

XUP Vitis Labs (2019.2)

1. Setup Vitis	2. Introduction to Vitis	3. Improving Performance	4. Optimization	5. RTL Kernel Wizard	6. Debugging	7. Vision Application	8. PYNQ Lab
----------------------	--------------------------------	-----------------------------	--------------------	----------------------------	-----------------	--------------------------	-------------------

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 AWS F1 instance using pre-generated and pre-registered bitstream.

Check [Creating a Vitis IDE Project](#) to know more about Vitis IDE.

Description of example application

This lab uses a standard application template available in Vitis. It consists of a C++ 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.

You will compile and check a software only version of the application. The *vector add* OpenCL kernel will then be implemented as a hardware kernel. You will first build an emulation version of the design and run a simulation of the hardware kernel. You will then test the application with the kernel running in the FPGA.

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. Launch Vitis GUI

Make sure Vitis and XRT environment has been setup. XRT setup is necessary before launching Vitis because building and running acceleration applications requires XRT. To do so, the following commands should return a valid path.

```
echo $XILINX_VITIS
echo $XILINX_XRT
```

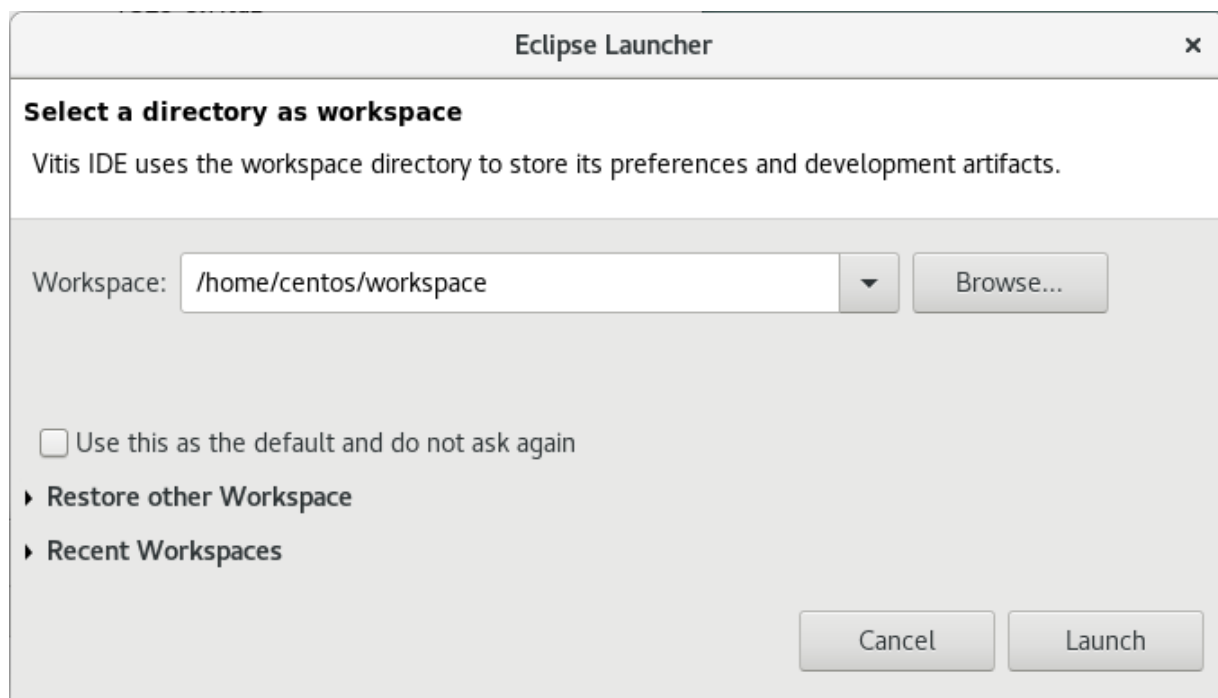
If any of the above command fails, then execute the following two commands from `aws-fpga` directory

```
cd ~/aws-fpga
source vitis_setup.sh
source vitis_runtime_setup.sh
```

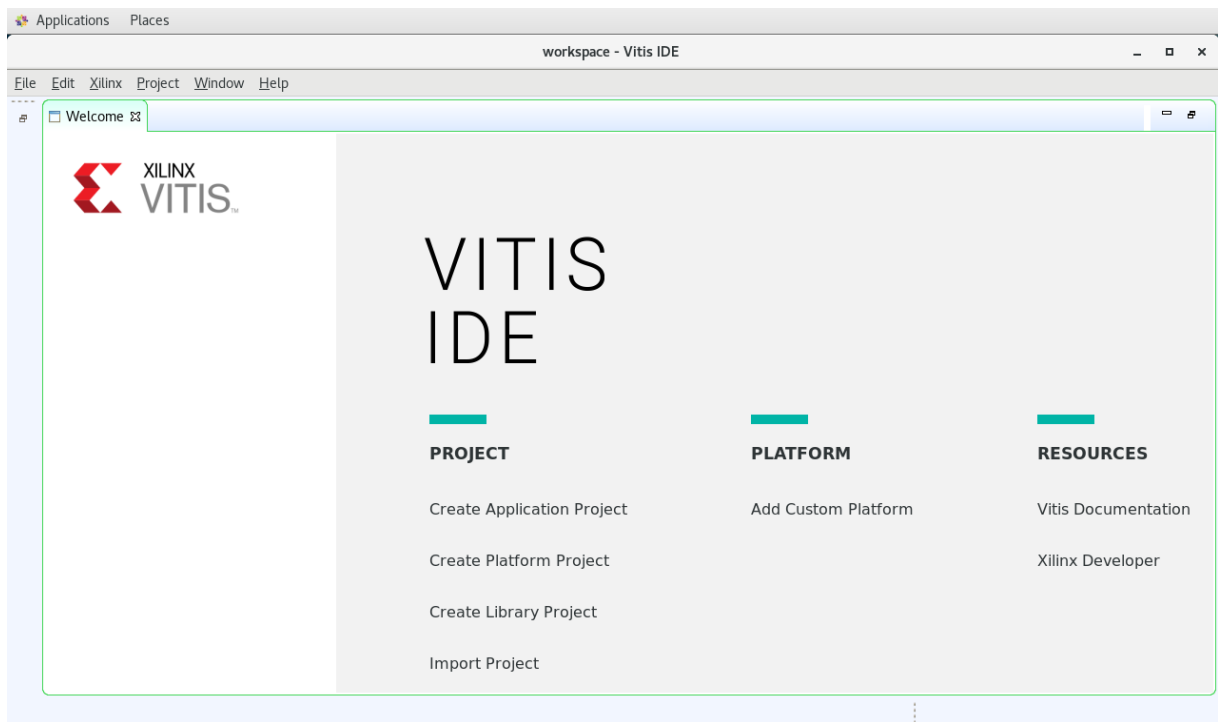
2. Invoke GUI by executing the following command:

```
vitis &
```

