Artifact for "HLS Taking Flight: Toward Using High-Level Synthesis Techniques in a Space-Borne Instrument"

Abstract

This artifact contains the kernel and test bench source code for high-level synthesis (HLS) of ADAPT's data preprocessing and reduction algorithms as presented in:

> M. Sudvarg, C. Zhao, Y. Htet, M. Konst, T. Lang, N. Song, R. D. Chamberlain, J. Buhler, and J. H. Buckley. "HLS Taking Flight: Toward Using High-Level Synthesis Techniques in a Space-Borne Instrument." In Proc. of 21st International Conference on Computing Frontiers. ACM, 2024.

Code is written in C++ for compilation in the AMD XILINX Vitis HLS version 2021.1, targeting the Kintex-7 KC 705 evaluation platform (XC7K325T). Additional data files encoding a sample data packet and analog memory pedestal values from the front-end analog waveform digitizer ASIC are also included for testing purposes.

Users of this artifact can reproduce the results shown in Table 1. Users will need to install the free Vitis HLS software, which will emulate the target FPGA. Users **do not** need a physical FPGA to reproduce this artifact.

Author Information

For questions about this artifact, please contact:

Marion Sudvarg
McKelvey School of Engineering
Washington University in St. Louis
St. Louis, Missouri, USA
msudvarg@wustl.edu (mailto:msudvarg@wustl.edu)

Artifact check-list (meta-information)

- **Algorithm:** We present implementations of data pre-processing algorithms (Listings 1–5) for high-energy particle telescopes.
- **Program:** C++ implementations of test-bench and HLS kernels.
- **Data set:** Sample data packet and pedestal files from front-end ALPHA ASICs.
- **Run-time environment:** AMD XILINX Vitis HLS 2021.1. Can be installed on Windows 10, 11, or Linux with desktop environment. The authors have used Ubuntu 22.04 and Rocky Linux 8.9.
- **Metrics:** FPGA kernel speed and area.

- **Output:** Synthesis reports with data in Table 1.
- **Approximate disk space required:** 50GB to install Vitis HLS.
- **Approximate time to prepare workflow:** 1–2 hours to install Vitis HLS.
- **Approximate time to complete experiments:** 1 hour.
- **Publicly available?:** Yes.
- **Code licenses (if publicly available)?:** MIT.
- **Data licenses (if publicly available)?:** MIT.
- **Workflow framework used?:** No.
- **Archived (provide DOI)?:** This artifact will be archived, with DOI, through Washington University in St. Louis libraries. DOI will be provided when available

Description

How to access

Note: This artifact is currently available on Github, but will be archived, with DOI, through Washington University in St. Louis libraries. This readme will be updated with a link to the archival repository, when it becomes available.

Retrieve from https://github.com/McKelvey-Engineering-CSE/adapt_fpga/tree/computing-frontiers-2024: (https://github.com/McKelvey-Engineering-CSE/adapt_fpga/tree/computing-frontiers-2024:)

```
git clone https://github.com/McKelvey-Engineering-CSE/adapt_fpga/
cd adapt_fpga
git checkout computing-frontiers-2024
```

Hardware dependencies

No special hardware needed. Only a laptop or desktop running Windows 10, 11, or Linux with a desktop environment that can support AMD XILINX Vitis HLS 2021.1.

The Vitis HLS installation requires around 50GB of free space.

Synthesis and co-simulation of all versions of the kernel pipeline requires around 1 hour on a Linux machine with 8 cores and 16GB of memory. However, 4 cores and 4GB of memory are sufficient.

Software dependencies

AMD XILINX Vitis HLS 2021.1. Installation instructions are provided below.

Data sets

Sample data packet, `EventStream.dat`, and pedestal value, `peds.dat`, files from the front-end ALPHA ASIC are included in the repository.

