Fedrik Durao
109302962
ESE 507 Project 3

1. It was quite easy to make the hardware generator flexible for handling arbitrary values of M, N, and T. These were simply parameterized in C, and all arithmetic related to these parameters were placed where needed. Although having values of T larger than 32 bits resulted in problems during saturation. This is because constants are assumed to be 32 bit in Verilog if not specified. Going forward, all constants have specified bit lengths. Making the program flexible for different values of P was quite difficult. Control logic and number of modules had to change with P. Since the speed of convolution would increase with P, the control logic has to account for this and manage how each convolution module receives data from memory. Having no maximum defined values for M, N, and T were fine. And since the assignment states that L/P values will be integers, maximum values for P were not considered (P > L not considered). There are practical limits on the T value though. Having extremely high values for T will result in very large arithmetic execution time, thus increasing the critical path considerably.
2. The structure of the control logic does not change at all from varying values of M and N. Rather, instances where M and N are used (port instantiations, conditional statements, and counters) the values would change. Designs with varying N and M are identical, only having different constant values where M and N are considered. The values that need to be changed are calculated in C++ and inserted where necessary.
3. For P > 1, the control logic would change a little bit. Whereas the control logic used counters to determine how long to wait before giving the accumulators new data, the counters needed to be divided by P (and be ceilinged). This is because P convolutions are happening at once, so P times more data needs to be given to the accumulators at once (otherwise the gain from parallelism is moot). The data path had to change considerably with P. P number of accumulators and memories (Xmem and Ymem) would be instantiated. Since there are P outputs, control logic was added so that the previously incrementing output memory address would remain the same, but each output was multiplexed according to the value of the incrementing address %P. Meaning that as the

address count increments, another output memory is used to produce output to the master-slave connection. Memory sizes should have been reduced since each memory will be holding less values as P increases. But this would require additional changes to control logic.

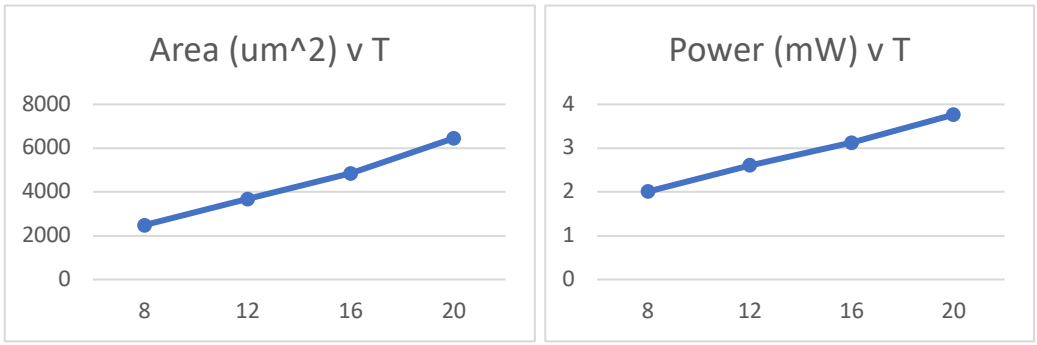
4.

a. N, M and T determine how big of a convolution can be computed. T determines how large the values that are being calculated can be, while N determines how many values will be convolved (as does M). Although if M increases, the number of convolutions decreases.
b. N, T, and P affect cost. If N increases, then memory sizes will be larger. If M increases, then input memories size increase, but then output memories size decrease, therefore not net effect is made by changing M. As T increases, combinational logic increases, as does all memory sizes. As P increases, memory and amount of combinational logic increases (although optimized memories would not increase). As these factors increase with N, T, and P, so does area and power. Energy increases with N and T as well, but as P increases, the time for solving convolutions decreases, resulting in either a minimal increase (due to increasing P) or decrease in energy depending on how effective P is.
c. Only T affects the precision of the system. Naturally as T increases, the number of bits on the output increases as well. Although, **if** we did not have saturating logic then large Ns and small Ms (large Ls) would increase precision since more bits would have to be on the output as to avoid overflow when making L accumulations.
d. Latency increases with T and M. T will increase the critical path in the combinational logic, resulting in low frequencies. M will increase the amount of multiply-additions being made, thus making the calculation of 1 convolution take longer (despite decreasing amount of convolutions). P and N do not affect latency of convolutions (although increasing N will increase the latency of finishing the whole set of convolutions). P increases throughput since more convolutions are happening per clock. M does not affect throughput because despite increasing latency, the throughput remains nearly the same since the amount of convolutions

decrease with M. Therefore M increases convolution time but increases number of convolutions, suggesting no net effect. T decreases throughput for the same reason latency increases with T. N has no direct effect on throughput.

