Improving Performance Lab

Introduction

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

Objectives

After completing this lab, you will be able to:

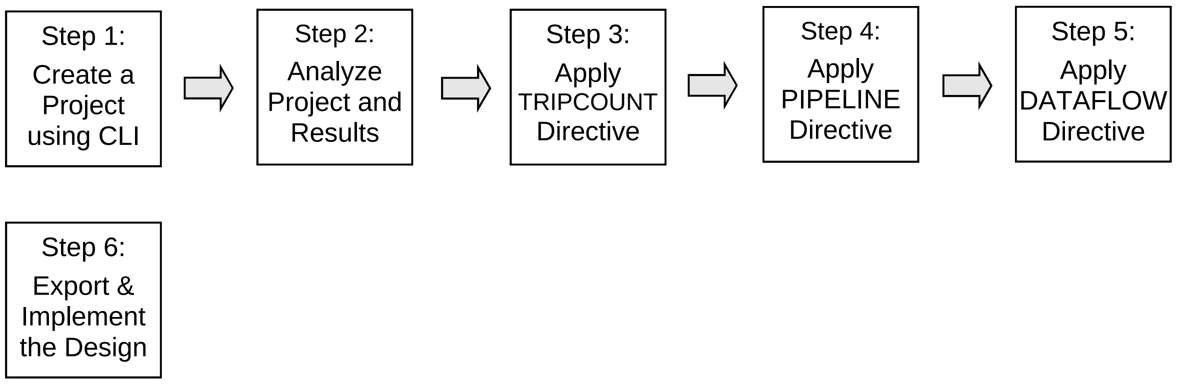
- Add directives in your design
- Understand the effect of INLINE directive
- Improve performance using PIPELINE directive
- Distinguish between DATAFLOW directive and Configuration Command functionality

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 6 primary steps: You will create a new project using Vivado HLS command prompt, analyze the created project and generated solution, turn off inlining and apply the TRIPCOUNT, PIPELINE, and DATAFLOW directives and command configuration, and finally export and implement the design.

General Flow for this Lab


Step 1: Create a Project using CLI
Step 2: Analyze Project and Results
Step 3: Apply TRIPCOUNT Directive
Step 4: Apply PIPELINE Directive
Step 5: Apply DATAFLOW Directive
Step 6: Export & Implement the Design

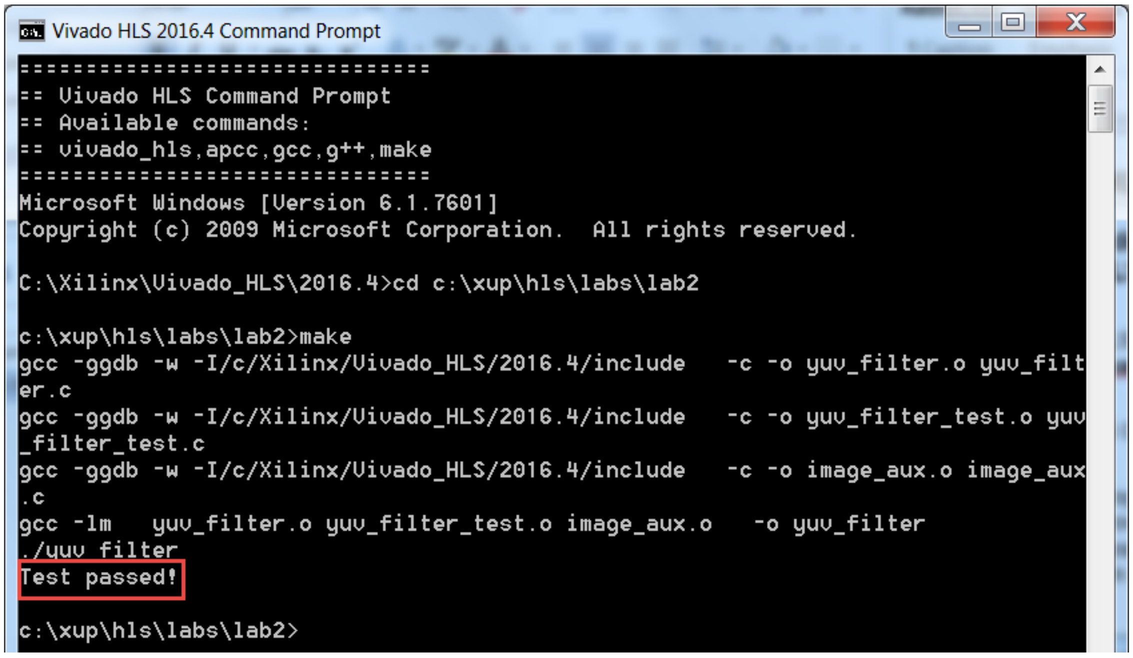
Create a Vivado HLS Project from Command Line Step 1

1-1. Validate your design using Vivado HLS command line window. Create a new Vivado HLS project from the command line.

1-1-1. Launch Vivado HLS: Select **Start > All Programs > Xilinx Design Tools > Vivado 2016.4 > Vivado HLS > Vivado HLS 2016.4 Command Prompt**.

1-1-2. In the Vivado HLS Command Prompt, change directory to **c:\xup\hls\labs\lab2.**

1-1-3. A self-checking program (yuv_filter_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. You can examine the contents of these files and the project directory. In the Vivado HLS Command Prompt, type **make** to compile and execute the program.



```
===============================
== Vivado HLS Command Prompt
== Available commands:
== vivado_hls,apcc,gcc,g++,make
===============================
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Xilinx\Vivado_HLS\2016.4>cd c:\xup\hls\labs\lab2

c:\xup\hls\labs\lab2>make
gcc -ggdb -w -I/c/Xilinx/Vivado_HLS/2016.4/include   -c -o yuv_filter.o yuv_filt
er.c
gcc -ggdb -w -I/c/Xilinx/Vivado_HLS/2016.4/include   -c -o yuv_filter_test.o yuv
_filter_test.c
gcc -ggdb -w -I/c/Xilinx/Vivado_HLS/2016.4/include   -c -o image_aux.o image_aux
.c
gcc -lm   yuv_filter.o yuv_filter_test.o image_aux.o   -o yuv_filter
./yuv_filter
Test passed!

c:\xup\hls\labs\lab2>
```

Figure 1. Validating the design

Note that the source files (yuv_filter.c, yuv_filter_test.c, and image_aux.c) were compiled, then yuv_filter executable program was created, and then it was executed. The program tests the design and outputs Test Passed message.

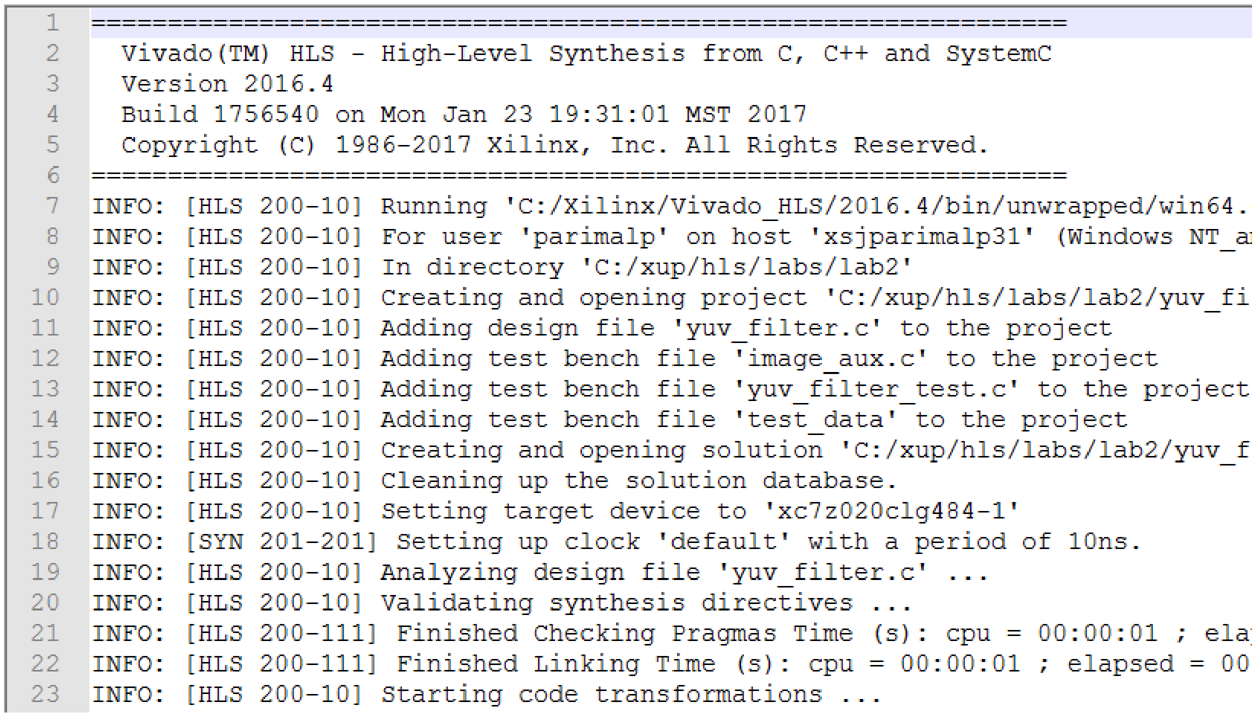
1-1-4. A Vivado HLS tcl script file (yuv_filter.tcl) is provided and can be used to create a Vivado HLS project.

