

# 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:

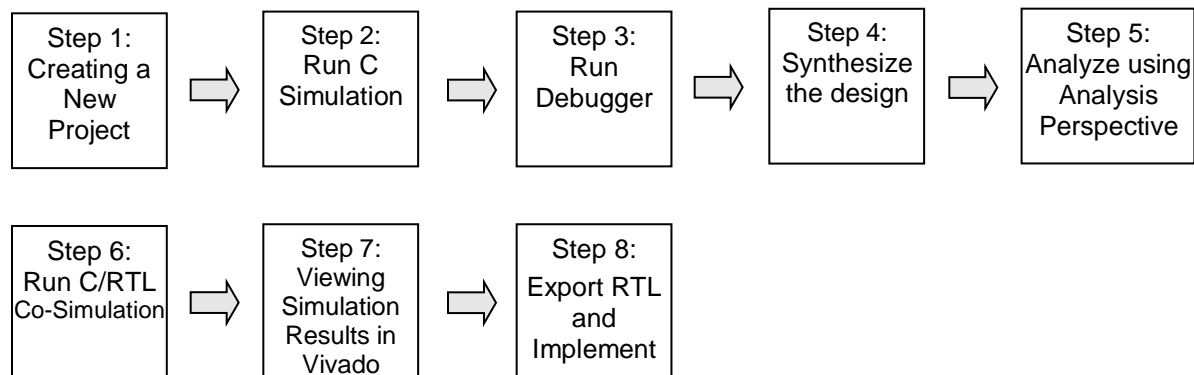
- Create a new project using Vivado HLS GUI
- Simulate a design
- Synthesize a design
- Implement a design
- Perform design analysis using the Analysis capability of Vivado HLS
- Analyze simulator output using Vivado and XSim simulator

## 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



## 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 2013.3 > Vivado HLS > Vivado HLS 2013.3**

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**

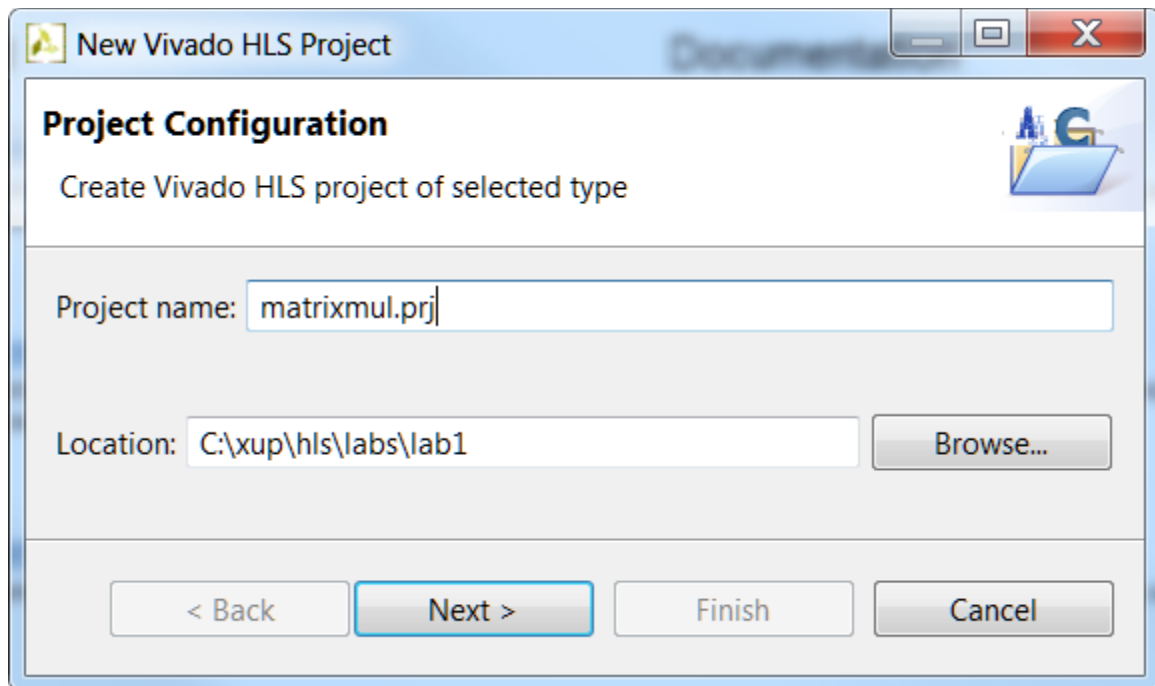
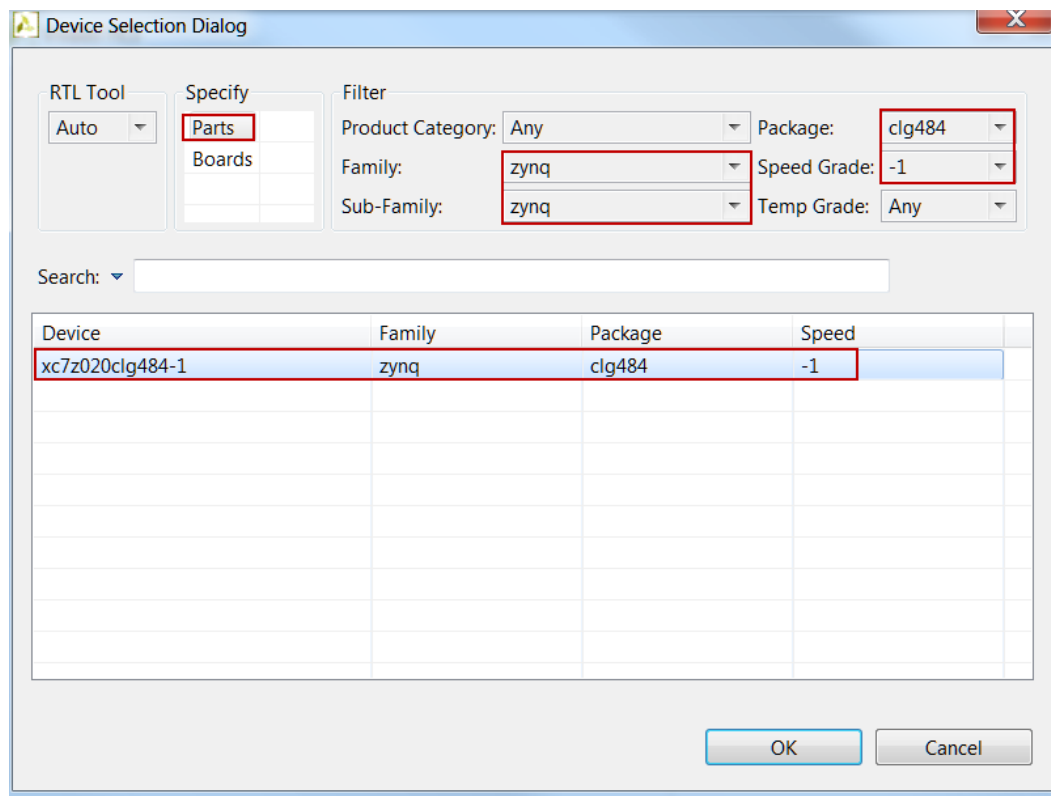


