Vivado HLS Design Flow Lab

Introduction

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

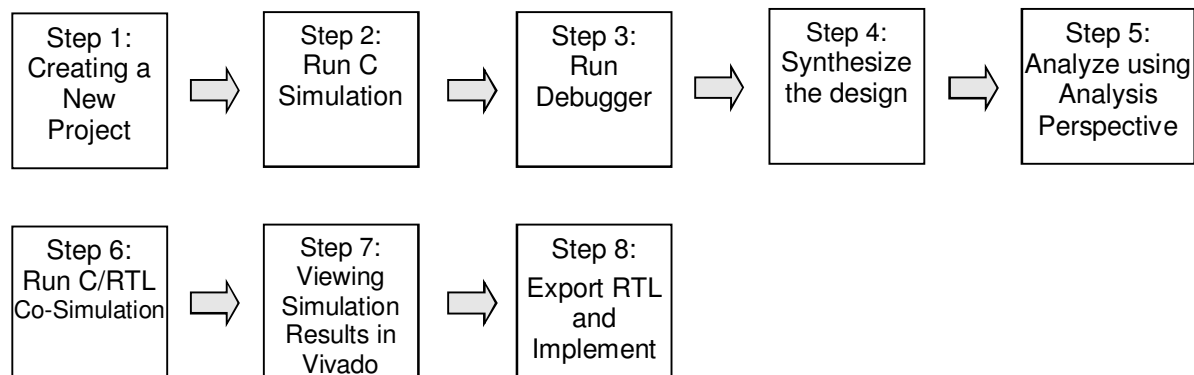## Objectives

After completing this lab, you will be able to:

- Create a new project using Vivado HLS GUI
- Simulate a design
- Synthesize a design
- Implement a design
- Perform design analysis using the Analysis capability of Vivado HLS
- Analyze simulator output using Vivado and XSim simulator

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 8 primary steps: You will create a new project in Vivado HLS, run simulation, run debug, synthesize the design, open an analysis perspective, run SystemC and RTL co-simulation, view simulation results using Vivado and XSim, and export and implement the design in Vivado HLS.

General Flow for this Lab

Step 1: Creating a New Project	Step 2: Run C Simulation	Step 3: Run Debugger	Step 4: Synthesize the design	Step 5: Analyze using Analysis Perspective

Step 6: Run C/RTL Co-Simulation	Step 7: Viewing Simulation Results in Vivado	Step 8: Export RTL and Implement

Create a New Project   Step 1

1-1.   Create a new project in Vivado HLS targeting Zynq xc7z020clg484-1.

1-1-1.   Launch Vivado HLS: Select **Start > All Programs > Xilinx Design Tools > Vivado 2014.2 > Vivado HLS > Vivado HLS 2014.2**

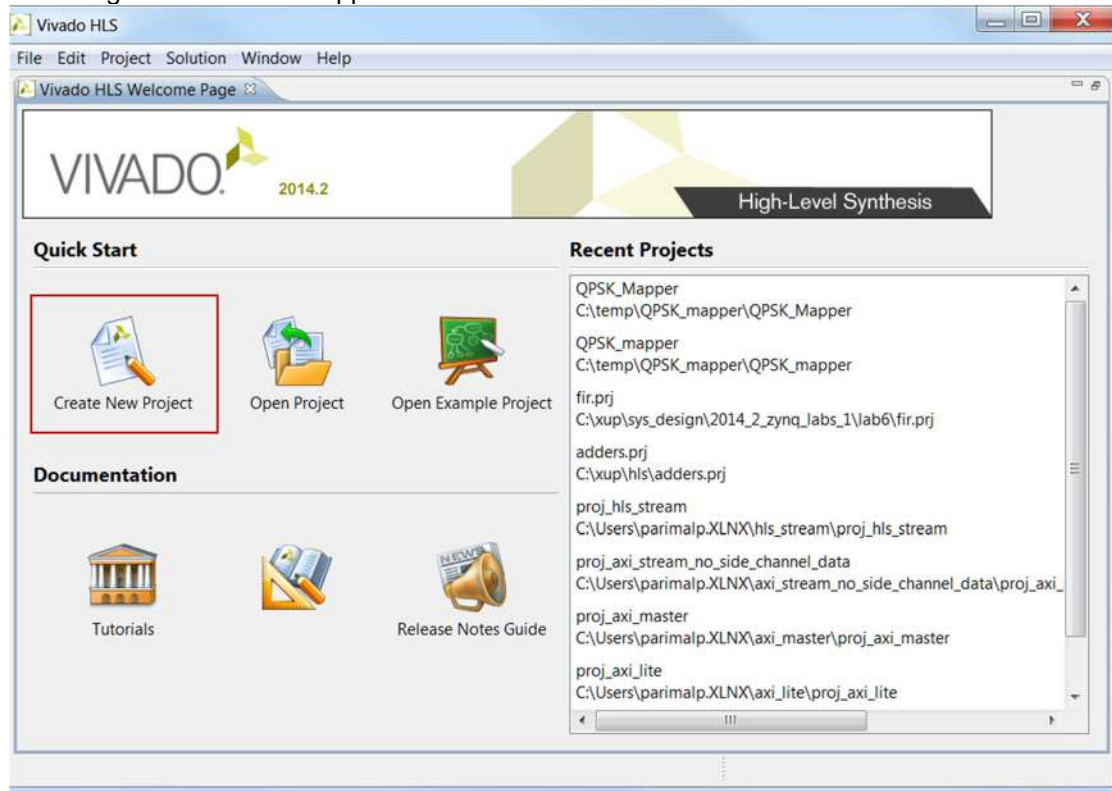A Getting Started GUI will appear.

Figure 1. Getting Started view of Vivado-HLS

1-1-2.   In the Getting Started GUI, click on **Create New Project.** The **New Vivado HLS Project** wizard opens.

1-1-3.   Click the **Browse…** button of the Location field and browse to **c:\xup\hls\labs\lab1** and then click **OK.**
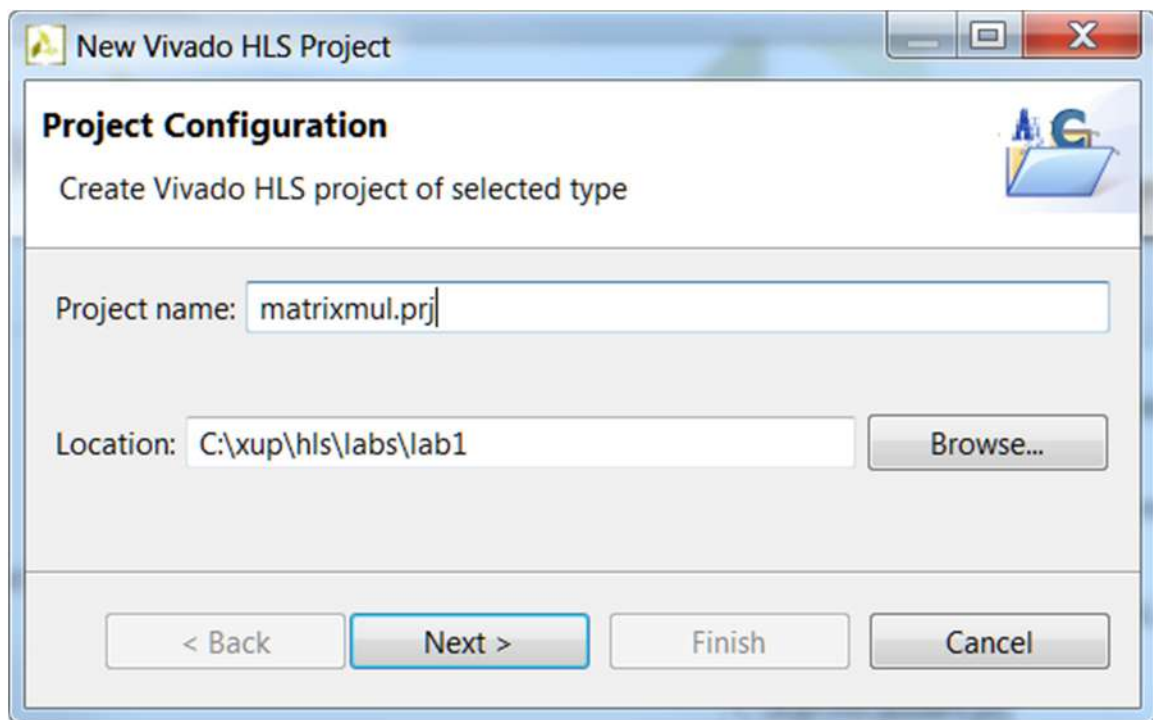
1-1-4.   For Project Name, type **matrixmul.prj**

XILINX

Figure 2. New Vivado HLS Project wizard

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

1-1-6. In the *Add/Remove Files* window, type **matrixmul** as the Top Function name (the provided source file contains the function, to be synthesized, called matrixmul).

1-1-7. Click the **Add Files…** button, select *matrixmul1.cpp* file from the **c:\xup\hls\labs\lab1** folder, and then click **Open**.

1-1-8. Click **Next**.

1-1-9. In the *Add/Remove Files* for the testbench, click the **Add Files…** button, select *matrixmul_test.cpp* file from the **c:\xup\hls\labs\lab1** folder and click **Open.**

1-1-10. Select the matrixmul1_test.cpp in the files list window and click the **Edit CFLAG…** button, type **-DHW_COSIM**, and click **OK**.

1-1-11. Click **Next.**

1-1-12. In the *Solution Configuration* page, leave Solution Name field as **solution1** and clock period as **10**. Leave Uncertainty field blank as it will take 1.25 as the default value.

1-1-13. In the *Device Selection Dialog* page, select *Parts* Specify field, and select the following filters to select the **xc7z020clg484-1** part, and click **OK**:
- o   Family: **Zynq**
- o   Sub-Family: **Zynq**
- o   Package: **clg484**
- o   Speed Grade: **–1**
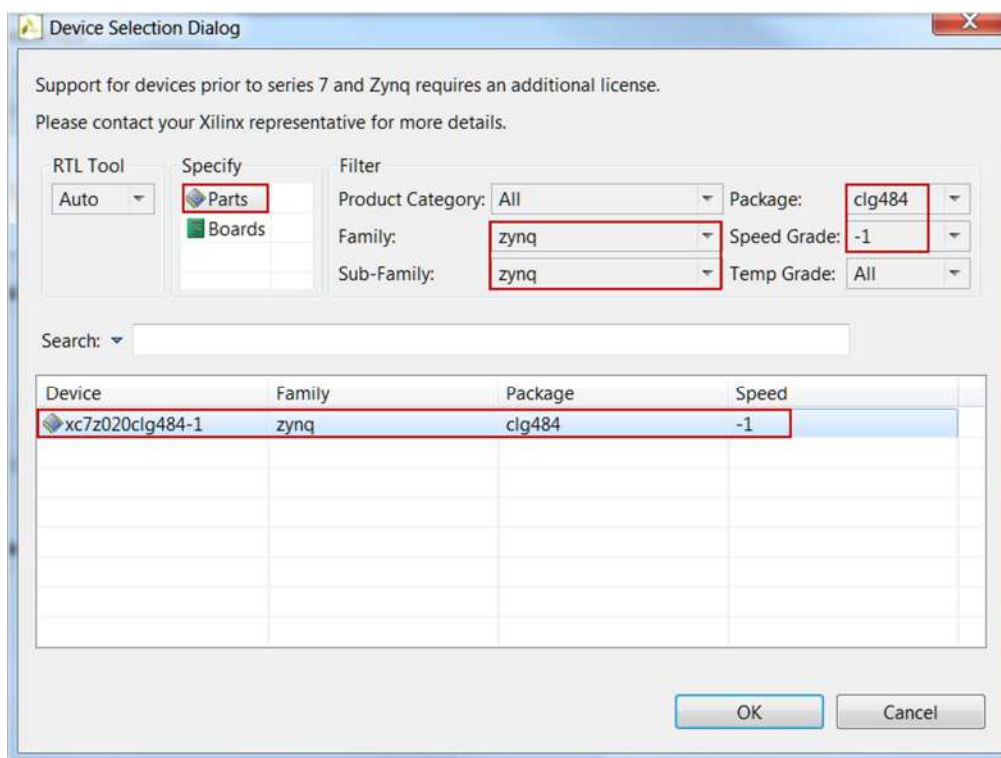
Figure 3. Using Parts Specify option in Part Selection Dialog

