CSE237C Lab3 Report

Hanyang Xu A92068025 hax032

Question1: What changes would this code require if you were to use a custom CORDIC similar to what you designed for Project 2? 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? (dft256_baseline)

To use CORDIC in this code, we need to include the CORDIC implementation files in this project, and replace the sin() and cos() function used with cordic(THETA TYPE theta, COS SIN TYPE &s, COS SIN TYPE &c).

Changing the accuracy of the CORDIC core will most likely NOT change the DFT hardware resource usage since more rotations in CORDIC does not require additional hardware UNLESS the rotation number is larger than the stored pre-computed Scaling Factor and CORDIC Gain in which case the CORDIC hardware does need more storage to store additional coefficients.

CORDIC should perform better on FPGA-based system than other implementations such as table lookup and power series since FPGA's lack of hardware multiplier. However, since the Xilinx FPGAs has onboard DSPs, the CORDIC algorithm should be slower performance wise but use less resources.

Question2: 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 area? What happens to the table lookup when you change the size of your DFT? (dft256_optimized1)

	Baseline	**Table Lookup**
Throughput	256/4788994*8.625*10e-9=6.197kDFTPS	256/1049089*7.424*10e-9=33 DFTPS
BRAM_18K	18	4
DSP48E	203	16
FF	12471	1474
LUT	17971	2495

The throughput increases since table lookup has much simpler logic than multipliers, so the combinational delay is reduced to speed up the frequency. The area decreases since table lookup has much simpler logic than multipliers, so less hardware resources are used to form the logic.

When DFT size changes, the look up table size changes by DFT_size^2. Therefore, when DFT size increase to 1024, it is not practical anymore to use table lookup method since the storage for lookup table is too large.

Question3: Modify the DFT function interface so that the input and outputs are stored in separate arrays. How does this affect the optimizations that you can perform? How does it change the performance? What about the area results? (dft256_optimized2)

	Table Lookup	**Separated Arrays on Input/Output**
Throughput	256/1049089*7.424*10e-9=33 DFTPS	256/1049089*7.424*10e-9=32869 DFTPS
BRAM_18K	4	2
DSP48E	16	16
FF	1474	1428
LUT	2495	2365

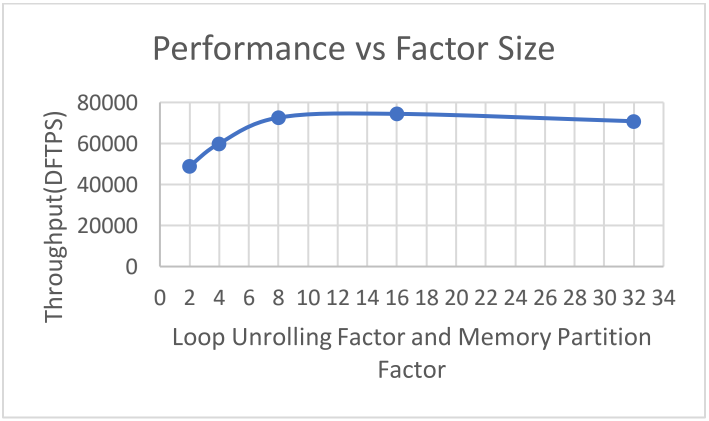
Separate the inputs and outputs array eliminate the usage of intermedia temp arrays and enable all the parallel computation optimization since now read and write can happens at the same time instead of sequentially.

