Report

For

Project 3

By

Aishwarya Gandhi

(111424452)

&

Kewal Raul

(111688087)

Q1.

Our Hardware generator very well meets the scalability & flexibility requirements. The variable sizes of Matrix are handled by using parameters M & N for any M x N Matrix. The entire code is written in terms of M & N so accordingly the generator generates for any given M x N matrix. Parallelism is also handled in terms of M & N while number of bits is handled by a T parameter. Generalizing for variable sizes of M x N & size of bits was quite easy whereas generalizing for any given P was quite tricky but manageable. In our design M & N are defined as parameter datatype of SystemVerilog so they can take any value up to 2^{16}

Q2.

Basically, our design has 2 states: The computation state & Non- computing state. In the non-computing state (Computing=0 / s_ready=1) the system reads input values for vector X while the MAC computation happens in in the computation state (Computing=1 / s_ready=0). The number of input values are dependent on the column size of the matrix (N) so reading of inputs is handled in terms of parameter N. So, in case of parallelism 1 now the total elements to be traversed becomes M*N instead of M^2. While computing the output of each row an output_counter is used which is defined in terms of N.

Q3.

For P>1 we instantiated multiple W_rom & B_rom P times instead splitting the rom itself. So now we have multiple address signals & data signals to access the data from the rom simultaneously. We added some extra logic to our generator to generate the address control for any M x N matrix with any possible P. So, depending on the parallelism the address pointer of each MAC unit controlled in terms of N is traversed through the respective rows of the matrix. In datapath, all the outputs of each MAC units arrive simultaneously, but as data_out is serial in nature all the outputs must be serialized. Thereby, we used a counter 'dataOutCounts' which depending upon the number of MACs keeps a track of pushing the output of each MAC to data_out when m_ready is high & also keeping m_valid asserted until the last MAC output is pushed to data_out.

Q4.

As M & N increase the problem size increases.

As the problem size i.e the size of M & N increases the cost such as power, area, energy increase. Cost also increases as P increases or T increases.

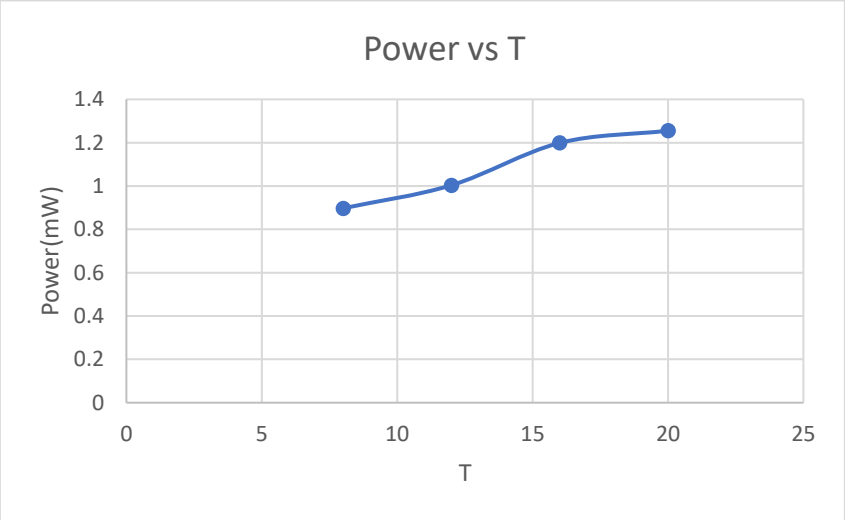
Precision is set by the T parameter which denotes the number of bits used. So precision increases with increase in T.