You can also select the *Boards* specify option and select one of the listed board if the desired target board is listed.
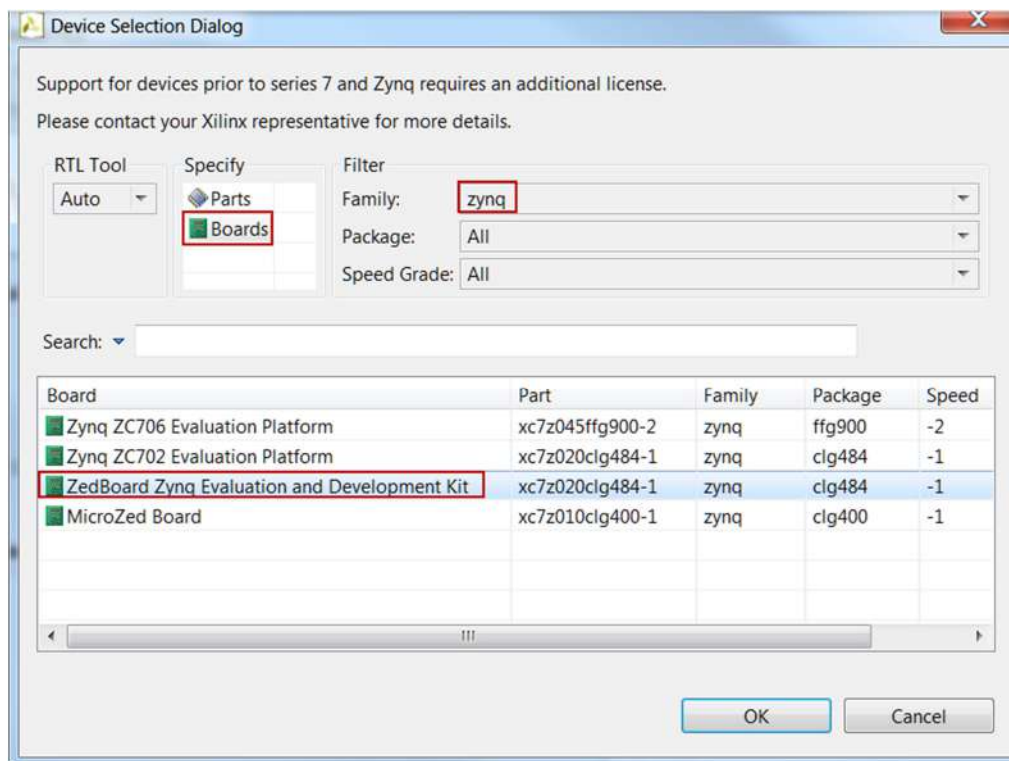
Figure 4. Using Boards Specify option in Part Selection Dialog

1-1-14. Click **Finish.**

You will see the created project in the Explorer view. Expand various sub-folders to see the entries under each sub-folder.
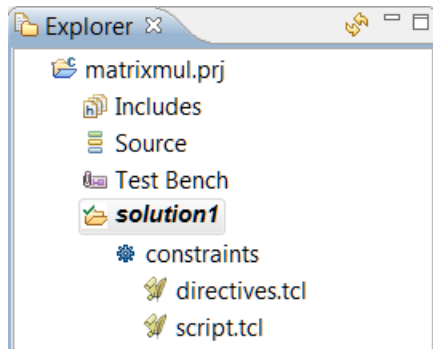

Figure 5. Explorer Window

1-1-15. Double-click on the **matrixmul.cpp** under the source folder to open its content in the information pane.

```
67 #include "matrixmul.h"
68
69 void matrixmul(
70        mat_a_t a[MAT_A_ROWS][MAT_A_COLS],
71        mat_b_t b[MAT_B_ROWS][MAT_B_COLS],
72        result_t res[MAT_A_ROWS][MAT_B_COLS])
73 {
74   // Iterate over the rows of the A matrix
75   Row: for(int i = 0; i < MAT_A_ROWS; i++) {
76     // Iterate over the columns of the B matrix
77     Col: for(int j = 0; j < MAT_B_COLS; j++) {
78       // Do the inner product of a row of A and col of B
79       res[i][j] = 0;
80       Product: for(int k = 0; k < MAT_B_ROWS; k++) {
81         res[i][j] += a[i][k] * b[k][j];
82       }
83     }
84   }
85 }
```

Figure 6. The Design under consideration

It can be seen that the design is a matrix multiplication implementation, consisting of three nested loops. The *Product* loop is the inner most loop performing the actual Matrix elements product and sum. The *Col* loop is the outer-loop which feeds the next column element data with the passed row element data to the Product loop. Finally, *Row* is the outer-most loop. The res[i][j]=0 (line 79) resets the result every time a new row element is passed and new column element is used.
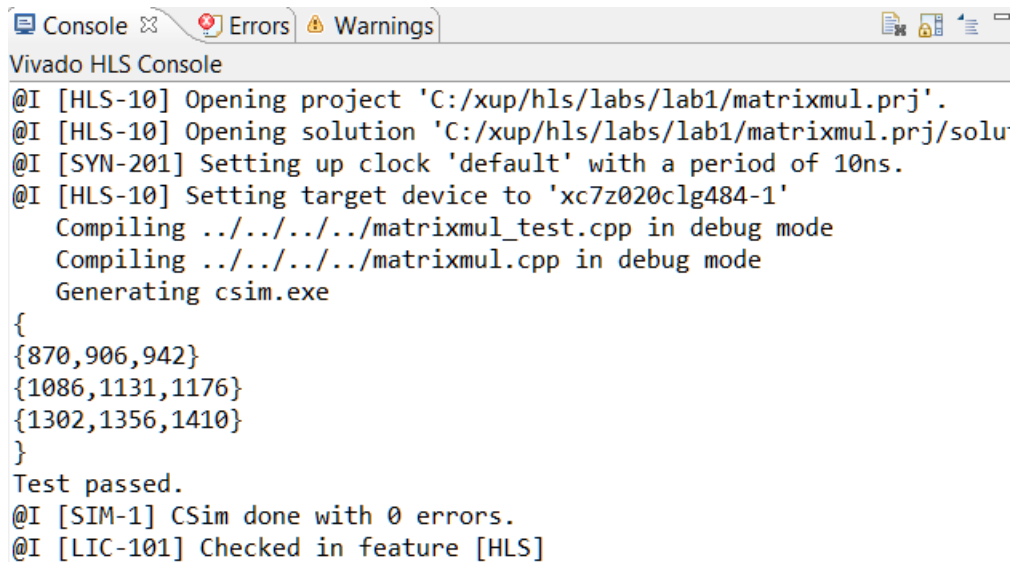
Run C Simulation   Step 2

2-1.   Run C simulation to view the expected output.

2-1-1.   Select **Project > Run C Simulation** or click on ⊡ from the tools bar buttons, and Click **OK** in the *C Simulation Dialog* window.

2-1-2.   The files will be compiled and you will see the output in the Console window.

```
🖳 Console ⊠   🔕 Errors  ⚠ Warnings                              📄 📑 ⁺⣿ ⁀
Vivado HLS Console
@I [HLS-10] Opening project 'C:/xup/hls/labs/lab1/matrixmul.prj'.
@I [HLS-10] Opening solution 'C:/xup/hls/labs/lab1/matrixmul.prj/solu⸱
@I [SYN-201] Setting up clock 'default' with a period of 10ns.
@I [HLS-10] Setting target device to 'xc7z020clg484-1'
   Compiling ../../../../matrixmul_test.cpp in debug mode
   Compiling ../../../../matrixmul.cpp in debug mode
   Generating csim.exe
{
{870,906,942}
{1086,1131,1176}
{1302,1356,1410}
}
Test passed.
@I [SIM-1] CSim done with 0 errors.
@I [LIC-101] Checked in feature [HLS]
```

Figure 7. Program output

2-1-3.   Double-click on **matrixmul_test.cpp** under **testbench** folder in the Explorer to see the content.

You should see two input matrices initialized with some values and then the code executes the algorithm. If HW_COSIM is defined then the matrixmul function is called and compares the output of the computed result with the one returned from the called function, and prints *Test passed* if the results match.

If HW_COSIM had not been defined then it will simply output the computed result and not call the matrixmul1 function.

Run Debugger   Step 3

3-1.   Run the application in debugger mode and understand the behavior of the program.

3-1-1.   Select **Project > Run C Simulation** or click on ⊡ from the tools bar buttons. Select the *Launch Debugger* option and click **OK**.

The application will be compiled with –g option to include the debugging information, the compiled application will be invoked, and the debug perspective will be opened automatically.

⚡ XILINX®

3-1-2. The Debug perspective will show the matrixmul1_test.cpp in the source view, argc and argv variables defined in the Variables view, Outline view showing the objects which are in the current scope, thread created and the program suspended at the main() function entry point.
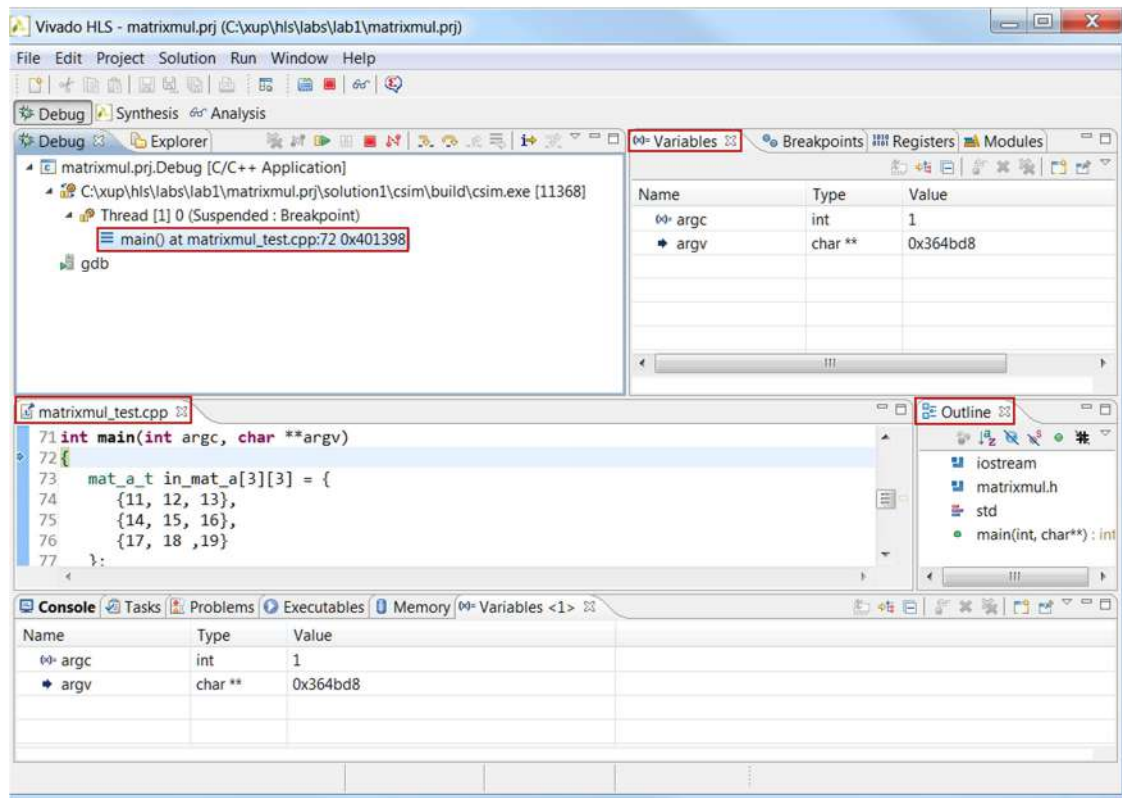
Figure 8. A Debug perspective

3-1-3. Scroll-down in the source view, and double-click in the blue margin at line 105 where it is about to output "{" in the output console window. This will set a break-point at line 105. .

The breakpoint is marked with a blue circle, and a tick

3-1-4. Similarly, set a breakpoint at line 101 on the matrixmul() function

3-1-5. Using the **Step Over (F6)** button ( ) several times, observe the execution progress. Do it for about 19 times and observe the variable values as well as computed software result.

Figure 9. Debugger's intermediate output view

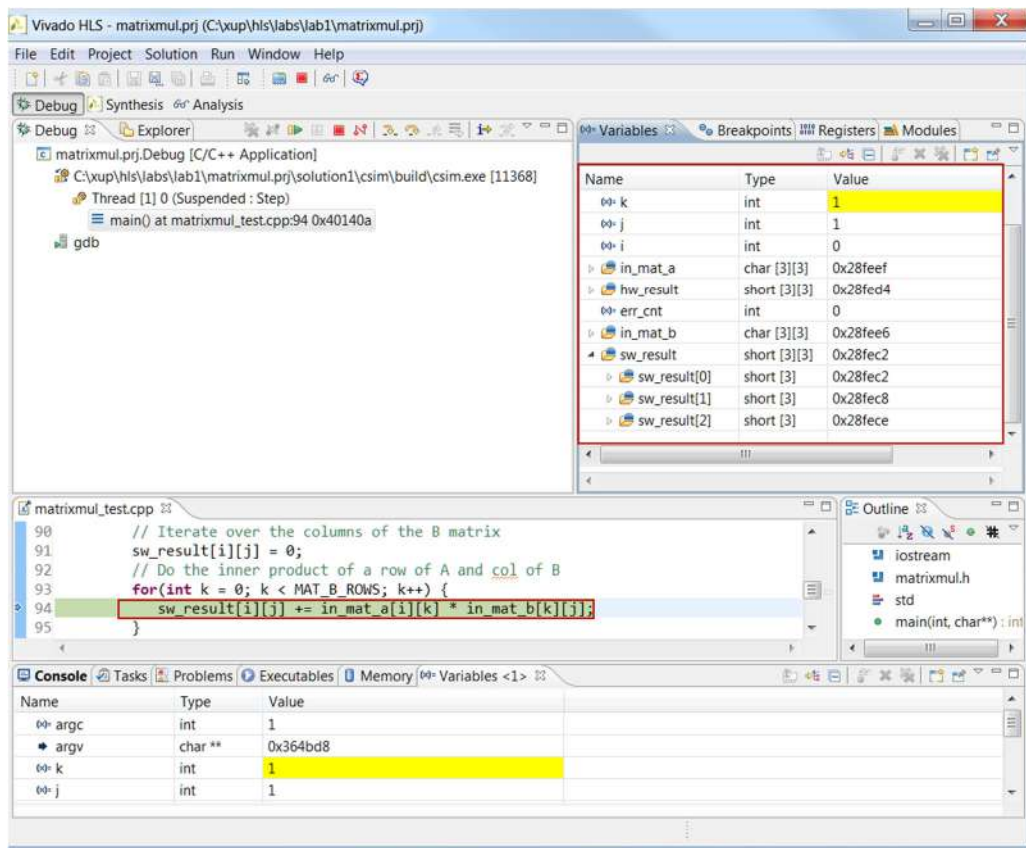**3-1-6.** Now click the **Resume** (▶) button to complete the software computation and stop at line 101.