5.

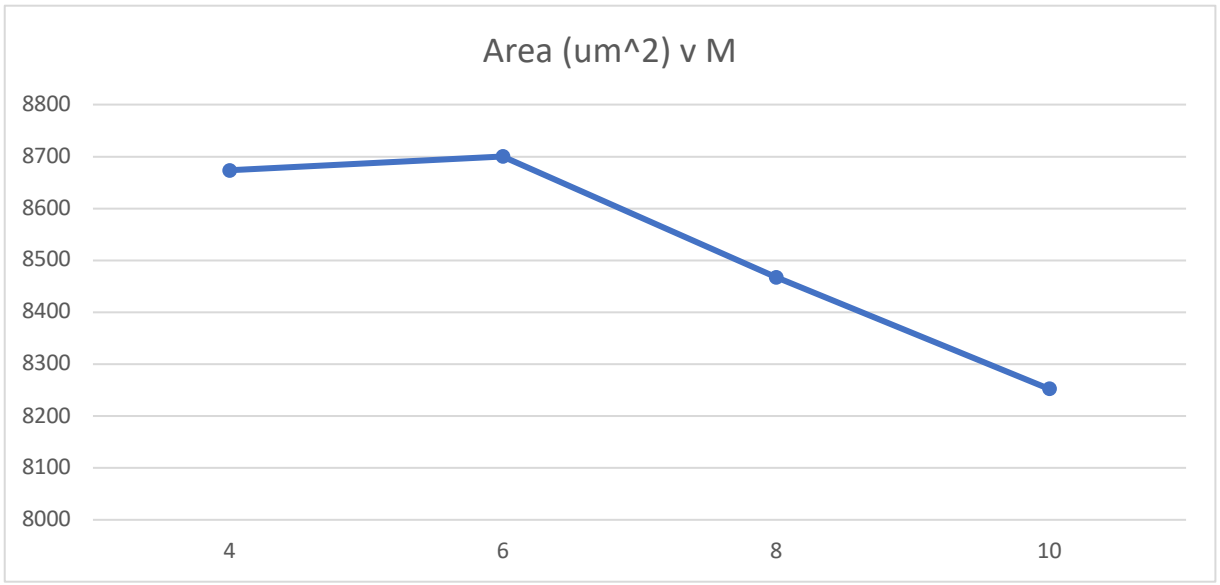
a.



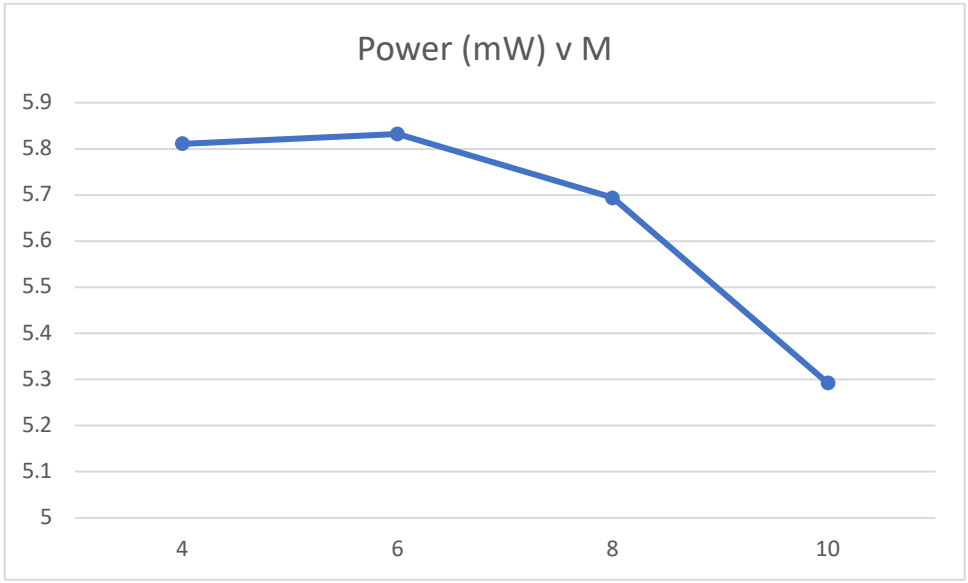
b. The critical path for T = 8 is from data coming from memX (stored X inputs) going through a multiplier and into a pipeline that stores the product. For all other T, the critical path was the same except it started from data leaving the ROM as opposed to memX.