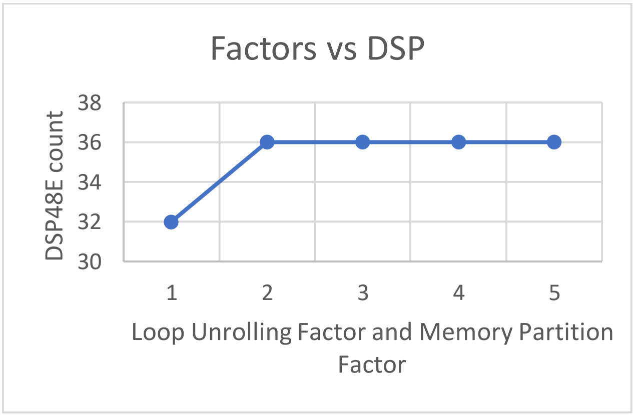
Separate the inputs and outputs alone does not improve performance since the implementation is still in a sequential manner, but it does reduce resource usage since it eliminates a read/write process. Specifically, it no longer needs to write to a temporary array first and then write back to the input array.

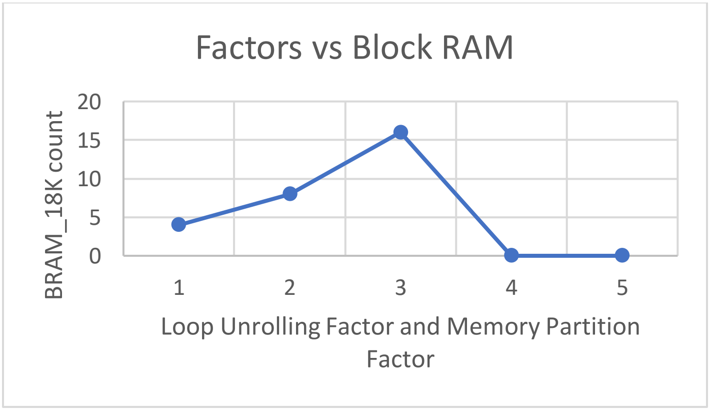
Question4: Study the effects of loop unrolling and array partitioning on the performance and area. What is the relationship between array partitioning and loop unrolling? Does it help to perform one without the other? Plot the performance in terms of number of matrix vector multiply operations per second (throughput) versus the unroll and array partitioning factor. Plot the same trend for area (showing LUTs, FFs, DSP blocks, BRAMs). What is the general trend in both cases? Which design would you select? Why? (dft256_optimized3)

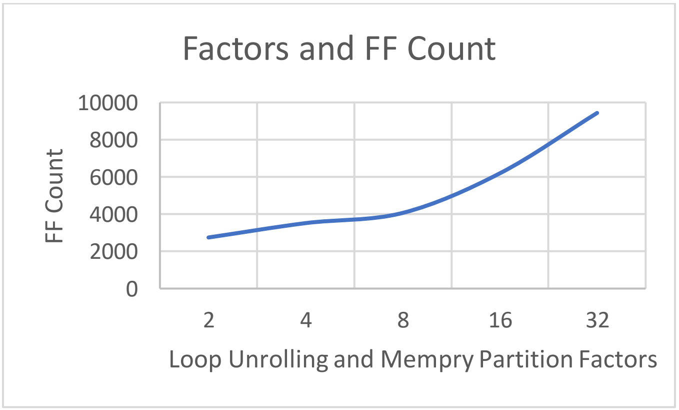
The relationship between array partitioning and loop unrolling is that array partitioning decides the effect loop unrolling can have on throughput. The parallelism that are provided by loop unrolling depends on parallel memory access. Therefore, if the design does not provide enough memory access concurrency with correct loop partitioning, then loop unrolling is serialized and can not achieve its full power.

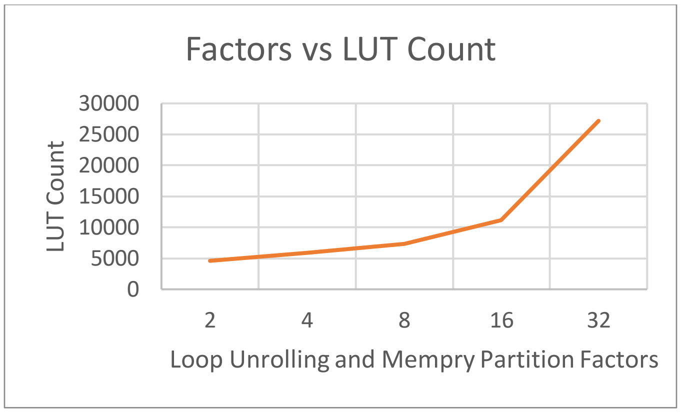
It does not help to do loop unrolling without array partitioning, experiments show that period increases when increase loop unrolling factor without array partitioning. It also does not help to do array partitioning without loop unrolling, experiments show that period stay the same when increase array partitioning factor without loop unrolling.







