Vitis Design Flow Lab

Introduction

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

Objectives

After completing this lab, you will be able to:

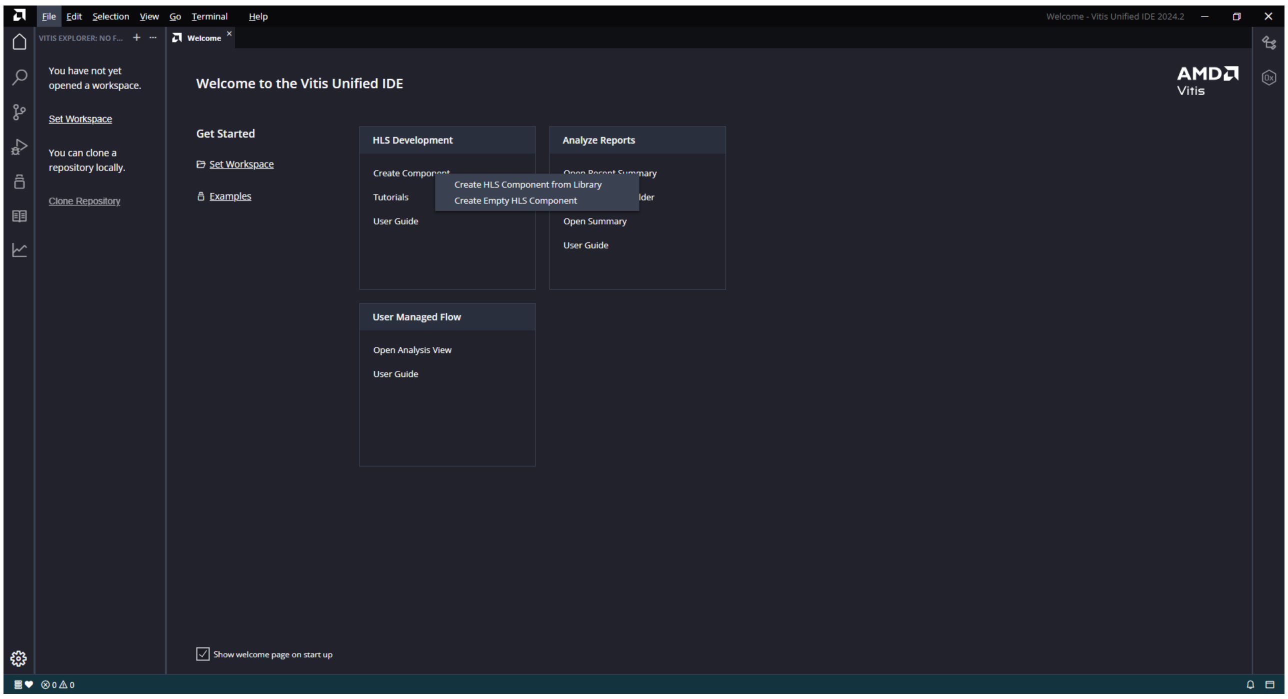
- Create a new project using Vitis
- Simulate a C design by using a self-checking test bench
- Synthesize the design
- Perform design analysis using the Analysis Perspective view
- Perform co-simulation on a generated RTL design by using a provided C test bench
- Implement a design

Steps

Create a New Project

Create a new project in Vitis HLS targeting PYNQ-Z2 board

1. Launch Vitis: Select **Create Component... > Create Empty HLS Component**

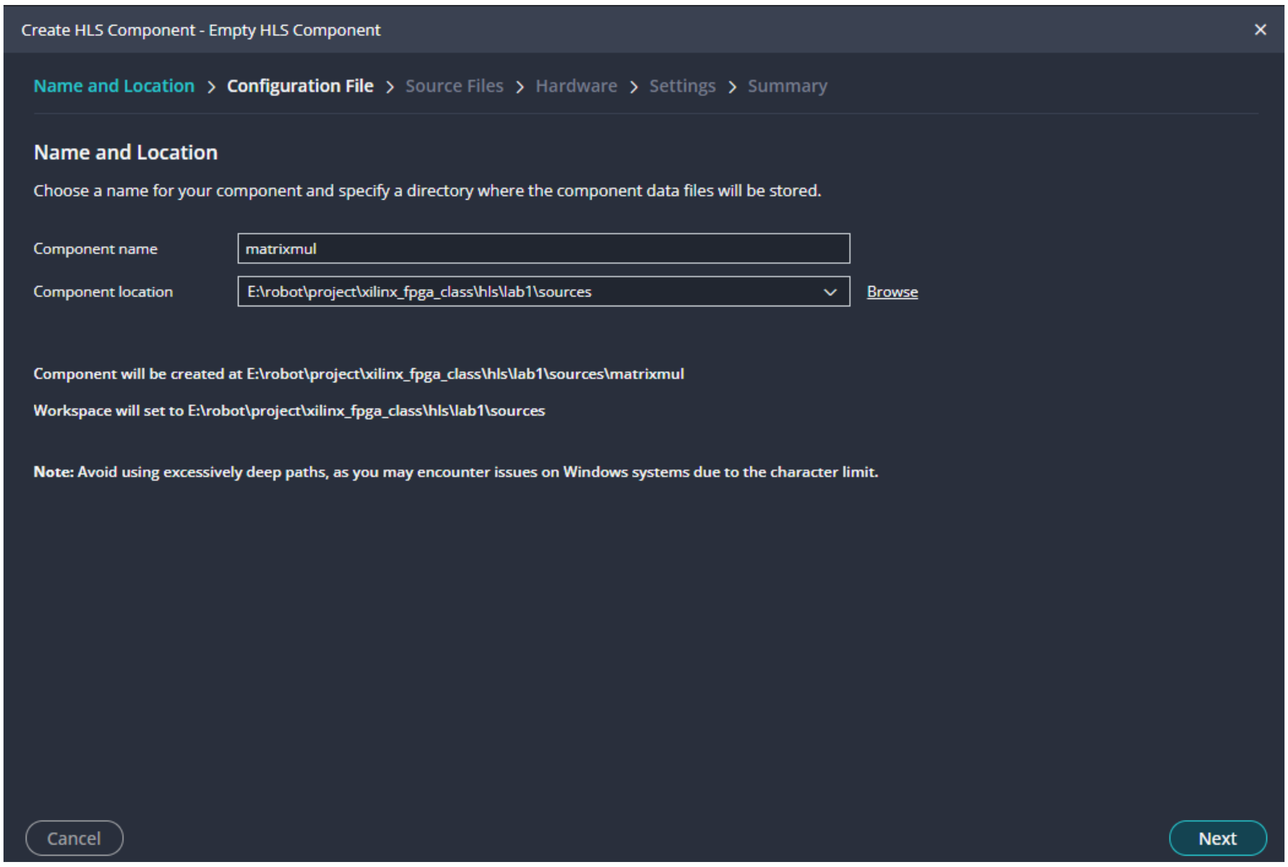


Getting Started view of Vitis

2. Click the *Browse...* button of the Location field and browse to **{labs}\lab1** on a Windows machine or **{labs}/lab1** on a Linux machine creating sub-folders as necessary, and then click **OK**.

Note: From this point onward reference will be made to Linux name.

3. For Project Name, type **matrixmul**.


Create HLS Component - Empty HLS Component

Name and Location > Configuration File > Source Files > Hardware > Settings > Summary

Name and Location

Choose a name for your component and specify a directory where the component data files will be stored.

Component name: matrixmul

Component location: E:\robot\project\xilinx_fpga_class\hls\lab1\sources

Component will be created at E:\robot\project\xilinx_fpga_class\hls\lab1\sources\matrixmul

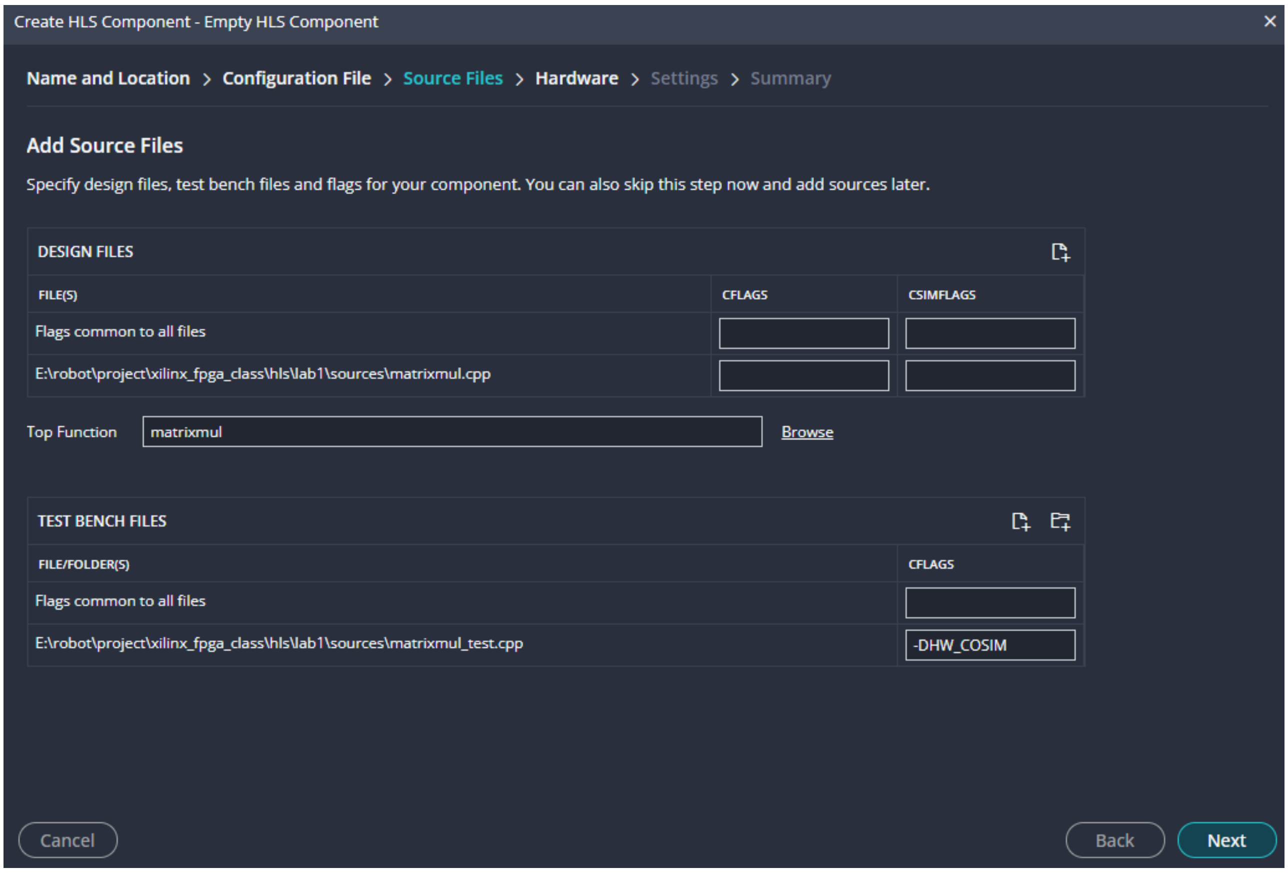
Workspace will set to E:\robot\project\xilinx_fpga_class\hls\lab1\sources

Note: Avoid using excessively deep paths, as you may encounter issues on Windows systems due to the character limit.

