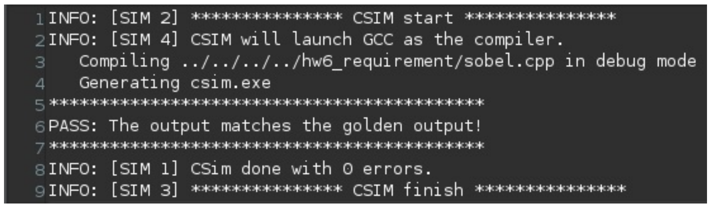
EE216 Re-configurable Computing Homework6 Report

Zhaojun Ni, 2022231102

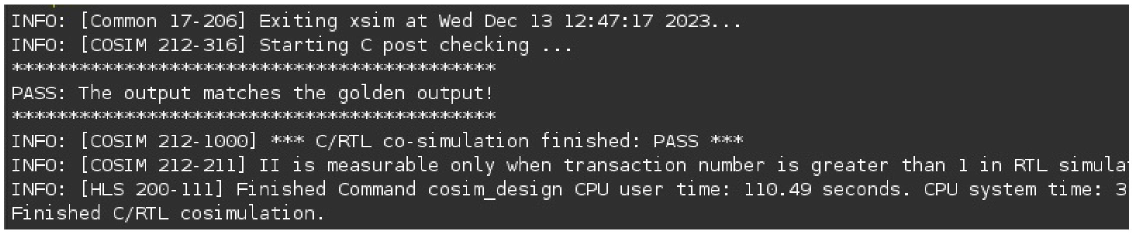
December 13, 2023

1 CSIM/CO-SIM Result



```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling ../../../../hw6_requirement/sobel.cpp in debug mode
   Generating csim.exe
*******************************************
PASS: The output matches the golden output!
*******************************************
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************
```

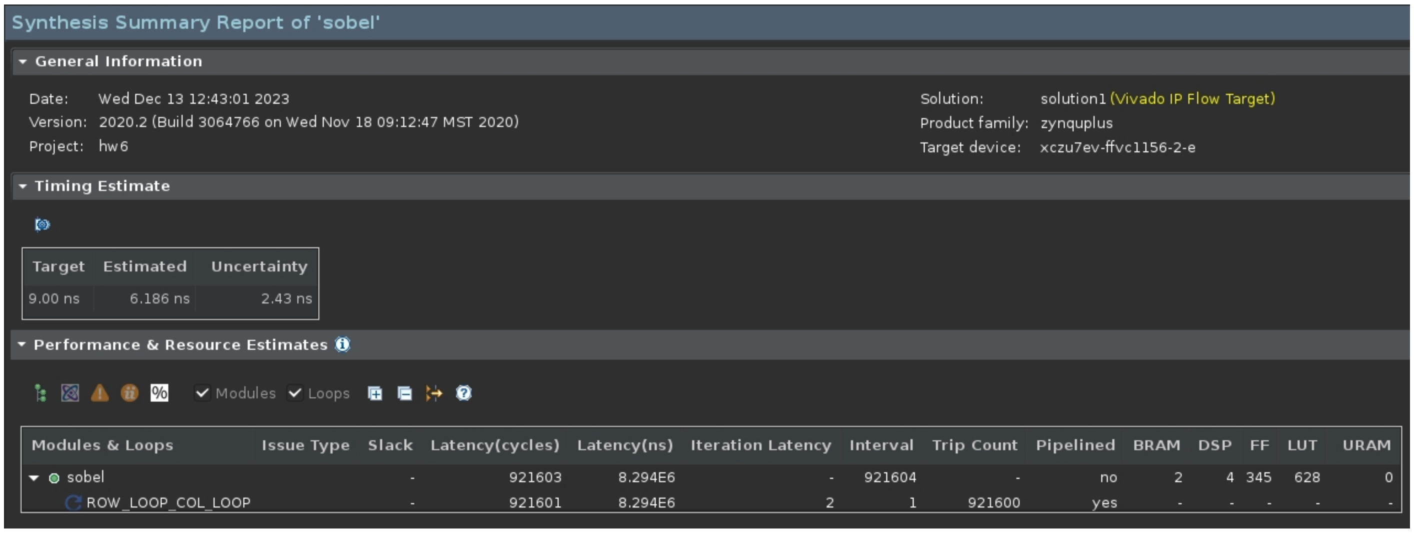
Figure 1: CSIM output matches the golden output.



```
INFO: [Common 17-206] Exiting xsim at Wed Dec 13 12:47:17 2023...
INFO: [COSIM 212-316] Starting C post checking ...
*******************************************
PASS: The output matches the golden output!
*******************************************
INFO: [COSIM 212-1000] *** C/RTL co-simulation finished: PASS ***
INFO: [COSIM 212-211] II is measurable only when transaction number is greater than 1 in RTL simula
INFO: [HLS 200-111] Finished Command cosim_design CPU user time: 110.49 seconds. CPU system time: 3
Finished C/RTL cosimulation.
```

