ECE4810J LAB 2 Group 8 Report

Jinlock Choi 520370990012 Yang Huancheng 521370910090

October 2024

Contents

1 Section 2

1.1 RTL multiplier and HLS multiplier

In this section we learned how to create and use Custom IP Blocks using High-Level Synthesis. Especially, we used Vitis HLS to generate HDL code from higher-level languages such as C or C++.

DELIVERABLE 1: The screenshot of the Implemented Design highlighting RTL multiplier and HLS multiplier leaf cells:

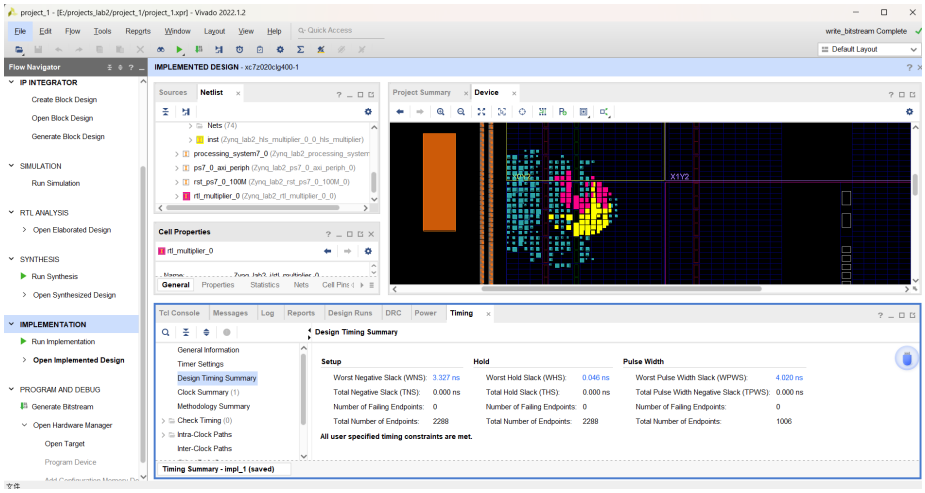

Figure 1: Highlighting RTL multiplier and HLS multiplier

DELIVERABLE 2: Utilization Summary and the Design Timing Summary of the Implemented Design leaf cells:

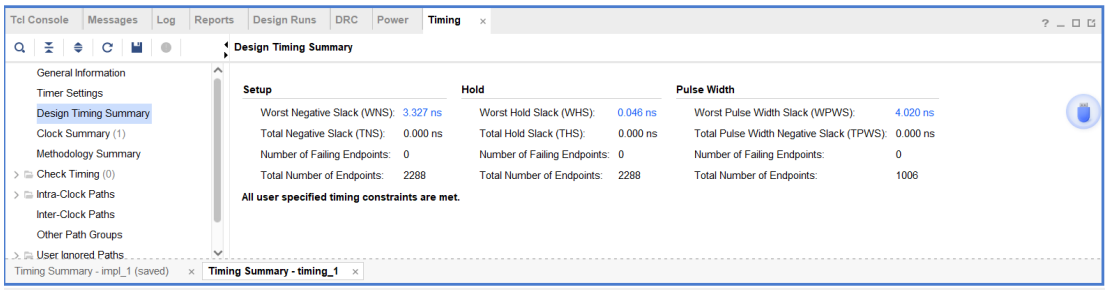

Figure 2: Timing Summary

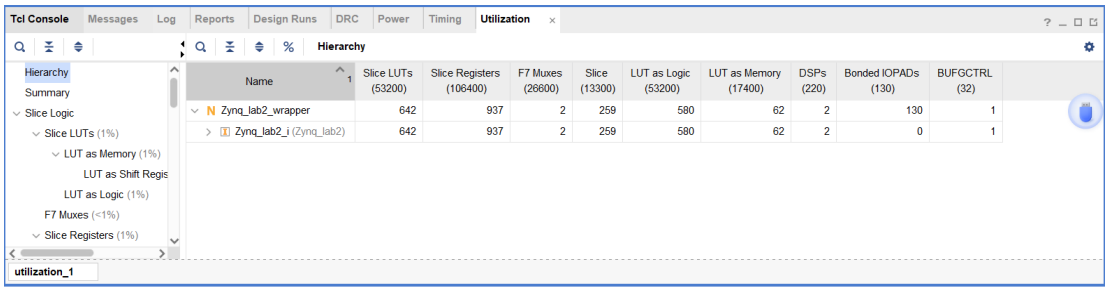

Figure 3: Implemented design

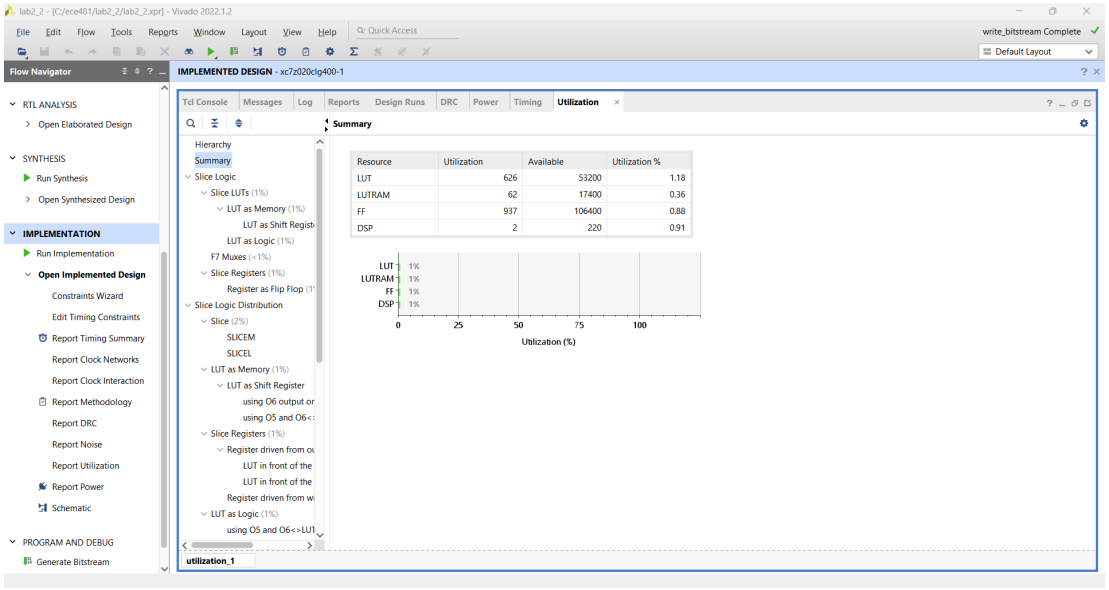

Figure 4: Utilization Summary

DELIVERABLE 3: Vitis Serial Terminal of *hello_hls_rtl_multiplier* Vitis project:

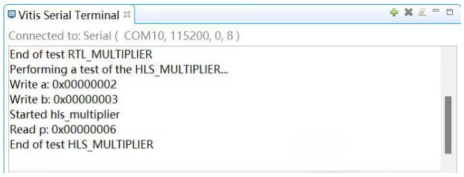

Figure 5: Serial Terminal of hello_hls_rtl_multiplier

1.2 RTL ALU and HLS ALU

DELIVERABLE 4: Screenshot of the Synthesis Summary Report of *hls_alu* containing Timing Estimate and Performance and Resource