1-1-5. Type **vivado_hls –f zed_yuv_filter.tcl** in the Vivado HLS Command Prompt window to create the project targeting the ZedBoard or type **vivado_hls –f zybo_yuv_filter.tcl** in the Vivado HLS Command Prompt window to create the project targeting the Zybo.

The project will be created and Vivado HLS.log file will be generated.

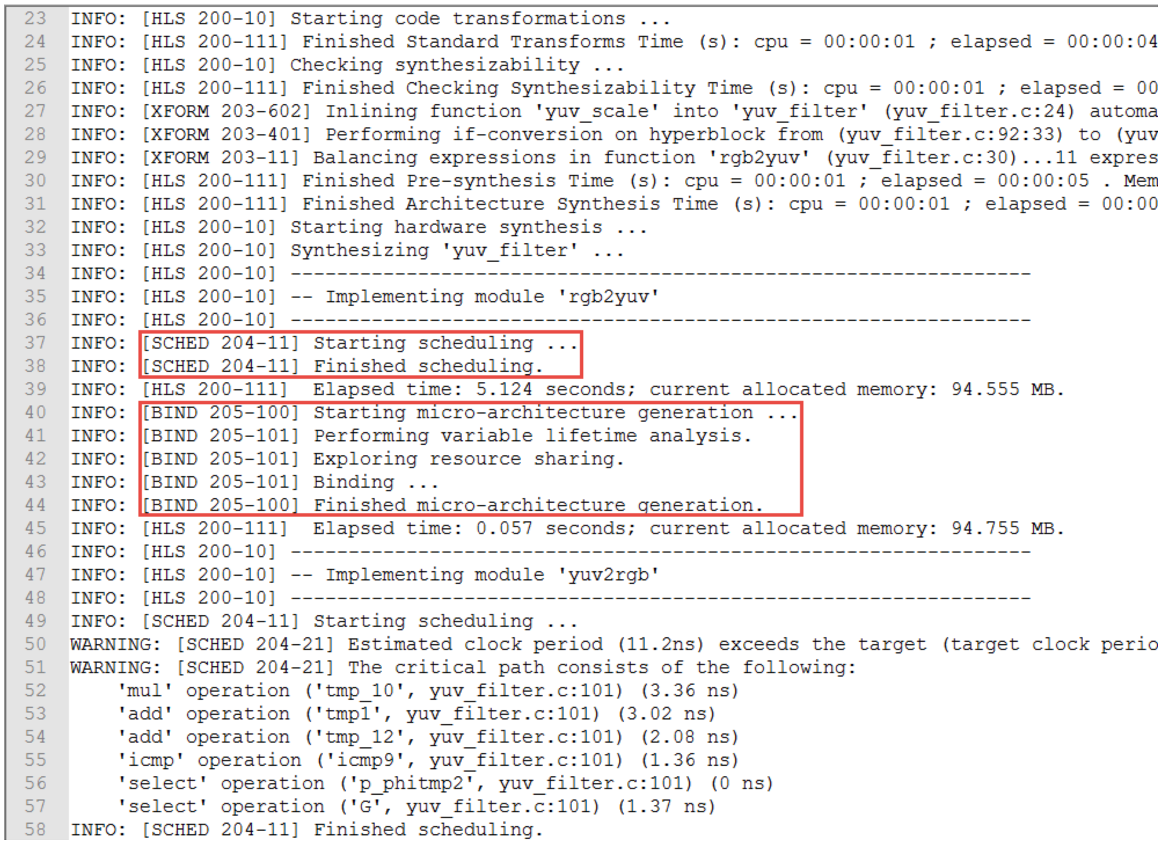
1-1-6. Open the **vivado_hls.log** file from **c:\xup\hls\labs\lab2** using any text editor and observe the following sections:

- o Creating directory and project called yuv_filter.prj within it, adding design files to the project, setting solution name as solution1, setting target device (Zynq-z020 for ZedBoard or Zynq-z010 for Zybo), setting desired clock period of 10 ns (for ZedBoard) or 8 ns (for Zybo), and importing the design and testbench files (Figure 2).
- o Synthesizing (Generating) the design which involves scheduling and binding of each functions and sub-function (Figure 3).
- o Generating RTL of each function and sub-function in SystemC, Verilog, and VHDL languages (Figure 4).



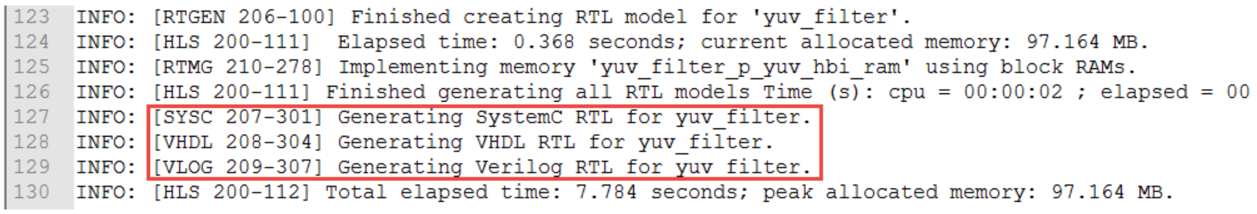
```
================================================================
  Vivado(TM) HLS - High-Level Synthesis from C, C++ and SystemC
  Version 2016.4
  Build 1756540 on Mon Jan 23 19:31:01 MST 2017
  Copyright (C) 1986-2017 Xilinx, Inc. All Rights Reserved.
================================================================
INFO: [HLS 200-10] Running 'C:/Xilinx/Vivado_HLS/2016.4/bin/unwrapped/win64.
INFO: [HLS 200-10] For user 'parimalp' on host 'xsjparimalp31' (Windows NT_a
INFO: [HLS 200-10] In directory 'C:/xup/hls/labs/lab2'
INFO: [HLS 200-10] Creating and opening project 'C:/xup/hls/labs/lab2/yuv_fi
INFO: [HLS 200-10] Adding design file 'yuv_filter.c' to the project
INFO: [HLS 200-10] Adding test bench file 'image_aux.c' to the project
INFO: [HLS 200-10] Adding test bench file 'yuv_filter_test.c' to the project
INFO: [HLS 200-10] Adding test bench file 'test_data' to the project
INFO: [HLS 200-10] Creating and opening solution 'C:/xup/hls/labs/lab2/yuv_f
INFO: [HLS 200-10] Cleaning up the solution database.
INFO: [HLS 200-10] Setting target device to 'xc7z020clg484-1'
INFO: [SYN 201-201] Setting up clock 'default' with a period of 10ns.
INFO: [HLS 200-10] Analyzing design file 'yuv_filter.c' ...
INFO: [HLS 200-10] Validating synthesis directives ...
INFO: [HLS 200-111] Finished Checking Pragmas Time (s): cpu = 00:00:01 ; ela
INFO: [HLS 200-111] Finished Linking Time (s): cpu = 00:00:01 ; elapsed = 00
INFO: [HLS 200-10] Starting code transformations ...
```

