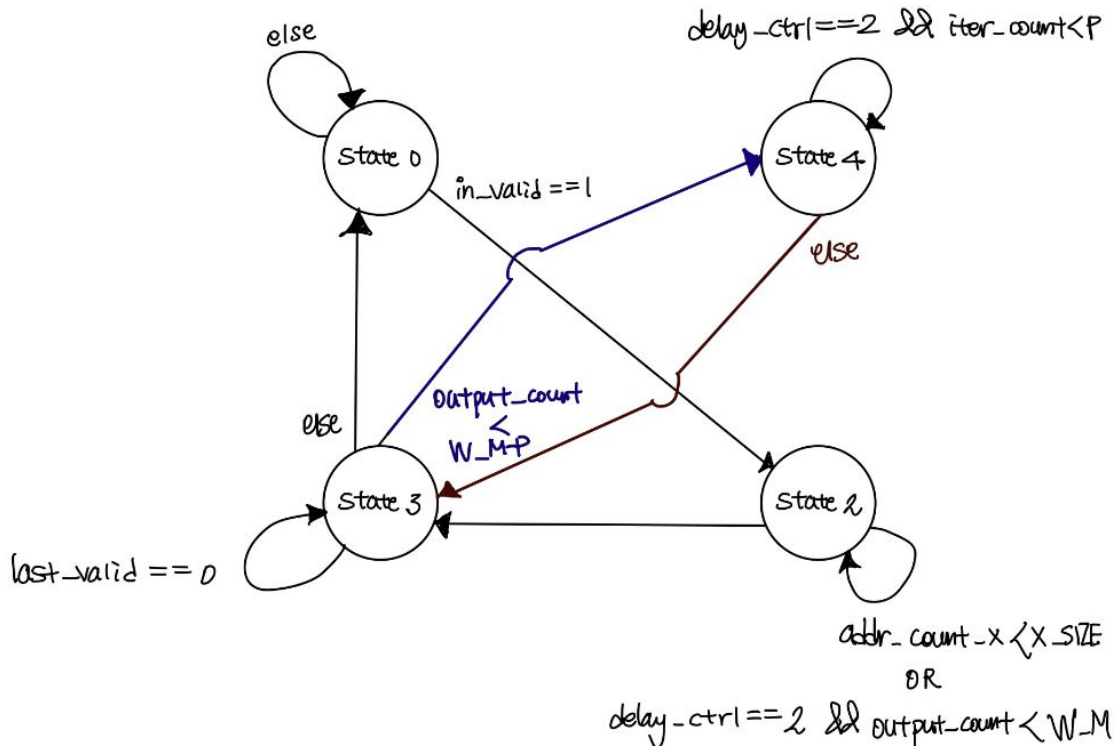
ESE 507 Project 3 Report

by Jinkyu Lee, Xinqi Wang
Fall 2020

1. Describe how you designed your control logic. How did you structure the design so that it can be changed as the M and N parameters change? Explain how the structure changes as these parameters grow.

State Transition for control logic:



The control logic is designed using a FSM.

To change the design as M and N parameters change, we declared genFCLayer void method with int M, int N, int T, int R, int P so that it can pass the given integer values from terminal to the local scope of genFClayer method. The state transition counters are then determined based on these local parameters. As these parameters grow, the time it takes for each corresponding state will increase since there are more inputs to load and process.

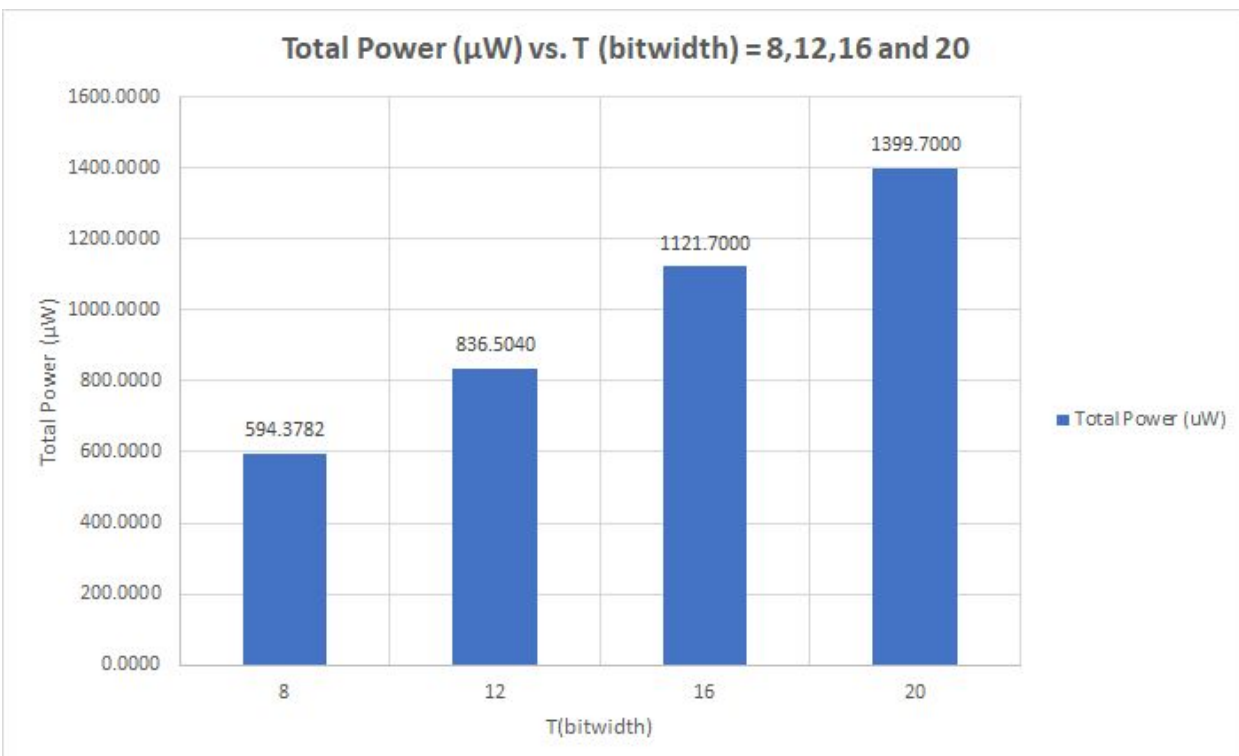
These parameters go through clog2 which returns the ceiling of the logarithm to the base 2. So, as M grows, LOGSIZE_X grows. LOGSIZE_W grows if the resultant of *M divided by P multiplied by N* grows. As P grows NUM_OUPUT grows. As they grow, the matrix dimensions(M and N), bit-width(T or the number of bits used everywhere in the system), the existence of Rectifier Linear Unit (R), and the weight matrix (W) grow. The W matrix uses M by N lines with base 10 integers per line.

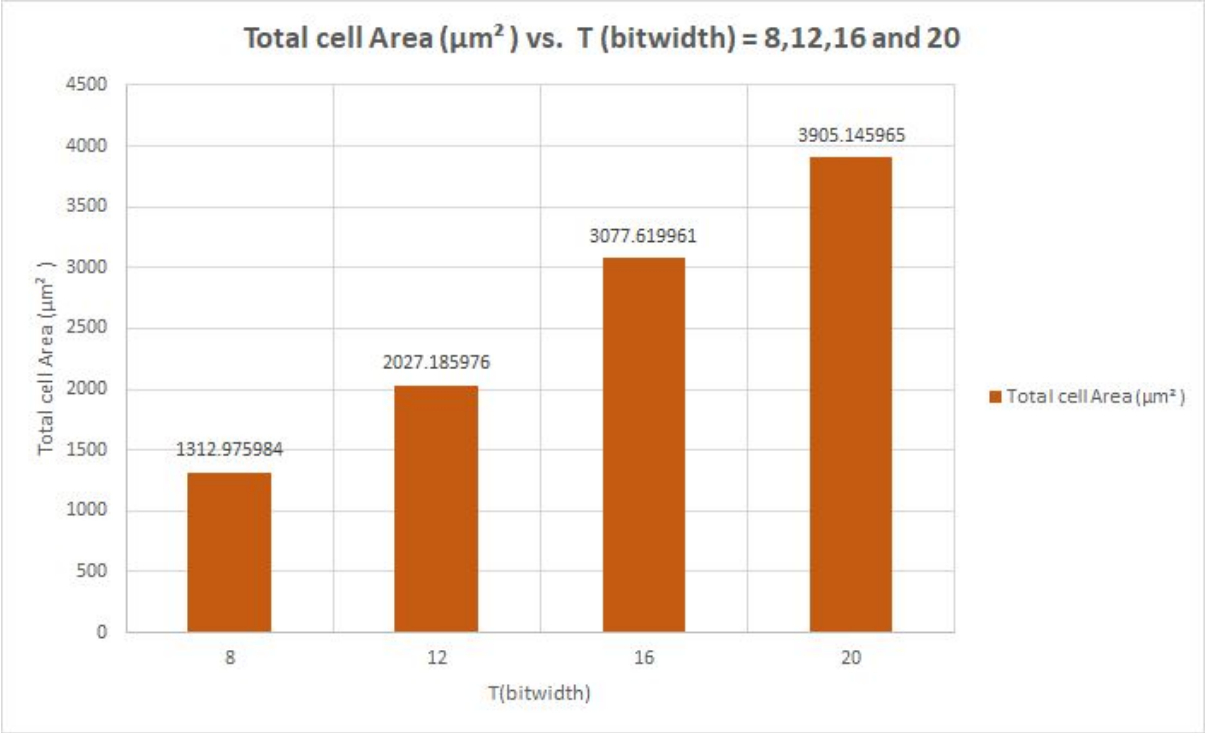
2. Describe how you implemented the parallel (P>1) designs. Explain your structure. What extra logic and storage elements did you need to add? Did you find any clever optimizations to reduce cost?

In genFClayer method, we added extra if statements to see when P is greater than 1 and passed the P value for each of the for loops so that they can iterate until the number of P to create multiple multiply accumulate units to compute P output at the same time. While iterating, multiple w_in, mac_out, mac_out_valid, W_rom, mac_mc, out_v, always_comb block for out from out_v are generated.

For generating constants, when P is greater than 1, rom_size is adjusted as an integer equal to *M divided by P multiplied by N.* Then the values of constVector will be passed into roms. And then, the genROM method will be invoked as many as the P number with different modName for each iteration of the for loop.

3. Now, we will use synthesis to evaluate how the area and power of a layer change as the data precision T changes. Use your generator to produce four designs (Part 1) with T = 8, 12, 16, and 20, while you keep the other parameters constant: (M, N, P, R) = (6, 6, 1, 1).
Then produce two graphs that illustrate: (1) power versus T and (2) area versus T for these designs. Describe where the critical path is located in each design.





Critical paths all lie in the MAC unit from the input register to the saturated product register.
Critical path for fc_6_6_8_1_1:

dp/fc_6_6_8_1_1_mc/**a_r_reg**[1]/CK (DFF_X1) →
dp/fc_6_6_8_1_1_mc/**product_sat_reg**[0]/CK (DFF_X1)

Critical path for fc_6_6_12_1_1:

dp/fc_6_6_12_1_1_mc/**b_r_reg**[5]/CK (DFF_X2) → dp/fc_6_6_12_1_1_mc/**product_sat_reg**[11]/CK (DFF_X1)

Critical path for fc_6_6_16_1_1:

dp/fc_6_6_16_1_1_mc/**b_r_reg**[11]/CK (DFF_X1)→ dp/fc_6_6_16_1_1_mc/**product_sat_reg**[0]/CK (DFF_X1)

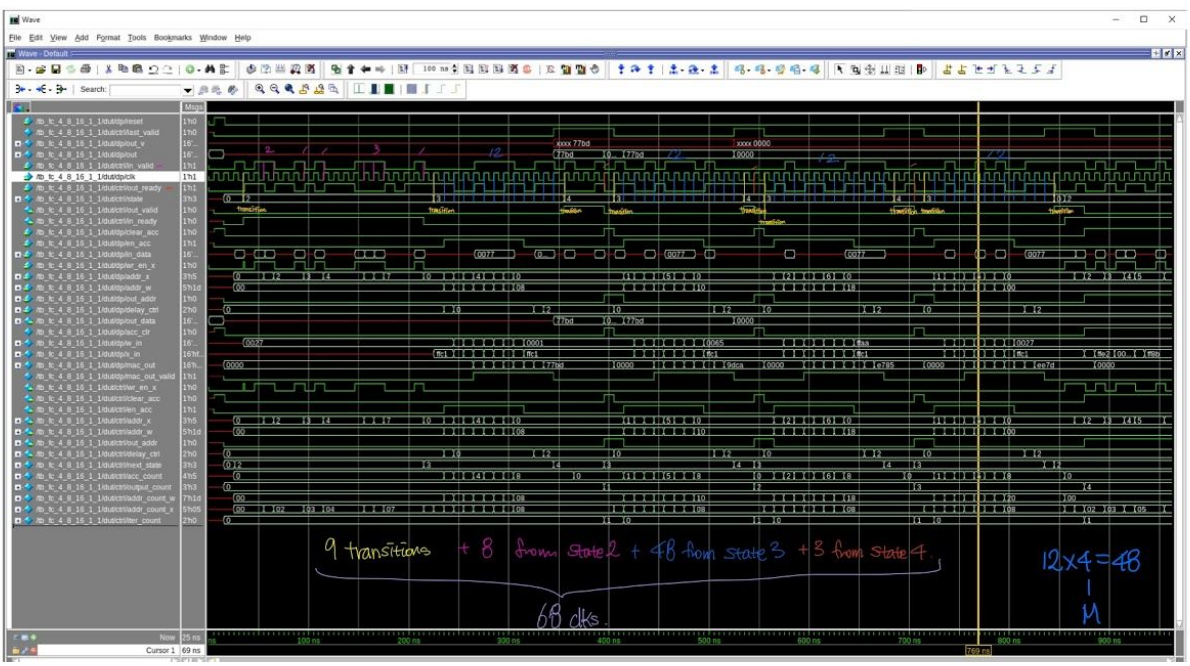
Critical path for fc_6_6_20_1_1:

dp/fc_6_6_20_1_1_mc/**b_r_reg**[5]/CK (DFF_X2) → dp/fc_6_6_20_1_1_mc/**product_sat_reg**[0]/CK (DFF_X1)

The critical paths for these designs lie in our MAC unit from the input registers to the saturated product register. It contains the multiplier and our saturation logic.

4. Next, we will evaluate how throughput, area, and power scale as M changes. Use your generator to produce four designs with M = 4, 6, 8, and 10, while you keep N=8, P=1 and T=16. Synthesize each design, and graph: (1) power versus M, (2) area versus M, and (3) throughput versus M. Does the location of the critical path change as M changes?

Latency is how much time elapses from the time we clock the inputs into the input registers to the time the result is clocked into the output registers. Throughput is the number of inputs or outputs the system processes per second. In our system, latency or a cycle is 1.4 ns as each synthesis resulted in 0.00 of slack, meaning it was MET. We know that the input vector x has length N while the output vector y has length M. We changed M values from 4, 6, 8 to 10 so each length of the output vector was increased for each synthesis. So, to get throughput, we can multiply the frequency by each M, output vector length to get each throughput as M changes like shown in the table and graph below. When M was 4, 6, 8, 10, the critical path was from b_r_reg to product_sat_reg. There was no change in each critical path. This is expected because it lies in the MAC unit.



