ACCELERATED CHIP DESIGN CYCLE WORKSHOP: HIGH-LEVEL SYNTHESIS

This workshop is intended to provide the participants with the necessary skills to create high-level synthesis IPs using the Vitis HLS tool flow. Various techniques and directives inherent to Vitis HLS to improve the design performance are also introduced in detail

The labs have been developed and tested on Windows 10 professional and Ubuntu 20.04 machines and using Vivado HLS 2018.2 and Vitis HLS 2022.1 version tools. Linux machine to run the Vitis HLS tool is more recommended though.

Labs Overview:

Lab1: Vitis HLS Design Flow

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

Lab2: Improving Performance

This lab introduces various techniques and directives which can be used in Vitis 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.

Lab3: Improving Area and Resource Utilization

This lab introduces various techniques and directives which can be used in Vitis HLS to improve design performance as well as area and resource utilization. The design under consideration performs discrete cosine transformation (DCT) on an 8x8 block of data.

Lab1: Vitis HLS Design Flow

Introduction:

This lab provides a basic introduction to high-level synthesis using the Vitis HLS tool flow. You will use Vitis 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:

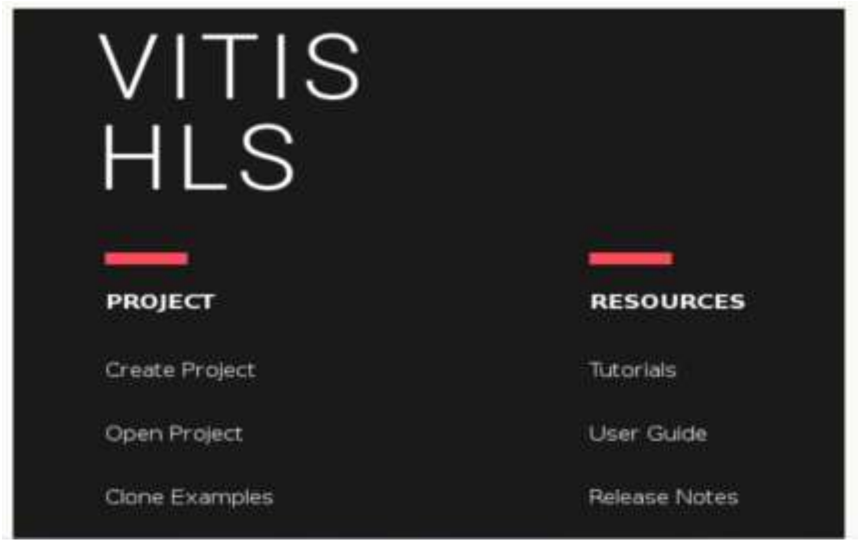
- Create a new project using Vitis HLS GUI
- Simulate a C design by using a self-checking test bench
- Synthesize the design
- Perform design analysis using the Analysis Perspective
- Perform co-simulation on a generated RTL design by using a provided C test bench
- Implement the design

Steps:

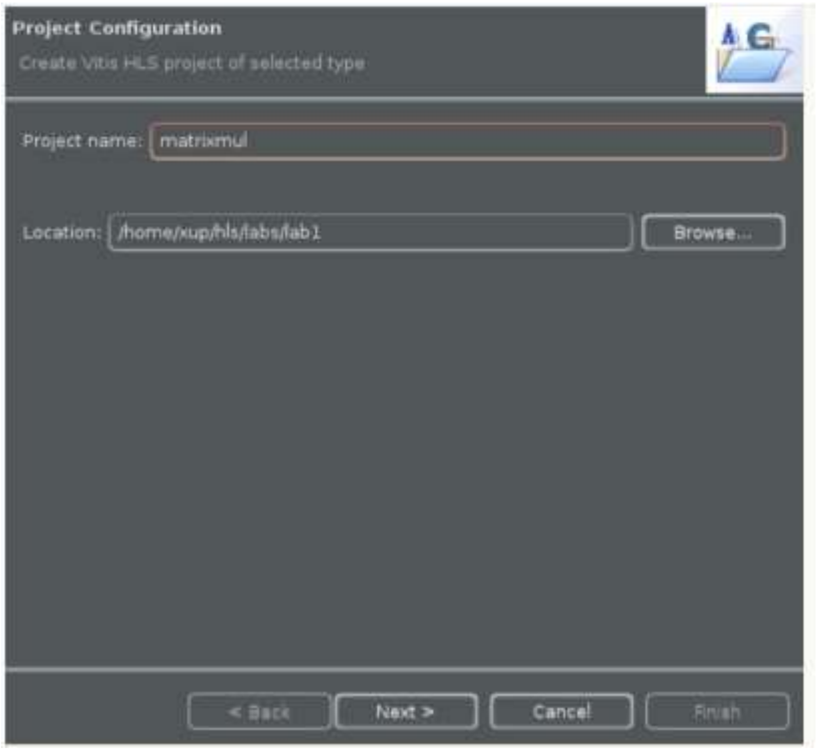
STEP 1: Create a New Project

Create a new project in Vitis HLS targeting Zynq xc7z020clg400-1.

1. Invoke Vitis HLS in GUI mode by typing **vitis_hls** in Linux or double clicking on **Vitis HLS 2022.1** icon in Windows environment.



2. In the Getting Started GUI, click on **Create Project**. The **New Vitis HLS Project** wizard opens.
3. Click the Browse… button of the Location and browse to **{labs}/lab1** creating sub-folders as necessary, and then click **OK**.
4. For project name, type **matrixmul**.

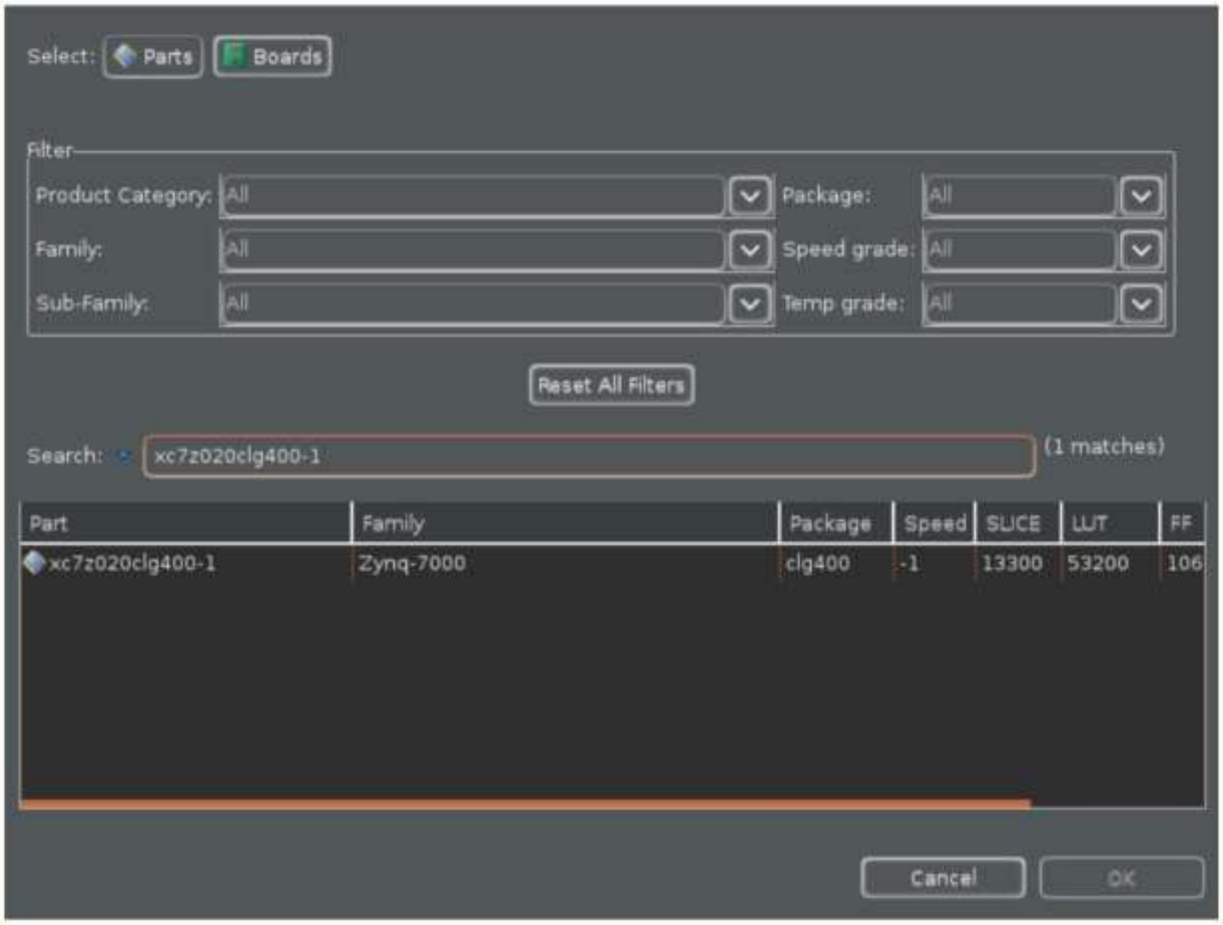

Project Configuration

Create Vitis HLS project of selected type

Project name: matrixmul

Location: /home/xup/hls/labs/lab1

5. Click **Next**
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).
7. Click the **Add Files…** button, select *matrixmul.cpp* file from the **{labs}\lab1** folder, and then click **Open**.
8. Click **Next**.
9. In the *Add/Remove Files* for the testbench, click the **Add Files…** button, select *matrixmul_test.cpp* file from the **{labs}\lab1** folder and click **Open.**
10. Select the **matrixmul_test.cpp** in the files list window and click the **Edit CSIMFLAG…** button, type **-DHW_COSIM**, and click **OK**. (This defines a custom flag that will be used later.)
11. Click **Next.**
12. In the *Solution Configuration* page, leave Solution Name field as **solution1** and set the clock period as **10**.
13. Click the **…** (browse) button in the *Part Selection* section.
14. In the *Device Selection Dialog* page, select **Parts Selection** field, enter **xc7z020clg400-1** in the *Search* field and click **OK**:
    Alternatively, click on **Boards** and select **pynq-z2.** This will set the **Part** to **xc7z020clg400-1**.



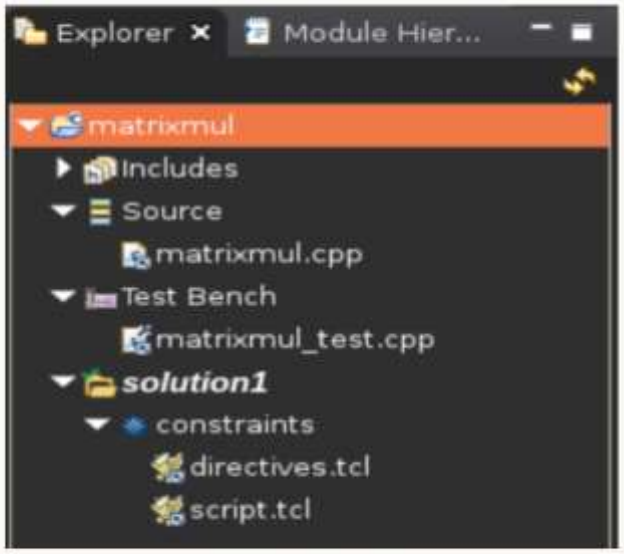
Product Category: All
Family: All
Sub-Family: All
Package: All
Speed grade: All
Temp grade: All

Search: xc7z020clg400-1 (1 matches)

Part	Family	Package	Speed	SLICE	LUT	FF
xc7z020clg400-1	Zynq-7000	clg400	-1	13300	53200	106