6.

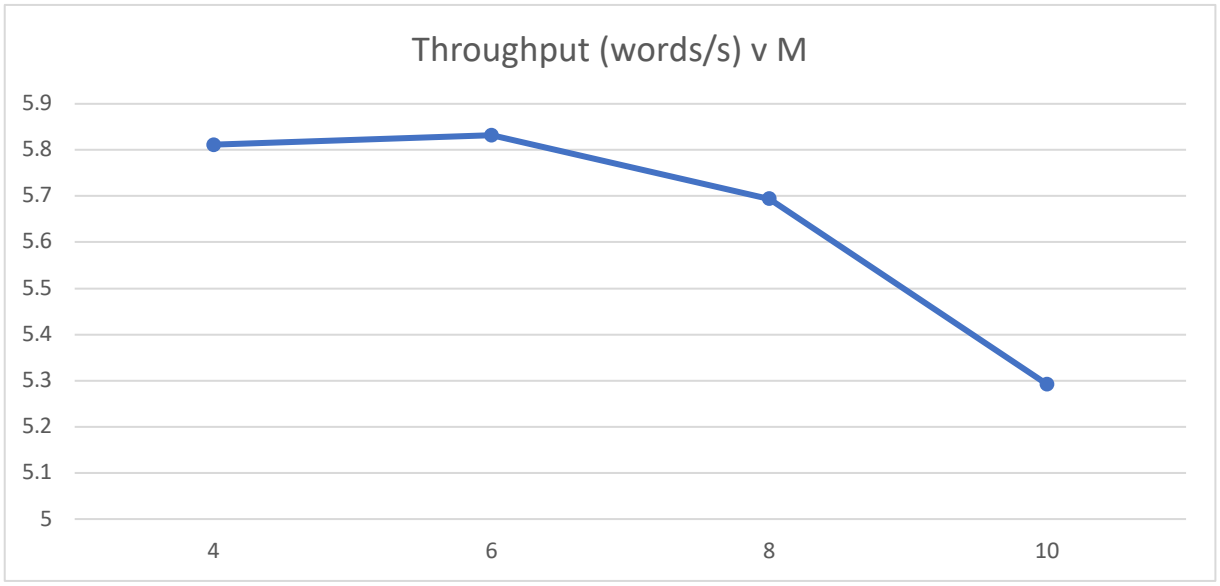
a.



b.



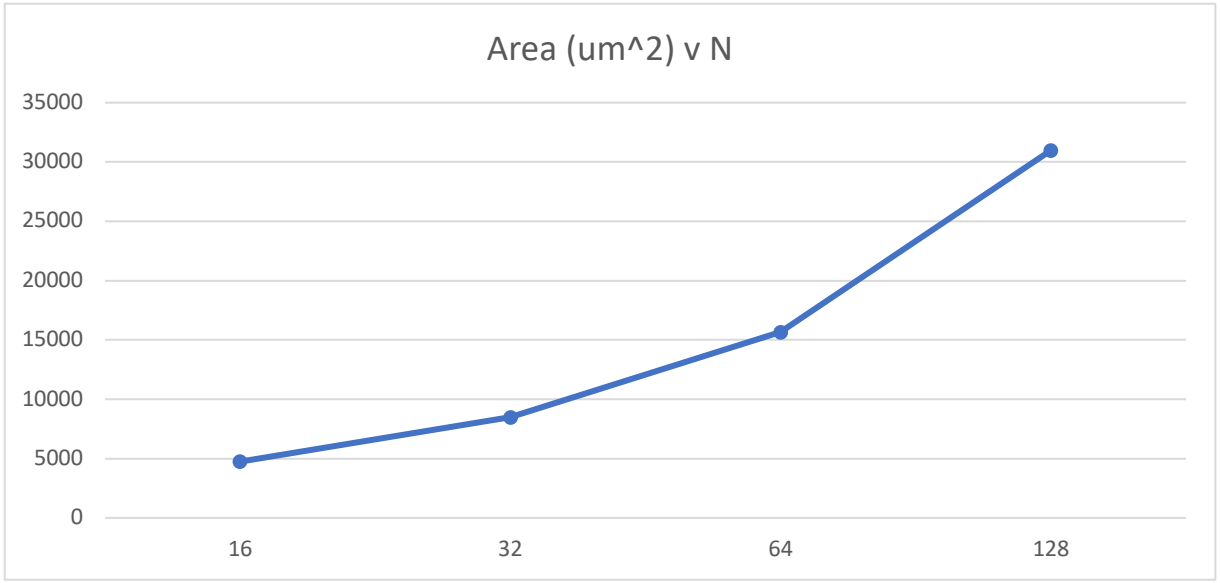
c.



