INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Advanced Computing Architectures – 2023/2024

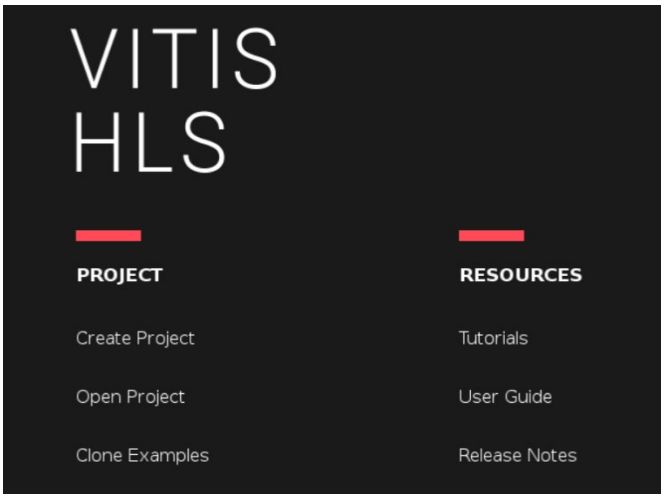
HLS Design Flow

Objectives

- Create a new project using Vitis HLS in GUI mode
- 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

Create a New Project

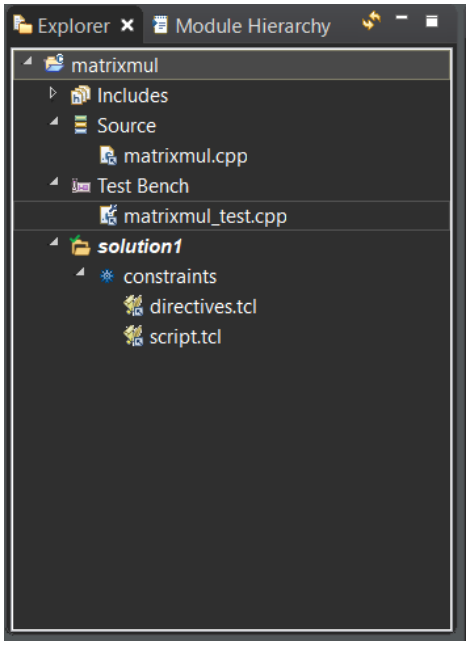
1. Launch Vitis HLS: Select **Start > Xilinx Design Tools > Vitis HLS 2023.2**



Getting Started view of Vitis HLS

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 field and browse to **{labs}\lab3** on a Windows machine, and then click **OK**.
4. For Project Name, type **matrixmul**.
5. Click **Next**.
6. In the *Add/Remove Design 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, select **matrixmul.cpp** file from the **{labs}/lab3** folder, and then click **Open**.
8. Click the *Add Files…* button, select **matrixmul.h** file from the **{labs}/lab3** folder, and then click **Open**.
9. Click **Next**.
10. In the *Add/Remove Testbench Files* for the testbench, click the *Add Files…* button, select **matrixmul_test.cpp** file and click **Open**.
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 of the *Part Selection* section.
14. In the *Board Selection* page, select **zybo** and click **OK**.
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.