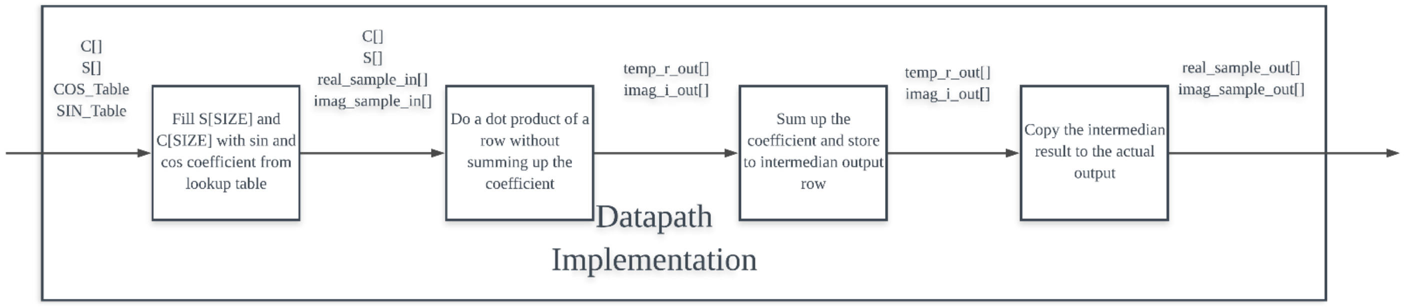




The general trend is throughput continue to rise with loop unrolling and memory partition factor until the factors are both 32. Resource usable goes up with factors since the parallelism increase. The trade of in area has diminish return in throughput. Therefore, I will choose factors to be 8 which is a sweet spot for performance vs area. Increasing factor 16 makes less sense since it shows less gain in performance as demonstrated with graph above.

Question5: Apply dataflow pragma to your design to improve throughput. How much improvement can you make with it? How much does your design use resources? What about BRAM usage? Please describe your architecture with figures on your report. (Make sure to add dataflow pragma on your top function.)

	Regular Version (factor = 8)	**Dataflow(factor = 8)**
Throughput	72578 DFTPS	256/353445*7.954 *10e-9=91061 DFTPS
BRAM_18K	16	32
DSP48E	36	36
FF	4067	5611
LUT	7374	8926


C[]
S[]
COS_Table
SIN_Table
Fill S[SIZE] and C[SIZE] with sin and cos coefficient from lookup table
C[]
S[]
real_sample_in[]
imag_sample_in[]
Do a dot product of a row without summing up the coefficient
temp_r_out[]
imag_i_out[]
Sum up the coefficient and store to intermedian output row
temp_r_out[]
imag_i_out[]
Copy the intermedian result to the actual output
real_sample_out[]
imag_sample_out[]
Datapath
Implementation

Question6: (Best architecture) Briefly describe your "best" architecture. In what way is it the best? What optimizations did you use to obtain this result? What is tradeoff you consider for the best architecture?

I choose to maximize throughput for my best architecture. For optimization, I choose Loop Unrolling, Memory partition and Datapath to achieve high throughput similar to question 5. Datapath breaks the entire implementation into four loops in a pipelined fashion. Loop unrolling and memory partition allows each element in each loop to run concurrently. The tradeoff for this architecture is the memory usage is high for storing lookup table and other intermediate results. The logic usage is also higher because of the parallelism it achieves requires a lot of hardware running concurrently.