CSE 237C: CORDIC Report

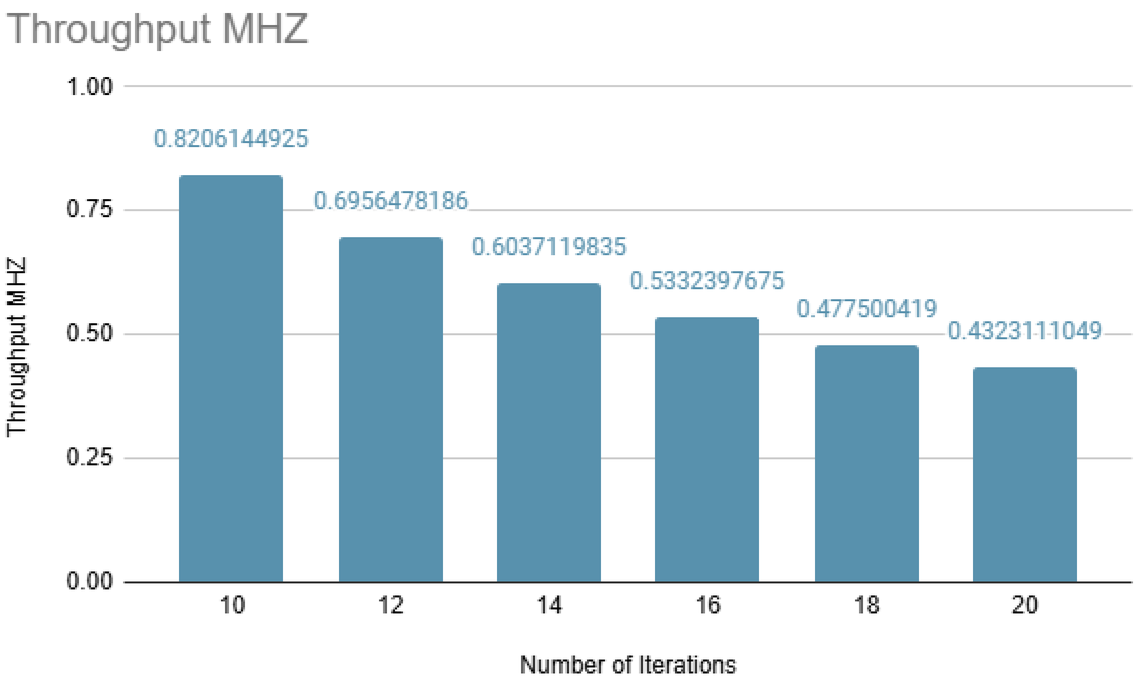
Harish Vasanth, Kaleigh Edusada, Shaurya Raswan

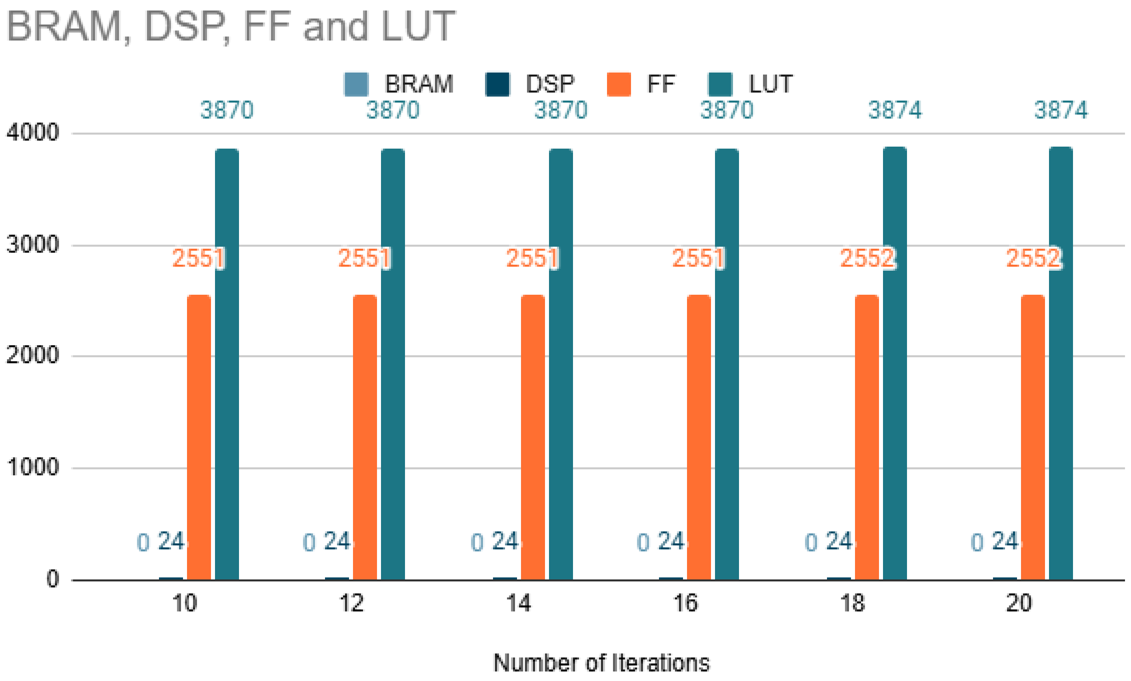
Link to GitHub: **kmogatas/cordic_powerpuffgurls**

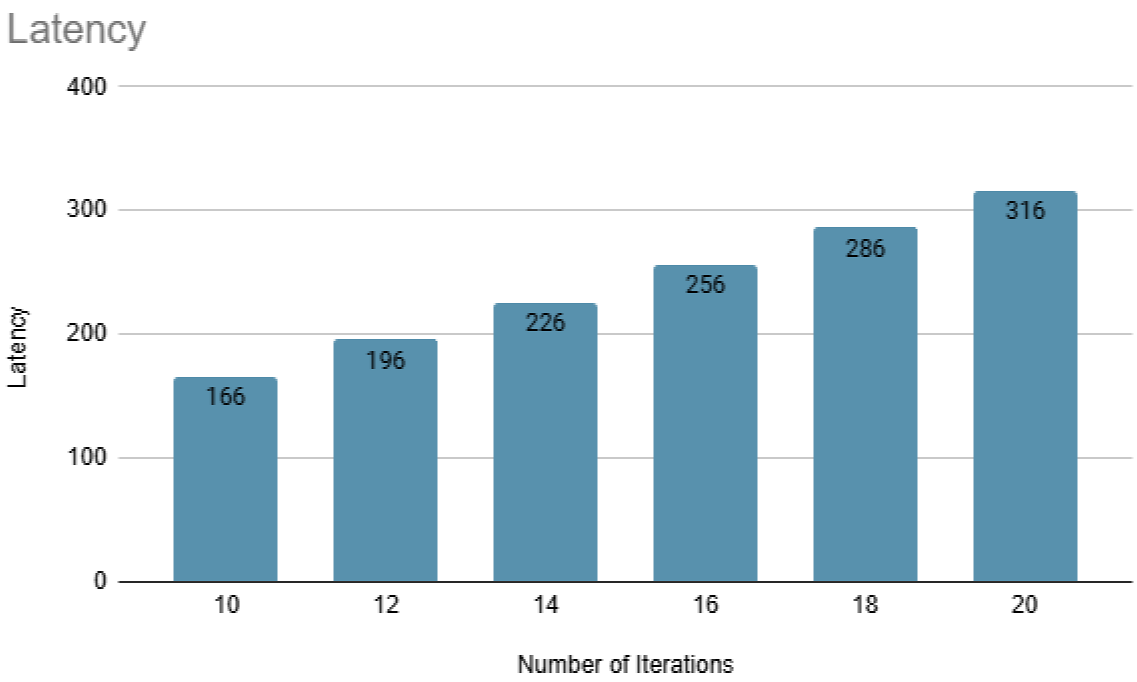
1a.

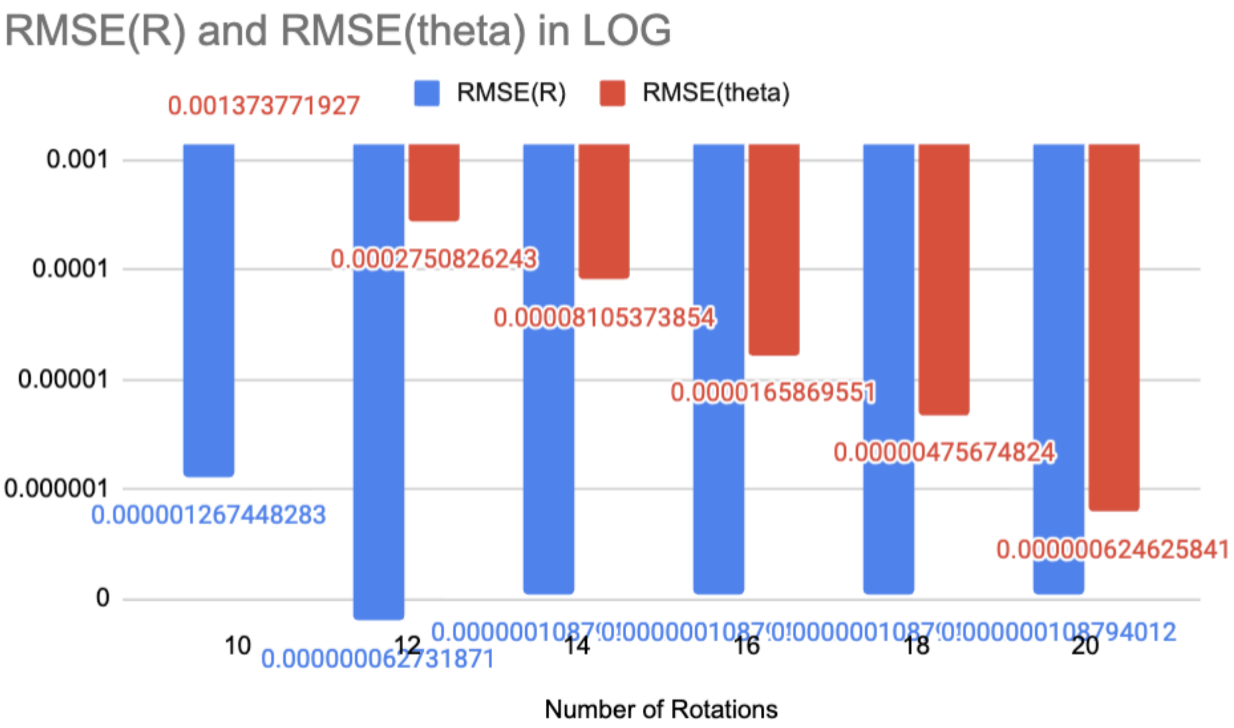
Rotations	10	12	14	16	18	20
BRAM	0	0	0	0	0	0
DSP	24	24	24	24	24	24
FF	2551	2551	2551	2551	2552	2552
LUT	3870	3870	3870	3870	3874	3874
Throughput MHZ	0.82061449254	0.69564781855	0.6037119835	0.53323976752	0.477500419	0.4323111049
Latency min/max	166/166	196/196	226/226	256/256	286/286	316/316
RMSE(R)	0.000001267448283	0.000000062731871	0.000000108794012	0.000000108794012	0.000000108794012	0.000000108794012
RMSE(theta)	0.001373771927319	0.000275082624285	0.000081053738541	0.000016586955098	0.000004756748240	0.000000624625841

