 to 255 and will be stored as 32-bit fixed-point numbers (1-bit signed bit, 15-bit integer part, 16-bit fractional part) in the pseudo memory in the testbench. In this project, the kernel and bias are fixed, and the values will be given. The kernel used in this project is shown in Figure 3 and the bias is set to 0.07446326 in decimal (0x00001310, fixed point, 1 signed bit, 15 integer bits, 16 fractional bits).

The ReLU (Rectified Linear Unit) function used in this project is defined as follows: If the input pixel value x is less than or equal to 0, the output value y is set to 0. If the input pixel value x is greater than 0, the output remains x. Every pixel resulting from the convolution operation (after bias calculation) must pass through the ReLU function, and the output after ReLU becomes the final result of Layer 0.

$$y = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{if } x \leq 0 \end{cases}$$

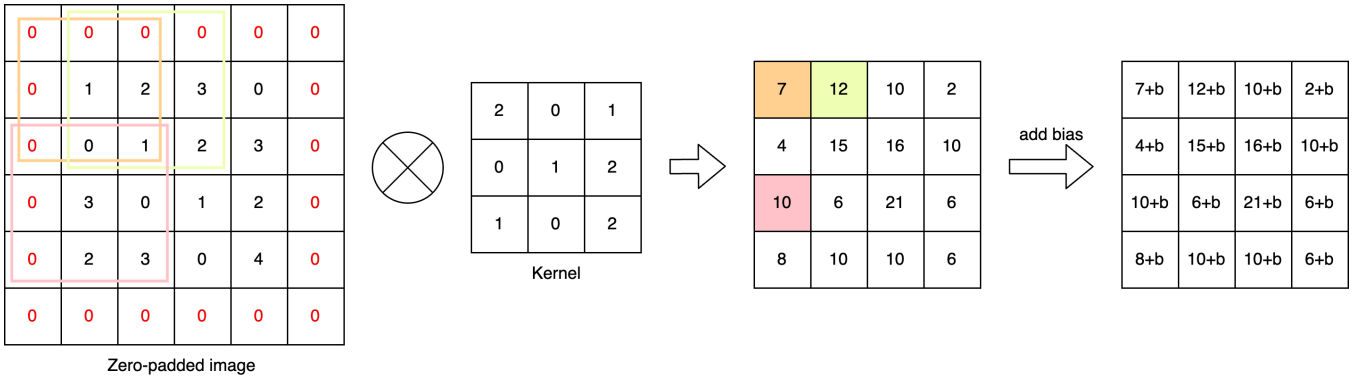

Figure 2: Example of zero-padding, 3x3 convolution, and bias calculation.

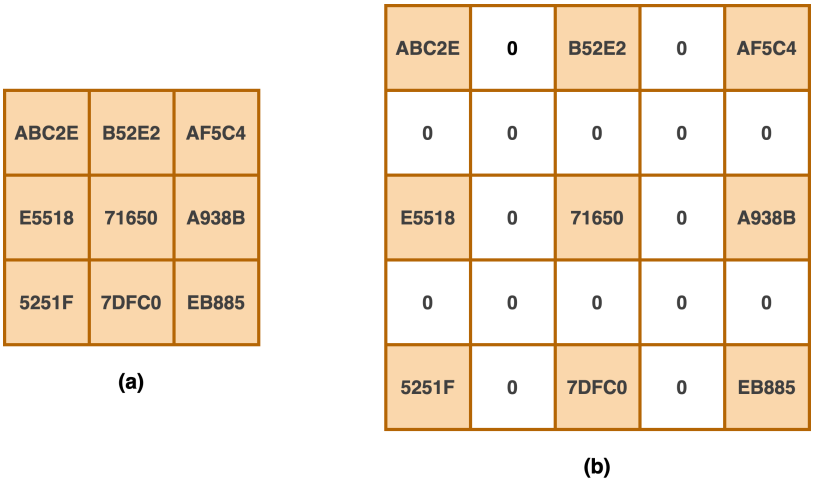

Figure 3: The kernel for this project. The weight is a 20-bit fixed-point number (1 signed bit, 3 integer bits, and 16 fractional bits). (a) A 3×3 kernel. (b) A dilated 3×3 kernel.