Figure 2: COSIM output matches the golden output.

2 C Synthesis Report



Synthesis Summary Report of 'sobel'

General Information

Date:	Wed Dec 13 12:43:01 2023	Solution:	solution1 (Vivado IP Flow Target)
Version:	2020.2 (Build 3064766 on Wed Nov 18 09:12:47 MST 2020)	Product family:	zynquplus
Project:	hw6	Target device:	xczu7ev-ffvc1156-2-e

Timing Estimate

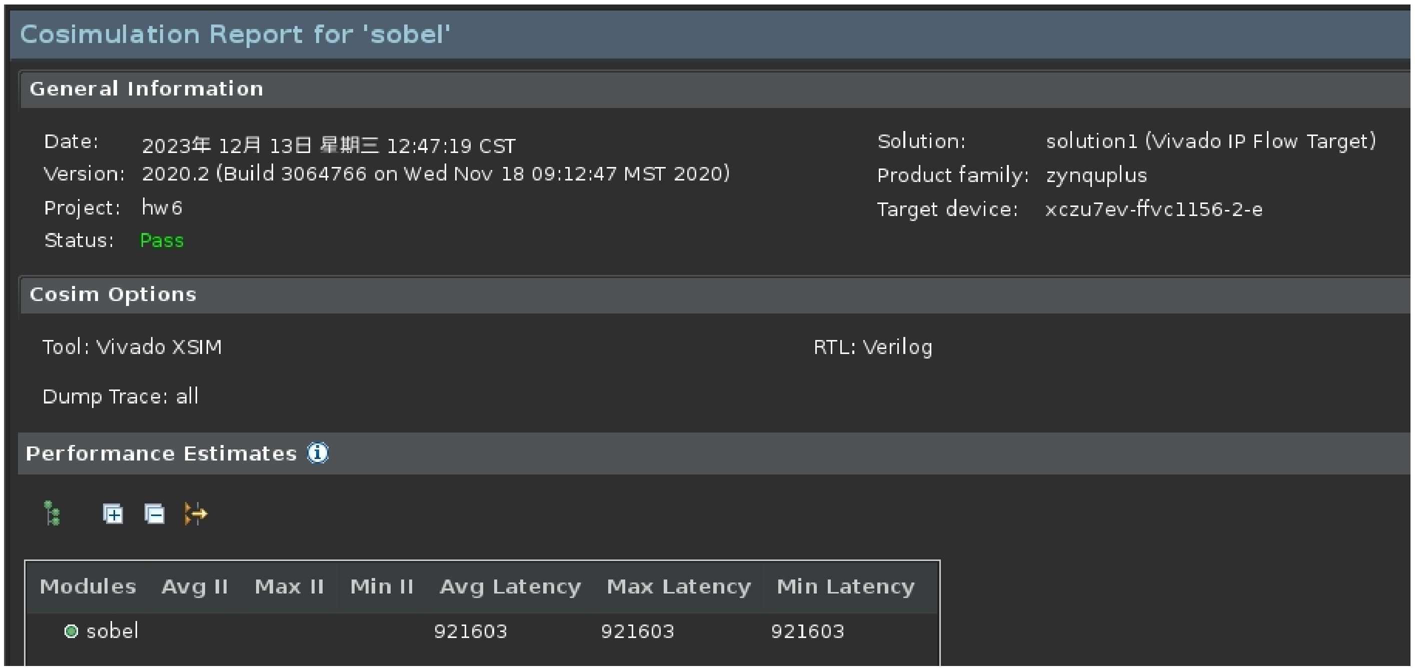
Target	Estimated	Uncertainty
9.00 ns	6.186 ns	2.43 ns