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

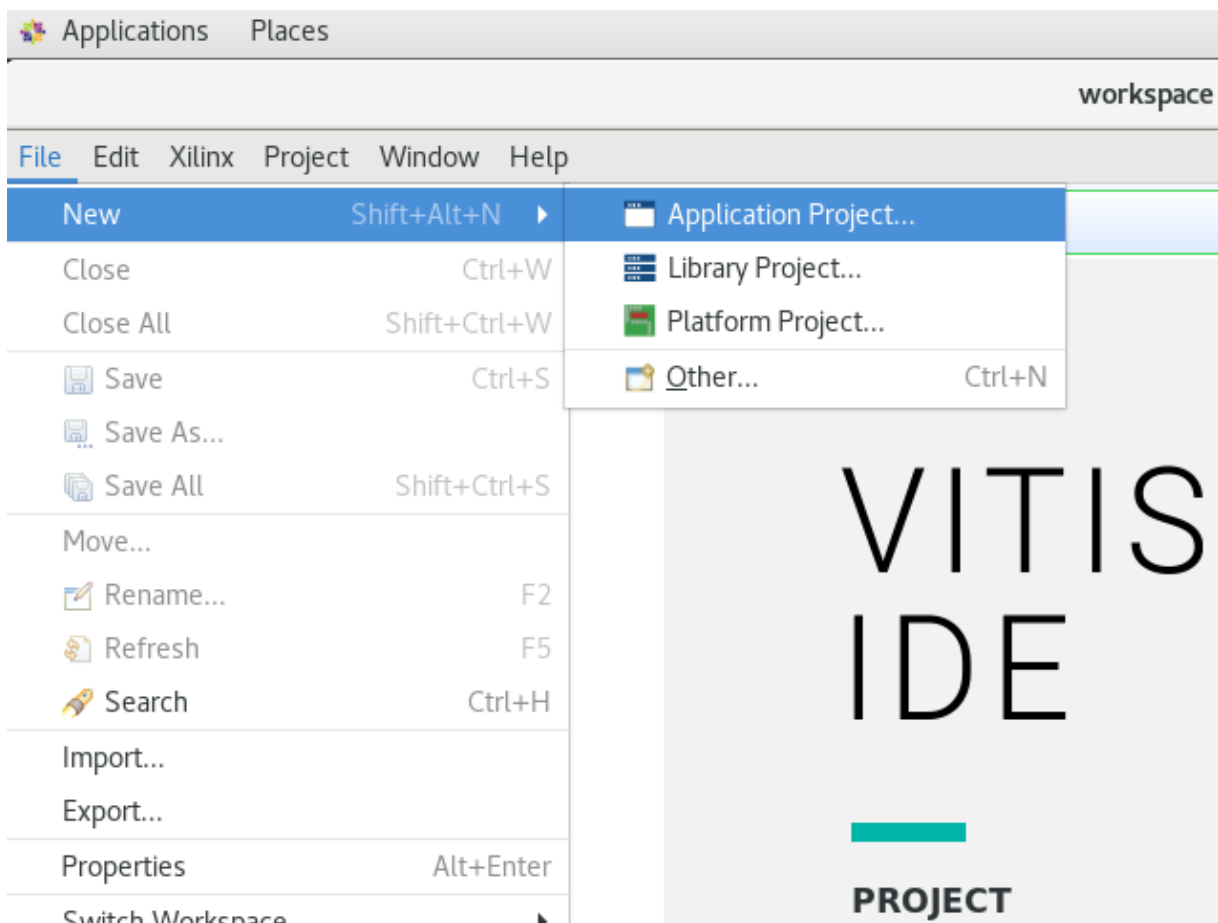


4. The Vitis IDE Welcome page will be displayed, if new a workspace is assigned



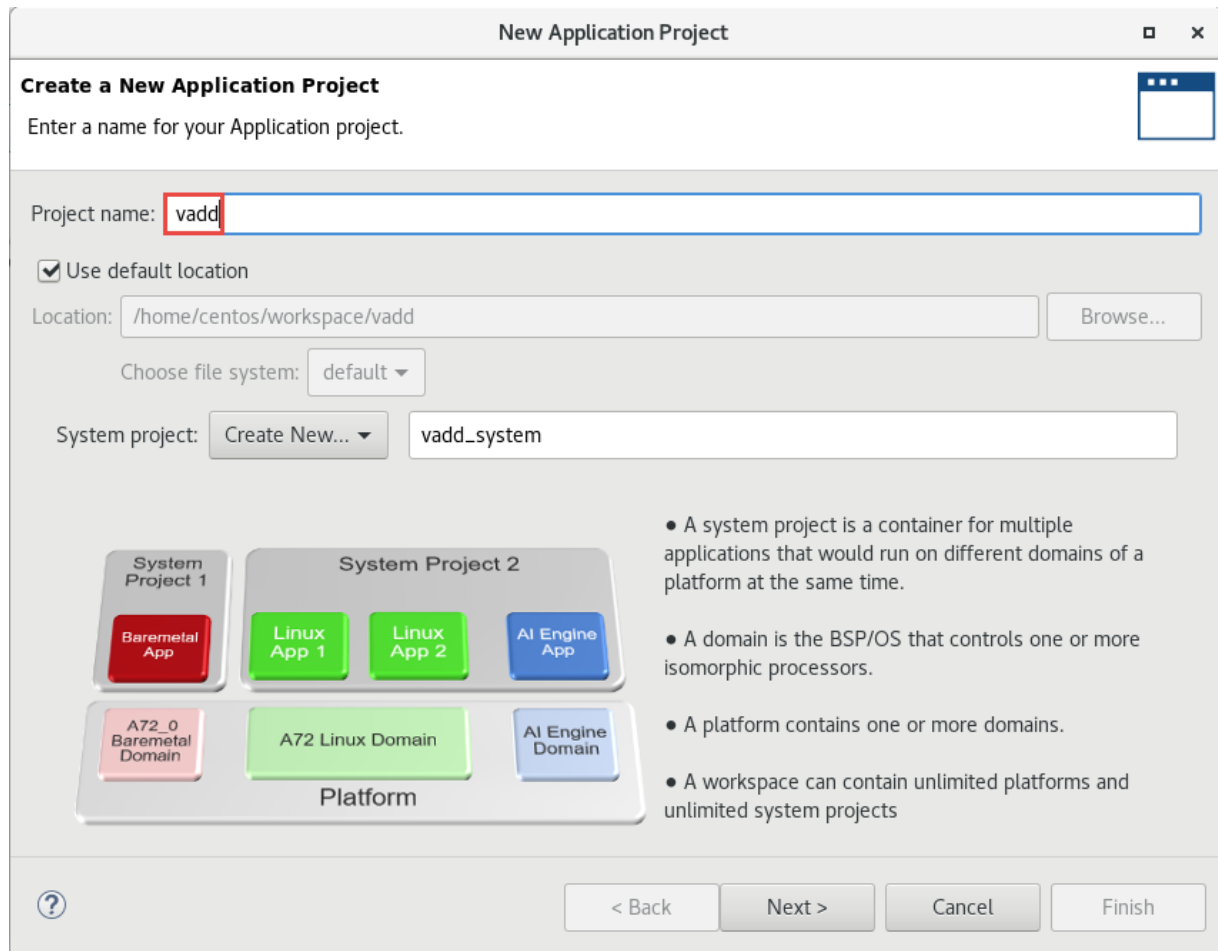
5. Create a new application project

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



6. Close Welcome page, if it was opened

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



New Application Project

Create a New Application Project

Enter a name for your Application project.