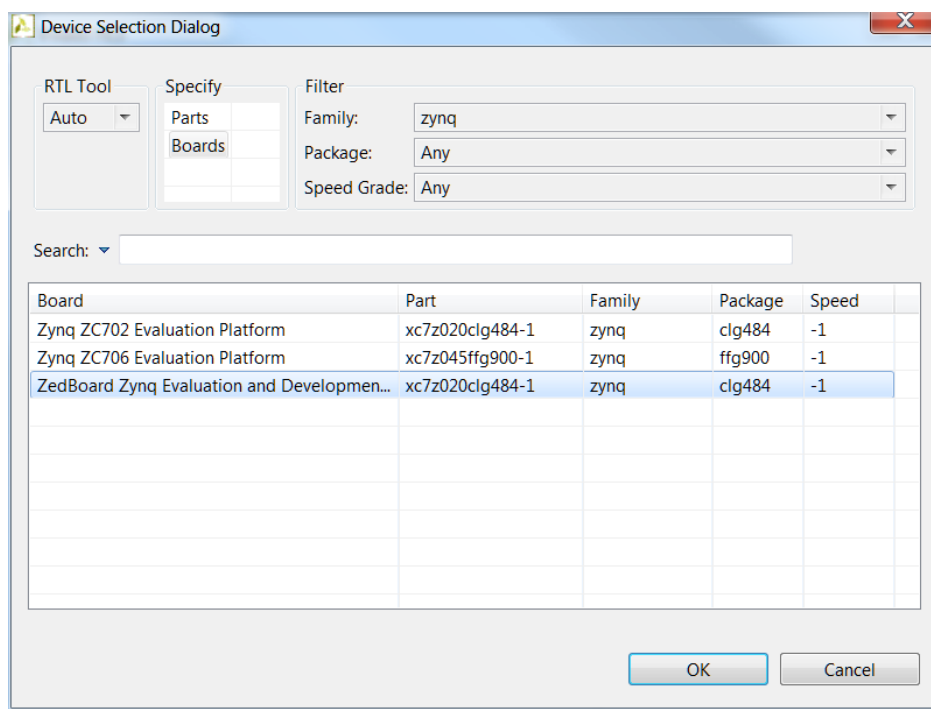
Figure 2. New Vivado HLS Project wizard

- 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.

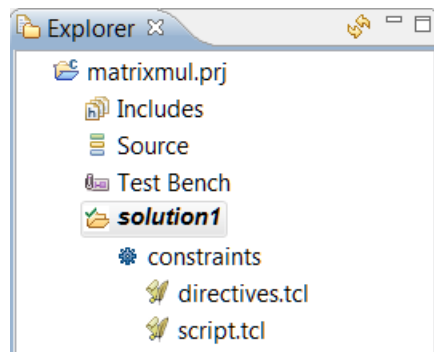


Figure 5. Explorer Window

**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
69void 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.

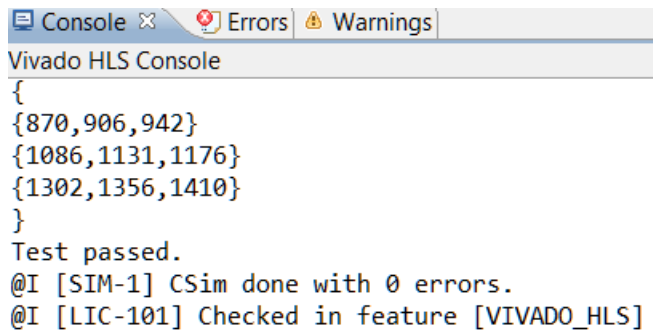
## Run C Simulation

## 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 x Errors Warnings
Vivado HLS Console
{
{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 [VIVADO_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 *Debug* 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.