New Vitis HLS Project wizard

4. Click **Next**.
5. In the *Configuration File* window, select the *Empty File* option and click **Next**.
6. In the *Source Files* window, type **matrixmul** as the *Top Function* name (the provided source file contains the function, called matrixmul, to be synthesized).
7. Click the *Add Files...* button (which in the line of DESIGN FILES), select **matrixmul.cpp** file from the **{labs}/lab1** folder, and then click **Open**.
8. Then we add the test file, in the next block click *Add Files...* button (which in the line of TEST BENCH FILES), select **matrixmul_test.cpp** file from the **/home/xup/hls/labs/lab1** folder and click **Open**.
9. Select the **matrixmul_test.cpp** in the files list window and click the *Edit CFLAG...* button, type

   -DHW_COSIM (there has a "-" in here, don't forget it), and click **OK**. (This defines a compiler flag that will be used later.)
10. Click **Next**.



Create HLS Component - Empty HLS Component
Name and Location > Configuration File > Source Files > Hardware > Settings > Summary

Add Source Files

Specify design files, test bench files and flags for your component. You can also skip this step now and add sources later.

DESIGN FILES

FILE(S)	CFLAGS	CSIMFLAGS
Flags common to all files		
E:\robot\project\xilinx_fpga_class\hls\lab1\sources\matrixmul.cpp		

Top Function: matrixmul

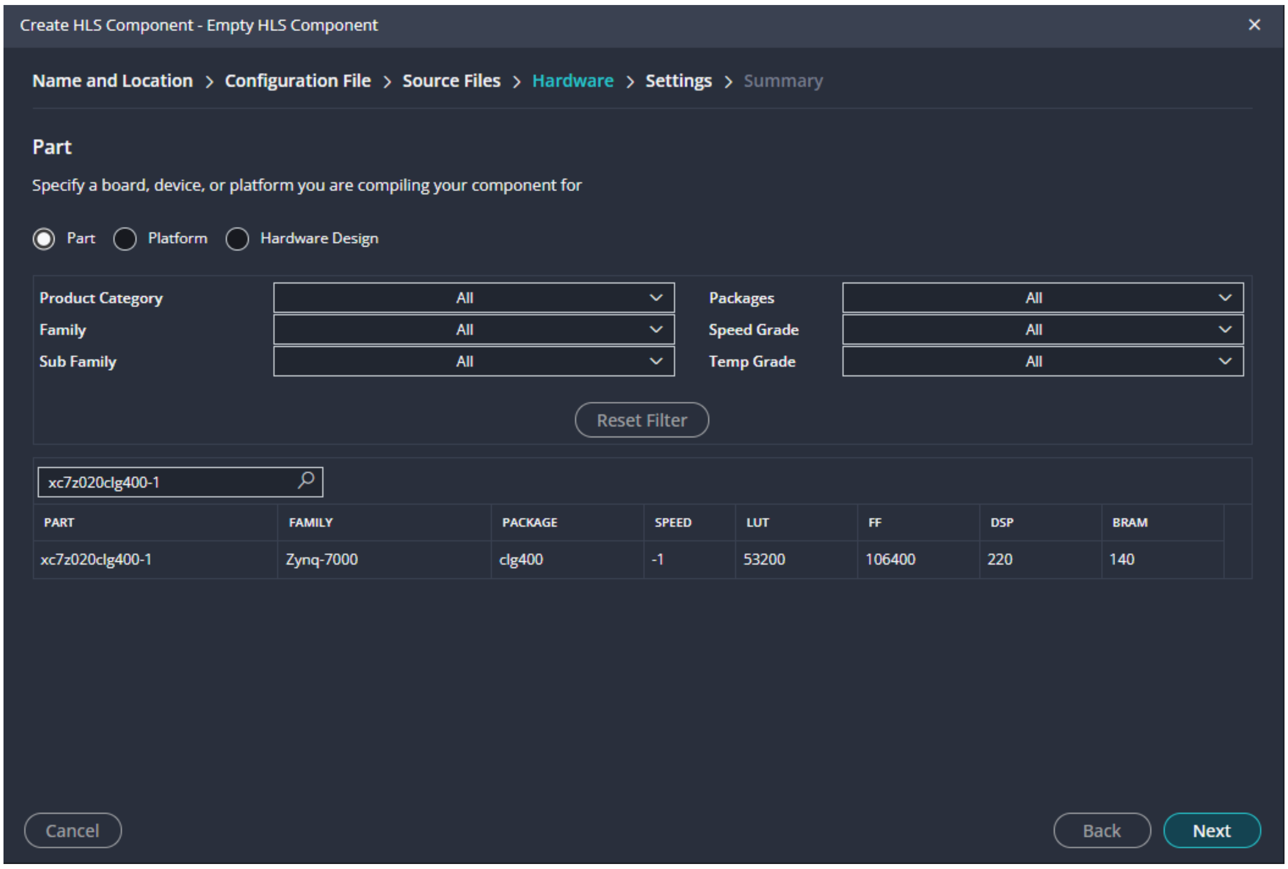
TEST BENCH FILES

FILE/FOLDER(S)	CFLAGS
Flags common to all files	
E:\robot\project\xilinx_fpga_class\hls\lab1\sources\matrixmul_test.cpp	-DHW_COSIM

Source Files setting

11. In the *Hardware* page, select **Part** field, enter **xc7z020clg400-1** in the *Search* field and click **next**.

here you should select the device that you are using, In my class Zedboard using xc7z020clg484-1



Create HLS Component - Empty HLS Component
Name and Location > Configuration File > Source Files > Hardware > Settings > Summary

Part

Specify a board, device, or platform you are compiling your component for

[✓ Part] Platform Hardware Design

Product Category	All	Packages	All
Family	All	Speed Grade	All
Sub Family	All	Temp Grade	All

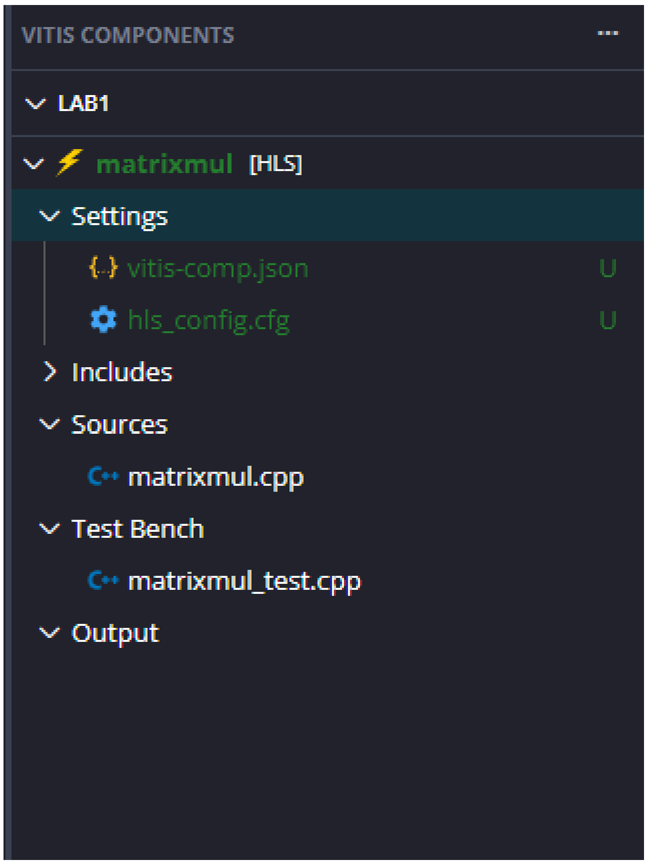
xc7z020clg400-1

PART	FAMILY	PACKAGE	SPEED	LUT	FF	DSP	BRAM
xc7z020clg400-1	Zynq-7000	clg400	-1	53200	106400	220	140

Using Search to selcet the chip

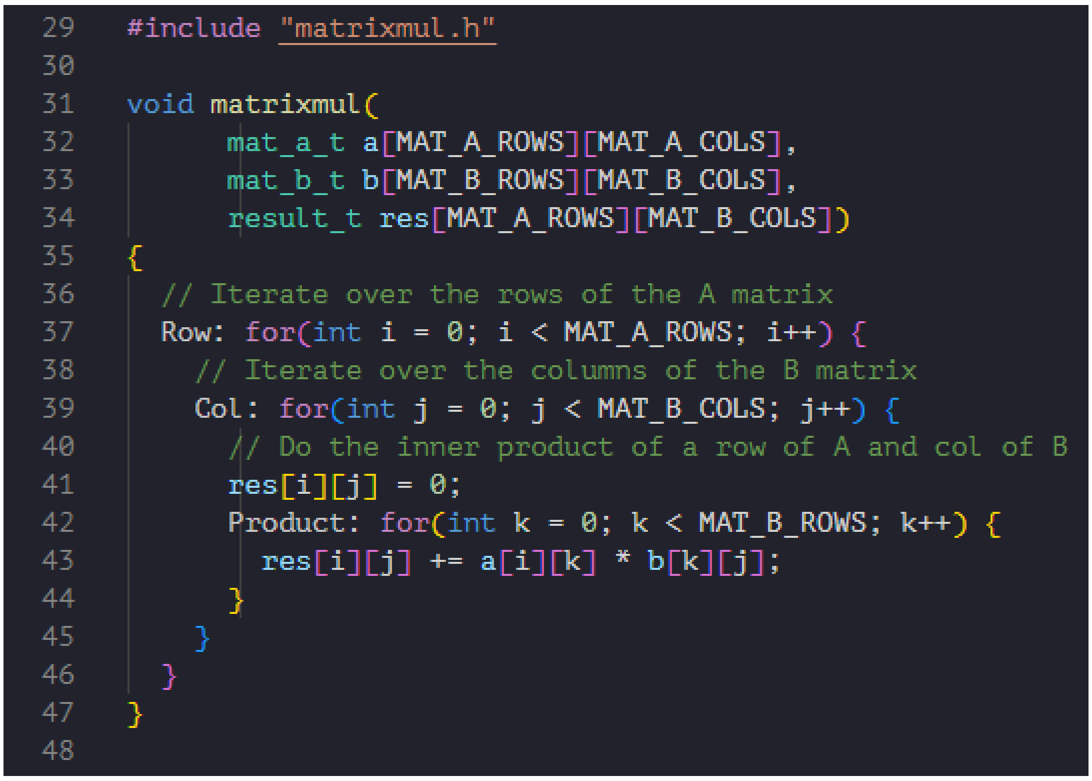
12. In the *Settings* page, tpye 10ns in the clock.Click **next**.

