

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

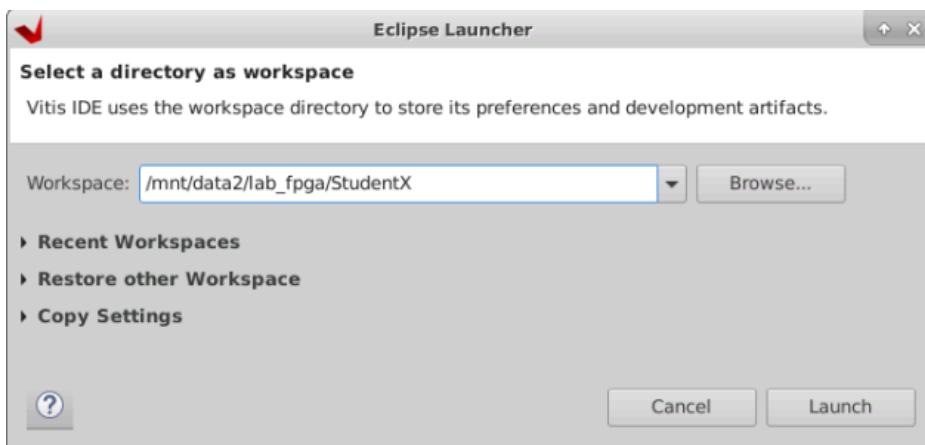
Steps

Create a Vitis Project

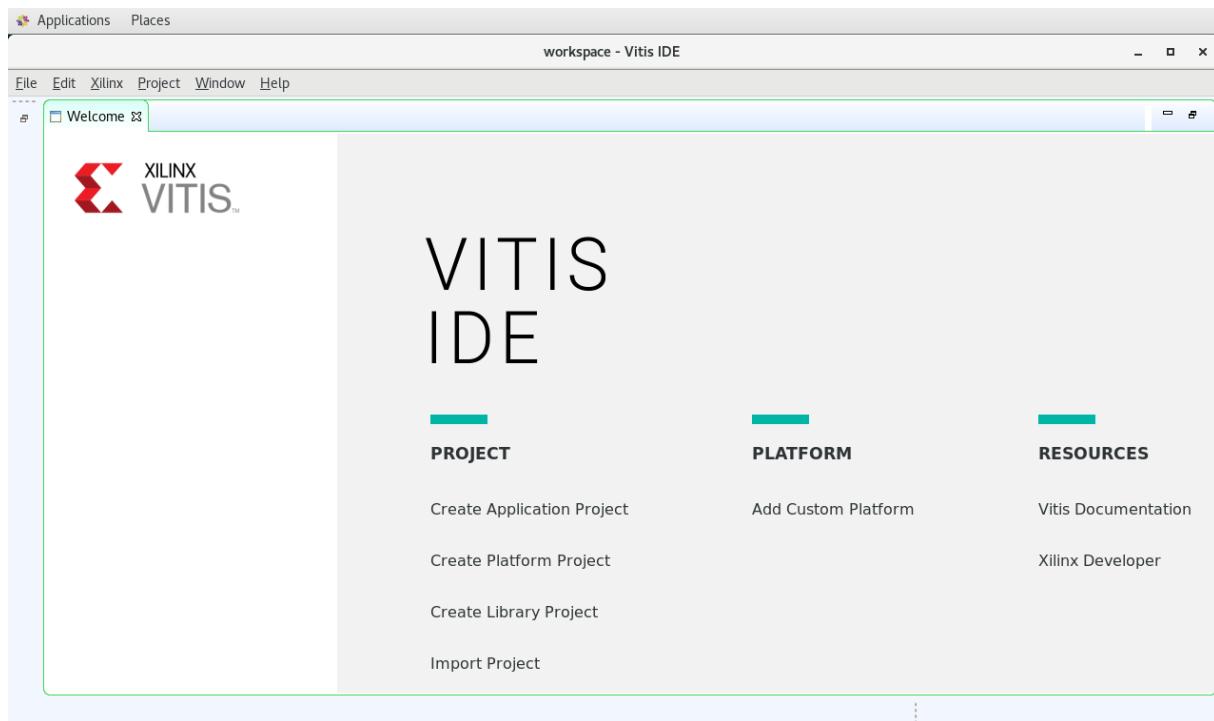
1. Invoke GUI by executing the following command:

```
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)

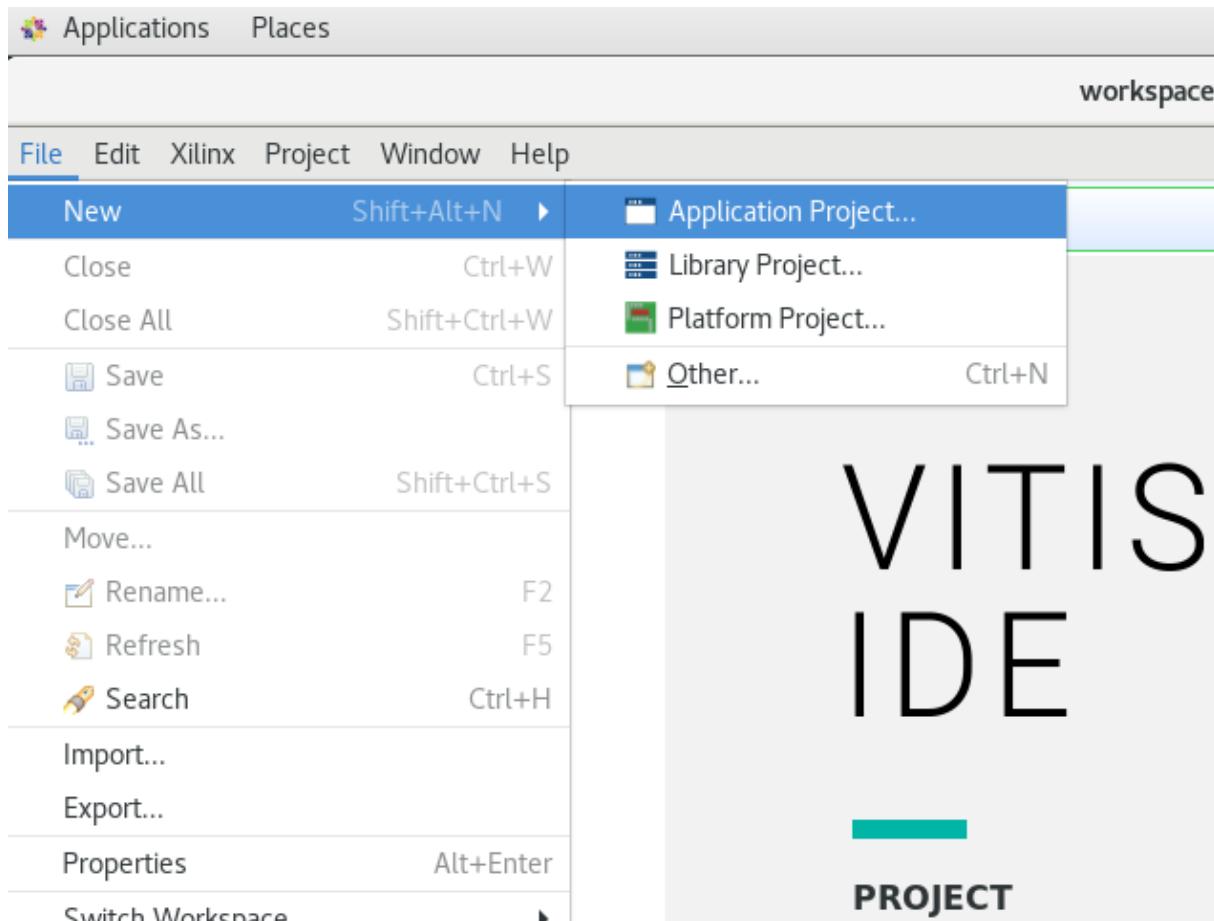


