School of Electrical & Computer Engineering - Aristotle University of Thessaloniki

Introduction to Vitis using Alveo U200 on PowerEdge R730 (2019.2)

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

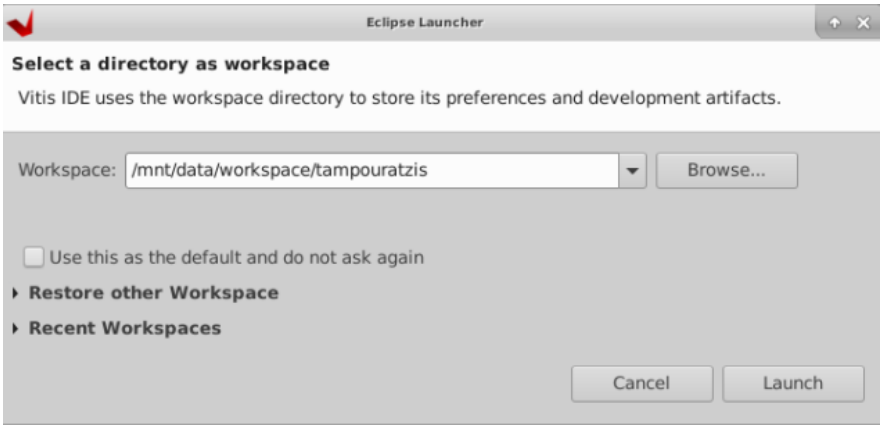
Steps

Create a Vitis Project

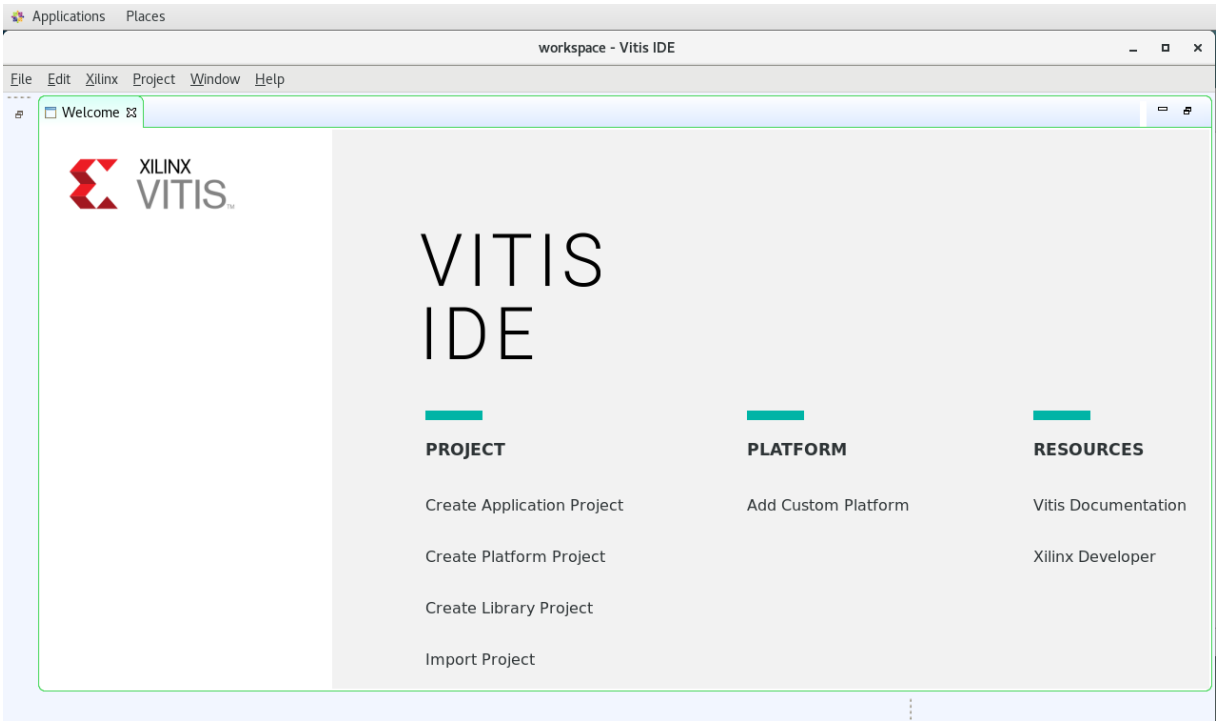
1. Invoke GUI by executing the following command:

```
vitis &
```

