SDAccel Environment Tutorial

Introduction

UG1021 (v2017.4) January 26, 2017

XILINX
ALL PROGRAMMABLE.

Send Feedback

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
01/26/2018	2017.4	Edited text for clarity.
12/20/2017	2017.4	• Edited text for clarity. • Updated Lab 2 to include more flows. • Updated screenshots to reflect changes for this release.
08/16/2017	2017.2	• Edited text for clarity. • Updated Lab 2 to include more flows. • Updated screenshots to reflect changes for this release.
06/20/2017	2017.1	Added new Lab 2 to tutorial.

Table of Contents

Chapter 1

Introduction

The Xilinx SDAccel™ Development Environment is part of the SDx™ Development Toolchain. This toolchain allows you to create FPGA accelerated designs using C/C++, OpenCL C, or RTL programming languages. You can create these designs in the SDx GUI environment or through a Makefile flow.

This tutorial walks you through the steps of building a basic OpenCL based design using the SDx GUI and learning some of the features that enable you to do performance profiling, and optimization.

Tutorial Design Description

This tutorial is based on the Xilinx SDAccel™ Github Examples, specifically the Vector Addition example. More information related to the Github examples can be found in *SDAccel Environment Profiling and Optimization Guide* (UG1207).

Hardware and Software Requirements

Refer to the *Vivado Design Suite User Guide: Release Notes, Installation, and Licensing*, (UG973) for a complete list and description of the system and software requirements for the Vivado® Design Suite.

Related information

Flow Overview

Chapter 2

Flow Overview

Lab 1: Introduction to the SDAccel Development Environment

This lab uses an example from the Xilinx SDAccel™ Example Github repository. To learn more about it refer to *SDAccel Environment User Guide* (UG1023) Chapter 5 "Getting Started with Examples."

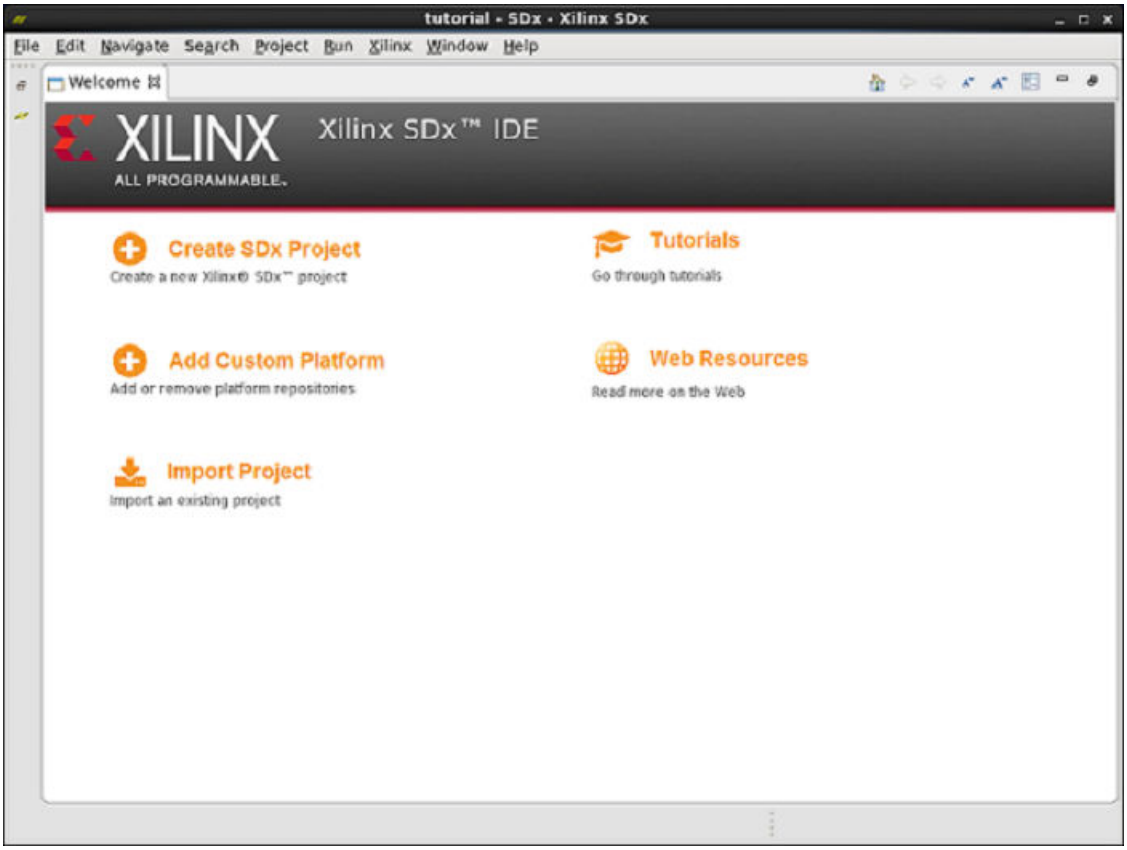
Step 1: Creating SDAccel Project From Github Example

1. Use the `sdx` command to launch SDx™ in a Terminal window and you will see the **Workspace Launcher** window appear. Select a location for your workspace, this is where the project will reside.

