EE6470 Electronic System Level Design and Synthesis
Homework1 Report

108062209 王鴻昱

- Github repo: https://github.com/PaulWang0513/Electronic-System-Level-Design-and-Synthesis

- Problem Description and Solution

  In this assignment, we are asked to implement a module that performing median filter and mean filter sequentially based on the given sobel filter module.

  As the first SystemC assignment, I think the main challenges are how to flexibly using channels to establish the communication between modules, properly arrange the order of two filters, and reasonably buffer the input to reduce data transfer.

  To solve these problems, I add two more enable signals to control two filter respectively, so I can start or stop the filters with at a proper moment. Also, I used an input buffer with size 18 to improve the data reuse, the achieve nearly 3 times improvement eventually.

- Implementation Details

  As previously mentioned, I add two enable signals (i_en_median, i_en_mean) in the filter module to control two filter process (do_median_filter, do_mean_filter) respectively.

```
class MedianMeanFilter : public sc_module {
public:
    sc_in_clk i_clk;
    sc_in<bool> i_rst;
    sc_in<bool> i_en_median;
    sc_in<bool> i_en_mean;
    sc_fifo_in<unsigned char> i_r;
    sc_fifo_in<unsigned char> i_g;
    sc_fifo_in<unsigned char> i_b;
    sc_fifo_out<int> o_result;

    SC_HAS_PROCESS(MedianMeanFilter);
    MedianMeanFilter(sc_module_name n);
    ~MedianMeanFilter() = default;

private:
    void do_median_filter();
    void do_mean_filter();
    int val[MASK_X * MASK_Y];   // make the size
};
#endif
```

  At the beginning of testbench, I set two enable signal as false to delay their execution by the conditional wait loop, and set to true when they should start.

```
void MedianMeanFilter::do_median_filter() {
    while (true) {
        // waiting for enable signal
        while (i_en_median.read() == false) {
            wait();
        }
```

Due to the 3x3 filter size, the filters will need 9 pixels to compute the result of one output position. For the no buffer version, we need to send 9 pixels from testbench to filter module, this cause a huge overhead on data transferring, since we continuously sending same data for different pixels (six out of nine data are the same to compute two neighboring location). So here we implement a version with input buffer to reduce the number of data transfer.

```
void MedianMeanFilter::do_median_filter() {
    while (true) {
        // waiting for enable signal
        while (i_en_median.read() == false) {
            wait();
        }
        int idx = 0;
        // buffer all grey values
        for (unsigned int v = 0; v < MASK_Y; ++v) {
            for (unsigned int u = 0; u < MASK_X; ++u) {
                unsigned char grey = (i_r.read() + i_g.read() + i_b.read()) / 3;
                val[idx++] = (int)grey;
            }
        }
```

(sending 9 pixels for one position from testbench ↓ to filter ↑)

```
// for each pixel
for (y = 0; y != height; ++y) {
    for (x = 0; x != width; ++x) {
        adjustX = (MASK_X % 2) ? 1 : 0; // 1
        adjustY = (MASK_Y % 2) ? 1 : 0; // 1
        xBound = MASK_X / 2;            // 1
        yBound = MASK_Y / 2;            // 1

        // send pixels in mask to filter
        for (v = -yBound; v != yBound + adjustY; ++v) {   //-1, 0, 1
            for (u = -xBound; u != xBound + adjustX; ++u) { //-1, 0, 1
                if (x + u >= 0 && x + u < width && y + v >= 0 && y + v < height) {
                    R = *(source_bitmap + bytes_per_pixel * (width * (y + v) + (x + u)) + 2);
                    G = *(source_bitmap + bytes_per_pixel * (width * (y + v) + (x + u)) + 1);
                    B = *(source_bitmap + bytes_per_pixel * (width * (y + v) + (x + u)) + 0);
```

We know that horizontal nearby and vertical nearby output positions both share 6 same pixels in a 3x3 filter, so if we buffer the 6 pixels in the filter, only 3 more pixels will be needed to compute the result for one output position.