Performance & Resource Estimates

Modules & Loops	Issue Type	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
sobel		-	921603	8.294E6	-	921604	-	no	2	4	345	628	0
ROW_LOOP_COL_LOOP		-	921601	8.294E6	2	1	921600	yes	-	-	-	-	-

Figure 3: C Synthesis Report.

3 CO-SIM Report



Cosimulation Report for 'sobel'

General Information

Date:	2023年 12月 13日 星期三 12:47:19 CST	Solution:	solution1 (Vivado IP Flow Target)
Version:	2020.2 (Build 3064766 on Wed Nov 18 09:12:47 MST 2020)	Product family:	zynquplus
Project:	hw6	Target device:	xczu7ev-ffvc1156-2-e
Status:	Pass		

Cosim Options

Tool: Vivado XSIM

RTL: Verilog

Dump Trace: all

Performance Estimates

Modules	Avg II	Max II	Min II	Avg Latency	Max Latency	Min Latency
sobel				921603	921603	921603

Figure 4: CO-SIM Report.

4 Optimization

This design's clock constraint is set to 9ns. The estimated max frequency of my design is 161.66MHz and the latency of the top function is 8.294ms. The interval is optimized to 921604 and the hardware resource consumption has also been significantly decreased.

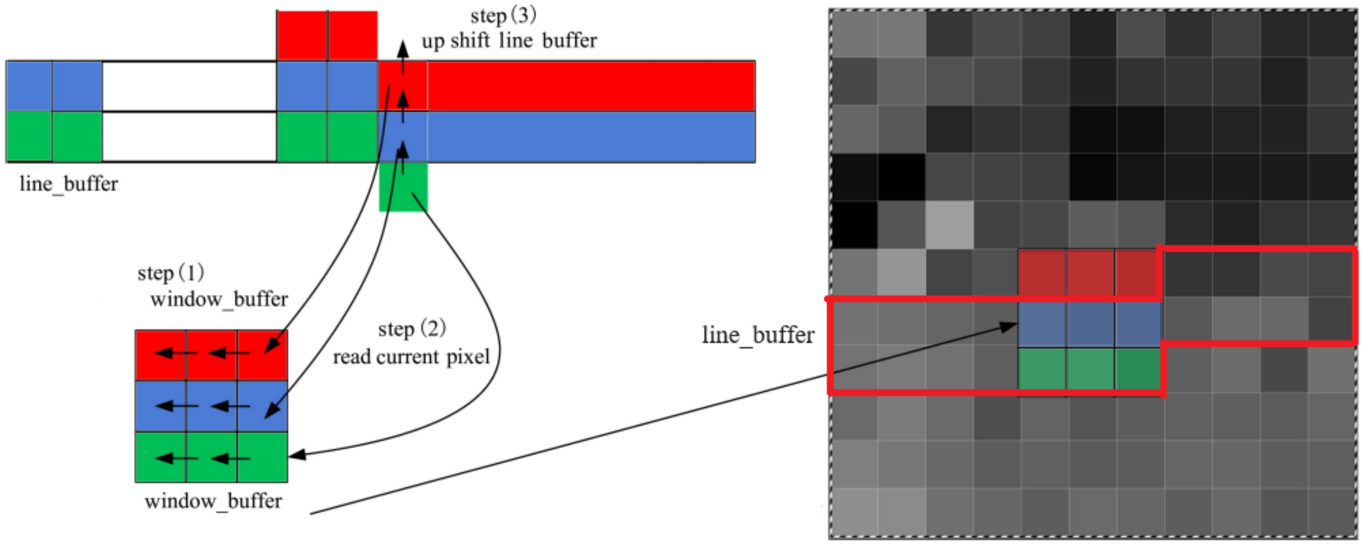


Figure 5: Line buffer and window buffer.

Optimization 1: Line buffer and window buffer. We can observe that the content in adjacent moving windows is often overlapped, implying a high locality. Thus the input pixels can be buffered and accessed multiple times. If the design doesn't use buffers, fetching the input for a 3x3 filter each time requires reading 9 data from memory, and the positions may not be contiguous. This leads to significant waste as, when processing the next window, it necessitates reading another 9 data again. Drawing a parallel to the design of accelerators in ASIC, the input buffer for neural networks is typically placed in SRAM; in the case of FPGA, it is implemented in BRAM. The weight buffer for neural networks is also usually stored in SRAM, but in this case, due to the fixed parameters of the filter kernel, it can be implemented using FFs. For a 3x3 filter, the line buffer only needs to store two lines of input.

The inspiration for this approach can be found in the chapter on Video Systems in the book 'Parallel Programming for FPGAs'. Fig.5 is from H. Shi and H. Hu, "Acceleration method of infrared image detail enhancement on FPGA", Journal of Northwestern Polytechnical University,2022 and modified.The line buffer is designed as a two-dimensional array with a height of 2 and a width of WIDTH, and the window buffer is designed as a two-dimensional array with a size of 3x3. Both arrays have a data type of PIXEL. The overall design involves a total of 921600 iterations through ROW_LOOP and COL_LOOP, where ROW_LOOP iterates 720 times and COL_LOOP iterates 1280 times.