2. Set workspace to any empty folder, such as `~/workspace` and click **Launch**

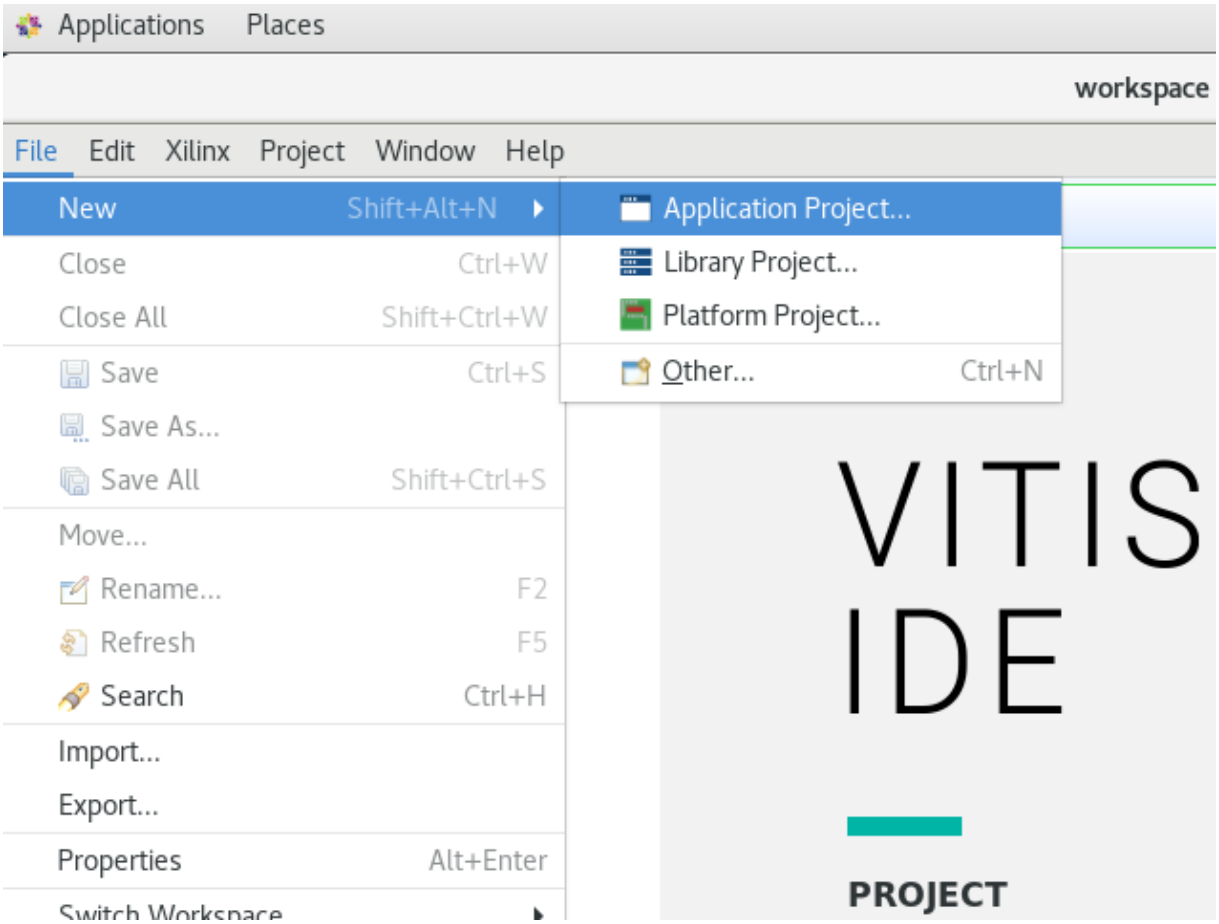


3. The Vitis IDE Welcome page will be displayed, if new a workspace is assigned



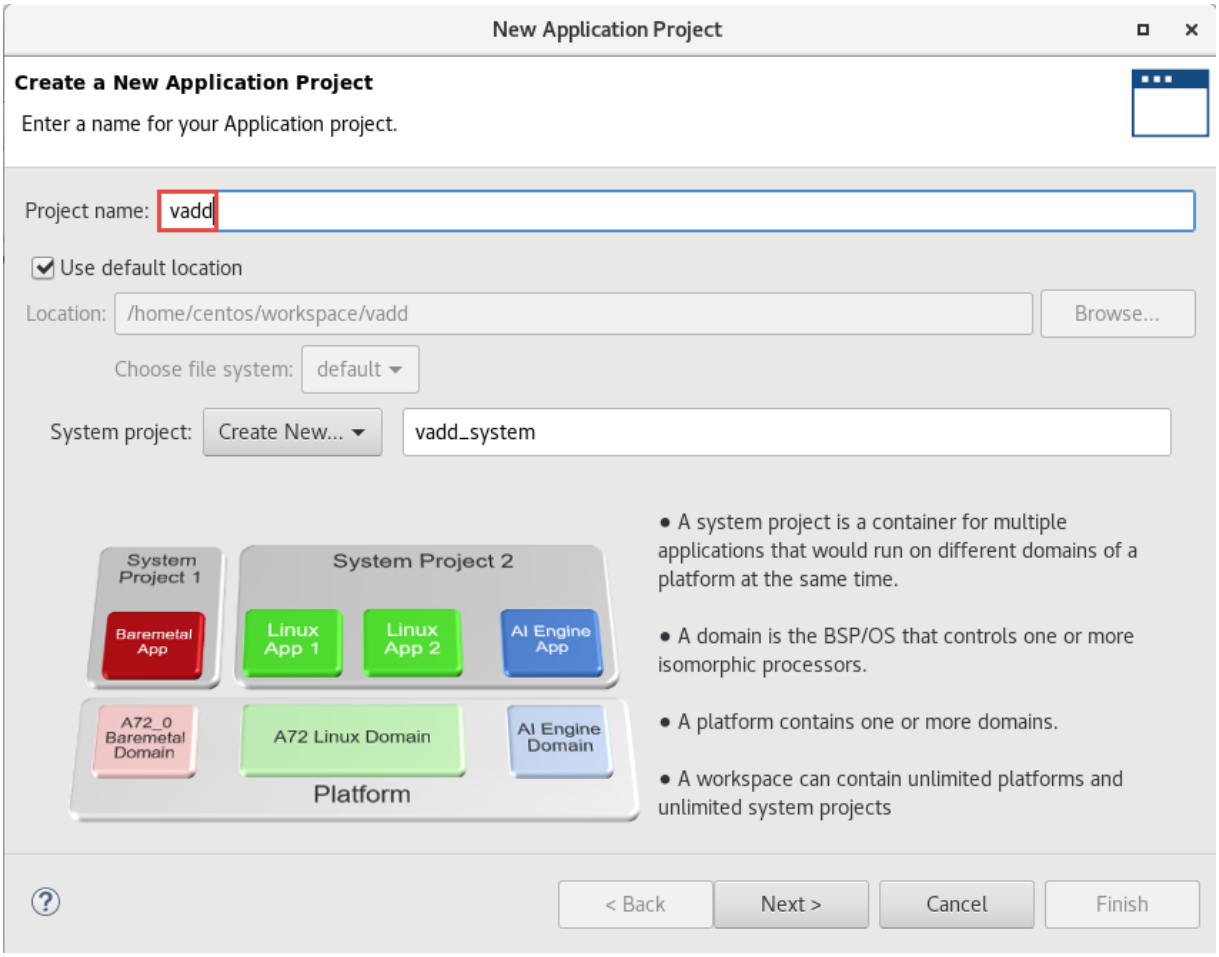
4. Create a new application project

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



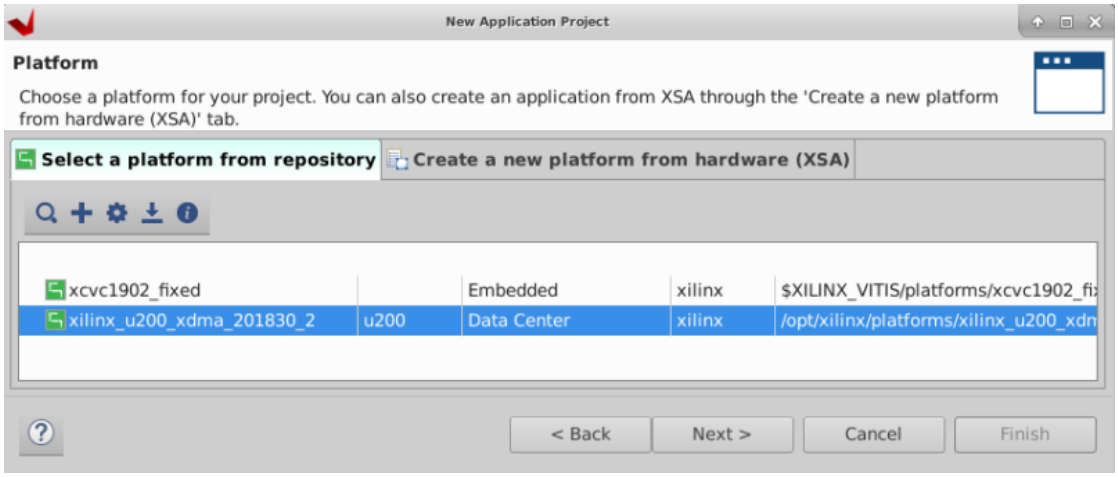
5. Close Welcome page, if it was opened

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

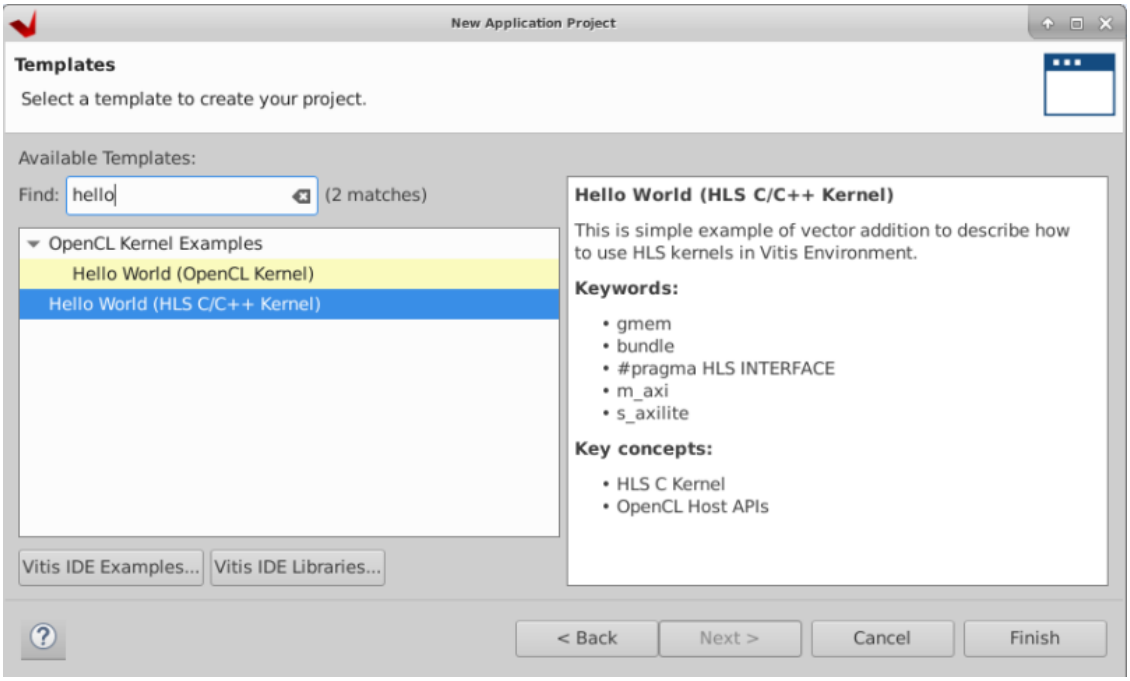


7. Select the platform `xilinx_u200_xdma_201830_2` , and click **OK**