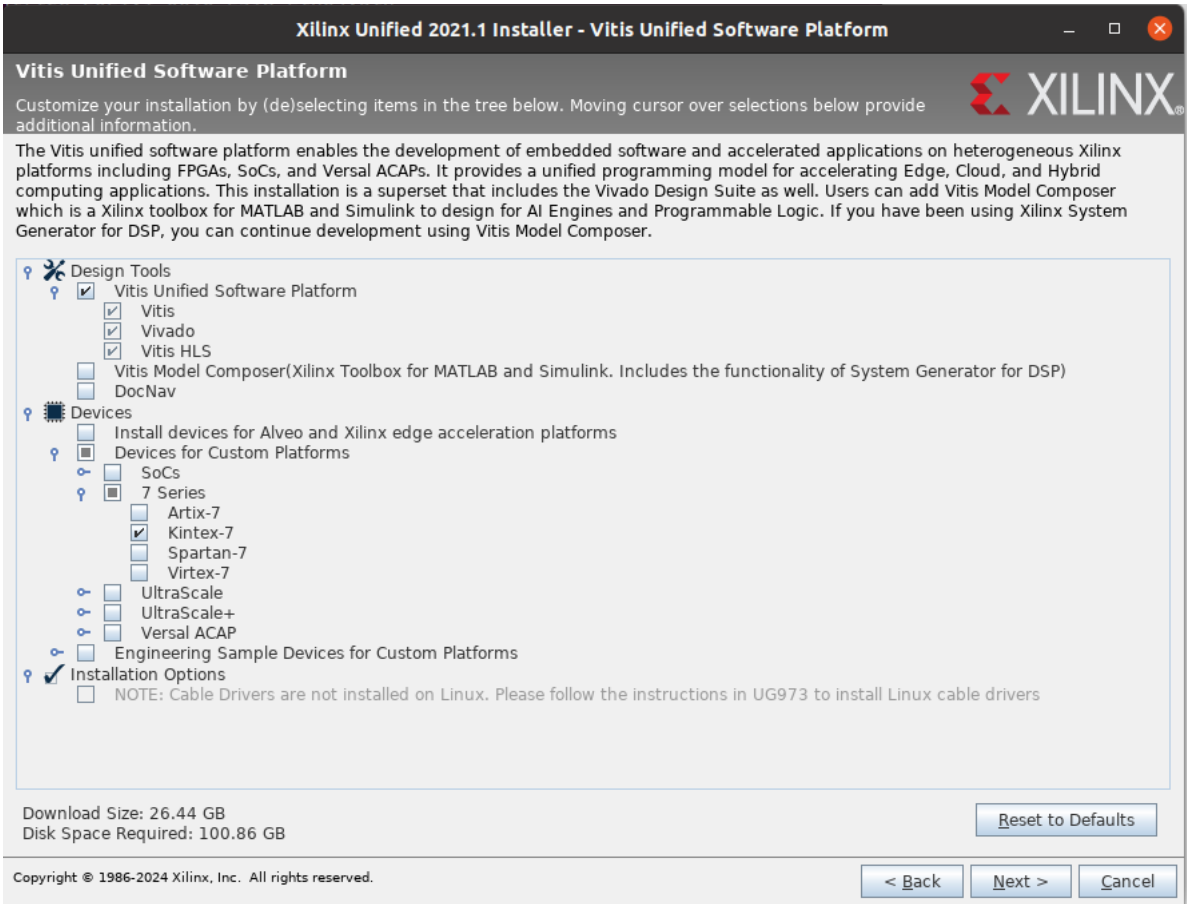
Installation

Install Vitis HLS

Install AMD XILINX Vitis HLS 2021.1 according to the instructions here: https://docs.xilinx.com/r/2021.1-English/ug1393-vitis-application-acceleration/Installing-the-Vitis-Software-Platform (https://docs.xilinx.com/r/2021.1-English/ug1393-vitis-application-acceleration/Installing-the-Vitis-Software-Platform)

We provide step-by-step instructions specific to Ubuntu 22.04.

1. Browse to the Vitis software archive: https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vitis/archive-vitis.html (https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vitis/archive-vitis.html)
2. Select 2021.1.
3. Download the "Xilinx Unified Installer 2021.1: Linux Self Extracting Web Installer."
4. Log in with an AMD account (it is free to register).
5. Verify your identity and click Download.
6. Navigate to the downloaded file and change its permissions to it is executable:
   ```
   chmod u+x Xilinx_Unified_2021.1_0610_2318_Lin64.bin
   ```
7. Execute the binary as root:
   ```
   sudo ./Xilinx_Unified_2021.1_0610_2318_Lin64.bin
   ```
8. A message may appear stating that "A Newer Version is Available." Click "Continue" to skip downloading the new version.
9. Enter your AMD account information, select "Download and Install Now," then click Next.
10. Select Product to Install: Vitis.
11. You may deselect all devices besides the Kintex-7.



12. Accept license agreements.
13. Use default options for installation directory, etc.
14. Wait for download and installation to complete (this may take a while).
15. Install additional libraries:
    ```
    sudo /tools/Xilinx/Vitis/2021.1/scripts/installLibs.sh
    ```
16. Set environment variables for paths, etc.:
    ```
    source /tools/Xilinx/Vitis/2021.1/settings64.sh
    ```
17. Launch Vitis HLS: `vitis_hls`

Note: we have observed on some Ubuntu-based installations that the Vitis HLS GUI may crash immediately after launching. A fix is detailed here: https://support.xilinx.com/s/question/0D52E00006hpY4PSAU/vitishls-20202-not-starting-only-splash-screen-visible?language=en_US (https://support.xilinx.com/s/question/0D52E00006hpY4PSAU/vitishls-20202-not-starting-only-splash-screen-visible?language=en_US)

Fix steps:

1. Open /tools/Xilinx/Vitis_HLS/2021.1/common/scripts/autopilot_init.tcl
2. Go to line 40, which reads: `----%r&-'%rl%&n$&lt'v-=`
3. Replace the last character `=` with `>`