Project name:

☒ Use default location

Location:

Choose file system:

System project:

System Project 1

Baremetal App

A72_0 Baremetal Domain

System Project 2

Linux App 1

Linux App 2

AI Engine App

A72 Linux Domain

AI Engine Domain

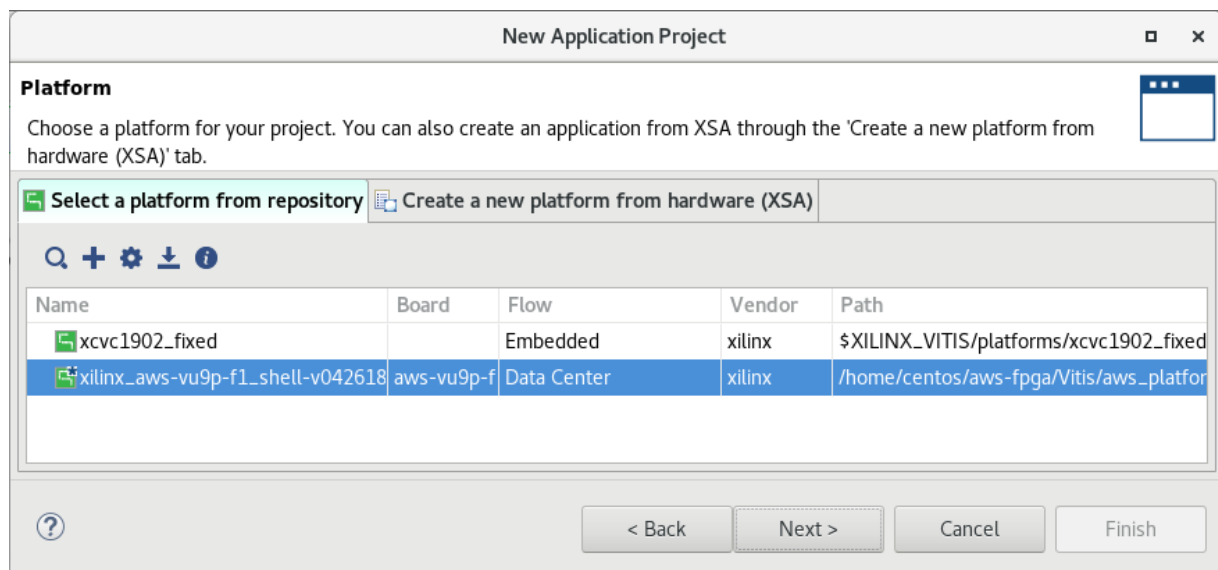
Platform

- A system project is a container for multiple applications that would run on different domains of a platform at the same time.
- A domain is the BSP/OS that controls one or more isomorphic processors.
- A platform contains one or more domains.
- A workspace can contain unlimited platforms and unlimited system projects

8. You will see one platform, named `xcvc1902_fixed`, for the embedded flow

9. Click on the '+' button and browse to `~/aws-fpga/Vitis/aws_platform`, select `xilinx_aws-vu9p-f1_shell-v04261818_201920_1`, and click **OK**