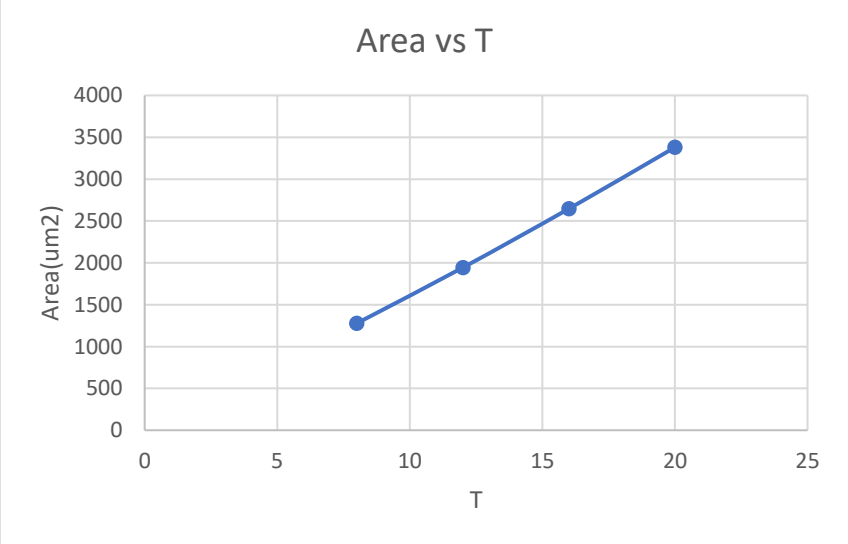
Latency increases with increase in problem size (size of M & N). While it decreases with increase in parallelism which given by parameter P. T has no effect on Latency

Throughput increases as P or N increases. Throughput decreases with increase in M while T has no effect on Throughput.

Q5.

T	Area (μm^2)	Power(mW)	Critical Path
8	1277.86	0.89712	dpth/mult2_out1_reg[0]] to dpth/data_out1_reg[2]
12	1943.12	1.0035	dpth/mem_x/data_out1_reg[0] to dpth/mult2_out1_reg[11]
16	2646.69	1.1988	dpth/data_out1_reg[0] to dpth/data_out1_reg[0]
20	3380.32	1.2552	dpth/data_out1_reg[0] to dpth/data_out1_reg[0]

