NCTU-EE IC LAB – Spring 2018

Online Test

Design: 2-Dimensional Convolution

Data Preparation

1. Extract test data from TA's directory:

% tar xvf ~iclabta01/OT.tar

2. The extracted LAB directory contains:
   a. EXERCISE/: your design

Design Description

In this test, we will be implementing a 2D-convolution between a 5-by-5 image and a 3-by-3 Gaussian Filter. An *example* of convolution is shown in the image below:

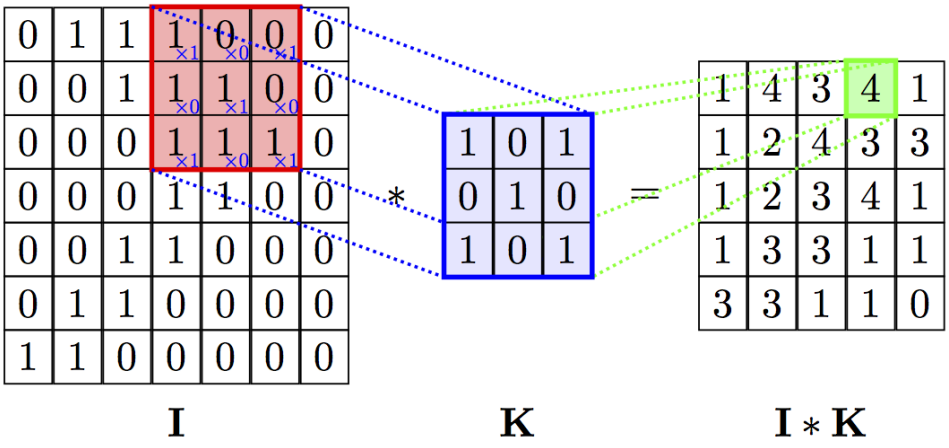


Figure 1: Example of 2D Convolution

https://camo.githubusercontent.com/

As we can see from the example above, the resulting output is always smaller than the original image. In some image processing application, we hope that we are able to get an output that has the same size as the input. A workaround for this problem is by using zero padding. Zero padding is done by adding 0's at the border of the input image before performing convolution. For the case of convolving the image with a 3x3 kernel, a padding of 1 pixel is sufficient. An example is shown below:

Figure 2: An example of zero padding

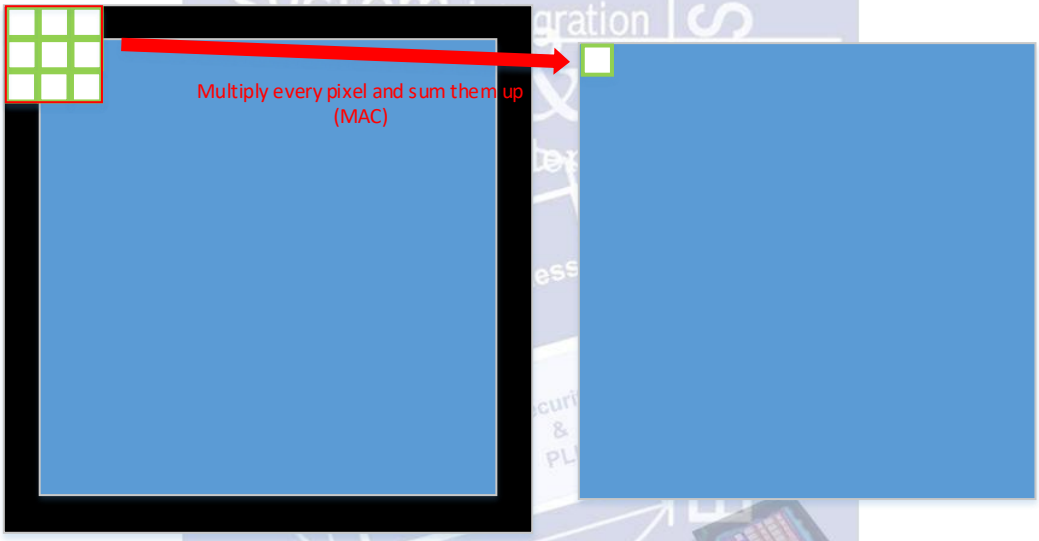


Figure 3: Convolution on a zero-padded image

As mentioned above, we pass our image into a Gaussian filter as shown below:

