Department of
Systems and Computer Engineering

SYSC-3320: Laboratory 6
Designing with Vivado HLS

Introduction
In previous labs we showed how to build SoC systems using VHDL (for PL part using FPGA technology) and C/C++ (for PS part). Building the PL (or hardware) parts using VHDL is a complex process for large projects. Therefore, to facilitate FPGA development, SoC CAD tools include a design automation capability to accelerate the project development and optimization. In this lab, you will use the Vivado™ HLS (High Level Synthesis) environment to build a relatively complex hardware project that multiplies two matrices. You will not write VHDL code to build such system. Instead, you will use normal C/C++ code to describe the system and the Vivado™ HLS will automatically convert your C/C++ code into VHDL and package this VHDL code into an IP that can be used by other designers in bigger projects. The lab will show you how complex PL (or hardware) sub-systems that performs computationally intense math or DSP operations can be developed. The lab is split into three parts, and is organised as follows:

Part A — This part concerns the creation of projects using the Vivado HLS GUI. It details the inclusion of relevant source and test files and generation of the project and performance reports.
Part B — This part involves design optimization of a matrix multiplication function through the use of PIPELINE.

Part A: Creating Projects in Vivado HLS
In this exercise we will present the creation of Vivado HLS projects using both the Vivado HLS GUI.
(a) Copy the files from ***<LAB_Folder>\sources\hls*** to a new directory of your choice. We will refer to this new directory by ***<PROJ_DIRECTORY>***. (Do NOT use spaces in the project directory path).
(b) Launch the Vivado HLS GUI by navigating to ***Start > All Programs > Xilinx Design Tools > Vivado 2016.4 > Vivado HLS > Vivado HLS 2016.4***
(c) When the Vivado HLS GUI loads, you will be presented with the Welcome screen as in Figure .1.

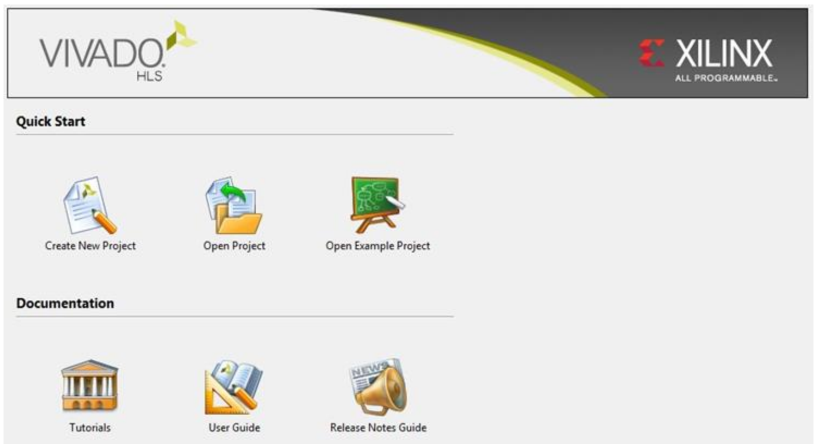
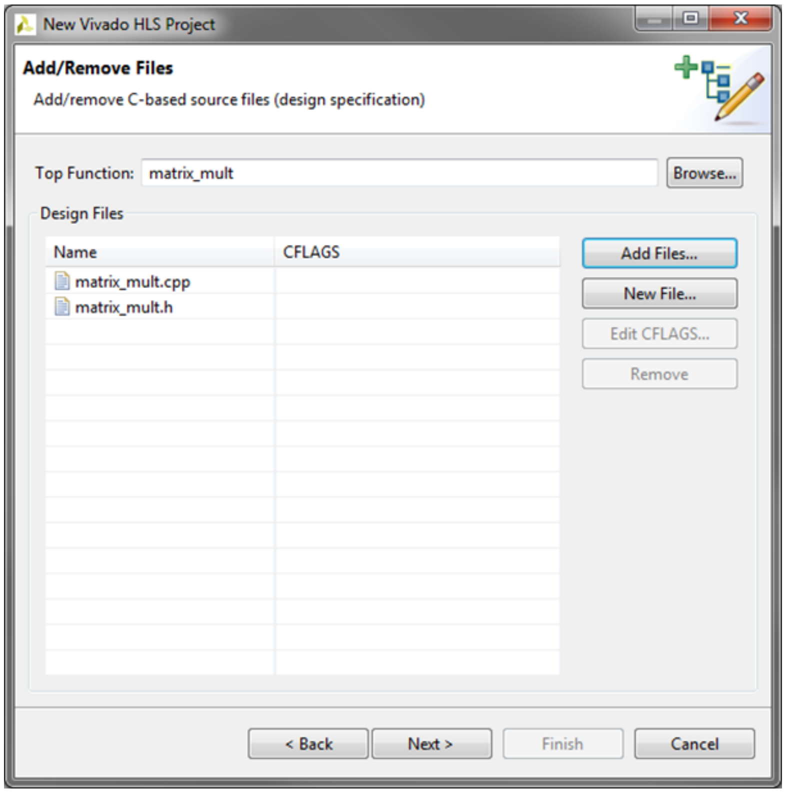