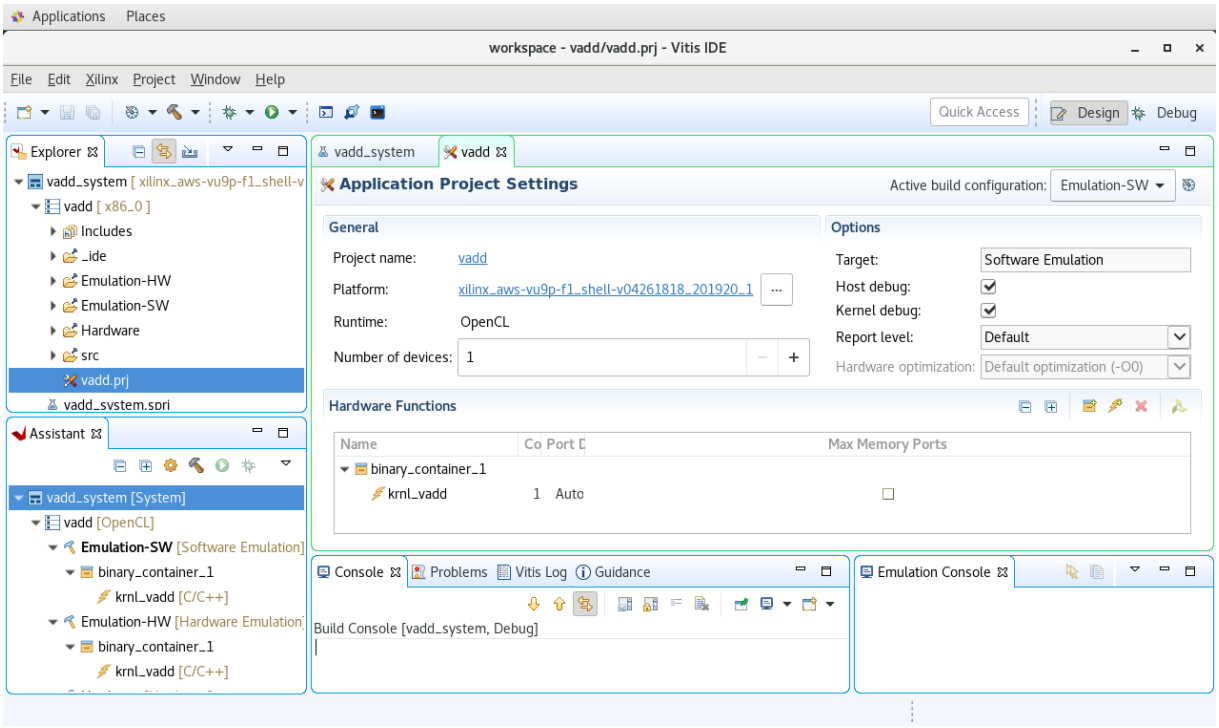
8. You will see the platform entry, select it and click **Next>**



9. Type "hello" and select `Hello World (HLS C/C++ Kernel)`, and click **Finish.** This is a simple example of vector addition.



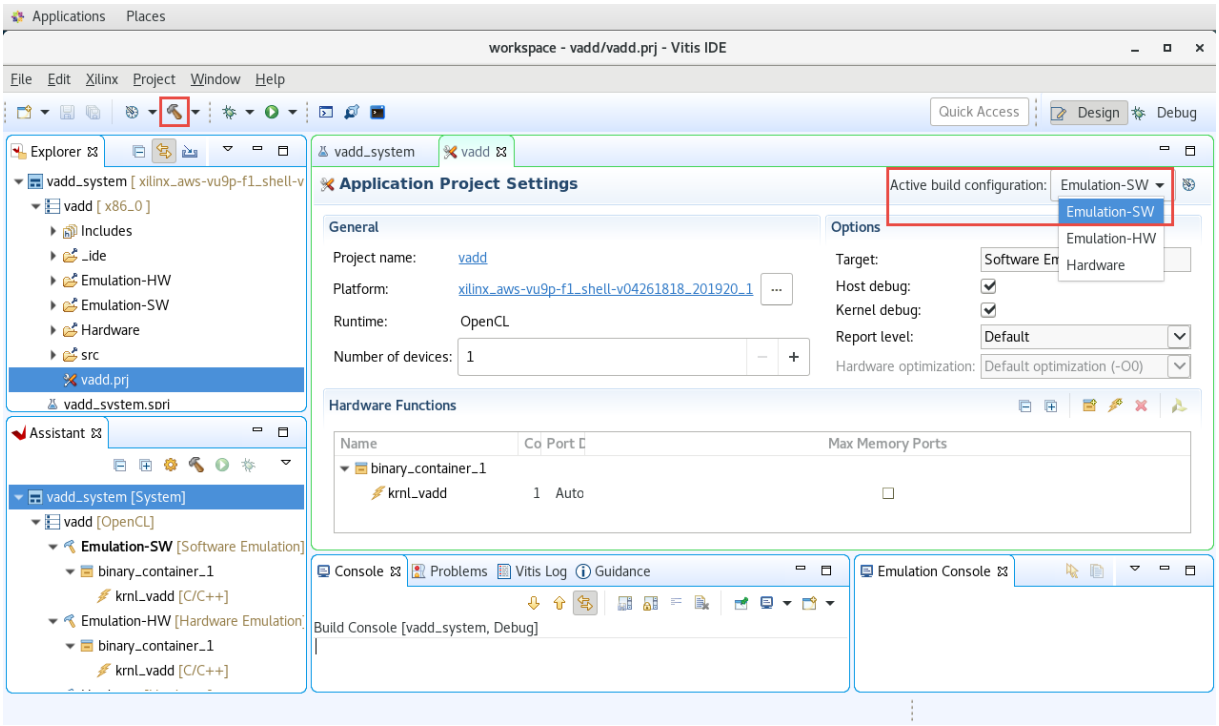
10. The project is generated. Notice that the *Hardware Function* in the *Project Editor* view is automatically set up to *krnl_vadd*



Build and Run Software Emulation

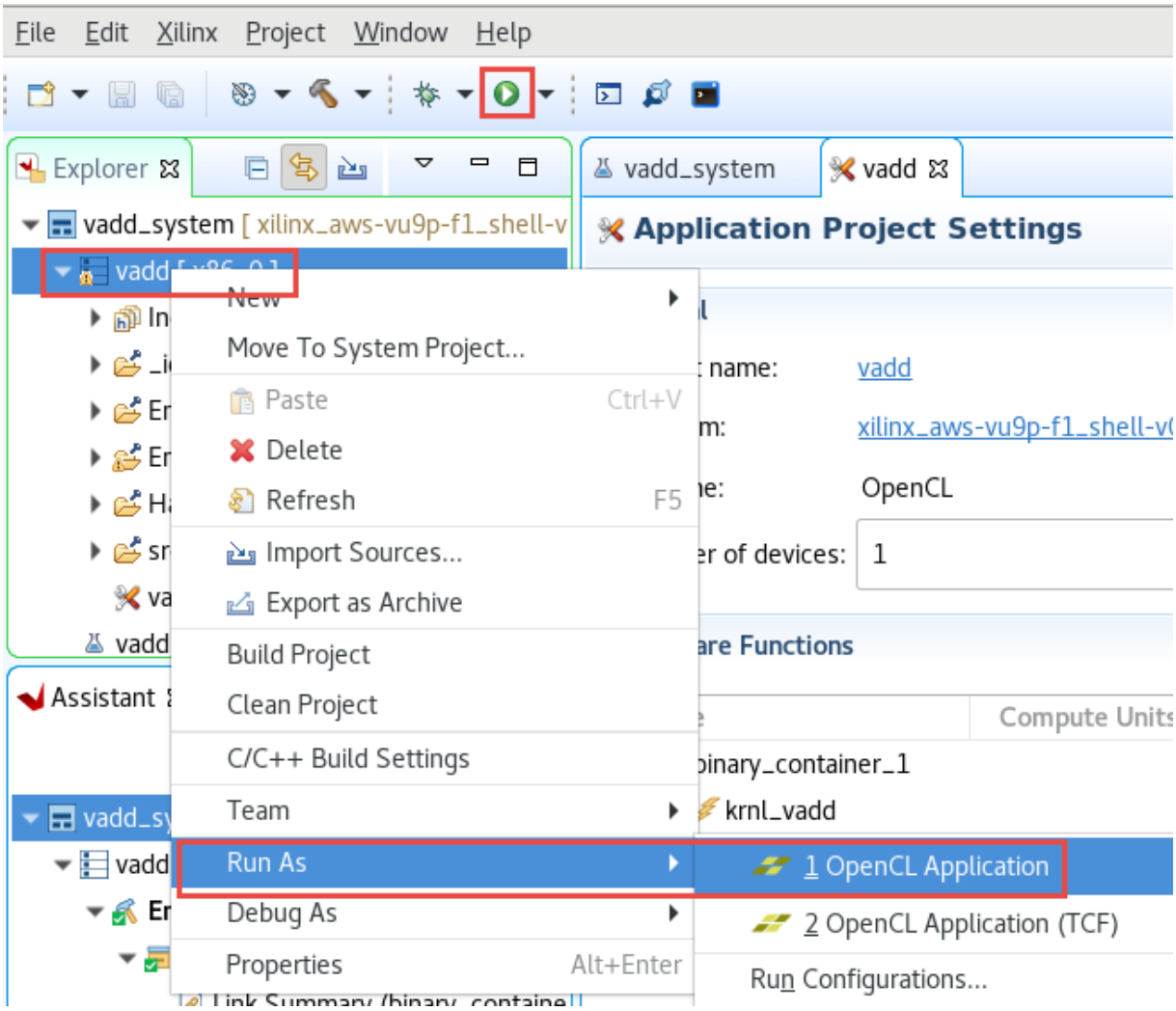
1. Set `Active build configuration:` to `Emulation-SW` on the upper right corner of *Project Editor* view