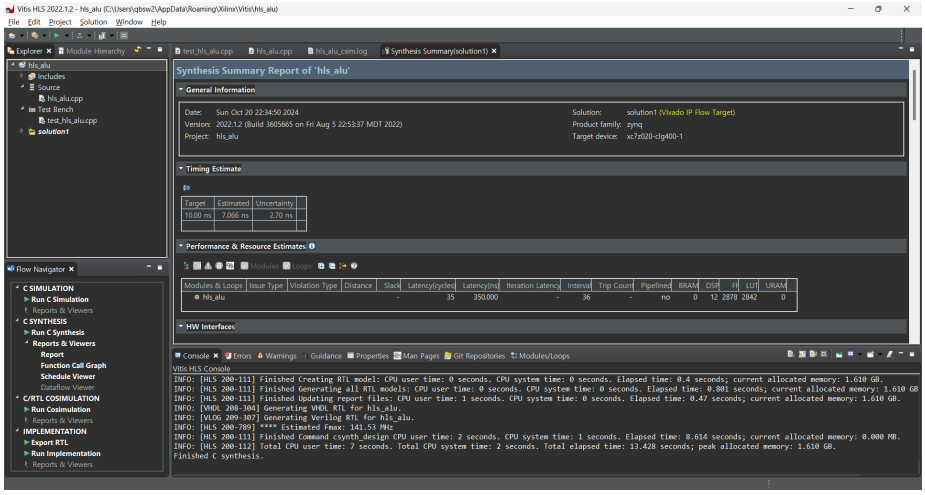

Figure 6: Timing Estimate and Performance and Resource Report

DELIVERABLE 5: The screenshot of the generated Cosimulation Report for *hls_alu*

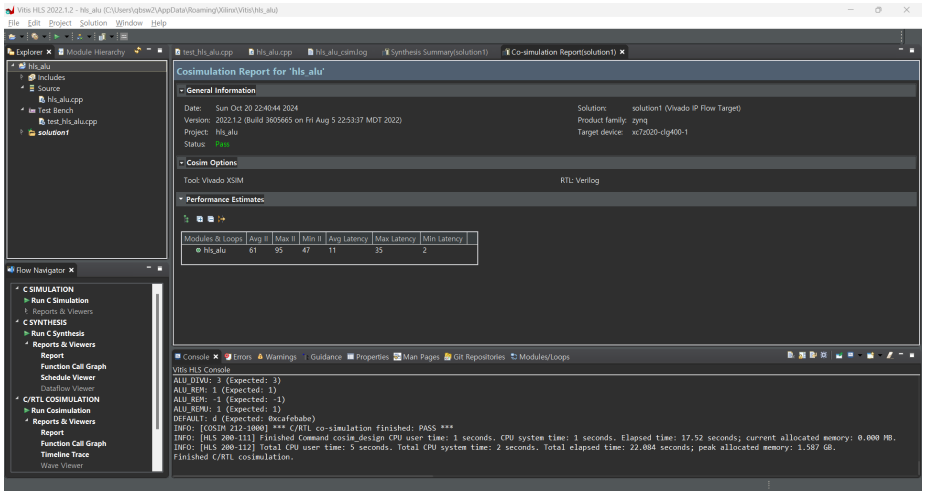

Figure 7: Cosimulation Report

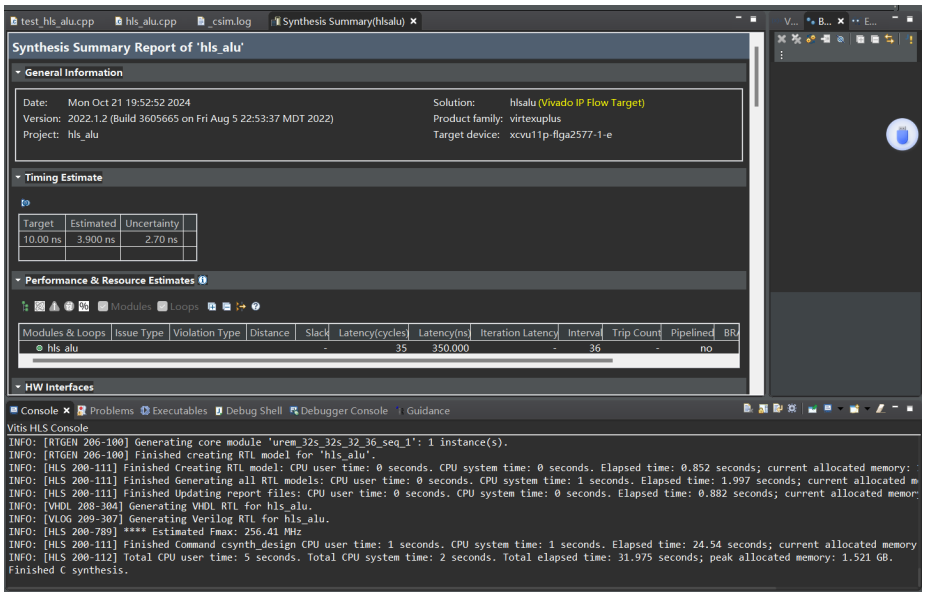

Figure 8: Synthesis Report

DELIVERABLE 6: The screenshot of the resource utilization of the Implemented Design highlighting RTL ALU and HLS ALU leaf cells