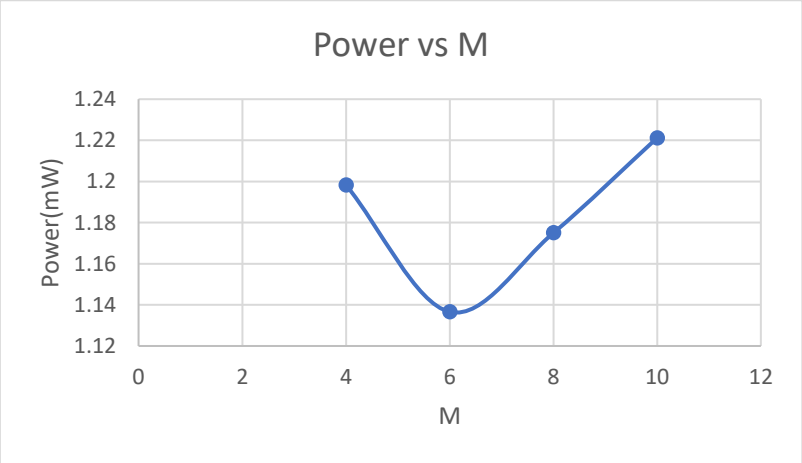


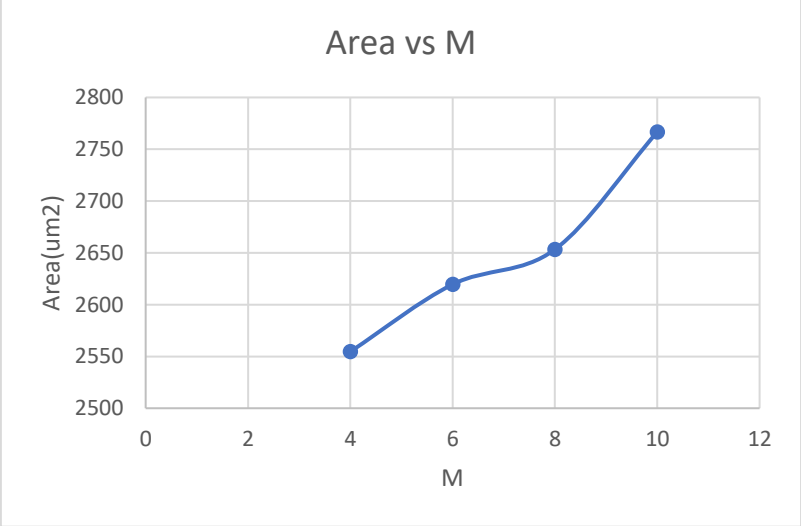


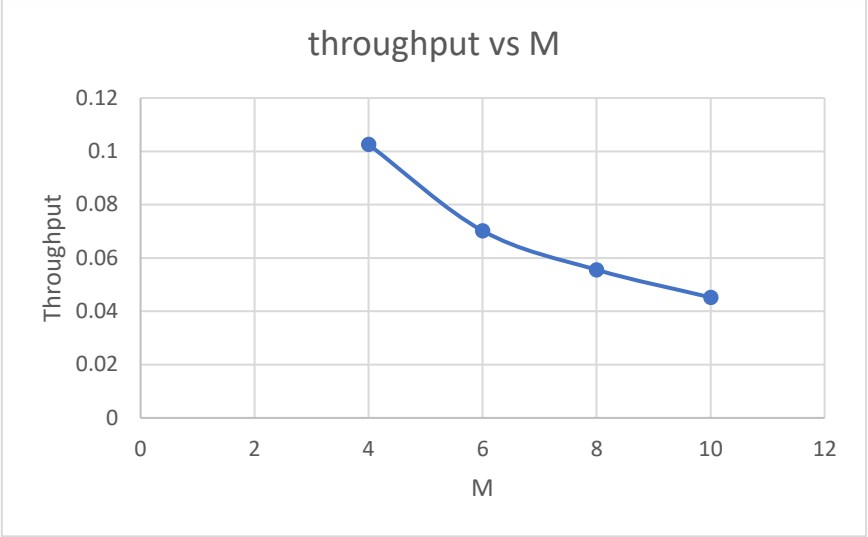
The critical path changes for different values of M.

Q6.

M	Area (μm^2)	Power(mW)	No. of cycles (c)	Throughput (words/s)	Critical Path
4	2554.66	1.1983	52	0.102564103	dpth/mem_x/data_out_reg[1] to dpth/mult2_out1_reg[15]
6	2619.56	1.1366	74	0.07020007	dpth/mem_x/data_out_reg[3] to dpth/mult2_out1_reg[15]
8	2653.34	1.175	96	0.055555556	dpth/data_out1_reg[0] to dpth/data_out1_reg[0]
10	2766.66	1.2212	118	0.04519774	dpth/data_out1_reg[0] to dpth/data_out1_reg[0]