2. Begin build by clicking the little hammer icon on top icon bar, or right click `vadd` and select `Build Project`



3. Run Software Emulation in GUI Mode

   To launch software emulation in GUI mode, first select the application in *Explorer* view, then click run icon on icon bar, or right click application and select `Run As -> Run Configuration...`

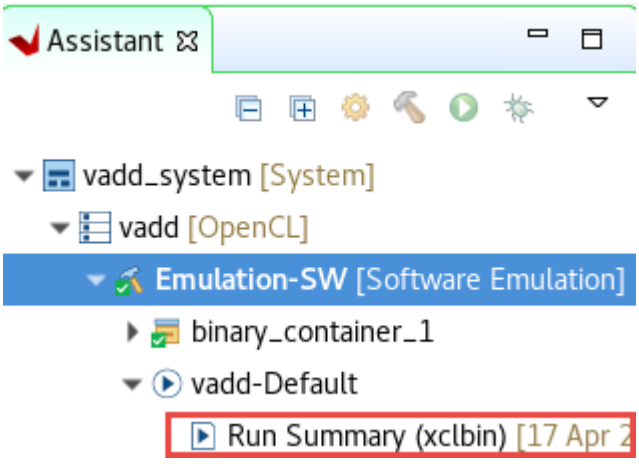


4. Observe the application is run and the output is displayed in the *Console* view

```
<terminated> (exit value: 0) vadd-launch [OpenCL] /mnt/data/workspace/tampouratzis/vadd/Emulation-SW/vadd (8/31/20, 6:28 PM)
[Console output redirected to file:/mnt/data/workspace/tampouratzis/vadd/Emulation-SW/vadd-launch.launch.log]
Found Platform
Platform Name: Xilinx
INFO: Reading ../vadd.xclbin
Loading: '../vadd.xclbin'
Trying to program device[0]: xilinx_u200_xdma_201830_2
Device[0]: program successful!
TEST PASSED
```

View Emulation Timeline

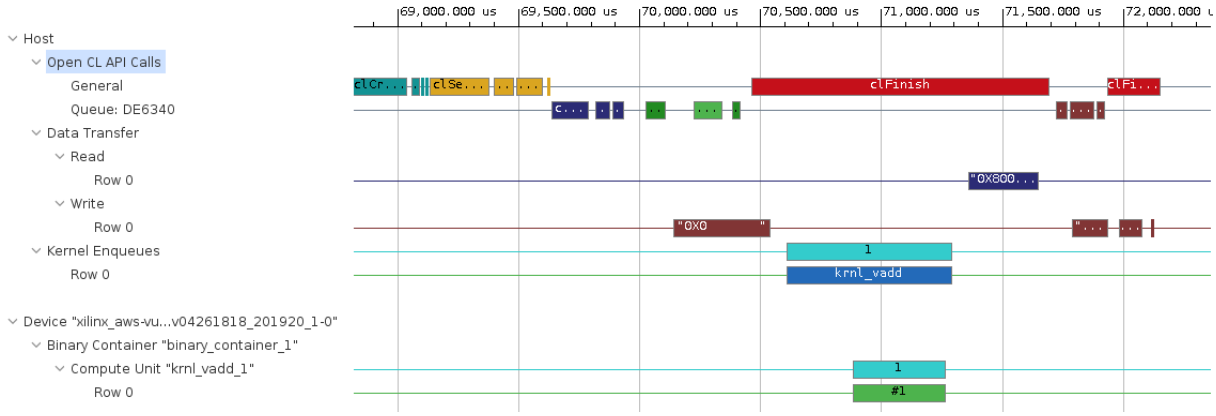
1. In the *Assistant* view, double click `Emulation-SW -> vadd-Default -> Run Summary (xclbin)` to open Vitis Analyzer



2. Vitis Analyzer shows **Profile Summary** and **Application Timeline** tabs on the left-hand side. Click **Application Timing**

3. Scroll right, click at around 75 ms (you may see different timeline depending on what else was executed earlier), then using mouse button, select the area of interest

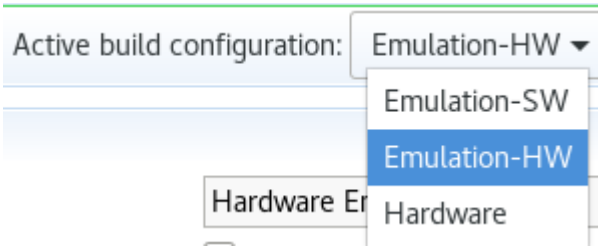
   Observe various activities in various regions



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

Build and run hardware emulation

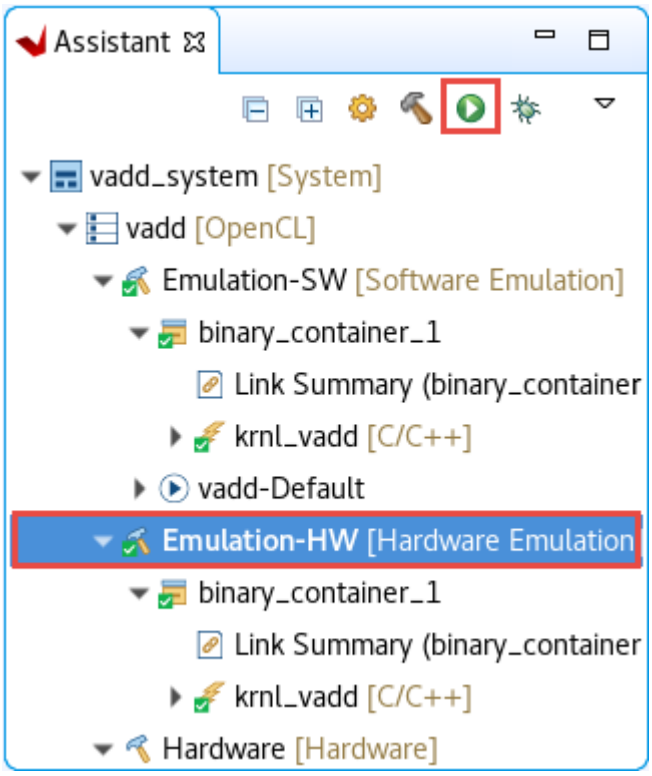
1. Set `Active build configuration:` to `Emulation-HW` on the upper right corner of *Project Editor* view



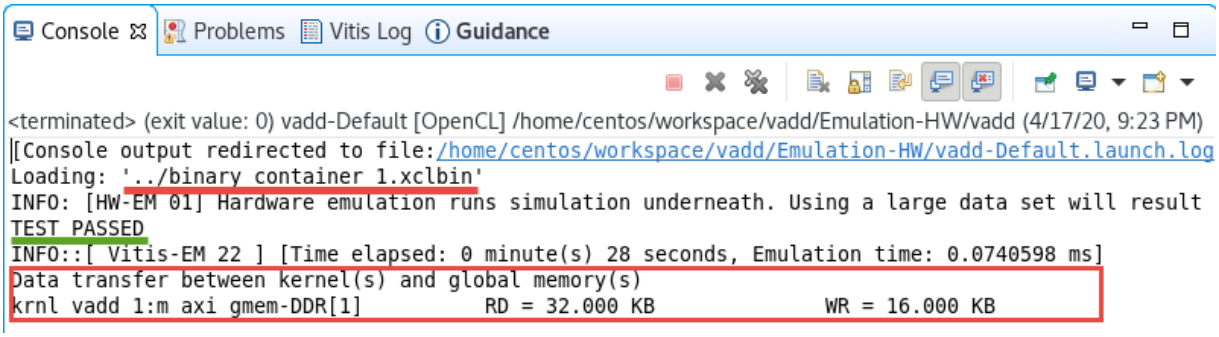
2. Build the project. This may take about 10 minutes

3. Run Hardware Emulation in GUI mode

   To launch hardware emulation in GUI mode, first select the application in *Explorer* view, then click run buttonn on icon bar, or select *Emulation-HW* in *Assistant* view and click on the Run button and select **vadd-Default (OpenCLK APplication)**



