Project-3

1. The control module is designed using an FSM. It has 5 states. RESET, LOAD_X, STALL1, EXEC, STALL. State changes from RESET to LOAD_X upon seeing is a valid input in the input bus. State remains in LOAD_X until N values of vector X is loaded into the memory, MEM_X. Upon load completion, the state moves to STALL1 state. This state helps is used to wait for one cycle so the correct value of MEM_X is available for execution state. In the next cycle the state changes to EXEC and remains here till all the multiply and accumulate operations are completed. After execution the next state which is STALL state is used to output the result of each row and column multiplication result into the output bus. During this state, the system wait for the destination to be ready for reading these results.
   Counter addr_x is used to control the address value passed to the mem x. This counter counts from 0 to N-1. Similarly, Counter addr_w is used to control the address value passed to the ROM containing the weights. This counter counts till ((M*N)/P) -1.

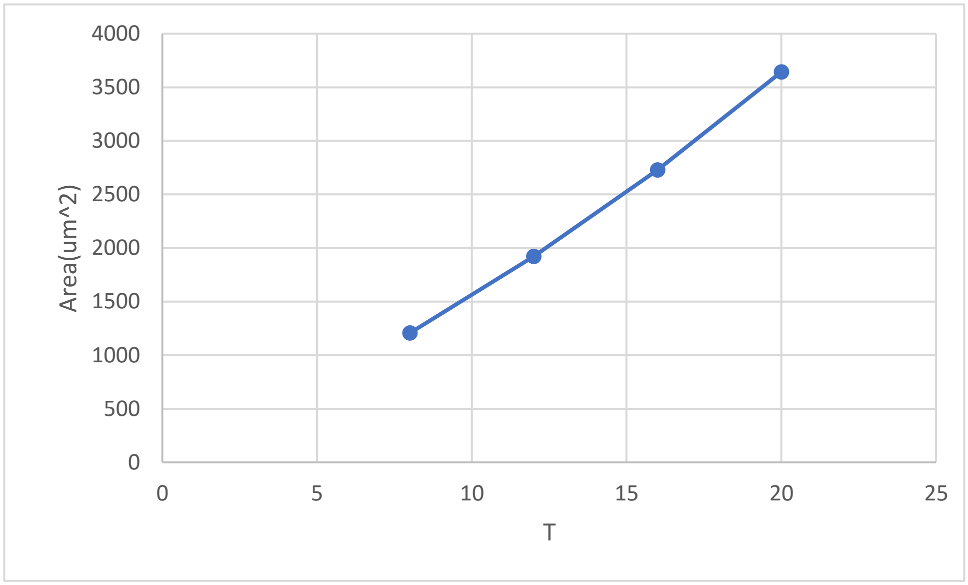
   Parameterized form of the control logic code is stored in the template.sv file. Here we have keywords like "_M_","_N_"and "_P_" which is used to represent the actual M,N and P parameters. These keywords are replaced with the actual M,N and P value during generation of each layer the generator. There is a find and replace function in the generator does this operation of replace the keywords with appropriate values.

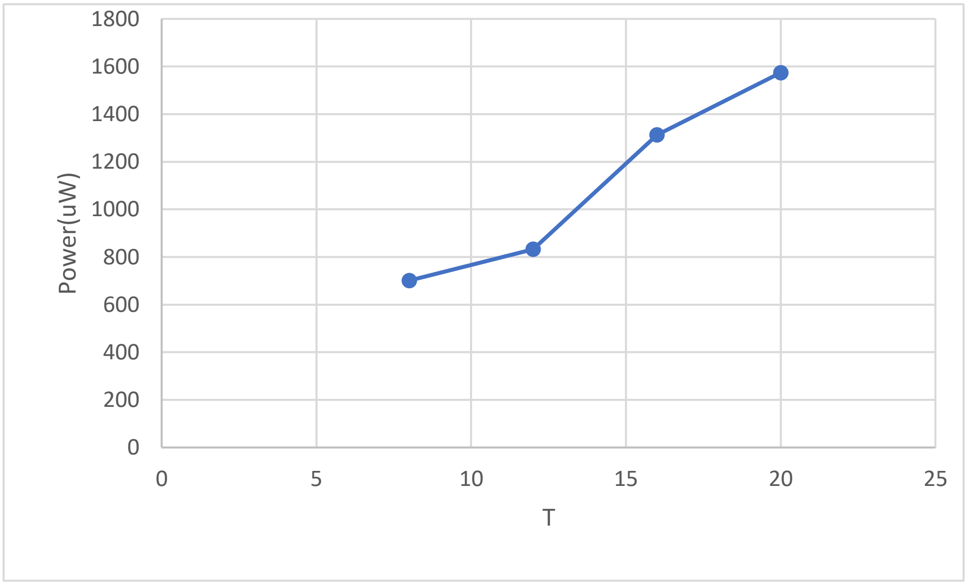
2. For the implementing case where P is greater than 1 we made following changes form part 1
   a. ROM generation: We split the W matrix into P groups. For example, if W is a 4X4 matrix, and P =2. Then there will be 2 ROM generated. First ROM will have the Row 1 and row 3 are stored in ROM1 and Row 2 and Row 4 are stored in ROM2. If P=4, 4 ROM containing each rows of W's in order is generated. If P=1, a single ROM containing all values of W is generated.
   b. The counter addr_x will count till N-1 just like part one. But counter addr_w now will count only till ((M*N)/P)-1 value to accommodate iterating through each ROM based on the P value.

