School of Electrical & Computer Engineering - Aristotle University of Thessaloniki

Introduction to Vitis using Alveo U200 on PowerEdge R730 (2022.2)

Nikolaos Tampouratzis

GUI Flow

Introduction

This lab guides you through the steps involved in creating a Vitis project using Graphical User Interface (GUI). After creating the project you will run software and hardware emulations to verify the functionality of the design. You will also test the design in hardware on Alveo U200 board.

Description of example application

This lab uses a standard application template available in Vitis. It consists of a OpenCL host application and a C++ kernel. The C++ kernel is a simple vector addition. The elements of 2 vectors (A & B) are added together, and the result returned in a third array (C). The host application initializes the two input arrays, send data to the kernel, and read back the result.

Objectives

After completing this lab, you will learn to:

- Create a project using the Vitis GUI flow
- Run Software Emulation to verify the functionality of a design
- Run Hardware Emulation to verify the functionality of the generated hardware
- Build the system and test it in hardware
- Perform profile and application timeline analysis in hardware emulation

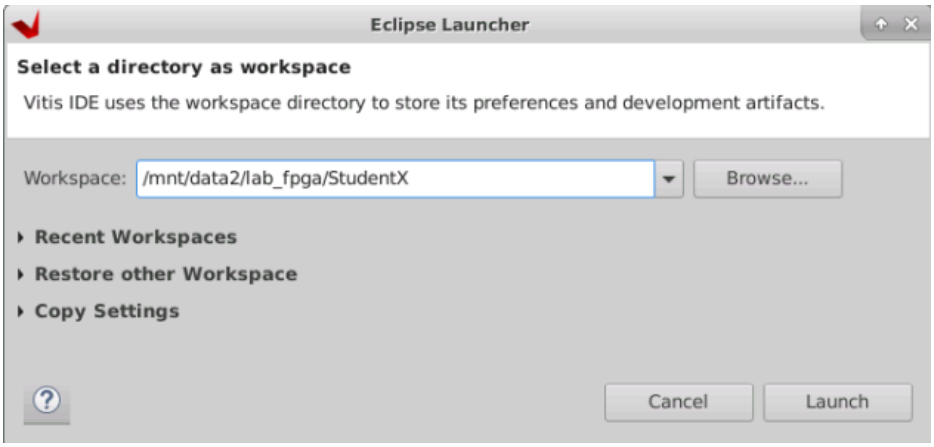
Steps

Create a Vitis Project

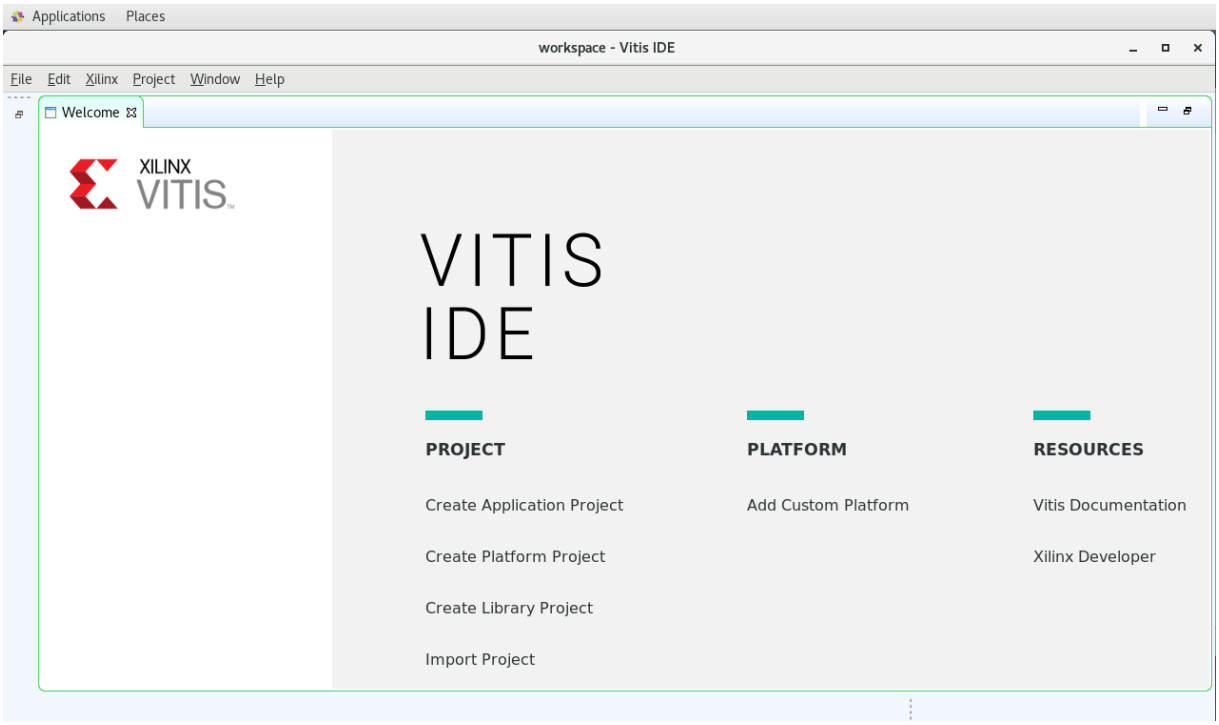
1. Invoke GUI by executing the following command:

```
vitis &
```

2. Set workspace to your team folder (where StudentX is your team number), as you can see in the following Figure and click **Launch. Please use the /mnt/data2/lab_fpga/StudentX (File -> Switch Workspace -> Other)**