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



Explorer Window

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



```
29  #include "matrixmul.h"
30
31  void matrixmul(
32        mat_a_t a[MAT_A_ROWS][MAT_A_COLS],
33        mat_b_t b[MAT_B_ROWS][MAT_B_COLS],
34        result_t res[MAT_A_ROWS][MAT_B_COLS])
35  {
36    // Iterate over the rows of the A matrix
37    Row: for(int i = 0; i < MAT_A_ROWS; i++) {
38      // Iterate over the columns of the B matrix
39      Col: for(int j = 0; j < MAT_B_COLS; j++) {
40        // Do the inner product of a row of A and col of B
41        res[i][j] = 0;
42        Product: for(int k = 0; k < MAT_B_ROWS; k++) {
43          res[i][j] += a[i][k] * b[k][j];
44        }
45      }
46    }
47  }
48
```

The Design under consideration

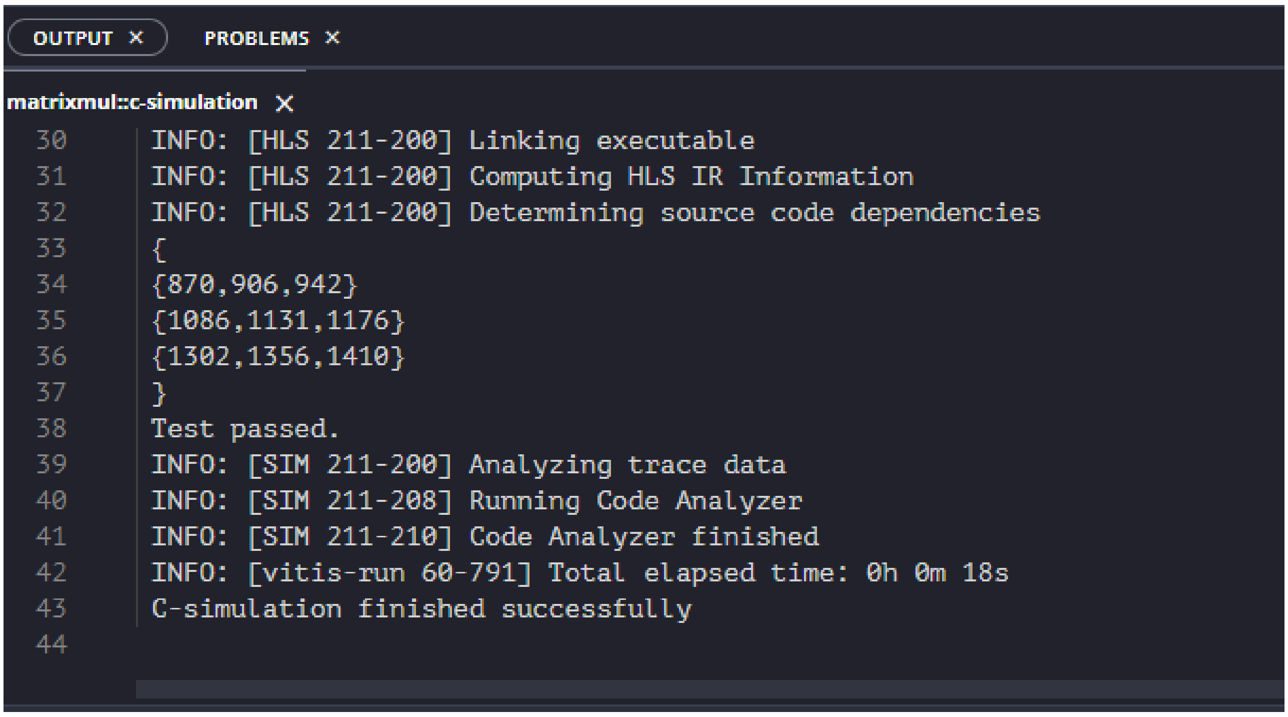
It can be seen that the design is a matrix multiplication implementation, consisting of three nested loops. The Product loop is the inner most loop performing the actual Matrix elements product and sum. The Col loop is the outer-loop which feeds the next column element data with the passed row element data to the Product loop. Finally, Row is the outer-most loop. The res[i][j]=0 (line 41) resets the result every time a new row element is passed and new column element is used.

Run C Simulation

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

   it may be have a windows ask if Enable Code Analyzer, click **Yes, enable Code Analyzer**

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



```
INFO: [HLS 211-200] Linking executable
INFO: [HLS 211-200] Computing HLS IR Information
INFO: [HLS 211-200] Determining source code dependencies
{
{870,906,942}
{1086,1131,1176}
{1302,1356,1410}
}
Test passed.
INFO: [SIM 211-200] Analyzing trace data
INFO: [SIM 211-208] Running Code Analyzer
INFO: [SIM 211-210] Code Analyzer finished
INFO: [vitis-run 60-791] Total elapsed time: 0h 0m 18s
C-simulation finished successfully
```

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.

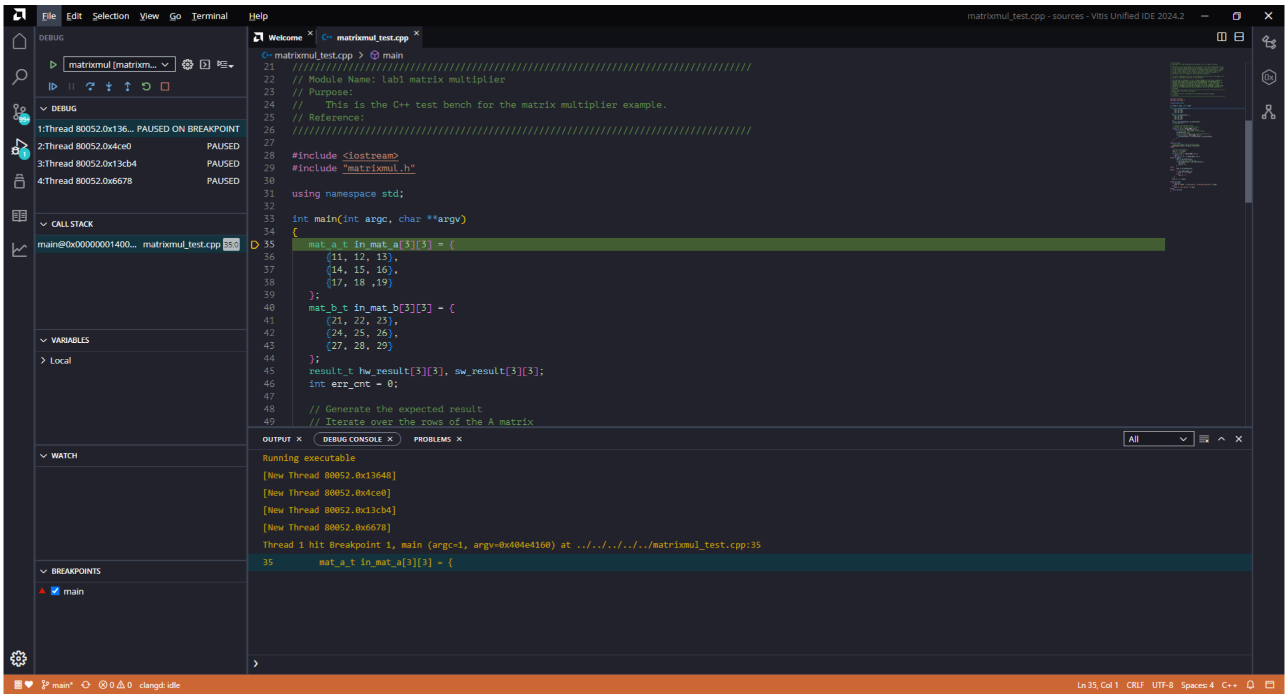
Run Debugger

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

1. Click **Debug** below the **Run** .

   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 > Local**, thread created and the program suspended at the main() function entry point in the *Debug* view.



```
DEBUG
1:Thread 80052.0x136... PAUSED ON BREAKPOINT
2:Thread 80052.0x4ce0          PAUSED
3:Thread 80052.0x13cb4         PAUSED
4:Thread 80052.0x6678          PAUSED
CALL STACK
main@0x00000001400... matrixmul_test.cpp 35:0
BREAKPOINTS
main
```

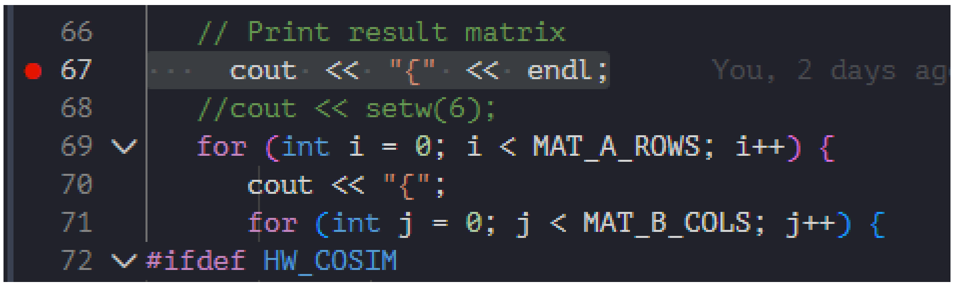
```cpp
////////////////////////////////////////////////////////////////////////////////
// Module Name: lab1 matrix multiplier
// Purpose:
//    This is the C++ test bench for the matrix multiplier example.
// Reference:
////////////////////////////////////////////////////////////////////////////////

#include <iostream>
#include "matrixmul.h"

using namespace std;

int main(int argc, char **argv)
{
   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
```

```
Running executable
[New Thread 80052.0x13648]
[New Thread 80052.0x4ce0]
[New Thread 80052.0x13cb4]
[New Thread 80052.0x6678]
Thread 1 hit Breakpoint 1, main (argc=1, argv=0x404e4160) at ../../../../../matrixmul_test.cpp:35
35         mat_a_t in_mat_a[3][3] = {
```

A Debug perspective

3. Scroll-down in the *source* view, and click in the left size of the line 67 where it is about to output **"{"** in the output console window. This will set a break-point at line 67.