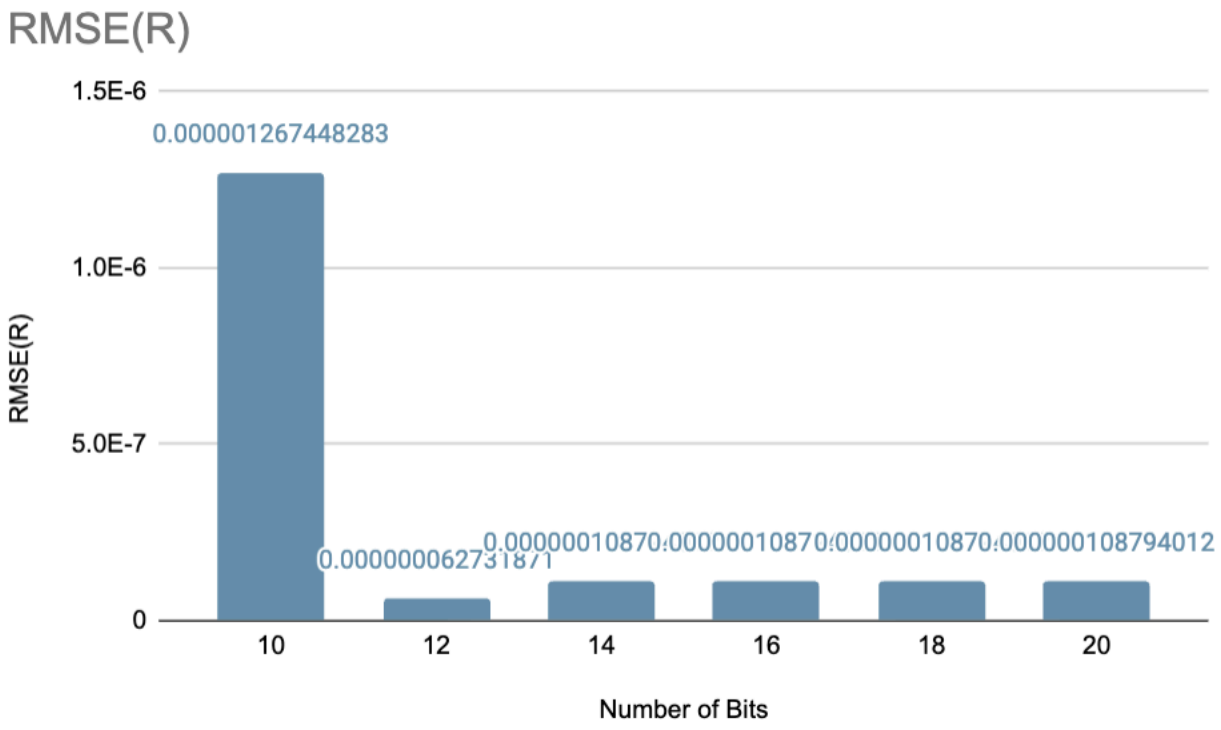
b.

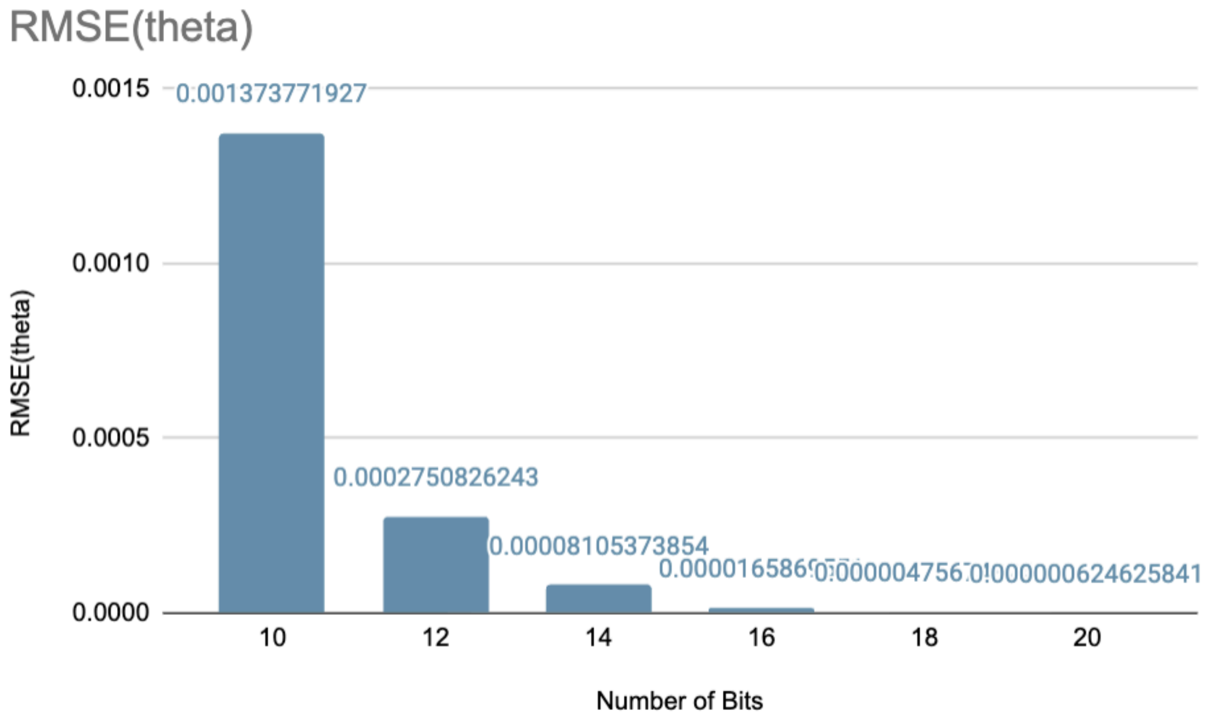

Throughput MHZ
1.00
0.75
0.50
0.25
0.00
Throughput MHZ
0.8206144925
0.6956478186
0.6037119835
0.5332397675
0.477500419
0.4323111049
10
12
14
16
18
20
Number of Iterations


BRAM, DSP, FF and LUT
BRAM
DSP
FF
LUT
4000
3000
2000
1000
0
3870
3870
3870
3870
3874
3874
2551
2551
2551
2551
2552
2552
0 24
0 24
0 24
0 24
0 24
0 24
10
12
14
16
18
20
Number of Iterations


Latency
400
300
200
100
0
Latency
166
196
226
256
286
316
10
12
14
16
18
20
Number of Iterations


RMSE(R) and RMSE(theta) in LOG
RMSE(R)
RMSE(theta)
0.001373771927
0.001
0.0001
0.00001
0.000001
0
0.000001267448283
0.0002750826243
0.00008105373854
0.0000165869551
0.00000475674824
0.000000624625841
0.000000062731871
0.0000001087'0!0000001087'0!0000001087'0!000000108794012
10
12
14
16
18
20
Number of Rotations





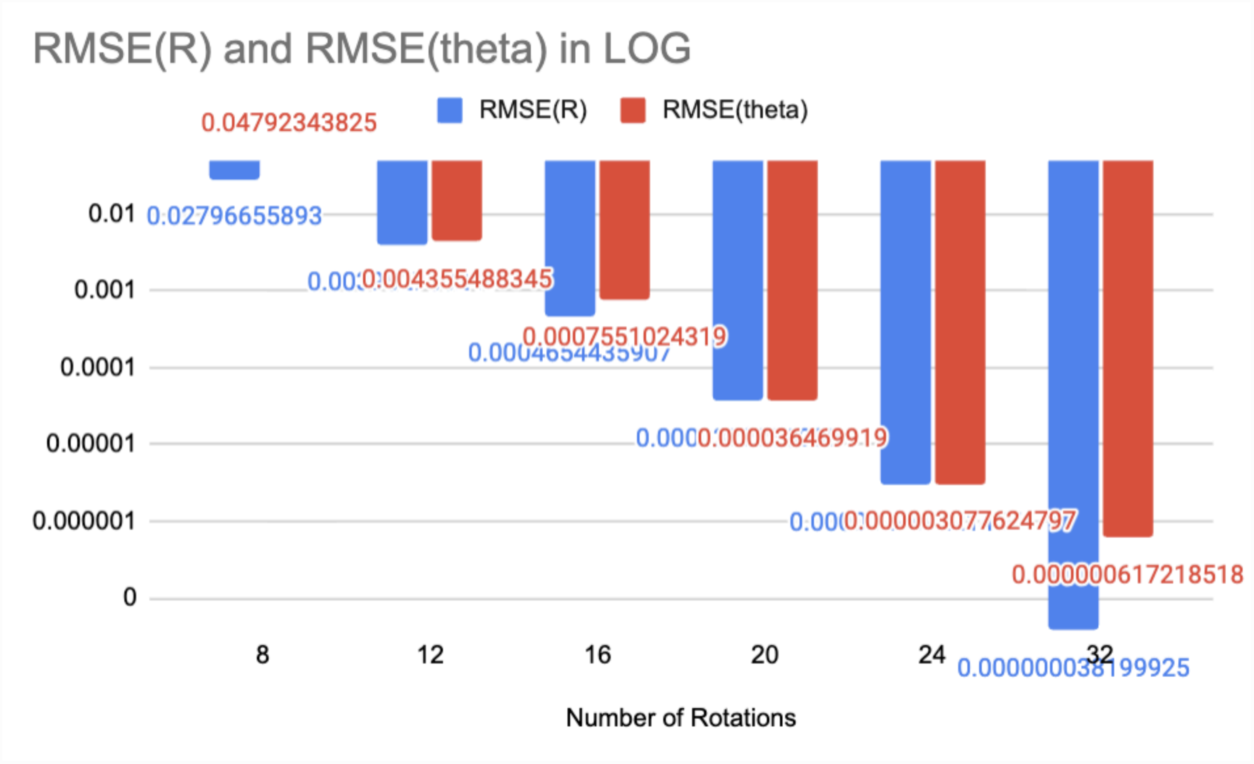
c.