Using Parts Specify option in Part Selection Dialog

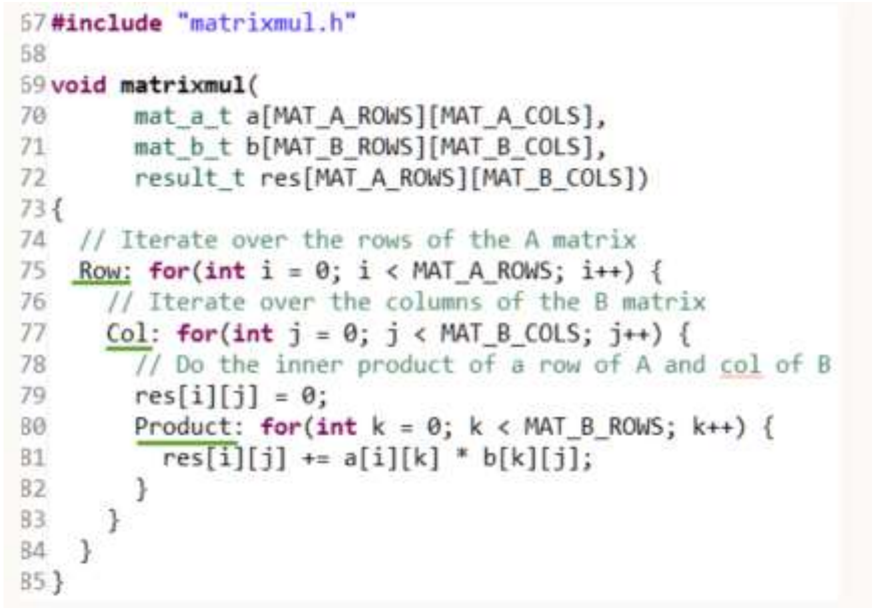
15. Click **Finish.** You will see the created project in the Explorer view. Expand various sub-folders to see the entries under each sub-folder.



```
matrixmul
  Includes
  Source
    matrixmul.cpp
  Test Bench
    matrixmul_test.cpp
  solution1
    constraints
      directives.tcl
      script.tcl
```

Explorer Window

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



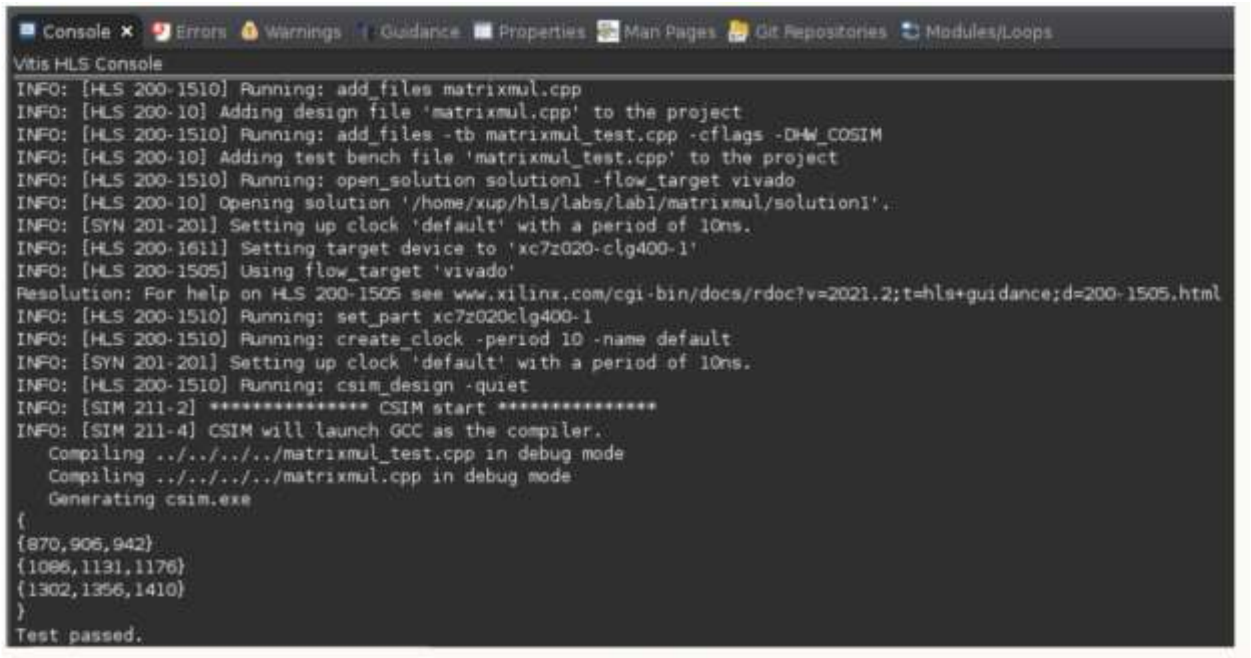
```
67 #include "matrixmul.h"
68
69 void matrixmul(
70       mat_a_t a[MAT_A_ROWS][MAT_A_COLS],
71       mat_b_t b[MAT_B_ROWS][MAT_B_COLS],
72       result_t res[MAT_A_ROWS][MAT_B_COLS])
73 {
74   // Iterate over the rows of the A matrix
75   Row: for(int i = 0; i < MAT_A_ROWS; i++) {
76     // Iterate 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 }
```

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.

STEP 2: Run C Simulation

1. Select **Project > Run C Simulation**, and click **OK** in the C Simulation Dialog windows. The files will be compiled and you will see the output in the Console window.



```
INFO: [HLS 200-1510] Running: add_files matrixmul.cpp
INFO: [HLS 200-10] Adding design file 'matrixmul.cpp' to the project
INFO: [HLS 200-1510] Running: add_files -tb matrixmul_test.cpp -cflags -DHW_COSIM
INFO: [HLS 200-10] Adding test bench file 'matrixmul_test.cpp' to the project
INFO: [HLS 200-1510] Running: open_solution solution1 -flow_target vivado
INFO: [HLS 200-10] Opening solution '/home/xup/hls/labs/lab1/matrixmul/solution1'.
INFO: [SYN 201-201] Setting up clock 'default' with a period of 10ns.
INFO: [HLS 200-1611] Setting target device to 'xc7z020-clg400-1'
INFO: [HLS 200-1505] Using flow_target 'vivado'
Resolution: For help on HLS 200-1505 see www.xilinx.com/cgi-bin/docs/rdoc?v=2021.2;t=hls+guidance;d=200-1505.html
INFO: [HLS 200-1510] Running: set_part xc7z020clg400-1
INFO: [HLS 200-1510] Running: create_clock -period 10 -name default
INFO: [SYN 201-201] Setting up clock 'default' with a period of 10ns.
INFO: [HLS 200-1510] Running: csim_design -quiet
INFO: [SIM 211-2] *************** CSIM start ***************
INFO: [SIM 211-4] CSIM will launch GCC as the compiler.
   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.
```

Program output

2. 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 that executes the algorithm. If HW_COSIM is defined (as was done during the project set-up) 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 **matrixmul** function.

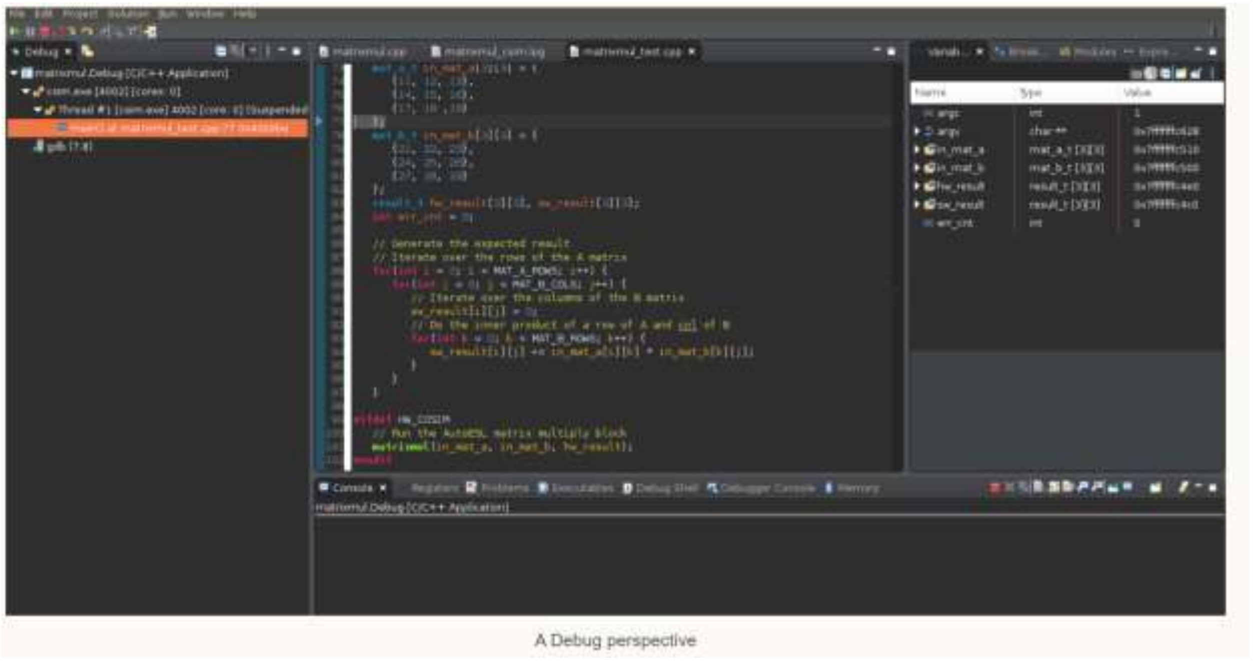
STEP 3: Run Debugger

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

1. Select **Project > Run C Simulation.** 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.

2. The *Debug* perspective will show the **matrixmul_test.cpp** in the source view, **argc** and **argv** variables defined in the *Variables* view, thread created and the program suspended at the main() function entry point in *Debug* view.



```
matrixmul.Debug [C/C++ Application]
  csim.exe [4002] [cores: 0]
    Thread #1 [csim.exe] 4002 [core: 0] (Suspended
      main() at matrixmul_test.cpp:77 0x4
  gdb (7.8)
```

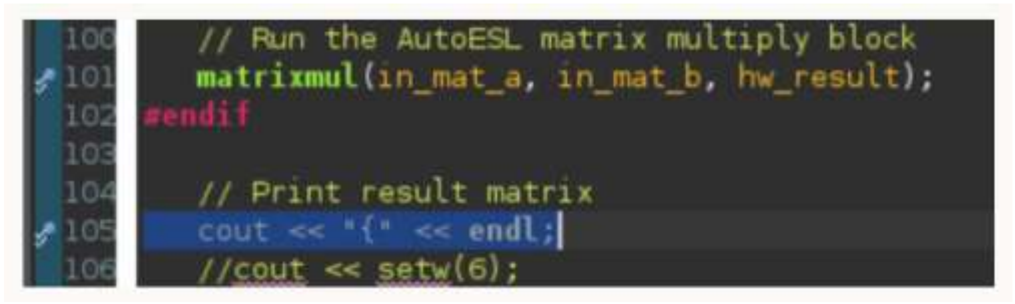
```
    mat_a_t in_mat_a[3][3] = {
        {11, 12, 13},
        {14, 15, 16},
        {17, 18 ,19}
    };
    mat_b_t in_mat_b[3][3] = {
        {21, 22, 23},
        {24, 25, 26},
        {27, 28, 29}
    };
    result_t hw_result[3][3], sw_result[3][3];
    int err_cnt = 0;

    // Generate the expected result
    // Iterate over the rows of the A matrix
    for(int i = 0; i < MAT_A_ROWS; i++) {
        for(int j = 0; j < MAT_B_COLS; j++) {
            // Iterate over the columns of the B matrix
            sw_result[i][j] = 0;
            // Do the inner product of a row of A and col of B
            for(int k = 0; k < MAT_B_ROWS; k++) {
                sw_result[i][j] += in_mat_a[i][k] * in_mat_b[k][j];
            }
        }
    }

#ifdef HW_COSIM
    // Run the AutoESL matrix multiply block
    matrixmul(in_mat_a, in_mat_b, hw_result);
#endif
```

Name	Type	Value
argc	int	1
argv	char **	0x7fffffffc628
in_mat_a	mat_a_t [3][3]	0x7fffffffc510
in_mat_b	mat_b_t [3][3]	0x7fffffffc500
hw_result	result_t [3][3]	0x7fffffffc4e0
sw_result	result_t [3][3]	0x7fffffffc4c0
err_cnt	int	0

```
matrixmul.Debug [C/C++ Application]
```