**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.

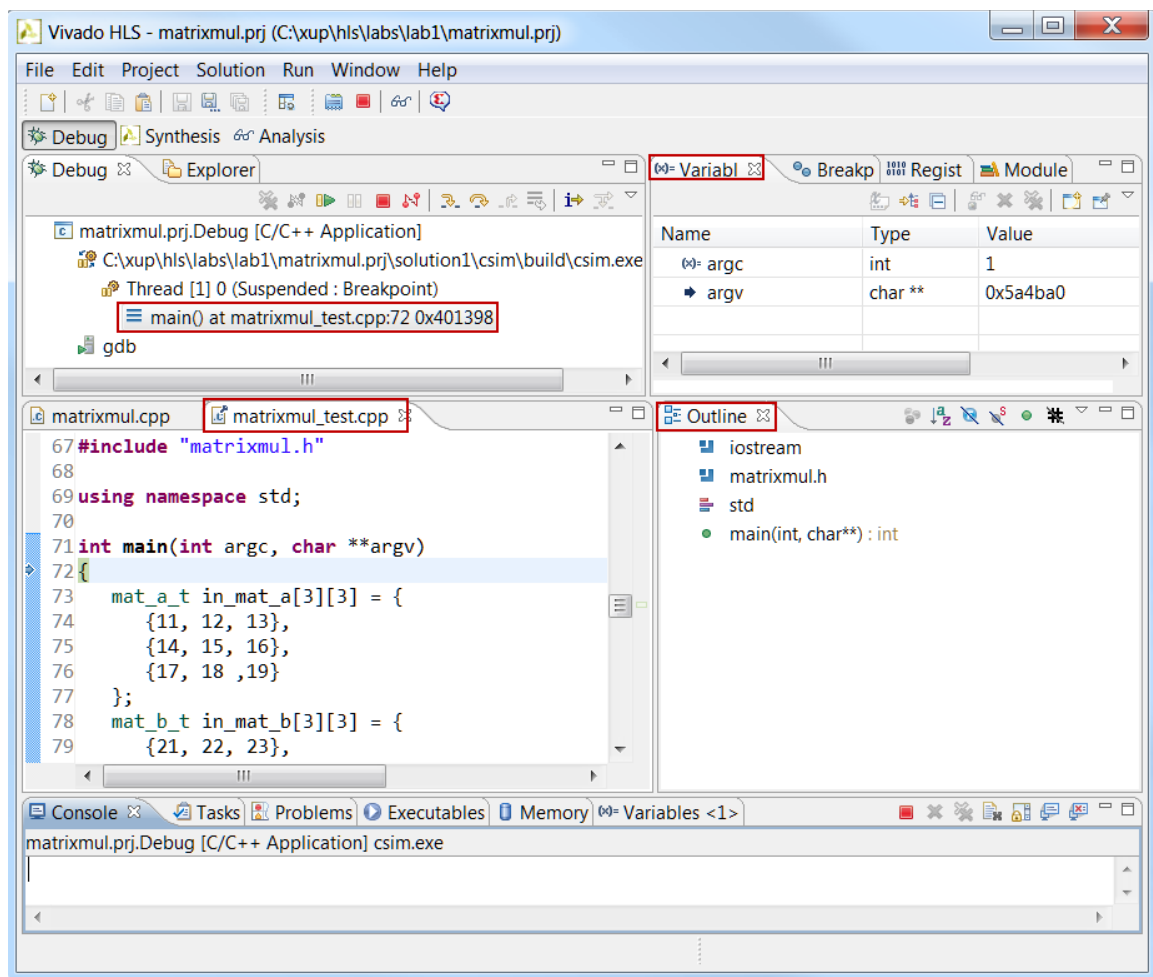



Figure 8. A Debug perspective

- 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

```
104 // Print result matrix
105 cout << "{" << endl;
```

- 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.

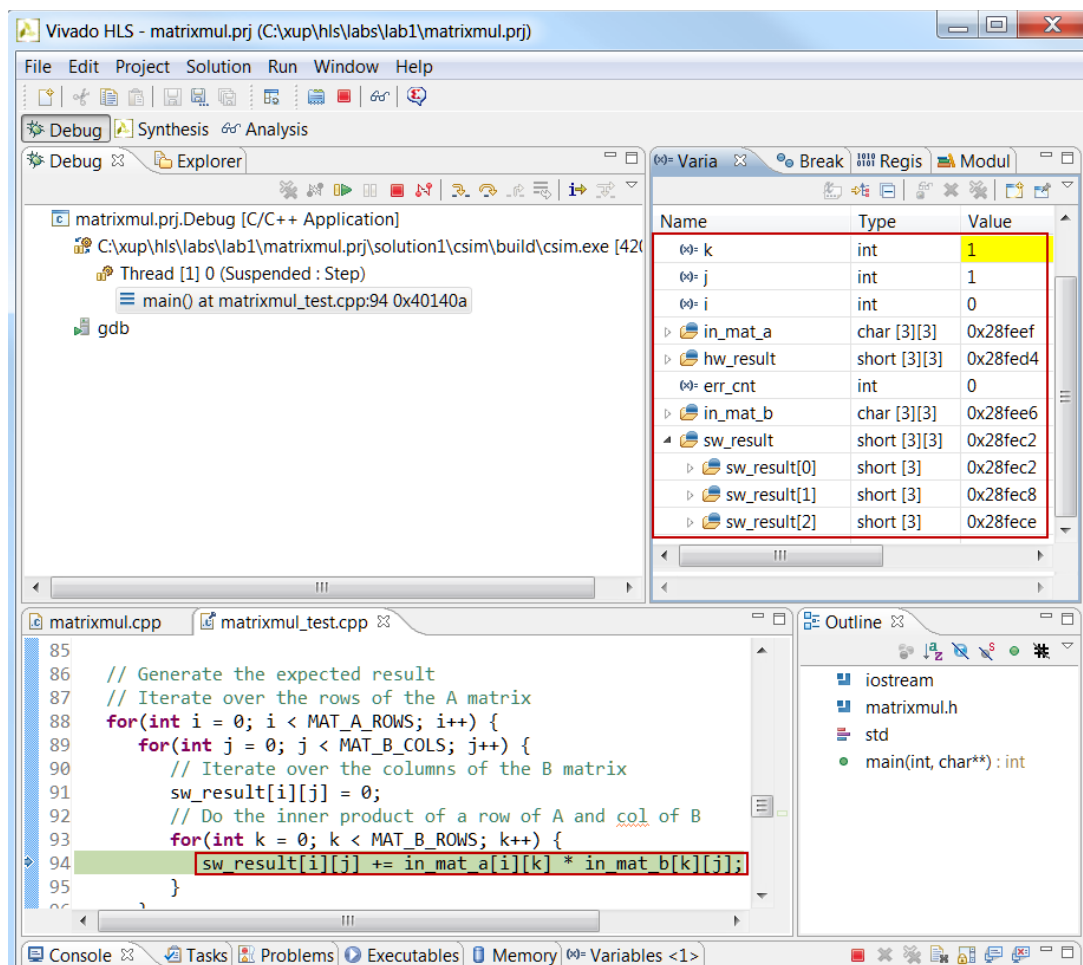


Figure 9. Debugger's intermediate output view


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.

Name	Type	Value
sw_result	short [3][3]	0x28fec2
sw_result[0]	short [3]	0x28fec2
sw_result[0][0]	short	870
sw_result[0][1]	short	906
sw_result[0][2]	short	942
sw_result[1]	short [3]	0x28fec8
sw_result[1][0]	short	1086
sw_result[1][1]	short	1131
sw_result[1][2]	short	1176
sw_result[2]	short [3]	0x28fece
sw_result[2][0]	short	1302
sw_result[2][1]	short	1356
sw_result[2][2]	short	1410

Figure 10. Software computed result



- 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.

hw_result	short [3][3]	0x28fed4
hw_result[0]	short [3]	0x28fed4
hw_result[0][0]	short	870
hw_result[0][1]	short	906
hw_result[0][2]	short	942
hw_result[1]	short [3]	0x28feda
hw_result[2]	short [3]	0x28fee0
err_cnt	int	0
in_mat_b	char [3][3]	0x28fee6
sw_result	short [3][3]	0x28fec2
sw_result[0]	short [3]	0x28fec2
sw_result[0][0]	short	870
sw_result[0][1]	short	906
sw_result[0][2]	short	942
sw_result[1]	short [3]	0x28fec8
sw_result[2]	short [3]	0x28fece