Figure 1: **Workspace Launcher**

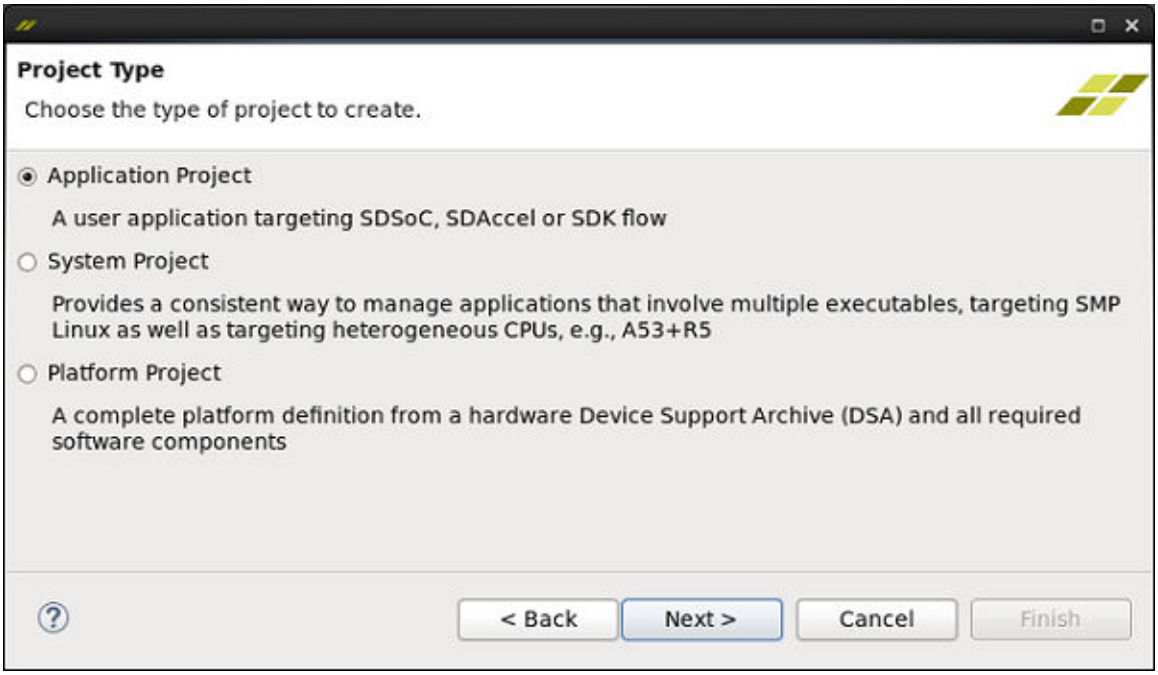
2. In the the **SDx Welcome** window, click **Create SDx Project**.

Figure 2: **SDx Welcome Window**



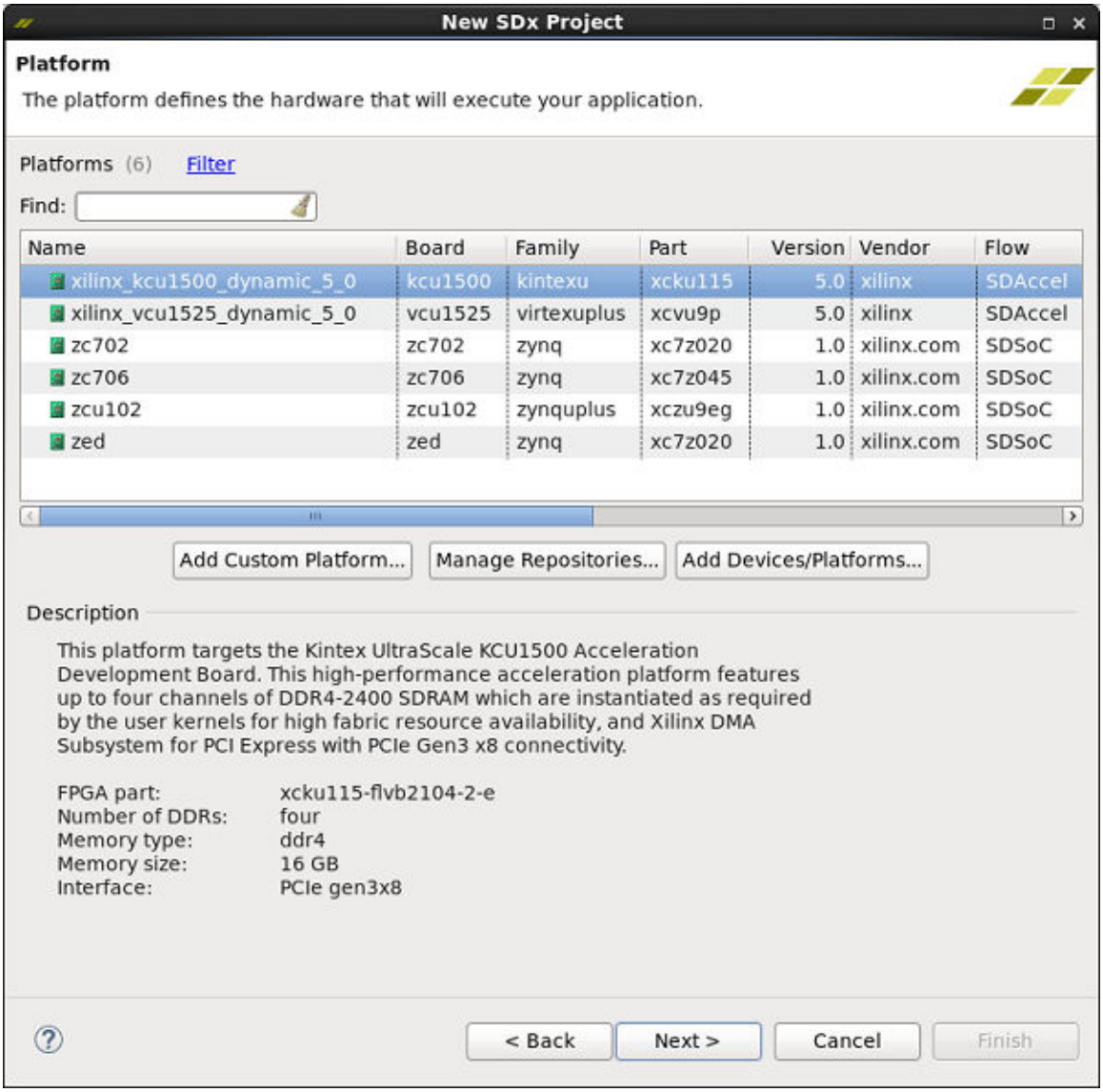
3. In the **Create a New SDx Project** window, in the **Project name** field, type **vadd**, and click **Next**.
4. In the **Project Type** dialog, select **Application Project** and click **Next**.

Figure 3: **Project Type**



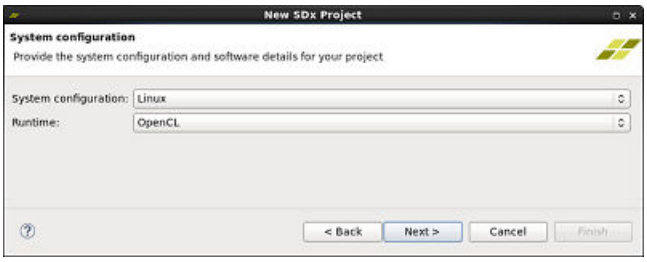
5. In the **Choose Hardware Platform** dialog choose the **xilinx_kcu1500_dynamic_5_0** platform and click **Next**.

