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

Step 1:

Validate the Design

Step 2:

Create a new project

Step 3:

Synthesize the design

Step 4:

Run RTL  
Simulation

Step 5:

Apply PIPELINE  
directive

Step 9:

Apply RESHAPE  
directive

Step 8:

Apply INLINE  
directive

Step 7:

Apply DATAFLOW  
directive

Step 6:

Improve the Memory Bandwidth

1. Validate the Design from Command Line Step 1
   1. Validate your design from Vivado HLS command line.
      1. Launch Vivado HLS: Select **Start > All Programs > Xilinx Design Tools > Vivado 2015.4 > Vivado HLS > Vivado HLS 2015.4 Command Prompt**.
      2. In the Vivado HLS Command Prompt, change directory to **c:\xup\hls\labs\lab3.**
      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.

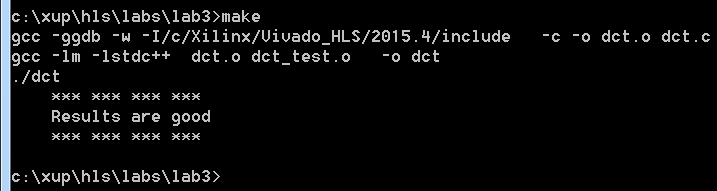


Figure 1. Validating the design

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. Close the command prompt window by typing **exit**.

1. Create a New Project Step 2
   1. Create a new project in Vivado HLS GUI targeting XC7Z020CLG484-1 (ZedBoard) or XC7Z010CLG400-1 (Zybo).
      1. Launch Vivado HLS: Select **Start > All Programs > Xilinx Design Tools > Vivado 2015.4 > Vivado HLS > Vivado HLS 2015.4**
      2. In the Vivado HLS GUI, select **File >** **New Project.** The New Vivado HLS Project wizard opens.
      3. Click **Browse…** button of the Location field and browse to **c:\xup\hls\labs\lab3** and then click **OK.**
      4. For Project Name, type ***dct.prj***
      5. Click **Next.**
      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).
      7. Click the **Add Files…** button, select *dct.c* file from the **c:\xup\hls\labs\lab3** folder, and then click **Open**.
      8. Click **Next**.
      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.**
      10. Click **Next.**
      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.
      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**

* + 1. Click **Finish.**
    2. Double-click on the **dct.c** under the *source* folder to open its content in the information pane.

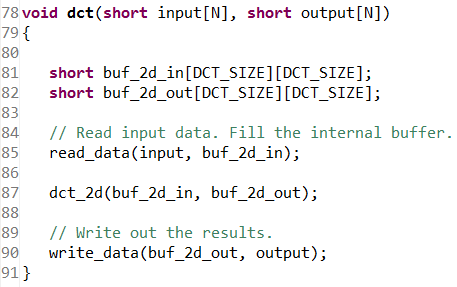


Figure 2. The design under consideration

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

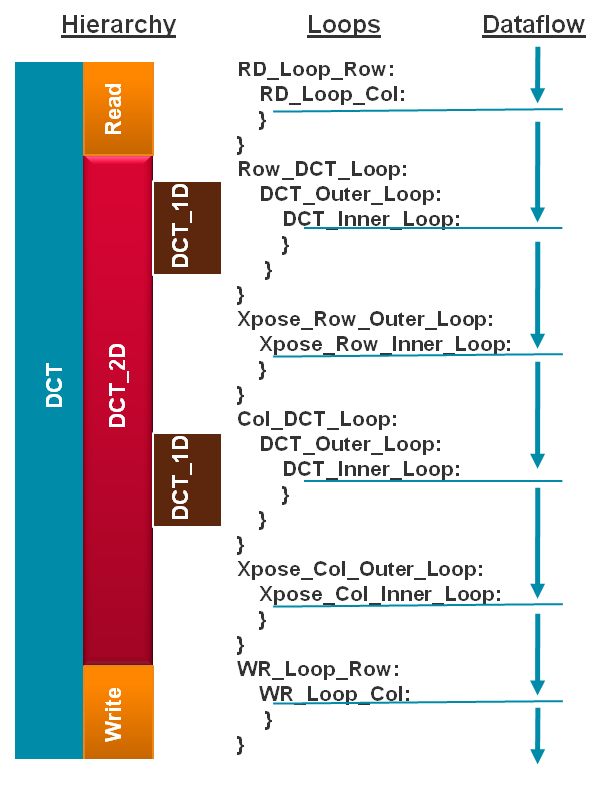


Figure 3. Design hierarchy and dataflow

1. Synthesize the Design Step 3
   1. 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** or click on the  button 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.

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.

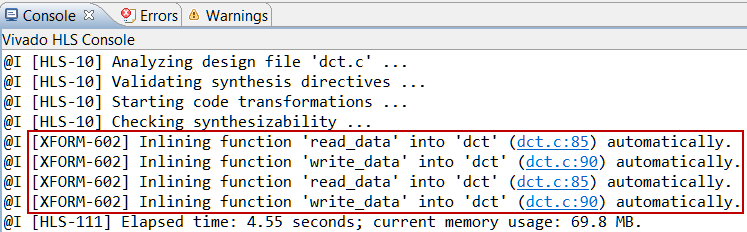


Figure 4. Inlining of read\_data and write\_data functions

* + 1. 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.

