Building Accelerated Applications with Vitis

Course Workbook

Table of Contents

About this Workbook

This workbook is designed to be used in conjunction with the Building Accelerated Applications with Vitis course.

The contents of this workbook are created by Adiuvo Engineering & Training, Ltd.

If you have any questions about the contents, or need assistance, please contact Adam Taylor at adam@adiuvoengineering.com.

Pre-Lab

Workshop Pre-requisites

Required Hardware

There is no required hardware for this course.

Downloads and Installations

Step 1 – Download and install the following at least 1 day prior to the workshop. This may take a significant amount of time and drive space.

Watch the video available here to show how to configure the installation

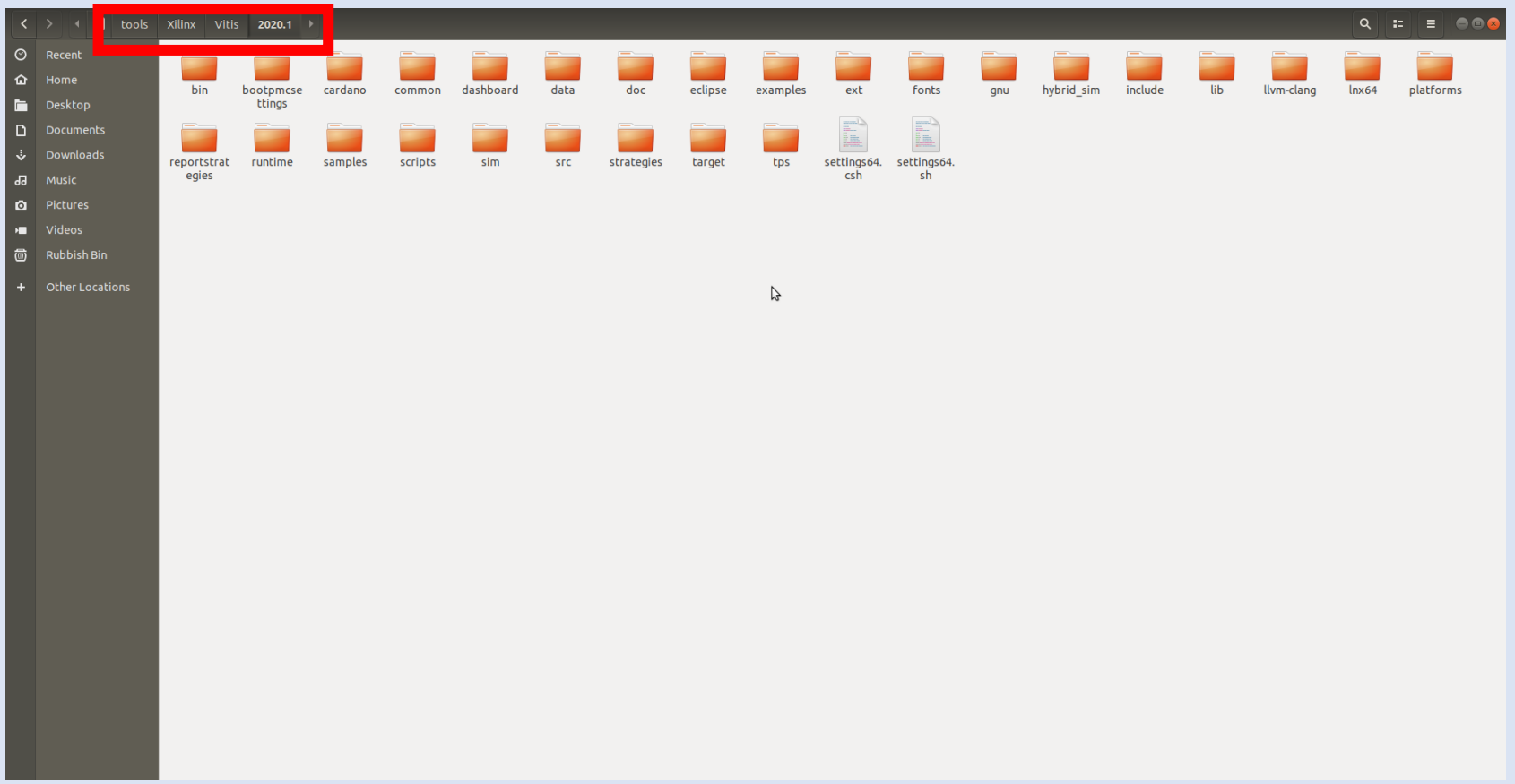
Vitis 2020.2	Download
XRT	Download
Embedded Platforms	Download
Common Images for Embedded Platforms	Download

Lab 1

Understanding Vitis Project creation & Flow

Lab 1: Understanding Vitis Project creation & Flow

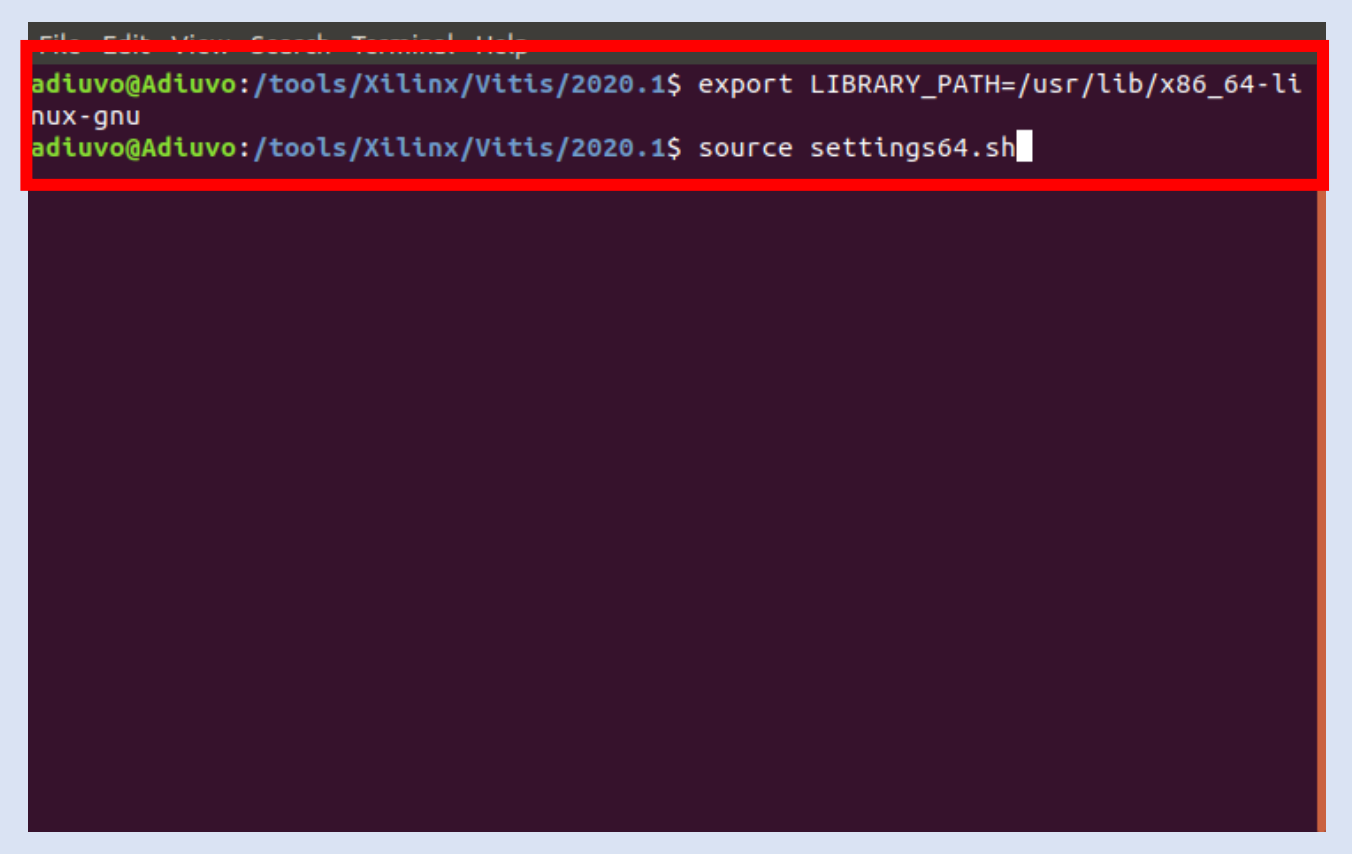
Step 1 – Open a file browser and navigate to /tools/Xilinx/Vitis/2020.1



Lab 1: Understanding Vitis Project creation & Flow

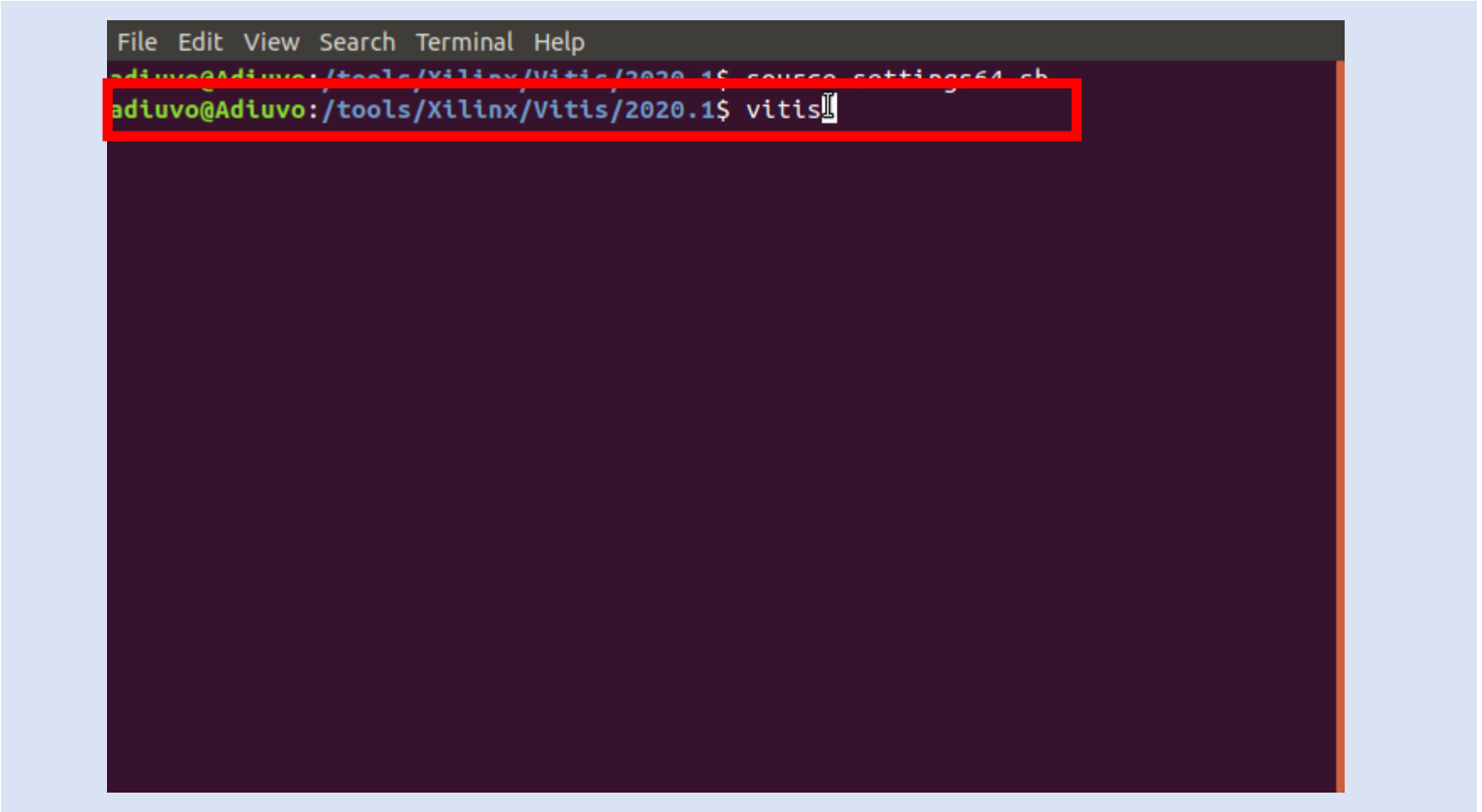
Step 2 – Open a terminal window, set the path of the LIBRARY_PATH and source the settings64.sh

export LIBRARY_PATH=/usr/lib/x86_64-linux-gnu



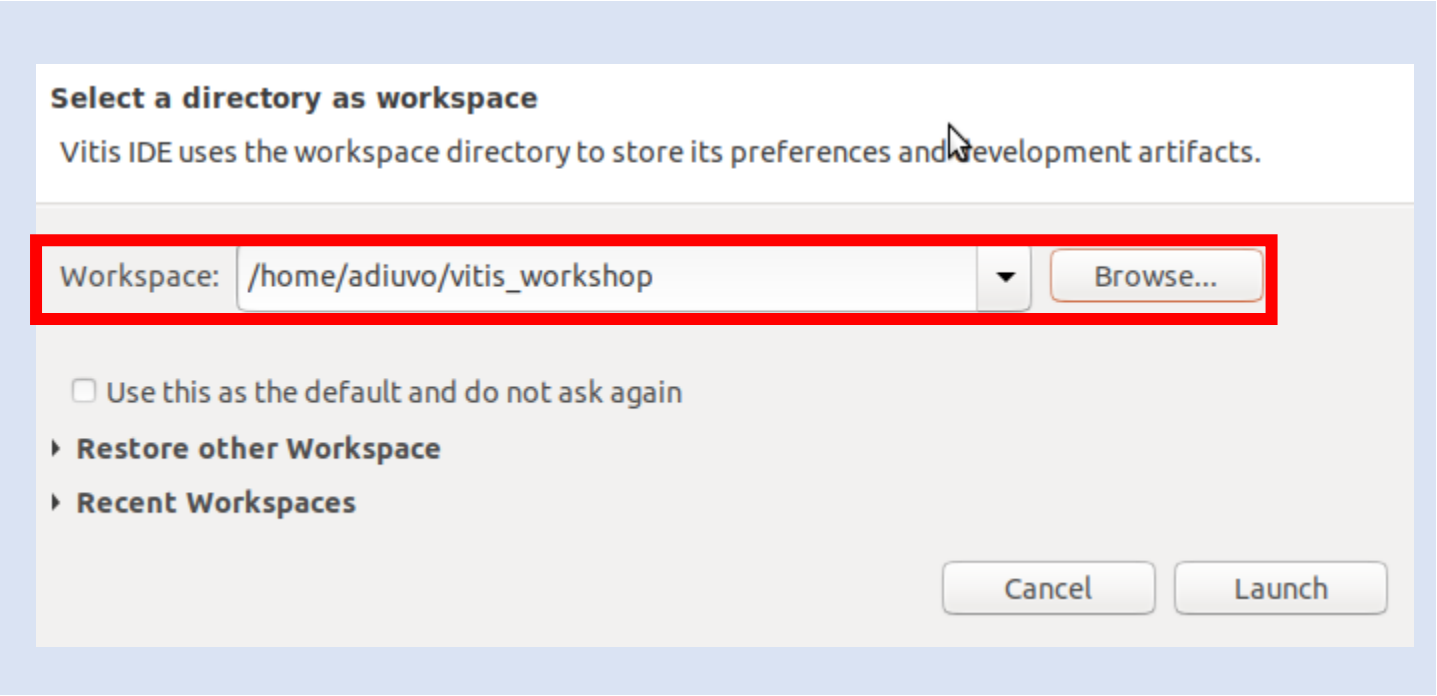
Lab 1: Understanding Vitis Project creation & Flow

Step 3 – Start Vitis from the command prompt by typing “vitis”



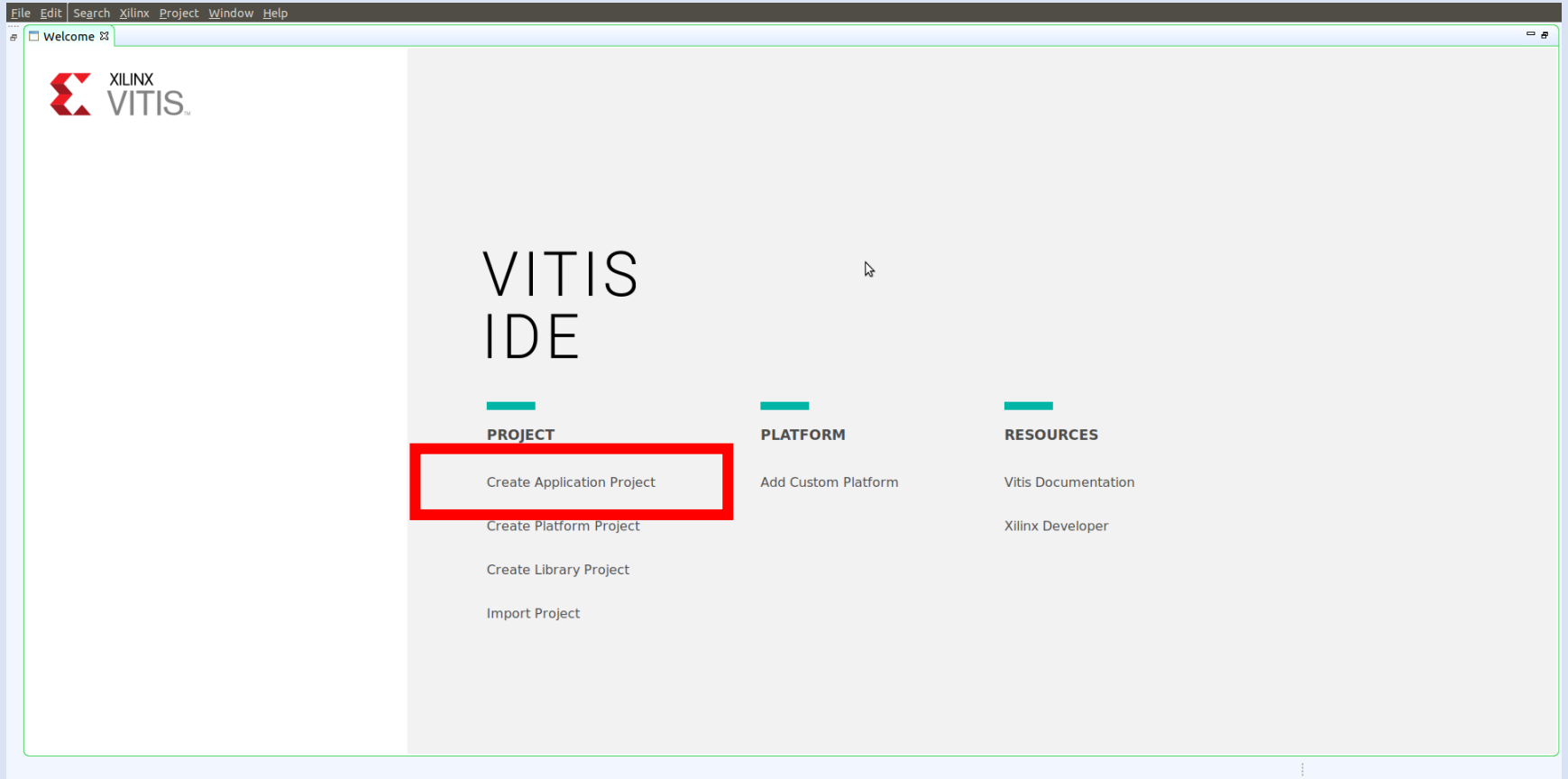
Lab 1: Understanding Vitis Project creation & Flow

Step 3 – Select the directory you which to use as your workspace, click launch



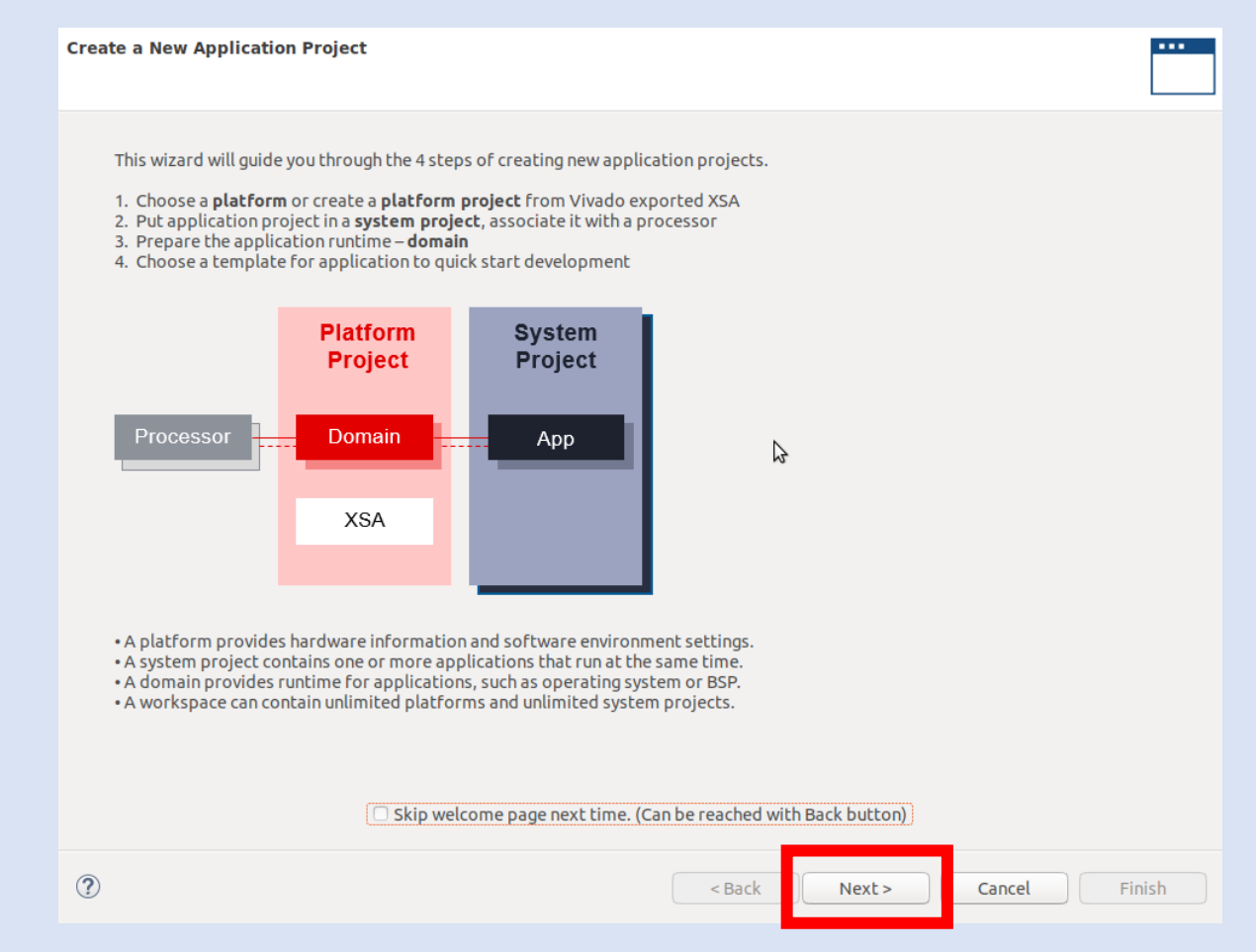
Lab 1: Understanding Vitis Project creation & Flow

Step 4 – When the IDE launches, select “Create Application Project”



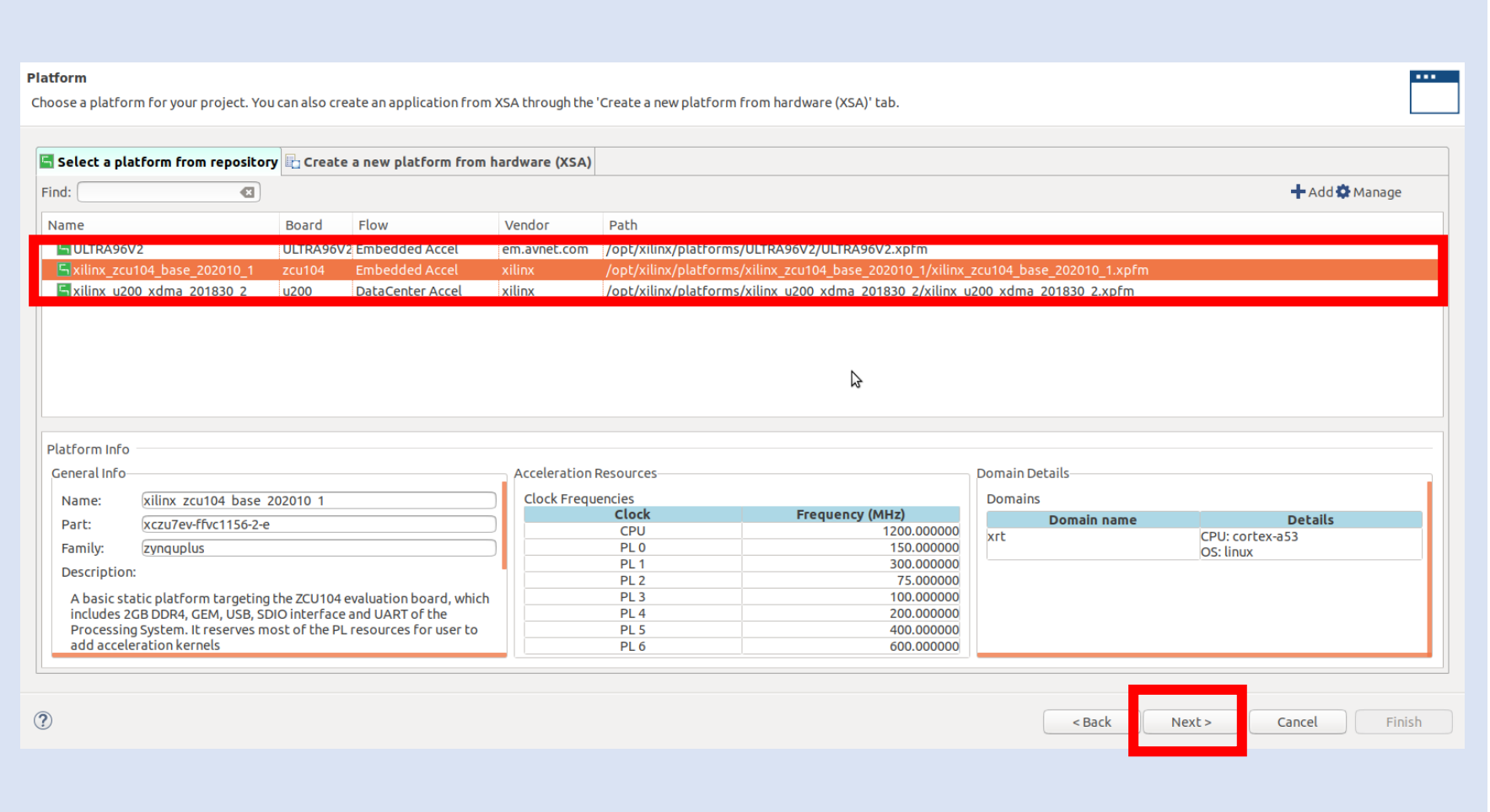
Lab 1: Understanding Vitis Project creation & Flow

Step 5 – This will launch the project creation wizard, click next on the first dialog