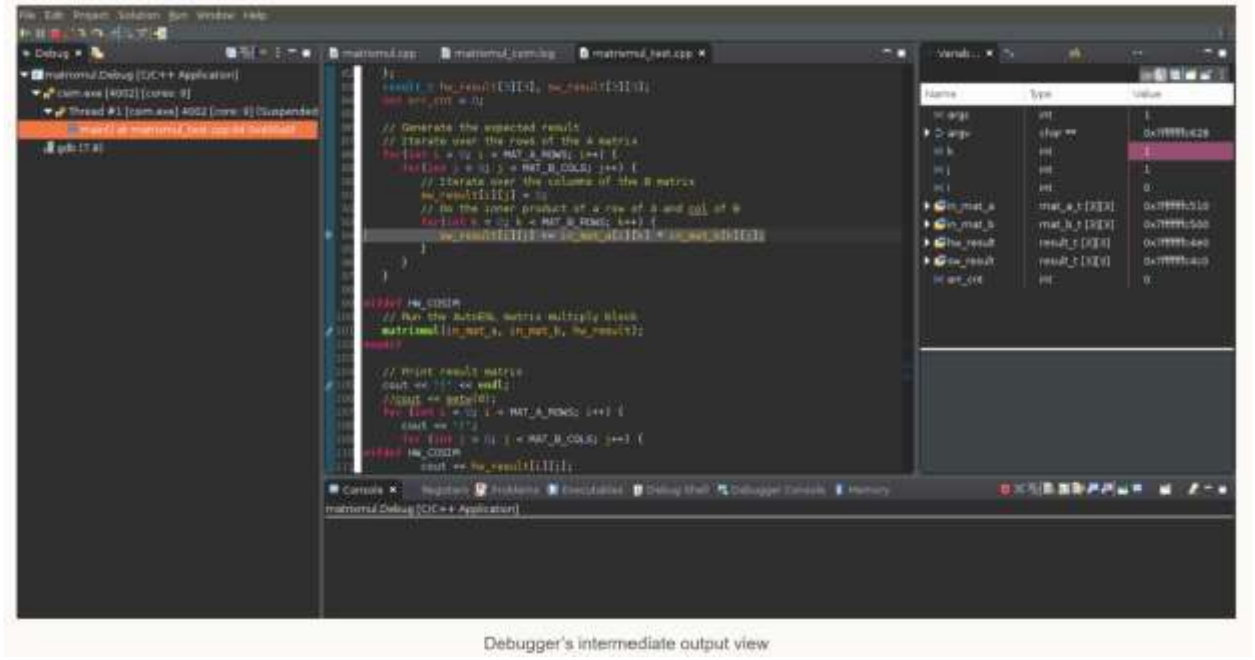
A Debug perspective

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.



```
100     // Run the AutoESL matrix multiply block
101     matrixmul(in_mat_a, in_mat_b, hw_result);
102 #endif
103
104     // Print result matrix
105     cout << "{" << endl;
106     //cout << setw(6);
```

4. Similarly, set a breakpoint at line 101 on the matrixmul() function
5. Using the **Step Over (F6)** button several times, observe the execution progress, and observe the variable values updating, as well as computed software result.



```
matrixmul Debug [C/C++ Application]
  csim.exe [4002] [cores: 0]
    Thread #1 [csim.exe] 4002 [core: 0] (Suspended
      main() at matrixmul_test.cpp:94 0x40...
  gdb (7.6)
```

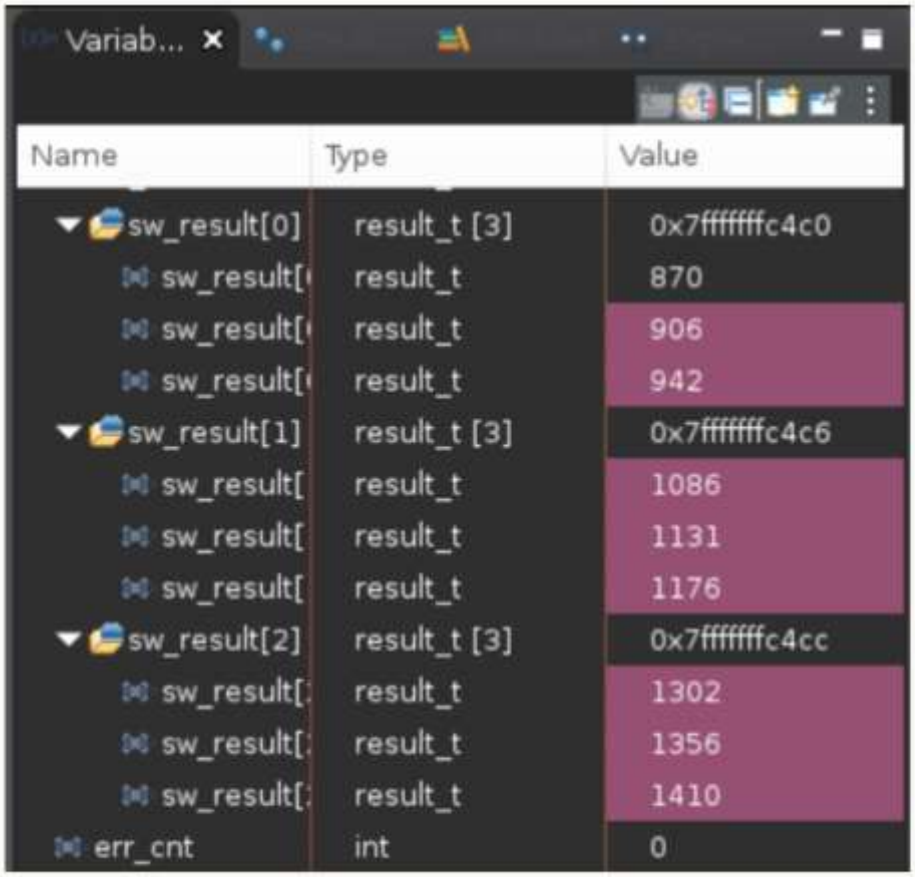
```
    };
    result_t hw_result[3][3], sw_result[3][3];
    int err_cnt = 0;

    // Generate the expected result
    // Iterate over the rows of the A matrix
    for(int i = 0; i < MAT_A_ROWS; i++) {
        for(int j = 0; j < MAT_B_COLS; j++) {
            // Iterate over the columns of the B matrix
            sw_result[i][j] = 0;
            // Do the inner product of a row of A and col of B
            for(int k = 0; k < MAT_B_ROWS; k++) {
                sw_result[i][j] += in_mat_a[i][k] * in_mat_b[k][j];
            }
        }
    }

#ifdef HW_COSIM
    // Run the AutoESL matrix multiply block
    matrixmul(in_mat_a, in_mat_b, hw_result);
#endif

    // Print result matrix
    cout << "{" << endl;
    //cout << setw(8);
    for (int i = 0; i < MAT_A_ROWS; i++) {
        cout << "{";
        for (int j = 0; j < MAT_B_COLS; j++) {
#ifdef HW_COSIM
            cout << hw_result[i][j];
```

Name	Type	Value
argc	int	1
argv	char **	0x7fffffffc628
k	int	1
j	int	1
i	int	0
in_mat_a	mat_a_t [3][3]	0x7fffffffc510
in_mat_b	mat_b_t [3][3]	0x7fffffffc500
hw_result	result_t [3][3]	0x7fffffffc4e0
sw_result	result_t [3][3]	0x7fffffffc4c0
err_cnt	int	0

```
matrixmul Debug [C/C++ Application]
```

Debugger's intermediate output view

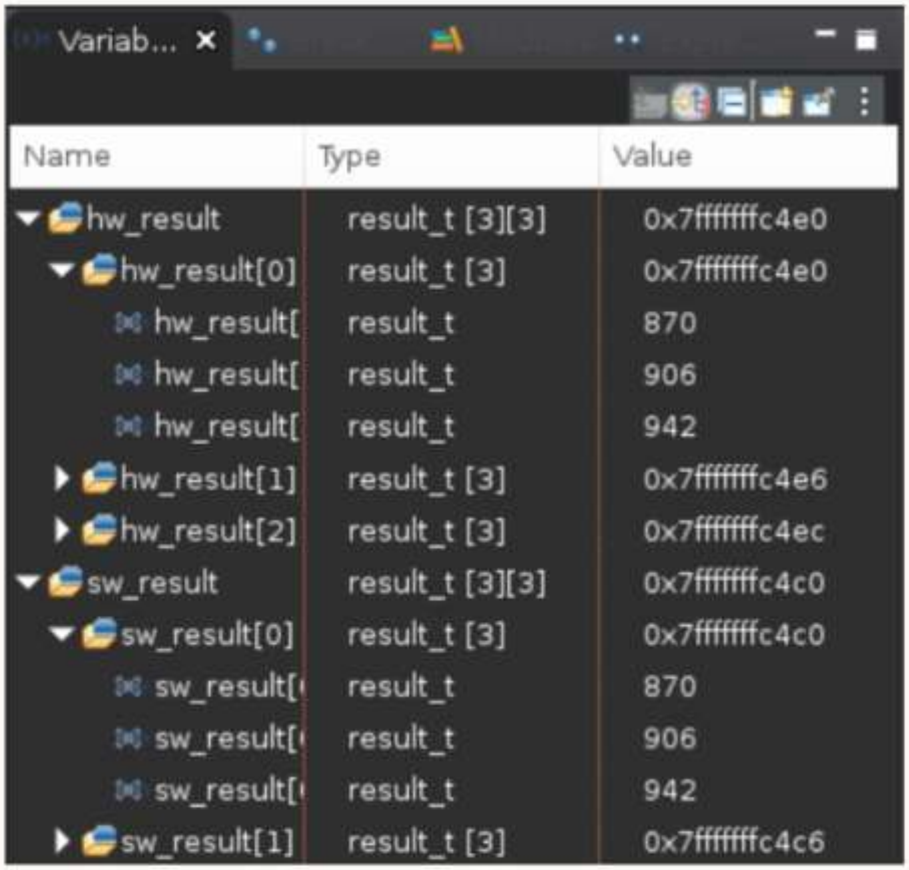
6. Now click the **Resume** button or **F8** to complete the software computation and stop at line 101.
7. Observe the following computed software result in the variables view.



Name	Type	Value
sw_result[0]	result_t [3]	0x7fffffffc4c0
sw_result[	result_t	870
sw_result[	result_t	906
sw_result[	result_t	942
sw_result[1]	result_t [3]	0x7fffffffc4c6
sw_result[	result_t	1086
sw_result[	result_t	1131
sw_result[	result_t	1176
sw_result[2]	result_t [3]	0x7fffffffc4cc
sw_result[	result_t	1302
sw_result[	result_t	1356
sw_result[	result_t	1410
err_cnt	int	0

Software computed result

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



Name	Type	Value
hw_result	result_t [3][3]	0x7fffffffc4e0
hw_result[0]	result_t [3]	0x7fffffffc4e0
hw_result[	result_t	870
hw_result[	result_t	906
hw_result[	result_t	942
hw_result[1]	result_t [3]	0x7fffffffc4e6
hw_result[2]	result_t [3]	0x7fffffffc4ec
sw_result	result_t [3][3]	0x7fffffffc4c0
sw_result[0]	result_t [3]	0x7fffffffc4c0
sw_result[	result_t	870
sw_result[	result_t	906
sw_result[	result_t	942
sw_result[1]	result_t [3]	0x7fffffffc4c6

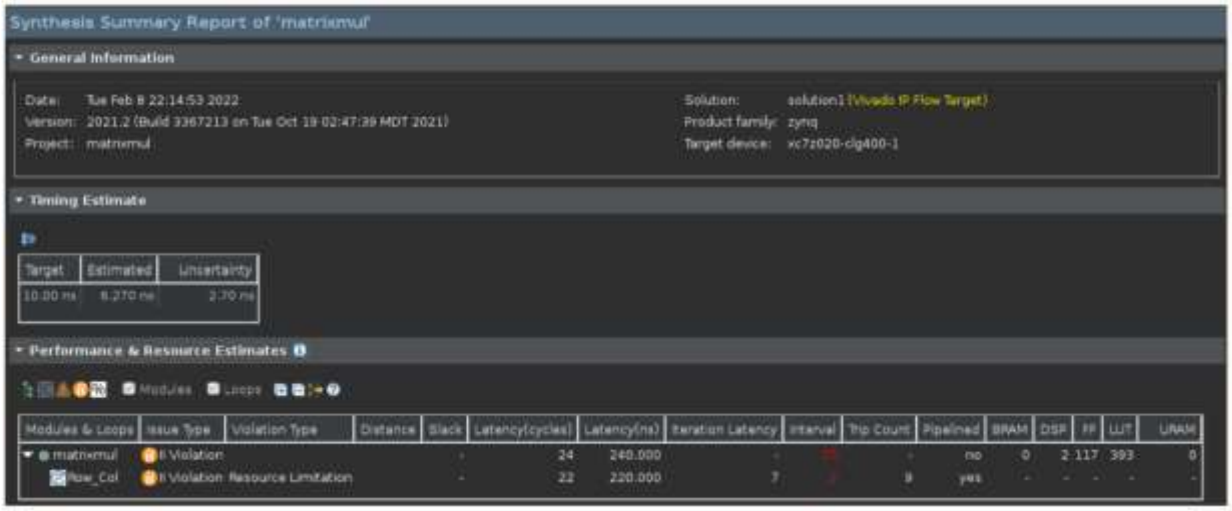
Computed results

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

STEP 4: Synthesize the Design

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.

