WES 237C - Hardware for Embedded Systems, Kastner
Project 3, DFT

Ricardo Lizárraga
rilizarraga@ucsd.edu **PID: A69028483**

Submit to:
Ryan Charles Kastner
(TA) Zhenghua Ma
(TA) Abarajithan Gnaneswaran

Project: Discrete Fourier Transform (DFT)

1) Introduction

The goal of this project is to design architectures that implement the Discrete Fourier Transform (DFT). DFT is a common operation in signal processing which generates a discrete frequency domain representation of the discrete input signal. The input signal is a vector of samples and the matrix is a set of basis functions corresponding to discrete cosine and sine waveforms of different frequencies. The multiplication of the input signal with these basis functions describes how well the input signal correlates with those waveforms, which is the value of the Fourier series at that frequency.

We start with the direct implementation of the DFT which is a form of a matrix-vector multiplication. Future projects describe how to refactor the code to perform a fast Fourier transform.

2) Materials

You can download the project files here:

2) Materials

You can download the project files here:

- DFT Project Repo

The repository is organized into five folders, dft_8_precomputed, dft_32_precomputed, dft_256_precomputed, dft_1024_precomputed, and Demo. Each of the first four folders has its own testbench, out.gold.dat file, and coefficients.h file.

Each dft_xx_precomputed folder contains following files:

- dft.cpp - the baseline implementation for the dft function.
- dft.h - header file
- dft_test.cpp - test bench
- coefficientsX.h - a file containing the values of corresponding to one sine/cosine period sampled based upon the DFT points. For example, an 8 point DFT has 8 samples across both the sine and cosine function evenly spaced across one period. This is equivalent to dividing one rotation in the complex plane equally by the number of points in the DFT.
- out.gold.dat - “Golden” output. The testbench (dft_test.cpp) generates a sample input and calls the function dft in dft.cpp with that sample input. This output of the function is compared to the expected output. This will indicate PASS or FAIL. If it fails, then the code in the folder Demo: dft.bit | dft.tcl | project3_host.ipynbn dft is incorrect. There are four different versions of depending on the DFT size and way in which the DFT coefficients were generated.
- script.tcl and directives.tcl file to create the project

Demo folder contains one file:

- DFT.ipynb - notebook for demo

3) Project Goal

You should modify the code to create a number of different architectures that perform tradeoffs between performance and resource utilization. For dft_256_precomputed and dft_1024_precomputed designs, you need to use precomputed values from coefficients256.h and coefficients1024.h

For 256-point and 1024-point DFTs, you will create a report describing how you generated these different architectures (code restructuring, pragmas utilized, etc.). For each architecture you should provide its results including the resource utilization (BRAMs, DSP48, LUT, FF), and performance in terms of throughput (number of DFT operations/second), latency, clock cycles, clock frequency (which is fixed to 10 ns). You can do most of the design space exploration on the 256 point DFT. You should pick your “best” 256 architecture and synthesize that as a 1024 DFT.

The 8 and 32 point folders are provided for your convenience. If you would like, you can do some of your initial design space optimization on these smaller architectures. But it is not necessary to use these at all.

The key in this project is to understand the tradeoffs between loop optimizations (unrolling and pipelining) and data partitioning. Therefore you should focus on these optimizations.

wes_2025_r_lizarraga@waiter:~$ cd project3_DFT/
wes_2025_r_lizarraga@waiter:~/project3_DFT/dft_8_precomputed$ vitis_hls -f script.tcl

5) Questions

- **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.

For this project to use CORDIC, I would have to include the CORDIC implementation in this project, for this implementation I would have to replace Sin() & Cos() functions with corresponding CORDIC ROTATIONS.
One of the main changes would be performance, resources usage and the accuracy, CORDIC method is very effective in performance since only uses SHIFT and ADDITIONS (no multiplications, less DSP usage).
Relative to the Accuracy, adding enough ROTATION data in the table to comply the required accuracy FPGA based system has limited number of DSPs that are used for multiplications, however CORDIC method would perform better with less resources by using the pre-computed scaling factors and gain

- **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?

See dft_256_precomputer_optimized1

```
// Calculate the jth frequency sample sequentially
for (j = 0; j < SIZE; j += 1) {
    // Utilize HLS tool to calculate sine and cosine values
    //~c = cos(j * w);
    //~s = -sin(j * w);
    // Lets use the const tables in header file:
    // cos_coefficients_table & sin_coefficients_table
    // Instead of calculation on the fly
    int index = (i * j) % SIZE;
    c = cos_coefficients_table[index];
    s = sin_coefficients_table[index];
```

```
INFO: [SIM 2] ************** CSIM start **************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
make: 'csim.exe' is up to date.
-----------------------------------------
   RMSE(R)           RMSE(I)
0.000322206469718 0.000299013743643
-----------------------------------------
*****************************************
PASS: The output matches the golden output!
*****************************************
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] ************** CSIM finish **************
```

Baseline	Target: 10.00 ns; Estimated: 7.958 ns; Uncertainty: 2.70 ns
1(Q2) Lookup tables	Target: 10.00 ns; Estimated: 7.256 ns; Uncertainty: 2.70 ns

Optimization	
Baseline	

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
dft				-0.66	524626	5.246E6	-	524627	-	no	12	68	10153	9615	0
dft_Pipeline_VITIS_LOOP_18_1_VITIS_LOOP_26_2	II Violation			-0.66	524362	5.244E6	-	524362	-	no	8	68	9864	9347	0
dft_Pipeline_VITIS_LOOP_39_3				-	261	2.610E3	-	261	-	no	0	0	283	95	0

1(Q2)
Lookup tables

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
dft			-		393493	3.935E6	-	393494	-	no	4	5	1229	1544	
dft_Pipeline_VITIS_LOOP_15_1_VITIS_LOOP_23_2	II Violation		-		393232	3.932E6	-	393232	-	no	2	5	1202	1308	
dft_Pipeline_VITIS_LOOP_39_3			-		258	2.580E3	-	258	-	no	0	0	21	63	

$$\text{Throughput}_{\text{Per\_Input}} = \frac{1}{(\text{Clock cycles})(\text{Interval})}$$

$$\text{Throughput}_{\text{Per\_DFTsize}} = \frac{(\text{DFTsize})}{(\text{Clock cycles}-\text{ns})(\text{Interval})}$$

	Baseline	Lookup Tables (Q1)
Clock Cycle Estimated (ns)	7.958	7.256
Interval	524627	393494
Throughput (KDFTPS)	$\frac{(256)}{(1/(7.958X10^{-9}))(524627)}$ = 61.31 KDFTPS	$\frac{(256)}{(1/7.256X10^{-9})(393494)}$ = 89.661
BRAM	12	4
DSP	68	5
FF	10153	1229
LUT	9615	1544

- **Throughput improved from 61.31 KDFTPS (kilo DFTs Per Second) to 89.661 KDFTPS**
- **Since we removed cos() & sin() functions, DSP usage was reduced from 68 to only 5**

Throughput improved (increased), since Lookup table logic is simpler and not requiring multipliers, the area and resources used is reduced significantly

- **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.**

See dft_256_precomputer_optimized2

dft.cpp:

```
void dft (DTYPE real_sample[SIZE], DTYPE imag_sample[SIZE], DTYPE real_out[SIZE], DTYPE imag_out[SIZE]) {
```