Within the inner COL_LOOP, with a column index j, each iteration involves the following steps: Move the last two columns of the window buffer to the first two columns. Pop the first row of the corresponding column from the line buffer to the first row of the window buffer. Move the second row of the corresponding column in the line buffer up to the first row and send it to the second row of the window buffer. Finally, read the current pixel value src[i][j] and send it to the second row of the line buffer and the third row of the window buffer.

Due to the lack of consideration for padding in this design, there is no need to output in cases where the window buffer contains 0 or intersects with the boundary. Therefore, output is performed only when $i \geq 2$ and $j \geq 2$ in the loop body. As a result, the total number of output values is (720-2) * (1280-2).

Optimization 2: Datatype optimization. Because one of the objectives of hardware design is to achieve optimal cost-effectiveness in resource utilization, considerations also need to be made regarding the data width.The PIXEL data type designed in the header file "sobel.h" is defined as short int, indicating that the data is stored in 16 bits. However, in reality, the input and output pixel values range from 0 to 255 and are of uint8 type. To prevent data range overflow during intermediate multiply-add calculations, the data type of PIXEL has been changed to ap_int with a width of 9 bits. Additionally, the loop counter registers have been changed from int to ap_uint type. Since the rows and cols parameters in the sobel function are defined as int and cannot be modified, and practical tests showed that changing the corresponding index registers to ap_uint with 10 and 11 bits would result in some delay increase, no modification has been made in this regard. A counter with a maximum value of 2 has been changed to ap_uint with a width of 2. These optimizations have resulted in a reduction in BRAM usage from 4 to 2, FF (Flip-Flops) from 380 to 345, and LUT (Look-Up Tables) from 659 to 628 after synthesis.