So, for fc 4 8 16 1:

N = 8 words

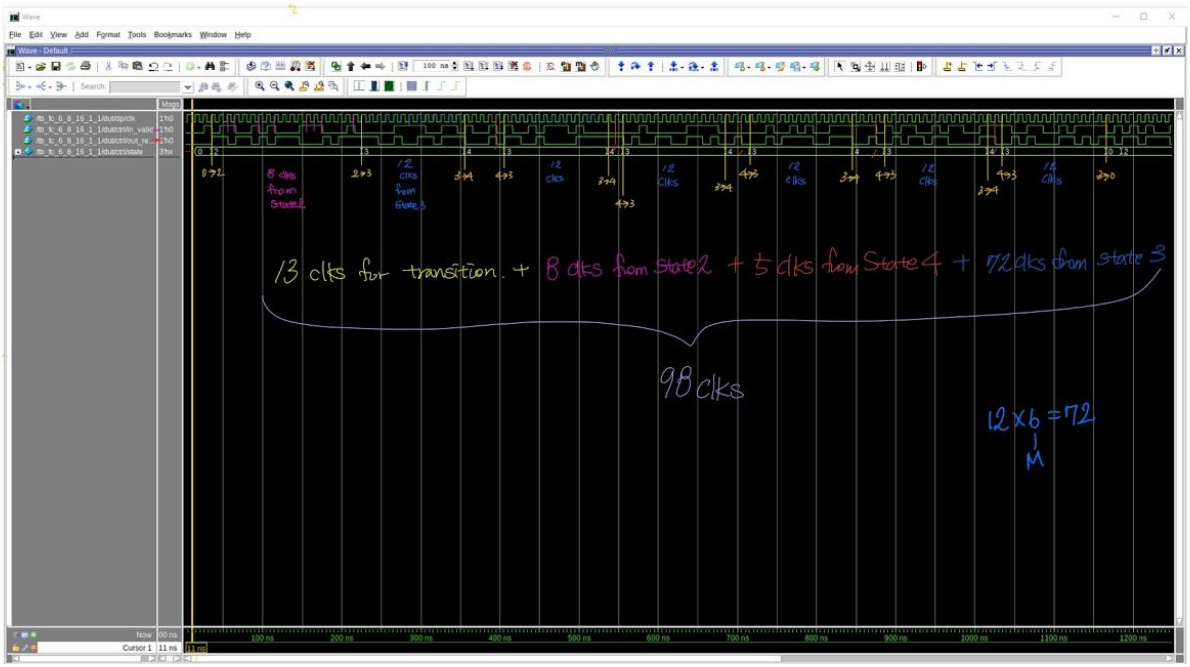
N/c = 8 words / 68 clks

Throughput = (N/c) x frequency

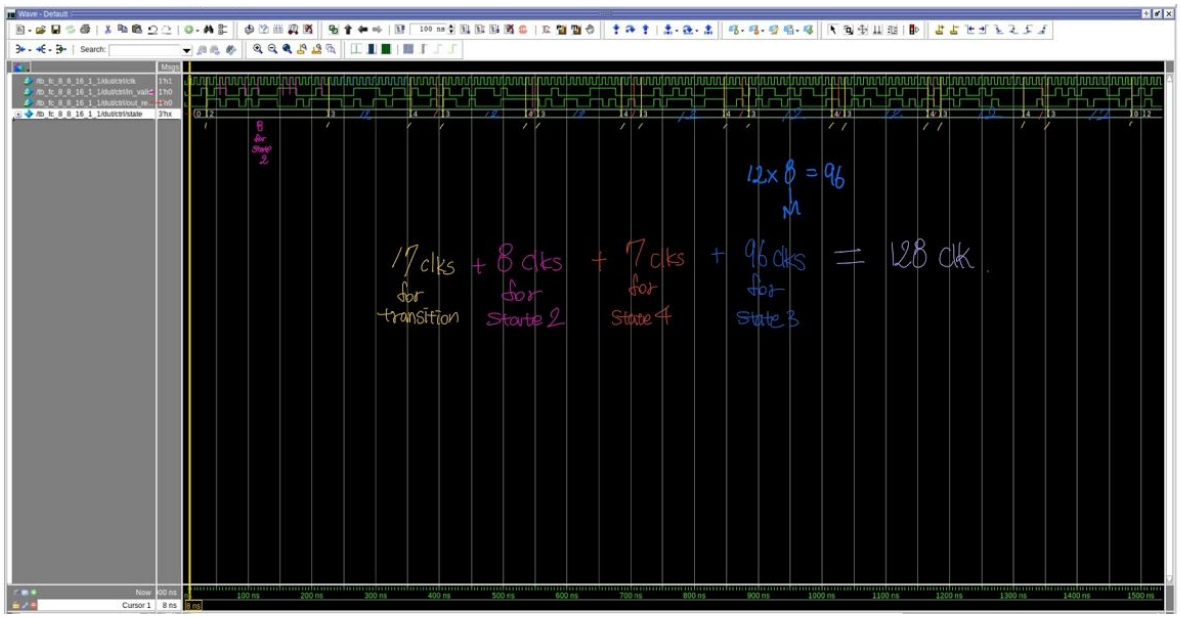
(N/c) x f = (8 words /68 clks) x 714.2857 MHz = 0.0840336 billion outputs / second

We repeat this calculation for the rest of the cases with different c.

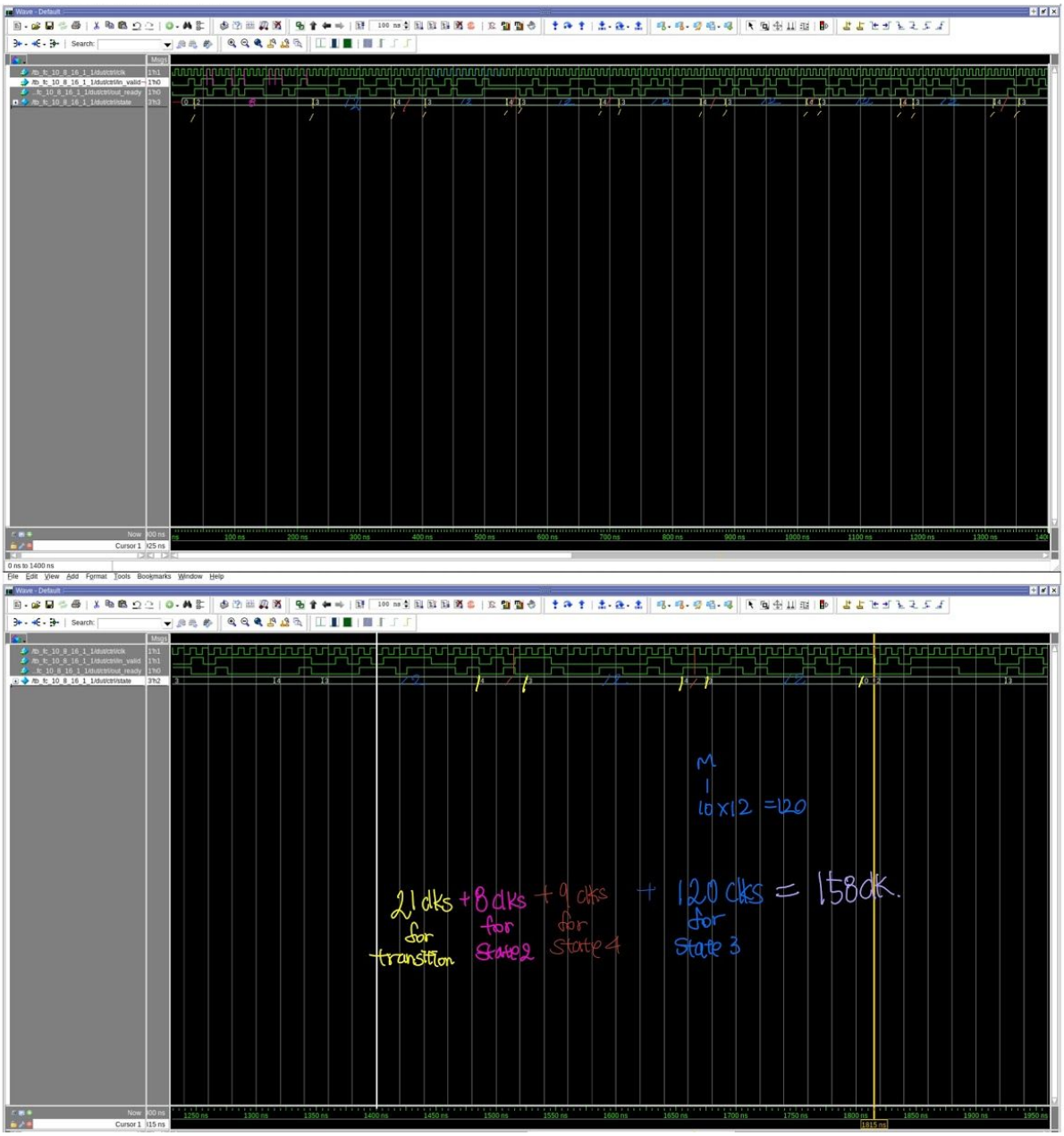
For fc 6 8 16 1: c= 98 clks



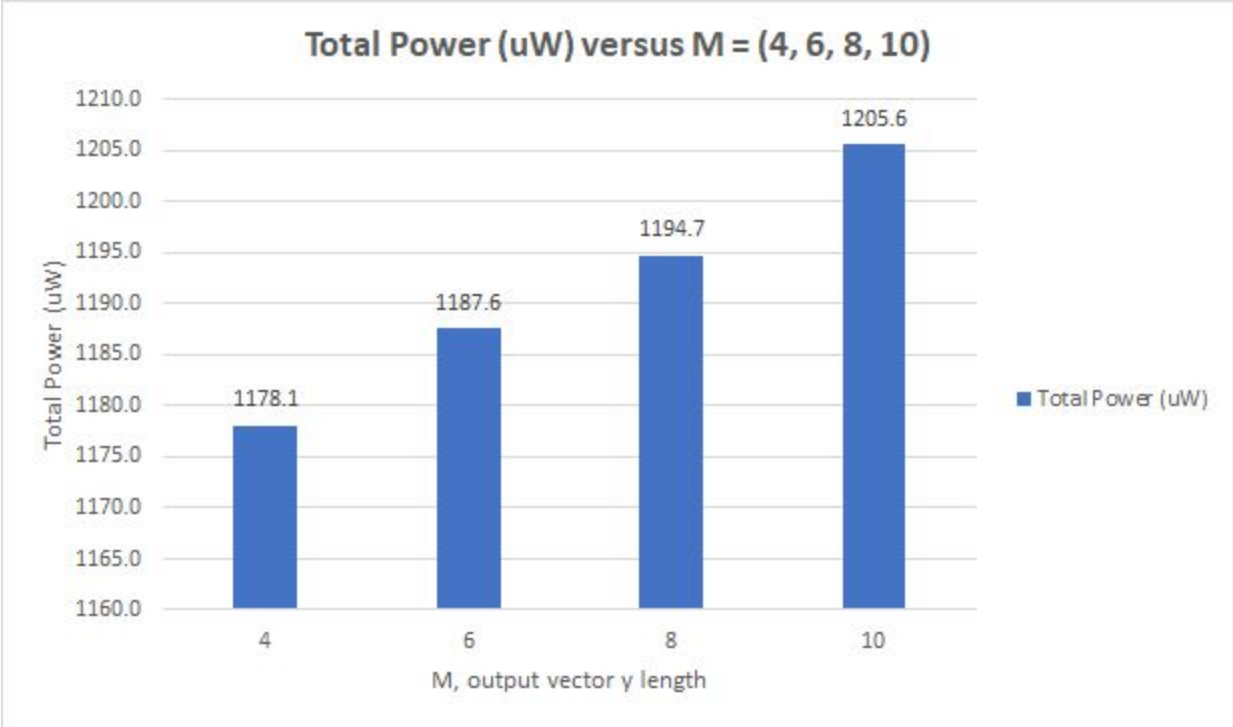
For fc 8 8 16 1: c= 128 clks

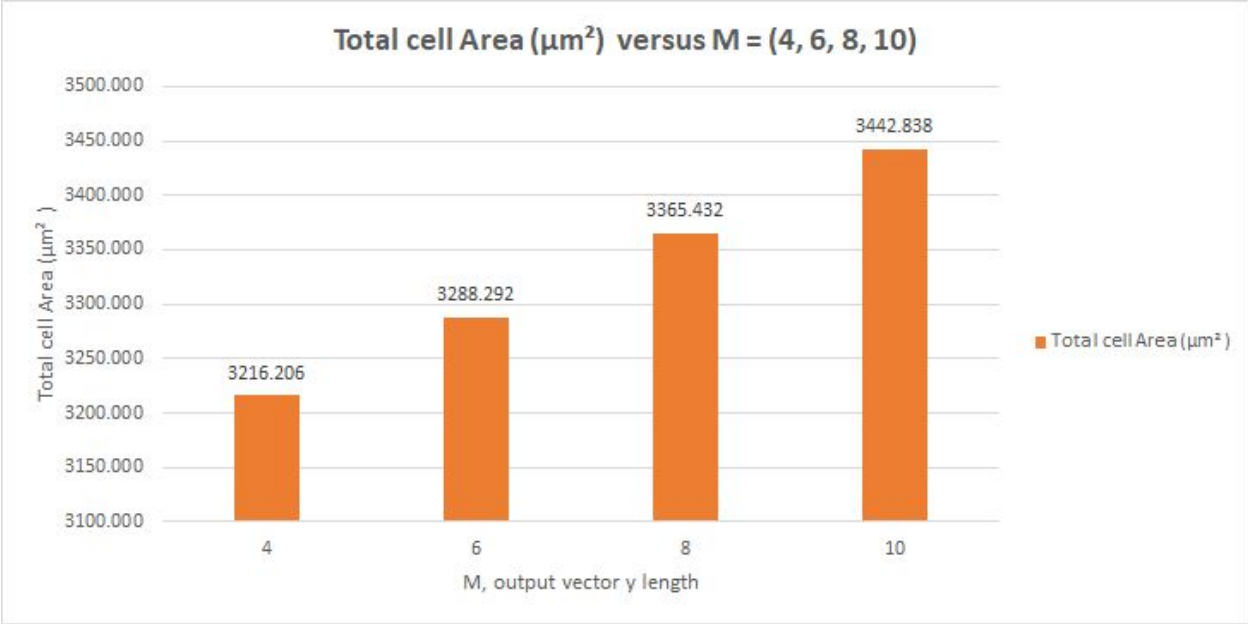


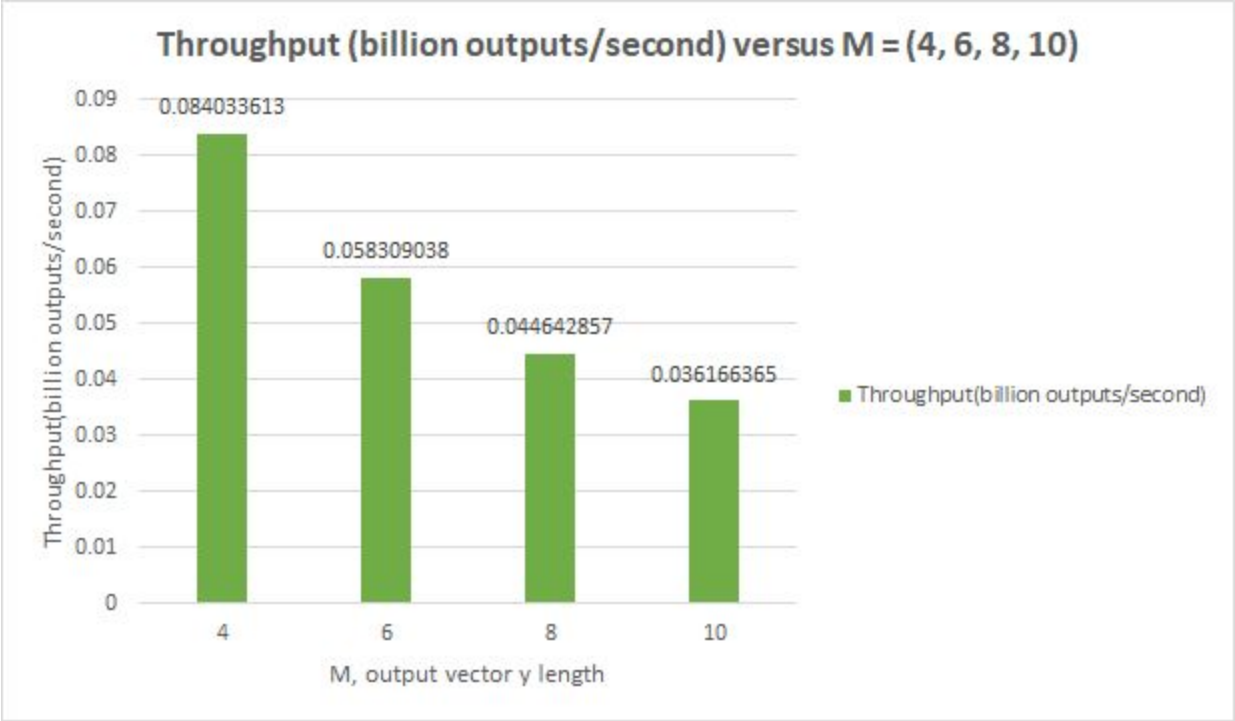
For fc 10 8 16 1: c= 158 clks



M N T R	Total Power (uW)	Total cell Area (μm²)	c (cycle)	Frequency (MHz)	Throughput (billion outputs/sec)
4 8 16 1	1178.1	3216.20596	68	714.2857	0.084033613
6 8 16 1	1187.6	3288.29196	98	714.2857	0.058309038
8 8 16 1	1194.7	3365.43196	128	714.2857	0.044642857
10 8 16 1	1205.6	3442.83796	158	714.2857	0.036166365


Total Power (uW) versus M = (4, 6, 8, 10)
1210.0
1205.0
1200.0
1195.0
1190.0
1185.0
1180.0
1175.0
1170.0
1165.0
1160.0
Total Power (uW)
1178.1
1187.6
1194.7
1205.6
4
6
8
10
M, output vector y length
Total Power (uW)


Total cell Area (µm²) versus M = (4, 6, 8, 10)
3500.000
3450.000
3400.000
3350.000
3300.000
3250.000
3200.000
3150.000
3100.000
Total cell Area (µm²)
3216.206
3288.292
3365.432
3442.838
4
6
8
10
M, output vector y length
Total cell Area (µm²)



Critical path for output_fc_4_8_16_1_1:

Startpoint: dp/fc_4_8_16_1_1_mc/**b_r_reg**[11]
(rising edge-triggered flip-flop clocked by clk)
Endpoint: dp/fc_4_8_16_1_1_mc/**product_sat_reg**[12]
(rising edge-triggered flip-flop clocked by clk)

Critical path for output_fc_6_8_16_1_1:

Startpoint: dp/fc_6_8_16_1_1_mc/**b_r_reg**[11]
(rising edge-triggered flip-flop clocked by clk)
Endpoint: dp/fc_6_8_16_1_1_mc/**product_sat_reg**[12]
(rising edge-triggered flip-flop clocked by clk)

Critical path for output_fc_8_8_16_1_1:

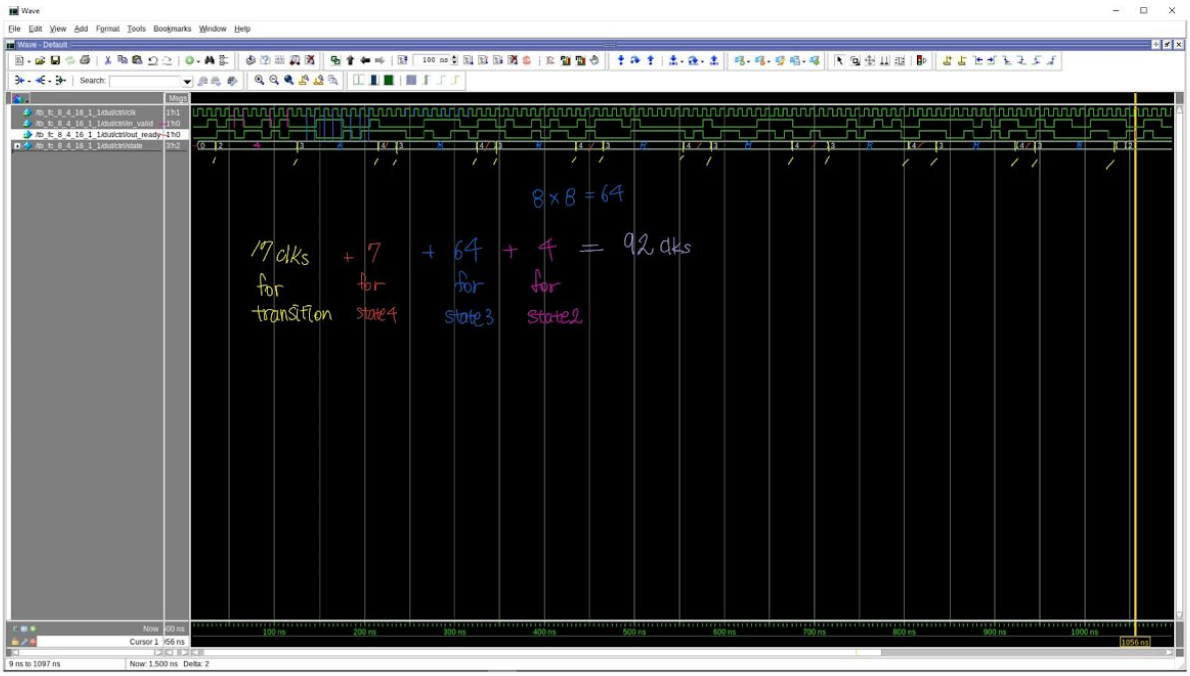
Startpoint: dp/fc_8_8_16_1_1_mc/**b_r_reg**[1]
(rising edge-triggered flip-flop clocked by clk)
Endpoint: dp/fc_8_8_16_1_1_mc/**product_sat_reg**[1]
(rising edge-triggered flip-flop clocked by clk)

Critical path for output_fc_10_8_16_1_1:

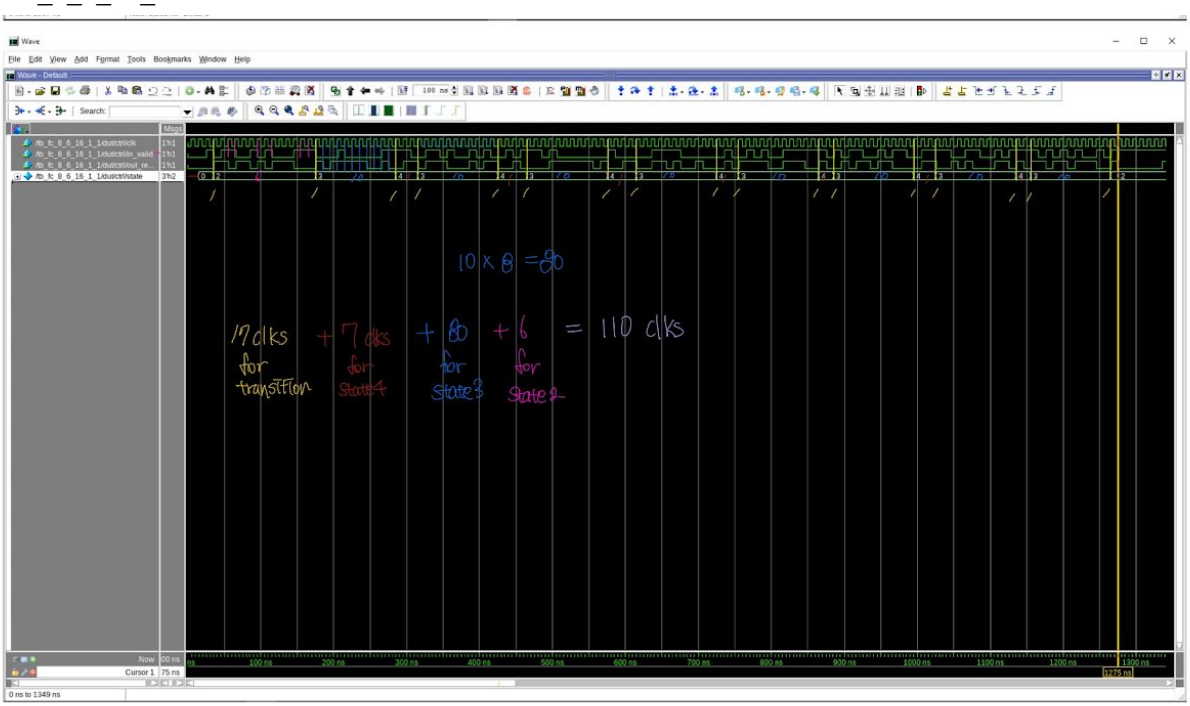
Startpoint: dp/fc_10_8_16_1_1_mc/**b_r_reg**[3]
(rising edge-triggered flip-flop clocked by clk)
Endpoint: dp/fc_10_8_16_1_1_mc/**product_sat_reg**[0]
(rising edge-triggered flip-flop clocked by clk)

5. Now, we will repeat the previous question while N changes. Use your generator to produce four designs with N=4, 6, 8, and 10. Set M=8, P=1, T=16, and R=1. Synthesize each design, and graph: (1) power versus N, (2) area versus N, and (3) throughput versus N. Does the location of the critical path change as N changes?

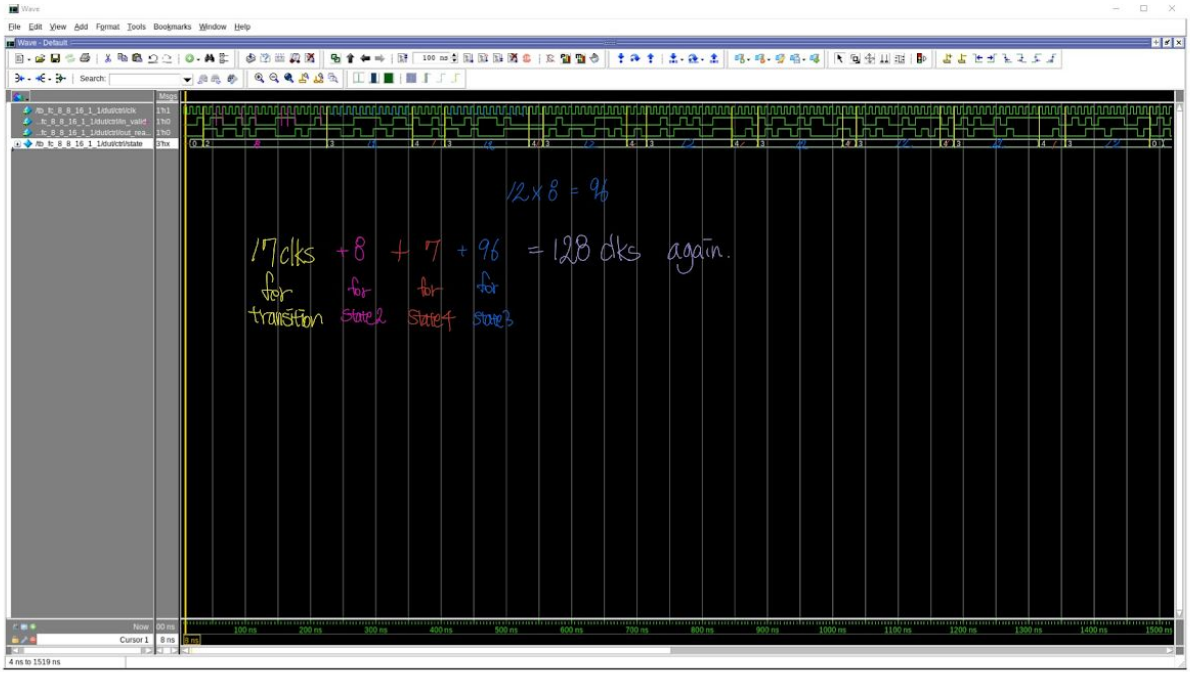
For fc_8_4_16_1:



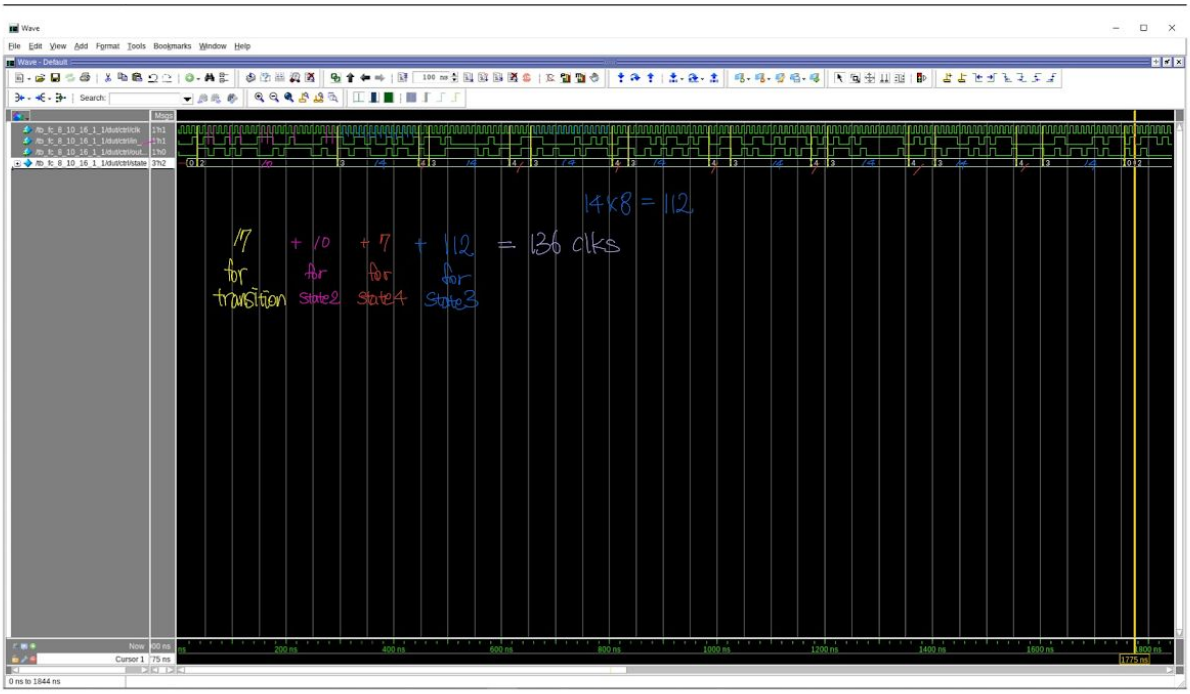
For fc_8_6_16_1:



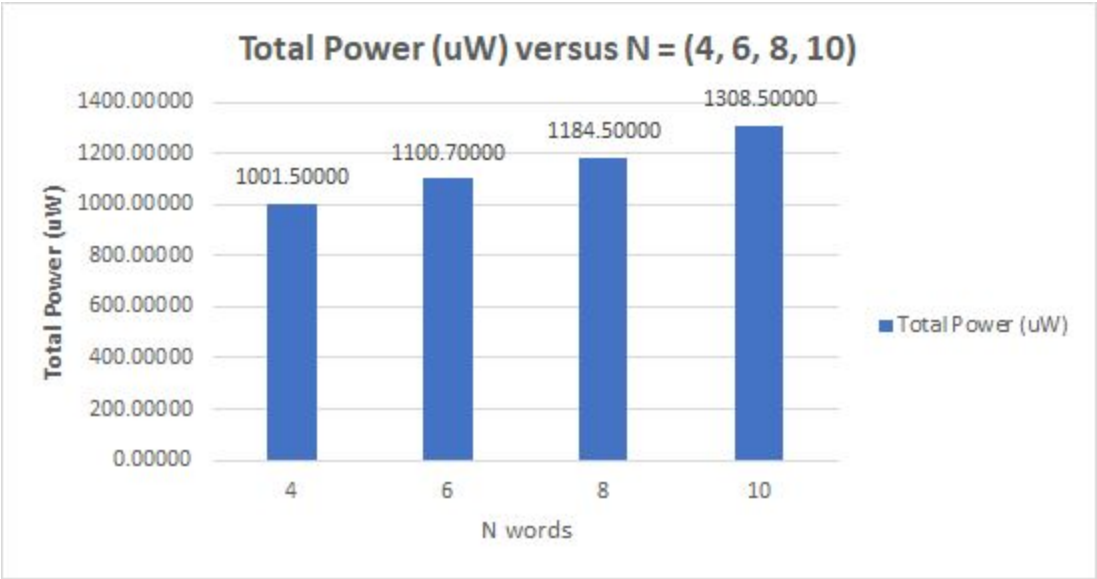
For fc_8_8_16_1:

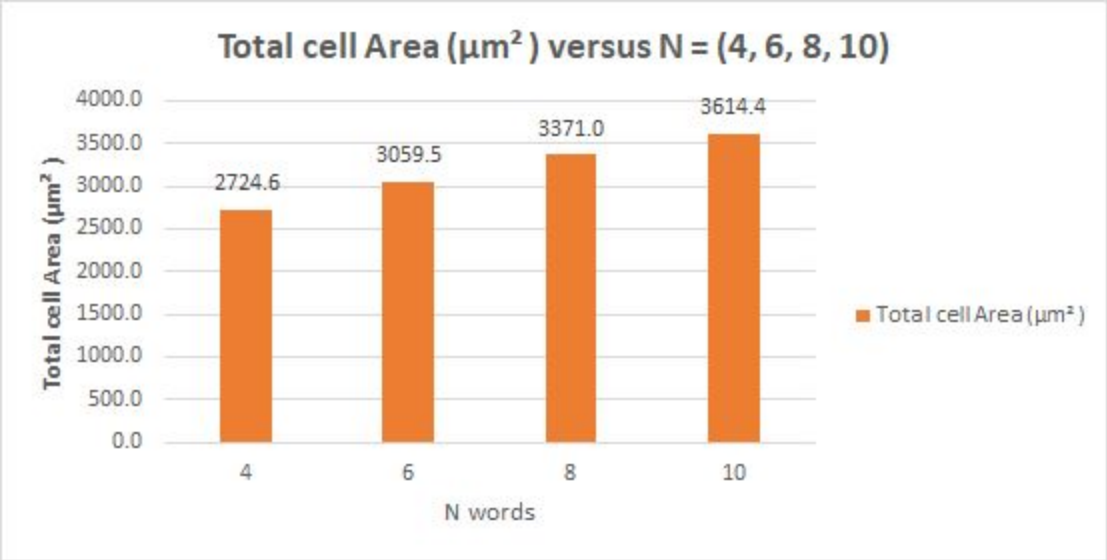


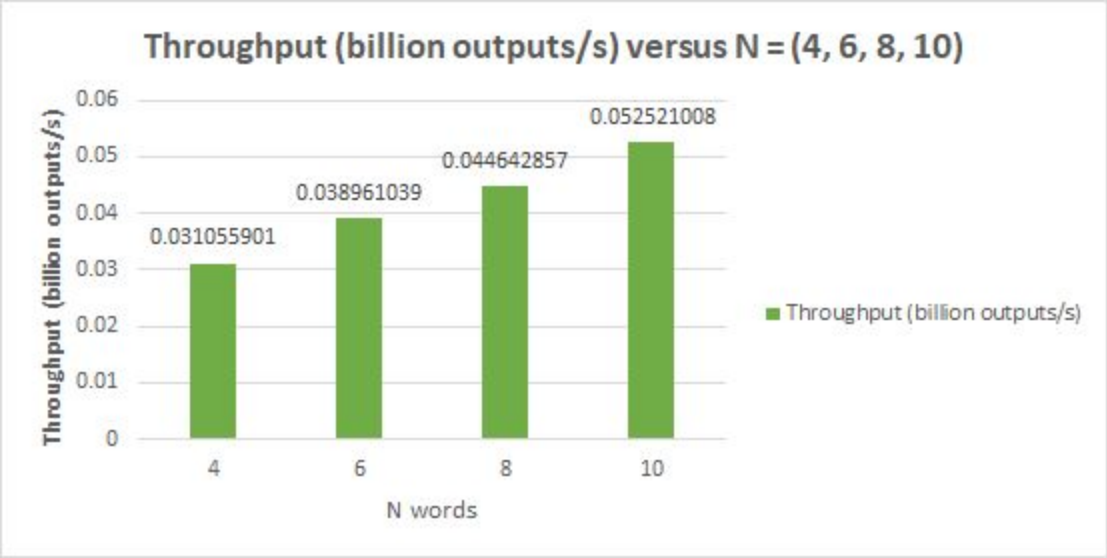
For fc_8_10_16_1:



M N T R	Total Power (uW)	Total cell Area (µm²)	Frequency (MHz)	c (clk)	Throughput (billion outputs/s)
8 4 16 1	1001.50000	2724.6	714.2857	92	0.031055901
8 6 16 1	1100.70000	3059.5	714.2857	110	0.038961039
8 8 16 1	1184.50000	3371.0	714.2857	128	0.044642857
8 10 16 1	1308.50000	3614.4	714.2857	136	0.052521008







Critical path for output_fc_8_4_16_1_1:

Startpoint: dp/fc_8_4_16_1_1_mc/**b_r_reg**[7]
(rising edge-triggered flip-flop clocked by clk)
Endpoint: dp/fc_8_4_16_1_1_mc/**product_sat_reg**[10]
(rising edge-triggered flip-flop clocked by clk)

