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. doi: 10.1145/3649153.3649209

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.

Release Title

FPGA Kernels for Front-End ASIC Pre-Processing on ADAPT V1

Author Information

Lead Author and Primary Contact:
Marion Baumli Sudvarg
Dept. of Computer Science and Engineering
Washington University in St. Louis
https://ror.org/01yc7t268 (https://ror.org/01yc7t268)
msudvarg@wustl.edu (mailto:msudvarg@wustl.edu)
OrcID: 0000-0003-2318-7763

Principal Investigator:
James H. Buckley
Dept. of Physics
Washington University in St. Louis
https://ror.org/01yc7t268 (https://ror.org/01yc7t268)
buckley@wustl.edu (mailto:buckley@wustl.edu)
OrcID: 0000-0001-6391-9661

Co-Principal Investigator:
Roger D. Chamberlain
Dept. of Computer Science and Engineering
Washington University in St. Louis
https://ror.org/01yc7t268 (https://ror.org/01yc7t268)
roger@wustl.edu (mailto:roger@wustl.edu)
OrcID: 0000-0002-7207-6106

Co-Principal Investigator:
Jeremy Buhler
Dept. of Computer Science and Engineering
Washington University in St. Louis
https://ror.org/01yc7t268 (https://ror.org/01yc7t268)
jbuhler@wustl.edu (mailto:jbuhler@wustl.edu)
OrcID: 0000-0002-4159-4226

Co-Author:
Chenfeng Zhao
Dept. of Computer Science and Engineering
Washington University in St. Louis
https://ror.org/01yc7t268 (https://ror.org/01yc7t268)
chenfeng.zhao@wustl.edu (mailto:chenfeng.zhao@wustl.edu)
OrcID: 0000-0001-9952-0628

Co-Author:
Ye Htet
Dept. of Computer Science and Engineering
Washington University in St. Louis
https://ror.org/01yc7t268 (https://ror.org/01yc7t268)
htet.ye@wustl.edu (mailto:htet.ye@wustl.edu)
OrcID: 0009-0006-8477-1460

Co-Author:

Meagan Konst
Dept. of Computer Science and Engineering
Washington University in St. Louis
https://ror.org/01yc7t268 (https://ror.org/01yc7t268)
meagan.konst@wustl.edu (mailto:meagan.konst@wustl.edu)
OrcID: 0009-0001-1508-8143

Co-Author:
Thomas Lang
Dept. of Computer Science and Engineering
Washington University in St. Louis
https://ror.org/01yc7t268 (https://ror.org/01yc7t268)
lang.thomas@wustl.edu (mailto:lang.thomas@wustl.edu)
OrcID: 0009-0005-5731-3969

Co-Author:
Nick Song
Dept. of Computer Science and Engineering
Washington University in St. Louis
https://ror.org/01yc7t268 (https://ror.org/01yc7t268)
qinzhounick@wustl.edu (mailto:qinzhounick@wustl.edu)
OrcID: 0009-0006-1589-693X

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

This artifact is available on GitHub and an archival repository through the Washington University in St. Louis libraries.

Archival repository links:

https://doi.org/10.7936/6RXS-103658 (https://doi.org/10.7936/6RXS-103658)

https://data.library.wustl.edu/record/103658 (https://data.library.wustl.edu/record/103658)

Retrieve from GitHub:

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.