Layer 1 – Max-Pooling

After ReLU computation, max-pooling will then be performed on the result, which will modify the size of the final image. The output from Layer 0 will be passed into a 2×2 max-pooling layer with a stride of 2. The result of this operation will be an image of a size 32×32. This image should also be stored in pseudo memory and will be checked by the testbench.

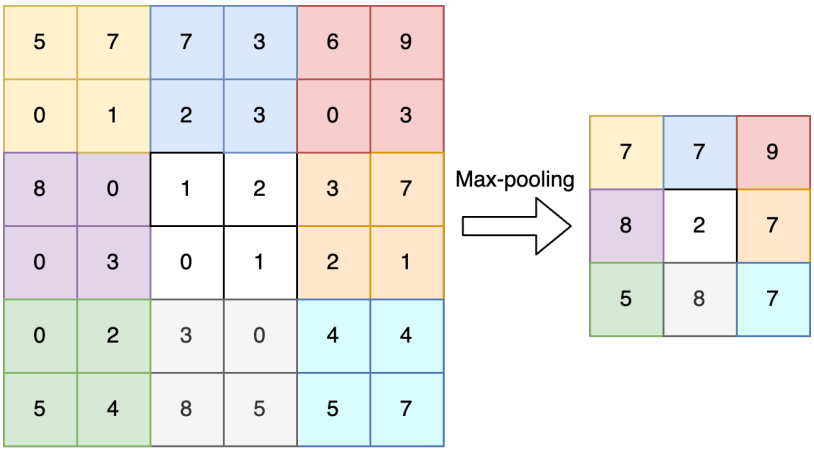

Figure 4: An example of Max-Pooling operation.

Pseudo Memory Allocation

In this project, the input and output images must be stored in the pseudo memory instantiated within the testbench. The pseudo memory is divided into three parts:

- **Memory 0:** Stores the input image, with a size of 64×64.
- **Memory 1:** Stores the output image after Layer 0 calculation, with a size of 64×64.
- **Memory 2:** Stores the output image after Layer 1 calculation, with a size of 32×32.

The testbench will automatically verify the contents of the two output memories when *out_valid* is high.

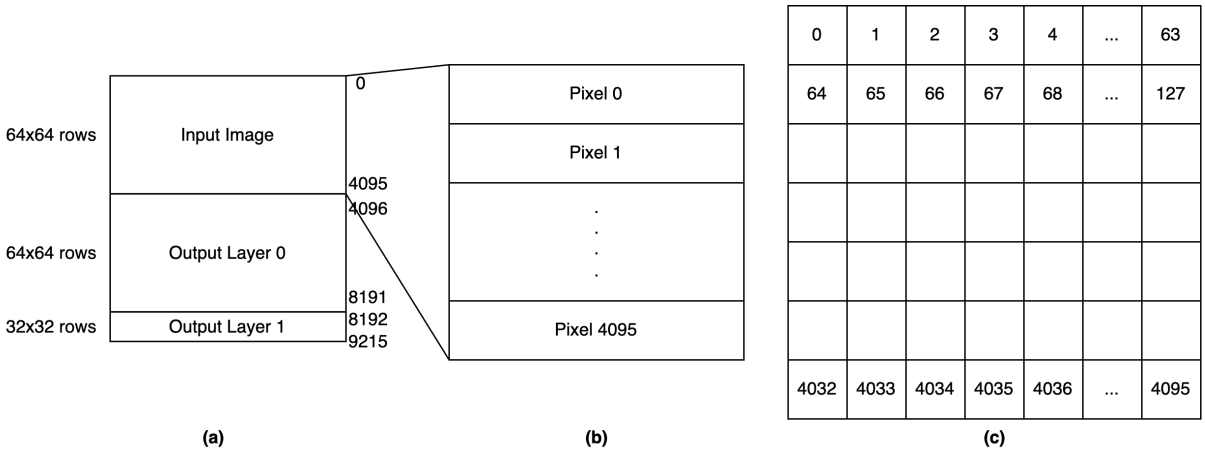

Figure 5: (a) The memory allocation for this project. (b) Detailed pixel index for each memory part. (c) Pixel index under the raster-scan order.