c. Output counter: We use an additional counter to output logic. This counter helps to output the parallelly generated result one at a time.

d. Data path instantiation: P number of data paths are instantiated in the top module using generate keyword and select parameter is passed during each data path instantiation so that each data path uses its corresponding ROMs.

3.





Critical path:

a. T =8
   Startpoint: data_name[0].data/genblk1.mem_w/z_0_reg[1]
   (rising edge-triggered flip-flop clocked by clk)
   Endpoint: data_name[0].data/mult_x_r_w_r_delay_reg[0]
   (rising edge-triggered flip-flop clocked by clk)

   Here the critical path is from ROM's 1st output bit to 0th output bit of the register between multiplier and adder .

b. T = 12

   Startpoint: data_name[0].data/genblk1.mem_w/z_0_reg[4]

   (rising edge-triggered flip-flop clocked by clk)

   Endpoint: data_name[0].data/mult_x_r_w_r_delay_reg[0]

   (rising edge-triggered flip-flop clocked by clk)

   Here the critical path is from ROM's 4th output bit to 0th output bit of the register between multiplier and adder .

c. T = 16

   Startpoint: mem_x/data_out_reg[13]

   (rising edge-triggered flip-flop clocked by clk)

   Endpoint: data_name[0].data/mult_x_r_w_r_delay_reg[1]

   (rising edge-triggered flip-flop clocked by clk)

   Here the critical path is from Mem X's 13th output bit to 1st output bit of the register between multiplier and adder.

d. T = 20

   Startpoint: data_name[0].data/genblk1.mem_w/z_0_reg[5]

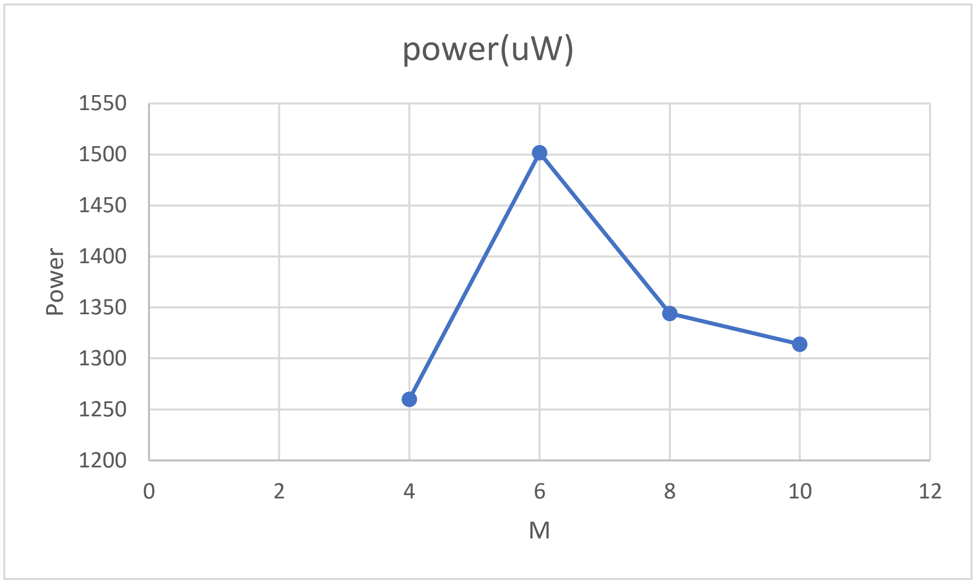
   (rising edge-triggered flip-flop clocked by clk)