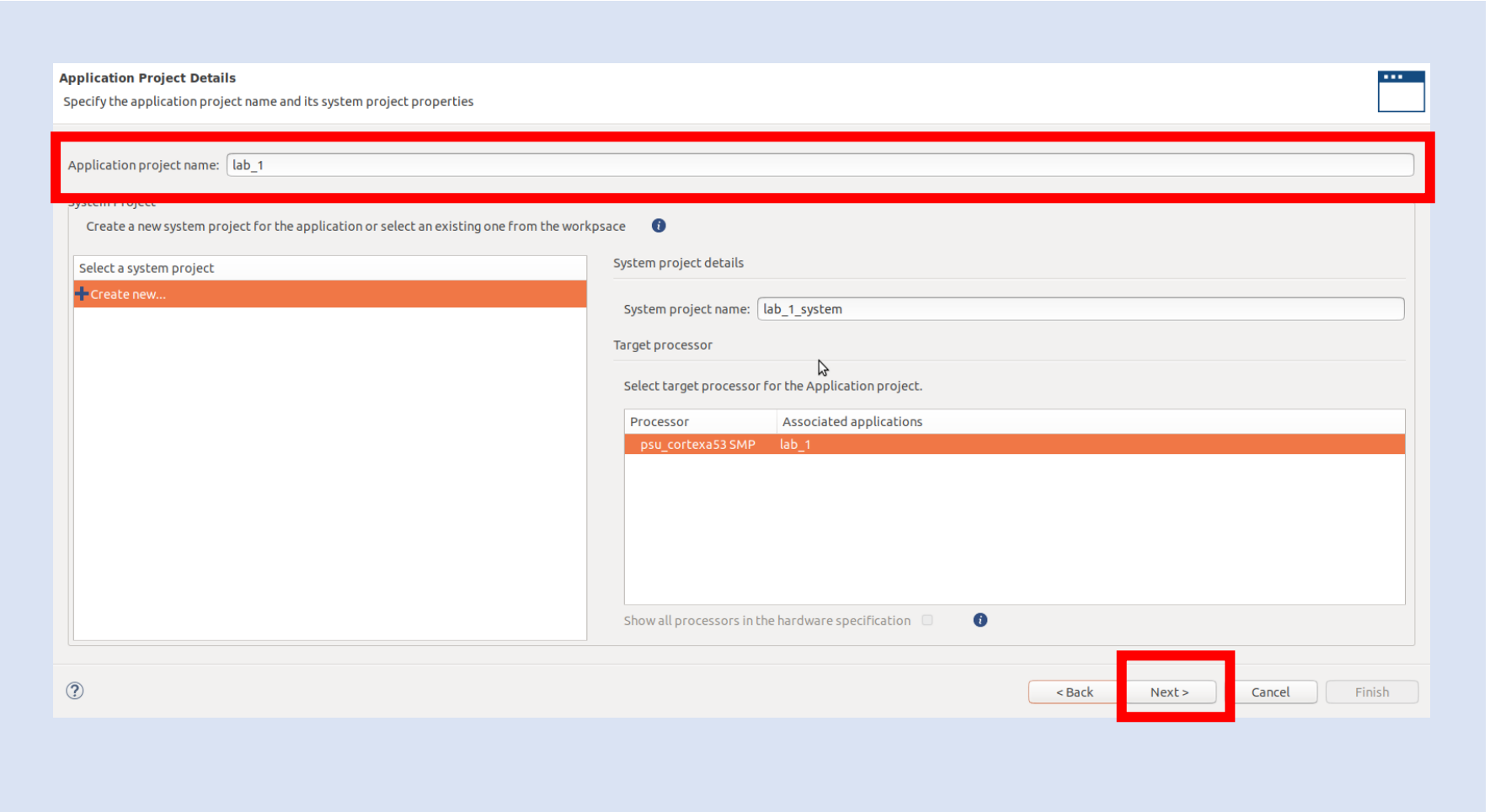
Lab 1: Understanding Vitis Project creation & Flow

Step 6 – You should see the platforms you installed; I will select ZCU104, click next



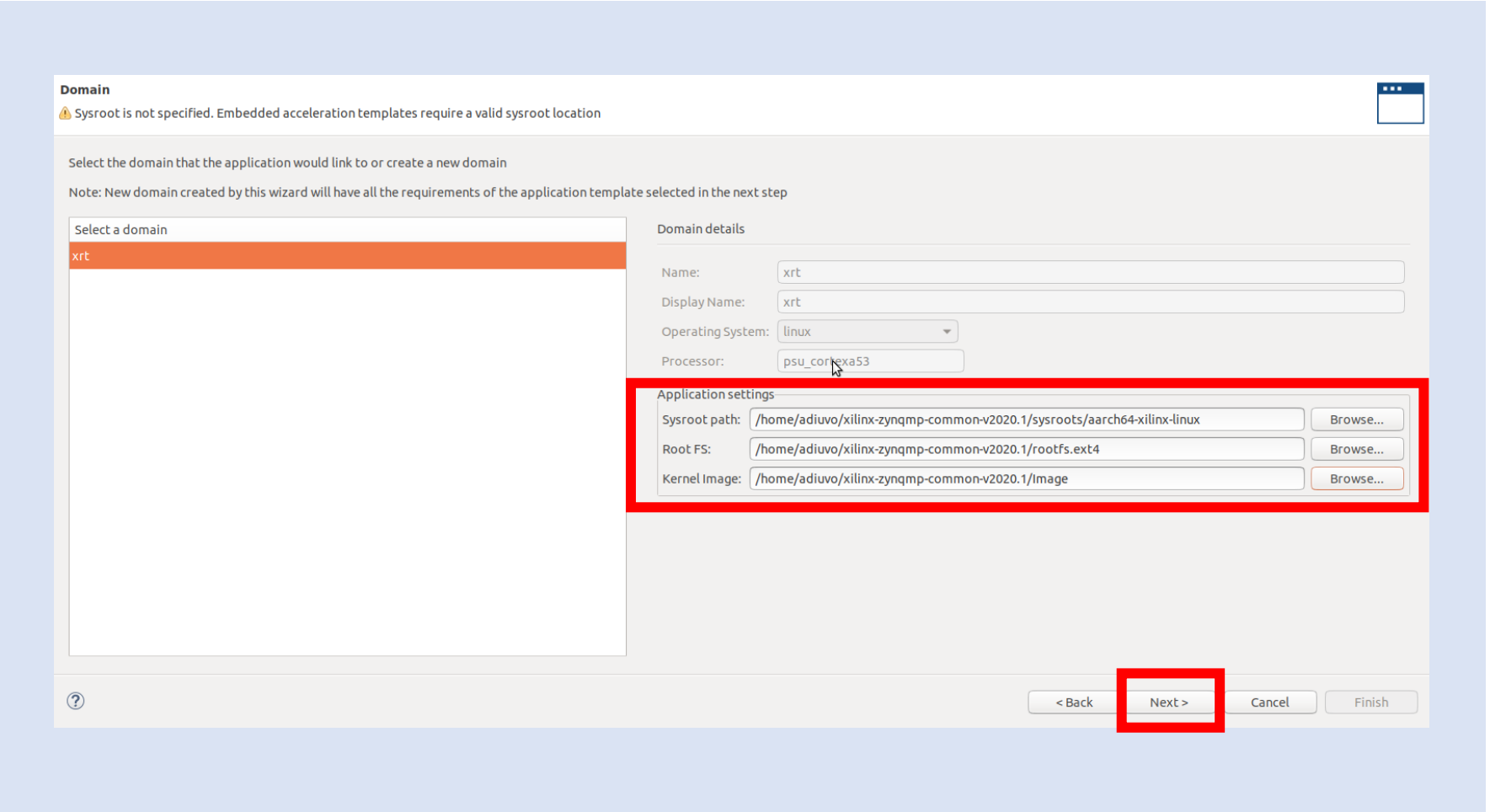
Lab 1: Understanding Vitis Project creation & Flow

Step 7 – Enter the name lab_1 for the project name and click next



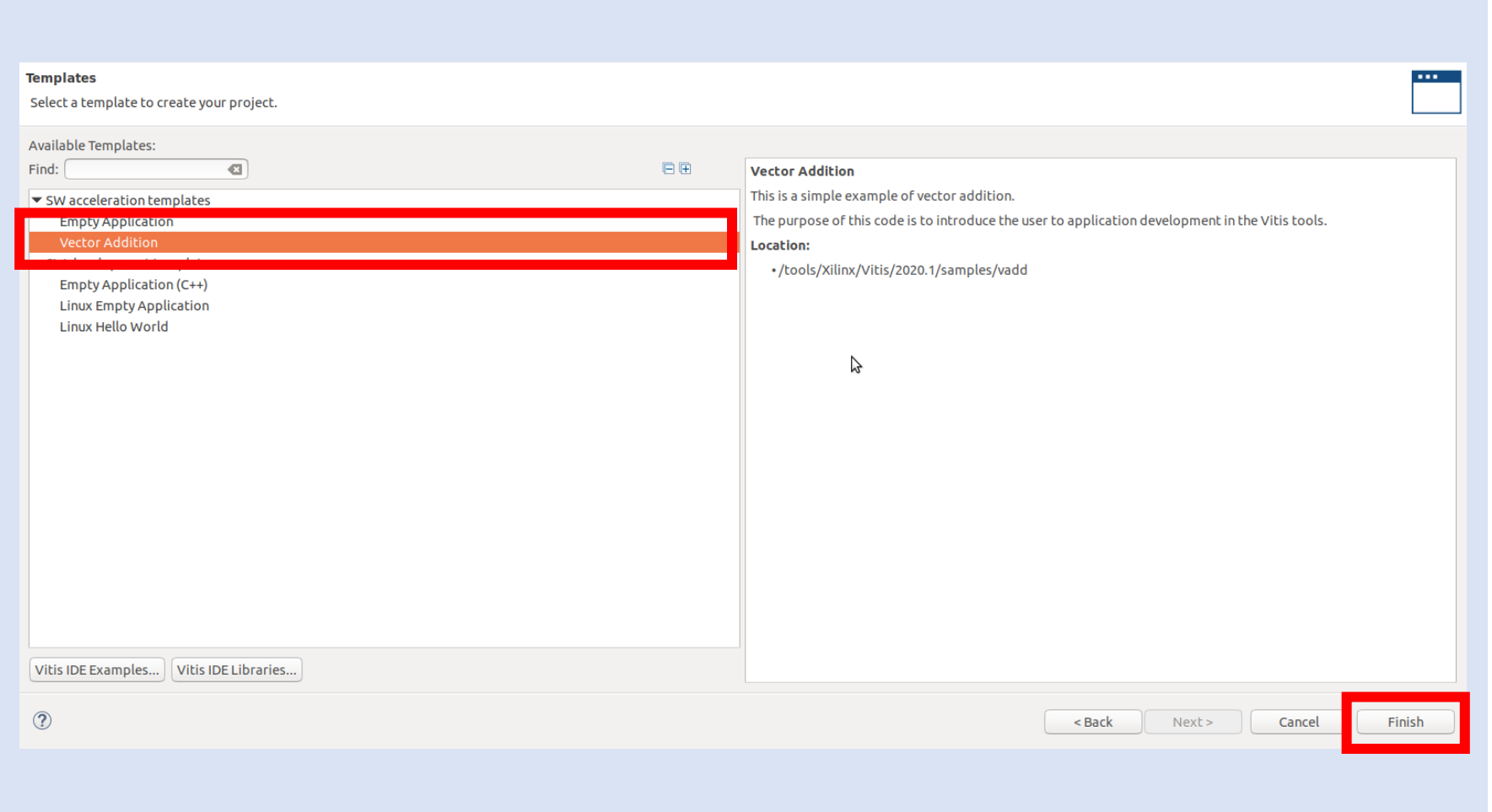
Lab 1: Understanding Vitis Project creation & Flow

Step 8 – Enter the Sysroot, Root FS and Kernel image previously installed, click next



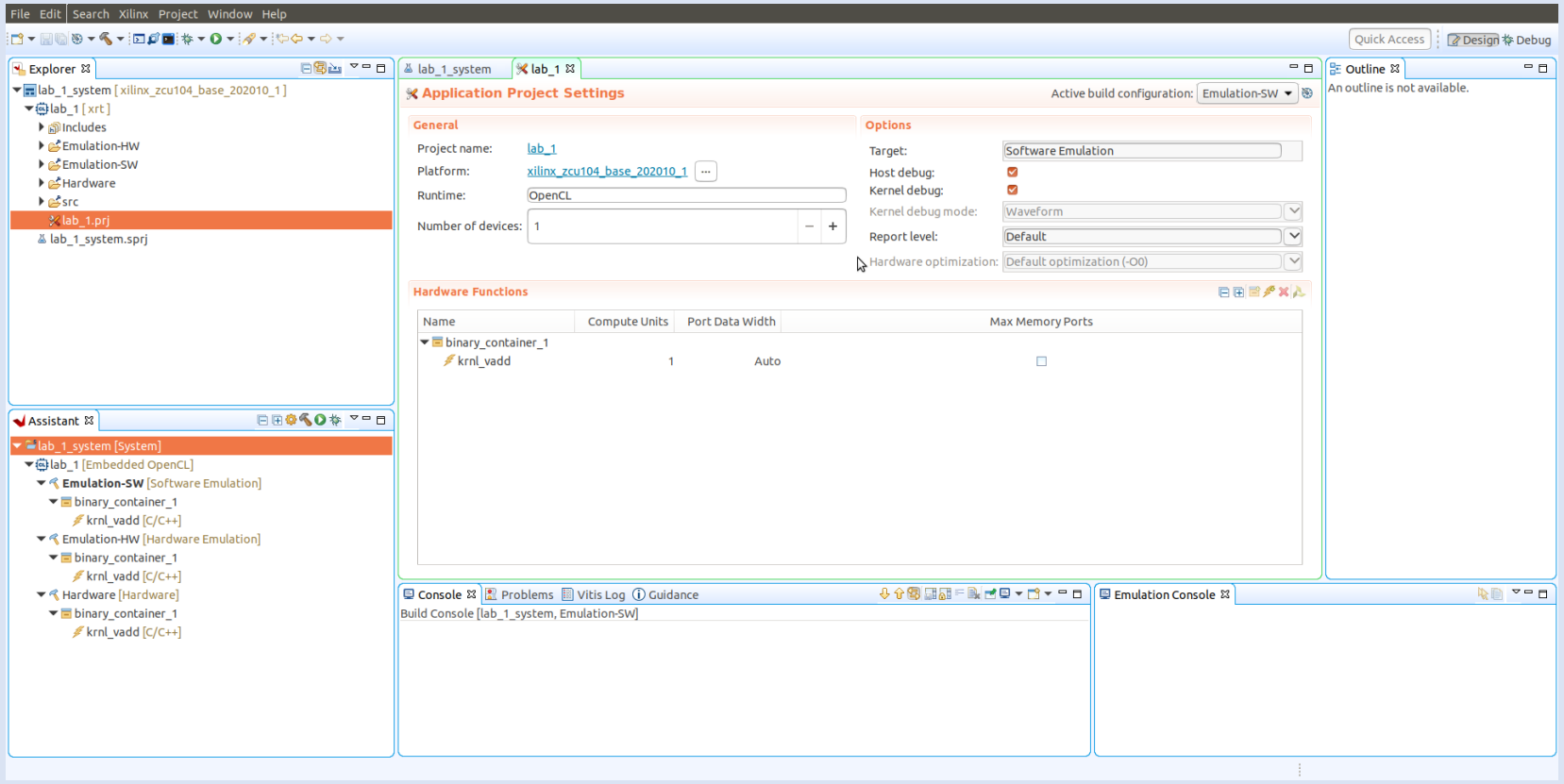
Lab 1: Understanding Vitis Project creation & Flow

Step 8 – Select the Vector Addition example and click finish



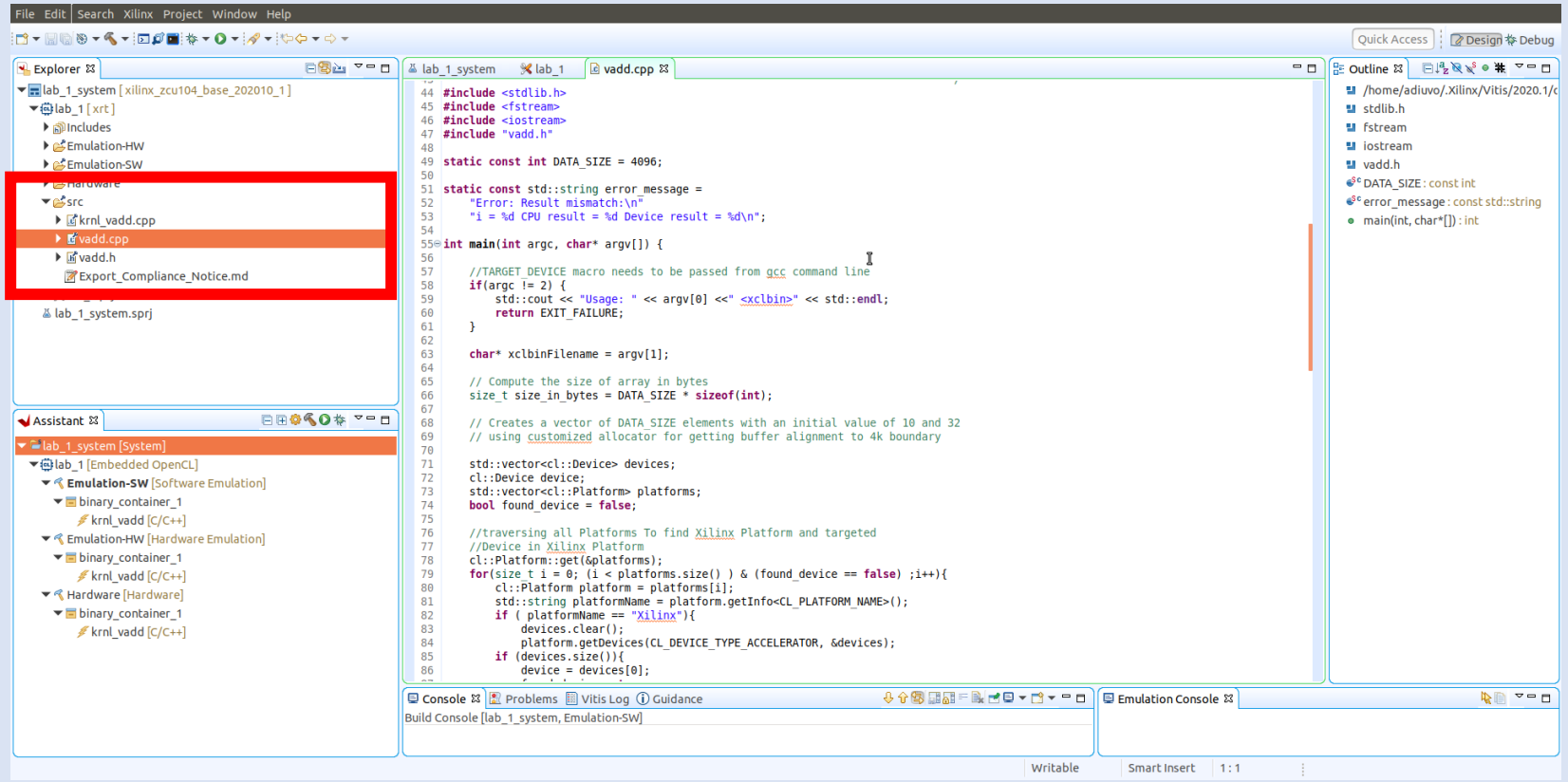
Lab 1: Understanding Vitis Project creation & Flow

Step 9 – This will open the project in the Vitis IDE



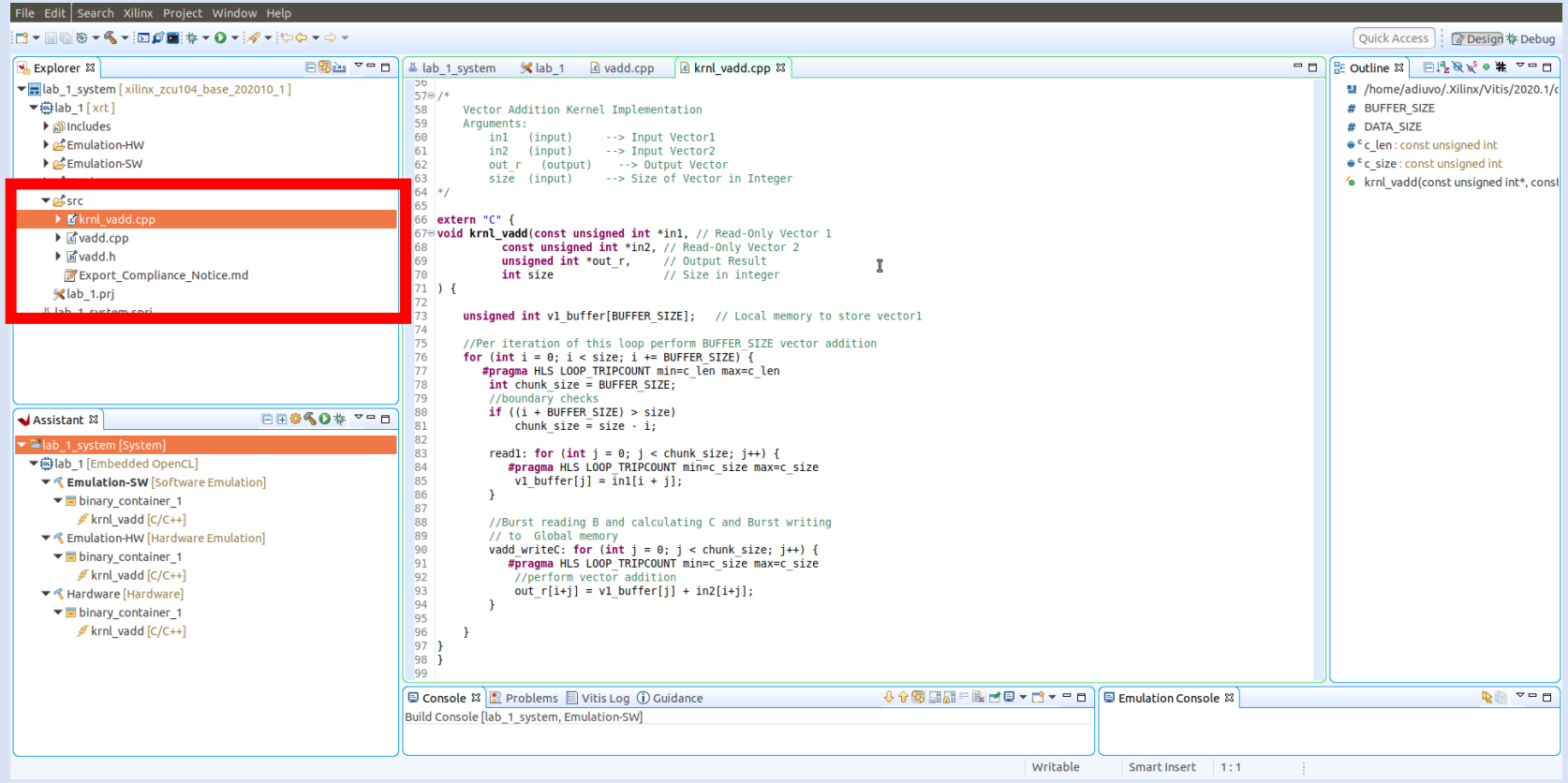
Lab 1: Understanding Vitis Project creation & Flow

Step 10 – Expand the src directory and double click on the Vadd.cpp file – this is the host SW



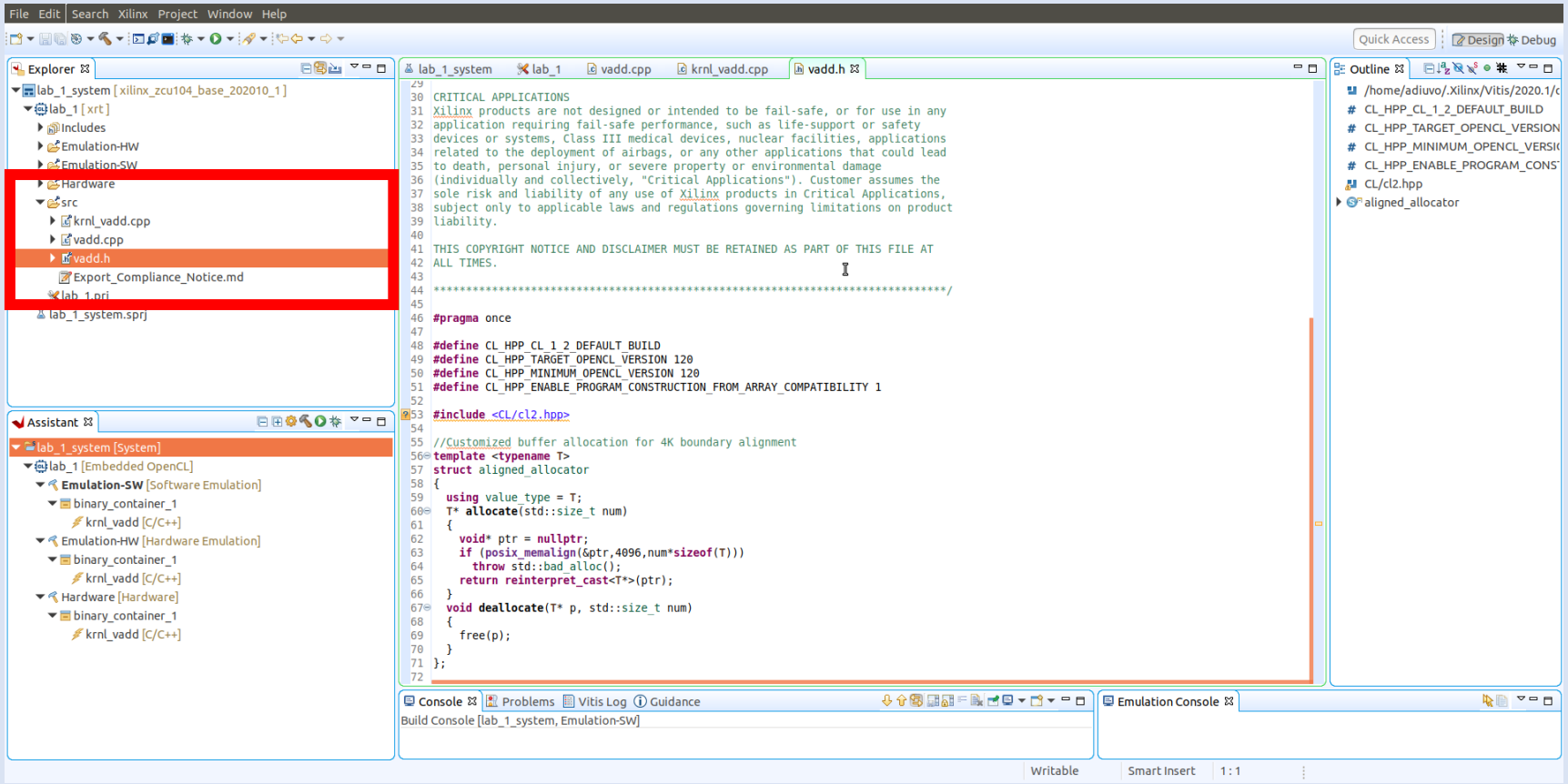
Lab 1: Understanding Vitis Project creation & Flow