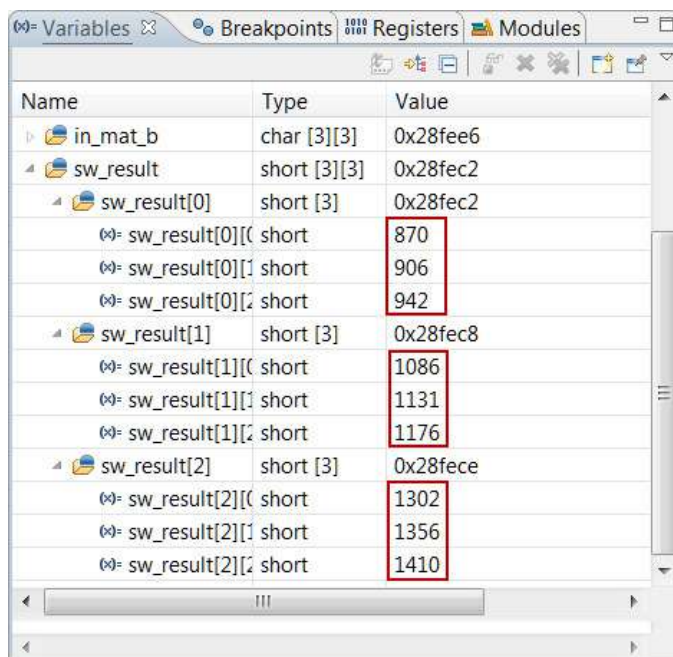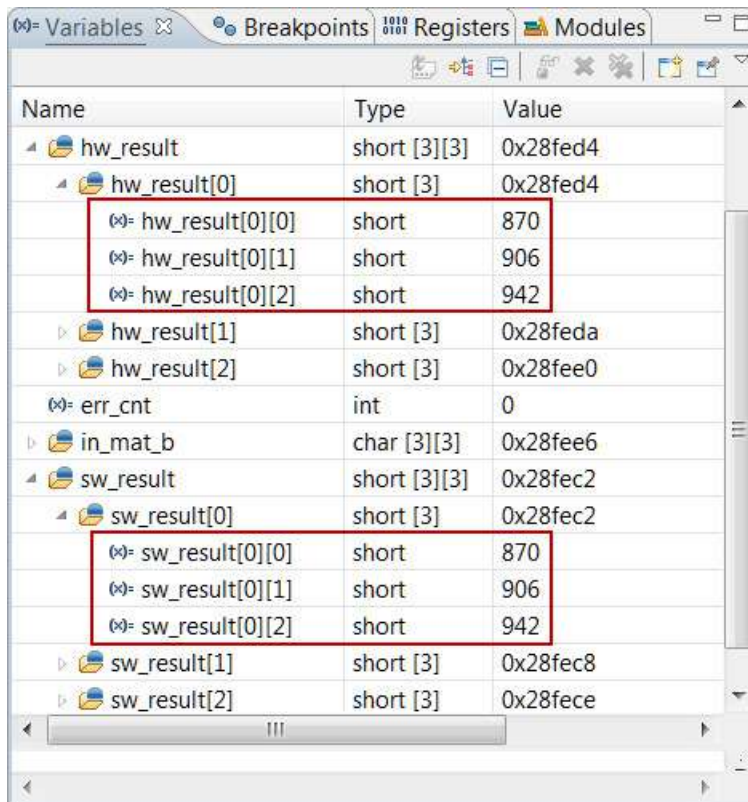**3-1-7.** Observe the following computed software result in the variables view.

Figure 10. Software computed result

XILINX

3-1-8. Click on the **Step Into (F5)** button (🔁) to traverse into the matrixmul module, the one that we will synthesize, and observe that the execution is paused on line 75 of the module.

3-1-9. Using the **Step Over (F6)** several times, observe the computed results. Once satisfied, you can use the **Step Return (F7)** button to return from the function.

3-1-10. The program execution will suspend at line 105 as we had set a breakpoint. Observe the software and hardware (function) computed results in Variables view.

Figure 11. Computed results

3-1-11. Set a breakpoint on line 134 (return err_cnt;), and click on the **Resume** button.

The execution will continue until the breakpoint is encountered. The console window will show the results as seen earlier (**Figure 7**).

3-1-12. Press the **Resume** button or **Terminate** button to finish the debugging session.

Synthesize the Design   Step 4

4-1.   Switch to Synthesis view and synthesize the design with the defaults. View the synthesis results and answer the question listed in the detailed section of this step.

4-1-1. Switch to the Synthesis view by clicking [Synthesis] on the tools bar.

4-1-2.   Select **Solution > Run C Synthesis > Active Solution** or click on the [▶] button to start the synthesis process.

4-1-3.   When synthesis is completed, the Synthesis Results will be displayed along with the Outline pane. Using the Outline pane, one can navigate to any part of the report with a simple click.
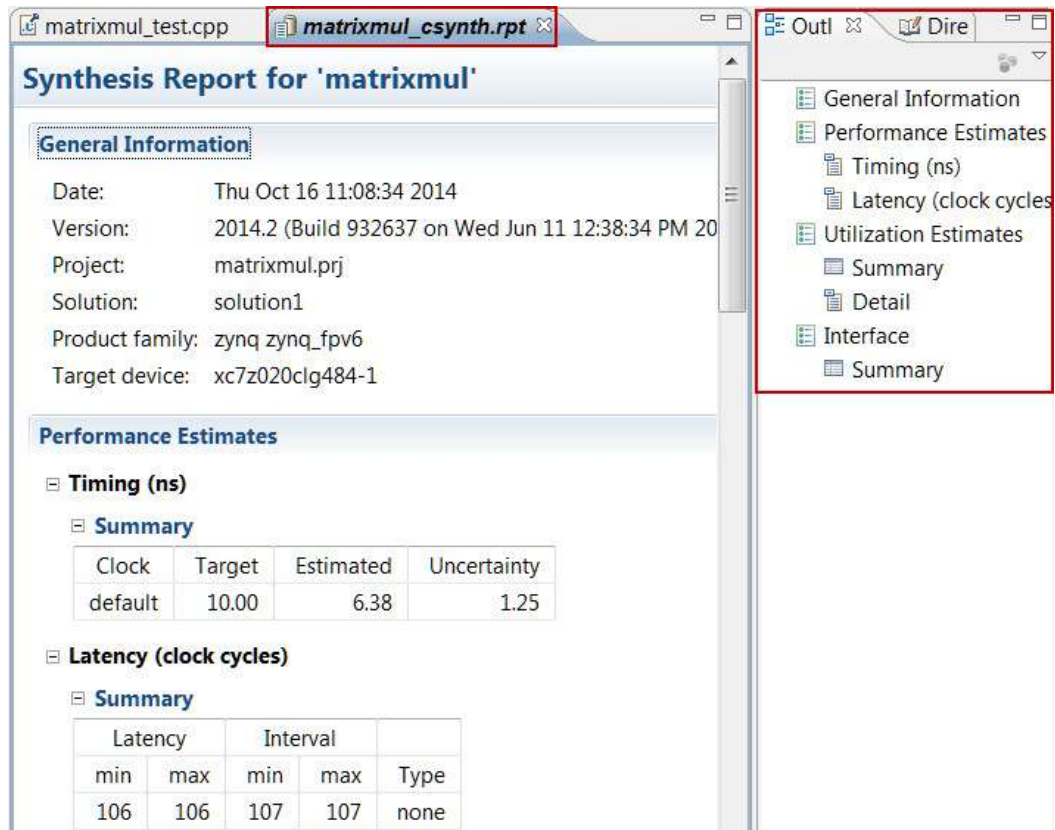
Figure 12. Report view after synthesis is completed

4-1-4.   If you expand **solution1** in Explorer, several generated files including report files will become accessible.
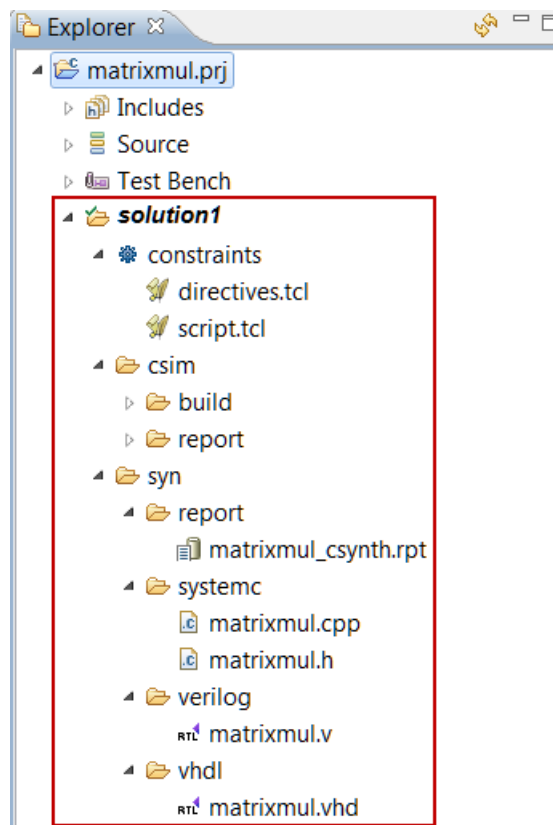
Figure 13. Explorer view after the synthesis process

Note that when the syn folder under the Solution1 folder is expanded in the Explorer view, it will show report, systemC, verilog, and vhdl sub-folders under which report files, and generated source (vhdl, verilog, header, and cpp) files. By double-clicking any of these entries will open the corresponding file in the information pane.

Also note that if the target design has hierarchical functions, reports corresponding to lower-level functions are also created.

4-1-5. The Synthesis Report shows the performance and resource estimates as well as estimated latency in the design.

4-1-6. 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 FFs used: _____
Number of LUTs used: _____

4-1-7. 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	matrixmul	return value
ap_rst	in	1	ap_ctrl_hs	matrixmul	return value
ap_start	in	1	ap_ctrl_hs	matrixmul	return value
ap_done	out	1	ap_ctrl_hs	matrixmul	return value
ap_idle	out	1	ap_ctrl_hs	matrixmul	return value
ap_ready	out	1	ap_ctrl_hs	matrixmul	return value
a_address0	out	4	ap_memory	a	array
a_ce0	out	1	ap_memory	a	array
a_q0	in	8	ap_memory	a	array
b_address0	out	4	ap_memory	b	array
b_ce0	out	1	ap_memory	b	array
b_q0	in	8	ap_memory	b	array
res_address0	out	4	ap_memory	res	array
res_ce0	out	1	ap_memory	res	array
res_we0	out	1	ap_memory	res	array
res_d0	out	16	ap_memory	res	array

Figure 14. Generated interface signals

You can see ap_clk, ap_rst and ap_* control signals are automatically added to every design by default. 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_ready), when the next computation is started (ap_start), and when the computation is completed (ap_done). Other signals are generated based on the design interface itself.

Analyze using Analysis Perspective   Step 5

5-1.   Switch to the Analysis Perspective and understand the design behavior.

5-1-1.   Select **Solution > Open Analysis Perspective** or click on ([⚙ Debug] [Synthesis] [Analysis])to open the analysis viewer.

The Analysis perspective consists of 5 panes as shown below. Note that the module and loops hierarchies are displayed unexpanded by default.

The Module Hierarchy pane shows both the performance and area information for the entire design and can be used to navigate through the hierarchy. The Performance Profile pane is visible and shows the performance details for this level of hierarchy. The information in these two panes is similar to the information reviewed earlier in the synthesis report.

