ESE 507 - Fall 2015
Proj 3 report

Pavan Vyas, Sushal Penugonda

1. In hardware generators, scalability and flexibility can be difficult. In your report, explain how you made your generator capable of handling the flexibility required by the parameters (k, p, b, g). Were any of these particularly easy or difficult to support? How did you handle the fact that there were no maximum defined values on k and b? Are there practical limits on these values? If so, what are they, and why?
Ans.
In our generator we are printing out a systemverilog design that is parameterized to k and b. Since p can only take two values 1 or k; and g only takes two values 0 or 1, we use if-else blocks in our gen.c file to generate relevant code.
Since no upper limits were imposed on k, b we defined any variable in system verilog depend on k or b as logic[n-1:0], where n is k or b. By parameterizing the variable in this manner there are no limitations on the design. But variable that are dependent on k or b in gen.c, especially for generating reference values for the test vectors.
For k, the data structure 'int' was used.We use a counter int j for counting k^2. In this scenario k is limited by the bounds of j. As j is int: $j < 2^{31} - 1,\ j < 2147483647$. So, $k < \sqrt{j},\ k < 46340$ This limit can be increased by using 'long long j' and 'long k'

The below statement is the limiting factor for b. Overflow variable is used to handle overflow conditions in generating test bench outputs.

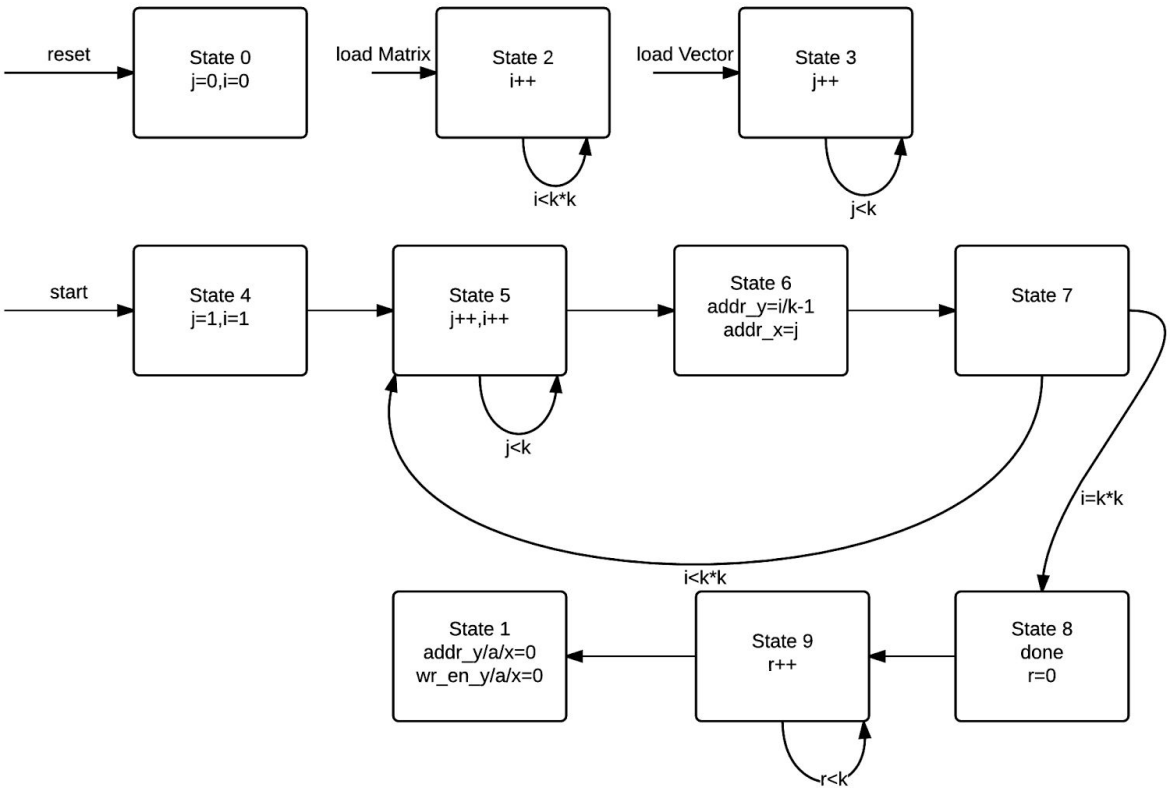
long long overflow = pow(2,b)*pow(2,b);

$overflow \leq 9223372036854775807,\ Hence\ b \leq log_2(overflow_{max})/2,\ b \leq 31$

2. Describe how you designed your control module/FSM. How did you structure the design so that it can be changed as the k parameter changes? Explain how the structure changes as k grows.