Figure 1: Vivado HLS welcome screen

(d) Select the option to Create New Project in Figure 1
(e) At the Project Name dialogue, enter ***matrix_mult_prj*** as the Project name and ***<PROJ_DIRECTORY>\hls\tut3A*** as Project location. Click Next.
(f) You will now be prompted to add or remove source files for the project. All C-based source files for this tutorial have been created in advance, as we seek to guide through the design flow rather than the programming itself. Click ***Add Files*** and navigate to ***<PROJ_DIRECTORY>\hls\tut3A***


New Vivado HLS Project

Add/Remove Files

Add/remove C-based source files (design specification)

Top Function: matrix_mult

Design Files

Name	CFLAGS
matrix_mult.cpp	
matrix_mult.h	

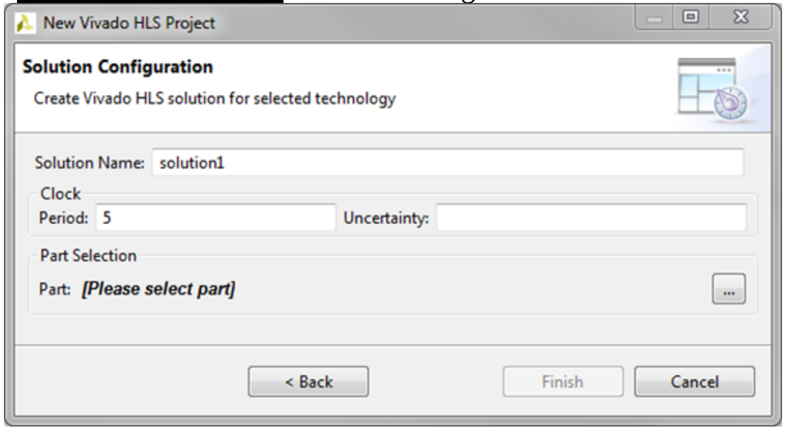
Figure 2: Adding files to a Vivado HLS project

Select the files ***matrix_mult.cpp and matrix_mult.h*** (hold down control to select multiple files) and click Open. Set the top function to ***matrix_mult*** as in Figure 2.

Click Next.

(g) You will now be prompted to add a *testbench file* for design testing. Once more, click ***Add Files*** and navigate to the previous directory this time adding the file ***matrix_mult_test.cpp*** and clicking Next.

The next step is configuring a solution for a specific FPGA technology. In this case, leave the solution name and ensure the ***clock period is set to 5*** as shown in Figure 3.