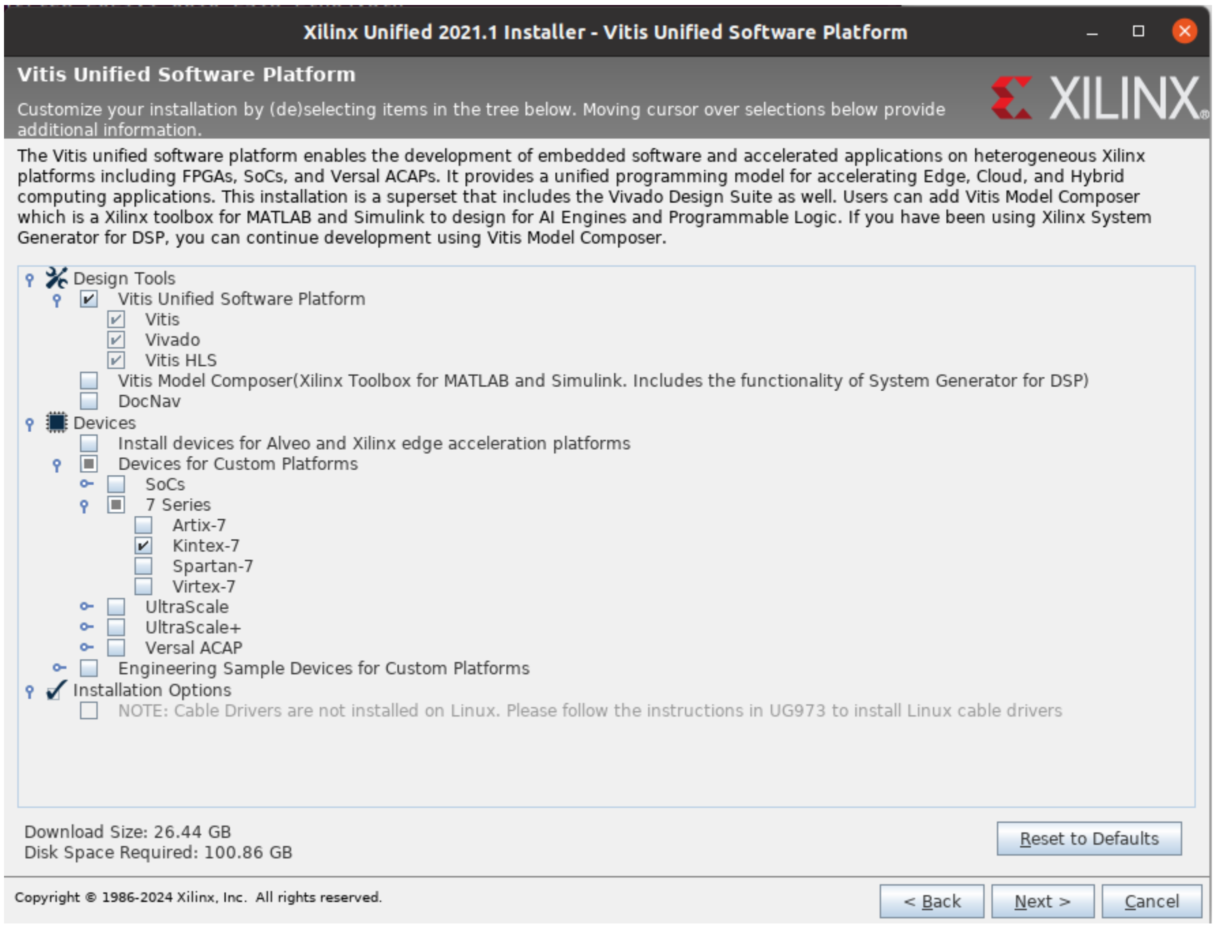
Installation

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.


Xilinx Unified 2021.1 Installer - Vitis Unified Software Platform
XILINX
Reset to Defaults
< Back
Next >
Cancel

Vitis Unified Software Platform

Customize your installation by (de)selecting items in the tree below. Moving cursor over selections below provide additional information.

The Vitis unified software platform enables the development of embedded software and accelerated applications on heterogeneous Xilinx platforms including FPGAs, SoCs, and Versal ACAPs. It provides a unified programming model for accelerating Edge, Cloud, and Hybrid computing applications. This installation is a superset that includes the Vivado Design Suite as well. Users can add Vitis Model Composer which is a Xilinx toolbox for MATLAB and Simulink to design for AI Engines and Programmable Logic. If you have been using Xilinx System Generator for DSP, you can continue development using Vitis Model Composer.

- Design Tools
  - [✓] Vitis Unified Software Platform
    - [✓] Vitis
    - [✓] Vivado
    - [✓] Vitis HLS
  - [ ] Vitis Model Composer(Xilinx Toolbox for MATLAB and Simulink. Includes the functionality of System Generator for DSP)
  - [ ] DocNav
- Devices
  - [ ] Install devices for Alveo and Xilinx edge acceleration platforms
  - [■] Devices for Custom Platforms
    - [ ] SoCs
    - [■] 7 Series
      - [ ] Artix-7
      - [✓] Kintex-7
      - [ ] Spartan-7
      - [ ] Virtex-7
    - [ ] UltraScale
    - [ ] UltraScale+
    - [ ] Versal ACAP
  - [ ] Engineering Sample Devices for Custom Platforms
- ✓ Installation Options
  - [ ] NOTE: Cable Drivers are not installed on Linux. Please follow the instructions in UG973 to install Linux cable drivers

Download Size: 26.44 GB
Disk Space Required: 100.86 GB

Copyright © 1986-2024 Xilinx, Inc. All rights reserved.

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&-'%r1%&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`

