

# Improving Area and Resource Utilization Lab

## Introduction

This lab introduces various techniques and directives which can be used in Vivado 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.

## Objectives

After completing this lab, you will be able to:

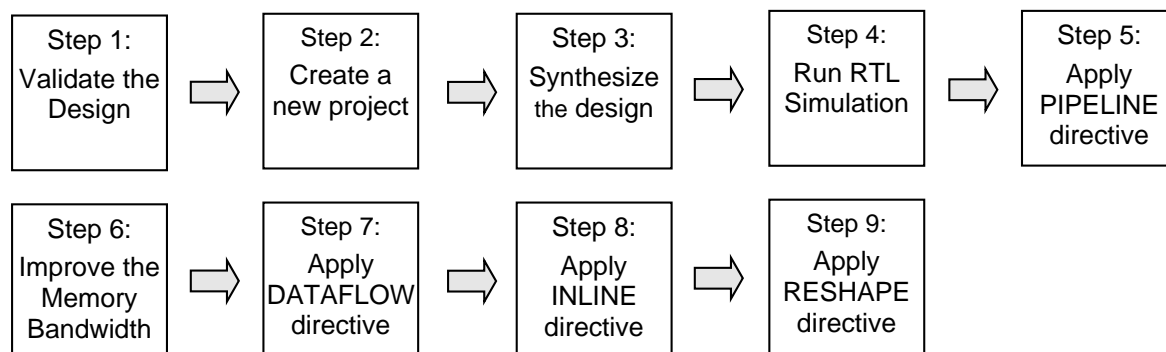
- Add directives in your design
- Improve performance using PIPELINE directive
- Distinguish between DATAFLOW directive and Configuration Command functionality
- Apply memory partitions techniques to improve resource utilization

## Procedure

This lab is separated into steps that consist of general overview statements that provide information on the detailed instructions that follow. Follow these detailed instructions to progress through the lab.

This lab comprises 9 primary steps: You will validate the design in Vivado HLS command prompt, create a new project using Vivado HLS GUI, synthesize the design, run RTL simulation, apply PIPELINE directive to improve performance, improve the memory bandwidth by applying PARTITION directive, apply DATAFLOW directive, apply INLINE directive, and finally apply RESHAPE directive.

## General Flow for this Lab



## Validate the Design from Command Line

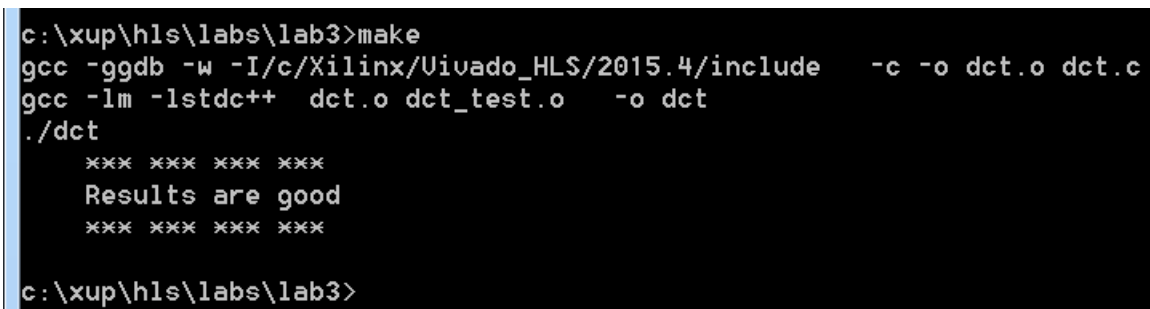
### Step 1

#### 1-1. Validate your design from Vivado HLS command line.

1-1-1. Launch Vivado HLS: Select **Start > All Programs > Xilinx Design Tools > Vivado 2015.4 > Vivado HLS > Vivado HLS 2015.4 Command Prompt**.

1-1-2. In the Vivado HLS Command Prompt, change directory to **c:\xup\hls\labs\lab3**.

1-1-3. 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. In the Vivado HLS Command Prompt, type **make** to compile and execute the program.



```
c:\xup\hls\labs\lab3>make
gcc -ggdb -w -I/c/Xilinx/Vivado_HLS/2015.4/include -c -o dct.o dct.c
gcc -lm -lstdc++ dct.o dct_test.o -o dct
./dct
*** ** Results are good ***
c:\xup\hls\labs\lab3>
```

Figure 1. 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.

1-1-4. Close the command prompt window by typing **exit**.

## Create a New Project

### Step 2

#### 2-1. Create a new project in Vivado HLS GUI targeting XC7Z020CLG484-1 (ZedBoard) or XC7Z010CLG400-1 (Zybo).

2-1-1. Launch Vivado HLS: Select **Start > All Programs > Xilinx Design Tools > Vivado 2015.4 > Vivado HLS > Vivado HLS 2015.4**

2-1-2. In the Vivado HLS GUI, select **File > New Project**. The New Vivado HLS Project wizard opens.

2-1-3. Click **Browse...** button of the Location field and browse to **c:\xup\hls\labs\lab3** and then click **OK**.

2-1-4. For Project Name, type **dct.prj**

2-1-5. Click **Next**.

- 2-1-6.** 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**).
- 2-1-7.** Click the **Add Files...** button, select *dct.c* file from the **c:\xup\hls\labs\lab3** folder, and then click **Open**.
- 2-1-8.** Click **Next**.
- 2-1-9.** 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 **c:\xup\hls\labs\lab3** folder and click **Open**.
- 2-1-10.** Click **Next**.
- 2-1-11.** In the *Solution Configuration* page, leave Solution Name field as **solution1** and set the clock period as **10** (for ZedBoard) or **8** (for Zybo). Leave Uncertainty field blank as it will take 1.25 as the default value for ZedBoard and 1 for Zybo.
- 2-1-12.** Click on Part's Browse button, and select the following filters, using the *Parts Specify* option, to select **xc7z020clg484-1** (ZedBoard) or **xc7z010clg400-1** (Zybo), and click **OK**:  
 Family: **Zynq**  
 Sub-Family: **Zynq**  
 Package: **clg484** (ZedBoard) or **clg400** (Zybo)  
 Speed Grade: **-1**
- 2-1-13.** Click **Finish**.
- 2-1-14.** Double-click on the **dct.c** under the *source* folder to open its content in the information pane.

```

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
89     // Write out the results.
90     write_data(buf_2d_out, output);
91 }

```

**Figure 2. The design under consideration**

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 executed and the flow of data on the right-hand side.

