Project 7 Final Report

[redacted]

Introduction

The purpose of this assignment was to significantly reduce the number of cycles required to run the matrix dot product. Through a combination of pipelining and parallelization, the group reduced the time to 190 clock cycles.

Methodology

Attempt #1: "In theory, this is easy" -- [redacted]

The first approach to this problem (depicted in Figure 1) is outlined below:

- Split the weight matrix in two
- Create two dot multipliers, and feed each multiplier the data inputs and one of the new weight matrices
- The dot product unit calculates the product on a high valid flag
    - The two multipliers had separate valid flags that would alternate based on the value of `INPUT_AXIS_TVALID`.
- Retrieve the output products, and output them in the correct order

The first step in this process was to determine in which way to split the weight matrix. Initial misunderstandings on which set of data was the weight matrix lead to a malformed algorithm which assumed the weights were split horizontally when they were in fact split vertically. Another error was believing the weights to be split every-other element rather than in chunks.

In this attempt, the valid flags were to be calculated by keeping track of which multiplier received the previous valid value, and alternated when `INPUT_AXIS_TVALID` was raised high (seen in Figure 1). Following this approach, the weight matrix splitting, valid flag assignment, and the beginnings of parallelization were written. There was also code written that allowed switching to horizontally-split matrices in case that it was more efficient.

Failures of this approach were the inability to determine how to go about adding the resulting outputs together in a manner that resulted in the correct solution.

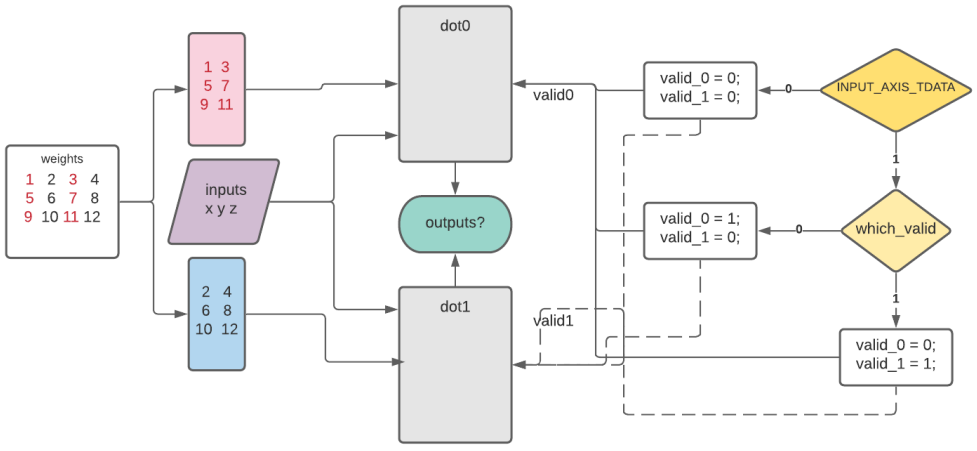

Figure 1. Flow chart describing attempt 1: *"In theory, this is easy"*.

Project 7 Final Report

[redacted]

Attempt #2: "Simplicity is better" -- [redacted]
This approach (depicted in Figure 2) is outlined below:

- The weight matrix was split vertically down the middle
- `FMAC_DELAY` was reduced from `8` to `8 - COLS`, the minimum necessary
- Send outputs as they are ready rather than waiting on all outputs to be finished
- This further cycle reduction was achieved by completely removing `ST_RX_WAIT`
- `dot0` sends its outputs first
- `dot1` waits on `dot0` to finish sending outputs
    - This was achieved through wire manipulation, as shown in Figure 2
- `dot1` sends outputs

Part 1 accel_dot:

Initially, it was thought that delays due to the inputs to each dot product unit being less than 8 columns could be alleviated by matrices switching between rows to run in the background while the other dot product unit calculated the other columns. Unfortunately, later analysis showed that both dot product units would run most efficiently both receiving the inputs simultaneously rather than staggered.

The wire manipulation described above takes the `INPUT_AXIS_TREADY` for `dot0` and uses a wire (`in_tready0`) to connect it to OUTPUT_AXIS_TREADY for `dot1` so that:

```
.OUTPUT_AXIS_TREADY(in_tready0 && OUTPUT_AXIS_TREADY)
```

In the instantiation of `dot1`.

Other wire manipulation done was to make `OUTPUT_AXIS_TVALID` be equal to `dot0`'s or `dot1`'s `OUTPUT_AXIS_TVALID` and `INPUT_AXIS_TREADY` to be equal to `dot0`'s and `dot1`'s `INPUT_AXIS_TREADY` (since the inputs are duplicated, the sender must wait on both to be ready before sending). The implementation is shown below.

```
assign INPUT_AXIS_TREADY = in_tready0 || in_tready1;
assign OUTPUT_AXIS_TVALID = valid1_out || valid0_out;
```

Finally, there needed to be code that sent out the correct output since there would be output streams from both dot products. To check this is simple: first, check whether `dot0` has a valid output, and send the output if it does. Otherwise, check `dot1` and send `dot1`'s output if it has a valid output. This sends all of `dot0`'s outputs first, and then all of `dot1`'s. This is implemented using a register and combinational logic.

```
reg [31:0] outputs;

always_comb begin
   outputs = 32'h0;

   if (valid0_out) begin
      outputs = output0;
   end
   else if (valid1_out) begin
      outputs = output1;
   end
end

assign OUPUT_AXIS_TDATA = outputs;
```

Part 2 dot:

The dot product unit supplied was also inefficient. The delay between rows was 8 clock cycles when, in fact, it could have been `8 - COLS` since by the time the computation reaches the end of a row, it has been `COLS` cycles since the first column.