Figure 2. Creating project and setting up parameters



```
INFO: [HLS 200-10] Starting code transformations ...
INFO: [HLS 200-111] Finished Standard Transforms Time (s): cpu = 00:00:01 ; elapsed = 00:00:04
INFO: [HLS 200-10] Checking synthesizability ...
INFO: [HLS 200-111] Finished Checking Synthesizability Time (s): cpu = 00:00:01 ; elapsed = 00
INFO: [XFORM 203-602] Inlining function 'yuv_scale' into 'yuv_filter' (yuv_filter.c:24) automa
INFO: [XFORM 203-401] Performing if-conversion on hyperblock from (yuv_filter.c:92:33) to (yuv
INFO: [XFORM 203-11] Balancing expressions in function 'rgb2yuv' (yuv_filter.c:30)...11 expres
INFO: [HLS 200-111] Finished Pre-synthesis Time (s): cpu = 00:00:01 ; elapsed = 00:00:05 . Mem
INFO: [HLS 200-111] Finished Architecture Synthesis Time (s): cpu = 00:00:01 ; elapsed = 00:00
INFO: [HLS 200-10] Starting hardware synthesis ...
INFO: [HLS 200-10] Synthesizing 'yuv_filter' ...
INFO: [HLS 200-10] ----------------------------------------------------------------
INFO: [HLS 200-10] -- Implementing module 'rgb2yuv'
INFO: [HLS 200-10] ----------------------------------------------------------------
INFO: [SCHED 204-11] Starting scheduling ...
INFO: [SCHED 204-11] Finished scheduling.
INFO: [HLS 200-111]  Elapsed time: 5.124 seconds; current allocated memory: 94.555 MB.
INFO: [BIND 205-100] Starting micro-architecture generation ...
INFO: [BIND 205-101] Performing variable lifetime analysis.
INFO: [BIND 205-101] Exploring resource sharing.
INFO: [BIND 205-101] Binding ...
INFO: [BIND 205-100] Finished micro-architecture generation.
INFO: [HLS 200-111]  Elapsed time: 0.057 seconds; current allocated memory: 94.755 MB.
INFO: [HLS 200-10] ----------------------------------------------------------------
INFO: [HLS 200-10] -- Implementing module 'yuv2rgb'
INFO: [HLS 200-10] ----------------------------------------------------------------
INFO: [SCHED 204-11] Starting scheduling ...
WARNING: [SCHED 204-21] Estimated clock period (11.2ns) exceeds the target (target clock perio
WARNING: [SCHED 204-21] The critical path consists of the following:
    'mul' operation ('tmp_10', yuv_filter.c:101) (3.36 ns)
    'add' operation ('tmp1', yuv_filter.c:101) (3.02 ns)
    'add' operation ('tmp_12', yuv_filter.c:101) (2.08 ns)
    'icmp' operation ('icmp9', yuv_filter.c:101) (1.36 ns)
    'select' operation ('p_phitmp2', yuv_filter.c:101) (0 ns)
    'select' operation ('G', yuv_filter.c:101) (1.37 ns)
INFO: [SCHED 204-11] Finished scheduling.
```

Figure 3. Synthesizing (Generating) the design



```
INFO: [RTGEN 206-100] Finished creating RTL model for 'yuv_filter'.
INFO: [HLS 200-111]  Elapsed time: 0.368 seconds; current allocated memory: 97.164 MB.
INFO: [RTMG 210-278] Implementing memory 'yuv_filter_p_yuv_hbi_ram' using block RAMs.
INFO: [HLS 200-111] Finished generating all RTL models Time (s): cpu = 00:00:02 ; elapsed = 00
INFO: [SYSC 207-301] Generating SystemC RTL for yuv_filter.
INFO: [VHDL 208-304] Generating VHDL RTL for yuv_filter.
INFO: [VLOG 209-307] Generating Verilog RTL for yuv_filter.
INFO: [HLS 200-112] Total elapsed time: 7.784 seconds; peak allocated memory: 97.164 MB.
```

Figure 4. Generating RTL

1-1-7. Open the created project (in GUI mode) from the Vivado HLS Command Prompt window, by typing **vivado_hls –p yuv_filter.prj**.

The Vivado HLS will open in GUI mode and the project will be opened.

Analyze the Created Project and Results Step 2

2-1. Open the source file and note that three functions are used. Look at the results and observe that the latencies are undefined (represented by ?).

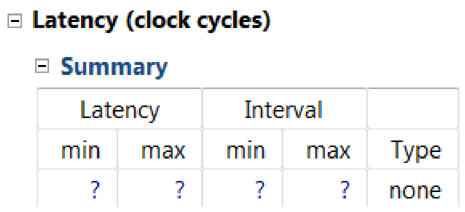
2-1-1. In Vivado HLS GUI, expand the **source** folder in the Explorer view and double-click **yuv_filter.c** to view the content.

- The design is implemented in 3 functions: `rgb2yuv`, `yuv_scale` and `yuv2rgb`.

- Each of these filter functions iterates over the entire source image (which has maximum dimensions specified in `image_aux.h`), requiring a single source pixel to produce a pixel in the result image.
- The scale function simply applies individual scale factors, supplied as top-level arguments to the Y'UV components.
- Notice that most of the variables are of user-defined (`typedef`) and aggregate (e.g. structure, array) types.
- Also notice that the original source used `malloc()` to dynamically allocate storage for the internal image buffers. While appropriate for such large data structures in software, `malloc()` is not synthesizable and is not supported by Vivado HLS.
- A viable workaround is conditionally compiled into the code, leveraging the `__SYNTHESIS__` macro. Vivado HLS automatically defines the `__SYNTHESIS__` macro when reading any code. This ensure the original `malloc()` code is used outside of synthesis but Vivado HLS will use the workaround when synthesizing.

2-1-2. Expand the **syn > report** folder in the Explorer view and double-click **yuv_filter_csynh.rpt** entry to open the synthesis report.

2-1-3. Each of the loops in this design has variable bounds – the width and height are defined by members of input type `image_t`. When variables bounds are present on loops the total latency of the loops cannot be determined: this impacts the ability to perform analysis using reports. Hence, "?" is reported for various latencies.



Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
?	?	?	?	none

Figure 5. Latency computation

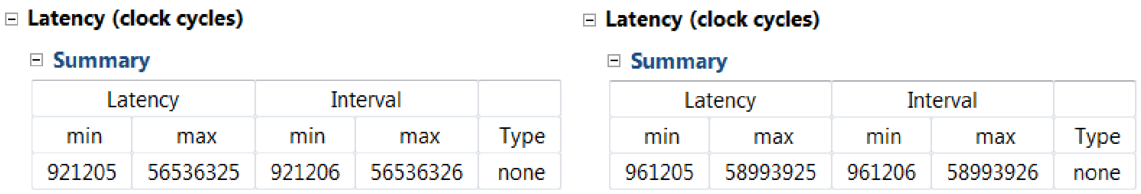
Apply TRIPCOUNT Pragma Step 3

3-1. Open the source file and uncomment pragma lines, re-synthesize, and observe the resources used as well as estimated latencies. Answer the questions listed in the detailed section of this step.

3-1-1. To assist in providing loop-latency estimates, Vivado HLS provides a TRIPCOUNT directive which allows limits on the variables bounds to be specified by the user. In this design, such directives have been embedded in the source code, in the form of #pragma statements.

3-1-2. Uncomment the #pragma lines (50, 53, 90, 93, 130, 133) to define the loop bounds and save the file.

3-1-3. Synthesize the design by selecting **Solution > Run C Synthesis > Active Solution.** View the synthesis report when the process is completed.



Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
921205	56536325	921206	56536326	none

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
961205	58993925	961206	58993926	none

(a) ZedBoard **(b) Zybo**

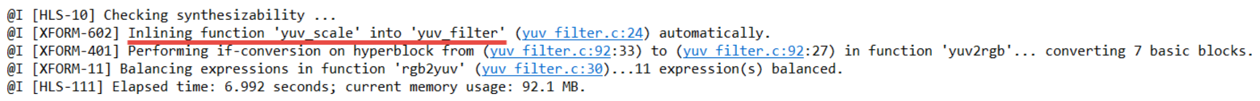
Figure 6. Latency computation after applying TRIPCOUNT pragma

3-1-4. Looking at 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. Scroll the Console window and note that yuv_scale function is automatically inline into the yuv_filter function.