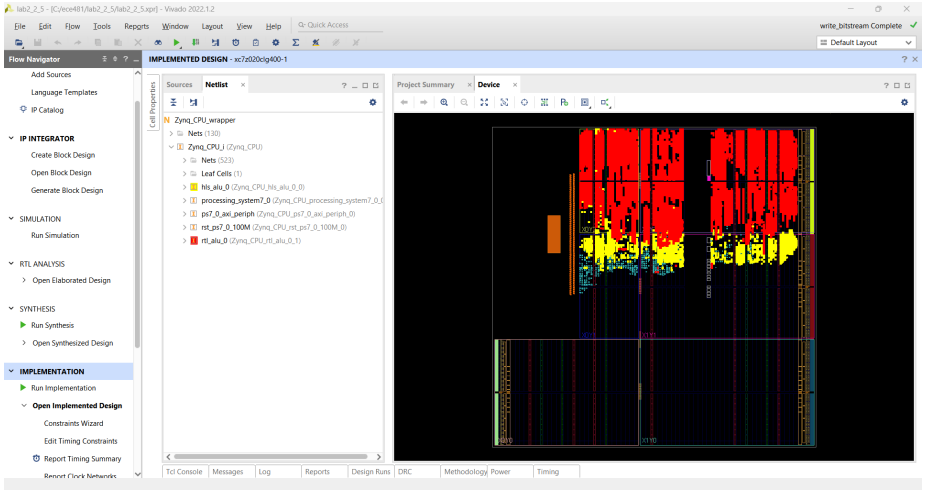

Figure 9: Highlighted RTL ALU and HLS ALU leaf cells

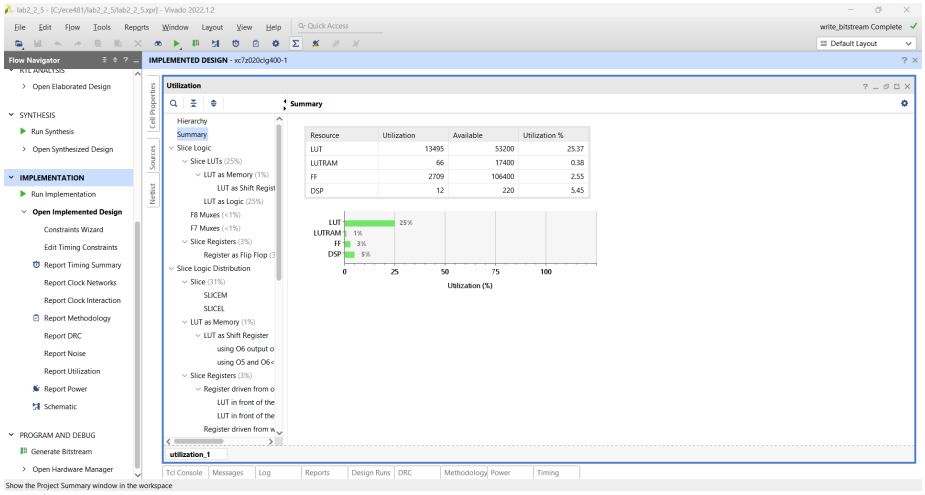

Figure 10: Utilization Summary of Implemented Design

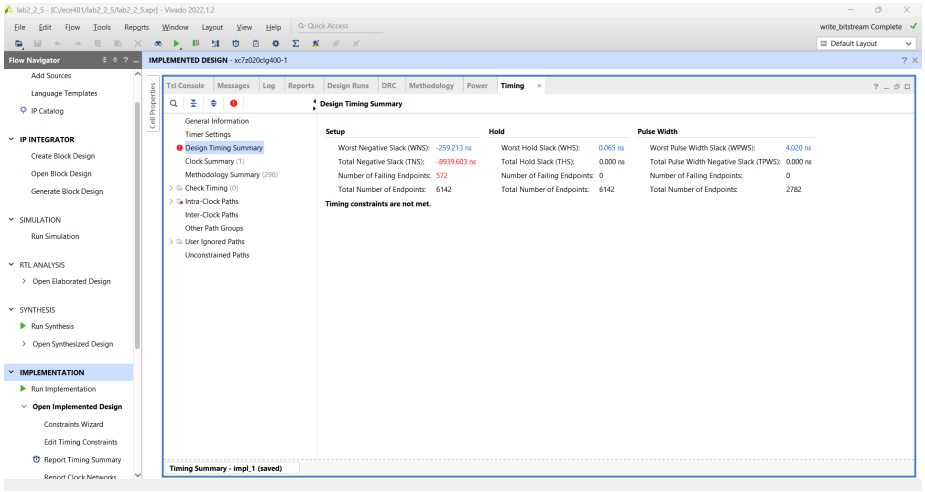

Figure 11: Design Timing Summary of Implemented Design

DELIVERABLE 7: The screenshot of your Vitis IDE window at the build-finished state of *hello_hls_rtl_alu* Vitis project

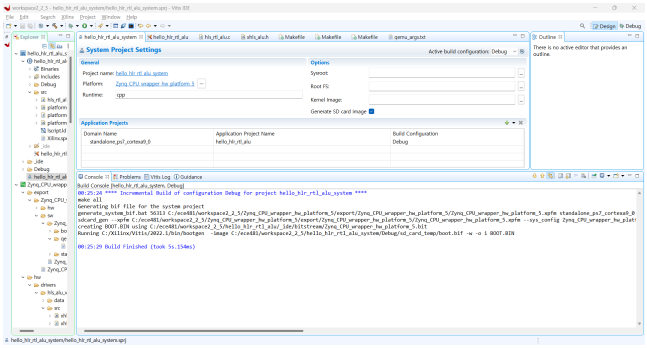

Figure 12: Vitis IDE window at the build-finished state