New Vivado HLS Project

Solution Configuration

Create Vivado HLS solution for selected technology

Solution Name: solution1

Clock

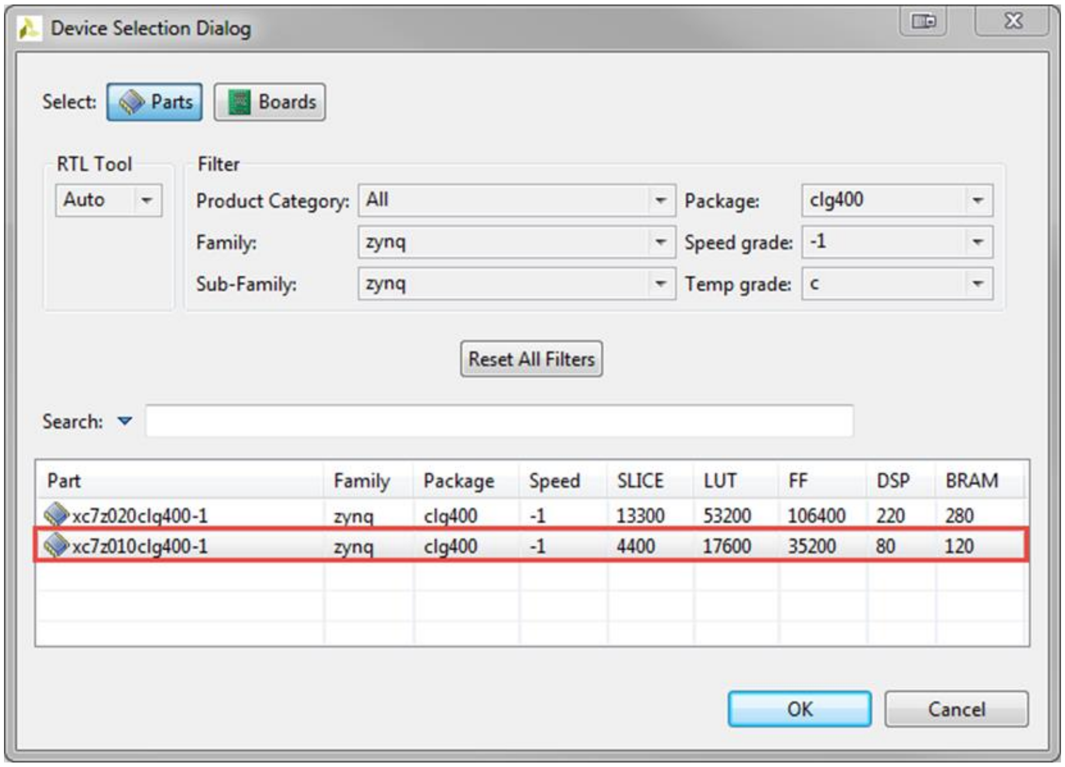
Period: 5 Uncertainty:

Part Selection

Part: *[Please select part]*

Figure 3: Solution Configuration Window

Since we are using the ***Zybo*** with the ***Zynq-7010 chip*** click, ... in the part selection panel.



RTL Tool: Auto

Product Category: All
Package: clg400
Family: zynq
Speed grade: -1
Sub-Family: zynq
Temp grade: c

Part	Family	Package	Speed	SLICE	LUT	FF	DSP	BRAM
xc7z020clg400-1	zynq	clg400	-1	13300	53200	106400	220	280
xc7z010clg400-1	zynq	clg400	-1	4400	17600	35200	80	120

Figure 5: Zybo Part selection dialogue

In the Select section click Parts and then filter the board parts using the filter drop down menus, as shown in Figure 5. The required part can be confirmed by inspecting the Zynq chip on the Zybo development board. The ***Z7010 Zynq*** chip with a ***clg400*** package should be selected. Click OK.
Click ***Finish***.

(h) The project will be generated, and the workspace will open in Synthesis mode for the generated project and solution as in Figure 6. Expanding the Source and Test Bench sections in the Explorer tab on the left side shows the inclusion of the source and test files from the previous steps. Double clicking on these files opens them in the editor view for examination and editing.
The project consists of a ***matrix multiplier***, which multiplies two matrices ***mat_a*** and ***mat_b*** to produce the output ***mat_prod***. The testbench performs the multiplication of two known matrices using normal software and checks the value of the hardware product against the expected software-generated values.