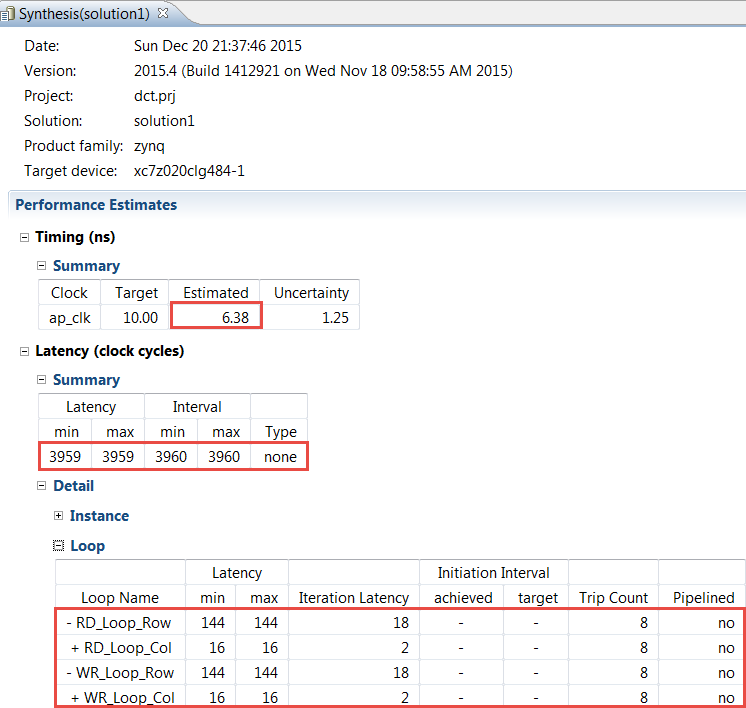


Figure 5. Synthesis report

* + 1. 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:

* + 1. The report also shows the top-level interface signals generated by the tools.

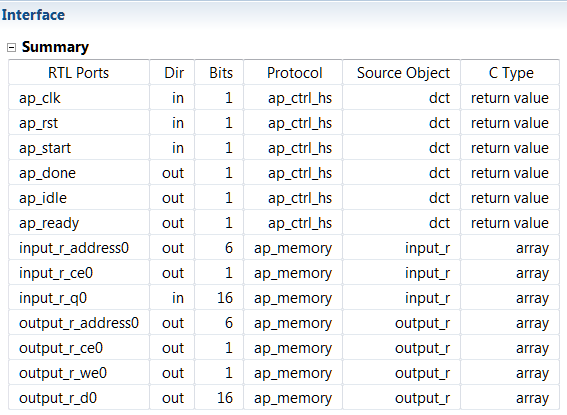


Figure 6. Generated interface signals

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.

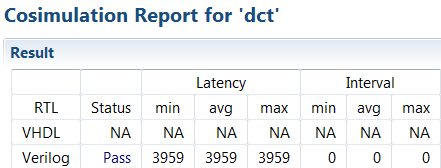
* + 1. Open dct\_1d.rpt and dct\_2d.rpt files either using the Explorer view or by using a hyperlink at the bottom of the dct.rpt in the information view. The report for dct\_2d clearly indicates that most of this design cycles (3668) are spent doing the row and column DCTs. Also the dct\_1d report indicates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

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

* + 1. 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.



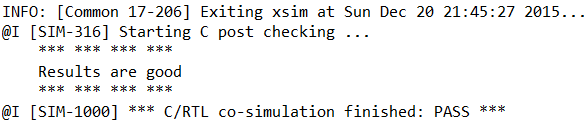


Figure 7. RTL Co-Simulation results

1. Apply PIPELINE Directive Step 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.
      1. Select **Project > New Solution** or click on () from the tools bar buttons.
      2. A *Solution Configuration* dialog box will appear. Click the **Finish** button (with *copy from* Solution1 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 *Insert Directive...*
      5. A pop-up menu shows up listing various directives. Select **PIPELINE** directive.
      6. Leave **II** (Initiation Interval) blank as Vivado 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. At this point, the Directive tab should look like as follows.

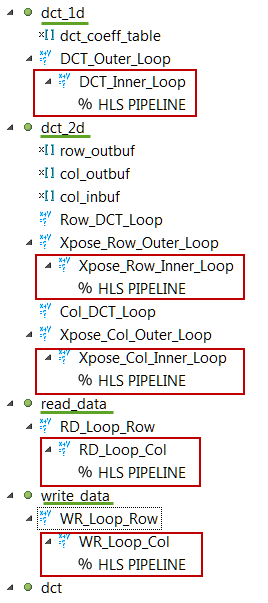
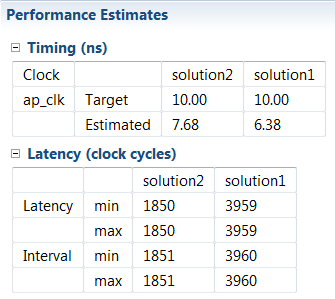
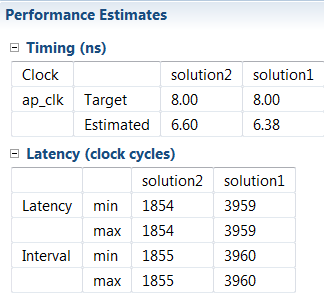