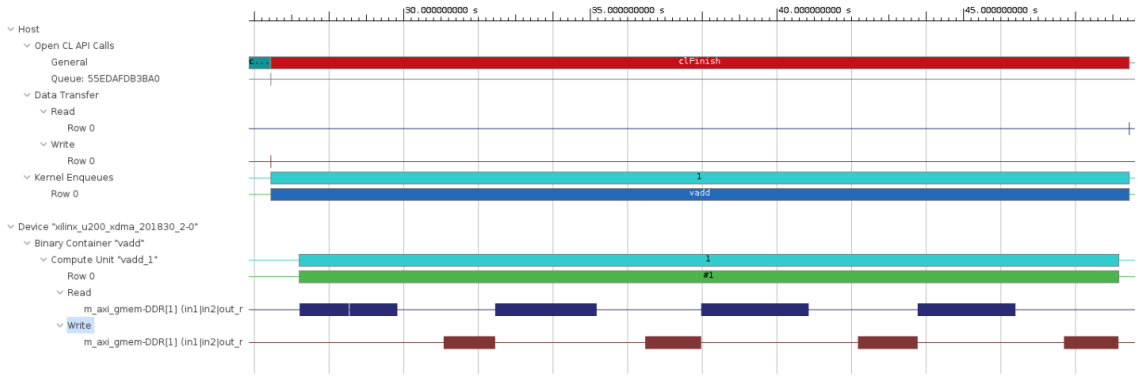
4. Observe the application is run and the output is displayed in the *Console* view. Compared to software emulation, the output also shows data transfer information. Notice the data transfer rate between kernel and global memory is 16 KB on each port



5. View Emulation Timeline

   In the *Assistant* view, double-click `Emulation-HW > vadd-Default > Run Summary (xclbin)` to open Vitis Analyzer

   Vitis Analyzer shows **System Diagram**, **Platform Diagram**, **Run Guidance**, **Profile Summary** and **Application Timeline** tabs on the left-hand side. Click **Application Timeline**. Zoom in between 25 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

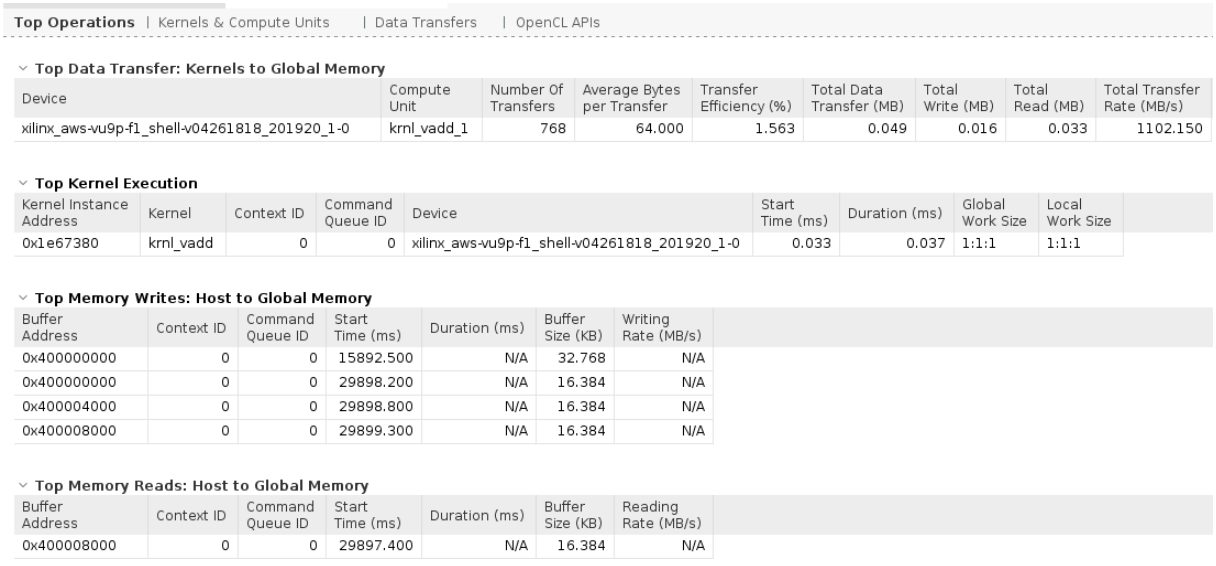


6. Click on the **Profile Summary** entry in the left panel, and observe multi-tab (four tabs) output

- **Top Operations** : Shows all the major top operations of 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
- **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. It is best to optimize the kernel to be as fast as it can be with the data it requires
- **Data Transfers** : This tab 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 throughput and bandwidth of the read/writes to the global memory that the host and kernel share
- **OpenCL APIs** : Shows all the OpenCL API command executions, how many time each was executed, and how long they take to execute

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

- Top Operations



Top Operations | Kernels & Compute Units | Data Transfers | OpenCL APIs

Top Data Transfer: Kernels to Global Memory

Device	Compute Unit	Number Of Transfers	Average Bytes per Transfer	Transfer Efficiency (%)	Total Data Transfer (MB)	Total Write (MB)	Total Read (MB)	Total Transfer Rate (MB/s)
xilinx_aws-vu9p-f1_shell-v04261818_201920_1-0	krnl_vadd_1	768	64.000	1.563	0.049	0.016	0.033	1102.150

Top Kernel Execution

Kernel Instance Address	Kernel	Context ID	Command Queue ID	Device	Start Time (ms)	Duration (ms)	Global Work Size	Local Work Size
0x1e67380	krnl_vadd	0	0	xilinx_aws-vu9p-f1_shell-v04261818_201920_1-0	0.033	0.037	1:1:1	1:1:1

Top Memory Writes: Host to Global Memory

Buffer Address	Context ID	Command Queue ID	Start Time (ms)	Duration (ms)	Buffer Size (KB)	Writing Rate (MB/s)
0x400000000	0	0	15892.500	N/A	32.768	N/A
0x400000000	0	0	29898.200	N/A	16.384	N/A
0x400004000	0	0	29898.800	N/A	16.384	N/A
0x400008000	0	0	29899.300	N/A	16.384	N/A

Top Memory Reads: Host to Global Memory

Buffer Address	Context ID	Command Queue ID	Start Time (ms)	Duration (ms)	Buffer Size (KB)	Reading Rate (MB/s)
0x400008000	0	0	29897.400	N/A	16.384	N/A

- Kernels & Compute Units

Top Operations | **Kernels & Compute Units** | Data Transfers | OpenCL APIs

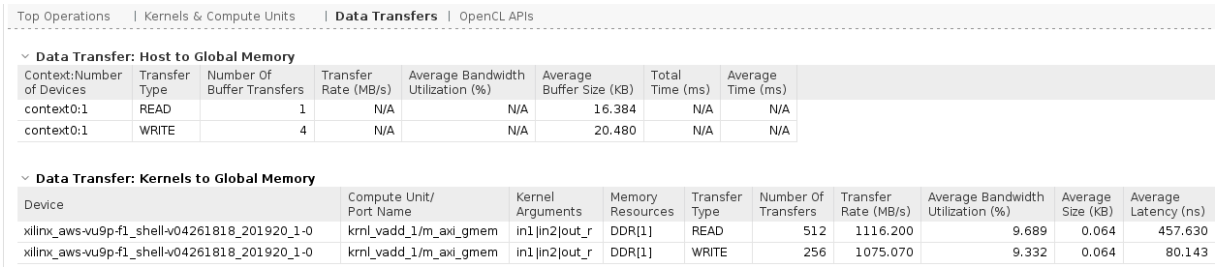
Kernel Execution (includes estimated device times)

Kernel	Number Of Enqueues	Total Time (ms)	Minimum Time (ms)	Average Time (ms)	Maximum Time (ms)
krnl_vadd	1	0.037	0.037	0.037	0.037

Compute Unit Utilization (includes estimated device times)

Device	Compute Unit	Kernel	Global Work Size	Local Work Size	Number Of Calls	Dataflow Execution	Max Parallel Executions	Dataflow Acceleration	Total Time (ms)	Minimum Time (ms)	Average Time (ms)	Maximum Time (ms)	Clock Freq (MHz)	CU Utilization (%)
xilinx_aws-vu9p-f1_shell-v04261818_201920_1-0	krnl_vadd_1	krnl_vadd	1:1:1	1:1:1	1	No	1	1.000000x	0.033	0.033	0.033	0.033	300	90.260

