# Implement of Many-Core System Homework 3

109061564 馬婕芸

## Introduction

In this homework, Transaction-Level Modeling (TLM2.0) sockets are used wrap Gaussian blur and testbench modules. Based on homework2, FIFO (first-in-first-out) channels and row buffers are also used in Gaussian blur processor to reduce number of data transaction, and the functionality of Gaussian blur and system architecture are quite the same. The implementation details and simulation results of both will be discussed in the following sections.

## Implementation details

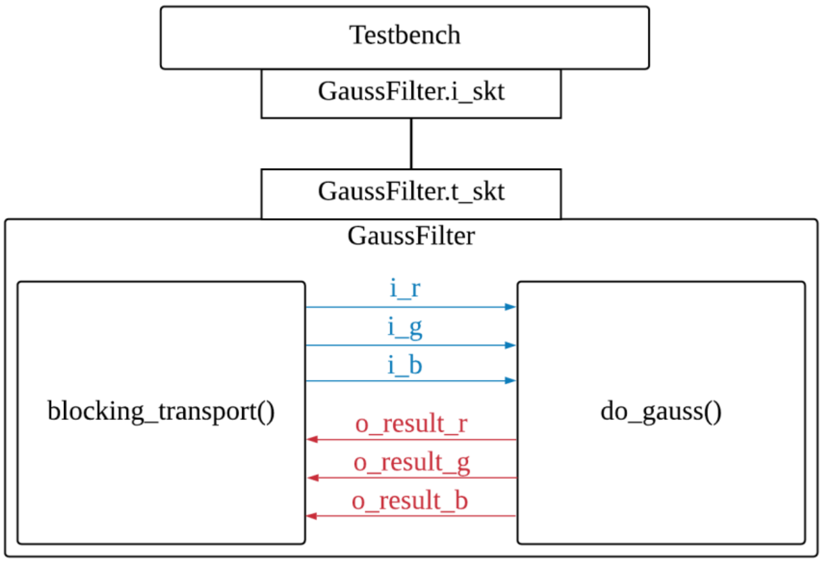
1. Gaussian blur filter is an image filter in digital signal processing. In this homework, we will implement a 3x3 Gaussian blur filter. The kernel is defined as shown below:

$$\text{kernel} = \begin{bmatrix} 1, & 2, & 1 \\ 2, & 4, & 2, \\ 1, & 2, & 1 \end{bmatrix}$$

After 2-D convolution with the original image (lena_std_short.bmp), Gaussian blur process finishes. Also, the 2-D convolution is defined as shown below: since there are three values (r, g, b) in one pixel, the convolution will be done on each value one by one.

$$\text{result}[m,n] = \sum_j \sum_i \text{original}[i,j] \cdot \text{filter}[m-i, n-j], \qquad 0 \le m,n,i,j \le 2$$

2. SystemC process with TLM 2.0 interfaces: The architecture is shown as follows.

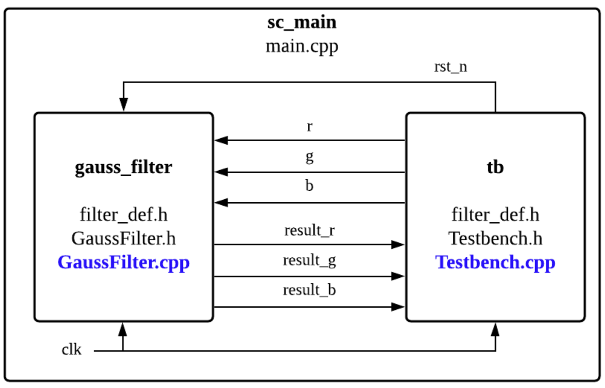
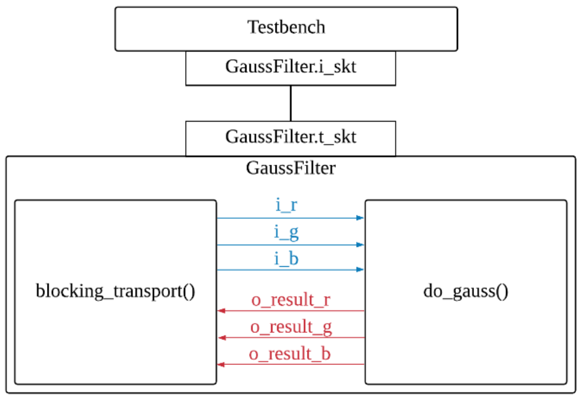


Between Testbench and GaussFilter modules, there are point-to-point TLM 2.0 interfaces, including initiator and target socket. The module Testbench works as TLM initiator, using SystemC module to handle TLM transaction; the module GaussFilter acts as TLM target, and additionally using SystemC FIFO to handle data usage between two function.

In Testbench.cpp, every pixel is sent row by row, and GaussFilter module will buffer them for further usage. As a result, the number of data access will be reduced compared to homework1.

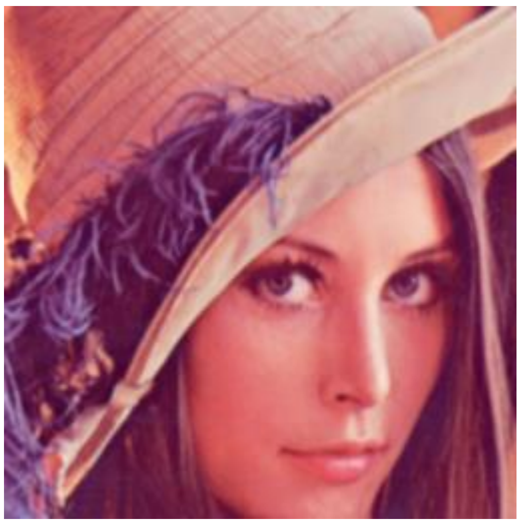
In GaussFilter.cpp, just as homework2, an additional buffer in the block: due to the nature of 2D convolution, each data(pixel) is used for about 9 times, as a result, there must be a buffer (256 pixels * 256 pixels * 3 color channels) to save data which is already input. However, the hardware system area may be increased in this way.

3. Comparison of architecture with and without TLM 2.0 interfaces: TLM 2.0 interfaces are used to wrap data transaction between two modules.

| Homework2 architecture | Homework3 architecture |
|---|---|
|  |  |

## Results

1. lena_std_short.bmp:

| Before Gaussian blur | After Gaussian blur |
|---|---|
| |  |

## Conclusion

In this homework, TLM-2.0 interfaces are a convenient and standard way to handle data transaction. Without changing the functionality and the data structure inside modules, we can address the data transaction issue more clearly using sockets, standard generic payload, and blocking transport. Also, compared to TLM-1 models, during simulation, TLM-2.0 uses blocking and non-blocking transport interface to address the shortcoming of calling “wait” by the addition of timing annotation. As a result, the simulation can be speed up.