ACCELERATED CHIP DESIGN CYCLE WORKSHOP: HIGH-LEVEL SYNTHESIS

This workshop is intended to provide the participants with the necessary skills to create high-level synthesis IPs using the Vitis HLS tool flow. Various techniques and directives inherent to Vitis HLS to improve the design performance are also introduced in detail

The labs have been developed and tested on Windows 10 professional and Ubuntu 20.04 machines and using Vivado HLS 2018.2 and Vitis HLS 2022.1 version tools. Linux machine to run the Vitis HLS tool is more recommended though.

Labs Overview:

Lab1: Vitis HLS Design Flow

This lab provides a basic introduction to high-level synthesis using the Vitis HLS tool flow. You will use Vitis HLS in GUI mode to create a project. You will simulate, synthesize, and implement the provided design.

Lab2: Improving Performance

This lab introduces various techniques and directives which can be used in Vitis HLS to improve design performance. The design under consideration accepts an image in a (custom) RGB format, converts it to the Y’UV color space, applies a filter to the Y’UV image and converts it back to RGB.

Lab3: Improving Area and Resource Utilization

This lab introduces various techniques and directives which can be used in Vitis HLS to improve design performance as well as area and resource utilization. The design under consideration performs discrete cosine transformation (DCT) on an 8x8 block of data.

Lab3: Improving Area and Resource Utilization

Introduction:

This lab introduces various techniques and directives which can be used in Vitis HLS to improve design performance as well as area and resource utilization. The design under consideration performs discrete cosine transformation (DCT) on an 8x8 block of data.

This design implements a discrete cosine transformation and it is provided as C source. The function leverages 2D DCT algorithm by first processing each row of the input array via a 1D DCT, then processing the columns of the resulting array through the same 1D DCT. It calls the **read_data**, **dct_2d** and **write_data** functions.

Objectives:

After completing this lab, you will be able to:

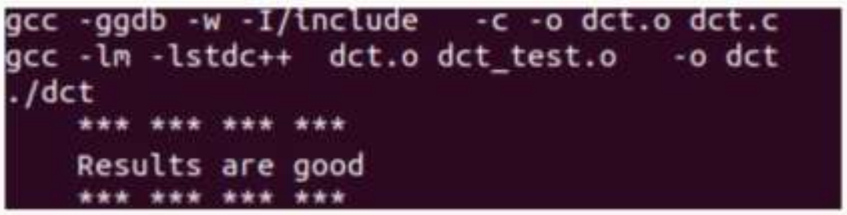
- Add directives in your design
- Improve performance using PIPELINE directive
- Distinguish between DATAFLOW directive and Configuration Command functionality
- Apply memory partitions techniques to improve resource utilization

Steps:

STEP 1: Validate the Design from Command Line

Validate your design in the terminal. (ONLY VALID FOR LINUX-BASED ENVIRONMENT, WINDOWS USERS MAY DIRECTLY GO TO STEP 2 and perform C simulation using the GUI as was done in earlier labs. You shall see the below message printed on the console.)

1. Open a terminal.
2. Change directory to **{labs}/lab3**.
   A self-checking program (dct_test.c) is provided**.** Using that we can validate the design. A Makefile is also provided. Using the Makefile, the necessary source files can be compiled and the compiled program can be executed.
3. Invoke Vitis HLS and in the terminal type **make** to compile and execute the program.



```
gcc -ggdb -w -I/include    -c -o dct.o dct.c
gcc -lm -lstdc++  dct.o dct_test.o    -o dct
./dct
    *** *** *** ***
    Results are good
    *** *** *** ***
```

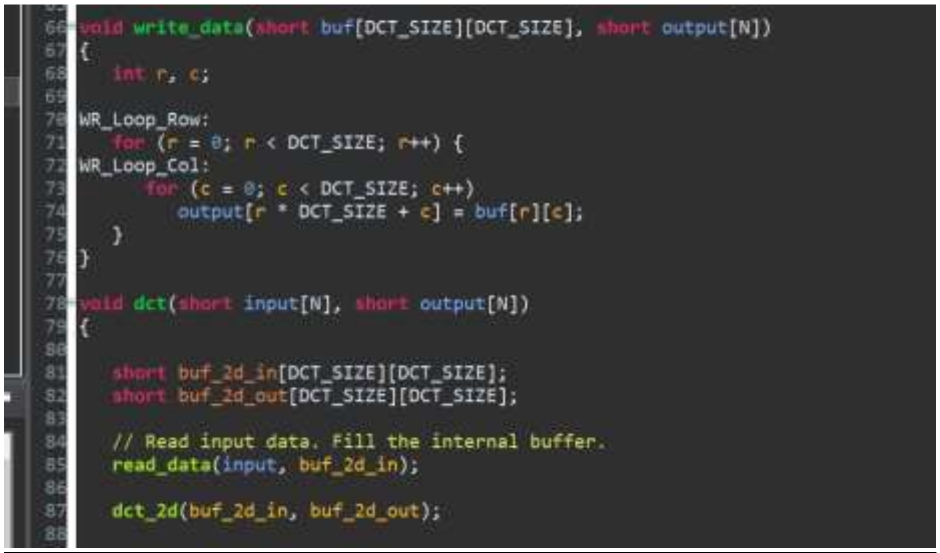
Validating the design

   Note that the source files (dct.c and dct_test.c) are compiled, then dct executable program was created, and then it was executed. The program tests the design and outputs **Results are good** message.
4. Close the command prompt window by typing **exit**.

STEP 2: Create a New Project

Create a new project in Vivado HLS GUI targeting xc7z020clg400-1.

1. Type **vitis_hls (Linux**) in the terminal or double click on **“Vitis HLS 2022.1”** icon in windows.
2. In the Vitis HLS GUI, click on **Create Project**. The New **Vitis HLS Project** wizard opens.
3. Click **Browse…** button of the Location field and browse to **{labs}\lab3** and then click **OK.**
4. For Project Name, type **dct** and click **Next**.
5. In the *Add/Remove Files* for the source files, type **dct** as the function name (the provided source file contains the function, to be synthesized, called dct).
6. Click the **Add Files…** button, select **dct.c** file from the **{labs}\lab3** folder, and then click **Open**.
7. Click **Next**.
8. In the *Add/Remove Files* for the testbench, click the **Add Files…** button, select **dct_test.c***,* **in.dat***,* **out.golden.dat** files from the **{lab}\lab3** folder and click **Open.**
9. Click **Next.**
10. In the *Solution Configuration* page, leave Solution Name field as **solution1** and set the clock period as **10**. Leave *Uncertainty* field blank.
11. Click on Part’s Browse button, and select the following filters, using the *Parts Specify* option, to select **xc7z020clg400-1.**
12. Click **Finish.**
13. Double-click on the **dct.c** under the *source* folder to open its content in the information pane.

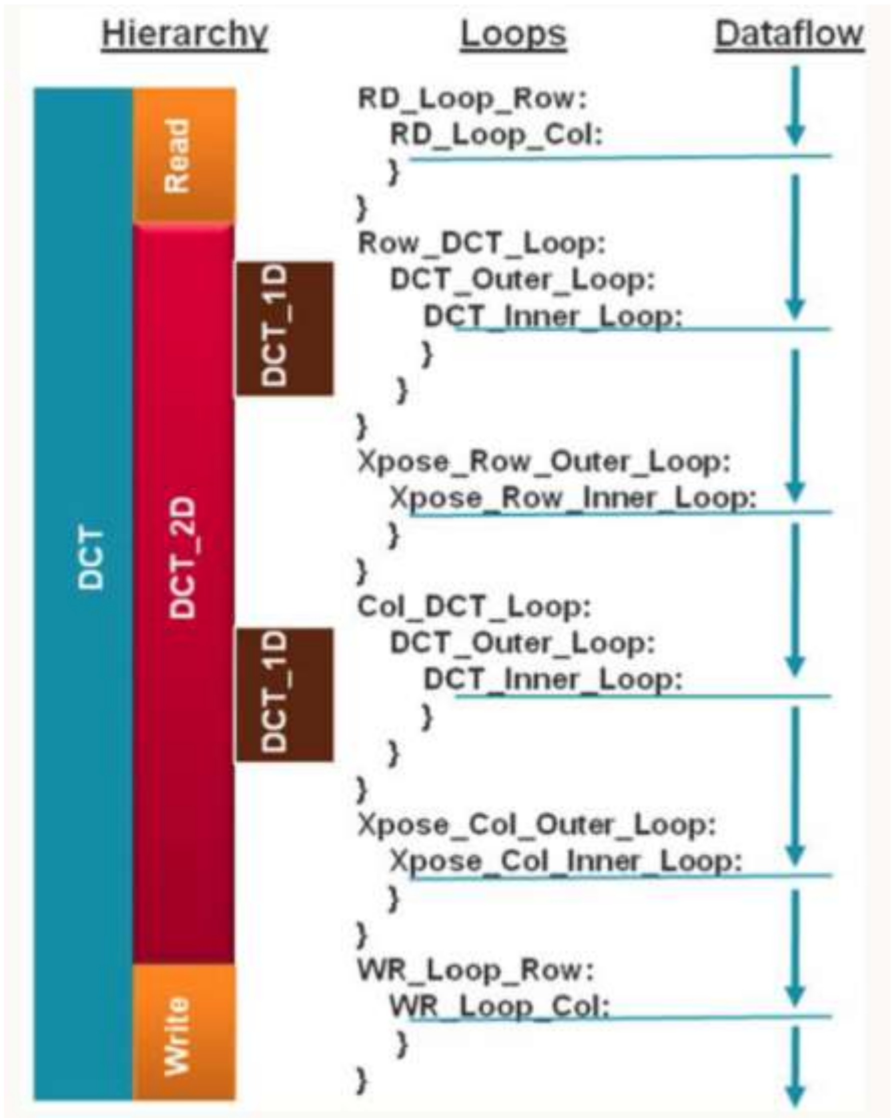


```
66 void write_data(short buf[DCT_SIZE][DCT_SIZE], short output[N])
67 {
68     int r, c;
69
70 WR_Loop_Row:
71     for (r = 0; r < DCT_SIZE; r++) {
72 WR_Loop_Col:
73         for (c = 0; c < DCT_SIZE; c++)
74             output[r * DCT_SIZE + c] = buf[r][c];
75     }
76 }
77
78 void dct(short input[N], short output[N])
79 {
80
81     short buf_2d_in[DCT_SIZE][DCT_SIZE];
82     short buf_2d_out[DCT_SIZE][DCT_SIZE];
83
84     // Read input data. Fill the internal buffer.
85     read_data(input, buf_2d_in);
86
87     dct_2d(buf_2d_in, buf_2d_out);
88
```

The top-level function **dct**, is defined at line **78**. It implements 2D DCT algorithm by first processing each row of the input array via a 1D DCT then processing the columns of the resulting array through the same 1D DCT. It calls **read_data**, **dct_2d**, and **write_data** functions.