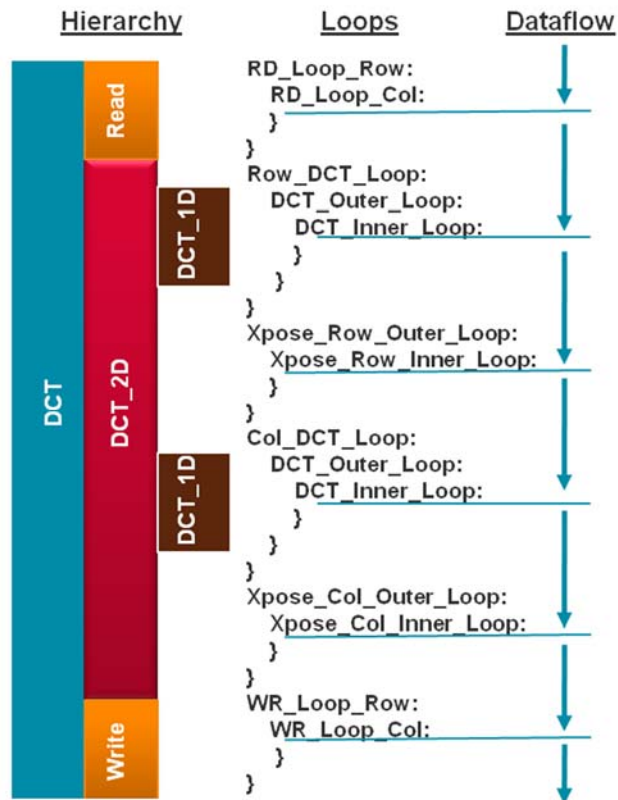


Figure 3. Design hierarchy and dataflow

## Synthesize the Design

## Step 3

**3-1. Synthesize the design with the defaults. View the synthesis results and answer the question listed in the detailed section of this step.**

**3-1-1.** Select **Solution > Run C Synthesis > Active Solution** or click on the  button to start the synthesis process.

**3-1-2.** When synthesis is completed, several report files will become accessible and the Synthesis Results will be displayed in the information pane.

Note that the Synthesis Report section in the Explorer view only shows dct\_1d.rpt, dct\_2d.rpt, and dct.rpt entries. The read\_data and write\_data functions reports are not listed. This is because these two functions are inlined. Verify this by scrolling up into the Vivado HLS Console view.

```

Vivado HLS Console
@I [HLS-10] Analyzing design file 'dct.c' ...
@I [HLS-10] Validating synthesis directives ...
@I [HLS-10] Starting code transformations ...
@I [HLS-10] Checking synthesizability ...
@I [XFORM-602] Inlining function 'read_data' into 'dct' (dct.c:85) automatically.
@I [XFORM-602] Inlining function 'write_data' into 'dct' (dct.c:90) automatically.
@I [XFORM-602] Inlining function 'read_data' into 'dct' (dct.c:85) automatically.
@I [XFORM-602] Inlining function 'write_data' into 'dct' (dct.c:90) automatically.
@I [HLS-111] Elapsed time: 4.55 seconds; current memory usage: 69.8 MB.

```

Figure 4. Inlining of read\_data and write\_data functions

- 3-1-3. The Synthesis Report shows the performance and resource estimates as well as estimated latency in the design. Note that the design is not optimized nor is pipelined.

**Performance Estimates**

▢ **Timing (ns)**

▢ **Summary**

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	6.38	1.25

▢ **Latency (clock cycles)**

▢ **Summary**

Latency		Interval		Type
min	max	min	max	
3959	3959	3960	3960	none

▢ **Detail**

⊕ **Instance**

⌘ **Loop**

Loop Name	Latency		Iteration Latency	Initiation Interval		Trip Count	Pipelined
	min	max		achieved	target		
- RD_Loop_Row	144	144	18	-	-	8	no
+ RD_Loop_Col	16	16	2	-	-	8	no
- WR_Loop_Row	144	144	18	-	-	8	no
+ WR_Loop_Col	16	16	2	-	-	8	no

Figure 5. Synthesis report

- 3-1-4. 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: \_\_\_\_\_

**3-1-5.** 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_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

**Figure 6. 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.

**3-1-6.** Open dct\_1d.rpt and dct\_2d.rpt files either using the Explorer view or by using a hyperlink at the bottom of the dct.rpt in the information view. The report for dct\_2d clearly indicates that most of this design cycles (3668) are spent doing the row and column DCTs. Also the dct\_1d report indicates that the latency is 209 clock cycles  $((24+2)*8+1)$ .

## Run Co-Simulation

## Step 4

**4-1. Run the Co-simulation, selecting Verilog. Verify that the simulation passes.**

**4-1-1.** Select **Solution > Run C/RTL Cosimulation** or click on the ☒ button to open the dialog box so the desired simulations can be run.

A C/RTL Co-simulation Dialog box will open.

**4-1-2.** Select the Verilog 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.

## Cosimulation Report for 'dct'


Result							
		Latency			Interval		
RTL	Status	min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	Pass	3959	3959	3959	0	0	0

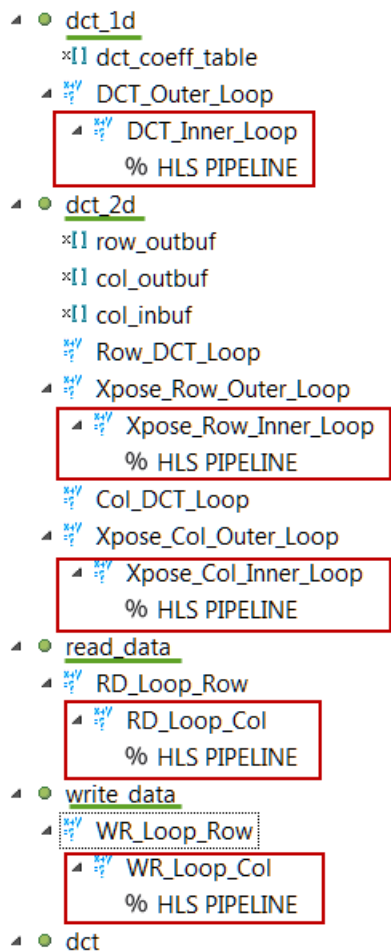
```
INFO: [Common 17-206] Exiting xsim at Sun Dec 20 21:45:27 2015...
@I [SIM-316] Starting C post checking ...
*** **
Results are good
*** **
@I [SIM-1000] *** C/RTL co-simulation finished: PASS ***
```

Figure 7. RTL Co-Simulation results

## Apply PIPELINE Directive

## Step 5

- 5-1. 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.
  - 5-1-1. Select **Project > New Solution** or click on (  ) from the tools bar buttons.
  - 5-1-2. A *Solution Configuration* dialog box will appear. Click the **Finish** button (with *copy from Solution1* selected).
  - 5-1-3. Make sure that the **dct.c** source is opened in the information pane and click on the **Directive** tab.
  - 5-1-4. Select **DCT\_Inner\_Loop** of the **dct\_1d** function in the Directive pane, right-click on it and select *Insert Directive...*
  - 5-1-5. A pop-up menu shows up listing various directives. Select **PIPELINE** directive.
  - 5-1-6. Leave **II** (Initiation Interval) blank as Vivado HLS will try for an **II=1**, one new input every clock cycle.
  - 5-1-7. Click **OK**.
  - 5-1-8. 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. At this point, the Directive tab should look like as follows.



**Figure 8. PIPELINE directive applied**

**5-1-9.** Click on the **Synthesis** button.

**5-1-10.** When the synthesis is completed, select **Project > Compare Reports...** or click on  to compare the two solutions.

**5-1-11.** Select *Solution1* and *Solution2* from the **Available Reports**, click on the **Add>>** button, and then click **OK**.