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



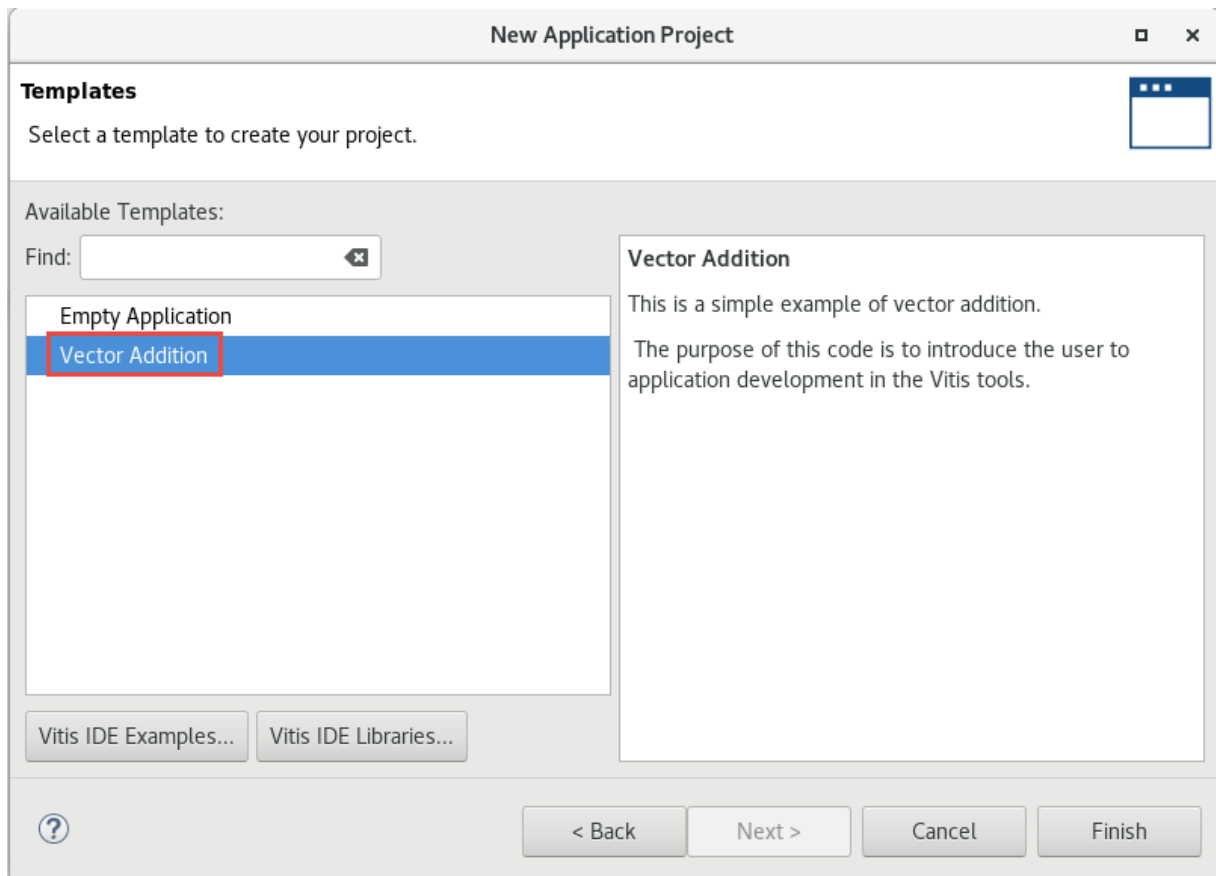
New Application Project

Platform

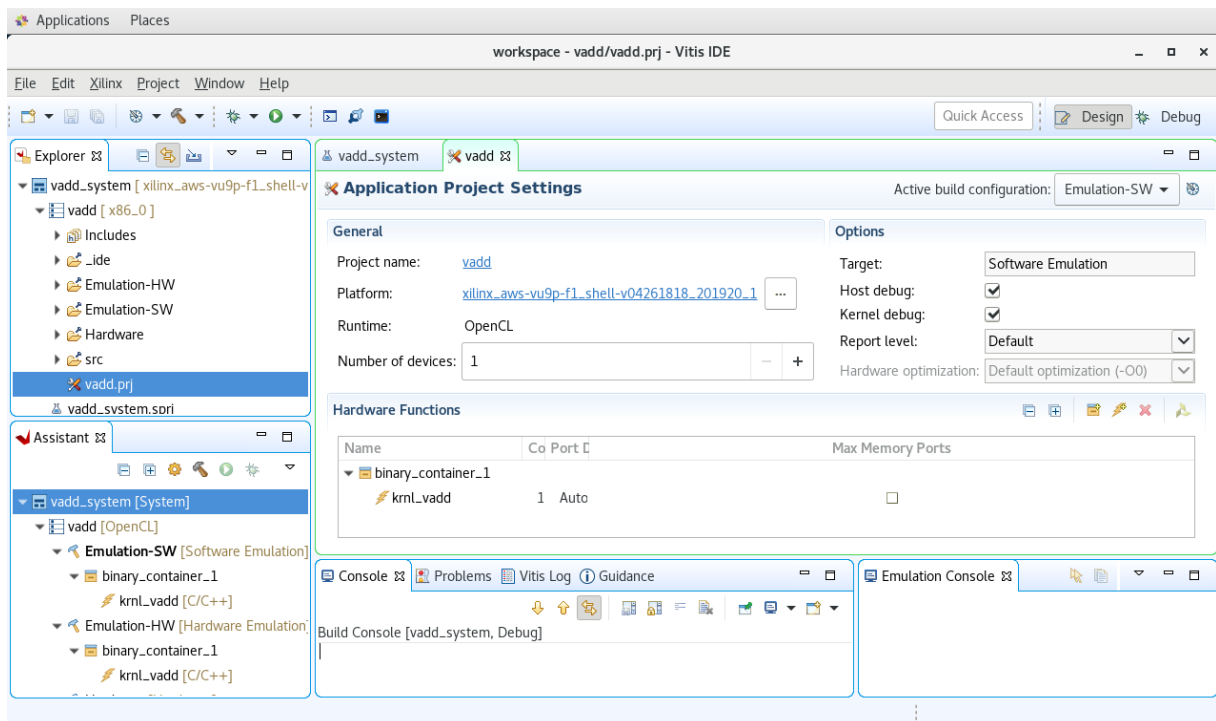
Choose a platform for your project. You can also create an application from XSA through the 'Create a new platform from hardware (XSA)' tab.

Name	Board	Flow	Vendor	Path
 xcvc1902_fixed		Embedded	xilinx	\$XILINX_VITIS/platforms/xcvc1902_fixed
 xilinx_aws-vu9p-f1_shell-v04261818_201920_1	aws-vu9p-f	Data Center	xilinx	/home/centos/aws-fpga/Vitis/aws_platform

11. Select `Vector Addition`, and click **Finish**

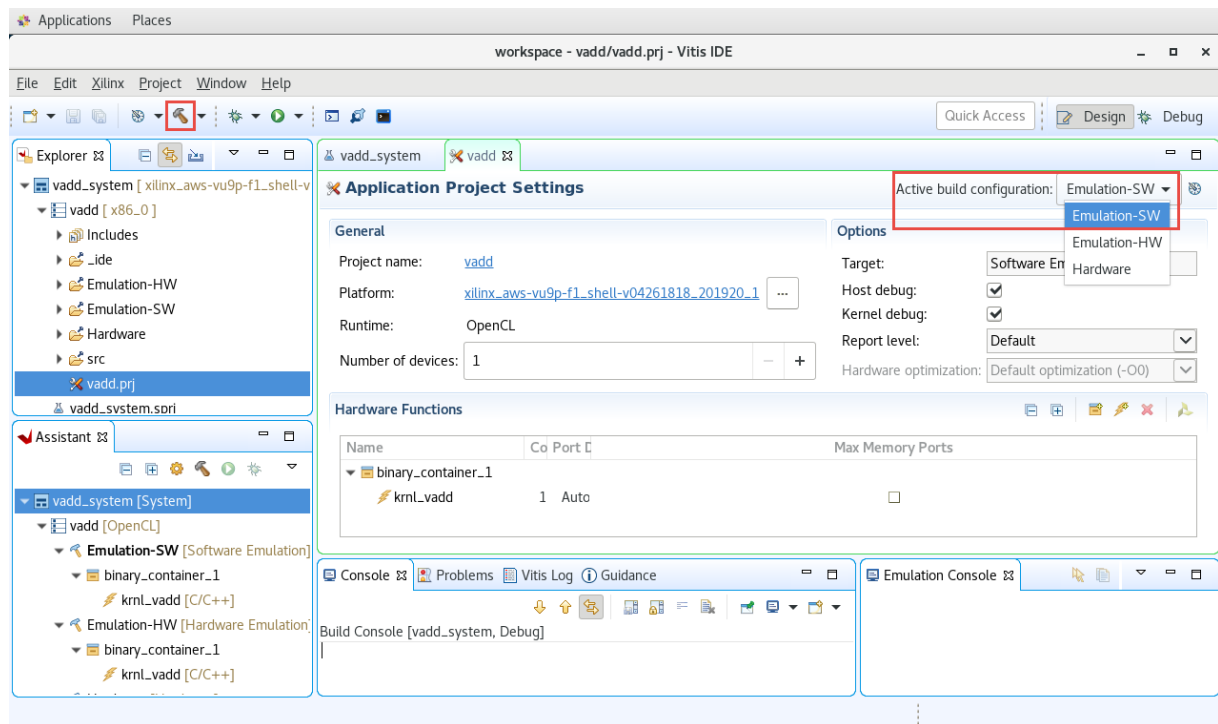


12. The project is generated. Notice that the *Hardware Function* in the *Project Editor* view is automatically set up to *knl_vadd*



