Steven Daniels

WES237C

1. **Question 1 – FIR 128 Design with arbitrary bit width optimization**

- Max Throughput: **298 kHz**

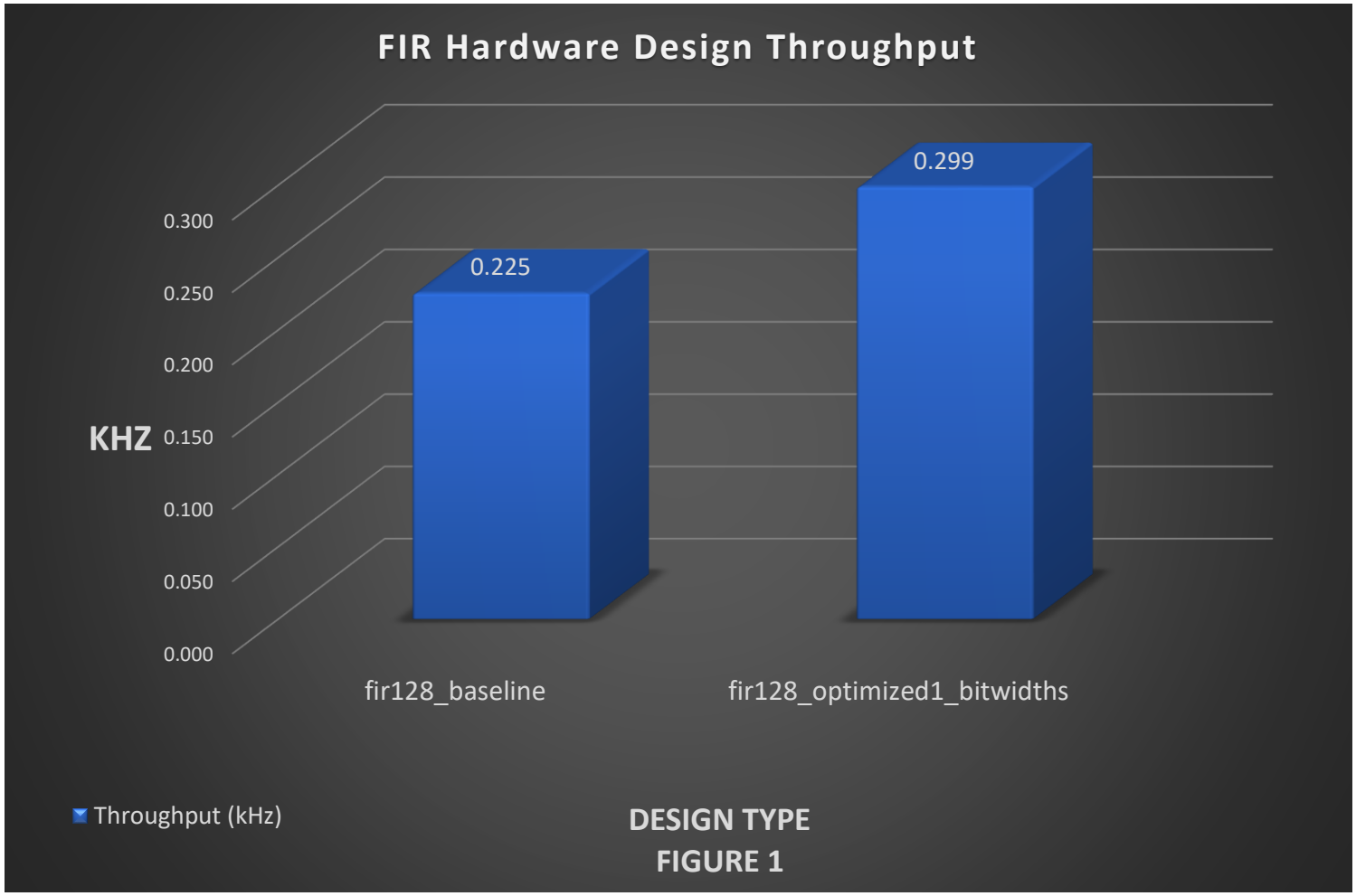


FIGURE 1

- Optimizations:
  - Changed the bit width of the coefficients to 5 bits.

    - Changed the bit width of the accumulator to 16 bits.
- Minimum usable data size without loss of accuracy:
    - Coefficients: 5 bits
    - Accumulator: 16 bits
- Analysis:
    - The goal of this design was to reduce the resource utilization by as much as possible. The lowest usable bit width for this 128 sample FIR filter is 5 bits for the coefficients and 16 bits for each accumulator. Decreasing the bit width provided a moderate increase in throughput for this design as can be seen in **Figure 1**, but provided a major reduction in resource utilization when compared to the baseline implementation. A plot demonstrating the effects that shrinking the bit width of variables within our top-level function had on each FPGA resource can been seen in **Figure 2**. To provide a clearer picture of the degree to which resource usage declined in our design we observe how much the resource usage for this design changed from our baseline metrics. According to the estimates produced by this design flip flops benefited the most from reducing the bit widths. Reducing the bit widths in the baseline design provides approximately a 75% reduction in Flip Flop usage as seen in **Figure 3**. This would produce a design that uses less power than the baseline as well as a moderately higher maximum throughput.

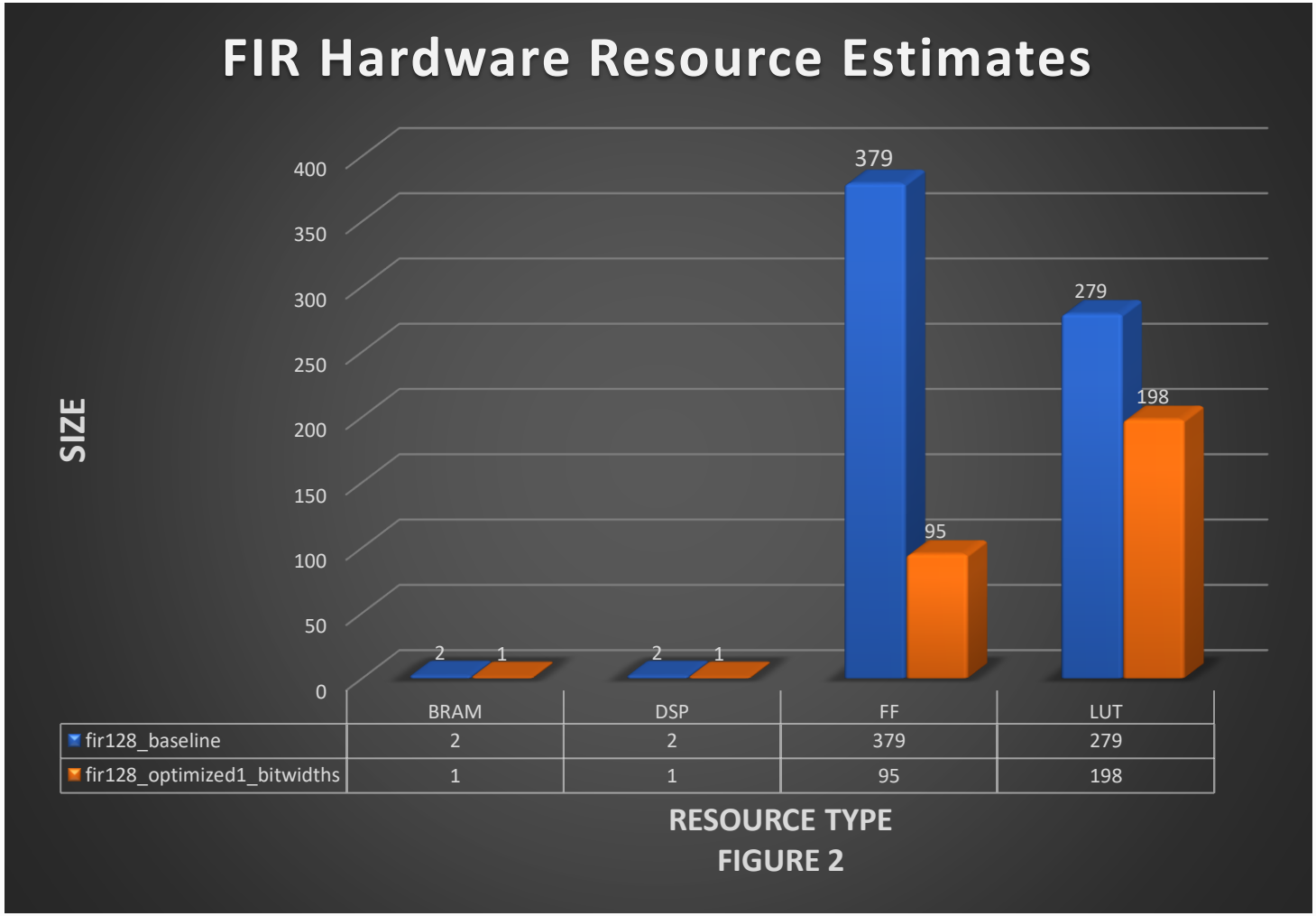

FIR Hardware Resource Estimates
SIZE
400
350
300
250
200
150
100
50
0
379
279
198
95
2
1
2
1
BRAM
DSP
FF
LUT
fir128_baseline
2
2
379
279
fir128_optimized1_bitwidths
1
1
95
198
RESOURCE TYPE