The **read_data** function is defined at line **54** and consists of two loops – **RD_Loop_Row** and **RD_Loop_Col**. The **write_data** function is defined at line **66** and consists of two loops to perform writing the result. The **dct_2d** function, defined at line **23**, calls **dct_1d** function and performs transpose.

Finally, **dct_1d** function, defined at line **4**, uses **dct_coeff_table** and performs the required function by implementing a basic iterative form of the 1D Type-II DCT algorithm. Following figure shows the function hierarchy on the left-hand side, the loops in the order they are executes and the flow of data on the right-hand side.



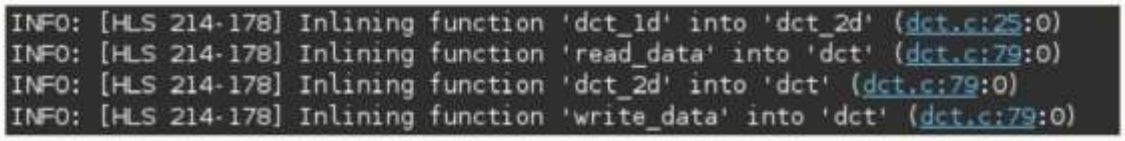
Design hierarchy and dataflow

STEP 3: Synthesize the Design

Synthesize the design with the defaults. View the synthesis results and answer the question listed in the detailed section of this step.

1. Select **Solution > Run C Synthesis > Active Solution** to start the synthesis process.
2. When synthesis is completed, several report files will become accessible and the *Synthesis Results* will be displayed in the information pane.

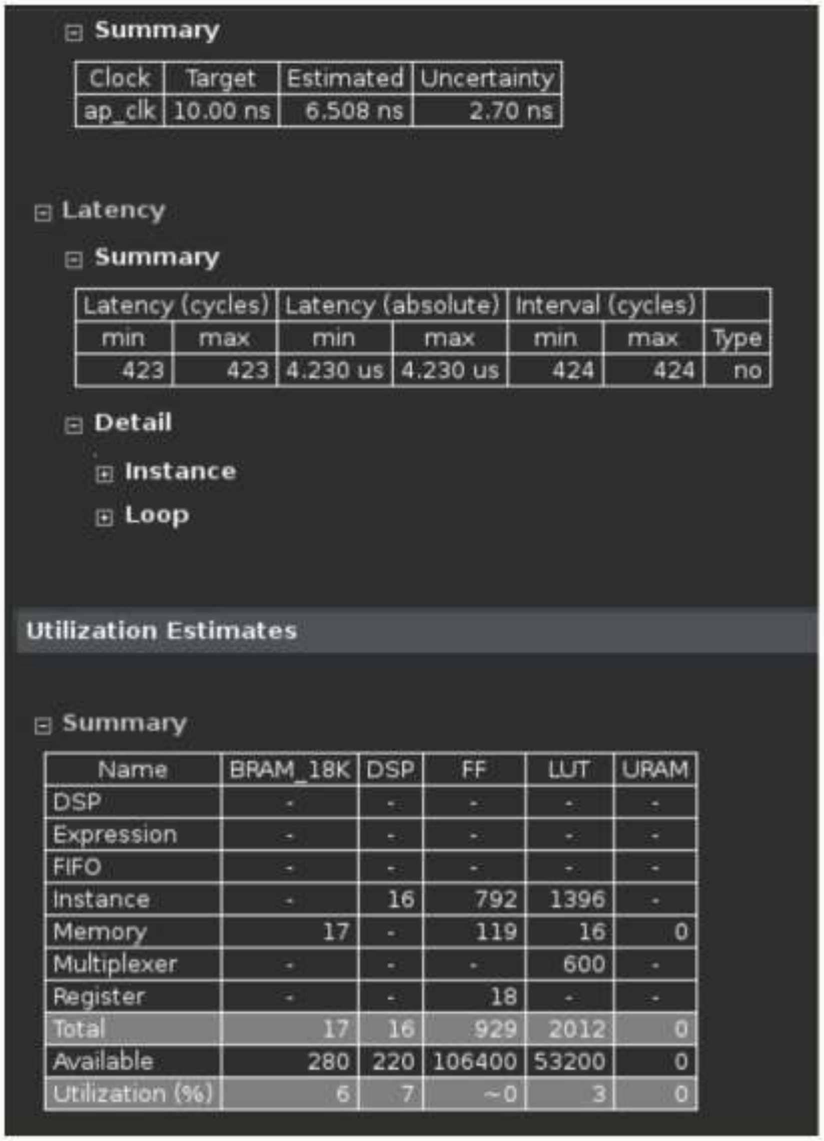
The **dct_1d, dct_2d, read_data** and **write_data** functions are inlined. Verify this by scrolling up into the Vitis HLS Console view.



```
INFO: [HLS 214-178] Inlining function 'dct_1d' into 'dct_2d' (dct.c:25:0)
INFO: [HLS 214-178] Inlining function 'read_data' into 'dct' (dct.c:79:0)
INFO: [HLS 214-178] Inlining function 'dct_2d' into 'dct' (dct.c:79:0)
INFO: [HLS 214-178] Inlining function 'write_data' into 'dct' (dct.c:79:0)
```

Inlining of dct_1d, dct_2d, read_data and write_data functions

The Synthesis Report shows the performance and resource estimates as well as estimated latency in the design.



Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	6.508 ns	2.70 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
423	423	4.230 us	4.230 us	424	424	no

Detail

Instance

Loop

Utilization Estimates

Summary

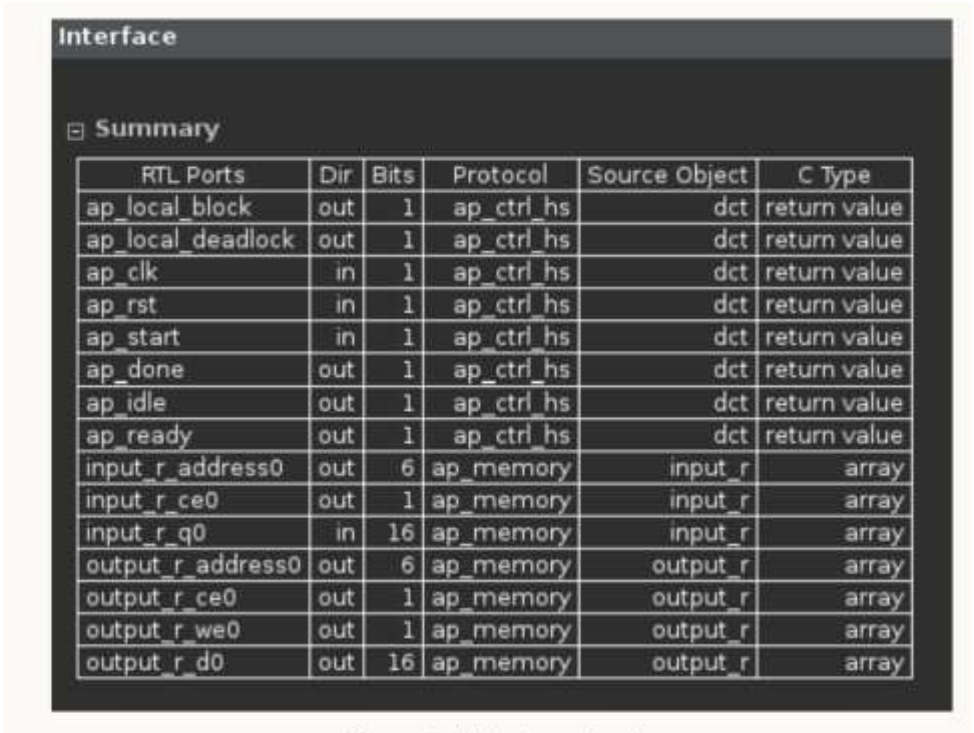
Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	-	-	-
FIFO	-	-	-	-	-
Instance	-	16	792	1396	-
Memory	17	-	119	16	0
Multiplexer	-	-	-	600	-
Register	-	-	18	-	-
Total	17	16	929	2012	0
Available	280	220	106400	53200	0
Utilization (%)	6	7	~0	3	0

Synthesis report

3. Using scroll bar on the right, scroll down into the report and answer the following question.
Question 1
Estimated clock period:
Worst case latency:

Number of DSP48E used:
Number of BRAMs used:
Number of FFs used:
Number of LUTs used:

4. The report also shows the top-level interface signals generated by the tools.



Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_local_block	out	1	ap_ctrl_hs	dct	return value
ap_local_deadlock	out	1	ap_ctrl_hs	dct	return value
ap_clk	in	1	ap_ctrl_hs	dct	return value
ap_rst	in	1	ap_ctrl_hs	dct	return value
ap_start	in	1	ap_ctrl_hs	dct	return value
ap_done	out	1	ap_ctrl_hs	dct	return value
ap_idle	out	1	ap_ctrl_hs	dct	return value
ap_ready	out	1	ap_ctrl_hs	dct	return value
input_r_address0	out	6	ap_memory	input_r	array
input_r_ce0	out	1	ap_memory	input_r	array
input_r_q0	in	16	ap_memory	input_r	array
output_r_address0	out	6	ap_memory	output_r	array
output_r_ce0	out	1	ap_memory	output_r	array
output_r_we0	out	1	ap_memory	output_r	array
output_r_d0	out	16	ap_memory	output_r	array

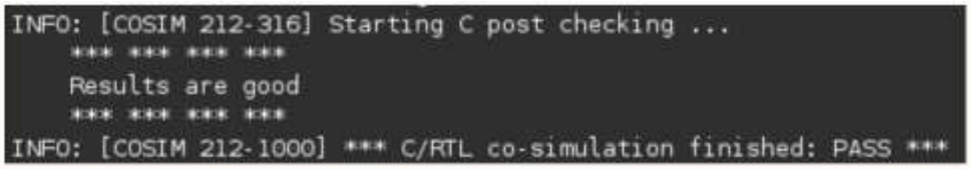
Generated interface signals

You can see **ap_clk, ap_rst** are automatically added. The **ap_start, ap_done, ap_idle**, and **ap_ready** are top-level signals used as handshaking signals to indicate when the design is able to accept next computation command (**ap_idle**), when the next computation is started (**ap_start**), and when the computation is completed (**ap_done**). The top-level function has input and output arrays, hence an **ap_memory** interface is generated for each of them.

STEP 4: Run Co-Simulation

Run the Co-simulation, selecting Verilog. Verify that the simulation passes.

1. Select **Solution > Run C/RTL Co-simulation** to open the dialog box so the desired simulations can be run. A C/RTL Co-simulation Dialog box will open.
2. Select the **VHDL** option, and click **OK** to run the Verilog simulation using XSIM simulator.
   The RTL Co-simulation will run, generating and compiling several files, and then simulating the design. In the console window you can see the progress and also a message that the test is passed.