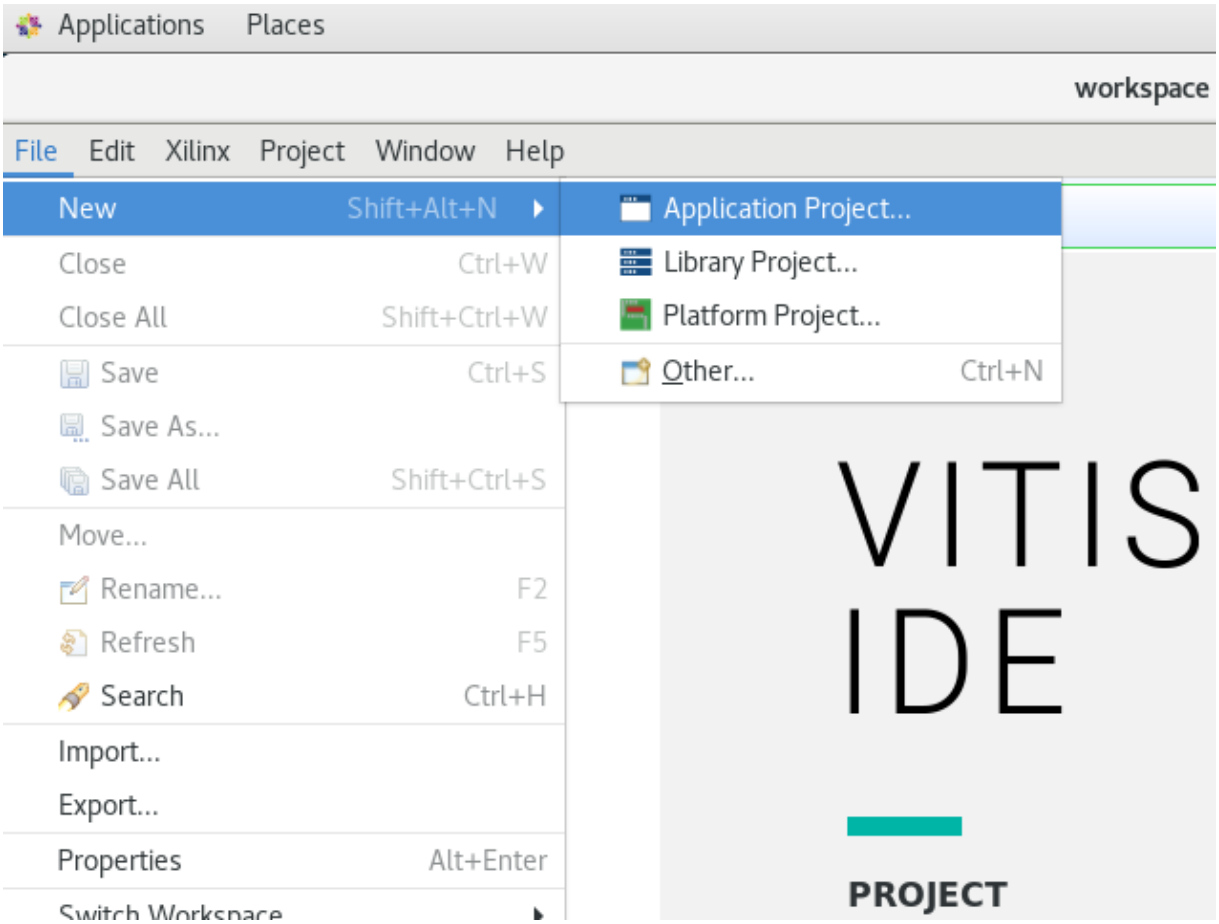


3. The Vitis IDE Welcome page will be displayed, if new a workspace is assigned



4. Create a new application project

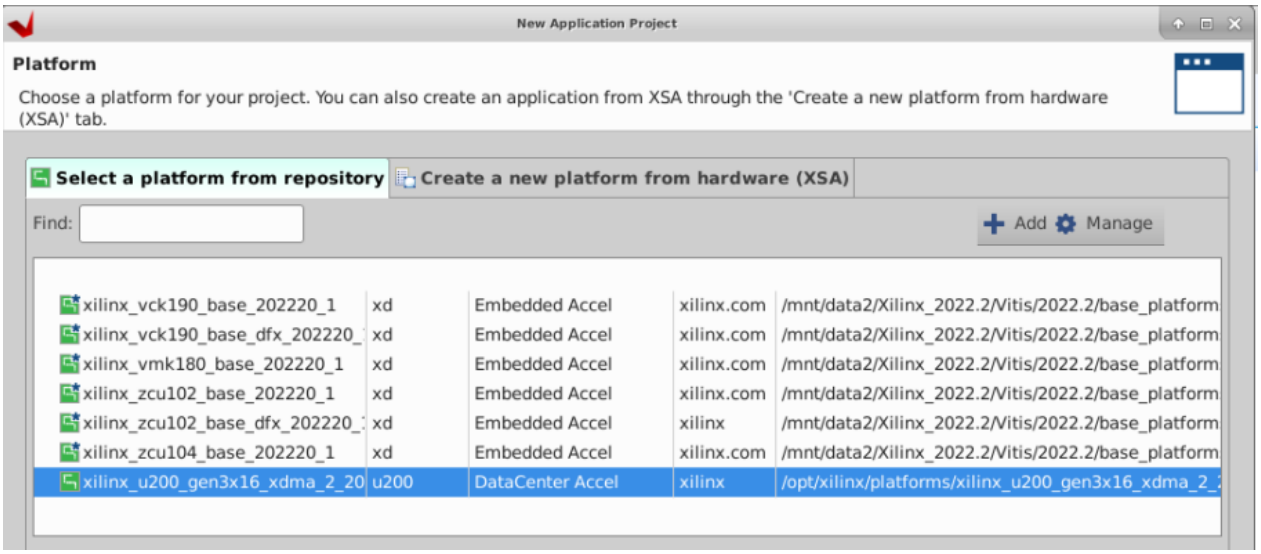
   Use `Create Application Project` from Welcome page, or use `File > New > Application Project` to create a new application.



