Fine-Tuning with Vivado HLS

Introduction

This lab introduces various techniques and directives of Vivado HLS which can be used in SDx to improve design performance. The design under consideration performs a discrete cosine transformation (DCT) on an 8x8 block of data.

Objectives

After completing this lab, you will be able to:

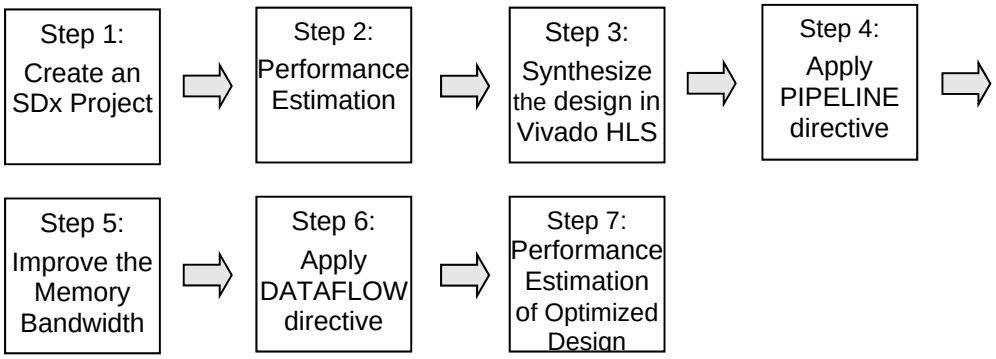
- Improve performance using the PIPELINE directive
- Understand the DATAFLOW directive functionality
- Apply memory partitioning techniques to improve data access

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 7 primary steps: You will create an SDx project with the provided dct example, carry out the performance estimation of the design which estimates the acceleration of the dct function in hardware, invoke Vivado HLS and resynthesize, apply the PIPELINE directive to improve performance, improve the memory bandwidth by applying the PARTITION directive, apply the DATAFLOW directive, and finally carry out the performance estimation of the improved model.

General Flow for this Lab



Create an SDx Project Step 1

1-1. Launch SDx and create a project, called *lab6*, using the *Empty Application* template and then using the provided source files, targeting the Zed or Zybo board and Standalone OS.

1-1-1. Open SDx, select **c:\xup\SDSoC\labs** as the workspace and click **OK**.

1-1-2. Create a new project called **lab6** targeting either *zybo* or *zed* board and *Standalone* OS, *Empty Application* template.

1-2. Import the provided source files from source\lab6\src folder.

1-2-1. Right click on *src* under **lab6** in the Project Explorer tab and select **Import…**

1-2-2. Click on **File System** under *General category* and then click **Next**.

1-2-3. Click on the **Browse** button, browse to *c:\xup\SDSoC\source\lab6\src* folder, and click **OK**.

1-2-4. Either select all the files in the right-side window or select *src* checkbox in the left-side window and click **Finish** to import the files into the project.

Performance Estimation Step 2

2-1. Mark *dct* for the hardware acceleration. Run an initial performance estimate of the hardware only.

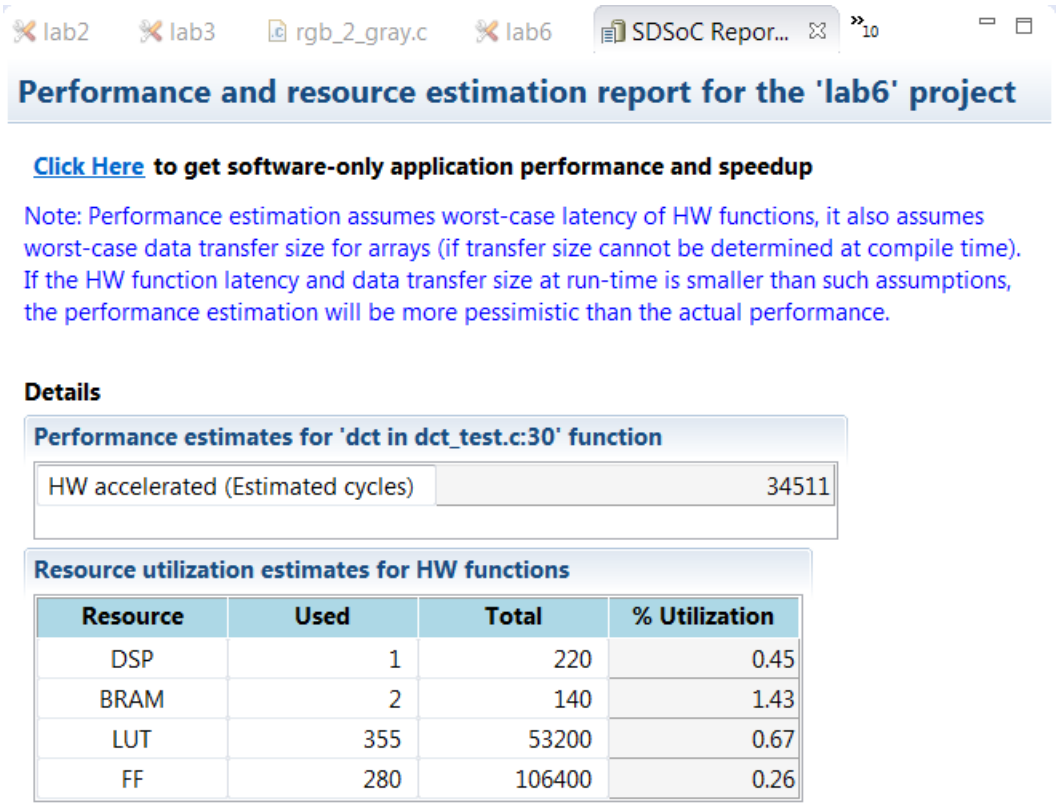
2-1-1. Add the **dct** function operating at 100 MHz in the *HW Function* pane.

2-1-2. In the *Options* panel of the *SDx Project Overview*, click on the *Estimate Performance* checkbox.

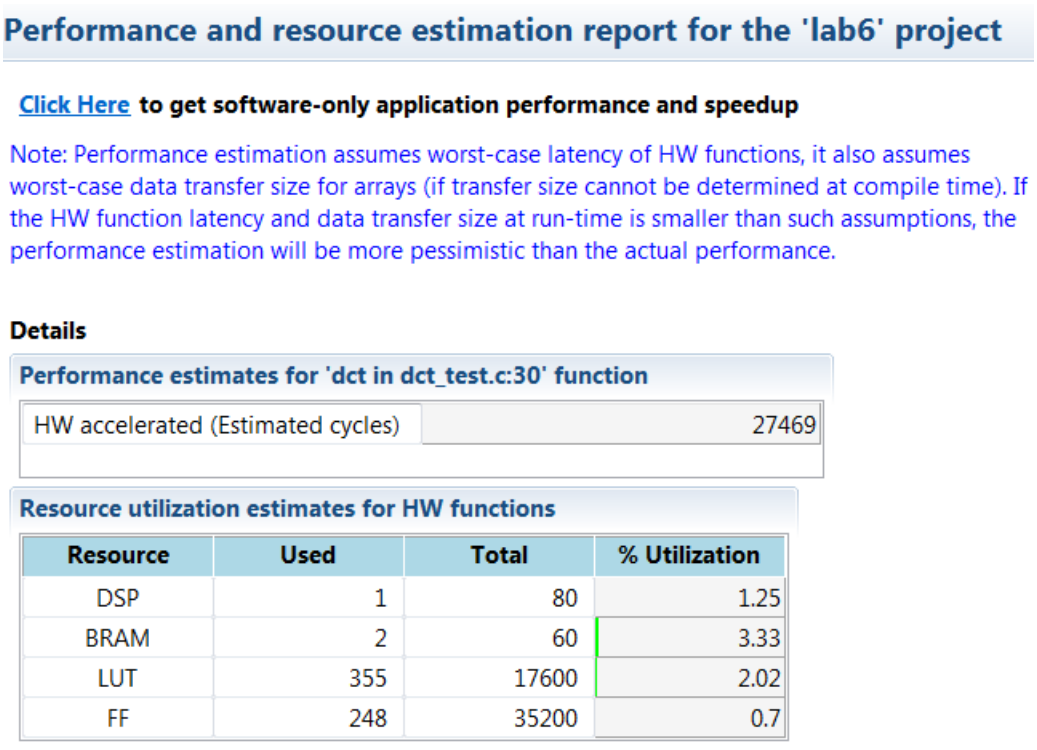
2-1-3. Set the build configuration to *Debug* and then build the project.