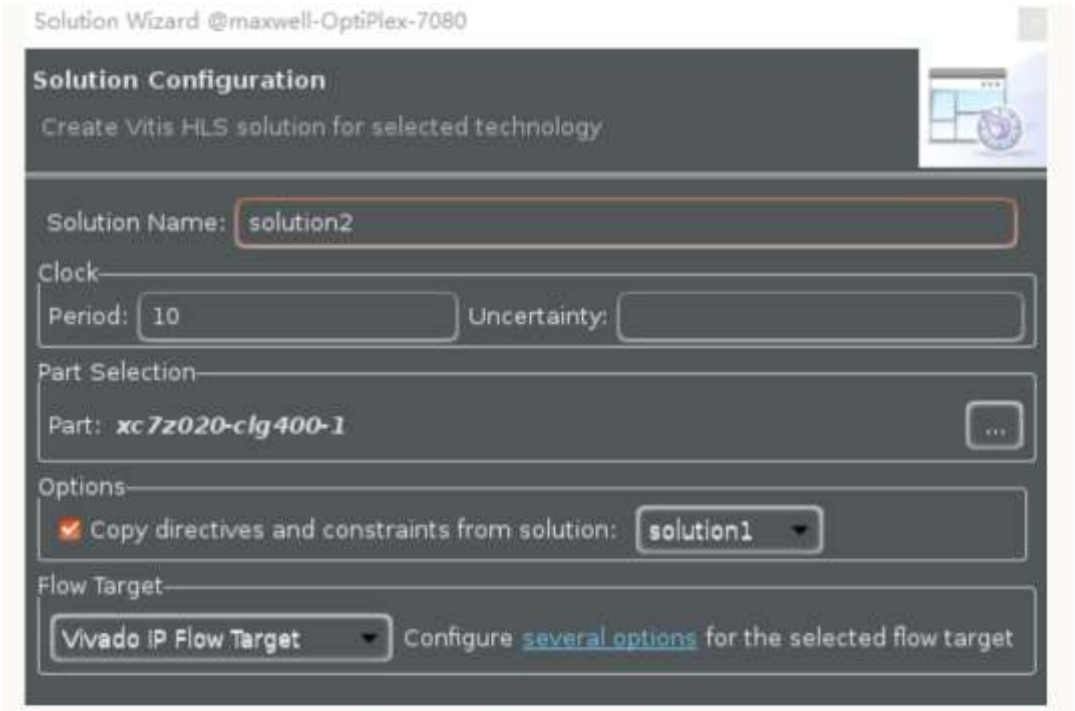


```
INFO: [COSIM 212-316] Starting C post checking ...
    *** *** *** ***
    Results are good
    *** *** *** ***
INFO: [COSIM 212-1000] *** C/RTL co-simulation finished: PASS ***
```

RTL Co-Simulation results

STEP 5: Remove the pipeline optimization done by Vitis HLS automatically by adding pipeline off pragma.

1. Select **Project > New Solution.**
2. A *Solution Configuration* dialog box will appear. Note that the check boxes of *Copy directives and constraints from solution* are checked with *solution1* selected. Click the **Finish** button to create a new solution with the default settings.



Solution Wizard @maxwell-OptiPlex-7080

Solution Configuration

Create Vitis HLS solution for selected technology

Solution Name: solution2

Clock

Period: 10 Uncertainty:

Part Selection

Part: xc7z020-clg400-1

Options

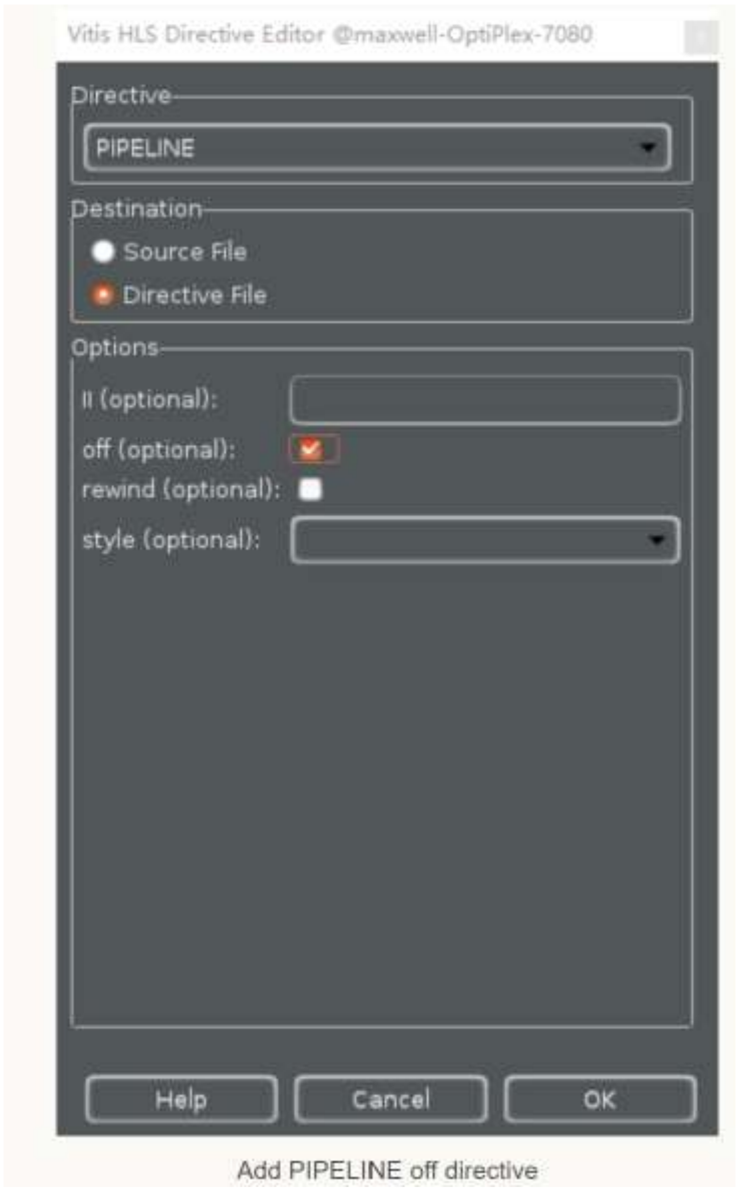
[✓] Copy directives and constraints from solution: solution1

Flow Target

Vivado IP Flow Target Configure several options for the selected flow target

Creating a new Solution after copying the existing solution

3. Make sure that the **dct.c** source is opened in the information pane and click on the **Directive** tab.
4. Select **DCT_Inner_Loop** in the directives pane, right-click on it and select **Insert Directive...**
5. Click on the drop-down button of the *Directive* field. A pop-up menu shows up listing various directives. Select **PIPELINE** directive.
6. In the *Vitis HLS Directive Editor* dialog box, click on the **off** option to turn off the automatic pipelining. Make sure that the *Directive File* is selected as destination. Click **OK**.


Vitis HLS Directive Editor @maxwell-OptiPlex-7080

Directive: PIPELINE

Destination: ( ) Source File [✓ Directive File]

Options:

II (optional):

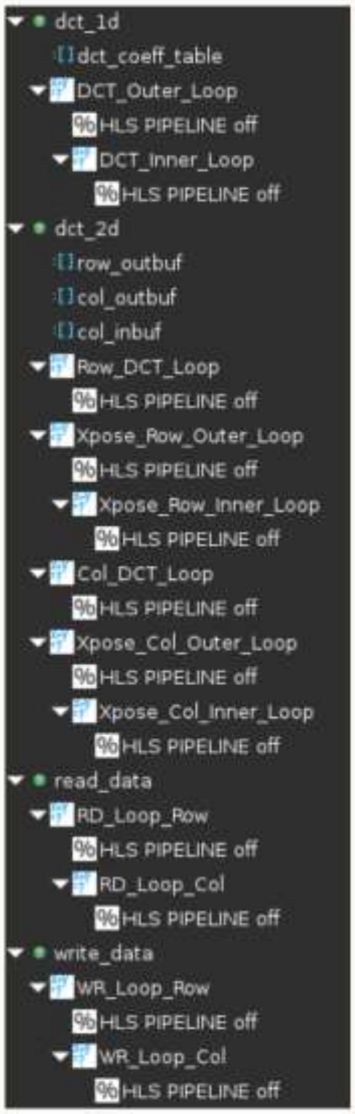
off (optional): ✓

rewind (optional):

style (optional):

Add PIPELINE off directive

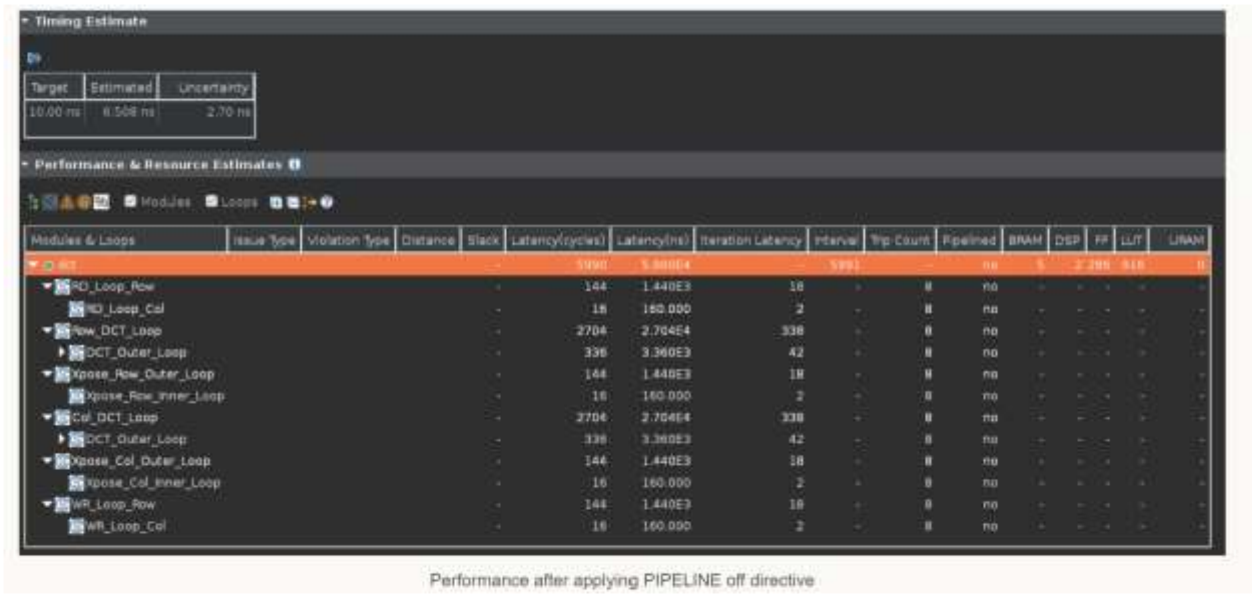
7. Similarly, apply the **PIPELINE off** directive to **DCT_Outer_Loop, Row_DCT_Loop, Xpose_Row_Outer_Loop, Xpose_Row_Inner_Loop, Col_DCT_Loop, Xpose_Col_Outer_Loop, Xpose_Col_Inner_Loop, RD_Loop_Row, RD_Loop_Col, WR_Loop_Row** and **WR_Loop_Col** objects. At this point, the *Directive* tab should look like as follows.



```
▼ ● dct_1d
    [] dct_coeff_table
  ▼ DCT_Outer_Loop
      % HLS PIPELINE off
    ▼ DCT_Inner_Loop
        % HLS PIPELINE off
▼ ● dct_2d
    [] row_outbuf
    [] col_outbuf
    [] col_inbuf
  ▼ Row_DCT_Loop
      % HLS PIPELINE off
  ▼ Xpose_Row_Outer_Loop
      % HLS PIPELINE off
    ▼ Xpose_Row_Inner_Loop
        % HLS PIPELINE off
  ▼ Col_DCT_Loop
      % HLS PIPELINE off
  ▼ Xpose_Col_Outer_Loop
      % HLS PIPELINE off
    ▼ Xpose_Col_Inner_Loop
        % HLS PIPELINE off
▼ ● read_data
  ▼ RD_Loop_Row
      % HLS PIPELINE off
    ▼ RD_Loop_Col
        % HLS PIPELINE off
▼ ● write_data
  ▼ WR_Loop_Row
      % HLS PIPELINE off
    ▼ WR_Loop_Col
        % HLS PIPELINE off
```