**5-1-12.** Observe that the latency reduced from 3959 to 1850 clock cycles (ZedBoard) and from 3959 to 1854 (Zybo).



**Performance Estimates**⊟ **Timing (ns)**

Clock		solution2	solution1
ap_clk	Target	10.00	10.00
	Estimated	7.68	6.38

⊟ **Latency (clock cycles)**

		solution2	solution1
Latency	min	1850	3959
	max	1850	3959
Interval	min	1851	3960
	max	1851	3960

(a) ZedBoard

**Performance Estimates**⊟ **Timing (ns)**

Clock		solution2	solution1
ap_clk	Target	8.00	8.00
	Estimated	6.60	6.38

⊟ **Latency (clock cycles)**

		solution2	solution1
Latency	min	1854	3959
	max	1854	3959
Interval	min	1855	3960
	max	1855	3960

(b) Zybo

**Figure 9. Performance comparison after pipelining**

**5-1-13.** 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	solution1
BRAM_18K	5	5
DSP48E	1	1
FF	255	278
LUT	458	354

(a) ZedBoard

**Utilization Estimates**

	solution2	solution1
BRAM_18K	5	5
DSP48E	1	1
FF	289	278
LUT	462	354

(b) Zybo

**Figure 10. Resources utilization after pipelining**

**5-2. Open the Analysis perspective and determine where most of the clock cycles are spend, i.e. where the large latencies are.**

**5-2-1.** Click on the *Analysis* perspective button.

**5-2-2.** In the Module Hierarchy, select the dct entry and observe the RD\_Loop\_Row\_RD\_Loop\_Col and WR\_Loop\_Row\_WR\_Loop\_Col entries. 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.

```
@I [XFORM-602] Inlining function 'read_data' into 'dct' (dct.c:85) automatically.
@I [XFORM-602] Inlining function 'write_data' into 'dct' (dct.c:90) automatically.
@I [XFORM-541] Flattening a loop nest 'Xpose_Row_Outer_Loop' (dct.c:38:1) in function 'dct_2d'.
@I [XFORM-541] Flattening a loop nest 'Xpose_Col_Outer_Loop' (dct.c:49:1) in function 'dct_2d'.
@W [XFORM-542] Cannot flatten a loop nest 'DCT_Outer_Loop' (dct.c:13:67) in function 'dct_1d' :
the outer loop is not a perfect loop because there is nontrivial logic in the loop latch.
@I [XFORM-541] Flattening a loop nest 'RD_Loop_Row' (dct.c:59:67) in function 'dct'.
@I [XFORM-541] Flattening a loop nest 'WR_Loop_Row' (dct.c:71:67) in function 'dct'.
```

**Figure 11. The console view content indicating loops flattening**

Module Hierarchy								
	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type	
└─ <b>dct</b>	5	1	255	458	1850	1851	none	
└─ <b>dct_dct_2d</b>	3	1	194	320	1717	1717	none	

Performance Profile					Resource Profile			
	Pipelined	Latency	Initiation Interval	Iteration Latency				
└─ <b>dct</b>	-	1850	1851	-				
└─ <b>RD_Loop_Row_RD_Loop_Col</b>	yes	64	1	2				
└─ <b>WR_Loop_Row_WR_Loop_Col</b>	yes	64	1	2				

**(a) ZedBoard**

Module Hierarchy								
	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type	
└─ <b>dct</b>	5	1	289	462	1854	1855	none	
└─ <b>dct_dct_2d</b>	3	1	214	322	1719	1719	none	
└─ <b>dct_dct_1d2</b>	0	1	117	122	97	97	none	

Performance Profile					Resource Profile			
	Pipelined	Latency	Initiation Interval	Iteration Latency				
└─ <b>dct</b>	-	1854	1855	-				
└─ <b>RD_Loop_Row_RD_Loop_Col</b>	yes	65	1	3				
└─ <b>WR_Loop_Row_WR_Loop_Col</b>	yes	65	1	3				

**(b) Zybo****Figure 12. The performance profile at the dct function level**

- 5-2-3.** In the Module Hierarchy tab, expand **dct > dct\_2d > dct\_1d**. Notice that the most of the latency occurs is in **dct\_2d** function.
- 5-2-4.** In the Module Hierarchy tab, notice that there still hierarchy exists in the **dct\_2d** module. Expand **dct > dct\_2d > dct\_1d**, and select the **dct\_1d** entry.

Module Hierarchy								
	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type	
└─ <b>dct</b>	5	1	255	458	1850	1851	none	
└─ <b>dct_dct_2d</b>	3	1	194	320	1717	1717	none	
└─ <b>dct_dct_1d2</b>	0	1	117	122	97	97	none	

Performance Profile					Resource Profile			
	Pipelined	Latency	Initiation Interval	Iteration Latency	Trip count			
└─ <b>dct_dct_1d2</b>	-	97	97	-	-			
└─ <b>DCT_Outer_Loop</b>	no	96	-	12	8			
└─ <b>DCT_Inner_Loc</b>	yes	9	1	3	8			

**ZedBoard**

Module Hierarchy								
	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type	
▲ ● dct	5	1	289	462	1854	1855	none	
▲ ● dct_dct_2d	3	1	214	322	1719	1719	none	
● dct_dct_1d2	0	1	117	122	97	97	none	

Performance Profile						
	Pipelined	Latency	Initiation Interval	Iteration Latency	Trip count	
▲ ● dct_dct_1d2	-	97	97	-	-	
▲ ● DCT_Outer_Loop	no	96	-	12	8	
● DCT_Inner_Loop	yes	9	1	3	8	

(b) Zybo

Figure 13. The dct\_1d function performance profile

- 5-2-5.** In the Performance Profile tab, select the DCT\_Inner\_Loop entry, right-click on the node\_60 (write) block in the C3 state in the Performance view, and select Goto Source. Notice that line 19 is highlighted which is preventing the flattening of the DCT\_Outer\_Loop.

**Current Module :** `dct > dct dct 2d > dct dct 1d2`

Operation\Control S...	C0	C1	C2	C3	C4
1 tmp 21 read(read)					
2 tmp 2 read(read)					
3 DCT Outer Loop					
4 k(phi mux)					
5 exitcond1(icmp)					
6 k 1(+)					
7 tmp 9(+)					
8-... DCT Inner Loop					
19 tmp s(+)					
20 node 60(write)					

Performance Resource

Properties C Source

File: C:\xup\hls\labs\lab3\dct.c

```

18 }
19 dst[k] = DESCALE(tmp, CONST_BITS);
20 }
21 }
22
23 void dct_2d(dct_data_t in_block[DCT_SIZE][DCT_SIZE],
24            dct_data_t out_block[DCT_SIZE][DCT_SIZE])
25 {
26     dct_data_t row_outbuf[DCT_SIZE][DCT_SIZE];
27     dct_data_t col_outbuf[DCT_SIZE][DCT_SIZE], col_inbuf[DCT_SIZE][DCT_SIZE];
28     unsigned i, j;

```

Figure 14. Understanding what is preventing DCT\_Outer\_Loop flattening

- 5-2-6.** Switch to the *Synthesis* perspective.

**5-3. 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.**

**5-3-1.** Select **Project > New Solution**.

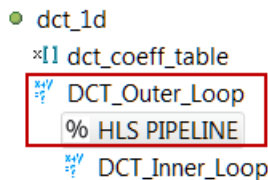
**5-3-2.** A *Solution Configuration* dialog box will appear. Click the **Finish** button (with Solution2 selected).