1. Switch to the Synthesis view by clicking the **Exit Debug** button.
2. Select **Solution > Run C Synthesis > Active Solution** or click on the button to start the synthesis process.
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.



Synthesis Summary Report of 'matrixmul'

General Information

Date:	Tue Feb 8 22:14:53 2022	Solution:	solution1 (Vivado IP Flow Target)
Version:	2021.2 (Build 3367213 on Tue Oct 19 02:47:39 MDT 2021)	Product family:	zynq
Project:	matrixmul	Target device:	xc7z020-clg400-1

Timing Estimate

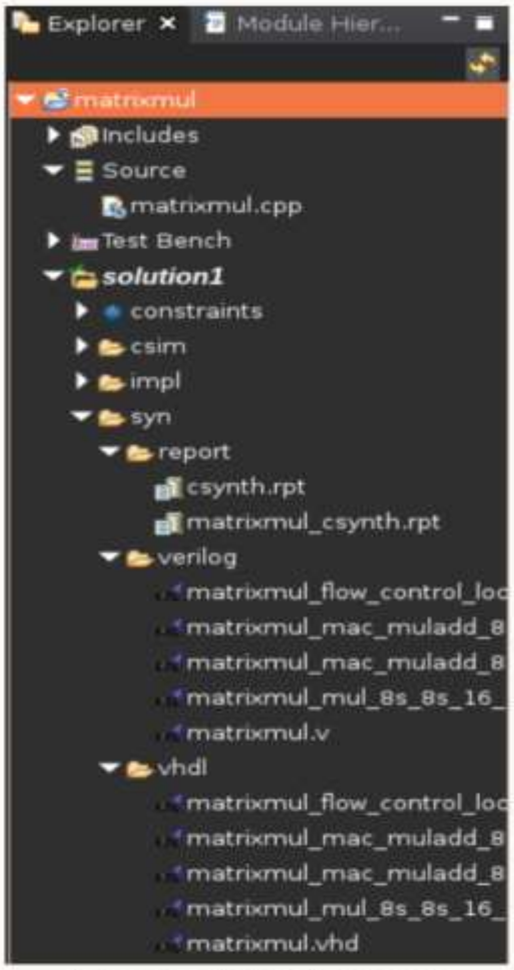
Target	Estimated	Uncertainty
10.00 ns	8.270 ns	2.70 ns

Performance & Resource Estimates

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
matrixmul	II Violation			-	24	240.000	-	[illegible]	-	no	0	2	117	393	0
Row_Col	II Violation	Resource Limitation		-	22	220.000	7	[illegible]	9	yes	-	-	-	-	-

Report view after synthesis is completed

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

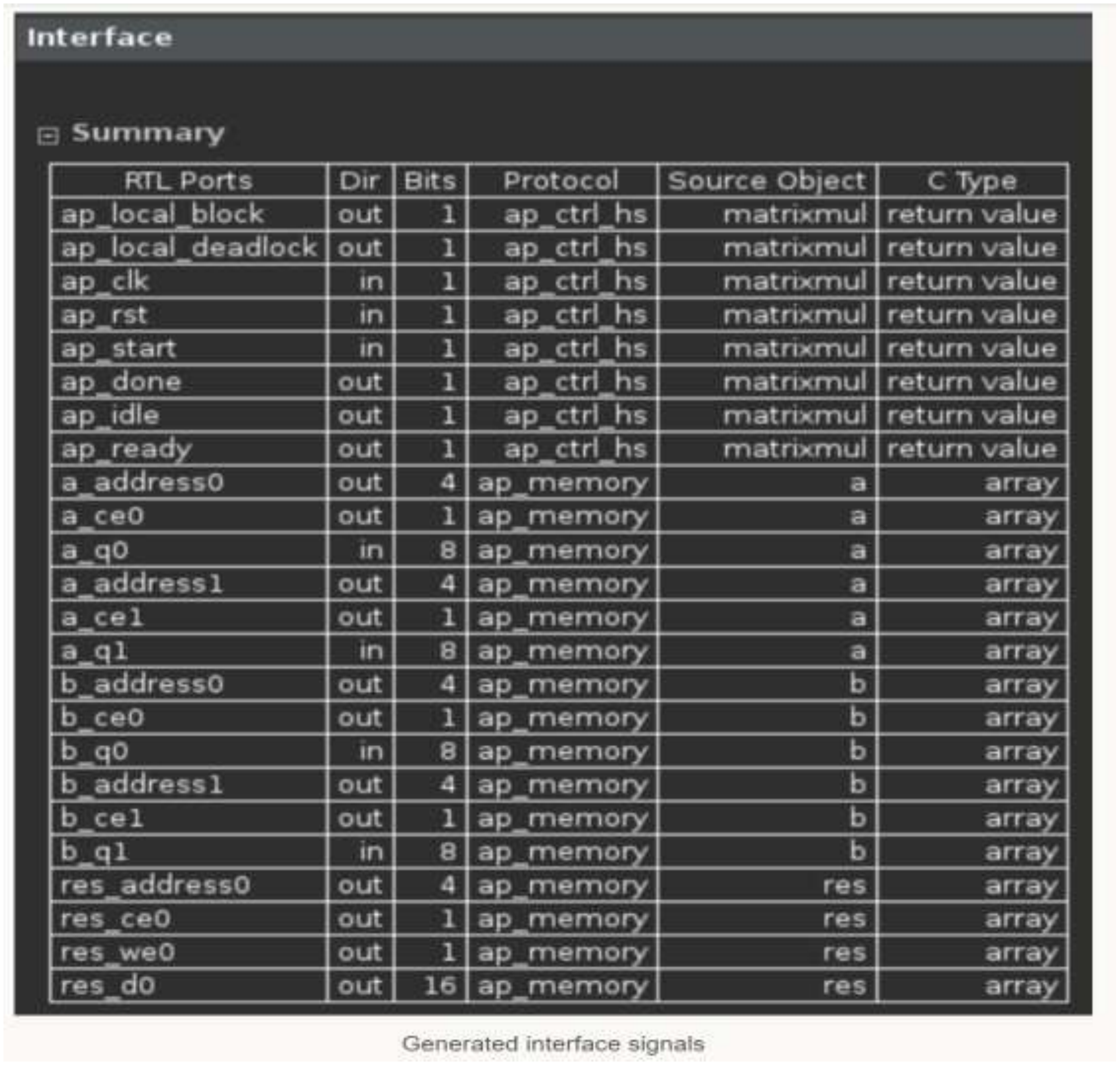


```
matrixmul
  Includes
  Source
    matrixmul.cpp
  Test Bench
  solution1
    constraints
    csim
    impl
    syn
      report
        csynth.rpt
        matrixmul_csynth.rpt
      verilog
        matrixmul_flow_control_loc
        matrixmul_mac_muladd_8
        matrixmul_mac_muladd_8
        matrixmul_mul_8s_8s_16_
        matrixmul.v
      vhdl
        matrixmul_flow_control_loc
        matrixmul_mac_muladd_8
        matrixmul_mac_muladd_8
        matrixmul_mul_8s_8s_16_
        matrixmul.vhd
```

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, verilog***, and ***vhdl*** sub-folders under which report files, and generated source (vhdl, verilog, header, and cpp) files can be found. 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.

5. The ***Synthesis*** Report shows the performance and resource estimates as well as estimated latency in the design.
6. Using scroll bar on the right, scroll down into the report and answer the following question.
   Question 1 Answer the following question:
   Estimated clock period:
   Worst case latency:
   Number of DSP48E used:
   Number of FFs used:
   Number of LUTs used:
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_local_block	out	1	ap_ctrl_hs	matrixmul	return value
ap_local_deadlock	out	1	ap_ctrl_hs	matrixmul	return value
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
a_address1	out	4	ap_memory	a	array
a_ce1	out	1	ap_memory	a	array
a_q1	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
b_address1	out	4	ap_memory	b	array
b_ce1	out	1	ap_memory	b	array
b_q1	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

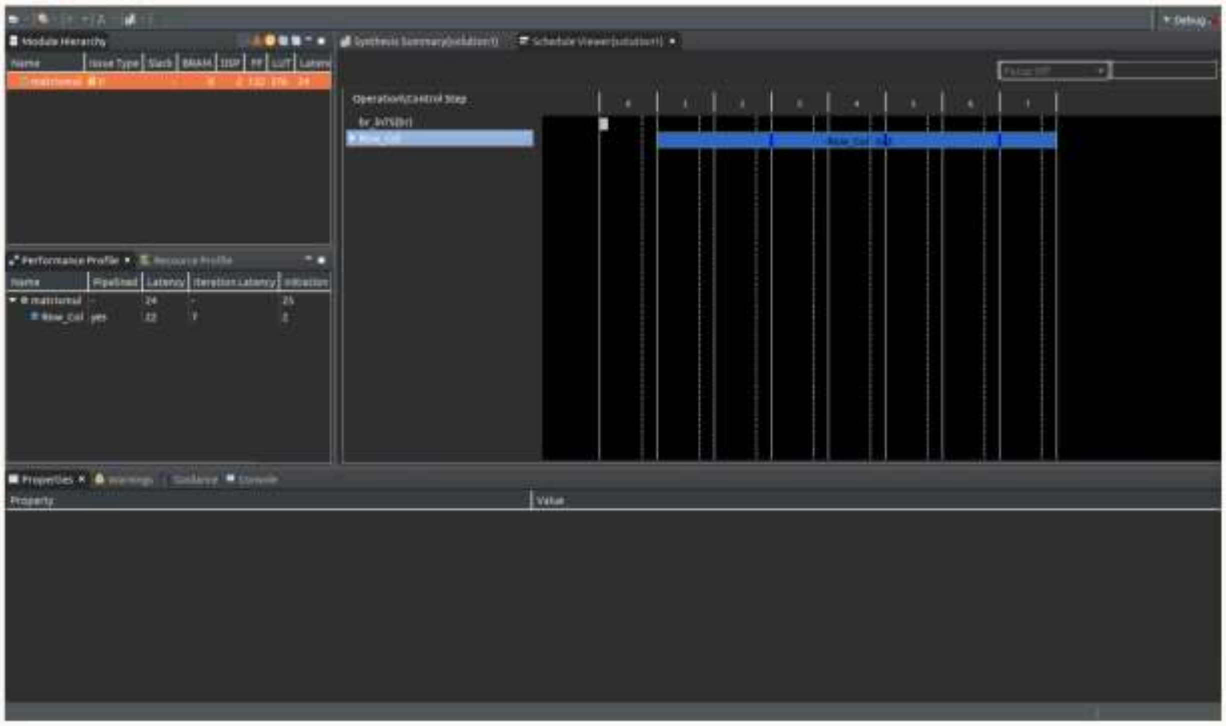
Generated interface signals

You can see **ap_clk, ap_rst**, **ap_idle** and **ap_ready** control signals are automatically added to the design by default. These signals are used as handshaking signals to indicate when the design is ready to begin the 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 input and output signals in the design and their default or specified interfaces.

STEP 5: Analyze using Analysis Perspective

Switch to the Analysis Perspective and understand the design behavior.