PIPELINE off directive applied

8. Click on **Synthesis** button.
9. When the synthesis is completed, report shows the performance and area without the automatic optimization of Vitis HLS.



Target	Estimated	Uncertainty
10.00 ns	6.508 ns	2.70 ns

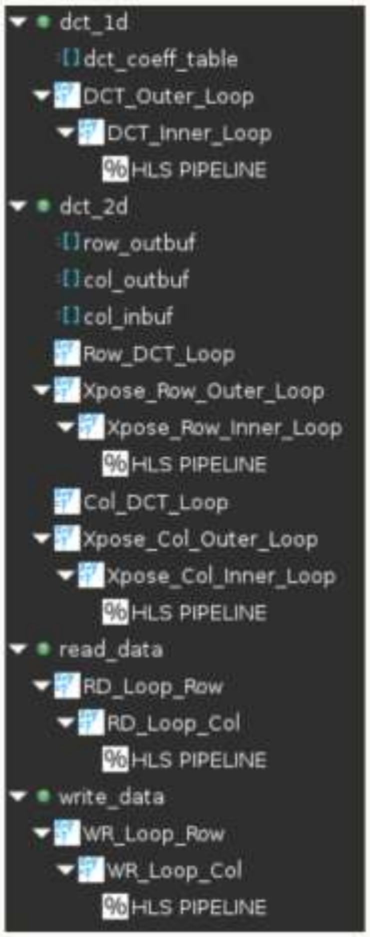
Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
dct				-	5990	5.990E4	-	5991	-	no	5	2	[illegible]	[illegible]	0
RD_Loop_Row				-	144	1.440E3	18	-	8	no	-	-	-	-	-
RD_Loop_Col				-	16	160.000	2	-	8	no	-	-	-	-	-
Row_DCT_Loop				-	2704	2.704E4	338	-	8	no	-	-	-	-	-
DCT_Outer_Loop				-	336	3.360E3	42	-	8	no	-	-	-	-	-
Xpose_Row_Outer_Loop				-	144	1.440E3	18	-	8	no	-	-	-	-	-
Xpose_Row_Inner_Loop				-	16	160.000	2	-	8	no	-	-	-	-	-
Col_DCT_Loop				-	2704	2.704E4	338	-	8	no	-	-	-	-	-
DCT_Outer_Loop				-	336	3.360E3	42	-	8	no	-	-	-	-	-
Xpose_Col_Outer_Loop				-	144	1.440E3	18	-	8	no	-	-	-	-	-
Xpose_Col_Inner_Loop				-	16	160.000	2	-	8	no	-	-	-	-	-
WR_Loop_Row				-	144	1.440E3	18	-	8	no	-	-	-	-	-
WR_Loop_Col				-	16	160.000	2	-	8	no	-	-	-	-	-

Performance after applying PIPELINE off directive

STEP 6: Apply PIPELINE Directive

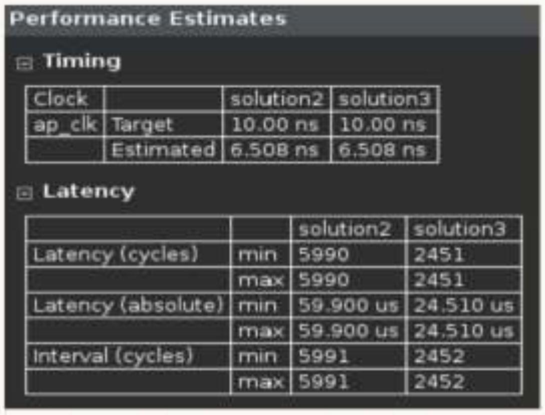
Create a new solution by copying the previous solution settings. Apply the PIPELINE directive to DCT_Inner_Loop, Xpose_Row_Inner_Loop, Xpose_Col_Inner_Loop, RD_Loop_Col, and WR_Loop_Col. Generate the solution and analyze the output.

1. Select **Project > New Solution.**
2. A *Solution Configuration* dialog box will appear. Click the **Finish** button (with *copy from* Solution2 selected).
3. Make sure that the **dct.c** source is opened in the information pane and click on the **Directive** tab.
4. Select **DCT_Inner_Loop** of the **dct_1d** function in the Directive pane, right-click on it and select **Modify Directive**.
5. In the Vitis HLS Directive Editor dialog box, click the **off** option to turn on the pipelining. Make sure the *Directive File* is selected as destination. Click **OK**.
6. Leave II (Initiation Interval) blank as Vitis HLS will try for an II=1, one new input every clock cycle.
7. Click **OK**.
8. Similarly, apply the **PIPELINE** directive to **Xpose_Row_Inner_Loop** and **Xpose_Col_Inner_Loop** of the dct_2d function, and **RD_Loop_Col** of the read_data function, and **WR_Loop_Col** of the write_data function. But remove the PIPELINE directive of **DCT_Outer_Loop, Row_DCT_Loop, Xpose_Row_Outer_Loop, Col_DCT_Loop, Xpose_Col_Outer_Loop, RD_Loop_Row** and **WR_Loop_Row**. At this point, the Directive tab should look like as follows:


- dct_1d
  - dct_coeff_table
  - DCT_Outer_Loop
    - DCT_Inner_Loop
      - HLS PIPELINE
- dct_2d
  - row_outbuf
  - col_outbuf
  - col_inbuf
  - Row_DCT_Loop
  - Xpose_Row_Outer_Loop
    - Xpose_Row_Inner_Loop
      - HLS PIPELINE
  - Col_DCT_Loop
  - Xpose_Col_Outer_Loop
    - Xpose_Col_Inner_Loop
      - HLS PIPELINE
- read_data
  - RD_Loop_Row
    - RD_Loop_Col
      - HLS PIPELINE
- write_data
  - WR_Loop_Row
    - WR_Loop_Col
      - HLS PIPELINE

PIPELINE directive applied

9. Click on the **Synthesis** button.
10. When the synthesis is completed, select **Project > Compare Reports…** to compare the two solutions.
11. Select *Solution2* and *Solution3* from the *Available Reports*, click on the **Add>>** button, and then click **OK**.
12. Observe the reduction in the latency from *5990* to ~*2450* clock cycles.


Performance Estimates

⊟ **Timing**

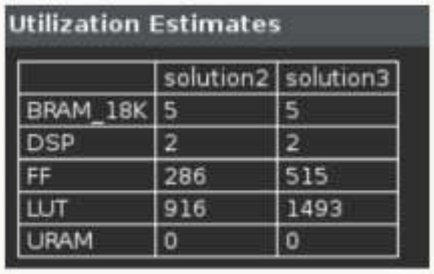
Clock		solution2	solution3
ap_clk	Target	10.00 ns	10.00 ns
	Estimated	6.508 ns	6.508 ns

⊟ **Latency**

		solution2	solution3
Latency (cycles)	min	5990	2451
	max	5990	2451
Latency (absolute)	min	59.900 us	24.510 us
	max	59.900 us	24.510 us
Interval (cycles)	min	5991	2452
	max	5991	2452

Performance comparison after pipelining

13. Scroll down in the comparison report to view the resources utilization. Observe that the FFs and/or LUTs utilization increased whereas BRAM and DSP48E remained same.



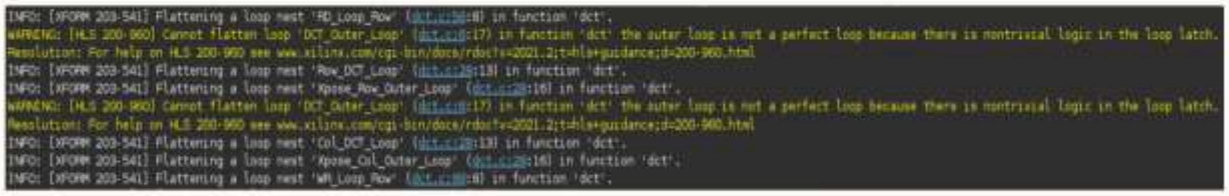
Utilization Estimates

	solution2	solution3
BRAM_18K	5	5
DSP	2	2
FF	286	515
LUT	916	1493
URAM	0	0

Resources utilization after pipelining

Open the Schedule Viewer and determine where most of the clock cycles are spend, i.e. where the large latencies are.