**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  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.

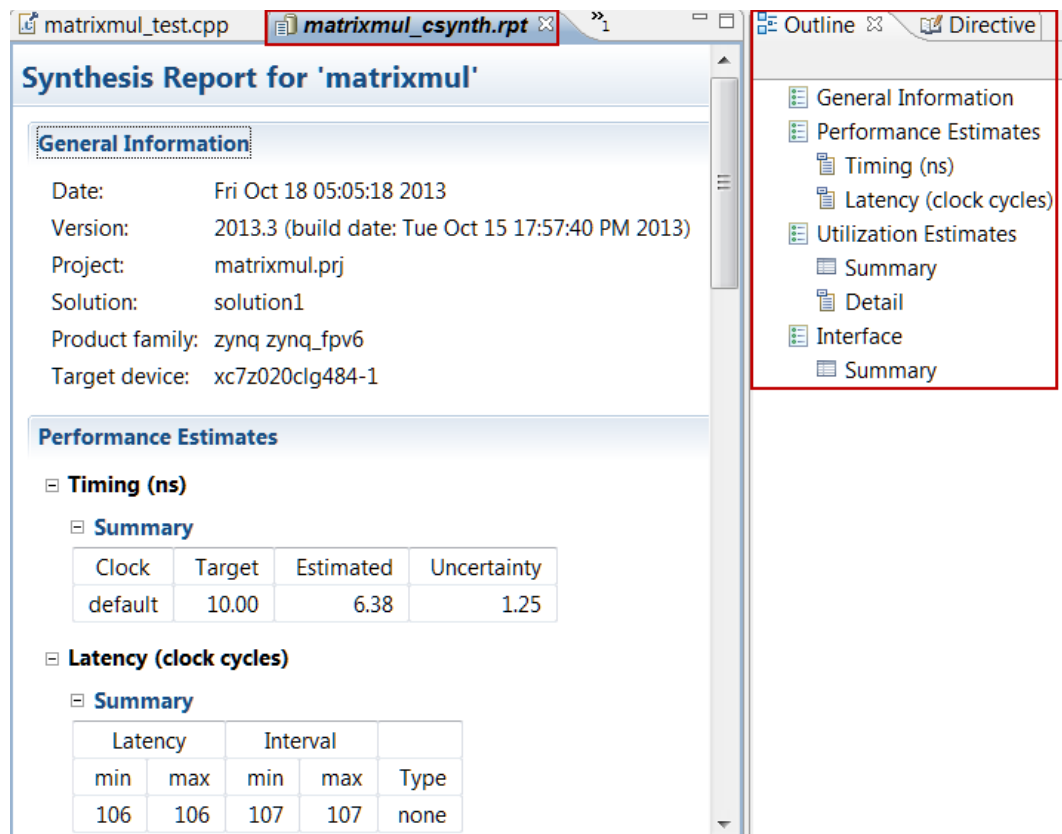


Figure 12. Report view after synthesis is completed

- 4-1-4. If you expand **solution1** in Explorer, several generated files including report files will become accessible.

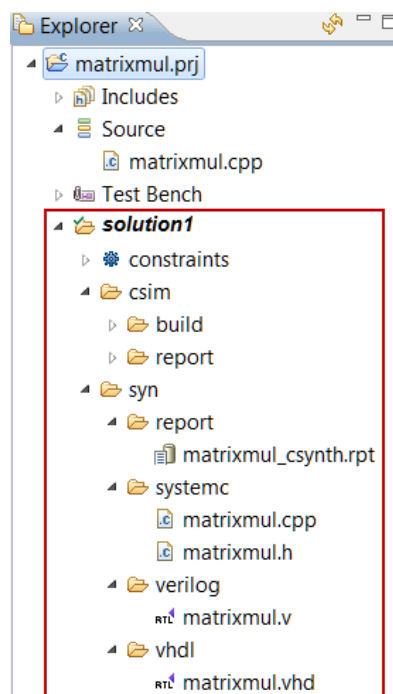


Figure 13. Explorer view after the synthesis process

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	matrixmul1	return value
ap_rst	in	1	ap_ctrl_hs	matrixmul1	return value
ap_start	in	1	ap_ctrl_hs	matrixmul1	return value
ap_done	out	1	ap_ctrl_hs	matrixmul1	return value
ap_idle	out	1	ap_ctrl_hs	matrixmul1	return value
ap_ready	out	1	ap_ctrl_hs	matrixmul1	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 (    ) 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.

- The left-hand column lists the resources
- 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

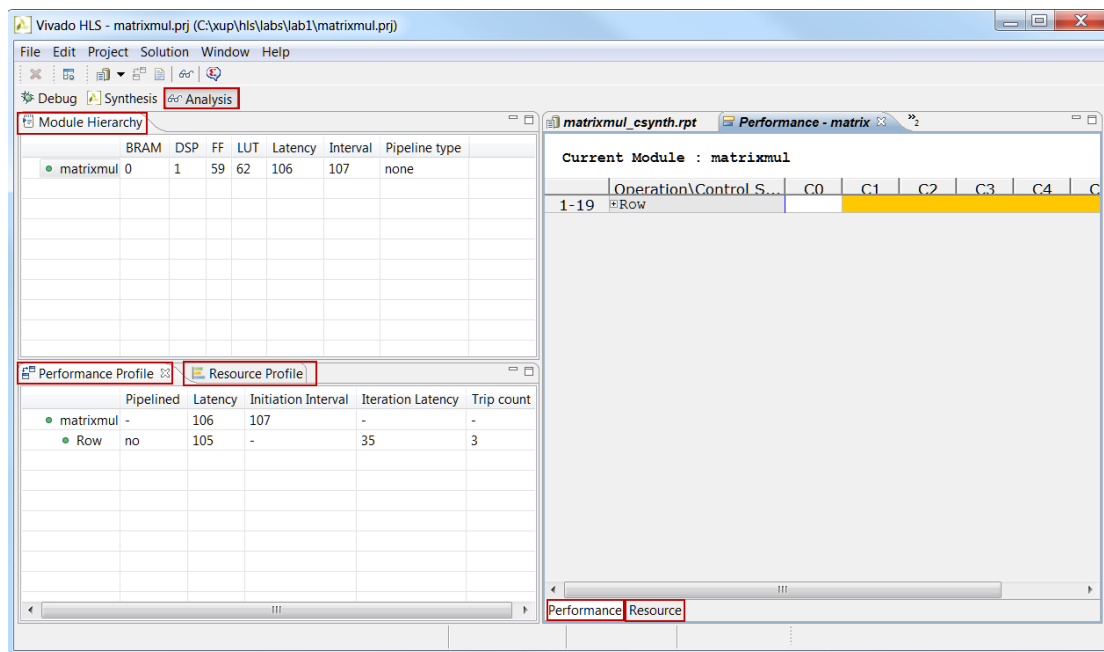


Figure 15. Analysis perspective

- 5-1-2. Click on loop **Row** to expand, and then click on sub-loops **Col** and **Product** to fully expand the loop hierarchy.