2-1-4. After the build is over, you can see an initial report. This report contains a hardware-only estimate summary and has a link that can be clicked to obtain the software run data, which updates the report with comparison of hardware implementation versus the software-only information.

Note that the hardware accelerator performs the function in approximately 32000 clock cycles (estimated) for Zybo, or 34500 clock cycles (estimated) for Zed.



(a) Zed



(b) Zybo

Figure 1. Initial estimate of hardware only performance

Synthesize the Design Using Vivado HLS Step 3

3-1. Analyze the source (*dct.c*) code.

3-1-1. 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 a 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 which write the result. The dct_2d function, defined at line 23, calls dct_1d function and performs a transpose.

Finally, the dct_1d function, defined at line 4, uses dct_coeff_table and implements a basic iterative form of the 1D Type-II DCT algorithm. The following figure shows the function hierarchy on the left-hand side, and the loops in the order they are executed, and the flow of data, on the right-hand side.

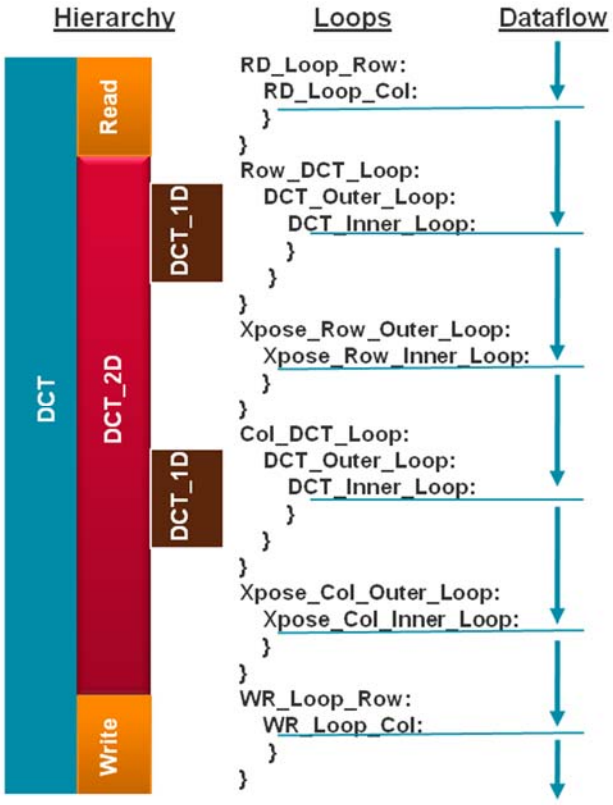


Figure 3. Design hierarchy and dataflow

3-2. Invoke Vivado HLS. Synthesize the design with the defaults. View the synthesis results.

3-2-1. In the *lab6 SDx Project Settings* pane, click on ( ) to invoke the Vivado HLS program.

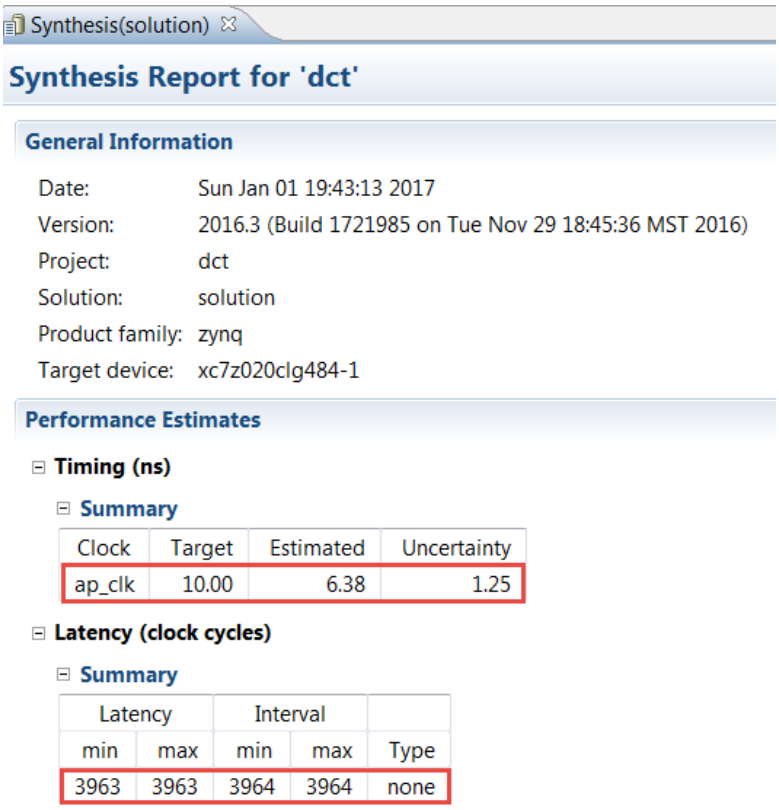
The HLS launcher dialog box will appear.

3-2-2. Click **OK**.

3-2-3. In Vivado HLS, select **Solution > Run C Synthesis > Active Solution** or click on the button to start the synthesis process.

This is just to view the log file.

3-2-4. When synthesis is completed, the results will be displayed in the **Synthesis(Solution)** tab.



Synthesis(solution)

Synthesis Report for 'dct'

General Information

Date:	Sun Jan 01 19:43:13 2017
Version:	2016.3 (Build 1721985 on Tue Nov 29 18:45:36 MST 2016)
Project:	dct
Solution:	solution
Product family:	zynq
Target device:	xc7z020clg484-1

Performance Estimates

Timing (ns)

Summary

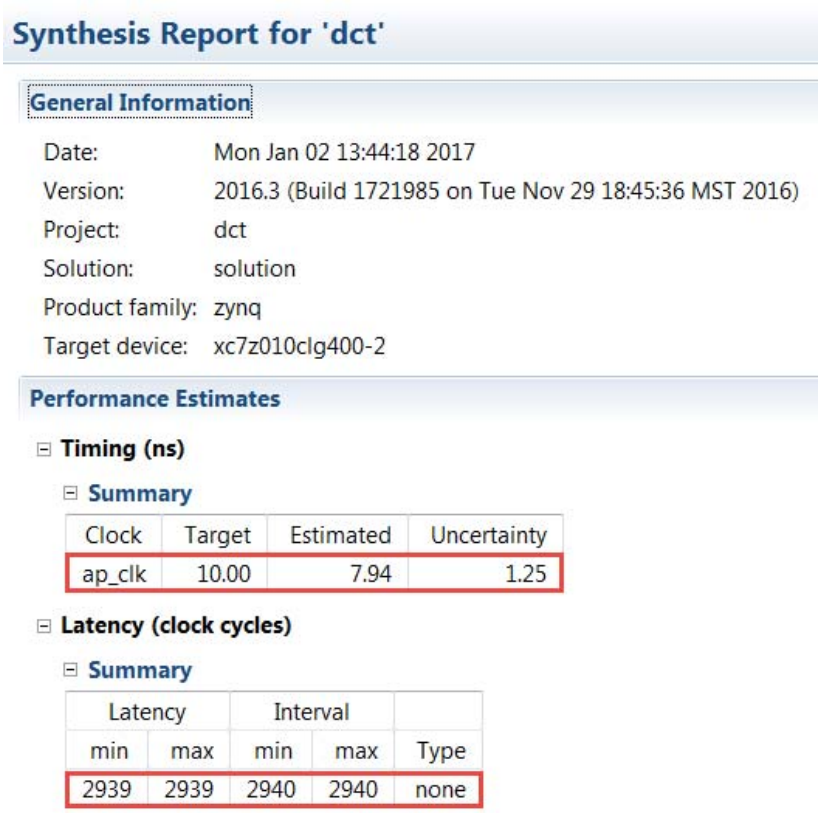
Clock	Target	Estimated	Uncertainty
ap_clk	10.00	6.38	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
3963	3963	3964	3964	none