Ans.

State diagram when p=1, states 5,6,7 are structured differently when p=k



As shown above, we made our counters in all states dependent on parameter k. In this manner, we can use the same control module code for different values of k. As k grows, the counters grow along with it. Some counters grow linearly, some counters grow in a quadratic manner.

3. Describe how you implemented the parallel (p=k) designs. Explain your structure. What extra logic and storage elements did you need to add? Did you find any clever optimizations to reduce cost?

Ans.

Parallelism was implemented by using existing memory modules. In our matrix A of our data path, we replaced the kxk memory module with k memory modules of size kx1. The data path was also modified to have the size of mem y to hold only one number The control module correctly routed incoming data values of A into the correct memory module. We simply used k such data paths to enable parallelism without changing our entire design. We also added code to our control module to select the correct memory y, after the done pulse.

4. Explain your testbench strategy. How do your testbenches work? Justify why your testbenches test the designs sufficiently. How difficult was it to incorporate the flexibility of the various parameters into the testbench itself?

Ans.

Our test bench asserts numerous combinations of four different types of valid inputs. Since the invalid inputs could not be enumerated and since keeping track of a completely random pattern of inputs for verification is complex, we opted to not include invalid assertions to our test bench design. The four different modes of valid inputs are:

1. load matrix followed by a random time gap between (0,10) periods, and then a load vector
2. load vector, random time gap, load matrix
3. load matrix
4. load vector

After each of these cases we are waiting for a random time and asserting the start signal.

The testbench then waits for the done signal before waiting again for a random time. The process repeats itself for the specified number of iterations.

Note that we have successfully passed the milestone test bench provided and we have tested a few of our designs with a fully random signal generator to ensure that we weren't getting stuck in any state.

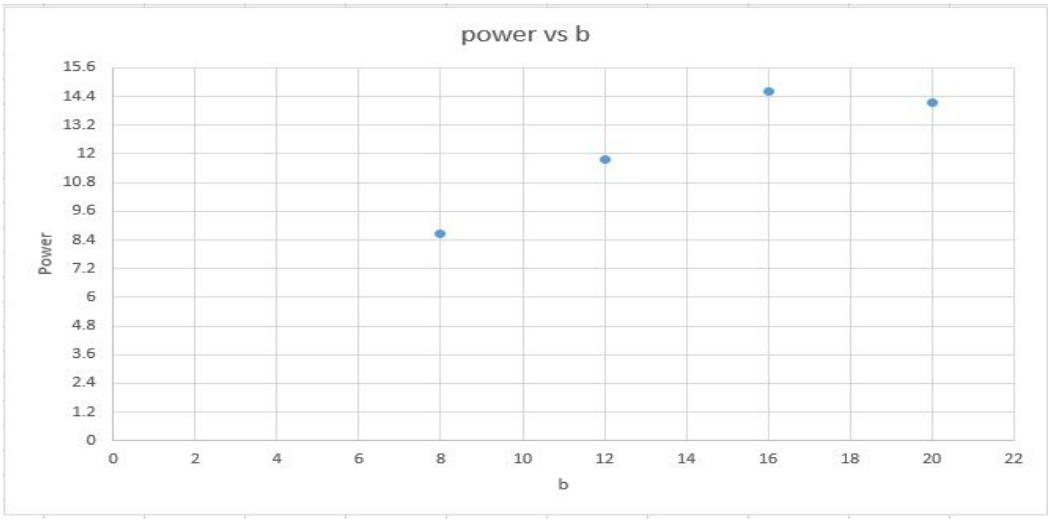
5. In Section 3, we discussed how the parameters (k, p, b, g) allow various tradeoffs between problem size, costs, precision, and performance. Explain how these tradeoffs work at a conceptual level. In other words, explain how you would expect changing each parameter to affect these metrics. Be specific
Ans.