- Data Transfers



Top Operations | Kernels & Compute Units | **Data Transfers** | OpenCL APIs

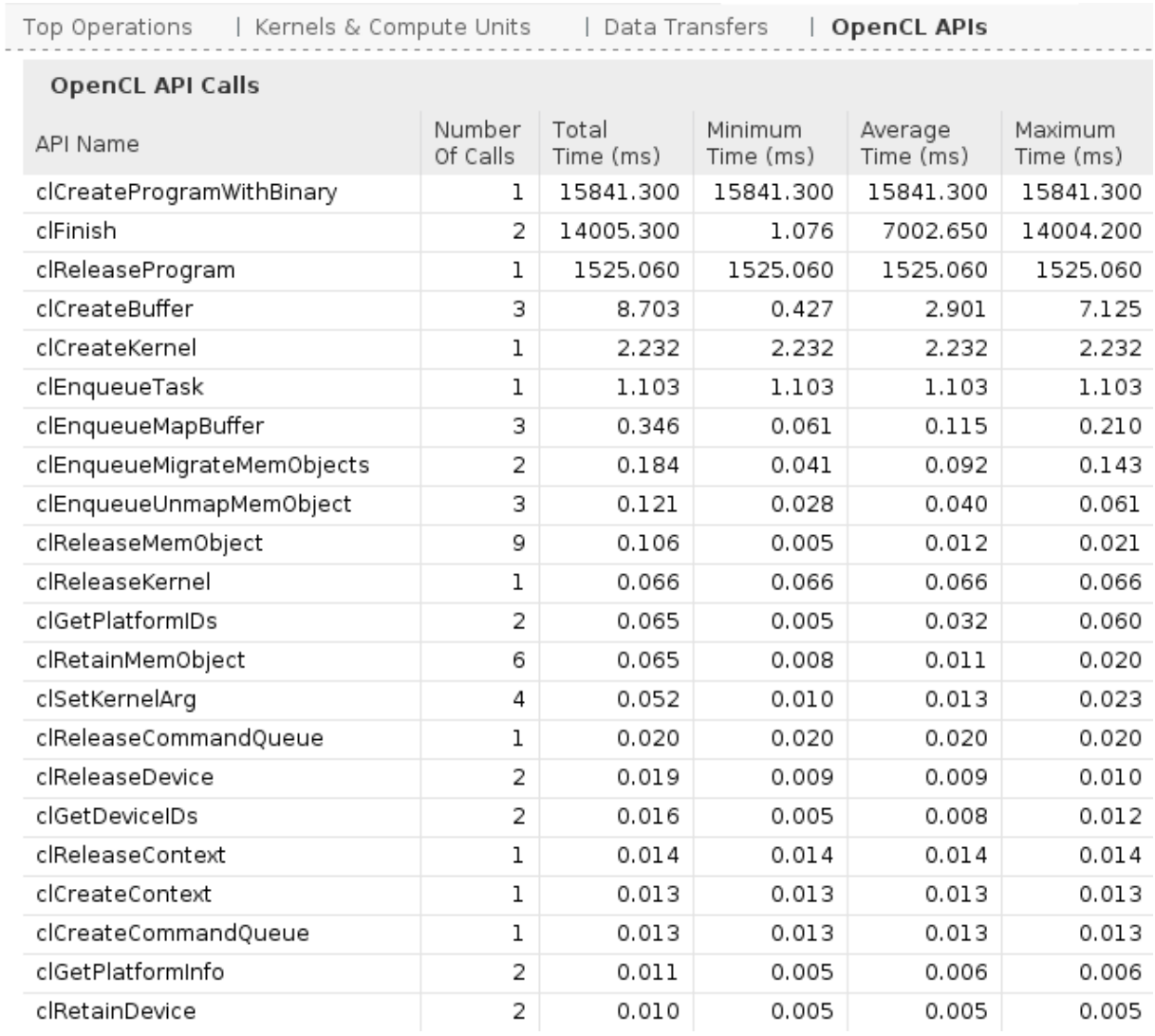
Data Transfer: Host to Global Memory

Context:Number of Devices	Transfer Type	Number Of Buffer Transfers	Transfer Rate (MB/s)	Average Bandwidth Utilization (%)	Average Buffer Size (KB)	Total Time (ms)	Average Time (ms)
context0:1	READ	1	N/A	N/A	16.384	N/A	N/A
context0:1	WRITE	4	N/A	N/A	20.480	N/A	N/A

Data Transfer: Kernels to Global Memory

Device	Compute Unit/ Port Name	Kernel Arguments	Memory Resources	Transfer Type	Number Of Transfers	Transfer Rate (MB/s)	Average Bandwidth Utilization (%)	Average Size (KB)	Average Latency (ns)
xilinx_aws-vu9p-f1_shell-v04261818_201920_1-0	krnl_vadd_1/m_axi_gmem	in1|in2|out_r	DDR[1]	READ	512	1116.200	9.689	0.064	457.630
xilinx_aws-vu9p-f1_shell-v04261818_201920_1-0	krnl_vadd_1/m_axi_gmem	in1|in2|out_r	DDR[1]	WRITE	256	1075.070	9.332	0.064	80.143

- OpenCL APIs



Top Operations | Kernels & Compute Units | Data Transfers | **OpenCL APIs**

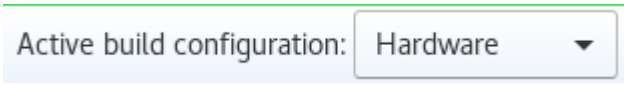
OpenCL API Calls

API Name	Number Of Calls	Total Time (ms)	Minimum Time (ms)	Average Time (ms)	Maximum Time (ms)
clCreateProgramWithBinary	1	15841.300	15841.300	15841.300	15841.300
clFinish	2	14005.300	1.076	7002.650	14004.200
clReleaseProgram	1	1525.060	1525.060	1525.060	1525.060
clCreateBuffer	3	8.703	0.427	2.901	7.125
clCreateKernel	1	2.232	2.232	2.232	2.232
clEnqueueTask	1	1.103	1.103	1.103	1.103
clEnqueueMapBuffer	3	0.346	0.061	0.115	0.210
clEnqueueMigrateMemObjects	2	0.184	0.041	0.092	0.143
clEnqueueUnmapMemObject	3	0.121	0.028	0.040	0.061
clReleaseMemObject	9	0.106	0.005	0.012	0.021
clReleaseKernel	1	0.066	0.066	0.066	0.066
clGetPlatformIDs	2	0.065	0.005	0.032	0.060
clRetainMemObject	6	0.065	0.008	0.011	0.020
clSetKernelArg	4	0.052	0.010	0.013	0.023
clReleaseCommandQueue	1	0.020	0.020	0.020	0.020
clReleaseDevice	2	0.019	0.009	0.009	0.010
clGetDeviceIDs	2	0.016	0.005	0.008	0.012
clReleaseContext	1	0.014	0.014	0.014	0.014
clCreateContext	1	0.013	0.013	0.013	0.013
clCreateCommandQueue	1	0.013	0.013	0.013	0.013
clGetPlatformInfo	2	0.011	0.005	0.006	0.006
clRetainDevice	2	0.010	0.005	0.005	0.005

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

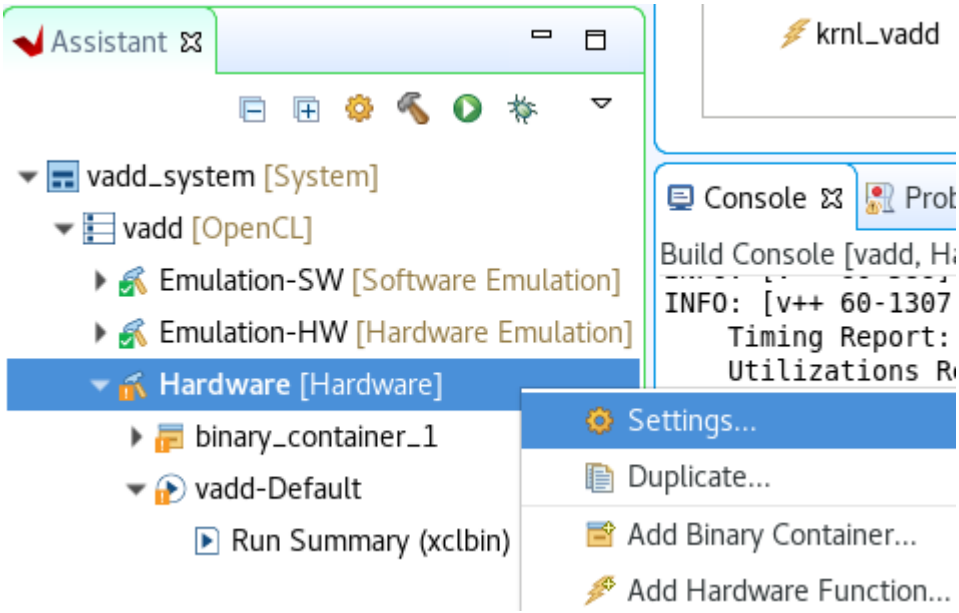
Build System hardware with profiling and timing analysis options