Step 11 – Double click on the krnl_vadd.cpp file – this is the accelerator



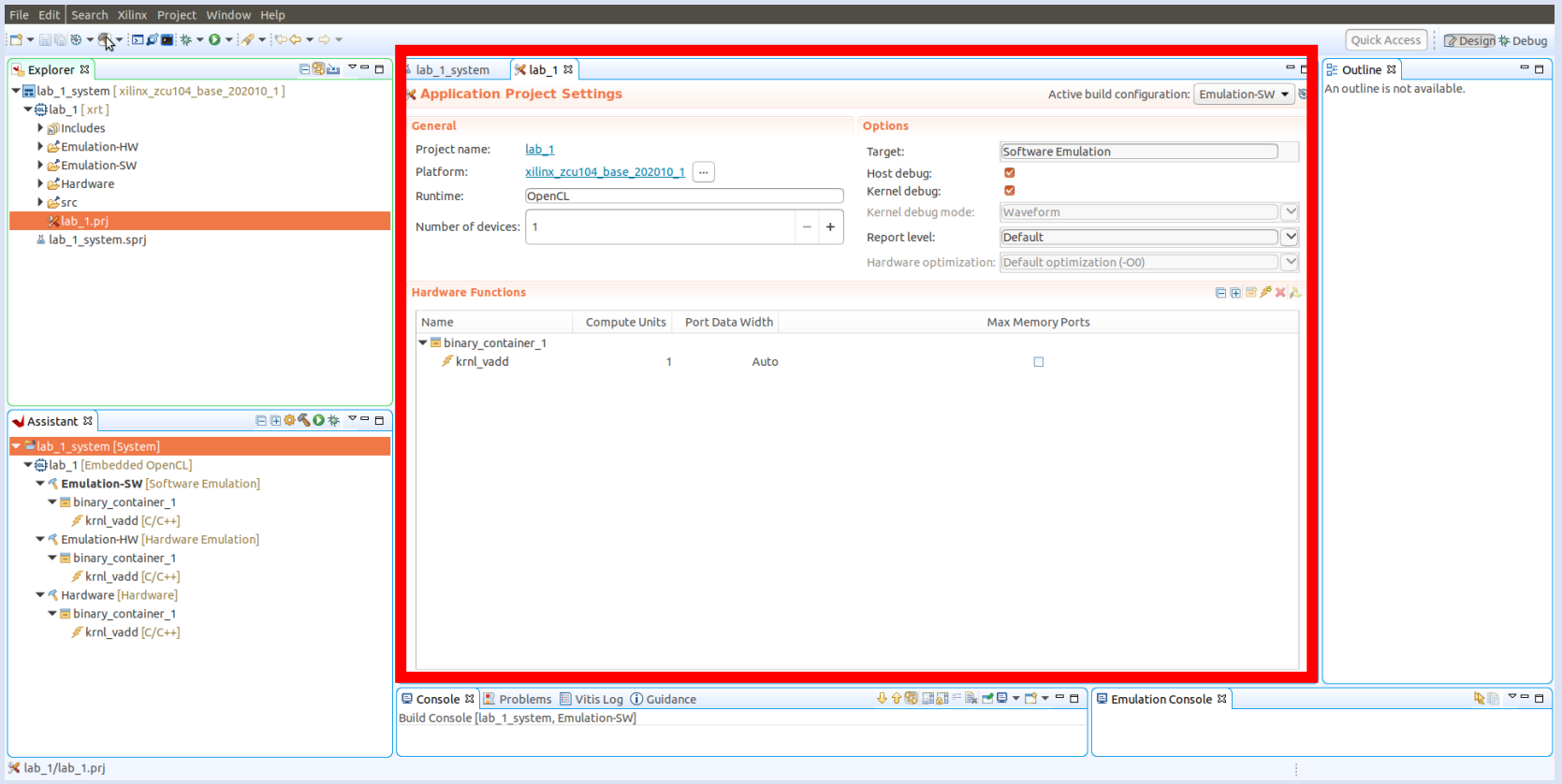
Lab 1: Understanding Vitis Project creation & Flow

Step 12 – Double click on the vadd.h



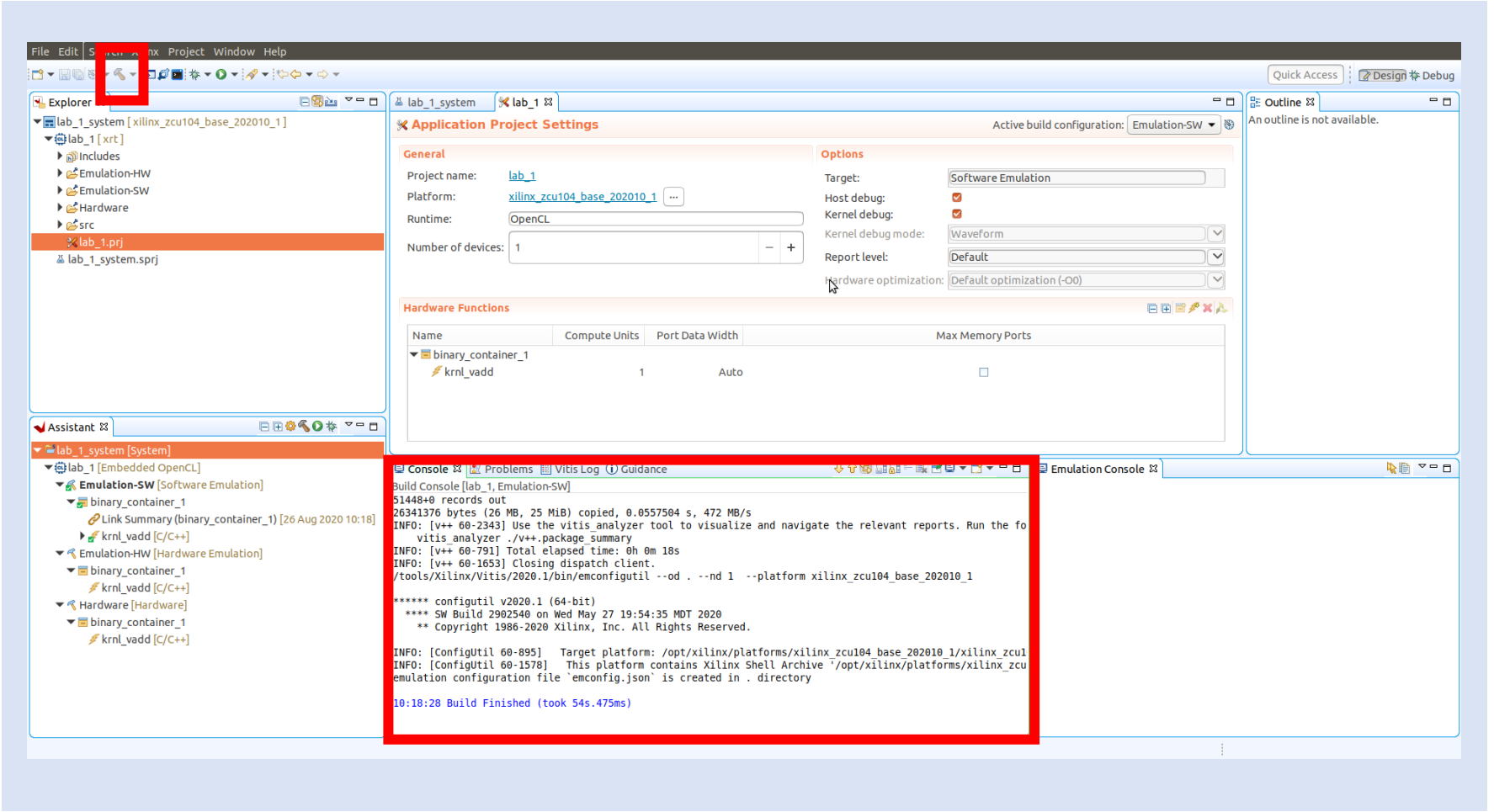
Lab 1: Understanding Vitis Project creation & Flow

Step 13 – Click on the lab_1 tab this where we set most configurations for building – make sure the active build configuration is set at Emulation-SW



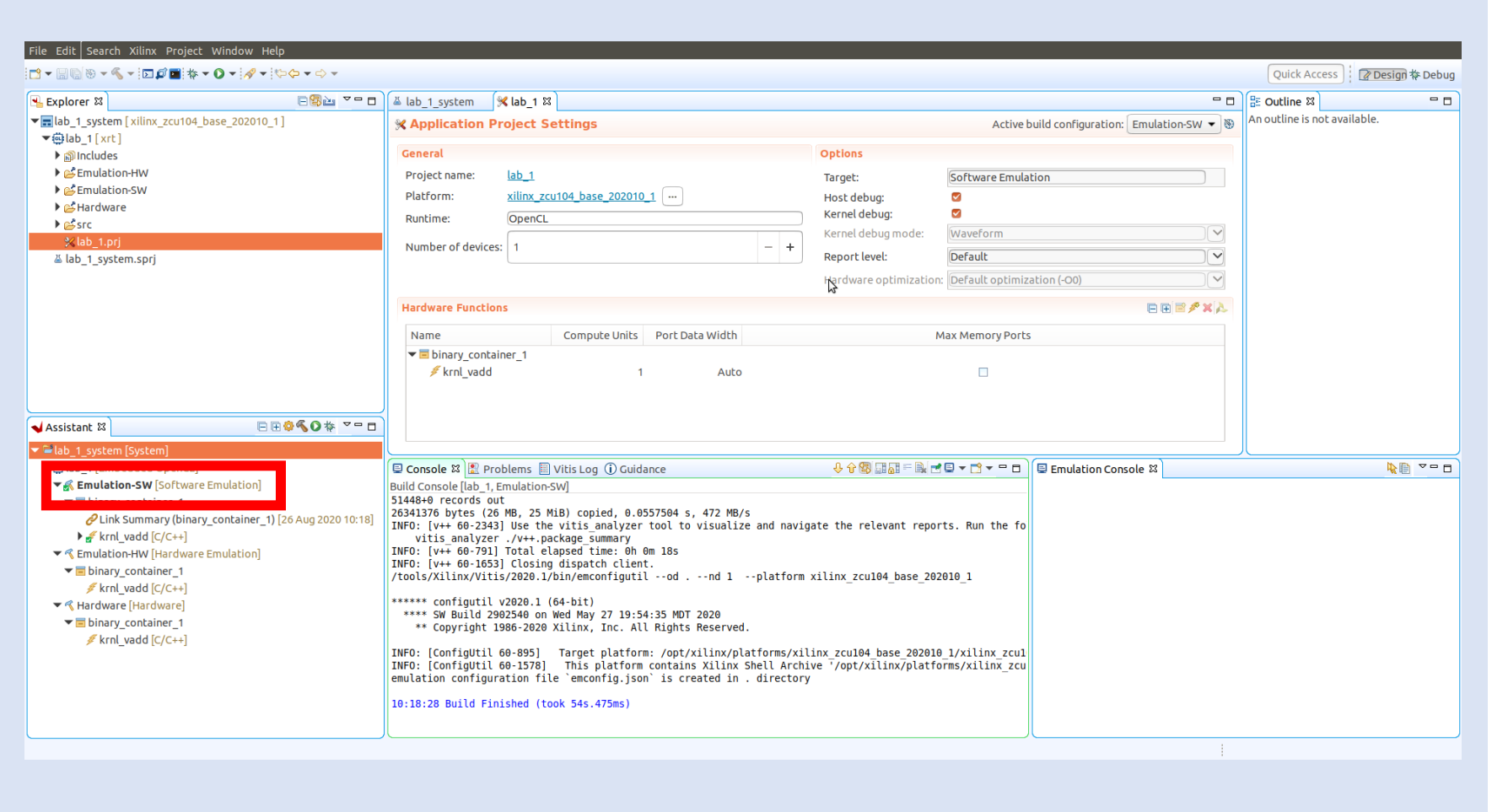
Lab 1: Understanding Vitis Project creation & Flow

Step 14 – Build the SW Emulation by clicking on the hammer



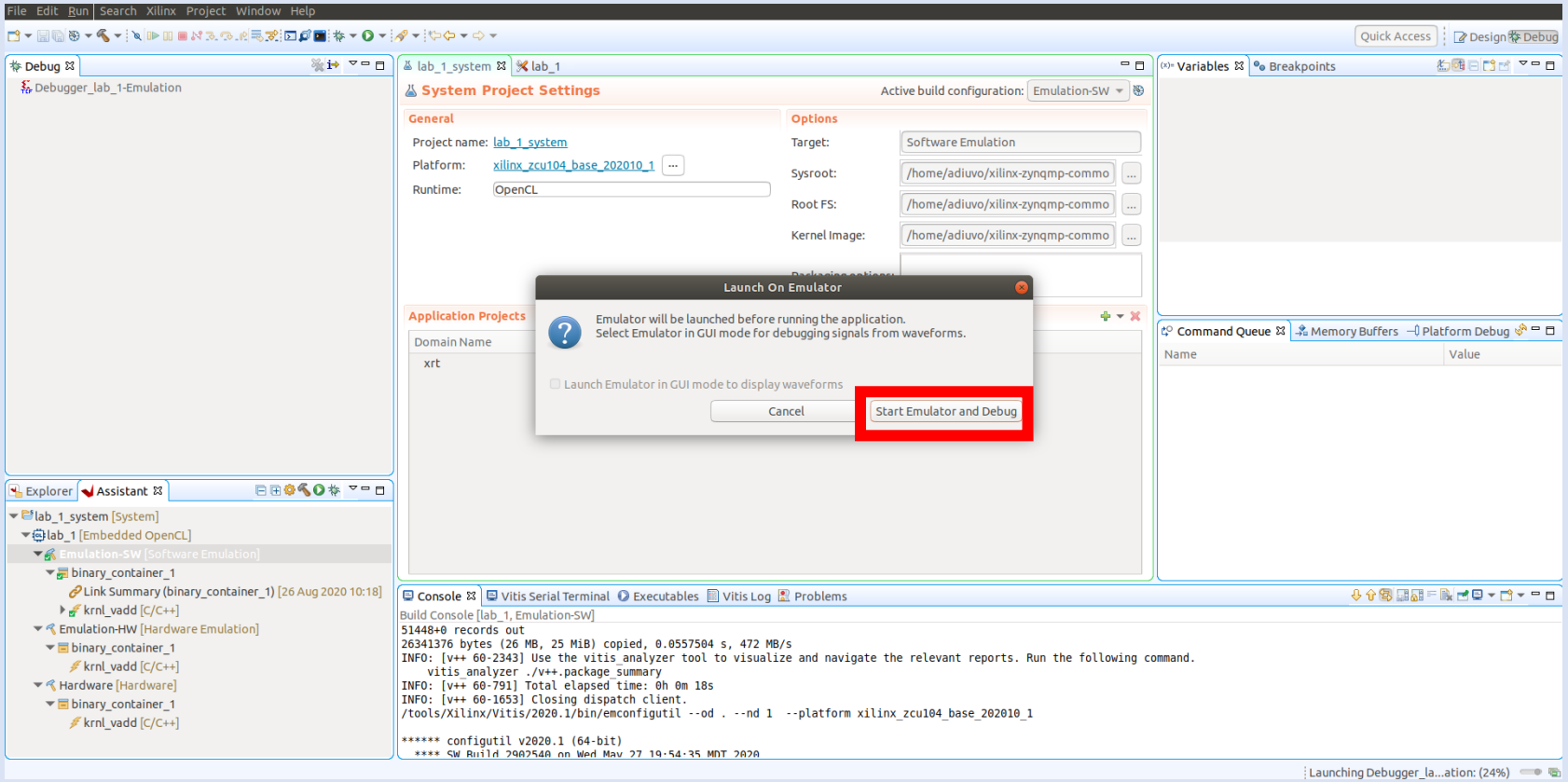
Lab 1: Understanding Vitis Project creation & Flow

Step 15 – Launch the debugger by right clicking on Emulation-SW and Launch on Emulator



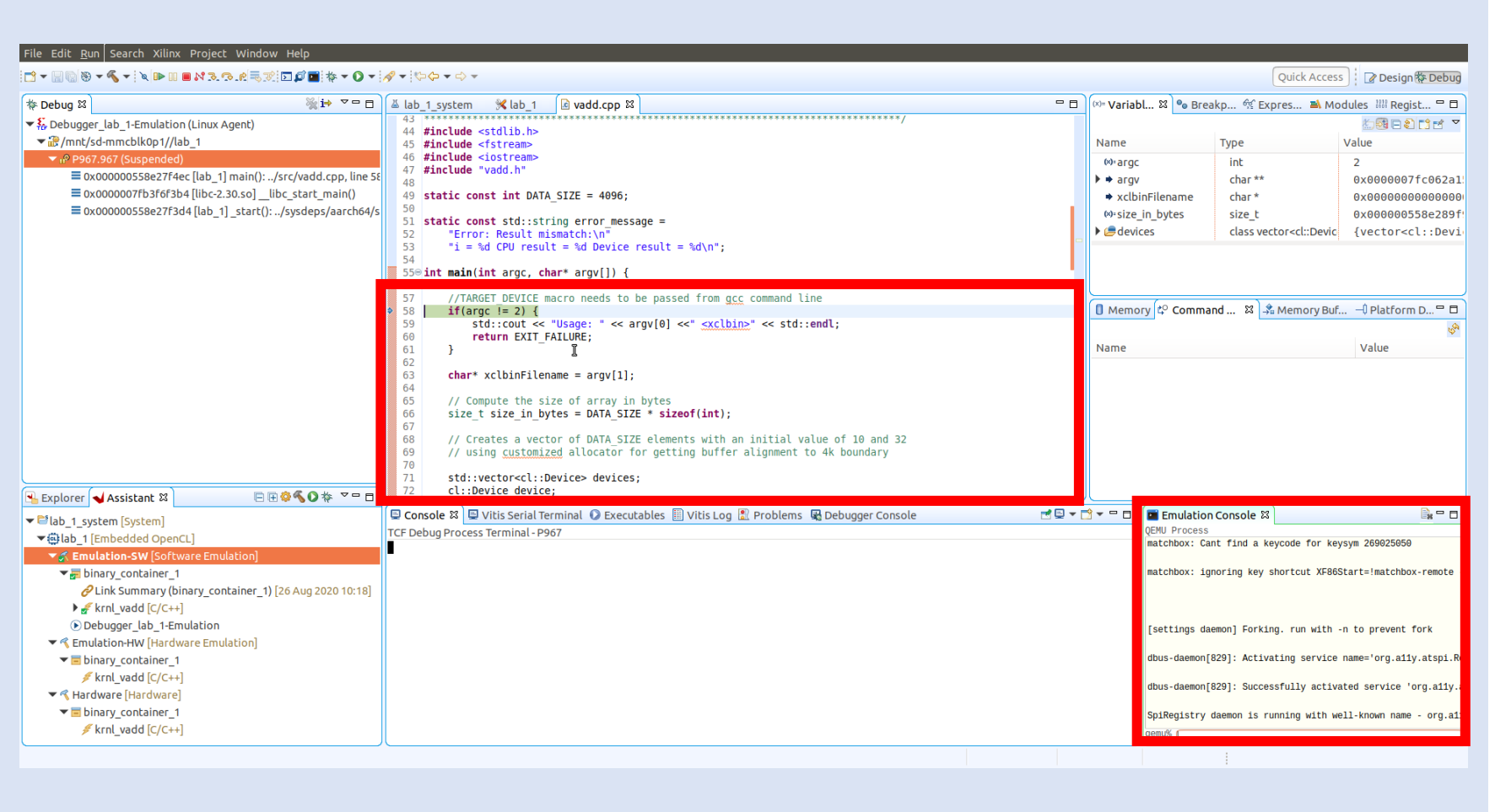
Lab 1: Understanding Vitis Project creation & Flow

Step 16 – This will open the Debug Perspective and click on start Emulator and Debug



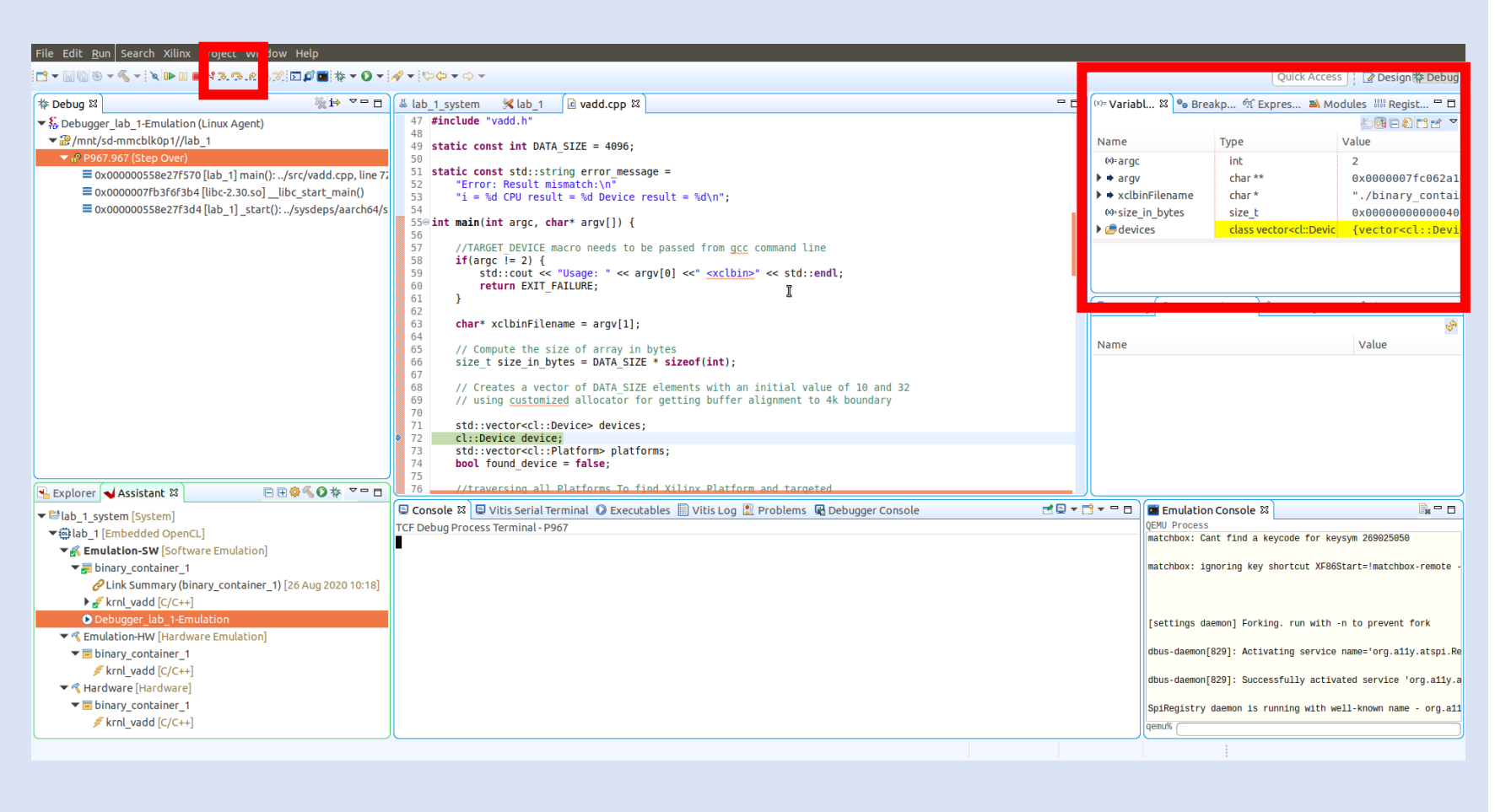
Lab 1: Understanding Vitis Project creation & Flow

Step 17 – Once the Emulator is loaded and the TCF agent connected the Host application will pause awaiting execution.



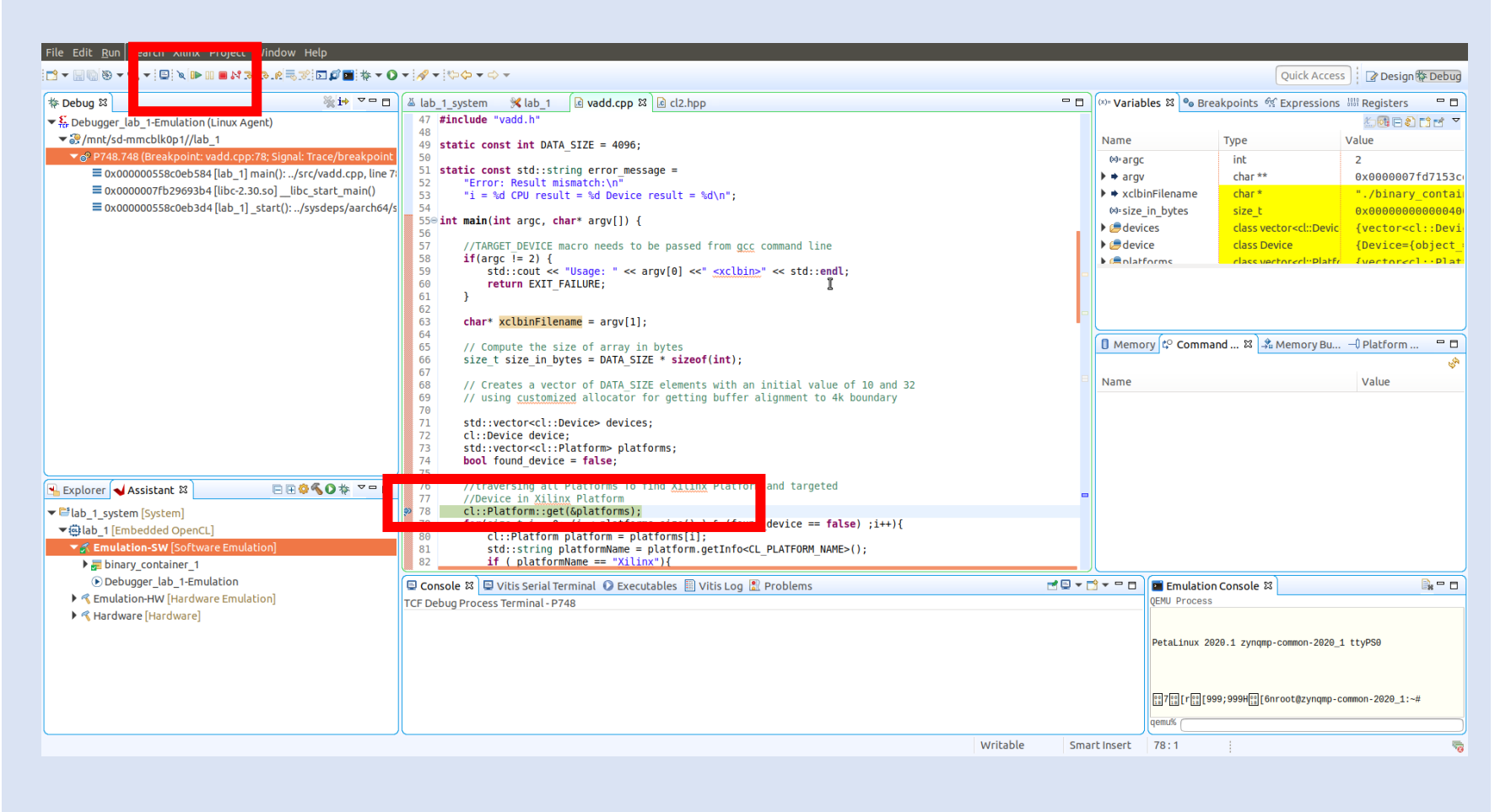
Lab 1: Understanding Vitis Project creation & Flow