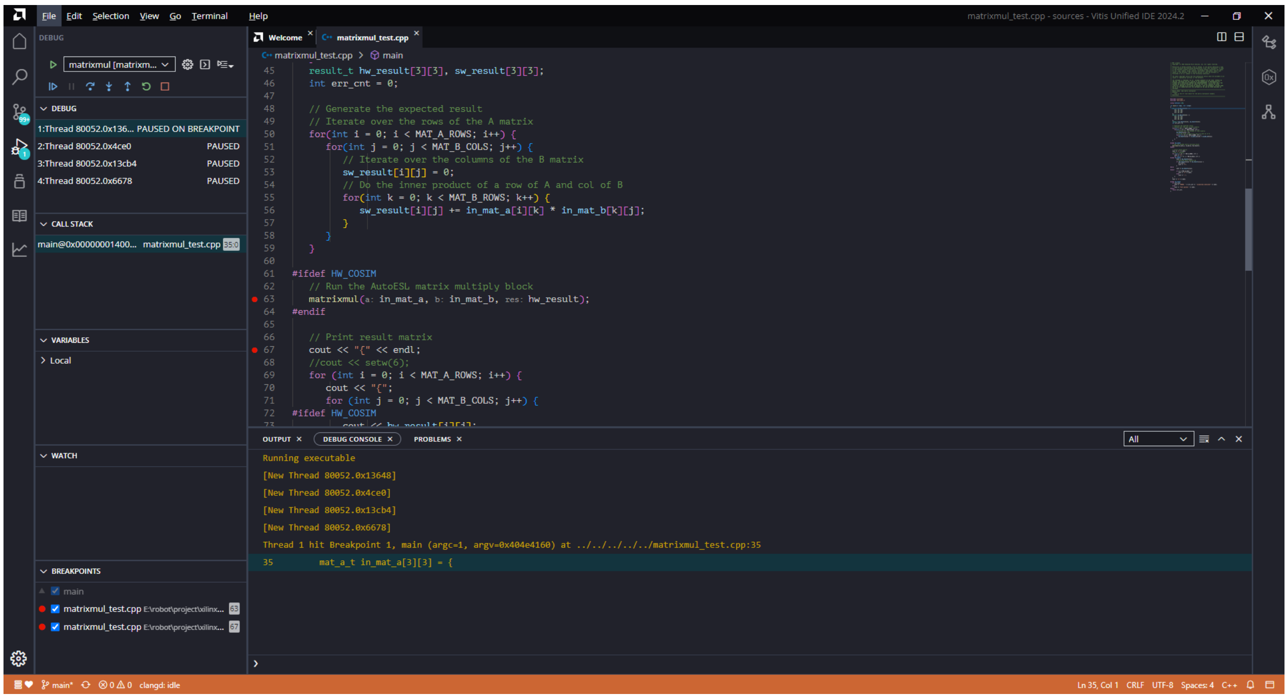
   The breakpoint is marked with a red circle.



```cpp
66     // Print result matrix
67     cout << "{" << endl;        You, 2 days ag
68     //cout << setw(6);
69     for (int i = 0; i < MAT_A_ROWS; i++) {
70        cout << "{";
71        for (int j = 0; j < MAT_B_COLS; j++) {
72  #ifdef HW_COSIM
```

4. Similarly, set a breakpoint at line 63 in the matrixmul() function.

5. Using the **Step Over** (F10) button several times, observe the execution progress, and observe the variable values updating, as well as computed software result.



```
45      result_t hw_result[3][3], sw_result[3][3];
46      int err_cnt = 0;
47
48      // Generate the expected result
49      // Iterate over the rows of the A matrix
50      for(int i = 0; i < MAT_A_ROWS; i++) {
51         for(int j = 0; j < MAT_B_COLS; j++) {
52            // Iterate over the columns of the B matrix
53            sw_result[i][j] = 0;
54            // Do the inner product of a row of A and col of B
55            for(int k = 0; k < MAT_B_ROWS; k++) {
56               sw_result[i][j] += in_mat_a[i][k] * in_mat_b[k][j];
57            }
58         }
59      }
60
61   #ifdef HW_COSIM
62      // Run the AutoESL matrix multiply block
63      matrixmul(a: in_mat_a, b: in_mat_b, res: hw_result);
64   #endif
65
66      // Print result matrix
67      cout << "{" << endl;
68      //cout << setw(6);
69      for (int i = 0; i < MAT_A_ROWS; i++) {
70         cout << "{";
71         for (int j = 0; j < MAT_B_COLS; j++) {
72   #ifdef HW_COSIM
```

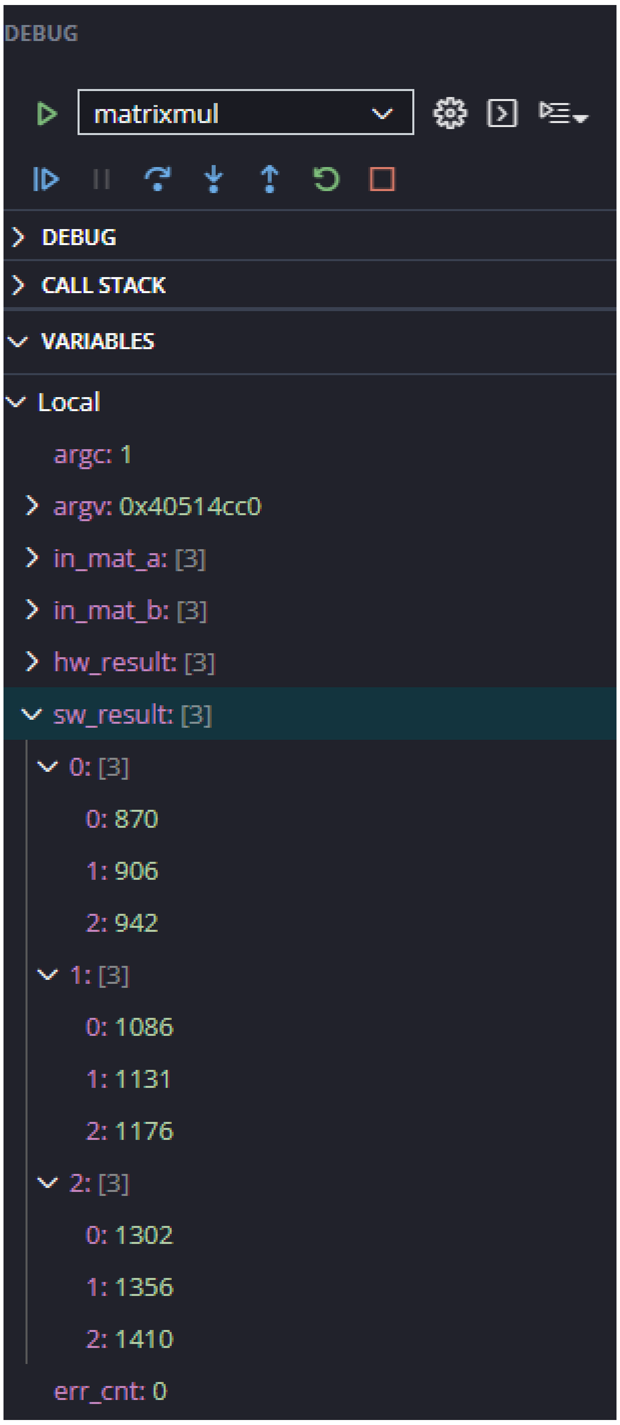
```
Running executable
[New Thread 80052.0x13648]
[New Thread 80052.0x4ce0]
[New Thread 80052.0x13cb4]
[New Thread 80052.0x6678]
Thread 1 hit Breakpoint 1, main (argc=1, argv=0x404e4160) at ../../../../../matrixmul_test.cpp:35
35        mat_a_t in_mat_a[3][3] = {
```

DEBUG
matrixmul [matrixm...
1:Thread 80052.0x136... PAUSED ON BREAKPOINT
2:Thread 80052.0x4ce0 PAUSED
3:Thread 80052.0x13cb4 PAUSED
4:Thread 80052.0x6678 PAUSED
CALL STACK
main@0x00000001400... matrixmul_test.cpp 35:0
VARIABLES
Local
WATCH
BREAKPOINTS
main
matrixmul_test.cpp E:\robot\project\xilinx... 63
matrixmul_test.cpp E:\robot\project\xilinx... 67
OUTPUT
DEBUG CONSOLE
PROBLEMS

Debugger's intermediate output view

6. Now click the **Restart** button or *ctrl+shift+F5* and then click the **Continue** or F5 to complete the software computation and stop at line 63.

7. Observe the following computed software result in the variables view.



```
Local
  argc: 1
> argv: 0x40514cc0
> in_mat_a: [3]
> in_mat_b: [3]
> hw_result: [3]
v sw_result: [3]
  v 0: [3]
      0: 870
      1: 906
      2: 942
  v 1: [3]
      0: 1086
      1: 1131
      2: 1176
  v 2: [3]
      0: 1302
      1: 1356
      2: 1410
  err_cnt: 0
```

Software computed result

8. Click on the **Step Into** (F11) button to traverse into the **matrixmul** module, the one that we will synthesize, and observe that the execution is paused on line 37 of the module.

9. Using the **Step Over** (F6) several times, observe the computed results. Once satisfied, you can use the **Restart** and **Continue** back to the line 63.