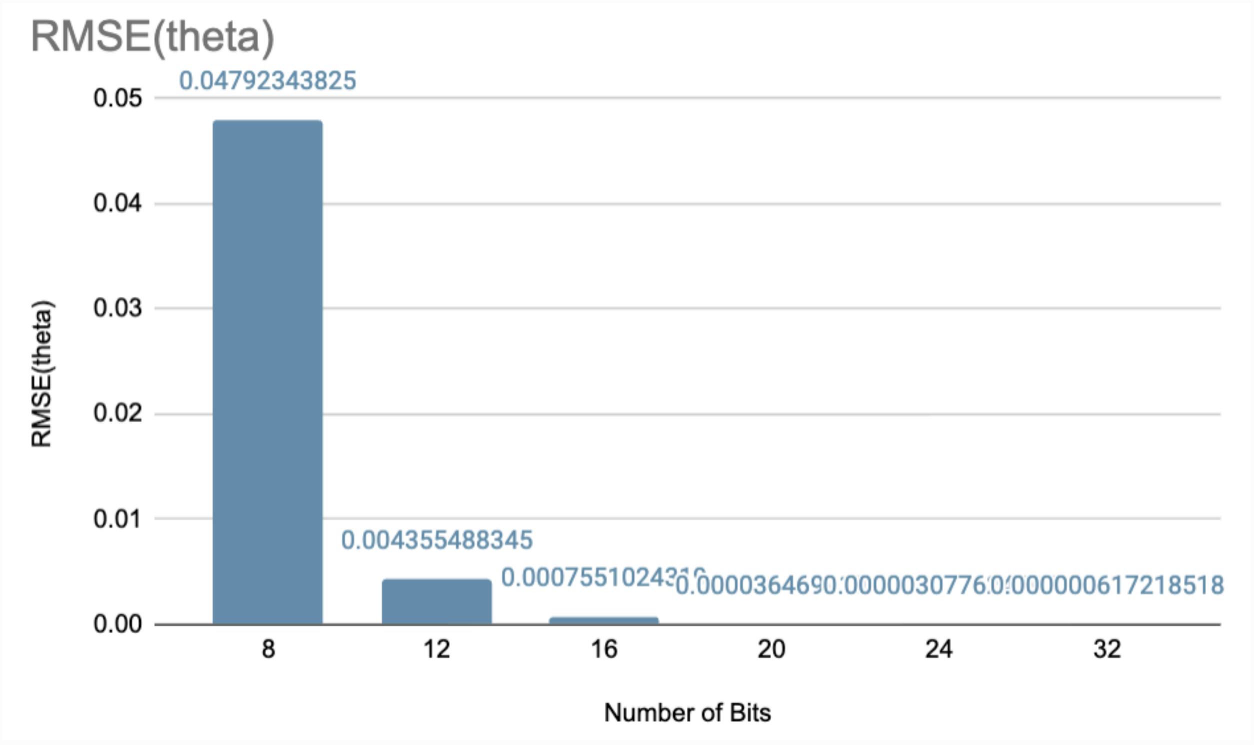
At around 14, the RMSE of radius does not change. The 12th rotation is relatively good for both radius and theta, however, theta keeps decreasing in RMSE even at 20 rotations. The 12th rotation RMSE for the radius is actually lower than the RMSE 14th rotation, most likely some coincidental overfitting with our testbench which just uses the 4 original tests.

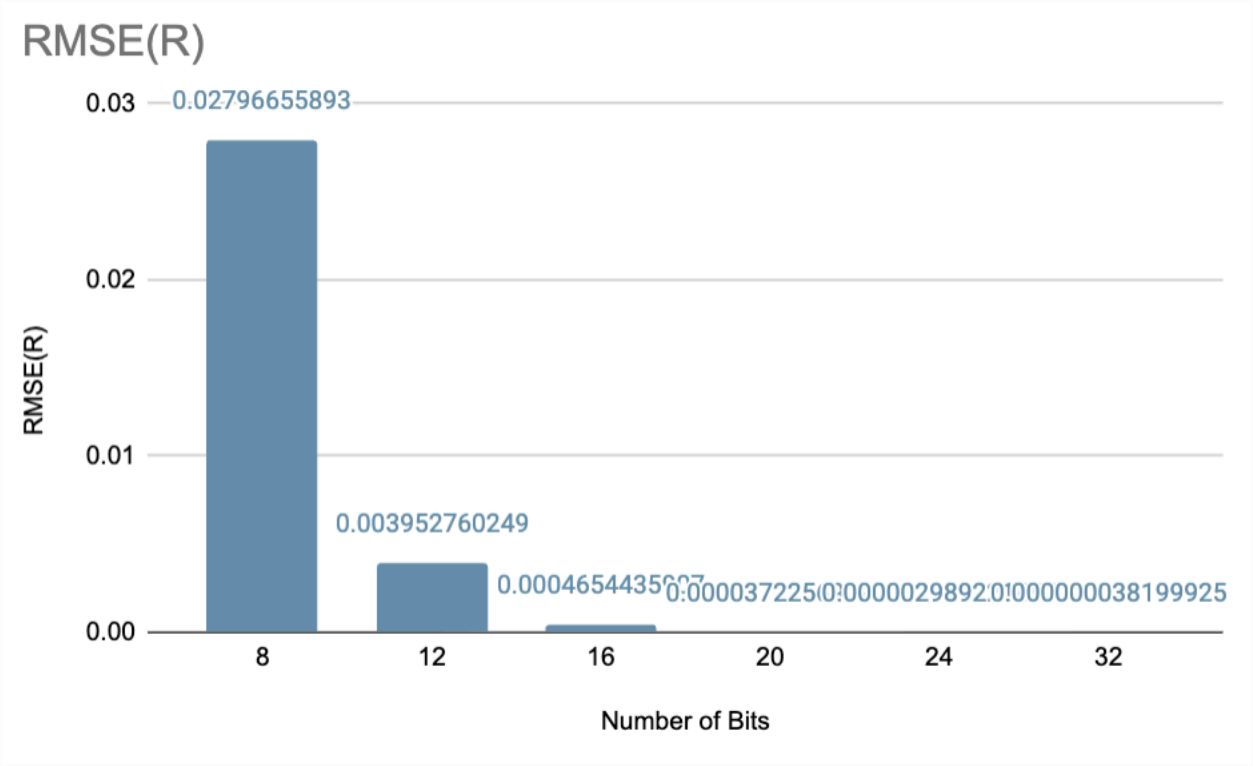
2a.
x:ap_fixed<32,3>
y:ap_fixed<32,3>
r: ap_fixed<32, 2>
theta:ap_fixed<32,3>
Kvalues:ap_fixed<32,3>
angles:ap_fixed<32, 3>

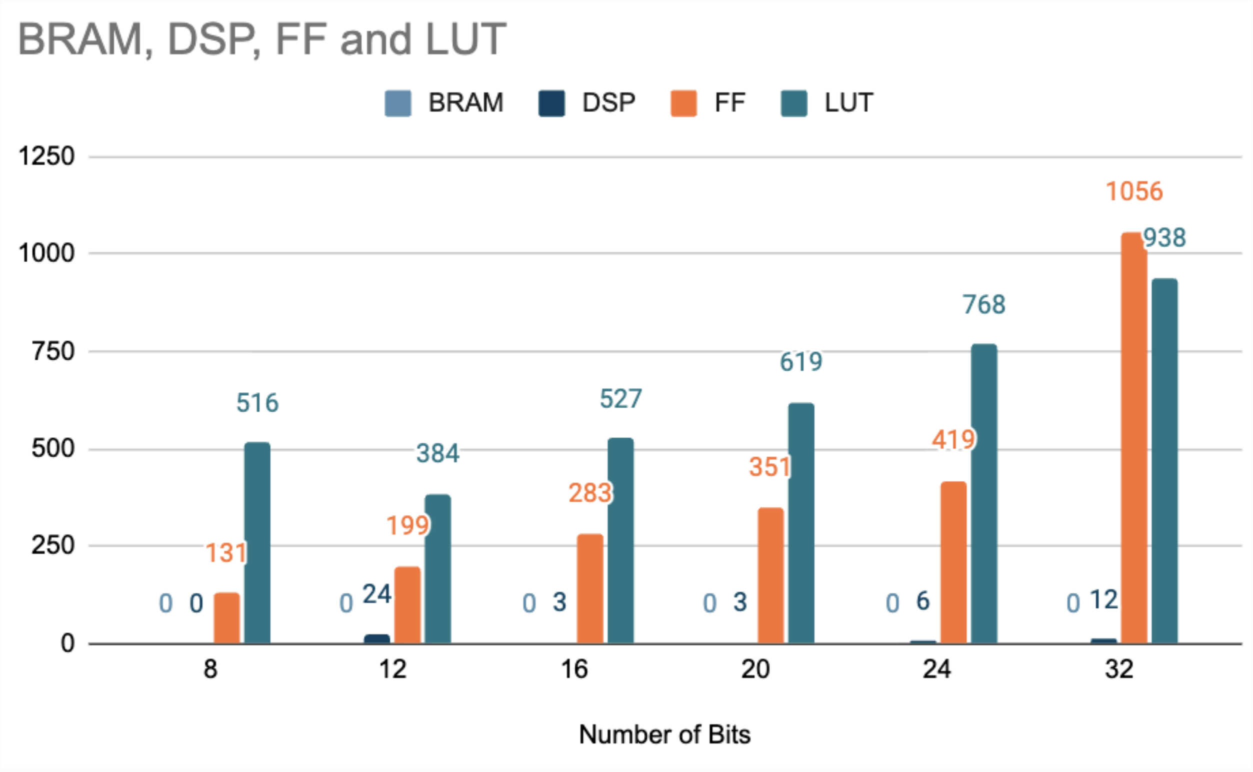
2b.