(a) Zed



Synthesis Report for 'dct'

General Information

Date:	Mon Jan 02 13:44:18 2017
Version:	2016.3 (Build 1721985 on Tue Nov 29 18:45:36 MST 2016)
Project:	dct
Solution:	solution
Product family:	zynq
Target device:	xc7z010clg400-2

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	7.94	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
2939	2939	2940	2940	none

(b) Zybo

Figure 4. Synthesis report showing performance estimate

Notice that the estimated period is 6.38 ns for Zed whereas 7.94 ns for Zybo. For Zed, the latency is 3963 clocks and the interval is 3964. For Zybo, the latency is 2939 clocks and the interval is 2940. The Type is none since no pipeline was implemented.

3-2-5. Expand the **Source** folder and double-click on the *dct.c* to view the source file.

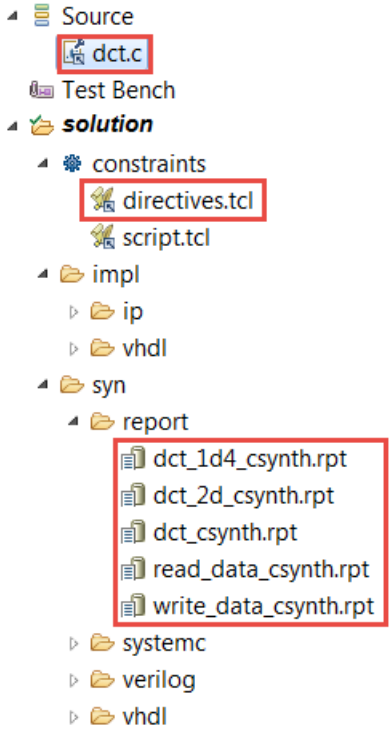


Figure 5. Project Explorer view

Note that the Synthesis Report section in the Explorer view shows dct_1d.rpt, dct_2d.rpt, dct_csynth.rpt, read_data, and write_data entries.

3-2-6. Double-click on the **directives.tcl** entry and examine its content.

Notice that input and output ports are using single-port block RAM (RAM_1P), and the desired latency is 1. You can verify this by selecting the *dct.c* tab and looking at the **Directive** tab. Also notice the "%" for the directives which indicate that they are passed via the *directives.tcl* file. Pragmas in the source code are indicated with a "#".

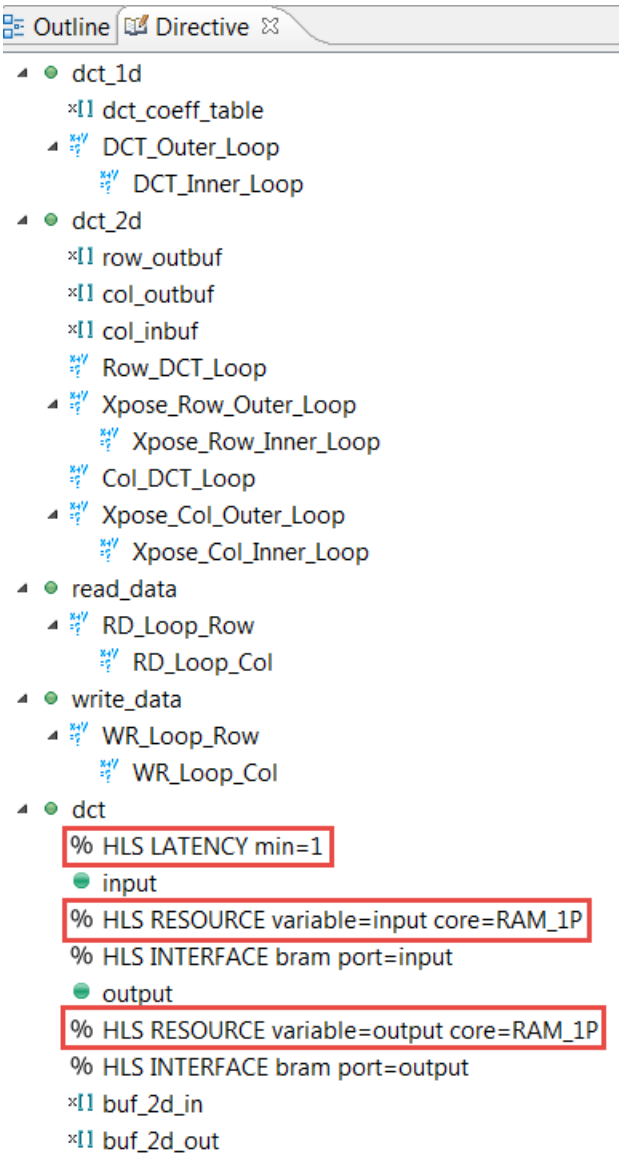
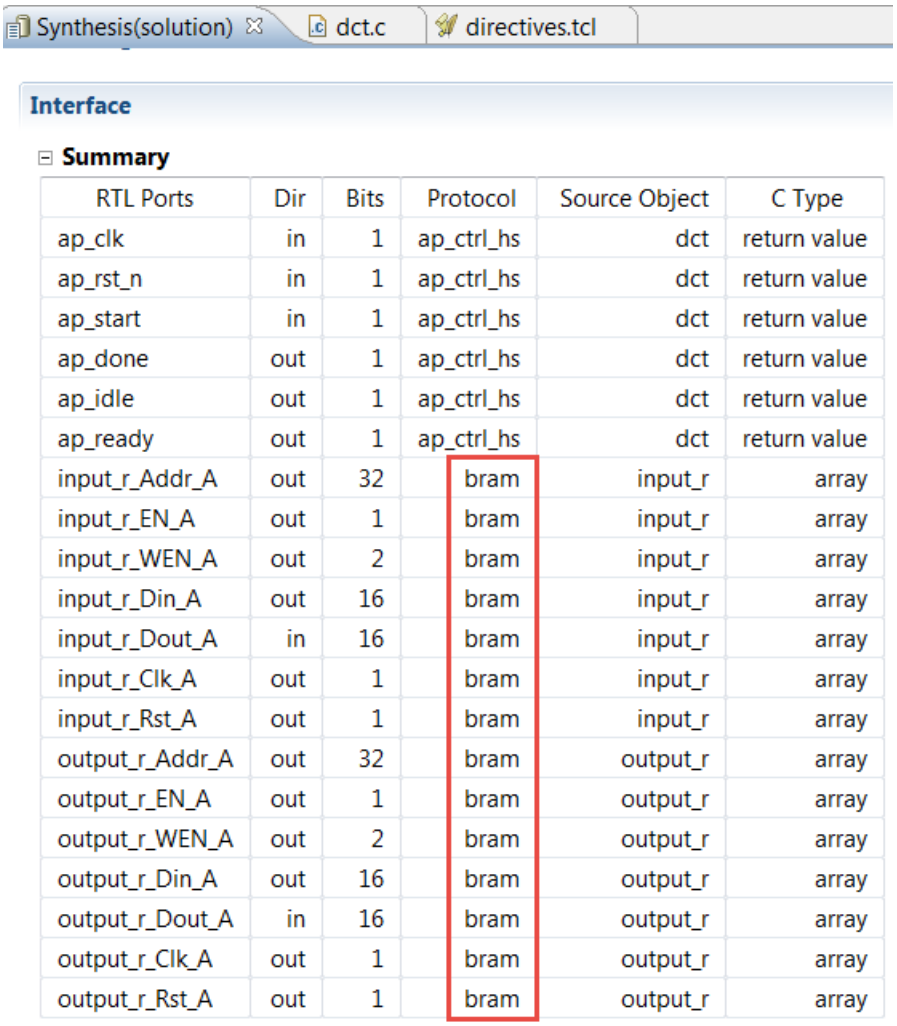


Figure 6. The Directive tab showing the directives passed from SDSoC

3-2-7. Select the *Synthesis(solution)* tab and then click on the **Interface** entry under the *Outline* tab.