2 I/O Specifications

Signals	I/O	Bit Width	Description
clk	I	1	Positive-edge triggered clock
rst_n	I	1	Asynchronous negative-edge reset
in_valid	I	1	Indicate input figure is valid, active high
mode	I	1	0: normal 3×3 kernel, 1: dilated 3×3 kernel
out_valid	O	1	Indicate the output is valid, active high
AWADDR	O	14	Write address
AWLEN	O	8	Write burst length (The total data transfer will be AWLEN+1)
AWVALID	O	1	Write address valid
AWREADY	I	1	Write address ready
WDATA	O	32	Write data
WVALID	O	1	Write data valid
WREADY	I	1	Write data ready
ARADDR	O	14	Read address
ARLEN	O	8	Read burst length (The total data transfer will be AWLEN+1)
ARVALID	O	1	Read address valid
ARREADY	I	1	Read address ready
RDATA	I	32	Read data (fixed point, 1 signed bit, 15 integer bits, 16 fractional bits)
RVALID	I	1	Read data valid
RREADY	O	1	Read data ready

3 Waveform

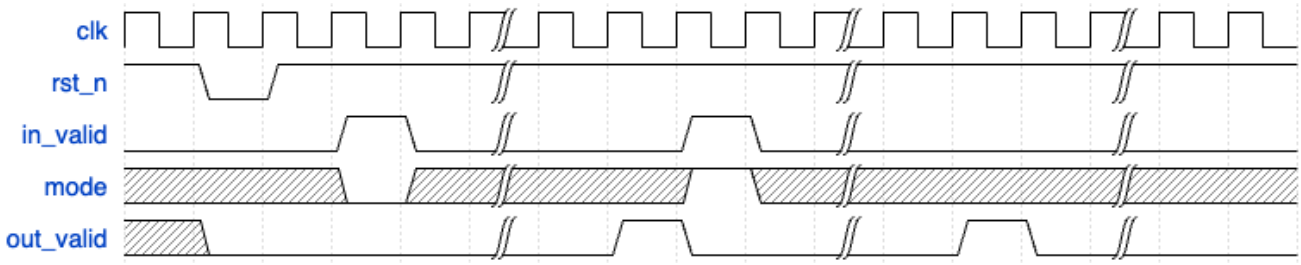

Figure 6: The waveforms of I/O signals.

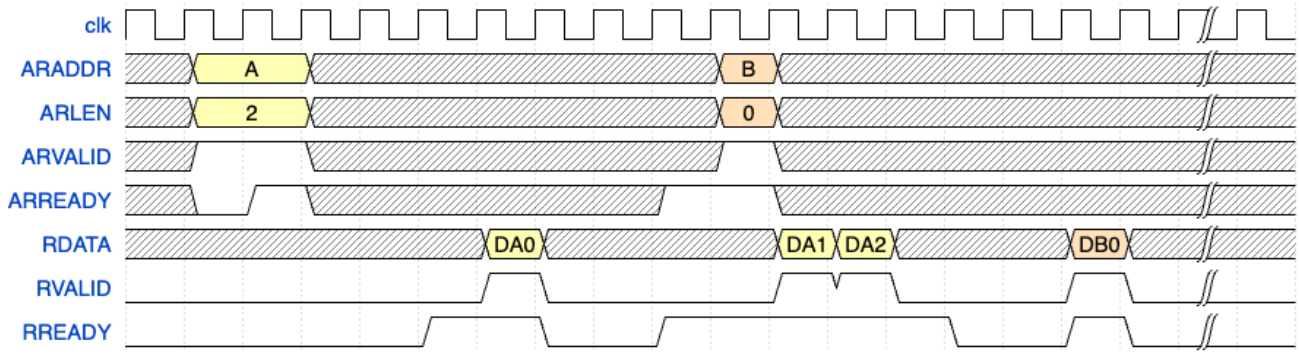

Figure 7: The waveforms of read I/O signals. ARADDR, ARLEN, ARVALID, and RREADY are controlled by your design.