**5-3-3.** Select PIPELINE directive of **DCT\_Inner\_Loop** of the **dct\_1d** function in the Directive pane, right-click on it and select *Remove Directive*.

**5-3-4.** Select **DCT\_Outer\_Loop** of the **dct\_1d** function in the Directive pane, right-click on it and select *Insert Directive...*

**5-3-5.** A pop-up menu shows up listing various directives. Select **PIPELINE** directive.

**5-3-6.** Click **OK**.



**Figure 15. 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.

**5-3-7.** Click on the **Synthesis** button.

**5-3-8.** When the synthesis is completed, select **Project > Compare Reports...** to compare the two solutions.

**5-3-9.** Select *Solution2* and *Solution3* from the **Available Reports**, click on the **Add>>** button, and then click **OK**.

**5-3-10.** Observe that the latency reduced from 1850 to 874 clock cycles for ZedBoard (1854 to 878 for Zybo).

**Performance Estimates**⊞ **Timing (ns)**

Clock		solution3	solution2
ap_clk	Target	10.00	10.00
	Estimated	9.40	7.68

⊞ **Latency (clock cycles)**

		solution3	solution2
Latency	min	874	1850
	max	874	1850
Interval	min	875	1851
	max	875	1851

**(a) ZedBoard****Performance Estimates**⊞ **Timing (ns)**

Clock		solution3	solution2
ap_clk	Target	8.00	8.00
	Estimated	9.40	6.60

⊞ **Latency (clock cycles)**

		solution3	solution2
Latency	min	878	1854
	max	878	1854
Interval	min	879	1855
	max	879	1855

**(b) Zybo****Figure 16. Performance comparison after pipelining**

**5-3-11.** Scroll down in the comparison report to view the resources utilization. Observe that the utilization of all resources (except BRAM) increased. Since the DCT\_Inner\_Loop was unrolled, the parallel computation requires 8 DSP48E.

**Utilization Estimates**

	solution3	solution2
BRAM_18K	5	5
DSP48E	8	1
FF	677	255
LUT	520	458

**(a) ZedBoard****Utilization Estimates**

	solution3	solution2
BRAM_18K	5	5
DSP48E	8	1
FF	711	289
LUT	524	462

**(b) Zybo****Figure 17. Resources utilization after pipelining**

**5-3-12.** Open dct\_1d report and observe that the pipeline initiation interval (II) is four (4) cycles, not one (1) as might be hoped and there are now 8 BRAMs being used for the coefficient table.

Looking closely at the synthesis log, notice that the coefficient table was automatically partitioned, resulting in 8 separate ROMs: this helped reduce the latency by keeping the unrolled computation loop fed, however the input arrays to the dct\_1d function were not automatically partitioned.

The reason the II is four (4) rather than the eight (8) one might expect, is because Vivado HLS automatically uses dual-port RAMs, when beneficial to scheduling operations.

## Performance Estimates

## Timing (ns)

## Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	9.40	1.25

## Latency (clock cycles)

## Summary

Latency		Interval		
min	max	min	max	Type
36	36	36	36	none

## Detail

## Instance

## Loop

Loop Name	Latency		Iteration Latency	Initiation Interval		Trip Count	Pipelined
	min	max		achieved	target		
- DCT_Outer_Loop	34	34	7	4	1	8	yes

## Utilization Estimates

## Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	8	-	-
Expression	-	-	0	128
FIFO	-	-	-	-
Instance	-	-	-	-
Memory	0	-	119	16
Multiplexer	-	-	-	21
Register	-	-	420	-
<b>Total</b>	<b>0</b>	<b>8</b>	<b>539</b>	<b>165</b>
Available	280	220	106400	53200
Utilization (%)	0	3	~0	~0

Figure 18. Increased resource utilization of dct\_1d

```

@I [HLS-10] Starting code transformations ...
@I [HLS-10] Checking synthesizability ...
@I [XFORM-602] Inlining function 'read_data' into 'dct' (dct.c:85) automatically.
@I [XFORM-602] Inlining function 'write_data' into 'dct' (dct.c:90) automatically.
@I [XFORM-502] Unrolling all sub-loops inside loop 'DCT_Outer_Loop' (dct.c:13) in function 'dct_1d' for pipelining.
@I [XFORM-501] Unrolling loop 'DCT_Inner_Loop' (dct.c:15) in function 'dct_1d' completely.
@I [XFORM-102] Partitioning array 'dct_coeff_table' in dimension 2 automatically.
@I [XFORM-602] Inlining function 'read_data' into 'dct' (dct.c:85) automatically.
@I [XFORM-602] Inlining function 'write_data' into 'dct' (dct.c:90) automatically.
@I [XFORM-11] Balancing expressions in function 'dct_1d' (dct.c:4)...8 expression(s) balanced.
@I [XFORM-541] Flattening a loop nest 'Xpose_Row_Outer_Loop' (dct.c:38:1) in function 'dct_2d'.
@I [XFORM-541] Flattening a loop nest 'Xpose_Col_Outer_Loop' (dct.c:49:1) in function 'dct_2d'.
@I [XFORM-541] Flattening a loop nest 'RD_Loop_Row' (dct.c:59:67) in function 'dct'.
@I [XFORM-541] Flattening a loop nest 'WR_Loop_Row' (dct.c:71:67) in function 'dct'.
@I [HLS-111] Elapsed time: 5.291 seconds; current memory usage: 92.4 MB.

```

Figure 19. Automatic partitioning of dct\_coeff\_table

```

@I [HLS-10] -----
@I [HLS-10] -- Scheduling module 'dct_dct_1d2'
@I [HLS-10] -----
@I [SCHED-11] Starting scheduling ...
@I [SCHED-61] Pipelining loop 'DCT_Outer_Loop'.
@W [SCHED-69] Unable to schedule 'load' operation ('src_load_5', dct.c:17) on array 'src' due to limited memory ports.
@I [SCHED-61] Pipelining result: Target II: 1, Final II: 4, Depth: 7.
@I [SCHED-11] Finished scheduling.

```

Figure 20. Initiation interval of 4

#### 5-4. Perform design analysis by switching to the Analysis perspective and looking at the `dct_1d` performance view.

5-4-1. Switch to the Analysis perspective, expand the *Module Hierarchy* entries, and select the `dct_1d` entry.

5-4-2. Expand, if necessary, the **Profile** tab entries and notice that the `DCT_Outer_Loop` is now pipelined and there is no `DCT_Inner_Loop` entry.