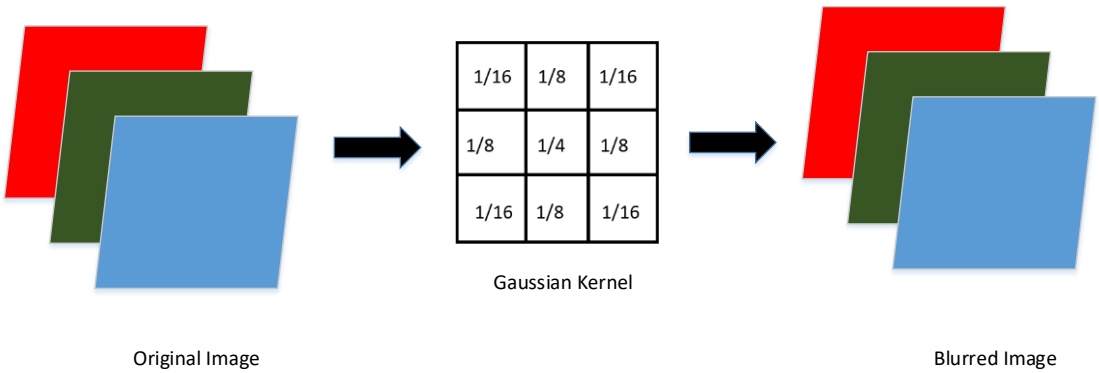


Figure 4: Gaussian Filter

As we can see from Figure 4, the input image will have 3 channels (red, green, blue). You can think of all 3 channels are independent images fed into the filter. The resulting output will also be a “Gaussian blurred image” with 3 channels.

If you notice the parameters of the filter, it’s representing by fractions, which also means that you’ll need a way to represent a number with decimals. One way to do that is by using a fixed-point representation as shown in the figure below.

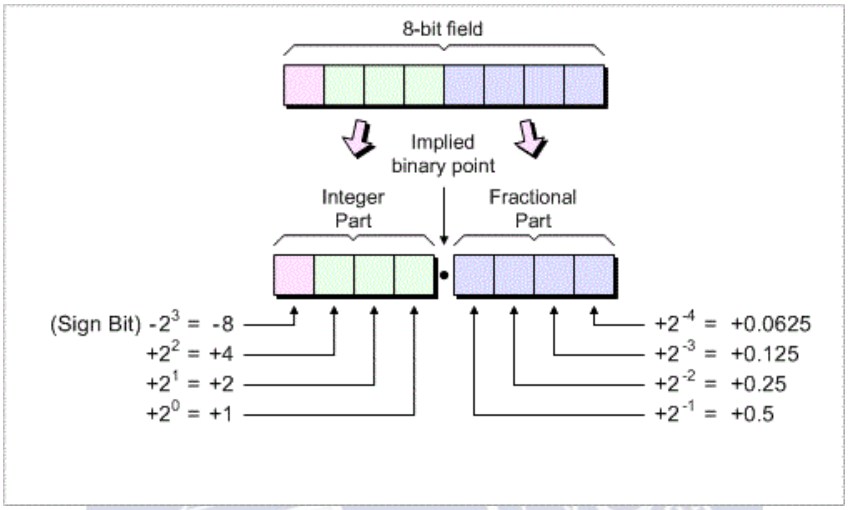


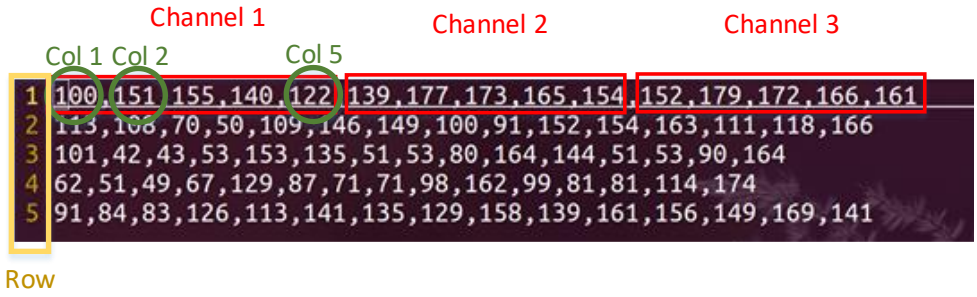
Figure 5: Fixed-point representation

https://www.allaboutcircuits.com/uploads/thumbnails/round-26.gif

Since we are only dealing with unsigned numbers, you can leave out the sign bit. In this test, we require that the first 4 bits (starting from LSB) will be fractional part and the rest will be integer part.