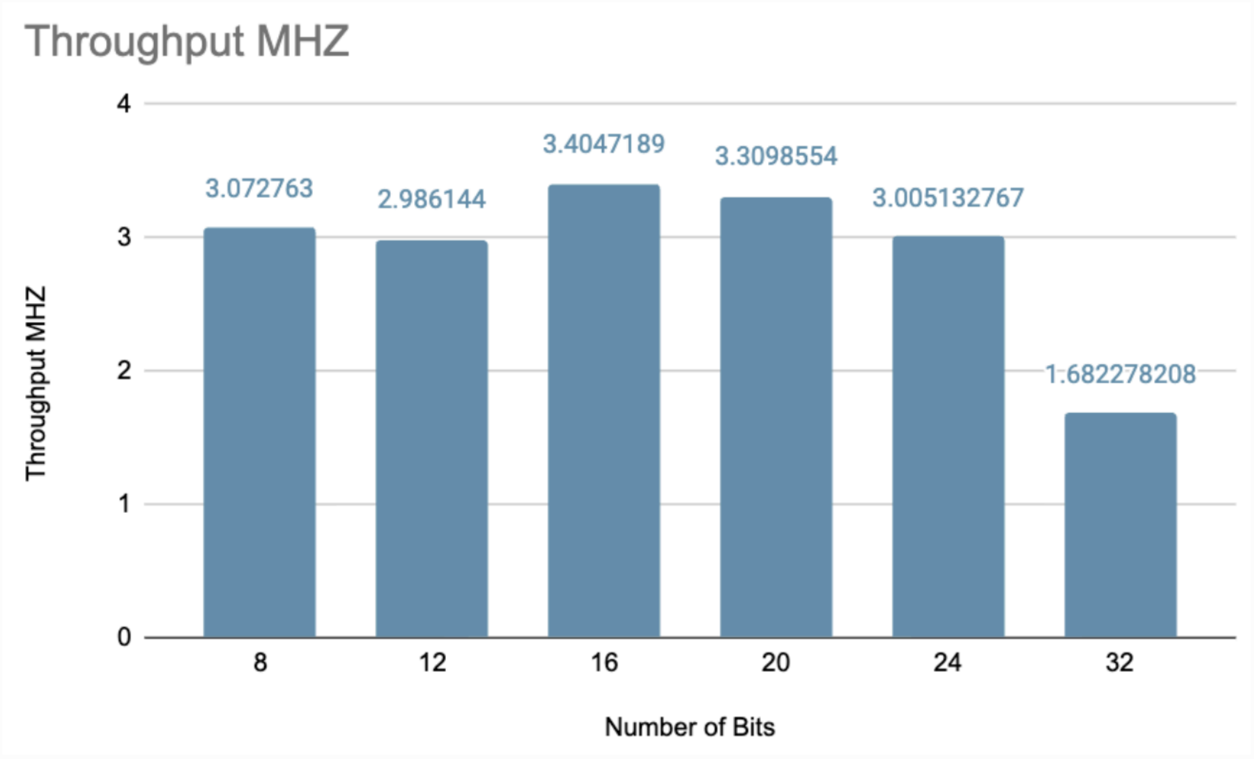
# of bits	8	12	16	20	24	32
BRAM	0	0	0	0	0	0
DSP	0	3	3	3	6	12
FF	131	199	283	351	419	1056
LUT	516	384	527	619	768	938
Throughput MHZ	3.072763	2.986144	3.4047189	3.3098554	3.005132767	1.6822782084
Latency min/max	44/44	45/45	45/45	45/45	45/45	85/85
RMSE(R)	0.027966558933258	0.003952760249376	0.000465443590656	0.000037225690903	0.000002989221457	0.000000038199925
RMSE(theta)	0.047923438251019	0.004355488345027	0.000755102431867	0.000036469918996	0.000003077624797	0.000000617218518

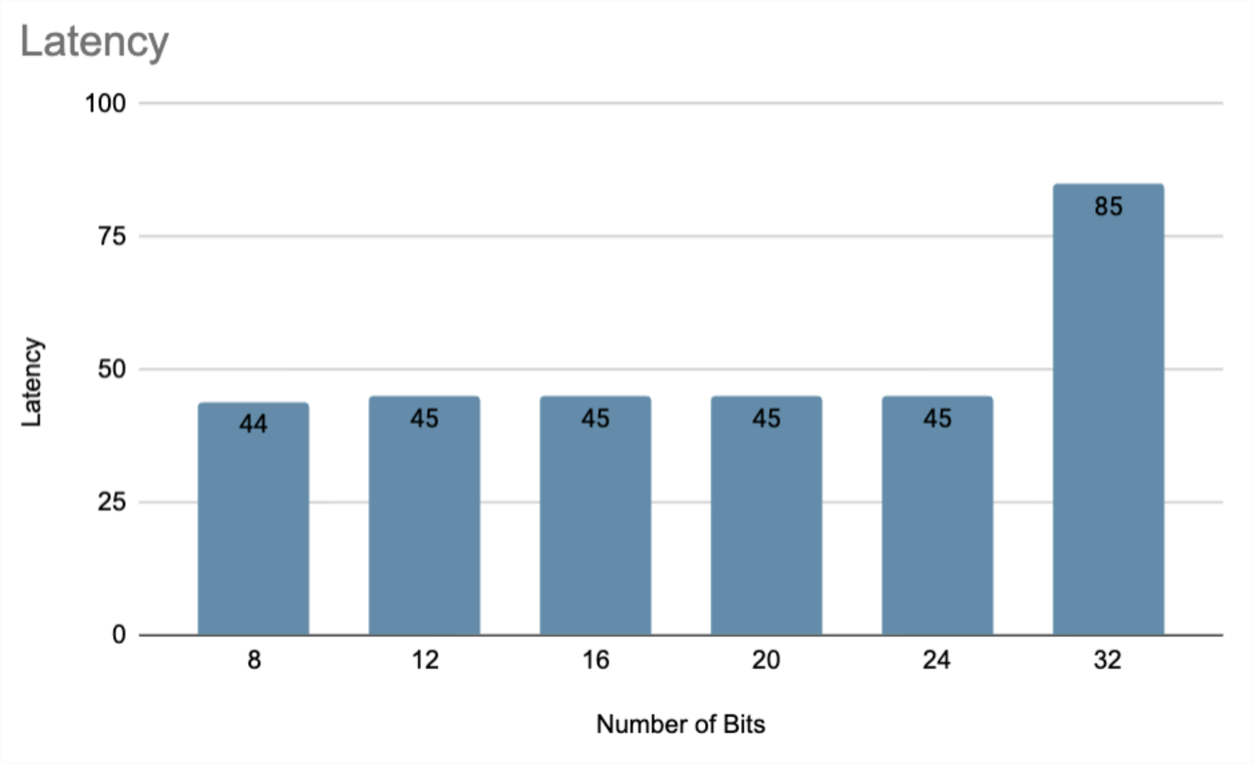

RMSE(R) and RMSE(theta) in LOG


RMSE(theta)


RMSE(R)
RMSE(R)
0.03
0.02
0.01
0.00
0.02796655893
0.003952760249
0.0004654435607
0.000037225
0.0000029892
0.000000038199925
8
12
16
20
24
32
Number of Bits

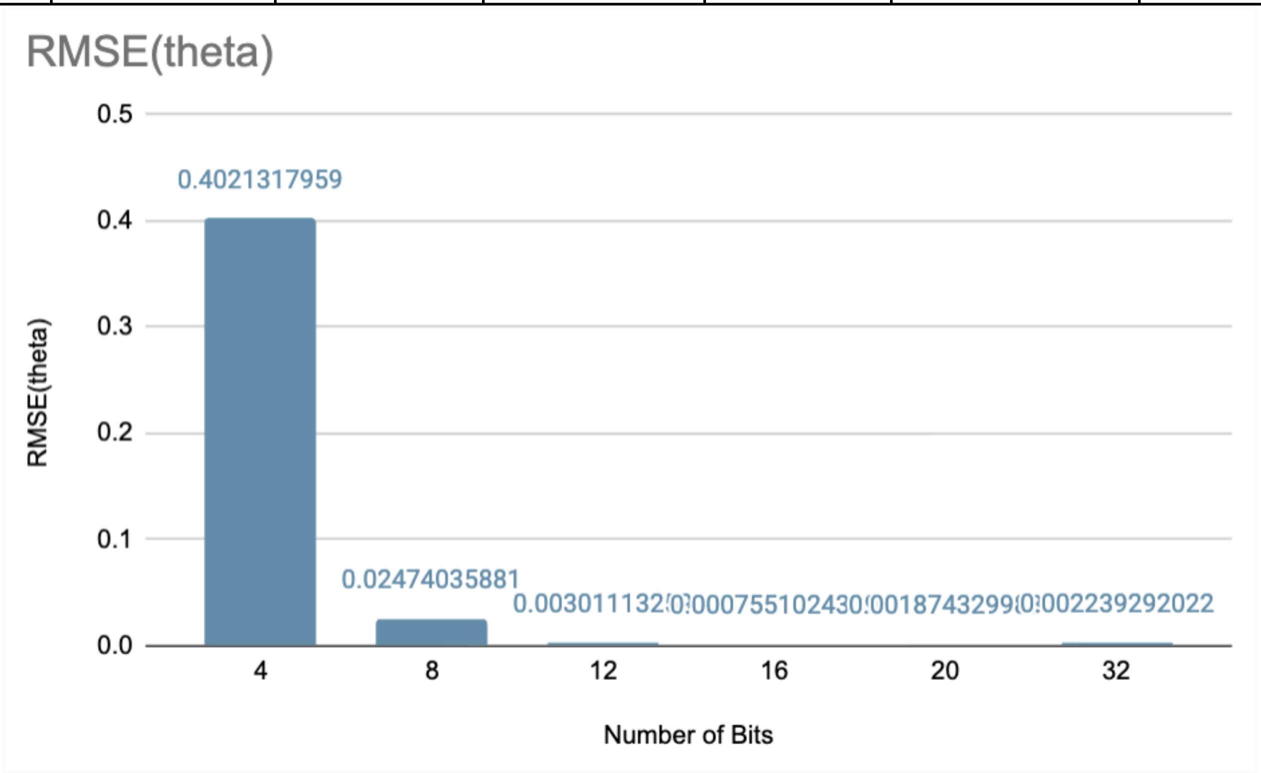

BRAM, DSP, FF and LUT
BRAM
DSP
FF
LUT
1250
1000
750
500
250
0
0
0
131
516
0
24
199
384
0
3
283
527
0
3
351
619
0
6
419
768
0
12
1056
938
8
12
16
20
24
32
Number of Bits