3. The Vitis IDE Welcome page will be displayed, if new a workspace is assigned



4. Create a new application project

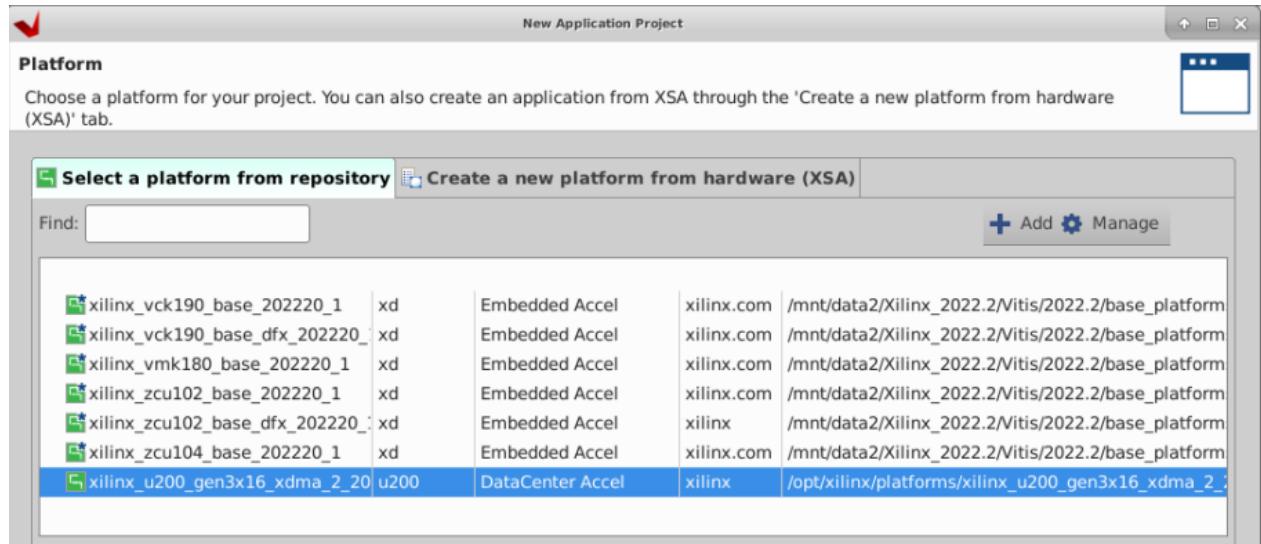
Use `Create Application Project` from Welcome page, or use `File > New > Application Project` to create a new application.



