https://youtu.be/8CkavD2o-2Q
https://github.com/bird-wizard/WES_237C/tree/main/fft

FFT Design Report:

Bit Reversal:
The main for loop is taken from the textbook, my only addition is wrapping it in another for loop to iterate over the real and imaginary values. I use HLS Pragma inline off to ensure the bit_reverse can be used efficiently. All arrays (4 total) in this function are partitioned moderately at a factor of 4. I used cyclic array partitioning to improve parallelism by spreading out the values in the arrays instead of having neighboring values in the same storage bank.

FFT:
When starting out with the large set of arrays, unrolling at any factor resulted in resource usage beyond what was available. I needed to find the minimum amount of arrays possible, while maintaining a decent pipeline size for max throughput. I figured, as long as the next functional call did not require the values of the previous function call I might be able to use the array values immediately. So in order to reduce the initial array requirements, I split the FFT stages into groups. I only need to partition my arrays when I need them instead of all at once at the top. First group would be the first 3 values, second group would be next 3, and last group would be the last 4 stages. The design is duplicated when calling the FFT stages as well, I have arrays that hold intermediate values until the end of the set.

The main portion of the FFT is in the fft_stages function. We follow the previous HLS pragma of cyclic array partitions and inline off. Most of the implementation is taken from the textbook. During development, I found that using HLS pipeline pragma anywhere besides the inner most loop causes high synthesis times. I used an unroll factor of 4 for the twiddle factors as a safe and moderate optimization.