	k	p	b	g
problem size	polynomial increase	remains the same	exponential increase	remains the same
costs	polynomial memory increase	increases k times	exponential increase in counter size	slight increase due to extra register
precision	unchanged	unchanged	unchanged	unchanged
performance	decrease	increase	decrease	increases

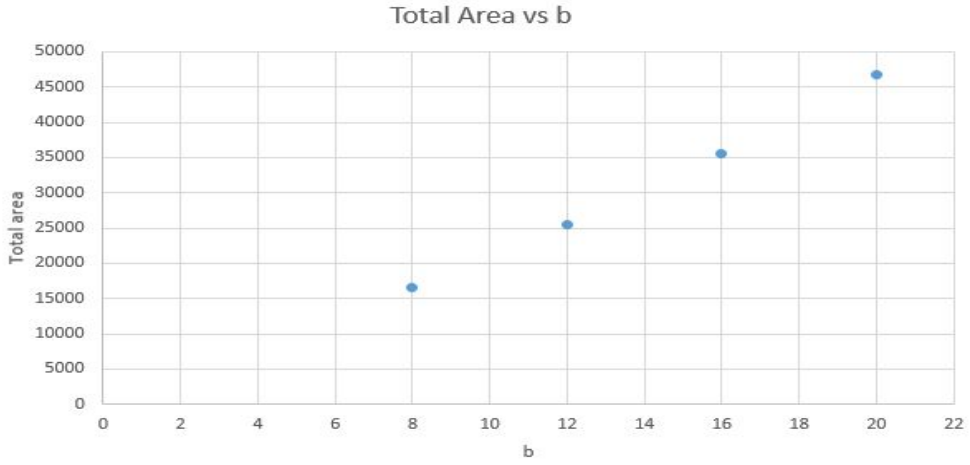
6. Now, we will use synthesis to evaluate how the area and power of an implementation scale as the input precision b changes. Use your generator to produce four designs with b=8, 12, 16, and 20, while you keep k=8, p=8, and g=1. Page 7 Then produce two graphs that illustrate: (1) power versus b and (2) area versus b for these designs. Describe where the critical path is located in each design.

Ans.

power increases with the increase in the value of b.



Area increases with the increase in memory sizes



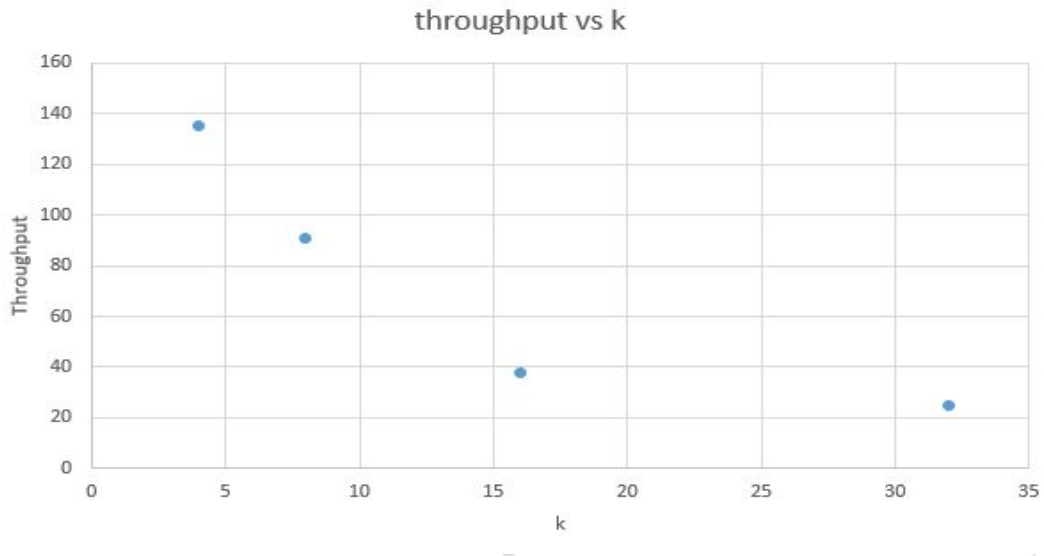
Design	Critical_Path
8_8_8_1	data_path7/memX/data_out_reg[1] => data_path7/mult_r_sig_reg[14]
8_8_12_1	data_path2/add_r_sig_reg[1] => data_path2/add_r_sig_reg[23]
8_8_16_1	data_path8/memX/data_out_reg[1] => data_path8/mult_r_sig_reg[31]
8_8_20_1	data_path5/memX/data_out_reg[7] => data_path5/mult_r_sig_reg[39]