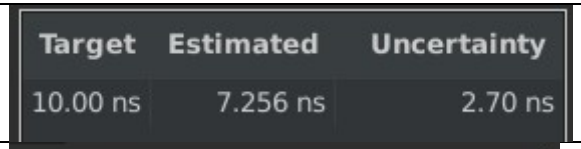
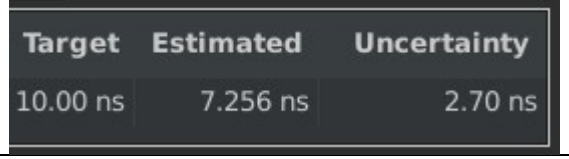
dft.h:

```
void dft(DTYPE real_sample[SIZE], DTYPE imag_sample[SIZE], DTYPE real_out[SIZE], DTYPE imag_out[SIZE]);
```

dft_test.cpp:

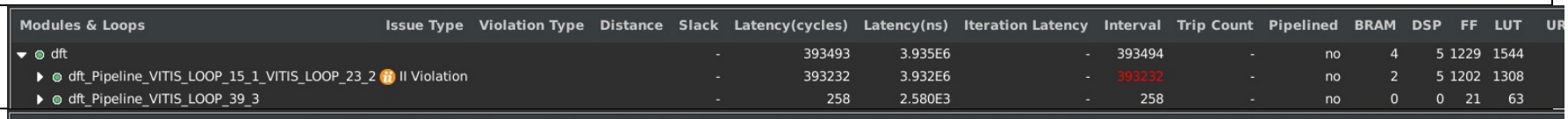
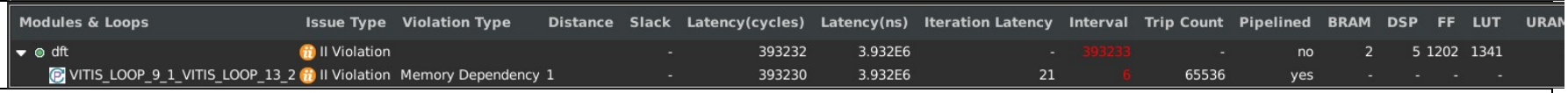
```
dft(In_R, In_I,Out_R,Out_I);
```

```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling ../../../../dft_test.cpp in debug mode
   Compiling ../../../../dft.cpp in debug mode
   Generating csim.exe
-----------------------------------------------
   RMSE(R)           RMSE(I)
0.000082730904978 0.000134893096401
-----------------------------------------------
*********************************************
PASS: The output matches the golden output!
*********************************************
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************
```

Optimization	
1(Q2) Lookup tables	
2 (Q3) I/O arrays separated	

Target	Estimated	Uncertainty
10.00 ns	7.256 ns	2.70 ns

Target	Estimated	Uncertainty
10.00 ns	7.256 ns	2.70 ns

Optimization	
1(Q2) Lookup tables	
2 (Q3) I/O arrays separated	

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	UR
dft				-	393493	3.935E6	-	393494	-	no	4	5	1229	1544	
dft_Pipeline_VITIS_LOOP_15_1_VITIS_LOOP_23_2	II Violation			-	393232	3.932E6	-	393232	-	no	2	5	1202	1308	
dft_Pipeline_VITIS_LOOP_39_3				-	258	2.580E3	-	258	-	no	0	0	21	63	

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
dft	II Violation			-	393232	3.932E6	-	393233	-	no	2	5	1202	1341	
VITIS_LOOP_9_1_VITIS_LOOP_13_2	II Violation	Memory Dependency	1	-	393230	3.932E6	21	6	65536	yes	-	-	-	-	

	Optimization1(Q2) Lookup tables	Optimization2(Q3) I/O arrays separated
Clock Cycle Estimated (ns)	7.256	7.256
Interval	393494	393233
Throughput (KDFTPS)	$\frac{(256)}{(1/(7.256X10^{-9}))(393494)} = 89.661$	$\frac{(256)}{(1/(7.256X10^{-9}))(393233)} = 89.720$
BRAM	4	2
DSP	5	5
FF	1229	1202
LUT	1544	1341

The separation of the input and output arrays didn't improve performance significantly (see throughput), but the separation of the input and output arrays, eliminate the intermediate temporal arrays and allow us to perform parallel processing, so instead of sequential read & write, now it can be in parallel.

- **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?

The relationship between array partitioning and loop unrolling is that loop unrolling allows parallelism on memory access, and this has a significant effect on throughput.
If the configuration doesn't provide enough memory access concurrency with the right loop partitioning, then the loop unrolling is serialized (sequentially) and cannot achieve its full potential.,

To do Loop unrolling without array partitioning causes period increases when increase loop unrolling factor without array partitioning.

HLS unroll factor=8 **without** Partition arrays:

Target	Estimated	Uncertainty
10.00 ns	8.567 ns	2.70 ns

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
dft				-1.27	332801	3.328E6	-	332802	-	no	2	7	2617	2272	0
VITIS_LOOP_20_1	II Violation			-	332800	3.328E6	1300	-	256	no	-	-	-	-	-

HLS unroll factor=2 with Partition arrays:

Target	Estimated	Uncertainty
10.00 ns	8.495 ns	2.70 ns

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
dft				-1.19	333058	3.331E6	-	333059	-	no	4	7	1579	1631	0
VITIS_LOOP_19_1	II Violation			-	333057	3.331E6	1301	-	256	no	-	-	-	-	-

HLS unroll factor=4 with Partition arrays:

Target	Estimated	Uncertainty
10.00 ns	8.352 ns	2.70 ns

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
dft				-1.05	332801	3.328E6	-	332802	-	no	8	7	2122	2034	0
VITIS_LOOP_19_1	II Violation			-	332800	3.328E6	1300	-	256	no	-	-	-	-	-

HLS unroll factor=6 with Partition arrays:

Target	Estimated	Uncertainty
10.00 ns	9.162 ns	2.70 ns

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
dft				-1.86	335105	3.351E6	-	335106	-	no	12	7	5753	3950	0
VITIS_LOOP_19_1	II Violation			-	335104	3.351E6	1309	-	256	no	-	-	-	-	-

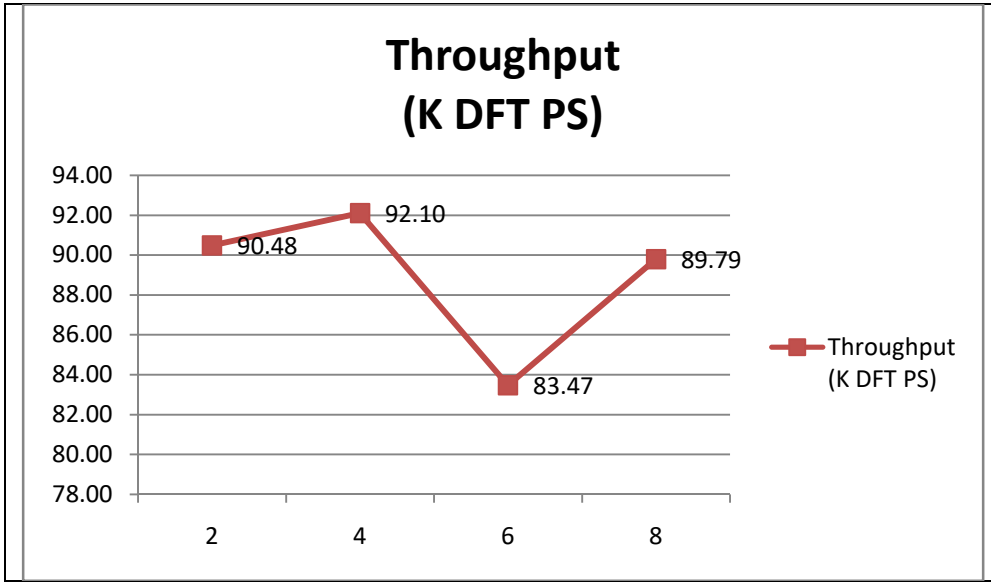
HLS unroll factor=8 with Partition arrays:

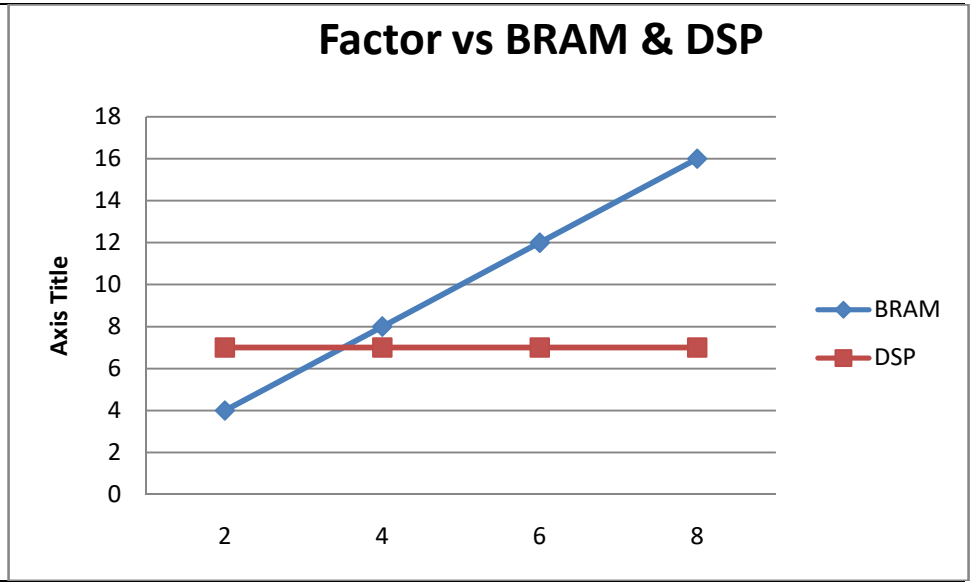
Target	Estimated	Uncertainty
10.00 ns	8.567 ns	2.70 ns

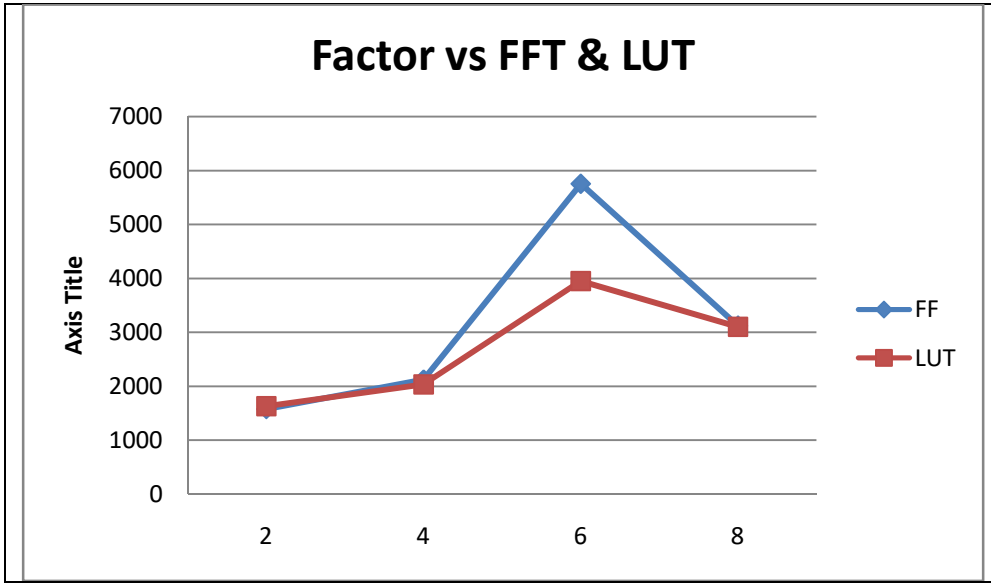
Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	U
dft				-1.27	332801	3.328E6	-	332802	-	no	16	7	3122	3103	
VITIS_LOOP_19_1	II Violation			-	332800	3.328E6	1300	-	256	no	-	-	-	-	

DFT size	Optimization Configuration	Unroll factor	Clock Cycle Estimated (ns)	Clock Cycle Estimated (ns)	Interval	Throughput (DFT PS)	Throughput (K DFT PS)	BRAM	DSP	FF	LUT
256	HLS unroll factor=8 WITHOUT Partition arrays	8	8.567	8.567E-09	332802	89789.44163	89.79	2	7	2617	2272
256	HLS unroll factor=2 with Partition arrays	2	8.495	8.495E-09	333059	90480.58677	90.48	4	7	1579	1631
256	HLS unroll factor=4 with Partition arrays	4	8.352	8.352E-09	332802	92100.83171	92.10	8	7	2122	2034
256	HLS unroll factor=6 with Partition arrays	6	9.152	9.152E-09	335106	83472.17887	83.47	12	7	5753	3950
256	HLS unroll factor=8 with Partition arrays	8	8.567	8.567E-09	332802	89789.44163	89.79	16	7	3122	3103

r







Code changes:

#pragma HLS array_partition

Partitions an array into smaller arrays or individual elements and provides the following:

- Results in RTL with multiple small memories or multiple registers instead of one large memory.
- Effectively increases the amount of read and write ports for the storage.
- Potentially improves the throughput of the design.
- Requires more memory instances or registers.

#pragma HLS array_partition variable=real_sample_in cyclic factor=8

Syntax

Place the pragma in the C source within the boundaries of the function where the array variable is defined.

```
#pragma HLS array_partition variable=<name> \
type=<type>  factor=<int>  dim=<int> off=true
```

`cyclic`

Cyclic partitioning creates smaller arrays by interleaving elements from the original array. The array is partitioned cyclically by putting one element into each new array before coming back to the first array to repeat the cycle until the array is fully partitioned.

For example, if `factor=3` is used:

- Element 0 is assigned to the first new array
- Element 1 is assigned to the second new array.
- Element 2 is assigned to the third new array.
- Element 3 is assigned to the first new array again.

`factor=<int>`

Specifies the number of smaller arrays that are to be created.

