Project 3: DFT

Question 1: ***What changes would this code require if you were to use a custom CORDIC similar to what you designed for Project: CORDIC? Compared to a baseline code with HLS math functions for cos() and sin(), would changing the accuracy of your CORDIC core make the DFT hardware resource usage change? How would it affect the performance? Note that you do not need to implement the CORDIC in your code, we are just asking you to discuss potential tradeoffs that would be possible if you used a CORDIC that you designed instead of the one from Xilinx.***

Question 2:

For this assignment, throughput was calculated as: DFT size * Fmax / interval

Implementation (256 point DFT)	**Resource Utilization**	**Performance**
Baseline for DFT	BRAM: 2 DSP: 25 LUT: 5113 FF: 4441	DFT/sec: 82.74 kHz Latency: 3.935e6 ns Clock Cycle: 393533
Math Functions Eliminated for DFT	BRAM: 4 DSP: 5 LUT: 1544 FF: 1229	DFT/sec: 89.66 kHz Latency: 3.935e6 ns Clock Cycle: 393493

When I change the size of DFT from 256 to 128, I get "FAIL: Output DOES NOT match the golden output". The precalculated coefficients no longer correspond to the new size of DFT.

Question 3:

Implementation (256 point DFT)	**Resource Utilization**	**Performance**
DFT Function Interface with 2 input arrays and 2 output arrays	BRAM: 4 DSP: 5 LUT: 1470 FF:1229	DFT/sec: 89.66 kHz Latency: 3.9356e6 ns Clock Cycle: 393493

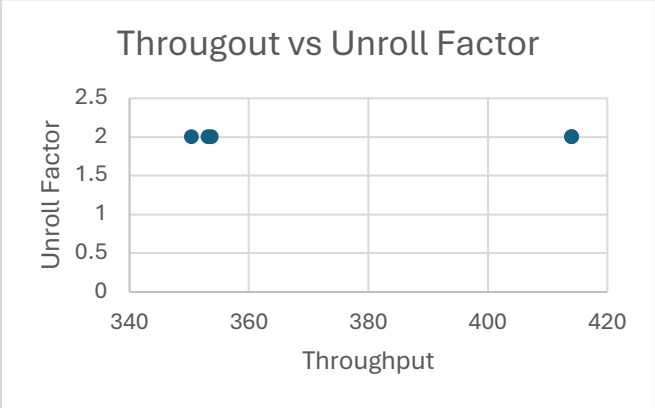
As expected, LUT utilization reduced, as the input and output memory are no longer shared.

Question 4:

The relationship between array partitioning and loop unrolling is that they both allow for parallel programming. In array partitioning, parallel programming happens by breaking the array up into smaller blocks and allowing simultaneous access to the array data. For loop unrolling, operations are able to running in parallel and simultaneously as well.

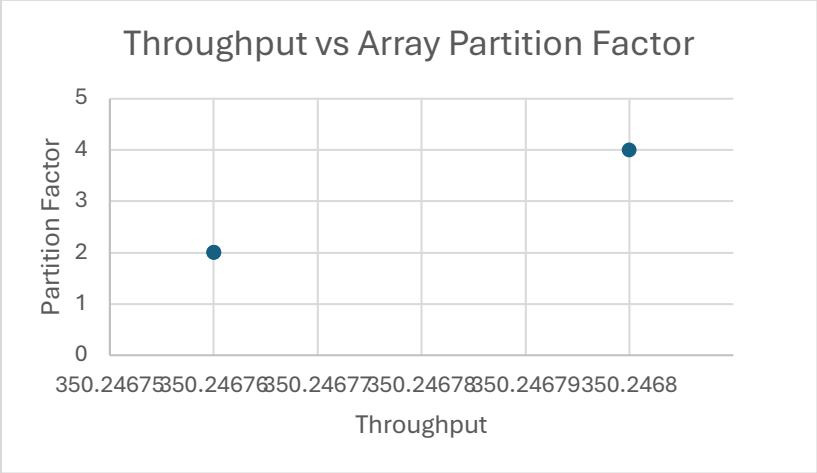
Implementation (256 point DFT)	**Resource Utilization**	**Performance**
#pragma HLS unroll factor=2 (first for loop)	BRAM: 4 DSP: 5 LUT: 1839 FF:1564	DFT/sec: 105.97kHz Latency: 3.329e6 ns Clock Cycle: 332932
#pragma HLS unroll factor=2 (second for loop)	BRAM: 4 DSP: 7 LUT: 1746 FF: 1523	DFT/sec: 90.41 kHz Latency: 3.333e6 ns Clock Cycle: 333316
#pragma HLS unroll factor=2 (last for loop)	BRAM: 6 DSP: 5 LUT: 1485 FF: 1236	DFT/sec: 89.69 kHz Latency: 3.934e6 ns Clock Cycle: 393365
#pragma HLS unroll factor=2 (3 for loops)	BRAM: 6 DSP: 7 LUT: 2566 FF:2454	DFT/sec: 90.544 kHz Latency: 3.328e6 ns Clock Cycle: 332804
#pragma HLS unroll factor=4 (first for loop)	BRAM: 4 DSP: 5 LUT: 2773 FF:2735	DFT/sec: 105.993 kHz Latency: 3.329e6 ns Clock Cycle: 332868

Observation: Depending on the complexity of the loop, a higher throughput is achievable when unrolling the loop. That is, you can make more calculations by unlooping and as a result get a higher throughput. For loops that are more simple and don't have much to calculate, the throughput isn't optimized very much.