matrixmul\_test.cpp    matrixmul\_csynth.rpt    Performance - matrix    »1

Current Module : matrixmul

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

**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.

Current Module : matrixmul

	Operation\Control S...	C0	C1	C2	C3	C4	C5
1	Row						
2	exitcond2 (icmp)						
3	i 1(+)						
4	Col						
5	exitcond1 (icmp)						
6	j 1(+)						
7	p_addr7 (-)						
8	p_addr8 (+)						
9	Product						
10	node 42 (write)						
11	exitcond (icmp)						
12	k 1(+)						
13	p_addr1 (+)						
14	a load (read)						
15	p_addr3 (-)						
16	p_addr4 (+)						
17	b load (read)						
18	tmp 7 (*)						
19	tmp 8 (+)						

File: C:\xup\hls\labs\lab1\matrixmul.cpp

```

63 //Reference:
64 //Revision History:
65 //*****
66
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 a column 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 17. Cross probing into the source file

5-1-4. In C2, click on the operations (e.g. p\_addr4) 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.

	Pipelined	Latency	Initiation Interval	Iteration Latency	Trip count
matrixmul1	-	106	107	-	-
Row	no	105	-	35	3
Col	no	33	-	11	3
Product	no	9	-	3	3

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).

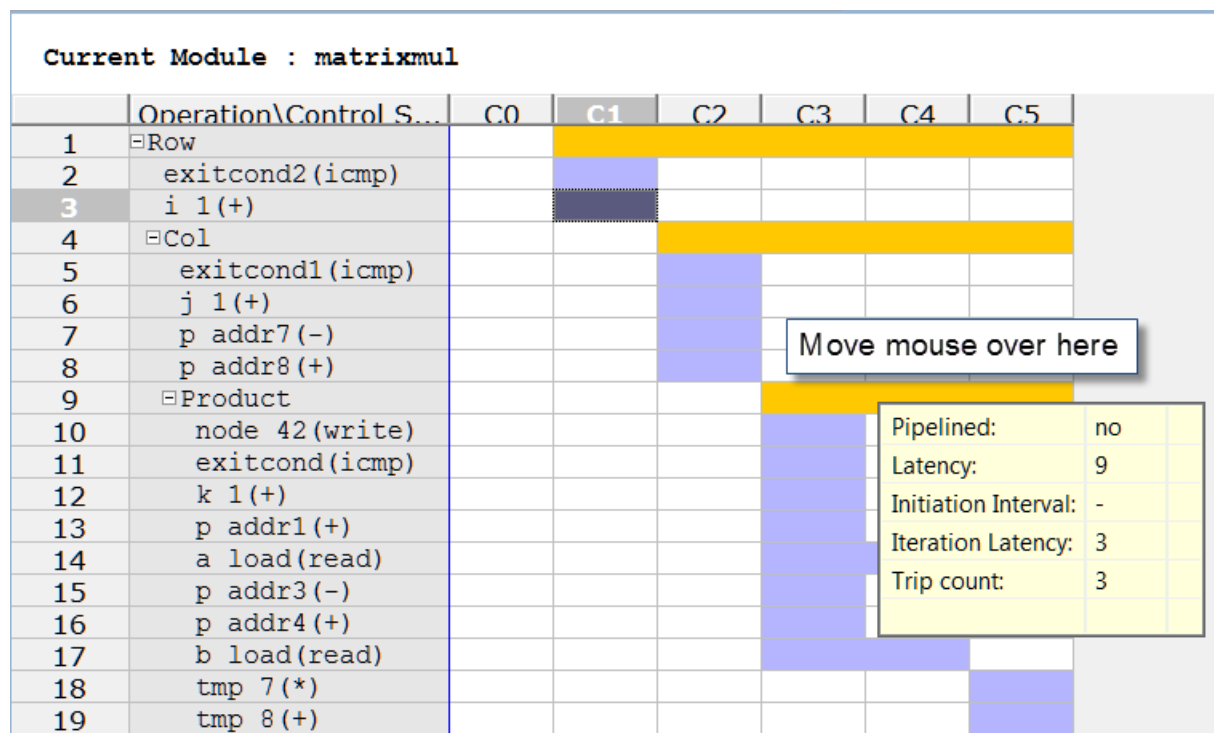


Figure 19. Loop information

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.

Performance Profile		Resource Profile						
		BRAM	DSP	FF	LUT	Bits P0	Bits P1	Bits P2
matrixmul		0	1	59	62			
I/O Ports(3)					32			
Instances(0)		0	0	0	0			
Memories(0)		0		0	0			0
Expressions(12)		0	1	0	40	48	45	0
Registers(13)				59	63			
FIFO(0)		0		0	0			0
Multiplexers(4)		0		0	22	22		0

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.

**Current Module : matrixmul**

	Resource\Control Step	C0	C1	C2	C3	C4	C5
1	[-] I/O Ports						
2	a						
3	res						
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	exitcond1 fu 135			icmp			
13	p addr7 fu 167			-			
14	p addr8 fu 177			+			
15	j 1 fu 141			+			
16	exitcond fu 192				icmp		
17	p addr3 fu 238				-		
18	k 1 fu 198				+		
19	p addr1 fu 212				+		
20	p addr4 fu 248				+		
21	tmp 7 fu 268						*
22	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

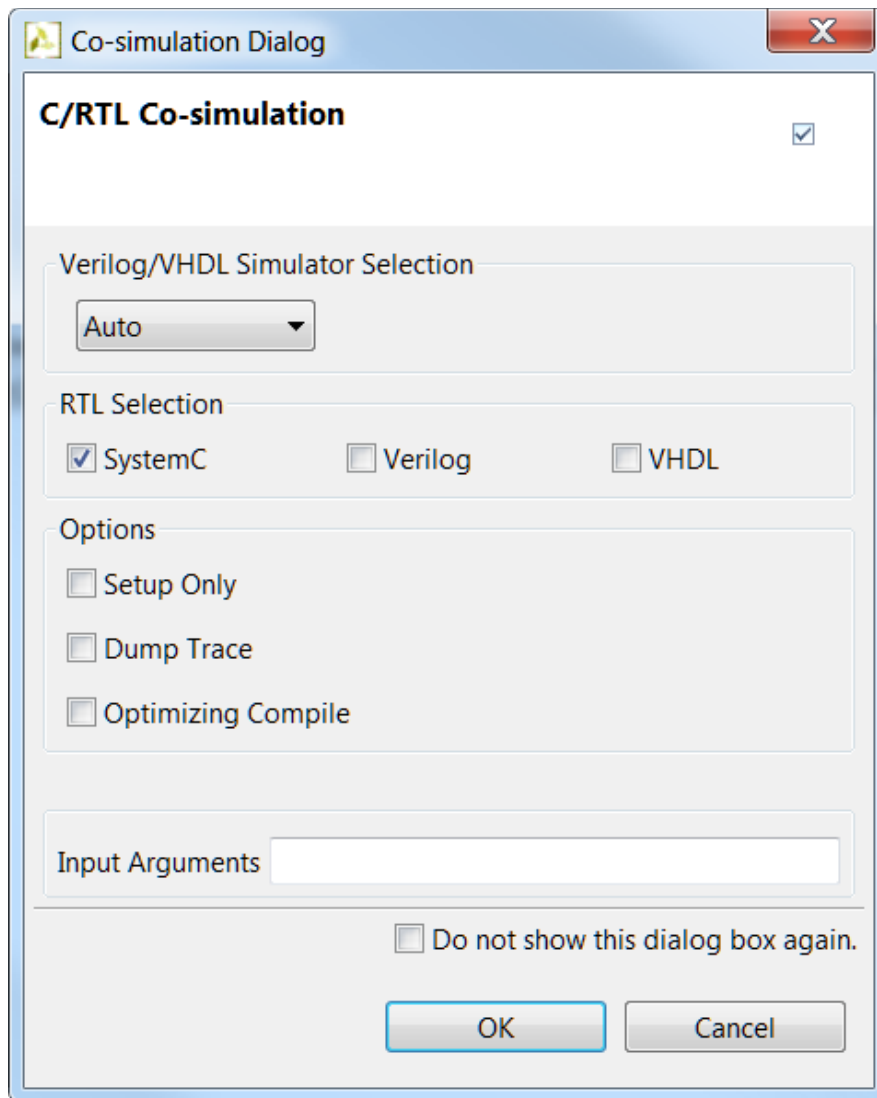
## Step 6

**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.





**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

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/2013.3/bin/vivado_hls.bat C:/xup/hls/labs/lab1/matrixmu
@I [LIC-101] Checked out feature [VIVADO_HLS]
@I [HLS-10] Running 'C:/Xilinx/Vivado_HLS/2013.3/bin/unwrapped/win64.o/vivac
      for user 'parimalp' on host 'xsjparimalp30' (Windows NT_amd64 ve
      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/2013.3/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 ...

@I [SIM-333] Generating C post check test bench ...
@I [SIM-12] Generating RTL test bench ...
      Build using "C:/Xilinx/Vivado_HLS/2013.3/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.

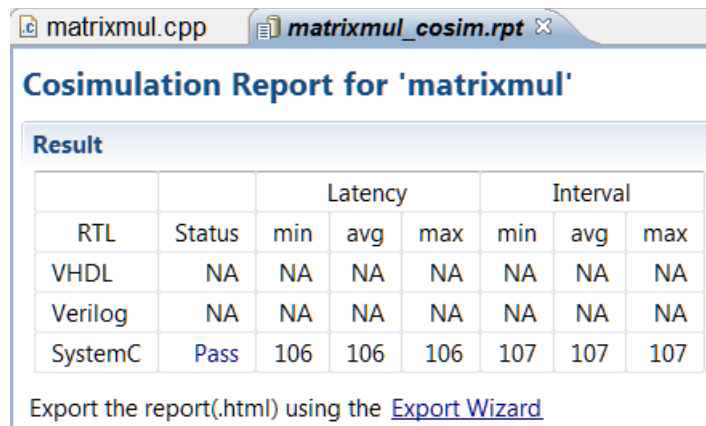
Info: /OSCI/SystemC: Simulation stopped by user.
@I [SIM-316] Starting C post checking ...
{
{870,906,942}
{1086,1131,1176}
{1302,1356,1410}
}
Test passed.
@I [SIM-1000] *** C/RTL co-simulation finished: PASS ***
@I [LIC-101] Checked in feature [VIVADO_HLS]