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

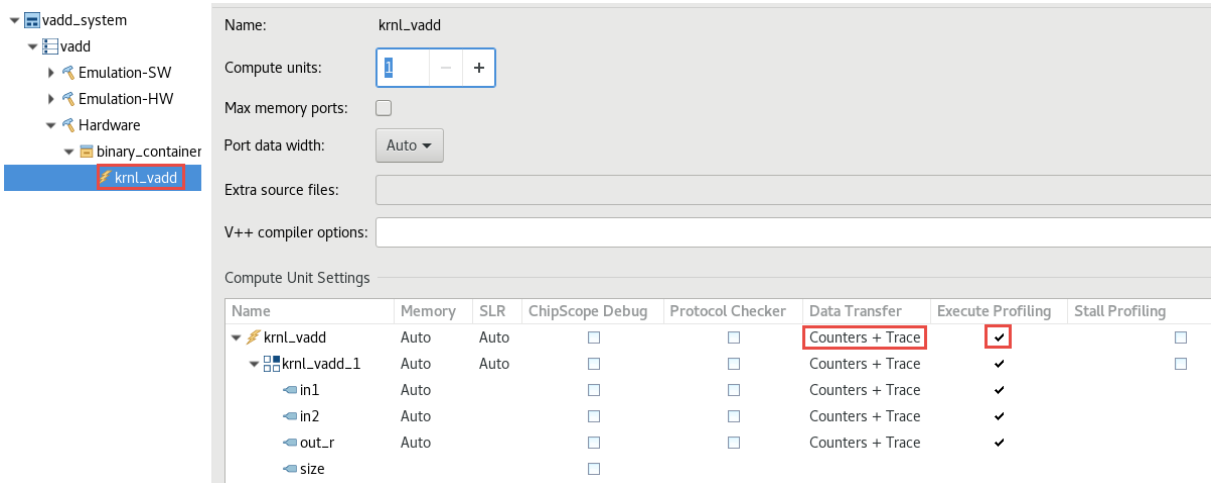


In order to collect the profiling data and run Timing Analyzer on the application run in hardware, we need to setup some options.

2. Right-click on Hardware in *Assistant* view and then click on *Settings*

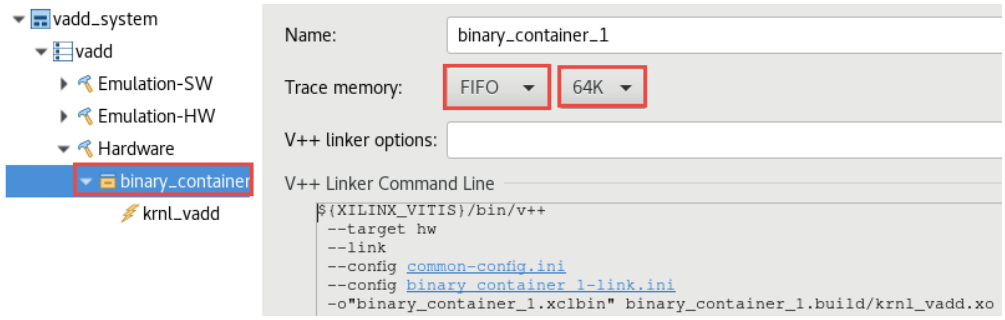


3. Expand Hardware in the left panel to see *binary_container* and *krnl_vadd* entries. Select *krnl_vadd* on the left-hand side, click on the *Data Transfer* drop-down button in krnl_vadd row and select *Counters+Trace* option. Notice that all lower-level entries get the same monitoring options. Similarly, click on *Execute Profiling* check-box in krnl_vadd row. At this point the settings should look like shown below



4. Click on *binary_container* on the left-hand side and select *Trace Memory* to be FIFO type and size of 64K. This is the memory where traces will be stored. You have options of storing also in DDR (max limit 2 GB) and PLRAM

5. Click **Apply and Close**



Build Full Hardware

Note that building the project can take around two hours.

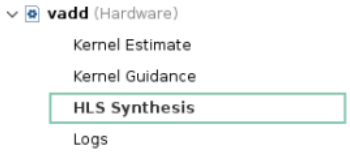
1. After having setting up all the options described above you can build the system

2. Click on the () button or select **Project > Build Project**

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

Get the FPGA Resources

In *Assistant* view, select `Hardware > vadd > Link Summary` in order to open Vitis Analyzer. From Vitis Analyzer you can get information about Utilization as well as other important information opening the Vivado HLS.



Utilization Estimates

Summary

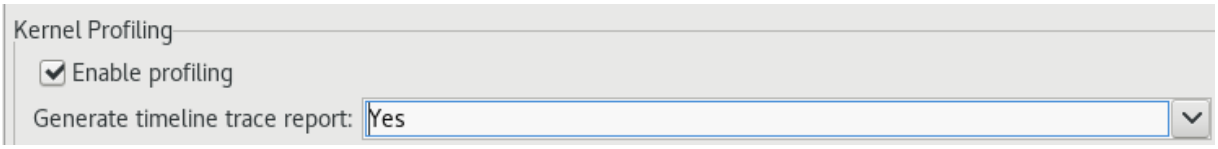
Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	722	-
FIFO	-	-	-	-	-
Instance	2	-	796	1068	-
Memory	6	-	0	0	0
Multiplexer	-	-	-	381	-
Register	-	-	1269	-	-
Total	8	0	2065	2171	0
Available	4320	6840	2364480	1182240	960
Available SLR	1440	2280	788160	394080	320
Utilization (%)	~0	0	~0	~0	0
Utilization SLR (%)	~0	0	~0	~0	0

Run the Design in target hardware and analyze output

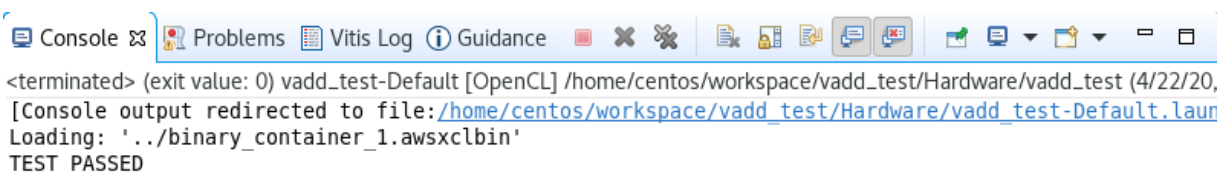
1. Setup the run configuration so you can run the application and then analyze results from GUI

2. Right-click on Hardware in *Assistant* view, select `Run > Run Configurations`

   Change Generate timeline trace report option from *Default* to *Yes* using the drop-down button in the Main tab.



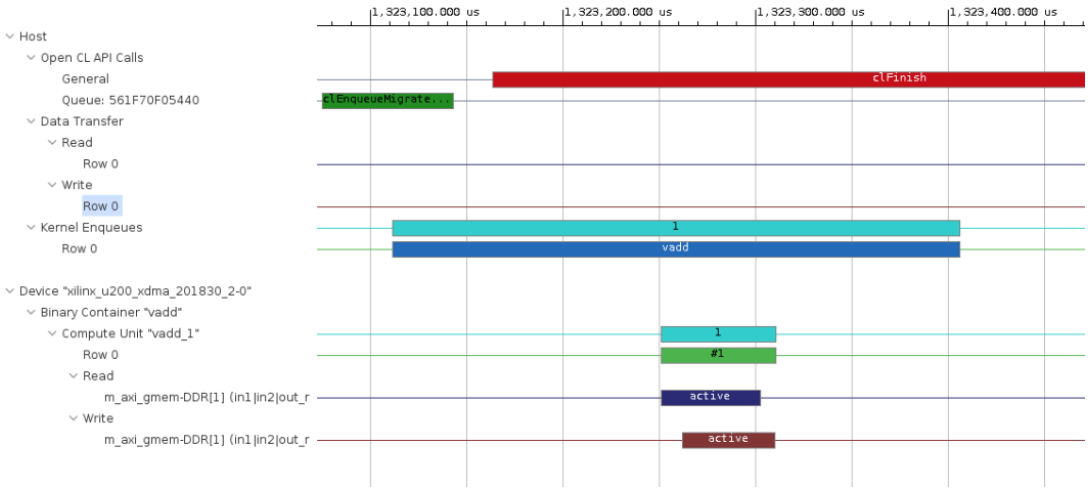
3. Execute the application by clicking **Apply** and then **Run**. The FPGA bitstream will be downloaded and the host application will be executed showing output similar to:



Analyze hardware application timeline and profile summary

1. In the *Assistant* view, double click `Hardware > vadd-Default > Run Summary (xclbin)` to open Vitis Analyzer

   Vitis Analyzer shows **Run Guidance**, **Profile Summary** and **Application Timeline** panels on the left-hand side. Click **Application Timeline**. Zoom in between 1,323,100 and 1,323,400 microsecond area (note for your output the range may differ depending on what else was executed on the instance) and observe the activities in various parts of the system. Note that the kernel processes data in one shot