Another inefficiency was the `RX_WAIT` state. This waits for all entries of the dot product to finish before being ready to send them out. Instead, it's possible to send each entry as it is calculated.

This was the hardest section to optimize, as initially, removing states led to indefinite hanging. This hanging was due to `run_fmac` not being set high at the right times. Turning it into sequential logic solved this problem and lowered the time to the final value of 190 cycles.

Project 7 Final Report

[redacted]

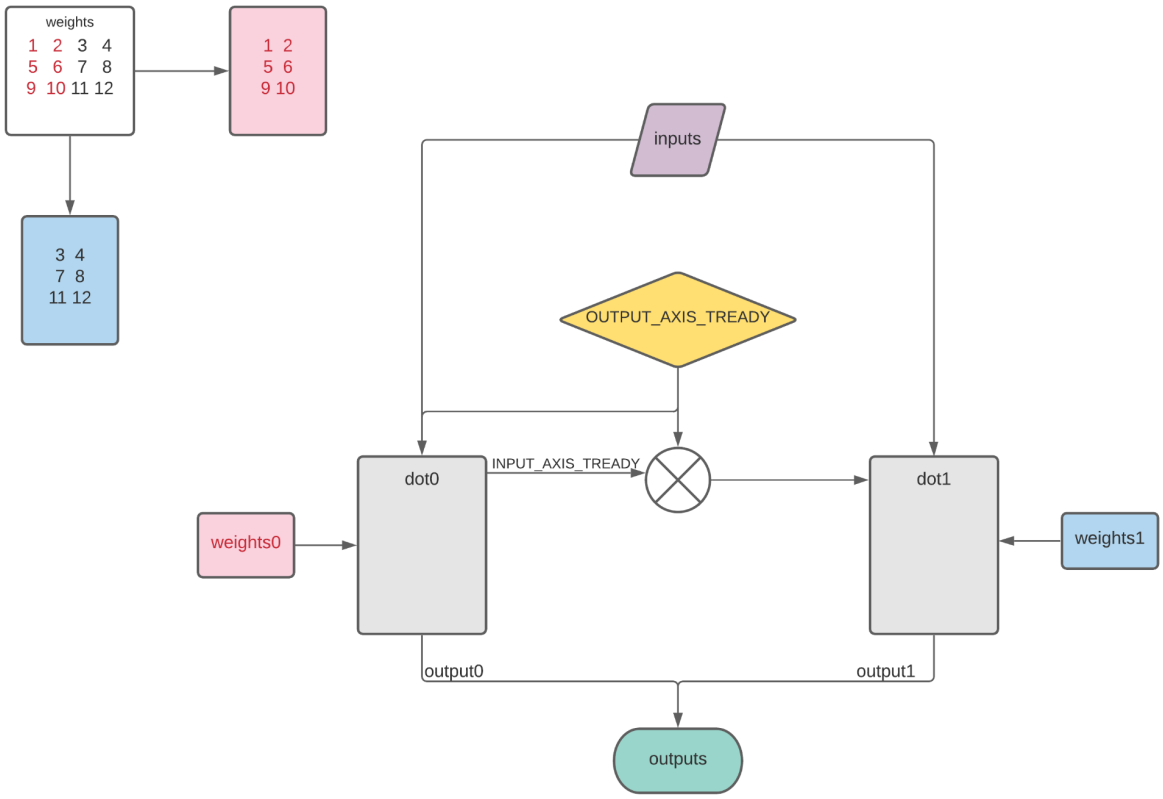

Figure 2. *Flow chart describing attempt 2, "Simplicity is Beautiful".*

Delegation of tasks

The majority of the changes present in the `accel_dot.sv` code and all of the changes in the `dot.sv` code were performed by [redacted]; however, some parallelization groundwork on the code was laid by [redacted], as well as some general peer-review.

The majority of the final report was written by [redacted], as well as both of the figures; however, some of the coding explanation was written by [redacted] to provide insight, as well as some general peer-review.

Project 7 Final Report

Conclusion

After optimizations, the program took 190 (compared to 2210 before optimization) cycles to finish the 20 x 10 weight matrix testbench. The group was able to reduce the cycle count by using pipelining and parallelization in concurrence, which resulted in an overall speedup 1163%. Apart from the general challenges inherent in optimizing code, the main challenges faced during this assignment included the general give-and-take of group work and the operation under incorrectly-assumed data. The former two challenges were resolved with healthy communication.

Unfortunately, the group was unable to meet the goal of 150 cycles, and it was decided that there was insufficient time to switch to the horizontal slicing, which would have been needed to get under 150 cycles. This could be solved in the future via more careful analysis and planning during the initial stages.