The Performance view is also shown in the right-hand side pane. This view shows how the operations in this particular block are scheduled into clock cycles.

o   The left-hand column lists the resources

o   The top row lists the control states (c0 to c5) in the design. Control states are the internal states used by High-Level Synthesis to schedule operations into clock cycles. There is a

close correlation between the control states and the final states in the RTL Finite State Machine(FSM) but there is no one-to-one mapping
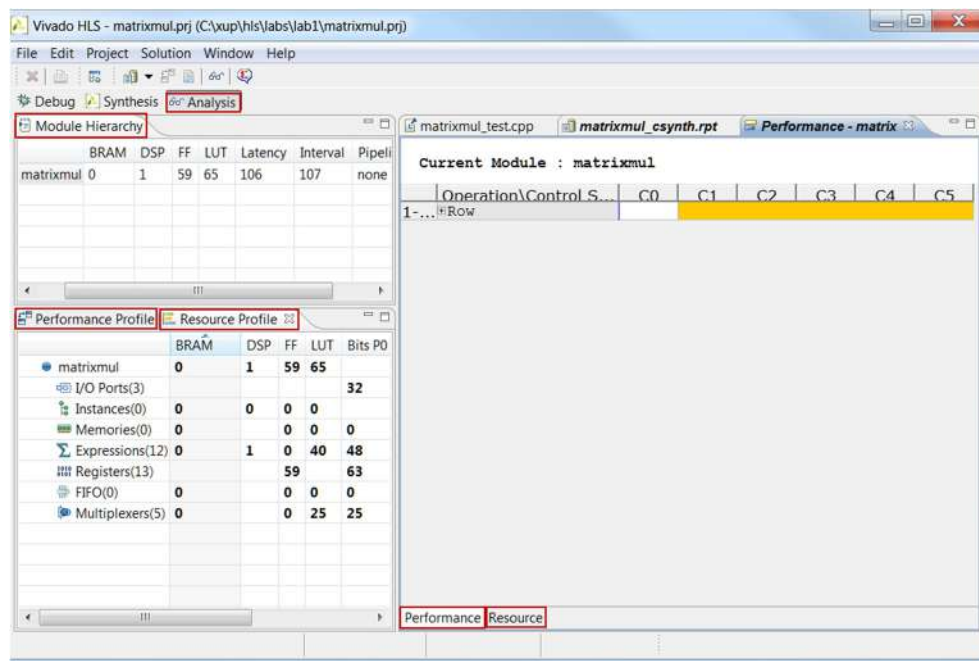
Figure 15. Analysis perspective

5-1-2. Click on loop **Row** to expand, and then click on sub-loops **Col** and **Product** to fully expand the loop hierarchy.
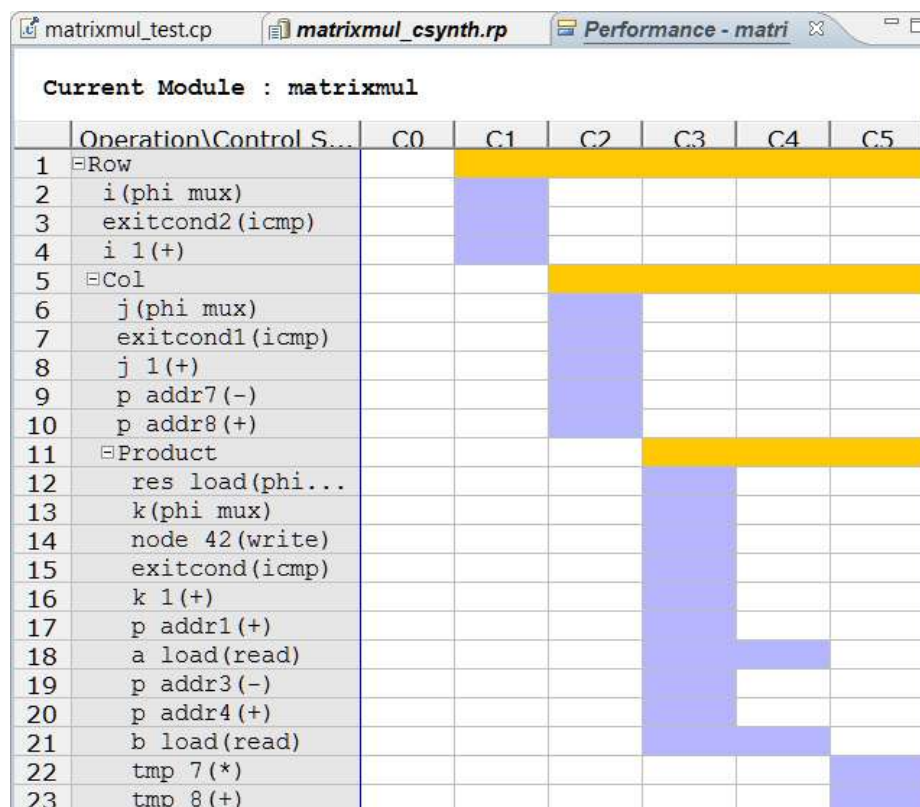
Figure 16. Performance matrix showing top-level Row operation

From this we can see that in the first state (C1) of the Row the loop exit condition is checked and there is an add operation performed. This addition is likely the counter to count the loop iterations, and we can confirm this.

The operations resulting from the loops are colored yellow, the standard operations are colored purple, and sub-blocks will be colored green (in our case we don't have any lower-level functions).

5-1-3. Select the grey block for the adder in state C1, right-click and select *Goto Source*.

The source code pane will be opened, highlighting line 75 where the *Row* loop index is being tested and incremented. In the next state (C2) it starts to execute the *Col* loop.

Figure 17. Cross probing into the source file

5-1-4. In C2, click on the operations (e.g. p_addr7) in the **Col** loop to see the source code highlighting (line 79) update.

5-1-5. Expand the *Performance Profile* hierarchy and note iteration latencies, Trip counts, and overall latencies for each of the nested loops.

Figure 18. The Performance Profile output

The number of iterations can also be noted by holding the mouse over the loop in the Performance view (a dialog box shows the loop statistics).

Operation\Control S…	C0	C1	C2	C3	C4	C5
1 □Row						
2 i(phi mux)						
3 exitcond2(icmp)						
4 i 1(+)						
5 □Col						
6 j(phi mux)						
7 exitcond1(icmp)						
8 j 1(+)						
9 p addr7(-)						
10 p addr8(+)						
11 □Product						
12 res load(phi...						
13 k(phi mux)						
14 node 42(write)						
15 exitcond(icmp)						
16 k 1(+)						
17 p addr1(+)						
18 a load(read)						
19 p addr3(-)						
20 p addr4(+)						
21 b load(read)						
22 tmp 7(*)						
23 tmp 8(+)						

Move mouse over here

Pipelined:	no
Latency:	9
Initiation Interval:	-
Iteration Latency:	3
Trip count:	3
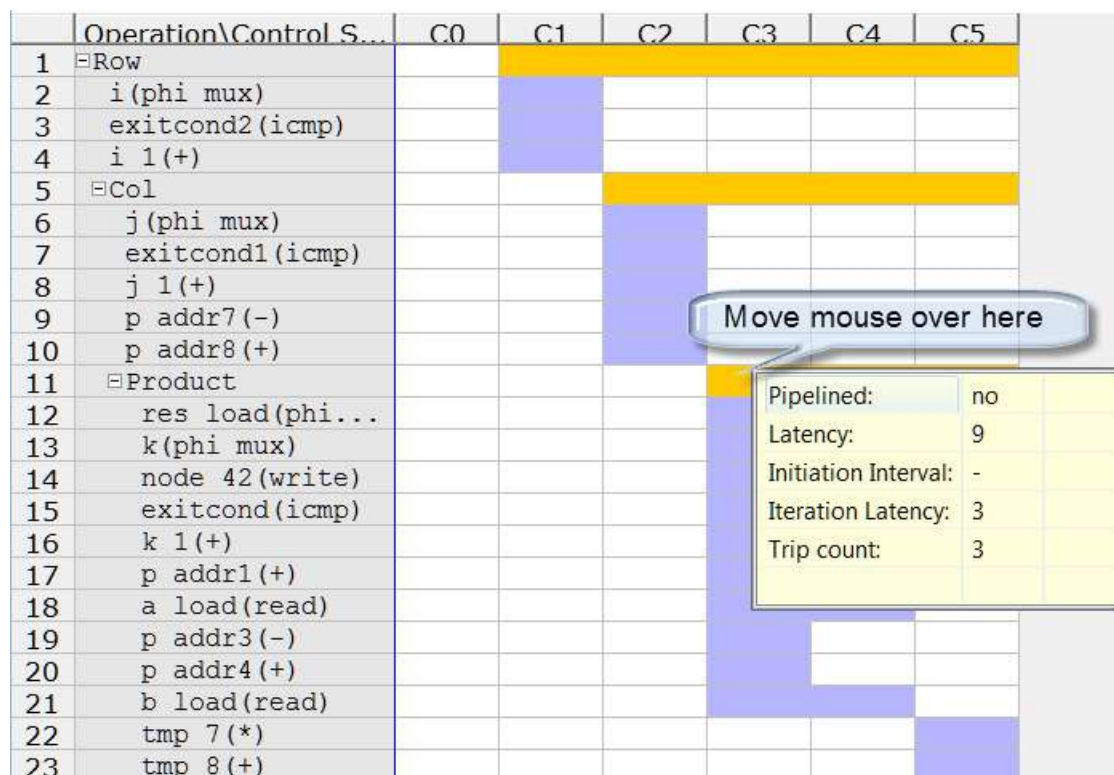
Figure 19. Loop information

Note that the initiation interval does not have a number as this loop is not pipelined.

5-1-6. Click next to the *matrixmul* entry in the **Module Hierarchy** and observe that the entry is not expanded, since there are no lower-level functions defined in the design.

5-1-7. Select the **Resource Profile** tab and observe various resources and where they have been used. You can expand Expressions and Registers sections to see how the resources are being used by which operations.

	BRAM	DSP	FF	LUT	Bits P0	Bits P1	Bits P2
▲ ● matrixmul	0	1	59	65			
▷ ⊕ I/O Ports(3)					32		
⊶ Instances(0)	0	0	0	0			
▦ Memories(0)	0		0	0	0		
▷ Σ Expressions(12)	0	1	0	40	48	45	0
▷ Registers(13)			59		63		
⊞ FIFO(0)	0		0	0	0		
▷ Multiplexers(5)	0		0	25	25		

Figure 20. The Resource Profile tab view

5-1-8. In the Performance Matrix tab, select the Resource tab (at the bottom of the page), and expand **Expressions, I/O Ports,** and **Memory Ports** entries to view the type of operations, resources used, and in which state they are being used.

	Resource\Control Step	C0	C1	C2	C3	C4	C5
1	⊟I/O Ports						
2	res						
3	a						
4	b						
5	⊟Memory Ports						
6	res				write		
7	b				read		
8	a				read		
9	⊟Expressions						
10	exitcond2 fu 123		icmp				
11	i 1 fu 129		+				
12	i phi fu 79	phi_mux					
13	exitcond1 fu 135			icmp			
14	p addr7 fu 167			-			
15	j 1 fu 141			+			
16	p addr8 fu 177			+			
17	j phi fu 91		phi_mux				
18	exitcond fu 192				icmp		
19	p addr3 fu 238				-		
20	p addr1 fu 212				+		
21	p addr4 fu 248				+		
22	k 1 fu 198				+		
23	k phi fu 115				phi_mux		
24	res load phi f...				phi_mux		
25	tmp 7 fu 268						*
26	tmp 8 fu 274						+

Figure 21. The Resource tab

5-1-9. Click on the **Synthesis** tool bar button to switch back to the Synthesis view.

Run C/RTL Co-simulation   Step 6

6-1. Run the C/RTL Co-simulation with the default settings of SystemC. Verify that the simulation passes.

6-1-1. Select **Solution > Run C/RTL Cosimulation** or if you are in the synthesis view, click on the ☑ toolbar button to open the dialog box so the desired simulations can be selected and run.

A C/RTL Co-simulation Dialog box will open. The default option for RTL Co-simulation is to perform the simulation using the SystemC RTL. This allows the simulation to be performed using the built-in C compiler. To perform the verification using Verilog and/or VHDL, you can select the HDL and choose the simulator from the drop-down menu or let the tools use the first simulator that appears in the PATH variable.

XILINX
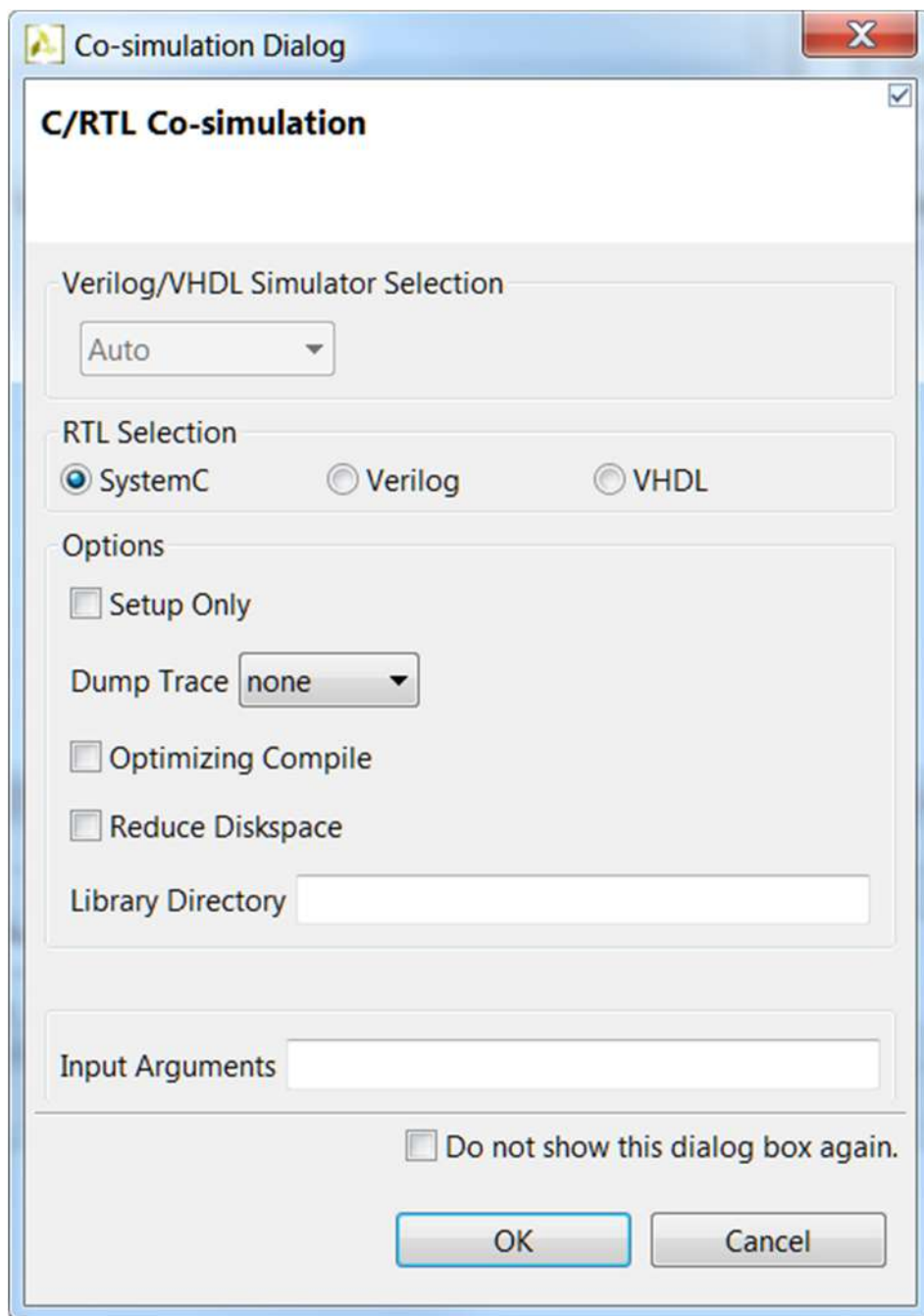
Figure 22. A C/RTL Co-simulation Dialog

6-1-2. Click **OK** to run the SystemC simulation.