```

**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.



**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.** Uncheck the *SystemC* option and 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 the *Dump Trace* option and click **OK**.

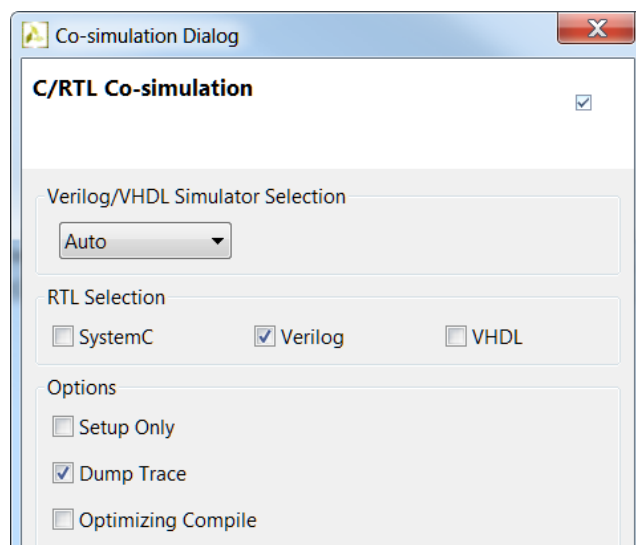
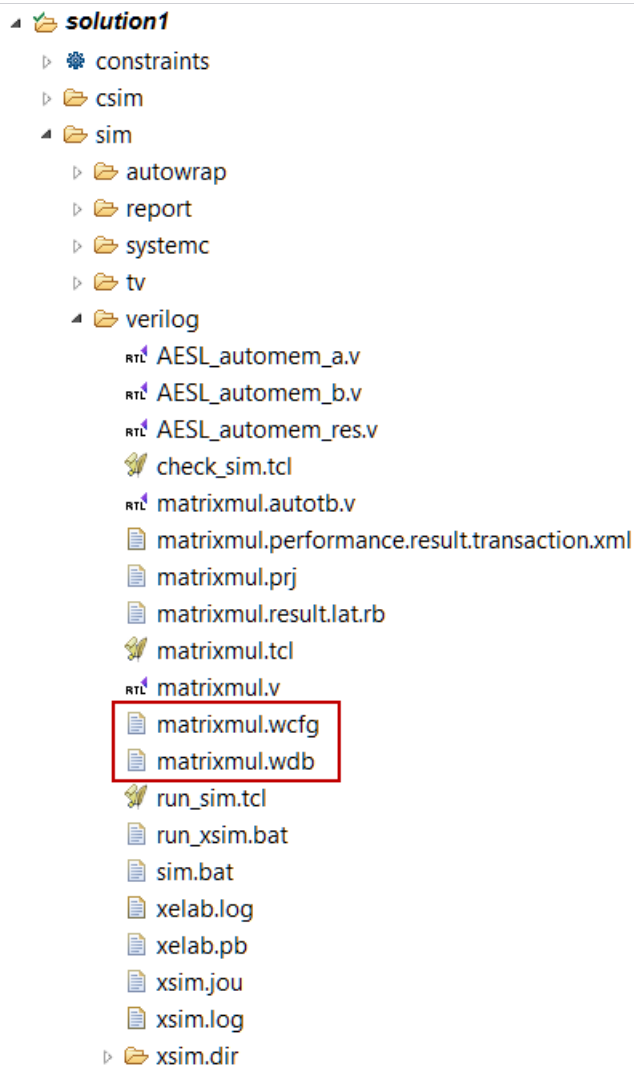


Figure 25. Setting up for Verilog simulation and dump trace

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.

matrixmul.cpp    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	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 2013.3 and enter Tcl commands to open and view the dumped traces.