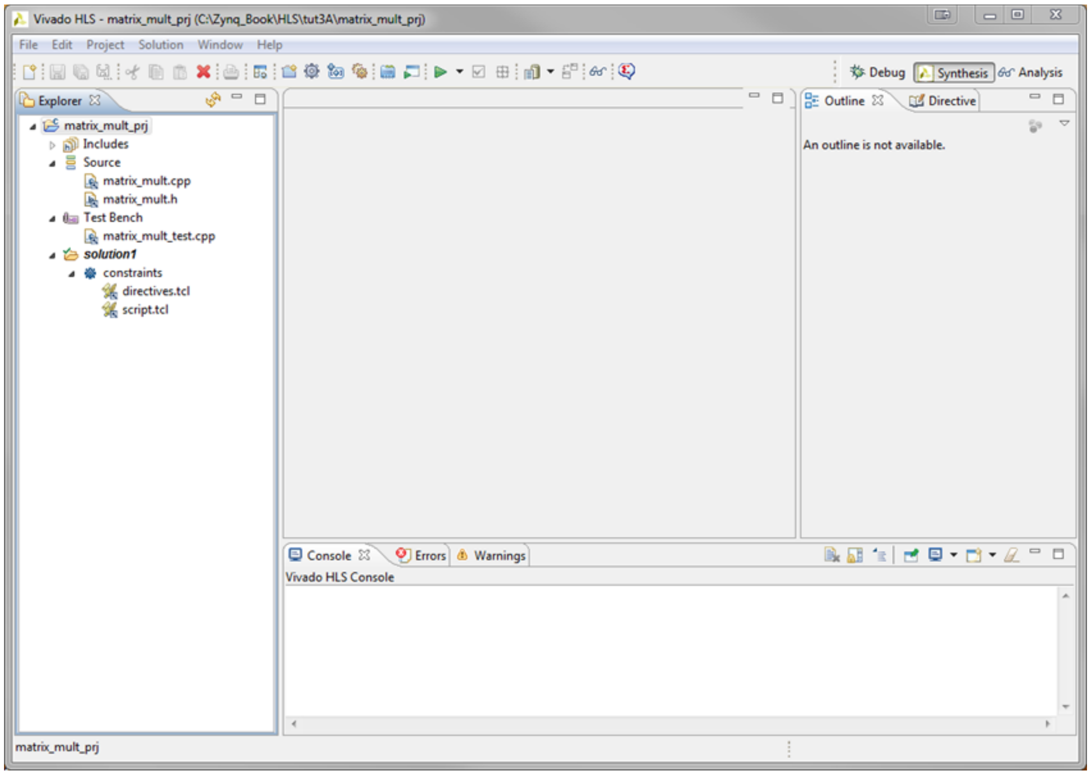
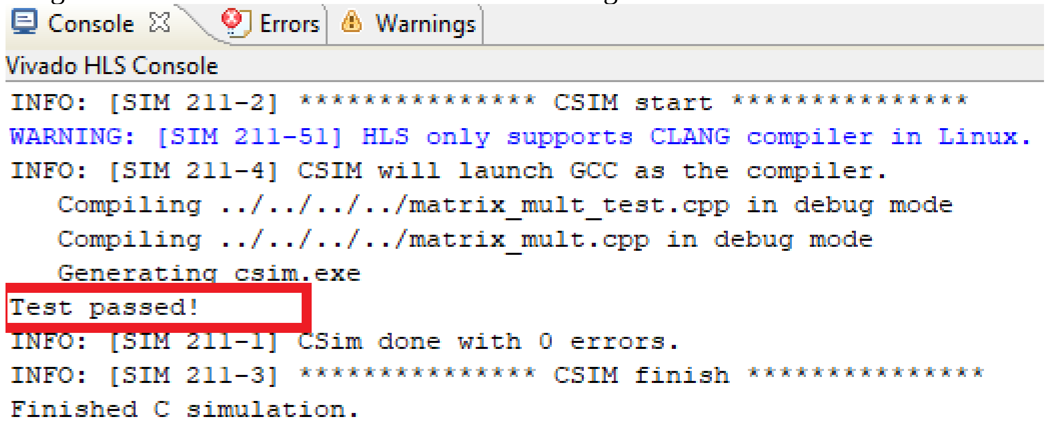


Figure 6: Synthesis view in the workspace

To test the accuracy of the hardware project described in ***matrix_mult.cpp*** file using the Test Bench code in ***matrix_mult_test.cpp*** file, run **"Run C Simulation"** from the toolbar or from ***Project-> Run C Simulation***. Leave all defaults in the ***"Run Simulation Dialog"*** window and click OK. You should see the following text in the Vivado HLS Console as seen in Figure 7.