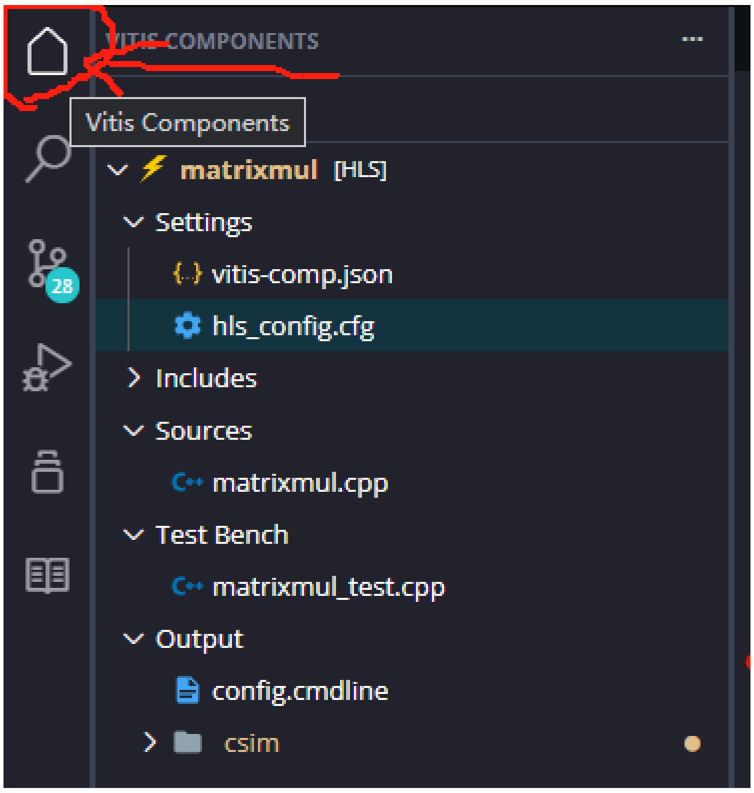
10. Set a breakpoint on line 96 (return err_cnt;), and click on the **Continue** button. The execution will continue until the breakpoint is encountered. The console window will show the results as seen earlier (Figure 11, which is up there).

11. Press the **Continue** button to finish the debugging session.

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.

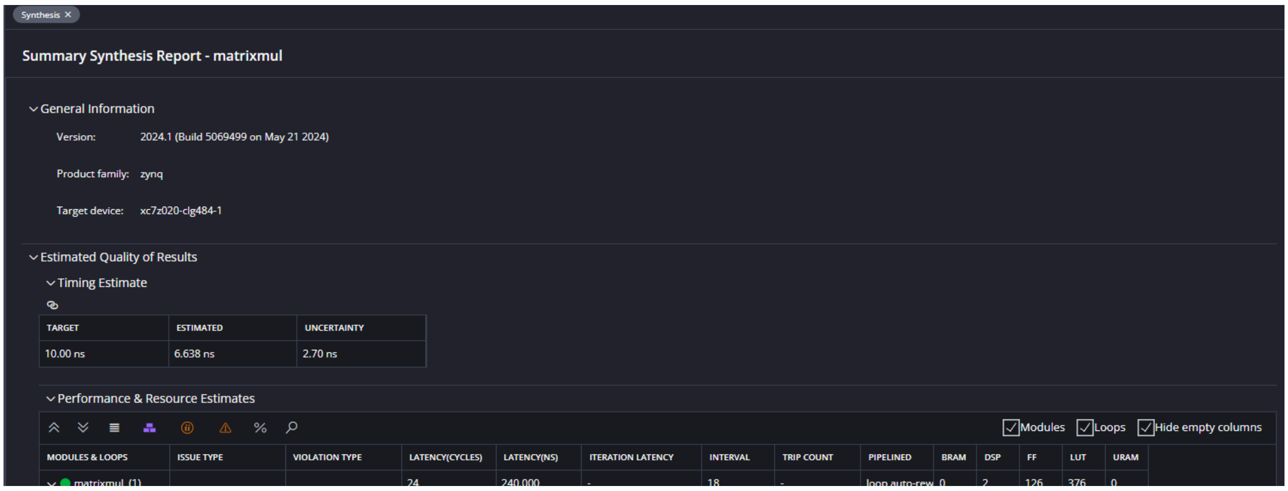
1. Switch to the *Synthesis* view by clicking **Vitis Components** button.



The botton

2. Select **Flow** > **C SYNTHSIS** > **Run** to start the synthesis process.

3. When the synthesis process is completed, Select **C SYNTHESIS** > **REPORTS** > **Synthesis** to oepn the synthesis page.



Summary Synthesis Report - matrixmul

General Information

Version: 2024.1 (Build 5069499 on May 21 2024)

Product family: zynq

Target device: xc7z020-clg484-1

Estimated Quality of Results

Timing Estimate

TARGET	ESTIMATED	UNCERTAINTY
10.00 ns	6.638 ns	2.70 ns

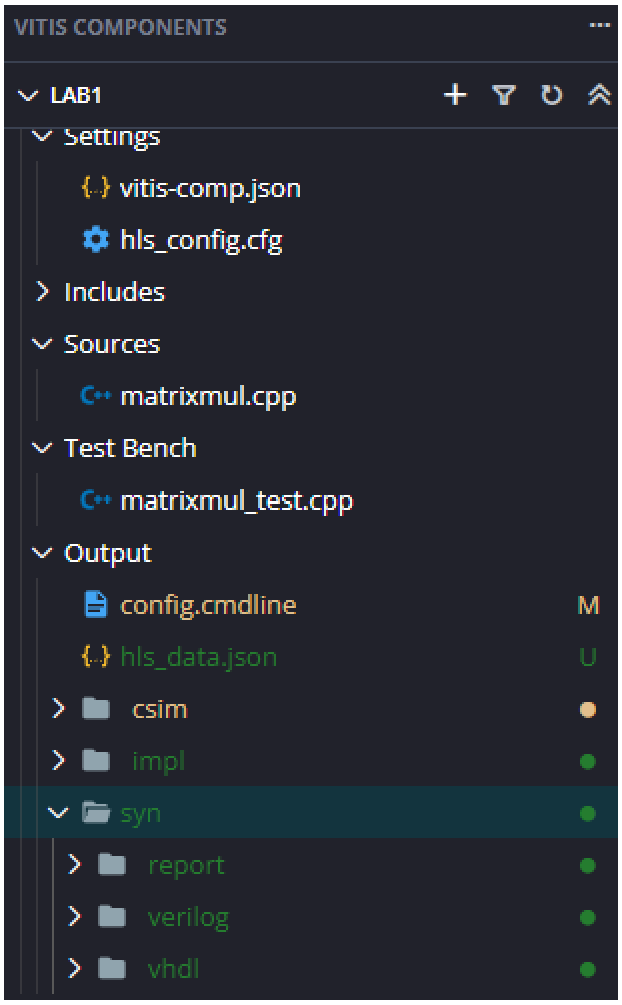
Performance & Resource Estimates

☑ Modules ☑ Loops ☑ Hide empty columns

MODULES & LOOPS	ISSUE TYPE	VIOLATION TYPE	LATENCY(CYCLES)	LATENCY(NS)	ITERATION LATENCY	INTERVAL	TRIP COUNT	PIPELINED	BRAM	DSP	FF	LUT	URAM
● matrixmul (1)			24	240.000	-	18	-	loop auto-rew	0	2	126	376	0

Report view after synthesis is completed

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



Explorer view after the synthesis process

Note that when the **syn** folder under the *Output* 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. By double-clicking any of these entries one can open the corresponding file in the information pane.

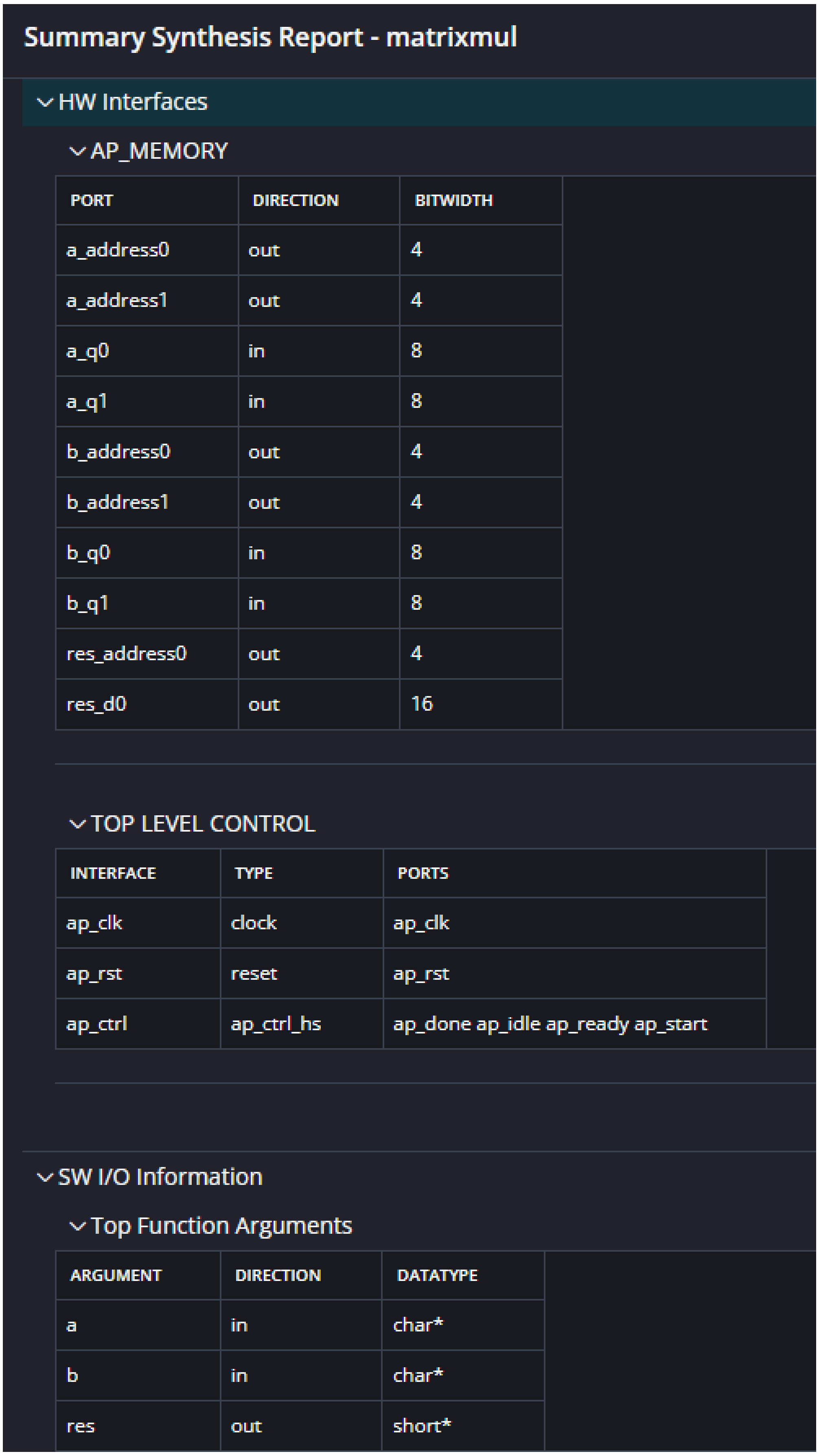
Also note that if the target design has hierarchical functions, reports corresponding to lower-level functions are also created.