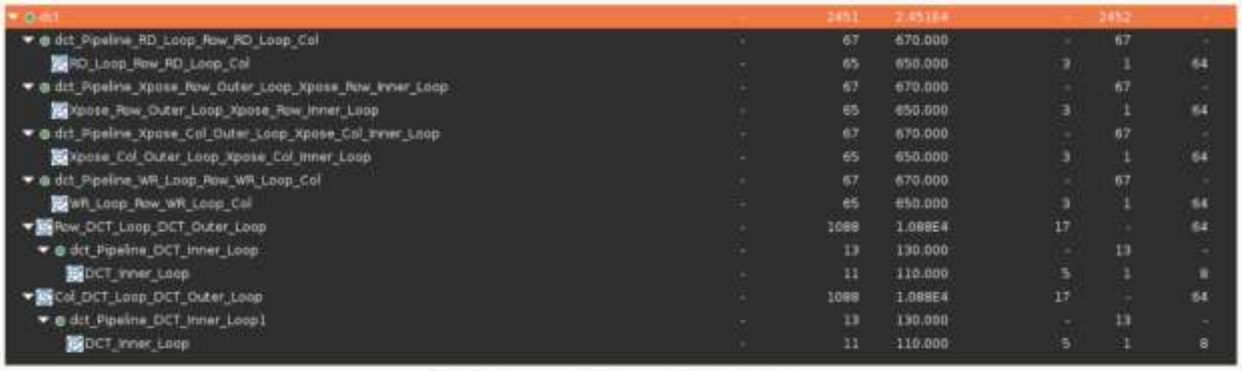
1. Click on the *Solution > Open Schedule Viewer*.
2. Select the **dct** entry and observe the **RD_Loop_Row_RD_Loop_Col** and **WR_Loop_Row_WR_Loop_Col** entries in the *Performance & Resource Estimates*. These are two nested loops flattened and given the new names formed by appending inner loop name to the out loop name. You can also verify this by looking in the *Console* view message.



```
INFO: [XFORM 203-541] Flattening a loop nest 'RD_Loop_Row' ([illegible]:8) in function 'dct'.
WARNING: [HLS 200-960] Cannot flatten loop 'DCT_Outer_Loop' ([illegible]:17) in function 'dct' the outer loop is not a perfect loop because there is nontrivial logic in the loop latch.
Resolution: For help on HLS 200-960 see www.xilinx.com/cgi-bin/docs/rdoc?v=2021.2;t=hls+guidance;d=200-960.html
INFO: [XFORM 203-541] Flattening a loop nest 'Row_DCT_Loop' ([illegible]:13) in function 'dct'.
INFO: [XFORM 203-541] Flattening a loop nest 'Xpose_Row_Outer_Loop' ([illegible]:16) in function 'dct'.
WARNING: [HLS 200-960] Cannot flatten loop 'DCT_Outer_Loop' ([illegible]:17) in function 'dct' the outer loop is not a perfect loop because there is nontrivial logic in the loop latch.
Resolution: For help on HLS 200-960 see www.xilinx.com/cgi-bin/docs/rdoc?v=2021.2;t=hls+guidance;d=200-960.html
INFO: [XFORM 203-541] Flattening a loop nest 'Col_DCT_Loop' ([illegible]:13) in function 'dct'.
INFO: [XFORM 203-541] Flattening a loop nest 'Xpose_Col_Outer_Loop' ([illegible]:16) in function 'dct'.
INFO: [XFORM 203-541] Flattening a loop nest 'WR_Loop_Row' ([illegible]:8) in function 'dct'.
```

The console view content indicating loops flattening

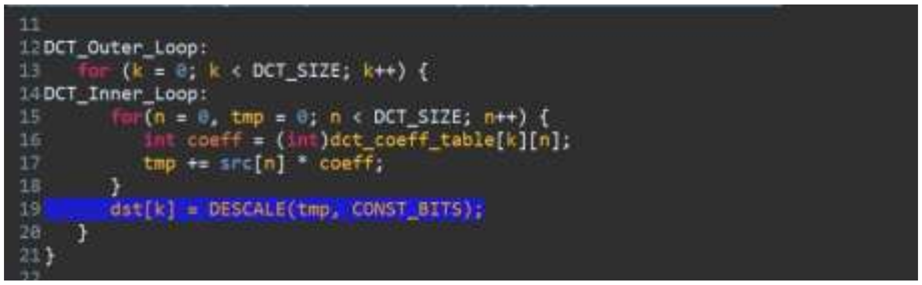
3. In the *Performance & Resource Estimates* pane, expand all items. Notice that the most of the latency is in **Row_DCT_Loop_DCT_Outer_Loop** and **Col_DCT_Loop_DCT_Outer_Loop** function.



dct	-	2451	2.451E4	-	2452	-
dct_Pipeline_RD_Loop_Row_RD_Loop_Col	-	67	670.000	-	67	-
RD_Loop_Row_RD_Loop_Col	-	65	650.000	3	1	64
dct_Pipeline_Xpose_Row_Outer_Loop_Xpose_Row_Inner_Loop	-	67	670.000	-	67	-
Xpose_Row_Outer_Loop_Xpose_Row_Inner_Loop	-	65	650.000	3	1	64
dct_Pipeline_Xpose_Col_Outer_Loop_Xpose_Col_Inner_Loop	-	67	670.000	-	67	-
Xpose_Col_Outer_Loop_Xpose_Col_Inner_Loop	-	65	650.000	3	1	64
dct_Pipeline_WR_Loop_Row_WR_Loop_Col	-	67	670.000	-	67	-
WR_Loop_Row_WR_Loop_Col	-	65	650.000	3	1	64
Row_DCT_Loop_DCT_Outer_Loop	-	1088	1.088E4	17	-	64
dct_Pipeline_DCT_Inner_Loop	-	13	130.000	-	13	-
DCT_Inner_Loop	-	11	110.000	5	1	8
Col_DCT_Loop_DCT_Outer_Loop	-	1088	1.088E4	17	-	64
dct_Pipeline_DCT_Inner_Loop1	-	13	130.000	-	13	-
DCT_Inner_Loop	-	11	110.000	5	1	8

The Performance & Resource Estimates of dct

4. In the Schedule Viewer pane, select the **Col_DCT_Loop_DCT_Outer_Loop** entry, right-click on the **col_outbuf_addr_write_ln69** (write) block in the **Schedule Viewer**, and select **Goto** Source. Notice that line **19** is highlighted which is preventing the flattening of the **DCT_Outer_Loop**.

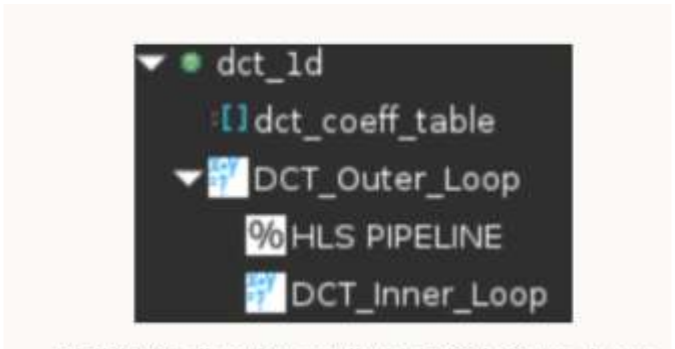


```
DCT_Outer_Loop:
   for (k = 0; k < DCT_SIZE; k++) {
DCT_Inner_Loop:
      for(n = 0, tmp = 0; n < DCT_SIZE; n++) {
         int coeff = (int)dct_coeff_table[k][n];
         tmp += src[n] * coeff;
      }
      dst[k] = DESCALE(tmp, CONST_BITS);
   }
}
```

5. Switch to the *Synthesis* perspective.

Create a new solution by copying the previous solution settings. Apply fine-grain parallelism of performing multiply and add operations of the inner loop of dct_1d using PIPELINE directive by moving the PIPELINE directive from inner loop to the outer loop of dct_1d. Generate the solution and analyze the output.

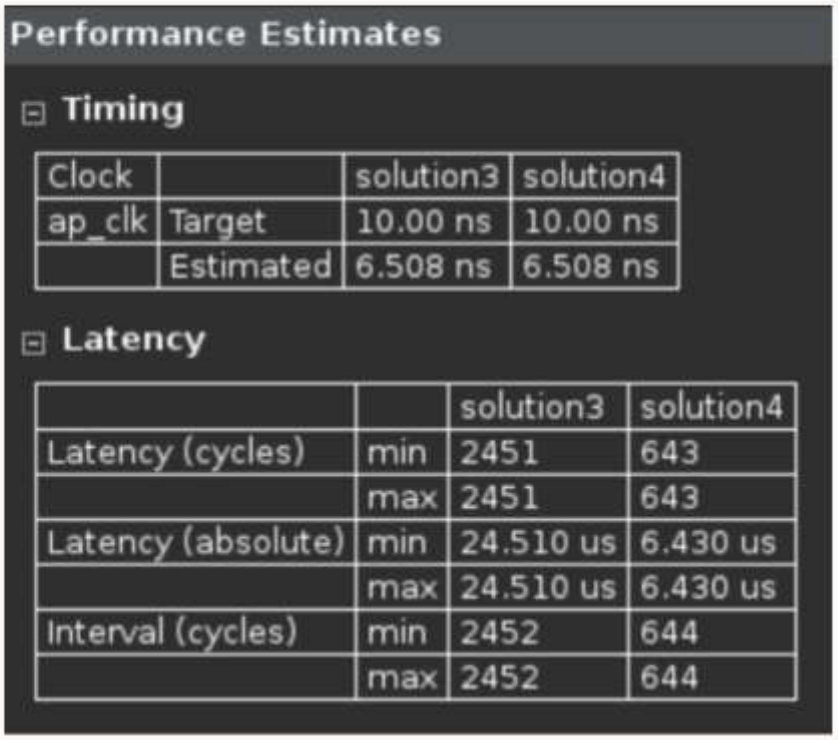
1. Select **Project > New Solution**.
2. A *Solution Configuration* dialog box will appear. Click the **Finish** button (with *Solution3* selected).
3. Select **PIPELINE** directive of **DCT_Inner_Loop** of the **dct_1d** function in the Directive pane, right-click on it and select **Remove** Directive.
4. Select **DCT_Outer_Loop** of the **dct_1d** function in the Directive pane, right-click on it and select **Insert Directive...**
5. A pop-up menu shows up listing various directives. Select **PIPELINE** directive.
6. Click **OK**



PIPELINE directive applied to DCT_Outer_Loop

By pipelining an outer loop, all inner loops will be unrolled automatically (if legal), so there is no need to explicitly apply an UNROLL directive to DCT_Inner_Loop. Simply move the pipeline to the outer loop: the nested loop will still be pipelined but the operations in the inner-loop body will operate concurrently.

7. Click on the **Synthesis** button.
8. When the synthesis is completed, select **Project > Compare Reports…** to compare the two solutions.
9. Select *Solution3* and *Solution4* from the **Available Reports,** click on the **Add>>** button, and then click **OK**.
10. Observe that the latency is reduced from *2450* to ~*642* clock cycles.