**7-2-1.** Select **Start > All Programs > Xilinx Design Tools > Vivado 2013.3 > Vivado 2013.3** 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_filesset
open_wave_database matrixmul.wdb
open_wave_config matrixmul.wcfg
```

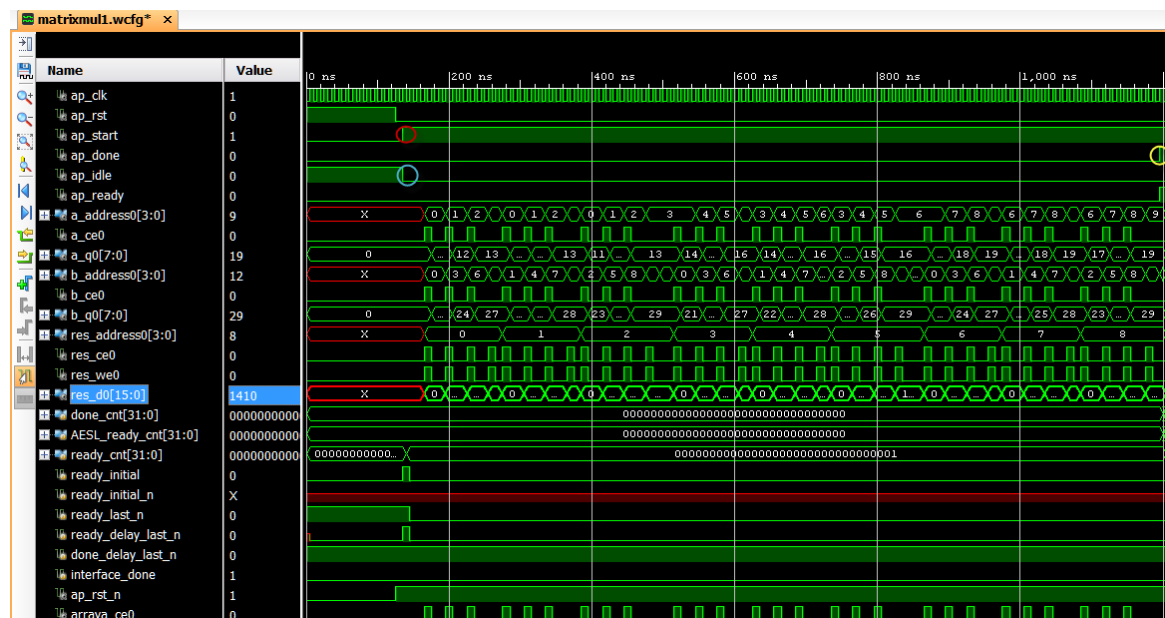
The above commands will load the project, simulation results, and open the waveform.

**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**.

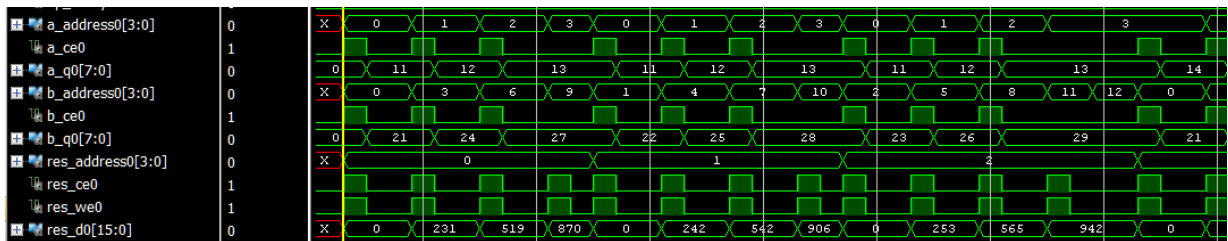
**7-2-6.** Scroll the waveform little, so you can view the main interface signals (*ap\_\**).



**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.



**Figure 29. Zoomed view**

Observe that the design expects element data by providing a\_address0, a\_ce0, b\_address0, b\_ce0 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.

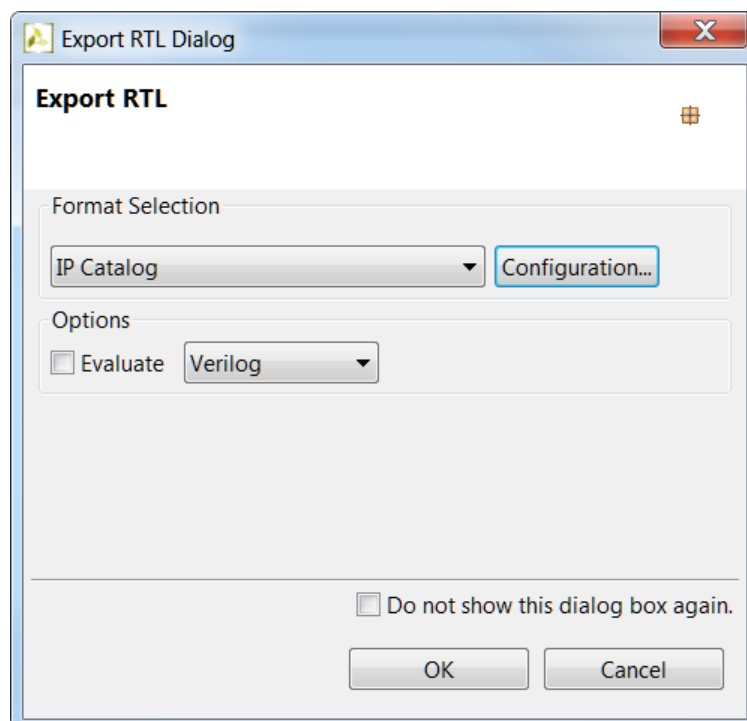
## Export RTL and Implement

## 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:

- Exporting RTL as an IP in the IP-XACT format
- RTL evaluation, since we selected Evaluate option
  - Goes through Synthesis
  - Goes through Placement and Routing

Starting export RTL ...