DELIVERABLE 8: Screenshot of the Vitis Serial Terminal of *hello_hls_rtl_alu* Vitis project

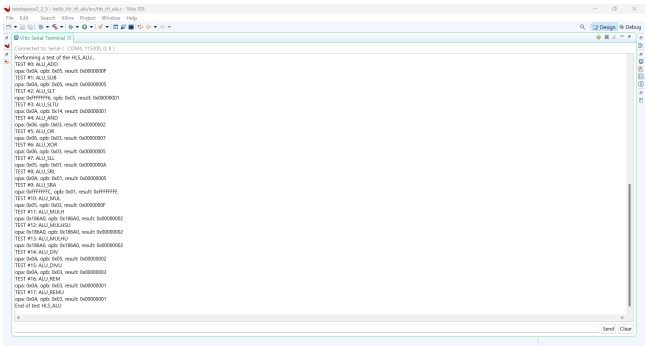

Figure 13: Serial Terminal showing HLS ALU result part

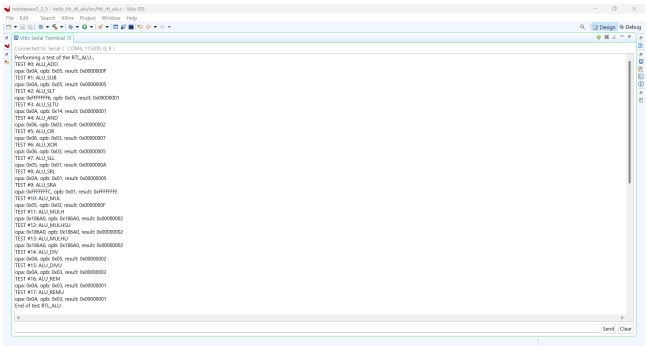

Figure 14: Serial Terminal showing RTL ALU result part

2 Section 3

2.1 Program the FPGA and Boot Software from the SD Card

In this section, we learned how to take developed C-code and boot it from the SD card inserted in the FPGA. By utilizing this way, we can save efforts of running the software from Vitis.