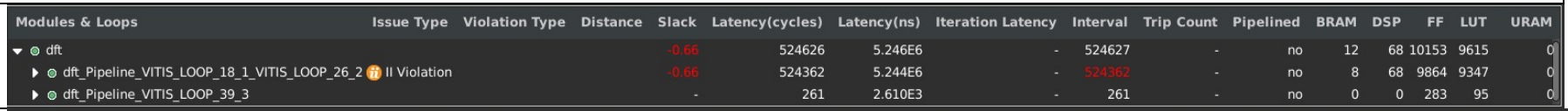
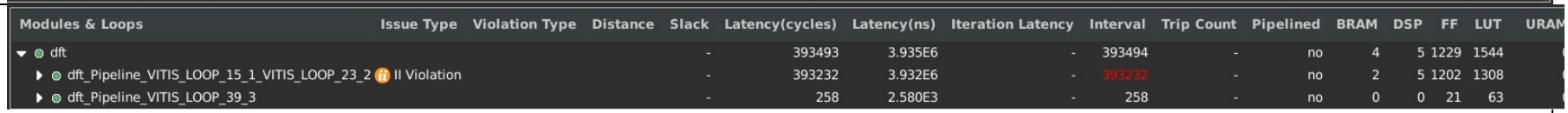
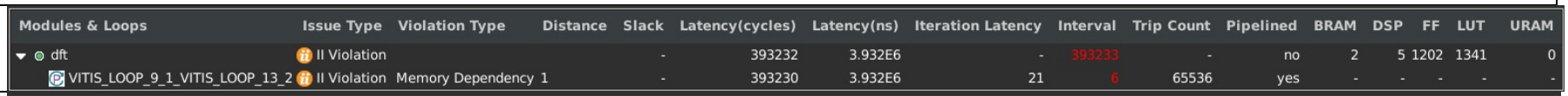
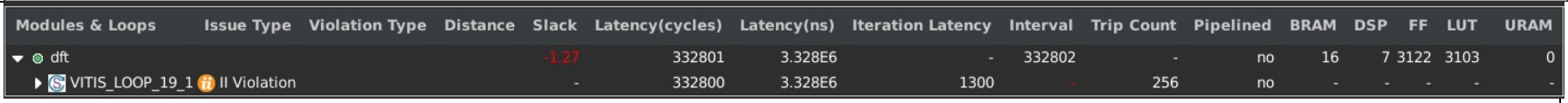
!! Important: For complete type partitioning, the factor is not specified. For block and cyclic partitioning, the `factor=` is required.

```
//Partition arrays into smaller arrays to individual elements to provide:
// multiple registers instead of one large memory to Potentially improves the throughput of the design
#pragma HLS array_partition variable=real_sample cyclic factor=8
#pragma HLS array_partition variable=imag_sample cyclic factor=8
#pragma HLS array_partition variable=real_out cyclic factor=8
#pragma HLS array_partition variable=imag_out cyclic factor=8
#pragma HLS array_partition variable=cos_coefficients_table cyclic factor=8
#pragma HLS array_partition variable=sin_coefficients_table cyclic factor=8
```

```
for (j = 0; j < SIZE; j += 1) {
    #pragma HLS unroll factor=8
    // Utilize HLS tool to calculate sine and cosine values
    int index = (i*j)%SIZE;
    cos = cos_coefficients_table[index];
    sin = sin_coefficients_table[index];

    // Multiply the current phasor with the appropriate input sample and keep
    // running sum
    temp_real += (real_sample[j] * cos - imag_sample[j] * sin);
    temp_imag += (real_sample[j] * sin + imag_sample[j] * cos);
}
```

```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling ../../../../dft_test.cpp in debug mode
   Compiling ../../../../dft.cpp in debug mode
   Generating csim.exe
-----------------------------------------
   RMSE(R)           RMSE(I)
0.000082730904978 0.000134893096401
-----------------------------------------
*****************************************
PASS: The output matches the golden output!
*****************************************
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************
```

Optimization	
Baseline	Target: 10.00 ns; Estimated: 7.958 ns; Uncertainty: 2.70 ns
1(Q2) Lookup tables	Target: 10.00 ns; Estimated: 7.256 ns; Uncertainty: 2.70 ns
2 (Q3) I/O arrays separated	Target: 10.00 ns; Estimated: 7.256 ns; Uncertainty: 2.70 ns
3 (Q4) Loop unrolling & Array partition	Target: 10.00 ns; Estimated: 8.567 ns; Uncertainty: 2.70 ns

Optimization	
Baseline	
1(Q2) Lookup tables	
2 (Q3) I/O arrays separated	
3 (Q4) Loop unrolling & Array partition	

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
dft				-0.66	524626	5.246E6	-	524627	-	no	12	68	10153	9615	0
dft_Pipeline_VITIS_LOOP_18_1_VITIS_LOOP_26_2	II Violation			-0.66	524362	5.244E6	-	524362	-	no	8	68	9864	9347	0
dft_Pipeline_VITIS_LOOP_39_3				-	261	2.610E3	-	261	-	no	0	0	283	95	0

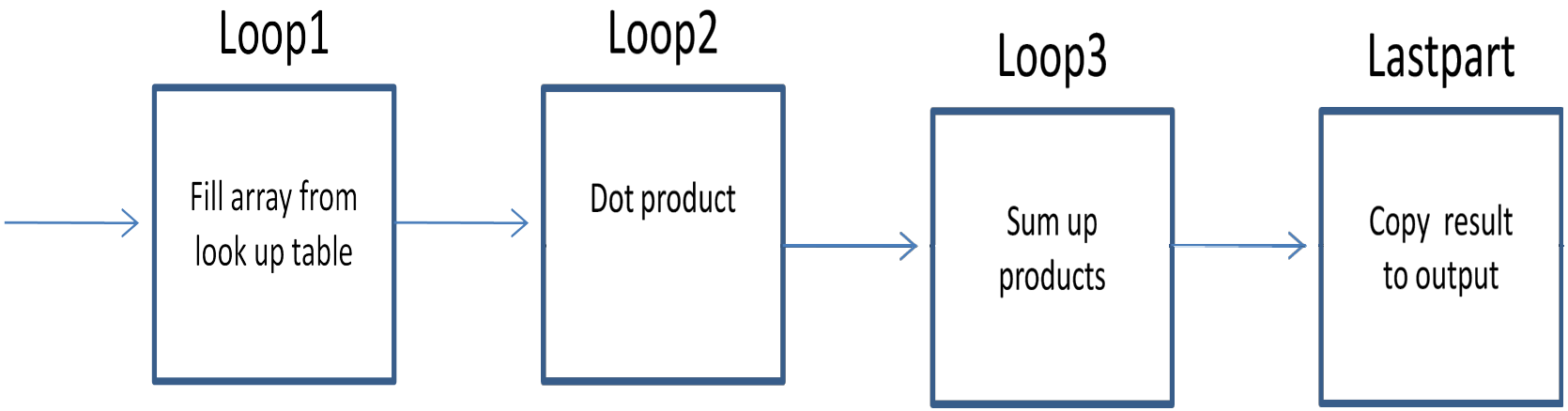
Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
dft				-	393493	3.935E6	-	393494	-	no	4	5	1229	1544	
dft_Pipeline_VITIS_LOOP_15_1_VITIS_LOOP_23_2	II Violation			-	393232	3.932E6	-	393232	-	no	2	5	1202	1308	
dft_Pipeline_VITIS_LOOP_39_3				-	258	2.580E3	-	258	-	no	0	0	21	63	

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
dft	II Violation			-	393232	3.932E6	-	393233	-	no	2	5	1202	1341	0
VITIS_LOOP_9_1_VITIS_LOOP_13_2	II Violation	Memory Dependency	1	-	393230	3.932E6	21	6	65536	yes	-	-	-	-	-

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
dft				-1.27	332801	3.328E6	-	332802	-	no	16	7	3122	3103	0
VITIS_LOOP_19_1	II Violation			-	332800	3.328E6	1300	-	256	no	-	-	-	-	-

- **Question 5: Dataflow:** Apply dataflow pragma to your design to improve throughput. You may need to change your code and make submodules so that it aligns with the task-level or function-level modularity that dataflow can exploit; Xilinx provides some examples of dataflow code. The Vitis HLS User Guide and this summary provide more information. How much improvement does dataflow provide? How does dataflow affect resource usage? What about BRAM usage specifically? Did you modify the code to make it more amenable to dataflow? If so, how? Please describe your architecture(s) with figures on your report.