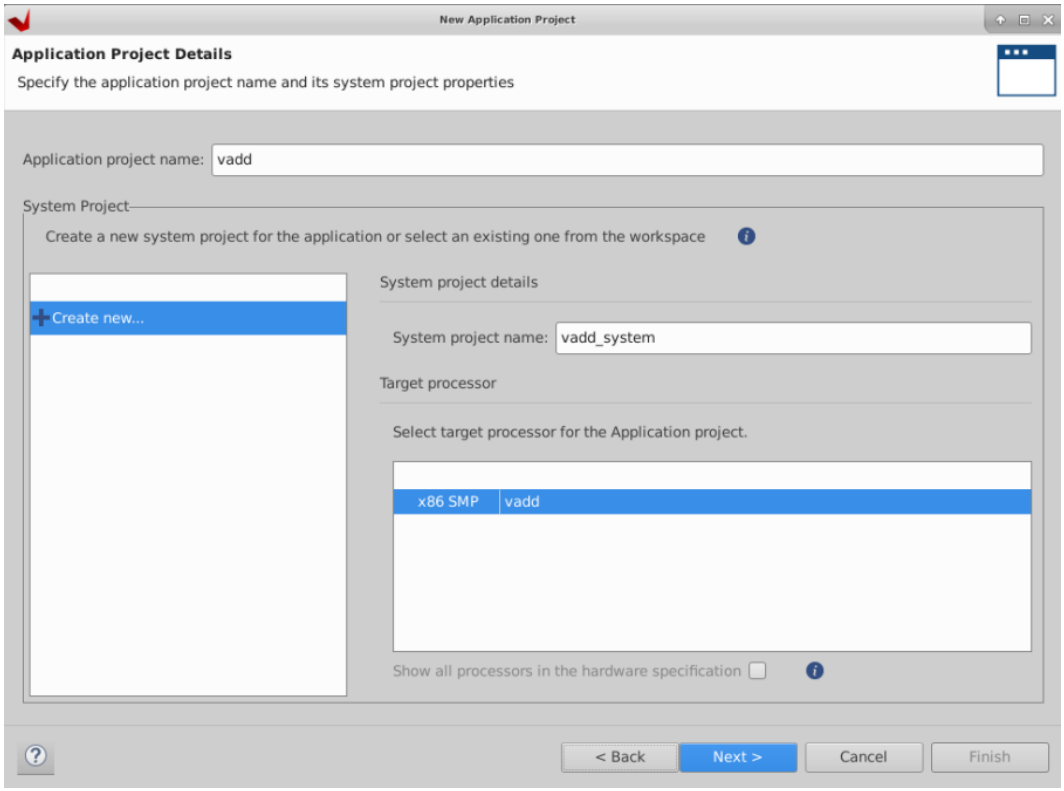
5. Close Welcome page, if it was opened

6. Select the platform `xilinx_u200_gen3x16_xdma...`, and click **OK**

7. You will see the platform entry, select it and click **Next>**

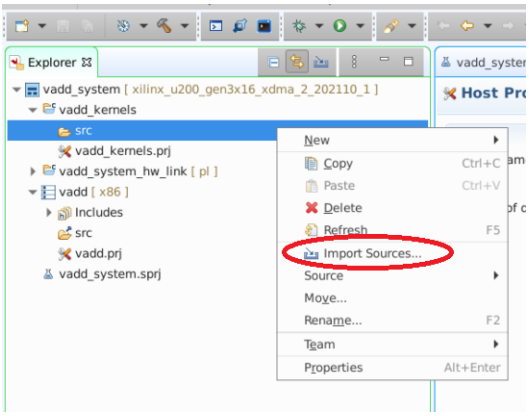


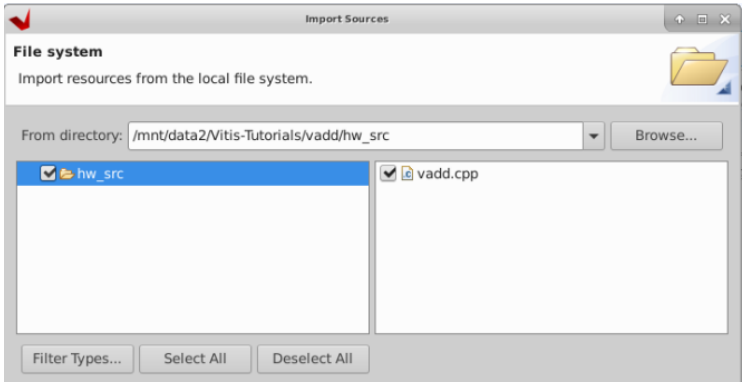
8. Give a name `vadd` and click **Next>**



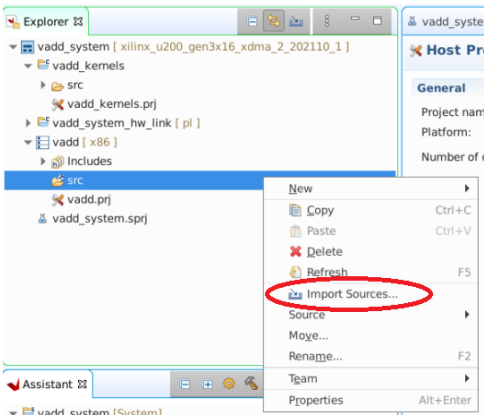
9. Select **Empty Application** as the template and click Finish. The project is generated.

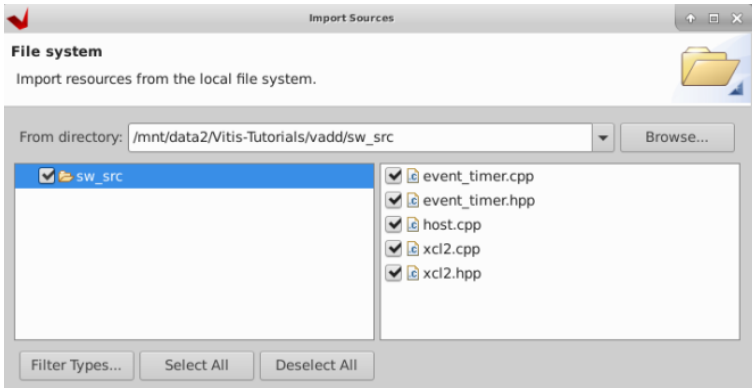
10. Import provided source files from vadd sub-directories.
    Select **vadd_kernels --> src** directory in Explorer view, right click and select Import Sources... in order to add the kernel Import `/mnt/data2/Vitis-Tutorials/vadd/hw_src/vadd.cpp` for **hardware accelerator** and press **Finish**



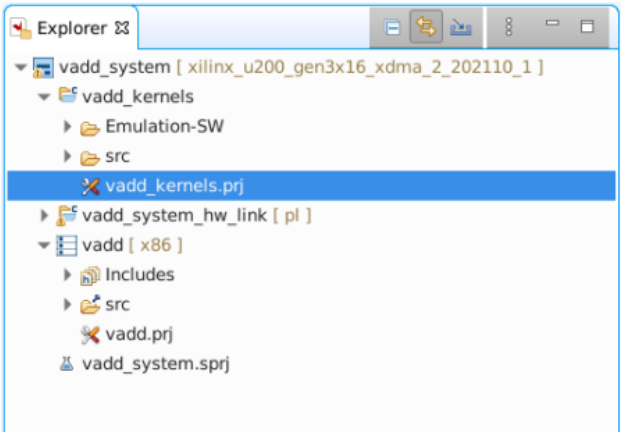


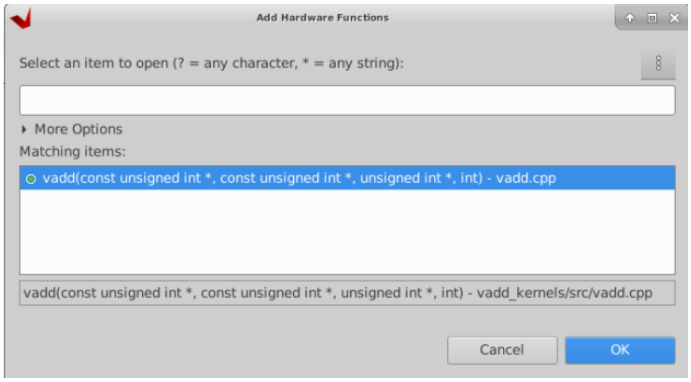
Select **vadd --> src** directory in Explorer view, right click and select Import Sources... in order to add the host files mport all `*.cpp` and `*.hpp` files in `/mnt/data2/Vitis-Tutorials/vadd/sw_src/` for **host code** application and press **Finish**





11. The project is generated. Double Click in vadd_kernels.prj and click on  in order to add the **vadd** as a *Hardware Function* (kernel).