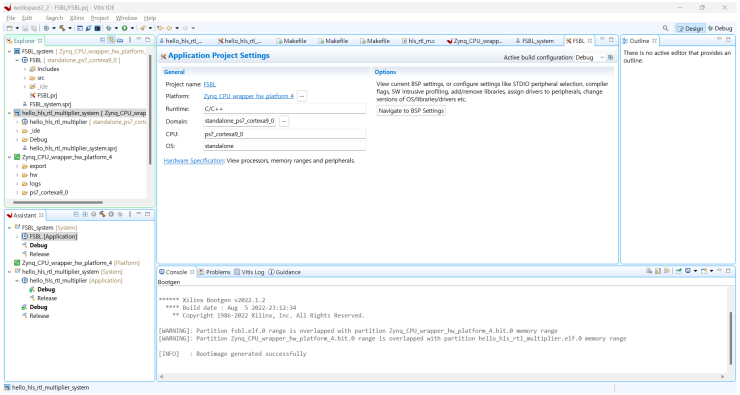

Figure 15: Boot Image

DELIVERABLE 9: the screenshot of the Vitis Serial terminal or PuTTY terminal

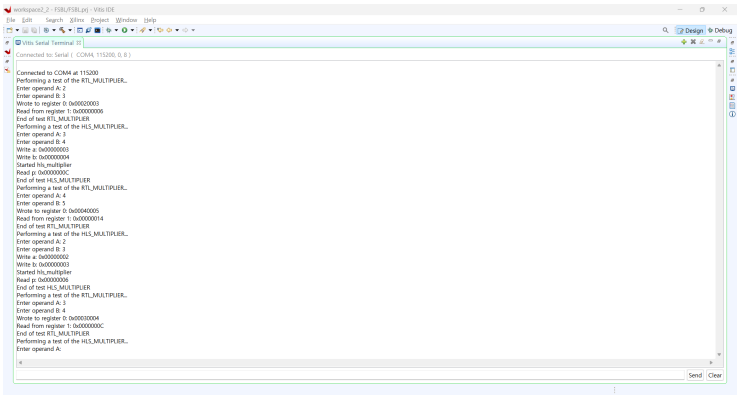

Figure 16: Vitis Serial Terminal

3 Section 4

3.1 Program Python on Arty Z7

In this section, we learned how to program Python on Arty Z7 by utilizing PYNQ framework.

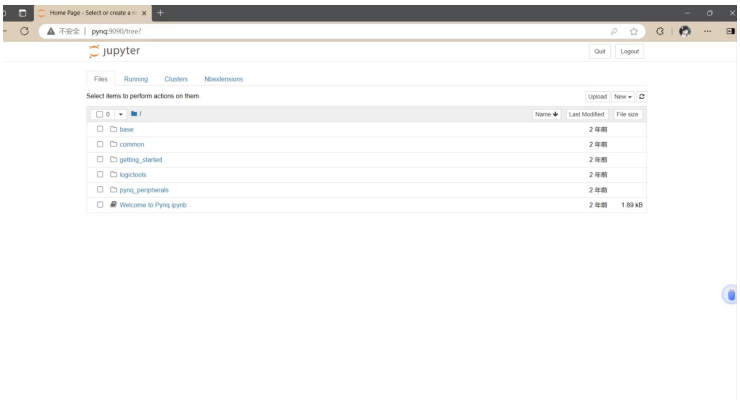

Figure 17: screenshot of Jupyter Notebook with address bar

4 Section 5

4.1 Optimize Your RTL ALU by Pipelining

DELIVERABLE 11: The screenshot of the Implemented Design highlighting RTL ALU and HLS ALU leaf cells