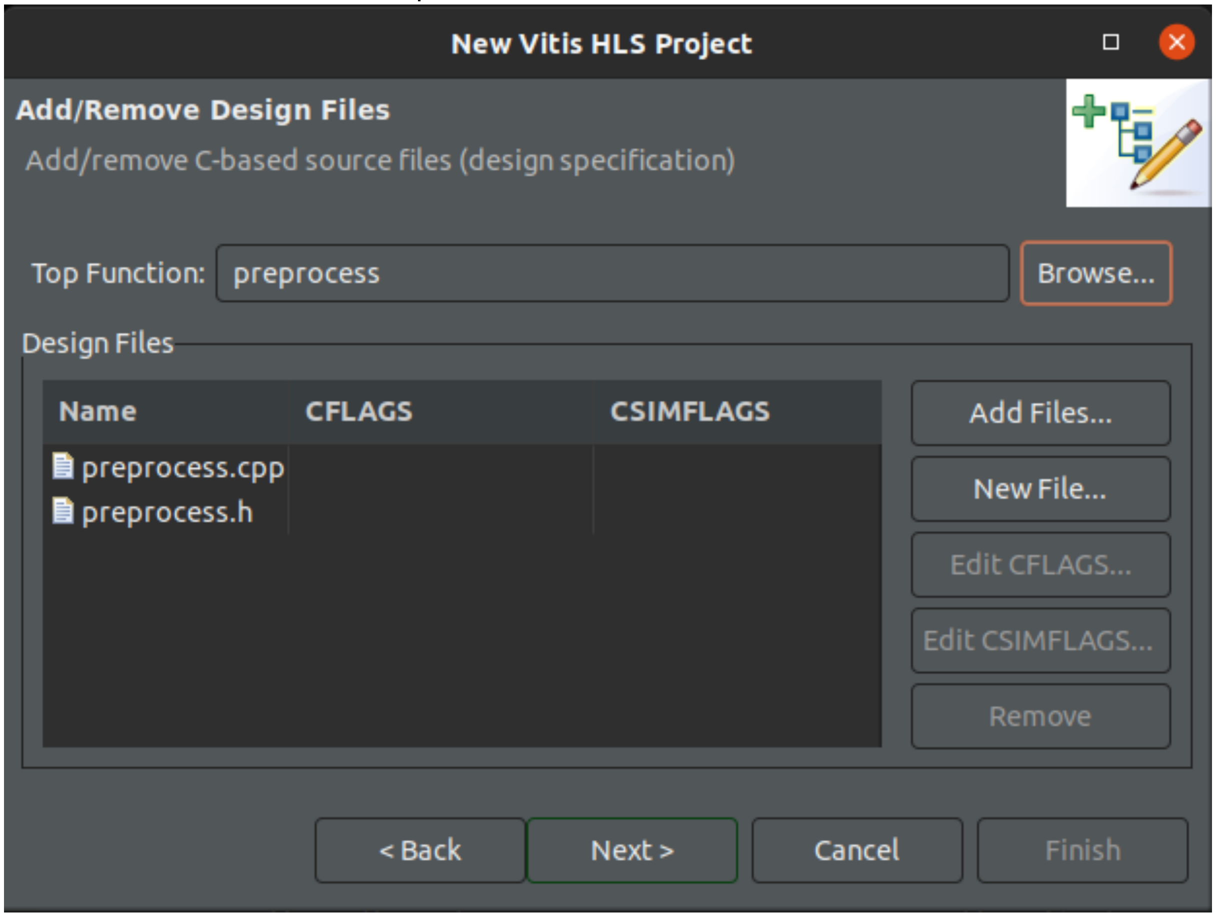
2. Click Create Project

3. Enter a project name and location

4. Add design files:

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

5. Select `preprocess` as the top function.



New Vitis HLS Project

Add/Remove Design Files

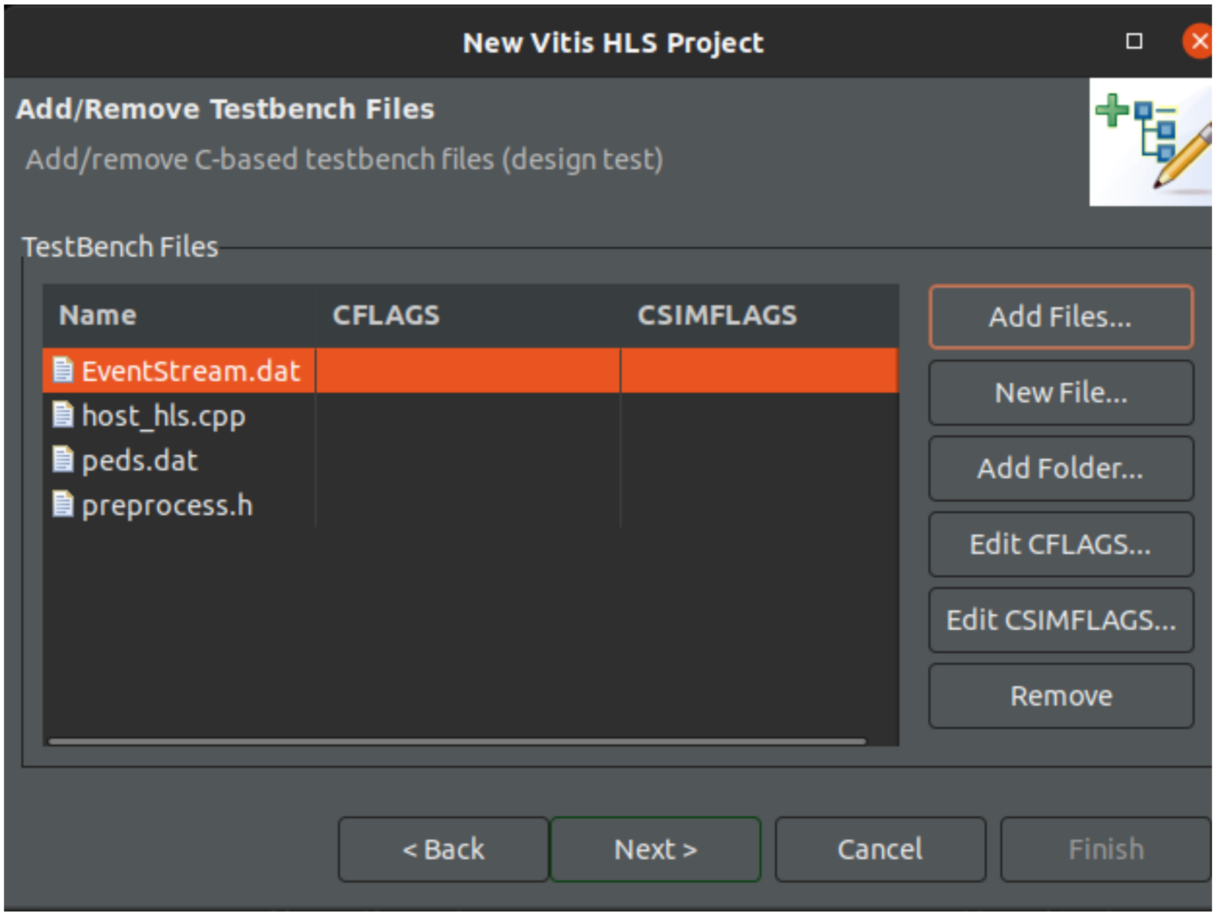
Add/remove C-based source files (design specification)