Figure 8. PIPELINE directive applied

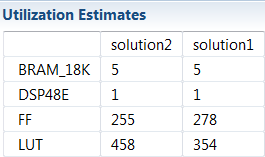
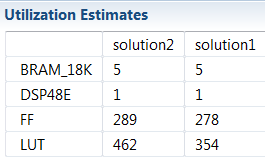
* + 1. Click on the **Synthesis** button.
    2. When the synthesis is completed, select **Project > Compare Reports…** or click on  to compare the two solutions.
    3. Select *Solution1* and *Solution2* from the **Available Reports,** click on the **Add>>** button, and then click **OK**.
    4. Observe that the latency reduced from 3959 to 1850 clock cycles (ZedBoard) and from 3959 to 1854 (Zybo).

(a) ZedBoard (b) Zybo

Figure 9. Performance comparison after pipelining

* + 1. 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.

(a) ZedBoard (b) Zybo

Figure 10. Resources utilization after pipelining

* 1. Open the Analysis perspective and determine where most of the clock cycles are spend, i.e. where the large latencies are.
     1. Click on the *Analysis* perspective button.
     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.

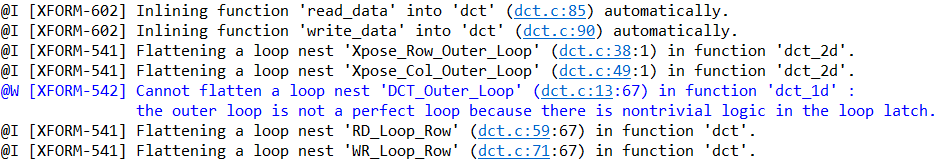
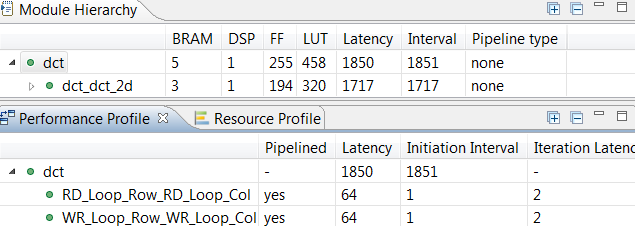
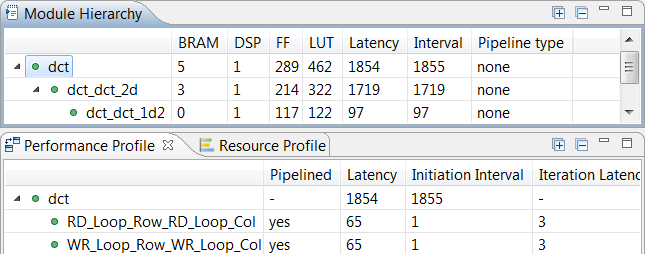


Figure 11. The console view content indicating loops flattening



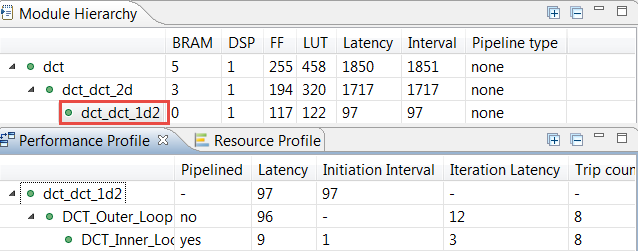
(a) ZedBoard

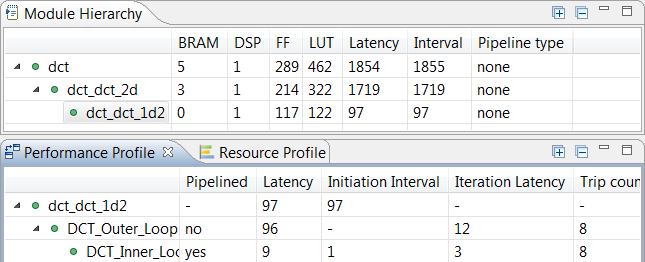


(b) Zybo

Figure 12. The performance profile at the dct function level

* + 1. In the Module Hierarchy tab, expand **dct > dct\_2d > dct\_1d**. Notice that the most of the latency occurs is in dct\_2d function.
    2. 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.

 (a) ZedBoard



(b) Zybo

Figure 13. The dct\_1d function performance profile

* + 1. 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.