Module Hierarchy								
	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type	
▲ ● dct	5	8	677	520	874	875	none	
▲ ● dct_dct_2d	3	8	616	381	741	741	none	
● <b>dct_dct_1d2</b>	0	8	539	165	36	36	none	

Performance Profile					
	Pipelined	Latency	Initiation Interval	Iteration Latency	Trip count
▲ ● dct_dct_1d2	-	36	36	-	-
● DCT_Outer_Loop	yes	34	4	7	8

##### (a) ZedBoard

Module Hierarchy								
	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type	
▲ ● dct	5	8	711	524	878	879	none	
▲ ● dct_dct_2d	3	8	636	383	743	743	none	
● <b>dct_dct_1d2</b>	0	8	539	165	36	36	none	

Performance Profile					
	Pipelined	Latency	Initiation Interval	Iteration Latency	Trip count
▲ ● dct_dct_1d2	-	36	36	-	-
● DCT_Outer_Loop	yes	34	4	7	8

##### (b) Zybo

Figure 21. DCT\_Outer\_Loop flattening

5-4-3. Select the `dct_1d` entry in the Module Hierarchy tab and observe that the `DCT_Outer_Loop` spans over eight states in the Performance view.



Current Module : `dct` > `dct dct 2d` > `dct dct 1d2`

	Operation\Control S...	C0	C1	C2	C3	C4	C5	C6	C7
1	tmp 21 read(read)								
2	tmp 2 read(read)								
3	tmp 10 ( )								
4	tmp 12 ( )								
5	tmp 14 ( )								
6	tmp 16 ( )								
7	tmp 18 ( )								
8	tmp 20 ( )								
9	tmp 23 ( )								
1...	+DCT Outer Loop								

Figure 22. The Performance view of the DCT\_Outer\_Loop function

- 5-4-4. Select the **Resource** tab, expand the *Memory Ports* entry and observe that the memory accesses on BRAM *src* are being used to the maximum in every clock cycle. (At most a BRAM can be dual-port and both ports are being used). This is a good indication the design may be bandwidth limited by the memory resource.

Current Module : `dct` > `dct dct 2d` > `dct dct 1d2`

	Resource\Control Step	C0	C1	C2	C3	C4	C5	C6	C7
1-6	+I/O Ports								
7	-Memory Ports								
8	src(p0)		read	read	read	read			
9	dct coeff tabl...		read						
10	dct coeff tabl...		read						
11	src(p1)		read	read	read	read			
12	dct coeff tabl...			read					
13	dct coeff tabl...			read					
14	dct coeff tabl...			read					
15	dct coeff tabl...			read					
16	dct coeff tabl...			read					
17	dct coeff tabl...			read					
18	dst(p0)								write
1...	+Expressions								

Performance **Resource**

Figure 23. The Resource tab

- 5-4-5. Switch to the *Synthesis* perspective.

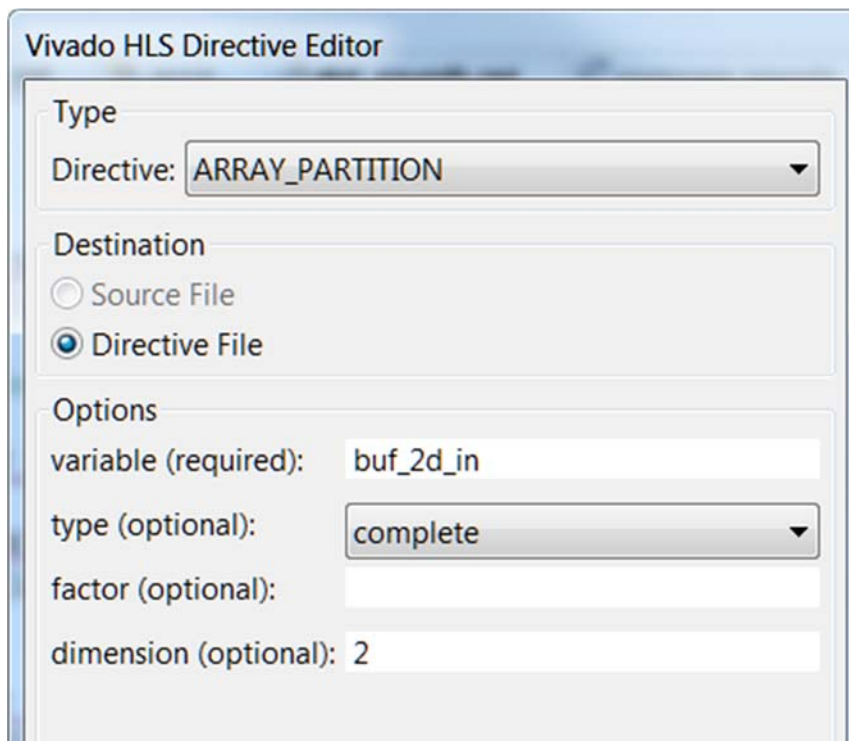
## Improve Memory Bandwidth

## Step 6

- 6-1. Create a new solution by copying the previous solution (Solution3) 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.



- 6-1-1.** Select **Project > New Solution** to create a new solution.
- 6-1-2.** A *Solution Configuration* dialog box will appear. Click the **Finish** button (with Solution3 selected).
- 6-1-3.** 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).
- 6-1-4.** A pop-up menu shows up listing various directives. Select **ARRAY\_PARTITION** directive.
- 6-1-5.** Make sure that the **type** is *complete*. Enter **2** in the *dimension* field and click **OK**.



**Figure 24. Applying ARRAY\_PARTITION directive to memory buffer**

- 6-1-6.** Similarly, apply the *ARRAY\_PARTITION* directive with dimension of 2 to the **col\_inbuf** array.
- 6-1-7.** Click on the **Synthesis** button.
- 6-1-8.** When the synthesis is completed, select **Project > Compare Reports...** to compare the two solutions.
- 6-1-9.** Select *Solution3* and *Solution4* from the **Available Reports**, and click on the **Add>>** button.
- 6-1-10.** Observe that the latency reduced from 874 to 508 clock cycles for ZedBoard (878 to 512 for Zybo).

**Performance Estimates**⊟ **Timing (ns)**

Clock		solution4	solution3
ap_clk	Target	10.00	10.00
	Estimated	10.79	9.40

⊟ **Latency (clock cycles)**

		solution4	solution3
Latency	min	508	874
	max	508	874
Interval	min	509	875
	max	509	875

**(a) ZedBoard****Performance Estimates**⊟ **Timing (ns)**

Clock		solution4	solution3
ap_clk	Target	8.00	8.00
	Estimated	9.40	9.40

⊟ **Latency (clock cycles)**

		solution4	solution3
Latency	min	512	878
	max	512	878
Interval	min	513	879
	max	513	879

**(b) Zybo****Figure 25. Performance comparison after array partitioning**

**6-1-11.** Scroll down in the comparison report to view the resources utilization. Observe the increase in the FF resource utilization (almost double).

**Utilization Estimates**

	solution4	solution3
BRAM_18K	3	5
DSP48E	8	8
FF	1243	677
LUT	634	520

**(a) ZedBoard****Utilization Estimates**

	solution4	solution3
BRAM_18K	3	5
DSP48E	8	8
FF	1284	711
LUT	638	524

**(b) Zybo****Figure 26. Resources utilization after array partitioning**

**6-1-12.** Expand the **Loop** entry in the **dct.rpt** entry and observe that the Pipeline II is now 1.

## **6-2. Perform resource analysis by switching to the Analysis perspective and looking at the dct resources profile view.**

**6-2-1.** Switch to the Analysis perspective, expand the Module Hierarchy entries, and select the *dct* entry.

**6-2-2.** Select the **Resource Profile** tab.

**6-2-3.** Expand the Memories and Expressions entries and observe that the most of the resources are consumed by instances. The *buf\_2d\_in* array is partitioned into multiple memories and most of the operations are done in addition and comparison.