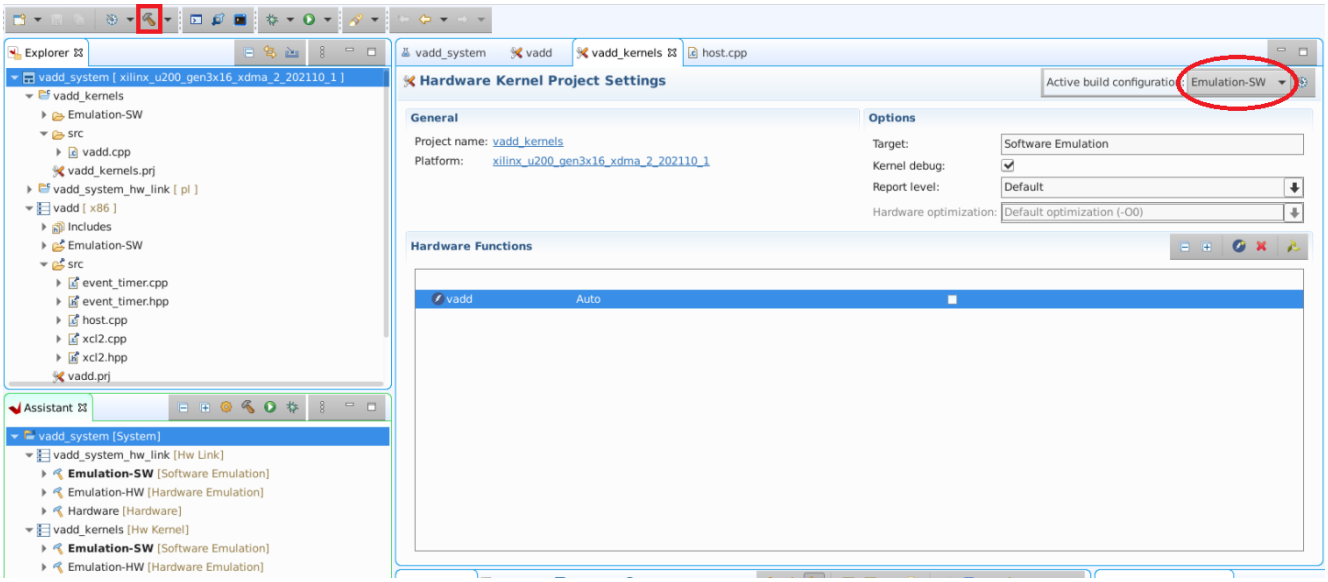




Build and Run Software Emulation

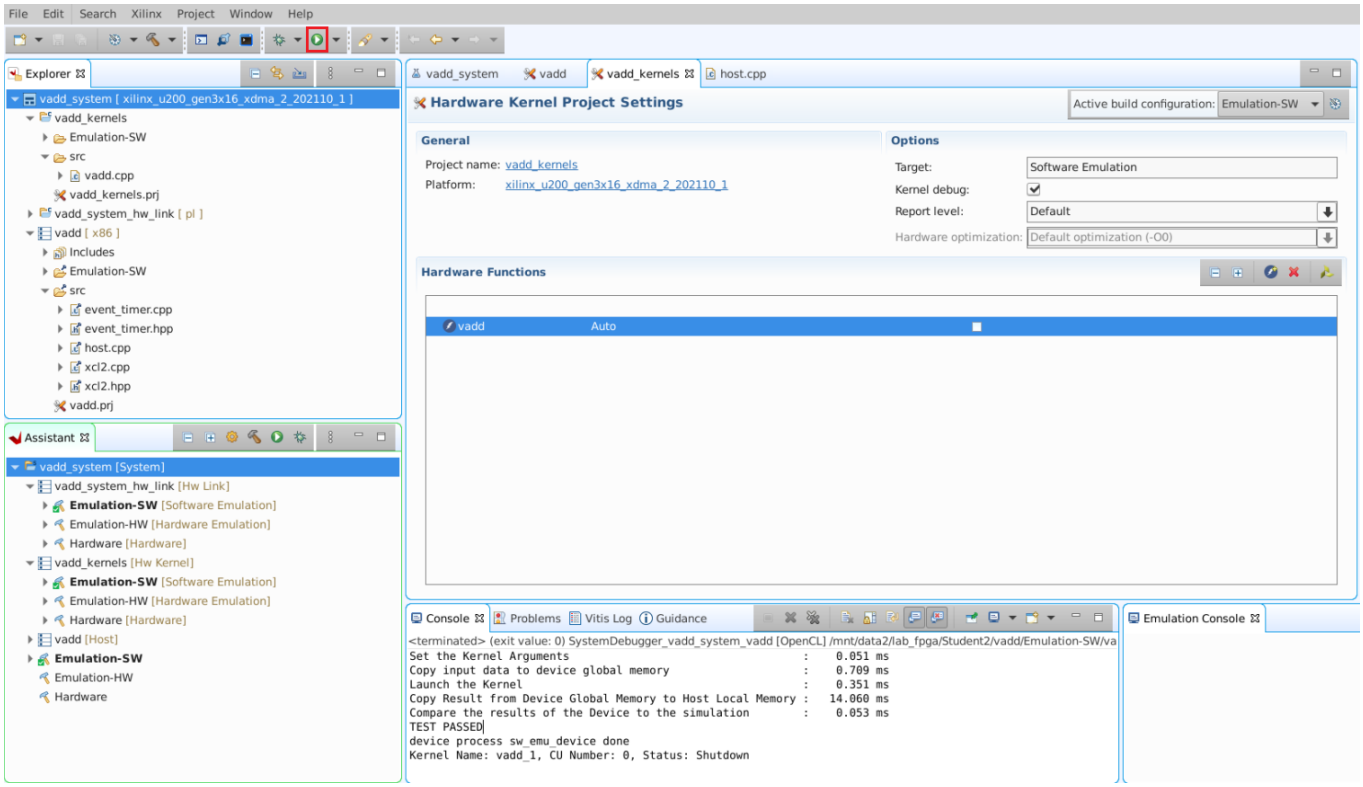
1. Set `Active build configuration:` to `Emulation-SW` on the upper right corner of *Project Editor* view

2. In the Explore view, select **vadd_system** and build the design by clicking the hammer button  on top buttons bar, or right click *vadd_system* and select Build Project.

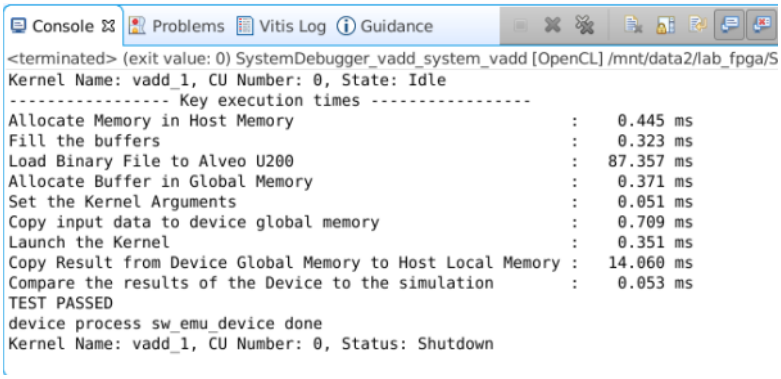


3. Run Software Emulation in GUI Mode

   To launch software emulation, select `vadd_system` either in the Assistant view or in the Explorer view and then click on the run button