Create Project

You will need to create a new Vitis HLS project to reproduce the results of the paper. Steps are:

1. Launch Vitis HLS: `vitis_hls`

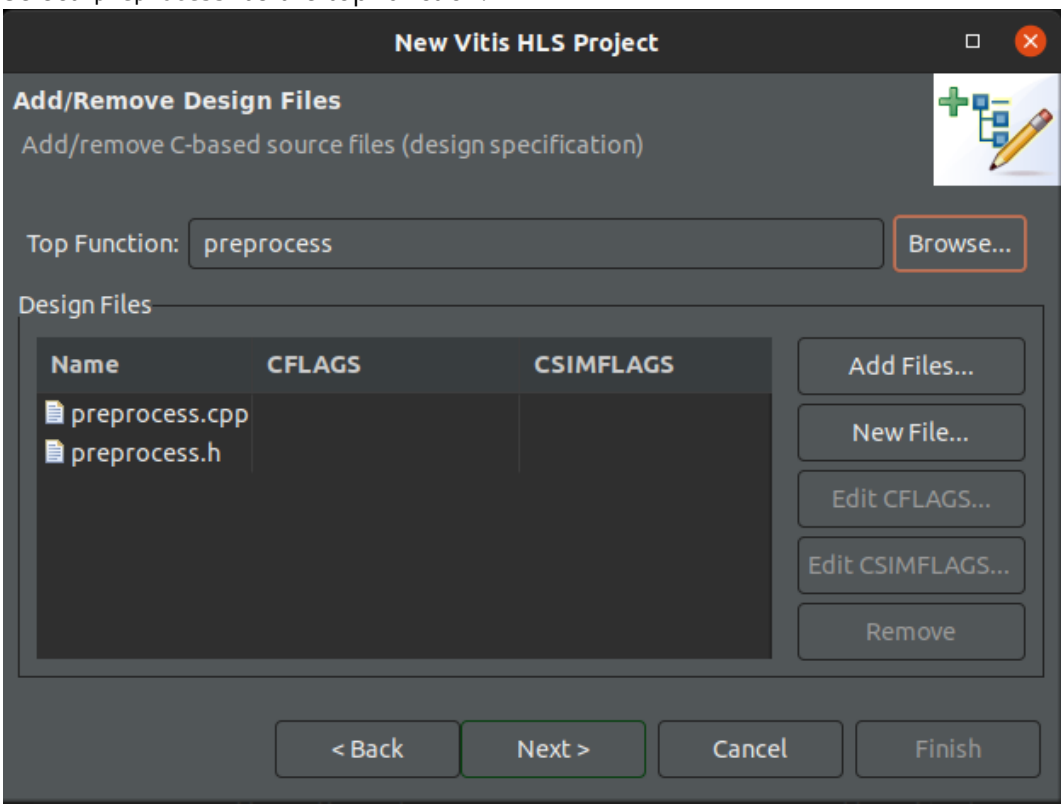
2. Click Create Project

3. Enter a project name and location

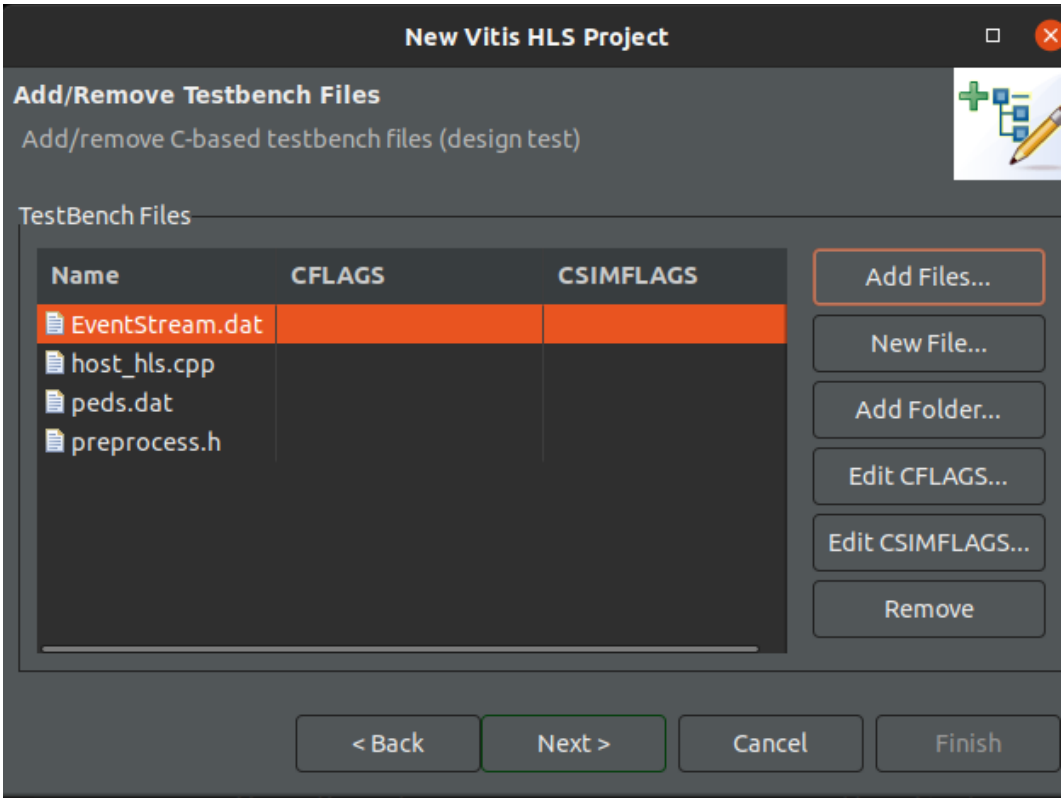
4. Add design files:

    - `preprocess.cpp`
    - `preprocess.h`

5. Select `preprocess` as the top function.

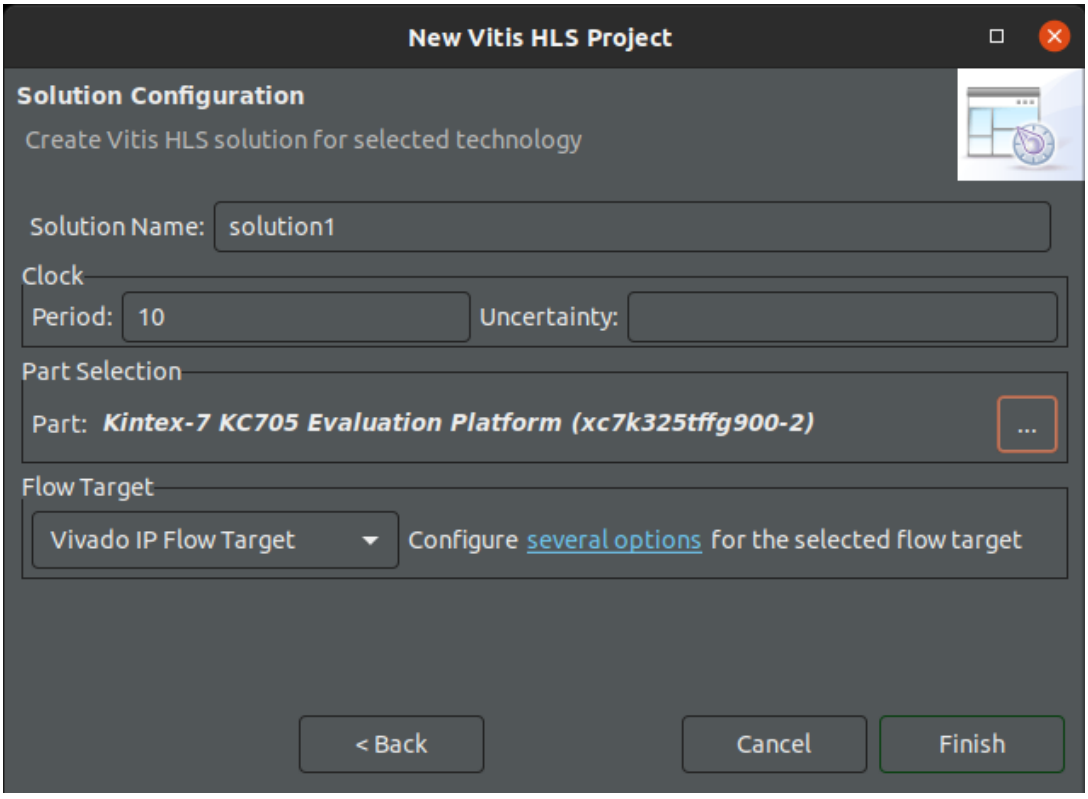


6. Add testbench files:

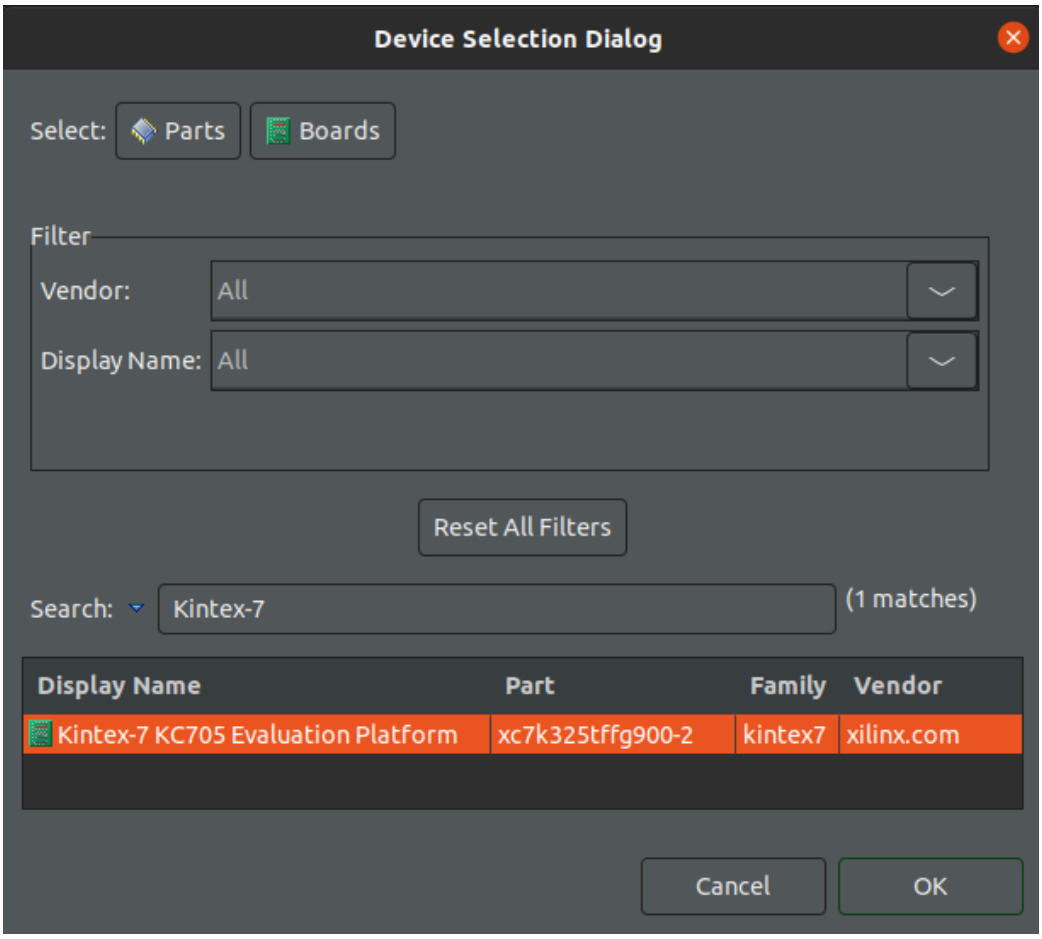


- `EventStream.dat`
- `host_hls.cpp`
- `peds.dat`
- `preprocess.h`

7. Clock Period: 10



8. Part Selection: Click to search by "Boards" then select the "Kintex-7 KC705 Evaluation Platform."



9. Finally, click "Finish." Your evaluation environment will launch.

Experiment workflow

We compare the reported performance of the seven different implementations of our data pre-processing pipeline listed in Table 1. The II (initiation interval in cycles) and resource utilization (BRAM, DSP, FF, and LUT) metrics are taken from the reported generated by Vitis HLS after C synthesis. The latency, in cycles, is taken from the report generated by Vitis HLS after C/RTL co-simulation. Therefore, the entire experimental workflow is performed in Vitis HLS. We did not analyze a deployment to the actual FPGA hardware platform.

The repository contains a copy of the testbench and kernel files for each implementation. For example:

- `1-preprocess.cpp` is the HLS code for the kernel corresponding to implementation **1. Functional Baseline**.
- `1-host_hls.cpp` is the C++ code for the testbench corresponding to **1. Functional Baseline**.
- `1-preprocess.h` is the common C++ header file for the two.

We also provide a single header file, `filepath.h`, that defines paths to the input data file and

output result file used by the simulation. Modify the following line of the file to reflect the actual path to the repository on your evaluation system:

```
const std::string path_to_repo = "/home/yourusername/adapt_fpga/";
```

Thus, the following steps can be used to evaluate one of our implementations (we use **1. Functional Baseline** as an example):