The interface summary table will be displayed. It shows the six handshaking signals (ap_clk, …, ap_ready) and then shows the single port bram ports for the input and output parameters.



RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	dct	return value
ap_rst_n	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_Addr_A	out	32	bram	input_r	array
input_r_EN_A	out	1	bram	input_r	array
input_r_WEN_A	out	2	bram	input_r	array
input_r_Din_A	out	16	bram	input_r	array
input_r_Dout_A	in	16	bram	input_r	array
input_r_Clk_A	out	1	bram	input_r	array
input_r_Rst_A	out	1	bram	input_r	array
output_r_Addr_A	out	32	bram	output_r	array
output_r_EN_A	out	1	bram	output_r	array
output_r_WEN_A	out	2	bram	output_r	array
output_r_Din_A	out	16	bram	output_r	array
output_r_Dout_A	in	16	bram	output_r	array
output_r_Clk_A	out	1	bram	output_r	array
output_r_Rst_A	out	1	bram	output_r	array

Figure 7. Interface summary showing single port bram interface for the input and output

3-2-8. Scroll through the Console tab to see the synthesis process log.

3-3. Create a new solution (solution1) copying the directives from the exisitng solution (solution). Synthesize the design again. View the synthesis results.

3-3-1. Select **Project > New Solution** or click on ( ) from the tools bar buttons.

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

3-3-3. With the source code file in focus, in the directives tab, under *dct*, right-click on the HLS INTERFACE directive of the *input* port, select **Remove Directive**.

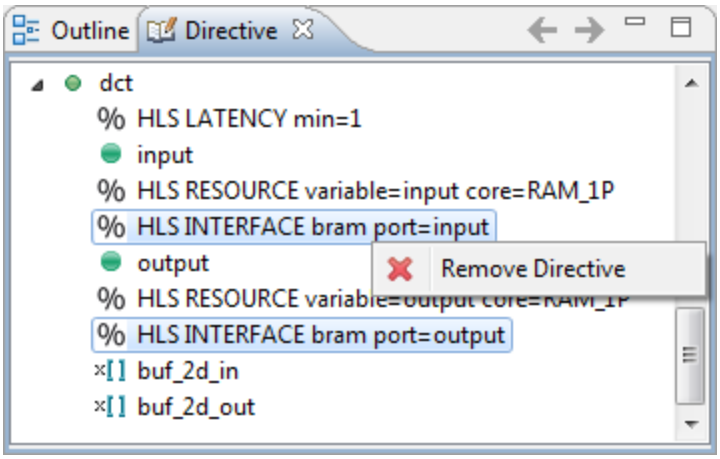


Figure 8. Remove HLS Interface directive

3-3-4. Similarly, remove the HLS INTERFACE directive of the *output* port.

3-3-5. Click on the  button to start the synthesis process.

3-3-6. When synthesis is completed, the results will be displayed in the **Synthesis(Solution1)** tab.

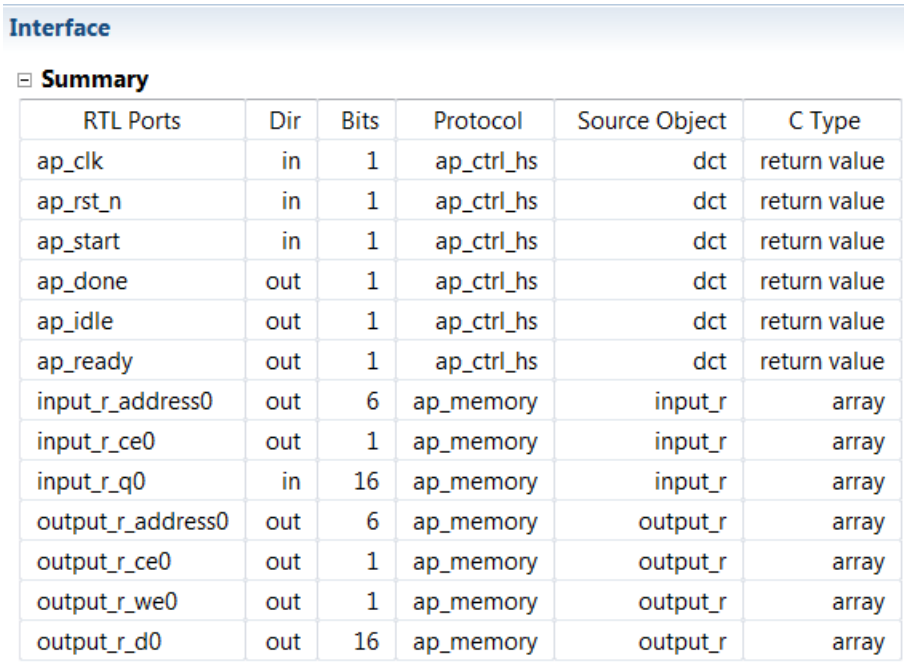
Notice that the performance estimations have changed slightly.

Note that the Synthesis Report section (under Solution1) in the Explorer view now 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.

```
INFO: [XFORM 203-602] Inlining function 'read_data' into 'dct' (../../../src/dct.c:85) automatically.
INFO: [XFORM 203-602] Inlining function 'write_data' into 'dct' (../../../src/dct.c:90) automatically.
INFO: [HLS 200-111] Finished Checking Synthesizability Time (s): cpu = 00:00:01 ; elapsed = 00:00:05 .
INFO: [XFORM 203-602] Inlining function 'read_data' into 'dct' (../../../src/dct.c:85) automatically.
INFO: [XFORM 203-602] Inlining function 'write_data' into 'dct' (../../../src/dct.c:90) automatically.
INFO: [HLS 200-111] Finished Pre-synthesis Time (s): cpu = 00:00:01 ; elapsed = 00:00:05 . Memory (MB)
```

Figure 9. Inlining of read_data and write_data functions

3-3-7. 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_n	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 10. Generated interface signals

The top-level function has input and output arrays without the HLS interface of *bram* directive. An *ap_memory* interface is generated for each of them instead.

3-3-8. Open dct_1d2.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_1d2` report indicates that the latency is 209 clock cycles ((24+2)*8+1) for ZedBoard.

Apply PIPELINE Directive — Step 4

4-1. 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.

4-1-1. Select **Project > New Solution** or click on ( ) from the tools bar buttons.

4-1-2. A *Solution Configuration* dialog box will appear. Click the **Finish** button (with *copy from* Solution1 selected).

4-1-3. Make sure that the **dct.c** source is opened in the information pane and click on the **Directive** tab.

4-1-4. Select **DCT_Inner_Loop** of the **dct_1d** function in the Directive pane, right-click on it and select *Insert Directive...*

4-1-5. A pop-up menu shows up listing various directives. Select the **PIPELINE** directive.

4-1-6. Leave **II** (Initiation Interval) blank as Vivado HLS will attempt to schedule the design with II = 1; one new input every clock cycle.

4-1-7. Click **OK**.

4-1-8. 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.