Explorer Window

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

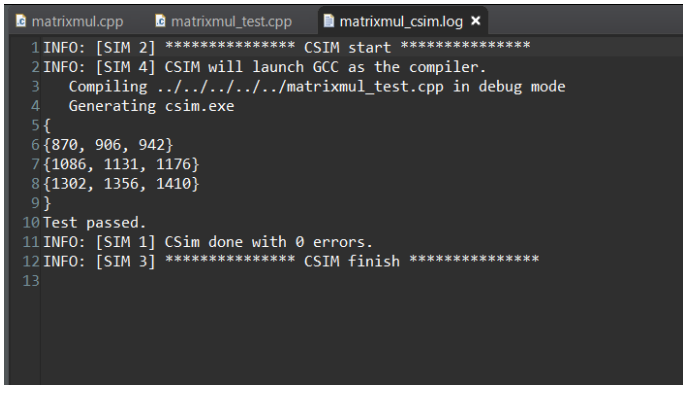
```
57 #include "matrixmul.h"
58
59 void matrixmul(
70       mat_a_t a[MAT_A_ROWS][MAT_A_COLS],
71       mat_b_t b[MAT_B_ROWS][MAT_B_COLS],
72       result_t res[MAT_A_ROWS][MAT_B_COLS])
73 {
74   // Iterate over the rows of the A matrix
75   Row: for(int i = 0; i < MAT_A_ROWS; i++) {
76     // Iterate 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

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



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

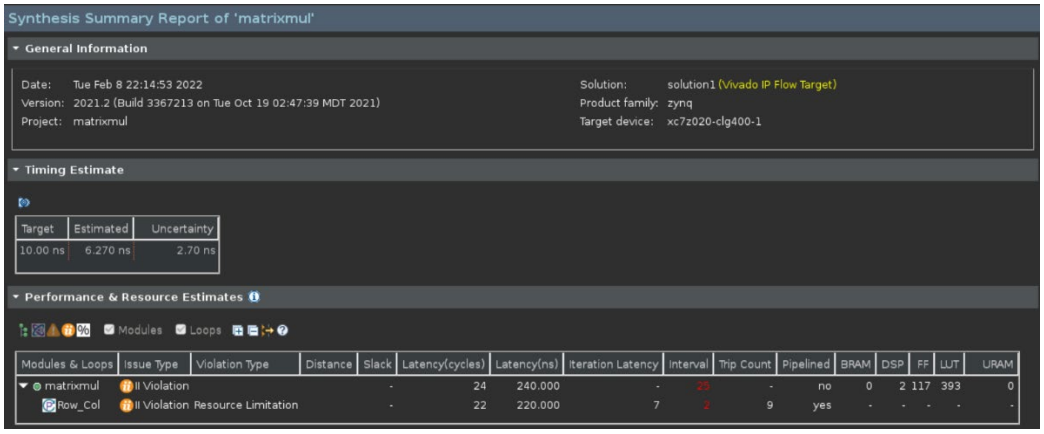
1. Select **Project > Run C Simulation**. Select the **Launch Debugger** option and click **OK**.

   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 the *Debug* view.
3. From here, you can debugged the function.
4. Press the **Resume** button or **Terminate** button to finish the debugging session.

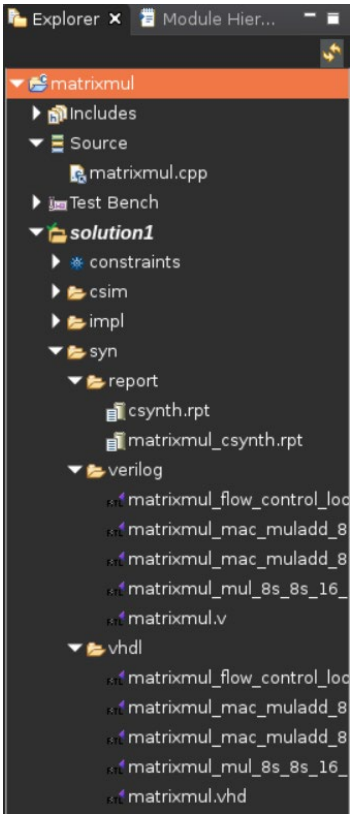
Synthesize the Design

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