Critical path for output_fc_8_6_16_1_1:

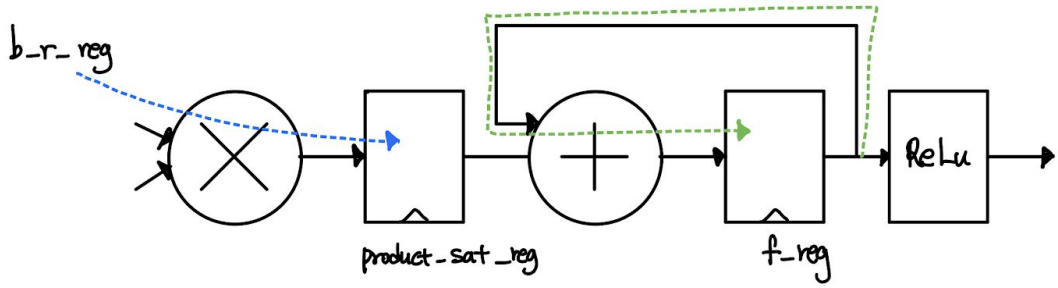
Startpoint: dp/fc_8_6_16_1_1_mc/**b_r_reg**[15]
(rising edge-triggered flip-flop clocked by clk)
Endpoint: dp/fc_8_6_16_1_1_mc/**product_sat_reg**[0]
(rising edge-triggered flip-flop clocked by clk)

Critical path for output_fc_8_8_16_1_1:

Startpoint: dp/fc_8_8_16_1_1_mc/**f_reg**[0]
(rising edge-triggered flip-flop clocked by clk)
Endpoint: dp/fc_8_8_16_1_1_mc/**f_reg**[10]
(rising edge-triggered flip-flop clocked by clk)

Critical path for output_fc_8_10_16_1_1:

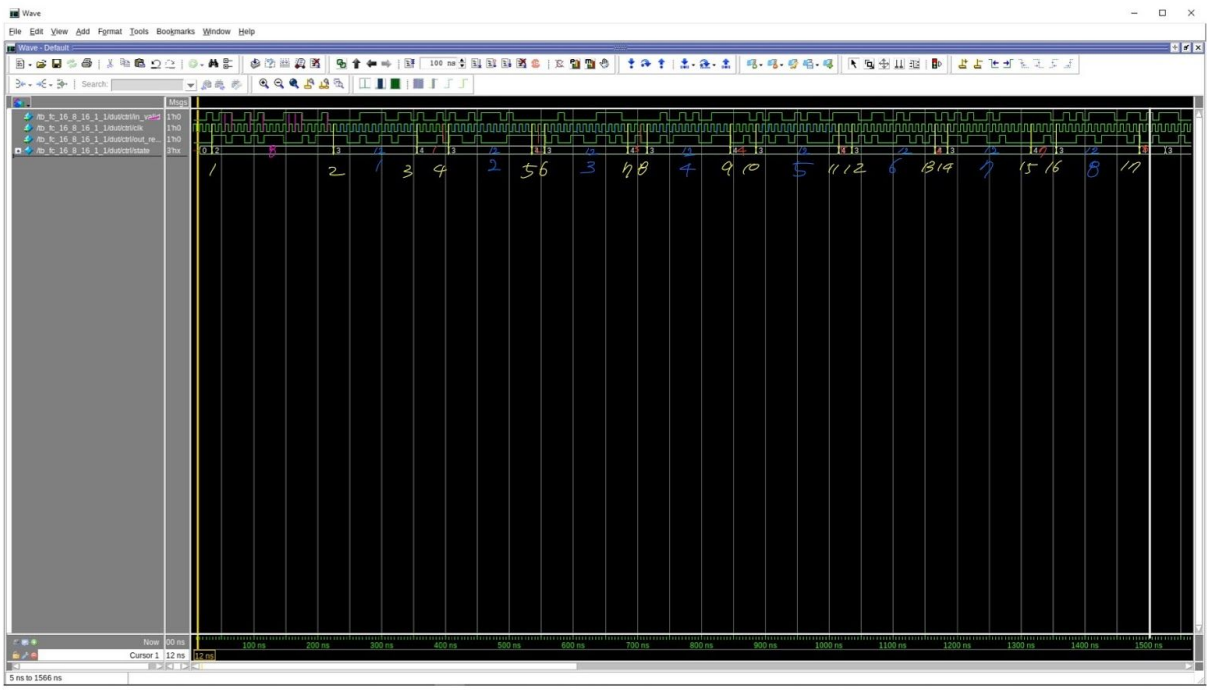
Startpoint: dp/fc_8_10_16_1_1_mc/**b_r_reg**[8]
(rising edge-triggered flip-flop clocked by clk)
Endpoint: dp/fc_8_10_16_1_1_mc/**product_sat_reg**[0]
(rising edge-triggered flip-flop clocked by clk)

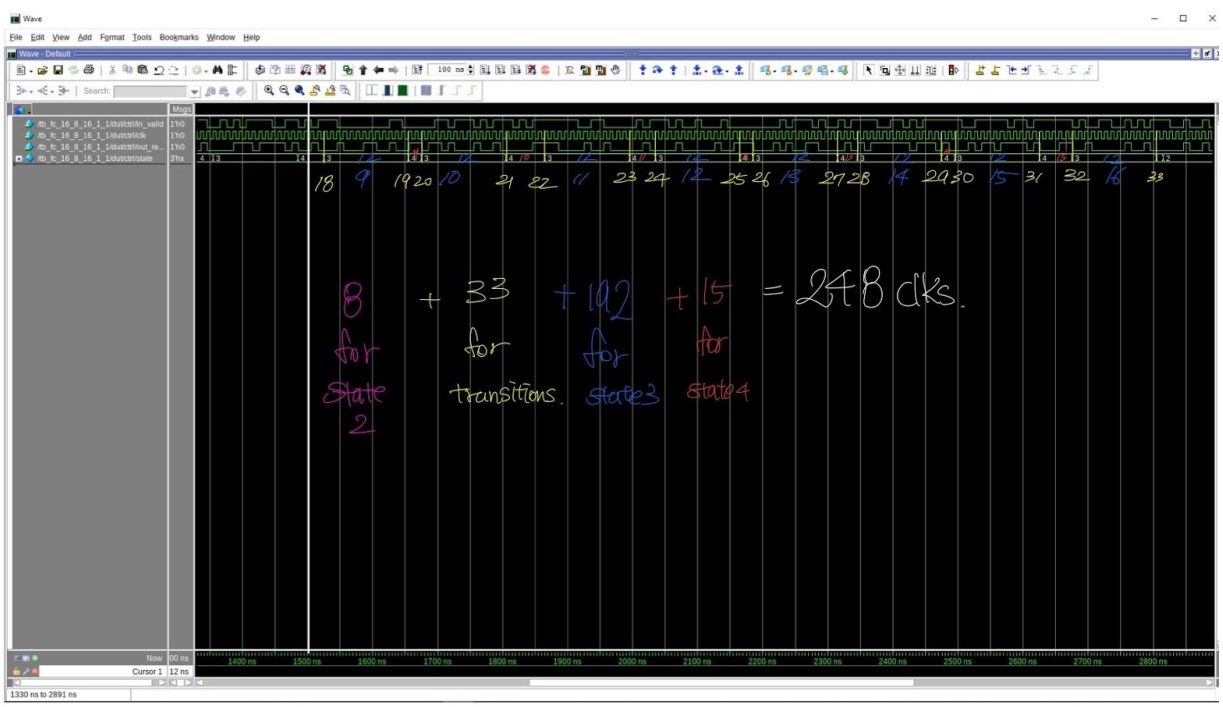


For design fc_8_8_16_1_1, we noticed the critical path we got was from f_reg to f_reg, which is the path containing the adder(green), instead of the one containing the multiplier(blue). Although we don't think this is an indication of an error in our design, since both paths in the above image could be the critical path, we are not sure of the reason why for this particular design the critical path turned out to be the adder path.

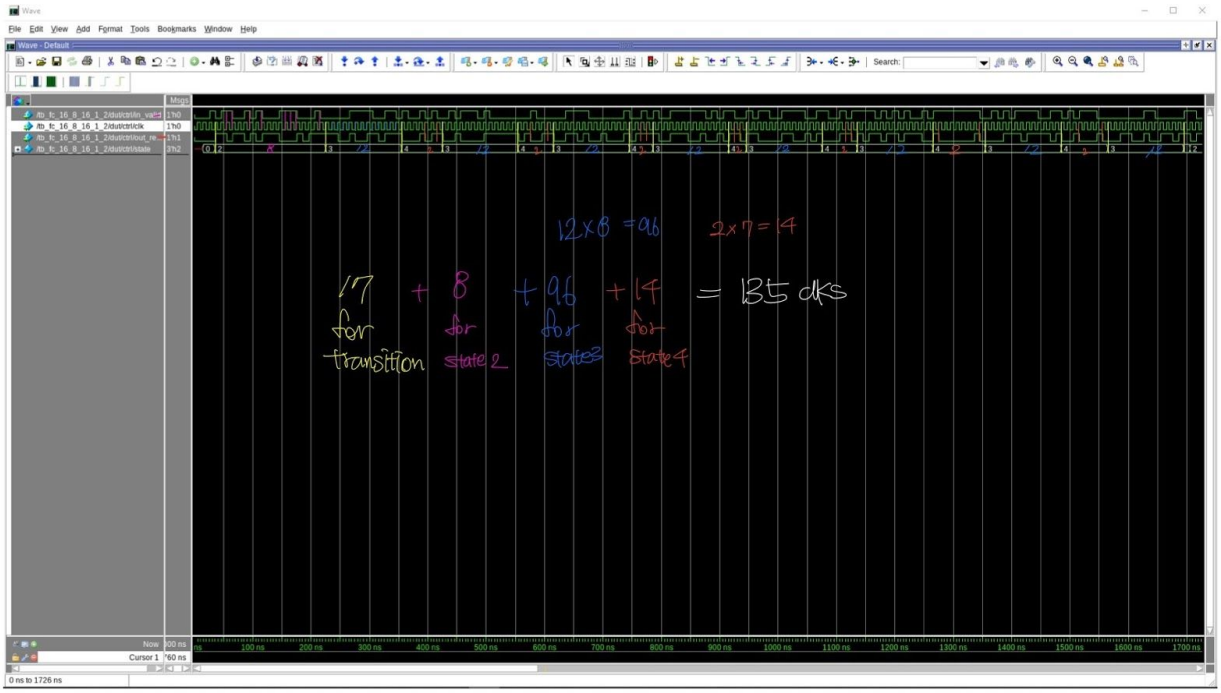
6. Evaluate how the designs change when you increase parallelism, by evaluating P=1, 2, 4, 8, 16, with M=16, N=8, R=1, and T=16. Graph: (1) power versus P, (2) area versus P, and (3) throughput versus P. As parallelism increases, the designs should get faster but also more expensive. Which of these designs the most efficient? Justify your answer quantitatively.

For fc 16 8 16 1 1:

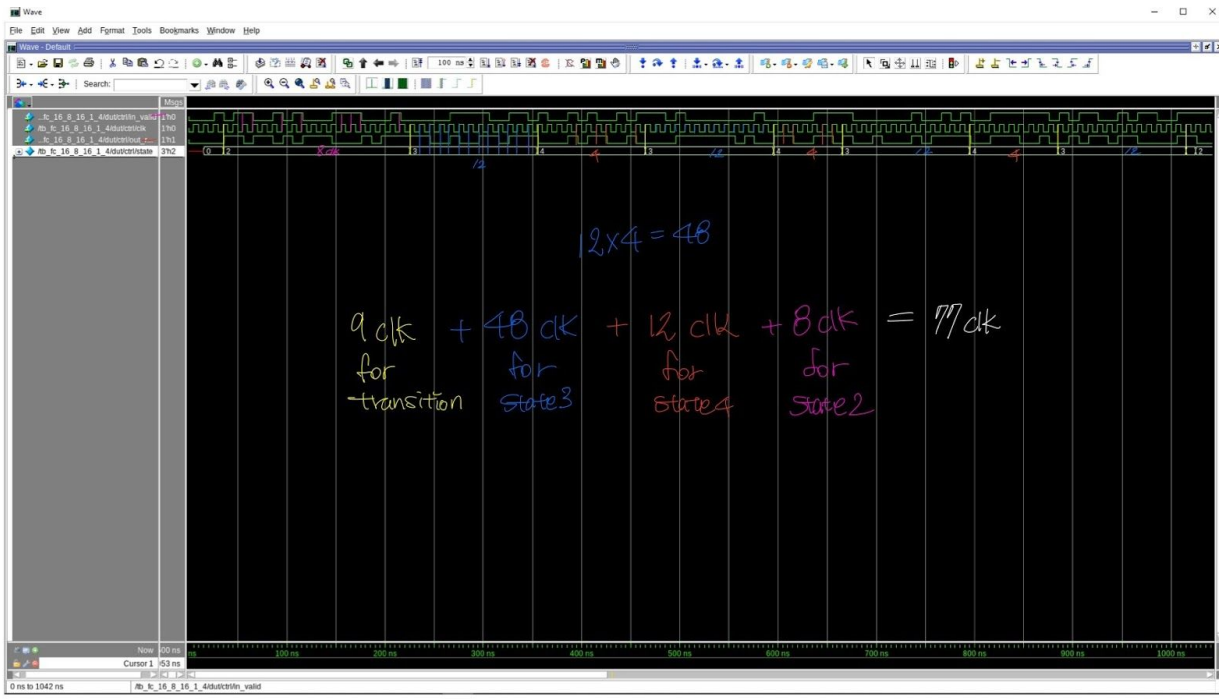




For fc 16 8 16 1 2:

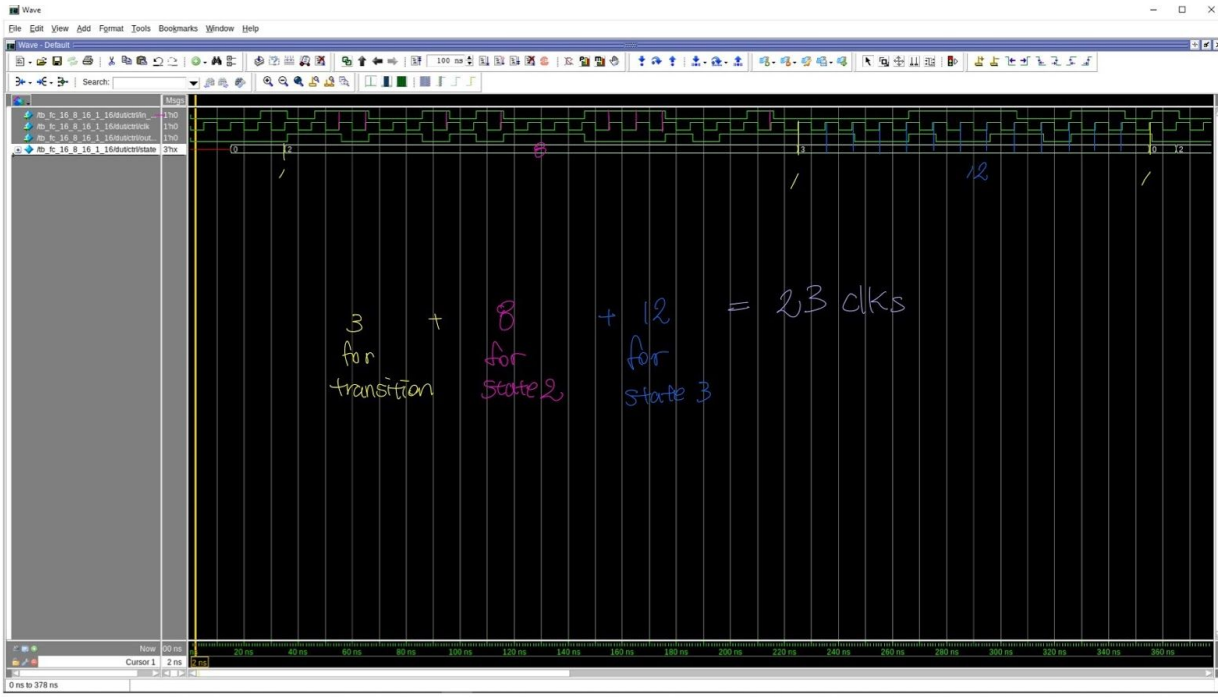


For fc 16 8 16 1 4:

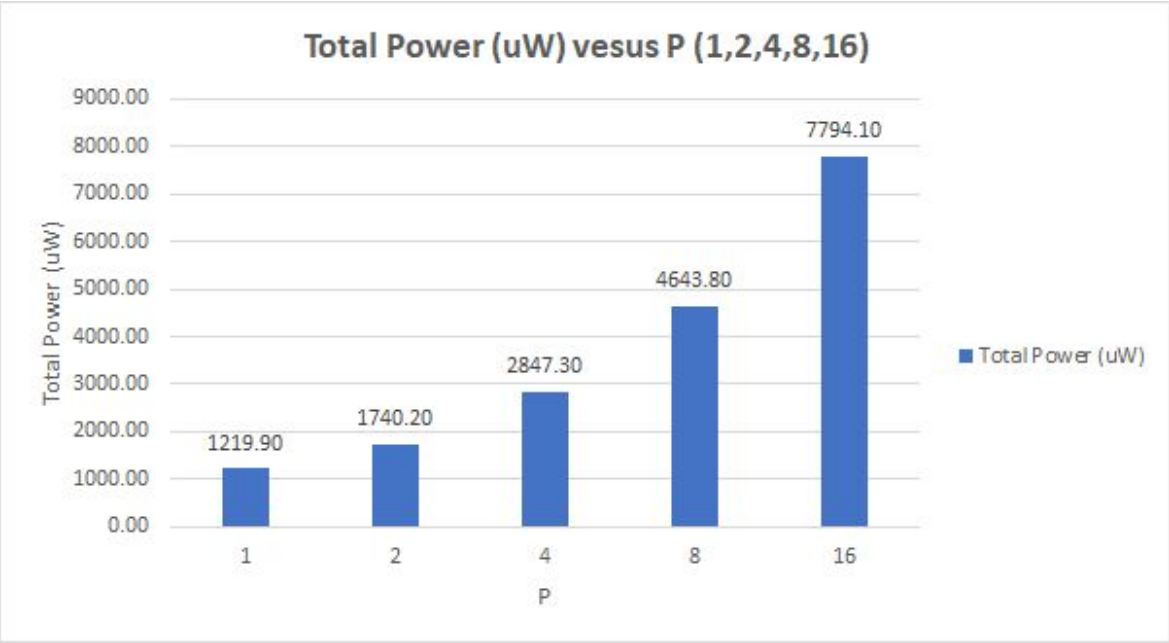


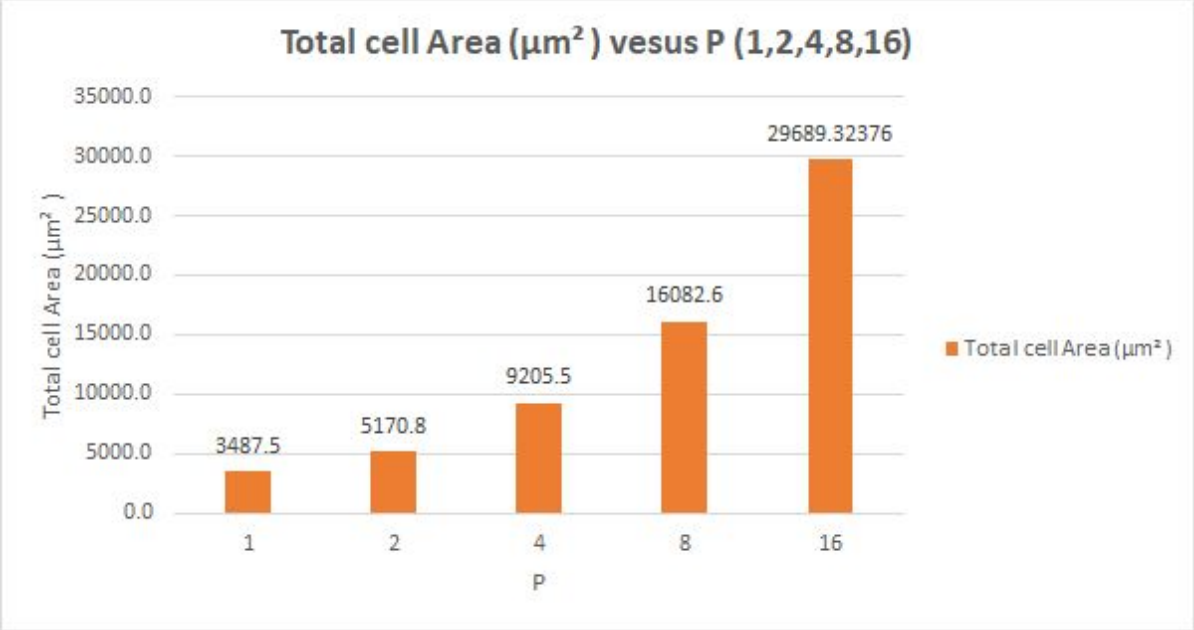
For fc 16 8 16 1 8:

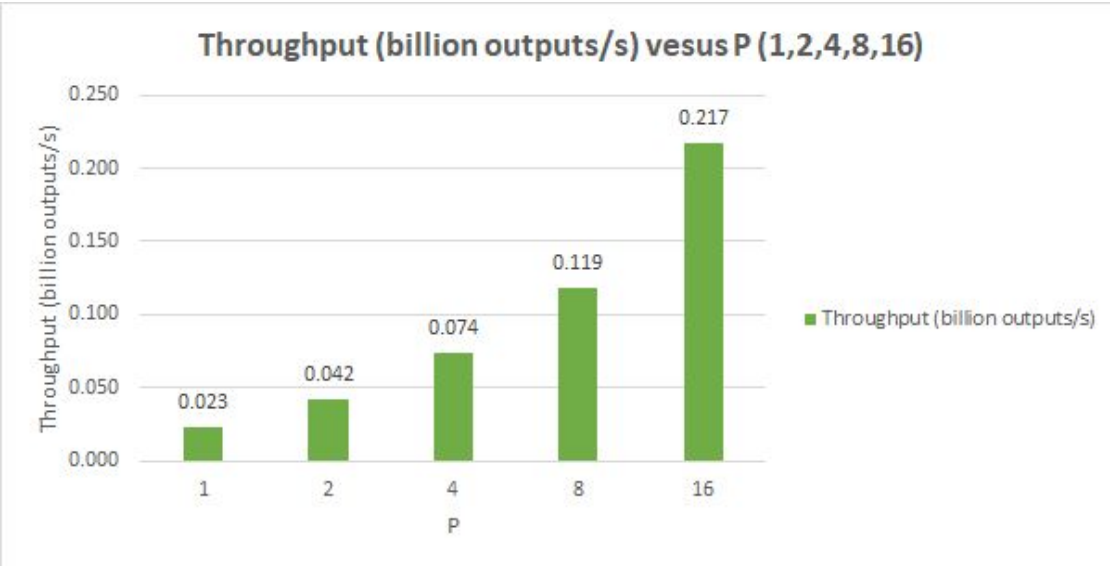
For fc 16 8 16 1 16:



M N T R P	Total Power (uW)	Total cell Area (µm²)	Frequncy (MHz)	clk (ns)	cycle (clk)	Throughput (billion outputs/s)
16 8 16 1 1	1219.90	3487.5	714.2857	1.4	248	0.023041475
16 8 16 1 2	1740.20	5170.8	714.2857	1.4	135	0.042328042
16 8 16 1 4	2847.30	9205.5	714.2857	1.4	77	0.074211503
16 8 16 1 8	4643.80	16082.6	666.6667	1.5	45	0.118518519
16 8 16 1 16	7794.10	29689.3	625.0000	1.6	23	0.217391304



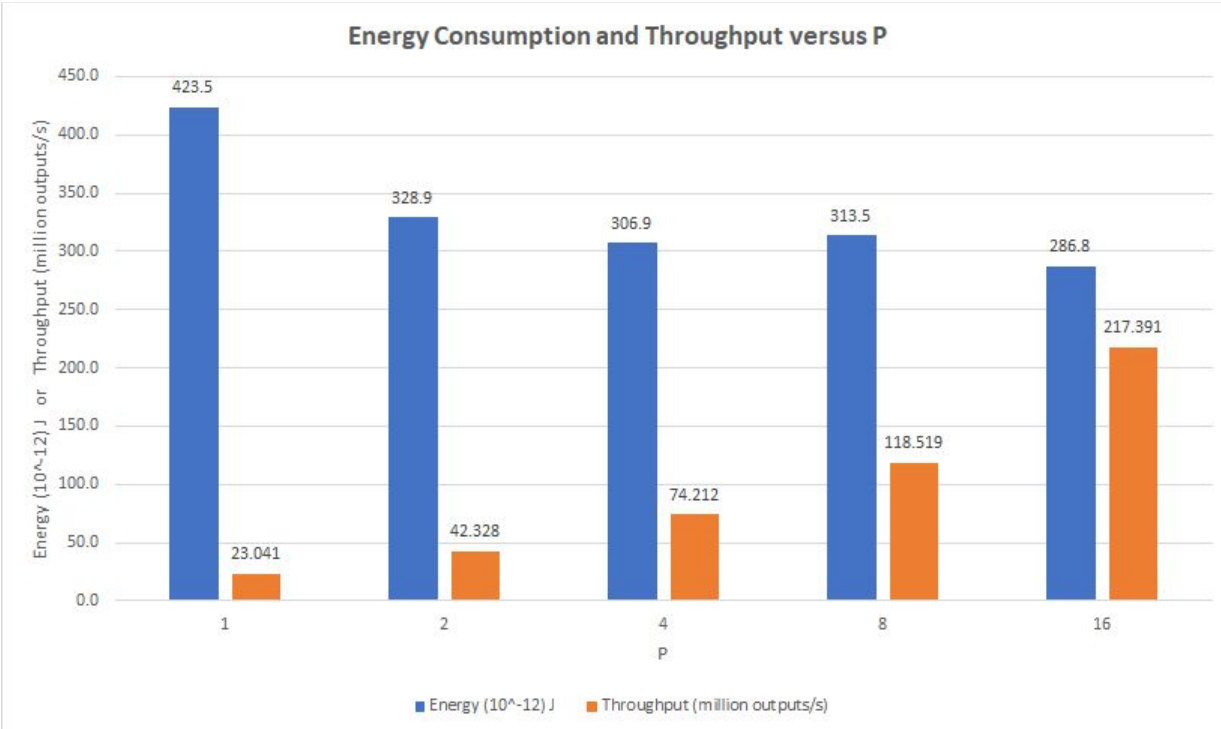




P->P'	Power increase	Throughput increase	Throughput increase divided by power increase
1 -> 2	x1.43	x1.84	1.29
2 -> 4	x1.64	x1.75	1.07
4 -> 8	x1.63	x1.6	0.98
8 -> 16	x1.68	x1.83	1.09

When P goes from 2 to 4 or 4 to 8, the improvement in throughput doesn't justify the increase in power as the other two designs do (P=1->2 and 8->16). For example, when P goes from 4->8 the throughput is 1.6 times greater but the design is actually consuming 1.63 times more power. We think fc_16_8_16_1_2 and fc_16_8_ 16_1_16 are more efficient under the assumption that we have an unlimited amount of area and what we aim for is less energy to compute high throughput. We calculated energy consumption for each case by multiplying each total power with each time taken to finish one cycle. The following table and graph shows that when P is 16 it costumes less energy and gives the highest throughput.

P	Energy (10^-12) J	Total Power (uW)	Total cell Area (µm²)	Throughput (million outputs/s)	clk multiplied by clks in one cycle (ns)
1	423.5	1219.9	3487.5	23.041	347.2
2	328.9	1740.2	5170.8	42.328	189.0
4	306.9	2847.3	9205.5	74.212	107.8
8	313.5	4643.8	16082.6	118.519	67.5
16	286.8	7794.1	29689.3	217.391	36.8



7. In Part 2, we defined parallelism for this system as having P parallel multiply accumulate units operating on P outputs concurrently. However, this limits us to usingonly M multipliers per layer. If you wanted to parallelize further, what could you do?

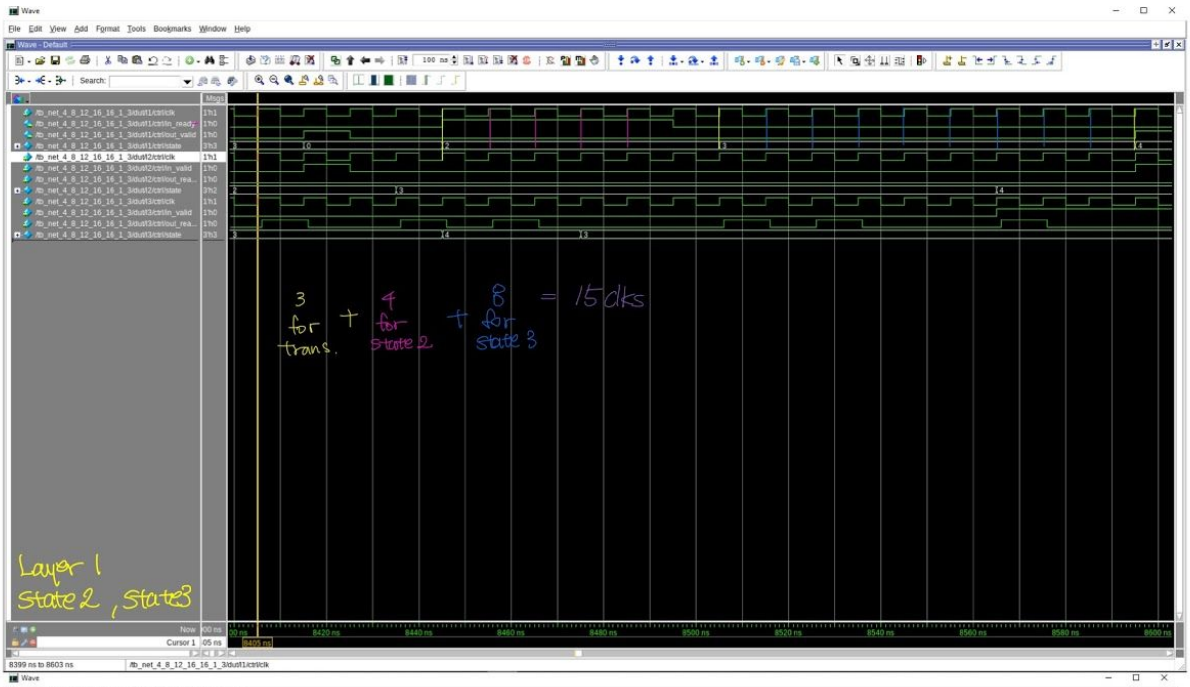
One way is to instead of using multiply and accumulate units, use N multipliers to process each row element concurrently and use M adders at the end to add all N values for each of the M outputs. This way, we save the time of passing each value sequentially into the MAC unit by doing N multiplications at the same time.

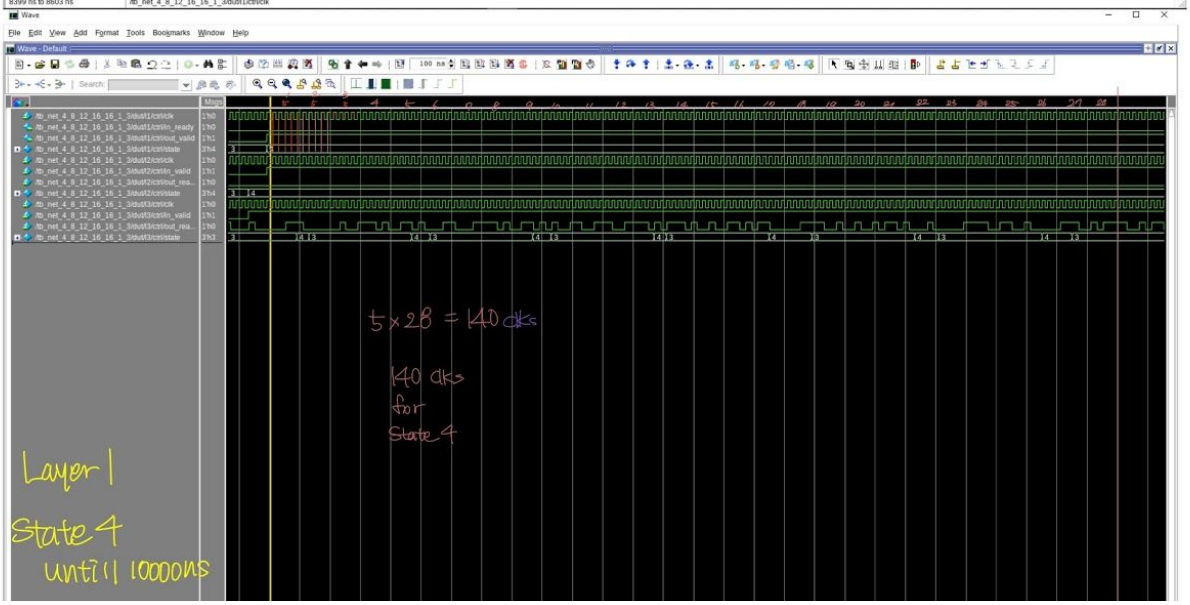
8. In Part 3, you were tasked with figuring out how to best optimize the parallelism parameters for your three-layer design, given a multiplier budget B. Explain how you did this, and why your approach is a good solution. Do you see any drawbacks to your approach? How did you evaluate whether or not your system worked well?