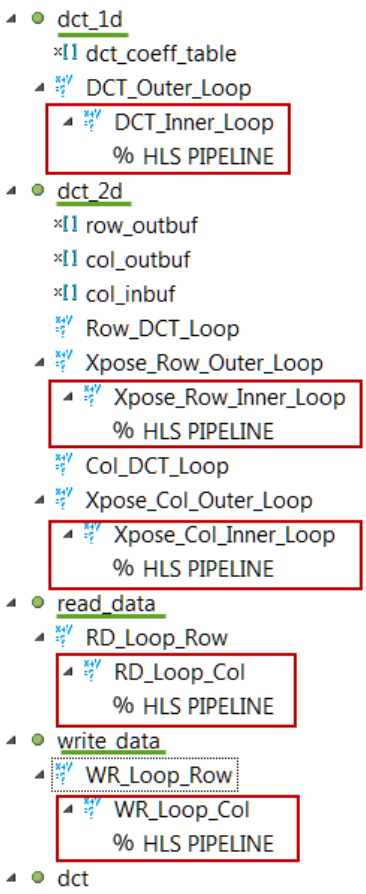


Figure 11. PIPELINE directive applied

4-1-9. Click on the **Synthesis** button.

4-1-10. When the synthesis is completed, select **Project > Compare Reports…** or click on  to compare the two solutions.

4-1-11. Select *Solution1* and *Solution2* from the **Available Reports,** click on the **Add>>** button, and then click **OK**.

4-1-12. Observe that the latency has reduced from 3959 to 1851 clock cycles for Zed or 2935 to 1723 clock cycles for 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	1851	3959
	max	1851	3959
Interval	min	1852	3960
	max	1852	3960

(a) Zed

Performance Estimates

Timing (ns)

Clock		solution2	solution1
ap_clk	Target	10.00	10.00
	Estimated	7.94	7.94

Latency (clock cycles)

		solution2	solution1
Latency	min	1723	2935
	max	1723	2935
Interval	min	1724	2936
	max	1724	2936

(b) Zybo

Figure 12. Performance comparison after pipelining

4-1-13. Scroll down in the comparison report to view the resources utilization. Observe that the LUTs utilization increased whereas BRAM and DSP48E remained same.

Utilization Estimates

	solution2	solution1
BRAM_18K	5	5
DSP48E	1	1
FF	256	272
LUT	462	353

(a) Zed

Utilization Estimates

	solution2	solution1
BRAM_18K	5	5
DSP48E	1	1
FF	223	240
LUT	461	353

(b) Zybo

Figure 13. Resources utilization after pipelining

4-2. Open the Analysis perspective and determine where most of the clock cycles are spent, i.e. where are the large latencies.

4-2-1. Click on the *Analysis* perspective button ( ).

4-2-2. 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. The new names are formed by appending the inner loop name to the outer loop name. You can also verify this by looking in the Console view message. Notice that the DCT_Outer_Loop could not be flattened.

```
@I [HLS-10] Checking synthesizability ...
@I [XFORM-602] Inlining function 'read_data' into 'dct' (../../../src/dct.c:85) automatically.
@I [XFORM-602] Inlining function 'write_data' into 'dct' (../../../src/dct.c:90) automatically.
@I [XFORM-602] Inlining function 'read_data' into 'dct' (../../../src/dct.c:85) automatically.
@I [XFORM-602] Inlining function 'write_data' into 'dct' (../../../src/dct.c:90) automatically.
@I [XFORM-541] Flattening a loop nest 'RD_Loop_Row' (../../../src/dct.c:59:67) in function 'dct'.
@I [XFORM-541] Flattening a loop nest 'WR_Loop_Row' (../../../src/dct.c:71:67) in function 'dct'.
@I [XFORM-541] Flattening a loop nest 'Xpose_Row_Outer_Loop' (../../../src/dct.c:38:1) in function 'dct_2d'.
@I [XFORM-541] Flattening a loop nest 'Xpose_Col_Outer_Loop' (../../../src/dct.c:49:1) in function 'dct_2d'.
@W [XFORM-542] Cannot flatten a loop nest 'DCT_Outer_Loop' (../../../src/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 [HLS-111] Elapsed time: 6.073 seconds; current memory usage: 90.2 MB.
```

Figure 14. The console view content indicating loops flattening

Module Hierarchy

	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type
dct	5	1	256	462	1851	1852	none
dct_2d	3	1	195	322	1718	1718	none
dct_	0	1	117	123	97	97	none

Performance Profile | Resource Profile

	Pipelined	Latency	Initiation Interval	Iteration Latency	Trip count
dct	-	1851	1852	-	-
RD_Loop_Row_RD_Loop_Col	yes	64	1	2	64
WR_Loop_Row_WR_Loop_Col	yes	64	1	2	64

(a) Zed

Module Hierarchy

	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type
dct	5	1	223	461	1723	1724	none
dct_2d	3	1	162	321	1590	1590	none
dct_1d2	0	1	84	122	89	89	none

Performance Profile | Resource Profile

	Pipelined	Latency	Initiation Interval	Iteration Latency	Trip count
dct	-	1723	1724	-	-
RD_Loop_Row_RD_Loop_Col	yes	64	1	2	64
WR_Loop_Row_WR_Loop_Col	yes	64	1	2	64