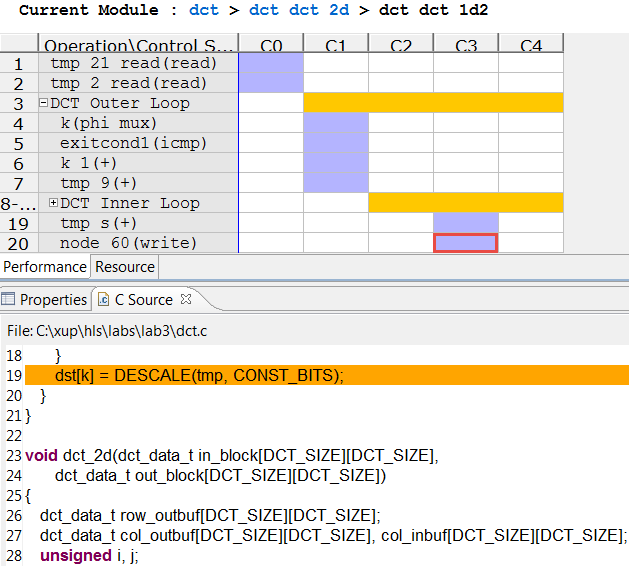


Figure 14. Understanding what is preventing DCT\_Outer\_Loop flattening

* + 1. Switch to the *Synthesis* perspective.
  1. 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 Solution2 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**.

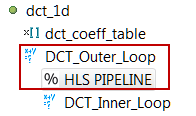
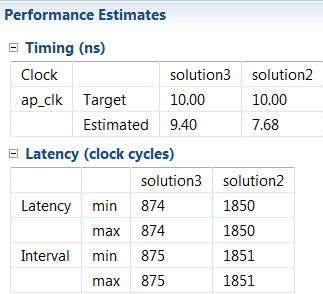
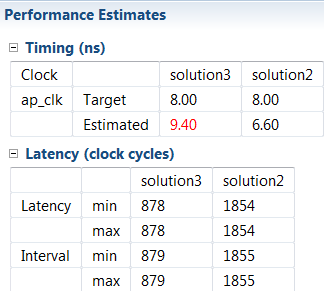


Figure 15. PIPELINE directive applied to DCT\_Outer\_Loop

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.

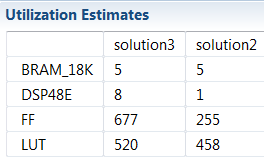
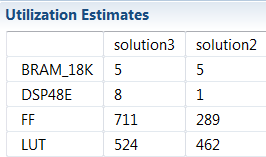
* + 1. Click on the **Synthesis** button.
    2. When the synthesis is completed, select **Project > Compare Reports…** to compare the two solutions.
    3. Select *Solution2* and *Solution3* from the **Available Reports,** click on the **Add>>** button, and then click **OK**.
    4. Observe that the latency reduced from 1850 to 874 clock cycles for ZedBoard (1854 to 878 for Zybo).

(a) ZedBoard (b) Zybo

Figure 16. Performance comparison after pipelining

* + 1. 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.

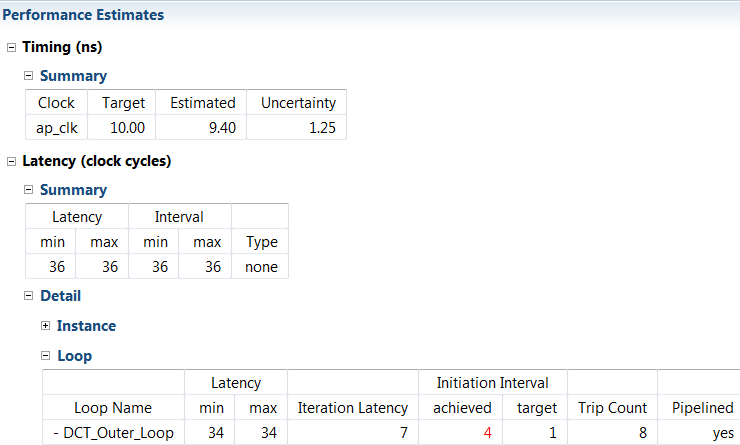
(a) ZedBoard (b) Zybo

Figure 17. Resources utilization after pipelining

* + 1. 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.