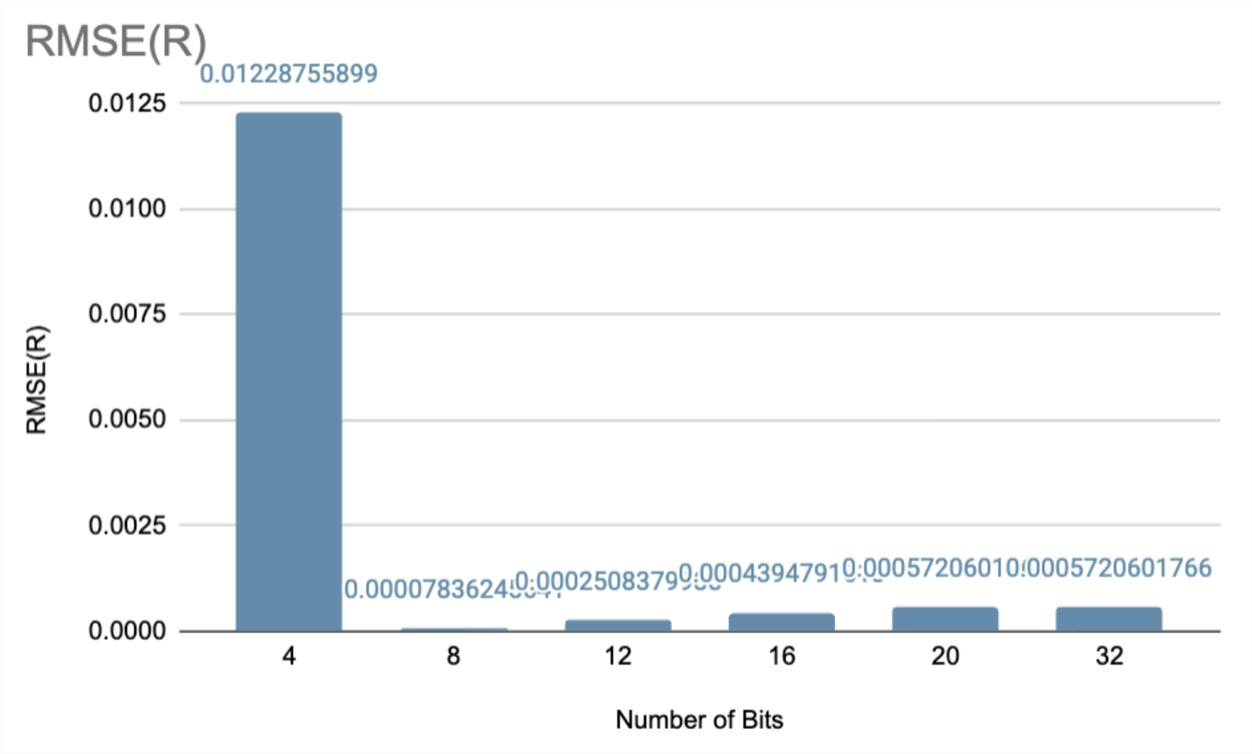

Throughput MHZ
4
3
2
1
0
Throughput MHZ
3.072763
2.986144
3.4047189
3.3098554
3.005132767
1.682278208
8
12
16
20
24
32
Number of Bits

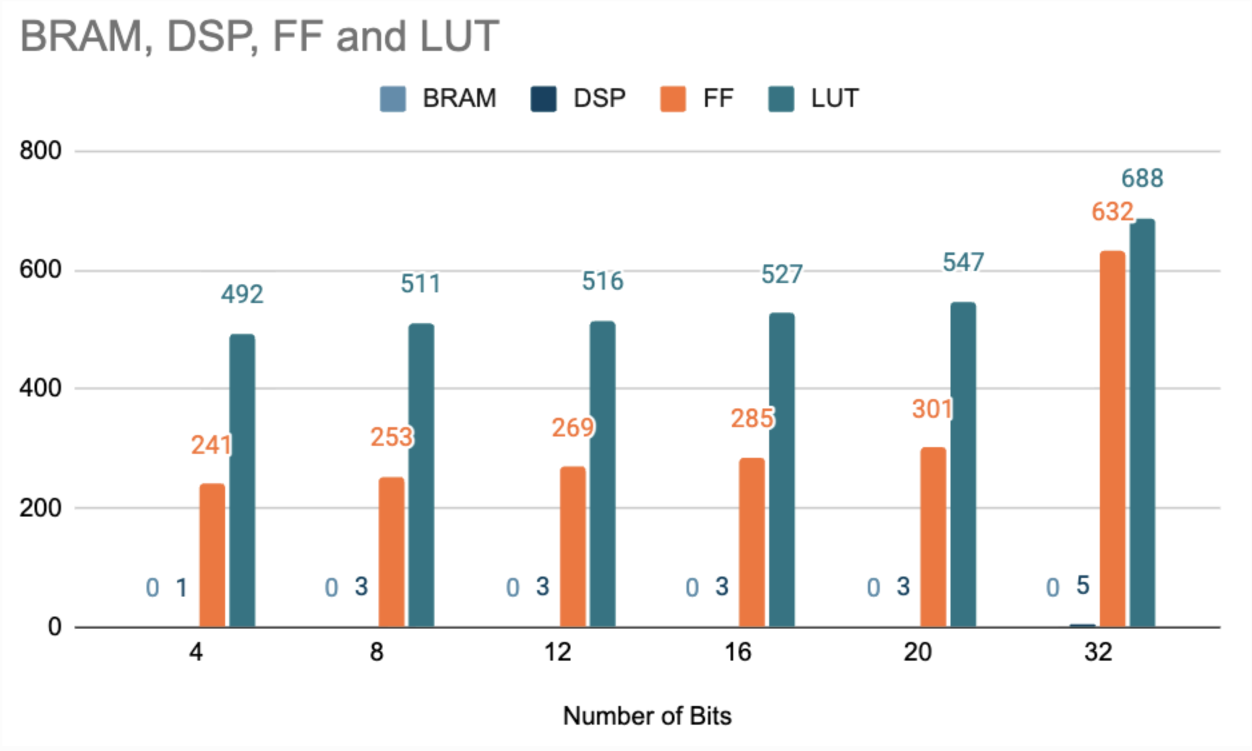

Latency
100
75
50
25
0
Latency
44
45
45
45
45
85
8
12
16
20
24
32
Number of Bits

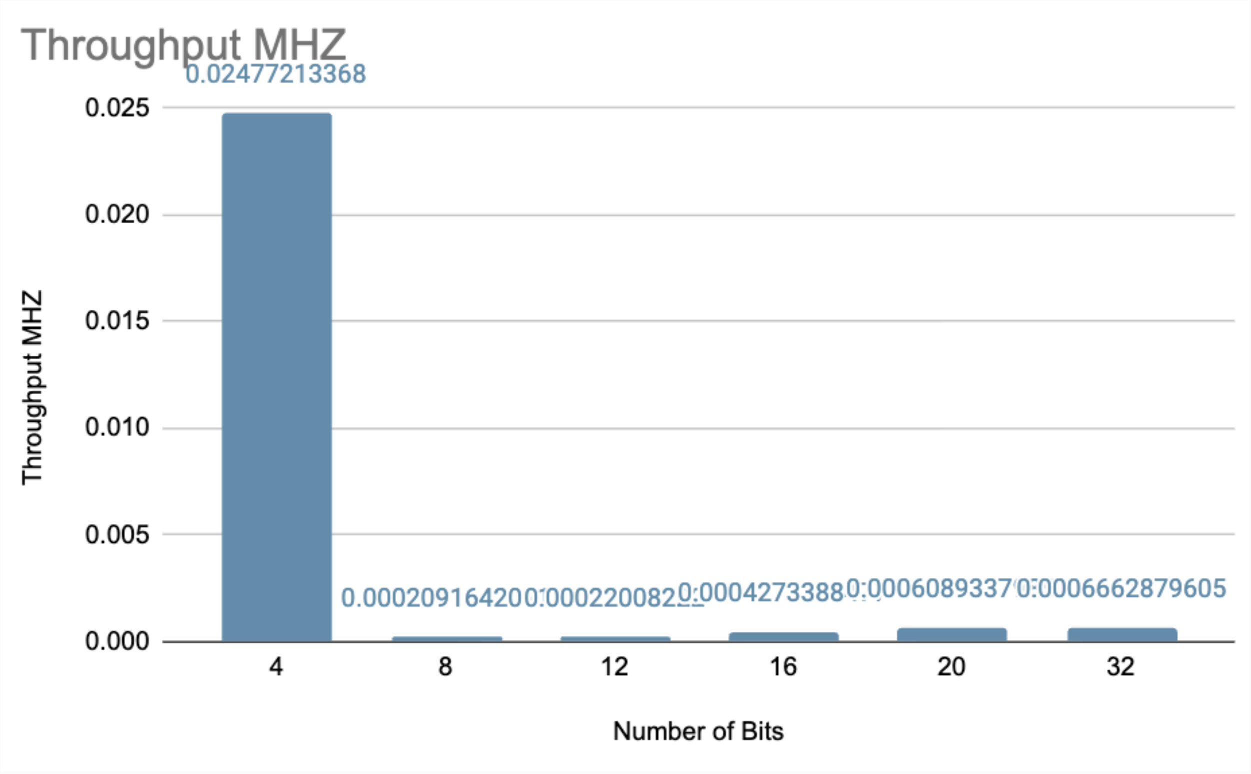
2c.

