Project 3 : DFT

Rajan Verma

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

DFT 256 Baseline : **project3/dft_256_precomputed**

```
RMSE(R)            RMSE(I)
0.000322531268466 0.000619540573098
-----------------------------------------------
***********************************************
PASS: The output matches the golden output!
***********************************************
```

Synthesis :

Target	Estimated	Uncertainty
10.00 ns	7.958 ns	2.70 ns

Performance & Resource Estimates

Modules Loops

ps	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
				-0.66	524626	5.246E6	-	524627	-	no	12	68	10153	9615	0
e_VITIS_LOOP_21_1_VITIS_LOOP_29_2	II Violation			-0.66	524362	5.244E6	-	524362	-	no	8	68	9864	9347	0
e_VITIS_LOOP_41_3				-	261	2.610E3	-	261	-	no	0	0	283	95	0

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

Target	Estimated	Uncertainty
10.00 ns	7.297 ns	2.70 ns

Performance & Resource Estimates

Modules Loops

	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
				-	524576	5.246E6	-	524577	-	no	6	14	2582	2622	0
LOOP_20_1_VITIS_LOOP_25_2	II Violation			-	524312	5.243E6	-	524312	-	no	2	14	2293	2354	0
LOOP_39_3				-	261	2.610E3	-	261	-	no	0	0	283	95	0

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.**

Github : project3/dft_256_opt2/dft_opt2/solution1

```
Generating csim.exe
-----------------------------------------------
RMSE(R)            RMSE(I)
0.000322206469718 0.000299013743643
-----------------------------------------------
***********************************************
PASS: The output matches the golden output!
***********************************************
```

Target	Estimated	Uncertainty
10.00 ns	7.297 ns	2.70 ns

Performance & Resource Estimates

Modules Loops

	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	UR
				-	524576	5.246E6	-	524577	-	no	6	14	2582	2548	
ITIS_LOOP_20_1_VITIS_LOOP_25_2	II Violation			-	524312	5.243E6	-	524312	-	no	2	14	2293	2354	
ITIS_LOOP_39_3				-	261	2.610E3	-	261	-	no	0	0	283	95	

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.

```
-----------------------------------------------
   RMSE(R)            RMSE(I)
0.000322206469718 0.000299013743643
-----------------------------------------------
***********************************************
PASS: The output matches the golden output!
```

- **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

```
-----------------------------------------------
   RMSE(R)            RMSE(I)
0.000322206469718 0.000299013743643
-----------------------------------------------
***********************************************
```

Target	Estimated	Uncertainty
10.00 ns	7.297 ns	2.70 ns

Performance & Resource Estimates

Modules Loops

& Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
				-	2451	2.451E4	-	2452	-	no	0	14280	1226186	2467887	0
t_Pipeline_Init_Loop				-	258	2.580E3	-	258	-	no	0	0	11	52	0
t_Pipeline_Freq_Loop				-	2061	2.061E4	-	2061	-	no	0	14252	1174880	2448946	0

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_;
// Modified interface with separate input/output arrays
void dft(DTYPE real_in[SIZE], DTYPE imag_in[SIZE],
         DTYPE real_out[SIZE], DTYPE imag_out[SIZE])
{
    //#pragma HLS INTERFACE axis port=real_in
    //#pragma HLS INTERFACE axis port=imag_in
    //#pragma HLS INTERFACE axis port=real_out
    //#pragma HLS INTERFACE axis port=imag_out
    //#pragma HLS INTERFACE s_axilite port=return

    // Partition coefficient tables for parallel access
    #pragma HLS ARRAY_PARTITION variable=cos_coefficients_table complete dim=1
    #pragma HLS ARRAY_PARTITION variable=sin_coefficients_table complete dim=1

    // Initialize output arrays
    Init_Loop:
    for(int i = 0; i < SIZE; i++) {
        #pragma HLS PIPELINE 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) {
       // #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) {
            #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?