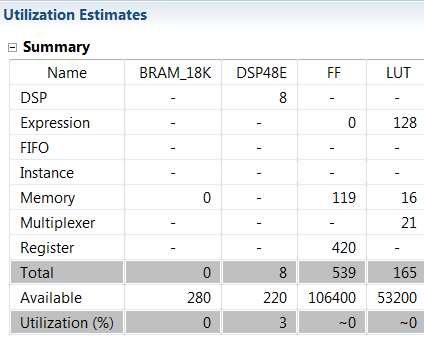


Figure 18. Increased resource utilization of dct\_1d

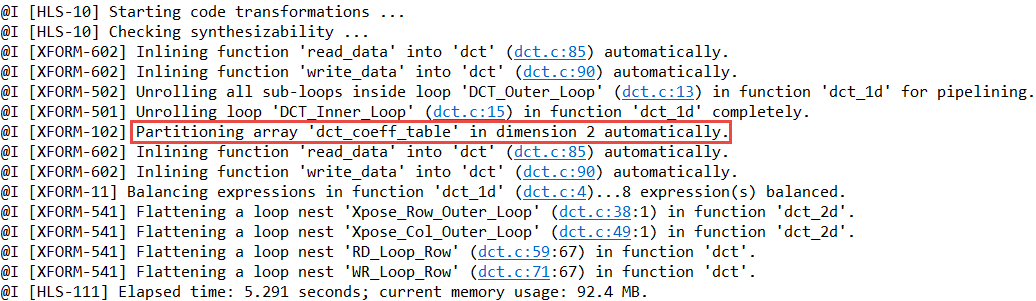


Figure 19. Automatic partitioning of dct\_coeff\_table

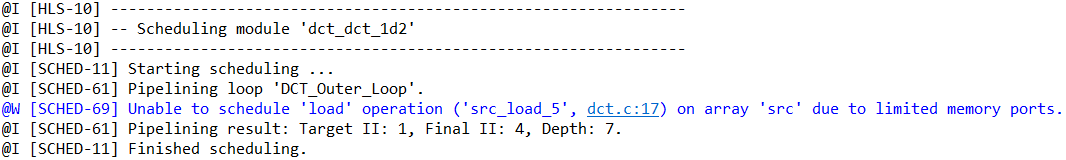
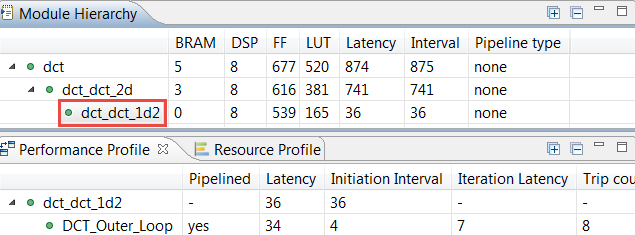
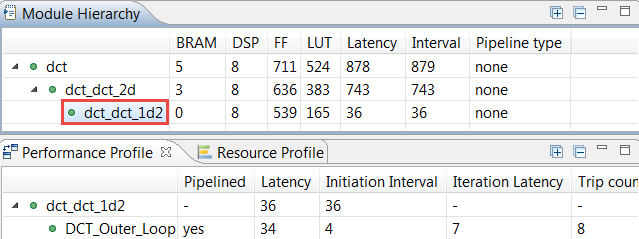


Figure 20. Initiation interval of 4

* 1. Perform design analysis by switching to the Analysis perspective and looking at the dct\_1d performance view.
     1. Switch to the Analysis perspective, expand the *Module Hierarchy* entries, and select the *dct\_1d* entry.
     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.



(a) ZedBoard



(b) Zybo

Figure 21. DCT\_Outer\_Loop flattening

* + 1. 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.

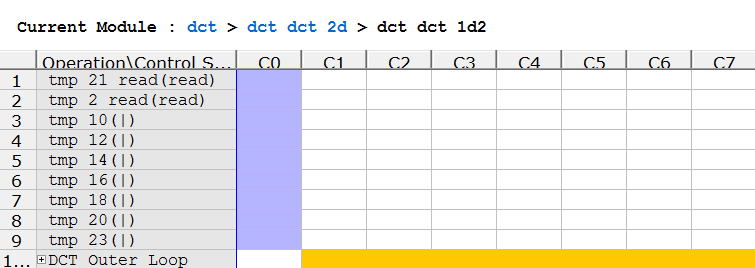


Figure 22. The Performance view of the DCT\_Outer\_Loop function

* + 1. 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.

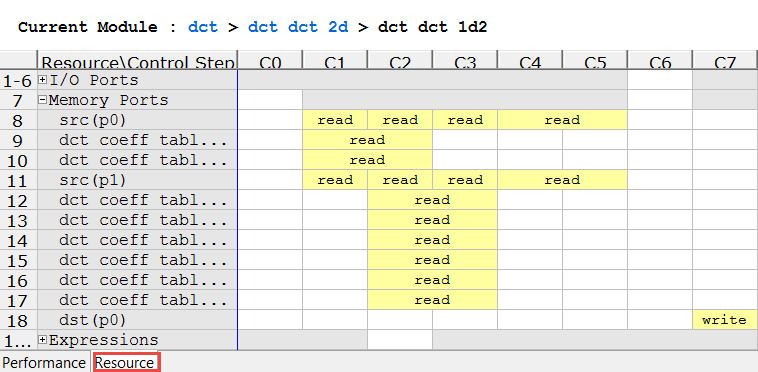


Figure 23. The Resource tab

* + 1. Switch to the *Synthesis* perspective.

1. Improve Memory Bandwidth Step 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.
      1. Select **Project > New Solution** to create a new solution.
      2. A *Solution Configuration* dialog box will appear. Click the **Finish** button (with Solution3 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).

* + 1. A pop-up menu shows up listing various directives. Select **ARRAY\_PARTITION** directive.
    2. Make sure that the **type** is *complete*. Enter **2** in the *dimension* field and click **OK**.