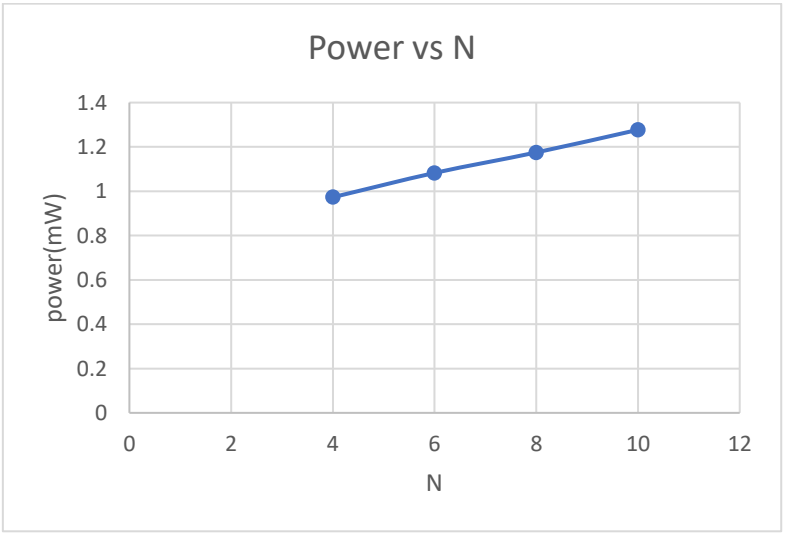


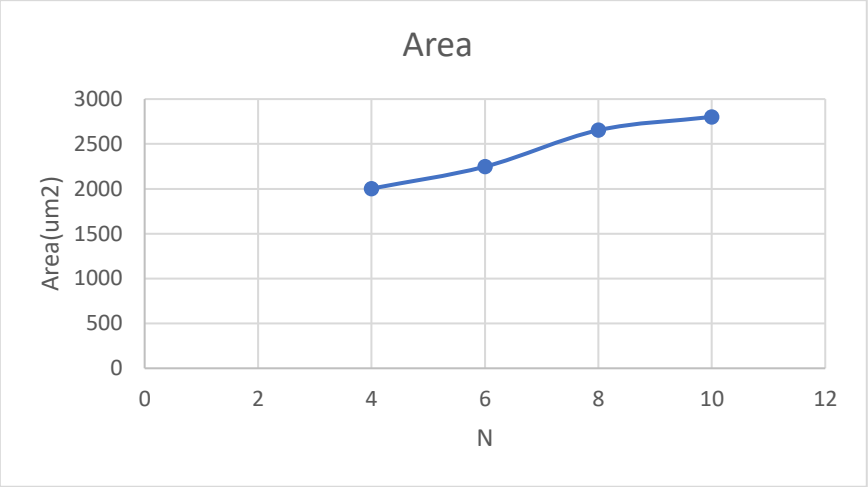


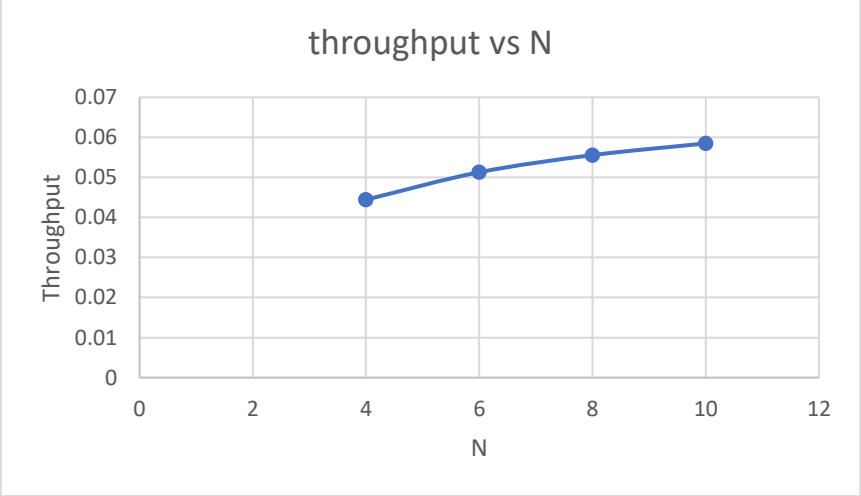
The critical path changes for different values of M.

Q7.

N	Area (µm²)	Power (mW)	No. of cycles (c)	Throughput (words/s)	Critical Path
4	2002.97	0.973	60	0.044444444	dpth/mem_m1/z_reg[1] to dpth/mult2_out1_reg[15]
6	2247.16	1.0826	78	0.051282051	dpth/mem_x/data_out_reg[1] to dpth/mult2_out1_reg[15]
8	2653.34	1.175	96	0.055555556	dpth/data_out1_reg[0] to dpth/data_out1_reg[0]
10	2801.77	1.2771	114	0.058479532	dpth/mem_x/data_out_reg[1] to dpth/mult2_out1_reg[15]