Example

A pattern is prepared for you in this test, you just need to worry about coming up with a good design. Your pattern takes the file below as input and it’s a representation of an image that you will be using.



The image is fed into your design in accord with the file format shown above. A brief description is as follows:

1. Only one pixel will be fed into your design per cycle.
2. Column 1 ~ 5 of row 1 of channel 1 will be fed.
3. `` 2 ``
4. `` 3 ``
5. Column 1 ~ 5 of row 2 of channel 1 will be fed.
6. `` 2 ``
7. `` 3 ``

   .
   .
   .

8. Column 1 ~ 5 of row 5 of channel 1 will be fed.
9. `` 2 ``
10. `` 3 ``

The output from your design will also follow the same order.

Input and Output

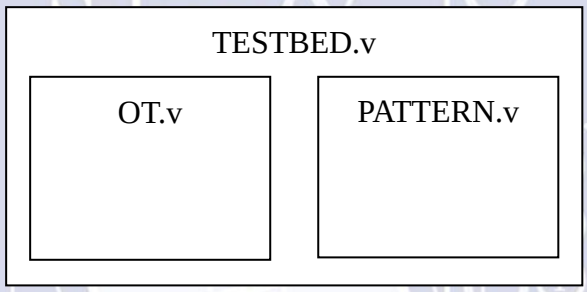
Input signal	Bit width	Definition
clk	1	Clock
rst_n	1	Asynchronous active-low reset
in_valid	1	Enable input signal
image_in	8	Single pixel of input image

Output signal	Bit width	Definition
out_valid	1	Enable output check
image_out	12	Single pixel of output image (Fixed point) First 4 bits (from LSB) are fractional

Specification

1. Top module name: **OT** (design file name: **OT**.v)
2. It is **asynchronous** reset and **active-low** architecture.
3. The reset signal would be given only once at the beginning of simulation. All output signals should be reset after the reset signal is asserted.
4. The maze signal will be given when in_valid pull up.
5. The input delay is set to **0.5*clock cycle**.
6. The output delay is set to **0.5*clock cycle**, and the output loading is set to **0.05**.
7. The synthesis result of data type **cannot** include any latches.
8. After synthesis, you can check **OT**.area and **OT**.timing. The area report is valid when the slack in the end of timing report should be **non-negative**.
9. Your design should output its result within **100 cycles**.
10. The default memory is 256x4 (words x bits) with mux 4 (area = 45372.905), and you can modify if you need it.

Block Diagram



Note

1. **Grading policy:**
   RTL and Gate-level simulation correctness: 70%
   Performance: 30%
   - Latency 5%
   - Area 25%
2. **Create a directory iclabXX and place the following files into it:**
   - **OT_iclabXX**.v
   - **iclabXX_your_clock_period.txt** (ex: iclab01_5.txt)
   - **memorysize.txt** (format: words_bits_mux.txt, ex: 256_2_4.txt)
   - **04_MEM (If you generated your own memory)**
3. **Compress the directory**
   - **tar cvf iclabXX.tar iclabXX**
4. **Upon completion, meet the TA in the front desk with your created directory and hand in the directory using the following commands**
   - **sftp iclabXX@linux01.ee.nctu.edu.tw**

- **[ENTER PASSWORD]**
- **Change directory (cd) to the directory containing your compressed file (iclabXX.tar)**
- **get iclabXX.tar**

5. **Template folders and reference commands:**

   01_RTL/ (RTL simulation) **./01_run**

   02_SYN/ (Synthesis) **./01_run_dc**

   (Check the design if there's latch or not in ***syn.log*****)**

   (Check the design's timing in /Report/**OT.*timing***)

   03_GATE/ (Gate-level simulation) **./01_run**

   04_MEM/ (Your RA1SH memory):

   ./01_mem_gen.sh (create the memory) (create the script by yourself)

   ./02_lib_gen_syntax_match.sh (make sure the format is correct)

Example wave form

Input:

Output:

GOOD LUCK!