Module Hierarchy								
	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type	
▲ dct	3	8	1243	634	508	509	none	
▶ dct_dct_2d	2	8	923	453	373	373	none	
▶ dct_read_data	0	0	28	54	66	66	none	

Performance Profile								
	BRAM	DSP	FF	LUT	Bits P0	Bits P1	Bits P2	Banks/Dep
▲ dct	3	8	1243	634				
▶ I/O Ports(2)					32			
▶ Instances(2)	2	8	951	507				
▲ Memories(9)	1		256	16	144			9
◆ buf_2d_out_U 1			0	0	16			1
◆ buf_2d_in_6_L 0			32	2	16			1
◆ buf_2d_in_5_L 0			32	2	16			1
◆ buf_2d_in_4_L 0			32	2	16			1
◆ buf_2d_in_3_L 0			32	2	16			1
◆ buf_2d_in_7_L 0			32	2	16			1
◆ buf_2d_in_2_L 0			32	2	16			1
◆ buf_2d_in_1_L 0			32	2	16			1
◆ buf_2d_in_0_L 0			32	2	16			1
▶ Σ Expressions(9)	0	0	0	42	36	41	8	
▶ 0000 Registers(11)			36		36			
▶ FIFO(0)	0		0	0	0			0
▶ Multiplexers(33)	0		0	69	69			0

(a) ZedBoard

Module Hierarchy								
	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type	
▲ dct	3	8	1284	638	512	513	none	
▶ dct_dct_2d	2	8	947	455	375	375	none	
▶ dct_read_data	0	0	38	55	67	67	none	

Performance Profile								
Resource Profile								
	BRAM	DSP	FF	LUT	Bits P0	Bits P1	Bits P2	Banks/De
▲ dct	3	8	1284	638				
▶ I/O Ports(2)					32			
▶ Instances(2)	2	8	985	510				
▲ Memories(9)	1		256	16	144			9
◆ buf_2d_out_U	1		0	0	16			1
◆ buf_2d_in_6_U	0		32	2	16			1
◆ buf_2d_in_5_U	0		32	2	16			1
◆ buf_2d_in_4_U	0		32	2	16			1
◆ buf_2d_in_3_U	0		32	2	16			1
◆ buf_2d_in_7_U	0		32	2	16			1
◆ buf_2d_in_2_U	0		32	2	16			1
◆ buf_2d_in_1_U	0		32	2	16			1
◆ buf_2d_in_0_U	0		32	2	16			1
▶ Expressions(9)	0	0	0	42	36	41	8	
▶ Registers(15)			43		43			
FIFO(0)	0		0	0	0			0
▶ Multiplexers(34)	0		0	70	70			0

(b) Zybo

Figure 27. Resource profile after partitioning buffers

6-2-4. Switch to the *Synthesis* perspective.

## Apply DATAFLOW Directive

## Step 7

**7-1. Create a new solution by copying the previous solution (Solution4) settings. Apply the DATAFLOW directive to improve the throughput. Generate the solution and analyze the output.**

**7-1-1.** Select **Project > New Solution**.

**7-1-2.** A *Solution Configuration* dialog box will appear. Click the **Finish** button (with Solution4 selected).

**7-1-3.** Close all inactive solution windows by selecting **Project > Close Inactive Solution Tabs**.

**7-1-4.** Select function **dct** in the directives pane, right-click on it and select *Insert Directive...*

**7-1-5.** Select **DATAFLOW** directive to improve the throughput.

7-1-6. Click on the **Synthesis** button.

7-1-7. When the synthesis is completed, the synthesis report is automatically opened.

7-1-8. Observe that dataflow type pipeline throughput is listed in the Performance Estimates.

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	10.79	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
507	507	374	374	dataflow

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	8.00	9.40	1.00

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
511	511	376	376	dataflow

(a) ZedBoard

(b) Zybo

(a) ZedBoard

(b) Zybo

**Figure 28. 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 current 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 top-level of the design will get benefit from dataflow optimization.

7-1-9. Scrolling down into the Area Estimates, observe that the number of BRAM\_18K required at the top-level has increased from 3 to 4.

Utilization Estimates					Utilization Estimates				
Summary					Summary				
Name	BRAM_18K	DSP48E	FF	LUT	Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-	DSP	-	-	-	-
Expression	-	-	0	1	Expression	-	-	0	1
FIFO	-	-	-	-	FIFO	-	-	-	-
Instance	2	8	985	572	Instance	2	8	1026	576
Memory	2	-	512	32	Memory	2	-	512	32
Multiplexer	-	-	-	16	Multiplexer	-	-	-	16
Register	-	-	12	-	Register	-	-	12	-
Total	4	8	1509	621	Total	4	8	1550	625
Available	280	220	106400	53200	Available	120	80	35200	17600
Utilization (%)	1	3	1	1	Utilization (%)	3	10	4	3

(a) ZedBoard

(b) Zybo

**Figure 29. Resource estimate with DATAFLOW directive applied**

7-1-10. Look at the console view and notice that `dct_coeff_table` is automatically partitioned in dimension 2. The `buf_2d_in` and `col_inbuf` arrays are partitioned as we had applied the directive in the

previous run. The dataflow is applied at the top-level which created channels between top-level functions read\_data, dct\_2d, and write\_data.

```
@I [HLS-10] Checking synthesizability ...
@I [XFORM-502] Unrolling all sub-loops inside loop 'DCT_Outer_Loop' (dct.c:13) in function 'dct_1d' for pipelining.
@I [XFORM-501] Unrolling loop 'DCT_Inner_Loop' (dct.c:15) in function 'dct_1d' completely.
@I [XFORM-102] Partitioning array 'dct_coeff_table' in dimension 2 automatically.
@I [XFORM-101] Partitioning array 'buf_2d_in' (dct.c:81) in dimension 2 completely.
@I [XFORM-101] Partitioning array 'col_inbuf' (dct.c:27) in dimension 2 completely.
@I [XFORM-712] Applying dataflow to function 'dct' (dct.c:78), detected/extracted 3 process function(s):
    'read_data'
    'dct_2d'
    'write_data'.
@I [XFORM-11] Balancing expressions in function 'dct_1d' (dct.c:4)...8 expression(s) balanced.
@I [XFORM-541] Flattening a loop nest 'WR_Loop_Row' (dct.c:71:67) in function 'write_data'.
@I [XFORM-541] Flattening a loop nest 'Xpose_Row_Outer_Loop' (dct.c:38:2) in function 'dct_2d'.
@I [XFORM-541] Flattening a loop nest 'Xpose_Col_Outer_Loop' (dct.c:49:2) in function 'dct_2d'.
@I [XFORM-541] Flattening a loop nest 'RD_Loop_Row' (dct.c:59:67) in function 'read_data'.
@I [HLS-111] Elapsed time: 4.52 seconds; current memory usage: 89.2 MB.
@I [HLS-10] Starting hardware synthesis ...
@I [HLS-10] Synthesizing 'dct' ...
```

**Figure 30. Console view of synthesis process after DATAFLOW directive applied**

## 7-2. Perform performance analysis by switching to the Analysis perspective and looking at the dct performance profile view.