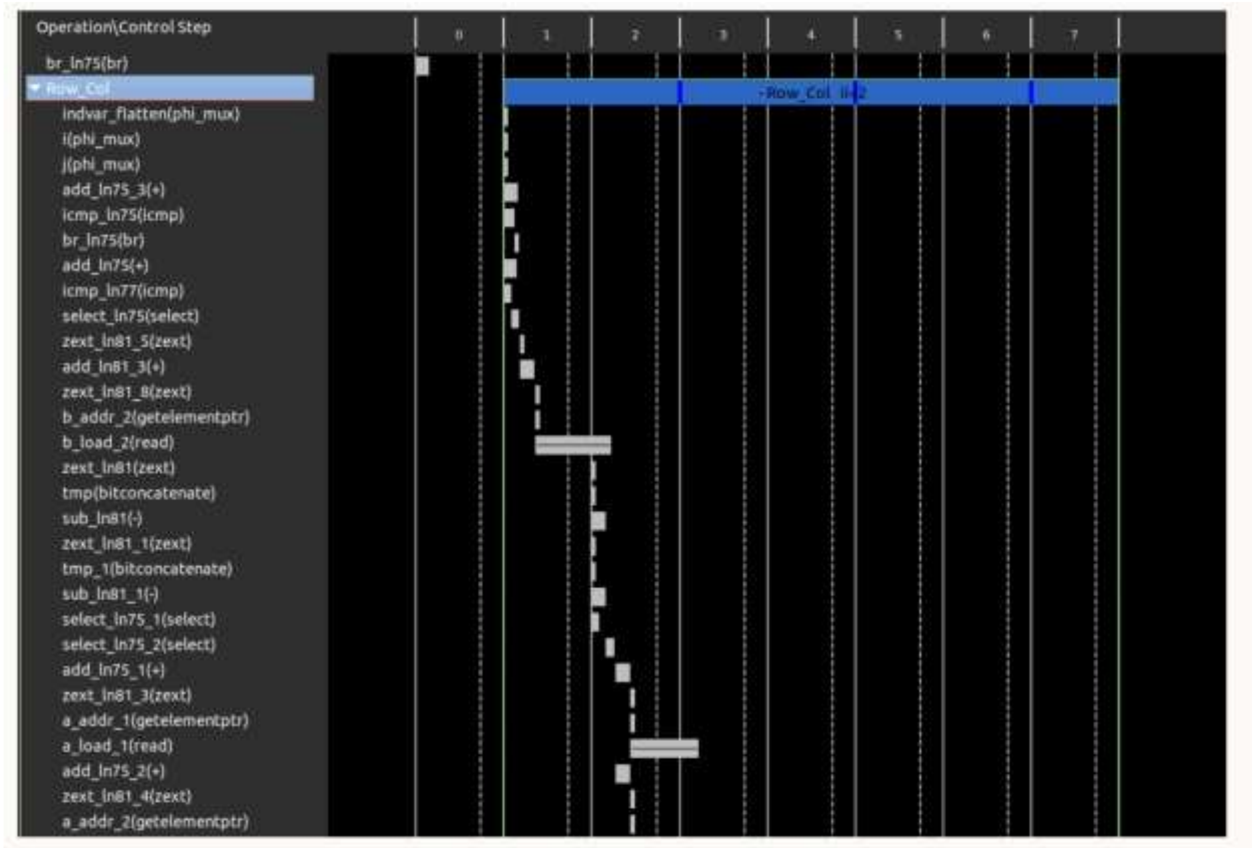
1. Select **Solution > Open Schedule Viewer** or click on *Analysis* button to open the analysis viewer. The Analysis perspective consists of 4 panes as shown below. Note that the module and loops hierarchies are displayed unexpanded by default. The **Module Hierarchy** pane shows both the performance 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 **Schedule Viewer** is also shown in the right-hand side pane. This view shows how the operations in this particular block are schedules into clock cycles.



Name	Pipelined	Latency	Iteration Latency	Initiation
matrixmul	-	24	-	25
Row_Col	yes	22	7	2

Analysis perspective

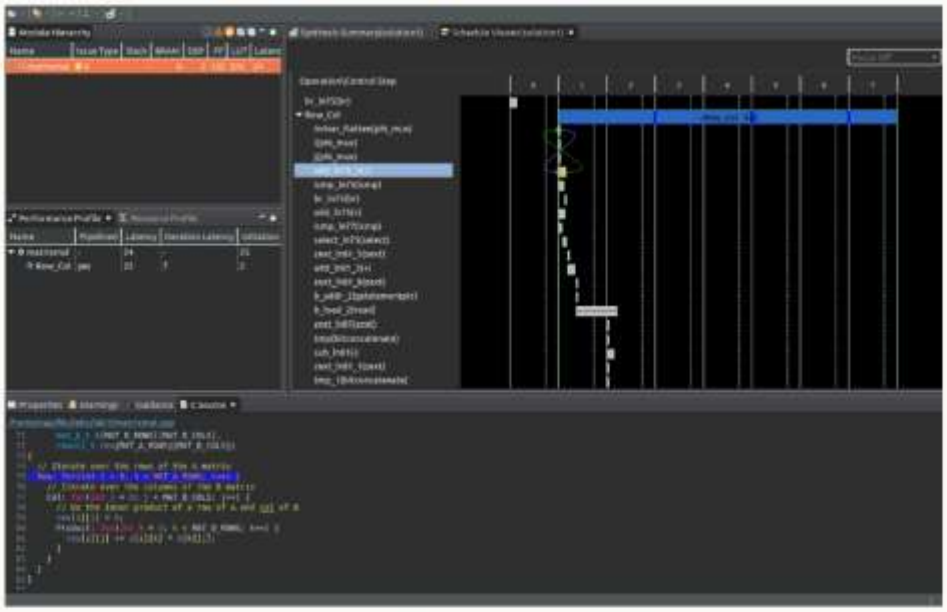
2. Click on > of loop **Row** to expand.



Performance matrix showing top-level Row operation

From this we can see that there is an add operation performed. This addition is likely the counter to count the loop iterations, and we can confirm this.

3. Select the block for the **adder**, right-click and select **Goto Source**. The source code pane will be opened, highlighting line 75 where the loop index is being tested and incremented.

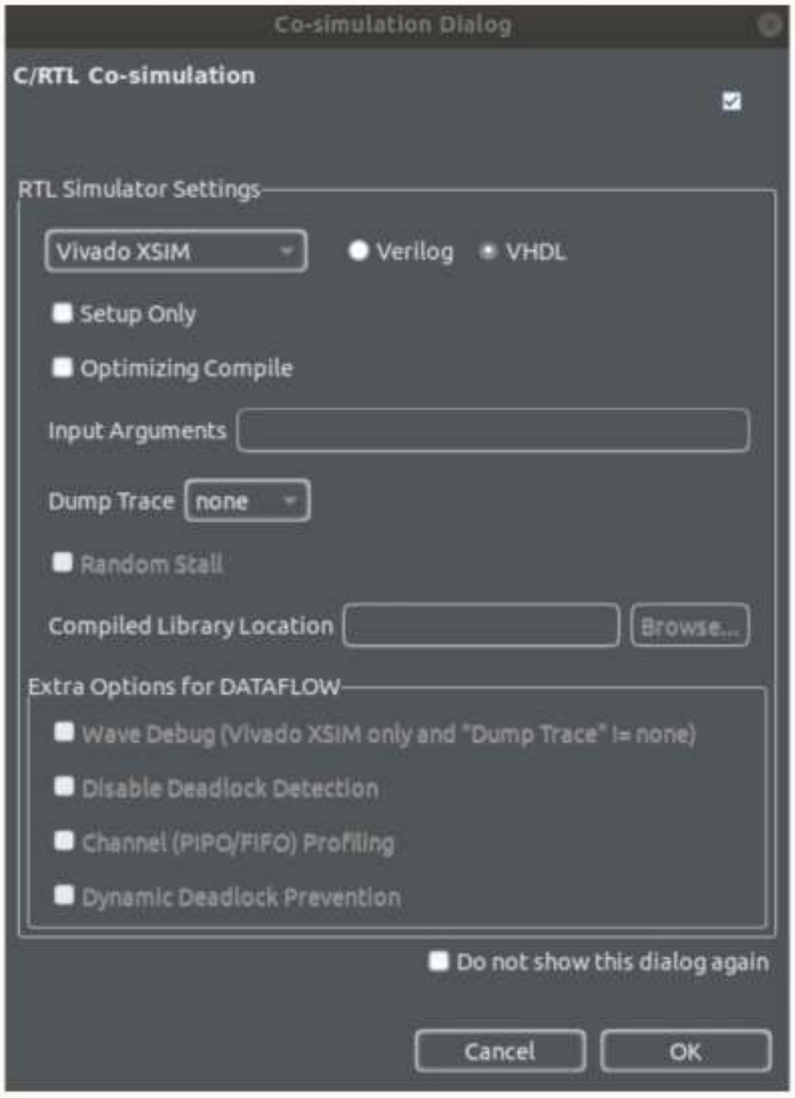

Cross probing into the source file

4. Click on the **Synthesis** tool bar button to switch back to the Synthesis view.

STEP 6: Run C/RTL Co-simulation

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

1. Select **Solution>Run C/RTL Cosimulation** to open the dialog box so the desired simulations can be selected and run. A **C/RTL Co-simulation Dialog box** will open.
2. Make sure the **VHDL** option is selected. This allows the simulation to be performed using VHDL. To perform the verification using Verilog, you can select Verilog.



Co-simulation Dialog

C/RTL Co-simulation

RTL Simulator Settings

Vivado XSIM   Verilog [✓ VHDL]

Setup Only

Optimizing Compile

Input Arguments

Dump Trace: none

Random Stall

Compiled Library Location   Browse...

Extra Options for DATAFLOW

Wave Debug (Vivado XSIM only and "Dump Trace" != none)

Disable Deadlock Detection

Channel (PIPO/FIFO) Profiling

Dynamic Deadlock Prevention

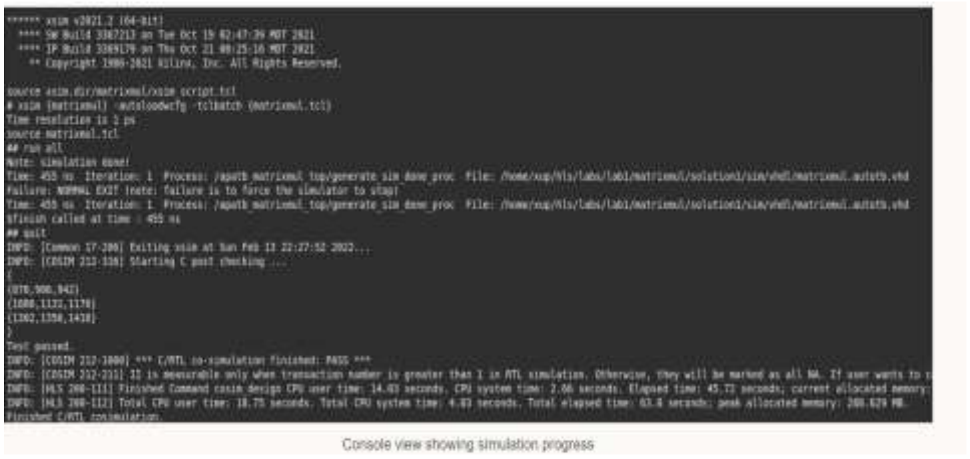
Do not show this dialog again

Cancel   OK

A C/RTL Co-simulation Dialog

3. Click **OK** to run the VHDL 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 VHDL 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

