May 9, 2018

**ECE 551 Final Project Report**

Our workflow began with Akshat and Naman writing snn\_core.sv while Jon and Shubham wrote a corresponding testbench. The original plan was for Jon and Shubham to then write snn.sv while the other two wrote their own testbench but debugging snn\_core took a disproportionately long time and limited the progress that could be made to snn. So, Jon wrote rough code for snn and testbench while snn\_core was debugged, and Shubham worked on a synthesis script. After this was complete, labor could not be divided as effectively so all four team members worked together for much of the time to get snn\_core and then snn working.

Initially, we planned to produce a working top-level as described in the project specification prior to making optimizations. However, because debugging snn\_core was behind schedule, we decided to replace some components with optimized modules, given that the originals were not working anyway. Firstly, we replaced the ram\_output\_unit with a simple combination of an 8-bit register storing the current maximum probability of any digit, and a 4-bit register storing the digit to which the maximum probability corresponds. The output signal from the lookup-table was connected directly to the max\_prob register, which updated based on an enable signal dependent on (lut\_out > max\_prob). This both saved ten clock cycles that would be required to read from ram\_output\_unit after it had valid values, as well as the extra area from the ram itself.

Our second optimization involved an analysis of the given hidden weights. We noticed while debugging snn\_core that many weights at early and late indexes of each hidden node were zero, and found that for all 32 nodes, this was the case for at least the first 36 and the last 11 weights. There was no reason to make any calculations at indexes corresponding to these weights because they could not contribute to any value being accumulated in the MAC. So, instead of iterating through 784 input values and weights, we simply began iterating from 36 and ended at (783 -11). This saved [(36 + 11) \* 32], or 1504 clock cycles per digit. By using this method, we also bypassed the need for back porch states between calculating hidden and output values by letting the input address counter go to two values higher than the address of the last relevant input, 774 instead of 772, before changing states.

We accounted for two extra clock cycles when calculating output unit values in the same way. Instead of back porch states in the state machine, we had a counter which detected the end of readings for an output node at a hidden weight count of 34 instead of 32. This is a little weird and was honestly done because we were having trouble producing the proper functionality with back porch states. However, based on our implementation, it does not sacrifice notable performance or area. The number of clock cycles needed is the same. We had to add a sixth bit to our hidden unit counter logic (still connecting only the first five to the address of ram\_hidden) to be able to reach 34, but we also were able to remove a bit from both the state and next\_state registers because it reduces the total number of needed states from 6 to 4.

Finally, while Jon and Shubham worked on FPGA testing, Akshat and Naman began implementing a parallel MAC and probability comparator structure in a separate branch to improve performance, but given time constraints, we opted to abandon this and make sure our functional code was in good order. If we had more time, this would be the next optimization to complete, as it would improve performance drastically more than any other single change what we could conceive.

The final program that we loaded to the FPGA for our presentation was similar to Professor Kim’s, based on his sample Quartus analysis being comparable to our own:

|  |  |  |
| --- | --- | --- |
| **Summary** | **Professor Kim’s design** | **Group 5’s design** |
| Total logic elements | 314 | 294 |
| Total registers | 152 | 152 |
| Total memory bits | 283,904 | 283,904 |
| Embedded Multiplier 9-bit elements | 1 | 1 |
| Cycles (“start” to “done”) | 25,535 | 24,065 |