EE6470 Electronic System Level Design and Synthesis
Homework3 Report

108062209 王鴻昱

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

- Problem Description and Solution

  i. Base Implementation:

  In this part, we are asked to write the buffer-based “flow” median and mean filter in a synthesizable way. I reused the previous implementation in hw2 to achieve this. Generally, I merged the separate R, G, B channels just like the one in lab6 SobelFilter. Then, changed the interface to a synthesizable version.

  ii. Improve Coding Styles:

  In this part, we are asked to modify the code itself with better coding style, to achieve a better QoR. So, I re-defined the data width of variables and arrays, made as much loop bound constant as possible, modified some algorithm, and modified some expression in a more efficient way. Then compared the QoR with base implementation version.

  iii. Optimized Implementation:

  I this part, we are asked to use Stratus directives to further improve the QoR. So, I tried different setting of clock period, array mapping, loop unrolling, etc. Then compared the QoR with previous versions.

- Implementation Details

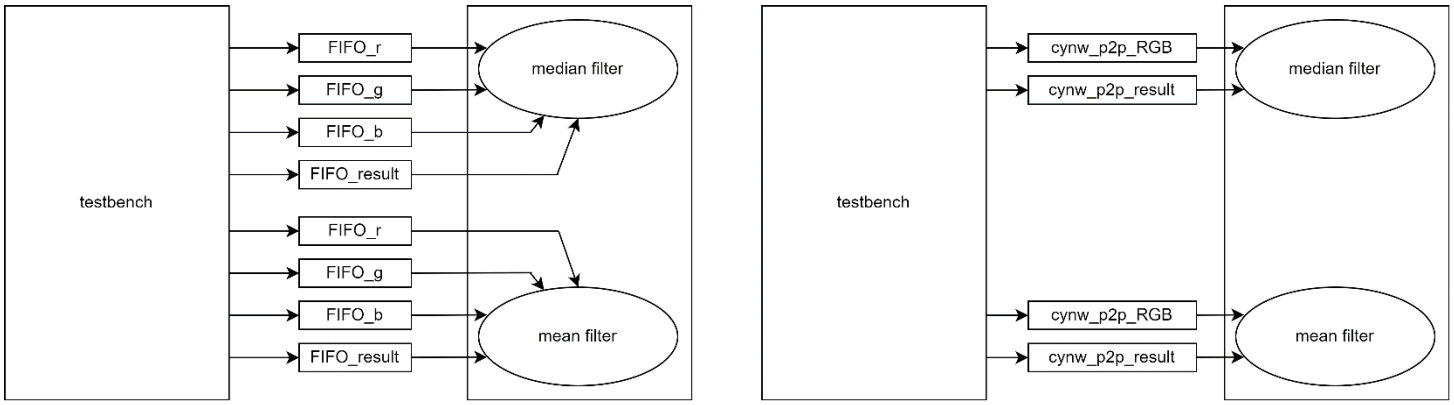

Diagram1. Block diagram of hw2 version (left) and hw3 version (right)
(Other channels are not shown for simplicity)

i. Base Implementation:

To make the design synthesizable, I merged the separated R, G, B FIFO channels into one RGB cynw_p2p<24> channel, and changed the FIFO channel for transferring results into cynw_p2p<32> channel, just like diagram1 show. Then, to properly use the cynw_p2p channel, I changed a few parts of the code like port reset and get/put data into channels.

```
15      i_rgb_median.clk_rst(i_clk, i_rst);
16      o_result_median.clk_rst(i_clk, i_rst);
17      i_rgb_mean.clk_rst(i_clk, i_rst);
18      o_result_mean.clk_rst(i_clk, i_rst);
```

Image1. Specify clock and reset signal to cynw_p2p port

```
23  void MedianMeanFilter::do_median_filter() {
24      {
25          HLS_DEFINE_PROTOCOL("median_reset");
26          i_rgb_median.reset();
27          o_result_median.reset();
28          wait();
29      }
```

```
140 void MedianMeanFilter::do_mean_filter() {
141     {
142         HLS_DEFINE_PROTOCOL("mean_reset");
143         i_rgb_mean.reset();
144         o_result_mean.reset();
145         wait();
146     }
```

Image2. Reset the cynw_p2p port at the begging of median filter thread (left) and mean filter thread (left)

```
sc_dt::sc_uint<24> rgb;
{
    HLS_DEFINE_PROTOCOL("input");
    rgb = i_rgb_median.get();
    wait();
}
```

Image3. Example of getting data from cynw_p2p port, the HLS_DEFINE_PROTOCOL directive is to specify the timing (1 cycle)

```
{
    HLS_DEFINE_PROTOCOL("output");
    o_result_median.put(median);
    wait();
}
```

Image4. Example of putting data into cynw_p2p port, the HLS_DEFINE_PROTOCOL directive is to specify the timing (1 cycle)

ii. Improve Coding Styles:

In this part, I modified the code itself to improve the QoR.

First, I constrained the data width of variables and arrays. Unsigned char instead of int is used for the buffer to store gray value of input data. Unsigned int instead of int is used for the accumulation register of mean filter. sc_uint<2> instead of int is used to store the value of mean filter mask, since the maximum value is 2. Most other variables in the filter use unsigned int to enable Stratus to perform bit trimming.