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.This can be see in the Synthesis page down below.



Summary Synthesis Report - matrixmul

HW Interfaces

AP_MEMORY

PORT	DIRECTION	BITWIDTH
a_address0	out	4
a_address1	out	4
a_q0	in	8
a_q1	in	8
b_address0	out	4
b_address1	out	4
b_q0	in	8
b_q1	in	8
res_address0	out	4
res_d0	out	16

TOP LEVEL CONTROL

INTERFACE	TYPE	PORTS
ap_clk	clock	ap_clk
ap_rst	reset	ap_rst
ap_ctrl	ap_ctrl_hs	ap_done ap_idle ap_ready ap_start

SW I/O Information

Top Function Arguments

ARGUMENT	DIRECTION	DATATYPE
a	in	char*
b	in	char*
res	out	short*

Generated interface signals

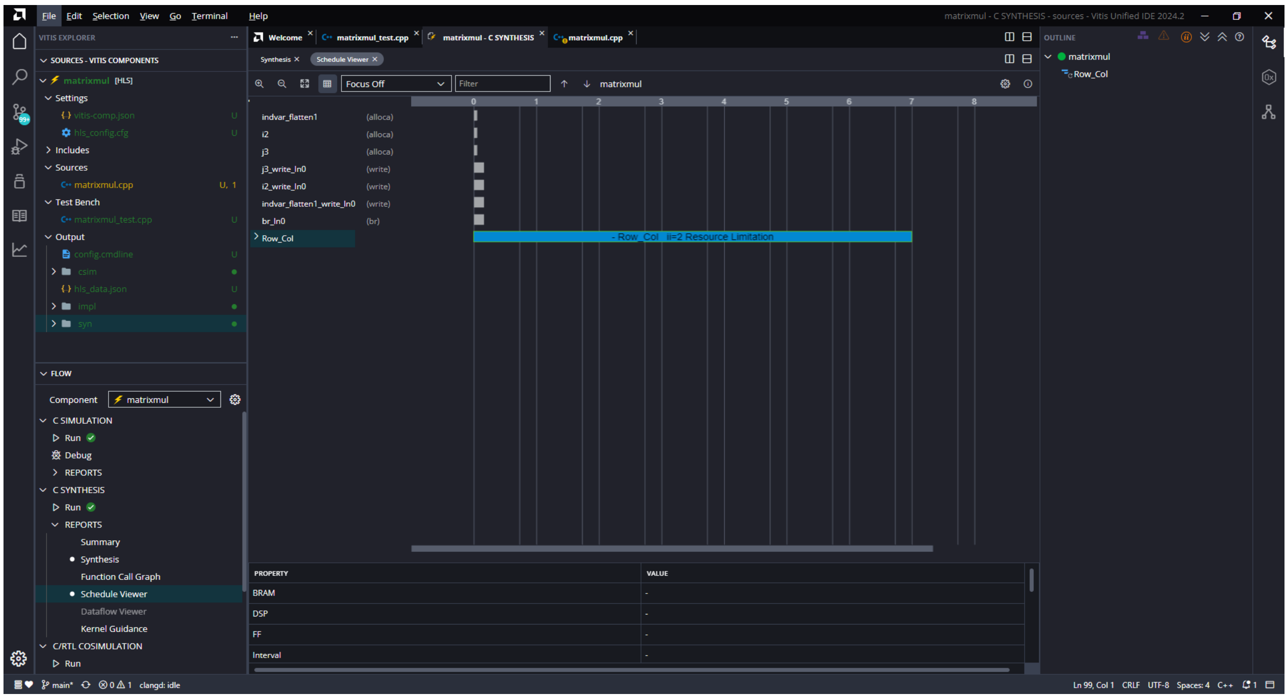
You can see **ap_clk, ap_rst, ap_ idle** and **ap_ready** control signals are automatically added to the design by default. These signals are used as handshaking signals to indicate when the design is ready to take the next computation command (ap_ready), when the next computation is started (ap_start), 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.

Analyze using Analysis Perspective

Switch to the Analysis Perspective and understand the design behavior.

1. Select **C SYNTHESIS > REPORTS > Schedule Viewer** 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 scheduled into clock cycles.

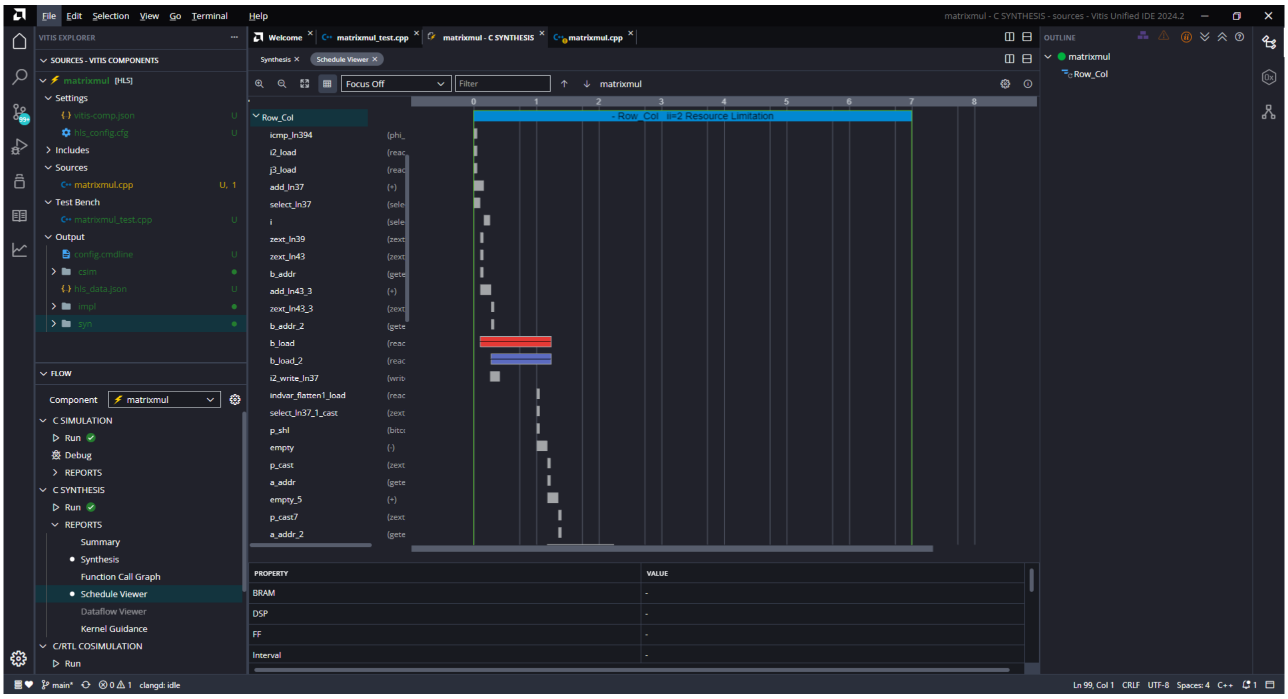


indvar_flatten1	(alloca)
i2	(alloca)
j3	(alloca)
j3_write_ln0	(write)
i2_write_ln0	(write)
indvar_flatten1_write_ln0	(write)
br_ln0	(br)
Row_Col	

PROPERTY	VALUE
BRAM	-
DSP	-
FF	-
Interval	-

Analysis perspective

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

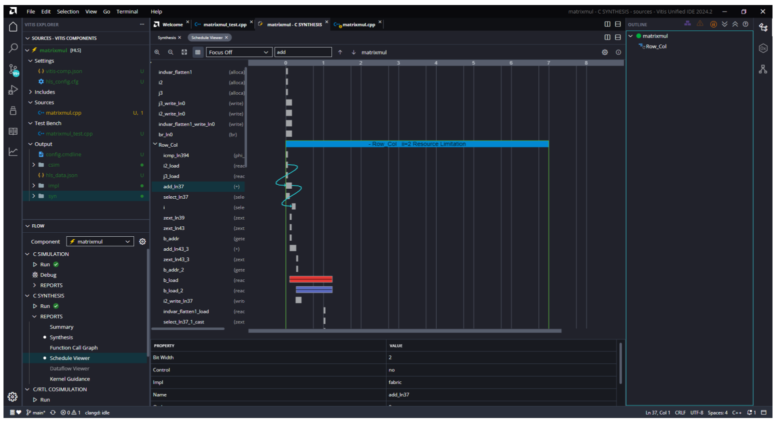


PROPERTY	VALUE
BRAM	-
DSP	-
FF	-
Interval	-

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** (**add_in75_3(+)**), right-click and select **Goto Source**. The source code pane will be opened, highlighting line 37 where the loop index is being tested and incremented.



PROPERTY	VALUE
Bit Width	2
Control	no
Impl	fabric
Name	add_ln37

Cross probing into the source file

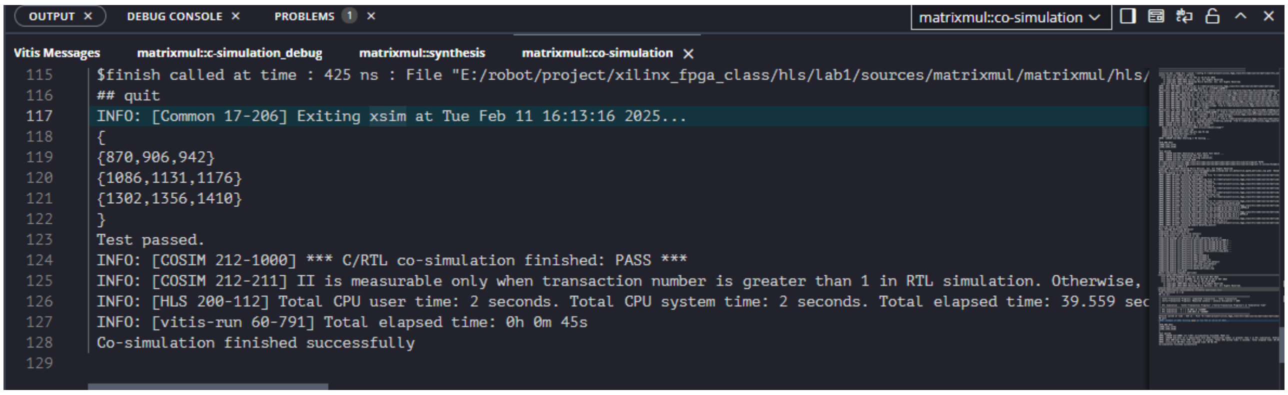
4. Click on the **Schedule Viewer** tool bar button to switch back to the *Synthesis* view.