FIGURE 2

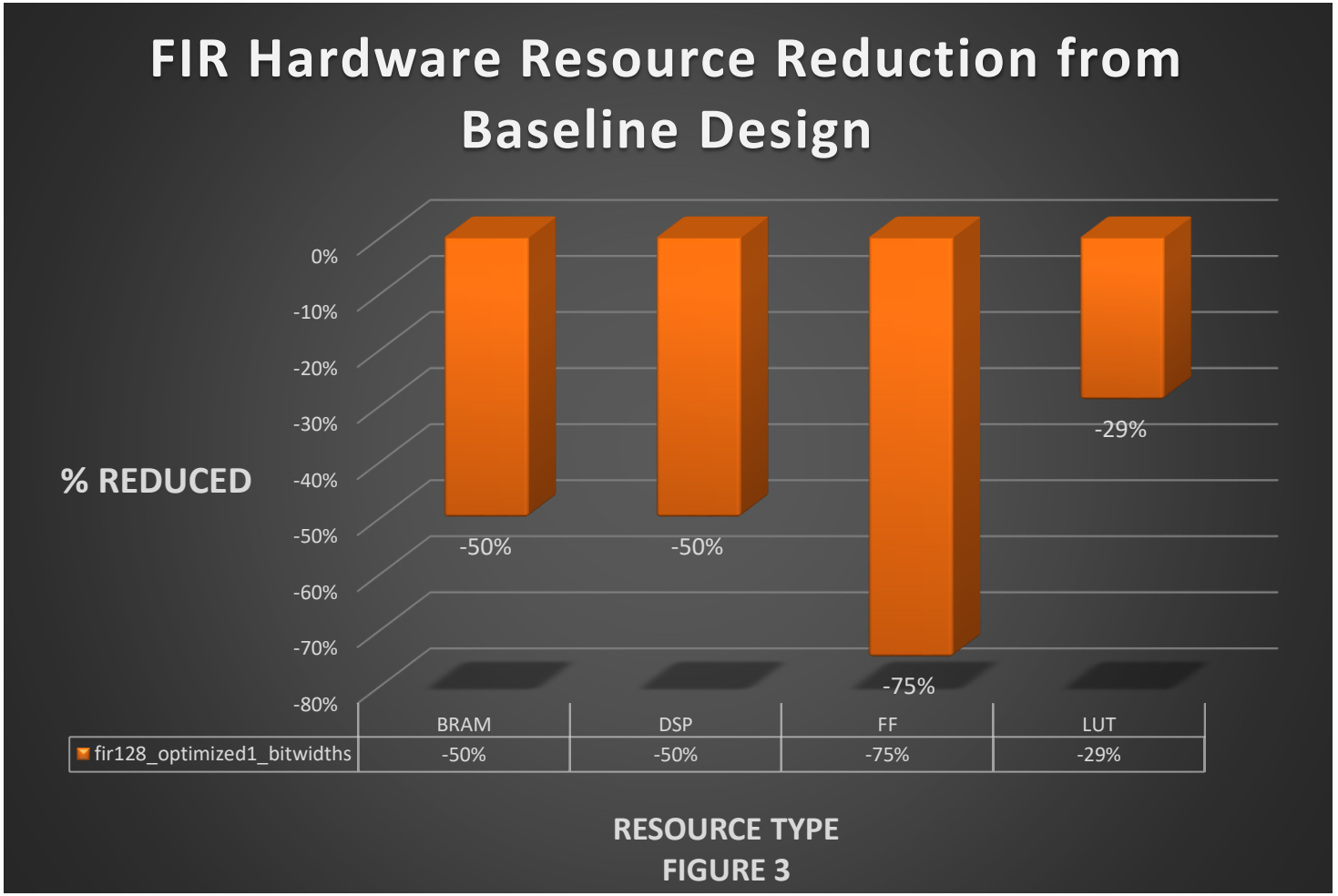


FIGURE 3

2. **Question 2 – FIR 128 Design with pipeline optimization**

   - Max Throughput: **1.064 MHz**

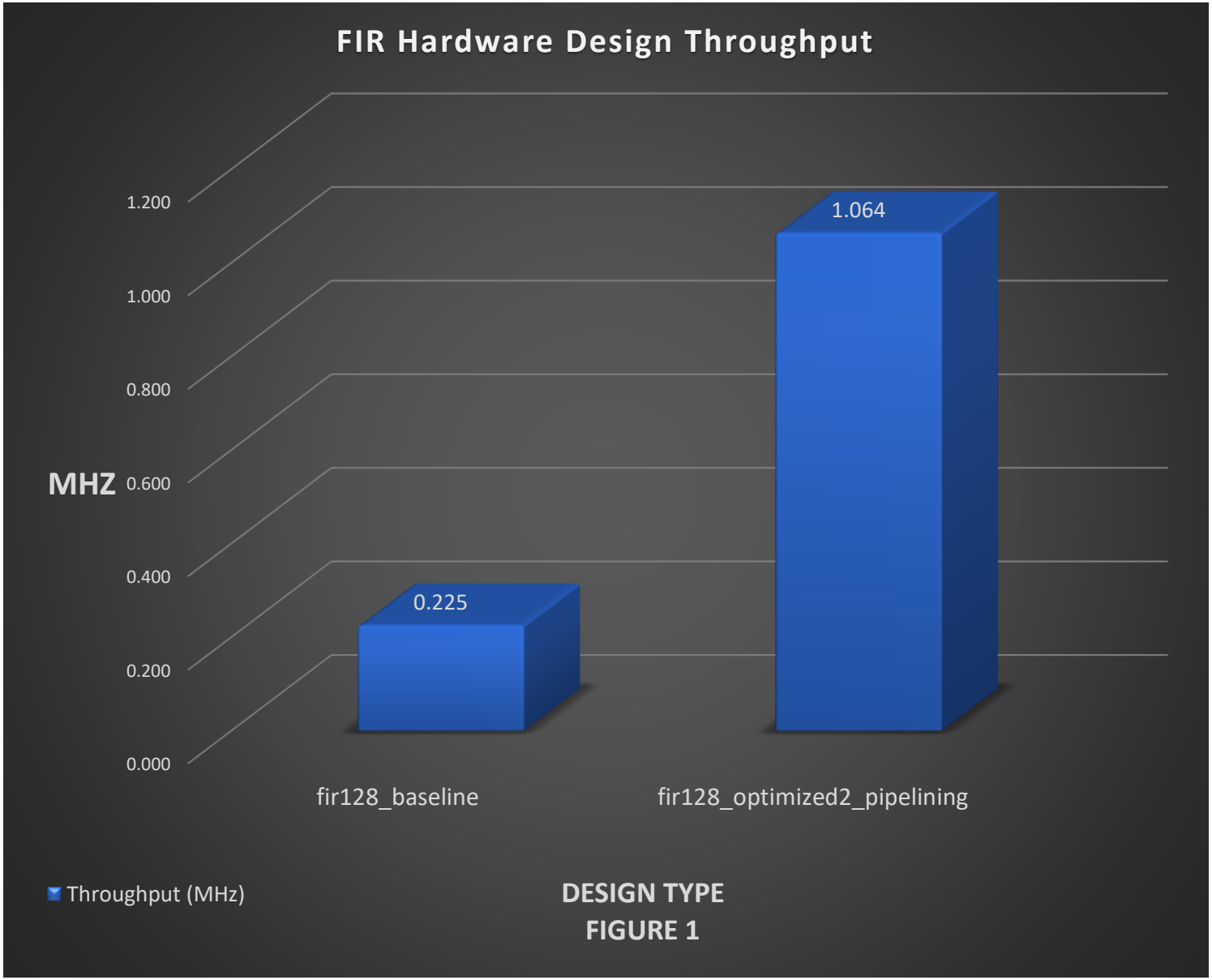


FIGURE 1

- Optimizations:
    - Pipelined the shift accumulator loop with an initiation interval of 1.
- Observations:

- Initiation intervals effect on loop latency
    - Adding a pipelined shift accumulator loop with an initiation interval of 1 drastically decreases the loop latency of the design by over 50% of the baseline value. However, as the initiation interval is increased in increments of 1 the improvements that were originally gained with and II=1 is lost. By the time an II=5 is reached the loop latency returns to approximately the baseline value as can be seen in **Figure 2**. The maximum initiation interval of a general loop can be found by multiplying the iteration latency value by the number of times the loop must iterate and then adding 1.

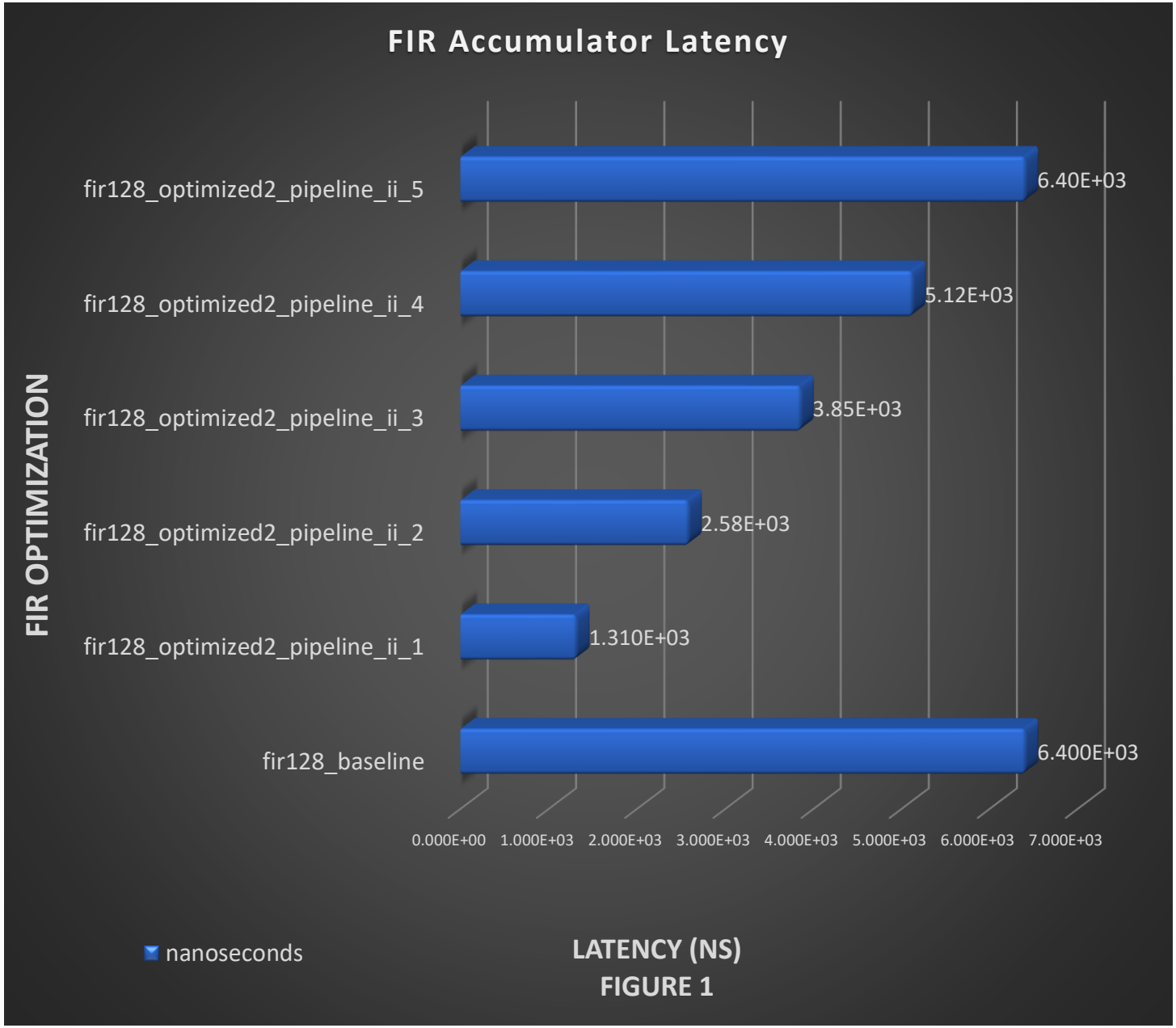
- 

FIGURE 1

- Analysis

- The goal of this design was to improve the throughput of the top module. Using an initiation interval of 1 for the shift accumulator loop provides a design that attempts to process a new input every clock cycle. Doing this leads to a higher overall throughput as seen in **Figure 1**. The downside to this design is that it requires a greater number of resources as can be seen in **Figure 2**. This leads to using approximately 61% more flip flops than the baseline design (**Figure 3**) which may not be ideal for systems that are resource restricted.