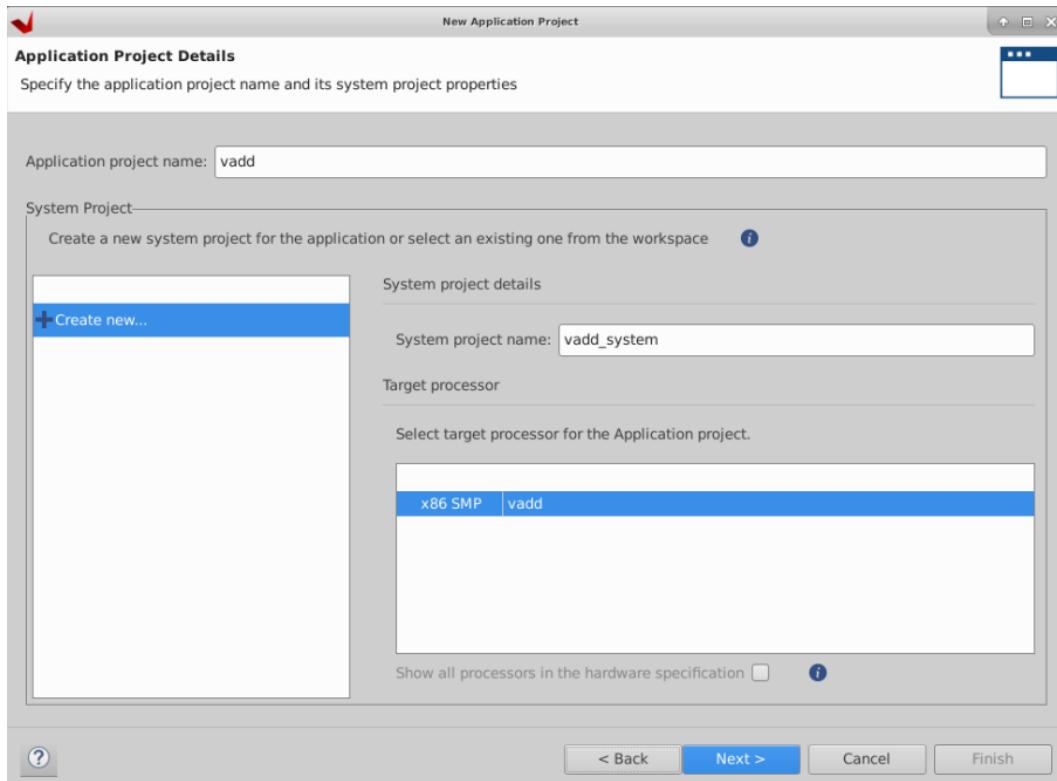
5. Close Welcome page, if it was opened

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>**

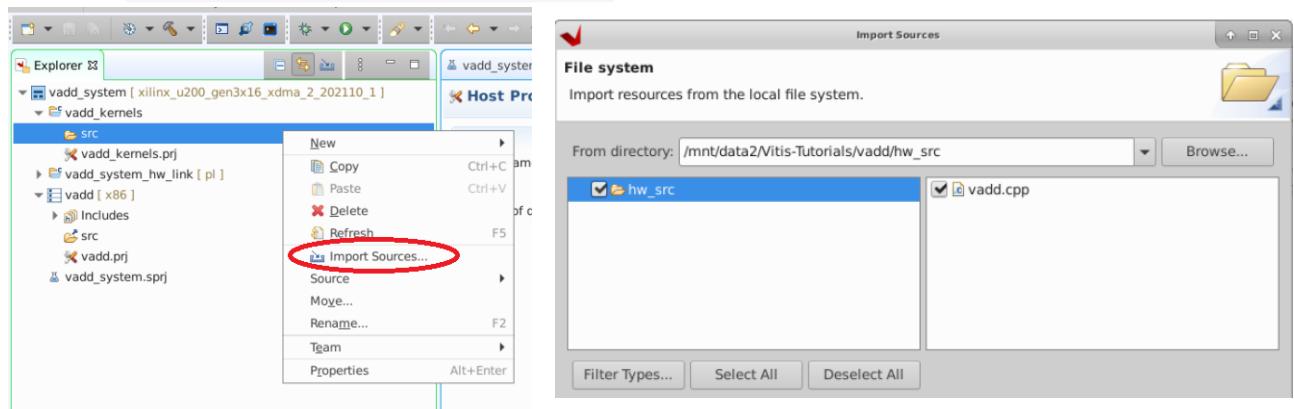


9. Select **Empty Application** as the template and click **Finish**. The project is generated.

10. Import provided source files from vadd sub-directories.

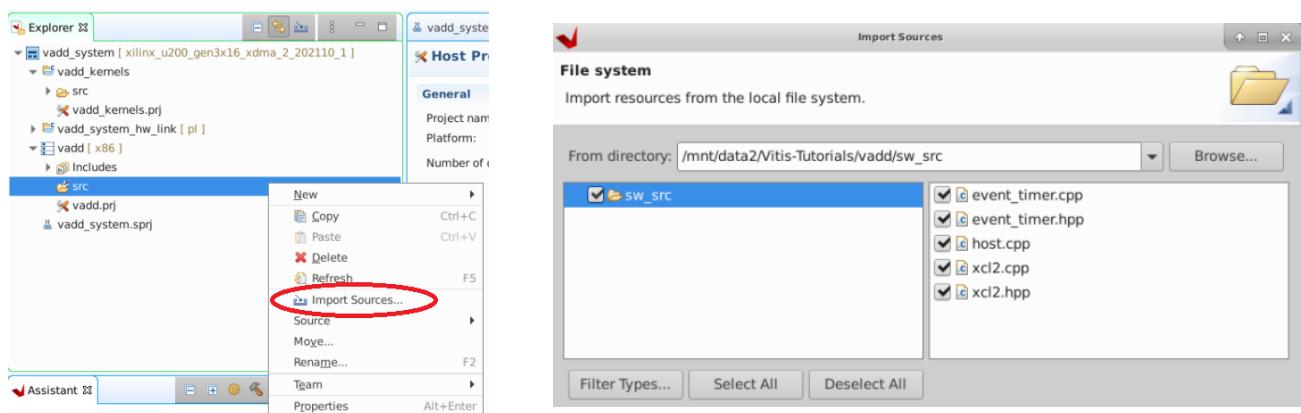
Select **vadd_kernels** --> **src** directory in Explorer view, right click and select Import Sources... in order to add the kernel