Run C/RTL Co-simulation

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

1. Select **Flow > C/RTL COSIMULATION > Run**, it will automatically run. Wait for tge COSIMULATION to complete. 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.

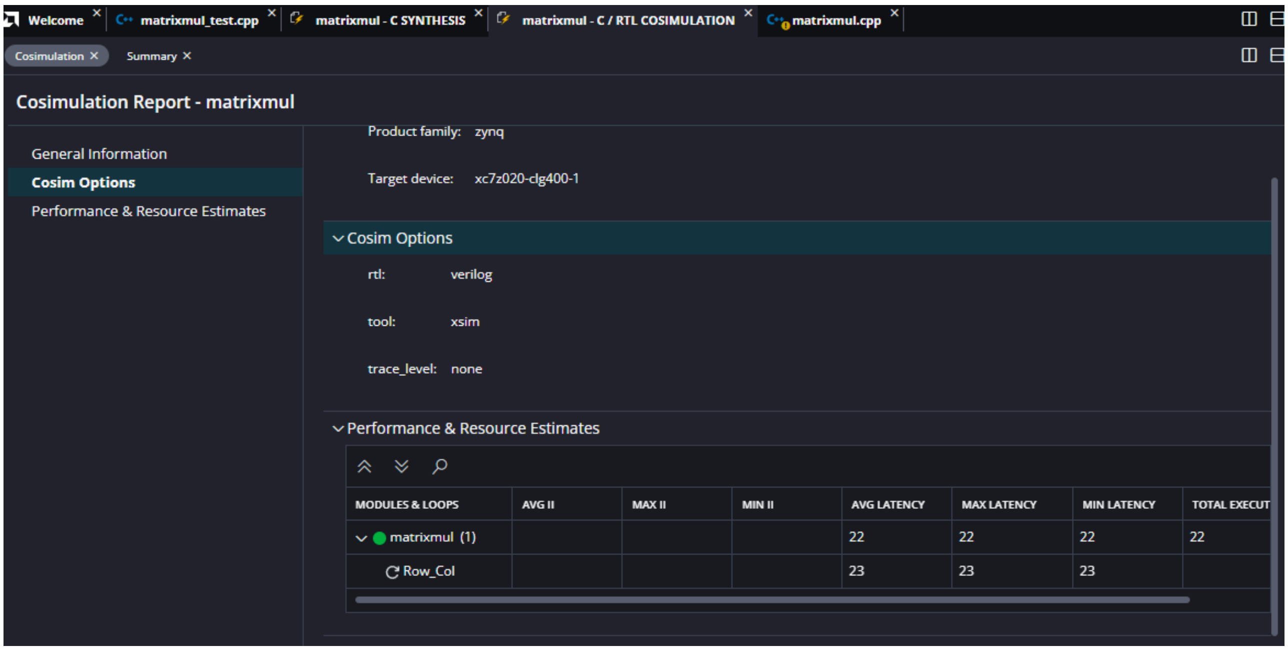


```
$finish called at time : 425 ns : File "E:/robot/project/xilinx_fpga_class/hls/lab1/sources/matrixmul/matrixmul/hls/
## quit
INFO: [Common 17-206] Exiting xsim at Tue Feb 11 16:13:16 2025...
{
{870,906,942}
{1086,1131,1176}
{1302,1356,1410}
}
Test passed.
INFO: [COSIM 212-1000] *** C/RTL co-simulation finished: PASS ***
INFO: [COSIM 212-211] II is measurable only when transaction number is greater than 1 in RTL simulation. Otherwise,
INFO: [HLS 200-112] Total CPU user time: 2 seconds. Total CPU system time: 2 seconds. Total elapsed time: 39.559 sec
INFO: [vitis-run 60-791] Total elapsed time: 0h 0m 45s
Co-simulation finished successfully
```

Console view showing simulation progress

2. Once the simulation verification is completed, the simulation report tab will open showing the results. Click **C/TRL COSIMULATION > REPORTS > Cosimulation** to see report.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 - matrixmul

Product family: zynq

Target device: xc7z020-clg400-1

Cosim Options

rtl: verilog

tool: xsim

trace_level: none

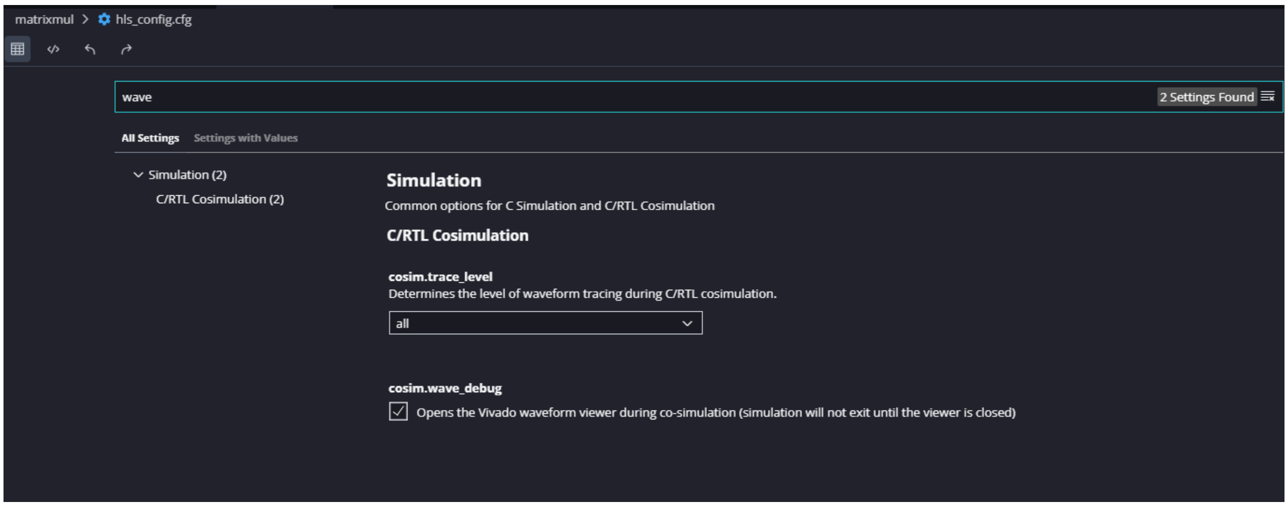
Performance & Resource Estimates

MODULES & LOOPS	AVG II	MAX II	MIN II	AVG LATENCY	MAX LATENCY	MIN LATENCY	TOTAL EXECUT
● matrixmul (1)				22	22	22	22
Row_Col				23	23	23	

Co-simulation results

Analyze the dumped traces.

1. You will see the Wave Viewer can't use with the ⚠, click it and we search the **wave** in the hls_config.cfg.And set the cosim.wave_debug.Then change the cosim.trace+level from None to all.



matrixmul > hls_config.cfg

wave

2 Settings Found

All Settings Settings with Values

Simulation (2)

C/RTL Cosimulation (2)

Simulation

Common options for C Simulation and C/RTL Cosimulation

C/RTL Cosimulation

cosim.trace_level

Determines the level of waveform tracing during C/RTL cosimulation.

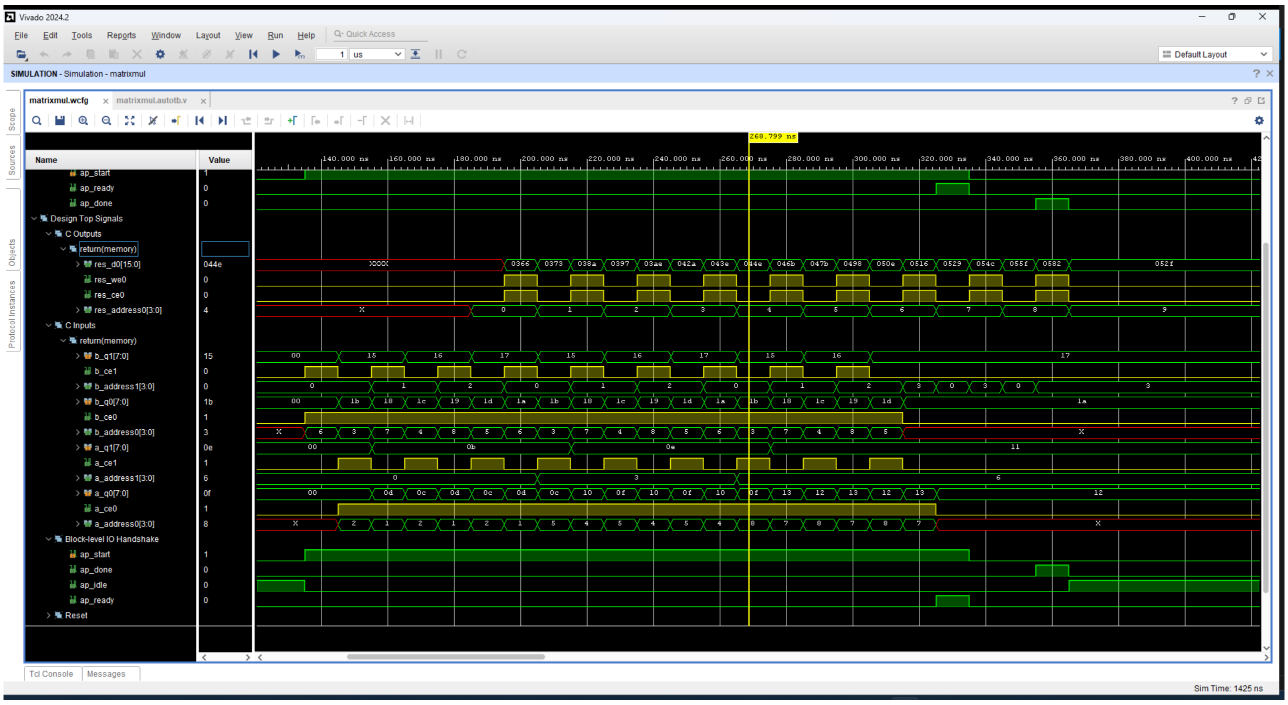
all

cosim.wave_debug

✓ Opens the Vivado waveform viewer during co-simulation (simulation will not exit until the viewer is closed)

Change the setting

2. Next click **C/TRL COSIMULATION > Run** again, the vivado will open.
3. You should click the **Run for 1µs** in the vivado.