The C/RTL Co-simulation will run, generating and compiling several files, and then simulating the design. It goes through three stages.

- First, the C test bench is executed to generate input stimuli for the RTL design

- Second, an RTL test bench with newly generated input stimuli is created and the RTL simulation is then performed

- Finally, the output from the RTL is re-applied to the C test bench to check the results

ZedBoard 1-17

In the console window you can see the progress and also a message that the test is passed. This eliminates writing a separate testbench for the synthesized design.

```
Starting C/RTL cosimulation ...
C:/Xilinx/Vivado_HLS/2014.2/bin/vivado_hls.bat C:/xup/hls/labs/lab1/matrixmul.prj/solution1/cosim.tcl
@I [LIC-101] Checked out feature [HLS]
@I [HLS-10] Running 'C:/Xilinx/Vivado_HLS/2014.2/bin/unwrapped/win64.o/vivado_hls.exe'
           for user 'parimalp' on host 'xsjparimalp30' (Windows NT_amd64 version 6.1) on Thu Oct 16
           in directory 'C:/xup/hls/labs/lab1'
@I [HLS-10] Opening project 'C:/xup/hls/labs/lab1/matrixmul.prj'.
@I [HLS-10] Opening solution 'C:/xup/hls/labs/lab1/matrixmul.prj/solution1'.
@I [SYN-201] Setting up clock 'default' with a period of 10ns.
@I [HLS-10] Setting target device to 'xc7z020clg484-1'
@I [SIM-14] Instrumenting C test bench ...
   Build using "C:/Xilinx/Vivado_HLS/2014.2/msys/bin/g++.exe"
   Compiling apatb_matrixmul.cpp
   Compiling matrixmul.cpp_pre.cpp.tb.cpp
   Compiling matrixmul_test.cpp_pre.cpp.tb.cpp
   Generating cosim.tv.exe
@I [SIM-302] Generating test vectors ...
{
{870,906,942}
{1086,1131,1176}
{1302,1356,1410}
}
Test passed.

@I [SIM-333] Generating C post check test bench ...
@I [SIM-12] Generating RTL test bench ...
   Build using "C:/Xilinx/Vivado_HLS/2014.2/msys/bin/g++.exe"
   Compiling AESL_automem_a.cpp
   Compiling AESL_automem_b.cpp
   Compiling AESL_automem_res.cpp
   Compiling matrixmul.autotb.cpp
   Compiling matrixmul.cpp
   Generating cosim.sc.exe
@I [SIM-11] Starting SystemC simulation ...


           SystemC 2.3.0-ASI --- Jul  4 2013 12:03:34
       Copyright (c) 1996-2012 by all Contributors,
       ALL RIGHTS RESERVED

Note: VCD trace timescale unit is set by user to 1.000000e-012 sec.

Info: /OSCI/SystemC: Simulation stopped by user.
@I [SIM-316] Starting C post checking ...
{
{870,906,942}
{1086,1131,1176}
{1302,1356,1410}
}
Test passed.
@I [SIM-1000] *** C/RTL co-simulation finished: PASS ***
@I [LIC-101] Checked in feature [HLS]
```

Figure 23. Console view showing simulation progress

6-1-3. Once the simulation verification is completed, the simulation report tab will open showing the results. The report indicates if the simulation passed or failed. In addition, the report indicates the measured latency and interval.

Since we have selected only SystemC, the result shows the latencies and interval (initiation) which indicates after how many clock cycles later the next input can be provided. Since the design is not pipelined, it will be latency+1 clock cycles.

matrixmul_csynth.rpt   **Resource - matrixmul**   **matrixmul_cosim.rpt** ☒

Cosimulation Report for 'matrixmul'

Result

RTL	Status	Latency			Interval		
		min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	NA	NA	NA	NA	NA	NA	NA
SystemC	Pass	106	106	106	107	107	107

Export the report(.html) using the Export Wizard

Figure 24. Co-simulation results

Viewing Simulation Results in Vivado   Step 7

7-1.   Run Verilog simulation with Dump Trace option selected.

7-1-1. Select **Solution > Run C/RTL Cosimulation** or click on the ☑ button in the Synthesis view to open the dialog box so the desired simulations can be run.

7-1-2. Click on the **Verilog** RTL Selection option, leaving Verilog/VHDL Simulator Section option to Auto.

Optionally, you can click on the drop-down button and select the desired simulator from the available list of XSim, ISim, ModelSim, and Riviera.

7-1-3. Select *All* for the *Dump Trace* option and click **OK**.
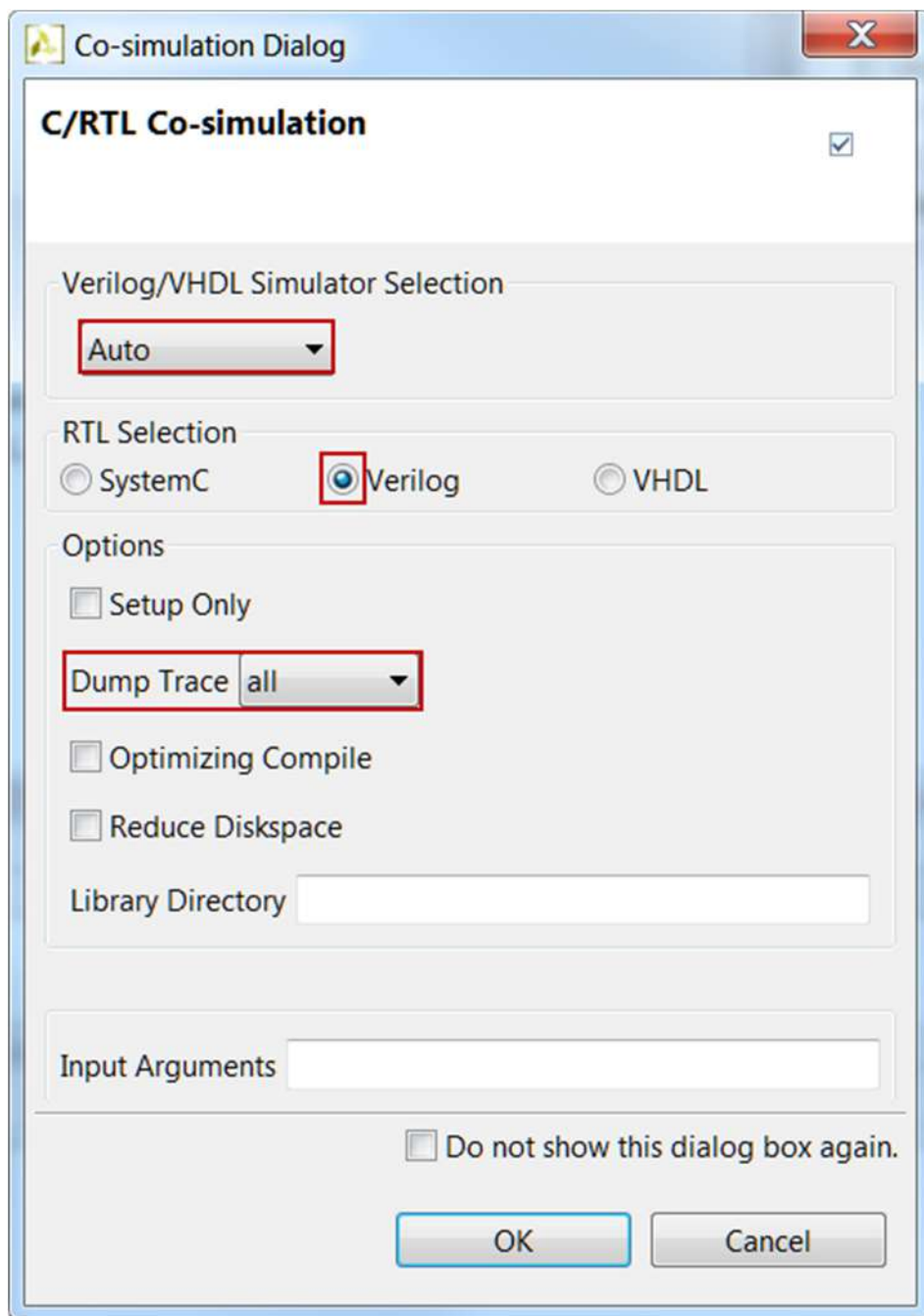
Figure 25. Setting up for Verilog simulation and dump trace

When RTL verification completes the co-simulation report automatically opens showing the Verilog simulation has passed (and the measured latency and interval). In addition, because the Dump Trace option was used and Verilog was selected, two trace files entries can be seen in the Verilog simulation directory
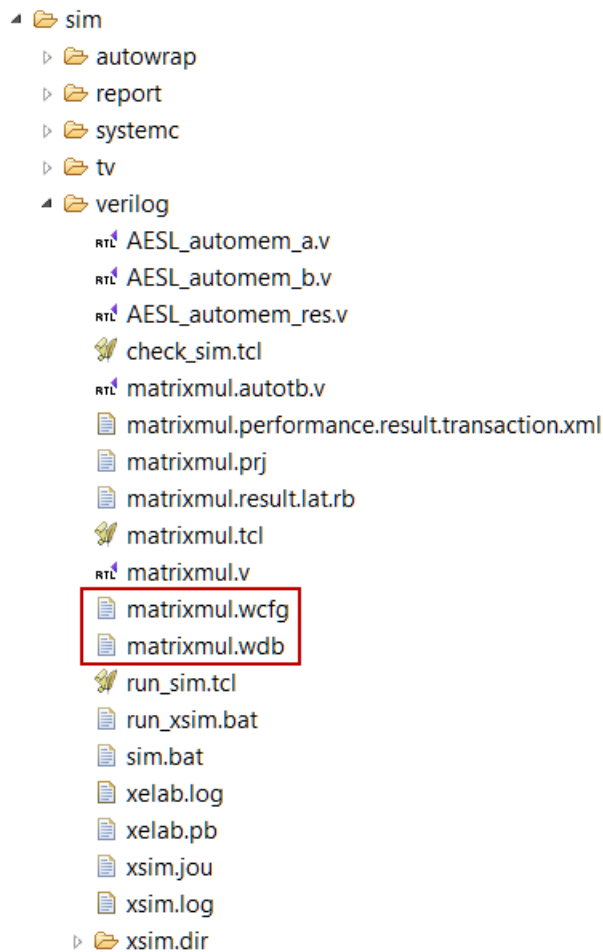
Figure 26. Explorer view after the Verilog RTL co-simulation run

The Cosimulation report shows the test was passed for Verilog along with latency and Interval results. It also shows the SystemC results of the previous run.
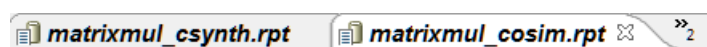
Cosimulation Report for 'matrixmul'

Result

RTL	Status	Latency			Interval		
		min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	Pass	106	106	106	107	107	107
SystemC	Pass	106	106	106	107	107	107

Export the report(.html) using the Export Wizard

Figure 27. Cosimulation report

7-2.   Start Vivado 2014.2 and enter Tcl commands to open and view the dumped traces.

7-2-1.   Select **Start > All Programs > Xilinx Design Tools > Vivado 2014.2 > Vivado 2014.2** to start the Vivado Design Suite program.

7-2-2. In the Vivado Tcl console, enter the following commands one by one:

cd c:/xup/hls/labs/lab1/matrixmul.prj/solution1/sim/Verilog
current_fileset
open_wave_database matrixmul.wdb
open_wave_config matrixmul.wcfg

The above commands will load the project, simulation results, and open the waveform.

7-2-3. In the waveform window, click on the full zoom tool button ( ) to see the entire simulation of one iteration.

7-2-4. Select *a_address0* in the waveform window, right-click and select **Radix > Unsigned Decimal**. Similarly, do the same for b_address0 and res_address0 signals.

7-2-5. Similarly, set the *a_q0, b_q0,* and *res_d0* radix to **Signed Decimal**.