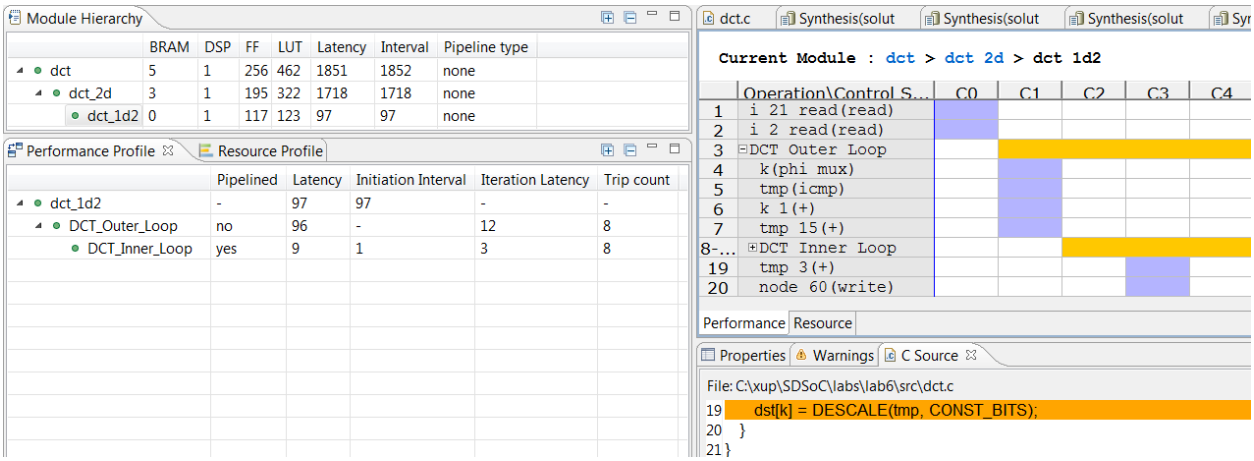
(b) Zybo

Figure 15. The performance profile at the dct function level

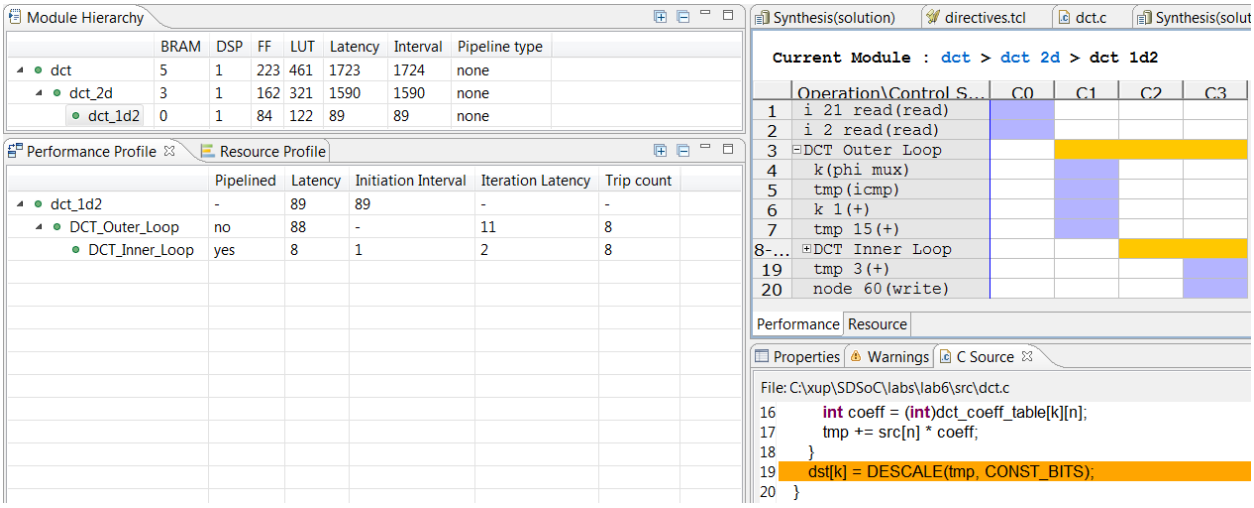
4-2-3. In the *Module Hierarchy* tab, expand the **dct** hierarchy completely. Notice that the most of the latency occurs is in dct_2d function. Select the *dct_1d2* entry.

4-2-4. In the *Performance Profile* tab, select the *DCT_Inner_Loop* entry

4-2-5. In the Performance view, right-click on the node_60 (write) block in the C3 state, and select Goto Source. Notice that line 19 is highlighted which is preventing the flattening of the DCT_Outer_Loop.



(a) Zed



(b) Zybo

Figure 16. Understanding what is preventing DCT_Outer_Loop flattening

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

4-3. 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.

4-3-1. Select **Project > New Solution**.

4-3-2. A *Solution Configuration* dialog box will appear. Click the **Finish** button (with Solution2 selected).

4-3-3. Close all inactive solution windows by selecting **Project > Close Inactive Solution Tabs.**

4-3-4. With the source code open, in the Directive pane, select the PIPELINE directive of the **DCT_Inner_Loop** of the **dct_1d** function, right-click on it and select *Remove Directive.*

4-3-5. Click **No**, if asked, to not to remove the label.

4-3-6. In the Directive pane again, select the **DCT_Outer_Loop** of the **dct_1d** function, right-click on it and select *Insert Directive...*

4-3-7. A pop-up menu shows up listing various directives. Select the **PIPELINE** directive

4-3-8. Click **Yes** and then **OK**.

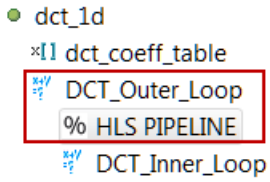


Figure 17. PIPELINE directive applied to DCT_Outer_Loop

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.

4-3-9. Click on the **Synthesis** button.

4-3-10. When the synthesis is completed, select **Project > Compare Reports…** to compare the two solutions.

4-3-11. Select *Solution2* and *Solution3* from the **Available Reports,** click on the **Add>>** button, and then click **OK**.

4-3-12. Observe that the latency reduced from 1851 to 875 for Zed whereas 1723 to 859 for Zybo.

Latency (clock cycles)

		solution3	solution2
Latency	min	875	1851
	max	875	1851
Interval	min	876	1852
	max	876	1852

(a) Zed

Latency (clock cycles)

		solution3	solution2
Latency	min	859	1723
	max	859	1723
Interval	min	860	1724
	max	860	1724

(b) Zybo

Figure 18. Performance comparison after pipelining

4-3-13. 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	678	256
LUT	523	462

(a) Zed

Utilization Estimates

	solution3	solution2
BRAM_18K	5	5
DSP48E	8	1
FF	615	223
LUT	537	461

(b) Zybo

Figure 19. Resources utilization after pipelining

4-3-14. Open the dct_1d2 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.

Loop

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

(a) Zed

Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- DCT_Outer_Loop	33	33	6	4	1	8	yes

(b) Zybo

Figure 20. Increased resource utilization of dct_1d

```
@I [RTMG-279] Implementing memory 'dct_dct_1d_dct_coeff_table_0_rom' using distributed ROMs.
@I [RTMG-279] Implementing memory 'dct_dct_1d_dct_coeff_table_1_rom' using distributed ROMs.
@I [RTMG-279] Implementing memory 'dct_dct_1d_dct_coeff_table_2_rom' using distributed ROMs.
@I [RTMG-279] Implementing memory 'dct_dct_1d_dct_coeff_table_3_rom' using distributed ROMs.
@I [RTMG-279] Implementing memory 'dct_dct_1d_dct_coeff_table_4_rom' using distributed ROMs.
@I [RTMG-279] Implementing memory 'dct_dct_1d_dct_coeff_table_5_rom' using distributed ROMs.
@I [RTMG-279] Implementing memory 'dct_dct_1d_dct_coeff_table_6_rom' using distributed ROMs.
@I [RTMG-279] Implementing memory 'dct_dct_1d_dct_coeff_table_7_rom' using distributed ROMs.
@I [RTMG-278] Implementing memory 'dct_dct_2d_row_outbuf_ram' using block RAMs.
@I [RTMG-278] Implementing memory 'dct_dct_2d_col_inbuf_ram' using block RAMs.
```

Figure 21. Automatic partitioning of dct_coeff_table

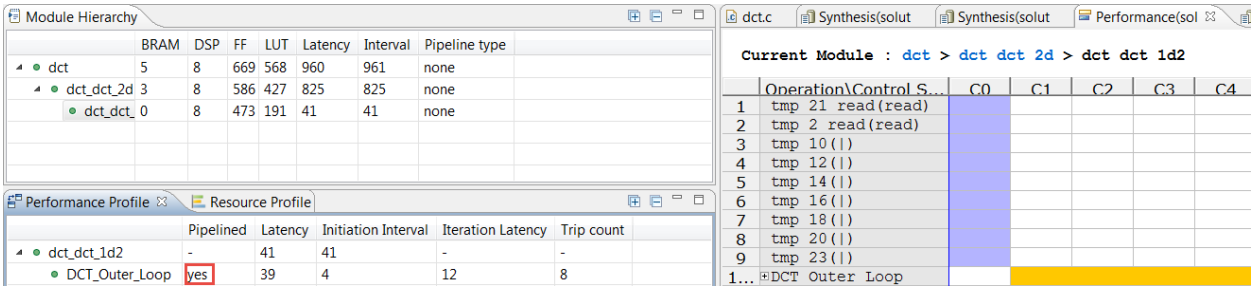
```
@W [SCHED-69] Unable to schedule 'load' operation ('src_load_5', ../../../src/dct.c:17) on array 'src' due to limited memory ports.
@I [SCHED-61] Pipelining result: Target II: 1, Final II: 4, Depth: 8.
```

Figure 22. Initiation interval of 4

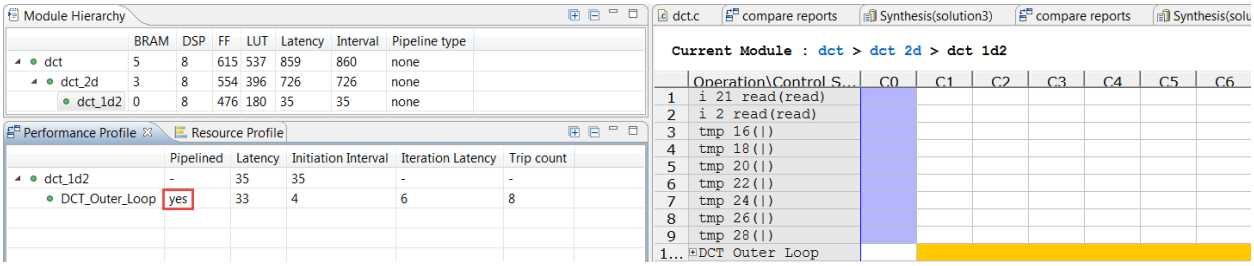
4-4. Perform design analysis by switching to the Analysis perspective and looking at the dct_1d performance view.