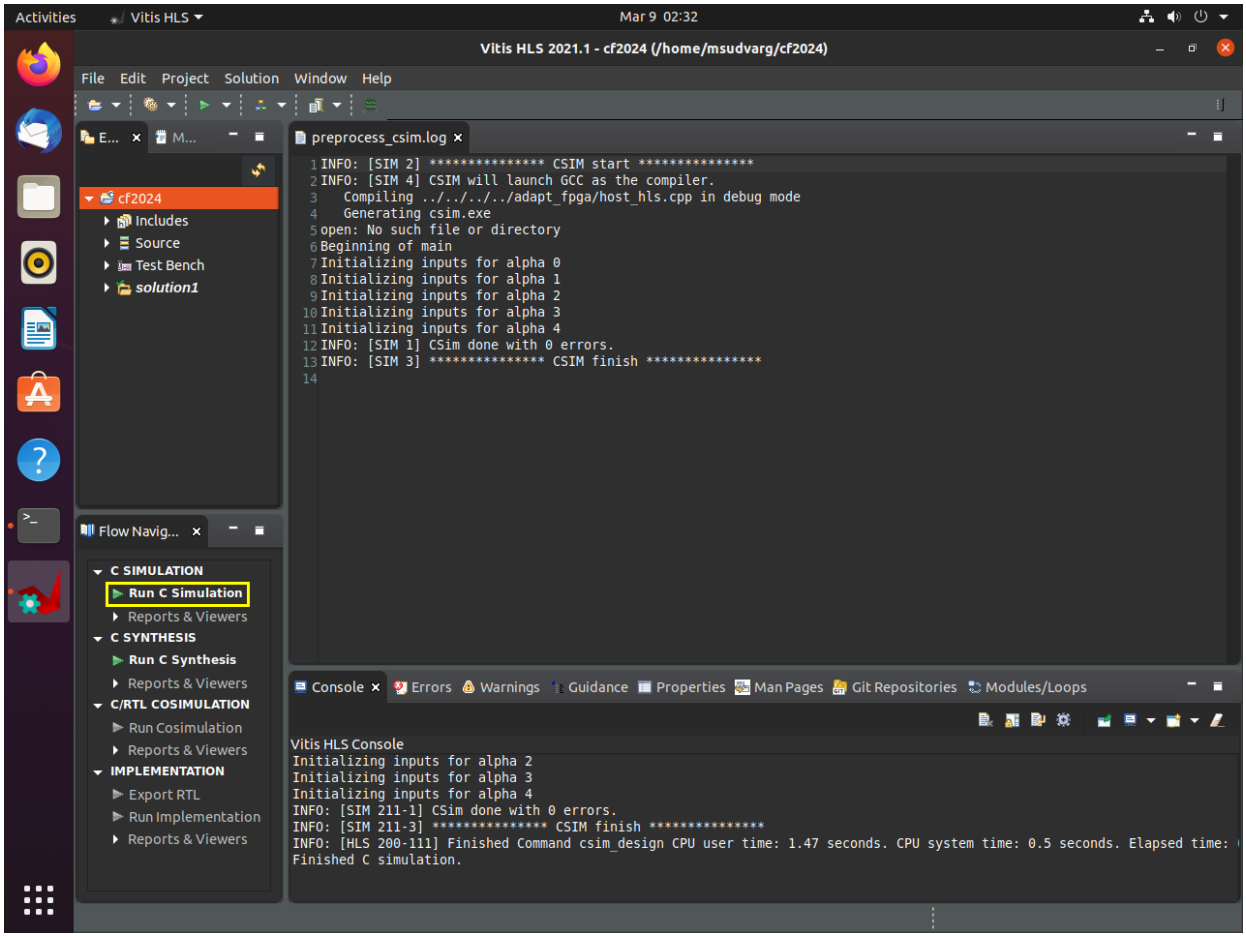
1. Modify `filepath.h` as appropriate.
2. Launch Vitis HLS: `vitis_hls`
3. Copy corresponding project files into project:
   ```
   cp 1-preprocess.cpp preprocess.cpp
   cp 1-preprocess.h preprocess.h
   cp 1-host_hls.cpp host_hls.cpp
   ```
4. Run C Simulation (typically takes < 1 minute).
5. Run C Synthesis (typically takes 3–5 minutes).
6. Run C/RTL Cosimulation (typically takes 5–10 minutes).

Evaluation and expected results

C Simulation

This process simply compiles the C++ testbench and kernel code into a software binary and runs it. At the end of a run of C Simulation, a new file, `output.txt`, will be produced in the repository directory pointed to by `filepath.h`. It contains debugging output information from the pre-processing pipeline. The correct output can be found at `output_five_centroiding.txt`. To compare the files to verify correct simulation, run:

```
diff output.txt output_five_centroiding.txt
```