Top Function: preprocess

Design Files

Name	CFLAGS	CSIMFLAGS
preprocess.cpp		
preprocess.h		

6. Add testbench files:


New Vitis HLS Project

Add/Remove Testbench Files

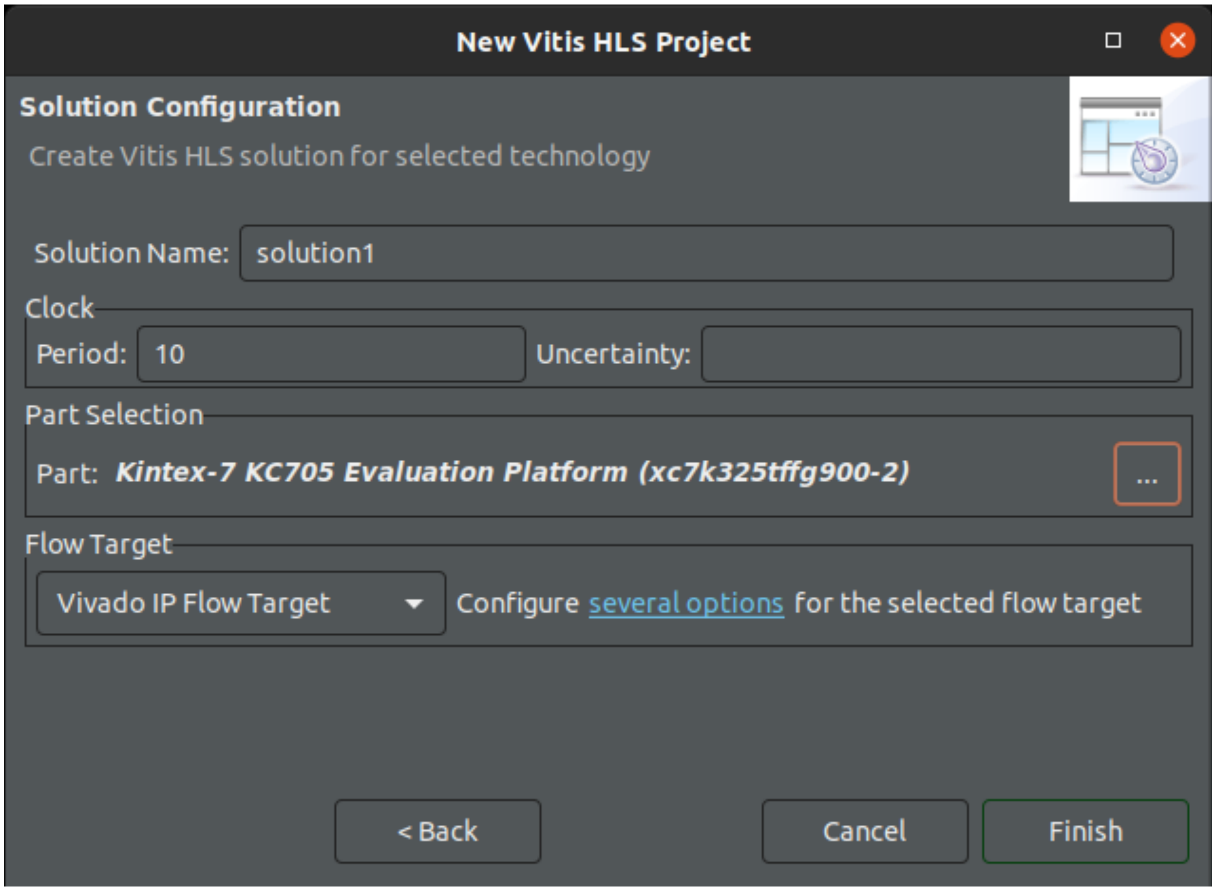
Add/remove C-based testbench files (design test)

TestBench Files

Name	CFLAGS	CSIMFLAGS
EventStream.dat		
host_hls.cpp		
peds.dat		
preprocess.h		

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

7. Clock Period: 10



New Vitis HLS Project

Solution Configuration

Create Vitis HLS solution for selected technology

Solution Name: solution1

Clock

Period: 10 Uncertainty:

Part Selection

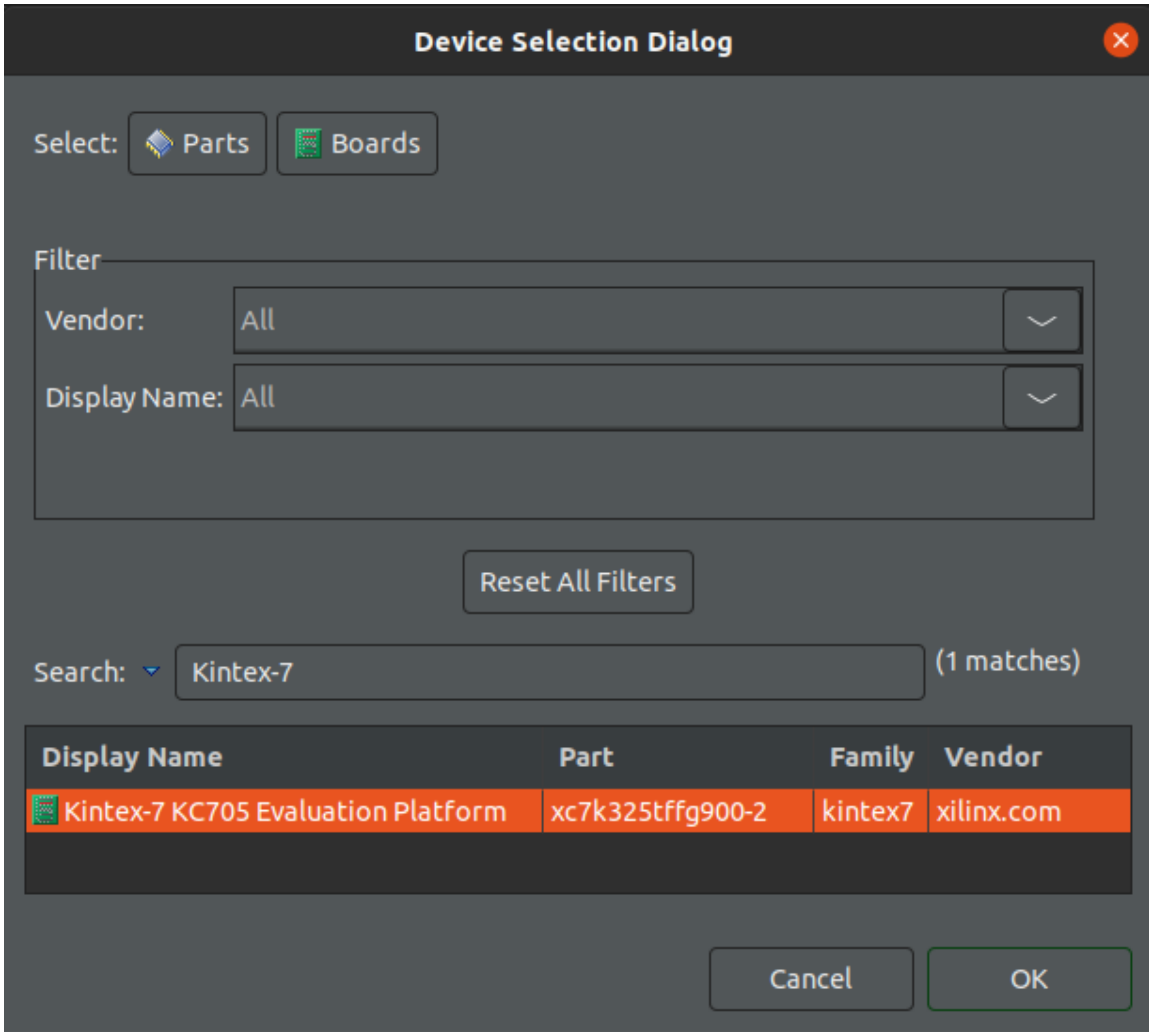
Part: ***Kintex-7 KC705 Evaluation Platform (xc7k325tffg900-2)***

Flow Target

Vivado IP Flow Target

Configure several options for the selected flow target

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


Device Selection Dialog

Select: Parts Boards

Filter

Vendor: All

Display Name: All

Reset All Filters

Search: Kintex-7 (1 matches)

Display Name	Part	Family	Vendor
Kintex-7 KC705 Evaluation Platform	xc7k325tffg900-2	kintex7	xilinx.com

Cancel OK

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):