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

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



(a) Zed



(b**) Zybo**

Figure 23. DCT_Outer_Loop flattening

4-4-3. 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 2d > dct 1d2

	Resource\Control Step	C0	C1	C2	C3	C4	C5	C6	C7
1-6	⊞I/O Ports								
7	⊟Memory Ports								
8	src(p1)		read	read	read	read			
9	dct coeff tabl...		read						
10	dct coeff tabl...		read						
11	src(p0)		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								

(a) Zed

Current Module : dct > dct_2d > dct_1d2

	Resource\Control Step	C0	C1	C2	C3	C4	C5	C6
1-6	⊞I/O Ports							
7	⊟Memory Ports							
8	src(p1)		read	read	read	read		
9	dct_coeff_tabl...		read					
10	src(p0)		read	read	read	read		
11	dct_coeff_tabl...		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

(b) Zybo

Figure 24. The Resource tab

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

Improve Memory Bandwidth — Step 5

5-1. 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.

5-1-1. Select **Project > New Solution** to create a new solution.

5-1-2. A *Solution Configuration* dialog box will appear. Click the **Finish** button (with Solution3 selected).

5-1-3. 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 the *src* port of the *dct_1d* function. This array was passed via *in_block* of the *dct_2d* function, which in turn was passed via *buf_2d_in* of the *dct* function).

5-1-4. A pop-up menu shows up listing various directives. Select the **ARRAY_PARTITION** directive.

5-1-5. Make sure that the **type** is *complete*. Enter **2** in the *dimension* field and click **OK**.

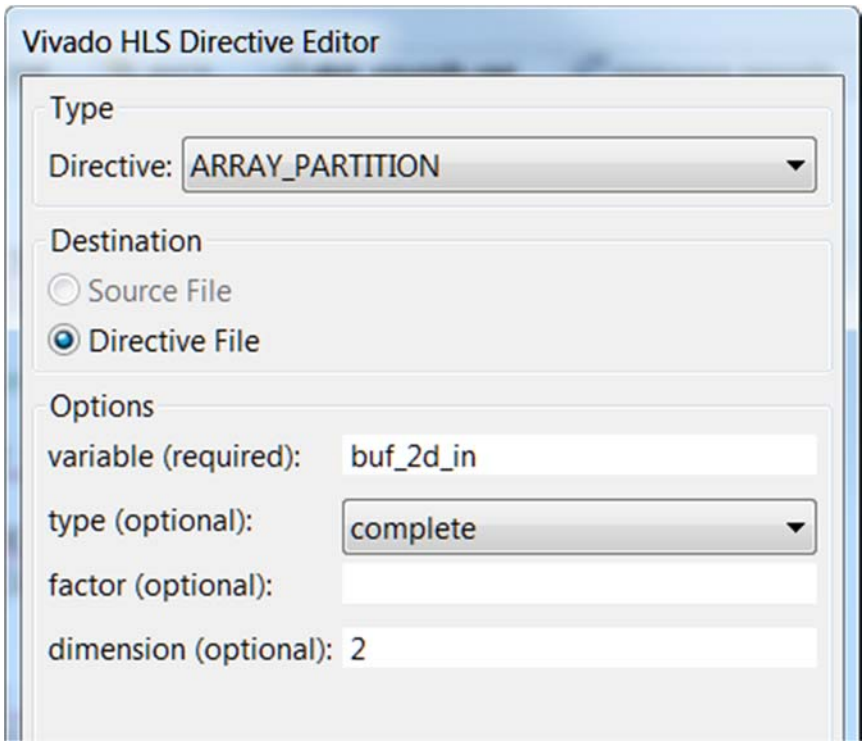


Figure 25. Applying ARRAY_PARTITION directive to memory buffer

5-1-6. Similarly, apply the *ARRAY_PARTITION* directive with dimension of *2* to the **col_inbuf** array of the dct_2d function.

5-1-7. Click on the **Synthesis** button.

5-1-8. When the synthesis is completed, select **Project > Compare Reports…** to compare the two solutions.

5-1-9. Select *Solution3* and *Solution4* from the **Available Reports,** and click on the **Add>>** button.

5-1-10. Observe that the latency reduced from 875 to 509 for Zed and from 859 to 493 for Zybo.

⊟ **Latency (clock cycles)**

		solution4	solution3
Latency	min	509	875
	max	509	875
Interval	min	510	876
	max	510	876

(a) Zed

⊟ **Latency (clock cycles)**

		solution4	solution3
Latency	min	493	859
	max	493	859
Interval	min	494	860
	max	494	860

(b) Zybo

Figure 26. Performance comparison after array partitioning

5-1-11. Scroll down in the comparison report to view the resources utilization. Observe the increase in the FF resource utilization (almost double) and BRAM_18K utilization reduced as the two selected arrays were completely partitioned.

Utilization Estimates

	solution4	solution3
BRAM_18K	3	5
DSP48E	8	8
FF	1204	678
LUT	630	523

(a) Zed

Utilization Estimates

	solution4	solution3
BRAM_18K	3	5
DSP48E	8	8
FF	1075	615
LUT	630	537

(b) Zybo

Figure 27. Resources utilization after array partitioning

5-1-12. Expand the **Loop** entry in the **dct.rpt** entry and observe that the Pipeline II is now *1*.

Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- WR_Loop_Row_WR_Loop_Col	64	64	2	1	1	64	yes

(a) Zed

Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- WR_Loop_Row_WR_Loop_Col	64	64	2	1	1	64	yes

(b) Zybo

Figure 28. II=1 achieved

Apply DATAFLOW Directive — Step 6

6-1. 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.

6-1-1. Select **Project > New Solution**.

6-1-2. A *Solution Configuration* dialog box will appear. Click the **Finish** button (with Solution4 selected).

6-1-3. Close all inactive solution windows by selecting **Project > Close Inactive Solution Tabs.**

6-1-4. Select function **dct** in the directives pane, right-click on it and select *Insert Directive...*

6-1-5. Select the **DATAFLOW** directive to improve the throughput.

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

6-1-7. When the synthesis is completed, the synthesis report is automatically opened.

6-1-8. Observe that dataflow type pipeline throughput is listed in the Performance Estimates.

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
508	508	375	375	dataflow

(a) Zed

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
492	492	359	359	dataflow

(b) Zybo

Figure 29. 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 top-level of the design will get benefit from dataflow optimization.

6-1-9. 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.

```
INFO: [XFORM 203-712] Applying dataflow to function 'dct' (../../../src/dct.c:78), detected/extracted 3 process function(s):
        'read_data'
        'dct_2d'
        'write_data'.
```

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

6-2. Save the directives as pragmas in the dct.c file and exit Vivado HLS.