Figure 4: **Hardware Platform**



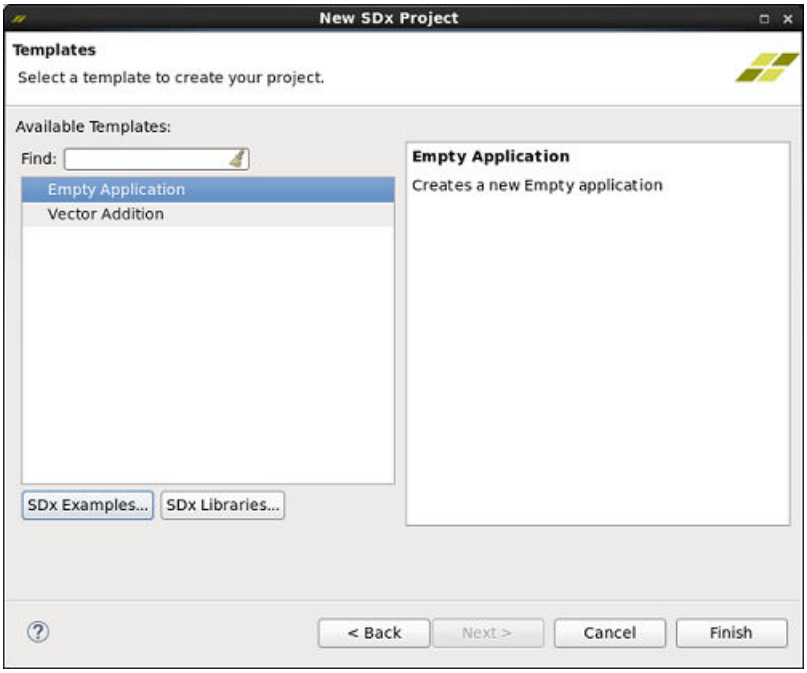
6. The **System configuration** window is where you define what type of system and runtime to use. For this lab, use the defaults of the **System configuration** which are set to **Linux** and **Runtime** of **OpenCL**. Click **Next**.

Figure 5: **System Configuration**



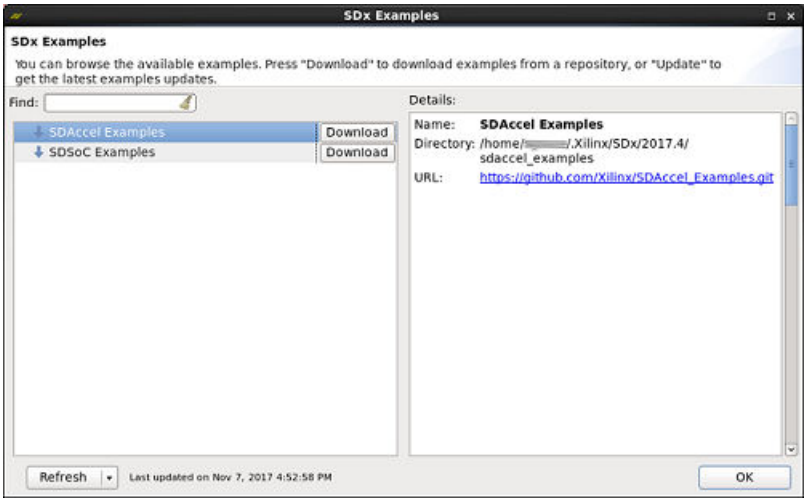
7. The **Templates** dialog has a list of possible templates that you can use to get started in building an SDAccel project. From a fresh start, you should only see **Empty Application** and **Vector Addition**. In this lab, you will be using the Vector Addition from the Github repository. To do this you need to download the examples. Click the **SDx Examples** button.

Figure 6: **Templates (Empty)**



8. The **SDx Examples** window shows that you can download both the SDAccel Examples and SDSoC Examples. Click the **Download** button for the SDAccel Examples and the system will begin to clone the Github repository to the location designated in the Details.

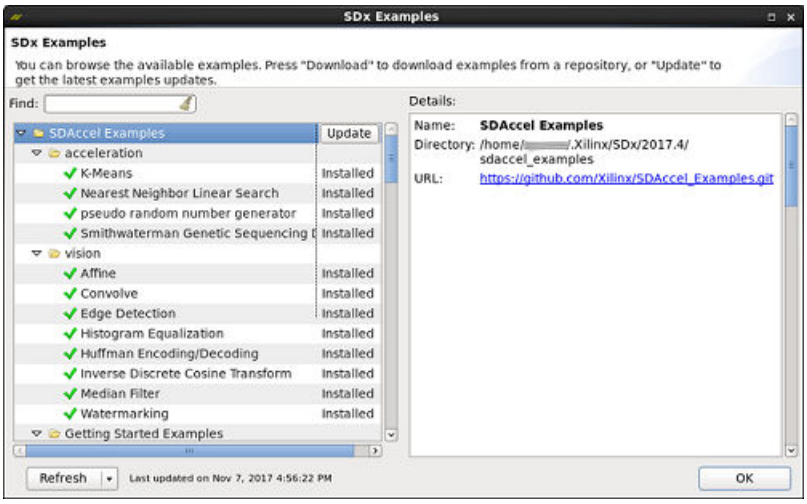
Figure 7: **SDx Examples (Not Populated)**



***Note*:** The download can take a while depending on connectivity speeds. The **Progress Information** dialog will be present until the cloning of the repository is complete.

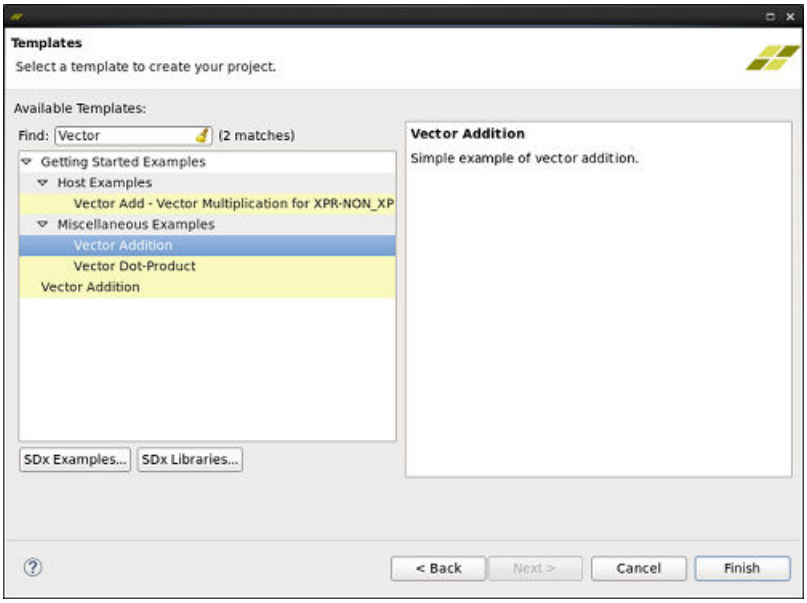
9. Once completed, you will see the **SDAccel Examples** tree table populated and expanded. Click **OK** to close the window and go back to the **Templates** window.