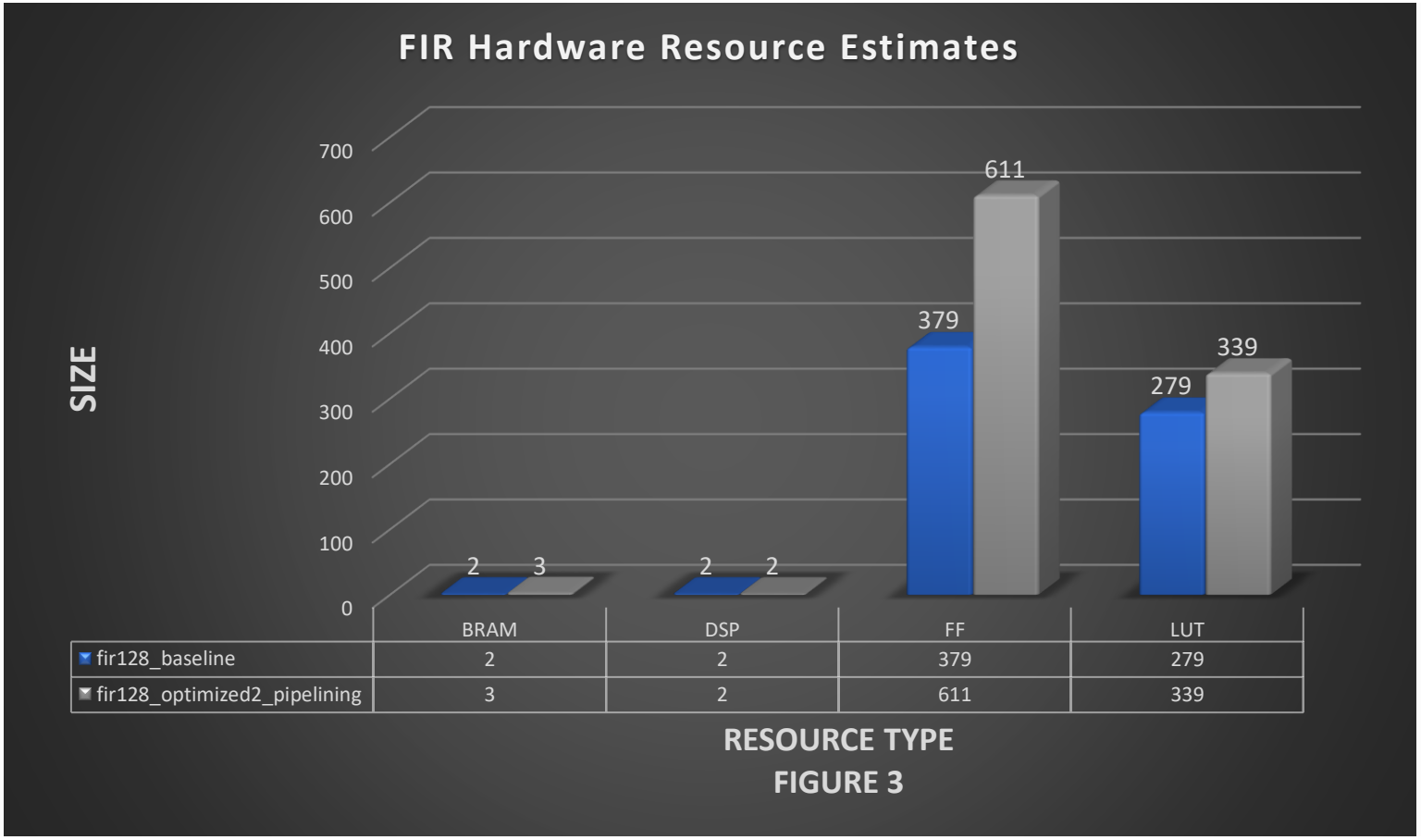


FIGURE 3

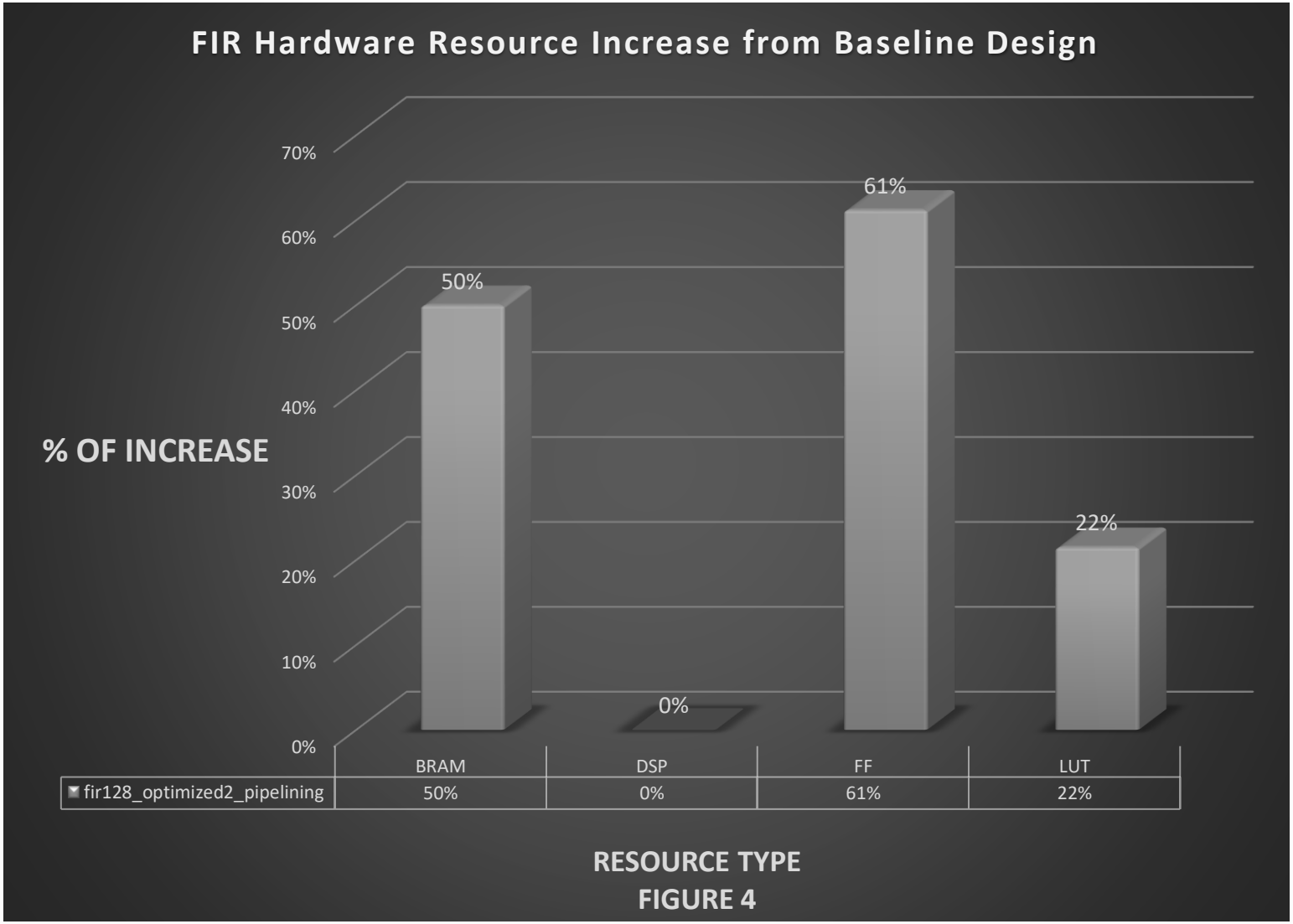


FIGURE 4

3. **Question 3 – FIR 128 Design with branching optimization**

    - Max Throughput: **227 kHz**

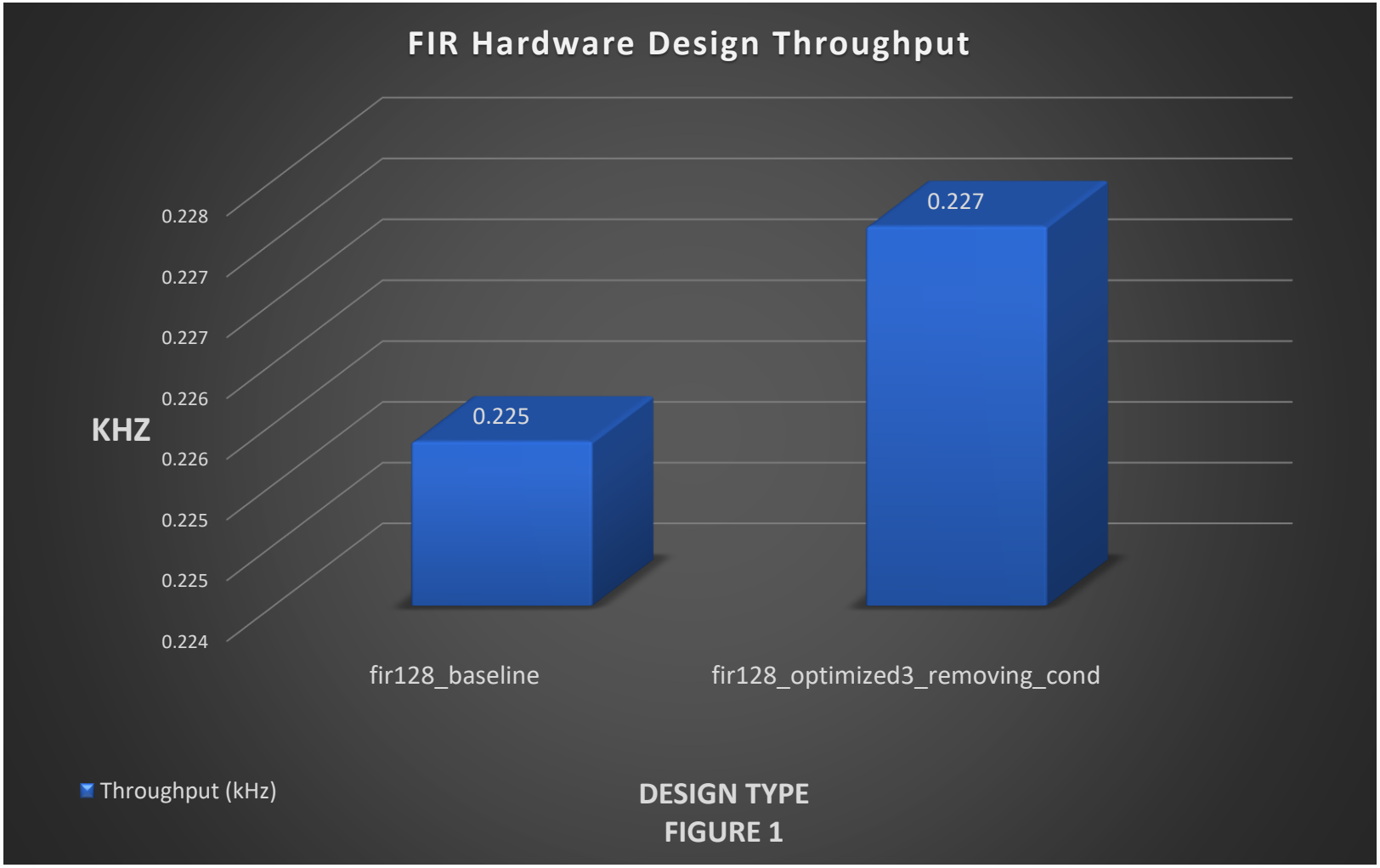


FIGURE 1

- Optimizations:
  - Hoisted code by removing if else conditional from shift accumulator loop.
- Analysis:
  - The goal of this design was to improve the efficiency of the code by removing conditional branching statements. . Branching adds an additional cycle to perform a comparison. Therefore, branching increases the number of clock cycles required to complete a set of instructions which increases the number of minimum and/or maximum cycles

in terms of latency. This increase in latency means that designs with tasks that use branch instructions take longer to complete than designs that exclude them. Excluding branch statements reduces the amount of required speculation which provides a moderate improvement of overall throughput as seen in **Figure 1**. The number of resources is also reduced due to no longer having to perform reads and writes associated with branching. **Figure 2** and **Figure 3** demonstrate the degree to which this design reduces the number of required resources when compared to the baseline design. The greatest improvement in resource usage was seen in flip flop usage with this design requiring 17% less flip flops than the baseline.