Implementation (256 point DFT)	Resource Utilization	Performance
#pragma HLS ARRAY_PARTITION type=block factor=4	BRAM: 10 DSP: 8 LUT: 1510 FF:1229	DFT/sec: 89.663kHz Latency: 3.935e6 ns Clock Cycle: 393493
#pragma HLS ARRAY_PARTITION type=block Factor=2	BRAM: 6 DSP: 5 LUT: 1488 FF: 1229	DFT/sec: 89.663kHz Latency: 3.935e6ns Clock Cycle: 393493
#pragma HLS ARRAY_PARTITION type=block Factor=2 Dim=2	BRAM: 4 DSP: 5 LUT: 1470 FF: 1229	DFT/sec: 89.663kHz Latency: 3.935e6 ns Clock Cycle: 393493

Observation: Array partitioning didn't yield a big difference in throughput. This might be because of the dimension of the array. It seemed intuitive to use block partitioning but there wasn't much change with factor and dimension value changes.



Implementation (256 point DFT)	Resource Utilization	Performance
#pragma HLS unroll factor=2 (first for loop) **AND** #pragma HLS ARRAY_PARTITION type=block factor=2 dim=2	BRAM: 4 DSP: 5 LUT: 1839 FF: 1564	DFT/sec: 105.97kHz Latency: 3.329e6 ns Clock Cycle: 332933
#pragma HLS unroll factor=4	BRAM: 4 DSP: 5	DFT/sec: 105.993kHz Latency: 3.329e6 ns

(first for loop) **AND** #pragma HLS ARRAY_PARTITION type=block factor=2 dim=2	LUT: 2773 FF:2735	Clock Cycle: 332868

Comparing throughput values between loop unrolling and array partitioning, I was able to achieve the exact same throughput with loop unrolling factor of 2 both when I used block array partitioning and when I didn't do any partitioning. When I ran the same array partition with a higher factor loop unrolling, I got a slightly higher throughput. This shows that the dimension of the array is a big factor in whether array partitioning would result in an optimization. For this case, it seems the array partition could be skipped.

Implementation (1024 point DFT)	**Resource Utilization**	**Performance**
#pragma HLS unroll factor=2 (first for loop)	BRAM: 8 DSP: 5 LUT: 1832 FF:1584	DFT/sec: 26.810kHz Latency: 5.264e7 ns Clock Cycle: 5263876
#pragma HLS unroll	N/A	N/A- This call caused the C Synthesis to take a very long time, so I stopped and moved on to the next optimization
ARRAY_PARTITION variable= cos_coefficients_table dim = 1 cyclic factor = 2 **AND** #pragma HLS unroll factor=2	BRAM: 8 DSP: 5 LUT: 1850 FF:1585	DFT/sec: 26.811kHz Latency: 5.264e7 ns Clock Cycle: 5263876
Same as above plus #pragma HLS pipeline II=1	BRAM: 8 DSP: 5 LUT: 1838 FF:1519	DFT/sec: 27.474kHz Latency: 4.214e7 ns Clock Cycle: 4241276

I was not very successful in getting a higher throughput with DFT 1024. I found that my optimizations were taking a long time when doing a C Synth. I do not know if this was expected behavior, but it seemed to me the code was hanging, so I stopped the synthesis.

Question 5:

Implementation (256 point DFT)	**Resource Utilization**	**Performance**
#pragma HLS unroll factor=2 (first for loop)	BRAM: 4 DSP: 5 LUT: 1839 FF:1564	DFT/sec: 413.957 Latency: 3.329e6 ns Clock Cycle: 332932

My best architecture was for the 256 DFT, which provided a higher throughput with only loop unrolling (with factor of 2). I chose to only do loop unrolling because array partitioning wasn't providing better results than by the optimizing by the loop unroll. I was able to get lower values for BRAM, DSP, LUT and FF as opposed to when I use the array partition pragma. Also, a value of 2 provided less resource use than value of 4, with about similar throughput results.

Question 6: ***Describe the major changes that you made to your code to implement the streaming interface. What benefits does the streaming interface provide? What are the drawbacks?***

To implement the streaming interface, I replaced all the DTYPE arrays with a HLS streaming data structure. I made changes to both the dft.h file and dft.cpp file. To make this easier to test, I implemented this on the 256 DFT. The code essentially reads in the data stream and stores a local copy. From there you do the DFT computations and store data in temp variables. Once the DFT computation is completed, you stream back the real and imaginary values from the computation.

A benefit of the streaming class is that there is no data management required. This means that you can treat the data as a FIFO. A drawback to this is that the data is not stored. Therefore, you have to keep a local copy of the data that was streamed in.