Dataflow implementation



```c
#include<math.h>
#include "dft.h"
#include"coefficients256.h"

void Loop1(int i, DTYPE cos[SIZE], DTYPE sin[SIZE]) {
    int j;
    for (j = 0; j < SIZE; j ++) {
#pragma HLS unroll factor=8
        //
        int index = (i * j) % SIZE;
        cos[j] = cos_coefficients_table[index];
        sin[j] = sin_coefficients_table[index];
    }
}

void Loop2(DTYPE cos[SIZE], DTYPE sin[SIZE], DTYPE real_in[SIZE], DTYPE imag_in[SIZE], DTYPE temp_real_out[SIZE], DTYPE temp_imag_out[SIZE]) {
    int j;
    for (j = 0; j < SIZE; j ++) {
#pragma HLS unroll factor=8
        DTYPE c_temp = cos[j];
        DTYPE s_temp = sin[j];
        DTYPE temp_real_in = real_in[j];
        DTYPE temp_imag_in = imag_in[j];
        temp_real_out[j] = (temp_real_in * c_temp - temp_imag_in * s_temp);
        temp_imag_out[j] = (temp_real_in * s_temp + temp_imag_in * c_temp);
    }
}

void Loop3(int i, DTYPE temp_real_out[SIZE], DTYPE temp_imag_out[SIZE], DTYPE sum_real[SIZE], DTYPE sum_imag[SIZE]) {
    int j;
    for (j = 0; j < SIZE; j ++) {
#pragma HLS unroll factor=8
        if (j==0) {
            sum_real[i]=temp_real_out[j];
            sum_imag[i]=temp_imag_out[j];
        } else {
            sum_real[i]+=temp_real_out[j];
            sum_imag[i]+=temp_imag_out[j];
        }
    }
}

void LastPart(int i, DTYPE sum_real[SIZE], DTYPE sum_imag[SIZE], DTYPE real_out[SIZE], DTYPE imag_out[SIZE]) {
    real_out[i] = sum_real[i];
    imag_out[i] = sum_imag[i];
}

void dft (DTYPE real_in[SIZE], DTYPE imag_in[SIZE], DTYPE real_out[SIZE], DTYPE imag_out[SIZE]) {
    //Partition arrays into smaller arrays to individual elements to provide:
    // multiple registers instead of one large memory to Potentially improves
    // the throughput of the design
    #pragma HLS array_partition variable=real_in cyclic factor=8
    #pragma HLS array_partition variable=imag_in cyclic factor=8
    #pragma HLS array_partition variable=real_out cyclic factor=8
    #pragma HLS array_partition variable=imag_out cyclic factor=8
    #pragma HLS array_partition variable=cos_coefficients_table cyclic factor=8
    #pragma HLS array_partition variable=sin_coefficients_table cyclic factor=8
    // Compute each frequency domain sample
    for (int i = 0; i < SIZE; i ++) {
        #pragma HLS dataflow
        DTYPE cos[SIZE], sin[SIZE];
        DTYPE temp_real_out[SIZE], temp_imag_out[SIZE];
        DTYPE sum_real[SIZE], sum_imag[SIZE];
        Loop1(i, cos, sin);                                          // Lookup tables
        Loop2(cos, sin, real_in, imag_in, temp_real_out, temp_imag_out);// Multiply the current phasor with the appropriate input sample
        Loop3(i, temp_real_out, temp_imag_out, sum_real, sum_imag);     // Summation of products
        LastPart(i,sum_real,sum_imag, real_out, imag_out);           // Return values
    }
}
```

Dataflow, HLS unroll factor=8 with Partition arrays

Target	Estimated	Uncertainty
10.00 ns	11.058 ns	2.70 ns

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
dft	Timing Violation			-3.76	338710	3.745E6	-	338711	-	no	34	34	7077	8629	0
VITIS_LOOP_56_1	II&Timing Violation			-	338709	3.745E6	338709	-	256	dataflow	-	-	-	-	-

Dataflow, HLS unroll factor=4 with Partition arrays

Target	Estimated	Uncertainty
10.00 ns	7.256 ns	2.70 ns

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
dft				-	346901	3.469E6	-	346902	-	no	26	34	5009	6871	0
VITIS_LOOP_59_1	II Violation			-	346900	3.469E6	346900	-	256	dataflow	-	-	-	-	-

	Optimization1(Q4) Loop unrolling &Array partition	Optimization2(Q5) Dataflow
Clock Cycle Estimated (ns)	11.058	7.256
Interval	338711	346902
Throughput (KDFTPS)	$\frac{(256)}{(1/(11.058X10^{-9}))(338711)} = 68.35$	$\frac{(256)}{(1/(7.256X10^{-9}))(346902)} = 101.70$
BRAM	34	26
DSP	34	34
FF	7077	5009
LUT	8629	6871

Significant improvement on Throughput from 68.35 KDFTPS to 101.70 KDFTPS .
BRAM, FF & LUT resources were reduced

- **Question 6: 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?

For my best optimization will be the one design with Dataflow implemented, lookup tables, I/O separated, array unrolling and array partition.
All these based on the design that provides the best Throughput.

"**ap_axis**" refers to a data type used to represent an AXI4-Stream interface, allowing you to define streaming data paths within your C/C++ code that will be automatically translated into optimized hardware logic on an FPGA, essentially enabling efficient data transfer between different IP blocks or system components with a streaming protocol
Key points about ap_axis:
Data structure:
It's a specialized data type within the "hls" namespace, typically defined as ap_axis<DATA_WIDTH, K, D, B> where:
DATA_WIDTH: The width of the data payload in bits.