Performance Estimates

Timing

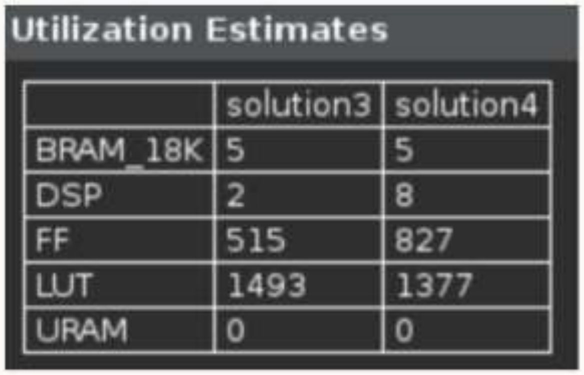
Clock		solution3	solution4
ap_clk	Target	10.00 ns	10.00 ns
	Estimated	6.508 ns	6.508 ns

Latency

		solution3	solution4
Latency (cycles)	min	2451	643
	max	2451	643
Latency (absolute)	min	24.510 us	6.430 us
	max	24.510 us	6.430 us
Interval (cycles)	min	2452	644
	max	2452	644

Performance comparison after pipelining

11. Scroll down in the comparison report to view the resources utilization. Observe that the utilization of DSP and FF has increased. Since the DCT_Inner_Loop was unrolled, the parallel computation requires 8 DSP48E.



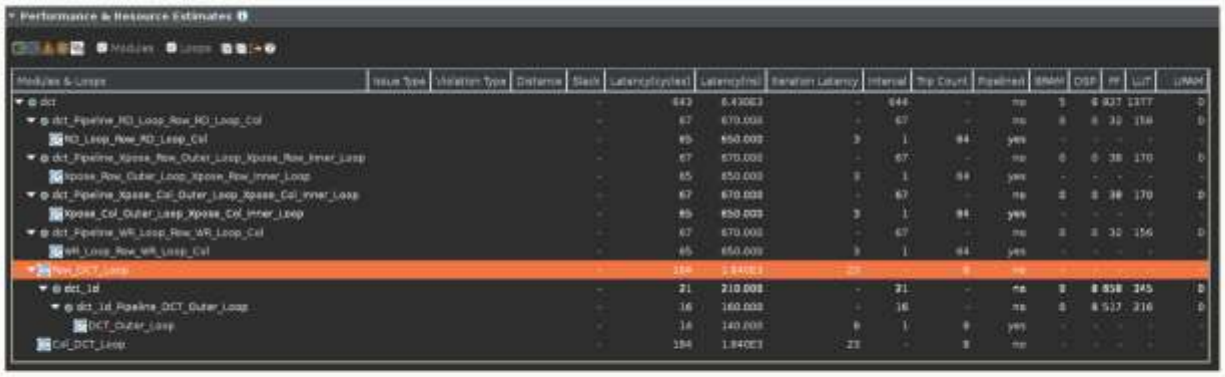
Utilization Estimates

	solution3	solution4
BRAM_18K	5	5
DSP	2	8
FF	515	827
LUT	1493	1377
URAM	0	0

Resources utilization after pipelining

Perform design analysis and look at the dct performance view.

1. Switch to the **Performance & Resource Estimates** in Synthesis Summary.
2. Expand, if necessary, and notice that the **DCT_Outer_Loop** is now pipelined and there is no **DCT_Inner_Loop** entry.



Performance & Resource Estimates

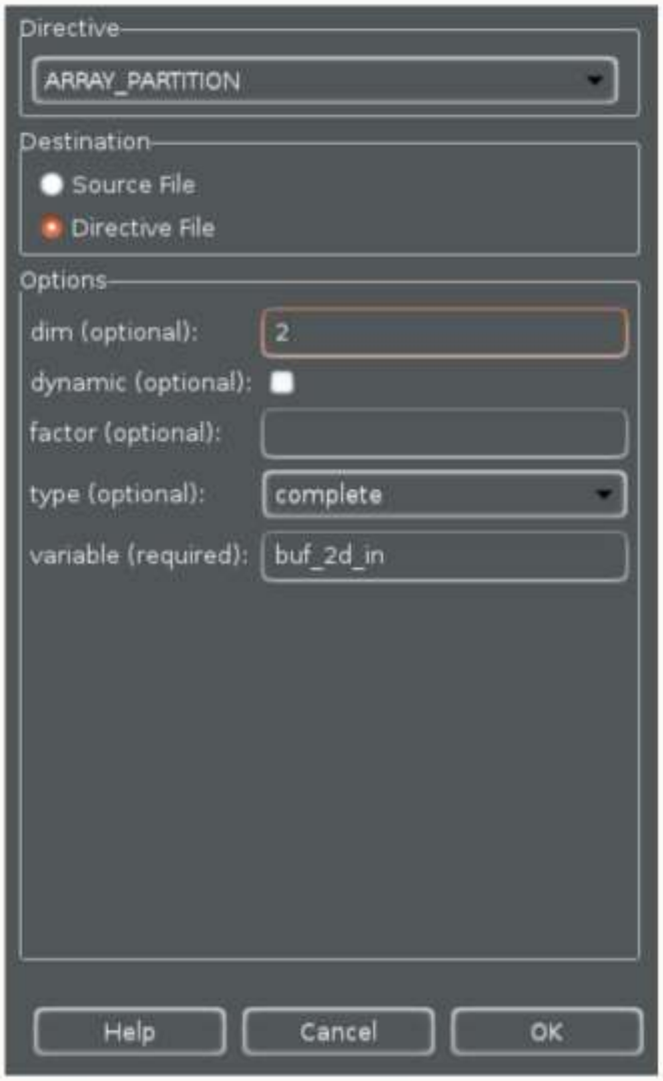
Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
dct				-	643	6.430E3	-	644	-	no	5	6	827	1377	0
dct_Pipeline_RD_Loop_Row_RD_Loop_Col				-	67	670.000	-	67	-	no	0	0	32	158	0
RD_Loop_Row_RD_Loop_Col				-	65	650.000	3	1	64	yes	-	-	-	-	-
dct_Pipeline_Xpose_Row_Outer_Loop_Xpose_Row_Inner_Loop				-	67	670.000	-	67	-	no	0	0	38	170	0
Xpose_Row_Outer_Loop_Xpose_Row_Inner_Loop				-	65	650.000	3	1	64	yes	-	-	-	-	-
dct_Pipeline_Xpose_Col_Outer_Loop_Xpose_Col_Inner_Loop				-	67	670.000	-	67	-	no	0	0	38	170	0
Xpose_Col_Outer_Loop_Xpose_Col_Inner_Loop				-	65	650.000	3	1	64	yes	-	-	-	-	-
dct_Pipeline_WR_Loop_Row_WR_Loop_Col				-	67	670.000	-	67	-	no	0	0	32	156	0
WR_Loop_Row_WR_Loop_Col				-	65	650.000	3	1	64	yes	-	-	-	-	-
Row_DCT_Loop				-	184	1.840E3	23	-	8	no	-	-	-	-	-
dct_1d				-	21	210.000	-	21	-	no	0	8	858	345	0
dct_1d_Pipeline_DCT_Outer_Loop				-	16	160.000	-	16	-	no	0	8	517	216	0
DCT_Outer_Loop				-	14	140.000	8	1	8	yes	-	-	-	-	-
Col_DCT_Loop				-	184	1.840E3	23	-	8	no	-	-	-	-	-

DCT_Outer_Loop flattening

STEP 7: Improve Memory Bandwidth

Create a new solution by copying the previous solution (Solution4) settings. Apply ARRAY_PARTITION directive to buf_2d_in of dct (since the bottleneck was on *src* port of the dct_1d function, which was passed via in_block of the dct_2d function, which in turn was passed via buf_2d_in of the dct function) and col_inbuf of dct_2d. Generate the solution.

1. Select **Project > New Solution** to create a new solution.
2. A *Solution Configuration* dialog box will appear. Click the **Finish** button (with *Solution4* selected).
3. With dct.c open, select **buf_2d_in** array of the **dct** function in the Directive pane, right-click on it and select **Insert Directive...**
   The *buf_2d_in* array is selected since the bottleneck was on *src* port of the *dct_1d* function, which was passed via *in_block* of the *dct_2d* function, which in turn was passed via *buf_2d_in* of the **dct** function).
4. A pop-up menu shows up listing various directives. Select **ARRAY_PARTITION** directive.
5. Make sure that the **type** is *complete*. Enter **2** in the *dimension* field and click **OK**.


Directive: ARRAY_PARTITION

Destination: Source File [✓ Directive File]

Options:

dim (optional): 2

dynamic (optional): ☐

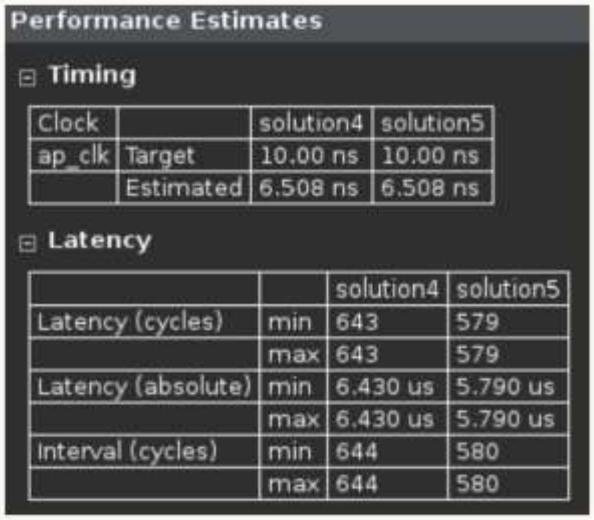
factor (optional):

type (optional): complete

variable (required): buf_2d_in

Applying ARRAY_PARTITION directive to memory buffer

6. Similarly, apply the **ARRAY_PARTITION** directive with dimension of *2* to the **col_inbuf** array of the **dct_2d** function.
7. Click on the **Synthesis** button.
8. When the synthesis is completed, select **Project > Compare Reports…** to compare the two solutions.
9. Select *Solution4* and *Solution5* from the **Available Reports,** and click on the **Add>>** button.
10. Observe the reduction in the latency from ***~642*** to ~**578** clock cycles.



Performance Estimates

Timing

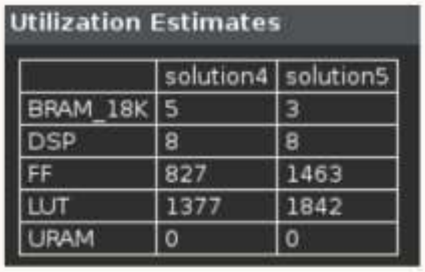
Clock		solution4	solution5
ap_clk	Target	10.00 ns	10.00 ns
	Estimated	6.508 ns	6.508 ns

Latency

		solution4	solution5
Latency (cycles)	min	643	579
	max	643	579
Latency (absolute)	min	6.430 us	5.790 us
	max	6.430 us	5.790 us
Interval (cycles)	min	644	580
	max	644	580

Performance comparison after array partitioning

11. Scroll down in the comparison report to view the resources utilization. Observe the increase in the FF resource utilization (almost double).