Step 18 – Single step through a few instructions and see the variables update



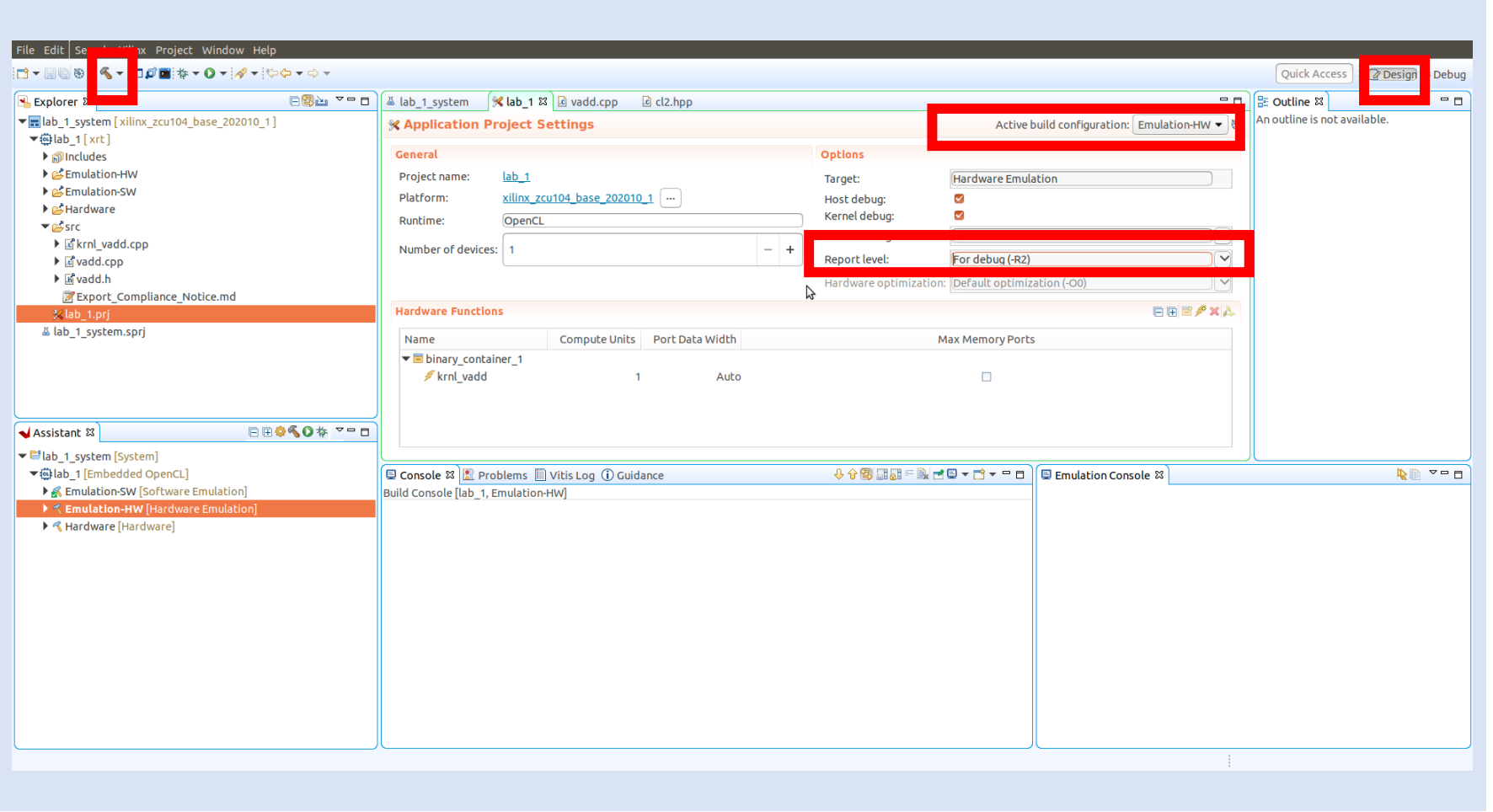
Lab 1: Understanding Vitis Project creation & Flow

Step 19 – Double click in the edge of the source code on line 78 to insert a breakpoint then select run, see the break point get hit and variables update. Clock on the red terminate button



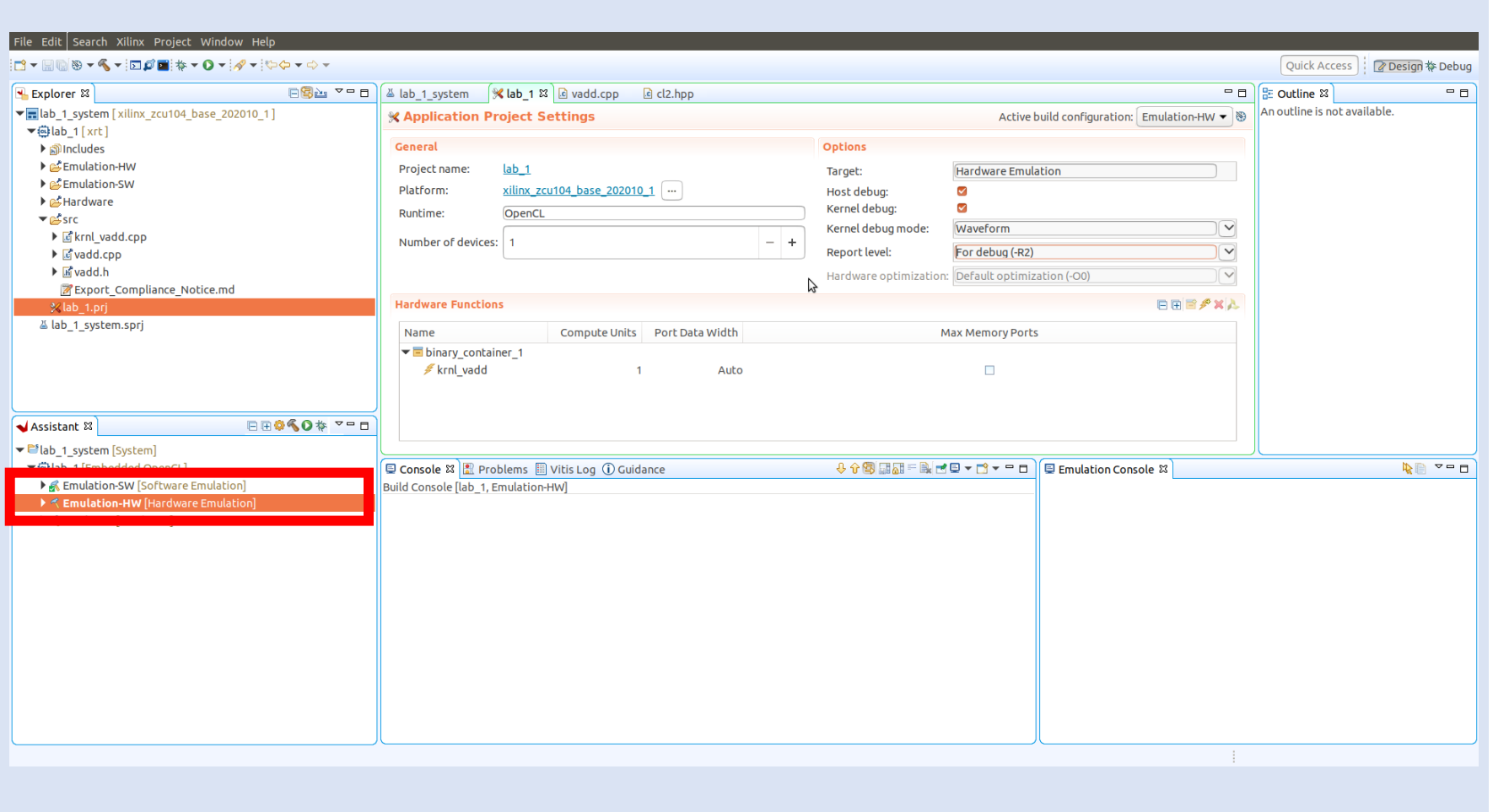
Lab 1: Understanding Vitis Project creation & Flow

Step 20 – Change the perspective back to design, set the active build configuration to Emulation-HW, Report Level to For-Debug and build the design



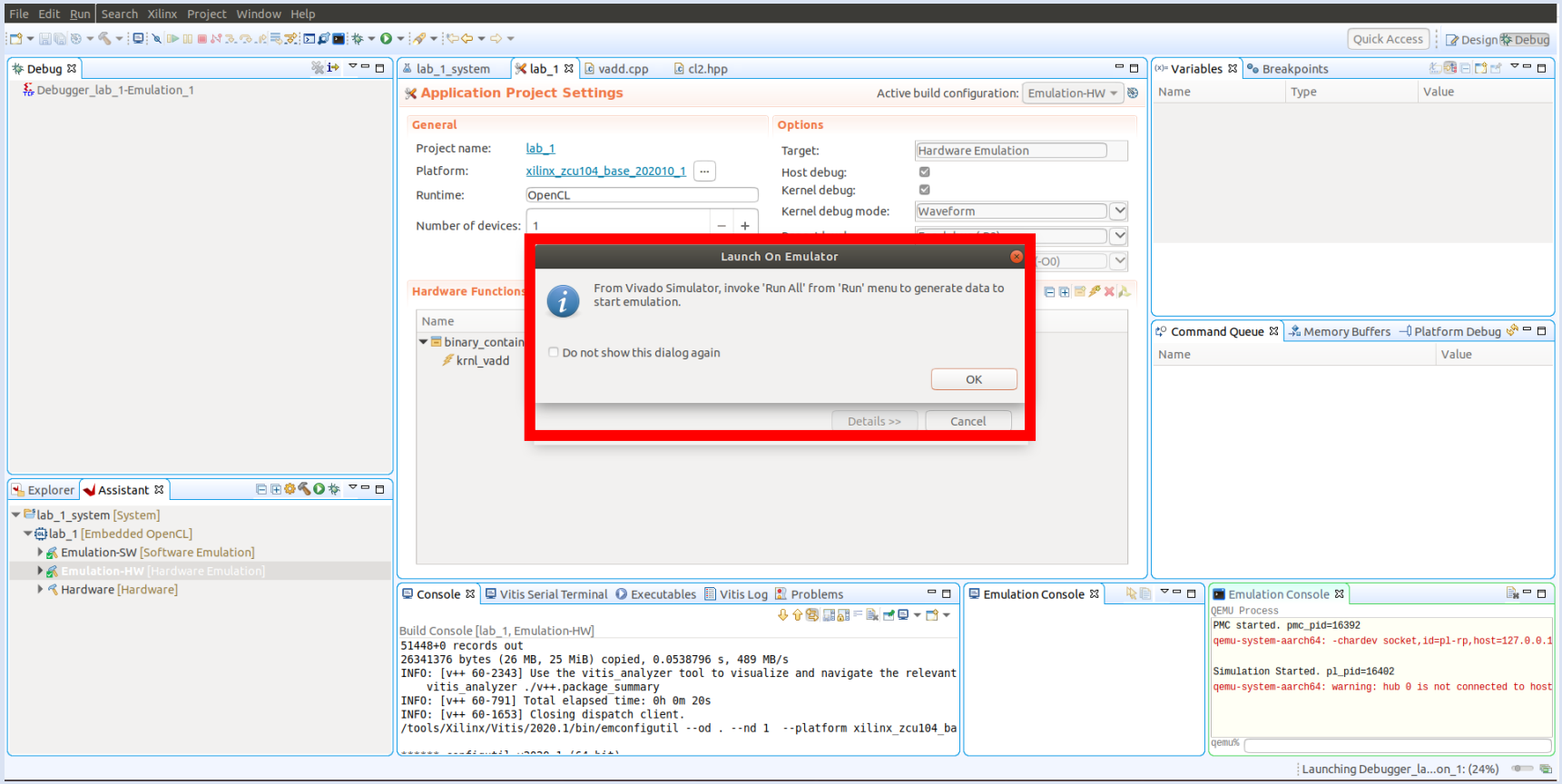
Lab 1: Understanding Vitis Project creation & Flow

Step 21 – Once build is complete, right click on Emulation-HW and launch a emulation debug session



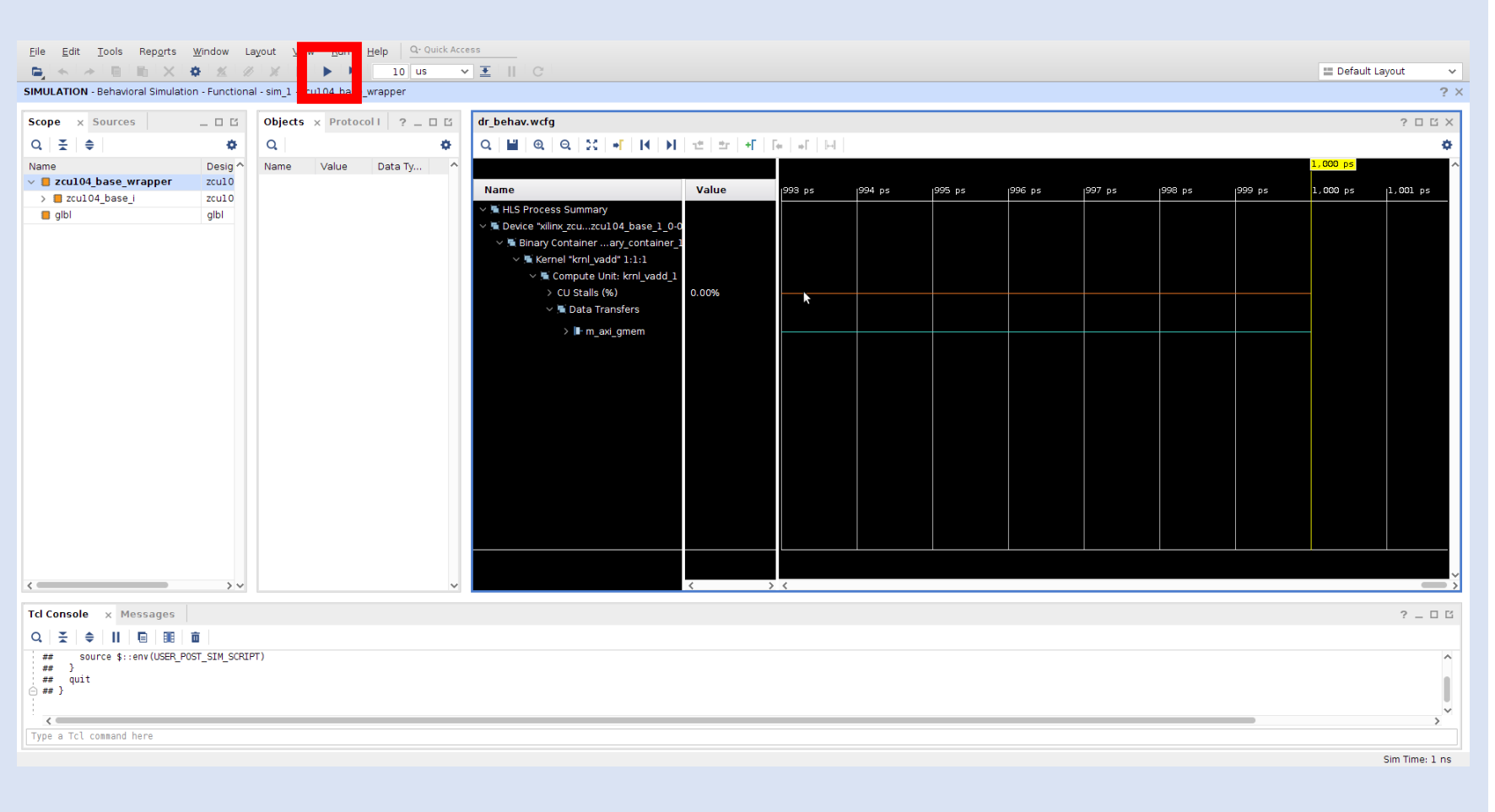
Lab 1: Understanding Vitis Project creation & Flow

Step 22 – This will launch Vivado, change views to Vivado when the dialog below appears



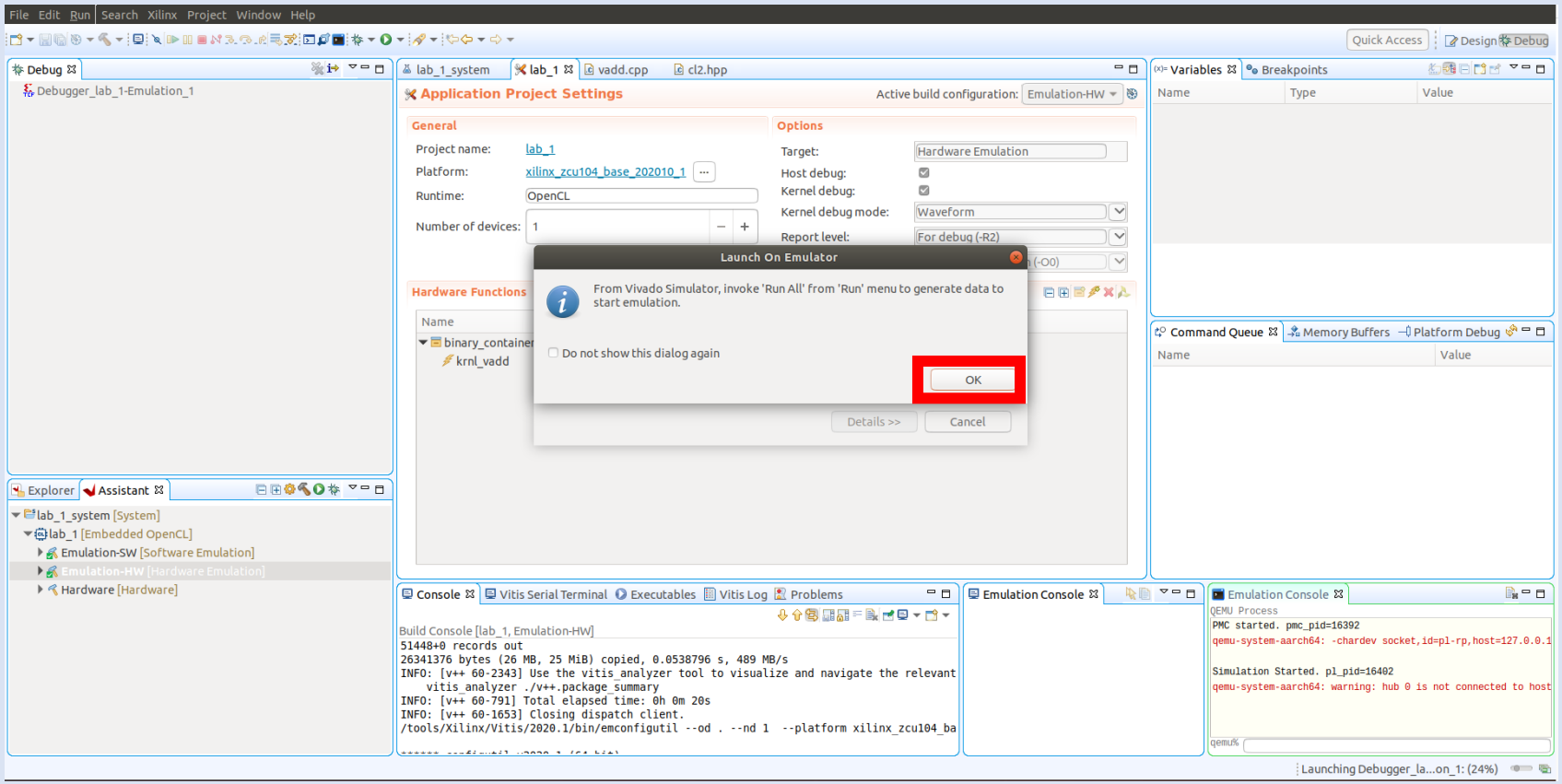
Lab 1: Understanding Vitis Project creation & Flow

Step 23 – Click on the run all button in Vivado and switch back to Vitis



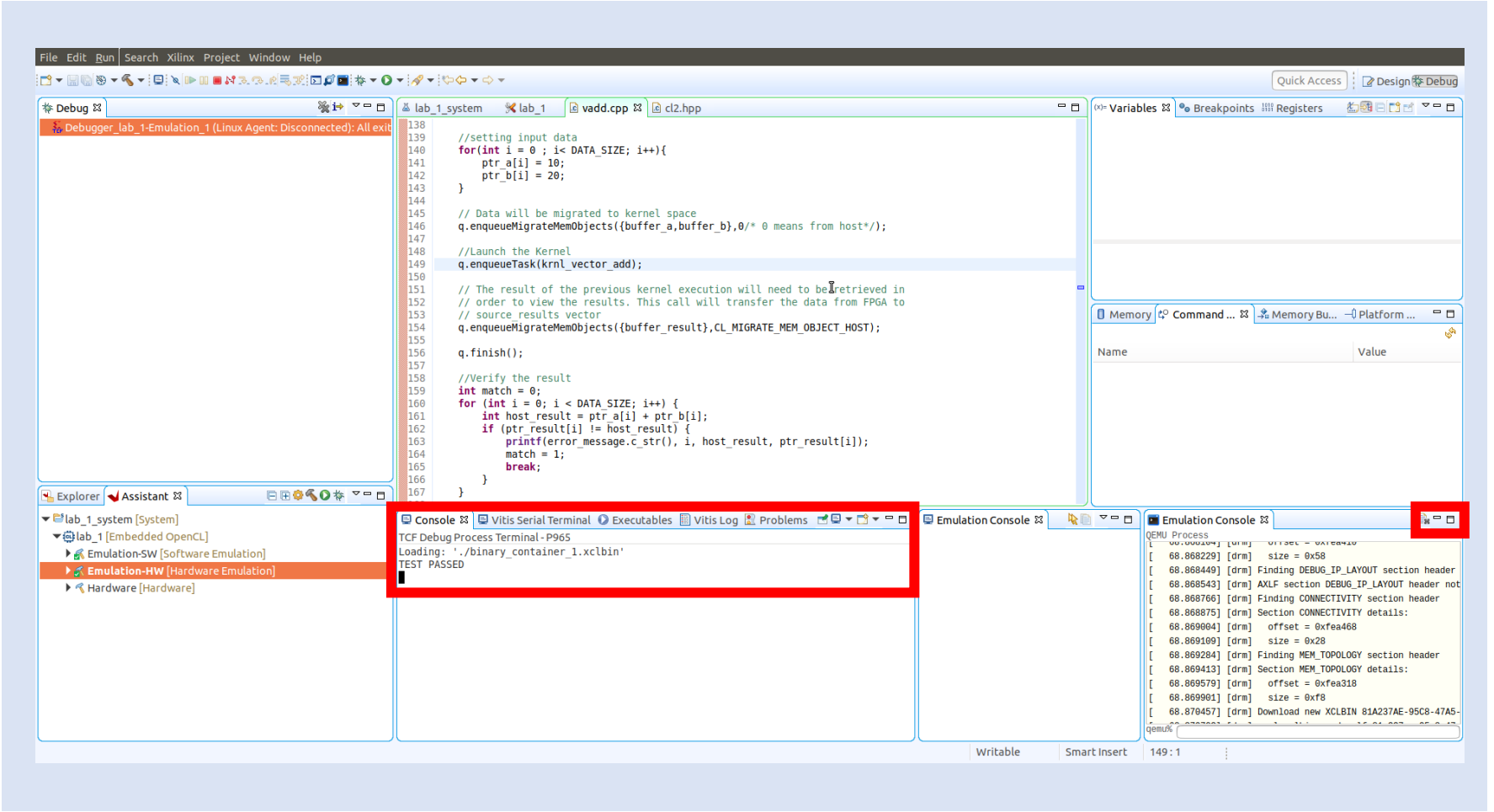
Lab 1: Understanding Vitis Project creation & Flow

Step 24 – Click OK on the dialog



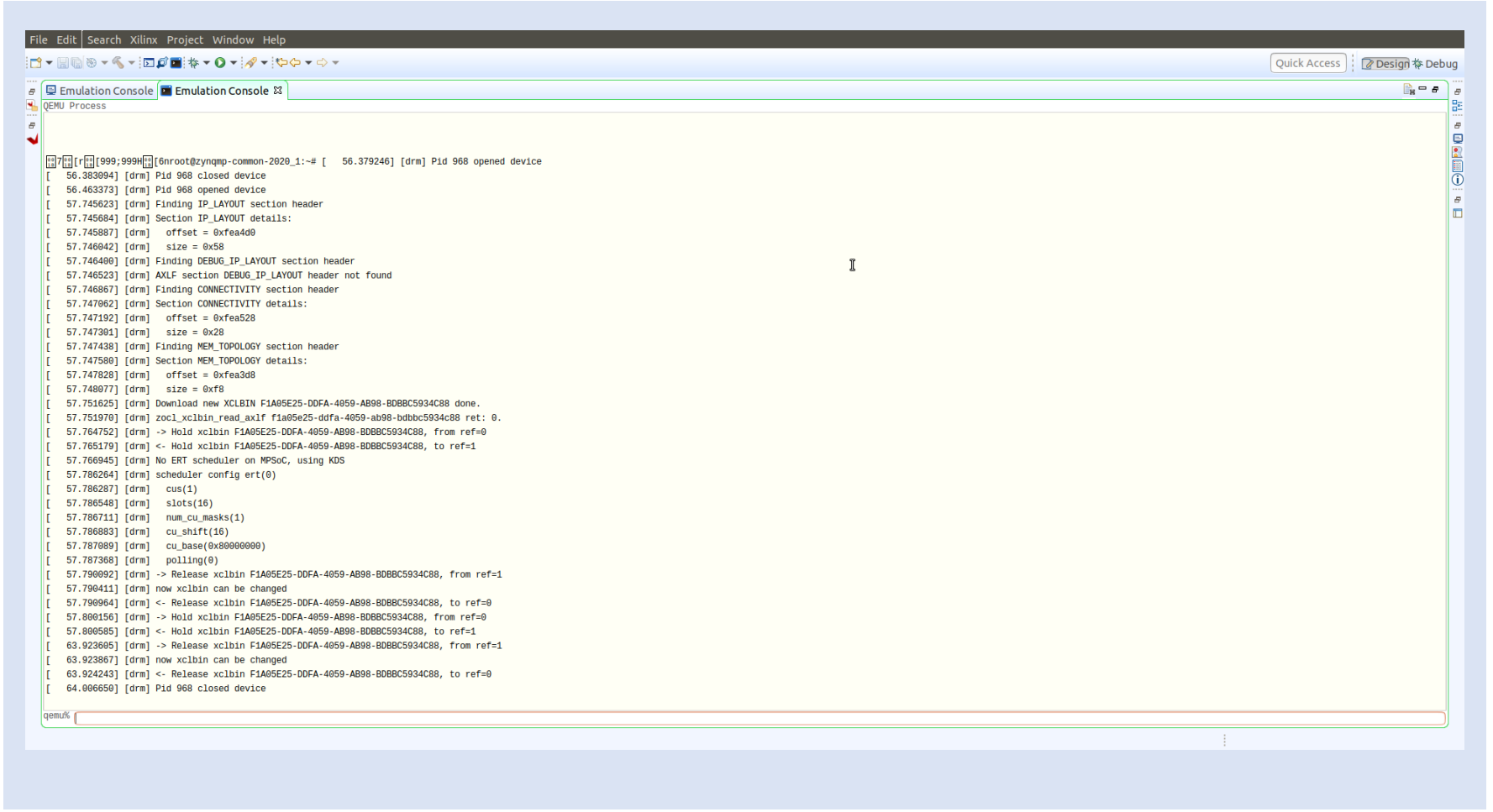
Lab 1: Understanding Vitis Project creation & Flow

Step 25 – When the Debugger loads, click Run and the Application will execute and should show a passed status, click on maximise in the Emulation Console



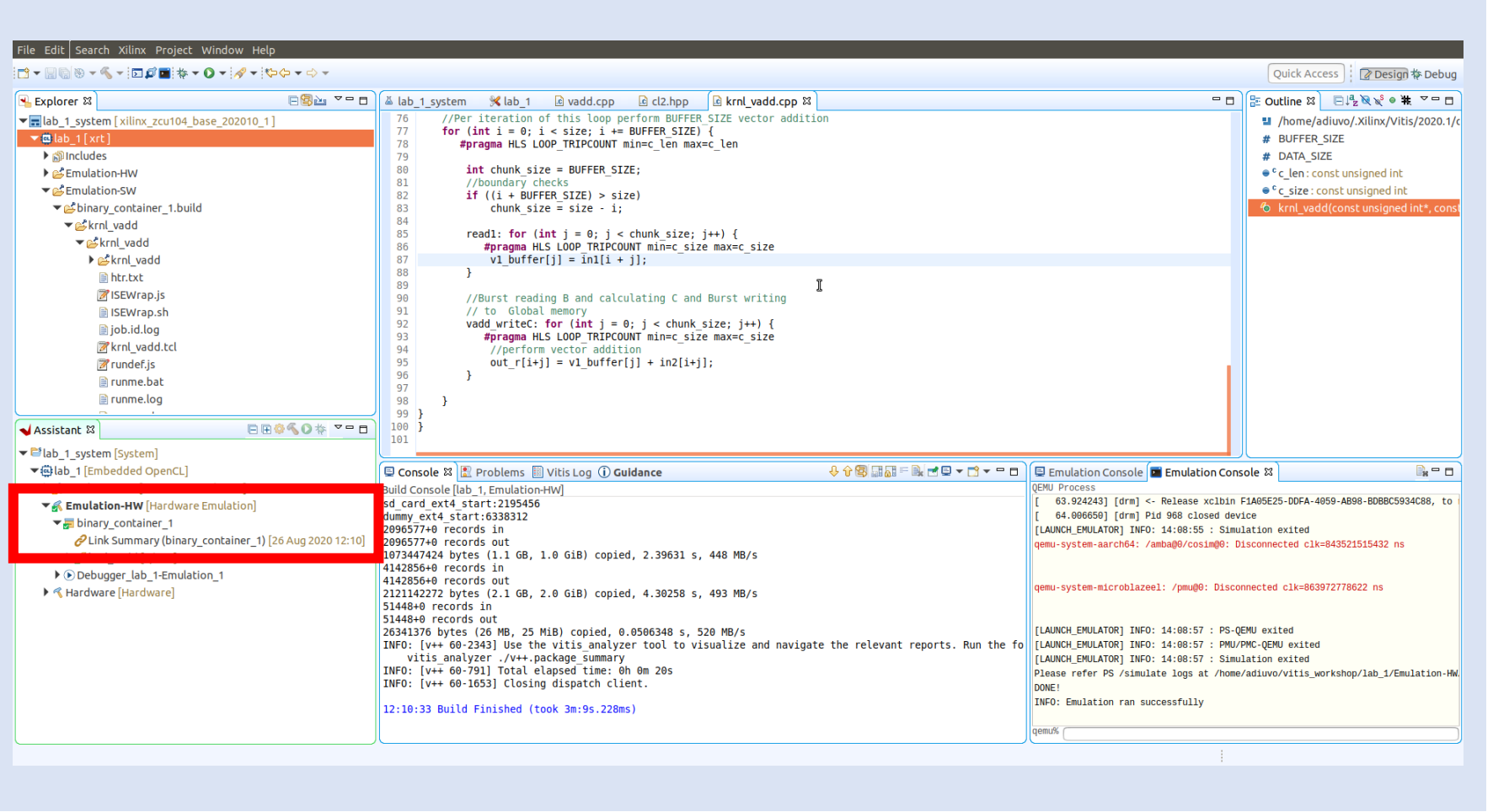
Lab 1: Understanding Vitis Project creation & Flow

Step 26 – In the Emulation console you will see the XCLBinary being loaded as you will on the real hardware.