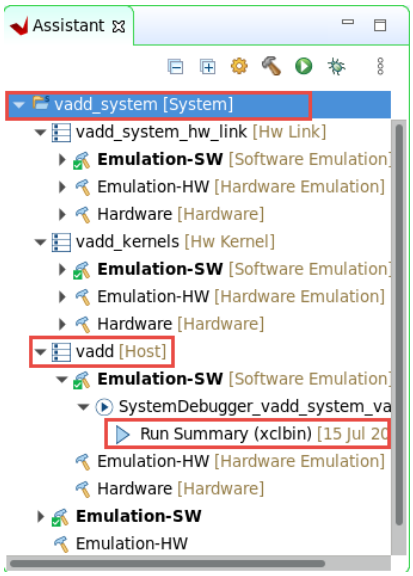


4. Observe the application is run and the output is displayed in the *Console* view



View Emulation Timeline

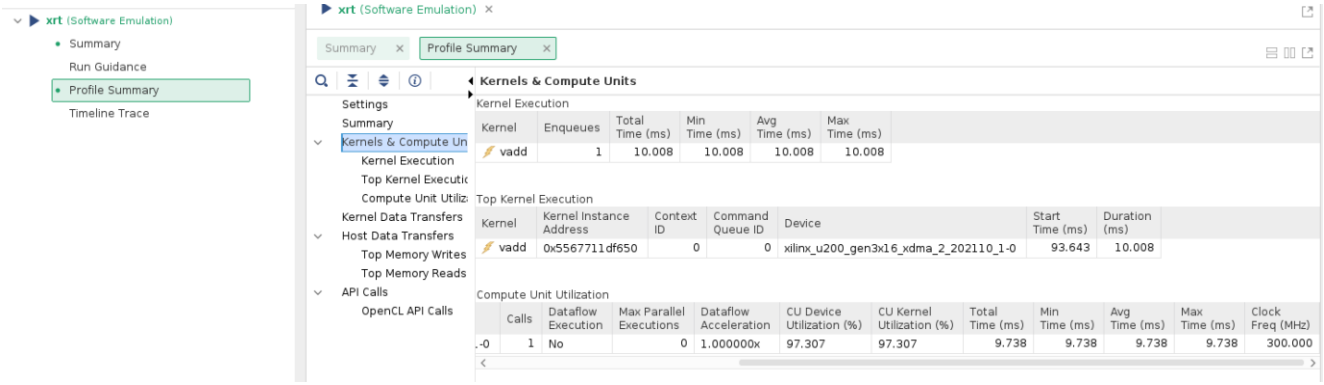
1. In the Assistant view, expand the following if necessary, and double-click vadd_system > vadd > Emulation-SW > SystemDebugger_vadd_system_vadd > Run Summary (xclbin) to open Vitis Analyzer



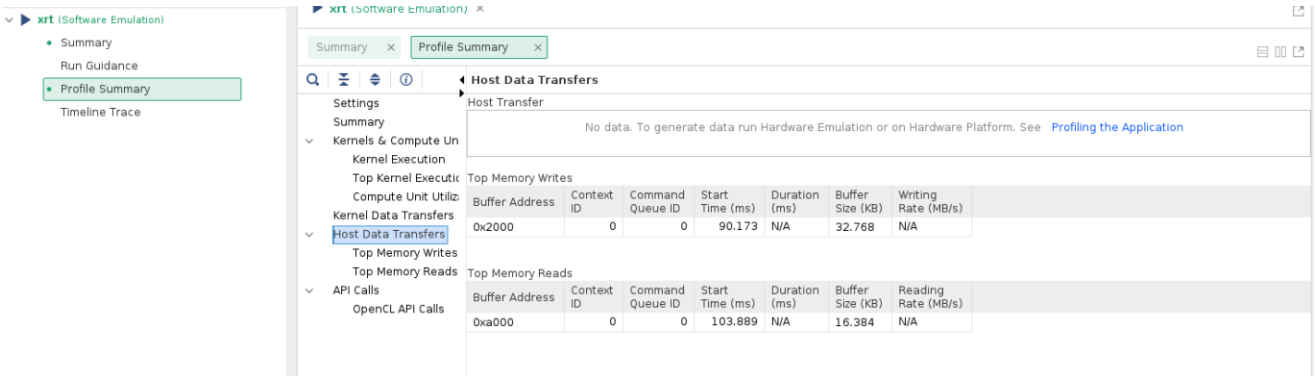
2. Vitis Analyzer shows **Summary**, **Run Guidance**, **Profile Summary** and **Timeline Trace** tabs on the left-hand side.

3. Select **Profile Summary** and then select **Kernels & Compute Units** to see kernel and compute units execution times.

   Notice the reported times.



4. Click **Host Data Transfer** to see read and write buffer sizes, buffer addresses, and the related execution parameters



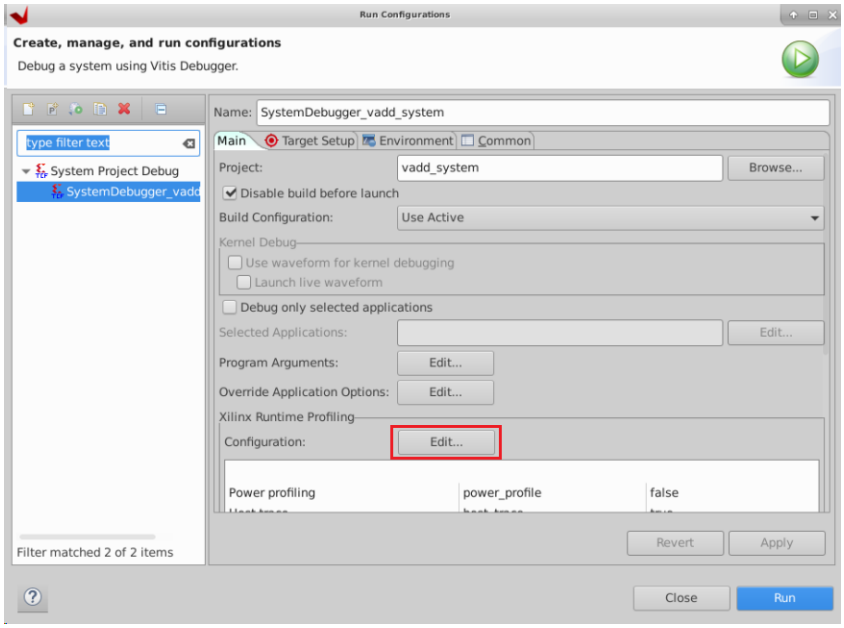
5. Select **File > Exit** to close the Analyzer

Timeline trace

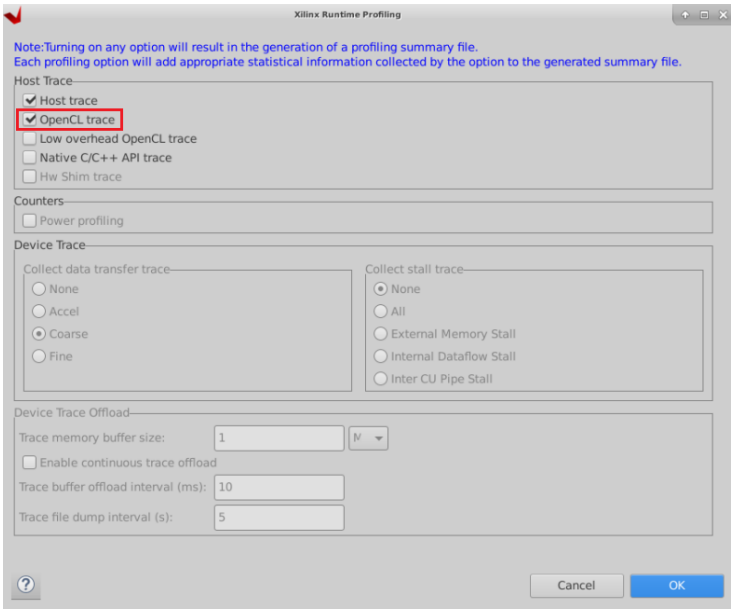
In order to see a Timeline Trace, you need to enable it in the Run configuration settings.