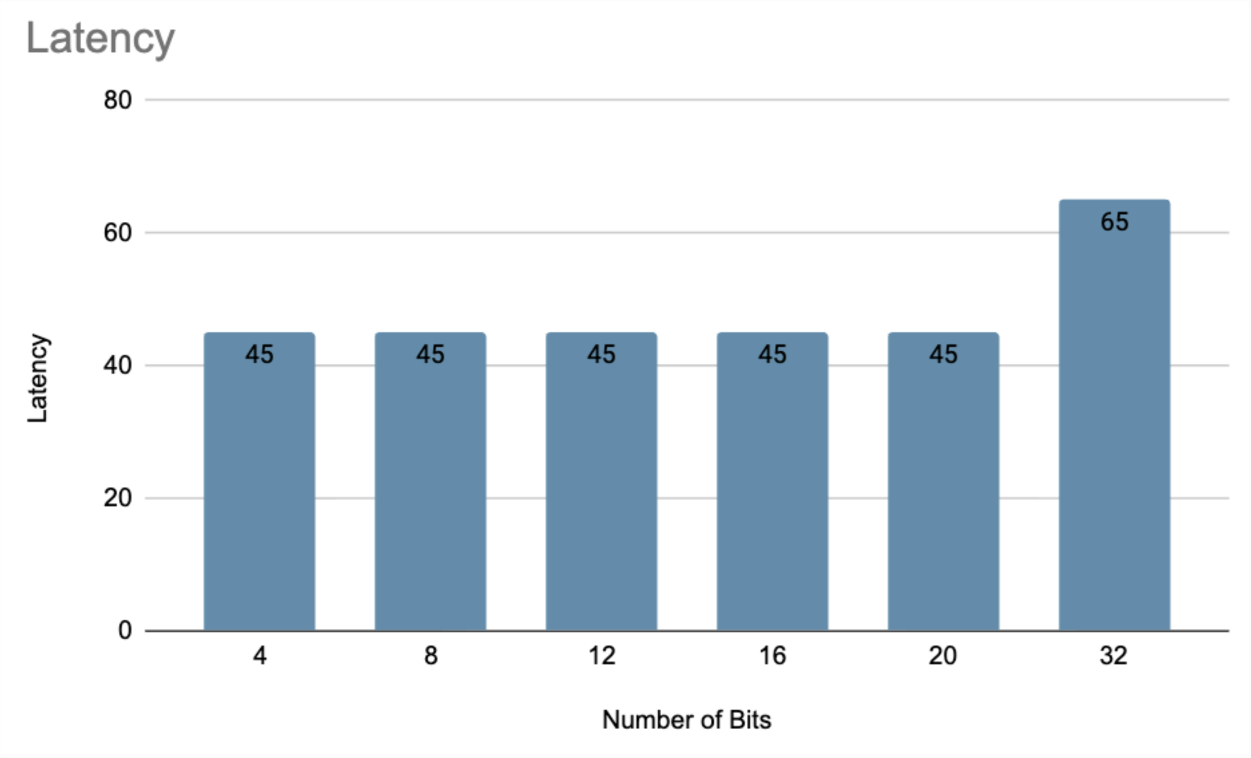
# of bits	4	8	12	16	20	32
BRAM	0	0	0	0	0	0
DSP	1	3	3	3	3	5
FF	241	253	269	285	301	632
LUT	492	511	516	527	547	688
Throughput MHZ	3.11448860097	3.40471894045	3.40471894045	3.40471894045	3.39408749958	2.19205948373
Latency min/max	45/45	45/45	45/45	45/45	45/45	65/65
RMSE(R)	0.012287558987737	0.000078362456406	0.000250837998465	0.000439479161287	0.000572060176637	0.000572060176637
RMSE(theta)	0.402131795883179	0.024740358814597	0.003011132590473	0.000755102431867	0.001874329987913	0.002239292021841




RMSE(R)
0.01228755899
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
RMSE(R)
0.0000783624
0.0002508379
0.0004394791
0.0005720601
0005720601766
4
8
12
16
20
32
Number of Bits


BRAM, DSP, FF and LUT
BRAM
DSP
FF
LUT
800
600
400
200
0
492
511
516
527
547
688
632
241
253
269
285
301
0
1
0
3
0
3
0
3
0
3
0
5
4
8
12
16
20
32
Number of Bits

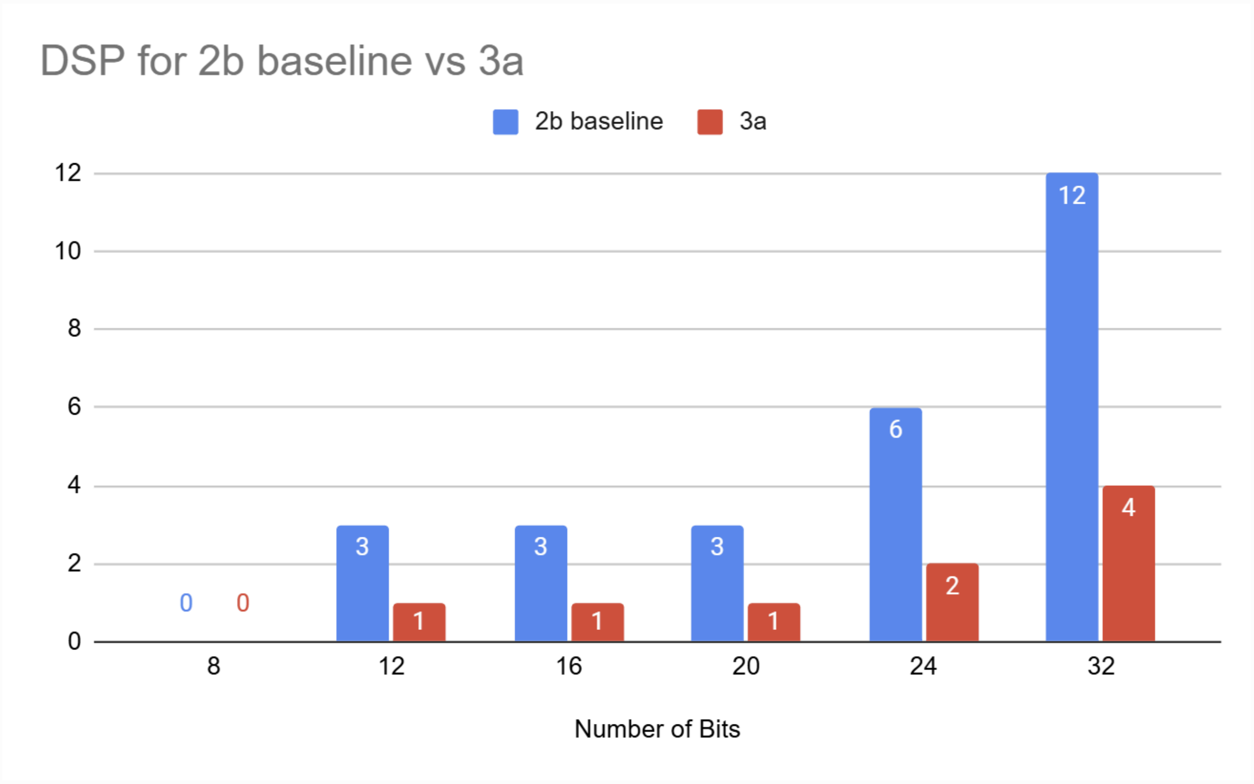

Throughput MHZ
0.02477213368
0.00020916420
0.000220082
0.0004273388
0.0006089337
0.0006662879605
0.025
0.020
0.015
0.010
0.005
0.000
Throughput MHZ
4
8
12
16
20
32
Number of Bits


Latency
80
60
40
20
0
Latency
45
45
45
45
45
65
4
8
12
16
20
32
Number of Bits

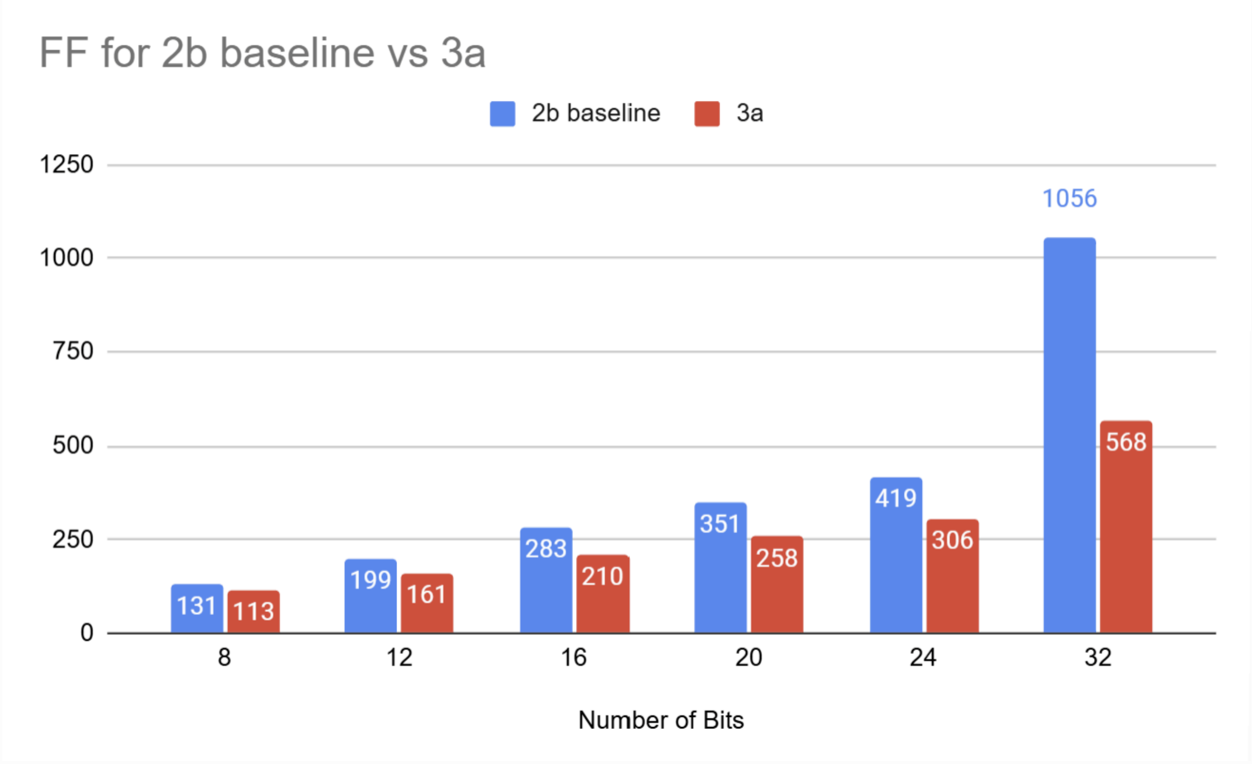
3a.