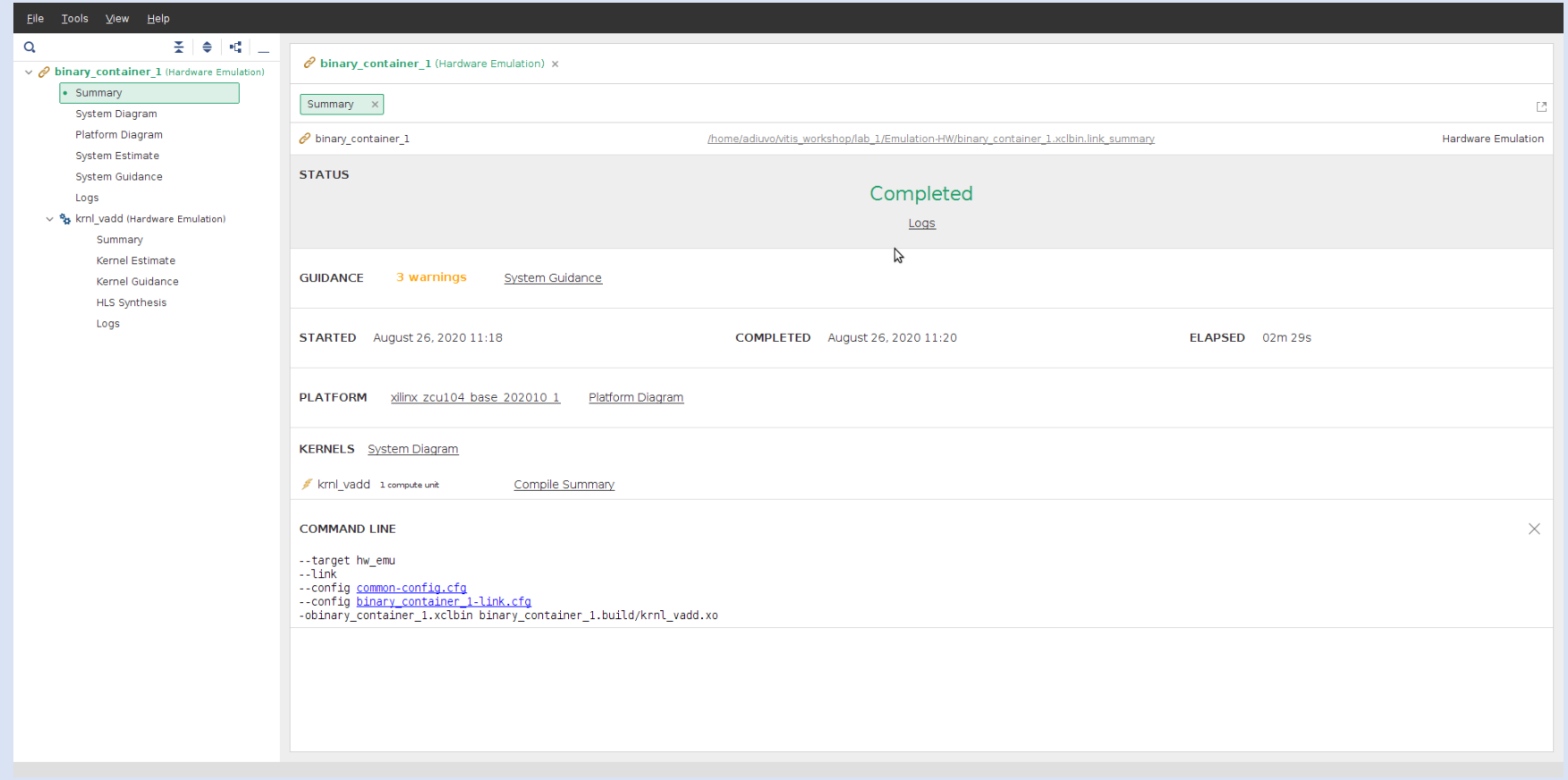
Lab 1: Understanding Vitis Project creation & Flow

Step 24 – Under the assistant view, Emulation hardware select the XCLBin, right click and open the Vitis Analyser.



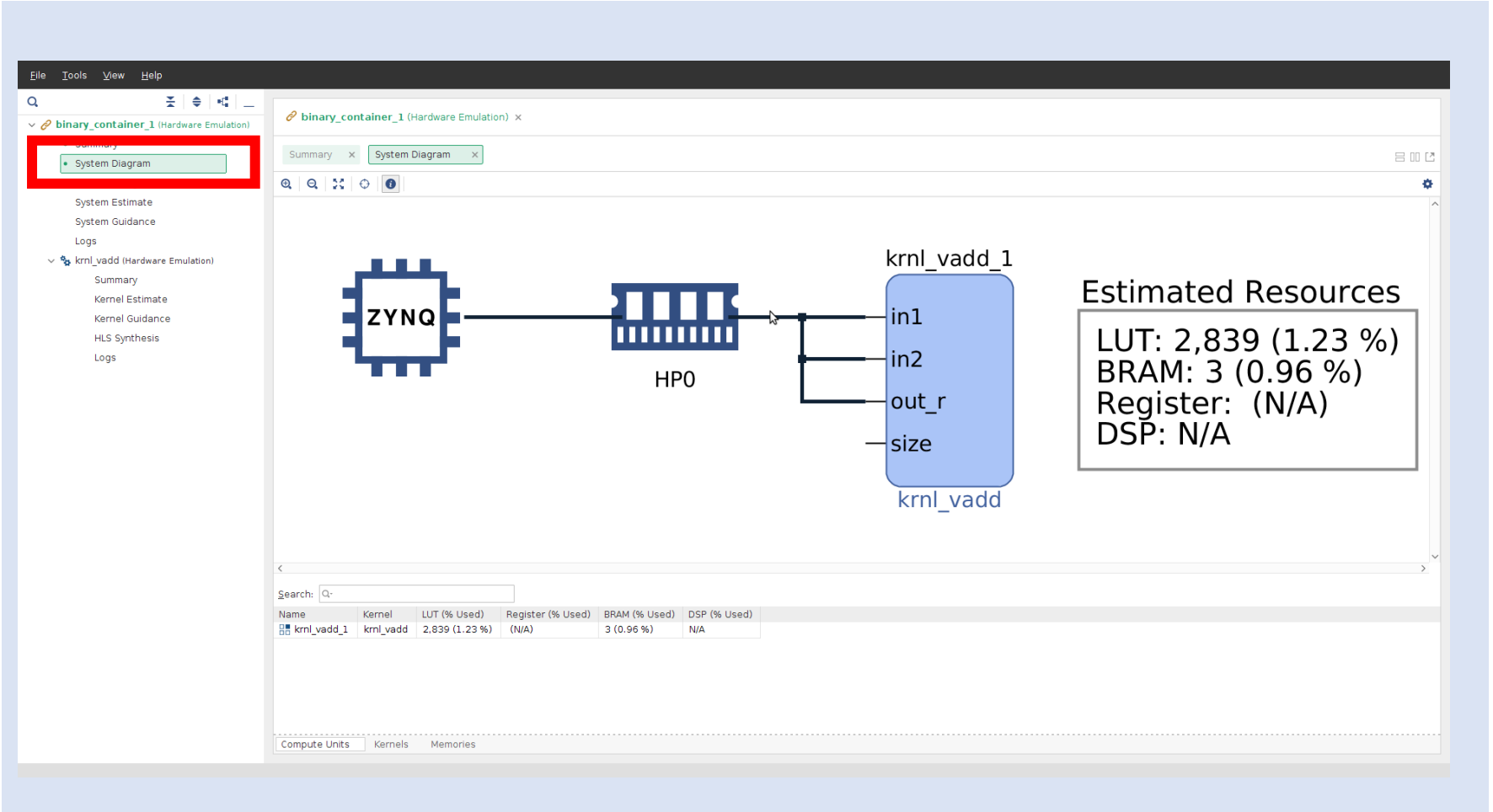
Lab 1: Understanding Vitis Project creation & Flow

Step 24 – The Vitis Analyzer will provide information on the System Performance and Accelerator performance.



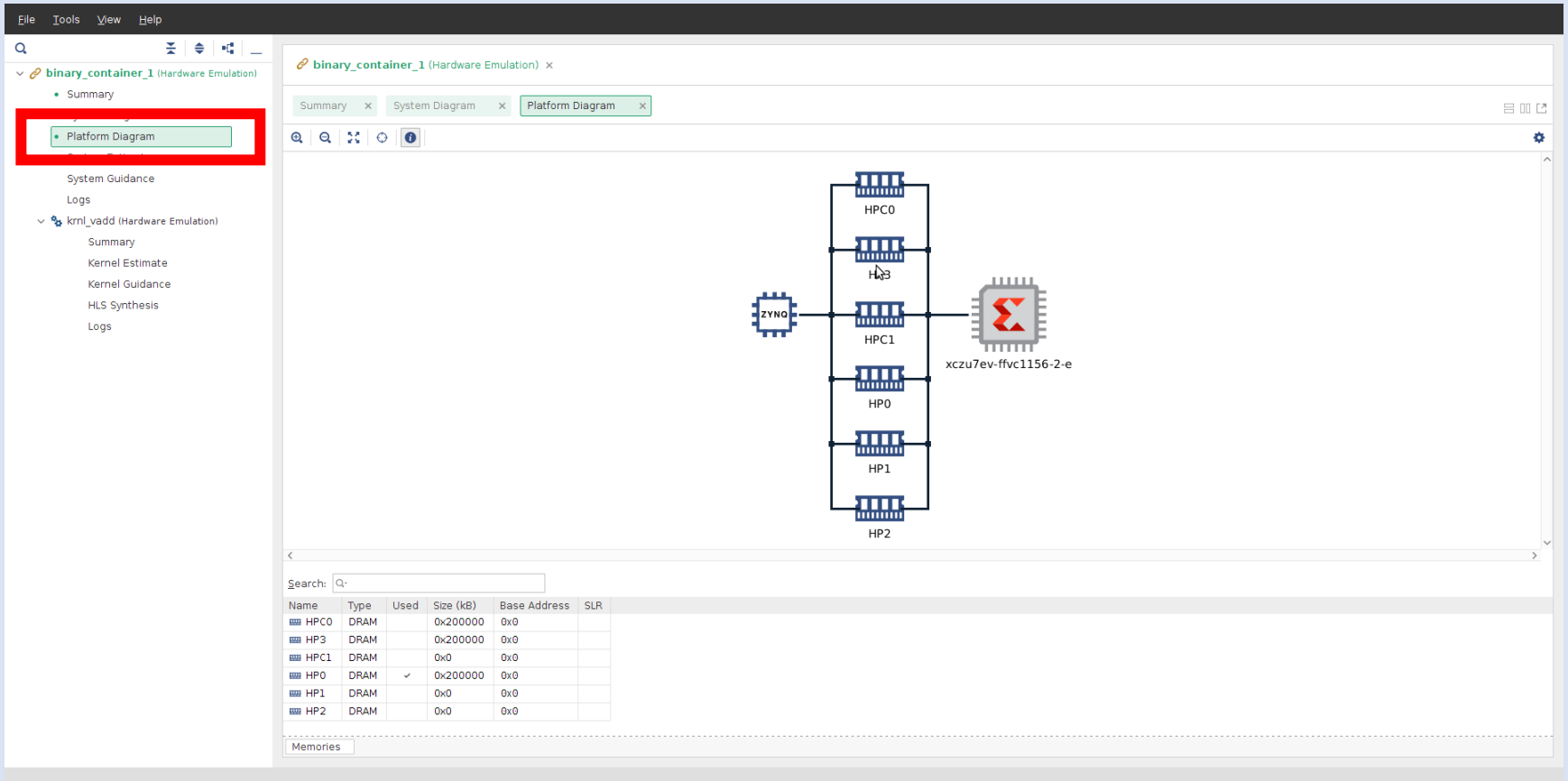
Lab 1: Understanding Vitis Project creation & Flow

Step 24 – Click on the System Design this will show how the accelerator is connected to the Host



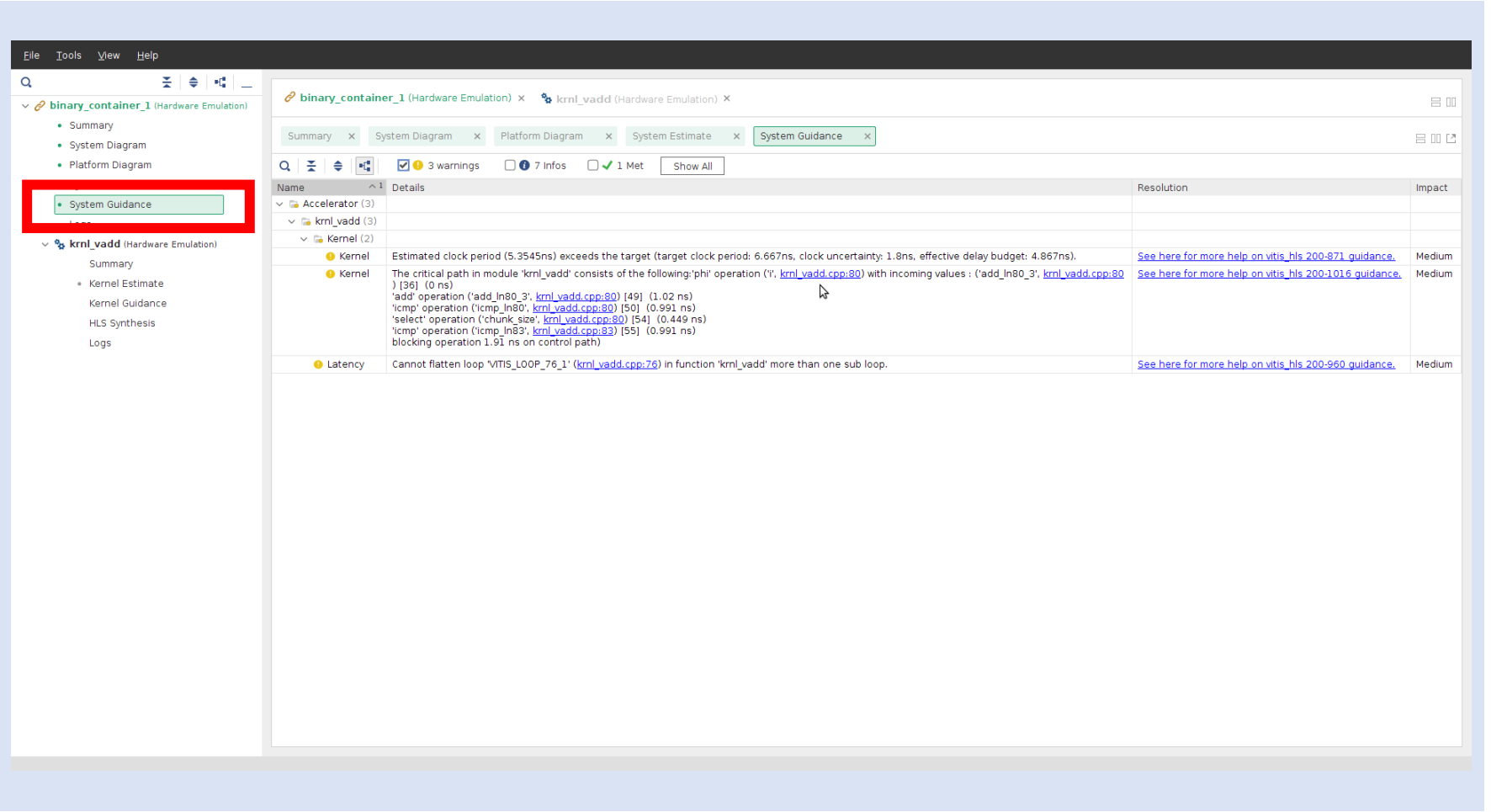
Lab 1: Understanding Vitis Project creation & Flow

Step 24 – Click on the Platform Diagram this will show the connections available in the system



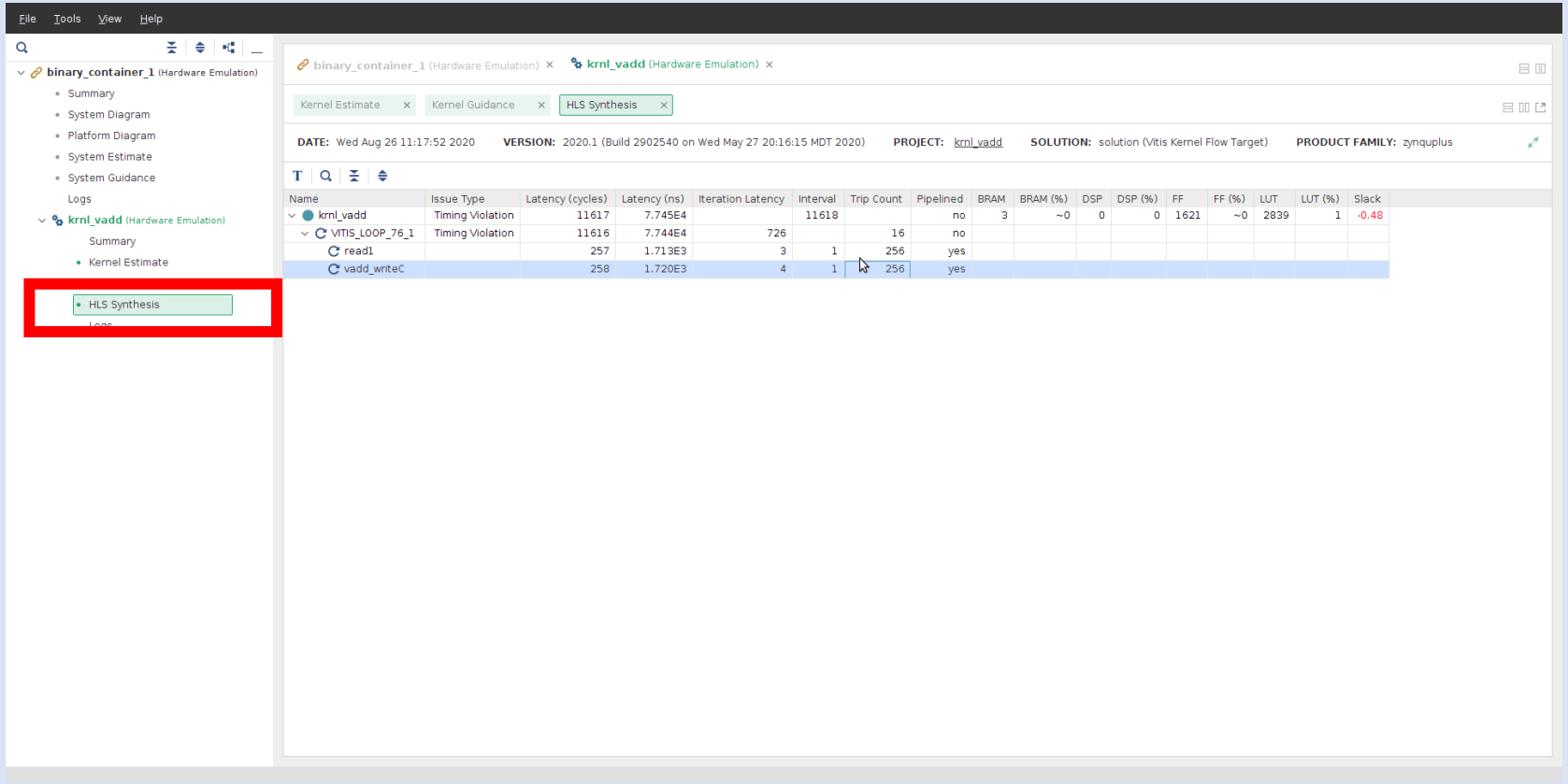
Lab 1: Understanding Vitis Project creation & Flow

Step 24 – Click on the System Guidance, this will report any warnings and resolutions



Lab 1: Understanding Vitis Project creation & Flow

Step 24 – Click on HLS Synthesis, this will show the performance of the accelerator, close the Vitis Analyser



Lab 1 Summary

The concludes lab 1, throughout this lab we have demonstrated

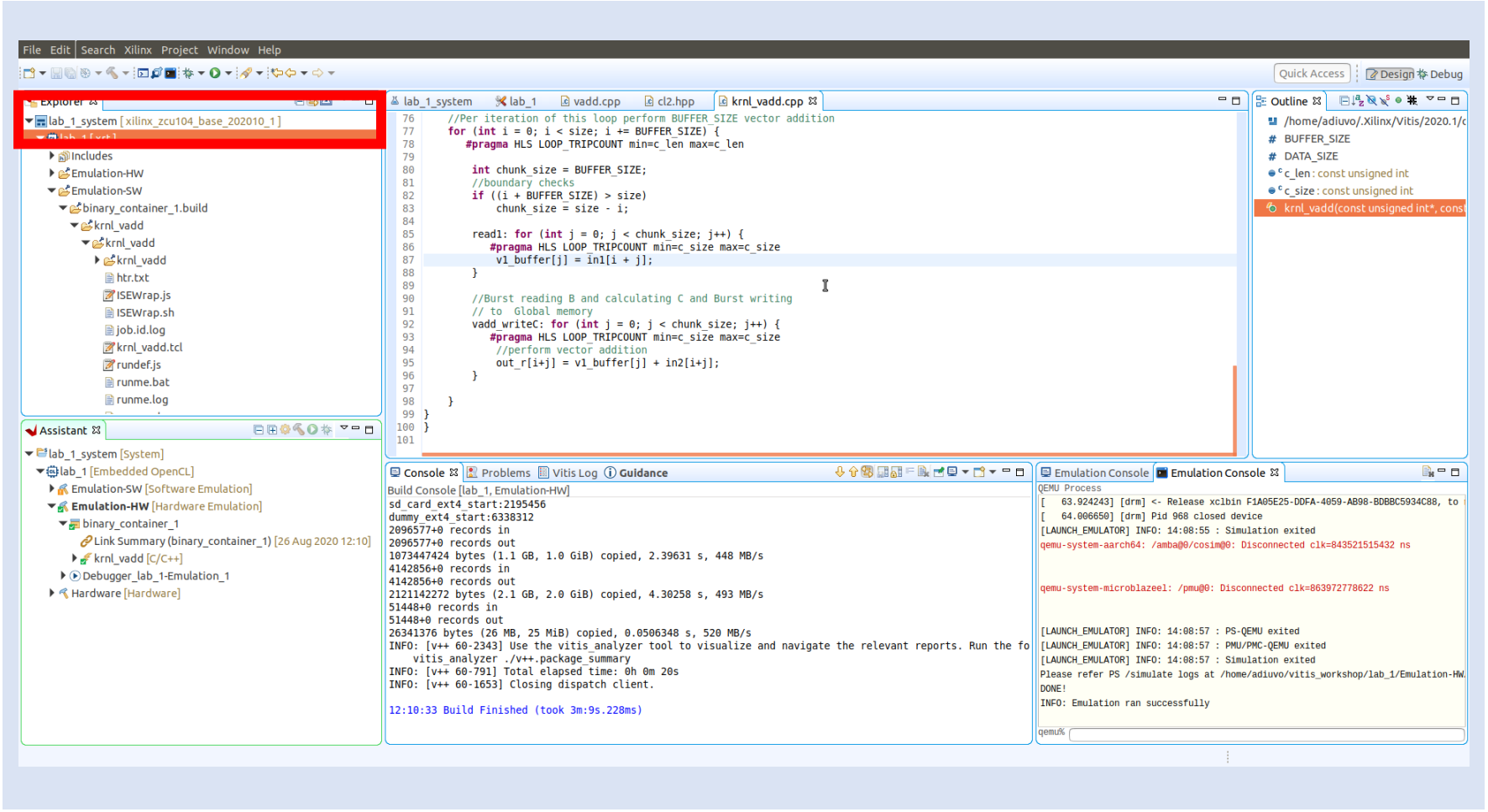
1. How to Open Vitis
2. How to create a project
3. How to Build & Debug a SW Emulation Flow – Including Breakpoints and Single Stepping
4. How to Build & Debug a HW Emulation Flow – Including Waveform
5. How to Examine the Vitis Analysis Report