1. In the Assistant view, right click on **vadd_system** and select **Run > Run configurations**…

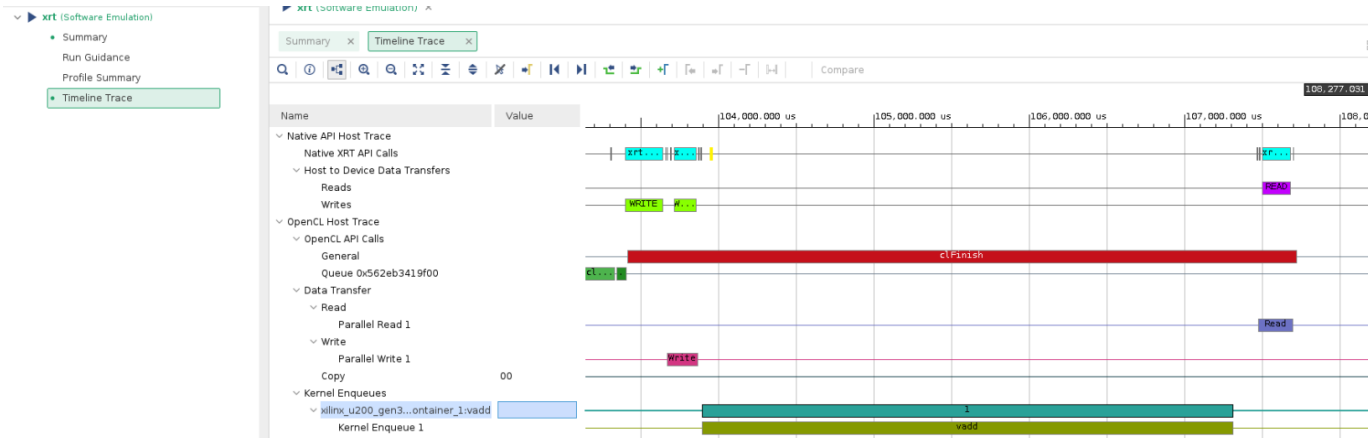
2. Under the Xilinx Runtime Profiling section, click the Configuration Edit… button



3. Select the **OpenCL** trace option, click **OK** to save the change and click **Run**



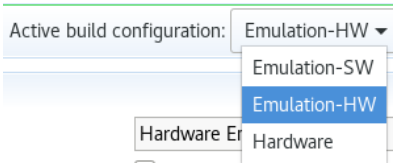
4. Open the Vitis Analyzer again (**vadd_system > vadd > Emulation-SW > SystemDebugger_vadd_system_vadd > Run Summary (xclbin)**) and select the **Timeline Trace**



5. You can close close the Vitis Analyzer when you are finished.

Build and run hardware emulation

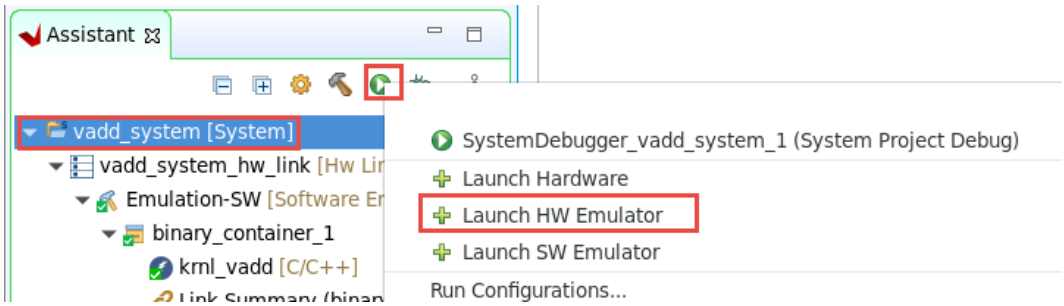
1. In the vadd_system panel, set the Active build configuration to Emulation-HW:



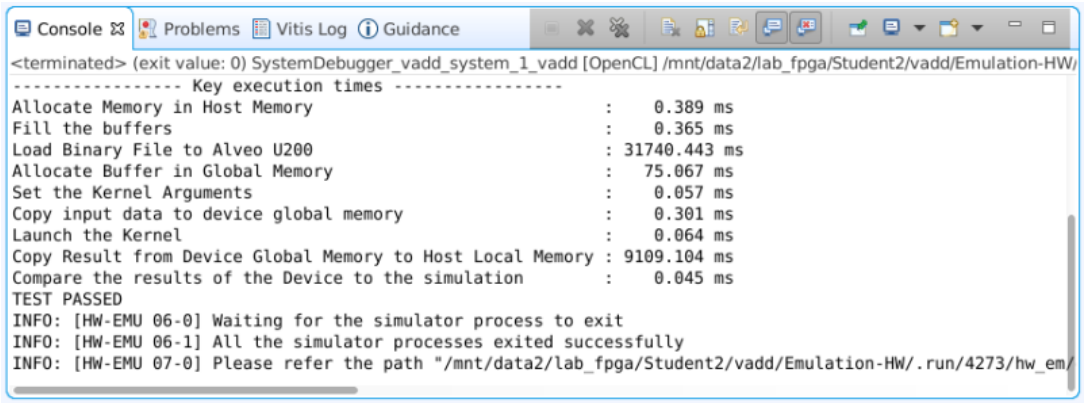
2. Build the project by selecting **vadd_system** in **Assistant view** and clicking the build button. This may take 10 minutes

Run hardware emulation

1. In the Assistant view, select v**add_system**, then click **run** the button on the icon bar and select **Launch HW Emulator**



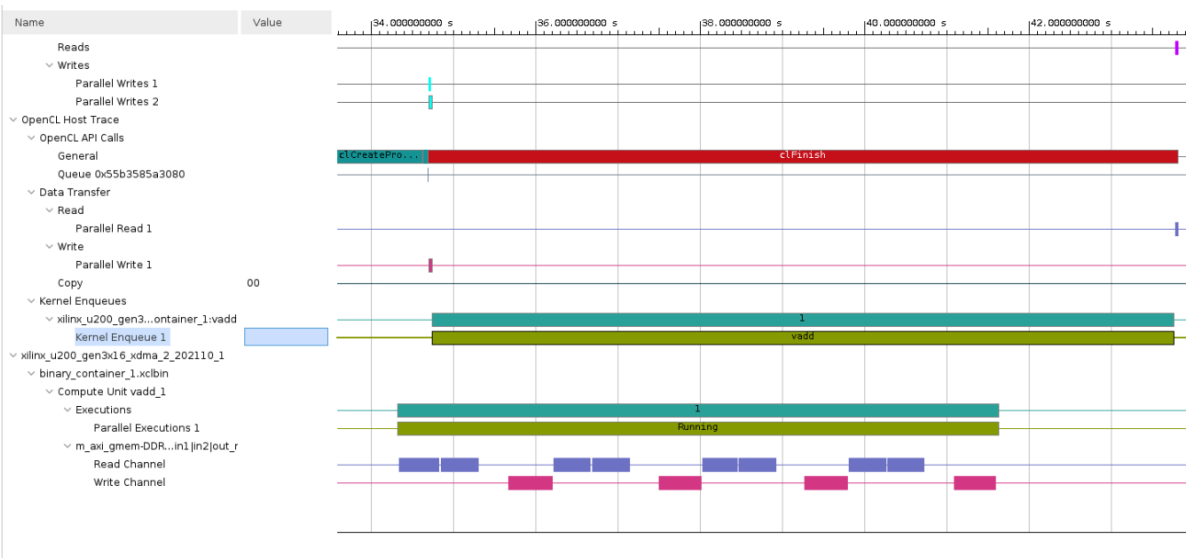
2. Observe the application is run and the output is displayed in the *Console* view. Compared to software emulation, the output also shows data transfer information.