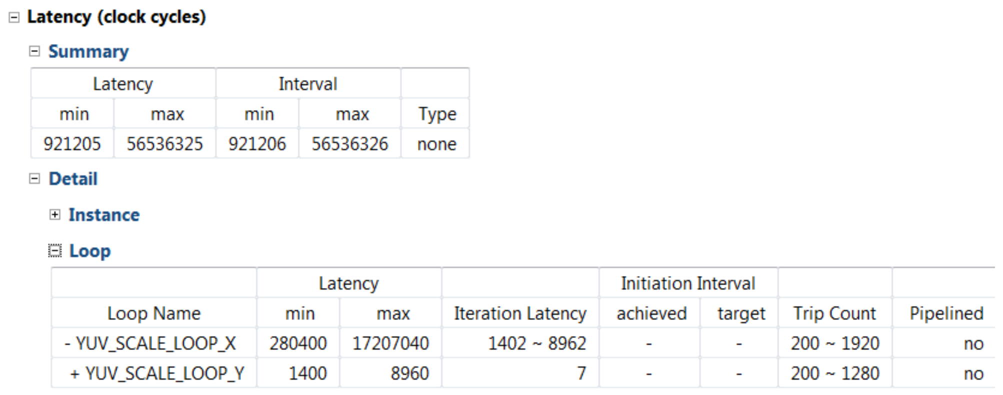
```
@I [HLS-10] Checking synthesizability ...
@I [XFORM-602] Inlining function 'yuv_scale' into 'yuv_filter' (yuv_filter.c:24) automatically.
@I [XFORM-401] Performing if-conversion on hyperblock from (yuv_filter.c:92:33) to (yuv_filter.c:92:27) in function 'yuv2rgb'... converting 7 basic blocks.
@I [XFORM-11] Balancing expressions in function 'rgb2yuv' (yuv_filter.c:30)...11 expression(s) balanced.
@I [HLS-111] Elapsed time: 6.992 seconds; current memory usage: 92.1 MB.
```

Figure 7. Vivado HLS automatically inlining function

3-1-6. Observe that there are three entries – rgb2yuv.rpt, yuv_filter.rpt, and yuv2rgb.rpt under the **syn** report folder in the Explorer view. There is no entry for yuv_scale.rpt since the function was inlined into the yuv_filter function.

You can access lower level module's report by either traversing down in the top-level report under components (under Utilization Estimates > Details > Component) or from the reports container in the project explorer.

3-1-7. Expand the Summary of loop latency and note the latency and trip count numbers for the yuv_scale function. Note that the `YUV_SCALE_LOOP_Y` loop latency is 7X the specified TRIPCOUNT, implying that 7 cycles are used for each of the iteration of the loop.



⊟ **Latency (clock cycles)**

⊟ **Summary**

Latency		Interval		
min	max	min	max	Type
921205	56536325	921206	56536326	none

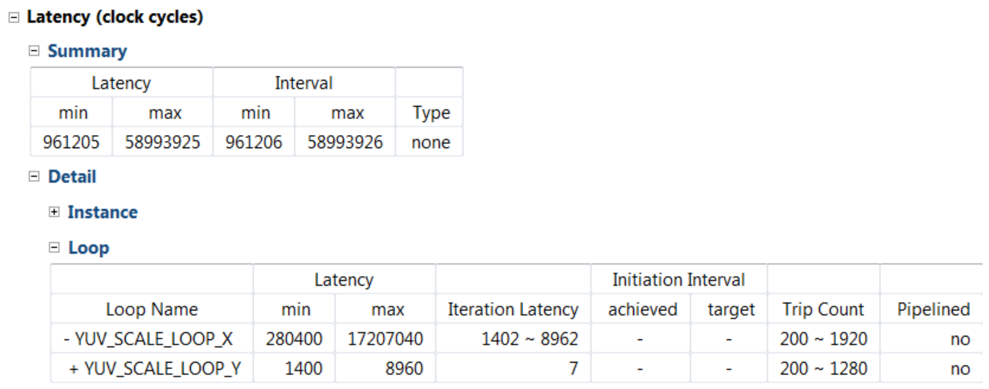
⊟ **Detail**

⊞ **Instance**

⊟ **Loop**

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- YUV_SCALE_LOOP_X	280400	17207040	1402 ~ 8962	-	-	200 ~ 1920	no
+ YUV_SCALE_LOOP_Y	1400	8960	7	-	-	200 ~ 1280	no

(a) ZedBoard



⊟ **Latency (clock cycles)**

⊟ **Summary**

Latency		Interval		
min	max	min	max	Type
961205	58993925	961206	58993926	none

⊟ **Detail**

⊞ **Instance**

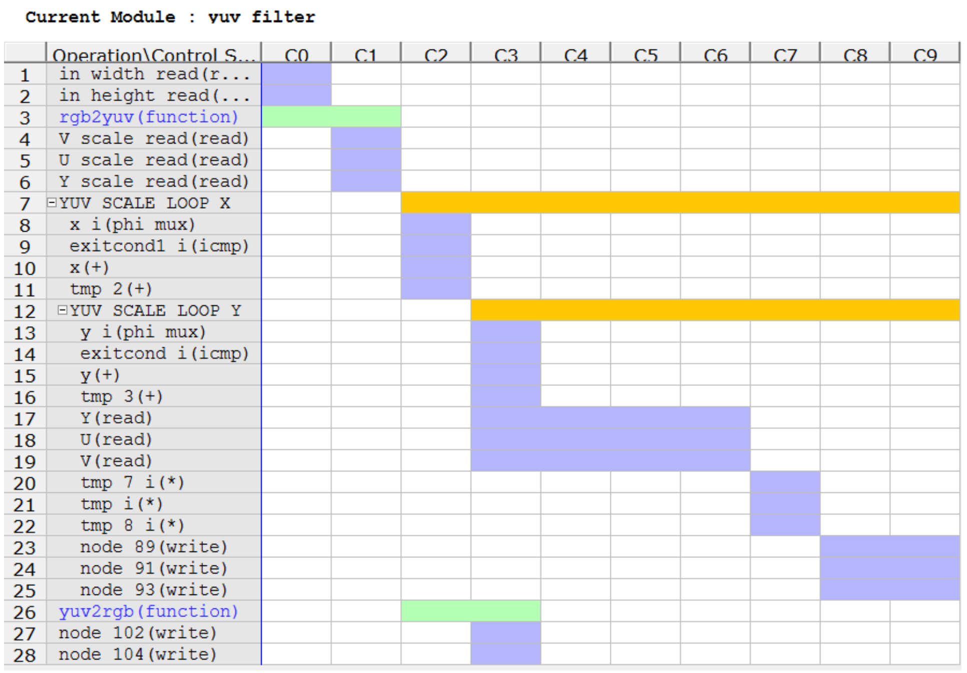
⊟ **Loop**

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- YUV_SCALE_LOOP_X	280400	17207040	1402 ~ 8962	-	-	200 ~ 1920	no
+ YUV_SCALE_LOOP_Y	1400	8960	7	-	-	200 ~ 1280	no

(b) Zybo

Figure 8. Loop latency

3-1-8. You can verify this by opening an analysis perspective view, expanding the **YUV_SCALE_LOOP_X** entry, and then expanding the **YUV_SCALE_LOOP_Y** entry.



Current Module : yuv filter

	Operation\Control S...	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9
1	in width read(r...	🟦									
2	in height read(...	🟦									
3	rgb2yuv(function)	🟩	🟩								
4	V scale read(read)		🟦								
5	U scale read(read)		🟦								
6	Y scale read(read)		🟦								
7	⊟YUV SCALE LOOP X			🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧
8	x i(phi mux)			🟦							
9	exitcond1 i(icmp)			🟦							
10	x(+)			🟦							
11	tmp 2(+)			🟦							
12	⊟YUV SCALE LOOP Y				🟧	🟧	🟧	🟧	🟧	🟧	🟧
13	y i(phi mux)				🟦						
14	exitcond i(icmp)				🟦						
15	y(+)				🟦						
16	tmp 3(+)				🟦						
17	Y(read)				🟦	🟦	🟦	🟦			
18	U(read)				🟦	🟦	🟦	🟦			
19	V(read)				🟦	🟦	🟦	🟦			
20	tmp 7 i(*)								🟦		
21	tmp i(*)								🟦		
22	tmp 8 i(*)								🟦		
23	node 89(write)									🟦	🟦
24	node 91(write)									🟦	🟦
25	node 93(write)									🟦	🟦
26	yuv2rgb(function)			🟩	🟩						
27	node 102(write)				🟦						
28	node 104(write)				🟦						

Figure 9. Design analysis view of the YUV_SCALE_LOOP_Y loop

3-1-9. In the report tab, expand **Detail > Instance** section of the *Utilization Estimates* and click on the **grp_rgb2yuv_fu_244** (rgb2yuv) entry to open the report.