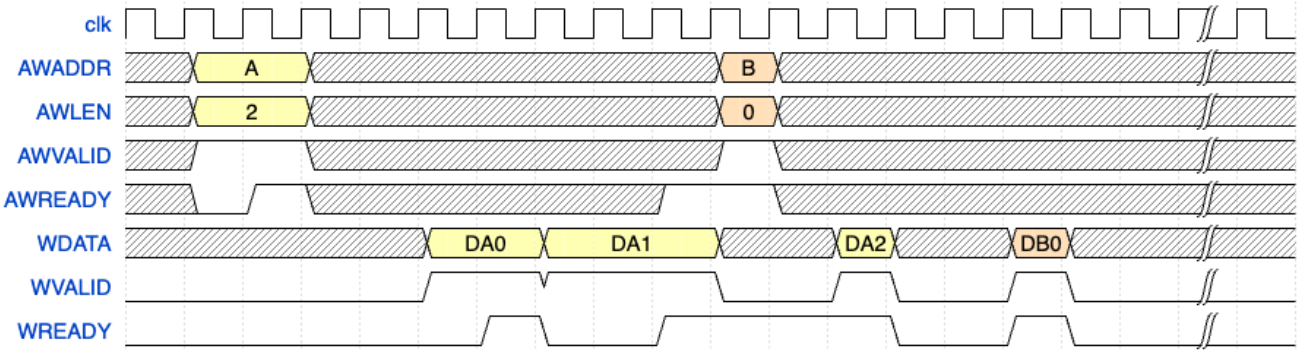

Figure 8: The waveforms of write I/O signals. AWADDR, AWLEN, AWVALID, WDATA, and WVALID are controlled by your design.

4 Files

Figure 9 shows the files you will be able to download from NTU Cool.

- tb_CONV.v: The testbench for this homework.
- ram.v: The pseudo RAM used in this project.
- pattern: The patterns used in this project.
- CONV.v: The design you need to work with.
- rtl.f: The files you need to use in RTL simulation.
- .synopsys_dc.setup: The setup file for Design Compiler.
- CONV.sdc: The design constraints.
- syn.tcl: The tcl file for synthesis.
- gate.f: The files needed for gate-level simulation.
- mmmc.view: The MMMC setting for APR.
- lab_script: The scripts for APR. (Copy to 04_APR from /nfshome/vlsi-ta2/ICD_final/)
- post.f: The files needed for post-route simulation.

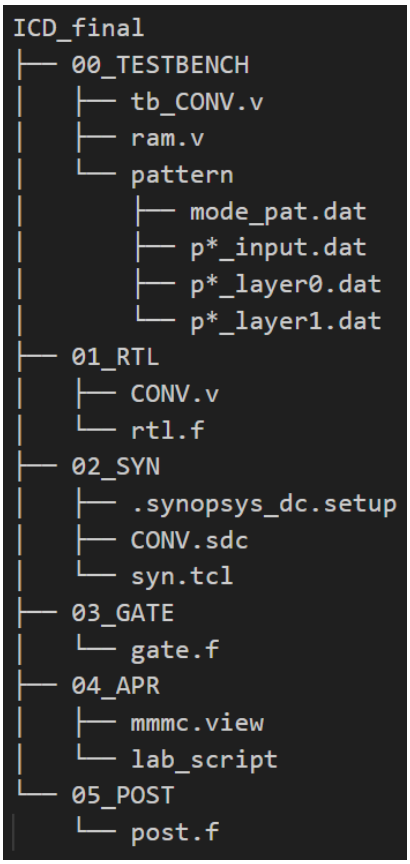

Figure 9: The files you will get.