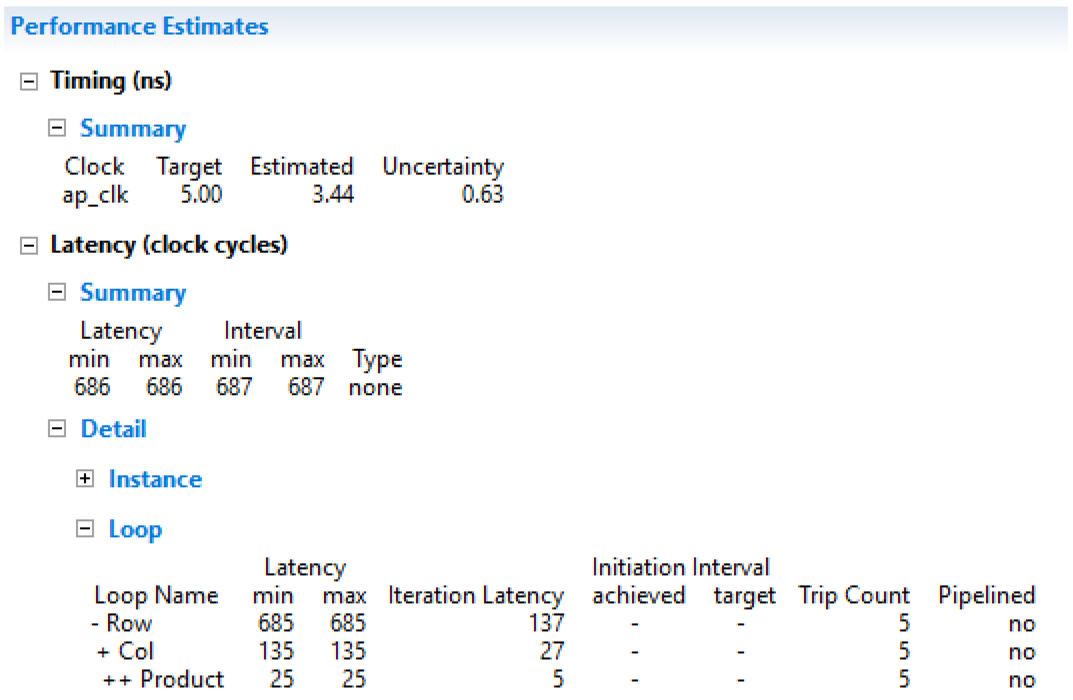
```
INFO: [SIM 211-2] *************** CSIM start ***************
WARNING: [SIM 211-51] HLS only supports CLANG compiler in Linux.
INFO: [SIM 211-4] CSIM will launch GCC as the compiler.
   Compiling ../../../../matrix_mult_test.cpp in debug mode
   Compiling ../../../../matrix_mult.cpp in debug mode
   Generating csim.exe
Test passed!
INFO: [SIM 211-1] CSim done with 0 errors.
INFO: [SIM 211-3] *************** CSIM finish ***************
Finished C simulation.
```

Figure 7: Vivado HLS command prompt

Using the project generated in the previous exercise, we will now investigate the process of design optimisation in Vivado HLS. This will also provide an insight into the flow from project creation to **C synthesis and C/RTL Co simulation**. **C synthesis** will convert the C code in ***matrix_mult.cpp*** into VHDL design and **C/RTL Co simulation** will test the generated VHDL code in simulation mode. Will also discuss the use of the **Analysis perspective** in analysing a HLS solution.

(i) The next step is to synthesise the C++ code using HLS. Click the ***C Synthesis*** button in the toolbar. Vivado HLS will begin the process of converting the C++ code into an RTL model (VHDL code). RTL

stands for "Register-transfer level" which is the level of design that VHDL code uses to design systems. The console details the steps performed in achieving this.
Upon completion, a Synthesis Report will open automatically as seen in figure 8. These details various aspects of the synthesised design, such as information concerning timing and latency and FPGA resource utilisation estimates. (You may require expanding sub-sections to see results.)