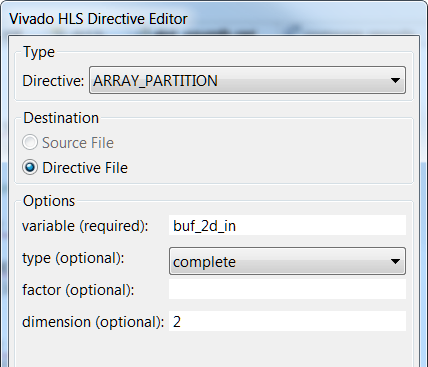
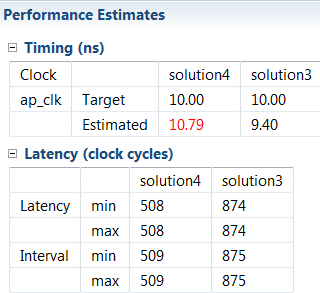
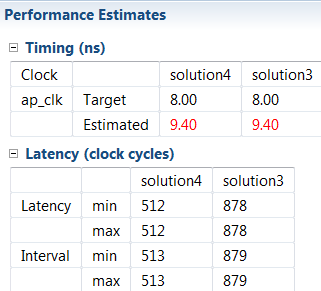


Figure 24. Applying ARRAY\_PARTITION directive to memory buffer

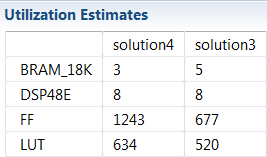
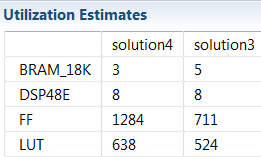
* + 1. Similarly, apply the *ARRAY\_PARTITION* directive with dimension of *2* to the **col\_inbuf** array.
    2. Click on the **Synthesis** button.
    3. When the synthesis is completed, select **Project > Compare Reports…** to compare the two solutions.
    4. Select *Solution3* and *Solution4* from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

(a) ZedBoard (b) Zybo

Figure 25. Performance comparison after array partitioning

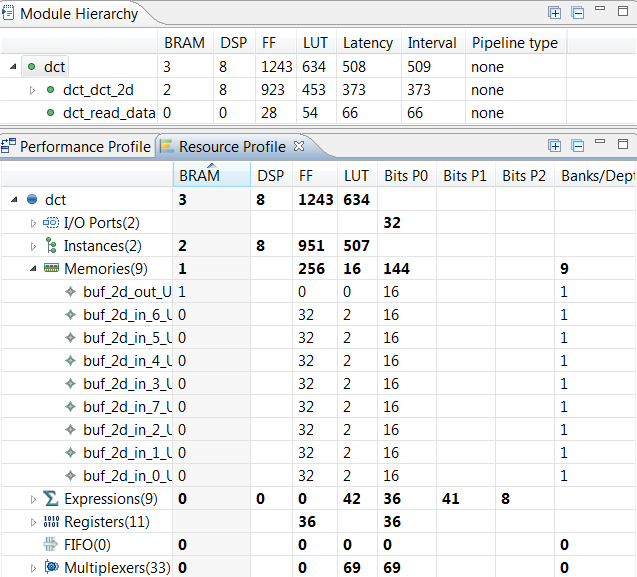
* + 1. Scroll down in the comparison report to view the resources utilization. Observe the increase in the FF resource utilization (almost double).

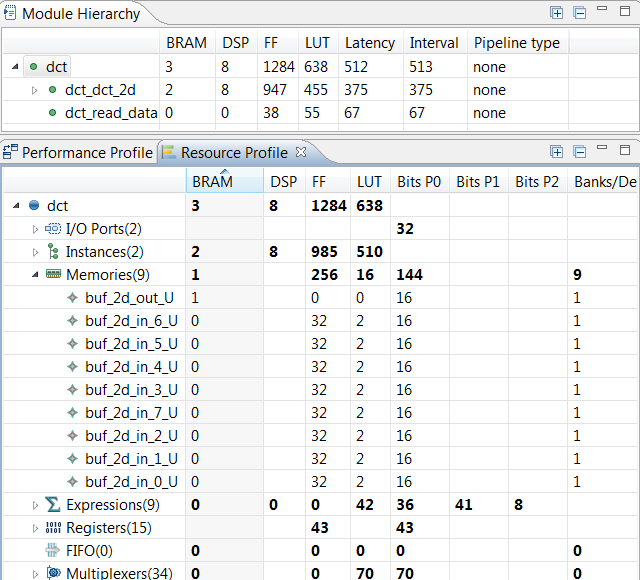
(a) ZedBoard (b) Zybo

Figure 26. Resources utilization after array partitioning

* + 1. Expand the **Loop** entry in the **dct.rpt** entry and observe that the Pipeline II is now *1*.
  1. Perform resource analysis by switching to the Analysis perspective and looking at the dct resources profile view.
     1. Switch to the Analysis perspective, expand the Module Hierarchy entries, and select the *dct* entry.
     2. Select the **Resource Profile** tab.
     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.