Utilization Estimates

	solution4	solution5
BRAM_18K	5	3
DSP	8	8
FF	827	1463
LUT	1377	1842
URAM	0	0

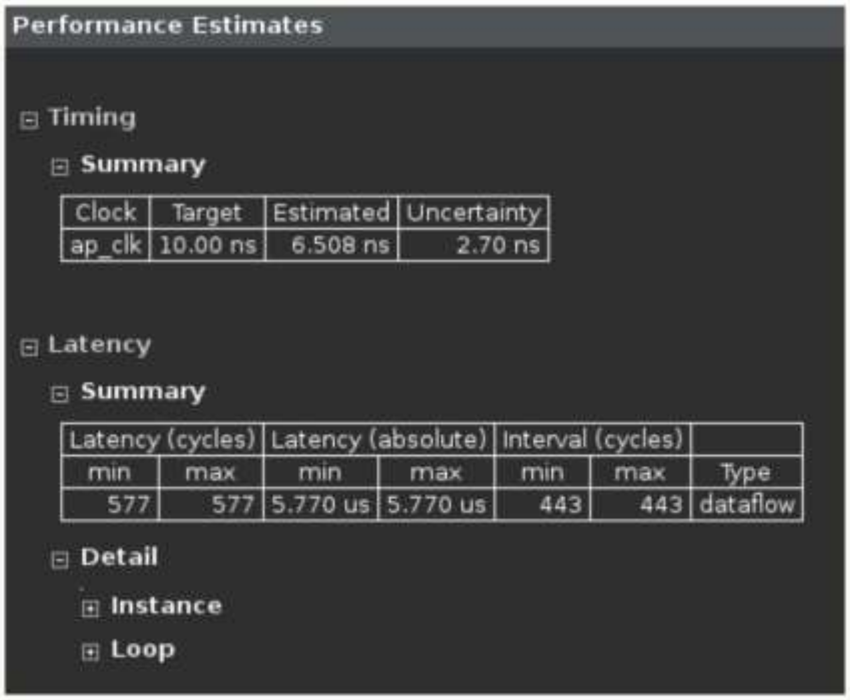
Resources utilization after array partitioning

12. Expand the Loop entry **DCT_Outer_Loop** of **dct_1d** function in the **Performance & Resource Estimates** in *Synthesis Summary* and observe that the Pipeline II is now 1.

STEP 8: Apply DATAFLOW Directive

Create a new solution by copying the previous solution (Solution5) settings. Apply the DATAFLOW directive to improve the throughput. Generate the solution and analyze the output.

1. Select **Project > New Solution**.
2. A *Solution Configuration* dialog box will appear. Click the **Finish** button (with *Solution5* selected).
3. Close all inactive solution windows by selecting **Project > Close Inactive Solution Tabs.**
4. Select function **dct** in the directives pane, right-click on it and select **Insert Directive...**
5. Select **DATAFLOW** directive to improve the throughput.
6. Click on the **Synthesis** button.
7. When the synthesis is completed, the synthesis report is automatically opened.
8. Observe that dataflow type pipeline throughput is listed in the “Performance Estimates”.



Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	6.508 ns	2.70 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
577	577	5.770 us	5.770 us	443	443	dataflow

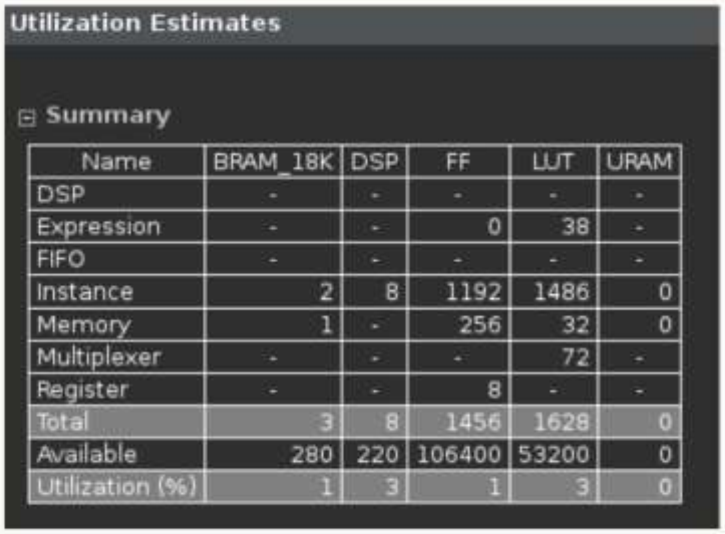
Detail

Instance

Loop

Performance estimate after DATAFLOW directive applied

- The Dataflow pipeline throughput indicates the number of clock cycles between each set of inputs reads (interval parameter). If this value is less than the design latency it indicates the design can start processing new inputs before the currents input data are output.
- Note that the dataflow is only supported for the functions and loops at the top-level, not those which are down through the design hierarchy. Only loops and functions exposed at the toplevel of the design will get benefit from dataflow optimization.
  1. Scrolling down into the *Utilization Estimates*, observe that the number of *BRAM_18K* required at the top-level remained at *3.*



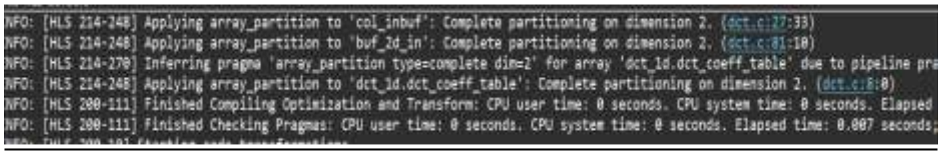
Utilization Estimates

Summary

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	38	-
FIFO	-	-	-	-	-
Instance	2	8	1192	1486	0
Memory	1	-	256	32	0
Multiplexer	-	-	-	72	-
Register	-	-	8	-	-
Total	3	8	1456	1628	0
Available	280	220	106400	53200	0
Utilization (%)	1	3	1	3	0

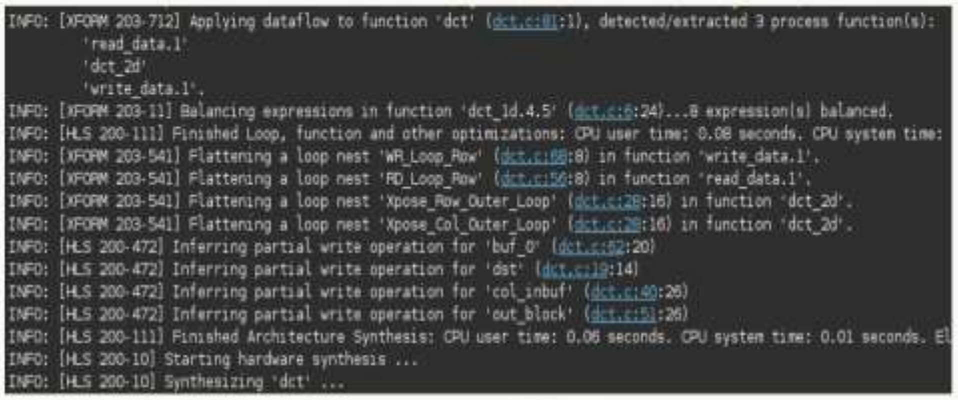
Resource estimate with DATAFLOW directive applied

2. Look at the console view and notice that **dct_coeff_table** is automatically partitioned in dimension *2*.



```
NFO: [HLS 214-248] Applying array_partition to 'col_inbuf': Complete partitioning on dimension 2. (dct.c:27:33)
NFO: [HLS 214-248] Applying array_partition to 'buf_2d_in': Complete partitioning on dimension 2. (dct.c:81:10)
NFO: [HLS 214-270] Inferring pragma 'array_partition type=complete dim=2' for array 'dct_1d.dct_coeff_table' due to pipeline pra
NFO: [HLS 214-248] Applying array_partition to 'dct_1d.dct_coeff_table': Complete partitioning on dimension 2. (dct.c:6:0)
NFO: [HLS 200-111] Finished Compiling Optimization and Transform: CPU user time: 0 seconds. CPU system time: 0 seconds. Elapsed
NFO: [HLS 200-111] Finished Checking Pragmas: CPU user time: 0 seconds. CPU system time: 0 seconds. Elapsed time: 0.007 seconds;
```

3. The *buf_2d_in* and *col_inbuf* arrays are partitioned as we had applied the directive in the previous run (shown in the figure above). The dataflow is applied at the top-level which created channels between top-level functions *read_data, dct_2d*, and *write_data*.

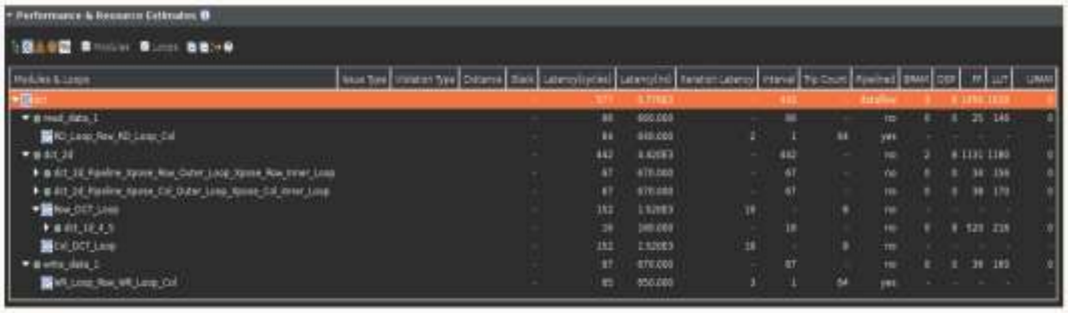


```
INFO: [XFORM 203-712] Applying dataflow to function 'dct' (dct.c:81:1), detected/extracted 3 process function(s):
         'read_data.1'
         'dct_2d'
         'write_data.1'.
INFO: [XFORM 203-11] Balancing expressions in function 'dct_1d.4.5' (dct.c:6:24)...8 expression(s) balanced.
INFO: [HLS 200-111] Finished Loop, function and other optimizations: CPU user time: 0.08 seconds. CPU system time:
INFO: [XFORM 203-541] Flattening a loop nest 'WR_Loop_Row' (dct.c:68:8) in function 'write_data.1'.
INFO: [XFORM 203-541] Flattening a loop nest 'RD_Loop_Row' (dct.c:56:8) in function 'read_data.1'.
INFO: [XFORM 203-541] Flattening a loop nest 'Xpose_Row_Outer_Loop' (dct.c:28:16) in function 'dct_2d'.
INFO: [XFORM 203-541] Flattening a loop nest 'Xpose_Col_Outer_Loop' (dct.c:28:16) in function 'dct_2d'.
INFO: [HLS 200-472] Inferring partial write operation for 'buf_0' (dct.c:62:20)
INFO: [HLS 200-472] Inferring partial write operation for 'dst' (dct.c:19:14)
INFO: [HLS 200-472] Inferring partial write operation for 'col_inbuf' (dct.c:40:26)
INFO: [HLS 200-472] Inferring partial write operation for 'out_block' (dct.c:51:26)
INFO: [HLS 200-111] Finished Architecture Synthesis: CPU user time: 0.06 seconds. CPU system time: 0.01 seconds. El
INFO: [HLS 200-10] Starting hardware synthesis ...
INFO: [HLS 200-10] Synthesizing 'dct' ...
```

Console view of synthesis process after DATAFLOW directive applied

Perform performance analysis by switching to the Synthesis Summary and looking at the dct Performance & Resource Estimates view.