**7-2-1.** Switch to the Analysis perspective, expand the Module Hierarchy entries, and select the dct\_2d entry.

**7-2-2.** Select the Performance Profile tab.

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. From the Performance Profile tab it can be seen that dct\_2d is not completely operating in parallel as Row\_DCT\_Loop and Col\_DCT\_Loop were not pipelined.

Module Hierarchy								
	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type	
▲ ● dct	4	8	1509	621	507	374	dataflow	
● dct_read_data	0	0	29	55	66	66	none	
▶ ● <b>dct_dct_2d</b>	2	8	924	454	373	373	none	
● dct_write_data	0	0	32	63	66	66	none	

Performance Profile				Resource Profile			
	Pipelined	Latency	Initiation Interval				
▲ ● <b>dct_dct_2d</b>	-	373	373				
● Row_DCT_Loop	no	120	-				
● Xpose_Row_Outer_Loop_Xpose_Row_Inner_Loop	yes	64	1				
● Col_DCT_Loop	no	120	-				
● Xpose_Col_Outer_Loop_Xpose_Col_Inner_Loop	yes	64	1				

**(a) ZedBoard**

Module Hierarchy								
	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type	
▲ • dct	4	8	1550	625	511	376	dataflow	
• dct_read_data	0	0	39	56	67	67	none	
▶ • <b>dct_dct_2d</b>	2	8	948	456	375	375	none	
• dct_write_data	0	0	39	64	67	67	none	

Performance Profile				Resource Profile			
	Pipelined	Latency	Initiation Interv:				
▲ • <b>dct_dct_2d</b>	-	375	375				
• Row_DCT_Loop	no	120	-				
• Xpose_Row_Outer_Loop_Xpose_Row_Inner_Loop	yes	65	1				
• Col_DCT_Loop	no	120	-				
• Xpose_Col_Outer_Loop_Xpose_Col_Inner_Loop	yes	65	1				

**(b) Zybo****Figure 31. 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.

7-2-3. Switch to the *Synthesis* perspective.

**Apply INLINE Directive****Step 8**

**8-1. Create a new solution by copying the previous solution (Solution5) settings. Apply INLINE directive to `dct_2d`. Generate the solution and analyze the output.**

**8-1-1.** Select **Project > New Solution**.

**8-1-2.** A *Solution Configuration* dialog box will appear. Click the **Finish** button (with Solution5 selected).

**8-1-3.** Select the function **dct\_2d** in the directives pane, right-click on it and select *Insert Directive...*

**8-1-4.** 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.

**8-1-5.** Click on the **Synthesis** button.

**8-1-6.** When the synthesis is completed, the synthesis report will be opened.



**8-1-7.** Observe that the latency reduced from 507 to 479 clock cycles for ZedBoard (511 to 483 clock cycles for Zybo), and the Dataflow pipeline throughput drastically reduced from 374 to 106 clock cycles (376 to 106 clock cycles for Zybo).

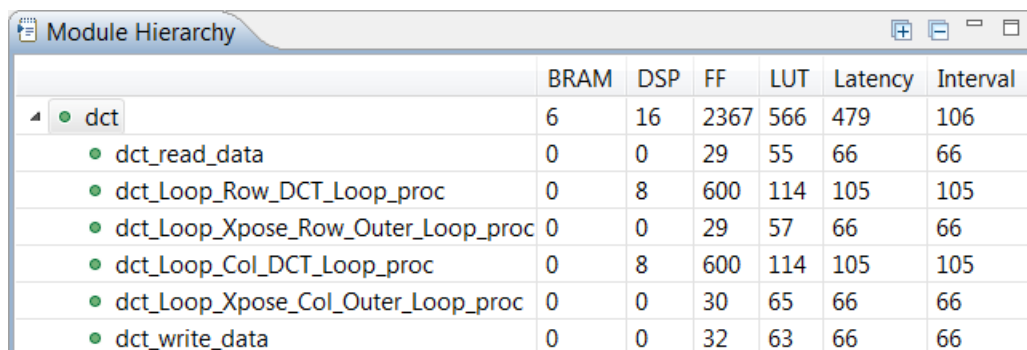
**8-1-8.** Examine the synthesis log to see what transformations were applied automatically.

- 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: Vivado HLS will seek to minimize the number of clocks, even if it means increasing the area.
  - BRAM usage has increased once again (from 4 to 6), due to ping-pong buffering between more dataflow processes.
- ```
@I [HLS-10] Starting code transformations ...
@I [XFORM-603] Inlining function 'dct_2d' into 'dct' (dct.c:87).
@I [HLS-10] Checking synthesizability ...
@I [XFORM-502] Unrolling all sub-loops inside loop 'DCT_Outer_Loop' (dct.c:13) in function 'dct_1d' for pipelining.
@I [XFORM-501] Unrolling loop 'DCT_Inner_Loop' (dct.c:15) in function 'dct_1d' completely.
@I [XFORM-102] Partitioning array 'dct_coeff_table' in dimension 2 automatically.
@I [XFORM-101] Partitioning array 'buf_2d_in' (dct.c:81) in dimension 2 completely.
@I [XFORM-101] Partitioning array 'col_inbuf' (dct.c:27) in dimension 2 completely.
@I [XFORM-721] Changing loop 'Loop_Row_DCT_Loop_proc' (dct.c:32) to a process function for dataflow in function 'dct'.
@I [XFORM-721] Changing loop 'Loop_Xpose_Row_Outer_Loop_proc' (dct.c:38) to a process function for dataflow in function 'dct'.
@I [XFORM-721] Changing loop 'Loop_Col_DCT_Loop_proc' (dct.c:43) to a process function for dataflow in function 'dct'.
@I [XFORM-721] Changing loop 'Loop_Xpose_Col_Outer_Loop_proc' (dct.c:49) to a process function for dataflow in function 'dct'.
@I [XFORM-712] Applying dataflow to function 'dct' (dct.c:78), detected/extracted 6 process function(s):
    'read_data'
    'Loop_Row_DCT_Loop_proc'
    'Loop_Xpose_Row_Outer_Loop_proc'
    'Loop_Col_DCT_Loop_proc'
    'Loop_Xpose_Col_Outer_Loop_proc'
    'write_data'.
@I [XFORM-602] Inlining function 'dct_1d' into 'Loop_Row_DCT_Loop_proc' (dct.c:33->dct.c:87) automatically.
@I [XFORM-602] Inlining function 'dct_1d' into 'Loop_Col_DCT_Loop_proc' (dct.c:44->dct.c:87) automatically.
@I [XFORM-11] Balancing expressions in function 'Loop_Row_DCT_Loop_proc' (dct.c:13:61)...8 expression(s) balanced.
@I [XFORM-11] Balancing expressions in function 'Loop_Col_DCT_Loop_proc' (dct.c:13:61)...8 expression(s) balanced.
```