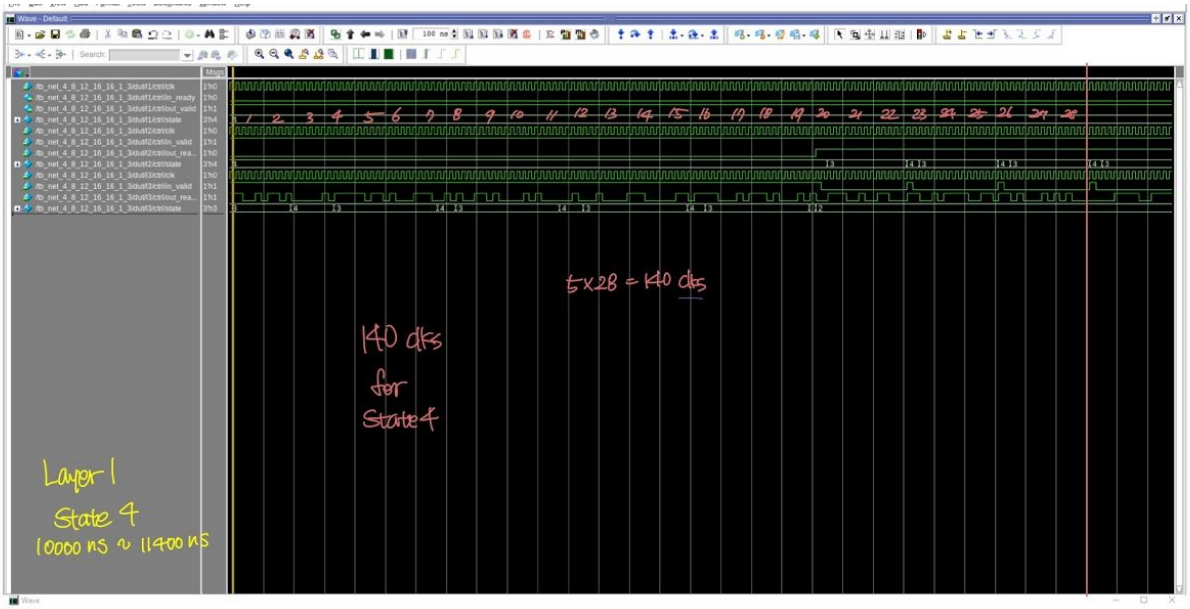
Since the overall system's latency will depend on the layer with the highest latency, to determine the best P for each layer, our logic takes all three P values and always optimizes the layer with the greatest M/P value, which is the number of rounds to go through to generate all M outputs with P compute units. While the remaining number of MAC units are enough, our logic continues to optimize the layer with the highest P until P is equal to the M value of that layer, by which that layer is considered fully parallelized along the output dimension. This solution works well because it ensures that the system only utilizes the amount of MAC units needed, and does not produce extreme results where one layer is fully parallelized while the other is poorly parallelized. One drawback of this approach is that we only considered the P and M values. If there is a case where one layer has a very large N and a small M and the other has a slightly larger M but a very small N, it might be better to optimize the layer with the larger N values because passing N values to the MAC units also take time and we want to overlap this time as much as possible.

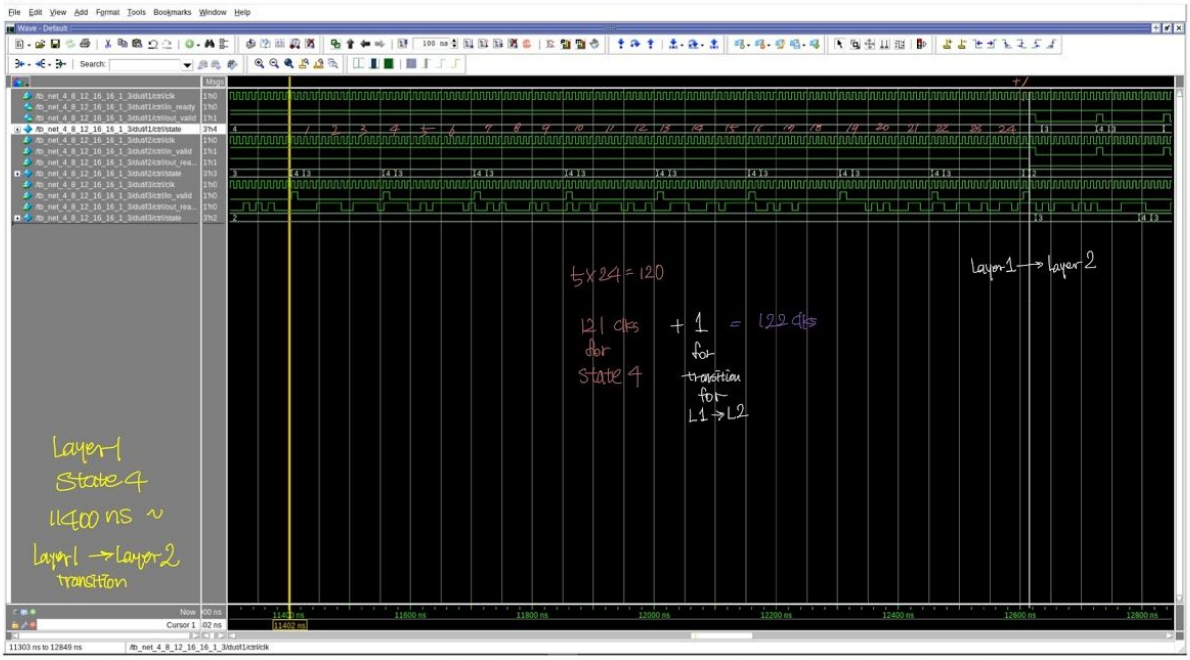
9. Lastly, we will evaluate the optimization method you developed for Part 3. For this experiment, set N=4, M1=8, M2=12, and M3=16. Set T=16 and R=1. Then, generate and synthesize five different designs, with B =3, 10, 20, 30, 36, and 50. Graph (1) power versus B, (2) area versus B, and (3) throughput versus B.

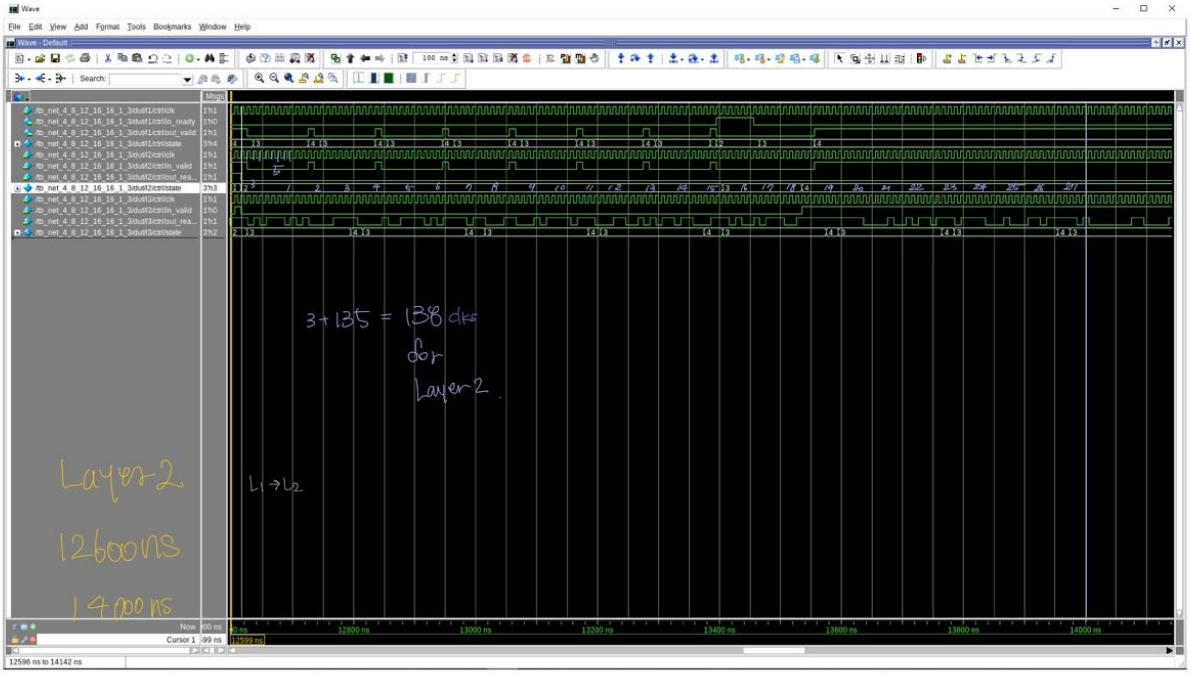
For net 4 8 12 16 1 3:

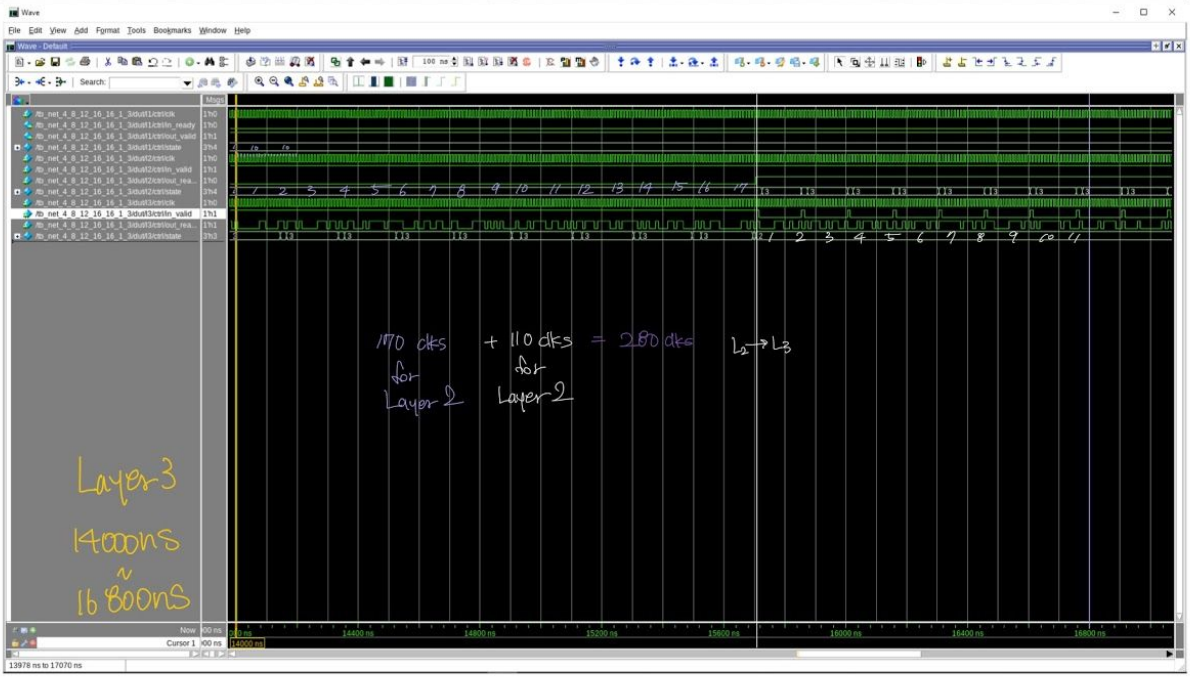


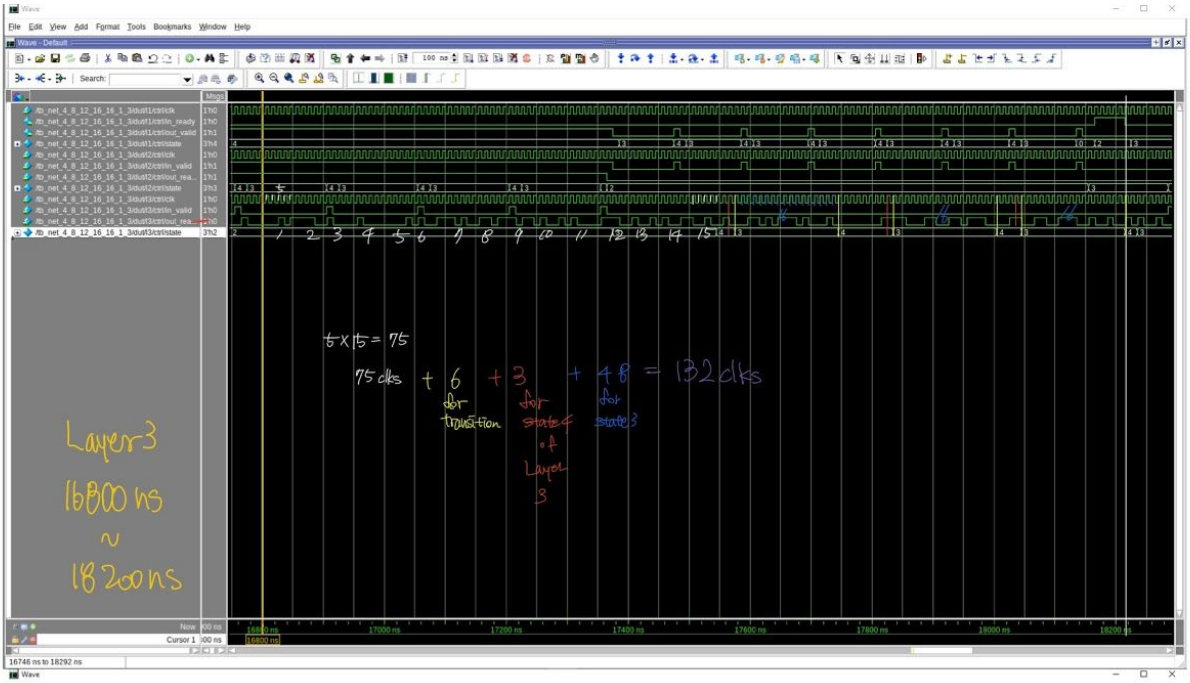


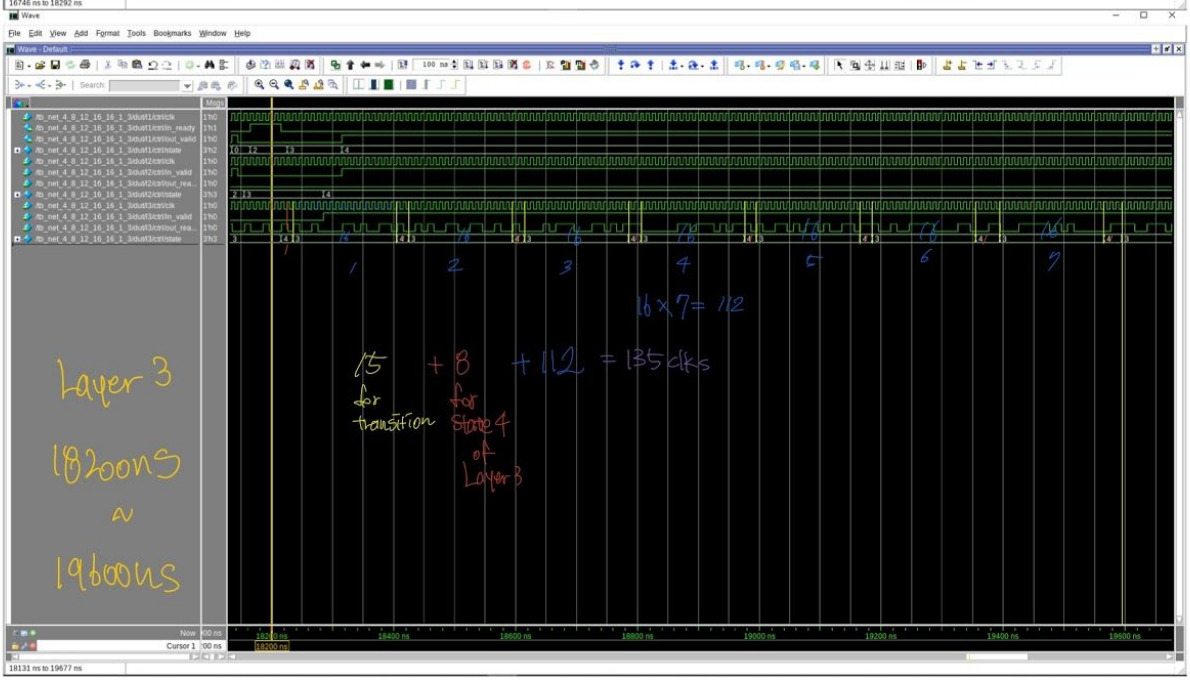

5x28 = 140 clks
140 clks
for
State 4
Layer 1
State 4
10000 ns ~ 11400 ns

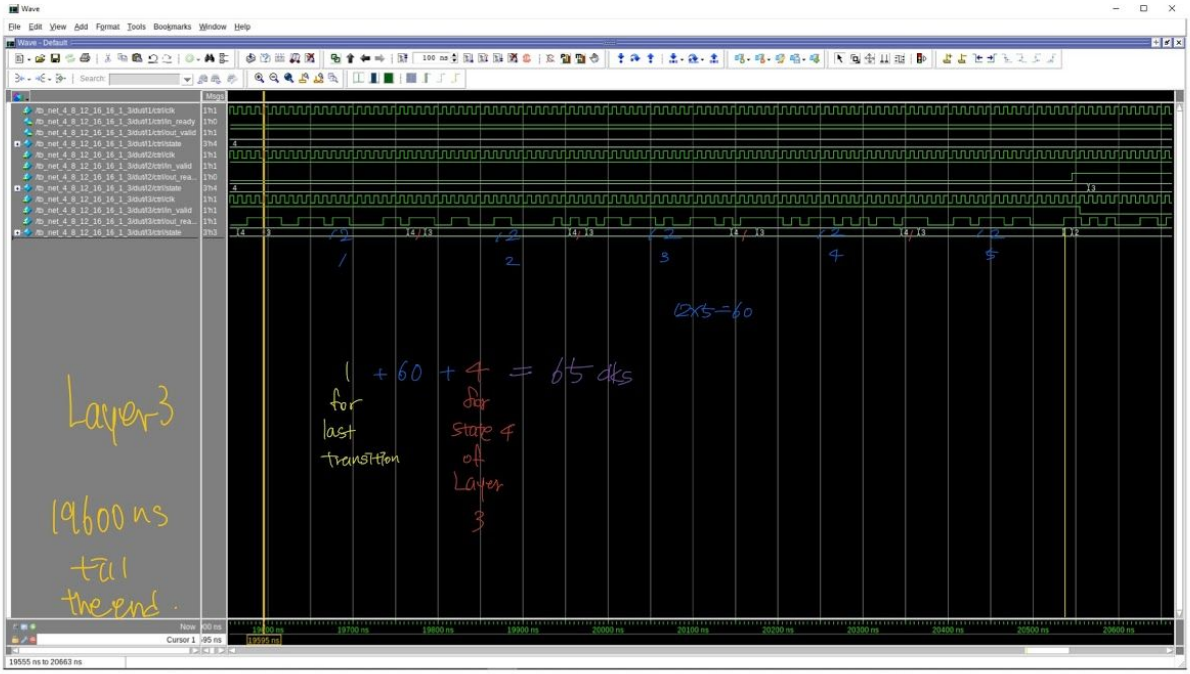

5x24 = 120
121 clks
for
State 4
+ 1
for
transition
for
L1 → L2
= 122 clks
Layer 1 → Layer 2
Layer 1
State 4
11400 ns ~
Layer 1 → Layer 2
transition