5. View Emulation Timeline

   In the Assistant view, select **vadd_system > vadd > Emulation-HW > SystemDebugger_vadd_system_vadd > Run Summary (xclbin) to open Vitis Analyzer**

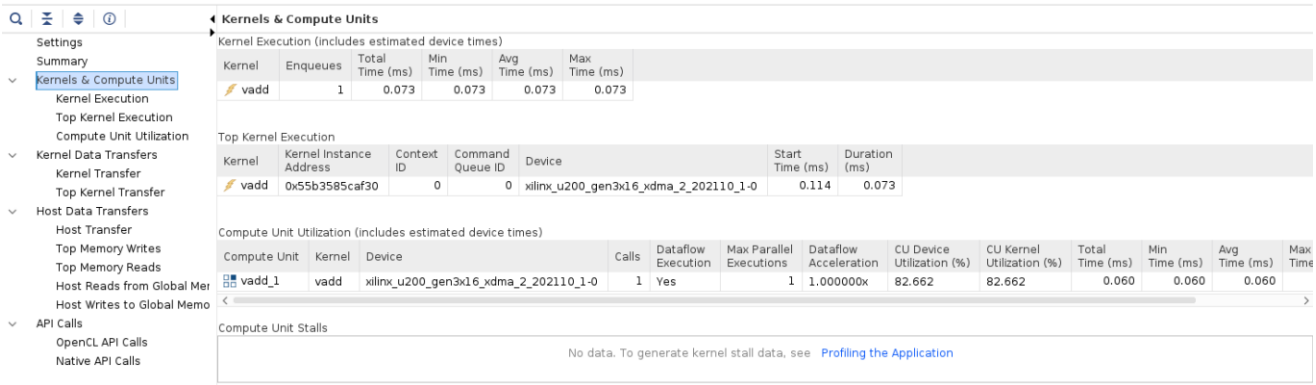
   Click on **Timeline Trace**. Zoom in between 35 and 45 second area and observe the activities in various parts of the system. Note that the data are processed in smaller chunks in the kernel and in a sequential manner



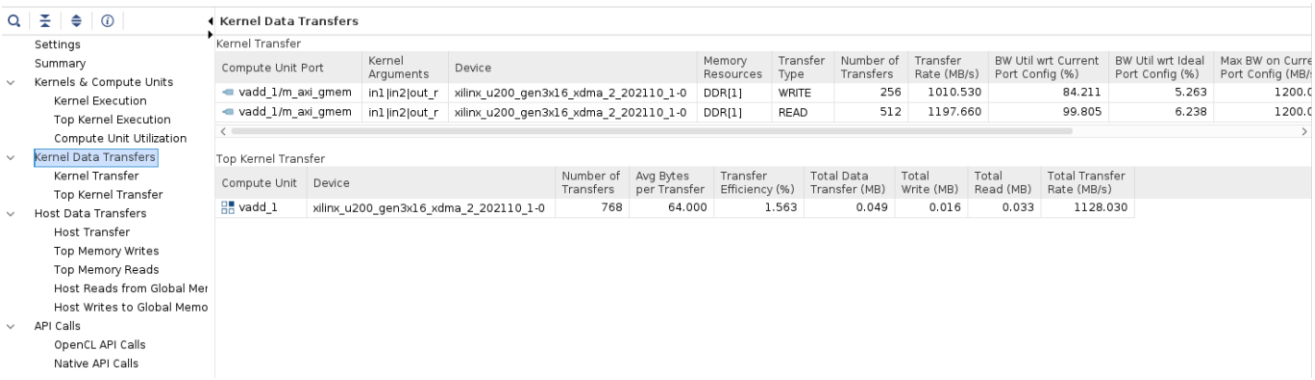
6. Click on **Profile Summary** entry in the left panel, and observe different entries, each containing reports on various performance metrics, we will focus on four of them
    - **Kernels & Compute Units :** Shows the number of times the kernel was executed. Includes the total, minimum, average, and maximum run times. If the design has multiple compute units, it will show each compute unit's utilization. When accelerating an algorithm, the faster the kernel executes, the higher the throughput which can be achieved.
    - **Kernel Data Transfers :** This report has no bearing in software emulation as no actual data transfers are emulated across the host to the platform. In hardware emulation, this shows the emulated throughput and bandwidth of the read/ writes to the global memory that the host and kernel share
    - **Host Data Transfer :** Shows the top operations related to memory transfer between the host and kernel to global memory, and kernel execution. This allows you to identify throughput bottlenecks when transferring data. Efficient transfer of data to the kernel/host allows for faster execution times
    - **API Calls :** Shows all the OpenCL API command executions, how many times each was executed, and how long they take to execute