Performance Estimates

⊟ **Timing (ns)**

⊟ Summary

Clock	Target	Estimated	Uncertainty
ap_clk	5.00	3.44	0.63

⊟ **Latency (clock cycles)**

⊟ Summary

Latency		Interval		
min	max	min	max	Type
686	686	687	687	none

⊟ Detail

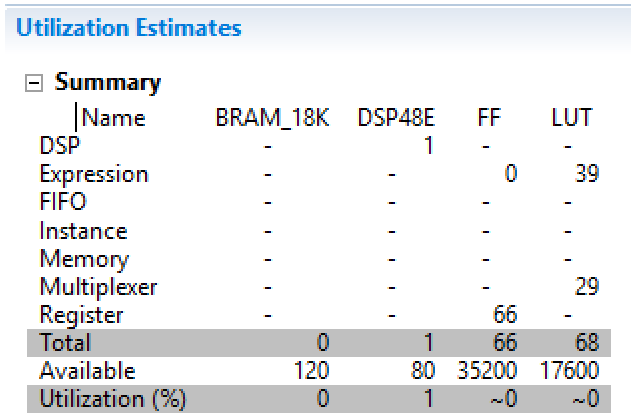
⊞ Instance

⊟ Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Row	685	685	137	-	-	5	no
+ Col	135	135	27	-	-	5	no
++ Product	25	25	5	-	-	5	no

Figure 8. Synthesis Report – Performance Estimates section

The synthesised design has an interval of **687** clock cycles. Each input array contains **25** elements (as it uses **5x5** matrices) and so this suggests roughly ***27 clock cycles per input read***. The hardware resources used to build the system (i.e. number of LUTs, Flipflops (FF), DSP (DSP48) …etc) is available in the summary report as shown in the following figure under ***Utilization Estimates*** section:



Utilization Estimates

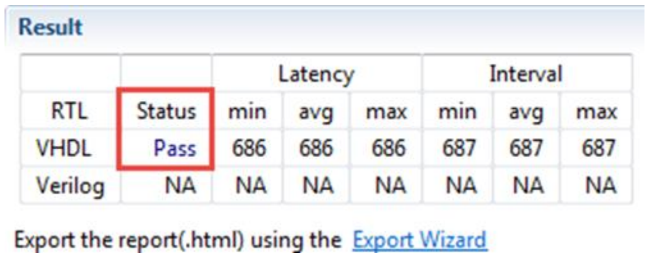
⊟ **Summary**

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	1	-	-
Expression	-	-	0	39
FIFO	-	-	-	-
Instance	-	-	-	-
Memory	-	-	-	-
Multiplexer	-	-	-	29
Register	-	-	66	-
Total	0	1	66	68
Available	120	80	35200	17600
Utilization (%)	0	1	~0	~0

Figure 9. Synthesis Report – Utilization Estimates section

We can now run a C/RTL cosimulation to ensure that the synthesised RTL behaves exactly the same as the C++ code in the Test Bench.

Click the Run ***C/RTL Cosimulation*** button . For the RTL selection, ensure **VHDL** is selected and click OK. Co simulation will now begin, with the RTL system being generated using **VHDL**. This process may take a short while to complete but progress can be viewed in the console. Upon completion, the C osimulation Report will be opened as in Figure 10



Result

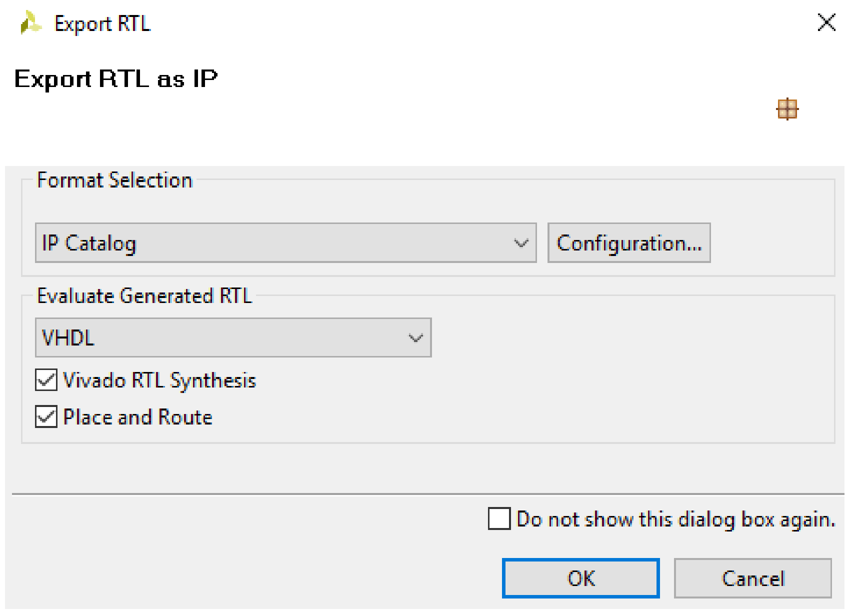
		Latency			Interval		
RTL	Status	min	avg	max	min	avg	max
VHDL	Pass	686	686	686	687	687	687
Verilog	NA	NA	NA	NA	NA	NA	NA