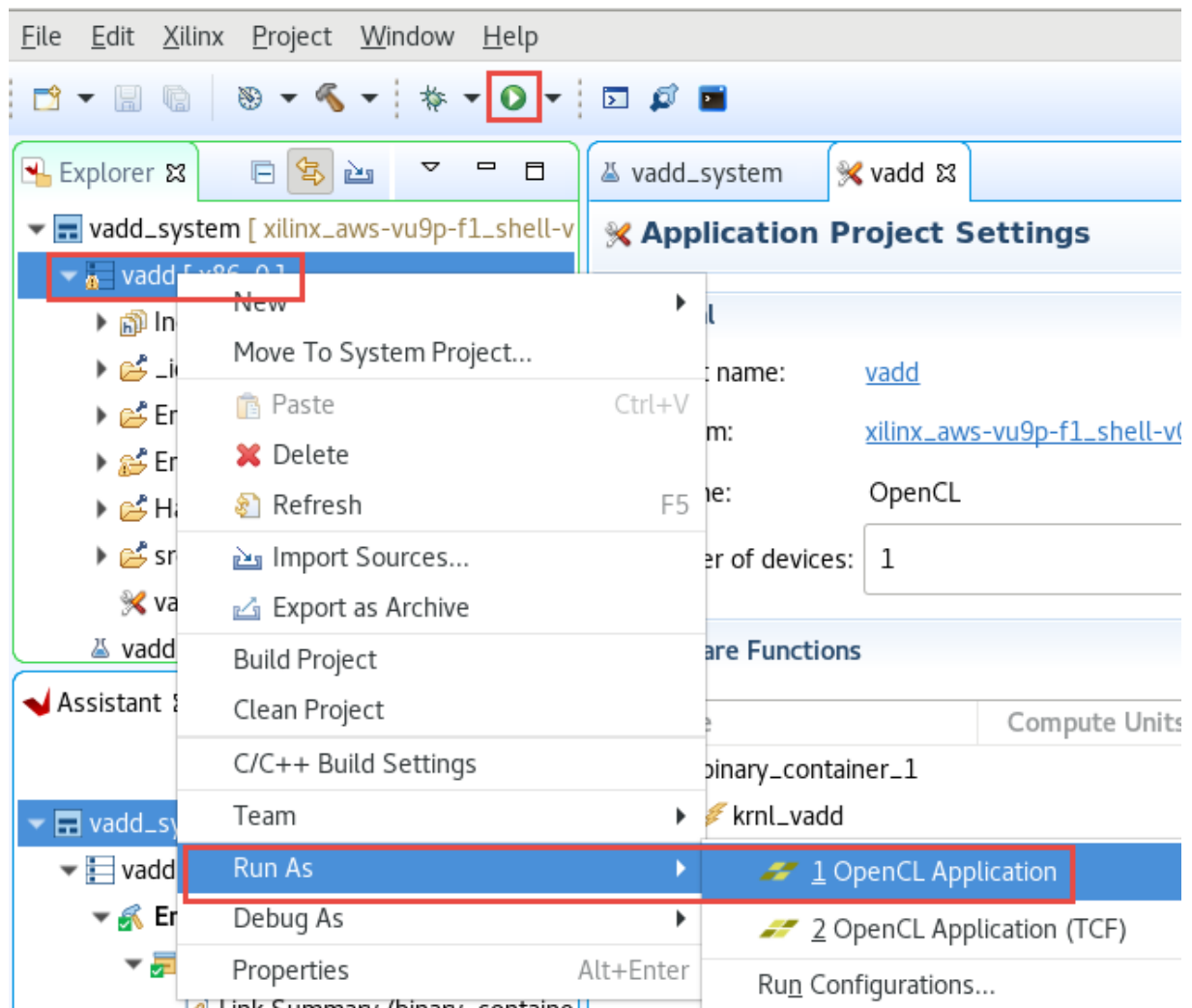
Build and Run Software Emulation

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...**

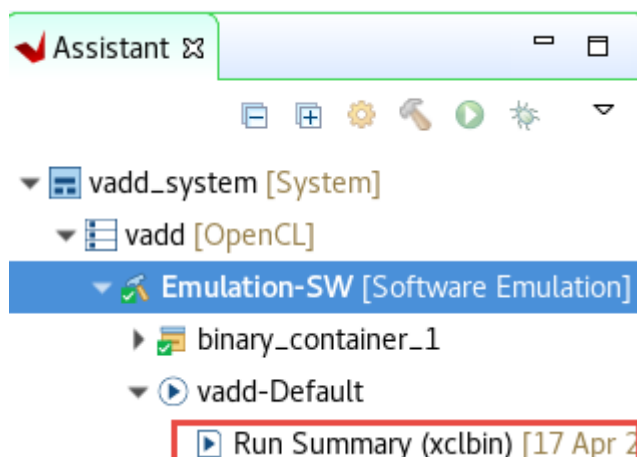


4. Observe the application is run and the output is displayed in the *Console* view

```
<terminated> Emulation-SW, vadd-Default, vadd (4/17/20, 9:03 PM)
Loading: '../binary_container_1.xclbin'
TEST PASSED
```

View Emulation Timeline

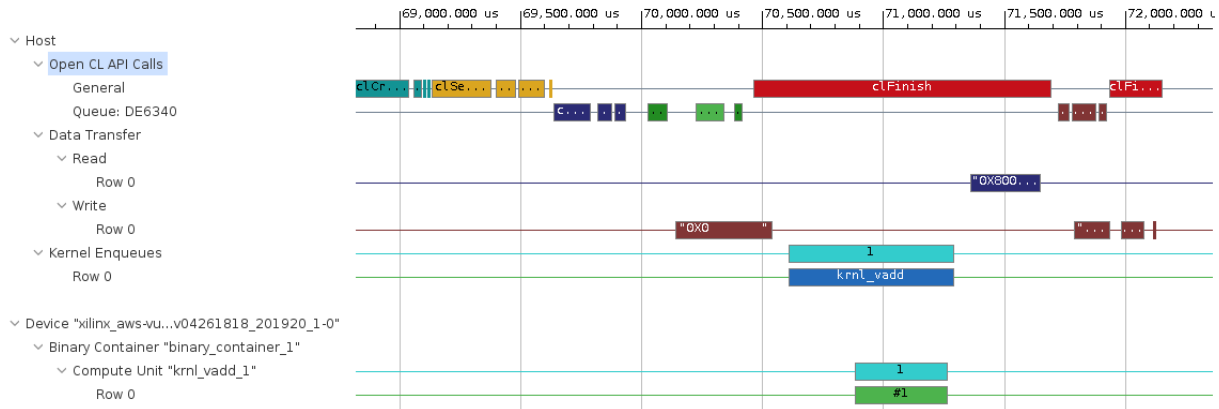
1. In the *Assistant* view, double click `Emulation-SW -> vadd-Default -> Run Summary (xclbin)` to open Vitis Analyzer