7-2-6. Scroll the waveform little, so you can view the main interface signals (ap_*).
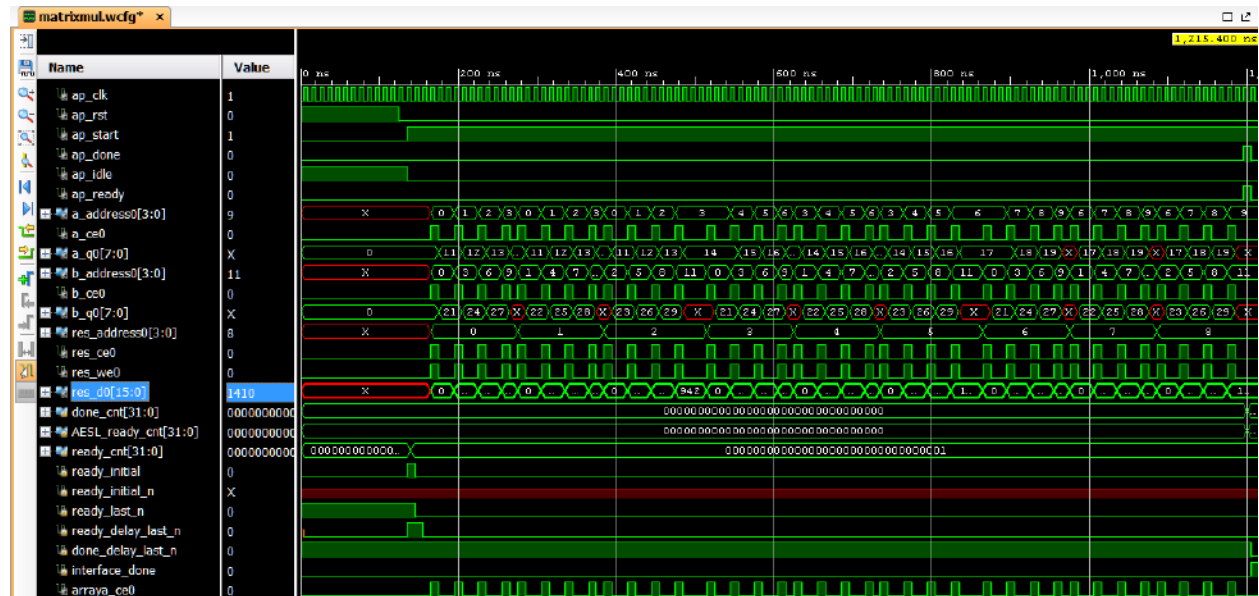
Figure 28. Full waveform showing iteration worth simulation

Simulation run was for 1205 ns. Note that as soon as ap_start is asserted, ap_idle has been de-asserted (at 135 ns) indicating that the design is in computation mode. The ap_idle signal remains de-asserted until ap_done is asserted at 1195 ns, indicating completion of the process. This indicates 106 clock cycles latency (1195 -135 => 1060 ns).

7-2-7. Using Zoom In button, view area of 165 ns and 550 ns.

XILINX
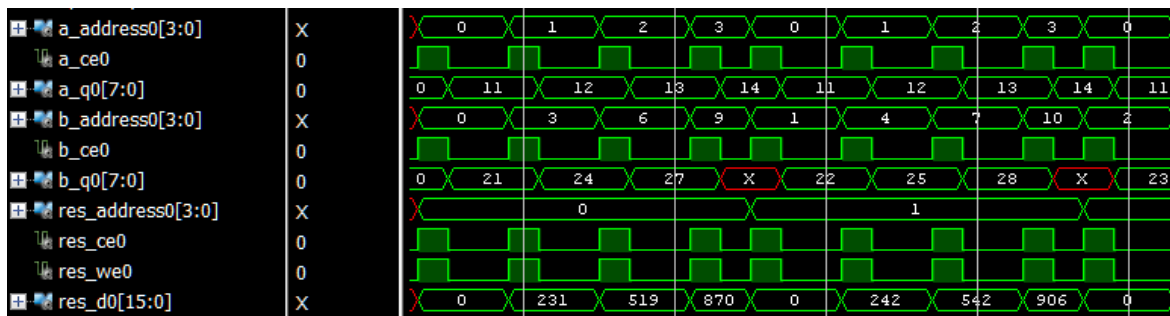
Figure 29. Zoomed view

Observe that the design expects element data by providing a_address0, a_ceo, b_address0, b_ceo signals and outputs result using res_d0, res_we0, and res_ce0.

7-2-8. View various part of the simulation and try to understand how the design works.

7-2-9. When done, close Vivado by selecting **File > Exit.** Click **OK** if prompted, and then **No** to close the program without saving.

Export RTL and Implement   Step 8

8-1.   In Vivado HLS, export the design, selecting VHDL as a language, and run the implementation by selecting Evaluate option.

8-1-1. 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.
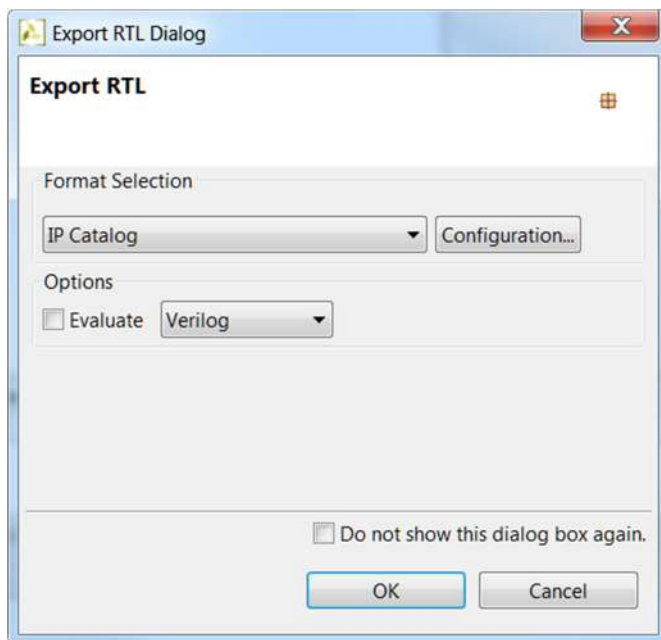
Figure 30. A Export RTL Dialog box

With default settings (shown above), the IP packaging process will run and create a package for the Vivado IP Catalog. Other options, available from the drop-down menu, are to create IP packages for System Generator for DSP/System Generator for DSP using ISE, create a pcore for Xilinx Platform Studio, or create a Synthesized checkpoint.

8-1-2. Click on the drop-down menu of the **Options** field, and select **VHDL** and click on the *Evaluate* check box as the preferred language and to run the implementation tool.

8-1-3. Click **OK** and the implementation run will begin.

You can observe the progress in the Vivado HLS Console window. It goes through several phases:
- o   Exporting RTL as an IP in the IP-XACT format
- o   RTL evaluation, since we selected Evaluate option
  - o   Goes through Synthesis
  - o   Goes through Placement and Routing

```
Starting export RTL ...
C:/Xilinx/Vivado_HLS/2014.2/bin/vivado_hls.bat C:/xup/hls/labs/lab1/matrixmul.prj/solution1
@I [LIC-101] Checked out feature [HLS]
@I [HLS-10] Running 'C:/Xilinx/Vivado_HLS/2014.2/bin/unwrapped/win64.o/vivado_hls.exe'
          for user 'parimalp' on host 'xsjparimalp30' (Windows NT_amd64 version 6.1) on T
          in directory 'C:/xup/hls/labs/lab1'
@I [HLS-10] Opening project 'C:/xup/hls/labs/lab1/matrixmul.prj'.
@I [HLS-10] Opening solution 'C:/xup/hls/labs/lab1/matrixmul.prj/solution1'.
@I [SYN-201] Setting up clock 'default' with a period of 10ns.
@I [HLS-10] Setting target device to 'xc7z020clg484-1'
@I [IMPL-8] Exporting RTL as an IP in IP-XACT.

@I [IMPL-8] Starting RTL evaluation using Vivado ...

C:\xup\hls\labs\lab1\matrixmul.prj\solution1\impl\vhdl>vivado

****** Vivado v2014.2 (64-bit)
  **** SW Build 932637 on Wed Jun 11 13:33:10 MDT 2014
  **** IP Build 924643 on Fri May 30 09:20:16 MDT 2014
    ** Copyright 1986-2014 Xilinx, Inc. All Rights Reserved.

source run_vivado.tcl -notrace
[Thu Oct 16 11:49:32 2014] Launched synth_1...

Implementation tool: Xilinx Vivado v.2014.2
Device target:       xc7z020clg484-1
Report date:         Thu Oct 16 11:51:22 -0700 2014

#=== Resource usage ===
SLICE:          12
LUT:            43
FF:             25
DSP:             1
BRAM:            0
SRL:             0
#=== Final timing ===
CP required:    10.000
CP achieved:    2.921
Timing met
INFO: [Common 17-206] Exiting Vivado at Thu Oct 16 11:51:22 2014...
@I [LIC-101] Checked in feature [HLS]
```

Figure 31. Console view

When the run is completed the implementation report will be displayed in the information pane.
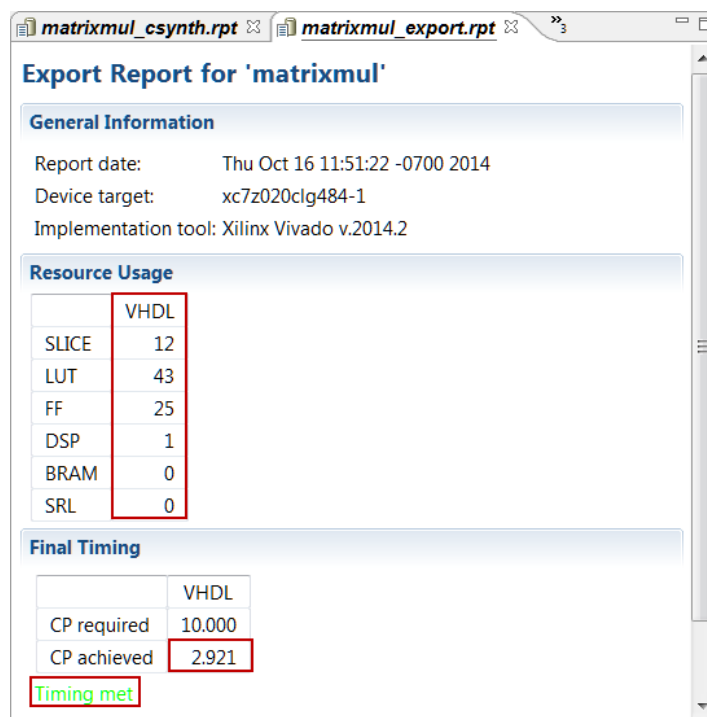
Figure 32. Implementation results in Vivado HLS

Observe that the timing constraint was met, the achieved period (6.176 ns), and the type and amount of resources used.

8-1-4. Collapse the Explorer view and observe that impl folder is created under which ip, report, Verilog, and vhdl sub-folders are created.
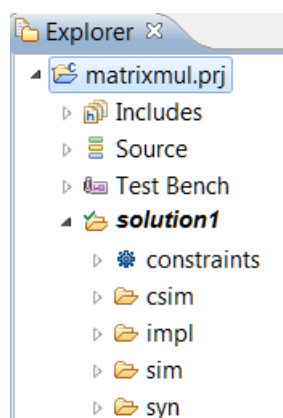
Figure 33. Explorer view after the RTL Export run

8-1-5. Expand the Verilog and vhdl sub-folders and observe that the Verilog sub-folder only has the rtl file whereas vhdl sub-folder has several files and sub-folders as the synthesis and implementation runs were made for it.

It includes project.xpr file (the Vivado project file), matrixmul1.xdc file (timing constraint file), project.runs folder (which includes synth_1 and impl_1 sub-folders created by the synthesis and implementation runs) among others.

```
◢ 🗁 solution1
    ▷ ⚙ constraints
    ▷ 🗁 csim
    ◢ 🗁 impl
        ▷ 🗁 ip
        ▷ 🗁 report
        ▷ 🗁 verilog
        ◢ 🗁 vhdl
            📄 autoimpl.log
            📜 extraction.tcl
            📄 impl.bat
            📄 matrixmul.result.rb
            📄 matrixmul.vhd
            📄 matrixmul.xdc
            📄 project.xpr
            📜 run_vivado.tcl
            📜 settings.tcl
            📄 vivado.jou
            📄 vivado.log
        ▷ 🗁 project.cache
        ◢ 🗁 project.runs
            ▷ 🗁 impl_1
            ▷ 🗁 synth_1
        ◢ 🗁 report
                📑 matrixmul_timing_routed.rpt
                📑 matrixmul_timing_synth.rpt
                📑 matrixmul_utilization_routed.rpt
                📑 matrixmul_utilization_synth.rpt
```

Figure 34. The implementation directory

8-1-6. Expand the ip folder and observe the IP packaged as a zip file (xilinx_com_hls_matrixmul1_1_0.zip), ready for adding to the Vivado IP catalog.