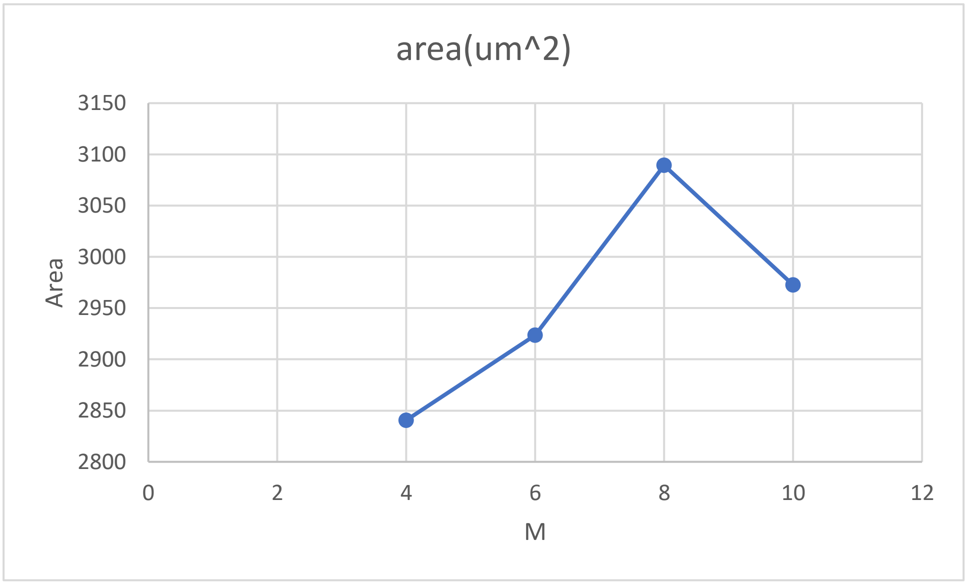
   Endpoint: data_name[0].data/mult_x_r_w_r_delay_reg[0]

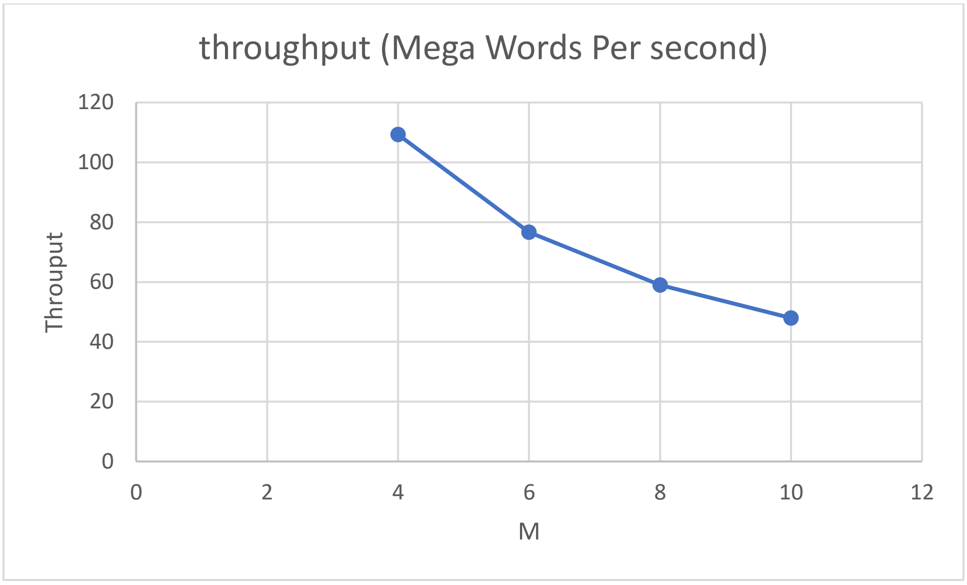
   (rising edge-triggered flip-flop clocked by clk)

Here the critical path is from ROM's 5th output bit to 0^{th} output bit of the register between multiplier and adder.

4.