Import /mnt/data2/Vitis-Tutorials/vadd/hw_src/vadd.cpp for **hardware accelerator** and press **Finish**

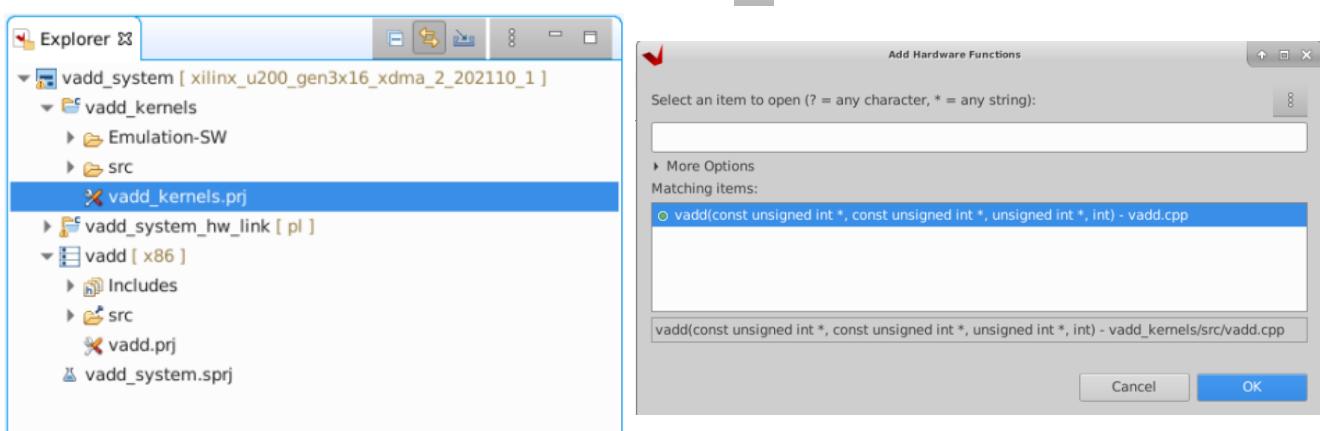


Select **vadd** --> **src** directory in Explorer view, right click and select Import Sources... in order to add the host files

import all *.cpp and *.hpp files in /mnt/data2/Vitis-Tutorials/vadd/sw_src/ for **host code** application and press **Finish**



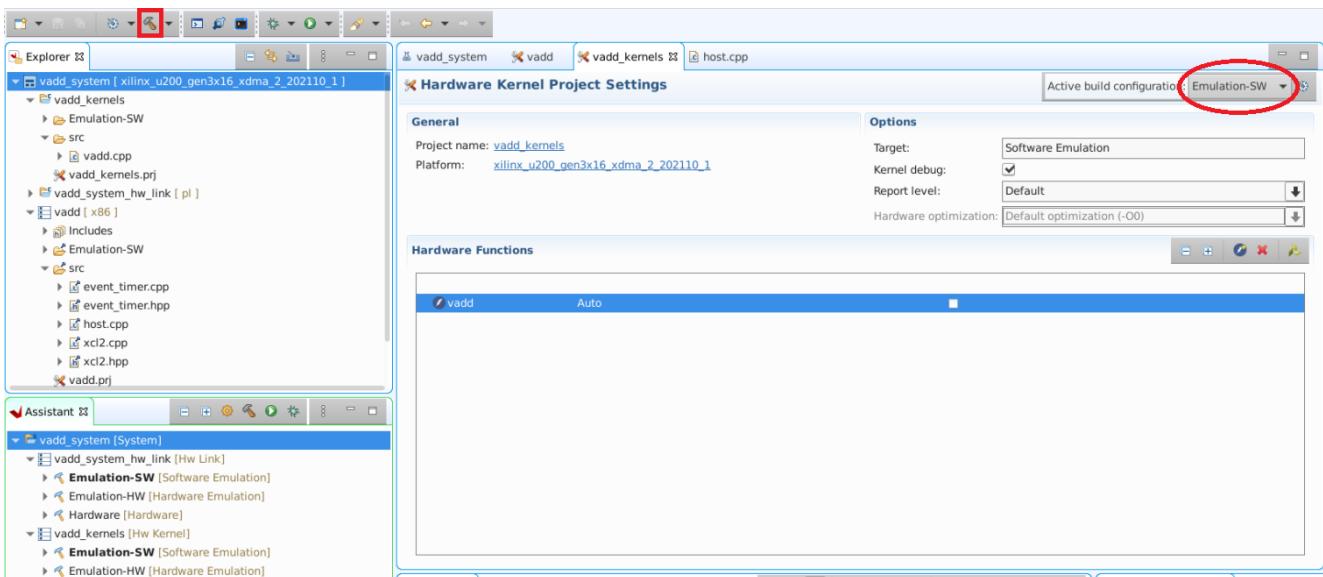
11. The project is generated. Double Click in vadd_kernels.prj and click on in order to add the **vadd** as a *Hardware Function* (kernel).