⚡ XILINX

```
▲ 📂 solution1
    ▷ ✲ constraints
    ▷ 📂 csim
    ▲ 📂 impl
        ▲ 📂 ip
            📄 autoimpl.log
            📄 auxiliary.xml
            📄 component.xml
            📄 pack.bat
            🖋 run_ippack.tcl
            📄 vivado.jou
            📄 vivado.log
            📄 xilinx_com_hls_matrixmul_1_0.zip
            📂 bd
            ▷ ✲ constraints
            ▷ 📂 doc
            ▷ 📂 example
            ▷ 📂 hdl
            ▷ 📂 misc
            📂 subcore
            ▷ 📂 xgui
        ▷ 📂 report
        ▷ 📂 verilog
        ▲ 📂 vhdl
```
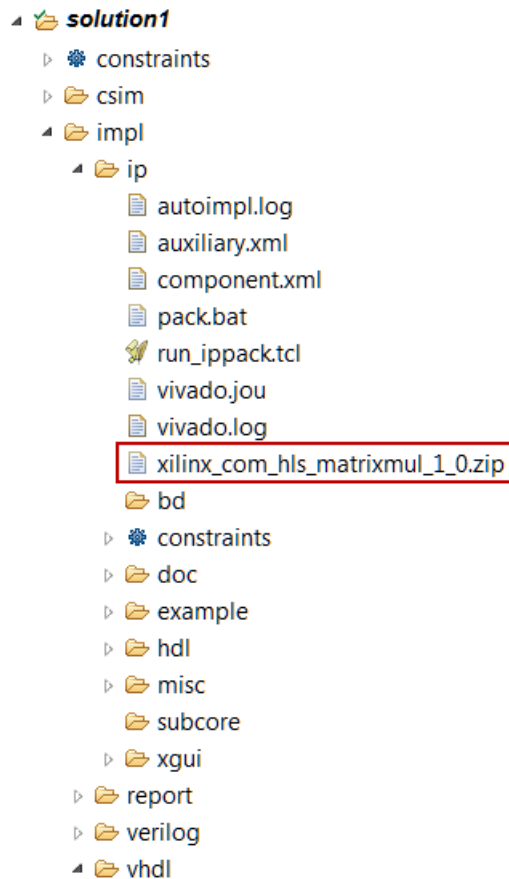
Figure 35. The ip folder content

8-1-7.   Close Vivado HLS by selecting **File > Exit.**

Conclusion

In this lab, you completed the major steps of the high-level synthesis design flow using Vivado HLS. You created a project, adding source files, synthesized the design, simulated the design, and implemented the design. You also learned how to use the Analysis capability to understand the scheduling and binding.

Answers

1.   Answer the following questions:

Estimated clock period: 6.38 ns

Worst case latency: 107 clock cycles

Number of DSP48E used: 1

Number of FFs used: 59

Number of LUTs used: 65

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 TRIPCOUNT directive, apply PIPELINE directive, apply DATAFLOW directive 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
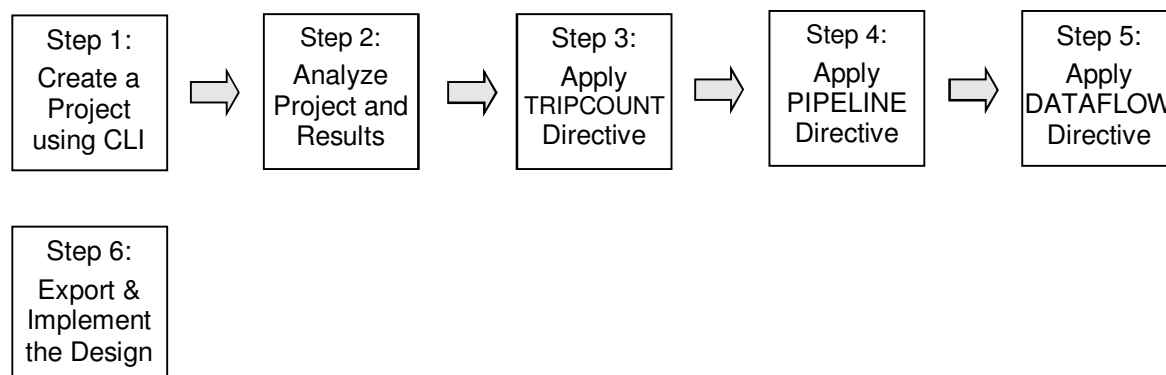
Create a Vivado HLS Project from Command Line   Step 1
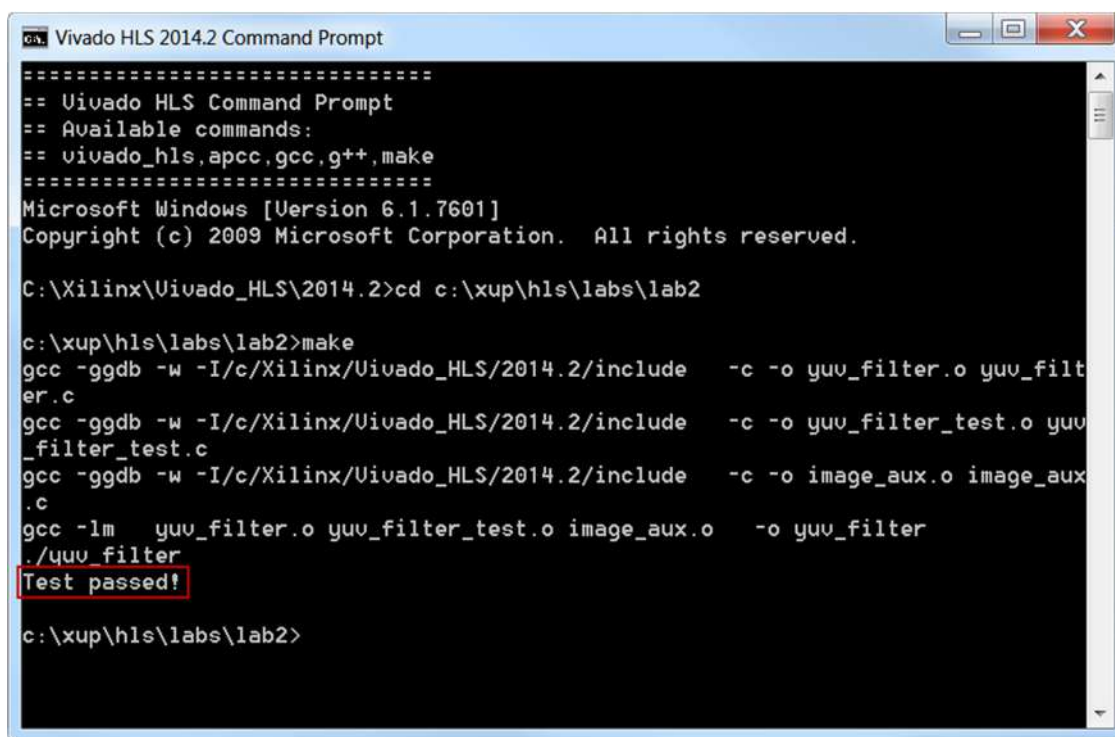
1-1.   Validate your design using Vivado HLS command line window. Create a new Vivado HLS project from the command line.

1-1-1. Launch Vivado HLS: Select **Start > All Programs > Xilinx Design Tools > Vivado 2014.2 > Vivado HLS > Vivado HLS 2014.2 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. In the Vivado HLS Command Prompt, type **make** to compile and execute the program.

Figure 1. Validating the design

Note that the source files (yuv_filter.c, yuv_filter_test.c, and image_aux.c are 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. Type **vivado_hls –f yuv_filter.tcl** in the Vivado HLS Command Prompt window to create the project.

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

1-1-5. 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), setting desired clock period of 10 ns, 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).

```
@I [LIC-101] Checked out feature [HLS]
@I [HLS-10] Running 'C:/Xilinx/Vivado_HLS/2014.2/bin/unwrapped/win64.o/vivado_hls.exe'
            for user 'parimalp' on host 'xsjparimalp30' (Windows NT_amd64 version 6.1)
            in directory 'C:/xup/hls/labs/lab2'
@I [HLS-10] Opening project 'C:/xup/hls/labs/lab2/yuv_filter.prj'.
@I [HLS-10] Adding design file 'yuv_filter.c' to the project
@I [HLS-10] Adding test bench file 'test_data' to the project
@I [HLS-10] Adding test bench file 'yuv_filter_test.c' to the project
@I [HLS-10] Adding test bench file 'image_aux.c' to the project
@I [HLS-10] Opening solution 'C:/xup/hls/labs/lab2/yuv_filter.prj/solution1'.
@I [SYN-201] Setting up clock 'default' with a period of 10ns.
@I [HLS-10] Setting target device to 'xc7z020clg484-1'
@I [HLS-10] Analyzing design file 'yuv_filter.c' ...
@I [HLS-10] Validating synthesis directives ...
@I [HLS-10] Starting code transformations ...
```

Figure 2. Creating project and setting up parameters

```
@I [HLS-10] Starting code transformations ...
@I [HLS-10] Checking synthesizability ...
@I [XFORM-602] Inlining function 'yuv_scale' into 'yuv_filter' (yuv_filter.c
@I [XFORM-401] Performing if-conversion on hyperblock from (yuv_filter.c:92:
@I [XFORM-11] Balancing expressions in function 'rgb2yuv' (yuv_filter.c:30).
@I [HLS-111] Elapsed time: 3.35 seconds; current memory usage: 69.1 MB.
@I [HLS-10] Starting hardware synthesis ...
@I [HLS-10] Synthesizing 'yuv_filter' ...
@I [HLS-10] -----------------------------------------------------------------
@I [HLS-10] -- Scheduling module 'yuv_filter_rgb2yuv'
@I [HLS-10] -----------------------------------------------------------------
@I [SCHED-11] Starting scheduling ...
@I [SCHED-11] Finished scheduling.
@I [HLS-111] Elapsed time: 0.21 seconds; current memory usage: 70.9 MB.
@I [HLS-10] -----------------------------------------------------------------
@I [HLS-10] -- Exploring micro-architecture for module 'yuv_filter_rgb2yuv'
@I [HLS-10] -----------------------------------------------------------------
@I [BIND-100] Starting micro-architecture generation ...
@I [BIND-101] Performing variable lifetime analysis.
@I [BIND-101] Exploring resource sharing.
@I [BIND-101] Binding ...
@I [BIND-100] Finished micro-architecture generation.
@I [HLS-111] Elapsed time: 0.04 seconds; current memory usage: 70.9 MB.
@I [HLS-10] -----------------------------------------------------------------
@I [HLS-10] -- Scheduling module 'yuv_filter_yuv2rgb'
@I [HLS-10] -----------------------------------------------------------------
@I [SCHED-11] Starting scheduling ...
@I [SCHED-11] Finished scheduling.
@I [HLS-111] Elapsed time: 0.14 seconds; current memory usage: 71.8 MB.
@I [HLS-10] -----------------------------------------------------------------
@I [HLS-10] -- Exploring micro-architecture for module 'yuv_filter_yuv2rgb'
@I [HLS-10] -----------------------------------------------------------------
@I [BIND-100] Starting micro-architecture generation ...
@I [BIND-101] Performing variable lifetime analysis.
@I [BIND-101] Exploring resource sharing.
@I [BIND-101] Binding ...
@I [BIND-100] Finished micro-architecture generation.
@I [HLS-111] Elapsed time: 0.04 seconds; current memory usage: 71.8 MB.
```

Figure 3. Synthesizing (Generating) the design

```
@I [HLS-10] -- Generating RTL for module 'yuv_filter_rgb2yuv'
@I [HLS-10] ------------------------------------------------------------------
@I [RTGEN-100] Generating core module 'yuv_filter_mul_8ns_7s_15_3': 1 instance(s).
@I [RTGEN-100] Generating core module 'yuv_filter_mul_8ns_8s_16_3': 1 instance(s).
@I [RTGEN-100] Finished creating RTL model for 'yuv_filter_rgb2yuv'.
@I [HLS-111] Elapsed time: 0.08 seconds; current memory usage: 72 MB.
@I [HLS-10] ------------------------------------------------------------------
@I [HLS-10] -- Generating RTL for module 'yuv_filter_yuv2rgb'
@I [HLS-10] ------------------------------------------------------------------
@I [RTGEN-100] Generating core module 'yuv_filter_mul_8s_8s_16_3': 1 instance(s).
@I [RTGEN-100] Finished creating RTL model for 'yuv_filter_yuv2rgb'.
@I [HLS-111] Elapsed time: 0.25 seconds; current memory usage: 72.5 MB.
@I [HLS-10] ------------------------------------------------------------------
@I [HLS-10] -- Generating RTL for module 'yuv_filter'
@I [HLS-10] ------------------------------------------------------------------
@I [RTGEN-500] Setting interface mode on port 'yuv_filter/in_channels_ch1' to 'ap_memory'.
@I [RTGEN-500] Setting interface mode on port 'yuv_filter/in_channels_ch2' to 'ap_memory'.
@I [RTGEN-500] Setting interface mode on port 'yuv_filter/in_channels_ch3' to 'ap_memory'.
@I [RTGEN-500] Setting interface mode on port 'yuv_filter/in_width' to 'ap_none'.
@I [RTGEN-500] Setting interface mode on port 'yuv_filter/in_height' to 'ap_none'.
@I [RTGEN-500] Setting interface mode on port 'yuv_filter/out_channels_ch1' to 'ap_memory'.
@I [RTGEN-500] Setting interface mode on port 'yuv_filter/out_channels_ch2' to 'ap_memory'.
@I [RTGEN-500] Setting interface mode on port 'yuv_filter/out_channels_ch3' to 'ap_memory'.
@I [RTGEN-500] Setting interface mode on port 'yuv_filter/out_width' to 'ap_vld'.
@I [RTGEN-500] Setting interface mode on port 'yuv_filter/out_height' to 'ap_vld'.
@I [RTGEN-500] Setting interface mode on port 'yuv_filter/Y_scale' to 'ap_none'.
@I [RTGEN-500] Setting interface mode on port 'yuv_filter/U_scale' to 'ap_none'.
@I [RTGEN-500] Setting interface mode on port 'yuv_filter/V_scale' to 'ap_none'.
@I [RTGEN-500] Setting interface mode on function 'yuv_filter' to 'ap_ctrl_hs'.
@I [RTGEN-100] Finished creating RTL model for 'yuv_filter'.
@I [HLS-111] Elapsed time: 0.24 seconds; current memory usage: 73.2 MB.
@I [RTMG-282] Generating pipelined core: 'yuv_filter_mul_8ns_7s_15_3_MAC3S_0'
@I [RTMG-282] Generating pipelined core: 'yuv_filter_mul_8ns_8s_16_3_MAC3S_1'
@I [RTMG-282] Generating pipelined core: 'yuv_filter_mul_8s_8s_16_3_MAC3S_2'
@I [RTMG-278] Implementing memory 'yuv_filter_p_yuv_channels_ch1_ram' using block RAMs.
@I [HLS-10] Finished generating all RTL models.
@I [WSYSC-301] Generating RTL SystemC for 'yuv_filter'.
@I [WVHDL-304] Generating RTL VHDL for 'yuv_filter'.
@I [WVLOG-307] Generating RTL Verilog for 'yuv_filter'.
```

Figure 4. Generating RTL

1-1-6. 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 don't have definite answer (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.