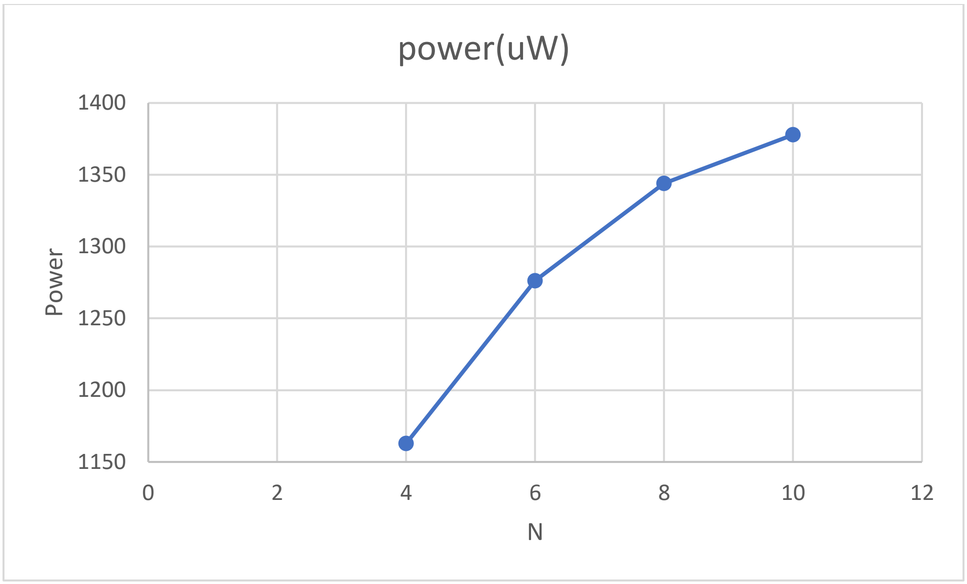


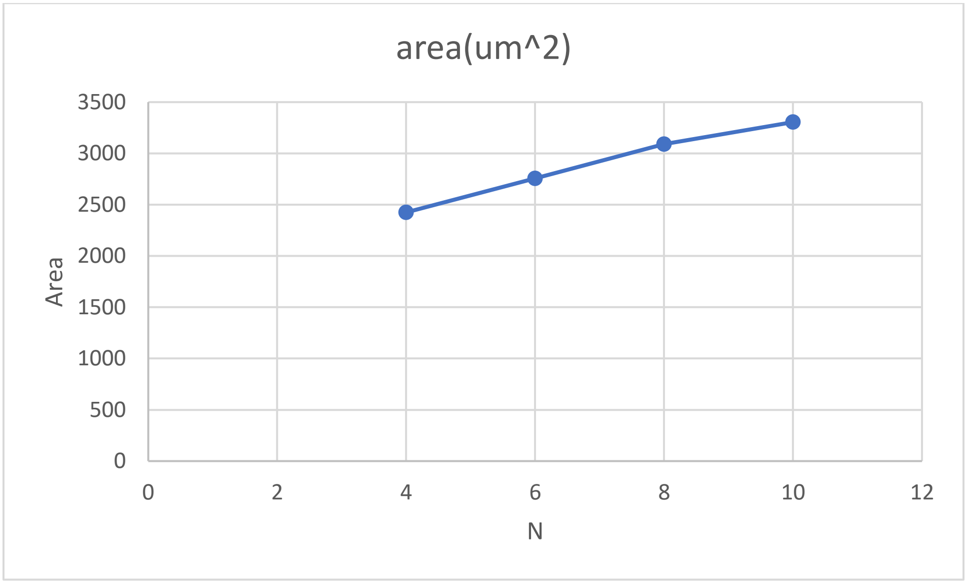
Yes, the critical path changes.
For M=4, Here the critical path is from ROM's 9th output bit to 15th output bit of the register between multiplier and adder.
For M=6, Here the critical path is from Mem X's 3rd output bit to 5th output bit of the register between multiplier and adder.
For M=8, Here the critical path is from ROM's 3rd output bit to 3rd output bit of the register between multiplier and adder.
For M=10, Here the critical path is from ROM's 11th output bit to 2nd output bit of the register between multiplier and adder.

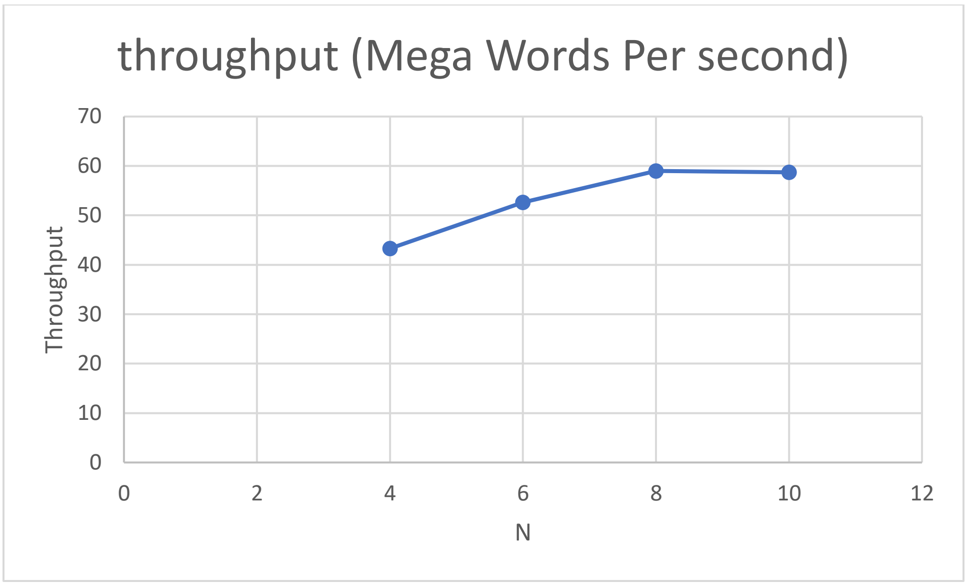
M	C
4	61
6	87
8	113
10	139

Here, C = ((N+1) + (N+3+2*P)M/P) is the formula used to calculate number of cycles. Where (N+1) is the number of cycles to load x vector to memory and (N+3+2*P) is the number of cycles taken to multiply and accumulate one row and column. (N+3+2*P) repeats for M/P times. We have verified this formula with the actual cycles from the simulation wave forms.

