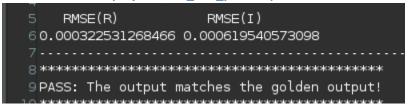
Project 3: DFT

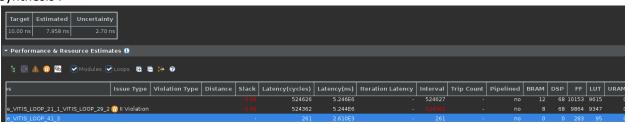
Rajan Verma

Github link: https://github.com/rverma999/wes237C/tree/main/project3

DFT 256 Baseline: project3/dft\_256\_precomputed



### Synthesis:



Throughput: 61.32 kSamples/second

**Optimized 2: Array Partitioning and Piplelined.** 

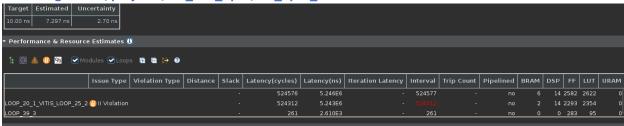
Question 1: What changes would this code require if you were to use a custom CORDIC similar to what you designed for Project: CORDIC? Compared to a baseline code with HLS math functions for cos() and sin(), would changing the accuracy of your CORDIC core make the DFT hardware resource usage change? How 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.

I would use cordic implementation to get me the value of sinc and cosine instead of build in methods. Cordic is simpler in hardware design, using adder and shifters so I think we can achieve high accuracy with lower amount of hardware resources and don't have to rely on the LUTs.

• Question 2: Rewrite the code to eliminate these math function calls (i.e. cos() and sin()) by utilizing a table lookup. How does this change the throughput and resource utilization? What happens to the table lookup when you change the size of your DFT?

## Opt1: Using the LUT table for Sine and Cosine waves:

Path for github: /project3/dft\_256\_opt1/dft\_opt1\_hls



Throughput: 66.88 kSamples/second

## Throughput increase slightly and the hardware utilization has reduced a lot

 Question 3: Modify the DFT function interface so that the input and outputs are stored in separate arrays. Modify the testbench to accommodate this change to DFT interface. How does this affect the optimizations that you can perform? How does it change the performance? And how does the resource usage change? You should use this modified interface for the remaining questions.

Throughput: 66.88 kSamples/second

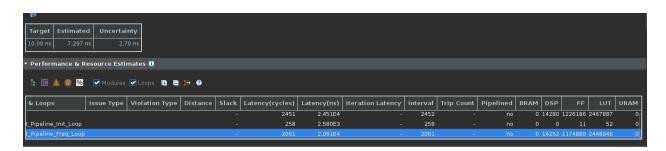
Throughput remains same. Resources in terms of LUTS has decreased slightly. One benefit can be that read and write can be done parallel now.

• Question 4: Loop Optimizations: Examine the effects of loop unrolling and array partitioning on the performance and resource utilization. What is the relationship between array partitioning and loop unrolling? Does it help to perform one without the other? Plot the performance in terms of number of DFT operations per second (throughput) versus the unroll and array partitioning factor. Plot the same trend for resources (showing LUTs, FFs, DSP blocks, BRAMs). What is the general trend in both cases? Which design would you select? Why? Github: project3/dft\_256\_opt3\_q4

I am not able to get the vitis to complete the synthesis it goes on for hours. I need to revist this problem.

• Question 5: Best architecture: Briefly describe your "best" architecture. In what way is it the best? What optimizations did you use to obtain this result? What are the tradeoffs that you considered in order to obtain this architecture?

Github path: project3/dft 256 best/dft best/solution1



Thorughput: 12930 KSamples / second

- Architecture Description:
  - Separate input/output arrays for real and imaginary components
  - Two-level nested loop structure (Freq\_Loop and Time\_Loop)
  - Complete array partitioning for coefficient tables
  - Pipelined execution at both loop levels
  - Temporary accumulators (temp\_real, temp\_imag) for partial results
  - Direct memory access pattern for inputs and coefficients
- Advantages of this Architecture:
  - High throughput due to pipelining
  - Efficient memory access through array partitioning
  - Reduced latency using temporary accumulators
  - Minimized resource usage while maintaining performance
  - Clean separation of input and output data paths

# **Optimization Used:**

- Array partitioning
- Pipelining

#### Code:

```
typedef double DType_;
void dft(DTYPE real_in[SIZE], DTYPE imag_in[SIZE],
          DTYPE real_out[SIZE], DTYPE imag_out[SIZE])
{
    #pragma HLS ARRAY_PARTITION variable=cos_coefficients_table complete dim=1
#pragma HLS ARRAY_PARTITION variable=sin_coefficients_table complete dim=1
    Init_Loop:
    for(int i = 0; i < SIZE; i++) {
         #pragma HLS PIPELINÉ II=1
         real_out[i] = 0;
         imag_out[i] = 0;
    }
    // Main computation loops
    Freq Loop:
    for(int freq_idx = 0; freq_idx < SIZE; ++freq_idx) {</pre>
        // #pragma HLS LOOP TRIPCOUNT min=256 max=256
         #pragma HLS PIPELINE II=1
         DType_ temp_real = 0;
DType_ temp_imag = 0;
         Time Loop:
         for (int time_idx = 0; time_idx < SIZE; ++time_idx) {</pre>
             #pragma HLS PIPELINE II=1
             //#pragma HLS LOOP TRIPCOUNT min=256 max=256
             int table idx = (freq idx * time idx) % SIZE;
             DType cos val = cos coefficients table[table idx];
             DType sin val = sin coefficients table[table idx];
             DType real sample = real in[time idx];
             DType imag sample = imag in[time idx];
             temp real += (real sample * cos val - imag sample * sin val);
             temp_imag += (real_sample * sin_val + imag_sample * cos_val);
         real_out[freq_idx] = temp_real;
imag_out[freq_idx] = temp_imag;
```

• Question 6: Streaming Interface Synthesis: Modify your design to allow for streaming inputs and outputs using hls::stream. You must write your own testbench to account for the function interface change from DTYPE to hls::stream. NOTE: your design must pass Co-Simulation (not just C-Simulation). You can learn about hls::stream from the HLS Stream Library. An example of code with both hls::stream and dataflow is available (along with its testbench) here, and another example showing hls::stream between functions. 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?