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

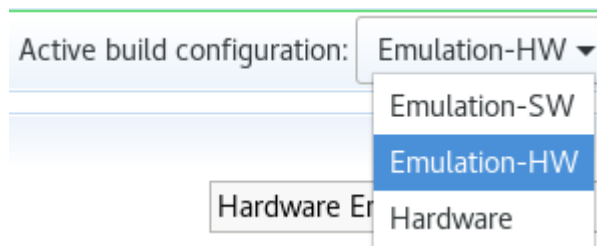
Observe various activities in various regions



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

Build and run hardware emulation

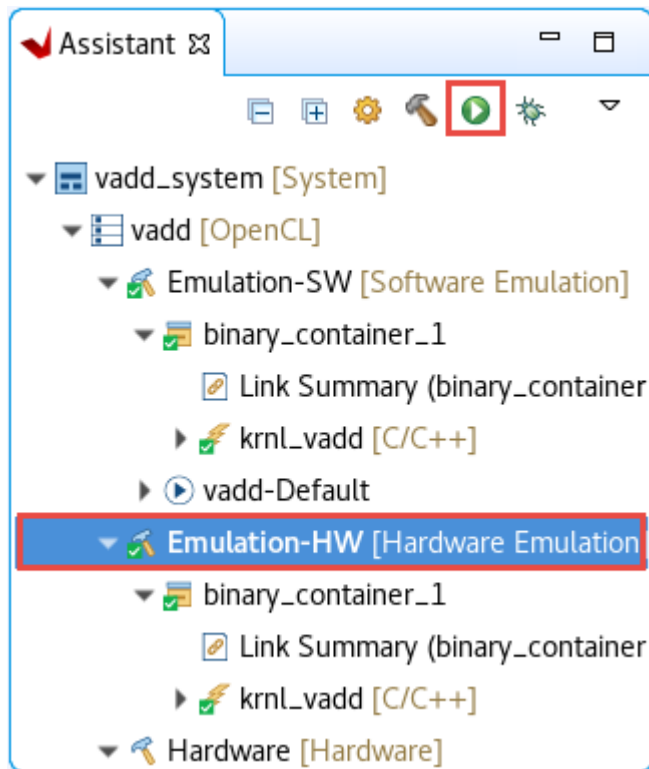
1. Set **Active build configuration:** to **Emulation-HW** on the upper right corner of *Project Editor* view



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 button on icon bar, or select *Emulation-HW* in *Assistant* view and click on the Run button and select **vadd-Default (OpenCL Application)**



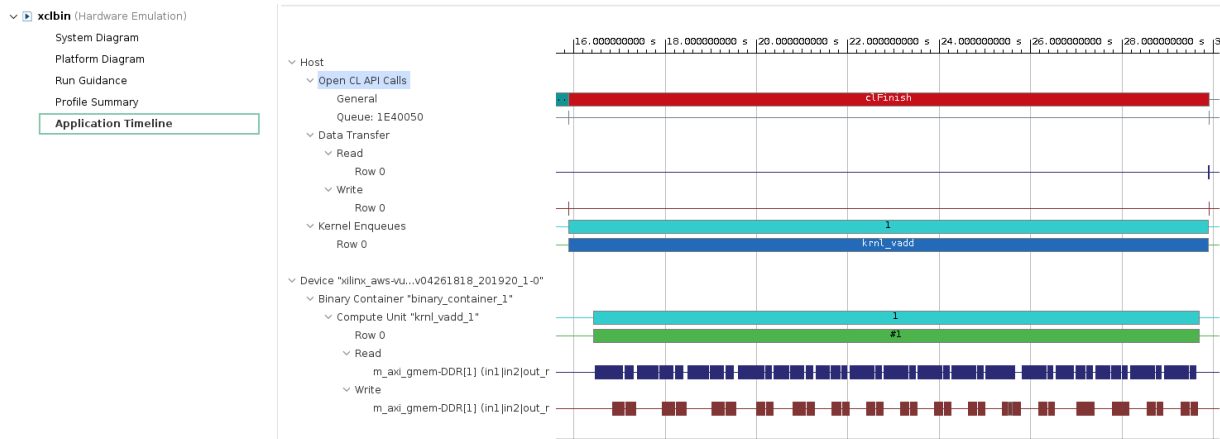
4. Observe the application is run and the output is displayed in the *Console* view. Compared to software emulation, the output also shows data transfer information. Notice the data transfer rate between kernel and global memory is 16 KB on each port

```
<terminated> (exit value: 0) vadd-Default [OpenCL] /home/centos/workspace/vadd/Emulation-HW/vadd (4/17/20, 9:23 PM)
[[Console output redirected to file:/home/centos/workspace/vadd/Emulation-HW/vadd-Default.launch.log
Loading: '../binary container 1.xclbin'
INFO: [HW-EM 01] Hardware emulation runs simulation underneath. Using a large data set will result
TEST PASSED
INFO::[ Vitis-EM 22 ] [Time elapsed: 0 minute(s) 28 seconds, Emulation time: 0.0740598 ms]
Data transfer between kernel(s) and global memory(s)
krnl vadd 1:m axi gmem-DDR[1]          RD = 32.000 KB          WR = 16.000 KB
```

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 16 and 30 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