Lab 2

Instantiating Xilinx Deep Learning Processor

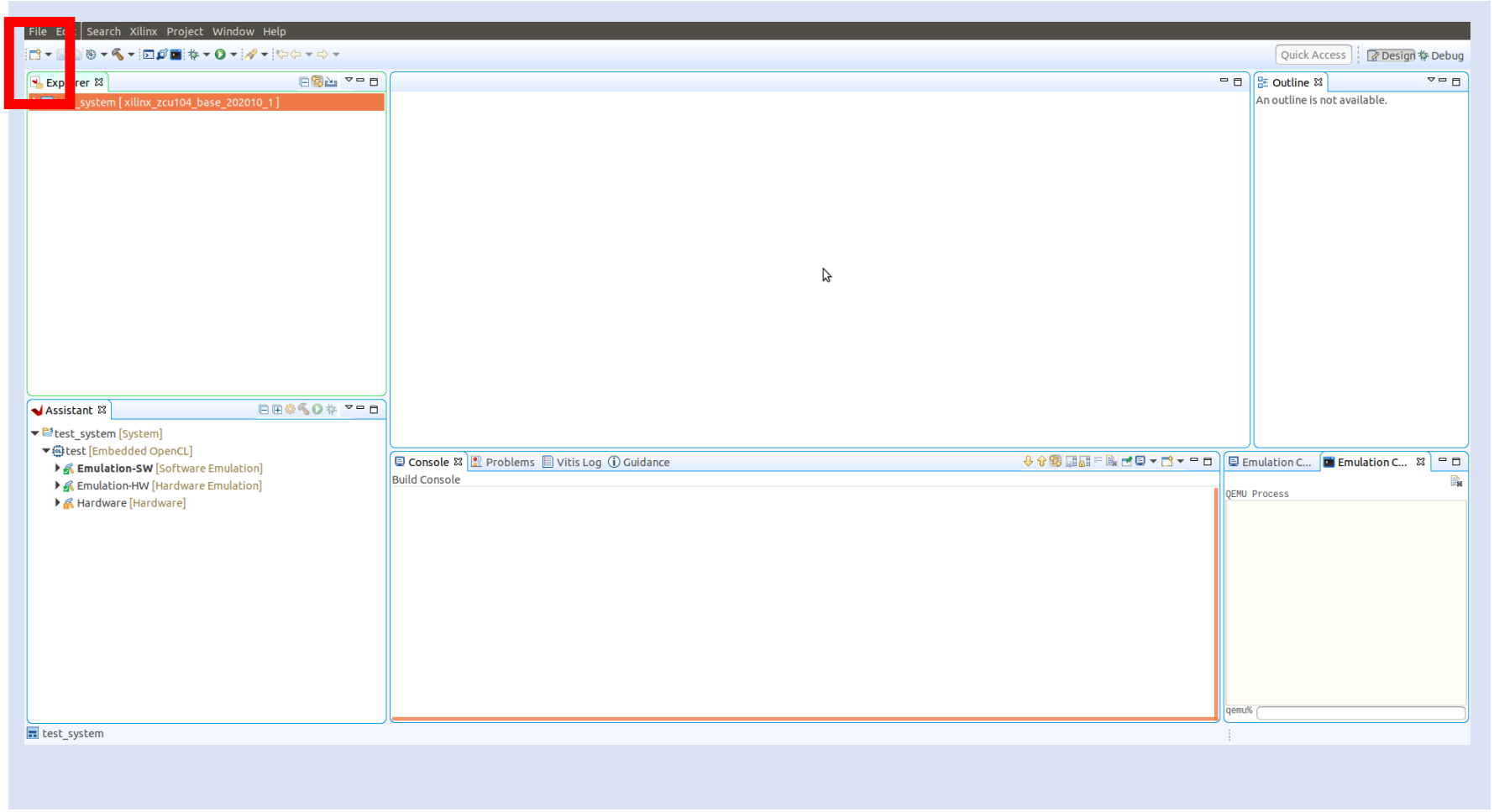
Lab 2: Instantiating Xilinx Deep Learning Processor

Step 1 – Minimize the Lab_1_system project



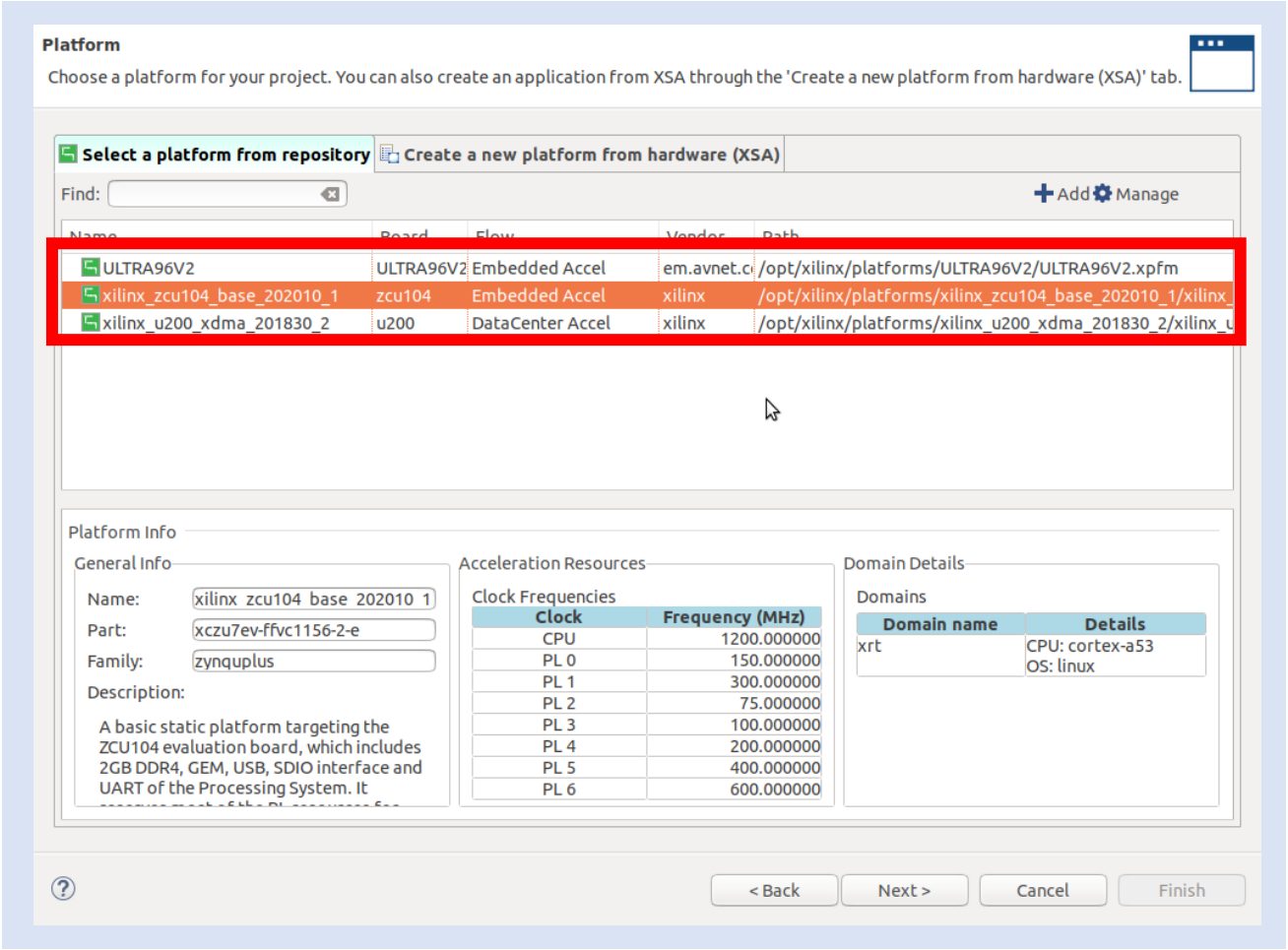
Lab 2: Instantiating Xilinx Deep Learning Processor

Step 2 – Click and create a new application project



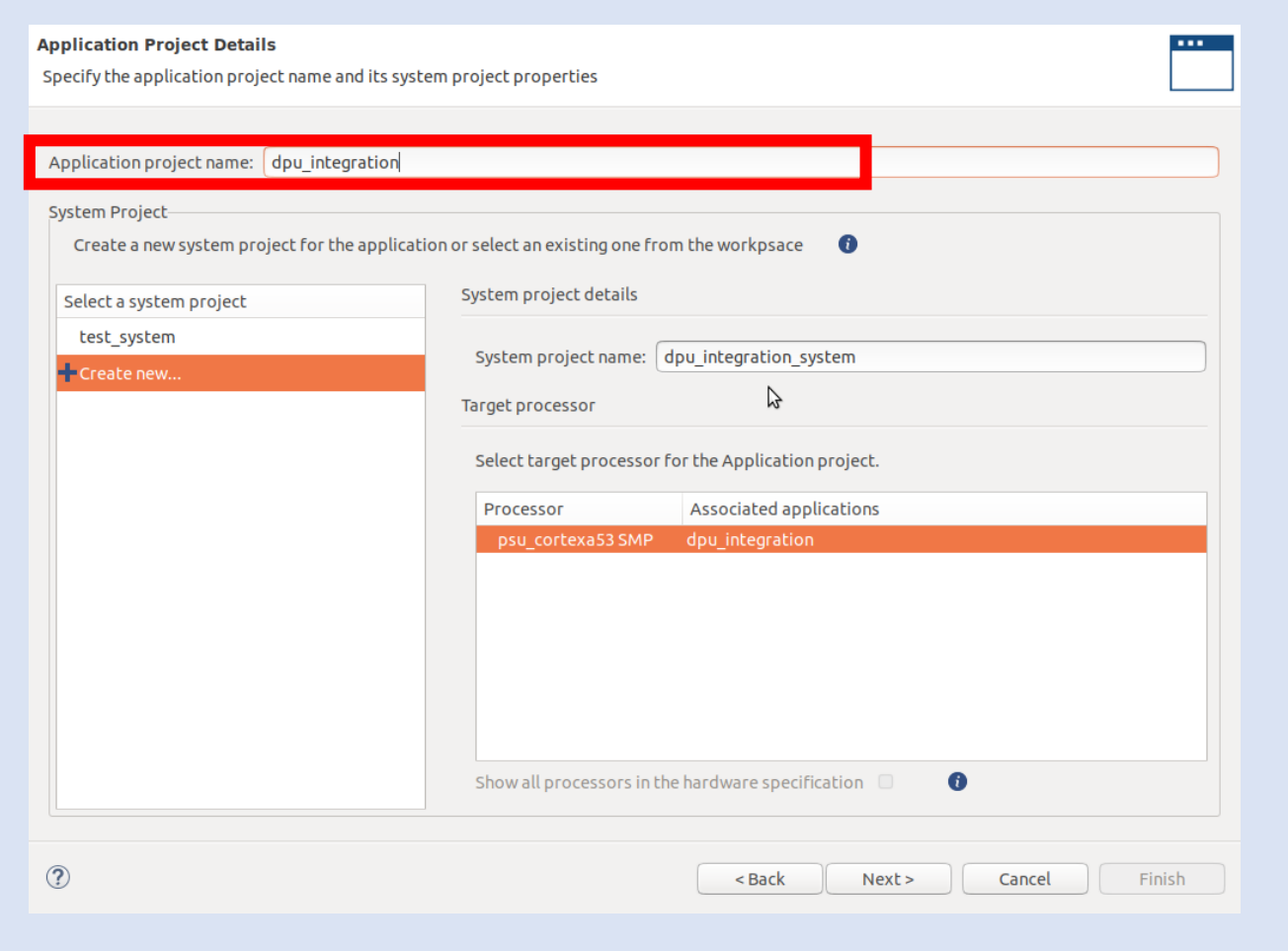
Lab 2: Instantiating Xilinx Deep Learning Processor

Step 3 – Select the platform you desire – for this example ZCU104 and click next



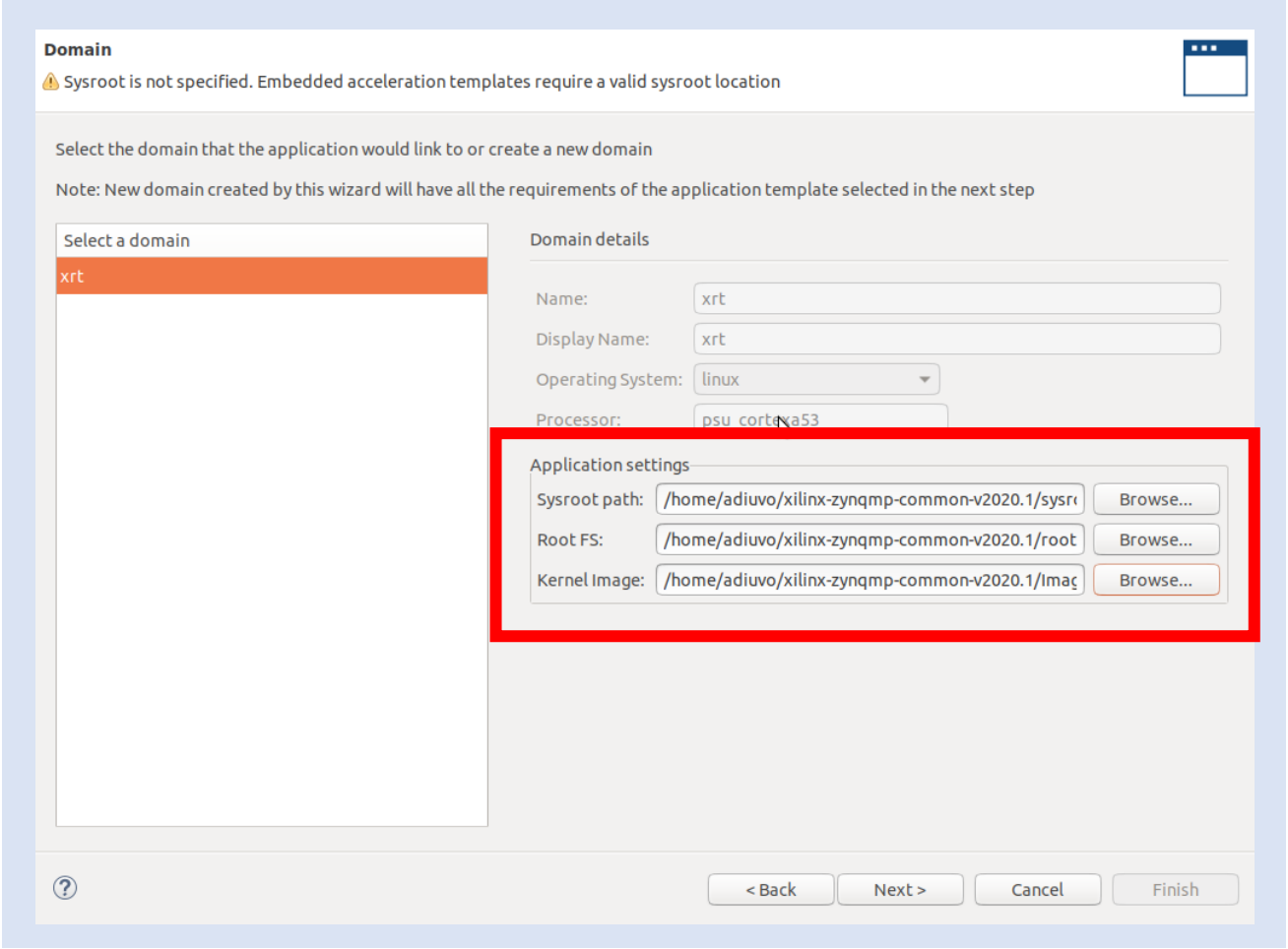
Lab 2: Instantiating Xilinx Deep Learning Processor

Step 4 – Enter a project name and click next



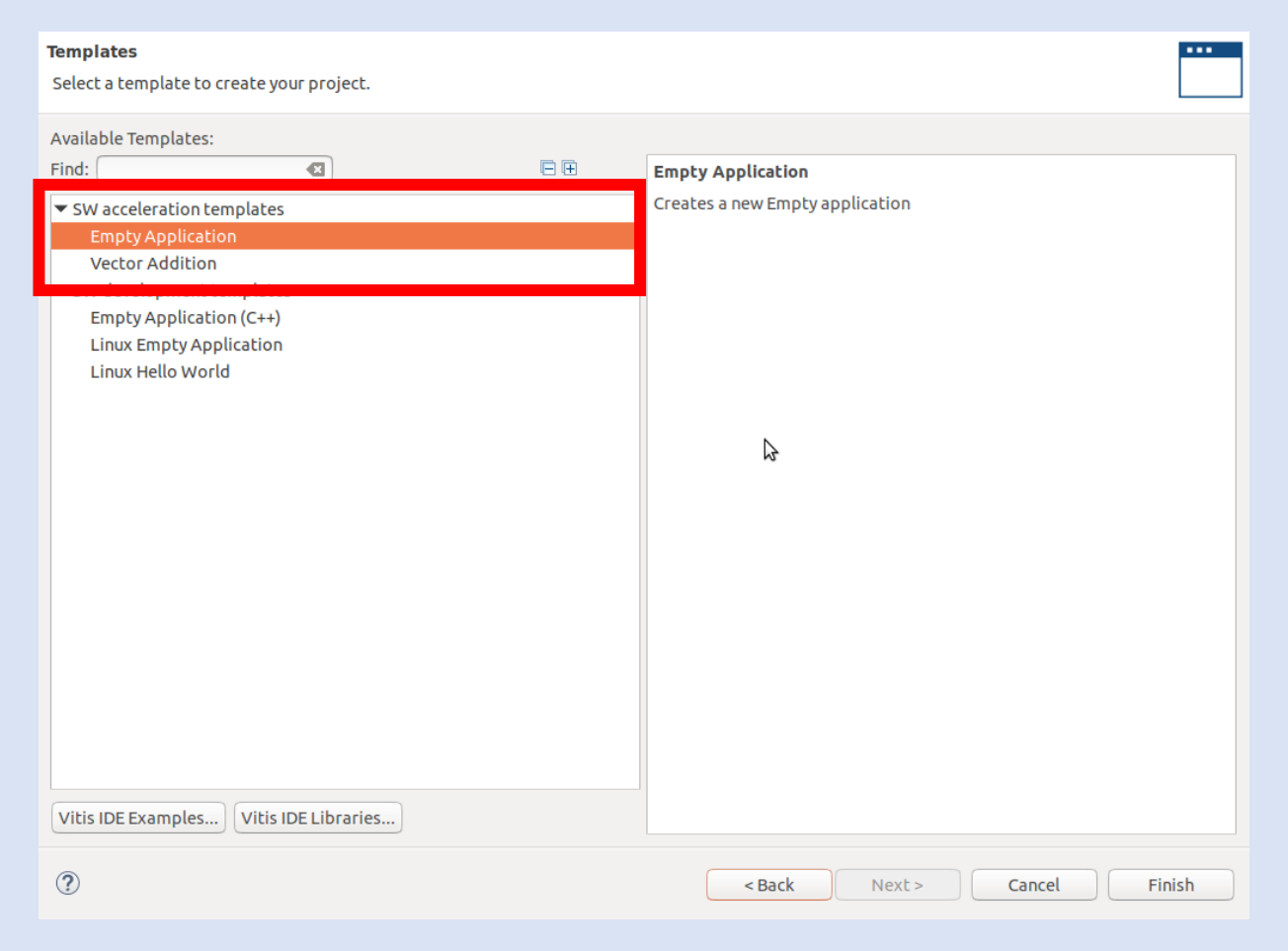
Lab 2: Instantiating Xilinx Deep Learning Processor

Step 5 – Enter the location of the SYSROOT, RootFS and Kernel Image



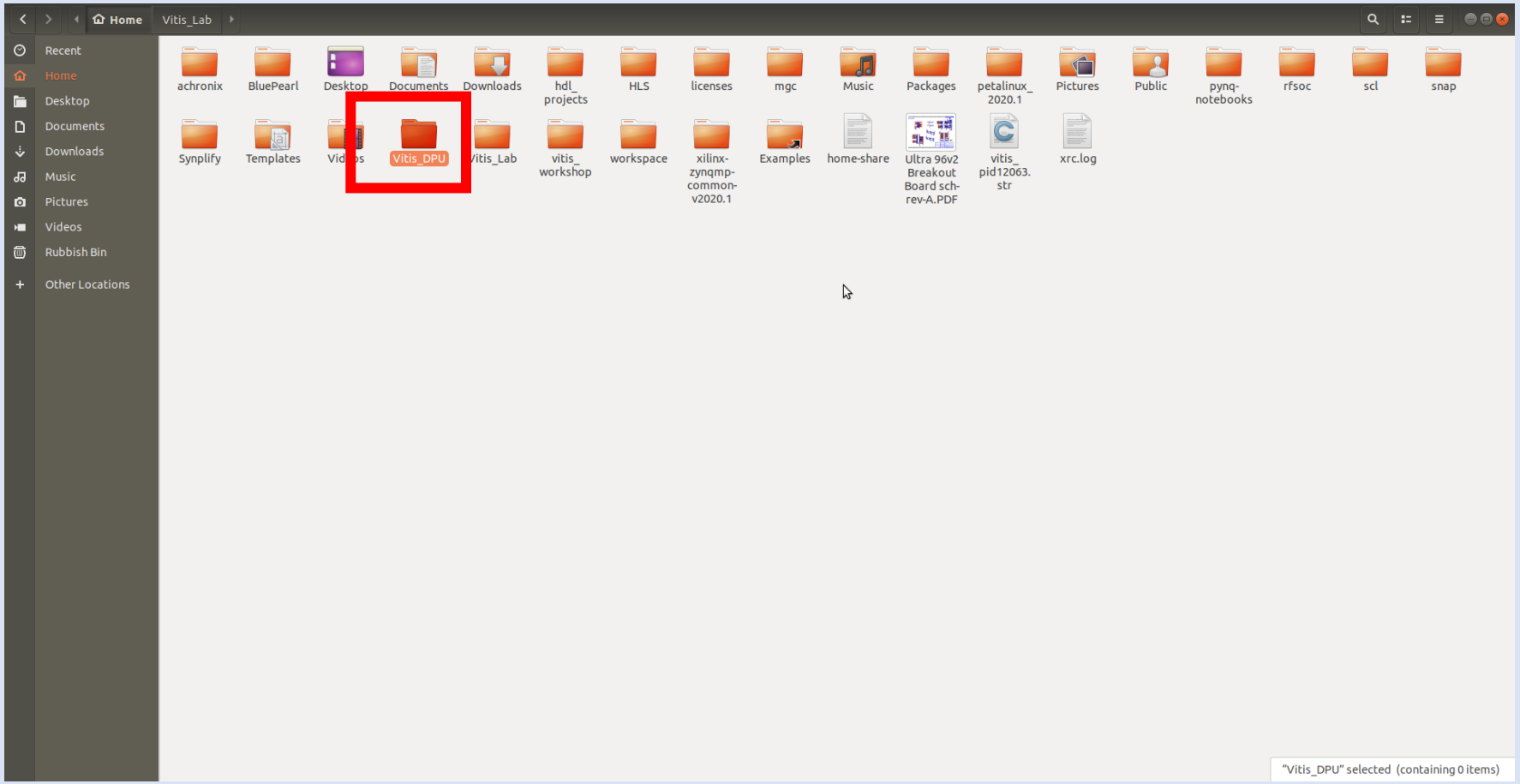
Lab 2: Instantiating Xilinx Deep Learning Processor

Step 6 – Select Empty Application and click on finish



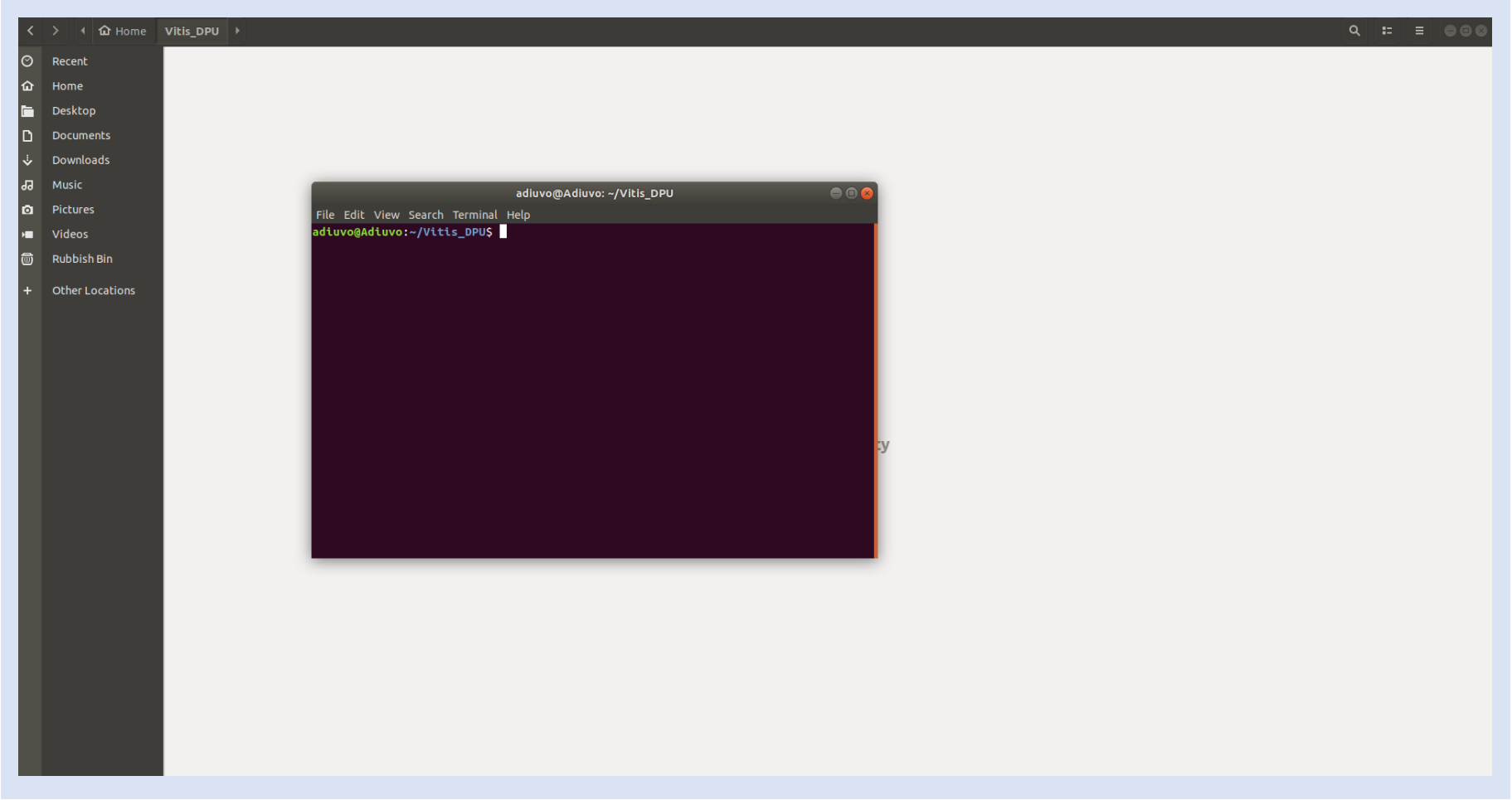
Lab 2: Instantiating Xilinx Deep Learning Processor

Step 7 – Create a new directory in your preferred directory – for example Vitis_DPU