Build and Run Software Emulation

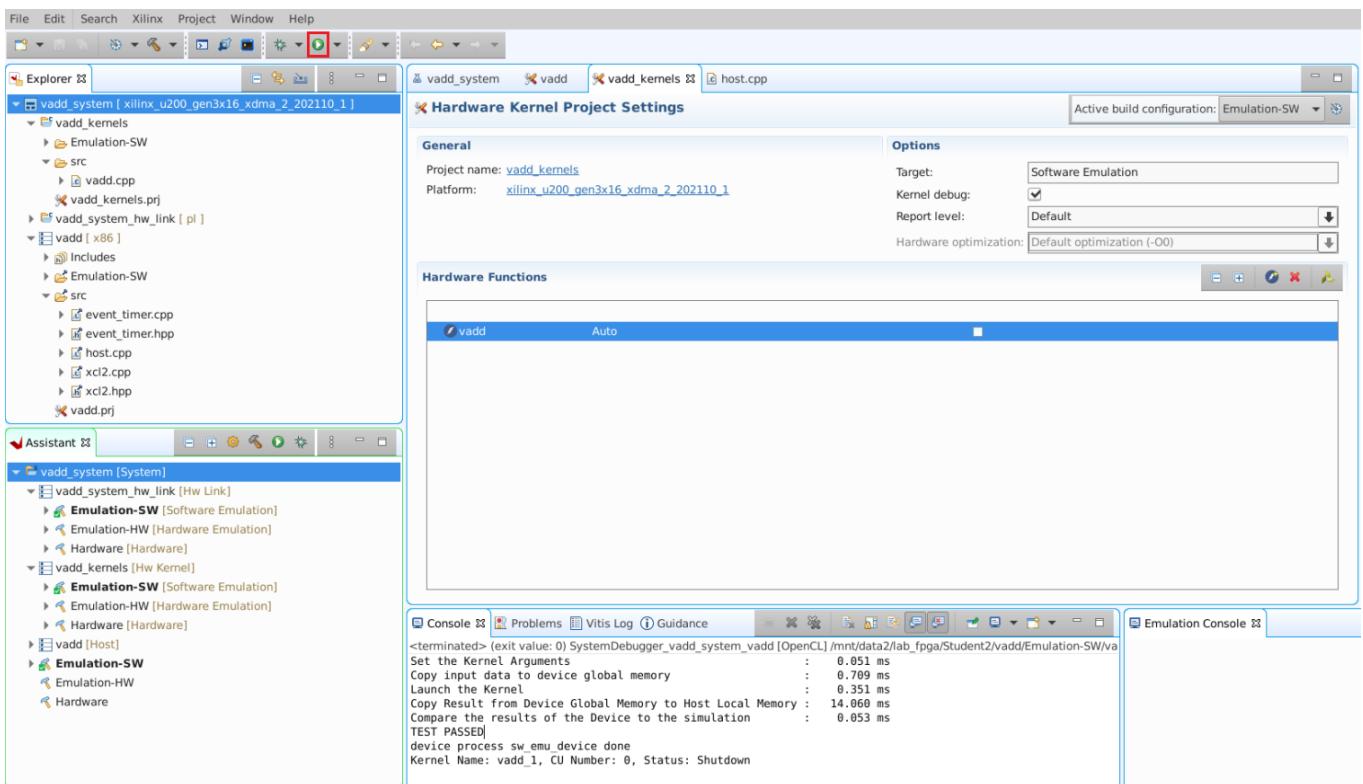
1. Set Active build configuration: to **Emulation-SW** on the upper right corner of *Project Editor* view

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

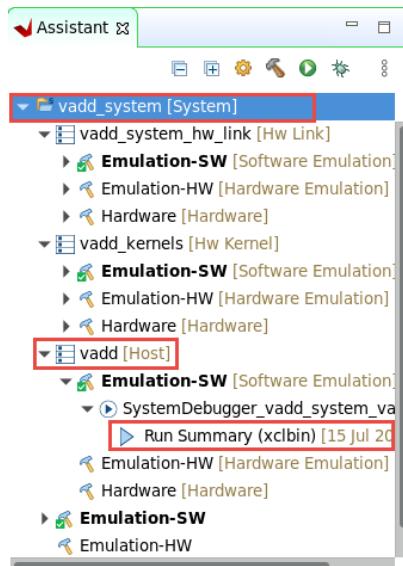
```

<terminated> (exit value: 0) SystemDebugger_vadd_system_vadd [OpenCL] /mnt/data2/lab_fpga/Student2/vadd/Emulation-SW/vadd
Kernel Name: vadd_1, CU Number: 0, State: Idle
----- Key execution times -----
Allocate Memory in Host Memory : 0.445 ms
Fill the buffers : 0.323 ms
Load Binary File to Alveo U200 : 87.357 ms
Allocate Buffer in Global Memory : 0.371 ms
Set the Kernel Arguments : 0.051 ms
Copy input data to device global memory : 0.709 ms
Launch the Kernel : 0.351 ms
Copy Result from Device Global Memory to Host Local Memory : 14.060 ms
Compare the results of the Device to the simulation : 0.053 ms
TEST PASSED
device process sw_emu_device done
Kernel Name: vadd_1, CU Number: 0, Status: Shutdown

```

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.

The screenshot shows the Vitis Analyzer interface with the 'Profile Summary' tab selected. Under the 'Kernels & Compute Units' section, the 'Kernel Execution' tab is active, displaying a table for the 'vadd' kernel. The table includes columns for Kernel, Enqueues, Total Time (ms), Min Time (ms), Avg Time (ms), and Max Time (ms). The 'Top Kernel Execution' table shows details for the 'vadd' kernel, including its address (0x5567711df650), context ID (0), command queue ID (0), device (xilinx_u200_gen3x16_xdma_2_202110_1-0), start time (93.643 ms), duration (10.008 ms), and clock frequency (300.000 MHz).

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

The screenshot shows the Vitis Analyzer interface with the 'Profile Summary' tab selected. Under the 'Host Data Transfers' section, the 'Host Transfer' tab is active, displaying tables for 'Top Memory Writes' and 'Top Memory Reads'. The 'Top Memory Writes' table shows a single entry for buffer address 0x2000, context ID 0, command queue ID 0, start time (90.173 ms), duration (N/A), buffer size (32.768 kB), and writing rate (N/A). The 'Top Memory Reads' table shows a single entry for buffer address 0xa000, context ID 0, command queue ID 0, start time (103.889 ms), duration (N/A), buffer size (16.384 kB), and reading rate (N/A).