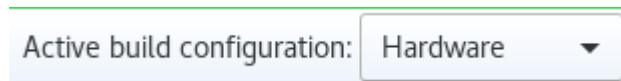
- OpenCL APIs

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**

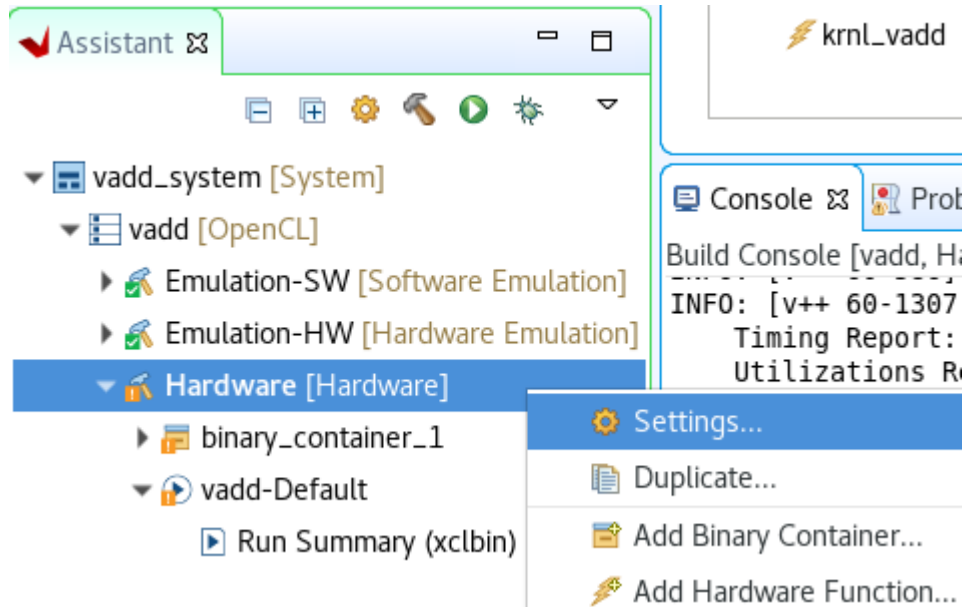
Build System hardware with profiling and timing analysis options

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

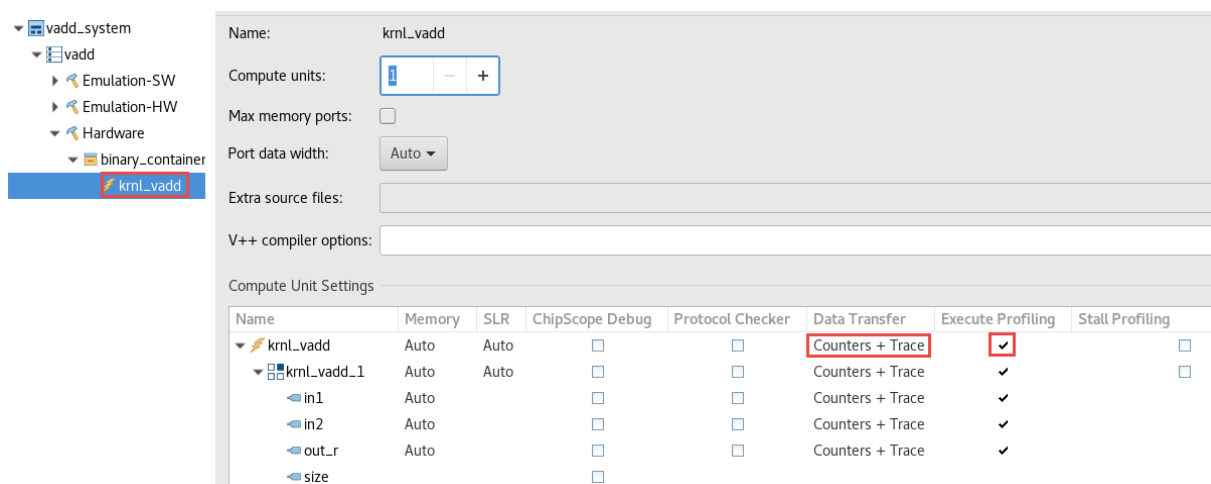


In order to collect the profiling data and run Timing Analyzer on the application run in hardware, we need to setup some options.

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

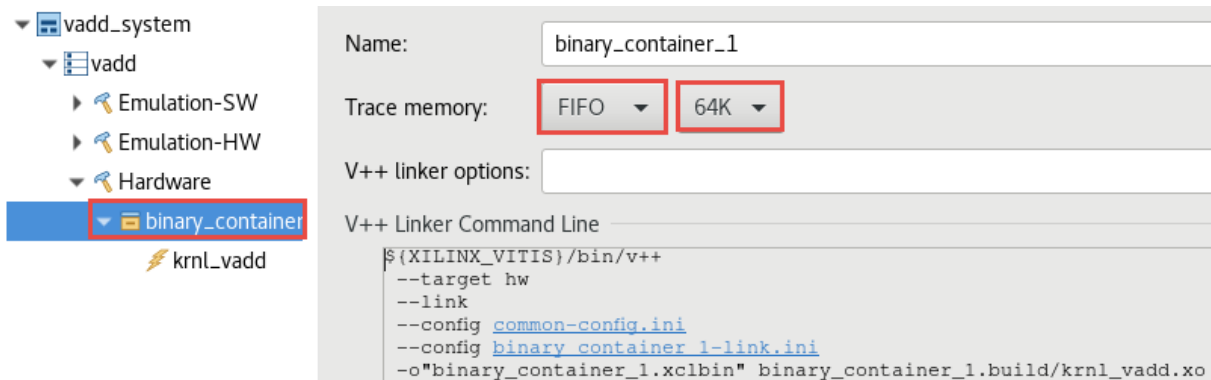


3. Expand **Hardware** in the left panel to see *binary_container* and *krrl_vadd* entries. Select *krrl_vadd* on the left-hand side, click on the *Data Transfer* drop-down button in *krrl_vadd* row and select *Counters+Trace* option. Notice that all lower-level entries get the same monitoring options. Similarly, click on *Execute Profiling* check-box in *krrl_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**



Normally, you would build the hardware, but since it will take approximately two hours you should **NOT BUILD** it here. Instead use the precompiled solution. See the [Appendix](#) below for instructions on how to build the hardware

Run the Design in target hardware and analyze output

Since the Hardware build and AFI availability takes considerable amount of time, a precompiled and preregistered version is provided. Use the precompiled solution directory to verify the functionality

1. Change to the solution directory by executing the following command

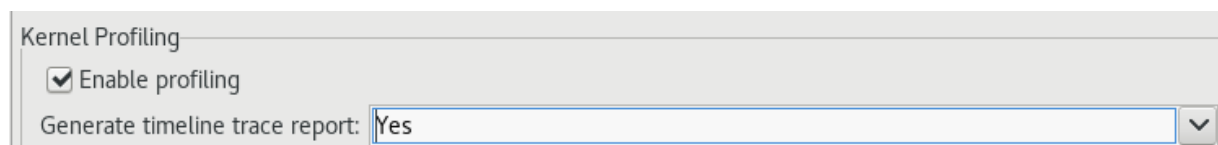
```
cd ~/compute_acceleration/solutions/gui_flow
```

2. Copy the *binary_container_1.xclbin*, *binary_container_1.awsxclbin*, and *vadd* files into `~/workspace/vadd/Hardware` folder. Make sure *vadd* has executable permissions. Use the following commands:

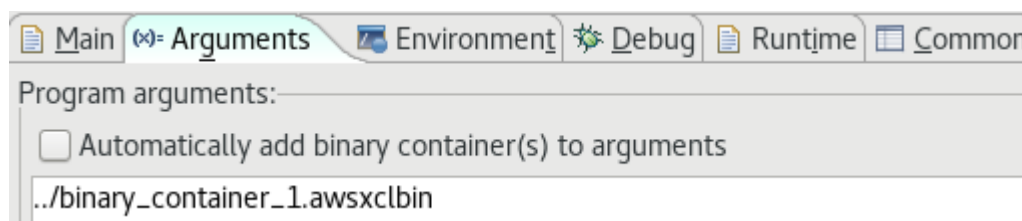
```
cp ~/compute_acceleration/solutions/gui_flow/* ~/workspace/vadd/Hardware/.
chmod +x ~/workspace/vadd/Hardware/vadd
```

3. Setup the run configuration so you can run the application and then analyze results from GUI
4. 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.