Report view after synthesis is completed

4. If you expand **solution1** 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 *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. By double-clicking any of these entries one can open the corresponding file in the information pane.

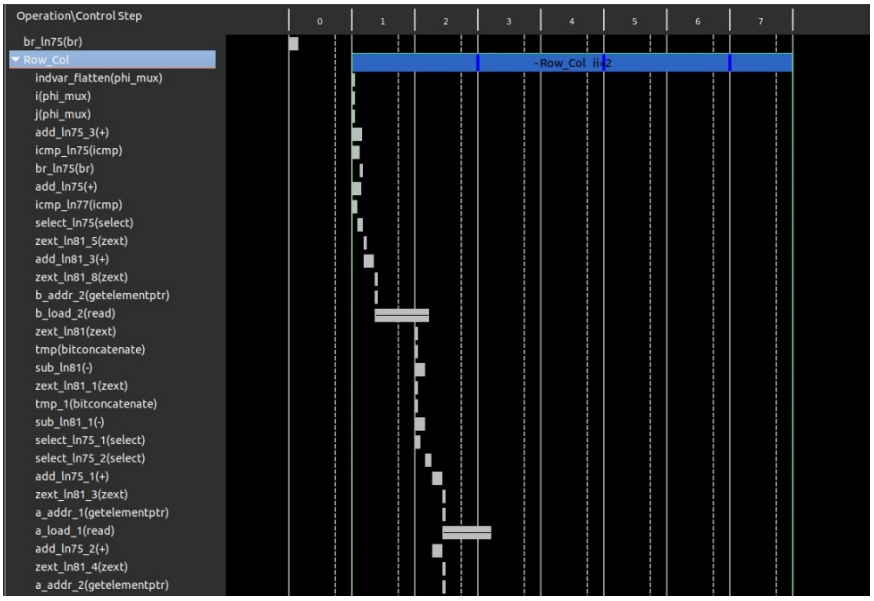
Also note that if the target design has hierarchical functions, reports corresponding to lower-level functions are also created.

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:

Analyze using Analysis Perspective

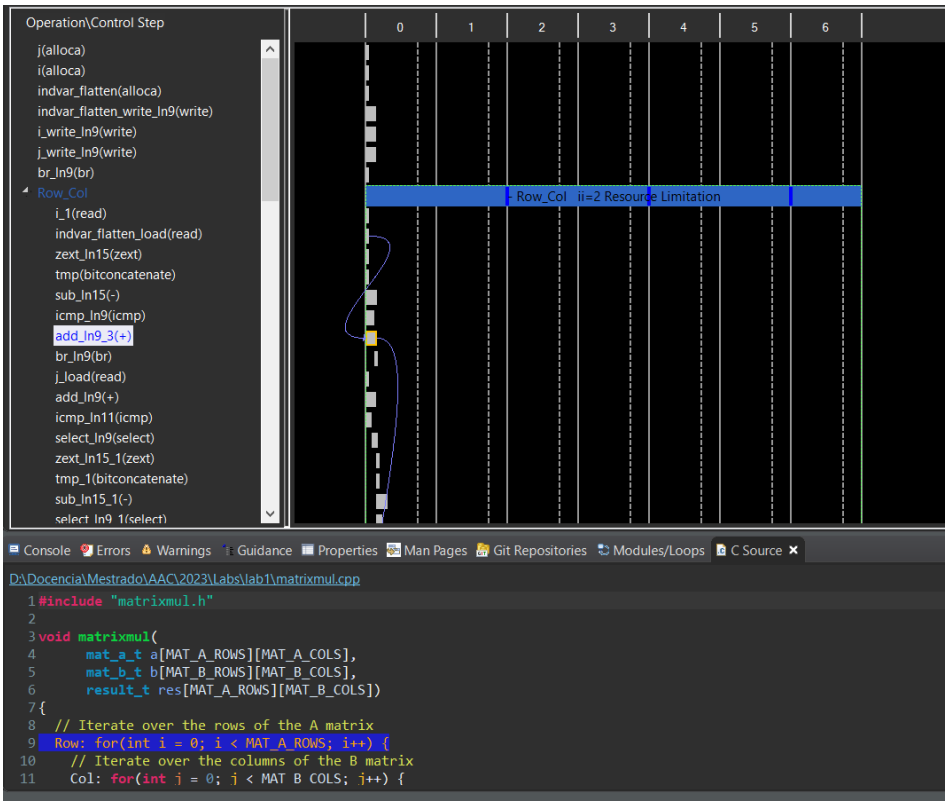
1. Select **Solution > Open Schedule Viewer** or click on *Analysis* button on tools bar to open the analysis viewer.
2. Click on **>** of loop *Row_Col* to expand.



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** (**add_in9_3(+)**), right-click and select **Goto Source**. The source code pane will be opened, highlighting the line where the loop index is being tested and incremented.



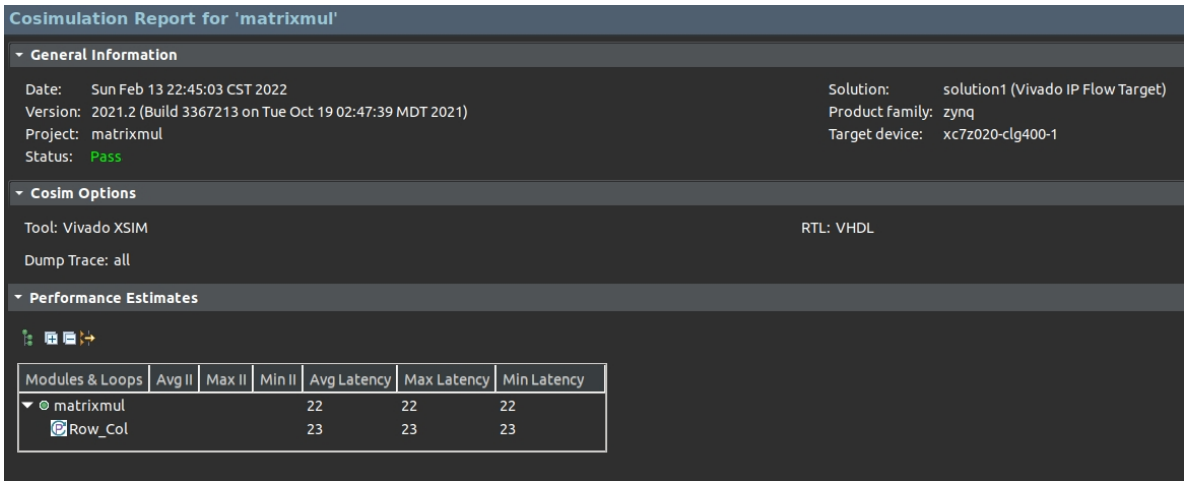
Cross probing into the source file