K: Number of user-defined "keep" bits (indicates which data bytes are valid).
D: Number of user-defined "last" bits (signals the end of a data burst).
B: Number of user-defined "user" bits (for custom signaling)

Target	Estimated	Uncertainty
10.00 ns	11.058 ns	2.70 ns

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT
dft	Timing Violation			-3.76	338710	3.745E6	-	338711	-	no	34	34	7077	8629
VITIS_LOOP_59_1	II&Timing Violation			-	338709	3.745E6	338709	-	256	dataflow	-	-	-	-

- **Question 7: 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?

dft.cpp | coefficients1024.h | dft_csim.log × | Synthesis Summary(solution1)

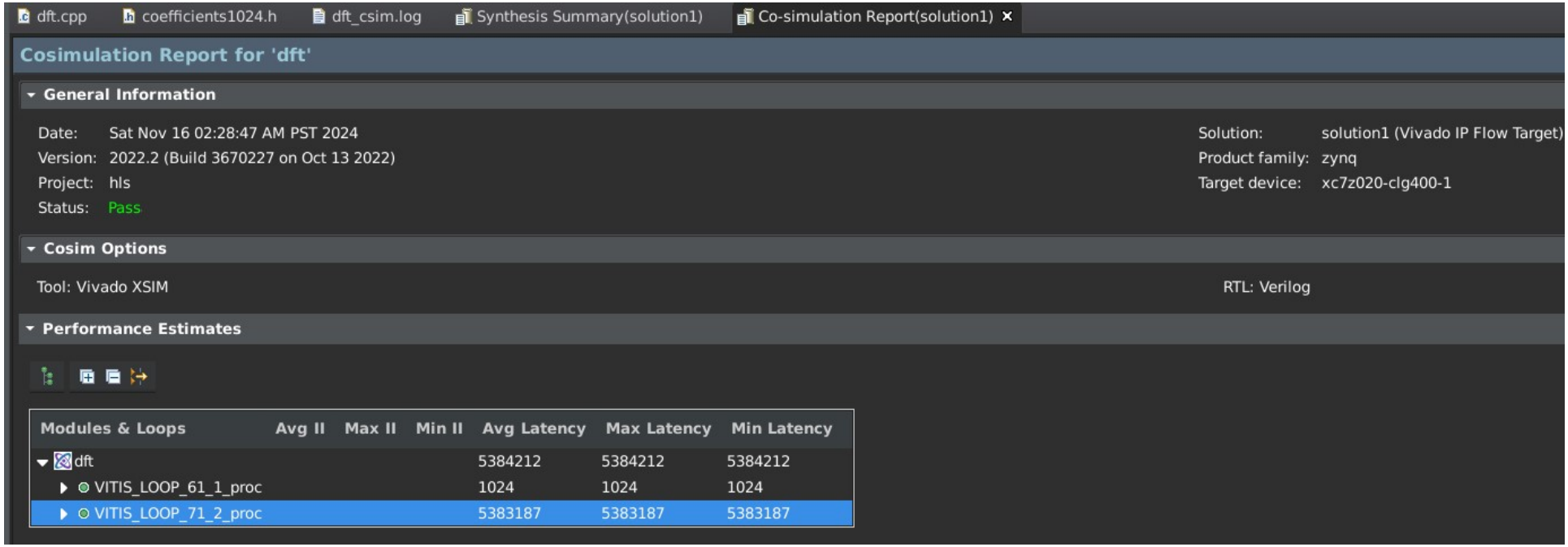
```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
make: 'csim.exe' is up to date.
-------------------------------------------
   RMSE(R)           RMSE(I)
0.001853297697380 0.001314980210736
-------------------------------------------
*******************************************
PASS: The output matches the golden output!
*******************************************
INFO [HLS SIM]: The maximum depth reached by any hls::stream() instance in the design is 1024
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************

```

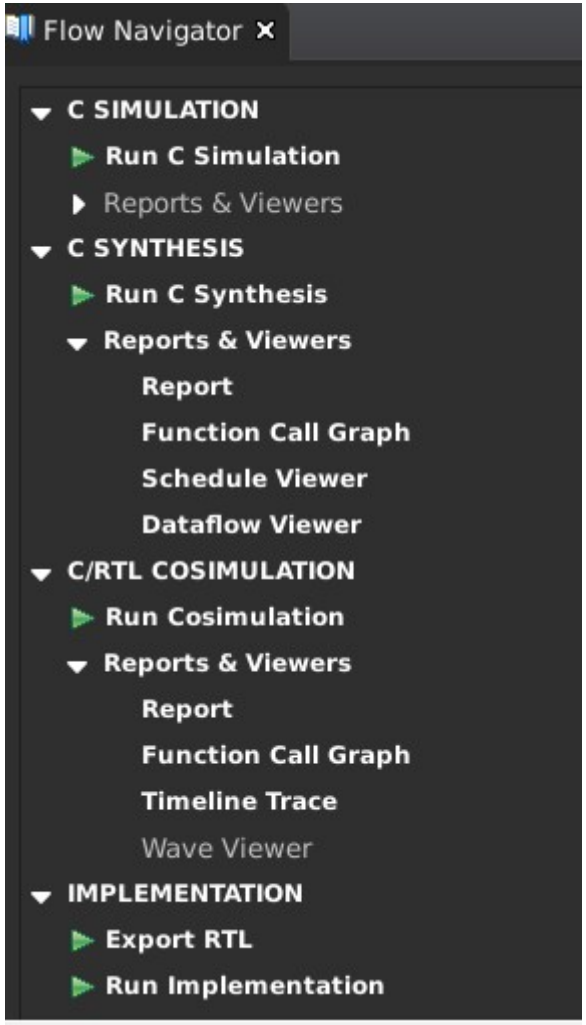
Target	Estimated	Uncertainty
10.00 ns	11.408 ns	2.70 ns

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
dft				-4.11	5386268	6.145E7	-	5385242	-	dataflow	48	34	7596	9289	0
VITIS_LOOP_61_1_proc				-	1026	1.026E4	-	1026	-	no	0	0	13	73	0
VITIS_LOOP_71_2_proc	II&Timing Violation			-4.11	5385241	6.144E7	-	5385241	-	no	40	34	7545	9144	0