Figure 8: **SDx Examples (Populated)**



10. The **Templates** window is now populated with the SDAccel Github examples. Using the **Find** window, type `Vector`, and locate the **Vector Addition** from the **Miscellaneous Examples**. Click **Finish**.

Figure 9: **Templates (Populated and Searched)**

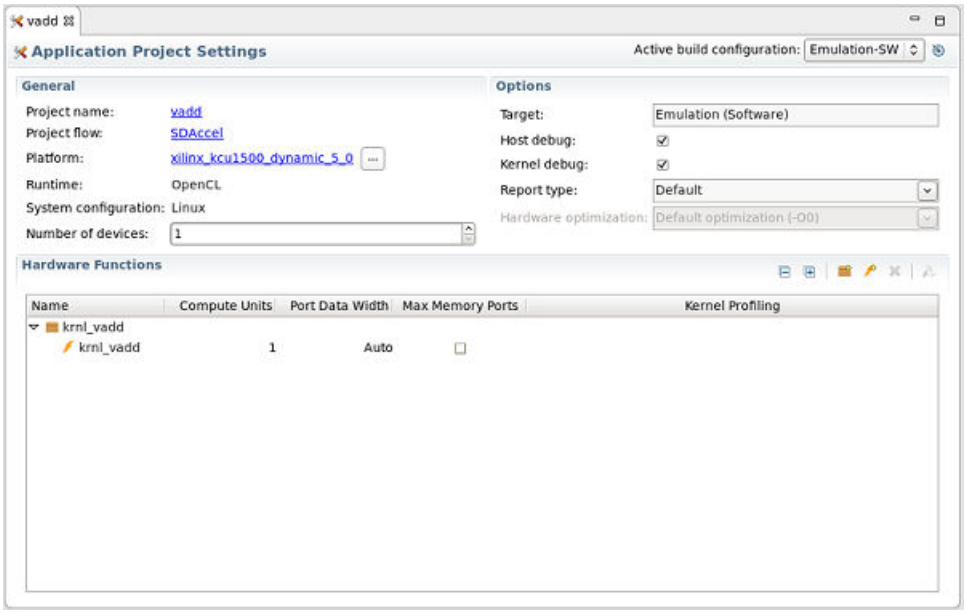


Step 2: Running Software Emulation

This step shows you how to run Software Emulation for a design, by setting Run Configuration settings, opening reports, and showing how to launch Debug. You can find details about reports and Debug in the *SDAccel Environment User Guide* (UG1023).

1. To run CPU Emulation, go to **Application Project Settings** and ensure that **Active build configuration** is set to **Emulation-SW**.

Figure 10: **SDx Project Settings**



2. From the Github example, an accelerator already exists for the design. To add a hardware function to a design that does not have one, start by clicking on the **Add Hardware Function** button: . This analyzes the C/C++ code and determines functions that can be used for acceleration.

3. Click the **Run** button: to run Software Emulation. This builds the project before running the emulation.

Note: In some cases, the design may need some command line inputs to function properly. To set these arguments, changes must be made in the **Run Configurations** dialog.

4. Go to the **Run** menu and select **Run Configurations**.

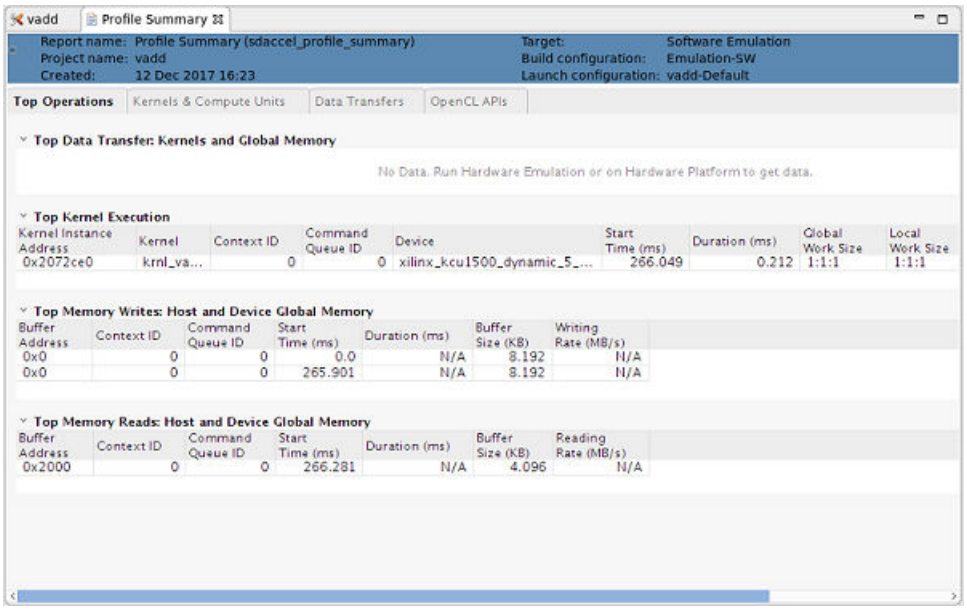
5. Under the **Arguments** tab, there is a section that allows you to add command line flags and switches. In this tutorial, no command line arguments are needed for the design to function.

6. Click the **Profile** tab, and notice that **Generate timeline report** and **Collect device data** have dropdown menus. Click each one and note what type of reports are generated, and make sure **Default** is set. Close the window without changing anything.

Note: If you make changes, you need to re-run the current emulation step in order to see the changes. You can do this by clicking the **Run** button.