Then I changed most of the loop bound into constant to enable loop-unrolling later. And remove unnecessary LCD (loop carry dependency) to improve register reuse.

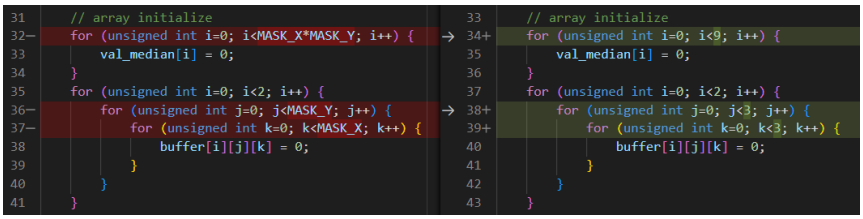

Image5. Example of loop bound modification.

Though this should not cause improvement due to constant propagation of HLS

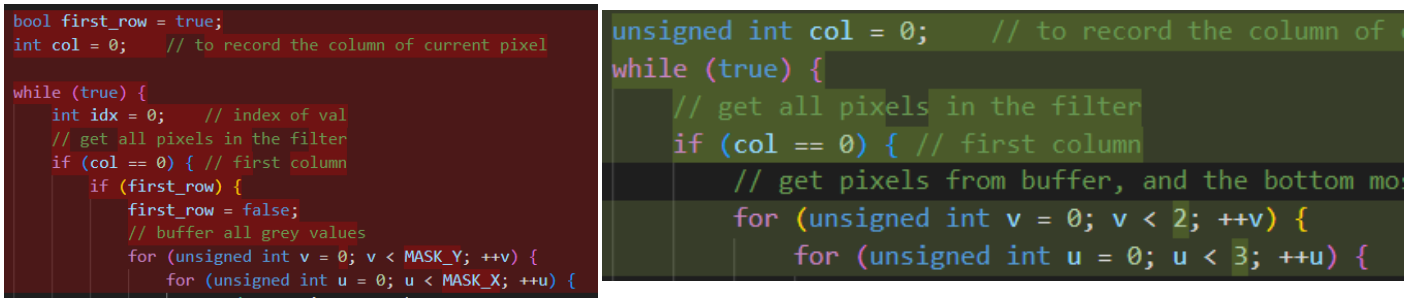

Image6. Example of removing LCD.

The first_row is removed by taking the corresponding tasks out of the loop.

The idx is removed because it's can be constant propagated from v and u

Then I modified some algorithm and operation to simplify hardware computation. For example, it's unnecessary to sort all 9 values to find the median in median filter, only 5 smallest or greatest values are enough.

Image7. Example of operation simplification.

iii. Optimized Implementation:

In this part, I tried 3 different optimizations on the previous work.

For the first version, I flatten all small arrays, and map the input buffer with size 18 into reg bank to prevent large area. I also turn on the unrolling global setting to enable the parallel processing as much as possible. Then I set dpopt_auto=all to make Stratus optimize the design automatically. Without loop pipelining, this version achieves the latency=3 cycle in both median and mean filter in clock_period=10ns. The latency cannot be further decrease because the 3 cycle is for data reading. So, I tried the next version.

For the second version, I used the same setting above with

clock_period=2ns, and got the latency of median filter and mean filter as 3 cycles and 5 cycles respectively. However, I failed to pipeline the design.

For the third version, I used the same setting with clock_period=1ns, and got the latency of median filter and mean filter as 10 cycles and 12 cycles respectively. And same with above, loop pipelining is failed.

- Additional Features

  No additional features this time.

- Experiment Results

	Clock period (ns)	Area	Median filter avg. latency (cycle)	Mean filter avg. latency (cycle)	Total avg. latency (cycle)	Throughput (pixels/us)
Base Implementation	10	16485.4	185	32	109	0.917
Improve Coding Styles	10	13386.7	133	9	71	1.408
Optimized1	10	9133.1	3	3	3	33.3
Optimized2	2	9727.9	3	5	4	125
Optimized3	1	11274.2	10	12	11	90.9

Table1. Comparison of QoR between different implementation

The way I compute the latency is compute the number of cycles between feeding input needed for a result and fetching the result. By averaging the latency of all median filter output, we obtain the median filter avg. latency. By averaging the latency of all mean filter output, we obtain the mean filter avg. latency. By averaging the latency of median and mean filter output, we obtain the total avg. latency. Throughput is computed by

*1000 / (clock period * total avg. latency)*

- Discussion and Conclusions

  In Table1, we can see that by using proper coding style, the area and latency of the design can have a large improvement, informing the importance of knowing the mechanism of HLS. On the basis of proper coding style, we can further apply HLS directives to agilely explore the optimization space.

  Furthermore, we can observe that low latency does not necessarily indicate the largest throughput, with proper manipulation, Optimized2 version provide the highest throughput while Optimized1 version have the lowest

latency and Optimized3 version have the lowest clock period. This can be even more obvious if loop pipelining is applied.

In this assignment, the only insufficient for me is not applying loop pipelining. Although I have tried different setting of hls_config and HLS_LOOP_PIPELINE parameter, I still can’t successfully finish a synthesis. I think the reason may be the module have two thread with dependency, but more research is needed to find the true reasons.