5.


power(uW)
Power
1400
1350
1300
1250
1200
1150
0
2
4
6
8
10
12
N


area(um^2)
Area
3500
3000
2500
2000
1500
1000
500
0
0
2
4
6
8
10
12
N



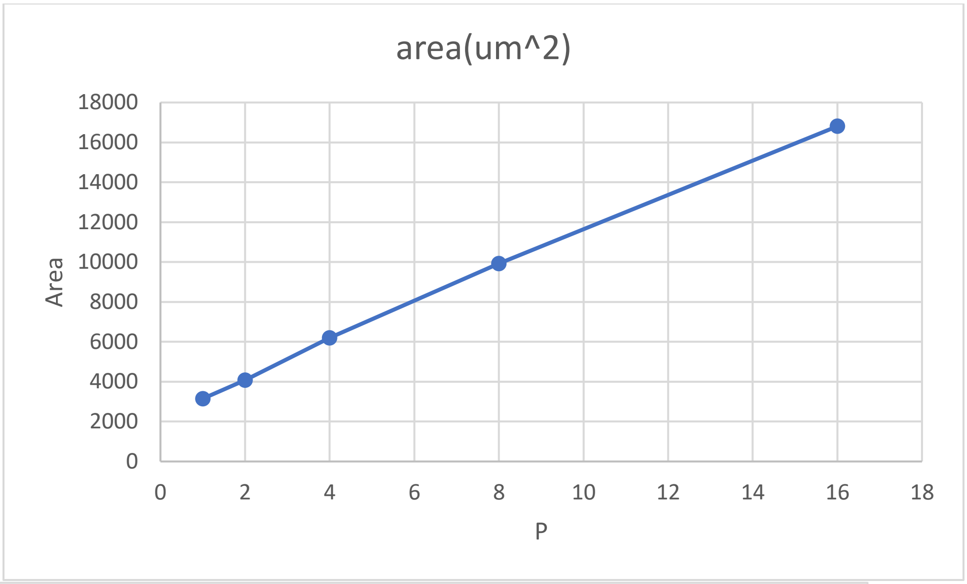
N	c
4	77
6	95
8	113
10	131

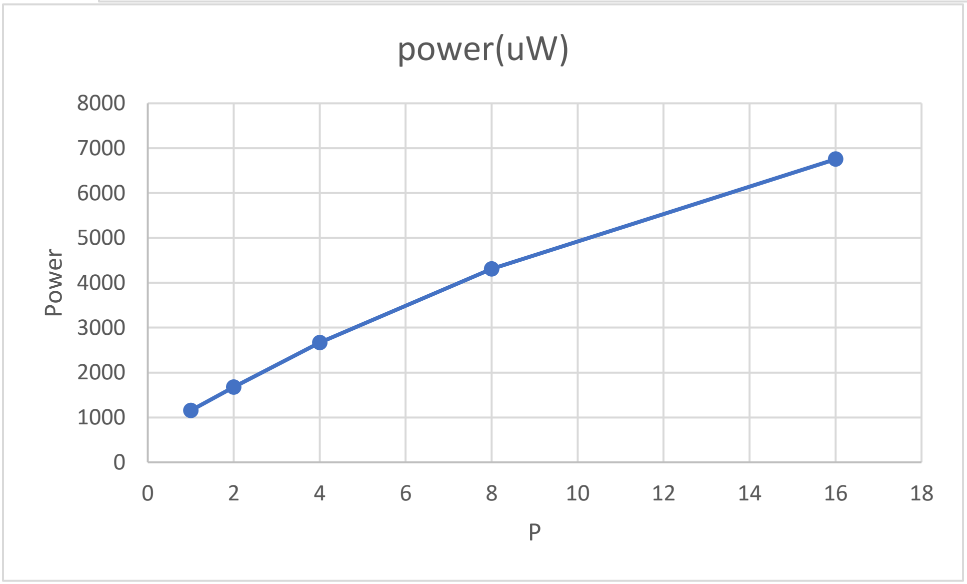
Like explained before c here is calculated using the formula, C = ((N+1) + (N+3+2*P)M/P).