**Figure 32. Console view after INLINE directive applied to `dct_2d`**

**8-1-9.** Switch to the Analysis perspective, expand the Module Hierarchy entries, and select the `dct` entry.

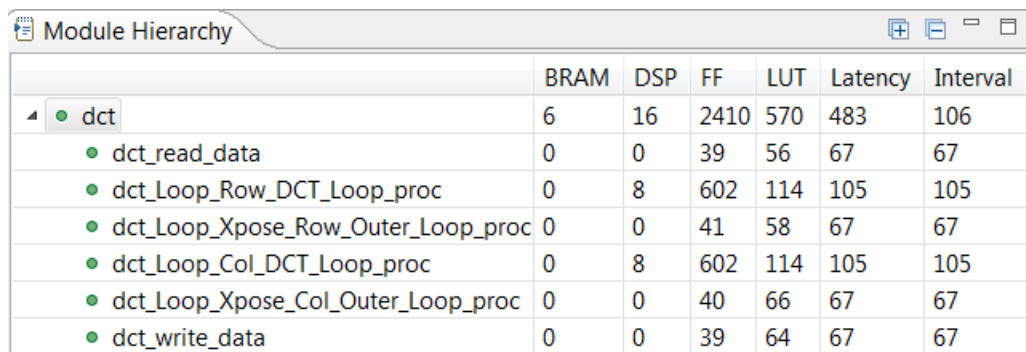
Observe that the `dct_2d` entry is now replaced with `dct_Loop_Row_DCT_Loop_proc`, `dct_Loop_Xpose_Row_Outer_Loop_proc`, `dct_Loop_Col_DCT_Loop_proc`, and `dct_Loop_Xpose_Col_Outer_Loop_proc` since the `dct_2d` function is inlined. Also observe that all the functions are operating in parallel, yielding the top-level function interval (throughput) of 106 clock cycles.



|                                                   | BRAM | DSP | FF   | LUT | Latency | Interval |
|---------------------------------------------------|------|-----|------|-----|---------|----------|
| ▲ ● <b>dct</b>                                    | 6    | 16  | 2367 | 566 | 479     | 106      |
| ● <code>dct_read_data</code>                      | 0    | 0   | 29   | 55  | 66      | 66       |
| ● <code>dct_Loop_Row_DCT_Loop_proc</code>         | 0    | 8   | 600  | 114 | 105     | 105      |
| ● <code>dct_Loop_Xpose_Row_Outer_Loop_proc</code> | 0    | 0   | 29   | 57  | 66      | 66       |
| ● <code>dct_Loop_Col_DCT_Loop_proc</code>         | 0    | 8   | 600  | 114 | 105     | 105      |
| ● <code>dct_Loop_Xpose_Col_Outer_Loop_proc</code> | 0    | 0   | 30   | 65  | 66      | 66       |
| ● <code>dct_write_data</code>                     | 0    | 0   | 32   | 63  | 66      | 66       |

**(a) ZedBoard**





|                                    | BRAM | DSP | FF   | LUT | Latency | Interval |
|------------------------------------|------|-----|------|-----|---------|----------|
| dct                                | 6    | 16  | 2410 | 570 | 483     | 106      |
| dct_read_data                      | 0    | 0   | 39   | 56  | 67      | 67       |
| dct_Loop_Row_DCT_Loop_proc         | 0    | 8   | 602  | 114 | 105     | 105      |
| dct_Loop_Xpose_Row_Outer_Loop_proc | 0    | 0   | 41   | 58  | 67      | 67       |
| dct_Loop_Col_DCT_Loop_proc         | 0    | 8   | 602  | 114 | 105     | 105      |
| dct_Loop_Xpose_Col_Outer_Loop_proc | 0    | 0   | 40   | 66  | 67      | 67       |
| dct_write_data                     | 0    | 0   | 39   | 64  | 67      | 67       |

(b) Zybo

Figure 33. Performance analysis after the **INLINE** directive

8-1-10. Switch to the *Synthesis* perspective.

## Apply RESHAPE Directive

## Step 9

**9-1. Create a new solution by copying the previous solution (Solution6) settings. Apply the RESHAPE directive. Generate the solution and understand the output.**

**9-1-1.** Select **Project > New Solution**.

**9-1-2.** A *Solution Configuration* dialog box will appear. Click the **Finish** button (with Solution6 selected).

**9-1-3.** Select **PARTITION** directive applied to the **buf\_2d\_in** array of the **dct** function in the Directive pane, right-click, and select **Modify Directive**. Select **ARRAY\_RESHAPE** directive, enter 2 as the dimension, and click **OK**.

**9-1-4.** Similarly, change **PARTITION** directive applied to the **col\_inbuf** array of the **dct\_2d** function in the Directive pane, to **ARRAY\_RESHAPE** with the dimension of **2**.

**9-1-5.** Assign the **ARRAY\_RESHAPE** directive with dimension of 2 to the **dct\_coeff\_table** array.

```

└─ dct_1d
    └─ [1] dct_coeff_table
        └─ % HLS ARRAY_RESHAPE variable=dct_coeff_table complete dim=2
            └─ DCT_Outer_Loop
└─ dct_2d
    └─ % HLS INLINE
    └─ [1] row_outbuf
    └─ [1] col_outbuf
    └─ [1] col_inbuf
        └─ % HLS ARRAY_RESHAPE variable=col_inbuf complete dim=2
            └─ Row_DCT_Loop
                └─ Xpose_Row_Outer_Loop
                    └─ Col_DCT_Loop
                        └─ Xpose_Col_Outer_Loop
└─ read_data
└─ write_data
└─ dct
    └─ % HLS DATAFLOW
    └─ input
    └─ output
    └─ [1] buf_2d_in
        └─ % HLS ARRAY_RESHAPE variable=buf_2d_in complete dim=2
            └─ [1] buf_2d_out

```

Figure 34. RESHAPE directive applied

9-1-6. Click on the **Synthesis** button.

9-1-7. When the synthesis is completed, the synthesis report is automatically opened.

9-1-8. Observe that both latency (increased from 479 to 607 for ZedBoard and from 483 to 611 for Zybo) and Dataflow pipeline throughput (increased from 106 to 131 for ZedBoard and 106 to 132 for Zybo) has regressed. The BRAM resource utilization increased from 6 to 22 for both ZedBoard and Zybo.

- Reviewing the synthesis log will provide some clues. There are warnings in the scheduling phase for read\_data stating that II=1 could not be achieved. In fact, read\_data complains about the conflict of read and write operations.
- The problem here is due to the fact that an update to a single element in a reshaped array requires that the entire word be read, the single element updated and the entire word written back: an array that has been reshaped requires a read-modify-write cycle (Vivado HLS does not implement byte-masking on writes).
- This operation negatively impacts the maximum write bandwidth for such an array.

9-1-9. Thus it can be seen the directives have to be applied carefully.

9-1-10. Close Vivado 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 RESHAPE directive will allow multiple accesses to BRAM, however, care should be taken if a single element requires modification as it will result in read-modify-write operation for the entire word. The Analysis perspective and console logs can provide insight on what is going on.

## Answers

1. Answer the following questions for dct:

|                         |                          |
|-------------------------|--------------------------|
| Estimated clock period: | <u>6.38 ns</u>           |
| Worst case latency:     | <u>3959 clock cycles</u> |
| Number of DSP48E used:  | <u>1</u>                 |
| Number of BRAMs used:   | <u>5</u>                 |
| Number of FFs used:     | <u>278</u>               |
| Number of LUTs used:    | <u>354</u>               |