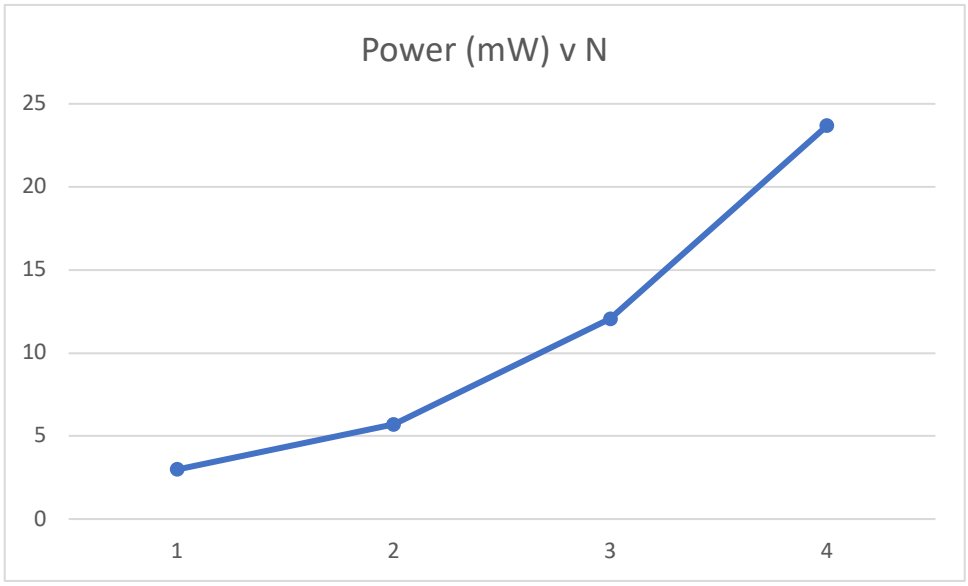
d. The critical path does seem to change with M. For M = 4, M = 6, and M =10 the critical path was from data coming from the rom, through the multiplier, and then into the product pipeline. For M = 8, the crtical path changed, starting at the product pipeline, through the accumulator, into the accumulator register f.

7.