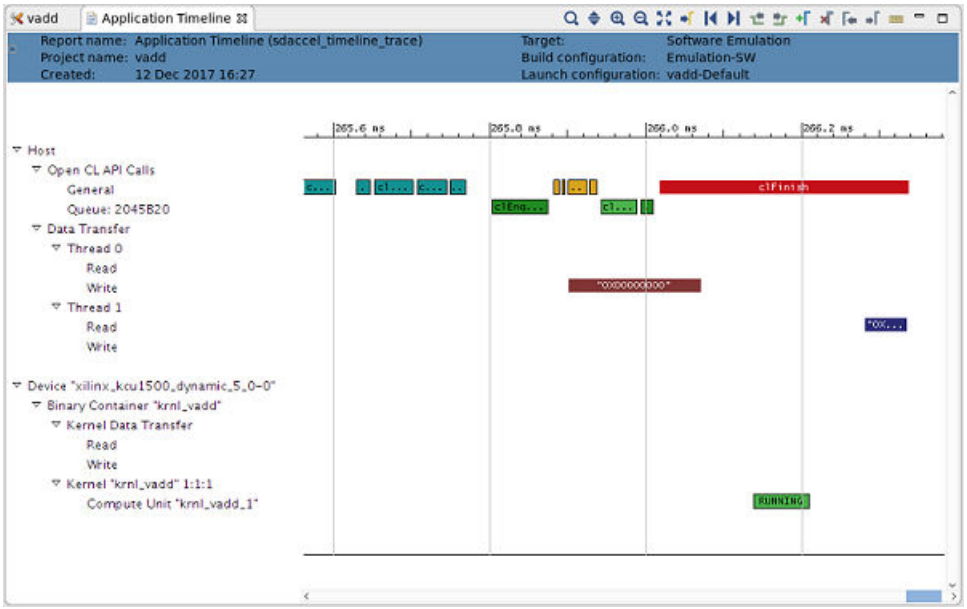
7. The **Console** window should now show **TEST PASSED**.
8. After the emulation run is complete, you can review the Profile Summary and Application Timeline reports for details on further optimizations. In the **Reports** window, double-click **Profile Summary**. Here, you can view operations, execution time, bandwidth, and other useful data that you can use to optimize the design. Note that your summary numbers may vary from the following figure.

Figure 11: **Emulation-SW Profile Summary**



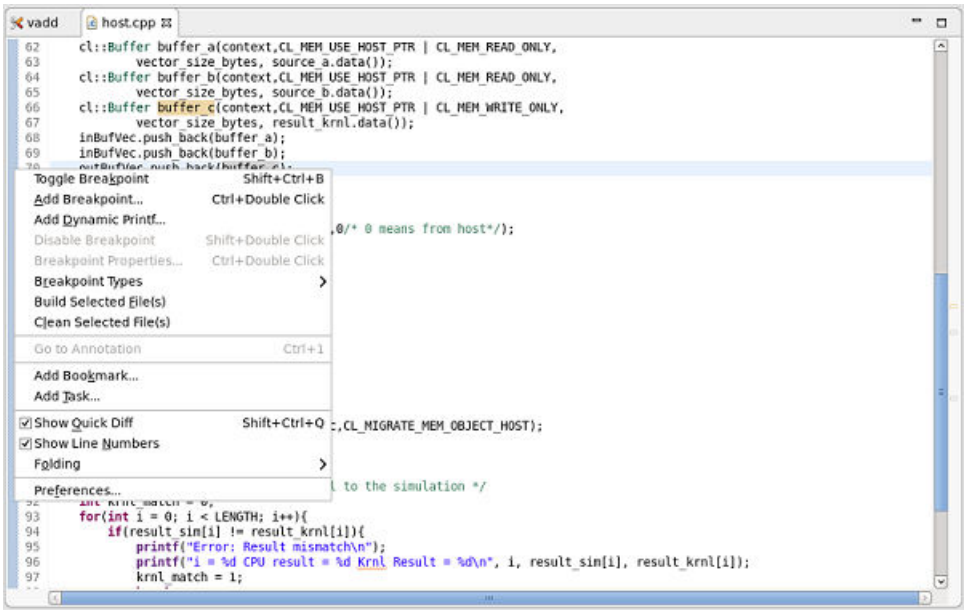
9. To view the **Application Timeline** report, in the **Reports** window, double-click **Application Timeline**. This shows a breakdown of the host code and the kernel code, and execution time for each. To zoom into a specific area, click and drag the mouse to the right.

Figure 12: **Application Timeline Report**



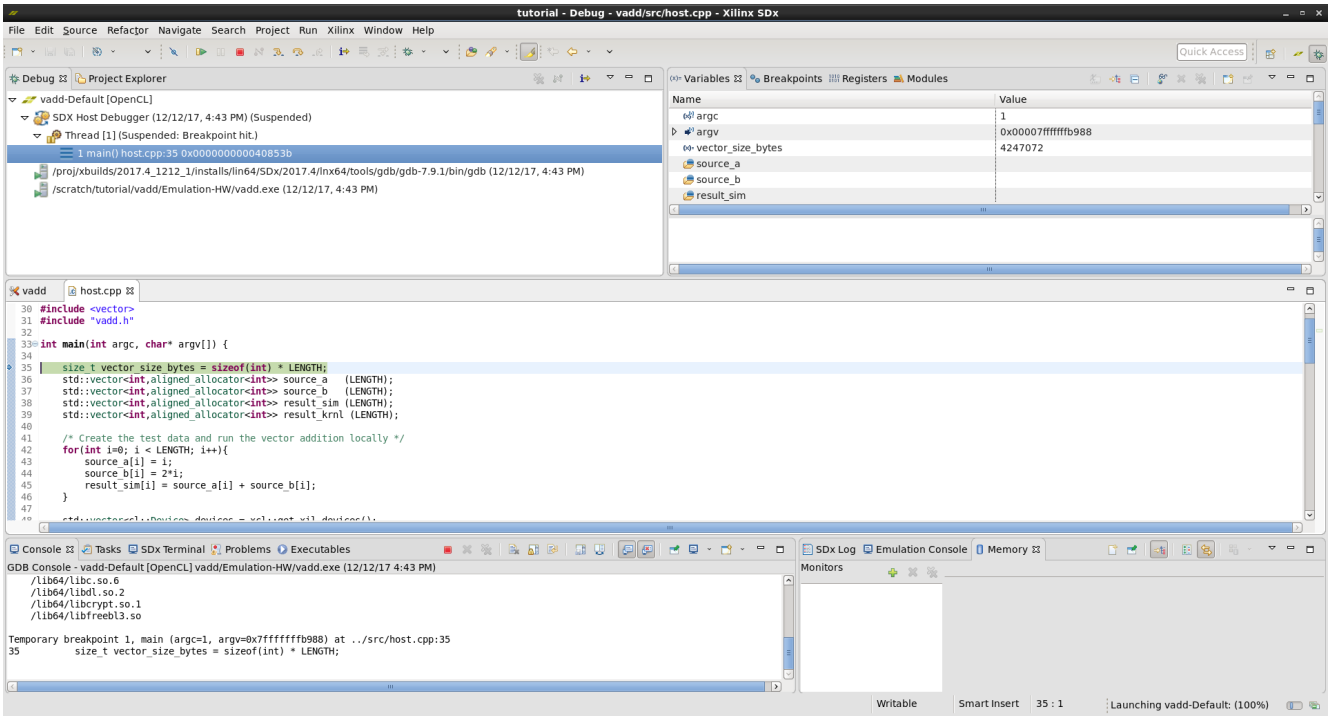
10. The **Profile Summary** and **Application Timeline** present data on how the host code and kernel communicate and process kernel information. Using the **Debug** feature can help you to step through host-kernel processing to pinpoint issues. In the **Project Explorer** window double-click the `host.cpp` (located in the Explorer > src directory) to open the file in the editor.
11. To run in Debug, you need to set a breakpoint. Setting breakpoints at key points in the execution helps identify problems. To pause the host code right before kernel debug begins, right-click on line 70 in the blue area (see figure below) on the `outBufVec.push_back(buffer_C)` and select **Toggle Breakpoint**.