Co-Simulation:

```
Warning: OPMODE Input Warning : The OPMODE 0110X0X with CARRYINSEL  000 to DSP48E1 instance is invalid.
Time: 9210875 ns  Iteration: 11  Process: /apatb_dft_top/AESL_inst_dft/VITIS_LOOP_71_2_proc_U0/dataflow_in_loop_VITIS_LOOP_71_2_U0/Loop3_U0/grp_Loop3_Pipeline_VITIS_LOOP_35_1_fu_88/fadd_32ns_32ns_32_5_full_dsp_1_U63/dft_fadd_32n
Warning: OPMODE Input Warning : The OPMODE 0110X0X with CARRYINSEL  000 to DSP48E1 instance is invalid.
Time: 9210925 ns  Iteration: 11  Process: /apatb_dft_top/AESL_inst_dft/VITIS_LOOP_71_2_proc_U0/dataflow_in_loop_VITIS_LOOP_71_2_U0/Loop3_U0/grp_Loop3_Pipeline_VITIS_LOOP_35_1_fu_88/fadd_32ns_32ns_32_5_full_dsp_1_U63/dft_fadd_32n
Warning: OPMODE Input Warning : The OPMODE 0110X0X with CARRYINSEL  000 to DSP48E1 instance is invalid.
Time: 9210975 ns  Iteration: 11  Process: /apatb_dft_top/AESL_inst_dft/VITIS_LOOP_71_2_proc_U0/dataflow_in_loop_VITIS_LOOP_71_2_U0/Loop3_U0/grp_Loop3_Pipeline_VITIS_LOOP_35_1_fu_88/fadd_32ns_32ns_32_5_full_dsp_1_U63/dft_fadd_32n
Warning: OPMODE Input Warning : The OPMODE 0110X0X with CARRYINSEL  000 to DSP48E1 instance is invalid.
Time: 9211025 ns  Iteration: 11  Process: /apatb_dft_top/AESL_inst_dft/VITIS_LOOP_71_2_proc_U0/dataflow_in_loop_VITIS_LOOP_71_2_U0/Loop3_U0/grp_Loop3_Pipeline_VITIS_LOOP_35_1_fu_88/fadd_32ns_32ns_32_5_full_dsp_1_U63/dft_fadd_32n
Warning: OPMODE Input Warning : The OPMODE 0110X0X with CARRYINSEL  000 to DSP48E1 instance is invalid.
Time: 9211075 ns  Iteration: 11  Process: /apatb_dft_top/AESL_inst_dft/VITIS_LOOP_71_2_proc_U0/dataflow_in_loop_VITIS_LOOP_71_2_U0/Loop3_U0/grp_Loop3_Pipeline_VITIS_LOOP_35_1_fu_88/fadd_32ns_32ns_32_5_full_dsp_1_U63/dft_fadd_32n
Warning: OPMODE Input Warning : The OPMODE 0110X0X with CARRYINSEL  000 to DSP48E1 instance is invalid.
Time: 9211135 ns  Iteration: 11  Process: /apatb_dft_top/AESL_inst_dft/VITIS_LOOP_71_2_proc_U0/dataflow_in_loop_VITIS_LOOP_71_2_U0/Loop3_U0/grp_Loop3_Pipeline_VITIS_LOOP_35_1_fu_88/fadd_32ns_32ns_32_5_full_dsp_1_U63/dft_fadd_32n
Warning: OPMODE Input Warning : The OPMODE 0110X0X with CARRYINSEL  000 to DSP48E1 instance is invalid.
Time: 9211185 ns  Iteration: 11  Process: /apatb_dft_top/AESL_inst_dft/VITIS_LOOP_71_2_proc_U0/dataflow_in_loop_VITIS_LOOP_71_2_U0/Loop3_U0/grp_Loop3_Pipeline_VITIS_LOOP_35_1_fu_88/fadd_32ns_32ns_32_5_full_dsp_1_U63/dft_fadd_32n
Warning: OPMODE Input Warning : The OPMODE 0110X0X with CARRYINSEL  000 to DSP48E1 instance is invalid.
Time: 9211235 ns  Iteration: 11  Process: /apatb_dft_top/AESL_inst_dft/VITIS_LOOP_71_2_proc_U0/dataflow_in_loop_VITIS_LOOP_71_2_U0/Loop3_U0/grp_Loop3_Pipeline_VITIS_LOOP_35_1_fu_88/fadd_32ns_32ns_32_5_full_dsp_1_U63/dft_fadd_32n
Warning: OPMODE Input Warning : The OPMODE 0110X0X with CARRYINSEL  000 to DSP48E1 instance is invalid.
Time: 9211285 ns  Iteration: 11  Process: /apatb_dft_top/AESL_inst_dft/VITIS_LOOP_71_2_proc_U0/dataflow_in_loop_VITIS_LOOP_71_2_U0/Loop3_U0/grp_Loop3_Pipeline_VITIS_LOOP_35_1_fu_88/fadd_32ns_32ns_32_5_full_dsp_1_U63/dft_fadd_32n
Warning: OPMODE Input Warning : The OPMODE 0110X0X with CARRYINSEL  000 to DSP48E1 instance is invalid.
Time: 9211335 ns  Iteration: 11  Process: /apatb_dft_top/AESL_inst_dft/VITIS_LOOP_71_2_proc_U0/dataflow_in_loop_VITIS_LOOP_71_2_U0/Loop3_U0/grp_Loop3_Pipeline_VITIS_LOOP_35_1_fu_88/fadd_32ns_32ns_32_5_full_dsp_1_U63/dft_fadd_32n
Warning: OPMODE Input Warning : The OPMODE 0110X0X with CARRYINSEL  000 to DSP48E1 instance is invalid.
Time: 9211385 ns  Iteration: 11  Process: /apatb_dft_top/AESL_inst_dft/VITIS_LOOP_71_2_proc_U0/dataflow_in_loop_VITIS_LOOP_71_2_U0/Loop3_U0/grp_Loop3_Pipeline_VITIS_LOOP_35_1_fu_88/fadd_32ns_32ns_32_5_full_dsp_1_U63/dft_fadd_32n
Warning: OPMODE Input Warning : The OPMODE 0110X0X with CARRYINSEL  000 to DSP48E1 instance is invalid.
Time: 9211435 ns  Iteration: 11  Process: /apatb_dft_top/AESL_inst_dft/VITIS_LOOP_71_2_proc_U0/dataflow_in_loop_VITIS_LOOP_71_2_U0/Loop3_U0/grp_Loop3_Pipeline_VITIS_LOOP_35_1_fu_88/fadd_32ns_32ns_32_5_full_dsp_1_U63/dft_fadd_32n
Warning: OPMODE Input Warning : The OPMODE 0110X0X with CARRYINSEL  000 to DSP48E1 instance is invalid.
Time: 9211485 ns  Iteration: 11  Process: /apatb_dft_top/AESL_inst_dft/VITIS_LOOP_71_2_proc_U0/dataflow_in_loop_VITIS_LOOP_71_2_U0/Loop3_U0/grp_Loop3_Pipeline_VITIS_LOOP_35_1_fu_88/fadd_32ns_32ns_32_5_full_dsp_1_U63/dft_fadd_32n
Warning: OPMODE Input Warning : The OPMODE 0110X0X with CARRYINSEL  000 to DSP48E1 instance is invalid.
Time: 9211545 ns  Iteration: 11  Process: /apatb_dft_top/AESL_inst_dft/VITIS_LOOP_71_2_proc_U0/dataflow_in_loop_VITIS_LOOP_71_2_U0/Loop3_U0/grp_Loop3_Pipeline_VITIS_LOOP_35_1_fu_88/fadd_32ns_32ns_32_5_full_dsp_1_U63/dft_fadd_32n
Warning: OPMODE Input Warning : The OPMODE 0110X0X with CARRYINSEL  000 to DSP48E1 instance is invalid.
Time: 9211595 ns  Iteration: 11  Process: /apatb_dft_top/AESL_inst_dft/VITIS_LOOP_71_2_proc_U0/dataflow_in_loop_VITIS_LOOP_71_2_U0/Loop3_U0/grp_Loop3_Pipeline_VITIS_LOOP_35_1_fu_88/fadd_32ns_32ns_32_5_full_dsp_1_U63/dft_fadd_32n
Warning: OPMODE Input Warning : The OPMODE 0110X0X with CARRYINSEL  000 to DSP48E1 instance is invalid.
Time: 9211645 ns  Iteration: 11  Process: /apatb_dft_top/AESL_inst_dft/VITIS_LOOP_71_2_proc_U0/dataflow_in_loop_VITIS_LOOP_71_2_U0/Loop3_U0/grp_Loop3_Pipeline_VITIS_LOOP_35_1_fu_88/fadd_32ns_32ns_32_5_full_dsp_1_U63/dft_fadd_32n
Warning: OPMODE Input Warning : The OPMODE 0110X0X with CARRYINSEL  000 to DSP48E1 instance is invalid.
Time: 9211695 ns  Iteration: 11  Process: /apatb_dft_top/AESL_inst_dft/VITIS_LOOP_71_2_proc_U0/dataflow_in_loop_VITIS_LOOP_71_2_U0/Loop3_U0/grp_Loop3_Pipeline_VITIS_LOOP_35_1_fu_88/fadd_32ns_32ns_32_5_full_dsp_1_U63/dft_fadd_32n
Warning: OPMODE Input Warning : The OPMODE 0110X0X with CARRYINSEL  000 to DSP48E1 instance is invalid.
Time: 9211745 ns  Iteration: 11  Process: /apatb_dft_top/AESL_inst_dft/VITIS_LOOP_71_2_proc_U0/dataflow_in_loop_VITIS_LOOP_71_2_U0/Loop3_U0/grp_Loop3_Pipeline_VITIS_LOOP_35_1_fu_88/fadd_32ns_32ns_32_5_full_dsp_1_U63/dft_fadd_32n
Warning: OPMODE Input Warning : The OPMODE 0110X0X with CARRYINSEL  000 to DSP48E1 instance is invalid.
Time: 9211795 ns  Iteration: 11  Process: /apatb_dft_top/AESL_inst_dft/VITIS_LOOP_71_2_proc_U0/dataflow_in_loop_VITIS_LOOP_71_2_U0/Loop3_U0/grp_Loop3_Pipeline_VITIS_LOOP_35_1_fu_88/fadd_32ns_32ns_32_5_full_dsp_1_U63/dft_fadd_32n
Warning: OPMODE Input Warning : The OPMODE 0110X0X with CARRYINSEL  000 to DSP48E1 instance is invalid.
Time: 9211845 ns  Iteration: 11  Process: /apatb_dft_top/AESL_inst_dft/VITIS_LOOP_71_2_proc_U0/dataflow_in_loop_VITIS_LOOP_71_2_U0/Loop3_U0/grp_Loop3_Pipeline_VITIS_LOOP_35_1_fu_88/fadd_32ns_32ns_32_5_full_dsp_1_U63/dft_fadd_32n
Warning: OPMODE Input Warning : The OPMODE 0110X0X with CARRYINSEL  000 to DSP48E1 instance is invalid.
```