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

Timeline trace

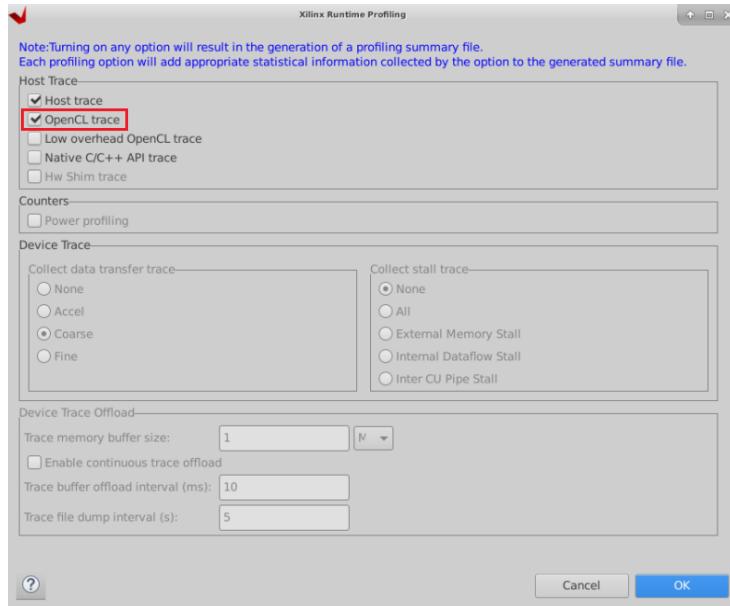
In order to see a Timeline Trace, you need to enable it in the Run configuration settings.

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

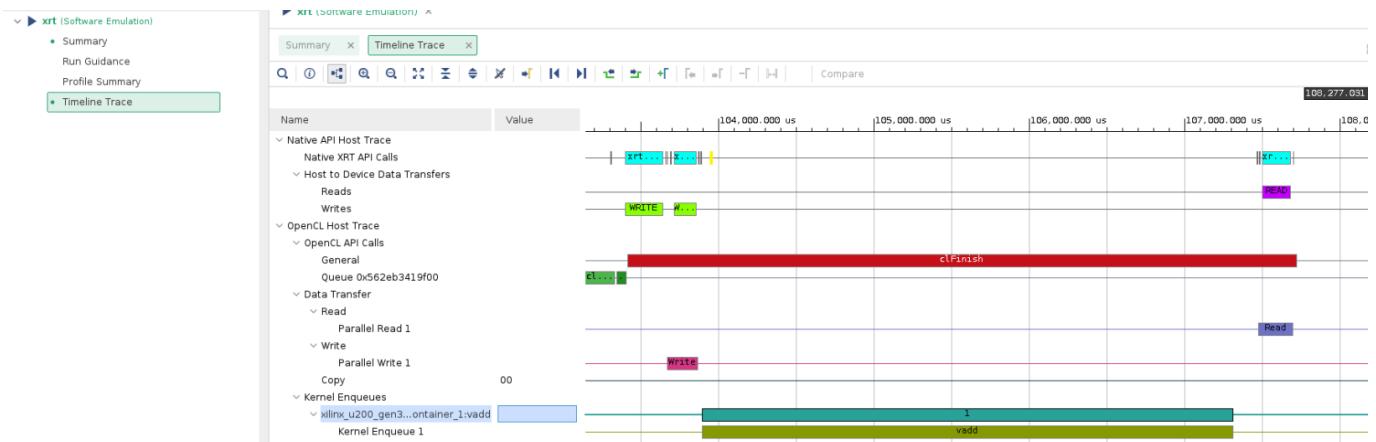
2. Under the Xilinx Runtime Profiling section, click the Configuration Edit... button

The screenshot shows the 'Run Configurations' dialog. In the 'Main' tab, under 'Xilinx Runtime Profiling', there is a 'Configuration:' field with an 'Edit...' button highlighted with a red box. Below this, there is a table for 'Power profiling' with columns for 'power_profile' and 'false'. At the bottom of the dialog are 'Revert', 'Apply', 'Close', and 'Run' buttons.

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



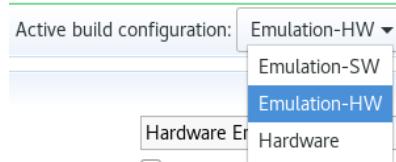
4. Open the Vitis Analyzer again (**vadd_system > vadd > Emulation-SW > SystemDebugger_vadd_system_vadd > Run Summary (xclbin)**) and select the **Timeline Trace**



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

Build and run hardware emulation

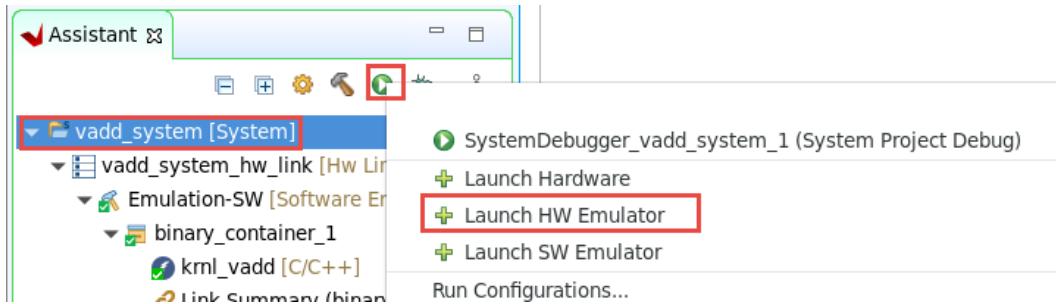
1. In the vadd_system panel, set the Active build configuration to Emulation-HW:



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 **vadd_system**, then click **run** the button on the icon bar and select **Launch HW Emulator**



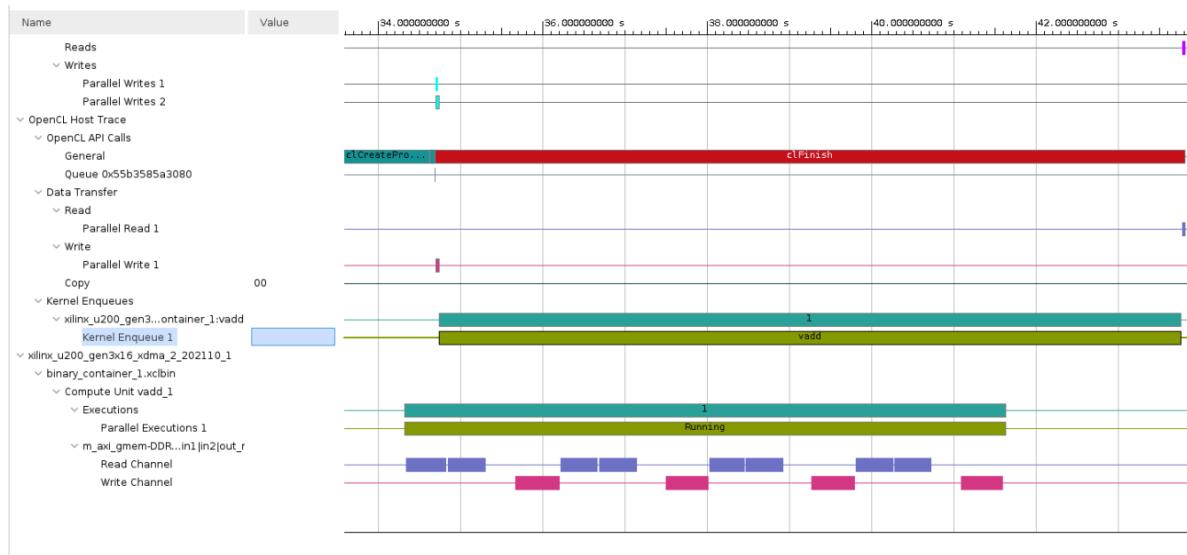
2. Observe the application is run and the output is displayed in the **Console** view. Compared to software emulation, the output also shows data transfer information.

```
<terminated> (exit value: 0) SystemDebugger_vadd_system_1_vadd [OpenCL] /mnt/data2/lab_fpga/Student2/vadd/Emulation-HW/
----- Key execution times -----
Allocate Memory in Host Memory : 0.389 ms
Fill the buffers : 0.365 ms
Load Binary File to Alveo U200 : 31740.443 ms
Allocate Buffer in Global Memory : 75.067 ms
Set the Kernel Arguments : 0.057 ms
Copy input data to device global memory : 0.301 ms
Launch the Kernel : 0.064 ms
Copy Result from Device Global Memory to Host Local Memory : 9109.104 ms