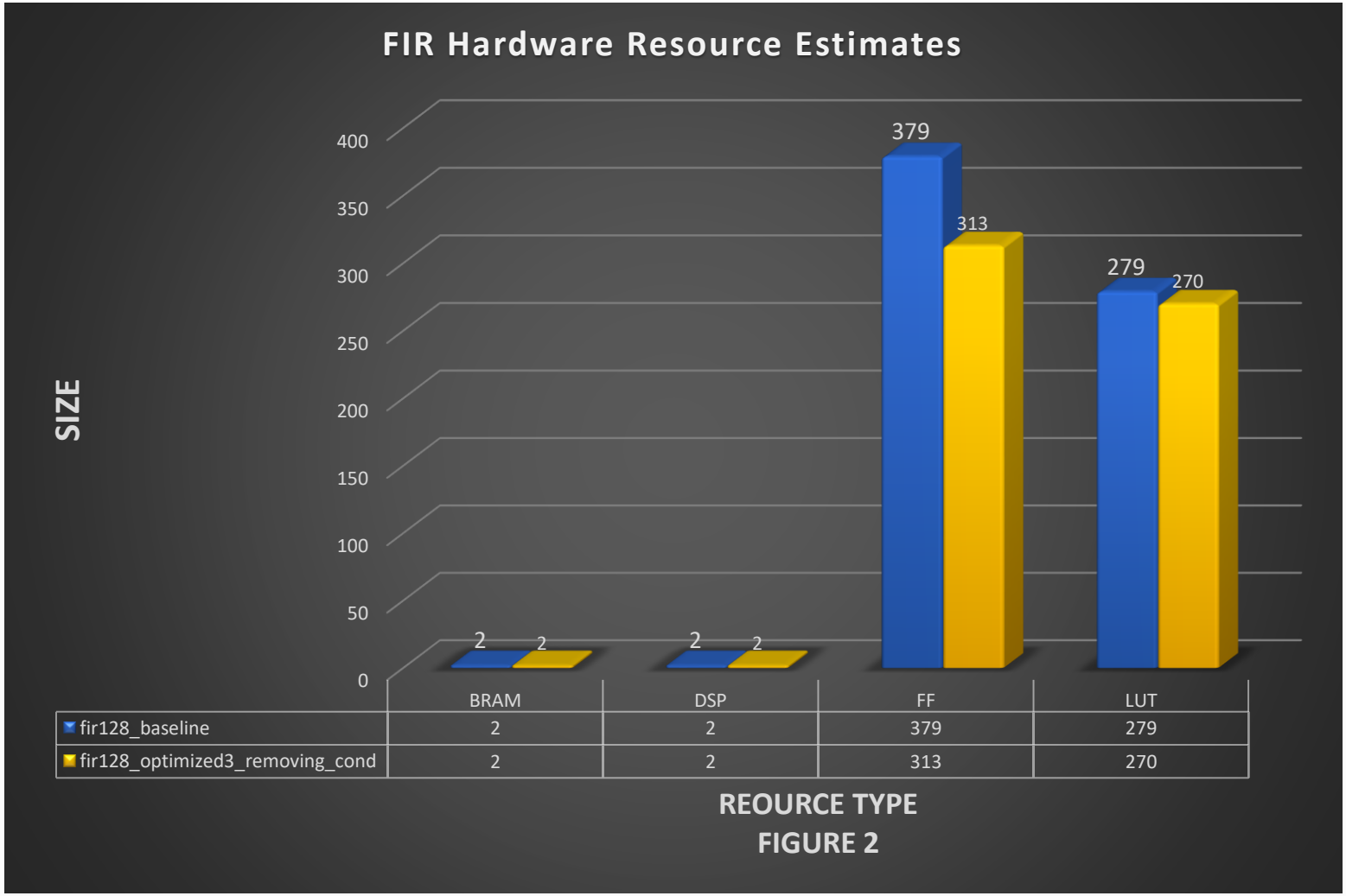


FIGURE 2

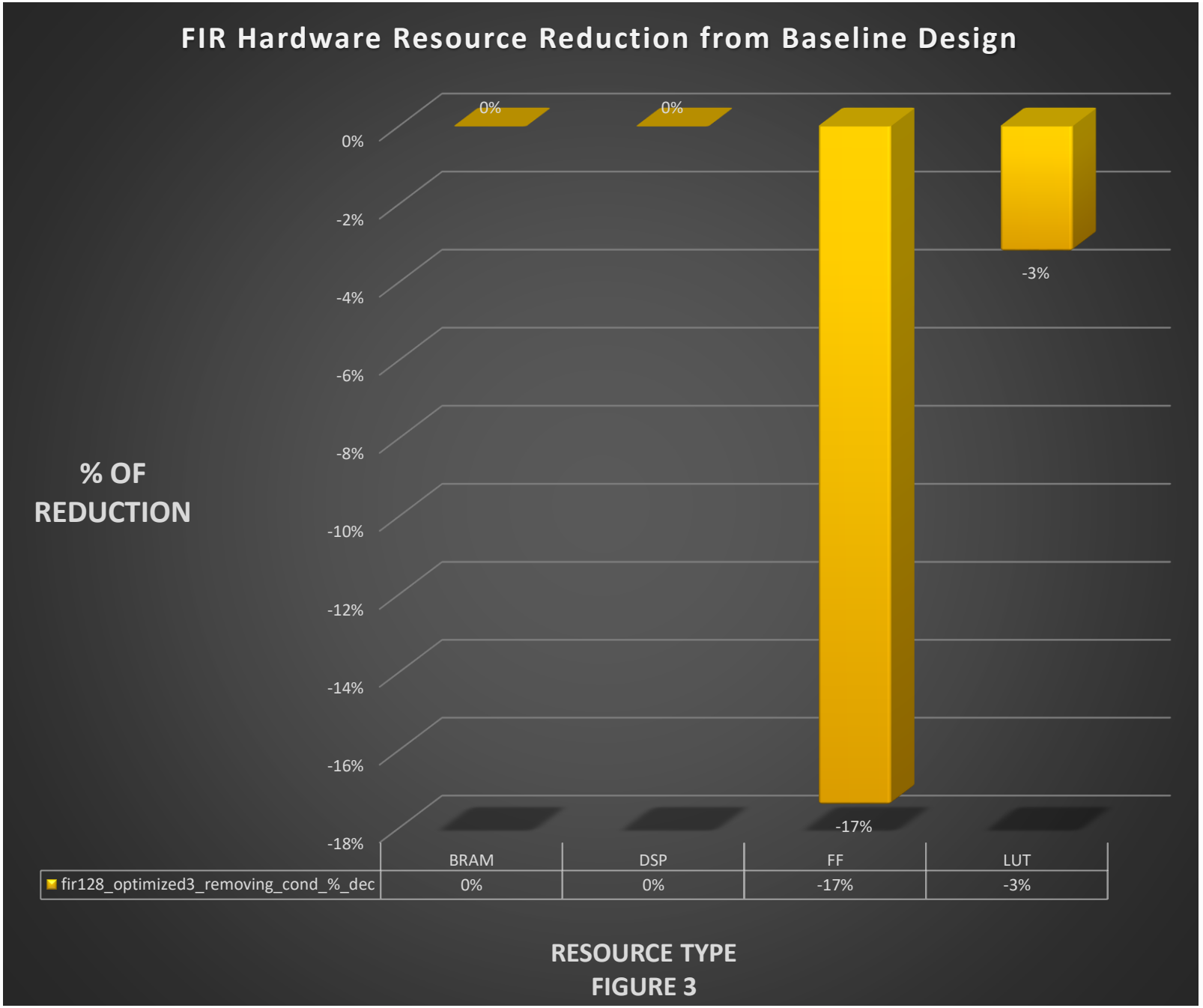


FIGURE 3

4. **Question 4 – FIR 128 Design with loop partitioning optimization**

- Max Throughput: **161 kHz**

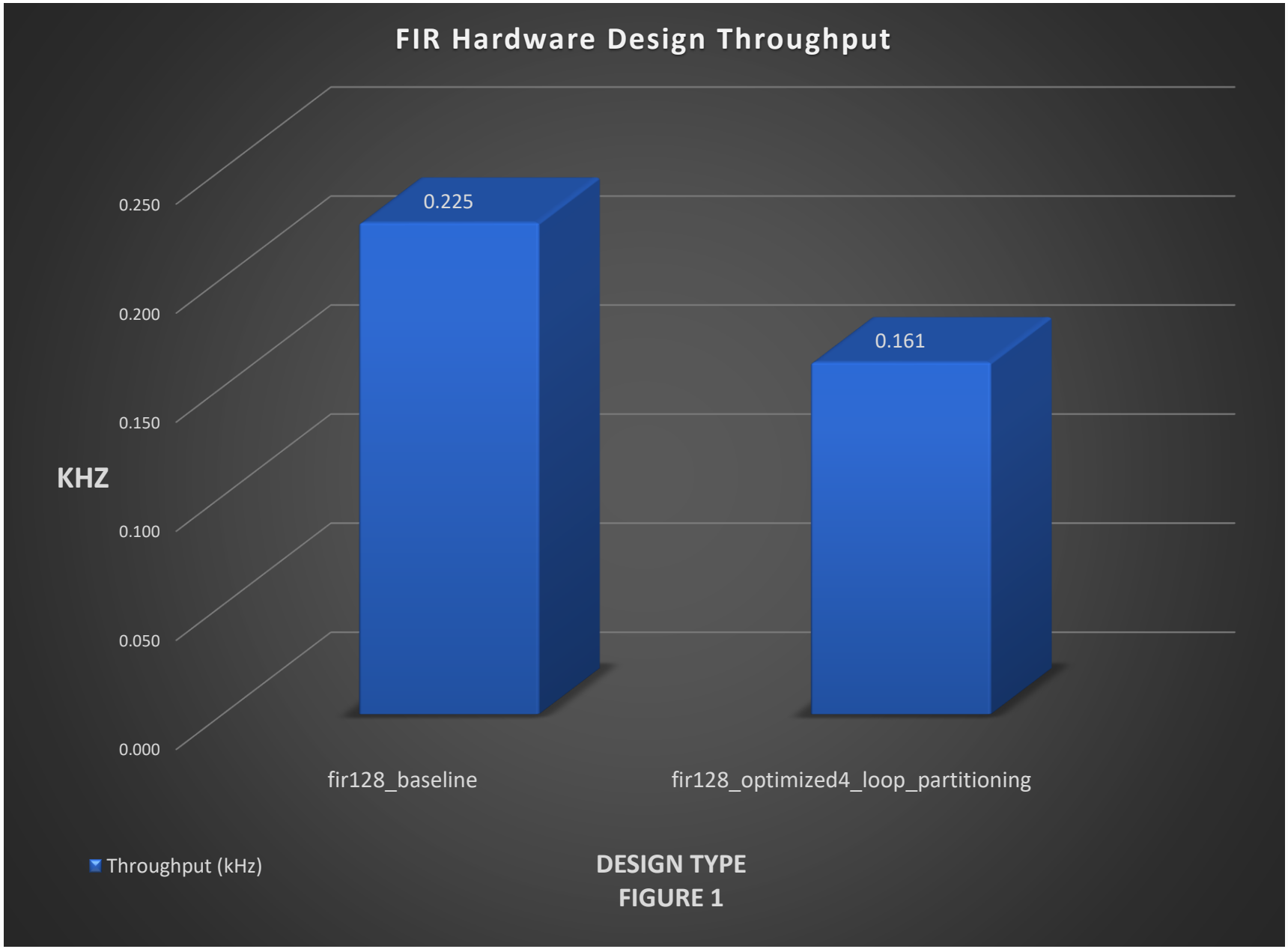


FIGURE 1

- Optimizations:

    - Separated the shift accumulator loop into 2 different loops. One for the tapped delay line logic and the other for the multiply accumulate logic.

- Analysis:
    - The goal of this design was to improve the performance and resource usage of the FIR. For this design the if/else statement used to check for last coefficient of the shift accumulator loop only needs to happen on the last iteration of the loop. Therefore, the