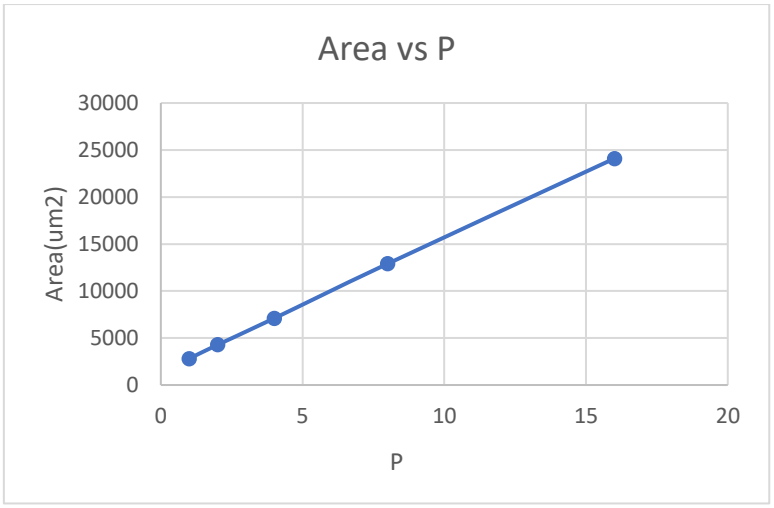


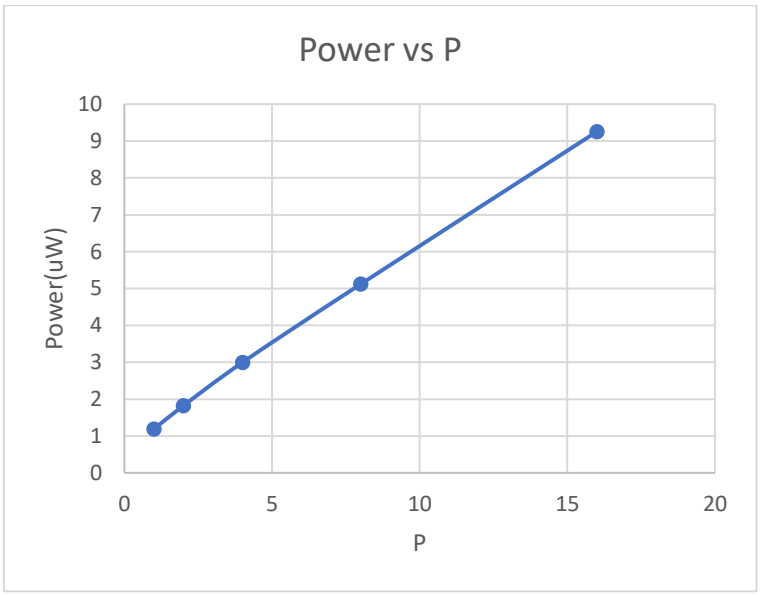


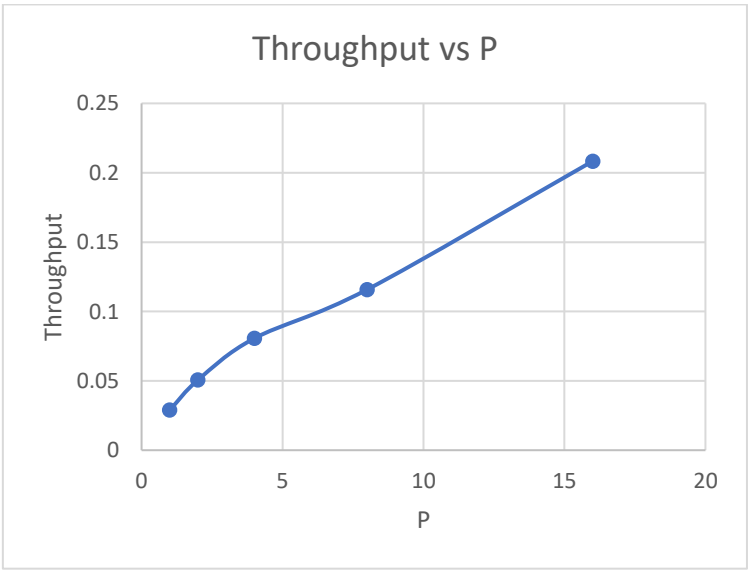
The critical path changes for different values of N.