For convenience, I set the buffer size to be 2x3x3 = 18. The first 3x3 is used to store the whole 9 pixels for the right output position, and the second 3x3 is used to store the whole 9 pixels for the down one. Although only 6 pixels are needed to achieve the data reuse scheme we claimed, this way is much easier

to implement since we just want to observe the reduction of data transfer.

```
private:
    void do_median_filter();
    void do_mean_filter();
    int val[MASK_X * MASK_Y];   // make the size sufficient to store pixels for median filter
    unsigned char buffer[2][MASK_Y][MASK_X];   // use the first 2D array to buffer the last 511 pixels of a row => xoo
                                               //                                                                xoo
                                               //                                                                xoo
                                               // use the last 2D array to buffer the first pixel of the next row => xxx
                                               //                                                                   ooo
                                               //                                                                   ooo
};
```

To make use of the buffer, the order of data transfer will be needed to be considered. For each filter, there is no possibility to reuse data for the very first output position, so the 9 pixels will be transferred from testbench to filter, and all be stored in the both horizontal and vertical input buffers. For the next 511 pixels on the righthand side, we can shift the horizontal buffer left, and fill in the right most column with three new pixels from testbench. For the 513th pixel, which is the leftmost pixel on the second row of the image, can reuse the 6 pixels we stored in vertical buffer previously, so the vertical buffer needs to be shift up and fill in with three new pixels from testbench.

```
if (col == 0) { // first column
    if (first_row) {
        first_row = false;
        // buffer all grey values
        for (unsigned int v = 0; v < MASK_Y; ++v) {
            for (unsigned int u = 0; u < MASK_X; ++u) {
                unsigned char grey = (i_r.read() + i_g.read() + i_b.read()) / 3;
                buffer[0][v][u] = grey;
                buffer[1][v][u] = grey; // for the first column of next row
                val[idx++] = (int)grey;
            }
        }
    } else {
        // get pixels from buffer, and the bottom most pixels from input
        for (unsigned int v = 0; v < MASK_Y-1; ++v) {
            for (unsigned int u = 0; u < MASK_X; ++u) {
                buffer[1][v][u] = buffer[1][v+1][u];
                buffer[0][v][u] = buffer[1][v][u];  // for the first column of next row
                val[idx++] = (int)buffer[1][v][u];
            }
        }
        for (unsigned int u = 0; u < MASK_X; ++u) {
            unsigned char grey = (i_r.read() + i_g.read() + i_b.read()) / 3;
            buffer[1][MASK_Y-1][u] = grey;
            buffer[0][MASK_Y-1][u] = grey;  // for the first column of next row
            val[idx++] = (int)grey;
        }
    }
} else {    // other columns
    // get pixels from buffer, and the right most pixels from input
    for (unsigned int v = 0; v < MASK_Y; ++v) {
        for (unsigned int u = 0; u < MASK_X-1; ++u) {
            buffer[0][v][u] = buffer[0][v][u+1];
            val[idx++] = (int)buffer[0][v][u];
        }
        unsigned char grey = (i_r.read() + i_g.read() + i_b.read()) / 3;
        buffer[0][v][MASK_X-1] = grey;
        val[idx++] = (int)grey;
    }
}
```

If we keep doing in this way, we can discover that except for the first output position, all other positions just need 3 more new pixels to complete the computation, the number of data transfer is reduced to nearly 1/3 than the no buffer version.

- Additional Features

In my implementation, all size is relative to MASK_Y and MASK_X. For example, the input buffer is declared as

unsigned char buffer[2][MASK_Y][MASK_X];

but not

unsigned char buffer[2][3][3];

, and the median is computed as

*median = val[(int)floor(MASK_X*MASK_Y / 2)];*

but not

median = val[4]; .

This means that it is extensible with respect to different filter size, providing more flexibility.

- Experiment Results

Functionality:

The median filter picks the median from 9 given pixel values, filtering out the extreme values, and hence remove the noise of the image.

The mean filter averages the value of 9 given pixel make each output position looks more similar with others, and hence produce a vague effect.

Both two filters work well to process the noised images. The results can be seen in the next page.

Degree of Improvement:

Before we applying an input buffer, one output location needs 9 pixels to compute, so the number of data transfer is:

*image size * 9 pixels * number of filters = (512*512)*9*2 = 4,718,592*

After applying input buffer, only the first output position needs 9 pixels to compute, other positions need only 3 pixels, so the number of data transfer is:

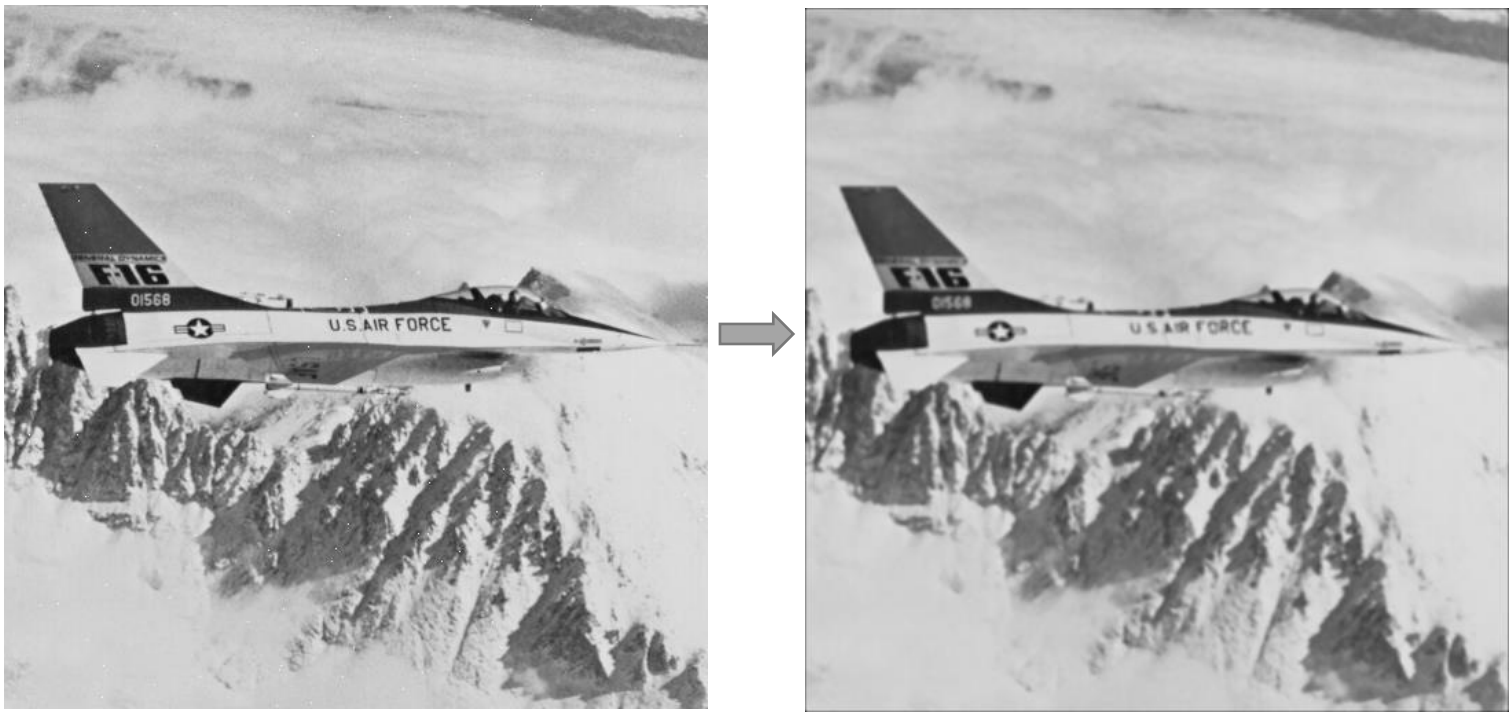
*((image size–1)*3+9)*number of filters = ((512*512-1)*3+9)*2 = 1,572,876*

Improvement: 4,718,592 / 1,572,876 = 2.999977112

We eventually achieve a nearly 3 times reduction on number of data transfer.

- Discussion and Conclusions

  In this assignment, I learned the way how median filter and mean filter work, and also getting more familiar with SystemC programming. At the end, we further apply an input buffer in filter module to try reducing number of data transfer. Although lots of code and functions are written with some software techniques, and many implementations can be further optimized to speed up on a hardware, it's still a valuable work as a simulation task and homework.



Filter_without_buffer:

```
[100%] Generating out.bmp

        SystemC 2.3.3-Accellera --- Feb 16 2023 13:55:28
        Copyright (c) 1996-2018 by all Contributors,
        ALL RIGHTS RESERVED
input file name : ../../noise_images/lake_noise.bmp
Image width=512, height=512
Median filter start ...
Median filter done
Mean filter start ...
Mean filter done
Total pixel transfer: 4718592
```

Filter_with_buffer

```
[100%] Generating out.bmp

        SystemC 2.3.3-Accellera --- Feb 16 2023 13:55:28
        Copyright (c) 1996-2018 by all Contributors,
        ALL RIGHTS RESERVED
input file name : ../../noise_images/jetplane_noise.bmp
Image width=512, height=512
Median filter start ...
Median filter done
Mean filter start ...
Mean filter done
Total pixel transfer: 1572876
```