statements within the if branch can be “hoisted” out of the loop before performing the multiply and accumulate logic. This design fails to improve on the performance of the baseline producing a moderate decrease in throughput as can be seen in **Figure 1**. However, **Figure 2** shows that a moderate number of resources were reduced because of the loop partitioning. The number of flip flops used saw the greatest reduction in resources having been reduced by approximately 17% of what the baseline design required as seen in **Figure 3**. The change in resources is a result of removing the branching statements from the design.

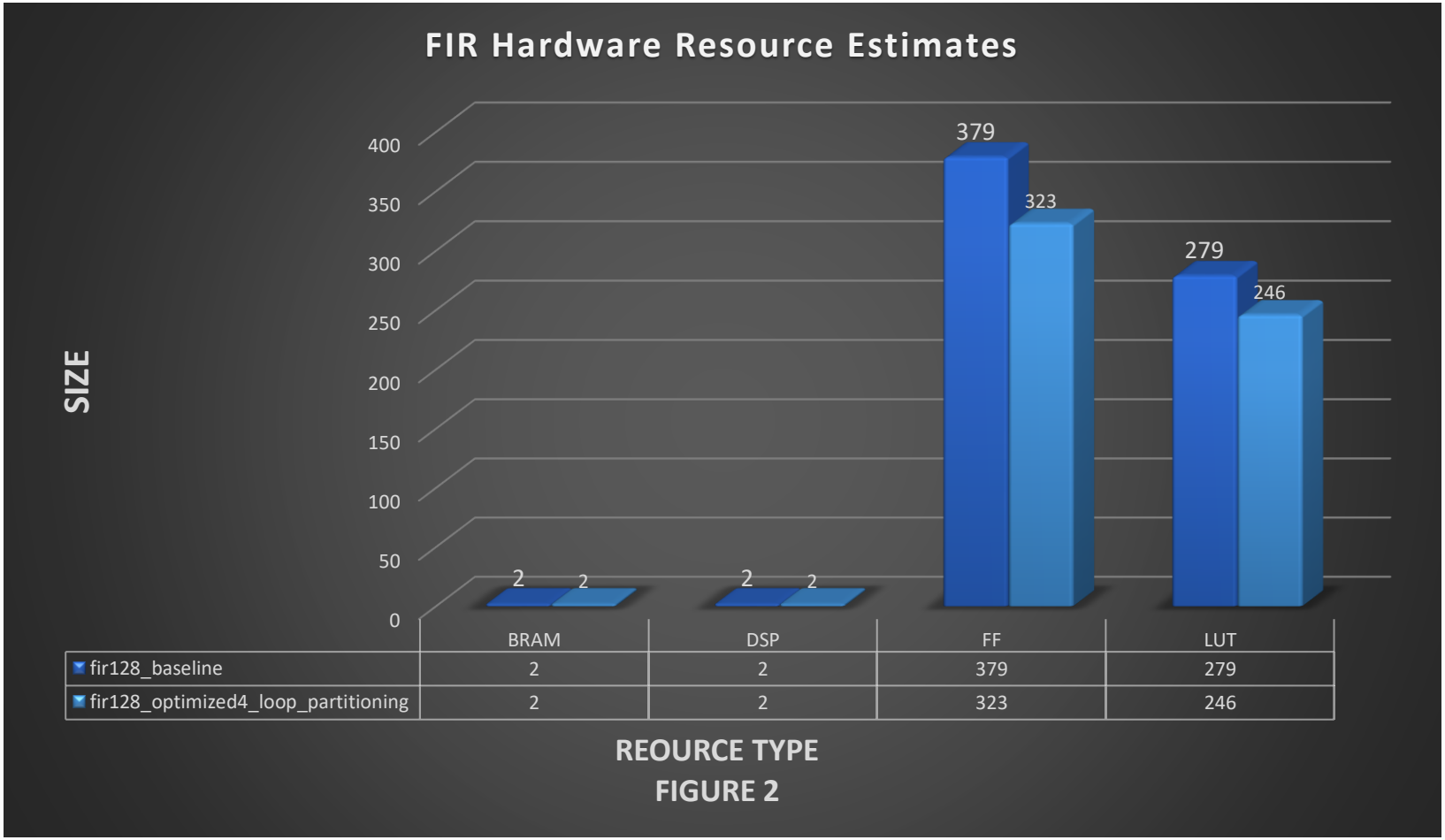
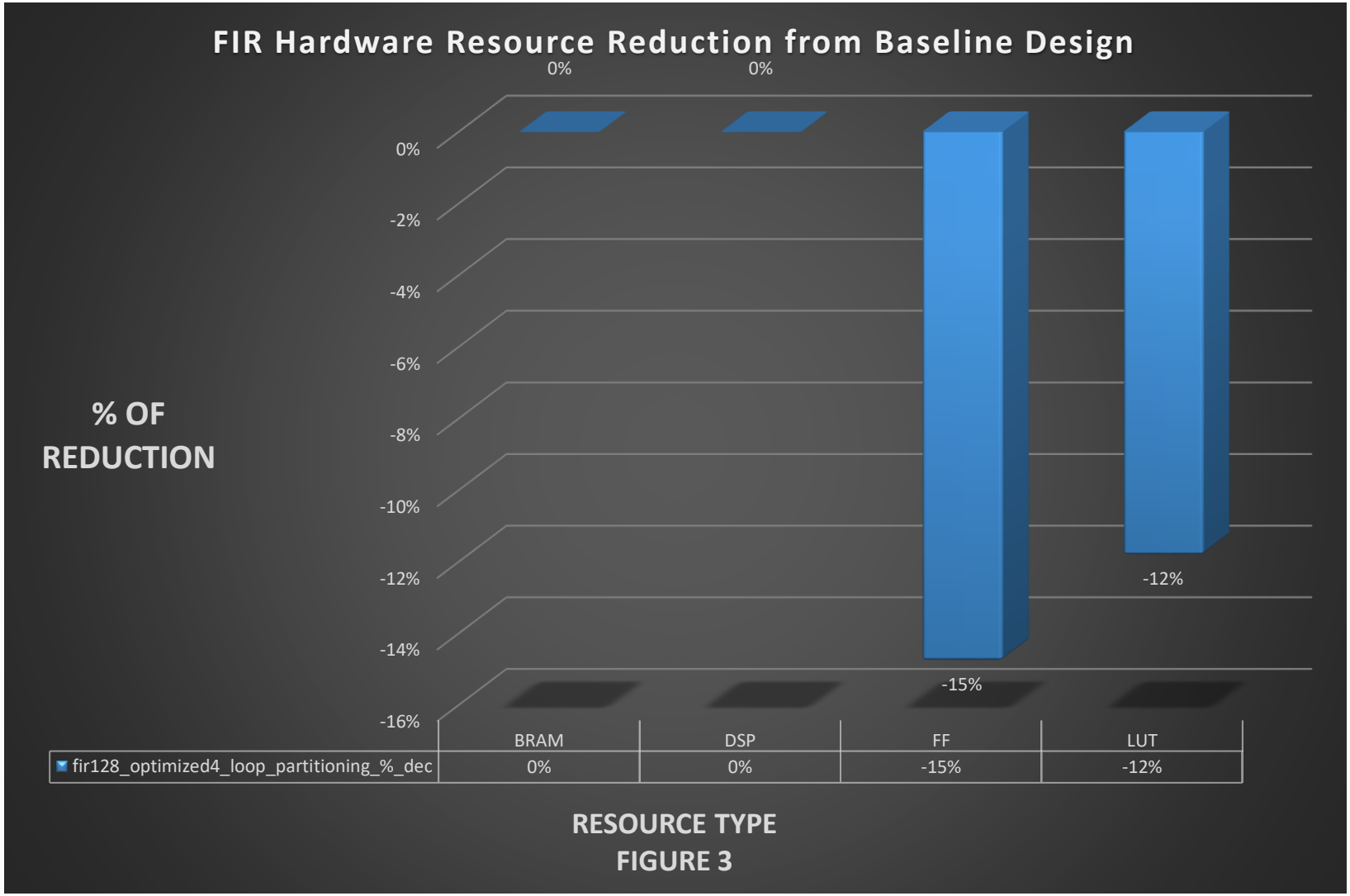