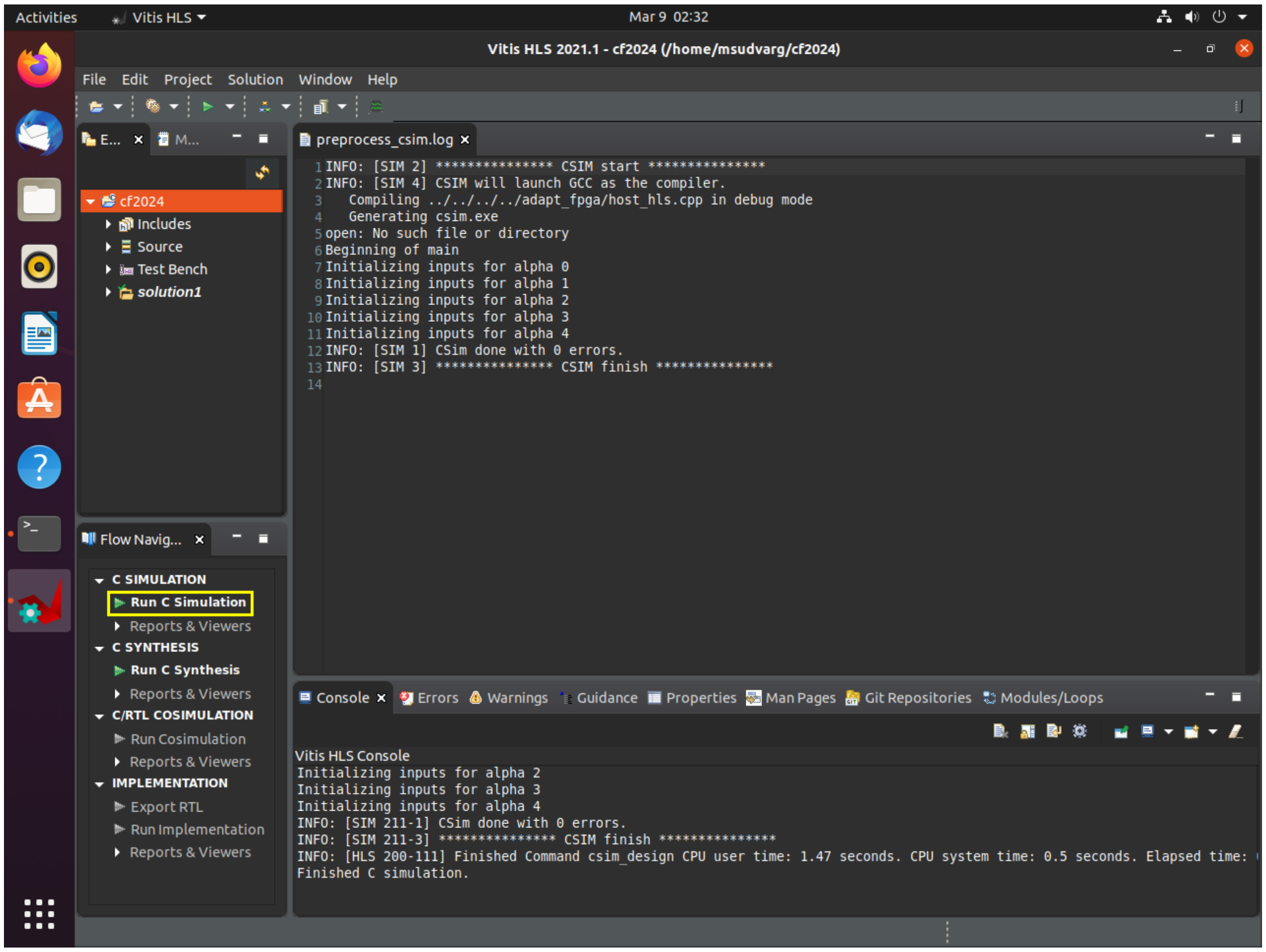
1. Modify `filepath.h` as appropriate.
2. Launch Vitis HLS: `vitis_hls`
3. Copy corresponding project files into project:
   ```
   cp 1-preprocess.cpp preprocess.cpp
   cp 1-preprocess.h preprocess.h
   cp 1-host_hls.cpp host_hls.cpp
   ```
4. Run C Simulation (typically takes < 1 minute).
5. Run C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



```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling ../../../../adapt_fpga/host_hls.cpp in debug mode
   Generating csim.exe
open: No such file or directory
Beginning of main
Initializing inputs for alpha 0
Initializing inputs for alpha 1
Initializing inputs for alpha 2
Initializing inputs for alpha 3
Initializing inputs for alpha 4
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************
```

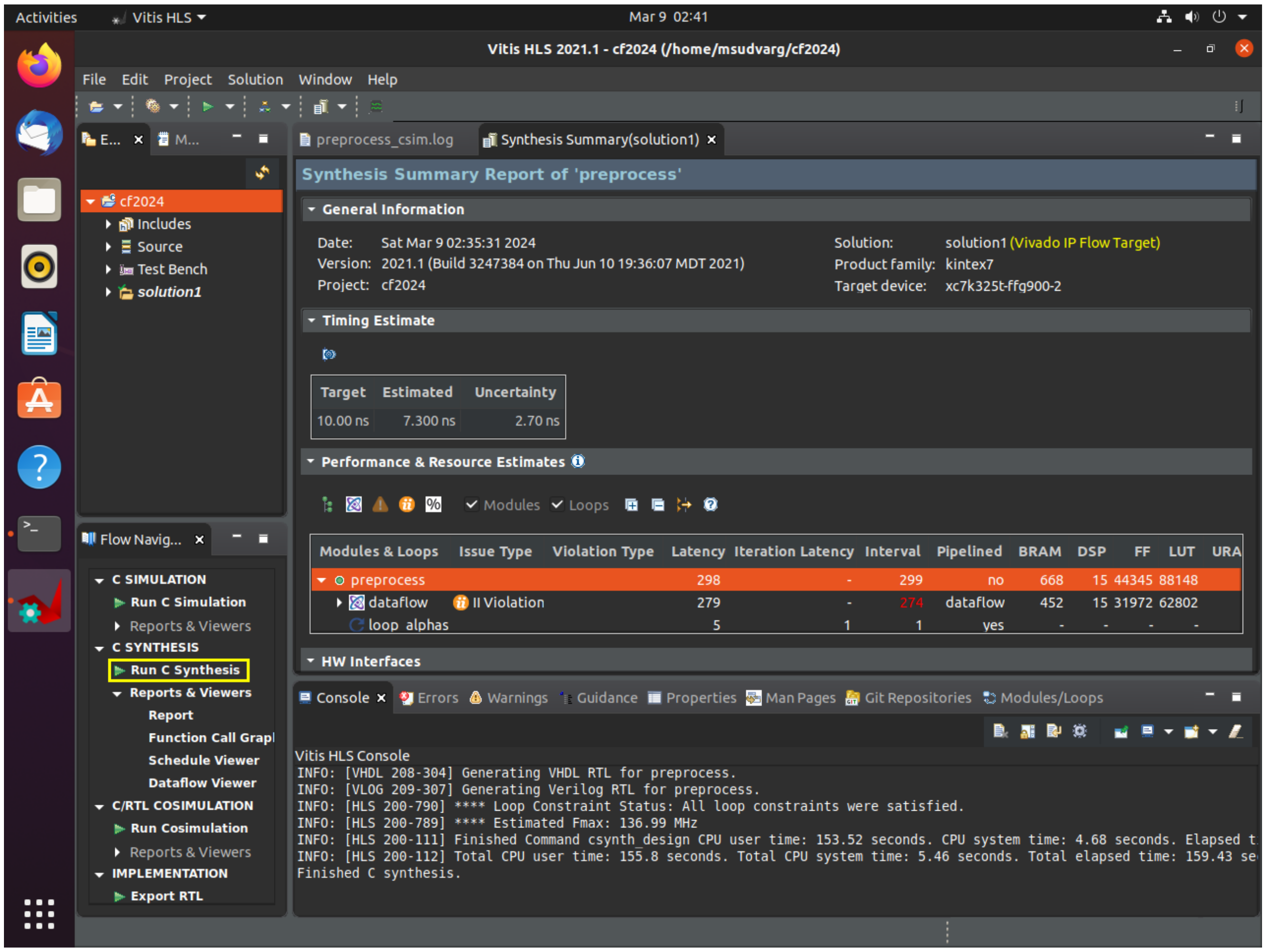
```
Vitis HLS Console
Initializing inputs for alpha 2
Initializing inputs for alpha 3
Initializing inputs for alpha 4
INFO: [SIM 211-1] CSim done with 0 errors.
INFO: [SIM 211-3] *************** CSIM finish ***************
INFO: [HLS 200-111] Finished Command csim_design CPU user time: 1.47 seconds. CPU system time: 0.5 seconds. Elapsed time:
Finished C simulation.
```