6-2-1. Double-click on the **directives.tcl** entry under *solutions5* > *constraints*.

```
1  ############################################################
2  ## This file is generated automatically by Vivado HLS.
3  ## Please DO NOT edit it.
4  ## Copyright (C) 1986-2016 Xilinx, Inc. All Rights Reserved.
5  ############################################################
6  set_directive_resource -core RAM_1P "dct" input
7  set_directive_resource -core RAM_1P "dct" output
8  set_directive_latency -min 1 "dct"
9  set_directive_pipeline "dct_2d/Xpose_Col_Inner_Loop"
10 set_directive_pipeline "dct_2d/Xpose_Row_Inner_Loop"
11 set_directive_pipeline "read_data/RD_Loop_Col"
12 set_directive_pipeline "write_data/WR_Loop_Col"
13 set_directive_pipeline "dct_1d/DCT_Outer_Loop"
14 set_directive_array_partition -type complete -dim 2 "dct" buf_2d_in
15 set_directive_array_partition -type complete -dim 2 "dct_2d" col_inbuf
16 set_directive_dataflow "dct"
```

Figure 31. The applied directives

6-2-2. Since SDx does not use the *directives.tcl* file, we will need to move all the desired directives and implement them as *pragmas* in the *dct.c* source file.

6-2-3. With *dct.c* open and in focus, in the **Directives** tab, select one directive at a time, right-click on it, select and **Modify Directive**.

6-2-4. Select **Source file** as the *destination* and click **OK**.

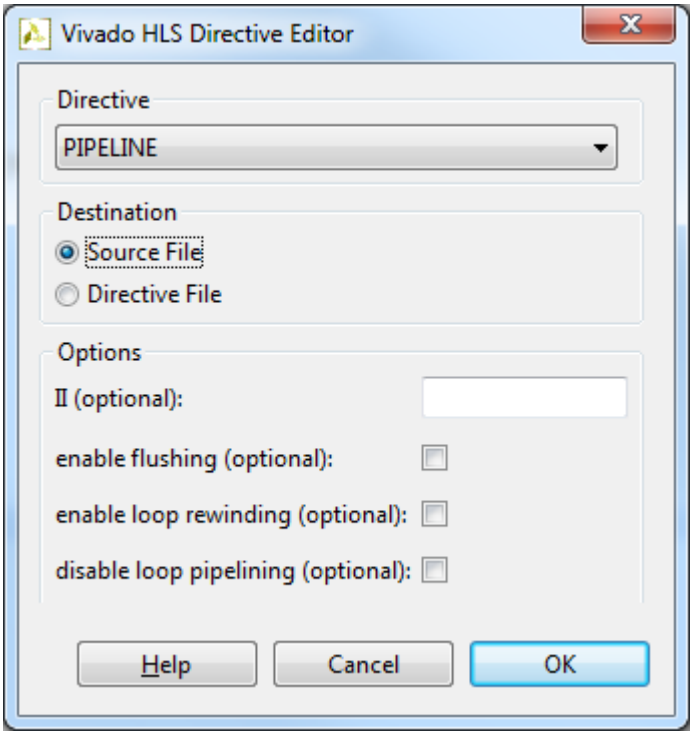


Figure 32. Move directives to source file

When all eleven directives are transferred to the source file, you should see #pragma directives on lines 14, 29, 43, 56, 69, 83, 90, 91, 92, 93, and 96.

6-2-5. Select **File > Save**.

6-2-6. Close Vivado HLS by selecting **File > Exit.**

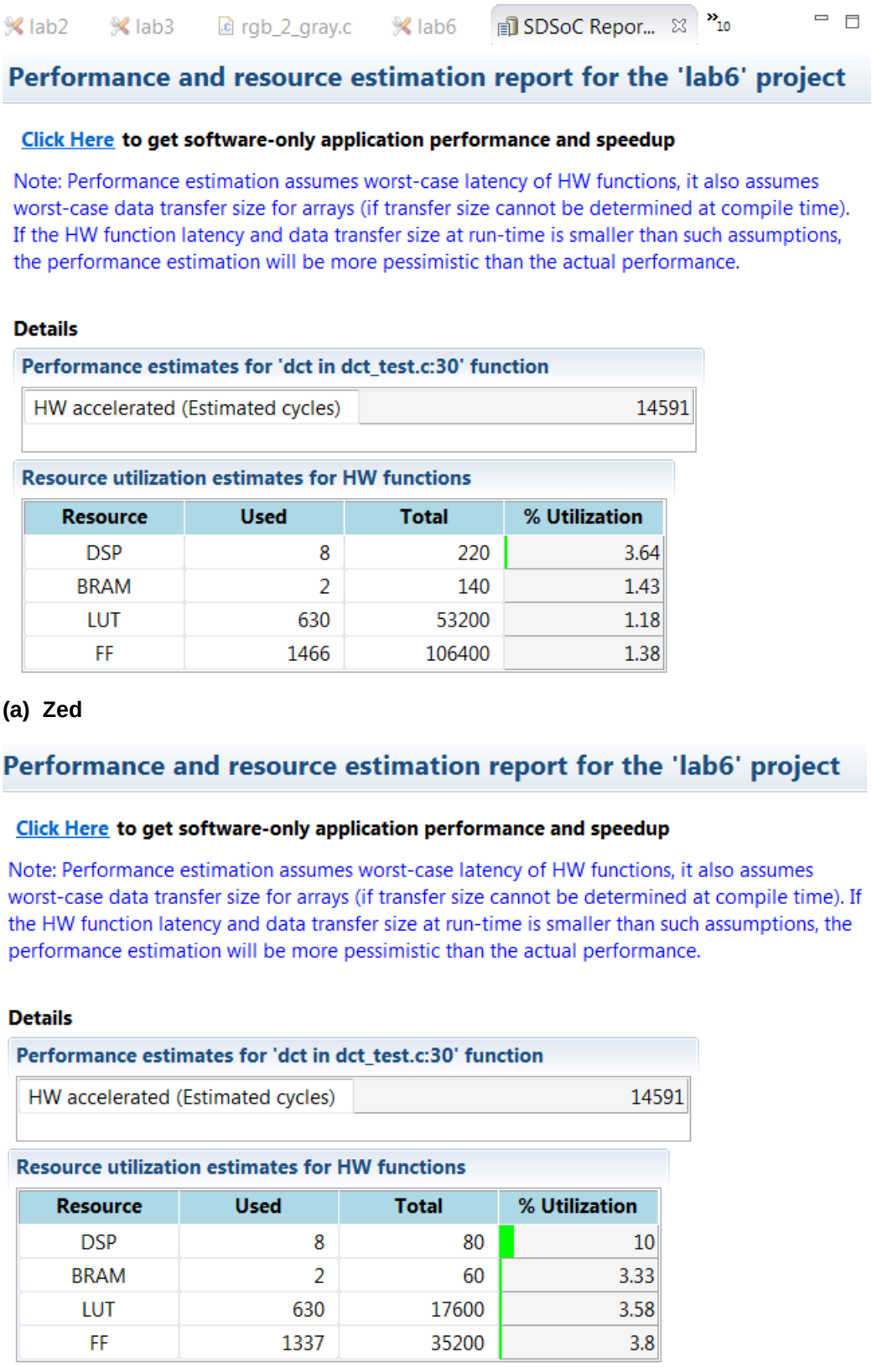
Performance Estimation of Optimized Model Step 7

7-1. Clean the project and re-estimate the performance.

7-1-1. Right-click the top-level folder for the project and click on **Clean Project** in the menu.

7-1-2. Build the project.

7-1-3. After the build is over, you can see an initial report. This report contains a hardware-only estimate summary.



(a) Zed

(b) Zybo

Figure 33. Initial estimate of hardware only performance of the optimized code

The Summary section shows that the estimated HW acceleration is 14591 compared to the initial hardware acceleration of 34511 for Zed, and 14895 compared to the initial hardware accelerator of 27469, yielding a 2.4x (for Zed) and 1.84x (for Zybo) improvement.

Conclusion

In this lab, you learned various techniques to improve the performance using Vivado HLS. These directives can be used with SDSoc through pragma statements. The 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 the DATAFLOW directive is applied, the default memory buffers (of ping-pong type) are automatically inserted between the top-level functions and loops. The Analysis perspective and console logs can provide insight on what is going on.