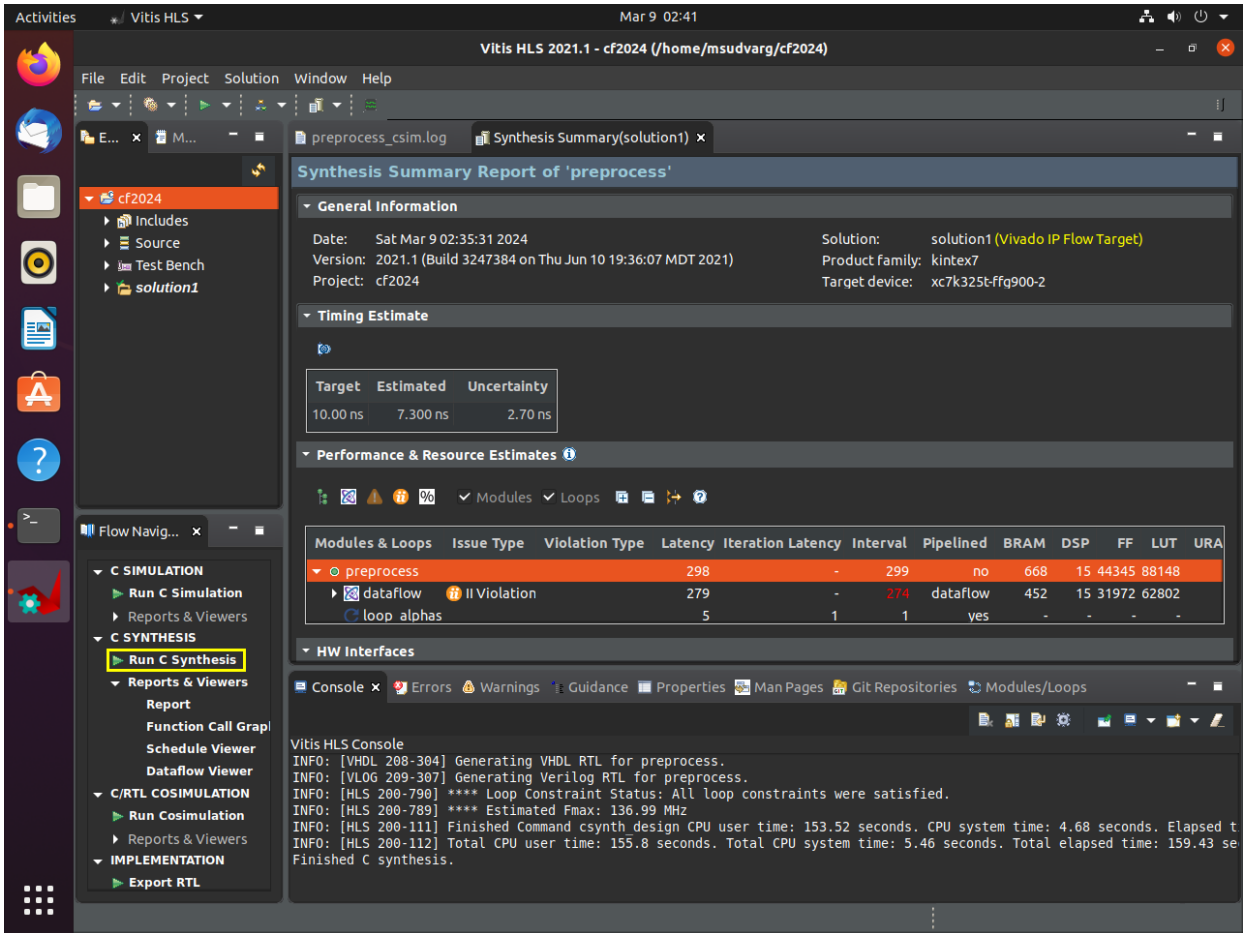


C Synthesis

This step generates and analyzes the FPGA kernel. At the end of a run of C Synthesis, Vitis HLS will display a report called "Synthesis Summary." This contains the II, BRAM, DSP, FF, and LUT values shown in Table 1.

The report file itself is generated in `/(solution path)/syn/report/csynth.rpt` where (`solution path`) is the path to the Vitis HLS solution that you specified when creating the project. The file has additional details, including the resource utilization percentage values reported in Table 1.

We have provided the corresponding report for each implementation in the repository. For example, `1-csynth.rpt` corresponds to implementation **1. Functional Baseline**.