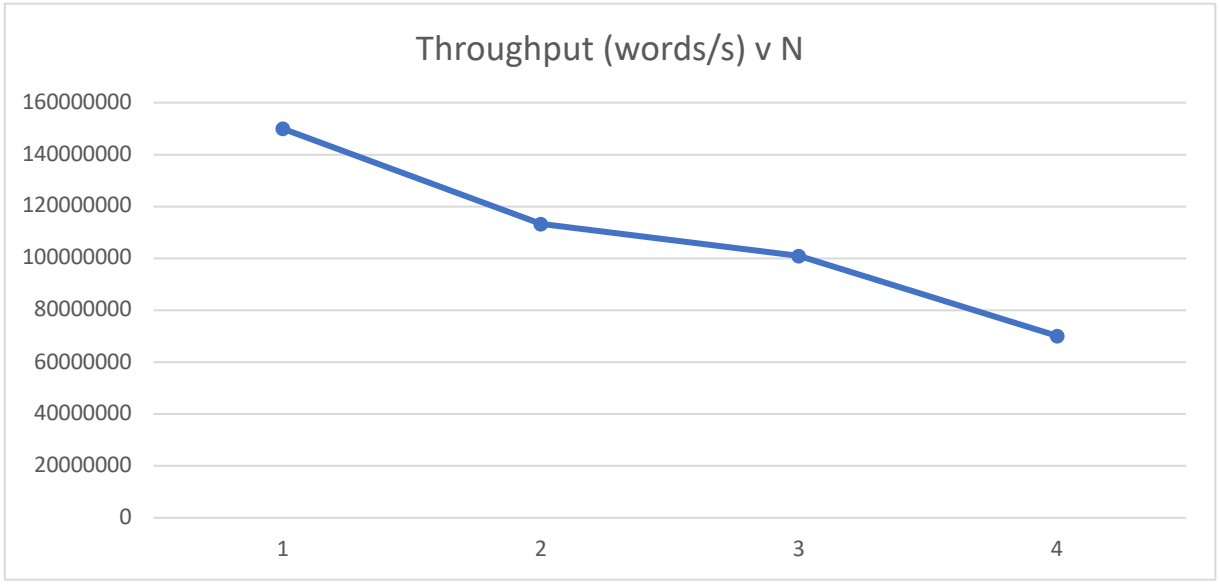
a.


Area (um^2) v N
35000
30000
25000
20000
15000
10000
5000
0
16
32
64
128

b.


Power (mW) v N
25
20
15
10
5
0
1
2
3
4

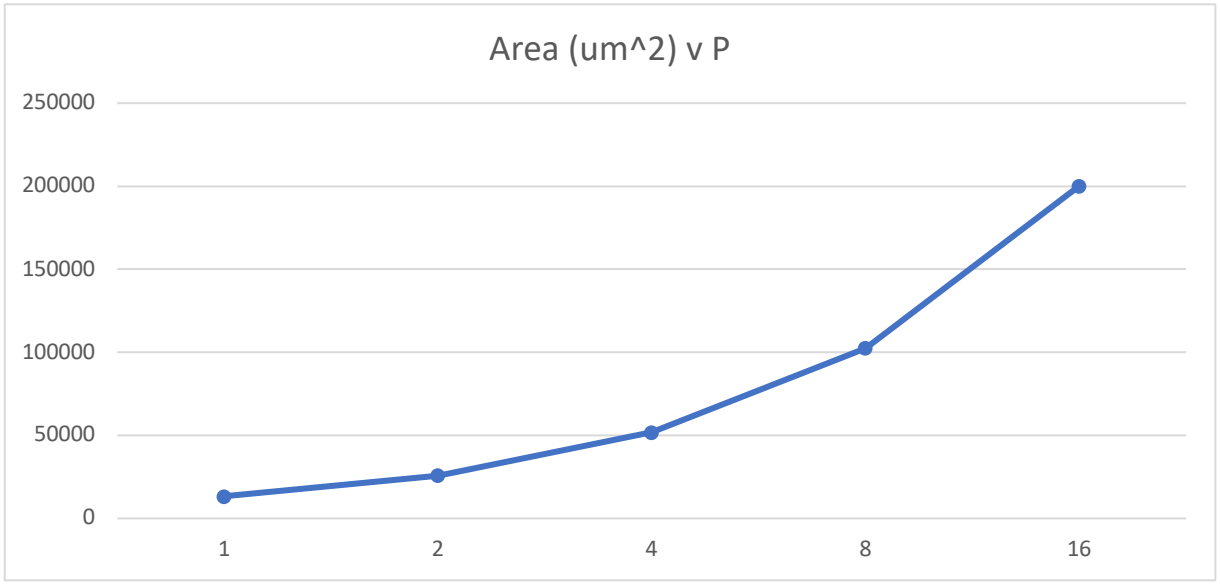
c.



