CSE237C Lab1 Report

Hanyang Xu A92068025 hax032

Question 1 - Variable Bitwidths: How does the bit width affect the performance? How does it affect the resource usage? What is the minimum data size that you can use without losing accuracy (i.e., your results still match the golden output)?

Performance wise, reducing the bitwidth improves both timing and latency. Compare with the baseline, the frequency increases from 8.51 to 7.604 and the max latency reduces from 513 to 385.

Resource Usage wise, the total DSP48E used reduces from 2 to 0, total FF used reduces from 215 to 104, total LUT used reduces from 254 to 241.

For **coef_t**, the minimum bit size is 5 since, within all the coefficient, the max coefficient is 11 and the min coefficient is -11. For **data_t**, the minimum bit size is 9. It is found by trials which 9 is the minimal data size for result to match the output. For **acc_t**, the minimum bit size is 16. It is found by trials which 9 is the minimal data size for result to match the output. I use fir128_baseline and fir128_optimized1 for this answer.

Question 2 - Pipelining: How does increasing the II affect the loop latency? What are the trends? At some point setting the II to a larger value does not make sense. What is that value in this example? How would you calculate that value for a general for loop?

For II = 1, HLS cannot achieve such goal; For II = 2, loop latency = 257+4=261; for II = 3, loop latency = 384+4=388; for II = 4, loop latency = 512+4=516. The loop latency increases with initiation interval. In the example, setting II equal or greater than 4 does not make sense since waiting every 4+ cycle before issue a new computation is identical/worse than the none pipelined version. For a general loop, I will calculate the loop latency of certain II values for that loop and compare to the latency of non-pipeline version, if the latency of pipelined version is greater, then that II value does not make sense. I use fir128_baseline and fir128_optimized2 for this answer.

Question 3 - Removing Conditional Statements: Rewrite the code to remove any conditional statements. Compare the designs with and without if/else condition. Is there a difference in performance and/or resource utilization? Does the presence of the conditional branch have any effect when the design is pipelined? If so, how and why?

Compared to baseline design, the max latency decreases from 513 to 509 while the min latency increases from 257 to 509. The total FF used reduces from 215 to 132, total LUT used reduces from 254 to 246.

After pipelining with II = 2, the max latency decreases from 509 to 257, the min latency from baseline of 257 is achieved in this design. The total FF used reduces from 215 to 135, total LUT used increased from 254 to 260. Once the conditional branch is removed, the pipeline architecture can then be fully optimized for throughput without the need to check the condition and therefore decrease the latency. I use fir128_baseline and fir128_optimized3 for this answer.

Question 4 - Loop Partitioning: Is there an opportunity for loop partitioning in FIR filters? Compare your hardware designs before and after loop partitioning. What is the difference in performance? How do the number of resources change? Why?

Yes, it can be partitioned into to part: 1) data shifting 2) sum and multiply. Compared two design, the total latency increased from 513 to 770 since the loops are not pipelined. However, compared with the pipelined version, the latency still increased from 259 to 390. Latency only decreases when combine loop partitioning with memory partitioning which allows Vivado to synthesizing the code so that two loops can run at the same time with no limit on memory bandwidth.

Resource Usage wise, total FF used reduces from 215 to 295, total LUT used reduces from 254 to 400. It increases since now the system needs two separate loop structure instead of one. I use fir128_baseline, fir128_ optimized2 and fir128_optimized4 for this answer.

Question 5 - Memory Partitioning: Compare the memory partitioning parameters: block, cyclic, and complete. What is the difference in performance and resource usage (particularly with respect to BRAMs and FFs)? Which one gives the best performance? Why?

Block:

2: Performance: Clock:8.51, Latency:(257:641)   Resources: BRAMs: 2, FFs:196, LUTs:334

4: Performance: Clock:8.51, Latency:(257:641)   Resources: BRAMs: 4, FFs:234, LUTs:363

8: Performance: Clock:8.51, Latency:(257:641)   Resources: BRAMs: 0, FFs:765, LUTs:536

Cyclic:

2: Performance: Clock:8.51, Latency:(257:641)   Resources: BRAMs: 2, FFs:190, LUTs:331

4: Performance: Clock:8.51, Latency:(257:641)   Resources: BRAMs: 4, FFs:228, LUTs:358

8: Performance: Clock:8.51, Latency:(257:641)   Resources: BRAMs: 0, FFs:761, LUTs:536

Complete:

Performance: Clock:8.51, Latency:(257:385)   Resources: BRAMs: 0, FFs:4286, LUTs:2201

Complete gives the best performance since it has the maximum memory channel throughput. Since it stores every data in separate registers, it can provide all the data at once with zero congestion. I use fir128_baseline, and fir128_optimized5 for this answer.

Question 6 - Best Design: In what way is it the best? What optimizations did you use to obtain this result? It is possible to create a design that outputs a result every cycle, i.e., get one sample per cycle, so a throughput of 100 MHz (assuming a 10 ns clock).

I optimized for throughput. I ajust Bitwidths to save resources as in question 1. I removed conditional statements in the loop. I partitioned the loop into two loops and pipelined them. I partitioned memory into register completely. I get 6.38ns clock period and 131 clock cycle as the result which gives 1.196 Mhz.