# of bits	8	12	16	20	24	32
BRAM	0	0	0	0	0	0
DSP	0	1	1	1	2	4
FF	113	161	210	258	306	568
LUT	439	403	568	671	794	1008
Throughput MHZ	3.14183829	3.61102083559	3.3461826748	3.33537903247	3.28523746	3.10366232
Latency min/max	44/44	44/44	43/43	43/43	43/43	44/44
RMSE(R)	0.019335353747010	0.002185192424804	0.000080280347902	0.000009209812561	0.000000697729604	0.000000033918731
RMSE(theta)	0.024768553674221	0.003023126861081	0.000559360603802	0.000010609828678	0.000001133646720	0.000000617218518

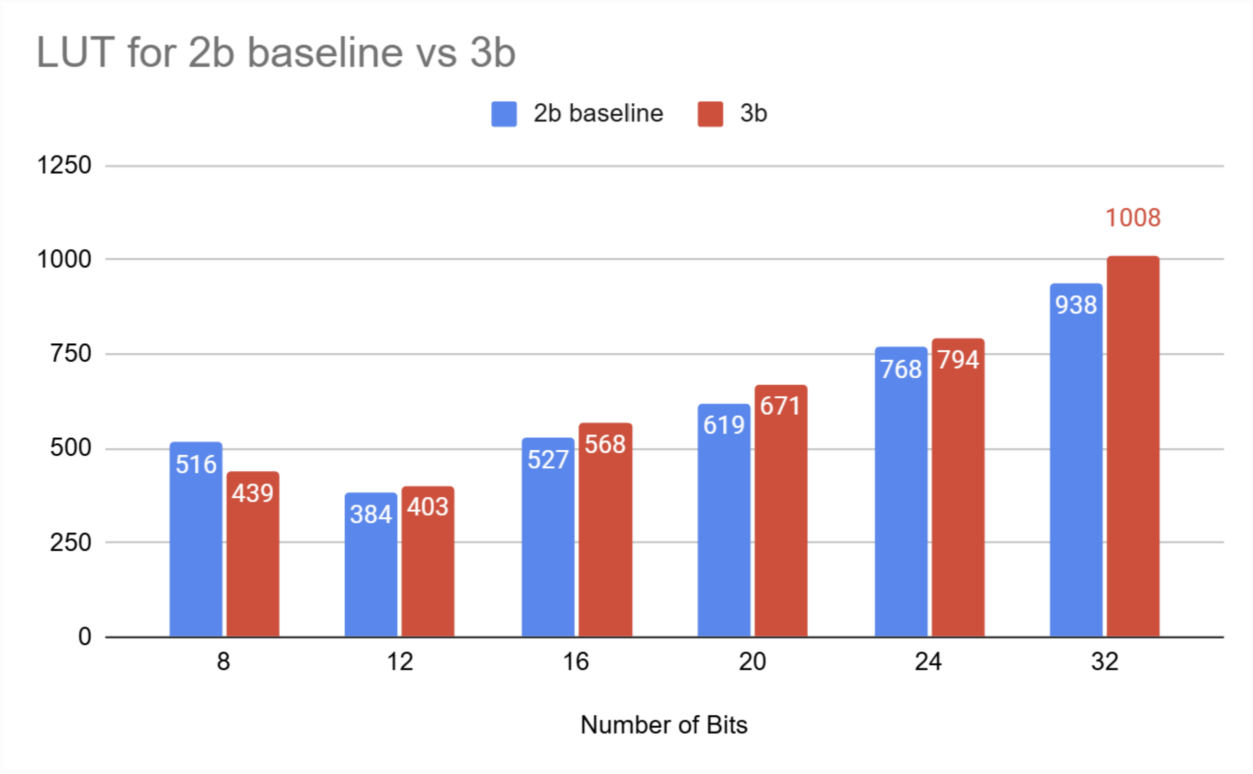
3b.


DSP for 2b baseline vs 3a

We can see that the DSP for 3a constantly stays below that of the baseline 2b. 2b starts at 0, remains at 3 until 20 bits, and then doubles for the final two intervals. In comparison, 3a consistently uses a third of the DSPs of 2b.



For 3a, we see that the total FFs remains consistently below that of the baseline. For the baseline it starts at 131 for 8 bits, and keeps increasing as the number of bits increases and has a huge jump from 24 bits to 32 bits. In comparison, the implementation from 3a scales much better, not witnessing as big of a jump for the higher number of bits leading to greater savings.



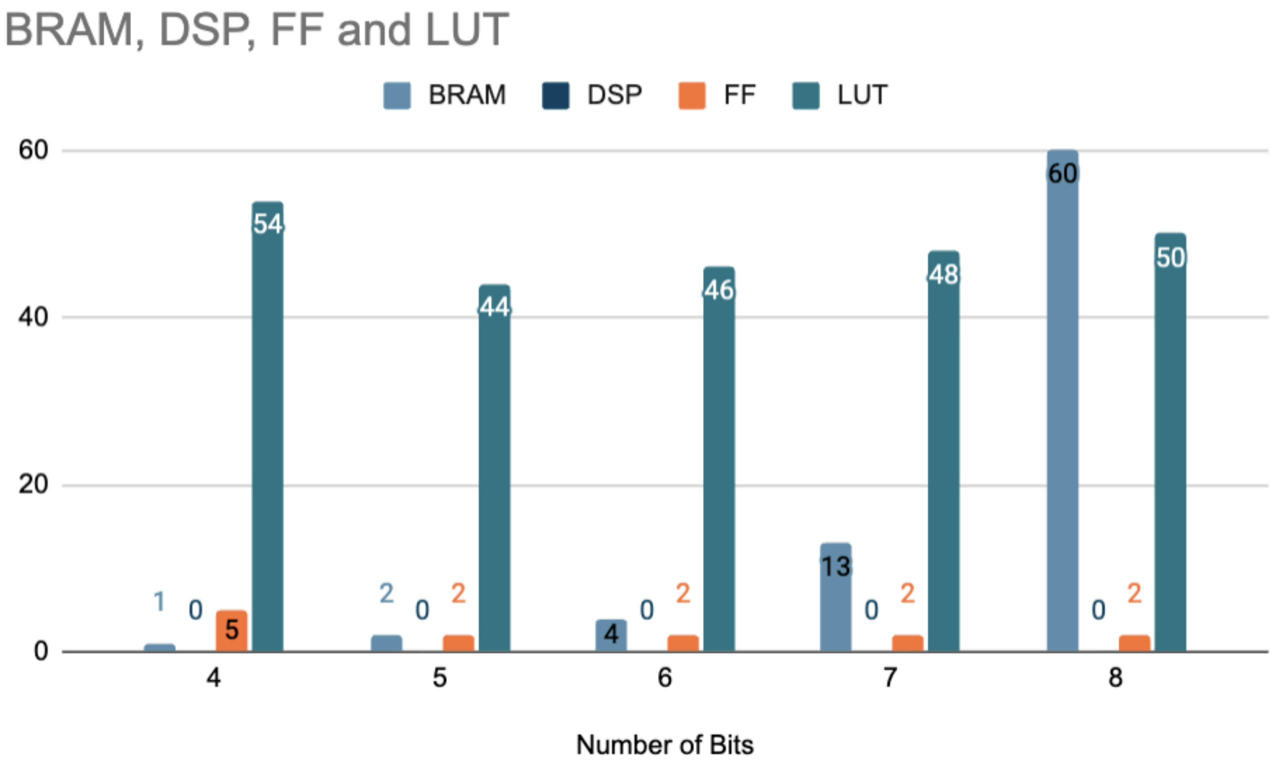
We can see that for 3a, the amount of LUTs generally tends to be marginally higher than the LUTs for 2b. For 8 bits, it is slightly lower but it begins to be slightly over for the rest. It is possible that because we are using shifts, we end up using more from the LUTs to handle the processing rather than the DSPs since we no longer are in need of multiplications. We save on the DSP resources and slightly increase our utilization of LUTs.

4a.
The input data type determines how big the LUT needs to be to contain its data, as larger input data sizes mean larger LUT, and smaller input means smaller LUT. If the total number of bits in for each value in the entry is W, then the size of the LUT must be 2^(2*W). The output data type does not affect the size of LUT in any way, since the rotations only actively take place over the input. However, the output affects the final level of precision for the result.