Q8.

P	Area (μm²)	Power (mW)	No. of cycles (c)	Throughput (words/s)	Area/Throughput	Power/Throughput
1	2806.56	1.1868	184	0.028985507	96826.32082	40.94460035
2	4275.15	1.8223	104	0.050607287	84476.96475	36.00864832
4	7095.01	2.9954	64	0.080645161	87978.12432	37.14296013
8	12900.2	5.1177	44	0.115807759	111393.2271	44.19133955
16	24094	9.2557	24	0.208333333	115651.2002	44.42736007


Area vs P
30000
25000
20000
15000
10000
5000
0
Area(um2)
0
5
10
15
20
P


Power vs P
10
9
8
7
6
5
4
3
2
1
0
Power(uW)
0
5
10
15
20
P


Throughput vs P
0.25
0.2
0.15
0.1
0.05
0
Throughput
0
5
10
15
20
P

As computed above, the ratio of area to throughput and power to throughput was found lowest for P=2. But there was a minor change in the ration for P=4. Thus, it is the most efficient design with high throughput and low area and power.

Q9.

Currently we parallelize the system row-wise, further parallelism can be added to the system by parallelizing column-wise. That is each MAC unit can be composed of mini MAC units which process parallelly. So, the output of each row is the sum of the output of the mini MACs. For example, 4 x 4 matrix say currently has P=4 and we further wish to parallelize it. Currently we have 4 MAC units. Now further parallelizing P=2 column-wise, each MAC layer will have 2 mini MACs.

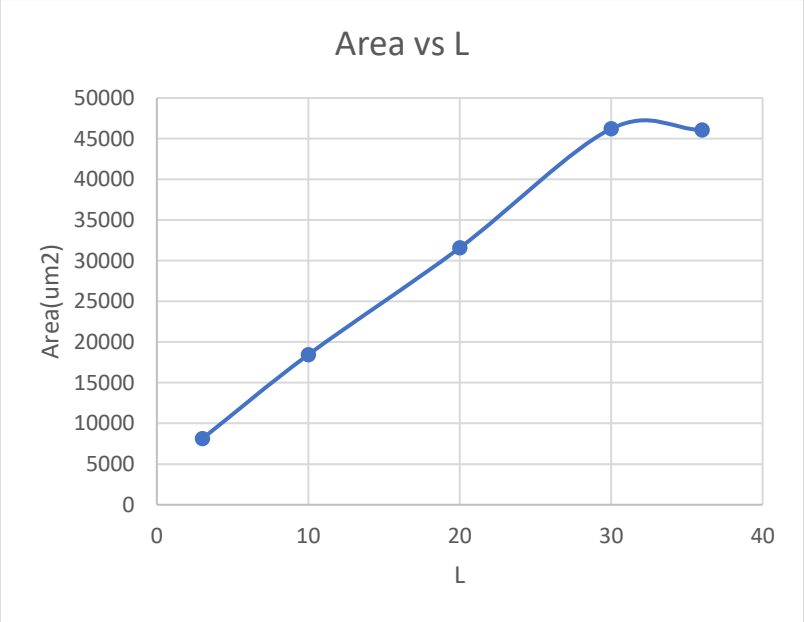
Q10.