5 Simulation

- Use the following code to run RTL simulations:

```
$ vcs -f rtl.f -full64 -R -debug_access+all +define+pat0
```

- Use the following code to run gate-level simulations:

```
$ vcs -f gate.f -full64 -R +maxdelays -negdelay +neg_tchk -debug_access+all +define+pat0+SYN
```

- Use the following code to run post-route simulations:

```
$ vcs -f post.f -full64 -R +maxdelays -negdelay +neg_tchk -debug_access+all +define+pat0+APR
```

- Change pat0 to pat1, pat2, ... to test other public patterns.

6 Synthesis

Use the following command to run synthesis.

```
$ dc_shell -f syn.tcl
```

The command needs to be run under the 02_SYN directory. Make sure there are no errors, no latch, and the timing is met.

7 Place and Route

Use the following command to open Innovus and follow the flow taught in class to finish the design.

```
$ innovus
```

Run the command under the 04_APR directory.

8 Rules

8.1 What you can do

- Explain the concept of this final project to others
- Discuss possible algorithms with others
- Search online for algorithms

8.2 What you cannot do

- Use brute force to solve the problem (directly output golden patterns)
- Use GPT to generate your code
- Share code with others
- Copy or read others' code
- Copy or read online code
- Help your classmate to debug his/her code

We will use a tool to check if there is any plagiarism. If the code fails the test, the team will receive no points for the final project.

9 Grading Policy

- (40%) Pass RTL simulations. Each pattern is worth 4 points. There are ten public patterns.
- (10%) Pass gate-level simulations. Need to pass all ten public patterns to receive the credit.
- (10%) Pass post-route simulations. Need to pass all ten public patterns to receive the credit.
- (10%) Written Report
  - Snapshot your clock tree plot in CCopt clock tree debugger.
  - Snapshot your APR result.
  - List the core area and the cycle time.
  - Describe how you implement your design.
    (If you pass all the public patterns in post-route simulation, you can skip the third part.)
- (30%) If your team passes all ten public and ten hidden patterns in post-route simulations, you will enter a contest, which is ranked by:

$$Performance = Time \times Layout\_Area.$$

The smaller value, the better. The best team will receive 30 points. For those who do not make it to the first contest but pass all ten public and ten hidden patterns in gate-level simulations, you will enter a second contest, which is similar to the first one but using the area reported by Design Compiler instead. If there are k teams in the first contest, the top position for the team in the second test will be $k+1$. So the j^{th} position in the second contest will have the final rank $j+k$.

Final Rank (FR) 1: 30, FR 2: 28, FR 3: 26, FR 4: 24, FR 5: 22, FR 6: 20, and FR (6+i):(20-i) for $i \geq 1$.

10 Submission

- Upload <student_id>_report.pdf to NTU COOL.
- Put <student_id>_final.tar to /nfshome/vlsi<your id>/ICD2025_submission
  - A wrong folder format will get -10 points as a penalty
  - Compress your files with the command below (replaced with your ID):

    $ tar -cvf b13901000_final.tar b13901000_final/

```
[vlsi073@vlsi-13 ~/ICD2025_submission]$ pwd
/nfshome/vlsi073/ICD2025_submission
[vlsi073@vlsi-13 ~/ICD2025_submission]$ ls
b13901000_final.tar
```

Figure 10: Make sure that you place the tar file under the correct directory.

```
b13901000_final.tar
└── b13901000_final
    ├── CONV.v
    ├── CONV_syn.v
    ├── CONV_syn.sdf
    ├── CONV_syn.ddc
    ├── CONV_syn.max.timing
    ├── CONV_APR.v
    ├── CONV_APR.sdf
    ├── CONV.gds
    ├── finish.enc
    └── finish.enc.dat
```

Figure 11: An example of the file/folder format.

TA

H.-Y. Tseng r12943119@ntu.edu.tw
P.-S. Yen r12943009@ntu.edu.tw