Figure 13: **Setting Breakpoint**



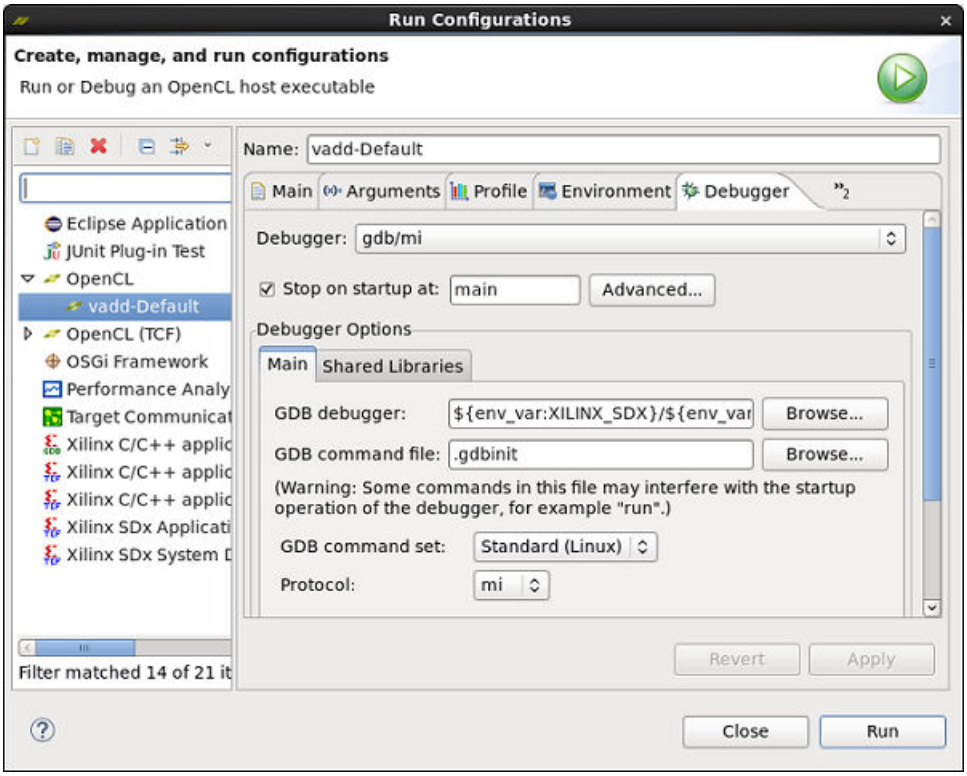
12. To run **Debug**, click the Debug icon: . A dialog box opens up asking you to switch to that perspective. Click **Yes**.
13. Using Eclipse debugging, you can examine the host and kernel code in more detail. All the controls with which to do step-by-step debugging are in the **Run** menu.

Figure 14: **Eclipse Debugging**



14. The debugger has an automatic breakpoint at the first line of `main`. In the **Runs Configuration** dialog, there is an option to stop on the `main` function (see the following figure). This is helpful in case of a problematic function in need of more thorough debugging. Press **F8** to resume to the next breakpoint or from the **Run** menu select **Resume**.

Figure 15: **Run Configuration**



15. After resuming debugging, SDx launches another **gdb** instance for the kernel code, and it also has a breakpoint at the beginning of the function. This allows for detailed analysis of the kernel and how the data looks being read into the function, and written out to memory. Once the kernel execution is done in **gdb**, that instance is terminated and you return to the main debugging thread. Press **F8** to continue.

***Note*:** The console view still shows the kernel debug outputs. Click the icon  to go back to the **vadd.exe** console and see the output from the host code.

16. Close the **Debug Perspective** by going to the upper-right of the window where it shows the Debug  button, right-click, and select **Close**.
17. Once back into the main **SDx Perspective**, close all tabs except the **SDx Project Settings** window.

Step 3: Running Hardware Emulation

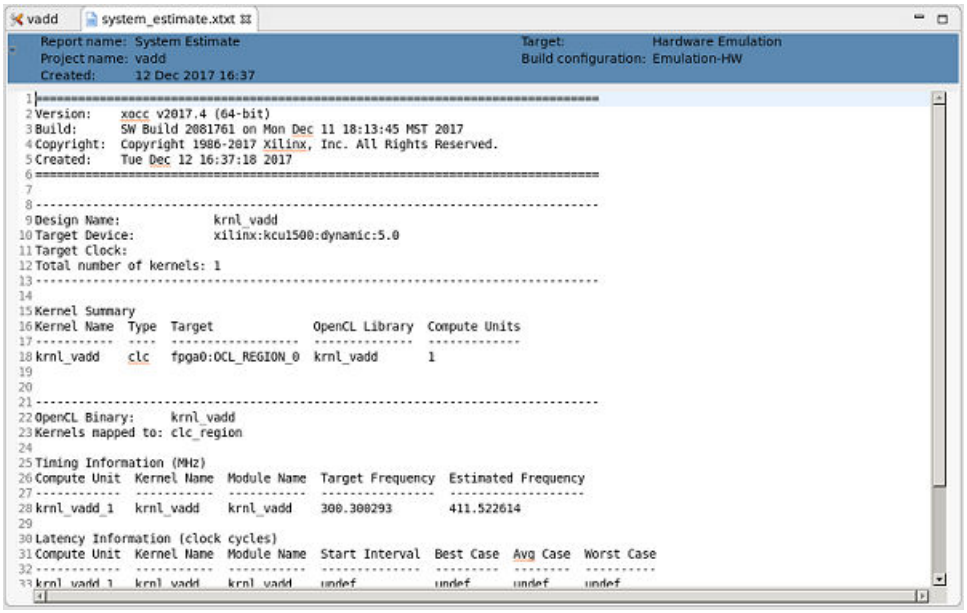
This step covers running Hardware Emulation feature as well as looking at the basics of profiling and reports.