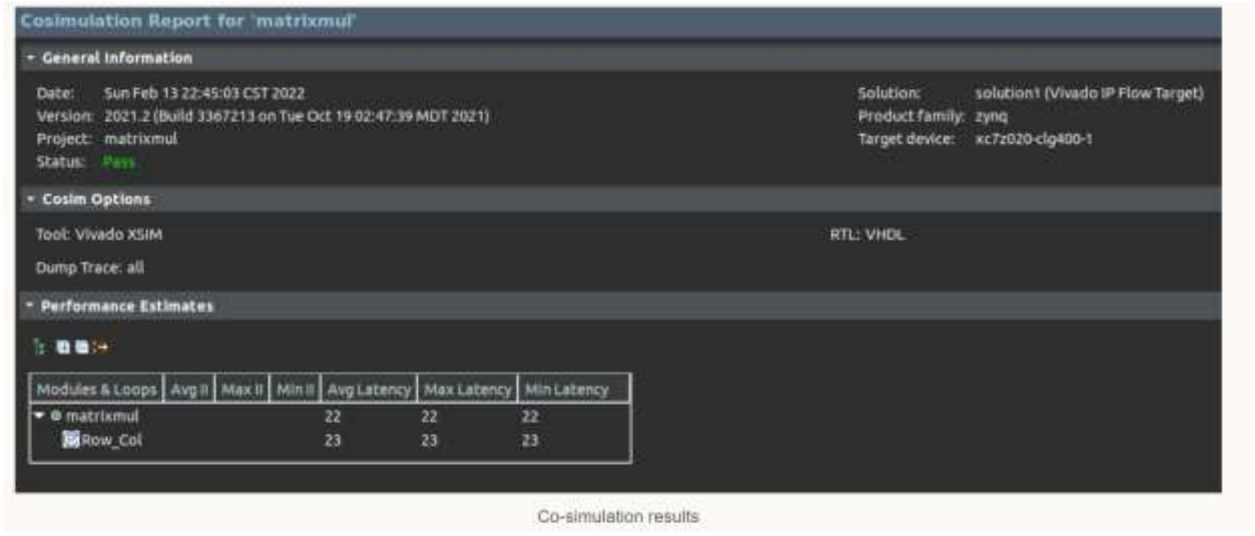
- Finally, the output from the RTL is re-applied to the VHDL test bench to check the results. In the console window you can see the progress and also a message that the test is passed. This eliminates writing a separate testbench for the synthesized design.



```
****** xsim v2021.2 (64-bit)
  **** SW Build 3367213 on Tue Oct 19 02:47:39 MDT 2021
  **** IP Build 3369179 on Thu Oct 21 08:25:16 MDT 2021
    ** Copyright 1986-2021 Xilinx, Inc. All Rights Reserved.

source xsim.dir/matrixmul/xsim_script.tcl
# xsim {matrixmul} -autoloadwcfg -tclbatch {matrixmul.tcl}
Time resolution is 1 ps
source matrixmul.tcl
## run all
Note: simulation done!
Time: 455 ns  Iteration: 1  Process: /apatb_matrixmul_top/generate_sim_done_proc  File: /home/xup/hls/labs/lab1/matrixmul/solution1/sim/vhdl/matrixmul.autotb.vhd
Failure: NORMAL EXIT (note: failure is to force the simulator to stop)
Time: 455 ns  Iteration: 1  Process: /apatb_matrixmul_top/generate_sim_done_proc  File: /home/xup/hls/labs/lab1/matrixmul/solution1/sim/vhdl/matrixmul.autotb.vhd
$finish called at time : 455 ns
## quit
INFO: [Common 17-206] Exiting xsim at Sun Feb 13 22:27:52 2022...
INFO: [COSIM 212-316] Starting C post checking ...
{
{870,906,942}
{1086,1131,1176}
{1302,1356,1410}
}
Test passed.
INFO: [COSIM 212-1000] *** C/RTL co-simulation finished: PASS ***
INFO: [COSIM 212-211] II is measurable only when transaction number is greater than 1 in RTL simulation. Otherwise, they will be marked as all NA. If user wants to
INFO: [HLS 200-111] Finished Command cosim_design CPU user time: 14.03 seconds. CPU system time: 2.06 seconds. Elapsed time: 45.72 seconds; current allocated memory:
INFO: [HLS 200-112] Total CPU user time: 18.75 seconds. Total CPU system time: 4.03 seconds. Total elapsed time: 63.6 seconds; peak allocated memory: 208.629 MB.
Finished C/RTL cosimulation.
```

Console view showing simulation progress

4. 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 VHDL, the result shows the latencies and interval (initiation) which indicates after how many clock cycles later the next input can be provided.



Cosimulation Report for 'matrixmul'

General Information

Date: Sun Feb 13 22:45:03 CST 2022
Version: 2021.2 (Build 3367213 on Tue Oct 19 02:47:39 MDT 2021)
Project: matrixmul
Status: Pass

Solution: solution1 (Vivado IP Flow Target)
Product family: zynq
Target device: xc7z020-clg400-1

Cosim Options

Tool: Vivado XSIM
Dump Trace: all
RTL: VHDL

Performance Estimates

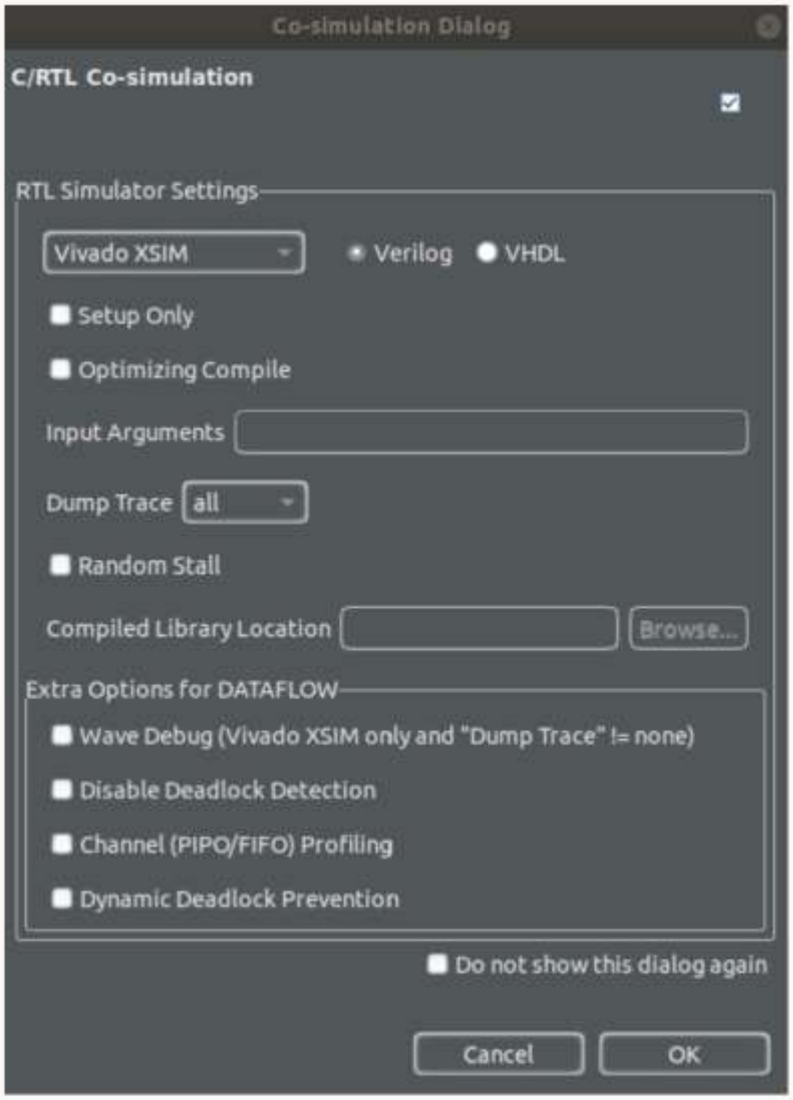
Modules & Loops	Avg II	Max II	Min II	Avg Latency	Max Latency	Min Latency
matrixmul				22	22	22
Row_Col				23	23	23

Co-simulation results

STEP 7: Viewing Simulation Results in Vivado

Run Verilog simulation with Dump Trace option selected.

1. Select **Solution > Run C/RTL Cosimulation** to open the dialog box so the desired simulations can be run.
2. Click on the **Verilog** Selection option.
   Optionally, you can click on the drop-down button and select the desired simulator from the available list of Vivado XSim, ModelSim and Riviera.
3. Select **All** for the *Dump Trace* option and click **OK.**



Co-simulation Dialog

C/RTL Co-simulation ✓

RTL Simulator Settings

Vivado XSIM [✓ Verilog] VHDL

☐ Setup Only

☐ Optimizing Compile

Input Arguments

Dump Trace: all

☐ Random Stall

Compiled Library Location

Extra Options for DATAFLOW

☐ Wave Debug (Vivado XSIM only and "Dump Trace" != none)

☐ Disable Deadlock Detection

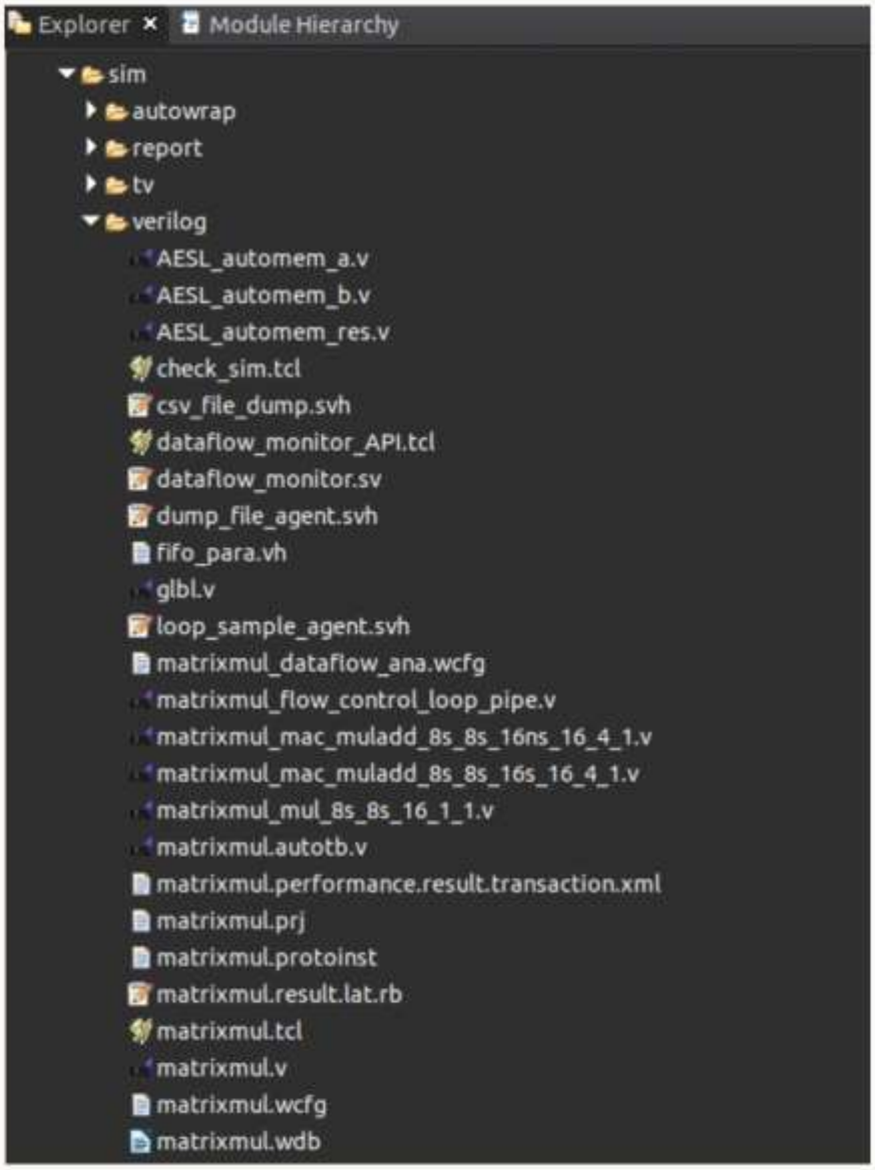
☐ Channel (PIPO/FIFO) Profiling

☐ Dynamic Deadlock Prevention

☐ Do not show this dialog again

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 (**matrixmul.wcfg** and **matrixmul.wdb**) can be seen in the Verilog simulation directory.