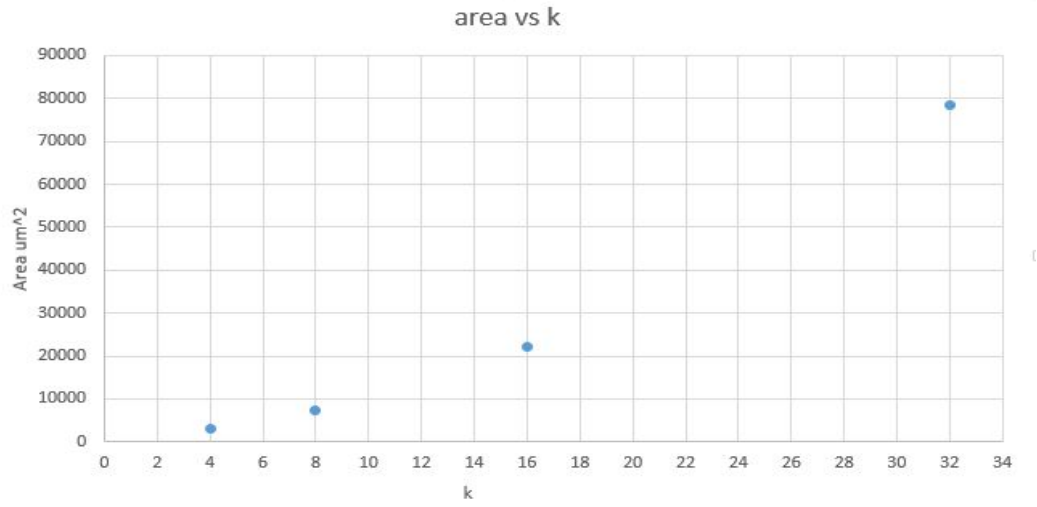
7. Next, we will evaluate how throughput, area, and power scale as k changes. Use your generator to produce four designs with k=4, 8, 16, and 32, while you keep p=1, b=8, and g=1. Synthesize each design, and graph: (1) power versus k, (2) area versus k, and (3) throughput versus k. Does the location of the critical path change as k changes? Throughput is defined as the number of data inputs processed per second. To calculate this, you will first determine the average number of data words per cycle, and multiply this by the clock frequency f. We will calculate the words per cycle assuming that our system keeps a fixed matrix A. (That is, assume that a matrix is stored in memory and that we will use loadVector followed by start for every input.) Under these assumptions, for every MVM computed, your system processes k input words. How many cycles does this computation take? Let c represent the number of cycles needed to process these k input words. Then, your throughput will be (k/c) words per cycle times f cycles per second. In your report, include the values of c that you found for each design, and explain how you found them.

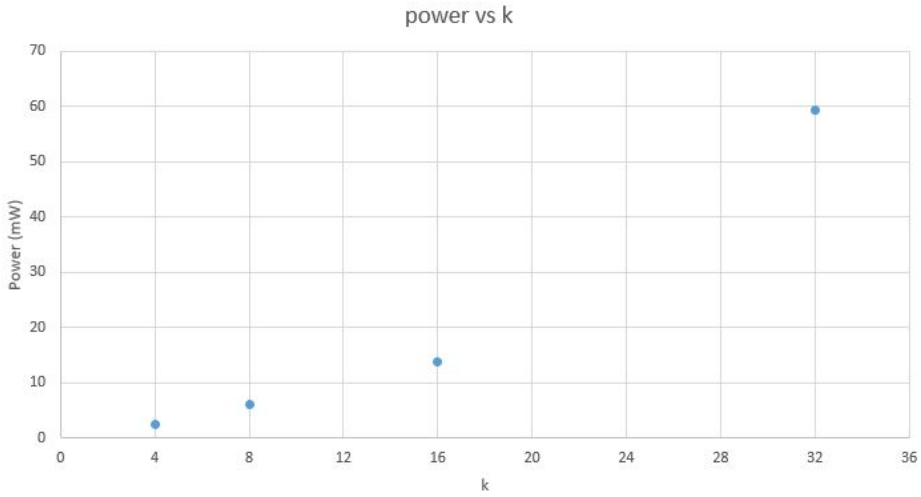
We created a test bench that measured the number of cycles, c, between load vector and done signal. See throughput.sv for source.