2. Click on the *Profile Summary* entry in the left panel, and observe multi-tab (four tabs) output

   - Top Operations

Top Operations | Kernels & Compute Units | Data Transfers | OpenCL APIs

Top Data Transfer: Kernels to Global Memory

Device	Compute Unit	Number Of Transfers	Average Bytes per Transfer	Transfer Efficiency (%)	Total Data Transfer (MB)	Total Write (MB)	Total Read (MB)	Total Transfer Rate (MB/s)
xilinx_aws-vu9p-f1_dynamic_5_0-0	krnl_vadd_1	768	64.000	1.563	0.049	0.016	0.033	760.443

Top Kernel Execution

Kernel Instance Address	Kernel	Context ID	Command Queue ID	Device	Start Time (ms)	Duration (ms)	Global Work Size	Local Work Size
0x25a3f50	krnl_vadd	0	0	xilinx_aws-vu9p-f1_dynamic_5_0-0	3164.900	0.253	1:1:1	1:1:1

Top Memory Writes: Host to Global Memory

Buffer Address	Context ID	Command Queue ID	Start Time (ms)	Duration (ms)	Buffer Size (KB)	Writing Rate (MB/s)
0x800000000	0	0	3158.730	0.107	32.768	305.948
0x800000000	0	0	3165.390	0.073	16.384	225.662
0x800004000	0	0	3165.490	0.059	16.384	277.728
0x800008000	0	0	3165.560	0.001	16.384	12154.303

Top Memory Reads: Host to Global Memory

Buffer Address	Context ID	Command Queue ID	Start Time (ms)	Duration (ms)	Buffer Size (KB)	Reading Rate (MB/s)
0x800008000	0	0	3165.220	0.077	16.384	212.195

   - Kernels & Compute Units

Top Operations | **Kernels & Compute Units** | Data Transfers | OpenCL APIs

Kernel Execution

Kernel	Number Of Enqueues	Total Time (ms)	Minimum Time (ms)	Average Time (ms)	Maximum Time (ms)
krnl_vadd	1	0.253	0.253	0.253	0.253

Compute Unit Utilization

Device	Compute Unit	Kernel	Global Work Size	Local Work Size	Number Of Calls	Dataflow Execution	Max Parallel Executions	Dataflow Acceleration	Total Time (ms)	Minimum Time (ms)	Average Time (ms)	Maximum Time (ms)	Clock Freq (MHz)	CU Utilization (%)
xilinx_aws-vu9p-f1_dynamic_5_0-0	krnl_vadd_1	krnl_vadd	1:1:1	1:1:1	1	No	1	1.000000x	0.052	0.052	0.052	0.052	250	20.450

Compute Units: Stall Information

No Data. Please use 'v++ -l --profile_kernel stall' to monitor and report kernel stall information.

- Data Transfers

Top Operations | Kernels & Compute Units | **Data Transfers** | OpenCL APIs

Data Transfer: Host to Global Memory

Context:Number of Devices	Transfer Type	Number Of Buffer Transfers	Transfer Rate (MB/s)	Average Bandwidth Utilization (%)	Average Buffer Size (KB)	Total Time (ms)	Average Time (ms)
context0:1	READ	1	212.195	2.210	16.384	0.077	0.077
context0:1	WRITE	4	341.265	3.555	20.480	0.240	0.060

Data Transfer: Kernels to Global Memory

Device	Compute Unit/ Port Name	Kernel Arguments	Memory Resources	Transfer Type	Number Of Transfers	Transfer Rate (MB/s)	Average Bandwidth Utilization (%)	Average Size (KB)	Average Latency (ns)
xilinx_aws-vu9p-f1_dynamic_5_0-0	krnl_vadd_1/m_axi_gmem	in1|in2|out_r	DDR[0]	READ	512	763.040	6.624	0.064	829.211
xilinx_aws-vu9p-f1_dynamic_5_0-0	krnl_vadd_1/m_axi_gmem	in1|in2|out_r	DDR[0]	WRITE	256	755.301	6.556	0.064	276.391

- OpenCL APIs

Top Operations | Kernels & Compute Units | Data Transfers | **OpenCL APIs**

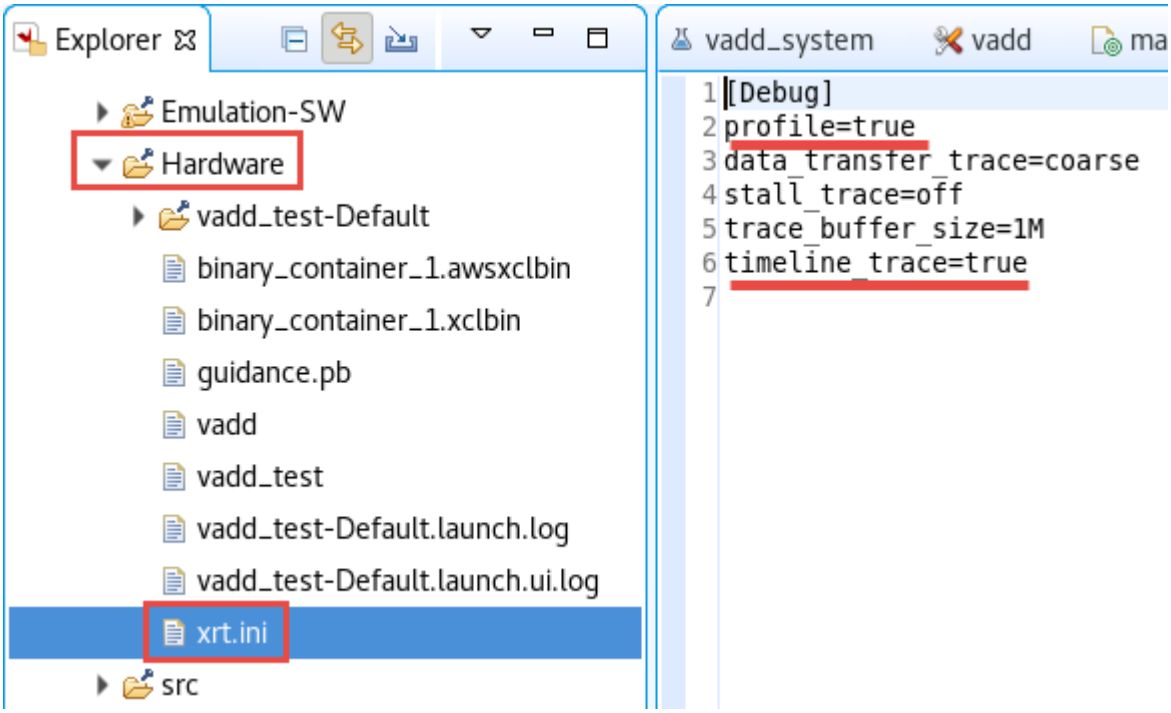
OpenCL API Calls

API Name	Number Of Calls	Total Time (ms)	Minimum Time (ms)	Average Time (ms)	Maximum Time (ms)
clCreateProgramWithBinary	1	3093.520	3093.520	3093.520	3093.520
clReleaseProgram	1	32.215	32.215	32.215	32.215
clEnqueueTask	1	6.265	6.265	6.265	6.265
clCreateKernel	1	1.343	1.343	1.343	1.343
clFinish	2	0.486	0.198	0.243	0.288
clEnqueueMapBuffer	3	0.237	0.045	0.079	0.132
clCreateBuffer	3	0.093	0.018	0.031	0.055
clReleaseKernel	1	0.077	0.077	0.077	0.077
clEnqueueMigrateMemObjects	2	0.068	0.029	0.034	0.039
clEnqueueUnmapMemObject	3	0.040	0.005	0.013	0.028
clGetPlatformIDs	2	0.018	0.002	0.009	0.016
clReleaseMemObject	9	0.016	0.001	0.002	0.003
clRetainMemObject	6	0.013	0.001	0.002	0.004
clCreateContext	1	0.013	0.013	0.013	0.013
clSetKernelArg	4	0.011	0.001	0.003	0.006
clGetDeviceIDs	2	0.009	0.002	0.004	0.007
clCreateCommandQueue	1	0.006	0.006	0.006	0.006
clReleaseCommandQueue	1	0.004	0.004	0.004	0.004
clReleaseDevice	2	0.004	0.002	0.002	0.002
clGetPlatformInfo	2	0.004	0.002	0.002	0.002
clReleaseContext	1	0.003	0.003	0.003	0.003
clRetainDevice	2	0.003	0.001	0.002	0.002

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

4. Review `xrt.ini` file in `Hardware` folder within *Explorer* view

   Earlier, when you set kernel profiling and trace settings, `xrt.ini` file gets updated. During the execution, this updated file is used to generate the profile and application timeline data which are seen using Vitis Analyzer.



Conclusion

In this lab, you used Vitis IDE to create a project using one of the application templates. You then ran the design using the software and hardware emulation flows, and reviewed the reports. You also analyzed profile and application timeline reports generated during running the application in actual hardware.