Export the report(.html) using the Export Wizard

Figure 10: Cosimulation report for the matrix multiplier, solution

Note the "Pass" message of Figure 10 indicating that the VHDL output behaves the same as the C++ source code in the Test Bench.

(j) Run Export RTL from the toolbar or from Solution-> Export RTL. Make sure the VHDL is selected and check the boxes as shown in the following figure. This step may take some time. Watch the progress in the progress bar on the right-bottom part of the Vivado HLS GUI.



Export RTL

Export RTL as IP

Format Selection: IP Catalog

Evaluate Generated RTL: VHDL

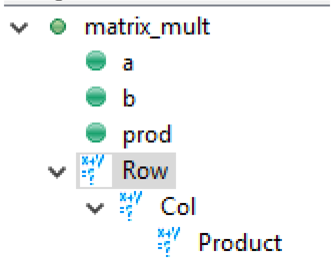
[✓] Vivado RTL Synthesis

[✓] Place and Route

[ ] Do not show this dialog box again.

This will create a folder under "Solution1" called "**impl**". Go to "***Solution1->impl->vhdl***" and examine the file matrix_mult.vhd. You do not have to understand all the VHDL codes in this file. However, this VHDL code will show you the general definition of the hardware component that you built. Have a look on the matrix_mult entity and examine the inputs and outputs defined for your hardware system.

Part B: Design Optimisation
Double click on ***matrix_mult.cpp*** in the Source section of the Explorer tab to ensure the code is visible in the workspace. We will now insert a directive which will pipeline the loops of the matrix multiplication code to be executed in parallel.
Open the Directives tab to the right of the workspace. Click on **Row** and you will observe the associated portion of code highlighted in the editor. *Right click on **Row** and select **Insert Directive***:



This will open the Directives Editor. Use the type drop-down menu to select the option PIPELINE. Click OK to accept the default options. The directives tab should now resemble Figure 11.

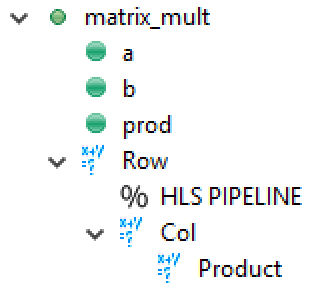
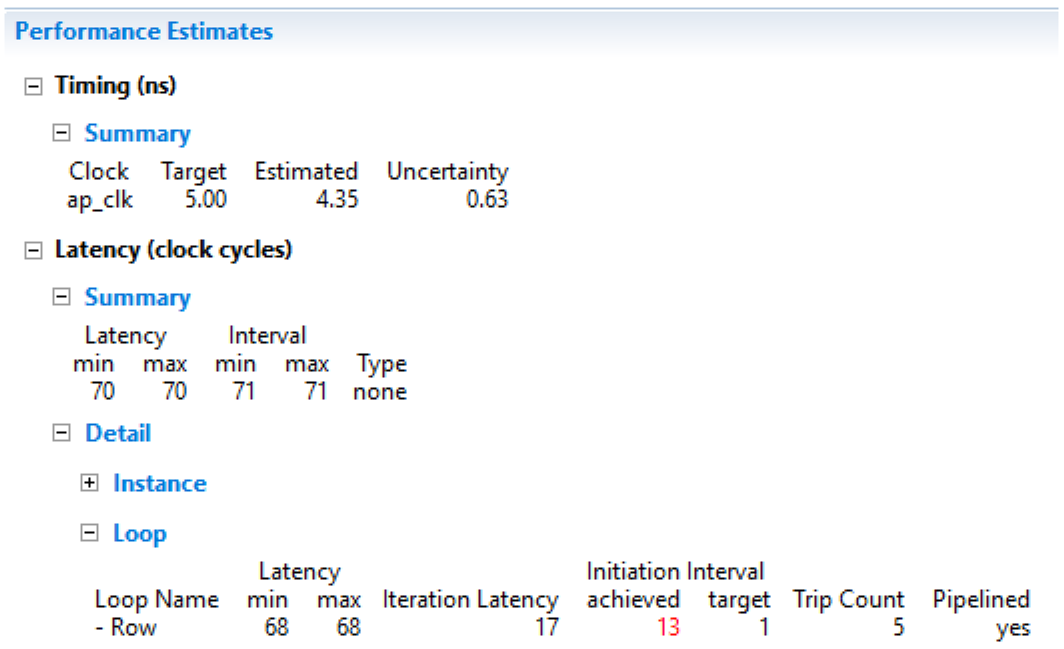


