Final Project-E315

1 What did we do? What worked?

The goal of this project was to accelerate the performance of an existing dot product implementation in Verilog. This task required the manipulation of two files within this library: *dot.sv*, which creates a module that calculates the dot product of streamed inputs with a weight matrix, and *accel_dot.sv*, which cuts the weight matrix in half and runs two dot modules in parallel.

To parallelize the computation in accel_dot.sv, we cut the weight matrix in half vertically, giving each half to a separate dot module. The dot modules share inputs and work with separate 5 column, 20 row weight matrices. The inputs are shared and we didn't have to worry about synchronizing the inputs, but outputs need to be concatenated in the end because of how the dot product operation works. To synchronize these outputs, we made a multiplexor so we can select the outputs of the dot module with the left half of the weight matrix first, then switch to the other dot product outputs when the first dot product signals that it's finished by raising OUTPUT_AXIS_TLAST. With this parallization and the given dot.sv, we got down to 208 clock cycles.

Next, we needed to alter the given dot.sv and get rid of unnecessary clock cycles so our implementation would be under 200 clock cycles. We found that in this case, rx_done was always high once the wait state is reached, so we got rid of the wait state. This got us down to 205 clock cycles. After that we noticed that there was a wasted clock cycle going from the TERM_ROW state to OUTPUT because the timer had already reached zero. To get rid of this wasted cycle, we made the timer equal to 7 instead of 8 so that the fmac first pipelined operation was finished on the transition to the OUTPUT state. This change got us down to 185 clock cycles.

2 What does not work?/What did we learn?

We were not able to get the accel_dot.sv to parallelize horizontally. We were able to split the weight matrix by rows and give every other input to the separate dot modules, but our fmac would output the first value correctly and never move on to later values. We suspect this was

because once our output of any dot was valid, the input valids of the two inputs of the fmac were never lowered, so it didn't accept a second value. In theory, the horizontal split implementation would work faster than the vertical split because there are no wasted clock cylces waiting on a result. When splitting by column, the number of columns becomes less than the number of cycles it takes to do an fmac calculation, so there are 3 cycles of waiting for the first output of each row.

We were also not able to use our implementation of dot.sv from project 6 in this project because our implementation did not work on weight matrix with less than 8 columns. Out implementation expected a result directly after the last value in a row was pipelined because the number of items in a row in project 6 was 10. When the dot running in parallel, the number of items in a row is 5, so it needs to hesitate before moving to output.

We learned that we need pay attention to the latency of an operation when pipelining that operation. Our implementation from project 6 did not work on project 7 because the latency of one fmac operation takes more clock cycles that it takes to give the fmac inputs 5 times. We also learned that there are multiple ways to parallelize problems and those implementations could force you to synchronize different aspects of the module (synchronize inputs or sychronize output).

3 What would we do differently in the future?

If we did this project again, we would continue to try to work on splitting weights by row. This approach seems to have more upside because we could create more than 2 dot modules and possible not get diminishing returns, and it would avoid ever waiting on outputs in dot.sv. We would also look at the fadd module and get a better understanding of how it works before going down this path.

4 Who did what?

[redacted] figured out all the changes we needed to make in dot.sv to go from 208 to 185 cycles. [redacted] got accel_dot.sv working when splitting by column. Writing the report was a team effort.