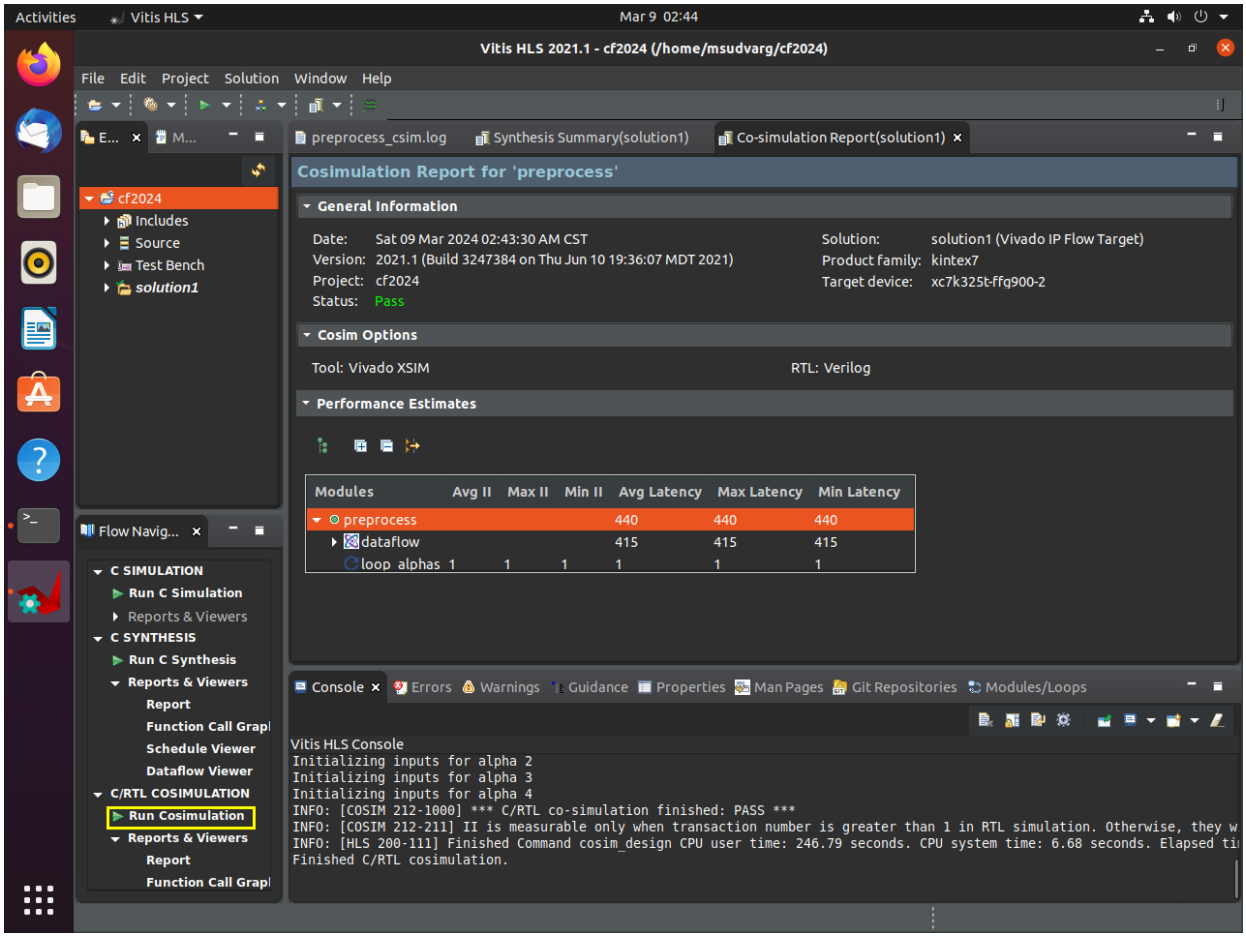
C/RTL Co-Simulation

This step compiles a software binary for the testbench and runs it simultaneously with the synthesized kernel in an emulation environment. At the end of a run of C/RTL Co-Simulation, Vitis HLS will display a report called "Co-Simulation Report." This contains the latency value shown in Table 1.

The report file itself is generated in `/(solution path)/sim/report/preprocess_cosim.rpt` where (`solution path`) is the path to the Vitis HLS solution that you specified when creating the project.

We have provided the corresponding report for each implementation in the repository. For example, `1-preprocess_cosim.rpt` corresponds to implementation **1. Functional Baseline**.

This step also creates an `output.txt` file similarly to C Synthesis. It can also be compared to `output_five_centroiding.txt` . As we have noted in the paper, the output produced by the **0. Naïve** implementation is incorrect. The output from that implementation is instead expected to match the file `output_naive.txt` .


Activities
Vitis HLS
Mar 9 02:44
Vitis HLS 2021.1 - cf2024 (/home/msudvarg/cf2024)
File Edit Project Solution Window Help
cf2024
Includes
Source
Test Bench
solution1
preprocess_csim.log
Synthesis Summary(solution1)
Co-simulation Report(solution1)
Cosimulation Report for 'preprocess'
General Information
Date: Sat 09 Mar 2024 02:43:30 AM CST
Version: 2021.1 (Build 3247384 on Thu Jun 10 19:36:07 MDT 2021)
Project: cf2024
Status: Pass
Solution: solution1 (Vivado IP Flow Target)
Product family: kintex7
Target device: xc7k325t-ffg900-2
Cosim Options
Tool: Vivado XSIM
RTL: Verilog
Performance Estimates
Modules Avg II Max II Min II Avg Latency Max Latency Min Latency
preprocess 440 440 440
dataflow 415 415 415
loop_alphas 1 1 1 1 1 1
Flow Navig...
C SIMULATION
Run C Simulation
Reports & Viewers
C SYNTHESIS
Run C Synthesis
Reports & Viewers
Report
Function Call Grapl
Schedule Viewer
Dataflow Viewer
C/RTL COSIMULATION
Run Cosimulation
Reports & Viewers
Report
Function Call Grapl
Console
Errors
Warnings
Guidance
Properties
Man Pages
Git Repositories
Modules/Loops
Vitis HLS Console
Initializing inputs for alpha 2
Initializing inputs for alpha 3
Initializing inputs for alpha 4
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 w
INFO: [HLS 200-111] Finished Command cosim_design CPU user time: 246.79 seconds. CPU system time: 6.68 seconds. Elapsed ti
Finished C/RTL cosimulation.