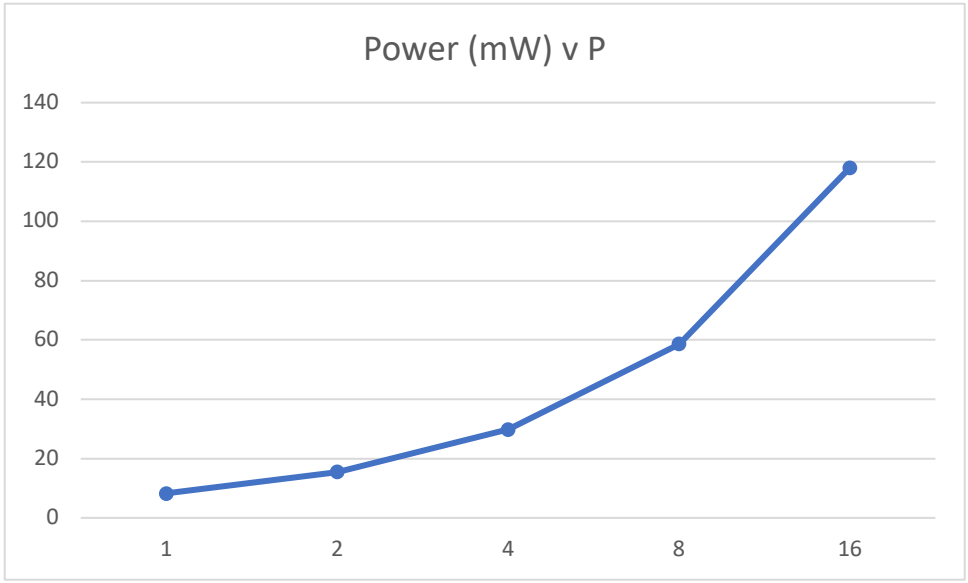
d. The critical path changes even more often with change in N. For N = 16, and N = 64, the critical path started from the rom and ended at the product pipeline once again. For N = 32, the critical path started from the product pipeline and ended at the accumulator register f. At N = 128, a new critical path was established, starting at the output of accumulator register f, through the feedback accumulator, back into register f.

8.

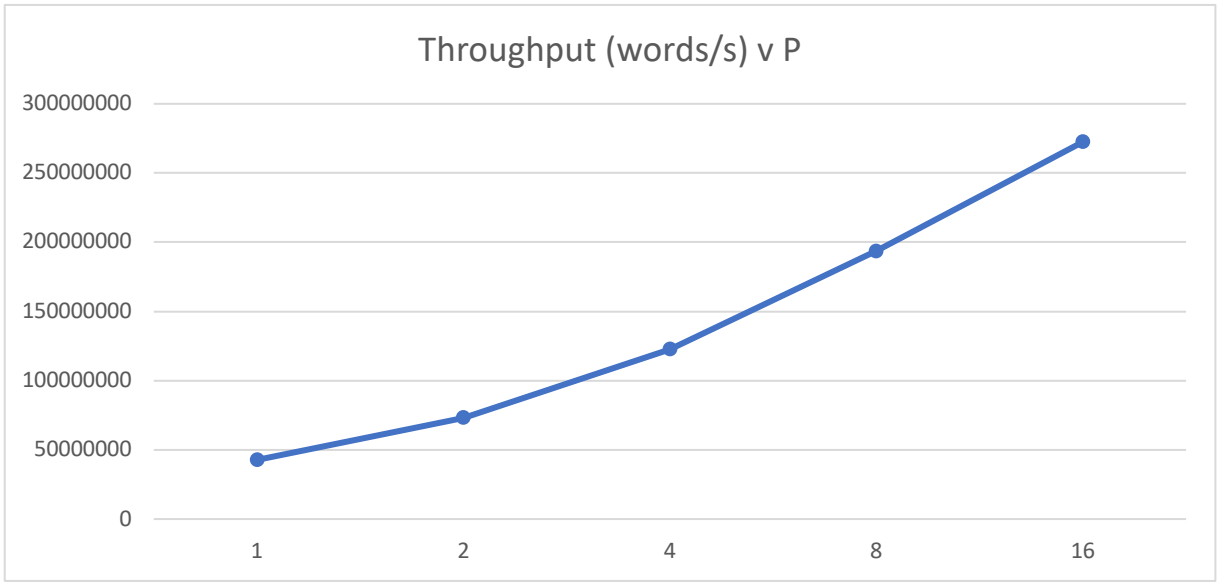
a.



b.



c.