k	c	f	Throughput (kf/c)
4	32	1.08GHz	135×10^6
8	94	1.08GHz	91.9×10^6
16	314	740MHz	37.7×10^6
32	1138	892MHz	25×10^6

Design	Critical_path
4_1_8_1	data_path/memX/data_out_reg[4] => data_path/mult_r_sig_reg[12]
8_1_8_1	data_path/memX/data_out_reg[5] => data_path/mult_r_sig_reg[11]
16_1_8_1	ctrl_path/addr_a_reg[1] => data_path/memA/data_out_reg[5]
32_1_8_1	ctrl_path/q_reg[1] => ctrl_path/q_reg[32]


throughput vs k
Throughput
k
160
140
120
100
80
60
40
20
0
0
5
10
15
20
25
30
35


area vs k
Area um^2
k
90000
80000
70000
60000
50000
40000
30000
20000
10000
0
0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34


power vs k
Power (mW)
k
70
60
50
40
30
20
10
0
0
4
8
12
16
20
24
28
32
36

8. In the previous step, you used pipelined designs (g=1). How do the costs and performance change if you use unpipelined designs? Generate and synthesize designs without pipelining using parameters (k, p, b, g) = (4, 1, 8, 0) and (32, 1, 8, 0) and compare these two designs to their pipelined counterparts from question 7. How does the critical path change?

Ans.

	4_1_8_0	4_1_8_1	32_1_8_1	32_1_8_0
freq	909.09 Mhz	1.08 Ghz	892.85 Mhz	833.33 Mhz
area	2932.117 um2	3000.47	78390.99	78400.57 um2
power	1.97 mW	2.41 mW	59.47mW	55.64 mW
critical path	data_path/memX/data_out_reg[5] => data_path/add_r_sig_reg[15]	data_path/memX/data_out_reg[4] => data_path/mult_r_sig_reg[12]	ctrl_path/q_reg[1] => ctrl_path/q_reg[32]	ctrl_path/addr_a_reg[1] => data_path/memA/data_out_reg[7]

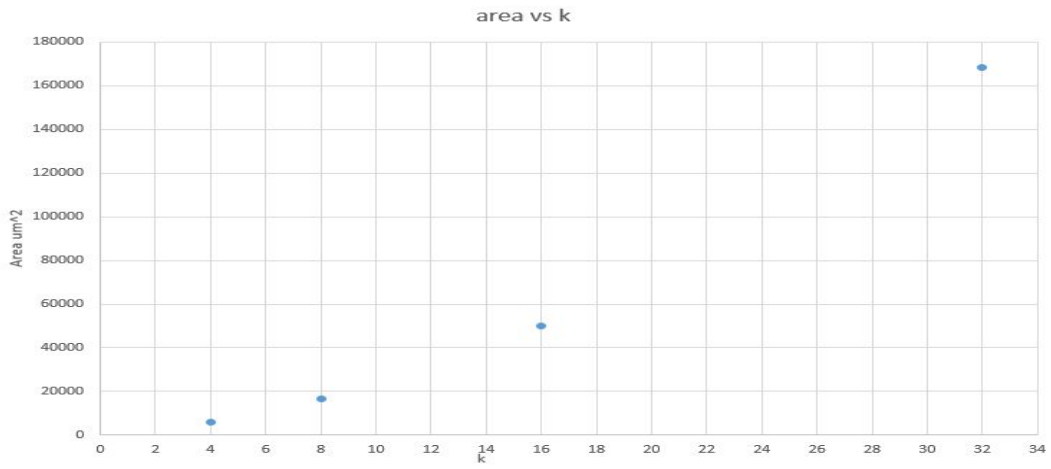
Pipelining does increase the frequency but cost also increases as the area and the power increases.
Performance irrelevant of power increases.

9. Lastly, you need to evaluate how the designs change when you increase parallelism (by setting p=k). Now, generate and synthesize four parallel designs: k=4, 8, 16, and 32, with p=k, b=8, and g=1. Graph: (1) power versus k, (2) area versus k, and (3) throughput versus k. These parallel designs will be faster but more expensive than their counterparts from question 7. Which are more efficient: the more-parallel p=k designs, or the less parallel p=1 designs? Justify your answer quantitatively.