dft.cpp
coefficients1024.h
dft_csim.log
Synthesis Summary(solution1)
Co-simulation Report(solution1)
Cosimulation Report for 'dft'
General Information
Date: Sat Nov 16 02:28:47 AM PST 2024
Version: 2022.2 (Build 3670227 on Oct 13 2022)
Project: hls
Status: Pass
Solution: solution1 (Vivado IP Flow Target)
Product family: zynq
Target device: xc7z020-clg400-1
Cosim Options
Tool: Vivado XSIM
RTL: Verilog
Performance Estimates
Modules & Loops
Avg II
Max II
Min II
Avg Latency
Max Latency
Min Latency
dft 5384212 5384212 5384212
VITIS_LOOP_61_1_proc 1024 1024 1024
VITIS_LOOP_71_2_proc 5383187 5383187 5383187

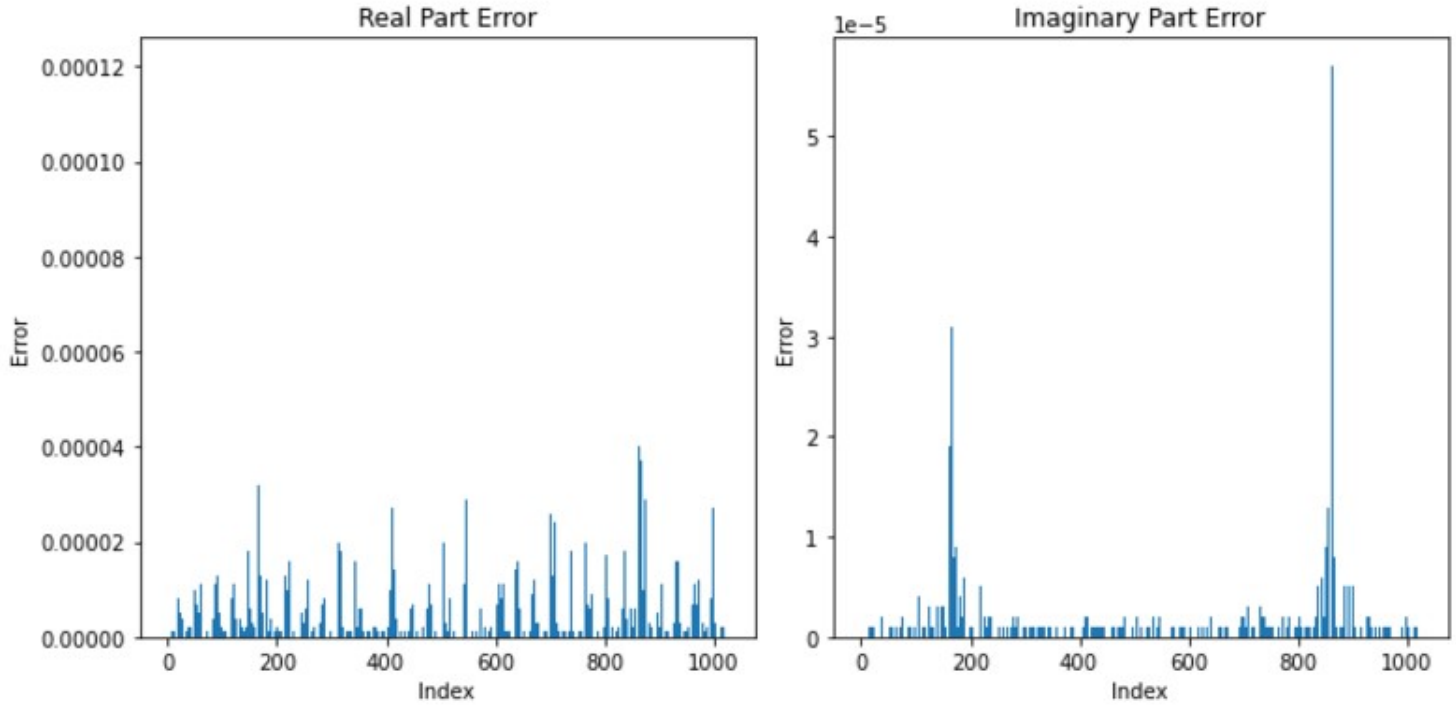

Flow Navigator
C SIMULATION
Run C Simulation
Reports & Viewers
C SYNTHESIS
Run C Synthesis
Reports & Viewers
Report
Function Call Graph
Schedule Viewer
Dataflow Viewer
C/RTL COSIMULATION
Run Cosimulation
Reports & Viewers
Report
Function Call Graph
Timeline Trace
Wave Viewer
IMPLEMENTATION
Export RTL
Run Implementation


Network 0
/processing_system7_0
/processing_system7_0/Data (32 address bits : 0x40000000 [1G])
/dft_0/s_axi_control
s_axi_control
Reg
0x4000_0000
64K
0x4000_FFFF

Displaying Error and Output

In [9]:

```
plt.figure(figsize=(10, 5))
plt.subplot(1,2,1)
plt.bar(ind,real_error)
plt.title("Real Part Error")
plt.xlabel("Index")
plt.ylabel("Error")
#plt.xticks(ind)
plt.tight_layout()

plt.subplot(1,2,2)
plt.bar(ind,imag_error)
plt.title("Imaginary Part Error")
plt.xlabel("Index")
plt.ylabel("Error")
#plt.xticks(ind)
plt.tight_layout()
```



In [10]:

```
freq=np.fft.fftfreq(1024)
```