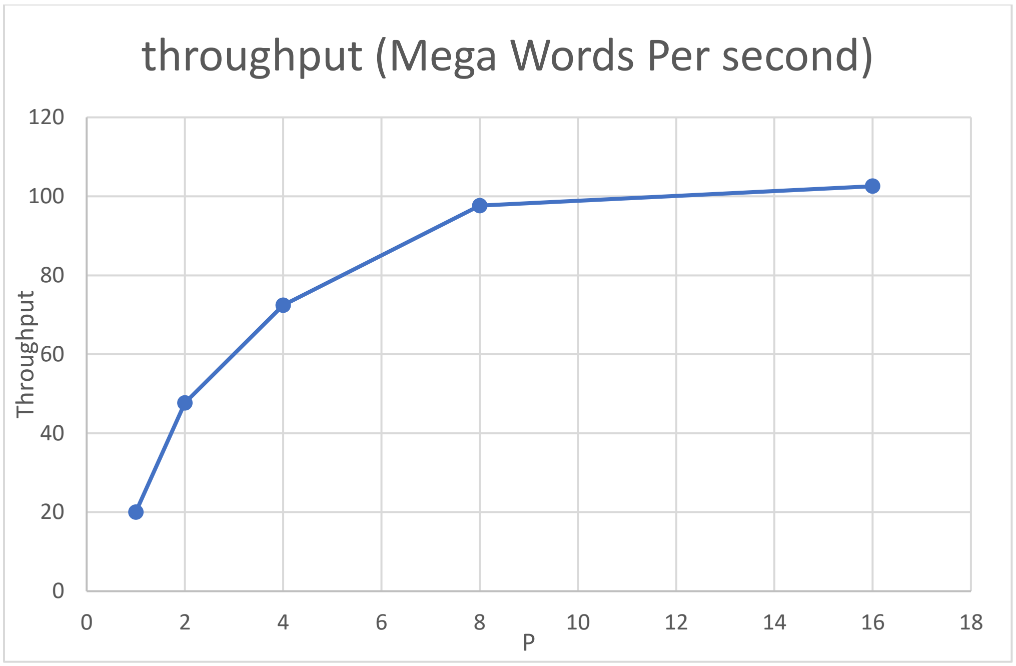
Yes, the critical path changes.
For N=4, Here the critical path is from ROM's 9th output bit to 0th output bit of the register between multiplier and adder.
For N=6, Here the critical path is from ROM's 7th output bit to 15th output bit of the register between multiplier and adder.
For N=8, Here the critical path is from ROM's 3rd output bit to 3rd output bit of the register between multiplier and adder.

For N=10, Here the critical path is from ROM's 13th output bit to 3rd output bit of the register between multiplier and adder.

6.


area(um^2)
Area
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
0
2
4
6
8
10
12
14
16
18
P

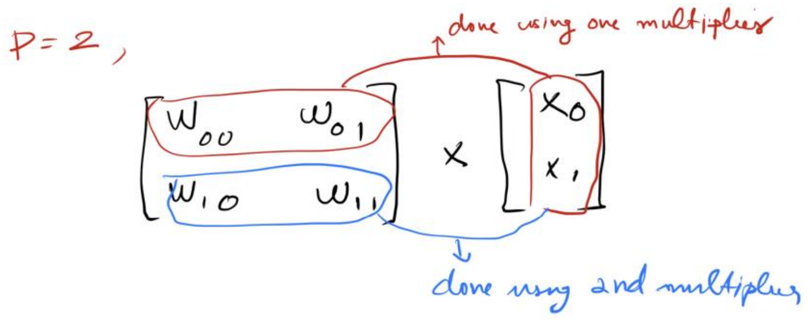

power(uW)
Power
8000
7000
6000
5000
4000
3000
2000
1000
0
0
2
4
6
8
10
12
14
16
18
P



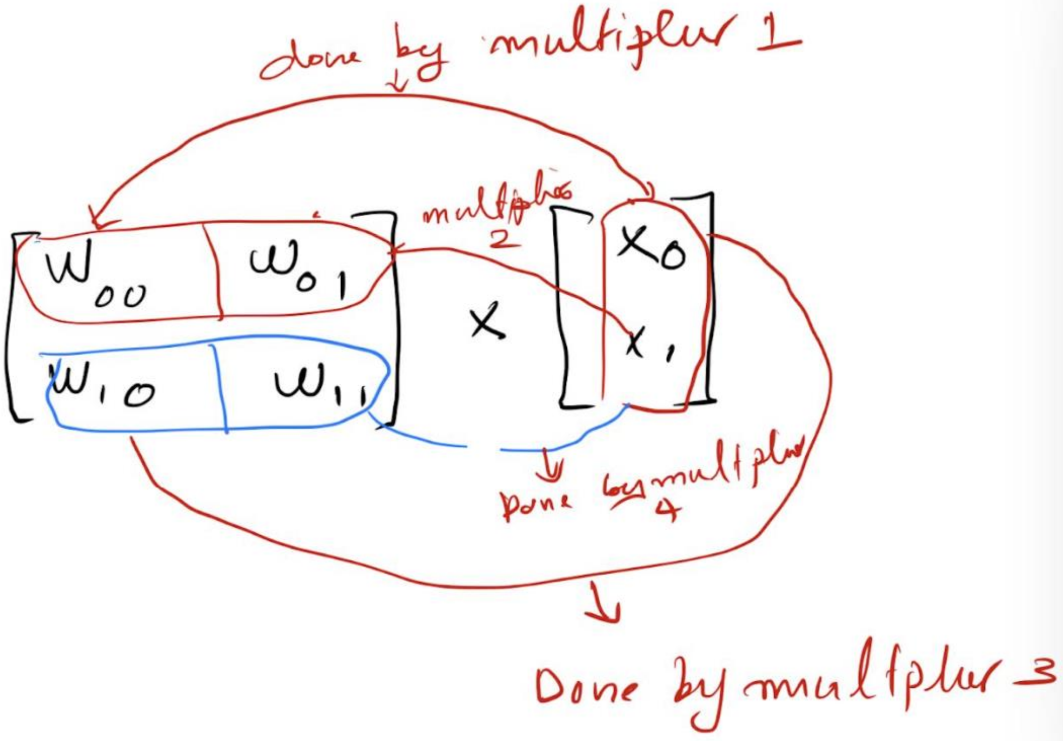
Percentage increases of throughput area and power are calculated with increase in P values.