3+135 = 138 clks
for
Layer2
Layer2
12600ns
14000ns
L1→L2


1170 clks
for
Layer 2
+ 110 clks
for
Layer 2
= 280 clks
L2→L3
Layer3
14000ns
~
16800ns

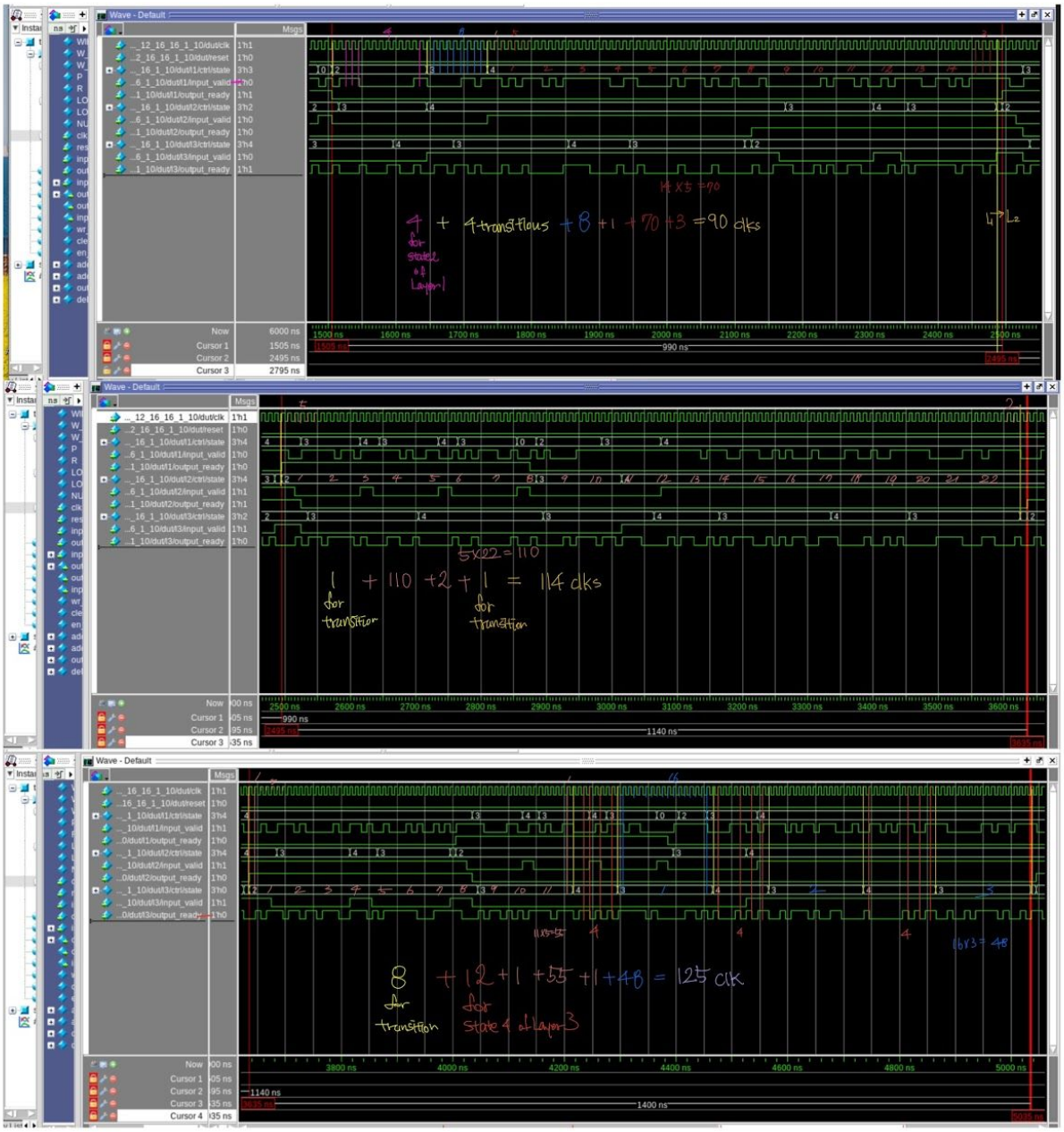

5 X 15 = 75
75 clks + 6 for transition + 3 for state 4 of Layer 3 + 48 for state 3 = 132 clks
Layer3
16800 ns
~
18200 ns


16 x 7 = 112
15 for transition + 8 for state 4 of Layer 3 + 112 = 135 clks
Layer 3
18200ns
~
19600ns



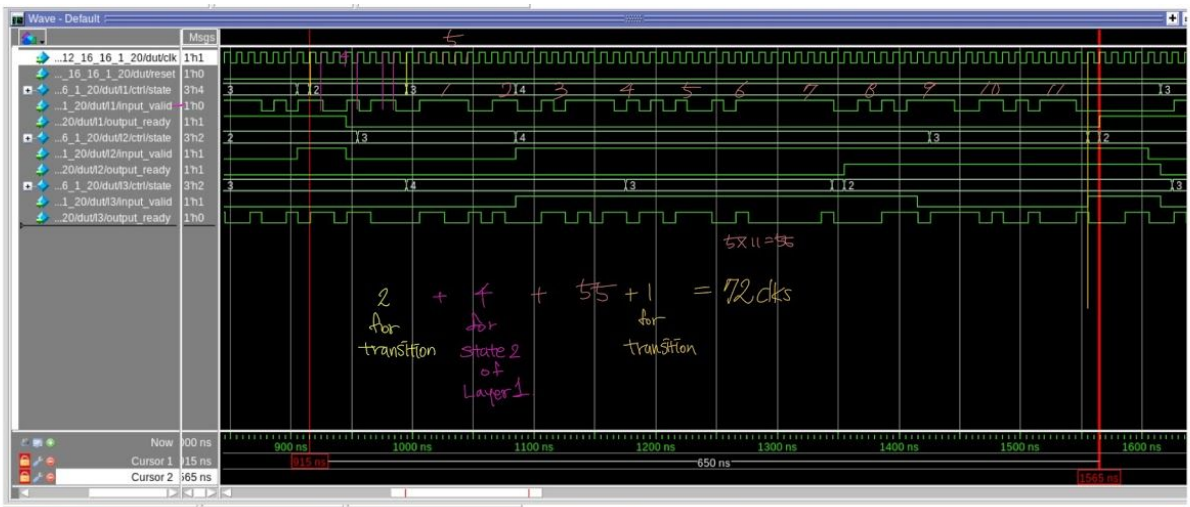
$15 + 140 + 140 + 122 + 138 + 280 + 132 + 135 + 65 = 1167$ clks

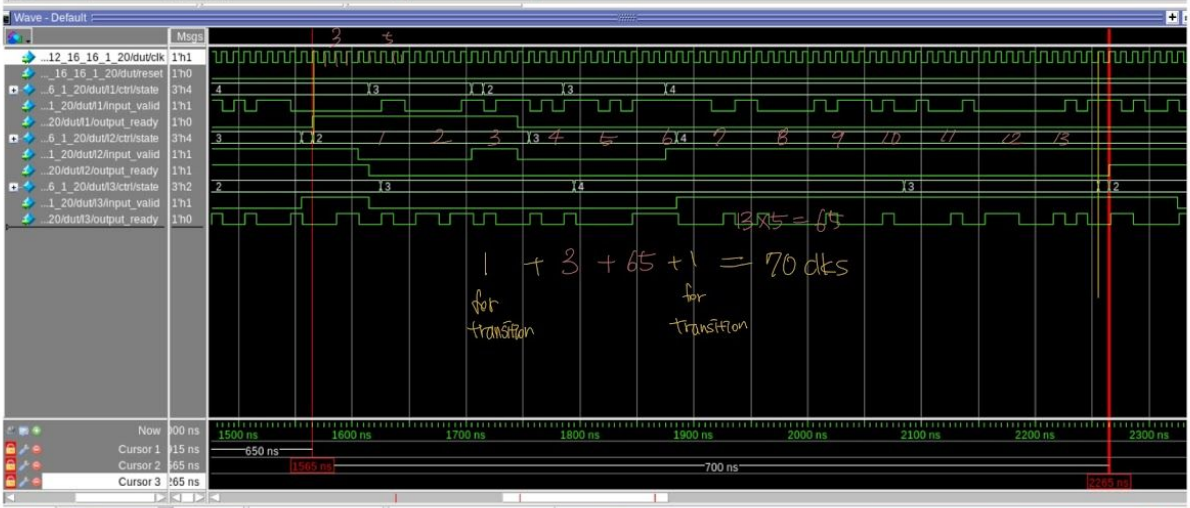
For net 4 8 12 16 1 10:

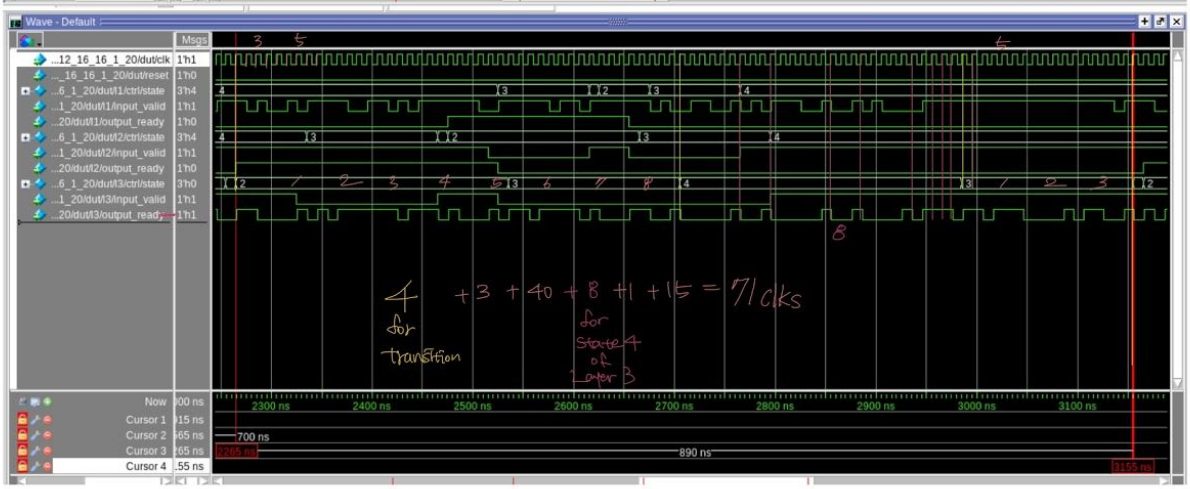


90 + 114 + 125 = 329 clks

For net 4 8 12 16 1 20:

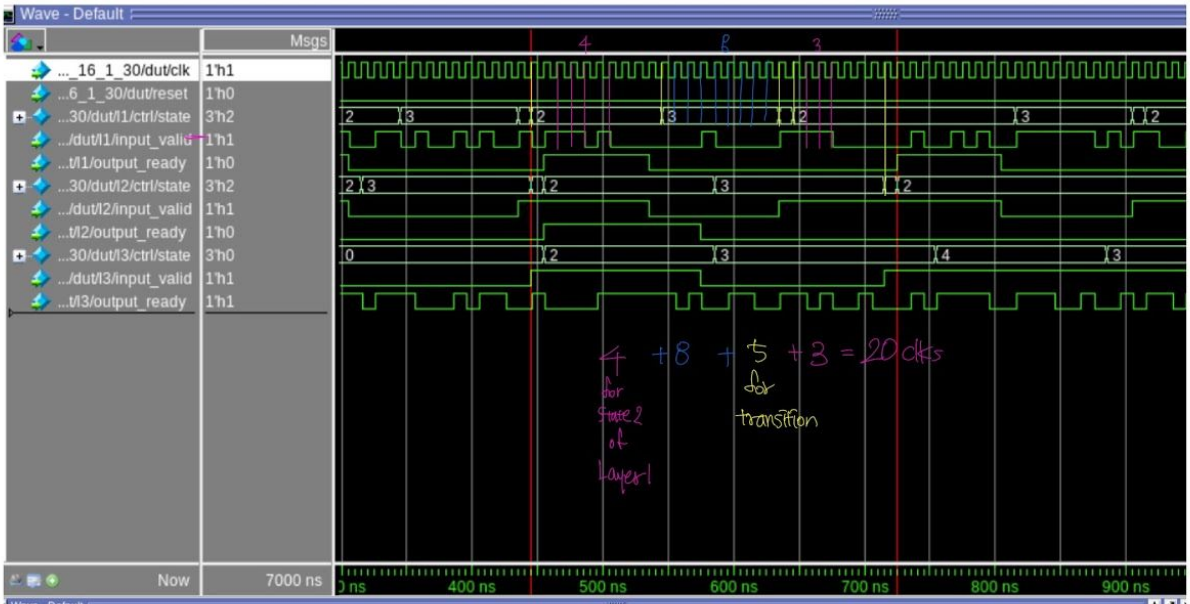


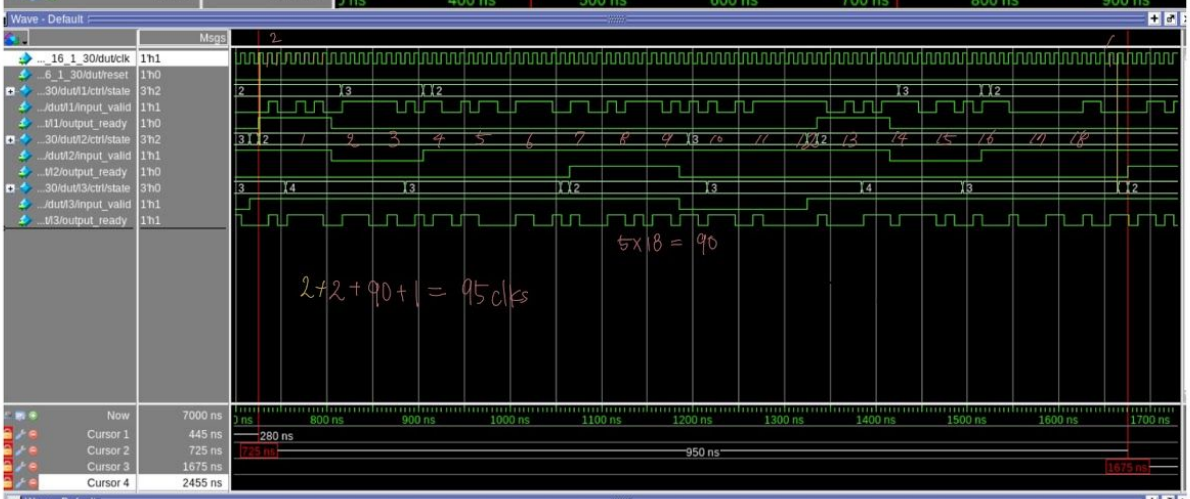


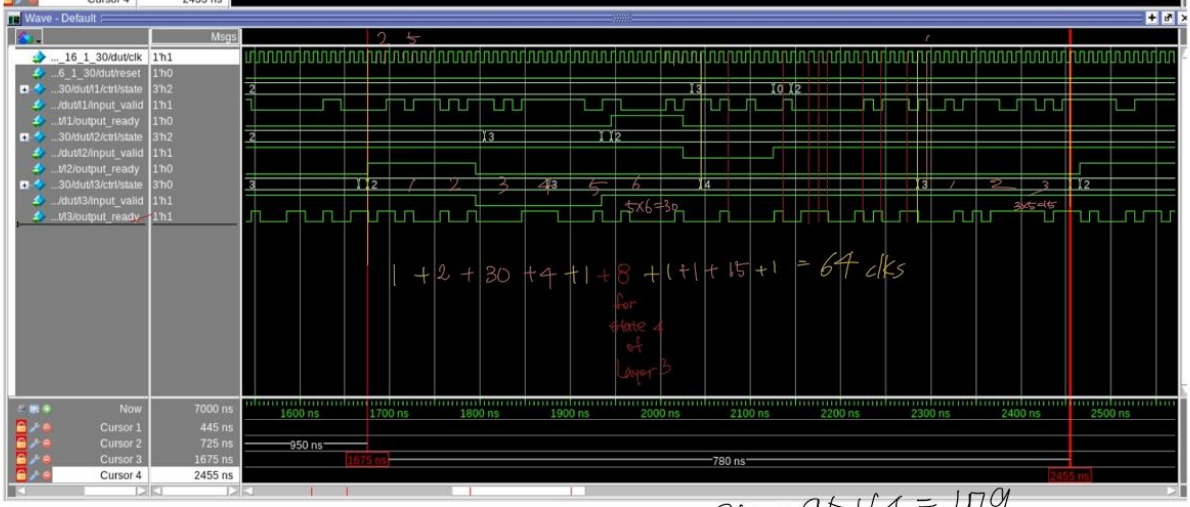


72 + 70 + 71 = 213 clks

For net 4 8 12 16 1 30:

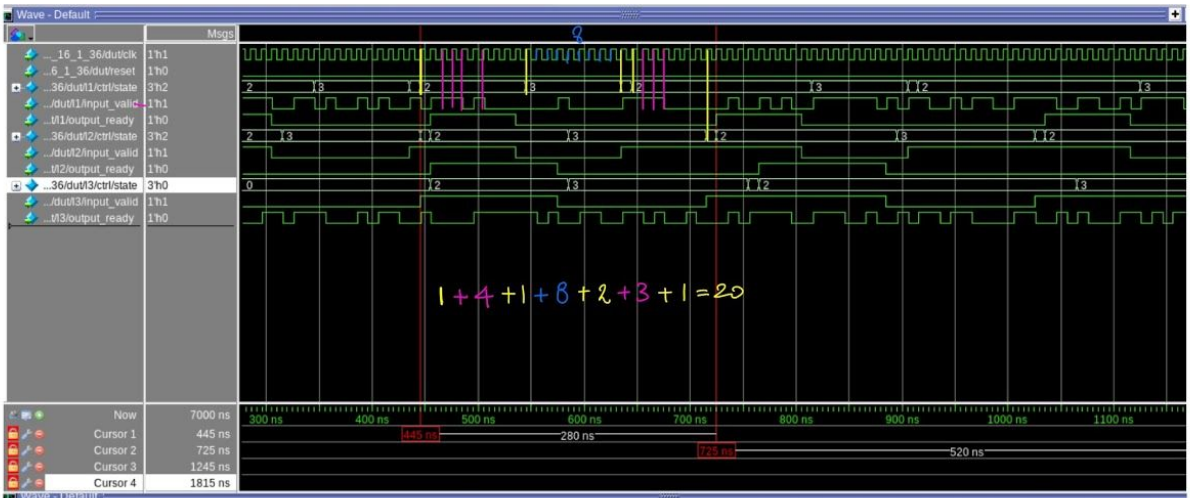


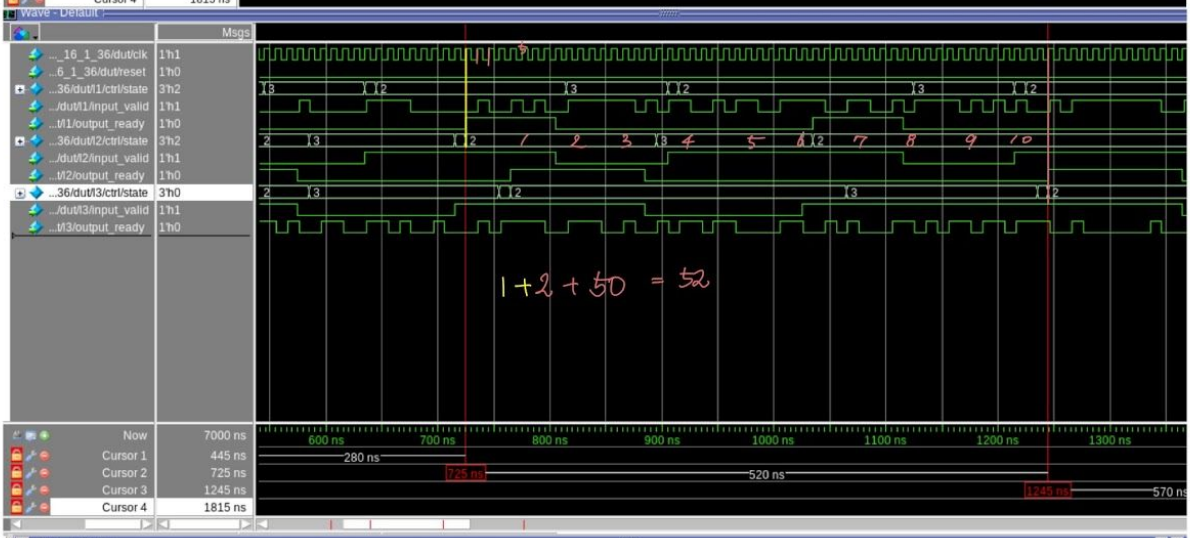


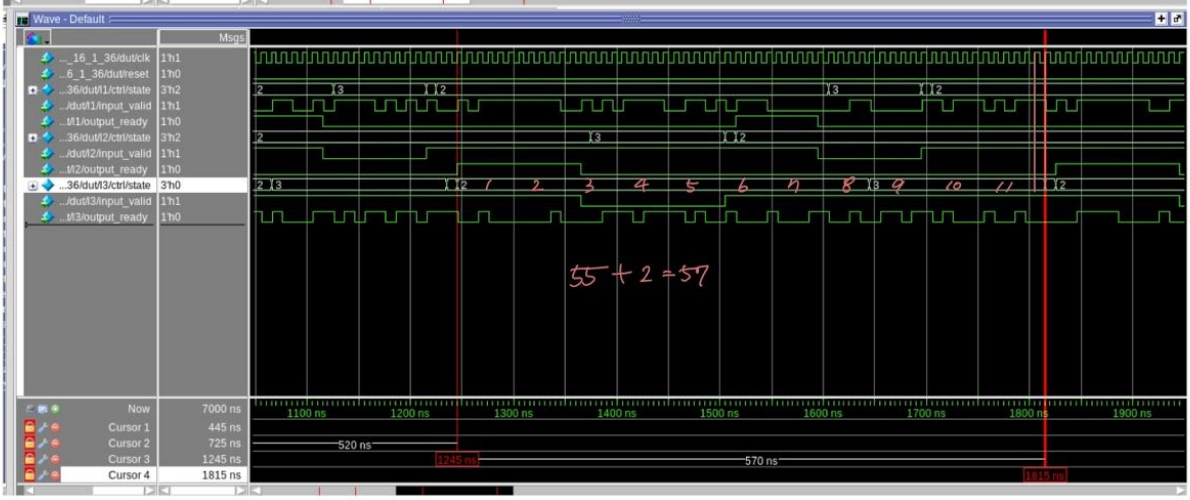


20 + 95 + 64 = 179

For net 4 8 12 16 1 36:

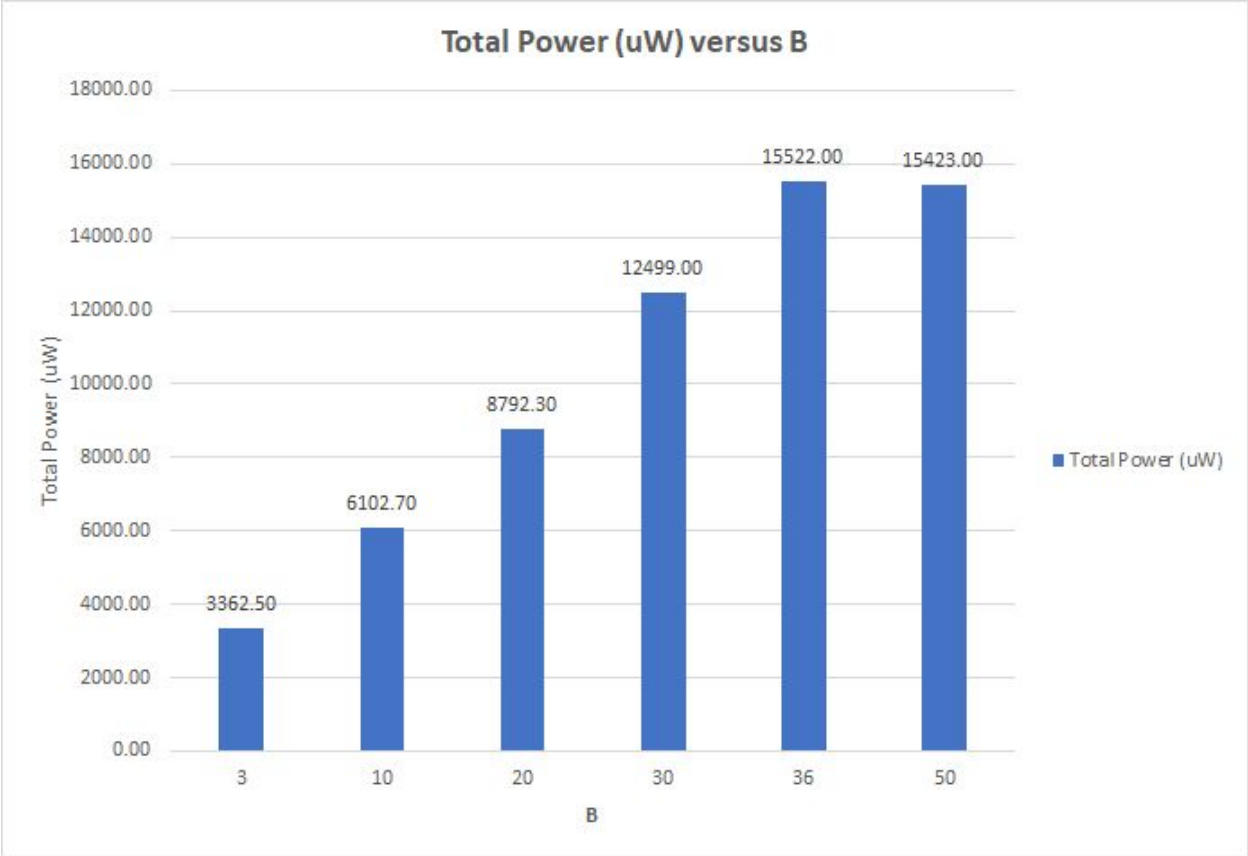


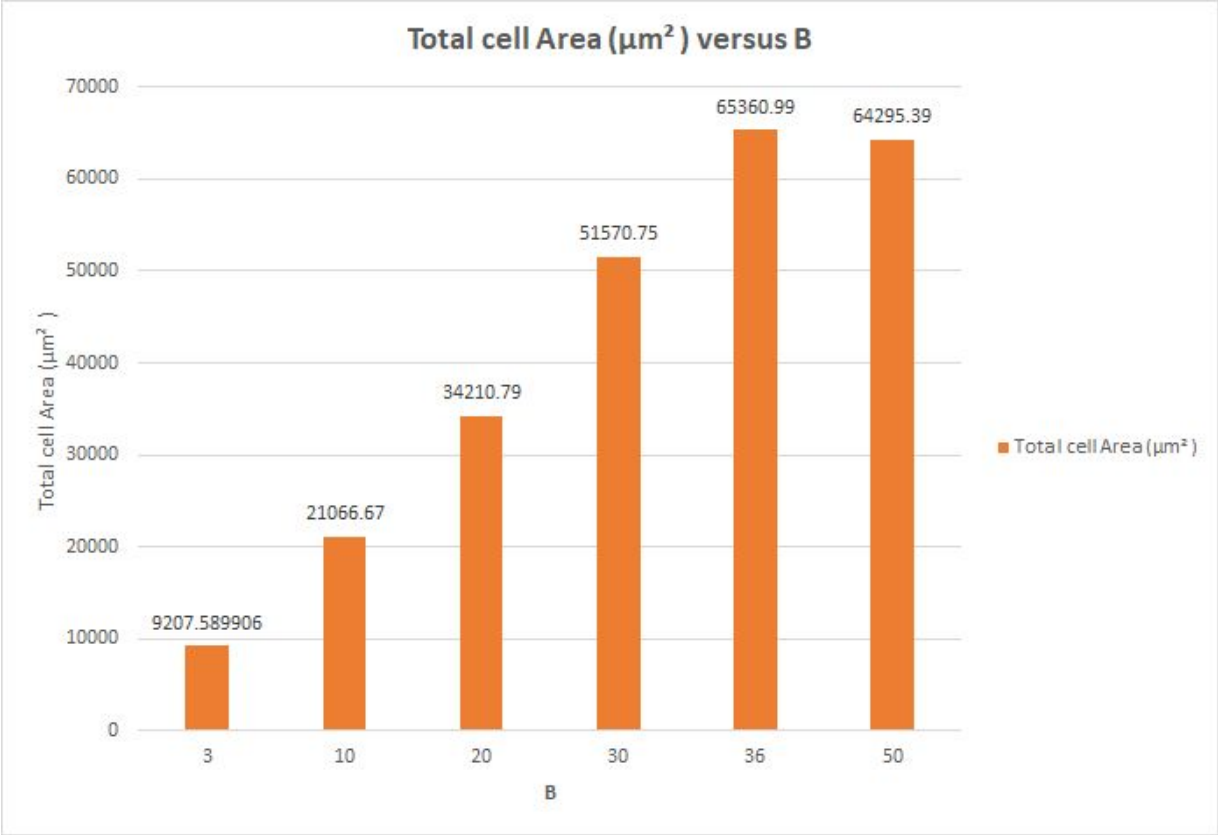


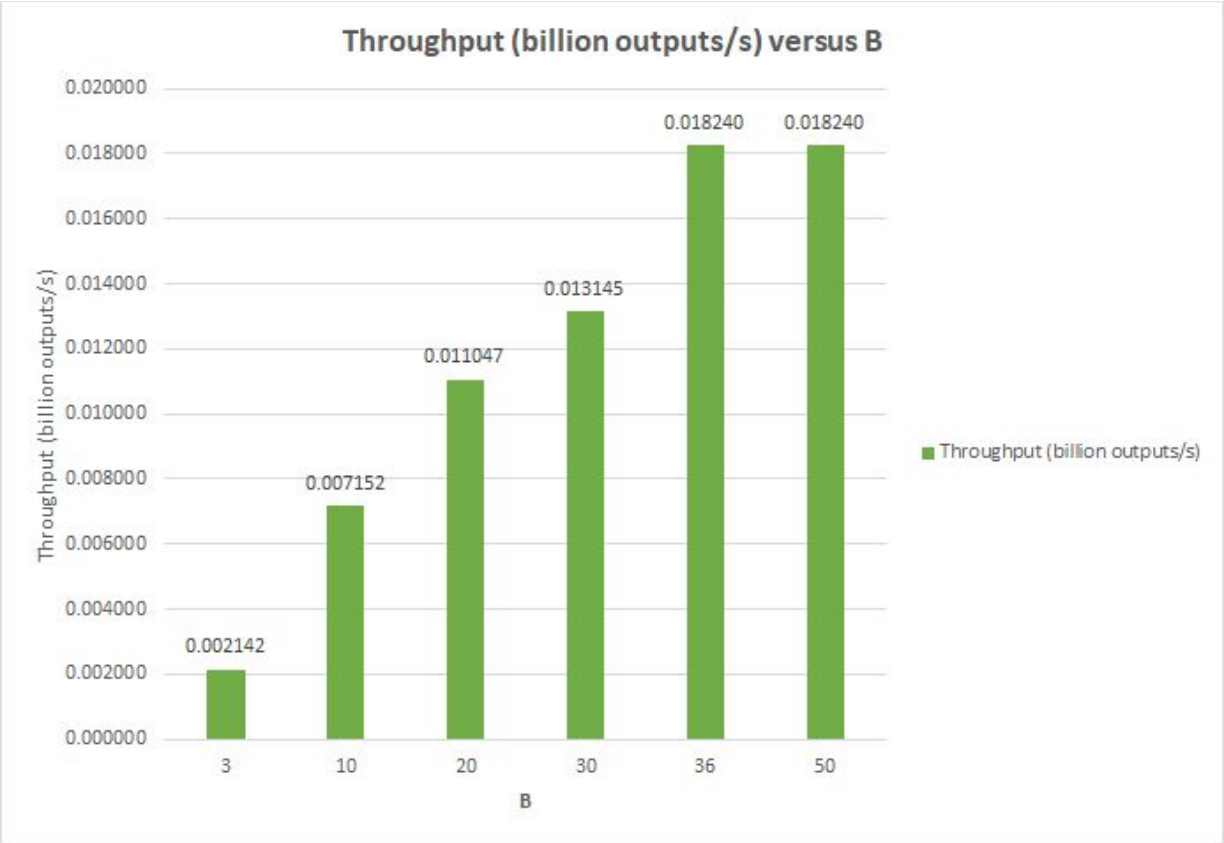


20 + 52 + 57 = 129

N M1 M2 M3 T R B	Total Power (uW)	Total cell Area (µm²)	Frequncy (MHz)	Clk (ns)	cycle (clks)	Throughput (billion outputs/s)
4 8 12 16 1 3	3362.50	9207.589906	625.00	1.60	1167	0.002142
4 8 12 16 1 10	6102.70	21066.67	588.24	1.60	329	0.007152
4 8 12 16 1 20	8792.30	34210.79	588.24	1.70	213	0.011047
4 8 12 16 1 30	12499.00	51570.75	588.24	1.70	179	0.013145
4 8 12 16 1 36	15522.00	65360.99	588.24	1.70	129	0.018240
4 8 12 16 1 50	15423.00	64295.39	588.24	1.70	129	0.018240




Total cell Area (µm²) versus B
70000
60000
50000
40000
30000
20000
10000
0
Total cell Area (µm²)
9207.589906
21066.67
34210.79
51570.75
65360.99
64295.39
3
10
20
30
36
50
B
Total cell Area (µm²)


Throughput (billion outputs/s) versus B
0.020000
0.018000
0.016000
0.014000
0.012000
0.010000
0.008000
0.006000
0.004000
0.002000
0.000000
Throughput (billion outputs/s)
0.002142
0.007152
0.011047
0.013145
0.018240
0.018240
3
10
20
30
36
50
B
Throughput (billion outputs/s)

10. In Part 3, your system assumed it would produce a system with exactly three layers. How would you adapt your generator to create a system with an arbitrary (user specified) number of layers? What challenges would it create?

If we want to generate a network with an arbitrary number of layers, one way is to create a lot of loops and use variables to store parameters from the previous layer to compute corresponding parameters for the current layer. One challenge this creates is if there are repeated layer specifications. For example, if there are 50 layers and only the first and last layer have different layer parameters, then generating 48 identical modules would be a waste of time and error prone, especially in our case where each layer has different ROM values.

11. If you worked with a partner: please explain each partner's contribution to the project. Be specific about what each partner did. (If you worked alone, skip this.)

Xinqi - System design simulation and debugging for all parts, Part 3 P optimization, report questions
Jinkyu - C program simulation for all parts, design syntheses, report questions