1. To run Hardware Emulation, go to **SDx Application Settings** and make sure that **Active build configuration** is set to **Emulation-HW**, then click **Run**. This takes some time to complete.

***Note*:** The main difference between **Emulation-SW** and **Emulation-HW** is that emulating hardware builds a design that is closer to what is seen on the platform. This means data related to bandwidth, throughput, and execution time are more accurate. This causes the design to take longer to compile.

2. In the **Reports** tab, open **System Estimate**. This is a text report that provides information related to kernel information, timing about the design, clock cycles, and area used in the device.

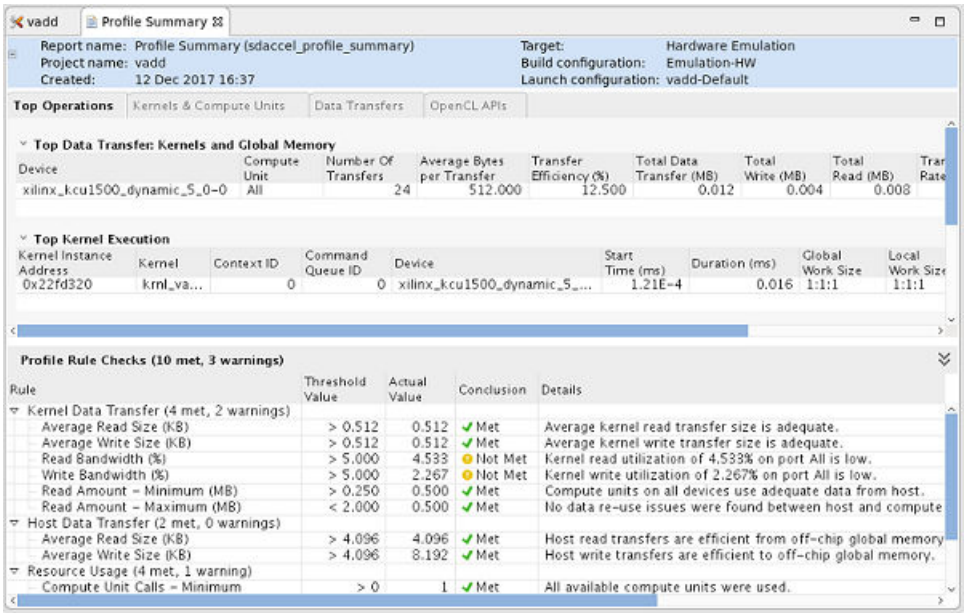
Figure 16: **System Estimate**



3. In the **Reports** tab, open **Profile Summary**. This summary report provides detailed information related to kernel operation, data transfers, and OpenCL™ API calls as well as profiling information related to the resource usage, and data transfer to/from the kernel/host. It also provides detailed guidance in how to meet the profile rule checks.

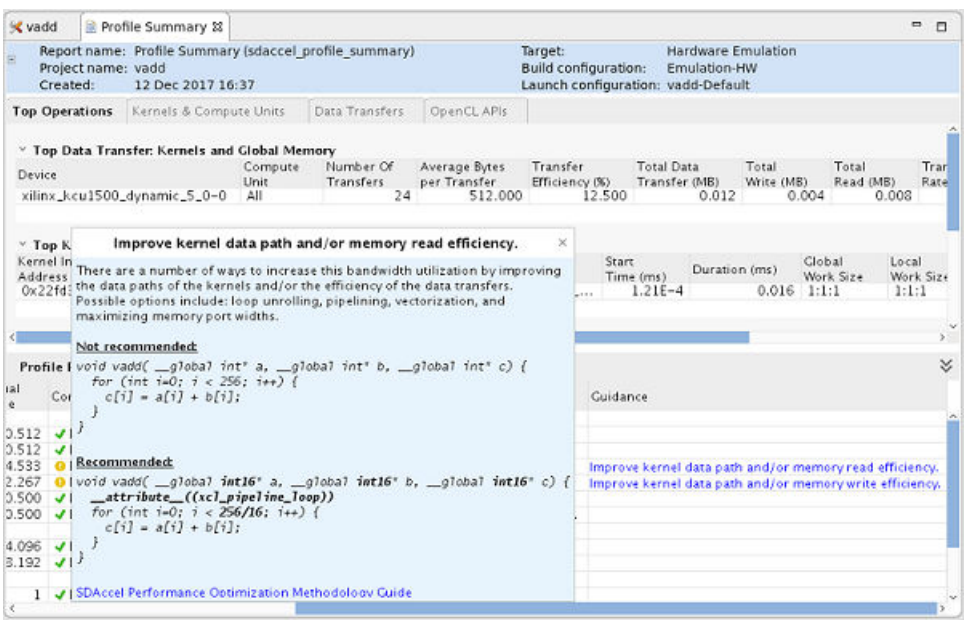
***Note*:** The simulation models used in HW Emulation are approximate. Profile numbers shown are just an estimate and may vary from results obtained in real HW.

Figure 17: **Profile Summary Report**



4. Scroll to the right in the **Profile Rule Checks** and look for the header column labeled **Guidance**. This is where unmet checks provide some information on how to optimize the kernel.

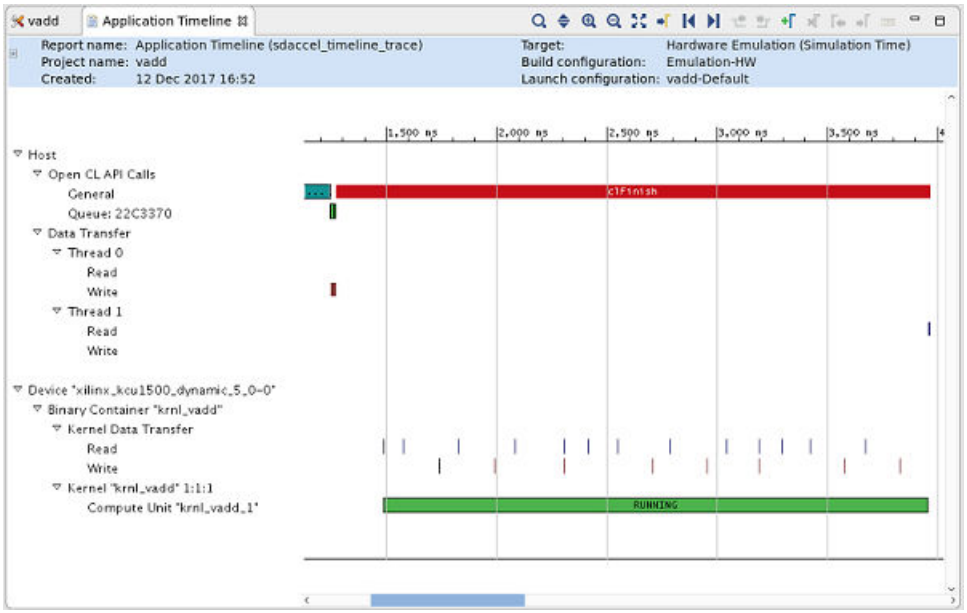
Figure 18: **Profile Rule Checks**



Note: To see other performance optimization techniques and methodologies, Refer to the *SDAccel Environment Profiling and Optimization Guide* (UG1207).