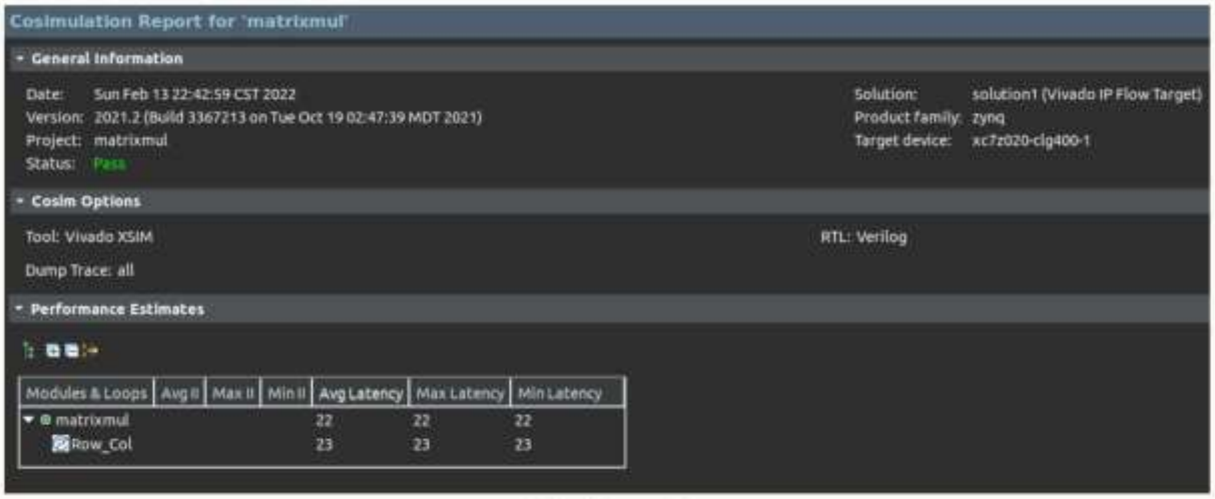


Explorer × Module Hierarchy

```
▾ sim
  ▸ autowrap
  ▸ report
  ▸ tv
  ▾ verilog
      AESL_automem_a.v
      AESL_automem_b.v
      AESL_automem_res.v
      check_sim.tcl
      csv_file_dump.svh
      dataflow_monitor_API.tcl
      dataflow_monitor.sv
      dump_file_agent.svh
      fifo_para.vh
      glbl.v
      loop_sample_agent.svh
      matrixmul_dataflow_ana.wcfg
      matrixmul_flow_control_loop_pipe.v
      matrixmul_mac_muladd_8s_8s_16ns_16_4_1.v
      matrixmul_mac_muladd_8s_8s_16s_16_4_1.v
      matrixmul_mul_8s_8s_16_1_1.v
      matrixmul.autotb.v
      matrixmul.performance.result.transaction.xml
      matrixmul.prj
      matrixmul.protoinst
      matrixmul.result.lat.rb
      matrixmul.tcl
      matrixmul.v
      matrixmul.wcfg
      matrixmul.wdb
```

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.



Cosimulation Report for 'matrixmul'

General Information

Date: Sun Feb 13 22:42:59 CST 2022
Version: 2021.2 (Build 3367213 on Tue Oct 19 02:47:39 MDT 2021)
Project: matrixmul
Status: Pass

Solution: solution1 (Vivado IP Flow Target)
Product family: zynq
Target device: xc7z020-clg400-1

Cosim Options

Tool: Vivado XSIM
RTL: Verilog
Dump Trace: all

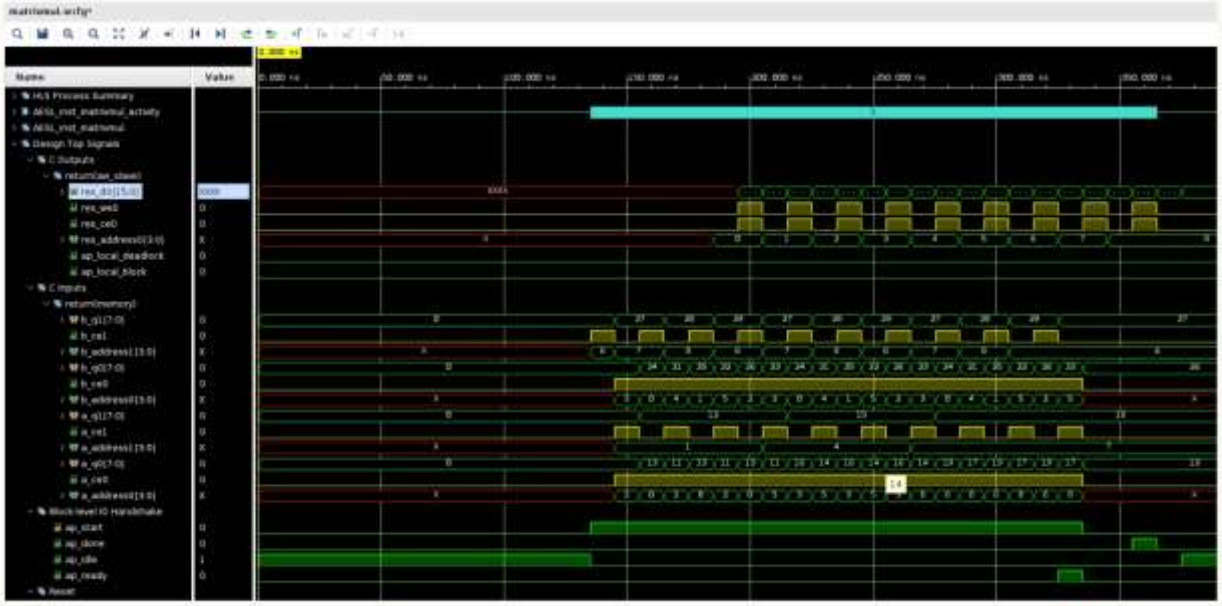
Performance Estimates

Modules & Loops	Avg II	Max II	Min II	Avg Latency	Max Latency	Min Latency
matrixmul				22	22	22
Row_Col				23	23	23

Cosimulation report

Analyze the dumped traces

1. Click on the button on tools bar to open the wave viewer. This will start Vivado and open the wave viewer.
2. Click on the **Zoom Fit** tool button to see the entire simulation of one iteration.
3. Select ***a_address0*** in the waveform window, right-click and select **Radix > Unsigned Decimal**. Similarly, do the same for ***b_address0***, ***a_address1***, ***b_address1*** and **res_address0** signals.
4. Similarly, set the ***a_q0, b_q0, a_q1, b_q1*** and ***res_d0*** radix to **Signed Decimal**. You should see the output similar to the one shown below:



Full waveform showing iteration worth simulation

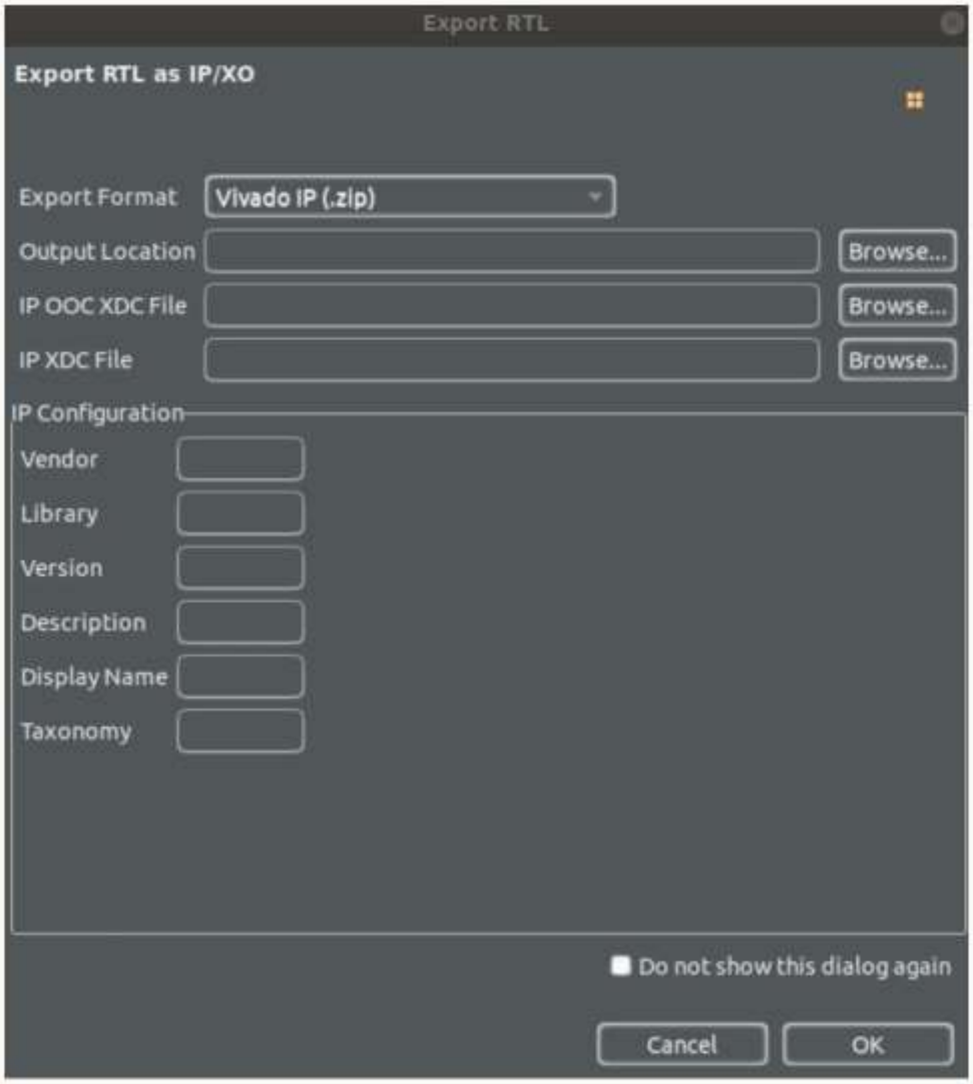
Note that as soon as ***ap_start*** is asserted, ***ap_idle*** has been de-asserted indicating that the design is in computation mode. The ***ap_idle*** signal remains de-asserted until ***ap_done*** is asserted, indicating completion of the process.

5. View various part of the simulation and try to understand how the design works.
6. When done, close Vivado by selecting **File > Exit.** Click **OK** if prompted, and then **Discard** to close the program without saving.

STEP 8: Export RTL and Implement

In Vitis HLS, export the design, selecting VHDL as a language, and run the implementation by selecting Evaluate option.

1. In Vitis HLS, select **Solution > Export RTL** to open the dialog box. An Export RTL Dialog box will open.



Export RTL

Export RTL as IP/XO

Export Format: Vivado IP (.zip)

Output Location:

IP OOC XDC File:

IP XDC File:

IP Configuration

Vendor:

Library:

Version:

Description:

Display Name:

Taxonomy:

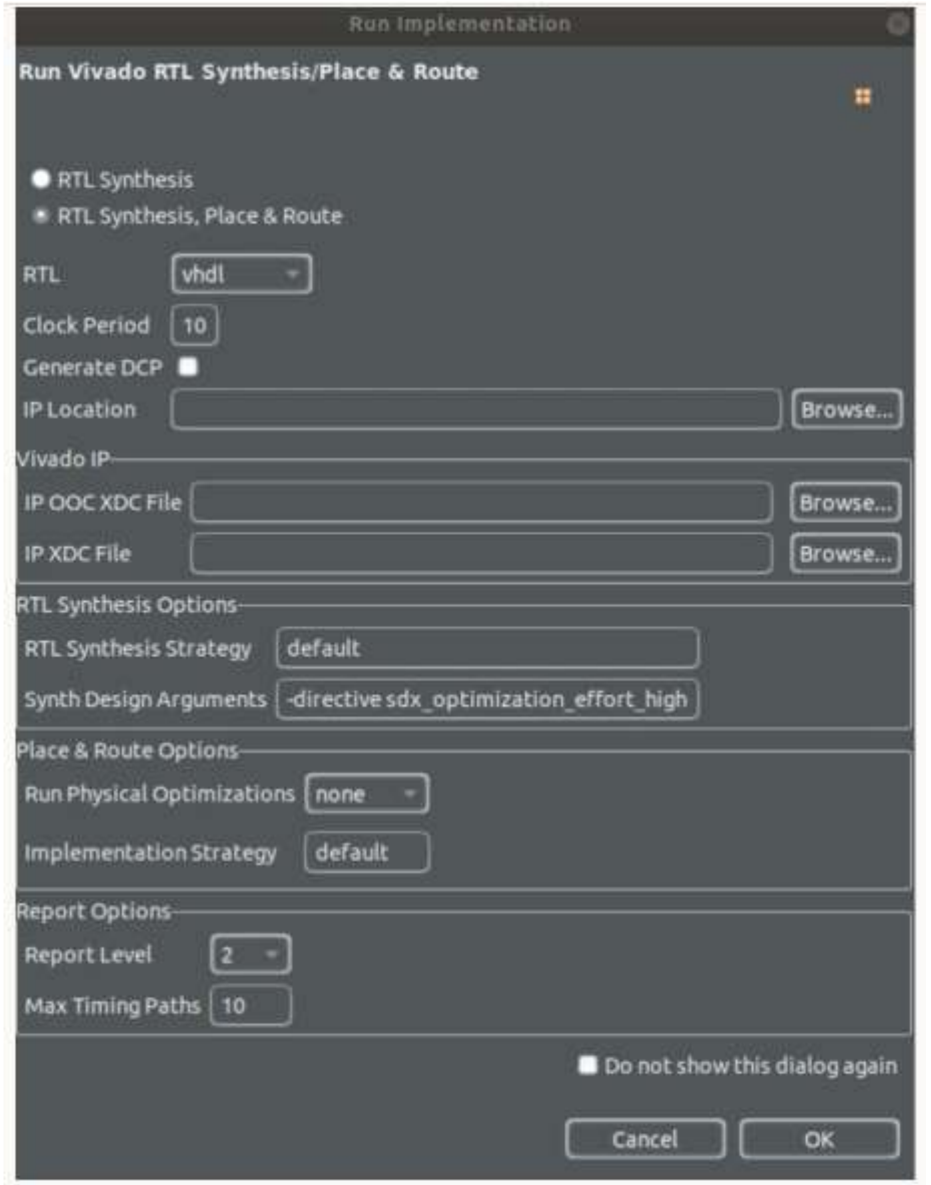
Do not show this dialog again

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 a Vivado IP for System Generator.