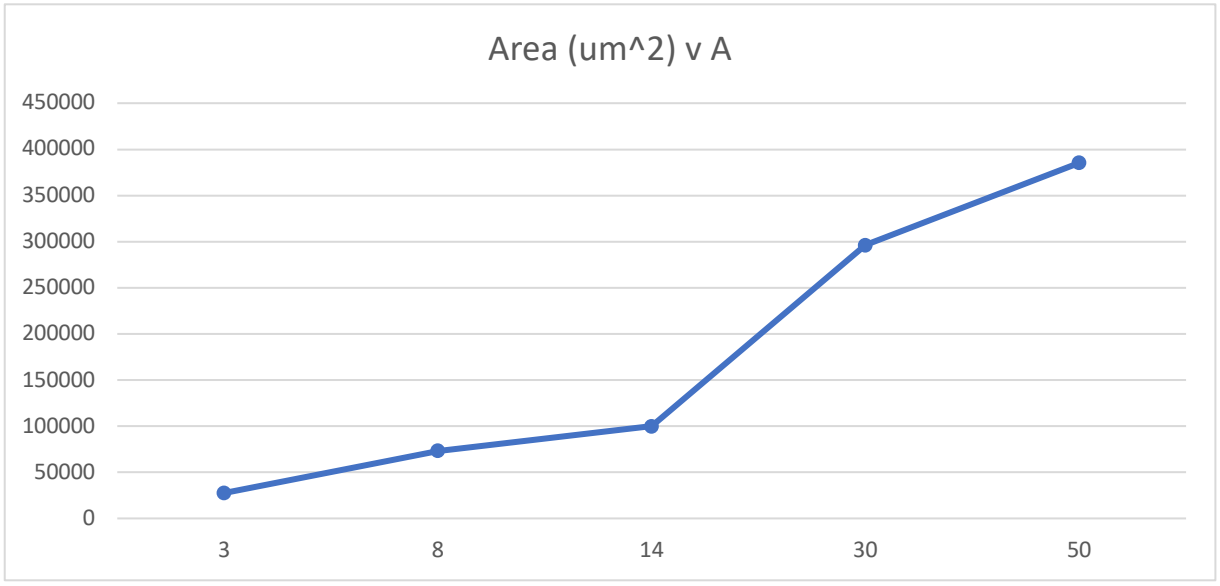
d. The first design with P = 1 is the most efficient. In this context, words/J would represent efficiency. This can be found by diving throughput by the power of the design. The design for P = 1 had the best calculated efficiency which suggests that cost of increase power outweighs the increase in throughput with increasing P.

9. To parallelize further, instead of performing M multiplication-sums per parallel block, the multiplications can be parallelized too. Meaning that inside each parallel block (up to L blocks), there can be up to M parallel multipliers and 1 adder (with high fan in).
10. For best optimization, the slowest layer at any instance would have its P incremented (by a value where L/P is and integer), until there is no budget left or if P is maxed out for