FIR Hardware Resource Estimates
SIZE
400
350
300
250
200
150
100
50
0
379
323
279
246
2
2
2
2
BRAM
DSP
FF
LUT
fir128_baseline
2
2
379
279
fir128_optimized4_loop_partitioning
2
2
323
246
REOURCE TYPE

FIGURE 2



5. **Question 1 – FIR 128 Design with memory partitioning optimization**

    - Max Throughput: **Complete Memory Partition** - **281 kHz**

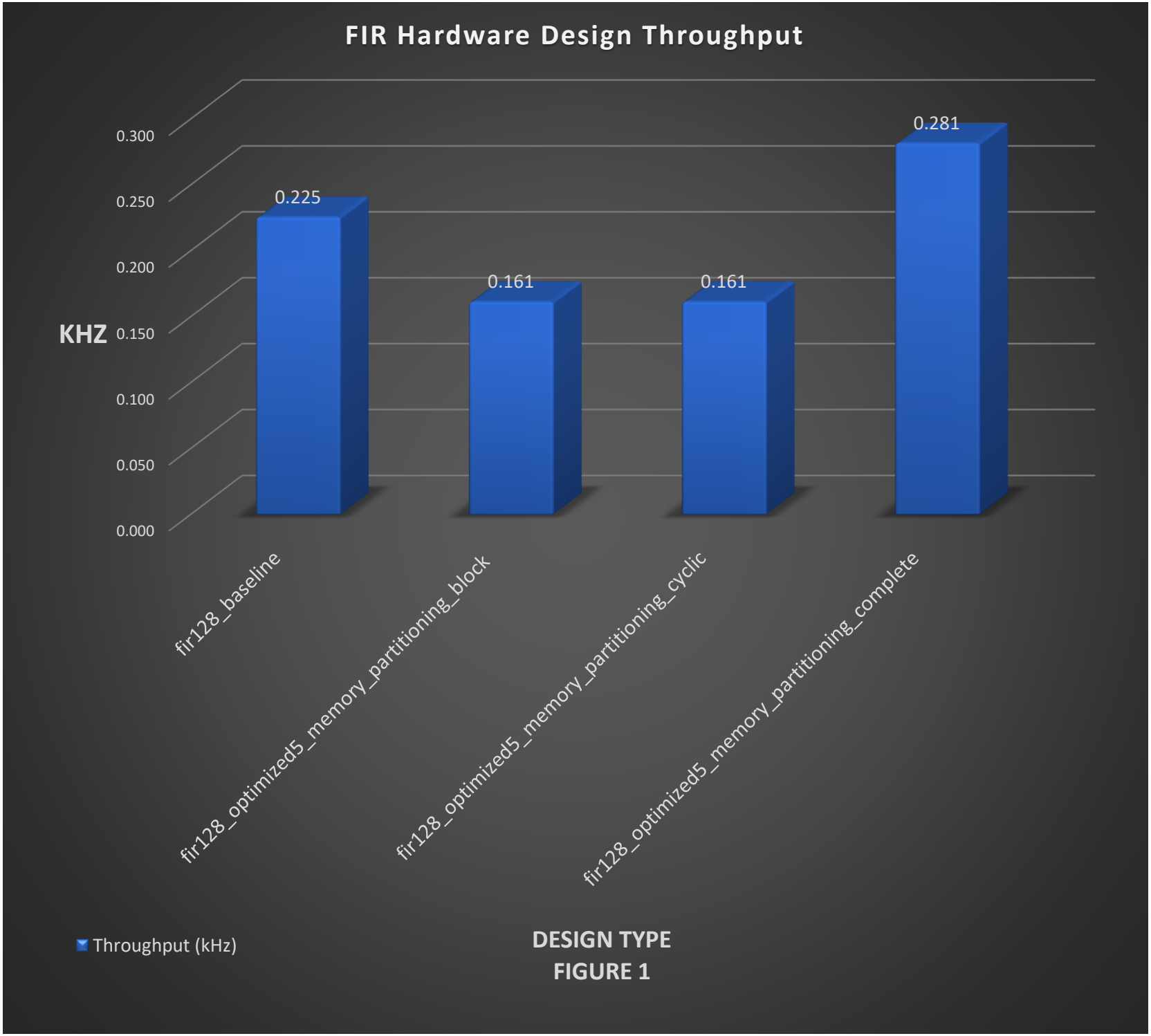

FIR Hardware Design Throughput
KHZ
0.300
0.250
0.200
0.150
0.100
0.050
0.000
0.225
0.161
0.161
0.281
fir128_baseline
fir128_optimized5_memory_partitioning_block
fir128_optimized5_memory_partitioning_cyclic
fir128_optimized5_memory_partitioning_complete
Throughput (kHz)
DESIGN TYPE

FIGURE 1

- Optimizations:
    - Updated the design to use block memory partitioning.
    - Updated the design to use cyclic memory partitioning.
    - Updated the design to use complete memory partitioning.
- Analysis:
    - The goal of this design was to improve performance by segregating the memory. The block partition optimization splits each array in this design into equally sized blocks of consecutive elements of the original array. This caused a moderate decrease in the performance of the baseline and a slight decrease in the number of resources used, reducing the number of flip flops needed by approximately 4%. However, it produced a moderate increase in the number of lookup tables required, enhancing its value by approximately 24%. Overall, it provided no significant benefits over the baseline design when compared to the other memory partitioning designs. **Figure 3** demonstrates the change in resource usage due to block memory partitioning. The cyclic partition optimization performed almost the same as the block memory partition providing a decrease in throughput compared to the baseline while using approximately the same number of resources as the memory partition design. The complete memory partition optimization is the only memory optimization that provides a moderate increase in throughput compared to the baseline design, but as trade off it uses an astronomical number of resources as seen in **Figure 3**. Performance of each can vary depending on the size of and shape of the array that needs to be partitioned. Cyclic would work well for single dimension arrays that store patterns of data. Block would work well for single- or double-dimension arrays that are an even size. Complete would work well for an array that contains data with elements meant to be used individually by a task as opposed to a task needing access to a range of elements from within the array.

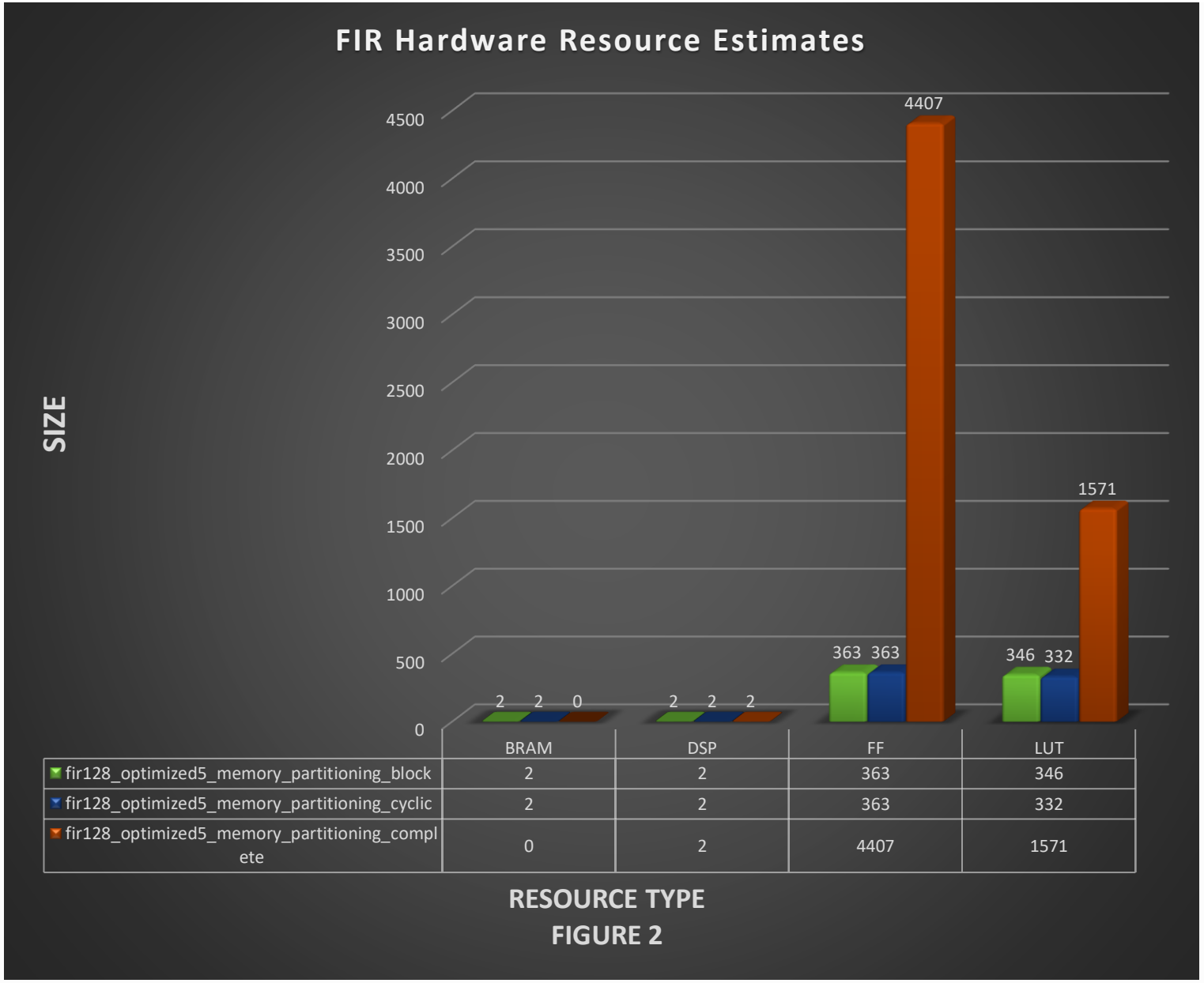

FIR Hardware Resource Estimates
SIZE
4500
4000
3500
3000
2500
2000
1500
1000
500
0
4407
1571
363 363
346 332
2 2 0
2 2 2
BRAM
DSP
FF
LUT
fir128_optimized5_memory_partitioning_block 2 2 363 346
fir128_optimized5_memory_partitioning_cyclic 2 2 363 332
fir128_optimized5_memory_partitioning_complete 0 2 4407 1571
RESOURCE TYPE