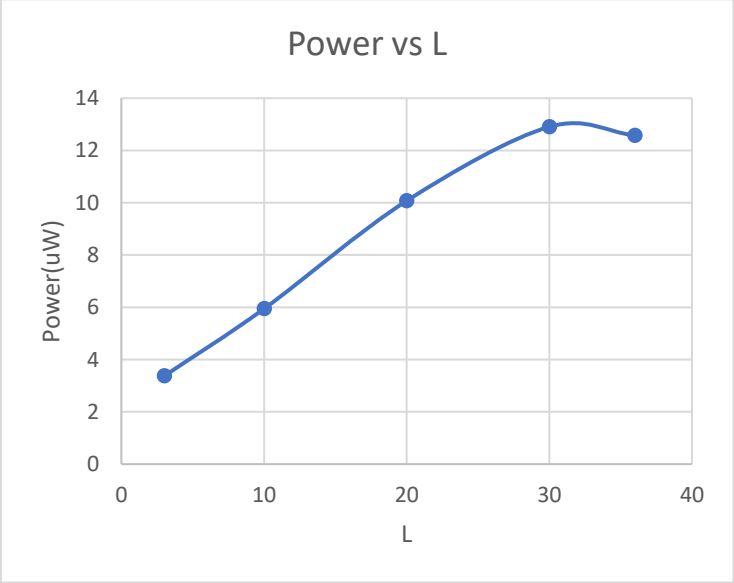
While assigning parallelism to the layers we consider the number of computations for each layer. We basically try to equalize the number of computation of all the layers. We approach this by sorting the layers in descending order and increase the parallelism of each layer to the first achievable P value in the same order. After assigning the first set of P values for each layer, the number of computations for each layer is recomputed. This process is reiterated number of times till L (multiplier budget) is optimally utilized.

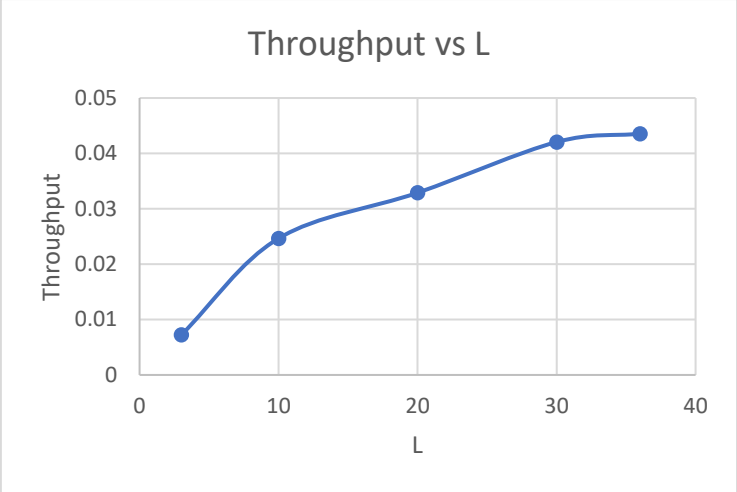
Our method of allocating parallelism at best tries to lower the overall number of computation of the system thereby increasing its throughput. The drawback we see in our approach arises when in case while dealing with prime values of M. In a certain scenario, most of the multiplier budget might be spent in parallelizing the layer with prime value of M. But however, in a few examples we tested by manually allotting values of P & comparing with the one's allocated by the code we didn't find any considerable change in the throughput. Thus, we implemented our code.

Q11.

M	Area (μm^2)	Power (mW)	No. of cycles (c)	Throughput (words/s)
3	8111.66	3.3709	362	0.007222042
10	18443.9	5.9433	104	0.024654832
20	31576.06	10.074	76	0.032894737
30	46215.9	12.9067	58	0.042052145
36	46068.53	12.5779	55	0.043549265


Area vs L
Area(um2)
50000
45000
40000
35000
30000
25000
20000
15000
10000
5000
0
0
10
20
30
40
L


Power vs L
Power(uW)
14
12
10
8
6
4
2
0
0
10
20
30
40
L



Q12.

The generator has enough scope to be modified to take any user defined arbitrary number of layers. The generator calls the genLayer() function is generalized so only multiple calls have to be made to the function depending on the number of layers requested. The major change would be of generalizing the definition of the Top- Module which stitches all the layers together. With these few changes our generator should be able to generate system for any number of layers.