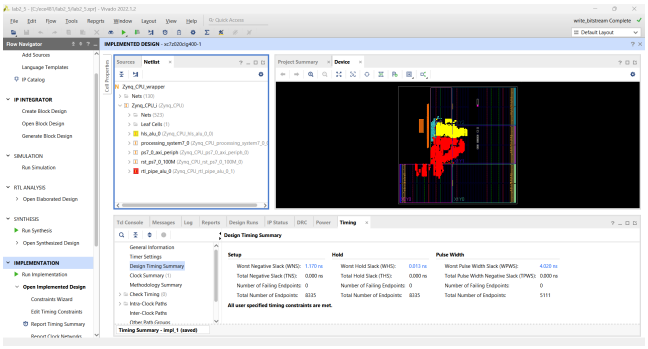

Figure 18: Implemented Design highlighting RTL ALU and HLS ALU leaf cells

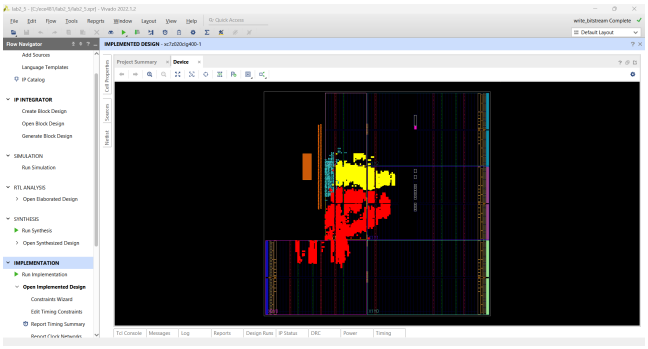

Figure 19: Implemented Design highlighting RTL ALU and HLS ALU leaf cells

DELIVERABLE 12: The screenshot of the Utilization Summary and the Design Timing Summary of the Implemented Design

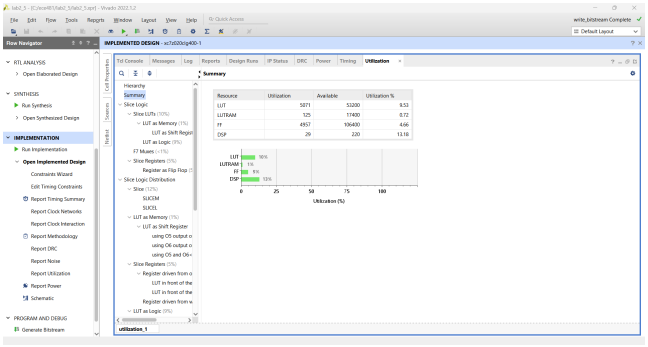

Figure 20: Utilization Summary

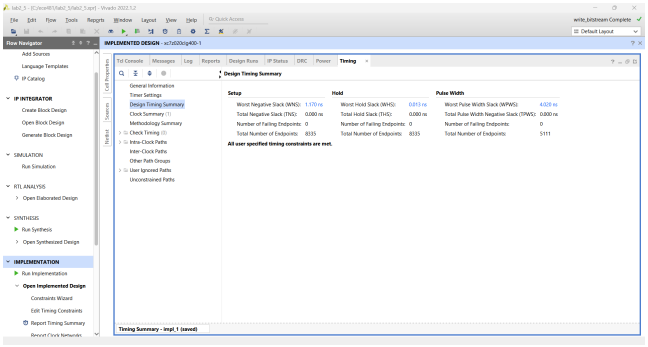

Figure 21: Design Timing Summary

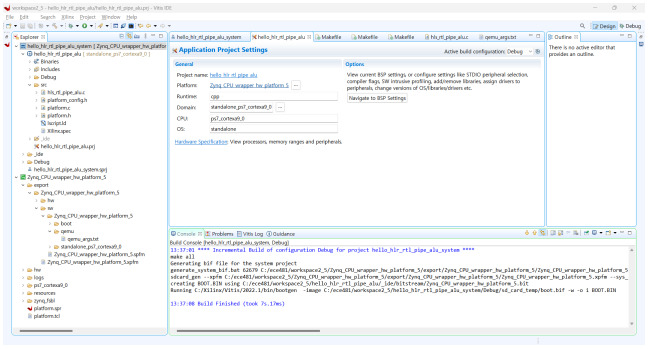

Figure 22: HLS_RTL_PIPE_ALU Build finished state

DELIVERABLE 13: The screenshots of the Vitis Serial Terminal of pipe_hls_rtl_alu Vitis project