(a) ZedBoard

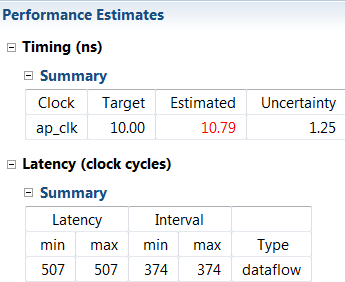
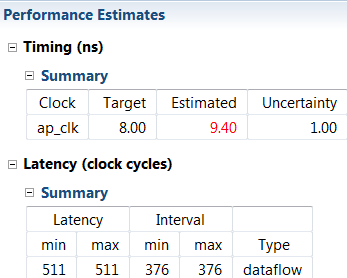


(b) Zybo

Figure 27. Resource profile after partitioning buffers

* + 1. Switch to the *Synthesis* perspective.

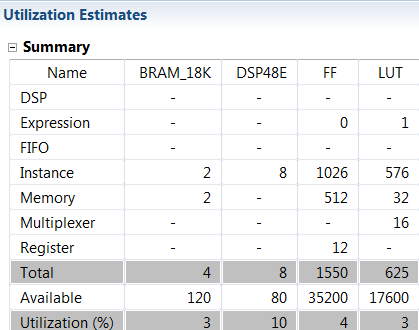
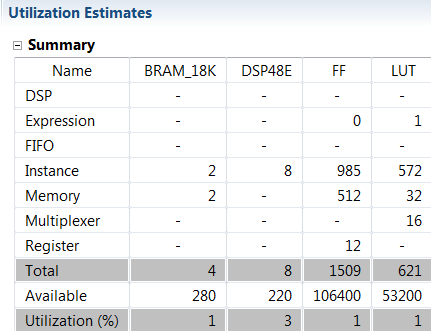
1. Apply DATAFLOW Directive Step 7
   1. Create a new solution by copying the previous solution (Solution4) settings. Apply the DATAFLOW directive to improve the throughput. Generate the solution and analyze the output.
      1. Select **Project > New Solution**.
      2. A *Solution Configuration* dialog box will appear. Click the **Finish** button (with Solution4 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.

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



(a) ZedBoard (b) Zybo

Figure 29. Resource estimate with DATAFLOW directive applied

* + 1. 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.

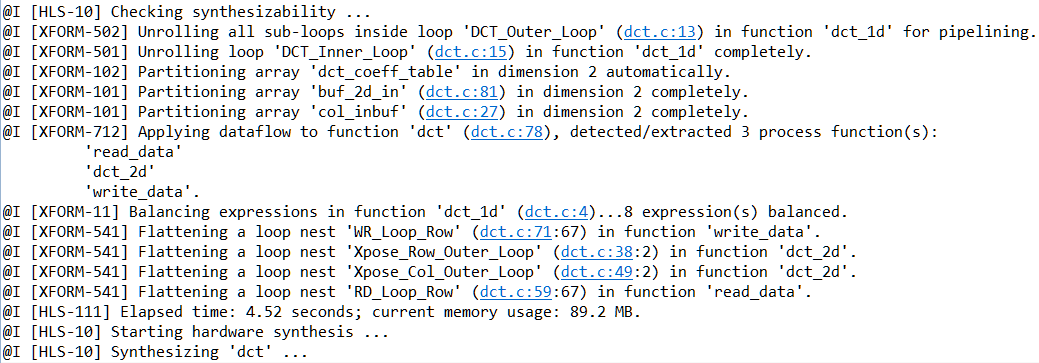
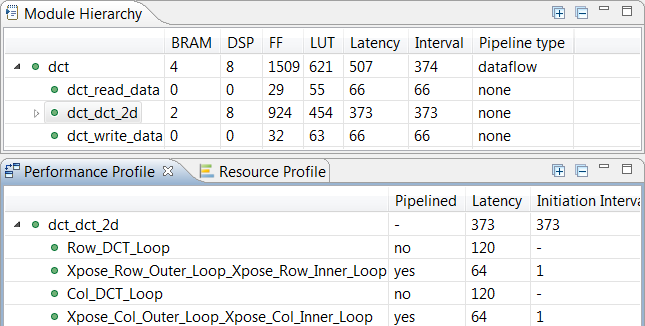


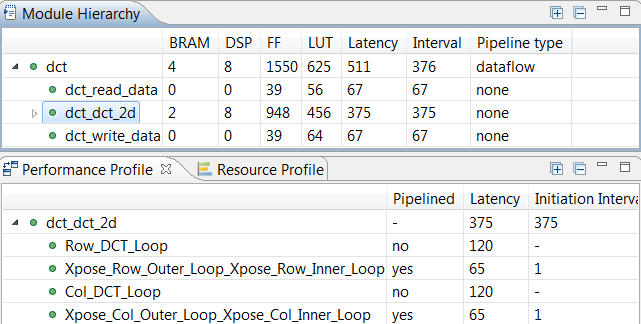
Figure 30. Console view of synthesis process after DATAFLOW directive applied

* 1. Perform performance analysis by switching to the Analysis perspective and looking at the dct performance profile view.
     1. Switch to the Analysis perspective, expand the Module Hierarchy entries, and select the dct\_2d entry.
     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.



(a) ZedBoard



(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.

* + 1. Switch to the *Synthesis* perspective.