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**.



Synthesis Summary Report of 'preprocess'

General Information

Date:	Sat Mar 9 02:35:31 2024	Solution:	solution1 (Vivado IP Flow Target)
Version:	2021.1 (Build 3247384 on Thu Jun 10 19:36:07 MDT 2021)	Product family:	kintex7
Project:	cf2024	Target device:	xc7k325t-ffg900-2

Timing Estimate

Target	Estimated	Uncertainty
10.00 ns	7.300 ns	2.70 ns

Performance & Resource Estimates

Modules & Loops	Issue Type	Violation Type	Latency	Iteration Latency	Interval	Pipelined	BRAM	DSP	FF	LUT	URA
preprocess			298	-	299	no	668	15	44345	88148	
dataflow	II Violation		279	-	274	dataflow	452	15	31972	62802	
loop_alphas			5	1	1	yes	-	-	-	-	

Vitis HLS Console

```
INFO: [VHDL 208-304] Generating VHDL RTL for preprocess.
INFO: [VLOG 209-307] Generating Verilog RTL for preprocess.
INFO: [HLS 200-790] **** Loop Constraint Status: All loop constraints were satisfied.
INFO: [HLS 200-789] **** Estimated Fmax: 136.99 MHz
INFO: [HLS 200-111] Finished Command csynth_design CPU user time: 153.52 seconds. CPU system time: 4.68 seconds. Elapsed t
INFO: [HLS 200-112] Total CPU user time: 155.8 seconds. Total CPU system time: 5.46 seconds. Total elapsed time: 159.43 se
Finished C synthesis.
```