3-1-10. Answer the following question pertaining to rgb2yuv function.

Question 2

Estimated clock period:	____________________________
Worst case latency:	____________________________
Number of DSP48E used:	____________________________
Number of FFs used:	____________________________
Number of LUTs used:	____________________________

3-1-11. Similarly, open the yuv2rgb report.

3-1-12. Answer the following question pertaining to yuv2rgb function.

Question 3

Estimated clock period:	____________________________
Worst case latency:	____________________________
Number of DSP48E used:	____________________________
Number of FFs used:	____________________________
Number of LUTs used:	____________________________

3-1-13. For the rgb2yuv function, in case of ZedBoard, the worst case latency is reported as 17207040 clock cycles. The reported latency can be estimated as follows.

- RGB2YUV_LOOP_Y total loop latency = 8 x 1280 = 10240 cycles
- 1 entry and 1 exit clock for loop RGB2YUV_LOOP_Y = 10242 cycles
- RGB2YUV_LOOP_X loop body latency = 10242 cycles
- RGB2YUV_LOOP_X total loop latency = 10242 x 1920 =19664640 cycles

3-1-14. For the rgb2yuv function, in case of ZYBO, the worst case latency is reported as 19664640 clock cycles. The reported latency can be estimated as follows.

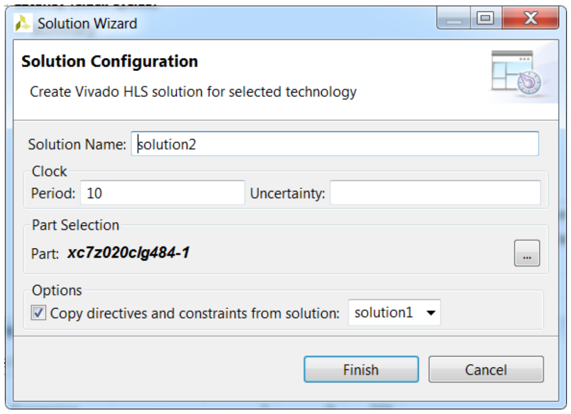
- RGB2YUV_LOOP_Y total loop latency = 8 x 1280 = 10240 cycles
- 1 entry and 1 exit clock for loop RGB2YUV_LOOP_Y = 10242 cycles
- RGB2YUV_LOOP_X loop body latency = 10242 cycles
- RGB2YUV_LOOP_X total loop latency = 10242 x 1920 =19664640 cycles

Turn OFF INLINE and Apply PIPELINE Directive Step 4

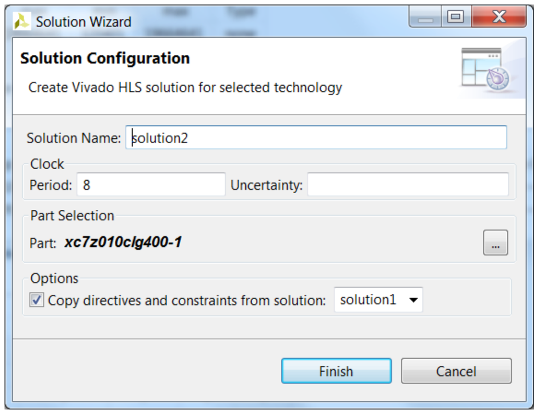
4-1. Create a new solution by copying the previous solution settings. Prevent the automatic INLINE and apply PIPELINE directive. Generate the solution and understand the output.

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. Note that the check boxes of *Copy existing directives from solution* and *Copy custom constraints directives from solution* are checked with Solution1 selected. Click the **Finish** button to create a new solution with the default settings.



Solution Configuration
Create Vivado HLS solution for selected technology
Solution Name: solution2
Clock
Period: 10 Uncertainty:
Part Selection
Part: xc7z020clg484-1
Options
☑ Copy directives and constraints from solution: solution1



Solution Configuration
Create Vivado HLS solution for selected technology
Solution Name: solution2
Clock
Period: 8 Uncertainty:
Part Selection
Part: xc7z010clg400-1
Options
☑ Copy directives and constraints from solution: solution1

(a) ZedBoard **(b) Zybo**

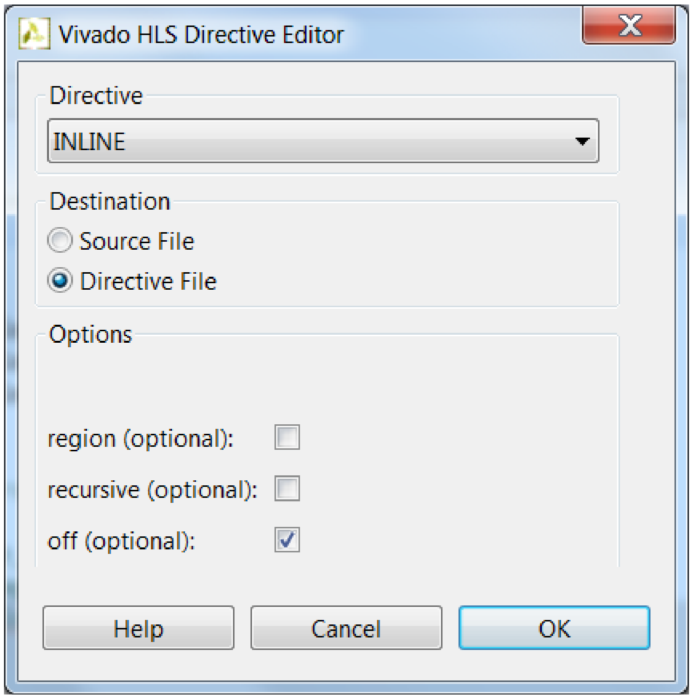
Figure 10. Creating a new Solution after copying the existing solution

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

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

4-1-5. Click on the drop-down button of the *Directive* field. A pop-up menu shows up listing various directives. Select **INLINE** directive.

4-1-6. In the *Vivado HLS Directive Editor* dialog box, click on the **off** option to turn OFF the automatic inlining. Make sure that the Directive File is selected as destination. Click **OK**.



Directive: INLINE

Destination: Source File [✓ Directive File]

region (optional): ☐

recursive (optional): ☐

off (optional): ✓

Figure 11. Turning OFF the inlining function

- o When an object (function or loop) is pipelined, all the loops below it, down through the hierarchy, will be automatically unrolled.
- o In order for a loop to be unrolled it must have fixed bounds: all the loops in this design have variable bounds, defined by an input argument variable to the top-level function.
- o Note that the TRIPCOUNT directive on the loops only influences reporting, it does not set bounds for synthesis.
- o Neither the top-level function nor any of the sub-functions are pipelined in this example.
- o The pipeline directive must be applied to the inner-most loop in each function – the inner-most loops have no variable-bounded loops inside of them which are required to be unrolled and the outer loop will simply keep the inner loop fed with data

4-1-7. Expand the *yuv_scale* in the Directives tab, right-click on *YUV_SCALE_LOOP_Y* object and select insert directives …, and select **PIPELINE** as the directive.

4-1-8. Leave **II** (Initiation Interval) blank as Vivado HLS will try for an II=1, one new input every clock cycle.

4-1-9. Click **OK**.

4-1-10. Similarly, apply the PIPELINE directive to *YUV2RGB_LOOP_Y* and *RGB2YUV_LOOP_Y* objects. At this point, the Directive tab should look like as follows.