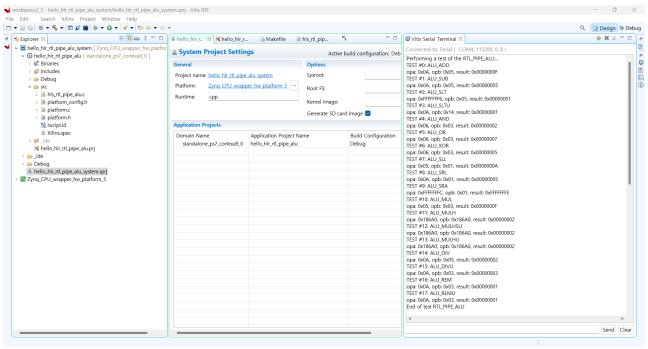

Figure 23: Vitis Serial Termina RTL Part

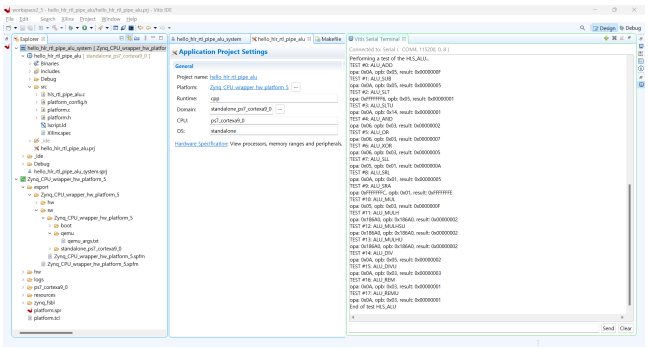

Figure 24: Vitis Serial Termina HLS Part

4.2 Observations

Compared to the non-pipelined RTL ALU implemented in section 2, Pipelined RTL ALU shows significant improvement in negative slacks, as multiplier and divisors go into several smaller stages. Also, Pipelined RTL ALU utilizes less LUT then non-pipelined one.

5 6 Post-lab Questions

5.1 What was the resource utilization of the RTL implementation vs. HLS implementation? If they are different, can you guess why there are differences?

RTL designs are hand-coded at a lower abstraction level, allowing precise control over resource usage. As a result, resource utilization is often optimized for the specific needs of the application, which may lead to more efficient use of hardware.HLS, on the other hand, starts from a higher-level description (e.g., C/C++), and then synthesizes hardware. Due to this higher-level abstraction, HLS tools may generate less optimized hardware since they need to make more generalized decisions during synthesis. Difference may due to the granularity of control and manual optimizations available in RTL but lacking in HLS.

5.2 Imagine that you have to create an IP block implementing ten different sorting algorithms. Would you rather use a hardware description language or High-level Synthesis? Why?

If I had to create an IP block that implements ten different sorting algorithms, I would prefer to use High-Level Synthesis (HLS) rather than a traditional Hardware Description Language (HDL): HLS allows you to design at a higher level of abstraction using languages like C or C++, which significantly speeds up the process of writing and debugging complex algorithms compared to the lower-level details of HDLs; What‘s more, HLS can make complex algorithm.Sorting algorithms, by nature, involve a lot of iterative and conditional logic, which is more naturally expressed in a high-level language. Also The HLS approach enables you to write more modular and reusable code.

5.3 Read more about PYNQ, and answer the following questions:

What is a Jupyter notebook? Why do we use it?

A Jupyter Notebook is an open-source web application that allows you to create and share documents. It supports a variety of programming languages, with Python being the most commonly used. Jupyter Notebooks have several advantages that make them suitable for many purposes, such as education, research, data analysis, and prototyping hardware designs.

What are overlays? What are the benefits of using overlays?

Overlays are pre-configured bitstreams or hardware designs that are used to program the programmable logic of an FPGA device, such as the ones found in

Zynq chips. Overlays provide several benefits, especially for users who are not experts in hardware design or who are looking to accelerate the development of applications that utilize FPGAs.

Please list available overlays on the Arty Z7 board.

LED control,audio processing, image control.....

Comparing part 2 where you design with C/C++ vs. part 4 (PYNQ framework with python), what are the major differences?

C/C++ with HLS (Part 2) involves designing custom hardware using C/C++, synthesizing to RTL, and provides lower-level control.PYNQ with Python (Part 4) offers a high-level interface where you can load pre-built bitstreams, use Python to control the FPGA, and focus on the application-level development rather than hardware design.