Lab 2: Instantiating Xilinx Deep Learning Processor

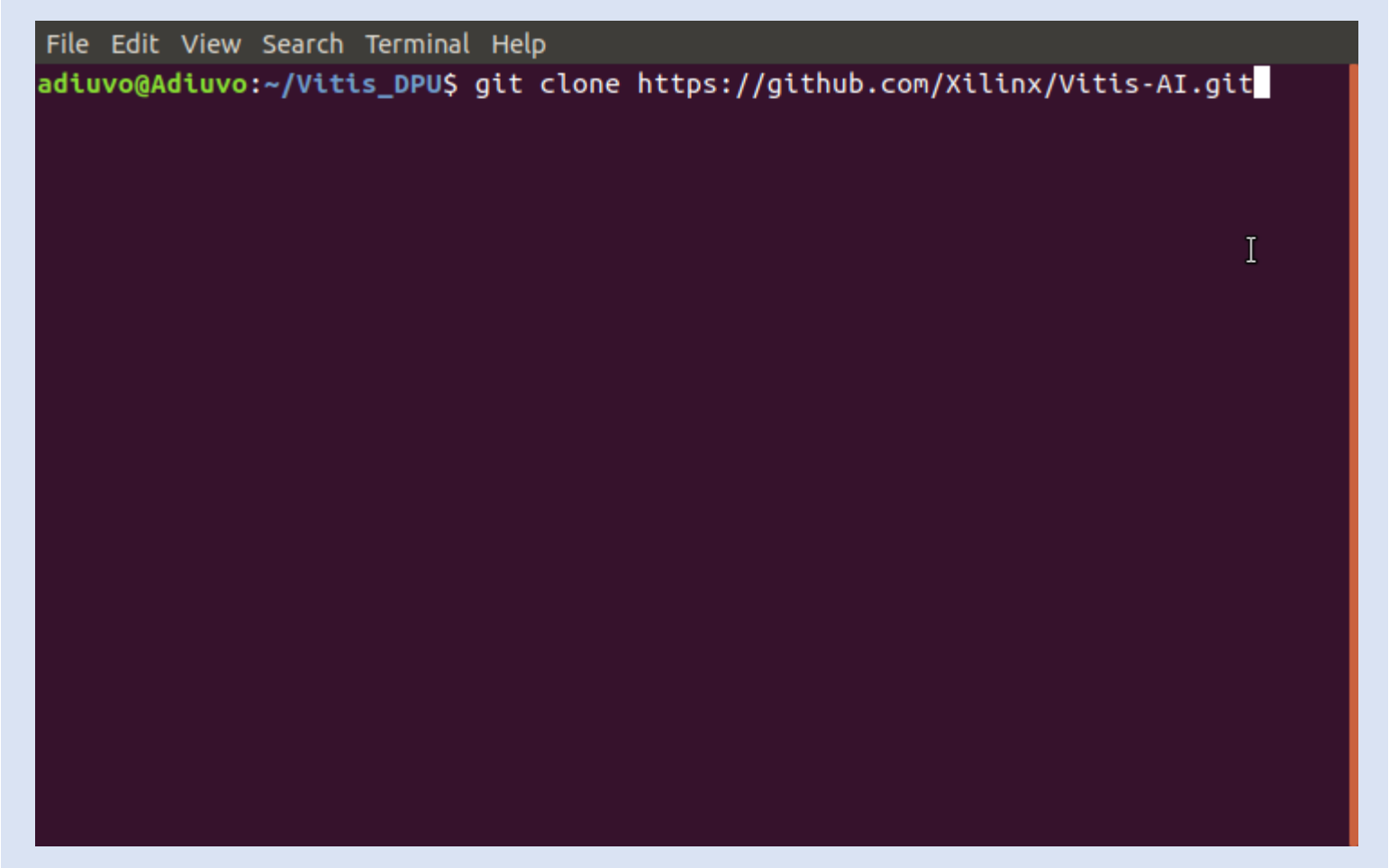
Step 8 – Change directory into the new folder – Open a new terminal in that folder



Lab 2: Instantiating Xilinx Deep Learning Processor

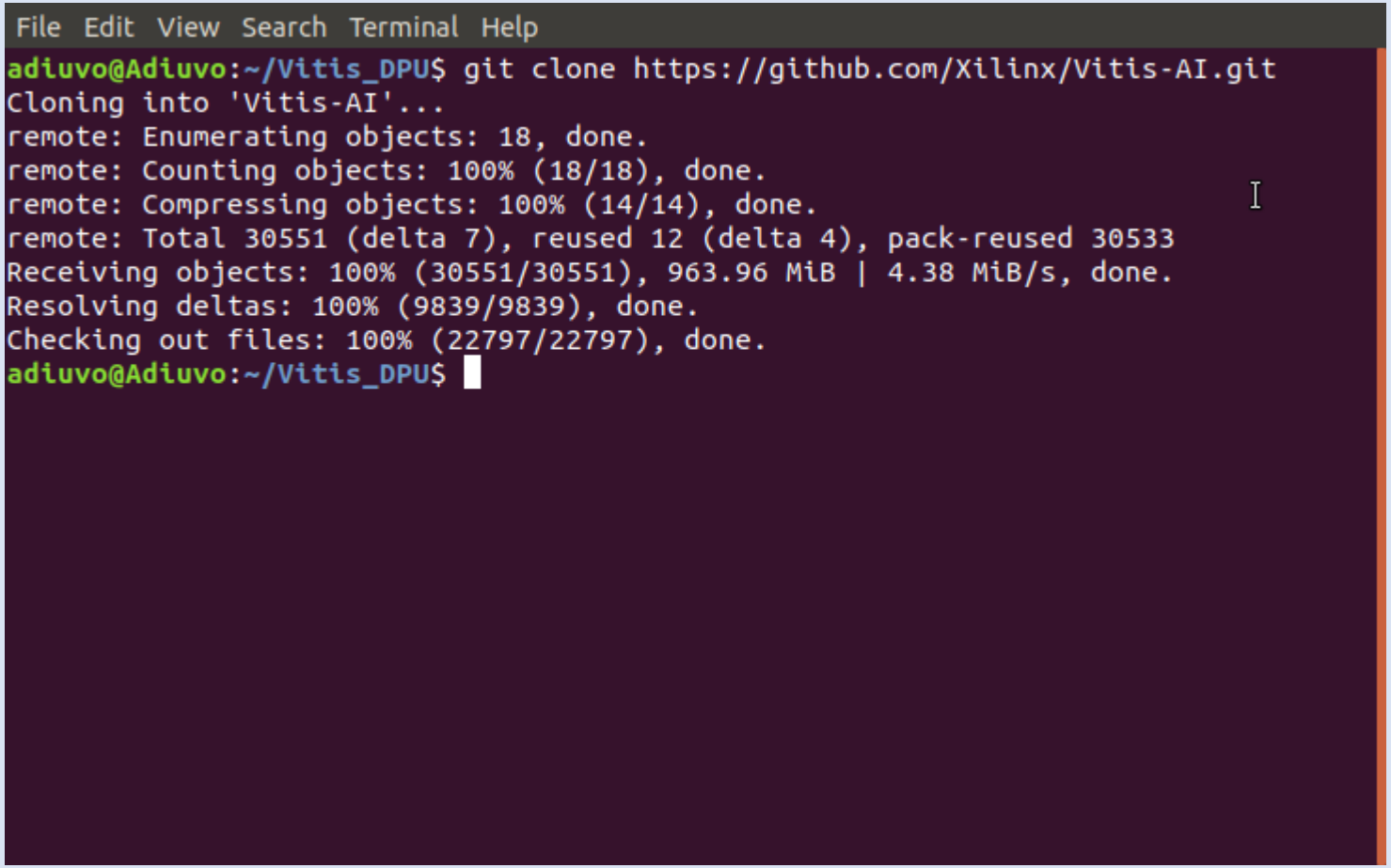
Step 9 – Clone the Vitis-AI github with the command

git clone https://github.com/Xilinx/Vitis-AI.git



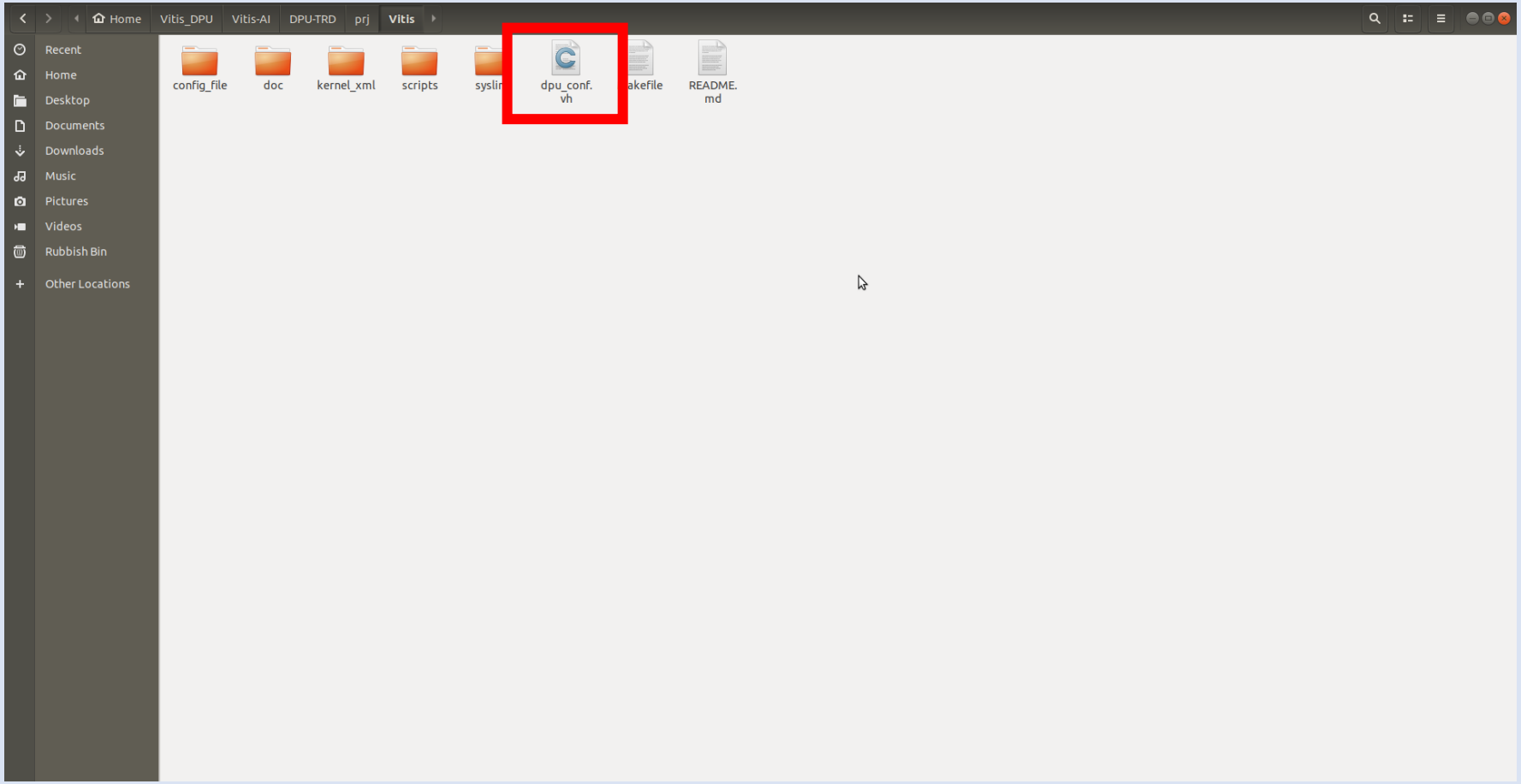
Lab 2: Instantiating Xilinx Deep Learning Processor

Step 10 – This will clone the Vitis-AI tool



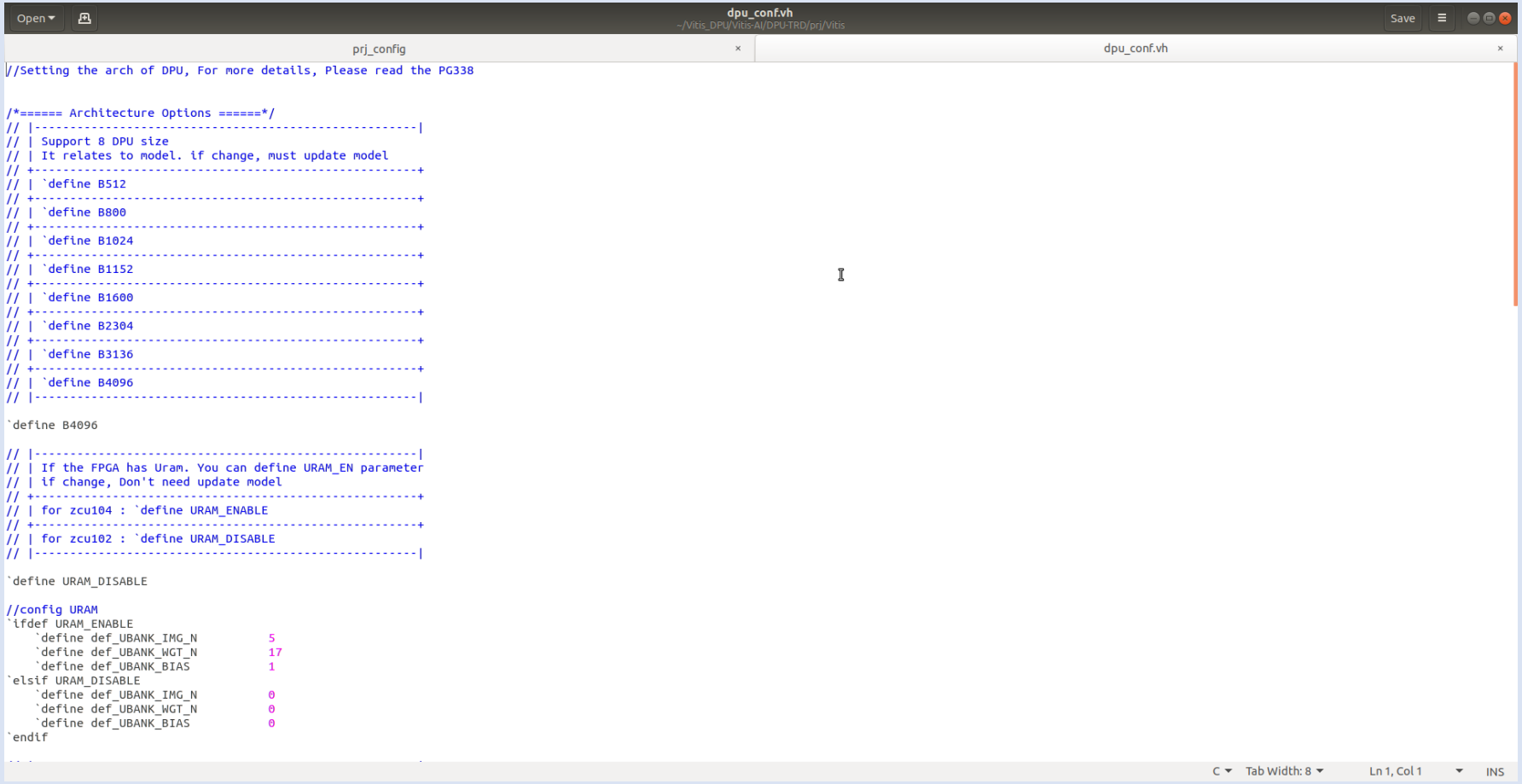
Lab 2: Instantiating Xilinx Deep Learning Processor

Step 11 – Change directory into the Vitis-AI/DSA/DPU-TRD/prj/Vitis and open dpu_conf.vh



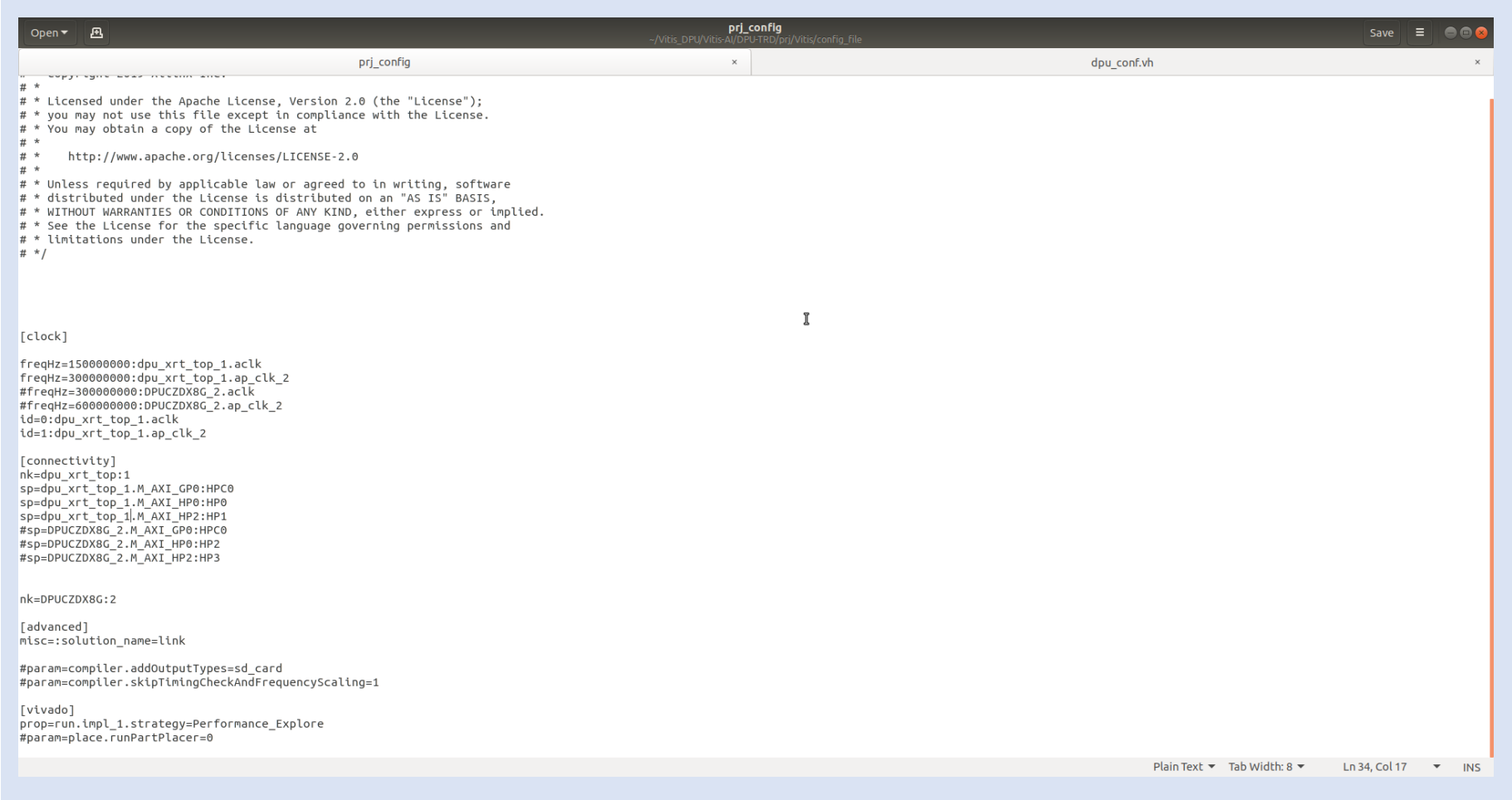
Lab 2: Instantiating Xilinx Deep Learning Processor

Step 12 – to configure the DPU we use dpu_conf.vh – examine the file and close it



Lab 2: Instantiating Xilinx Deep Learning Processor

Step 13 – Change directory to the config_file open the file prj_config for editing – update the file to be as shown below



Lab 2: Instantiating Xilinx Deep Learning Processor

Step 14 – In the terminal window export the platform location and make the XO

export SDX_PLATFORM=<location>

make binary_container_1/dpu.xo DEVICE=zcu104

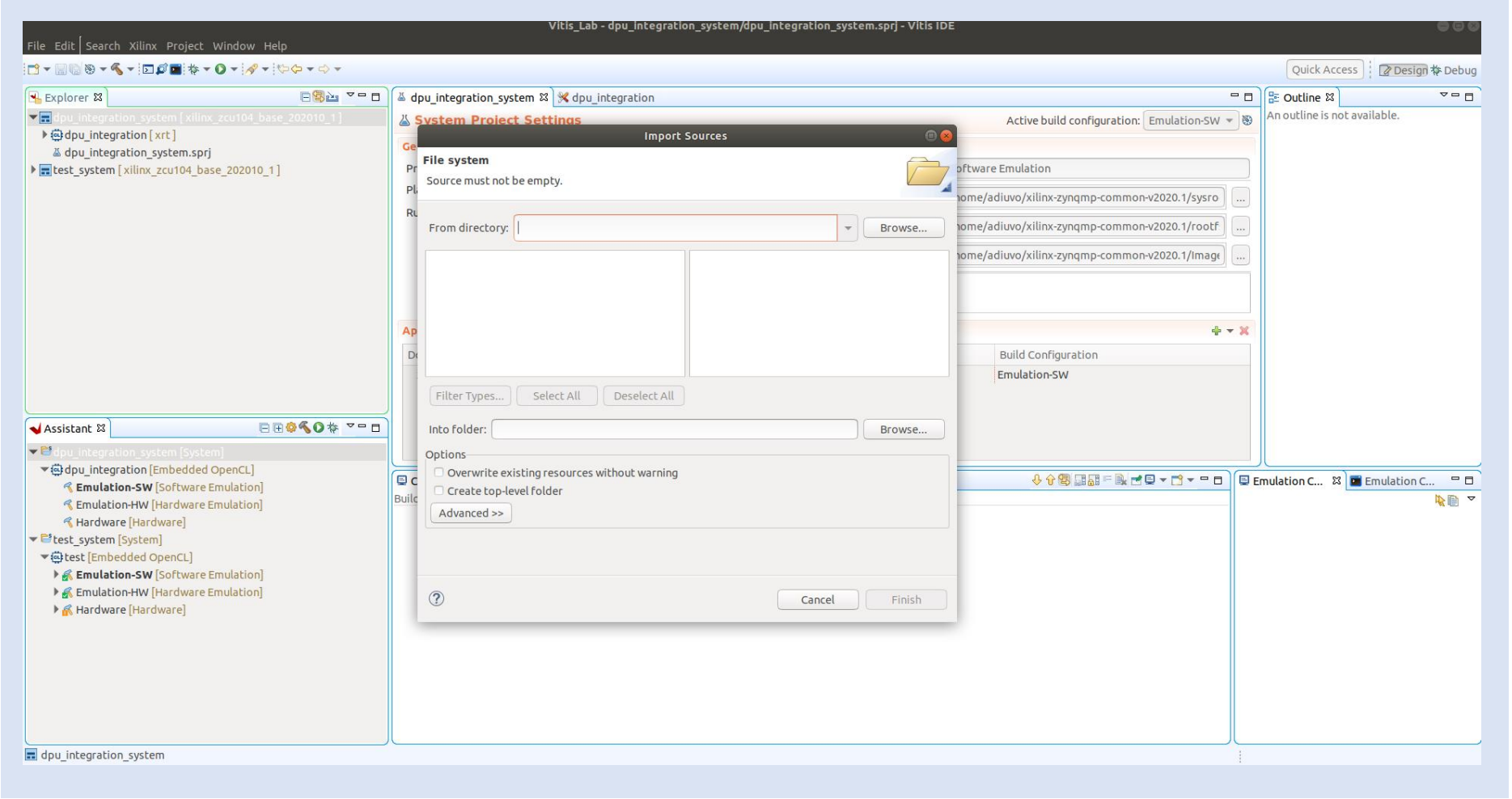
```
File  Edit  View  Search  Terminal  Help
adiuvo@Adiuvo:~/Vitis_DPU/Vitis-AI/DPU-TRD/prj/Vitis$ export SDX_PLATFORM=/opt/xilinx/platforms/xilinx_zcu104_base_202010_1/xilinx_zcu104_base_202010_1.xpfm
adiuvo@Adiuvo:~/Vitis_DPU/Vitis-AI/DPU-TRD/prj/Vitis$ echo $SDX_PLATFORM
/opt/xilinx/platforms/xilinx_zcu104_base_202010_1/xilinx_zcu104_base_202010_1.xpfm
adiuvo@Adiuvo:~/Vitis_DPU/Vitis-AI/DPU-TRD/prj/Vitis$ make binary_container_1/dpu.xo DEVICE=zcu104
/tools/Xilinx/Vivado/2020.1/bin/vivado -mode batch -source scripts/gen_dpu_xo.tcl -tclargs binary_container_1/dpu.xo DPUCZDX8G hw zcu104

****** Vivado v2020.1 (64-bit)
  **** SW Build 2902540 on Wed May 27 19:54:35 MDT 2020
  **** IP Build 2902112 on Wed May 27 22:43:36 MDT 2020
    ** Copyright 1986-2020 Xilinx, Inc. All Rights Reserved.

source scripts/gen_dpu_xo.tcl
# if { $::argc != 4 } {
#     puts "ERROR: Program \"$::argv0\" requires 4 arguments!\n"
#     puts "Usage: $::argv0 <xoname> <krnl_name> <target> <device>\n"
#     exit
# }
# set xoname    [lindex $::argv 0]
# set krnl_name [lindex $::argv 1]
# set target    [lindex $::argv 2]
# set device    [lindex $::argv 3]
# set suffix "${krnl_name}_${target}_${device}"
# source -notrace ./scripts/package_dpu_kernel.tcl
INFO: [IP_Flow 19-5654] Module 'DPUCZDX8G' uses SystemVerilog sources with a Verilog top file. These SystemVerilog files will not be analysed by the packager.
INFO: [IP_Flow 19-1842] HDL Parser: Found include file "src/arch_def.vh" from the top-level HDL file.
INFO: [IP_Flow 19-1842] HDL Parser: Found include file "/home/adiuvo/Vitis_DPU/Vitis-AI/DPU-TRD/prj/Vitis/dpu_conf.vh" from the top-level HDL file.
INFO: [IP_Flow 19-1841] HDL Parser: Add include file "/home/adiuvo/Vitis_DPU/Vitis-AI/DPU-TRD/prj/Vitis/dpu_conf.vh" to file group xilinx_anylanguagesynthesis.
INFO: [IP_Flow 19-1841] HDL Parser: Add include file "/home/adiuvo/Vitis_DPU/Vitis-AI/DPU-TRD/prj/Vitis/dpu_conf.vh" to file group xilinx_anylanguagebehavioralsimulation.
INFO: [IP_Flow 19-234] Refreshing IP repositories
INFO: [IP_Flow 19-1704] No user IP repositories specified
INFO: [IP_Flow 19-2313] Loaded Vivado IP repository '/tools/Xilinx/Vivado/2020.1/data/ip'.
INFO: [IP_Flow 19-5107] Inferred bus interface 'aclk' of definition 'xilinx.com:signal:clock:1.0' (from X_INTERFACE_INFO parameter from HDL file).
INFO: [IP_Flow 19-5107] Inferred bus interface 'aclk' of definition 'xilinx.com:signal:clock:1.0' (from 'X_INTERFACE_INFO' attribute).
INFO: [IP_Flow 19-5107] Inferred bus interface 'ap_clk_2' of definition 'xilinx.com:signal:clock:1.0' (from X_INTERFACE_INFO parameter from HDL file).
INFO: [IP_Flow 19-5107] Inferred bus interface 'ap_clk_2' of definition 'xilinx.com:signal:clock:1.0' (from 'X_INTERFACE_INFO' attribute).
INFO: [IP_Flow 19-5107] Inferred bus interface 'ap_rst_n_2' of definition 'xilinx.com:signal:reset:1.0' (from X_INTERFACE_INFO parameter from HDL file).
INFO: [IP_Flow 19-5107] Inferred bus interface 'ap_rst_n_2' of definition 'xilinx.com:signal:reset:1.0' (from 'X_INTERFACE_INFO' attribute).
INFO: [IP_Flow 19-5107] Inferred bus interface 'aresetn' of definition 'xilinx.com:signal:reset:1.0' (from X_INTERFACE_INFO parameter from HDL file).
INFO: [IP_Flow 19-5107] Inferred bus interface 'aresetn' of definition 'xilinx.com:signal:reset:1.0' (from 'X_INTERFACE_INFO' attribute).
INFO: [IP_Flow 19-5107] Inferred bus interface 'M_AXI_GP0' of definition 'xilinx.com:interface:aximm:1.0' (from Xilinx Repository).
INFO: [IP_Flow 19-5107] Inferred bus interface 'M_AXI_HP0' of definition 'xilinx.com:interface:aximm:1.0' (from Xilinx Repository).
INFO: [IP_Flow 19-5107] Inferred bus interface 'M_AXI_HP2' of definition 'xilinx.com:interface:aximm:1.0' (from Xilinx Repository).
INFO: [IP_Flow 19-5107] Inferred bus interface 'S_AXI_CONTROL' of definition 'xilinx.com:interface:aximm:1.0' (from Xilinx Repository).
INFO: [IP_Flow 19-5107] Inferred bus interface 'interrupt' of definition 'xilinx.com:signal:interrupt:1.0' (from Xilinx Repository).
```