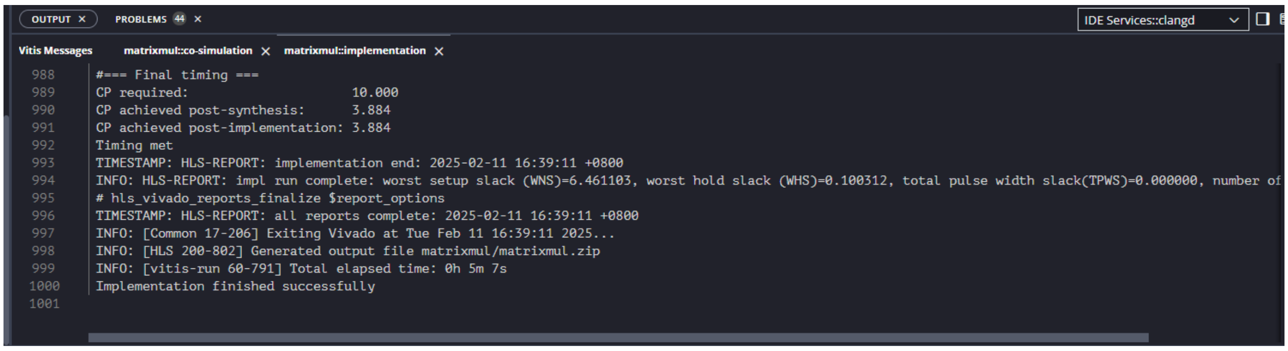
Full waveform showing iteration worth simulation

4. View various part of the simulation and try to understand how the design works.
5. When done, close Vivado, click **ok** if prompted.

Export RTL and Implement

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

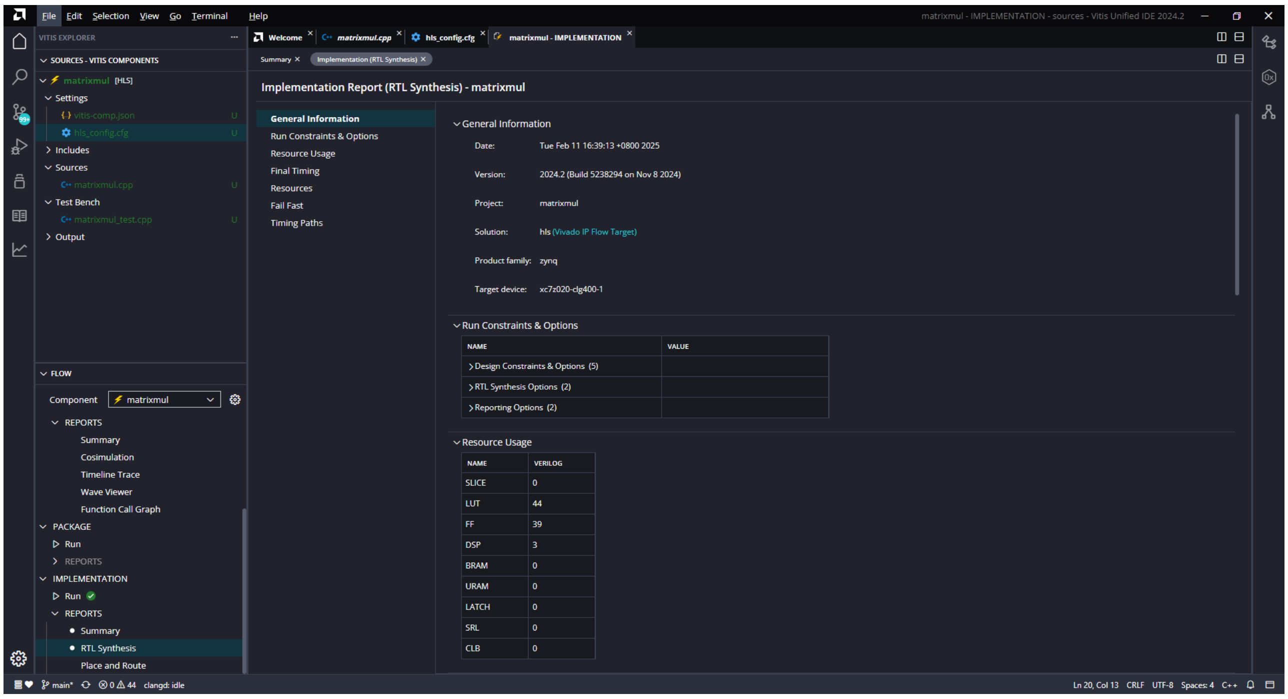
1. In Vitis HLS, select **Flow > IMPLEMENTATION >** wait for implemntation finished. An Export RTL Dialog box will open.



```
#=== Final timing ===
CP required:                    10.000
CP achieved post-synthesis:     3.884
CP achieved post-implementation: 3.884
Timing met
TIMESTAMP: HLS-REPORT: implementation end: 2025-02-11 16:39:11 +0800
INFO: HLS-REPORT: impl run complete: worst setup slack (WNS)=6.461103, worst hold slack (WHS)=0.100312, total pulse width slack(TPWS)=0.000000, number of
# hls_vivado_reports_finalize $report_options
TIMESTAMP: HLS-REPORT: all reports complete: 2025-02-11 16:39:11 +0800
INFO: [Common 17-206] Exiting Vivado at Tue Feb 11 16:39:11 2025...
INFO: [HLS 200-802] Generated output file matrixmul/matrixmul.zip
INFO: [vitis-run 60-791] Total elapsed time: 0h 5m 7s
Implementation finished successfully
```

Run Implemntation Done

2. Click the **Flow > IMPLEMENTATION > REPORTS > RTL Synthesis** to see the report.



Implementation Report (RTL Synthesis) - matrixmul

General Information

Date:	Tue Feb 11 16:39:13 +0800 2025
Version:	2024.2 (Build 5238294 on Nov 8 2024)
Project:	matrixmul
Solution:	hls (Vivado IP Flow Target)
Product family:	zynq
Target device:	xc7z020-clg400-1

Run Constraints & Options

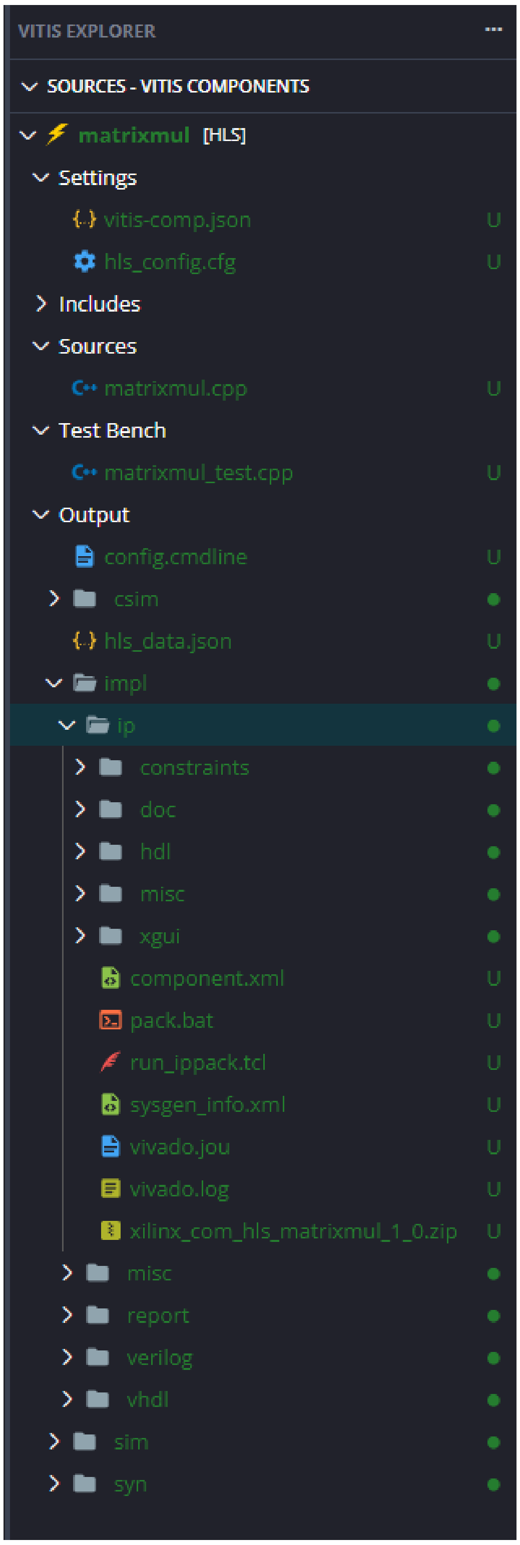
NAME	VALUE
Design Constraints & Options (5)	
RTL Synthesis Options (2)	
Reporting Options (2)	

Resource Usage

NAME	VERILOG
SLICE	0
LUT	44
FF	39
DSP	3
BRAM	0
URAM	0
LATCH	0
SRL	0
CLB	0

RTL Synthesis report

3. You can see the **xilinx_com_hls_matrixmul_1_0.zip** on **impl > ip**.which can be added to the Vivado IP catalog.



```
VITIS EXPLORER
SOURCES - VITIS COMPONENTS
matrixmul [HLS]
  Settings
    vitis-comp.json          U
    hls_config.cfg           U
  Includes
  Sources
    matrixmul.cpp            U
  Test Bench
    matrixmul_test.cpp       U
  Output
    config.cmdline           U
    csim
    hls_data.json            U
    impl
      ip
        constraints
        doc
        hdl
        misc
        xgui
        component.xml        U
        pack.bat             U
        run_ippack.tcl       U
        sysgen_info.xml      U
        vivado.jou           U
        vivado.log           U
        xilinx_com_hls_matrixmul_1_0.zip   U
      misc
      report
      verilog
      vhdl
    sim
    syn
```

The ip folder content

4. Close Vitis by selecting **File > Close Window**.

Conclusion

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

Answers for question 1:
Estimated clock period: **6.816 ns**
Worst case latency: **24 clock cycles**
Number of DSP48E used: **2**
Number of FFs used: **66**
Number of LUTs used: **365**