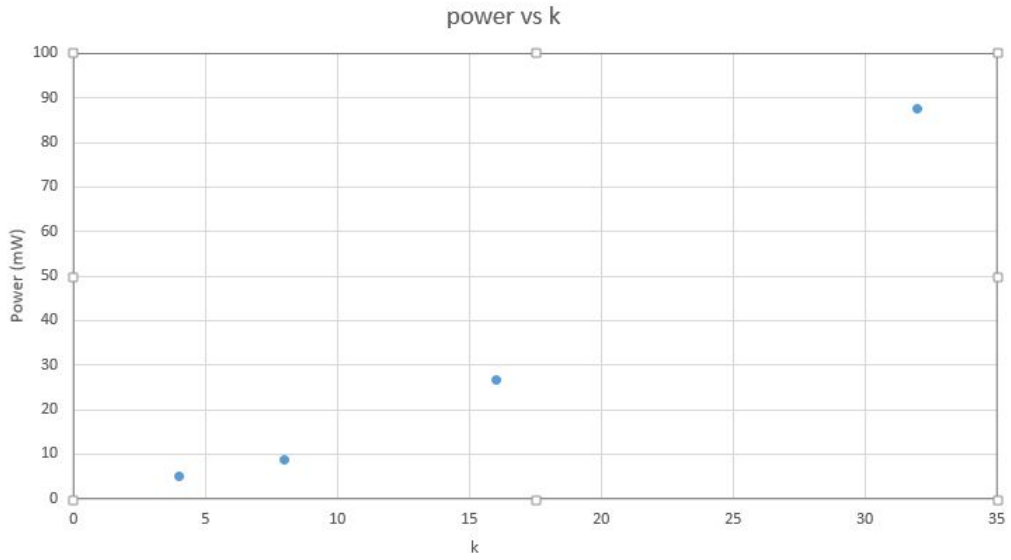
Design	area um2	power mW	throughput
4_4_8_1	5993.77	4.09 mW	378×10^6
8_8_8_1	16455.55	8.68	353×10^6
16_16_8_1	50057.20	26.69	408×10^6
32_32_8_1	168496.10	87.56	416×10^6

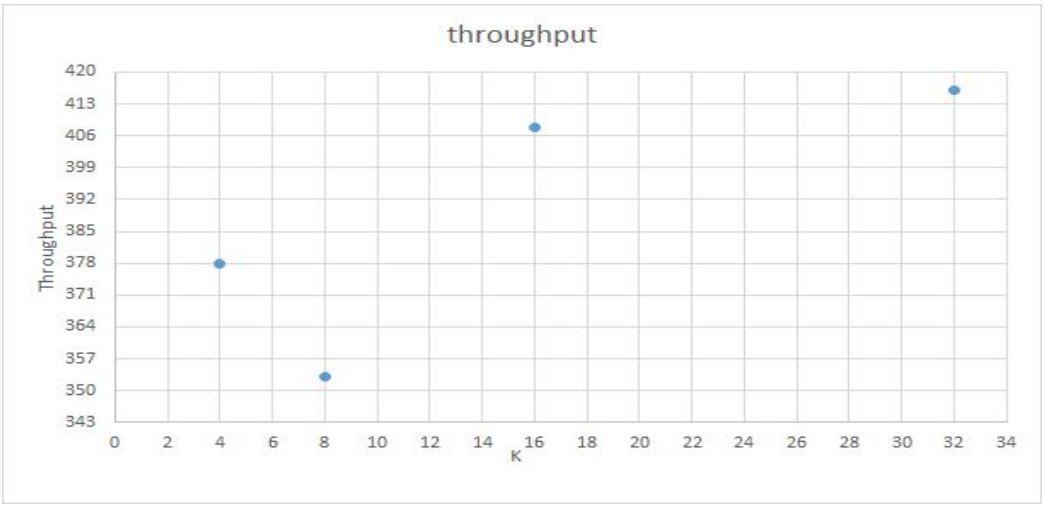
k	performance per watt for p=1	perf. per watt for p=k
4	56	92.42
8	14.82	40.66
16	2.73	15.28
32	0.42	4.75

Parallel designs are more efficient.


area vs k
Area um^2
k

.


power vs k
Power (mW)
k


throughput
Throughput
K