As we can observe from the slopes of the above graph, from 8 to 16 there is only 5Mega words per second increase with 56% of increase in power and 69% increase in area. From 4 to 8 there is 25 Mega words per second increase with 60 % in area and power. From 2 to 4 there is 25 Mega words per second increase with 51 % in area and 59 % increase in power. From 1 to 2 for 27 Mega words per second increase with 30 % increase in area and 45% increase in power. Here P=2 can achieve higher throughput by paying lesser cost. Therefore, efficiency wise, P=2 is the most efficient.

7. This can be explained with an example of 4X4 matrix and 4X1 vector multiplication. Based on what is followed in this project for below example, number of multiplier can't be more than 2.



But if we want to achieve high parallelism the multplication of row with column can be further broken down. This can be achieved by increasing the multipliers as seen below.



For the above example,

for the first case it takes 12 cycles for loading and executing. This was calculated using the C = ((N+1) + (N+3+2*P)M/P) formula. Now throughput is N/c *f= f/6

Now for the second improved approach, the formula can be re-written as C = ((N+1) + (N/2+3+2*P)M/P) formula. And therefore approximately takes, 11 cycles. Now throughput is N/c *f= f/5.5.

For bigger dimension, the gain with be higher.

8. Throughput = (N/c) words per cycle. Since N is constant for a system, we need to minimize c to get the maximum throughput.
Here we used a brute force mechanism. First, we found the possible P values for all the three-layer based on its dimensions. Then we calculate the delay(C) of each layer when

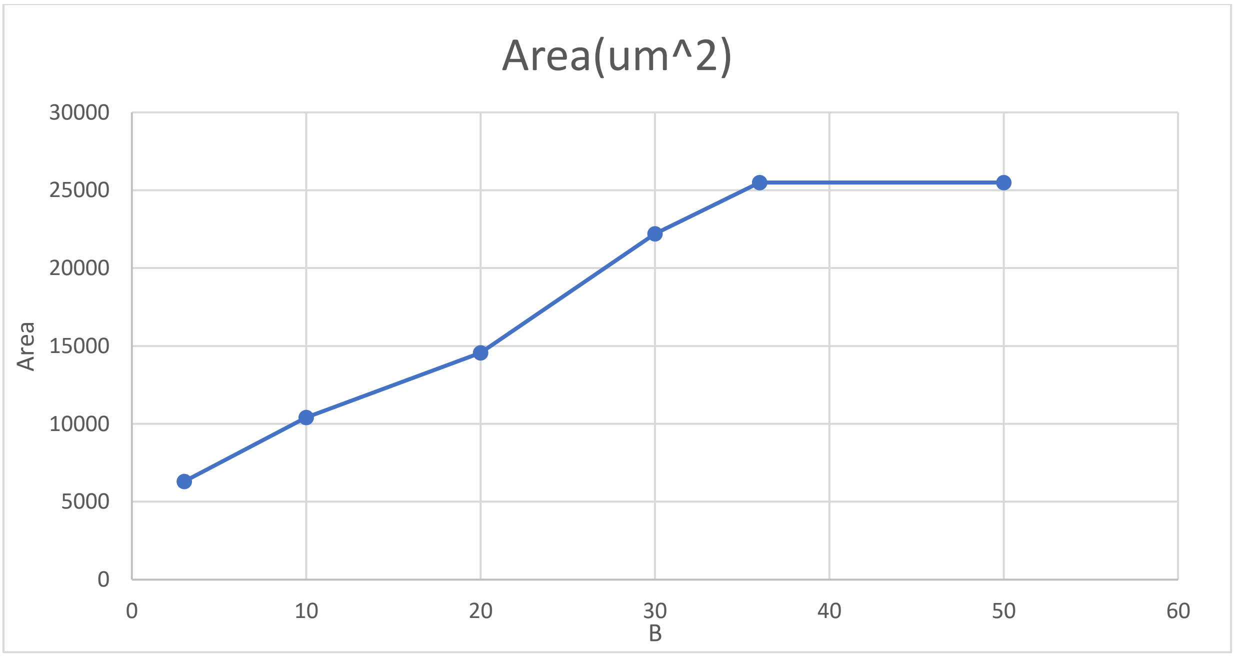
pipeline is full. Now the max value of these three delays is considered as the overall delay of the system.
Formula for finding the delay for layer 1 is ((N+3+(2*P))*M/P + (N +1))