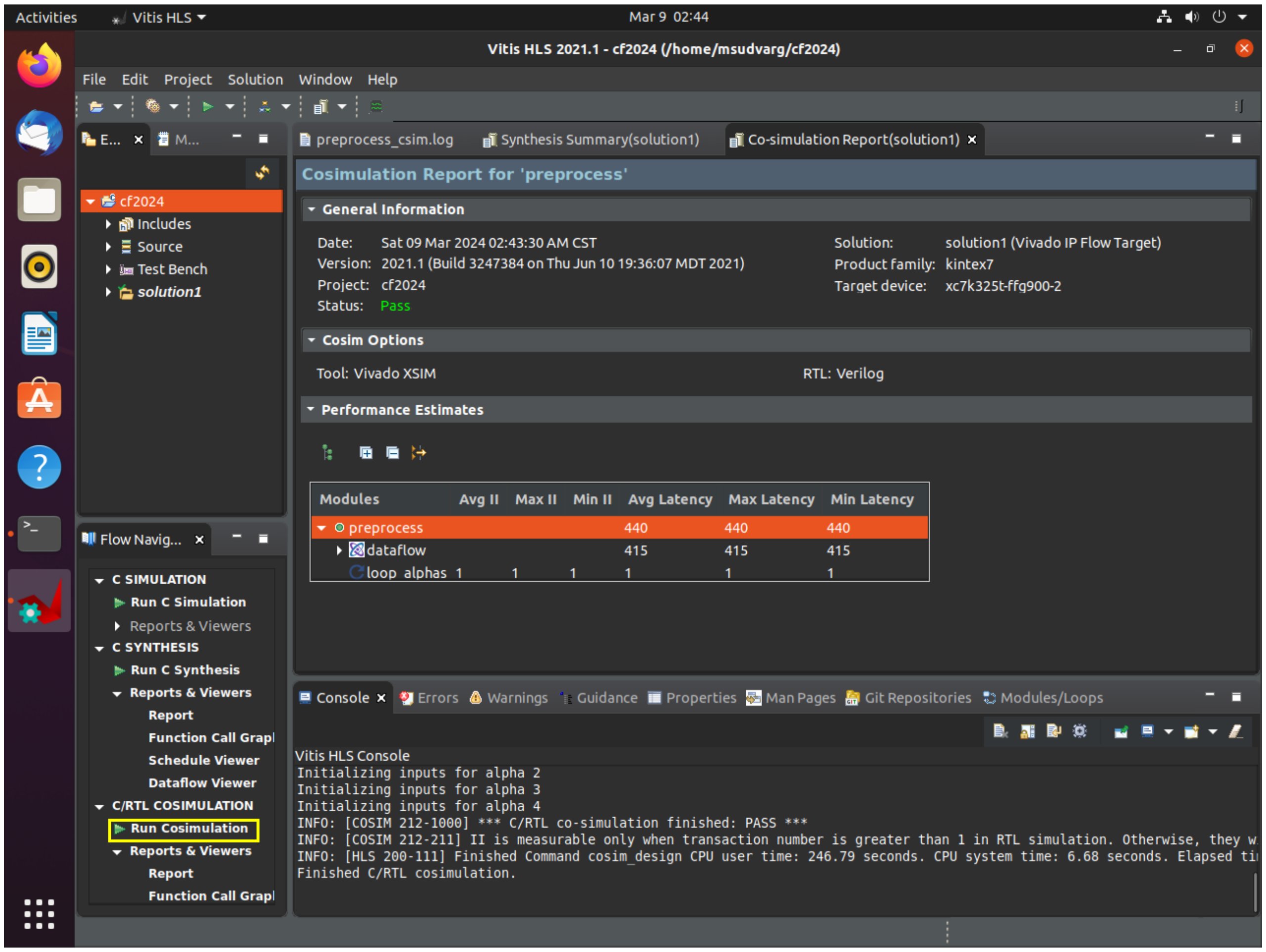
C/RTL Co-Simulation

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` .



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

```
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.
```

Additional Details

Dates

Software code and hardware specifications were developed during 2022 - 2024.

All result data were produced from December 2023 - January 2024.

Geographic Location

All experiments were performed at the Washington University Danforth Campus in St. Louis and St. Louis County, MO, USA.

Funding

This work was supported by NASA award 80NSSC21K1741 and NSF award CNS-1763503.

Contextual description of the data

This data contains the kernel and test bench source code for high-level synthesis (HLS) of ADAPT's data preprocessing and reduction algorithms, as well as measured FPGA timing latency and resource utilization numbers, to reproduce the results 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. doi: 10.1145/3649153.3649209

License

All source code is released under the MIT license.

The input data sets are in the public domain.

Original sources for input data

University of Hawaii Varner Lab's ALPHA ASIC simulator.

Data and File Overview

This is version 1 of the pre-processing software and FPGA kernel.

There are 45 files in total. These include:

1. Test Bench Files

   These include the C++ source code to implement host-side testing, e.g., sending data packets and calibration values to the FPGA, and retrieving output.

   Files include:

   - `n-host_hls.cpp` , where 'n' is a number 0-6. Each value of 'n' corresponds to one of the tested implementations (a summary of which are in Table 1 of the referenced publication).

   - `filepath.h` , which defines file paths for the test bench to read and write data.

2. Kernel Files

These include the C++ source code for high-level synthesis into FPGA kernels.

Files include:

- `n-preprocess.cpp` , which implements the kernel.

- `n-preprocess.h` , a header that includes definitions of common data formats, constant values, and function signatures between the kernel and testbench.

Again, 'n' is a number 0-6 corresponding to one of the tested implementations.

3. Reports

   These are reports on timing and resource utilization from the FPGA kernel synthesis tools.

   Files include:

   - `n-csynth.rpt` , which is the report from C synthesis and includes initiation interval and resource utilization/FPGA area numbers.

   - `n-preprocess_cosim.rpt` , which is the report from C/RTL co-simulation (FPGA kernel emulation in software) and includes more realistic latency numbers.

   Again, 'n' is a number 0-6 corresponding to one of the tested implementations.

   Though these have a '.rpt' extension, they are plain-text, human-readable files.

4. Input Data

   These are inputs to the FPGA produced by the
   University of Hawaii Varner Lab's ALPHA ASIC simulator.

   Files include:

   - `peds.dat` , a set of values representing the pedestal values for the ALPHA ASIC's analog memory cells.

   - `EventStream.dat` , a stream of bits representing a single data packet sent via serial interface from the ALPHA ASIC to the FPGA.

5. Output Data

   These are debug outputs against which the FPGA simulation output can be compared.

   Files include:

   - `output_five_centroiding.txt` , which is the correct output resulting from a run of the FPGA kernel for the supplied data packet.

   - `output_naive.txt` , which is incorrect output, but matches the output from the naive implementation discussed in the referenced paper.

6. Instructional Files

   `README.md` and `README.pdf` include instructions for reproducing the results in the referenced paper.

   The PDF includes more details, including screenshots.