4. Close the Schedule Viewer and return to the *Synthesis* view.

Run C/RTL Co-simulation

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 generated model. To perform the verification using Verilog, you can select Verilog.
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.
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.



Co-simulation results

Viewing Simulation Results in Vivado

1. Select **Solution > Run C/RTL Co-simulation** 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 - Vivado XSim.

3. Select **All** for the *Dump Trace* option and click **OK**.

   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.

   The Co-simulation report shows the test was passed for Verilog along with latency and Interval results.

Viewing Waveform Results

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 shown below.

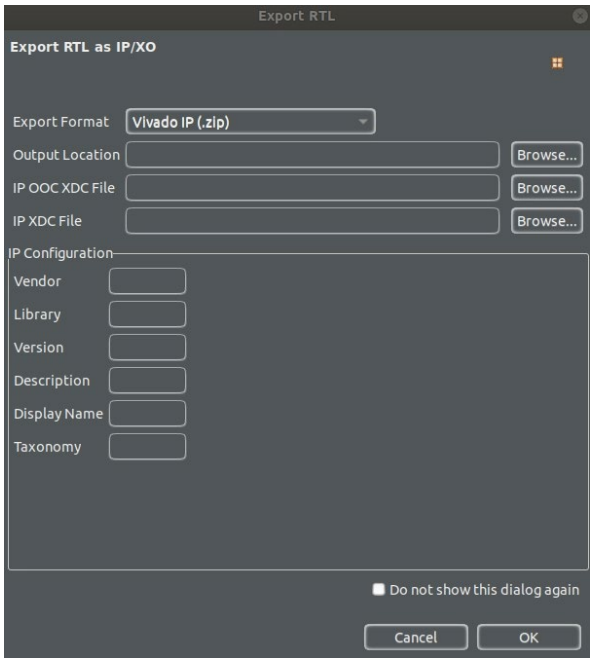
Full waveform showing iteration worth simulation

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. This indicates 24 clock cycles latency.

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.