FIGURE 2

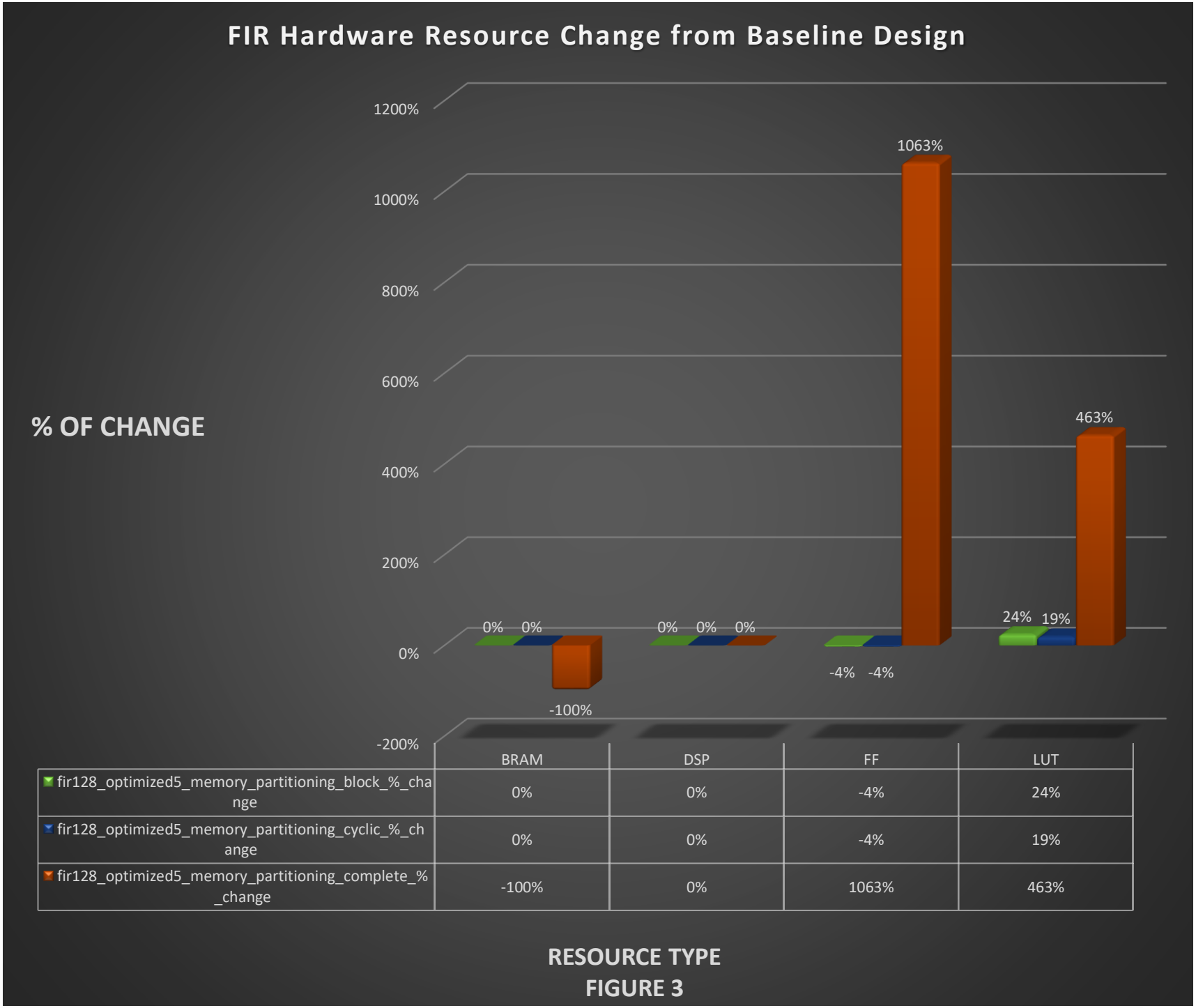

FIR Hardware Resource Change from Baseline Design
% OF CHANGE
1200%
1000%
800%
600%
400%
200%
0%
-200%
0% 0%
-100%
0% 0% 0%
-4% -4%
1063%
24% 19%
463%
BRAM
DSP
FF
LUT
fir128_optimized5_memory_partitioning_block_%_change 0% 0% -4% 24%
fir128_optimized5_memory_partitioning_cyclic_%_change 0% 0% -4% 19%
fir128_optimized5_memory_partitioning_complete_%_change -100% 0% 1063% 463%
RESOURCE TYPE

FIGURE 3

6. **Question 1 – FIR 128 Design with arbitrary bit width optimization**

- Max Throughput: **5.744 MHz**

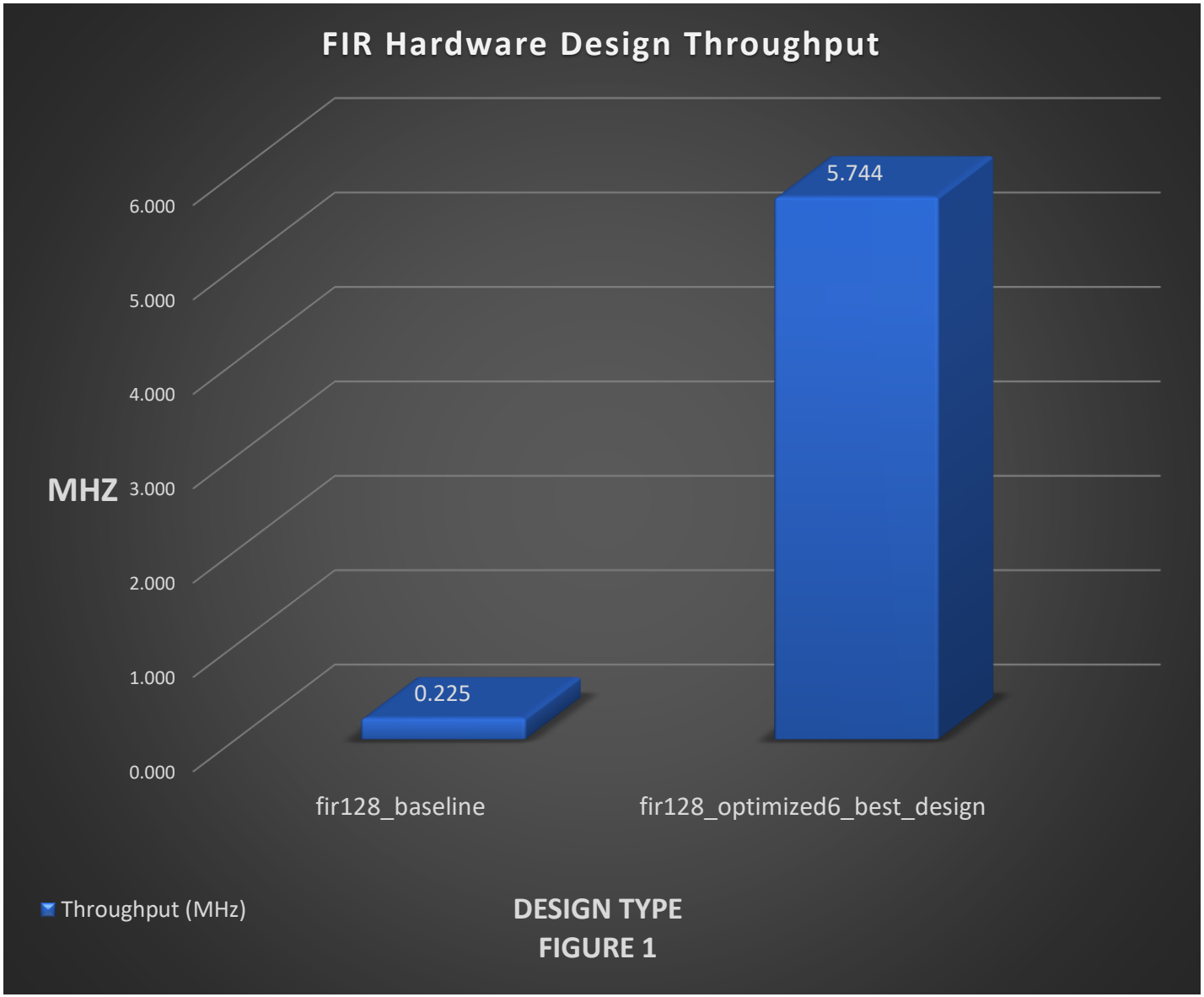


FIGURE 1

- Optimizations:
    - Reduced the number of bits used for coefficients to 5 which is the minimum usable bit width.
    - Reduced the number of bits used for the accumulator to 16 which is the minimum usable bit width.
    - Applied multiple optimizations to the coefficient and shift register memory:
        - Bound the coefficient storage to dual-port RAM.
        - Partitioned the coefficient storage by splitting the array into its individual elements. Which corresponds to resolving a memory into registers.
        - Stored the shift register in dual-port RAM.
        - Partitioned the shift register in cyclic fashion which allows the original array to be split into equally sized blocks interleaving the elements of the original array. Used a factor of 16 which creates 16 separate arrays.
        - Partitioned the shift accumulator loop into 2 separate loops.
        - Pipelined the tapped delay line loop with an initiation interval of 1.
        - Unrolled the tapped delay line loop.
        - Pipelined the multiply and accumulate loop with an initiation interval of 1.
        - Unrolled the multiply and accumulate loop with a factor of 32.
        - Removed branching statement from shift accumulator loop.
- Analysis:
    - The goal of this design was to increase the performance of the baseline design by enhancing the throughput metric. Decreasing the width of the bits used within the FIR helped reduce the likelihood of resource usage violations when synthesizing the design which also serves to help reduce the amount of power consumed by the hardware. The biggest challenge was determining the correct memory optimization. Dual-port RAM was selected as storage mechanism because it allows read operations on one port and both read and write operations on the