2. Select Solution>Implementation to open the dialog box
3. Click on the drop-down menu of the **RTL** field and select **VHDL.**

4. Click on the *RTL Synthesis*, *Place & Route* check box to run the implementation tool.



Run Vivado RTL Synthesis/Place & Route

RTL Synthesis
[✓ RTL Synthesis, Place & Route]

RTL: vhdl
Clock Period: 10
Generate DCP: ☐
IP Location:

Vivado IP
IP OOC XDC File:
IP XDC File:

RTL Synthesis Options
RTL Synthesis Strategy: default
Synth Design Arguments: -directive sdx_optimization_effort_high

Place & Route Options
Run Physical Optimizations: none
Implementation Strategy: default

Report Options
Report Level: 2
Max Timing Paths: 10

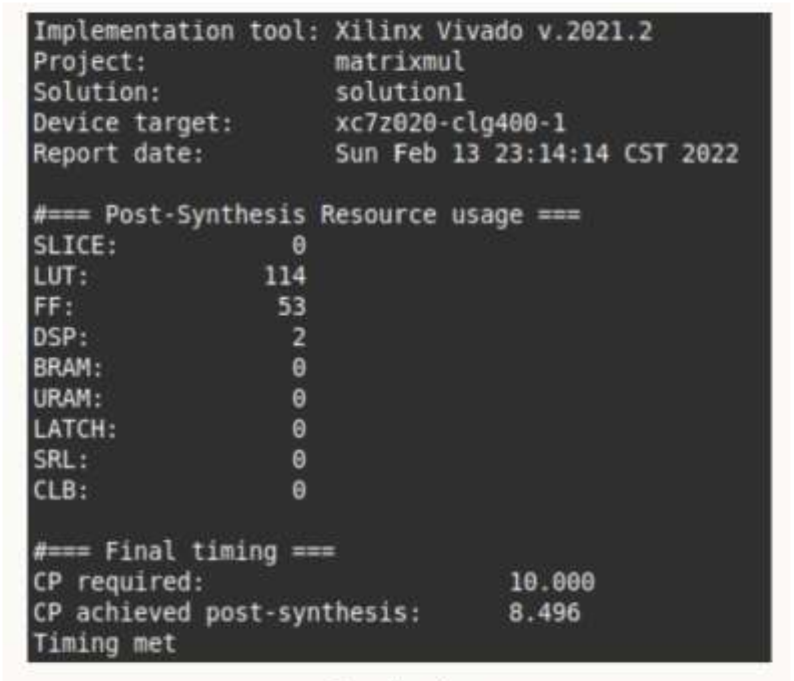
☐ Do not show this dialog again

Selecting Evaluate options

5. Click **OK** and the implementation run will begin.

You can observe the progress in the Vitis HLS Console window. It goes through several phases:

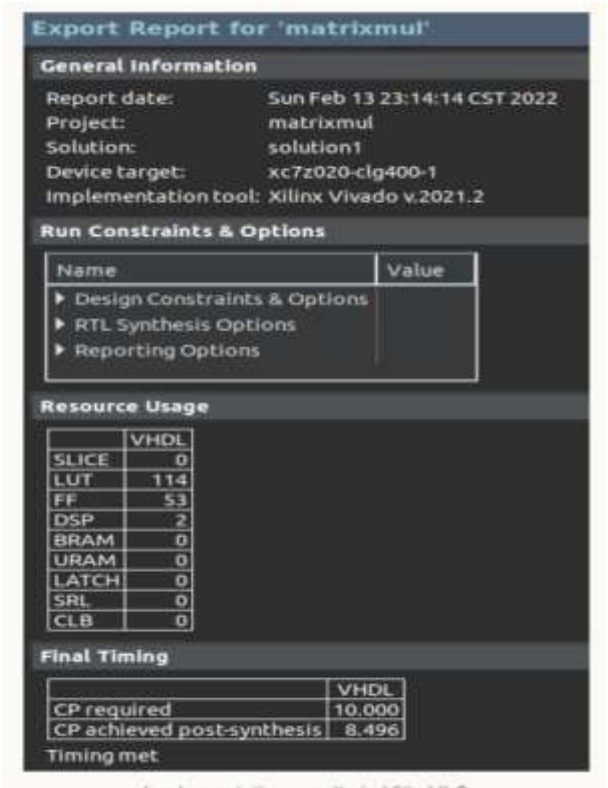
- Exporting RTL as an IP in the IP-XACT format
- RTL evaluation, where it goes through Synthesis and then through Placement and Routing.



```
Implementation tool: Xilinx Vivado v.2021.2
Project:             matrixmul
Solution:            solution1
Device target:       xc7z020-clg400-1
Report date:         Sun Feb 13 23:14:14 CST 2022

#=== Post-Synthesis Resource usage ===
SLICE:           0
LUT:           114
FF:             53
DSP:             2
BRAM:            0
URAM:            0
LATCH:           0
SRL:             0
CLB:             0

#=== Final timing ===
CP required:                     10.000
CP achieved post-synthesis:      8.496
Timing met
```

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:	Sun Feb 13 23:14:14 CST 2022
Project:	matrixmul
Solution:	solution1
Device target:	xc7z020-clg400-1
Implementation tool:	Xilinx Vivado v.2021.2

Run Constraints & Options

Name	Value
▸ Design Constraints & Options	
▸ RTL Synthesis Options	
▸ Reporting Options	

Resource Usage

	VHDL
SLICE	0
LUT	114
FF	53
DSP	2
BRAM	0
URAM	0
LATCH	0
SRL	0
CLB	0

Final Timing

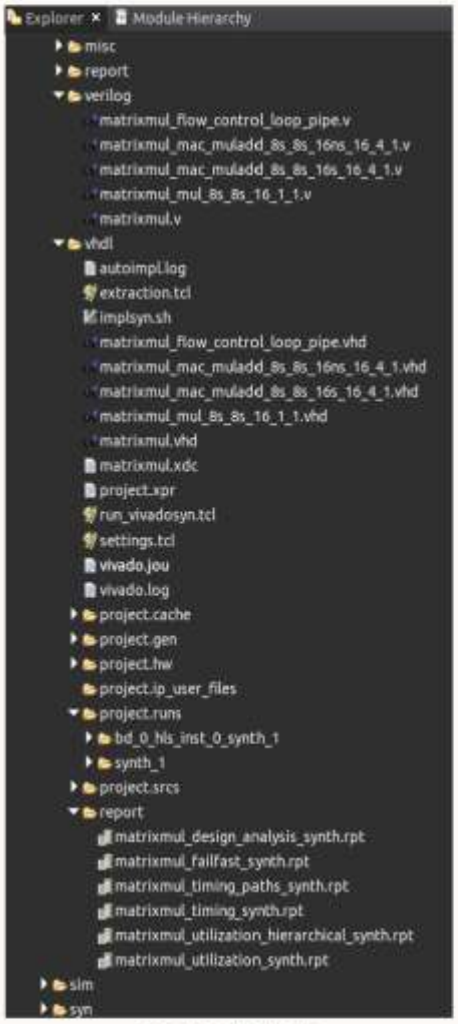
	VHDL
CP required	10.000
CP achieved post-synthesis	8.496

Timing met

Implementation results in Vitis HLS

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

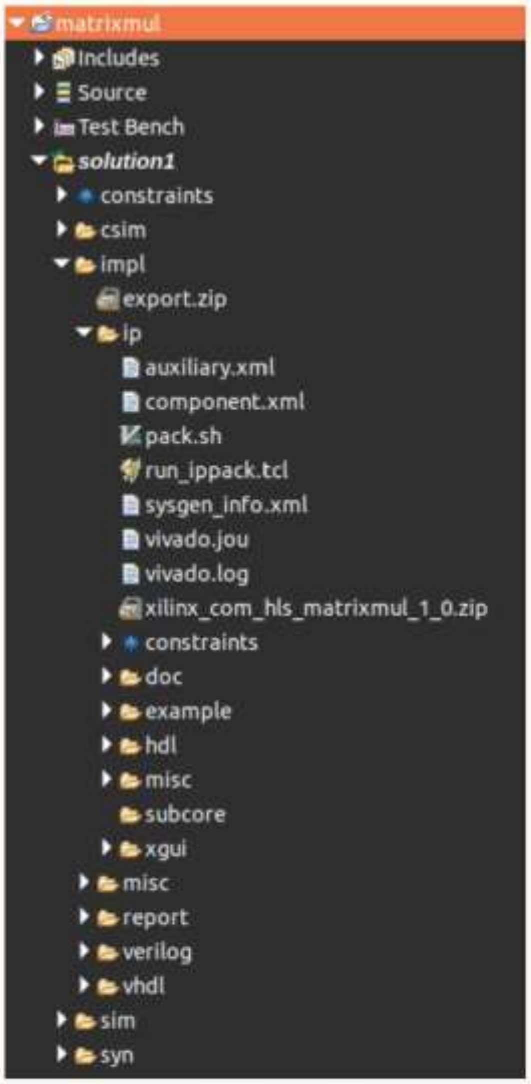
6. Expand the Verilog and vhdl sub-folders and observe that the Verilog sub-folder only has the rtl file whereas the vhdl sub-folder has several files and sub-folders as the synthesis and implementation runs were made for it.
   It includes **project.xpr** file (the Vivado project file), **matrixmul.xdc** 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.



```
misc
report
verilog
  matrixmul_flow_control_loop_pipe.v
  matrixmul_mac_muladd_8s_8s_16ns_16_4_1.v
  matrixmul_mac_muladd_8s_8s_16s_16_4_1.v
  matrixmul_mul_8s_8s_16_1_1.v
  matrixmul.v
vhdl
  autoimpl.log
  extraction.tcl
  implsyn.sh
  matrixmul_flow_control_loop_pipe.vhd
  matrixmul_mac_muladd_8s_8s_16ns_16_4_1.vhd
  matrixmul_mac_muladd_8s_8s_16s_16_4_1.vhd
  matrixmul_mul_8s_8s_16_1_1.vhd
  matrixmul.vhd
  matrixmul.xdc
  project.xpr
  run_vivadosyn.tcl
  settings.tcl
  vivado.jou
  vivado.log
  project.cache
  project.gen
  project.hw
  project.ip_user_files
  project.runs
    bd_0_hls_inst_0_synth_1
    synth_1
  project.srcs
  report
    matrixmul_design_analysis_synth.rpt
    matrixmul_failfast_synth.rpt
    matrixmul_timing_paths_synth.rpt
    matrixmul_timing_synth.rpt
    matrixmul_utilization_hierarchical_synth.rpt
    matrixmul_utilization_synth.rpt
sim
syn
```

The implementation directory

7. Expand the **ip** folder and observe the IP packaged as a zip file (**xilinx_com_hls_matrixmul_1_0.zip**), ready for adding to the Vivado IP catalog.



```
▾ matrixmul
  ▸ Includes
  ▸ Source
  ▸ Test Bench
  ▾ solution1
    ▸ constraints
    ▸ csim
    ▾ impl
        export.zip
      ▾ ip
          auxiliary.xml
          component.xml
          pack.sh
          run_ippack.tcl
          sysgen_info.xml
          vivado.jou
          vivado.log
          xilinx_com_hls_matrixmul_1_0.zip
        ▸ constraints
        ▸ doc
        ▸ example
        ▸ hdl
        ▸ misc
          subcore
        ▸ xgui
      ▸ misc
      ▸ report
      ▸ verilog
      ▸ vhdl
    ▸ sim
    ▸ syn
```

The ip folder content

8. Close Vitis HLS by selecting **File > Exit.**

CONCLUSION

In this lab, you completed the major steps of the high-level synthesis design flow using Vitis 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.

References:

https://xilinx.github.io/xup_high_level_synthesis_design_flow/Lab1.html