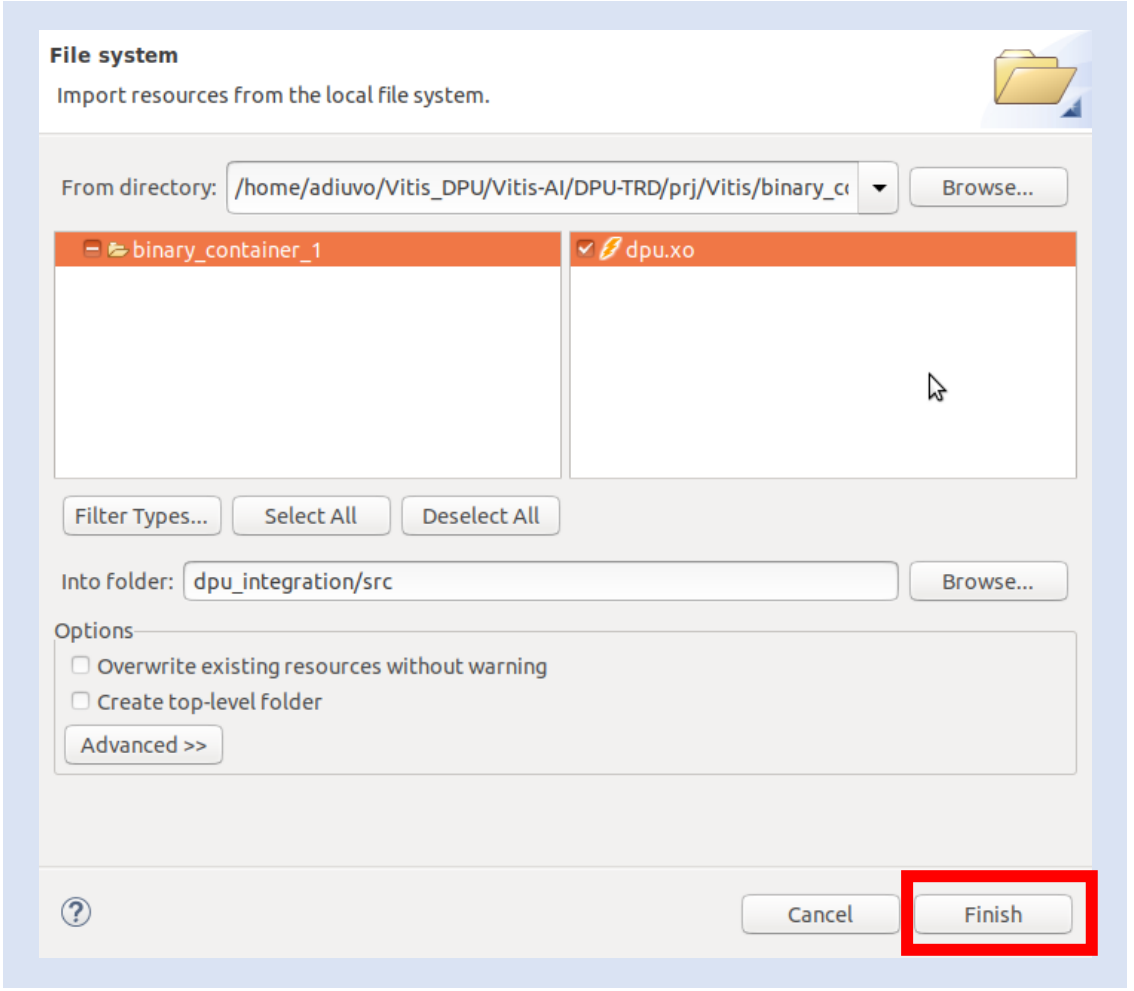
Lab 2: Instantiating Xilinx Deep Learning Processor

Step 15 – Back in Vitis, click on the src folder in the project, right click and select import sources



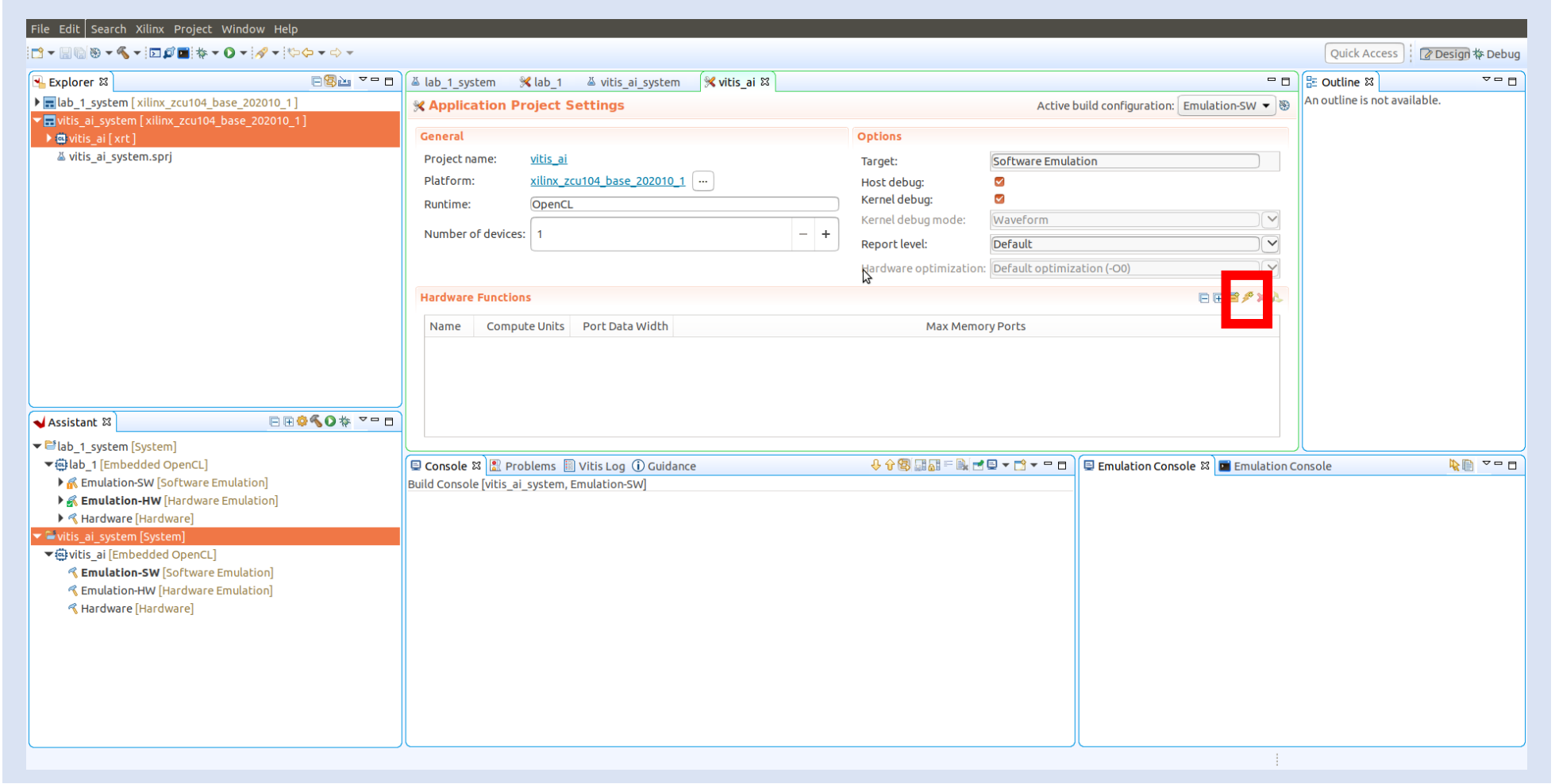
Lab 2: Instantiating Xilinx Deep Learning Processor

Step 16 – Navigate to the Vitis-AI directory and the binary_container_1 folder. Select the DPU.xo and click finish



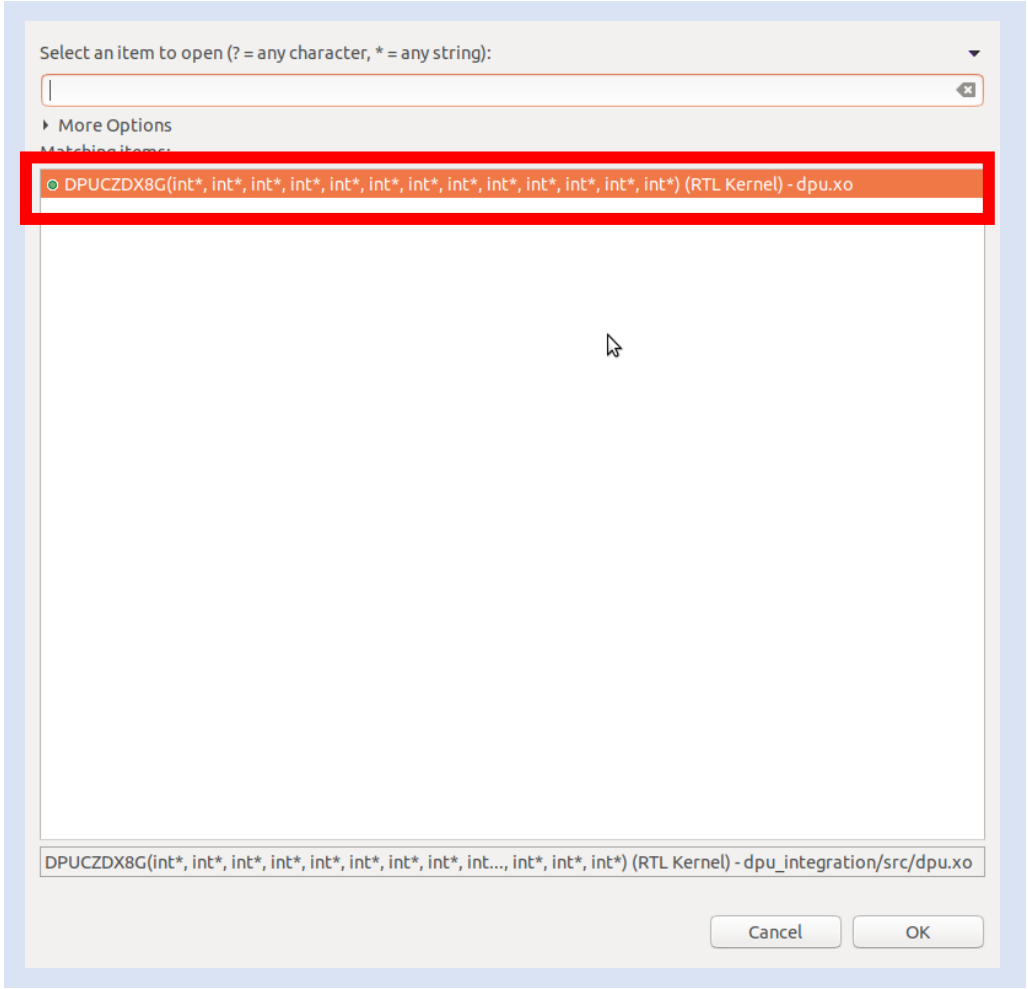
Lab 2: Instantiating Xilinx Deep Learning Processor

Step 17 – Click on the lightening bolt icon to select the function to accelerate to hardware



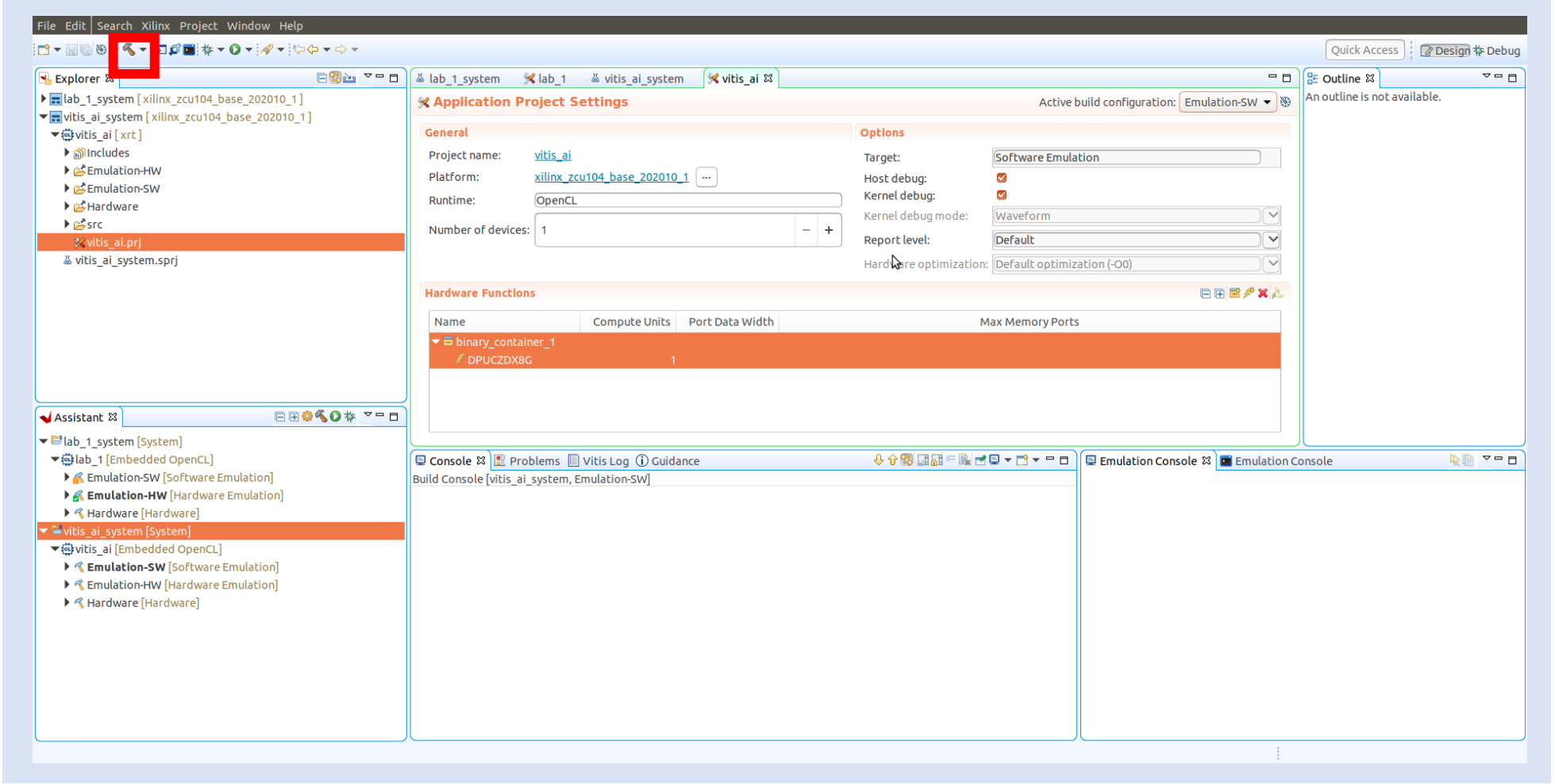
Lab 2: Instantiating Xilinx Deep Learning Processor

Step 18 – Select the DPU function from the list of matching items



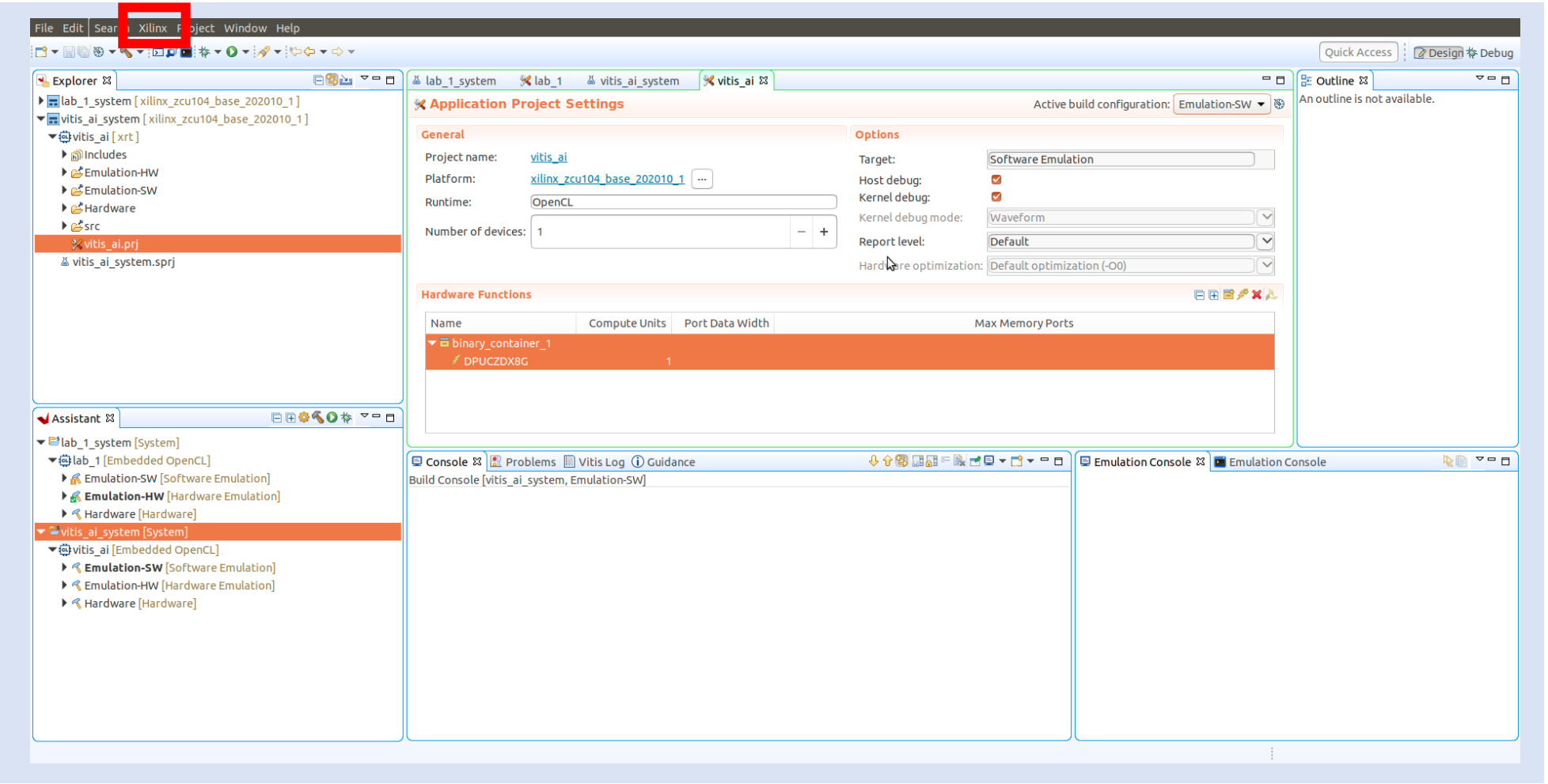
Lab 2: Instantiating Xilinx Deep Learning Processor

Step 19 – Click on build and the design will implement.



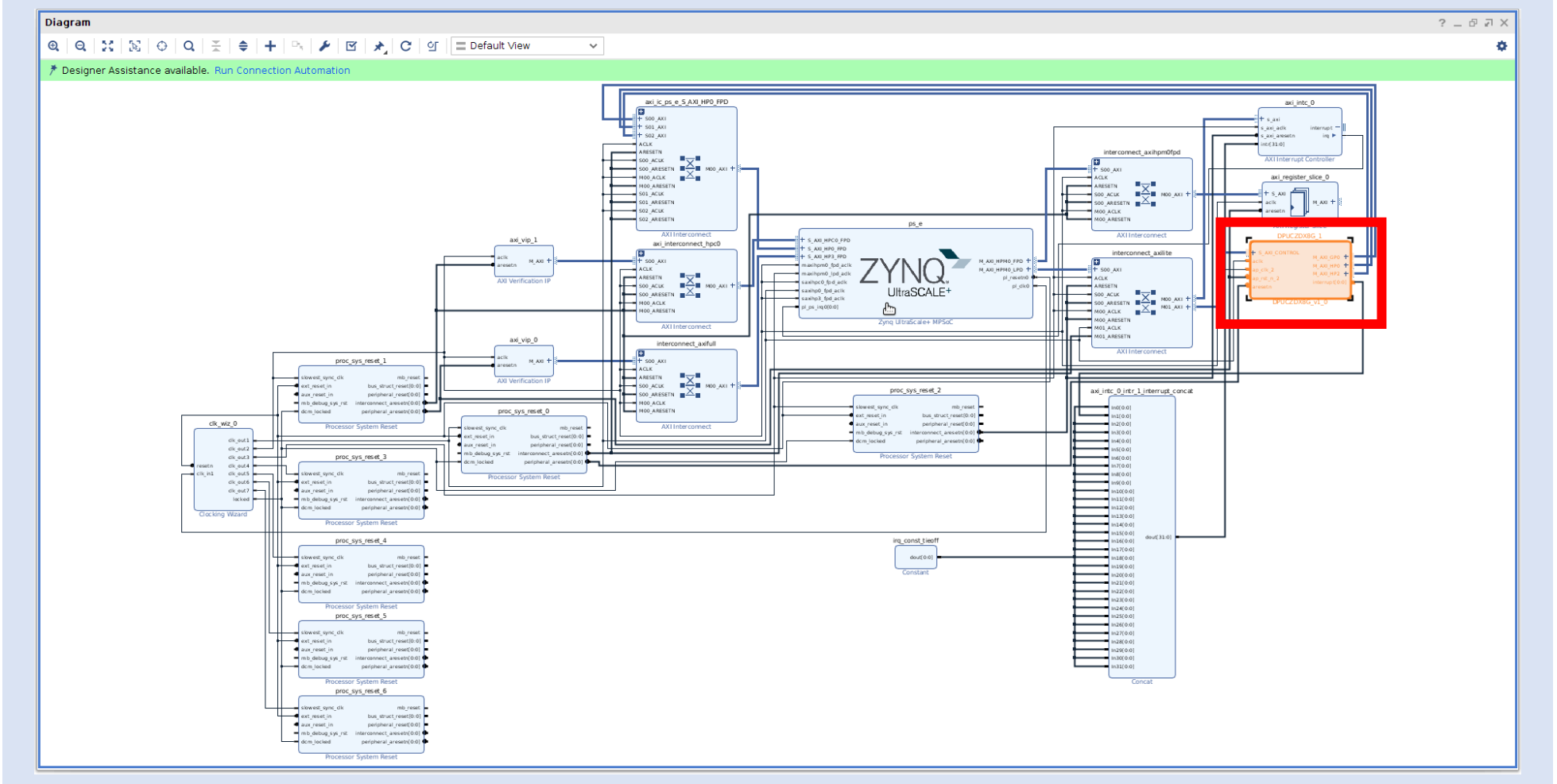
Lab 2: Instantiating Xilinx Deep Learning Processor

Step 20 – Once built, from the Xilinx Menu select Vivado integration and open Vivado project



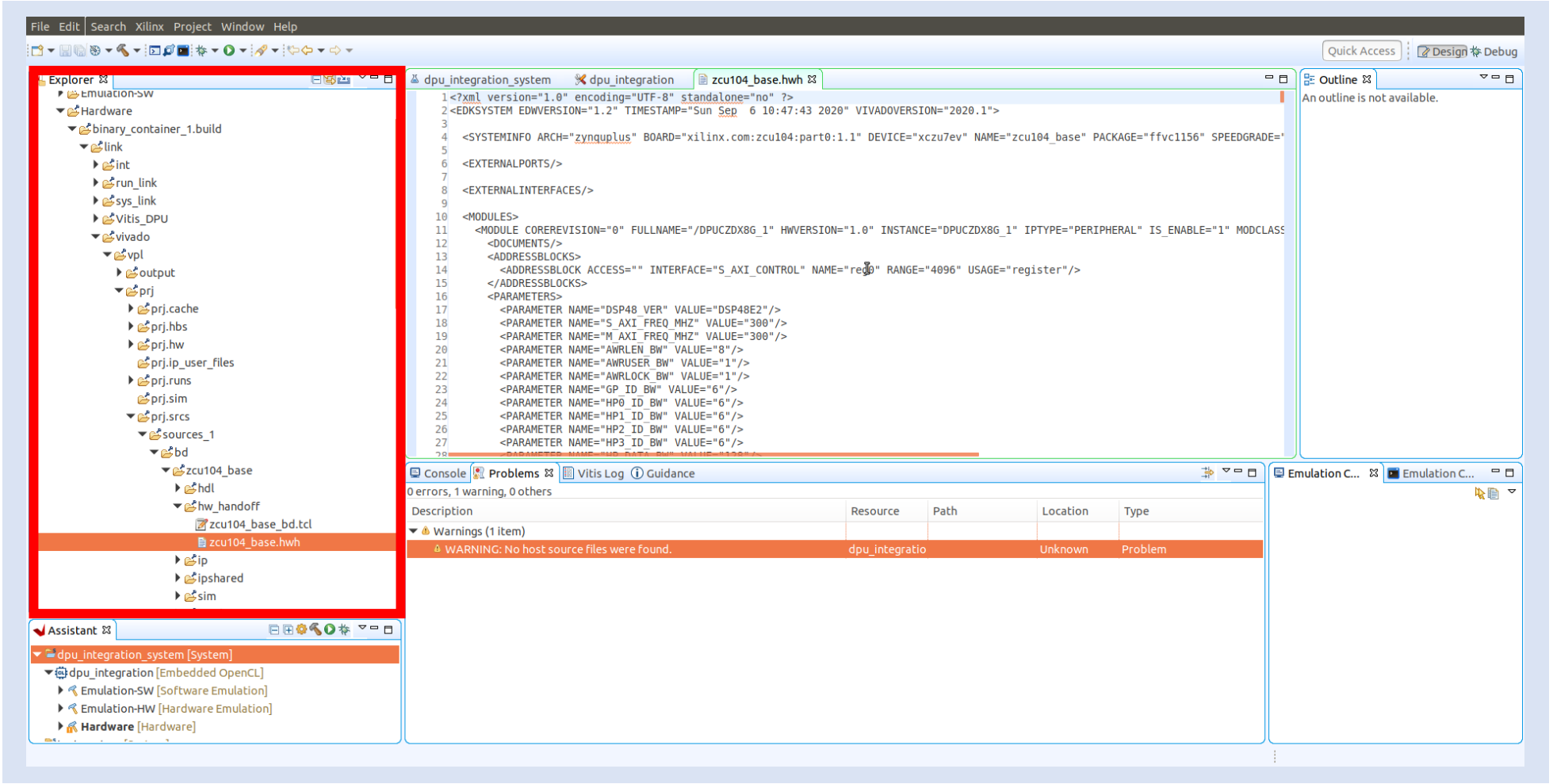
Lab 2: Instantiating Xilinx Deep Learning Processor

Step 21 – Once Vivado opens, open the block diagram and you will see the DPU



Lab 2: Instantiating Xilinx Deep Learning Processor

Step 21 – With the DPU platform created, the HWH file is now created for use with the remaining Vitis-AI stack



Lab 2 Summary

The concludes lab 2, throughout this lab we have demonstrated

1. How to clone the Vitis-AI tools
2. Modify the DPU configuration
3. Create the DPU.Xo for use in Vitis bottom up flow
4. Import the DPU into Vitis
5. Build the Xclbin and HWH file – required for higher levels Vitis-AI stack
6. Explore the implemented design in Vivado

Further Support

The following resources may be of interest

1. https://www.adiuvoengineering.com/blog - Adiuvo engineering blog updated weekly
2. https://www.adiuvoengineering.com/microzed-chronicles-archive - Complete Adiuvo engineering blog backlog
3. https://github.com/Xilinx/Vitis-AI-Tutorials - Vitis AI Tutorials
4. https://www.hackster.io/AlbertaBeef/projects - Good source of Vitis AI Ultra96 projects