other port. The coefficient array could be optimized in multiple ways. Coefficients in digital filters typically don't change once the FIR implementation begins and are used in a convolution so that means that a multiply accumulate step will happen during the execution of the FIR logic. Knowing this along with the fact that we have small number of coefficients allows for this design to partition the coefficient array completely in memory. The shift register was partitioned in a cyclical fashion with a factor of 16. This produced 8 arrays with interleaving elements each with 16 values in them. This increased the read and write access in parts of the design that are pipelined. Loop partitioning was done to allow more work to be done in parallel. Implementing a pipeline in both loops increases the amount of concurrency and as a result latency of the design. The tapped delay line could be unrolled by a factor of 16 without memory access violations. The multiply accumulate loop could be unrolled by two times the factor of the tapped delay line loop because it does not rely on previous values as much. Performance was the primary objective of this design, so it does not optimize resources. However, care was taken to ensure that this design did not approach resource boundaries. This design proved to have the highest throughput of all the designs while also using some of the highest number of resources as seen in **Figure 2** and **Figure 3**.

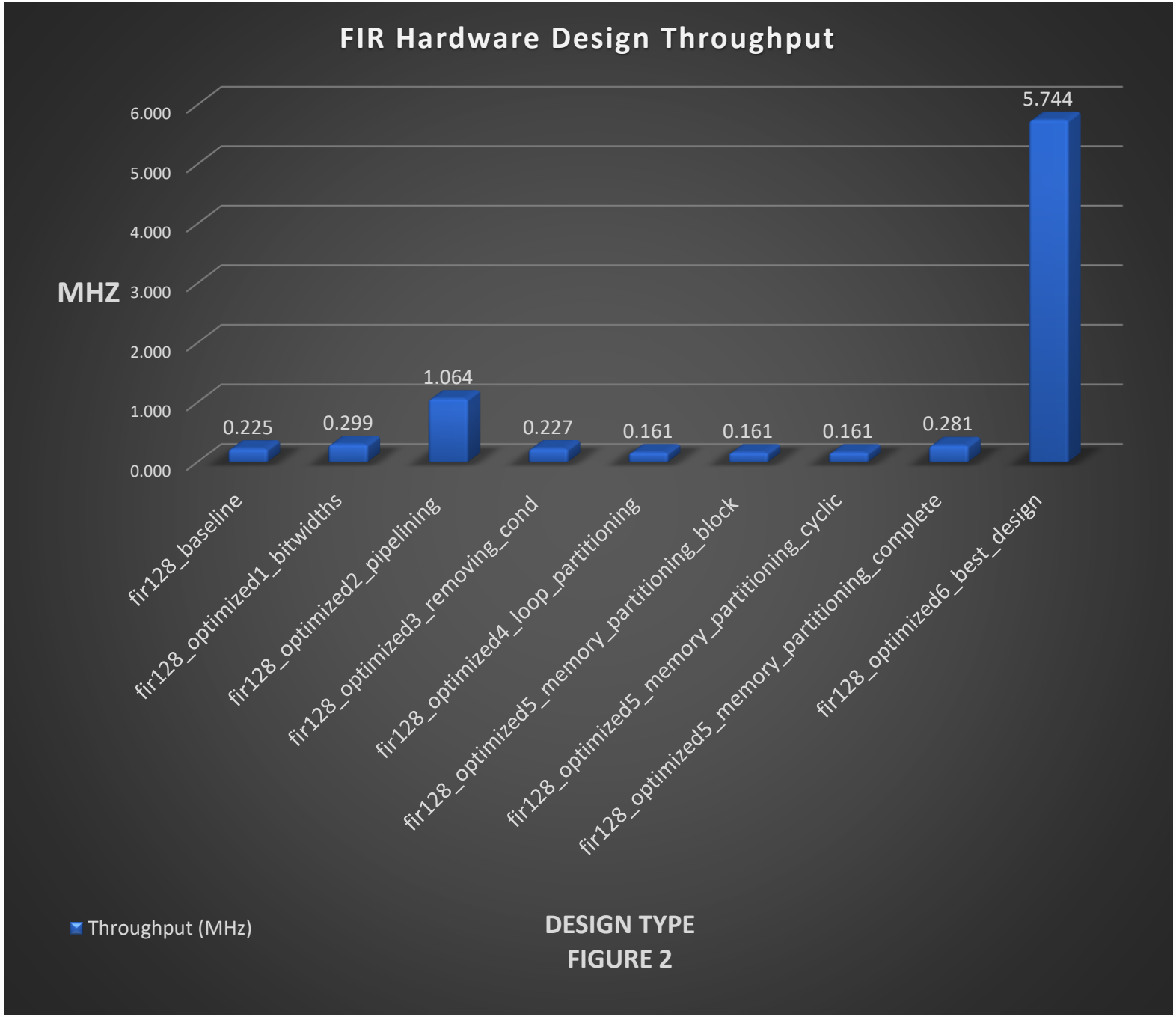
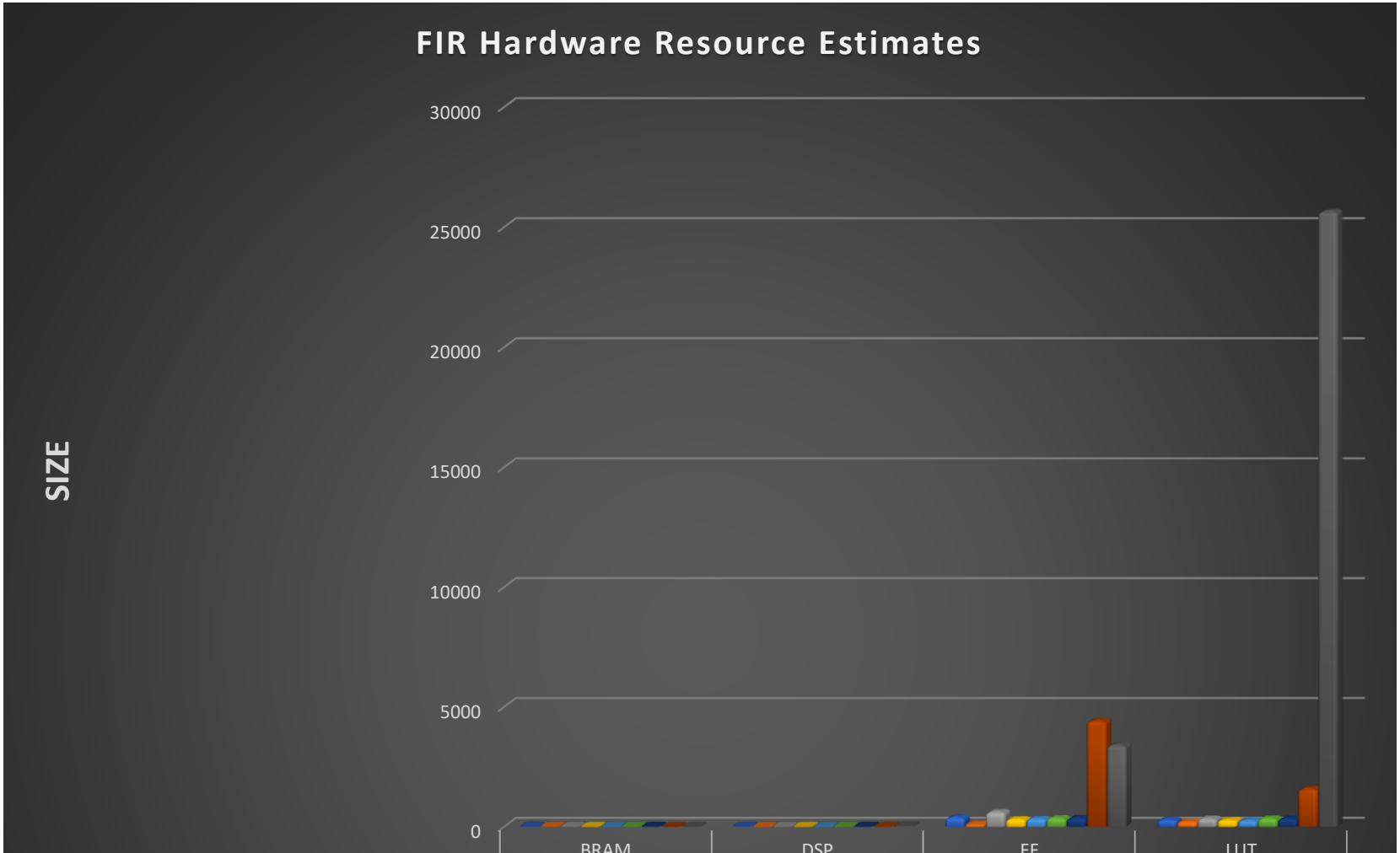

FIR Hardware Design Throughput
MHZ
6.000
5.000
4.000
3.000
2.000
1.000
0.000
0.225
0.299
1.064
0.227
0.161
0.161
0.161
0.281
5.744
fir128_baseline
fir128_optimized1_bitwidths
fir128_optimized2_pipelining
fir128_optimized3_removing_cond
fir128_optimized4_loop_partitioning
fir128_optimized5_memory_partitioning_block
fir128_optimized5_memory_partitioning_cyclic
fir128_optimized5_memory_partitioning_complete
fir128_optimized6_best_design
Throughput (MHz)
DESIGN TYPE

FIGURE 2



	BRAM	DSP	FF	LUT
fir128_baseline	2	2	379	279
fir128_optimized1_bitwidths	1	1	95	198
fir128_optimized2_pipelining	3	2	611	339
fir128_optimized3_removing_cond	2	2	313	270
fir128_optimized4_loop_partitioning	2	2	323	246
fir128_optimized5_memory_partitioning_block	2	2	363	346
fir128_optimized5_memory_partitioning_cyclic	2	2	363	332
fir128_optimized5_memory_partitioning_complete	0	2	4407	1571
fir128_optimized6_best_design	0	32	3397	25632

RESOURCE TYPE

FIGURE 3