5. Click Arguments tab, uncheck the *Automatically add binary container(s) to arguments* option, and then enter `../binary_container_1.awsxclbin`



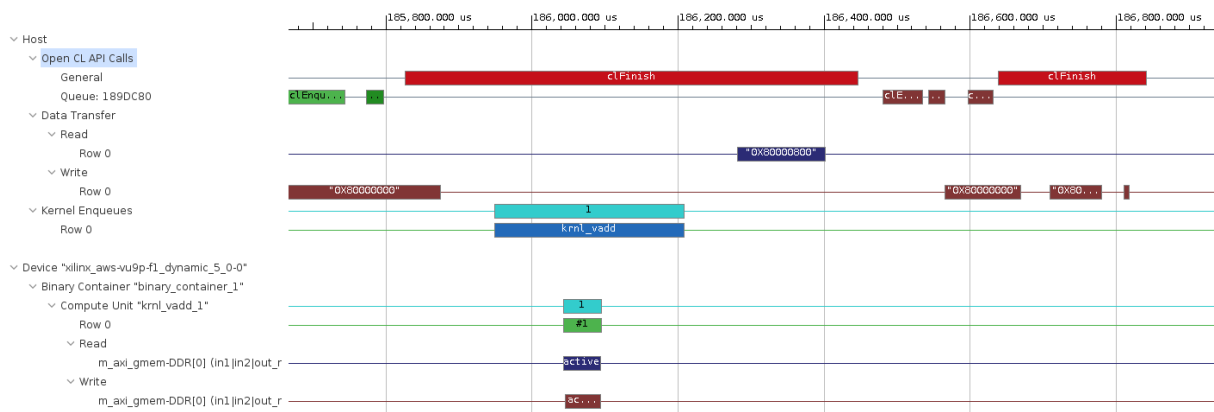
6. Execute the application by clicking **Apply** and then **Run**. The FPGA bitstream will be downloaded and the host application will be executed showing output similar to:

```
Console Problems Vitis Log Guidance
<terminated> (exit value: 0) vadd_test-Default [OpenCL] /home/centos/workspace/vadd_test/Hardware/vadd_test (4/22/20,
[Console output redirected to file:/home/centos/workspace/vadd_test/Hardware/vadd_test-Default.laun
Loading: './binary_container_1.awsxc1bin'
TEST PASSED
```

Analyze hardware application timeline and profile summary

1. In the Assistant view, double click Hardware > vadd-Default > Run Summary (xc1bin) 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 185,800,000 and 186,800,000 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	knl_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	knl_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

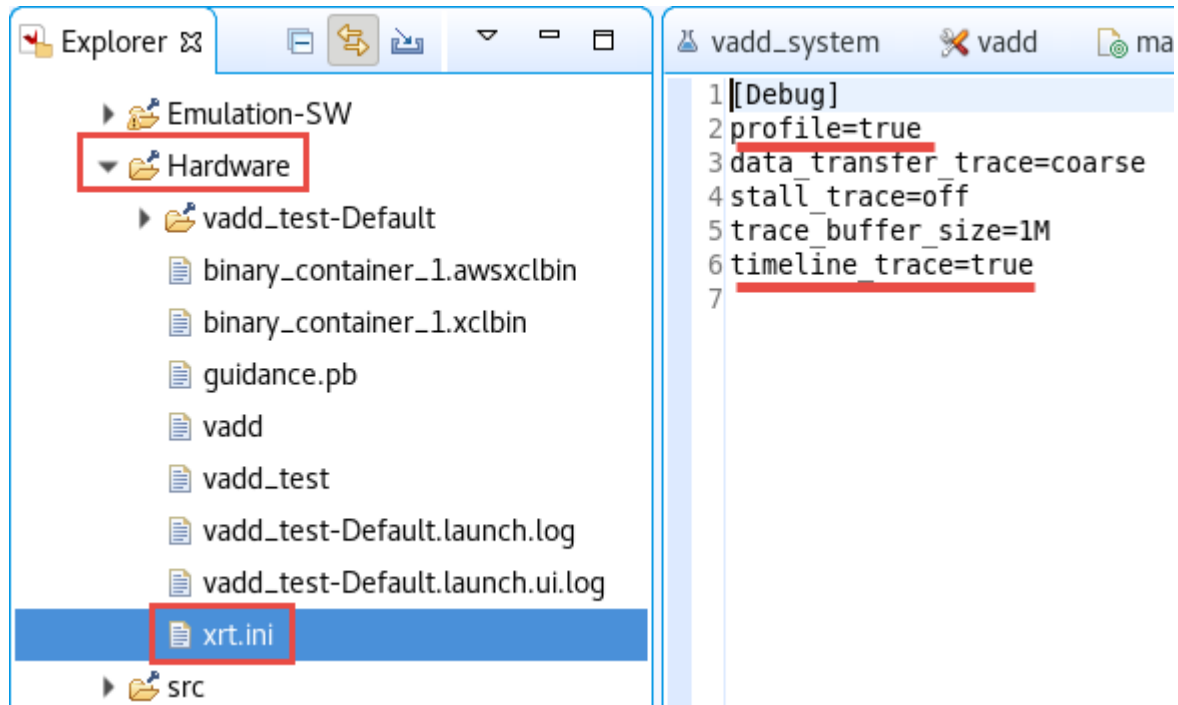
• OpenCL APIs

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. Since the system build and AFI creation takes relatively long time, you used the provided solution to download the application and kernel on the F1 instance and validated the functionality in hardware. You also analyzed profile and application timeline reports generated during running the application in actual hardware.

Start the next lab: [Improving Performance Lab](#)

Appendix: Build Full Hardware

Note that building the project can take around two hours. Skip this step in a tutorial environment.

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

Test on AWS (create AFI)

Before the design can be run on AWS an AFI (Amazon FPGA Image) is required

Once the full system is built, you can create an AFI by following the steps listed in [create an AFI](#)