Figure 11: Pipelining nested loops in HLS

Click the C Synthesis button to synthesise the RTL design. The console yields some information about the process of flattening the Row loop. It also explains that the default Initiation Interval (II) target of 1 could not be met for the Product loop. This is due to loop dependency. Watch the new Synthesis report which should be like the following figure:



Performance Estimates

⊟ **Timing (ns)**

⊟ **Summary**

Clock	Target	Estimated	Uncertainty
ap_clk	5.00	4.35	0.63

⊟ **Latency (clock cycles)**

⊟ **Summary**

Latency		Interval		
min	max	min	max	Type
70	70	71	71	none

⊟ **Detail**

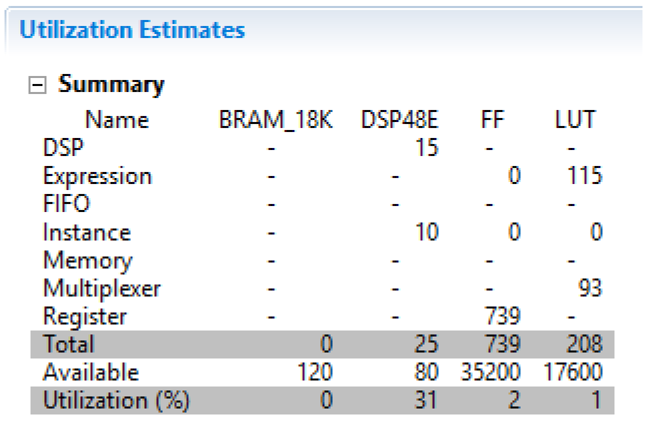
⊞ **Instance**

⊟ **Loop**

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Row	68	68	17	13	1	5	yes

12: Synthesis report for the optimized matrix multiplier

From the synthesis report shown in Figure 12 it is observed that the latency has been reduced to 70 and the interval has been reduced to 71. Check the ***Utilization Estimates section*** which should look like the following figure. As seen in the figure, the hardware resources (i.e. number of LUTs, Flipflops (FF), DSP (DSP48) …etc) used to build the system has been increased. This shows how performance gain is obtained on the expense of extra hardware expenses.

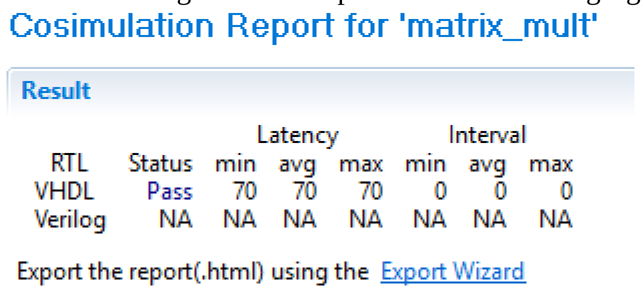


Utilization Estimates

⊟ **Summary**

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	15	-	-
Expression	-	-	0	115
FIFO	-	-	-	-
Instance	-	10	0	0
Memory	-	-	-	-
Multiplexer	-	-	-	93
Register	-	-	739	-
Total	0	25	739	208
Available	120	80	35200	17600
Utilization (%)	0	31	2	1

Run ***C/RTL Cosimulation*** which should generates a report like the following figure:



Cosimulation Report for 'matrix_mult'

Result

		Latency			Interval		
RTL	Status	min	avg	max	min	avg	max
VHDL	Pass	70	70	70	0	0	0
Verilog	NA	NA	NA	NA	NA	NA	NA

Export the report(.html) using the Export Wizard