```
C:/Xilinx/Vivado_HLS/2013.3/bin/vivado_hls.bat C:/xup/hls/labs/lab1/matrixmul.prj/
@I [LIC-101] Checked out feature [VIVADO_HLS]
@I [HLS-10] Running 'C:/Xilinx/Vivado_HLS/2013.3/bin/unwrapped/win64.o/vivado_hls.
           for user 'parimalp' on host 'xsjparimalp30' (Windows NT_amd64 version
           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 -notrace
```

```
***** Vivado v2013.3 (64-bit)
**** SW Build 328945 on Tue Oct 15 18:15:46 MDT 2013
**** IP Build 192451 on Mon Oct 14 10:53:07 MDT 2013
** Copyright 1986-1999, 2001-2013 Xilinx, Inc. All Rights Reserved.
```

```
INFO: [Common 17-78] Attempting to get a license: Implementation
INFO: [Common 17-81] Feature available: Implementation
INFO: [Device 21-36] Loading parts and site information from c:/Xilinx/
Parsing RTL primitives file [c:/Xilinx/Vivado/2013.3/data/parts/xilinx/r
Finished parsing RTL primitives file [c:/Xilinx/Vivado/2013.3/data/parts
source run_vivado.tcl -notrace
[Fri Oct 18 05:37:09 2013] Launched synth_1...
```

```

Implementation tool: Xilinx Vivado v.2013.3
Device target:      xc7z020clg484-1
Report date:       Fri Oct 18 05:39:28 -0700 2013

#=== Resource usage ===
SLICE:             14
LUT:               50
FF:               28
DSP:               1
BRAM:              0
SRL:               0
#=== Final timing ===
CP required:       10.000
CP achieved:       5.553
Timing met
INFO: [Common 17-206] Exiting Vivado at Fri Oct 18 05:39:28 2013...
@I [LIC-101] Checked in feature [VIVADO_HLS]

```

**Figure 31. Console view**

When the run is completed the implementation report will be displayed in the information pane.

**Export Report for 'matrixmul'**

**General Information**

Report date: Fri Oct 18 05:39:27 -0700 2013  
Device target: xc7z020clg484-1  
Implementation tool: Xilinx Vivado v.2013.3

**Resource Usage**

	VHDL
SLICE	14
LUT	50
FF	28
DSP	1
BRAM	0
SRL	0

**Final Timing**

	VHDL
CP required	10.000
CP achieved	5.553

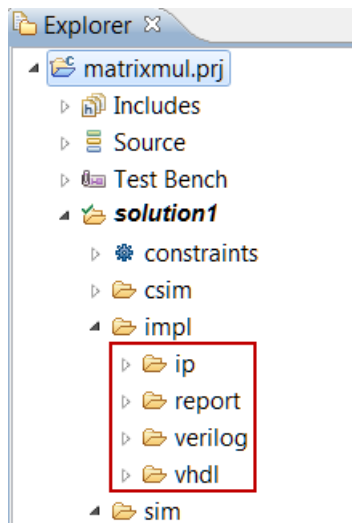
**Timing met**

**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.



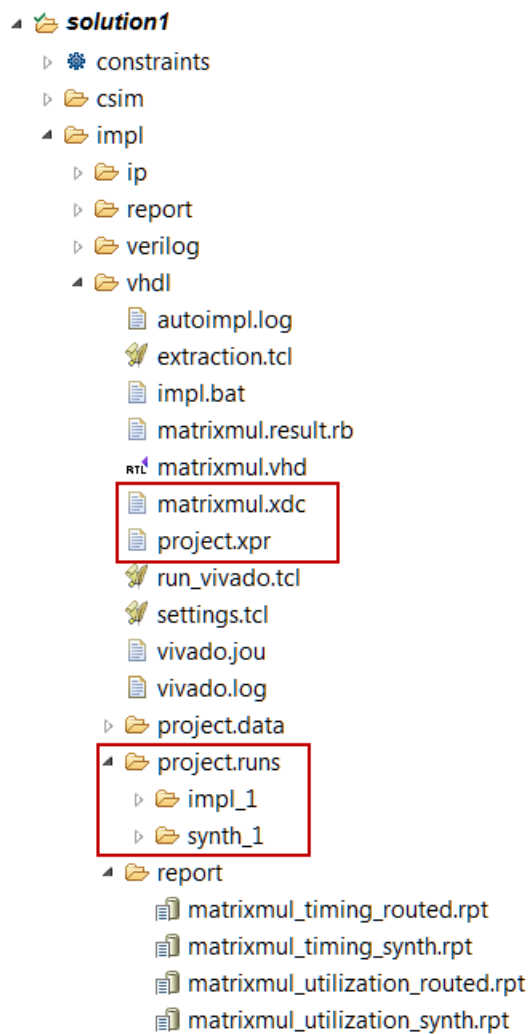
- 8-1-4.** Collapse the Explorer view and observe that impl folder is created under which ip, report, Verilog, and vhdl sub-folders are created.



**Figure 33. Explorer view after the RTL Export run**

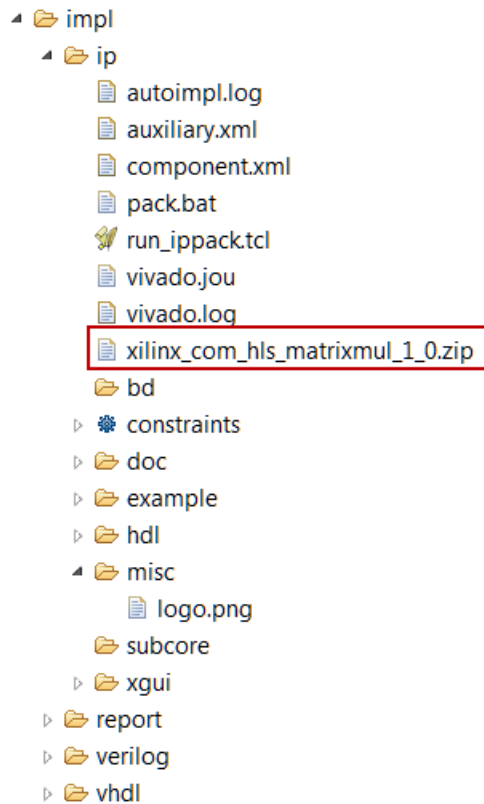
- 8-1-5.** Expand the Verilog and vhdl sub-folders and observe that the Verilog sub-folder only has the rtl file whereas vhdl sub-folder has several files and sub-folders as the synthesis and implementation runs were made for it.

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.



**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.



**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: \_\_\_\_\_

Worst case latency: \_\_\_\_\_

Number of DSP48E used: \_\_\_\_\_

Number of FFs used: \_\_\_\_\_

Number of LUTs used: \_\_\_\_\_