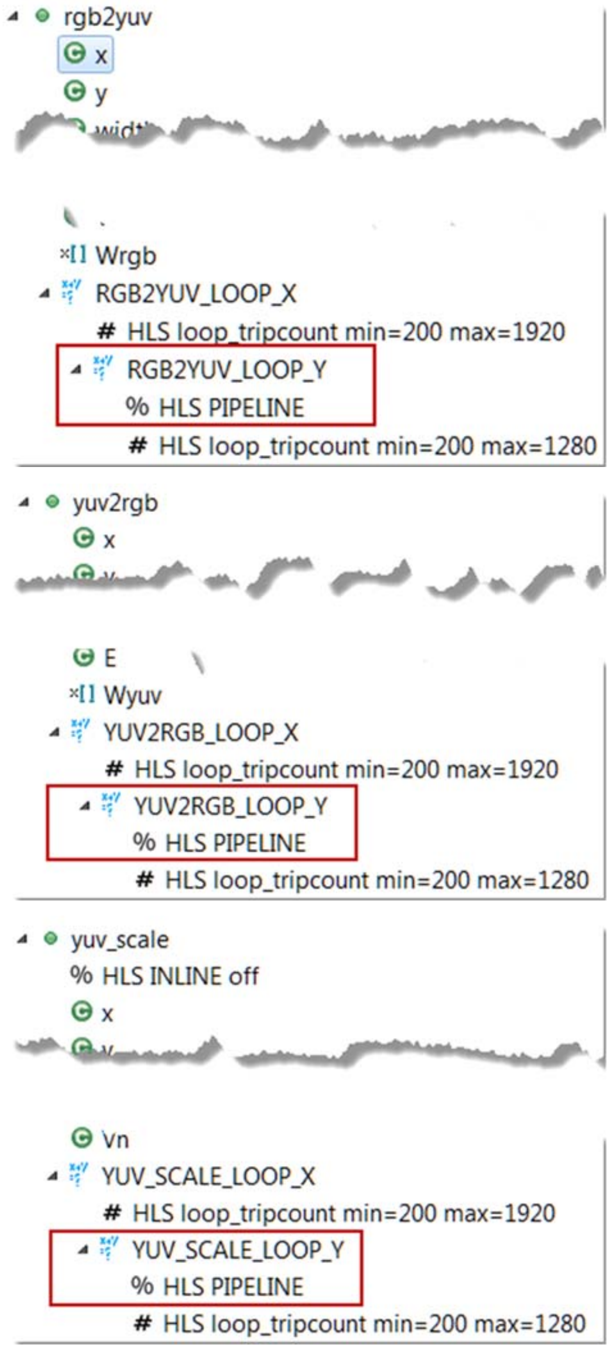

rgb2yuv
x
y
Wrgb
RGB2YUV_LOOP_X
HLS loop_tripcount min=200 max=1920
RGB2YUV_LOOP_Y
% HLS PIPELINE
HLS loop_tripcount min=200 max=1280
yuv2rgb
x
E
Wyuv
YUV2RGB_LOOP_X
HLS loop_tripcount min=200 max=1920
YUV2RGB_LOOP_Y
% HLS PIPELINE
HLS loop_tripcount min=200 max=1280
yuv_scale
% HLS INLINE off
x
Vn
YUV_SCALE_LOOP_X
HLS loop_tripcount min=200 max=1920
YUV_SCALE_LOOP_Y
% HLS PIPELINE
HLS loop_tripcount min=200 max=1280

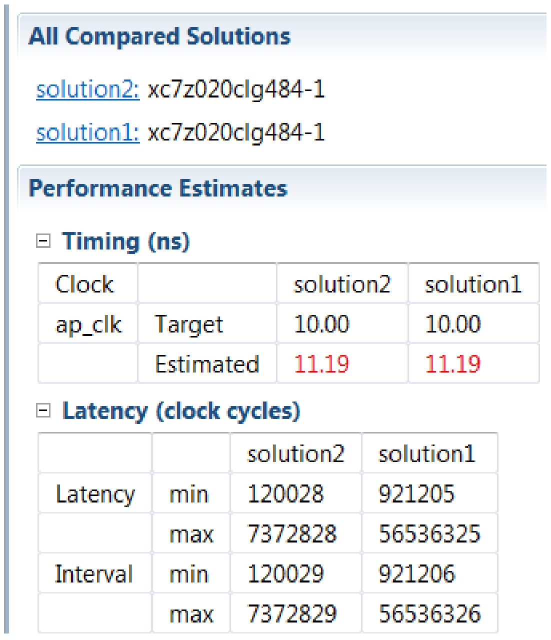
Figure 12. PIPELINE directive applied

4-1-11. Click on the **Synthesis** button.

4-1-12. When the synthesis is completed, select **Project > Compare Reports…** or click on to compare the two solutions.

4-1-13. Select *Solution1* and *Solution2* from the **Available Reports,** and click on the **Add>>** button.

4-1-14. Observe that the latency reduced from 56536325 to 7372831 (ZedBoard), and 58993925 to 7372828 (ZYBO) clock cycles.



All Compared Solutions

solution2: xc7z020clg484-1

solution1: xc7z020clg484-1

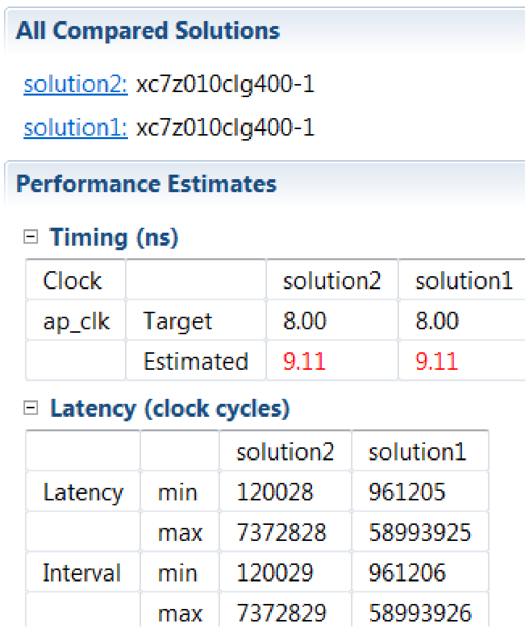
Performance Estimates

Timing (ns)

Clock		solution2	solution1
ap_clk	Target	10.00	10.00
	Estimated	11.19	11.19

Latency (clock cycles)

		solution2	solution1
Latency	min	120028	921205
	max	7372828	56536325
Interval	min	120029	921206
	max	7372829	56536326



All Compared Solutions

solution2: xc7z010clg400-1

solution1: xc7z010clg400-1

Performance Estimates

Timing (ns)

Clock		solution2	solution1
ap_clk	Target	8.00	8.00
	Estimated	9.11	9.11

Latency (clock cycles)

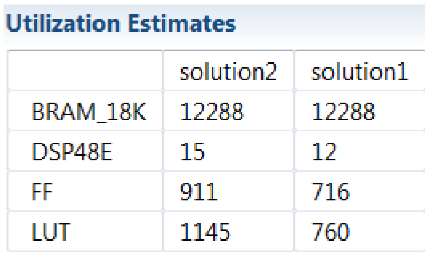
		solution2	solution1
Latency	min	120028	961205
	max	7372828	58993925
Interval	min	120029	961206
	max	7372829	58993926

(a) ZedBoard **(b) Zybo**

Figure 13. Performance comparison after pipelining

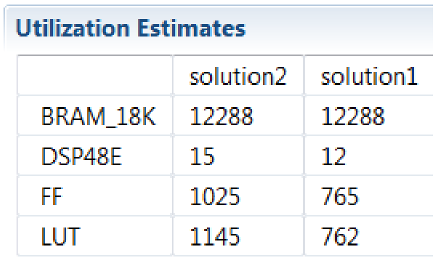
In Solution1, the total loop latency of the inner-most loop was loop_body_latency x loop iteration count, whereas in Solution2 the new total loop latency of the inner-most loop is loop_body_latency + loop iteration count.

4-1-15. Scroll down in the comparison report to view the resources utilization. Observe that the FFs, LUTs, and DSP48E utilization increased whereas BRAM remained same.



Utilization Estimates

	solution2	solution1
BRAM_18K	12288	12288
DSP48E	15	12
FF	911	716
LUT	1145	760



Utilization Estimates

	solution2	solution1
BRAM_18K	12288	12288
DSP48E	15	12
FF	1025	765
LUT	1145	762

(a) ZedBoard **(b) Zybo**

Figure 14. Resources utilization after pipelining

Apply DATAFLOW Directive and Configuration Command Step 5

5-1. Create a new solution by copying the previous solution (Solution2) settings. Apply DATAFLOW directive. Generate the solution and understand 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 Solution2 selected).