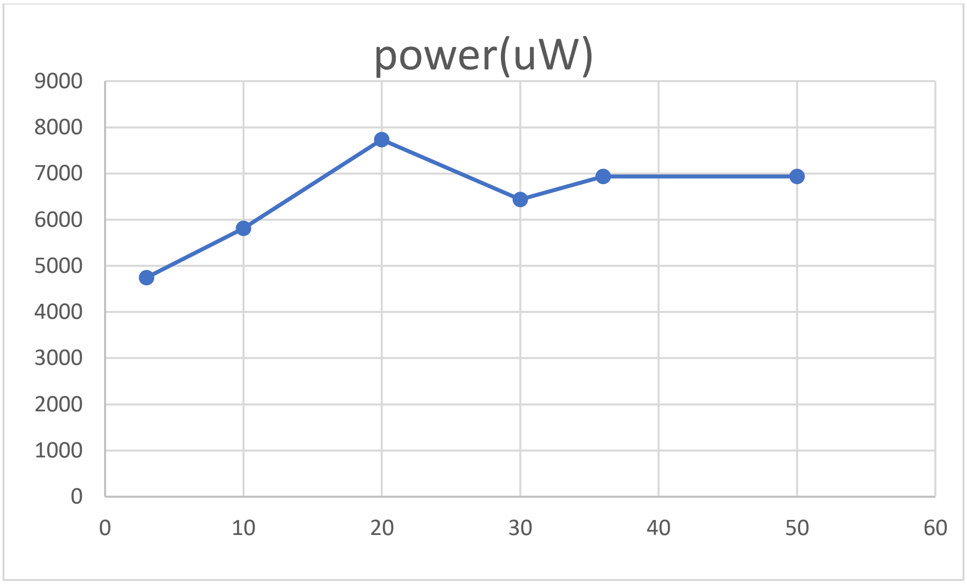
Formula for finding the delay for layer 2 and 3, (((M1+3+(2*P2))*M2/P2) + ((N1+3+(2*P1))*M1/P1) -N1 -3).

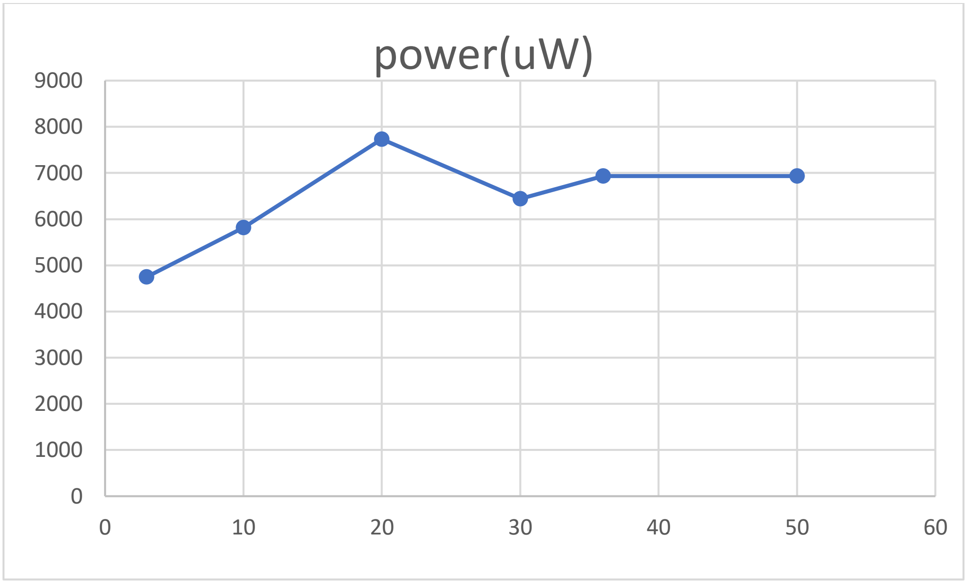
We find the combination of P1, P2 and P3, which gives the least overall c value. If there are multiple combination which gives the same overall c value, then the combination which uses the least budget will be chosen.

We came up with many Network parameters (N, M1, M2,M3)and for each of these networks we manually verified the overall c value from the simulation graph. For B=3,20,30,50 we tried simulations with all possible P values and noted down the best possible P values giving the best overall C. This matched with the optimizer code we came up with.

9. We will add a new parameter L which indicates the number of layers of the network. In genNetwork API of the generator, we will call genFLlayer() L times. For example, if L=5, the user has to provide values for N, M1, M2, M3 , M4 and M5.Also,Changes will be made in the optimizer to calculate the best delay for the network with L layers.
10.







11. Project contribution:

    George: Part 1, Part2(Rom code,Control path changes) , Part3(Optimizer brute force approach)

    Lahari: Part 1, Part 3(Optimizer Delay calculation) , Part2 (Output logic, data path changes)