Export RTL and Implement

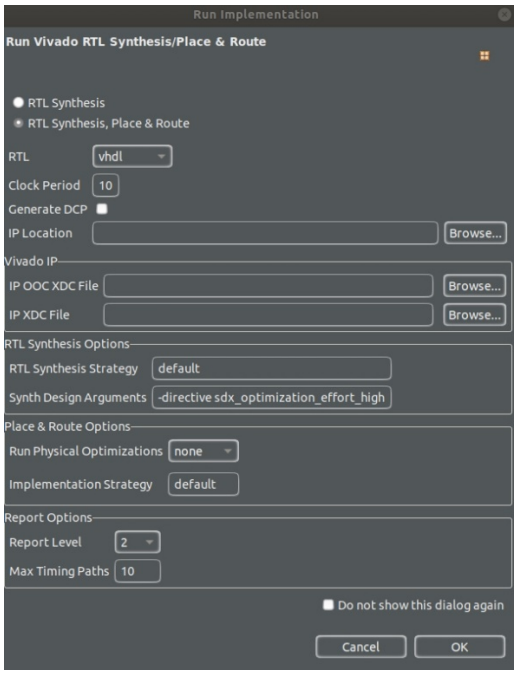
1. In Vitis HLS, select **Solution > Export RTL** to open the dialog box. An Export RTL Dialog box will open.



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. Another option available from the Export Format drop-down menu, is 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.



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, since we selected Evaluate option, it goes through Synthesis and then through Placement and Routing

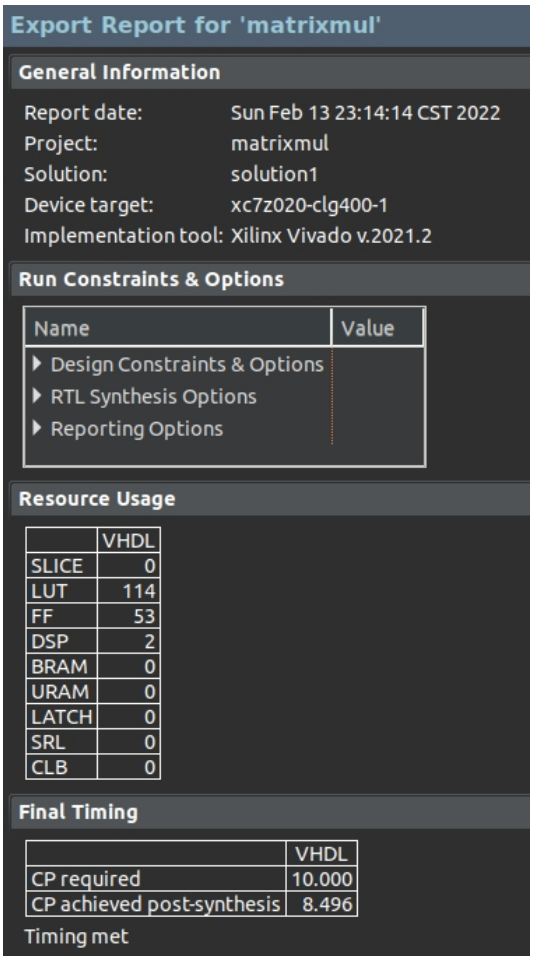
```
Implementation tool: Xilinx Vivado v.2021.2
Project:             matrixmul
Solution:            solution1
Device target:       xc7z020-clg400-1
Report date:         Sun Feb 13 23:14:14 CST 2022

#=== 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
```

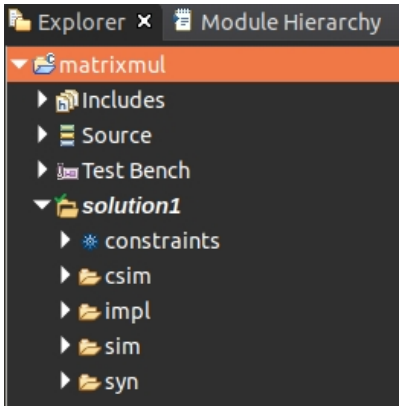
Console view

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



Implementation results in Vitis HLS

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



Explorer view after the RTL Export run