5-1-3. Close all inactive solution windows by selecting **Project > Close Inactive Solution Tabs.**

5-1-4. Make sure that the *yuv_filter.c* source is opened in the information pane and select the Directive tab.

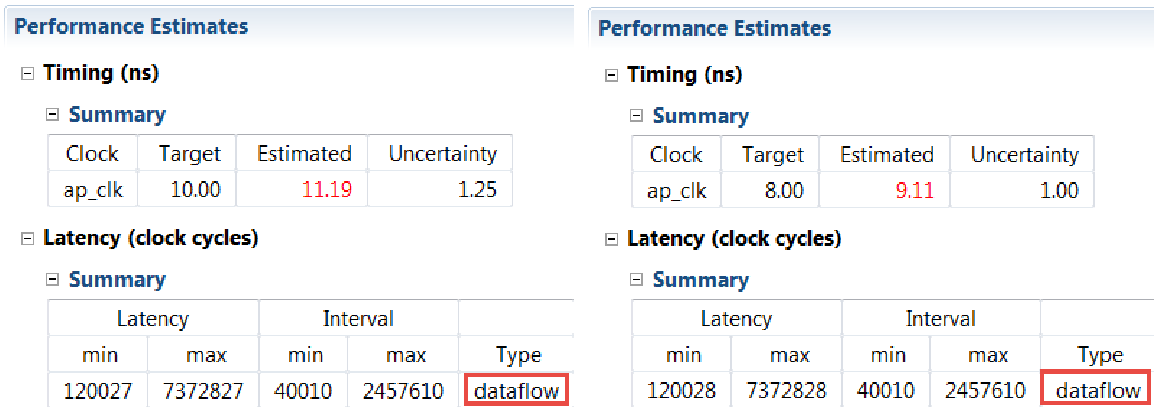
5-1-5. Select function **yuv_filter** in the directives pane, right-click on it and select *Insert Directive...*

5-1-6. A pop-up menu shows up listing various directives. Select **DATAFLOW** directive and click **OK**.

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

5-1-8. When the synthesis is completed, the synthesis report is automatically opened.

5-1-9. Observe additional information, Dataflow Type, in the Performance Estimates section is mentioned.



Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	11.19	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
120027	7372827	40010	2457610	dataflow

(a) ZedBoard

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	8.00	9.11	1.00

Latency (clock cycles)

Summary

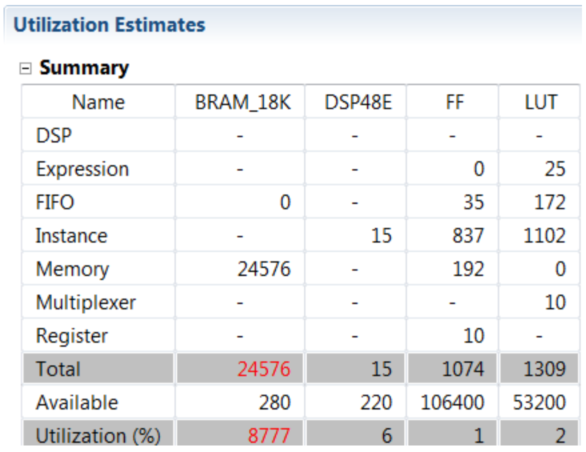
Latency		Interval		
min	max	min	max	Type
120028	7372828	40010	2457610	dataflow

(b) Zybo

Figure 15. Performance estimate after DATAFLOW directive applied

- The Dataflow pipeline throughput indicates the number of clocks cycles between each set of inputs reads. If this throughput value is less than the design latency it indicates the design can start processing new inputs before the currents input data are output.
- While the overall latencies haven't changed significantly, the dataflow throughput is showing that the design can achieve close to the theoretical limit (1920x1280 = 2457600) of processing one pixel every clock cycle.

5-1-10. Scrolling down into the Utilization Estimates, observe that the number of BRAMs required has doubled. This is due to the default dataflow ping-pong buffering.

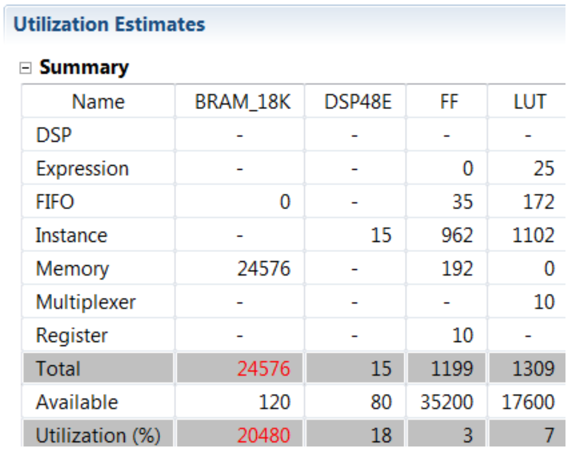


Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	25
FIFO	0	-	35	172
Instance	-	15	837	1102
Memory	24576	-	192	0
Multiplexer	-	-	-	10
Register	-	-	10	-
Total	24576	15	1074	1309
Available	280	220	106400	53200
Utilization (%)	8777	6	1	2

(a) ZedBoard



Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	25
FIFO	0	-	35	172
Instance	-	15	962	1102
Memory	24576	-	192	0
Multiplexer	-	-	-	10
Register	-	-	10	-
Total	24576	15	1199	1309
Available	120	80	35200	17600
Utilization (%)	20480	18	3	7

(b) Zybo

Figure 16. Resource estimate with DATAFLOW directive applied

- When DATAFLOW optimization is performed, memory buffers are automatically inserted between the functions to ensure the next function can begin operation before the previous function has finished. The default memory buffers are ping-pong buffers sized to fully accommodate the largest producer or consumer array.
- Vivado HLS allows the memory buffers to be the default ping-pong buffers or FIFOs. Since this design has data accesses which are fully sequential, FIFOs can be used. Another advantage to using FIFOs is that the size of the FIFOs can be directly controlled (not possible in ping-pong buffers where random accesses are allowed).

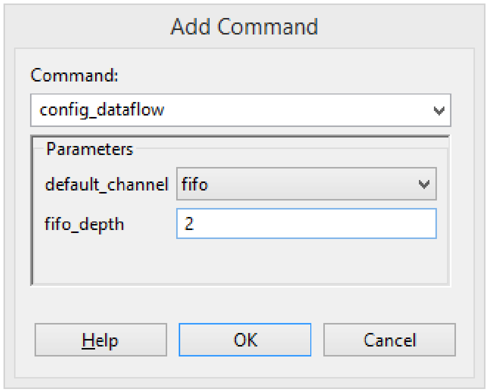
5-1-11. The memory buffers type can be selected using Vivado HLS Configuration command.

5-2. Apply Dataflow configuration command, generate the solution, and observe the improved resources utilization.

5-2-1. Select **Solution > Solution Settings…** or click on to access the configuration command settings.

5-2-2. In the *Configuration Settings* dialog box, select **General** and click the **Add…** button.

5-2-3. Select *config_dataflow* as the command using the drop-down button and **fifo** as the default_channel. Enter **2** as the fifo_depth. Click **OK**.



Add Command

Command: config_dataflow

Parameters

default_channel: fifo

fifo_depth: 2

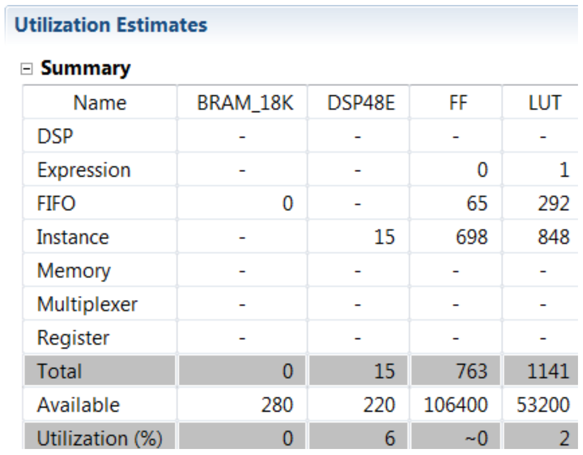
Figure 17. Selecting Dataflow configuration command and FIFO as buffer

5-2-4. Click **OK** again.