1. Apply INLINE Directive Step 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.
      1. Select **Project > New Solution**.
      2. A *Solution Configuration* dialog box will appear. Click the **Finish** button (with Solution5 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.

* + 1. Click on the **Synthesis** button.
    2. When the synthesis is completed, the synthesis report will be opened.
    3. 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).
    4. 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.

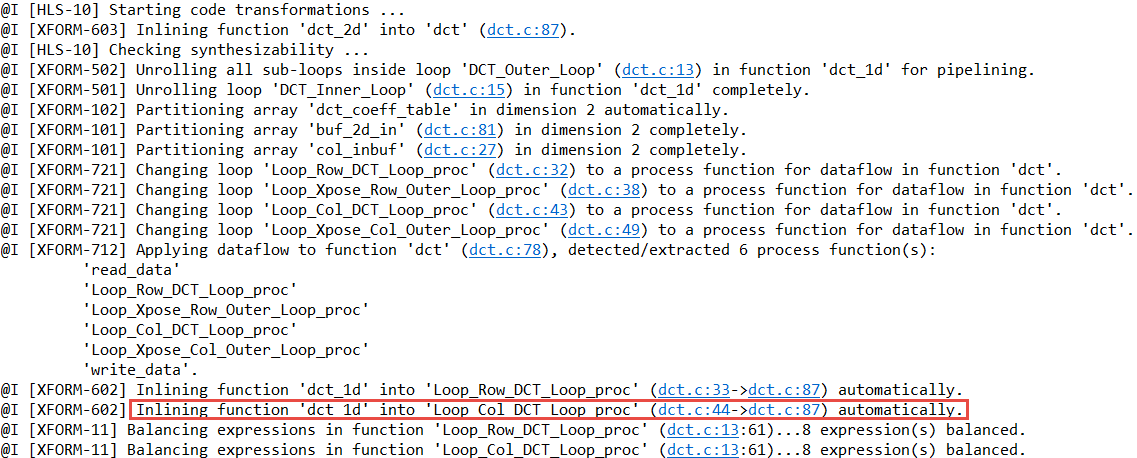
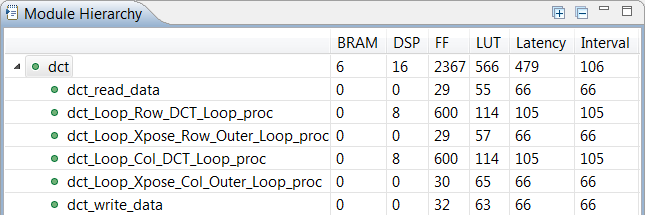


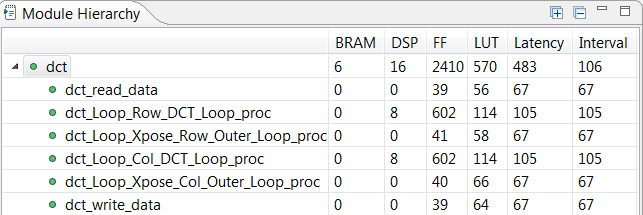
Figure 32. Console view after INLINE directive applied to dct\_2d

* + 1. 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.



(a) ZedBoard



(b) Zybo

Figure 33. Performance analysis after the INLINE directive

* + 1. Switch to the *Synthesis* perspective.

1. Apply RESHAPE Directive Step 9
   1. Create a new solution by copying the previous solution (Solution6) settings. Apply the RESHAPE directive. Generate the solution and understand the output.
      1. Select **Project > New Solution**.
      2. A *Solution Configuration* dialog box will appear. Click the **Finish** button (with Solution6 selected).
      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**.
      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**.
      5. Assign the **ARRAY\_RESHAPE** directive with dimension of 2 to the **dct\_coeff\_table** array.

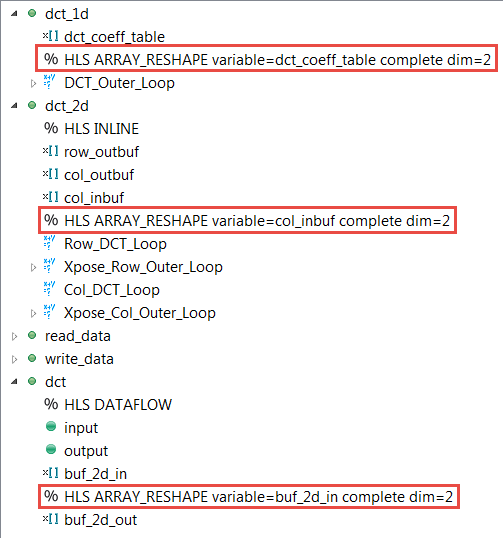


Figure 34. RESHAPE directive applied

* + 1. Click on the **Synthesis** button.
    2. When the synthesis is completed, the synthesis report is automatically opened.
    3. 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.
  + 1. Thus it can be seen the directives have to be applied carefully.
    2. 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: 6.38 ns

Worst case latency: 3959 clock cycles

Number of DSP48E used: 1

Number of BRAMs used: 5

Number of FFs used: 278

Number of LUTs used: 354