Compare the results of the Device to the simulation : 0.045 ms
TEST PASSED
INFO: [HW-EMU 06-0] Waiting for the simulator process to exit
INFO: [HW-EMU 06-1] All the simulator processes exited successfully
INFO: [HW-EMU 07-0] Please refer the path "/mnt/data2/lab_fpga/Student2/vadd/Emulation-HW/.run/4273/hw_em/
```

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

Kernels & Compute Units						
Kernel Execution (includes estimated device times)						
Kernel	Enqueues	Total Time (ms)	Min Time (ms)	Avg Time (ms)	Max Time (ms)	
vadd	1	0.073	0.073	0.073	0.073	

Top Kernel Execution						
Kernel	Kernel Instance Address	Context ID	Command Queue ID	Device	Start Time (ms)	Duration (ms)
vadd	0x55b3585caf30	0	0	xilinx_u200_gen3x16_xdma_2_202110_1-0	0.114	0.073

Compute Unit Utilization (includes estimated device times)												
Compute Unit	Kernel	Device	Calls	Dataflow Execution	Max Parallel Executions	Dataflow Acceleration	CU Device Utilization (%)	CU Kernel Utilization (%)	Total Time (ms)	Min Time (ms)	Avg Time (ms)	Max Time
vadd_1	vadd	xilinx_u200_gen3x16_xdma_2_202110_1-0	1	Yes	1	1.000000x	82.662	82.662	0.060	0.060	0.060	

Compute Unit Stalls											
No data. To generate kernel stall data, see Profiling the Application											

- Kernal Data Transfers

Kernel Data Transfers

Compute Unit Port	Kernel Arguments	Device	Memory Resources	Transfer Type	Number of Transfers	Transfer Rate (MB/s)	BW Util wrt Current Port Config (%)	BW Util wrt Ideal Port Config (%)	Max BW on Current Port Config (MB/s)
vadd_1/m_axi_gmem	in1 in2 out_r	xilinx_u200_gen3x16_xdma_2_202110_1-0	DDR[1]	WRITE	256	1010.530	84.211	5.263	1200.0
vadd_1/m_axi_gmem	in1 in2 out_r	xilinx_u200_gen3x16_xdma_2_202110_1-0	DDR[1]	READ	512	1197.660	99.805	6.238	1200.0

Top Kernel Transfer

Compute Unit	Device	Number of Transfers	Avg Bytes per Transfer	Transfer Efficiency (%)	Total Data Transfer (MB)	Total Write (MB)	Total Read (MB)	Total Transfer Rate (MB/s)
vadd_1	xilinx_u200_gen3x16_xdma_2_202110_1-0	768	64.000	1.563	0.049	0.016	0.033	1128.030

- Host Data Transfers

Host Data Transfers