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

- o 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.

- o The scale function simply applies individual scale factors, supplied as top-level arguments to the Y'UV components.

- o Notice that most of the variables are of user-defined (`typedef`) and aggregate (e.g. structure, array) types.

- o 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.

- o 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.

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 lines (50, 53, 90, 93, 130, 133) to bring the #pragma statements into the design to define the variable bounds.

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

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
default	10.00	8.32	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
561205	34417925	561206	34417926	none

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 [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)
@I [XFORM-11] Balancing expressions in function 'rgb2yuv' (yuv_filter.c:30)...11 expression(s) balanced
@I [HLS-111] Elapsed time: 3.35 seconds; current memory usage: 68.7 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 `Area` 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 4X the specified TRIPCOUNT, implying that 4 cycles are used for each of the iteration of the loop.

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
561205	34417925	561206	34417926	none

Detail

Instance

		Latency		Interval		
Instance	Module	min	max	min	max	Type
grp_yuv_filter_rgb2yuv_fu_246	yuv_filter_rgb2yuv	200401	12291841	200401	12291841	none
grp_yuv_filter_yuv2rgb_fu_266	yuv_filter_yuv2rgb	200401	12291841	200401	12291841	none

Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- YUV_SCALE_LOOP_X	160400	9834240	802 ~ 5122	-	-	200 ~ 1920	no
+ YUV_SCALE_LOOP_Y	800	5120	4	-	-	200 ~ 1280	no

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.

XILINX

```
Current Module : yuv filter
```

	Operation\Control S...	C0	C1	C2	C3	C4	C5	C6
1	in width read(r...							
2	in height read(...							
3	yuv filter rgb2...							
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	☐YUV SCALE LOOP Y							
12	y i(phi mux)							
13	exitcond i(icmp)							
14	y(+)							
15	p addr(+)							
16	p addr1(+)							
17	Y(read)							
18	U(read)							
19	V(read)							
20	tmp 32 i(*)							
21	tmp 34 i(*)							
22	tmp 36 i(*)							
23	node 87(write)							
24	node 90(write)							
25	node 93(write)							
26	yuv filter yuv2...							
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_246** (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 the worst case latency is reported as 12291841 clock cycles. The reported latency can be estimated as follows.

- o   RGB2YUV_LOOP_Y total loop latency = 5 x 1280 = 6400 cycles

- o   1 entry and 1 exit clock for loop RGB2YUV_LOOP_Y = 6402 cycles

- o   RGB2YUV_LOOP_X loop body latency = 6402 cycles

- o   RGB2YUV_LOOP_X total loop latency = 6402 x 1920 =12291840 cycles

- o   1 entry clock for RGB2YUV_LOOP_X = 12291841 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.

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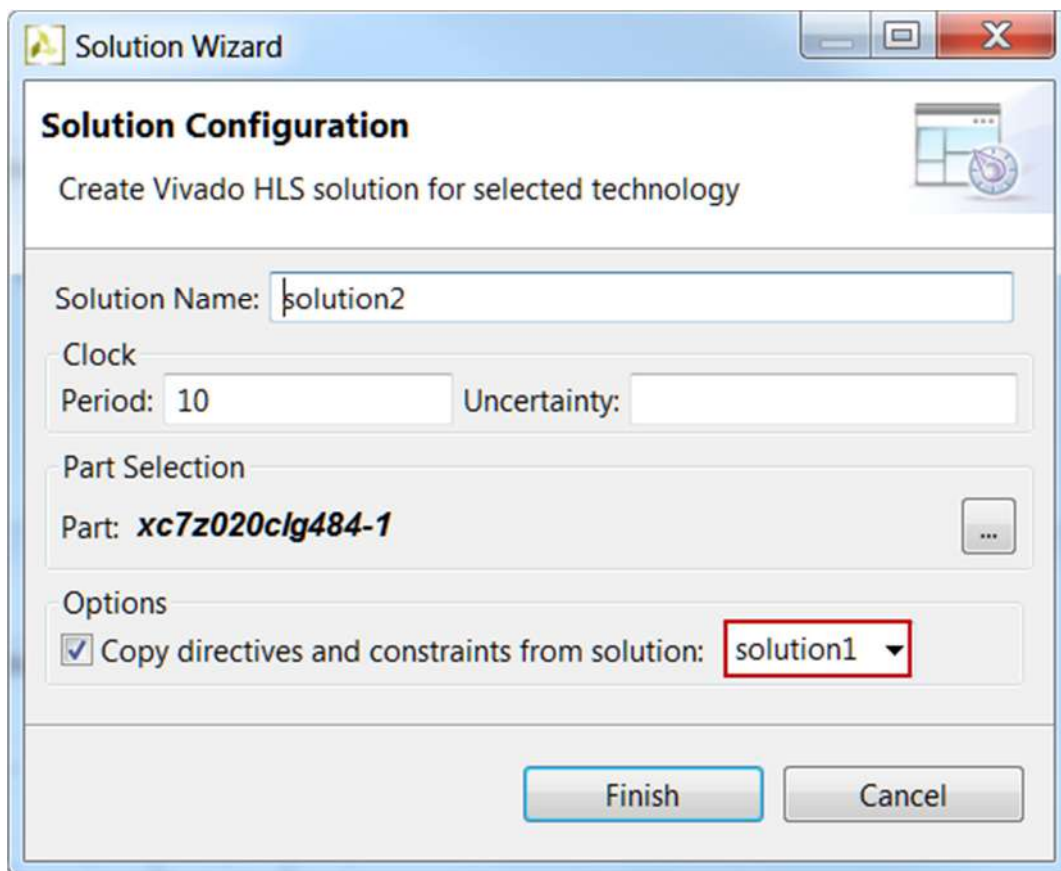
Figure 10. Creating a new Solution after copying the existing solution

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**.
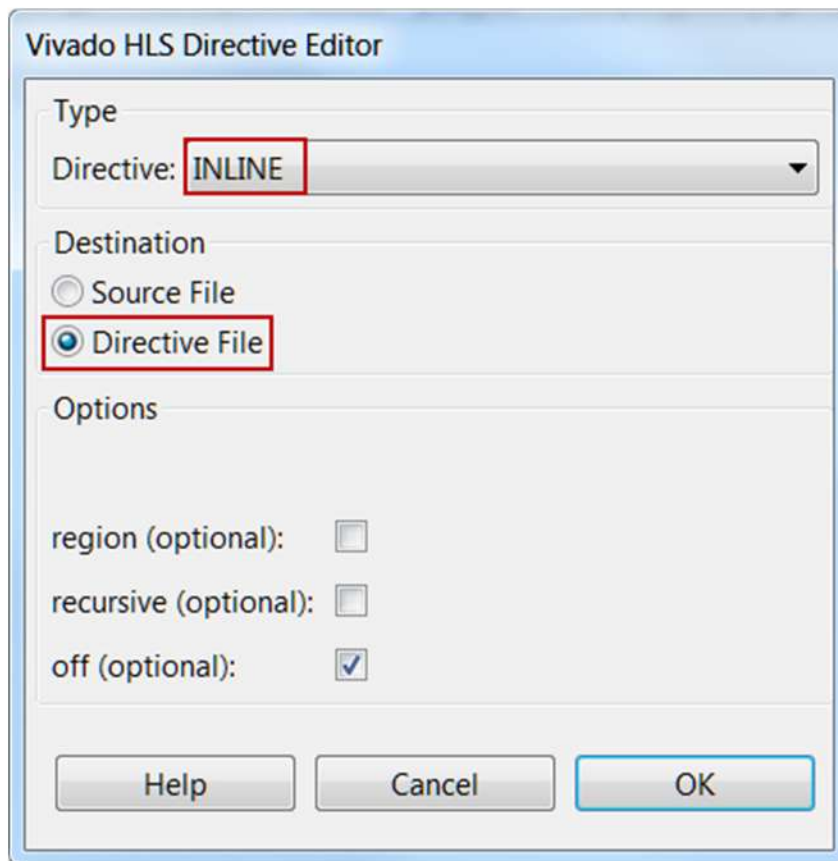
Figure 11. Turning OFF the inlining function

- 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.

- ○ yuv_filter
  - ◉ in
  - ◉ out
  - ◉ Y_scale
  - ◉ U_scale
  - ◉ V_scale
- ○ rgb2yuv
  - ×[] Wrgb
  - RGB2YUV_LOOP_X
    - # HLS loop_tripcount min=200 max=1920
    - RGB2YUV_LOOP_Y
      - % HLS PIPELINE
      - # HLS loop_tripcount min=200 max=1280
- ○ yuv2rgb
  - ×[] 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
  - 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 34417926 to 7372823 clock cycles.

Vivado HLS Report Comparison

All Compared Solutions

solution2: xc7z020clg484-1

solution1: xc7z020clg484-1

Performance Estimates

⊟ **Timing (ns)**

Clock		solution2	solution1
default	Target	10.00	10.00
	Estimated	8.32	8.32

⊟ **Latency (clock cycles)**

		solution2	solution1
Latency	min	120022	561205
	max	7372822	34417925
Interval	min	120023	561206
	max	7372823	34417926

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	7200	7200
DSP48E	15	12
FF	800	566
LUT	1208	833

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 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.

Figure 15. Performance estimate after DATAFLOW directive applied

- o 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.

- o 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 Area 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
Expression	-	-	0	2
FIFO	0	-	20	112
Instance	-	15	837	1143
Memory	14400	-	0	0
Multiplexer	-	-	-	-
Register	-	-	16	-
Total	14400	15	873	1257
Available	280	220	106400	53200
Utilization (%)	5142	6	~0	2

Figure 16. Resource estimate with DATAFLOW directive applied

o 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.

o 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**.
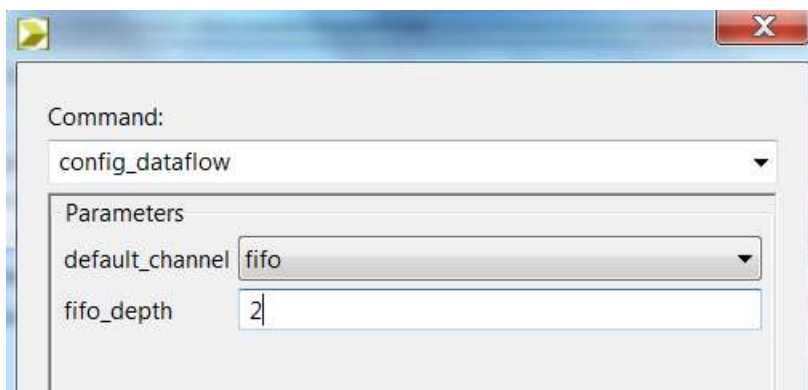

Figure 17. Selecting Dataflow configuration command and FIFO as buffer

⚡ XILINX ®

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
Expression	-	-	-	-
FIFO	0	-	50	232
Instance	-	15	623	869
Memory	-	-	-	-
Multiplexer	-	-	-	-
Register	-	-	7	-
Total	0	15	680	1101
Available	280	220	106400	53200
Utilization (%)	0	6	~0	2

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 **Option** field and select **VHDL** as the language and tick **Evaluate**.

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.

Figure 19. Implementation results in Vivado HLS

Note that the implementation was successful, meeting the expected timings.

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: 8.32 ns

    Worst case latency: 34417925 clock cycles

    Number of DSP48E used: 12

    Number of BRAMs used: 7200

    Number of FFs used: 566

    Number of LUTs used: 833

2. Answer the following questions rgb2yuv:

    Estimated clock period: 8.32 ns

    Worst case latency: 12291841 clock cycles

    Number of DSP48E used: 5

    Number of FFs used: 180

    Number of LUTs used: 314

3. Answer the following questions for yuv2rgb:

    Estimated clock period: 7.75 ns

    Worst case latency: 12291841 clock cycles

    Number of DSP48E used: 4

    Number of FFs used: 186

    Number of LUTs used: 250