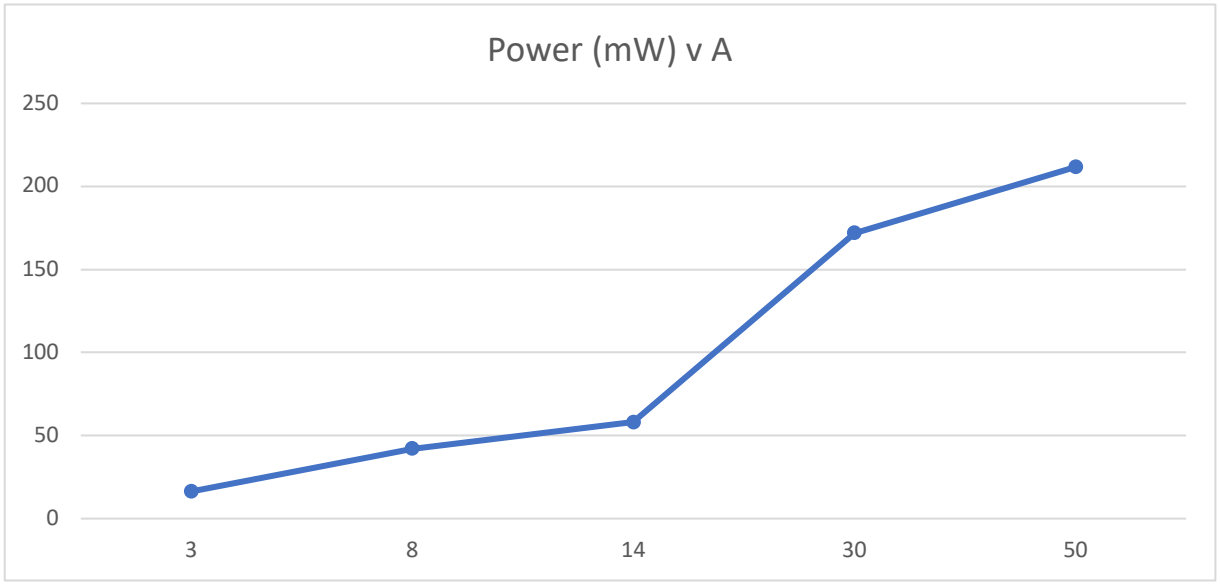
each layer. This was done by making a while loop checking if the budget is zero. If not then the loop continues by taking L/P for each layer, picking the largest (the slowest) and then adding the smallest value to P (where L/P is an integer), and then subtracting the addition from the budget. Also, if L/P = 1 for all layers, then the loop is exited. There is no drawbacks to this approach. This approach ensures that the weakest layer in all 3 layers is optimized, after every optimization. The finishing time of the testbench simulations helped evaluate whether these optimizations were effective.

11.

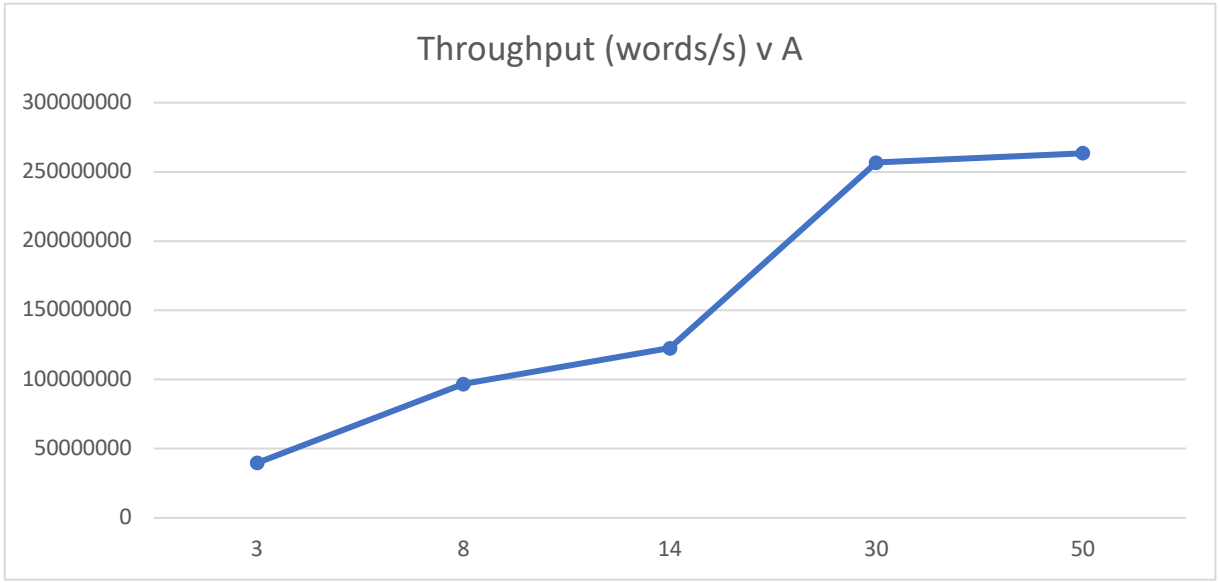
a.



b.



c.



12. Not much would have to change in order to support an arbitrary number of layers. Gen_layer will have to be called in a loop, and the modules connecting each layer would have to be made in a loop as well. The optimization logic would grow in proportion to how many layers there are, and sufficient M values will need to be given.

*Note that the throughputs were based off of calculated cycle values, not measured cycle values.