1. Switch to the *Synthesis Summary* perspective, expand the *Performance & Resource Estimates* entries, and select the **dct_2d** entry.
2. Observe that most of the latency and interval (throughput) is caused by the **dct_2d** function. The interval of the top-level function **dct**, is less than the sum of the intervals of the **read_data**, **dct_2d**, and **write_data** functions indicating that they operate in parallel and **dct_2d** is the limiting factor. It can be seen that **dct_2d** is not completely operating in parallel as **Row_DCT_Loop** and **Col_DCT_Loop** were not pipelined.



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
dct				-	577	5.770E3	-	442	-	dataflow	3	0	1350	1510	0
read_data_1				-	88	880.000	-	88	-	no	0	0	25	146	0
RD_Loop_Row_RD_Loop_Col				-	84	840.000	2	1	64	yes	-	-	-	-	-
dct_2d				-	442	4.420E3	-	442	-	no	2	0	1131	1180	0
dct_2d_Pipeline_Xpose_Row_Outer_Loop_Xpose_Row_Inner_Loop				-	67	670.000	-	67	-	no	0	0	38	156	0
dct_2d_Pipeline_Xpose_Col_Outer_Loop_Xpose_Col_Inner_Loop				-	67	670.000	-	67	-	no	0	0	38	170	0
Row_DCT_Loop				-	152	1.520E3	19	-	8	no	-	-	-	-	-
dct_1d_4_5				-	18	180.000	-	18	-	no	0	0	520	218	0
Col_DCT_Loop				-	152	1.520E3	19	-	8	no	-	-	-	-	-
write_data_1				-	87	870.000	-	87	-	no	0	0	38	183	0
WR_Loop_Row_WR_Loop_Col				-	85	850.000	3	1	64	yes	-	-	-	-	-

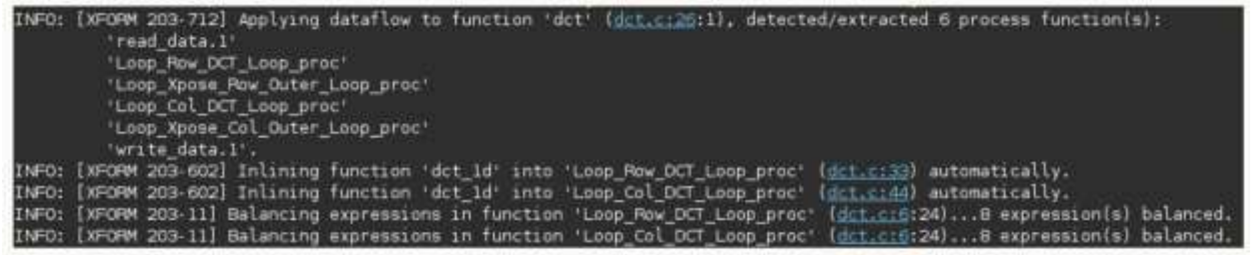
Performance analysis after the DATAFLOW directive

One of the limitations of the dataflow optimization is that it only works on top-level loops and functions. One way to have the blocks in **dct_2d** operate in parallel would be to pipeline the entire function. This however would unroll all the loops and can sometimes lead to a large area increase. An alternative is to raise these loops up to the top-level of hierarchy, where dataflow optimization can be applied, by removing the **dct_2d** hierarchy, i.e. inline the **dct_2d** function.

STEP 9: Apply INLINE Directive

Create a new solution by copying the previous solution (Solution6) settings. Apply INLINE directive to dct_2d. Generate the solution and analyze the output.

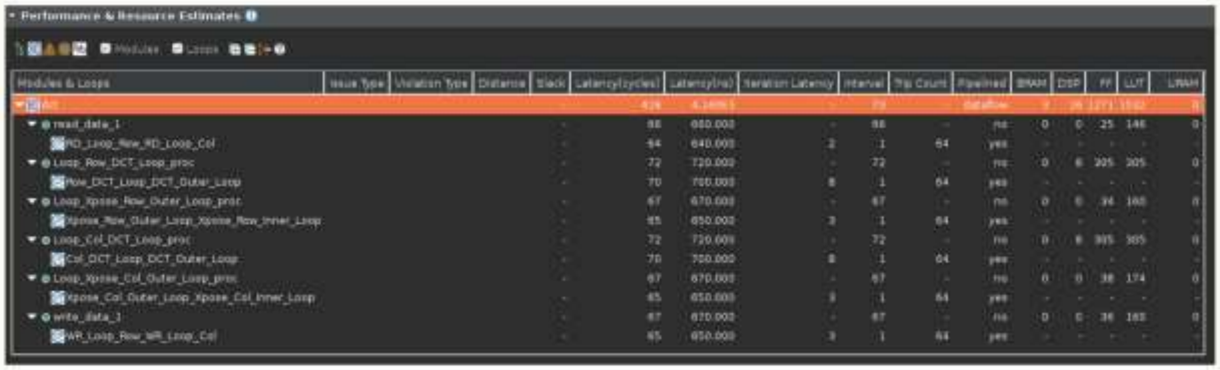
1. Select **Project > New Solution**.
2. A *Solution Configuration* dialog box will appear. Click the **Finish** button (with *Solution6* selected).
3. Select the function **dct_2d** in the directives pane, right-click on it and select **Insert Directive...**
4. A pop-up menu shows up listing various directives. Select **INLINE** directive. The INLINE directive causes the function to which it is applied to be inlined: its hierarchy is dissolved.
5. Click on the *Synthesis* button.
6. When the synthesis is completed, the synthesis report will be opened.
7. Observe by comparing the reports of *solution6* and *solution7* that the latency reduced from *577* to *416* clock cycles, and the Dataflow pipeline throughput drastically reduced from *443* to *73* clock cycles.
8. Examine the synthesis log to see what transformations were applied automatically.
   - The **dct_1d** function calls are now automatically inlined into the loops from which they are called, which allows the loop nesting to be flattened automatically.
   - Note also that the DSP48E usage has doubled (**from 8 to 16**). This is because, previously a single instance of dct_1d was used to do both row and column processing; now that the row and column loops are executing concurrently, this can no longer be the case and two copies of dct_1d are required: Vitis HLS will seek to minimize the number of clocks, even if it means increasing the area.



```
INFO: [XFORM 203-712] Applying dataflow to function 'dct' (dct.c:26:1), detected/extracted 6 process function(s):
         'read_data.1'
         'Loop_Row_DCT_Loop_proc'
         'Loop_Xpose_Row_Outer_Loop_proc'
         'Loop_Col_DCT_Loop_proc'
         'Loop_Xpose_Col_Outer_Loop_proc'
         'write_data.1'.
INFO: [XFORM 203-602] Inlining function 'dct_1d' into 'Loop_Row_DCT_Loop_proc' (dct.c:33) automatically.
INFO: [XFORM 203-602] Inlining function 'dct_1d' into 'Loop_Col_DCT_Loop_proc' (dct.c:44) automatically.
INFO: [XFORM 203-11] Balancing expressions in function 'Loop_Row_DCT_Loop_proc' (dct.c:6:24)...8 expression(s) balanced.
INFO: [XFORM 203-11] Balancing expressions in function 'Loop_Col_DCT_Loop_proc' (dct.c:6:24)...8 expression(s) balanced.
```

Console view after INLINE directive applied to dct_2d

9. Switch to the *Synthesis Summary* perspective, expand the *Performance & Resource Estimates* entries, and select the **dct** entry.
   Observe that the dct_2d entry is now replaced with ***dct_Loop_Row_DCT_Loop_proc***, ***dct_Loop_Xpose_Row_Outer_Loop_proc***, ***dct_Loop_Col_DCT_Loop_proc***, and ***dct_Loop_Xpose_Col_Outer_Loop_proc*** since the dct_2d function is inlined. Also observe that all the functions are operating in parallel, yielding the top-level function interval (throughput) of *73* clock cycles.



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
dct				-	[illegible]	[illegible]	-	[illegible]	-	dataflow	[illegible]	[illegible]	[illegible]	[illegible]	0
read_data_1				-	88	880.000	-	88	-	no	0	0	25	146	0
RD_Loop_Row_RD_Loop_Col				-	64	640.000	2	1	64	yes	-	-	-	-	-
Loop_Row_DCT_Loop_proc				-	72	720.000	-	72	-	no	0	6	305	305	0
Row_DCT_Loop_DCT_Outer_Loop				-	70	700.000	8	1	64	yes	-	-	-	-	-
Loop_Xpose_Row_Outer_Loop_proc				-	67	670.000	-	67	-	no	0	0	36	160	0
Xpose_Row_Outer_Loop_Xpose_Row_Inner_Loop				-	65	650.000	3	1	64	yes	-	-	-	-	-
Loop_Col_DCT_Loop_proc				-	72	720.000	-	72	-	no	0	6	305	305	0
Col_DCT_Loop_DCT_Outer_Loop				-	70	700.000	8	1	64	yes	-	-	-	-	-
Loop_Xpose_Col_Outer_Loop_proc				-	67	670.000	-	67	-	no	0	0	38	174	0
Xpose_Col_Outer_Loop_Xpose_Col_Inner_Loop				-	65	650.000	3	1	64	yes	-	-	-	-	-
write_data_1				-	67	670.000	-	67	-	no	0	0	36	165	0
WR_Loop_Row_WR_Loop_Col				-	65	650.000	3	1	64	yes	-	-	-	-	-

Performance analysis after the INLINE directive

10. Close Vitis HLS by selecting **File > Exit**.

CONCLUSION

In this lab, you learned various techniques to improve the performance and balance resource utilization. PIPELINE directive when applied to outer loop will automatically cause the inner loop to unroll. When a loop is unrolled, resources utilization increases as operations are done concurrently. Partitioning memory may improve performance but will increase BRAM utilization. When INLINE directive is applied to a function, the lower level hierarchy is automatically dissolved. When DATAFLOW directive is applied, the default memory buffers (of ping-pong type) are automatically inserted between the top-level functions and loops. The console logs can provide insight on what is going on.

References:

https://xilinx.github.io/xup_high_level_synthesis_design_flow/Lab3.html