Context: Number of Devices	Transfer Type	Number of Buffer Transfers	Transfer Rate (MB/s)	Avg Bandwidth Utilization (%)	Avg Size (kB)	Total Time (ms)	Avg Time (ms)
context:0:1	READ	2	0.704	N/A	16.384	N/A	N/A
context:0:1	WRITE	3	0.625	N/A	21.845	N/A	N/A

Top Memory Writes

Buffer Address	Context ID	Command Queue ID	Start Time (ms)	Duration (ms)	Buffer Size (kB)	Writing Rate (MB/s)
0x8000000000	0	0	34692.900	N/A	32.768	N/A
0x0	0	0	34693.400	N/A	16.384	N/A
0x0	0	0	34692.900	N/A	16.384	N/A

Top Memory Reads

Buffer Address	Context ID	Command Queue ID	Start Time (ms)	Duration (ms)	Buffer Size (kB)	Reading Rate (MB/s)
0x800008000	0	0	43778.700	N/A	16.384	N/A
0x0	0	0	43778.700	N/A	16.384	N/A

Host Reads from Global Mem

Number of Reads	Maximum Buffer Size (kB)	Minimum Buffer Size (kB)	Average Buffer Size (kB)
2	16.384	16.384	16.384

Host Writes to Global Memory

Number of Writes	Maximum Buffer Size (kB)	Minimum Buffer Size (kB)	Average Buffer Size (kB)
3	32.768	16.384	21.845

- API calls

API Calls: OpenCL API Calls

API call total time as percentage of total time for all API calls

API Name	Calls	Total Time (ms)	Min Time (ms)	Avg Time (ms)	Max Time (ms)
clCreateBuffer	3	66.282	21.157	22.094	23.082
clCreateCommandQueue	1	0.018	0.018	0.018	0.018
clCreateContext	1	1.694	1.694	1.694	1.694
clCreateKernel	1	1.525	1.525	1.525	1.525
clCreateProgramWithBinary	1	34560.600	34560.600	34560.600	34560.600
clEnqueueMigrateMemObjects	2	0.680	0.011	0.340	0.669
clEnqueueTask	1	0.219	0.219	0.219	0.219
clFinish	1	9112.050	9112.050	9112.050	9112.050
clGetDeviceIDs	2	0.007	0.002	0.003	0.005
clGetDeviceInfo	2	0.025	0.003	0.013	0.022
clGetPlatformIDs	2	0.027	0.002	0.013	0.025
clGetPlatformInfo	2	0.004	0.002	0.002	0.003
clReleaseCommandQueue	1	0.013	0.013	0.013	0.013
clReleaseContext	1	14479.600	14479.600	14479.600	14479.600

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

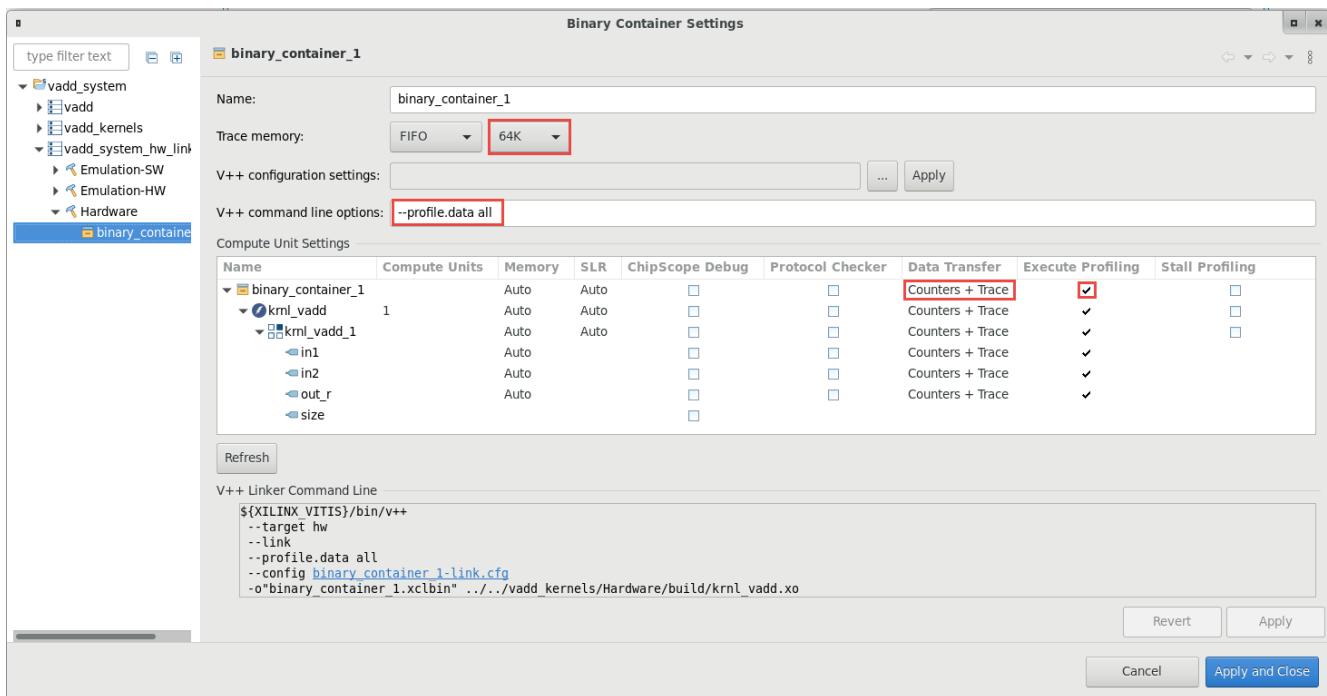


In order to collect the profiling data and run Timing Analyzer on the application run in hardware, we need to setup some options.

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.xlbin file

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