7. Click on each of tabs and review the report:

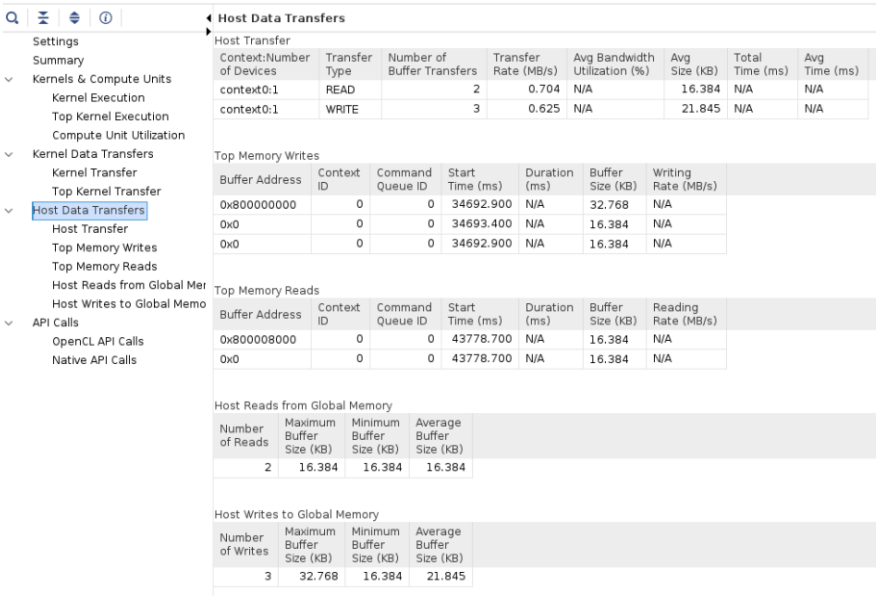
    - Kernels & Compute Units



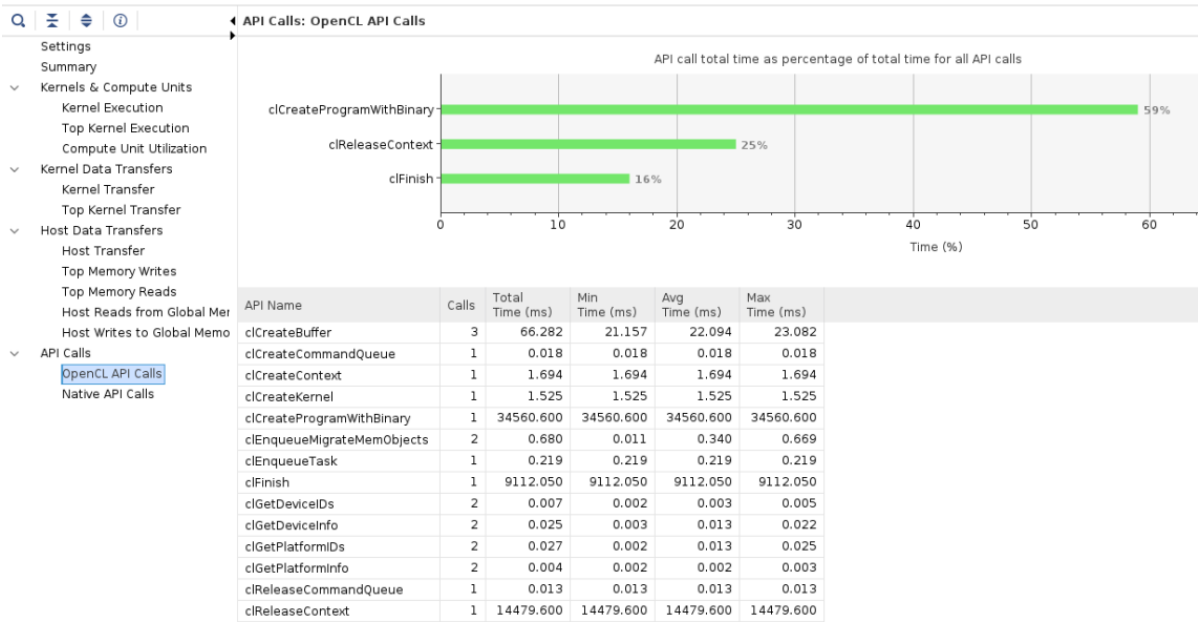
- Kernal Data Transfers



- Host Data Transfers



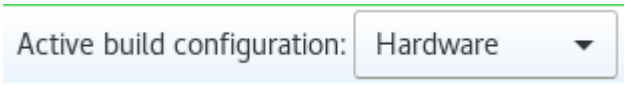
- API calls



8. When finished, close the analyzer by clicking `File -> Exit` and clicking **OK**

Build Full Hardware System

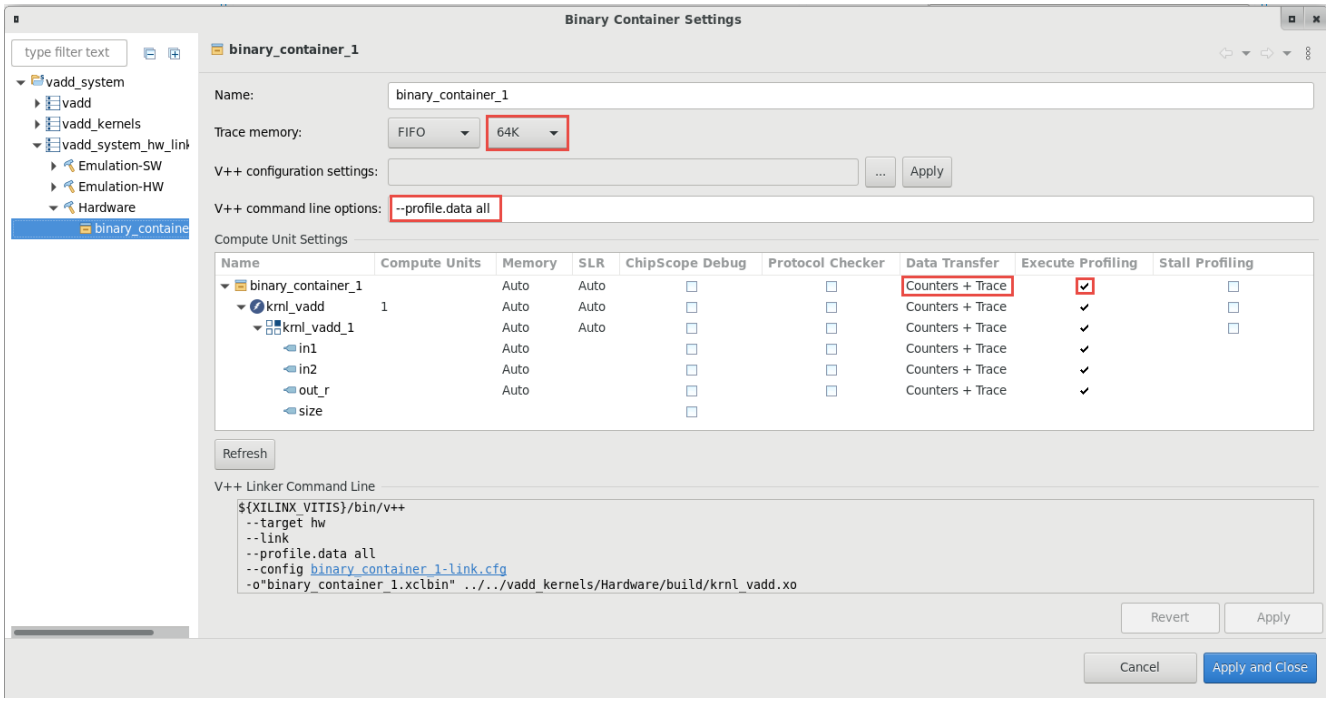
1. Set `Active build configuration:` to `Hardware` on the upper right corner of *Project Editor* view



   In order to collect the profiling data and run Timing Analyzer on the application run in hardware, we need to setup some options.

2. Select *vadd_system* > *vadd_system_hw_link* > *Hardware* > *binary_container_1* in Assistant view and then click on Settings. Click on the Data Transfer drop-down button in binary_container_1 row and select Counters+Trace option. This should also enable Execute Profiling option. If not, then click on the corresponding check box.
3. In the V++ command line options: field, enter --profile.data all to enable kernel profiling
4. Select Trace Memory to be FIFO type and size of 64K. This is the memory where traces will be stored. You also have the option to store this information in DDR (max limit 4 GB) and PLRAM

   At this point the settings should look like shown below



5. Click **Apply and Close**
6. Build the project by selecting vadd_system in Assistant view and clicking the build button ( )

   This will build the project under the Hardware directory. The built project will include vadd (executable) file along with the binary_container_1.xclbin file

7. A **binary_container_1.xclbin** and vadd application will be generated in the *vadd/Hardware* directory