5-2-5. Click on the **Synthesis** button.

5-2-6. When the synthesis is completed, the synthesis report is automatically opened.

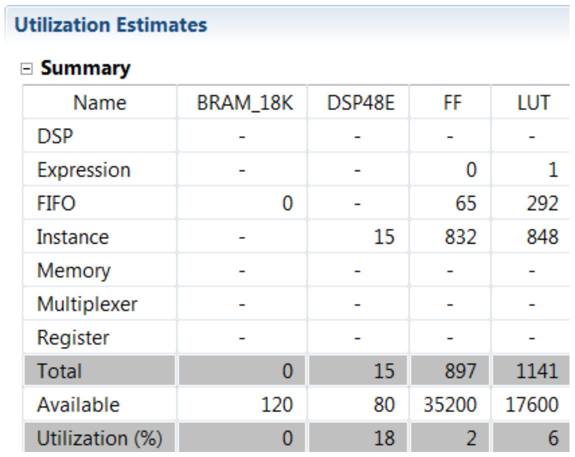
5-2-7. Note that the performance parameter has not changed; however, resource estimates show that the design is not using any BRAM and other resources (FF, LUT) usage has also reduced.



Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	1
FIFO	0	-	65	292
Instance	-	15	698	848
Memory	-	-	-	-
Multiplexer	-	-	-	-
Register	-	-	-	-
Total	0	15	763	1141
Available	280	220	106400	53200
Utilization (%)	0	6	~0	2



Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	1
FIFO	0	-	65	292
Instance	-	15	832	848
Memory	-	-	-	-
Multiplexer	-	-	-	-
Register	-	-	-	-
Total	0	15	897	1141
Available	120	80	35200	17600
Utilization (%)	0	18	2	6

(a) ZedBoard **(b) Zybo**

Figure 18. Resource estimation after Dataflow configuration command

Export and Implement the Design in Vivado HLS Step 6

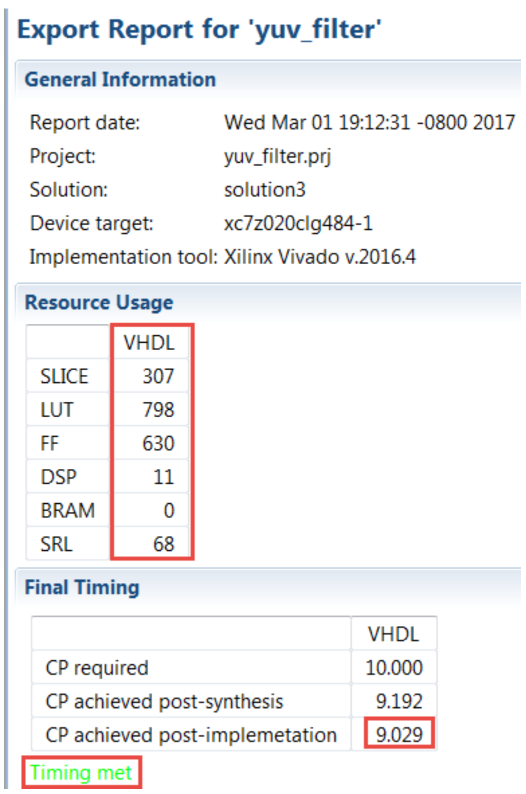
6-1. In Vivado HLS, export the design, selecting VHDL as a language, and run the implementation by selecting Evaluate option.

6-1-1. In Vivado HLS, select **Solution > Export RTL** or click on the button to open the dialog box so the desired implementation can be run.

An Export RTL Dialog box will open.

6-1-2. Click on the drop-down button of the **Evaluate Generated RTL** field and select **VHDL** as the language and click on the check boxes underneath.

6-1-3. Click **OK** and the implementation run will begin. You can observe the progress in the Vivado HLS Console window. When the run is completed the implementation report will be displayed in the information pane.



Export Report for 'yuv_filter'

General Information

Report date:	Wed Mar 01 19:12:31 -0800 2017
Project:	yuv_filter.prj
Solution:	solution3
Device target:	xc7z020clg484-1
Implementation tool:	Xilinx Vivado v.2016.4

Resource Usage

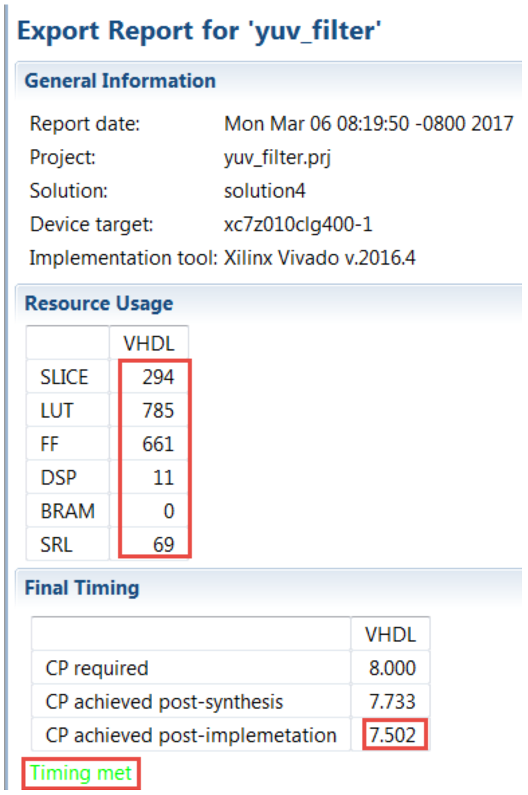
	VHDL
SLICE	307
LUT	798
FF	630
DSP	11
BRAM	0
SRL	68

Final Timing

	VHDL
CP required	10.000
CP achieved post-synthesis	9.192
CP achieved post-implemetation	9.029

Timing met

(a) ZedBoard



Export Report for 'yuv_filter'

General Information

Report date:	Mon Mar 06 08:19:50 -0800 2017
Project:	yuv_filter.prj
Solution:	solution4
Device target:	xc7z010clg400-1
Implementation tool:	Xilinx Vivado v.2016.4

Resource Usage

	VHDL
SLICE	294
LUT	785
FF	661
DSP	11
BRAM	0
SRL	69

Final Timing

	VHDL
CP required	8.000
CP achieved post-synthesis	7.733
CP achieved post-implemetation	7.502

Timing met

(b) Zybo

Figure 19. Implementation results in Vivado HLS

Note that the implementation was successful in case of ZedBoard but failed in case of Zybo.

6-1-4. Close Vivado HLS by selecting **File > Exit.**

Conclusion

In this lab, you learned that even though this design could not be pipelined at the top-level, a strategy of pipelining the individual loops and then using dataflow optimization to make the functions operate in parallel was able to achieve the same high throughput, processing one pixel per clock. When DATAFLOW directive is applied, the default memory buffers (of ping-pong type) are automatically inserted between the functions. Using the fact that the design used only sequential (streaming) data accesses allowed the costly memory buffers associated with dataflow optimization to be replaced with simple 2 element FIFOs using the Dataflow command configuration.

Answers

1. Answer the following questions for yuv_filter:

Estimated clock period:	11.19 ns (ZedBoard) 9.11 ns (Zybo)
Worst case latency:	56536325 (ZedBoard) 58993925 (Zybo) clock cycles
Number of DSP48E used:	12
Number of BRAMs used:	12288
Number of FFs used:	716 (ZedBoard) 765 (Zybo)
Number of LUTs used:	760 (ZedBoard) 762 (Zybo)

2. Answer the following questions rgb2yuv:

Estimated clock period:	8.34 ns (ZedBoard) 8.11 ns (Zybo)
Worst case latency:	19664641 (ZedBoard) 19664641 (Zybo) clock cycles
Number of DSP48E used:	5
Number of FFs used:	198 (ZedBoard) 219 (Zybo)
Number of LUTs used:	253 (ZedBoard) 253 (Zybo)

3. Answer the following questions for yuv2rgb:

Estimated clock period:	11.19 ns (ZedBoard) 9.11 ns (Zybo)
Worst case latency:	19664641 (ZedBoard) 22122241 (Zybo) clock cycles
Number of DSP48E used:	4
Number of FFs used:	203 (ZedBoard) 231 (Zybo)
Number of LUTs used:	238 (ZedBoard) 240 (Zybo)