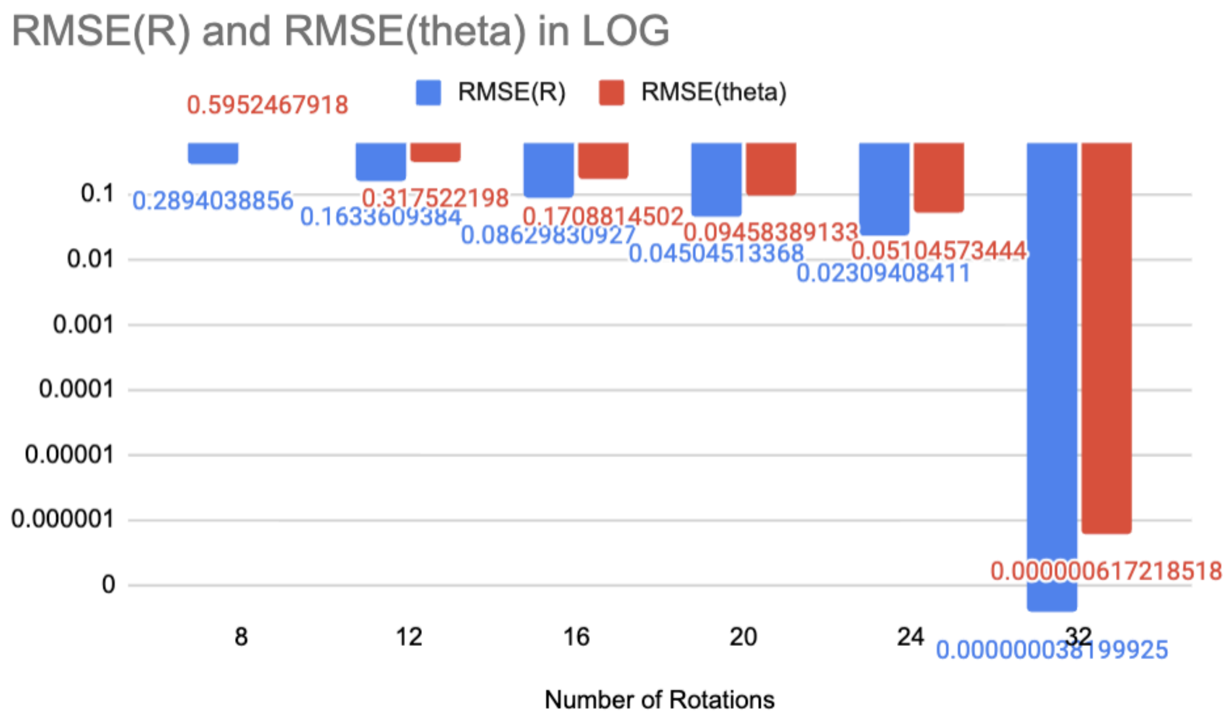
4b.

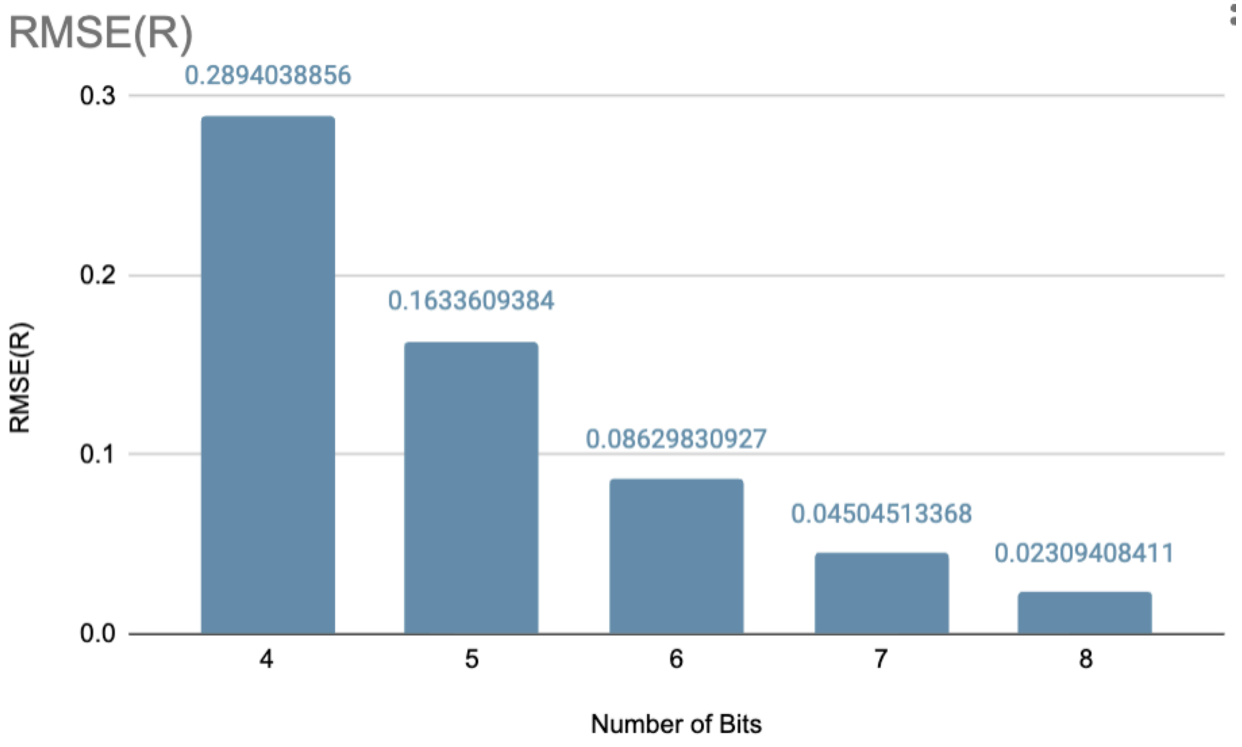
# of bits	4	5	6	7	8
BRAM	1	2	4	13	60
DSP	0	0	0	0	0
FF	5	2	2	2	2
LUT	54	44	46	48	50
Throughput MHZ	95.129375951	91.79364788	92.250922509	95.693779904	91.240875912
Latency min/max	1/1	1/1	1/1	1/1	1/1
RMSE(R)	0.289403885602951	0.163360938429832	0.086298309266567	0.045045133680105	0.023094084113836
RMSE(theta)	0.595246791839600	0.317522197961807	0.170881450176239	0.094583891332150	0.051045734435320

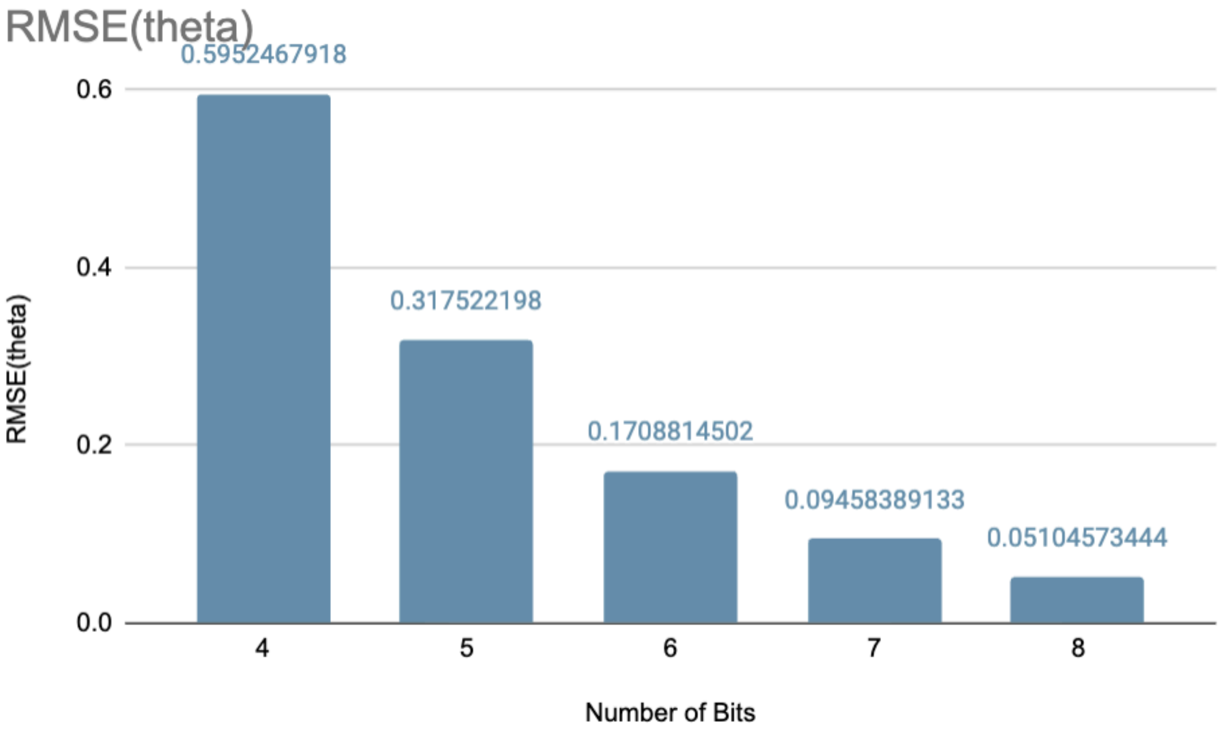
4c.



4d.


RMSE(R) and RMSE(theta) in LOG
RMSE(R)
RMSE(theta)
0.5952467918
0.2894038856
0.317522198
0.1633609384
0.1708814502
0.08629830927
0.09458389133
0.04504513368
0.05104573444
0.02309408411
0.000000617218518
0.000000038199925
0.1
0.01
0.001
0.0001
0.00001
0.000001
0
8
12
16
20
24
32
Number of Rotations


RMSE(R)
0.2894038856
0.1633609384
0.08629830927
0.04504513368
0.02309408411
0.3
0.2
0.1
0.0
RMSE(R)
4
5
6
7
8
Number of Bits



4e.

The LUT-based implementation struggles more with scale, as when you increase the number of bits on the input, you need a far bigger table in order to store the polar coordinates for every x and y pair. The LUT size directly corresponds with the input size W: 2^(2*W). It grows exponentially, meaning it is much worse from a resource utilization perspective in how it needs to store all these values in BRAM. However, for smaller numbers of bits, it can run far quicker than traditional CORDIC (much higher throughput) as it is just directly indexing into a table and grabbing the correct values. Initialization cost is there, but during runtime there can be bigger savings.

In conclusion, the LUT-based implementation has the advantage of higher throughput by utilizing simple value lookups rather than computation but this comes with a serious disadvantage of exponential scaling with resource utilization. CORDIC has much better scaling and resource utilization by taking advantage of rotations to compute the final values instead of having to store values in tables, but at the cost of much, much lower throughput.