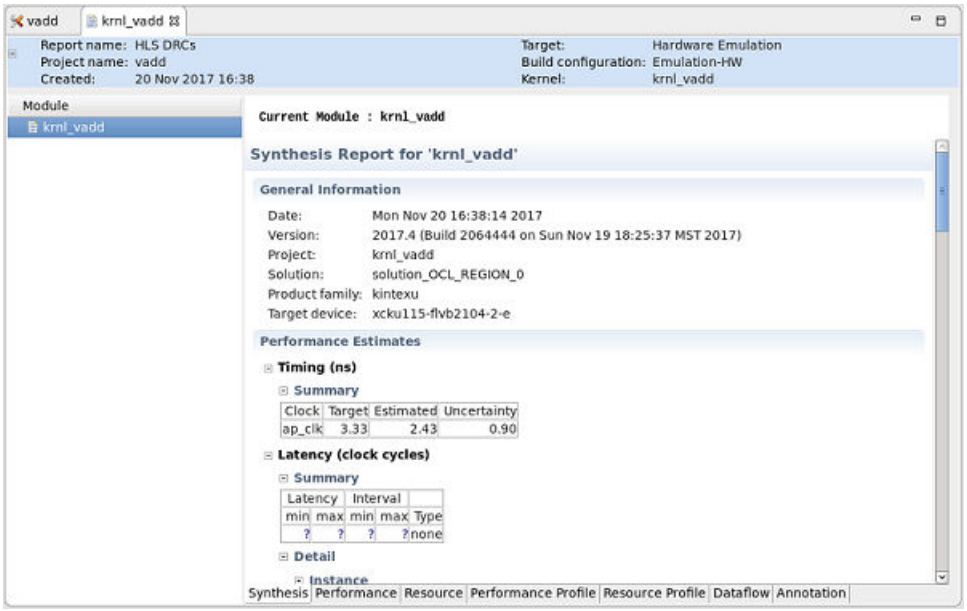
5. Open the **Application Timeline** report. This report shows the estimated time it takes for the host and kernel to complete the task and provides finer grained information on where bottlenecks can be. In this example, it is iterated twice and this timeline shows the kernel is run twice. Adding a marker, zooming, and expanding signals can help in identifying bottlenecks.

Figure 19: **Application Timeline Report**



6. Open the **HLS Report**. This report provides detailed information provided by Vivado® HLS on the kernel transformation and synthesis. The tabs at the bottom provide more information on where most of the time is spent in the kernel and other performance related data. Some performance data may include latency and clock period.

Figure 20: **HLS Report**



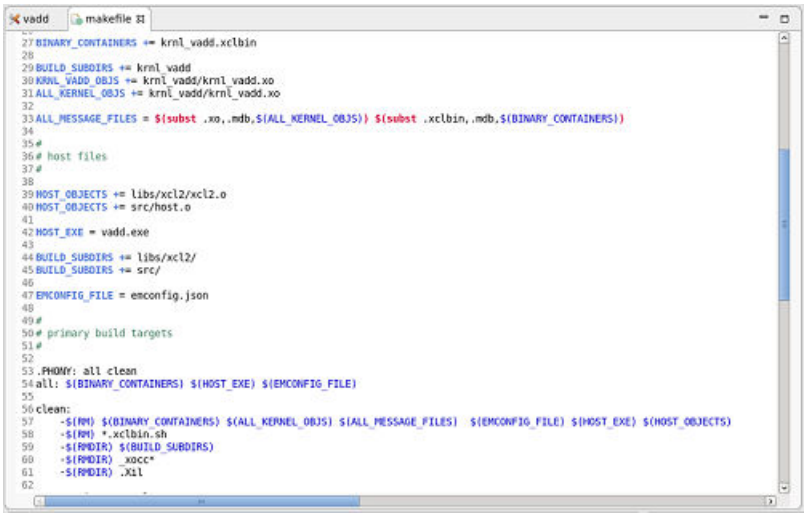
Step 4: Makefile Flow

This step explains the basics of the Makefile flow and how SDx™ uses it. The advantages of using this flow include the following:

- Easy automation into any system.
- Faster turnaround time on small design changes.

1. In the **Project Explorer**, navigate to the **Emulation-SW** directory and look for the **makefile** file. Double-click the file to open it in the editor. This is the makefile that SDx creates and uses for building and running emulations. Alternatively, navigate to the **Emulation-HW** directory and look for the **makefile** file. Open the file.
2. Notice that in there is a unique **makefile** for each build. In the opened **makefile** in the editor window, look at line 21. Notice that it contains either `hw_emu` or `sw_emu`.

Figure 21: **Makefile Editor window**



3. The **makefile** can also access SDx without using the GUI.

   Open up a new terminal session and navigate to the workspace. Then navigate to the **Emulation-SW** directory and type: `make incremental`. The process produces a typical SDx log output.

***Note*:** If no changes are made to the host or kernel code, this re-runs the entire software emulation flow.

Lab 2 of this tutorial goes into more detail on how to use the makefile and command line flow.

Summary

After completing this tutorial, you should be able to do the following:

- Create an SDAccel™ project from a Github example design.
- Create a binary container and accelerator for the design.
- Run Software Emulation and use the Debug environment on host and kernel code.
- Run Hardware Emulation and use the reports to understand possible optimization.
- Understand differences between Software and Hardware Emulation reports.
- Read the project makefile and run the makefile command line.

Lab 2: Introduction to the SDAccel Makefile

The following lab uses an example from the Xilinx SDAccel™ Example Github repository. To learn more about it refer to *SDAccel Environment User Guide* (UG1023) Chapter 5 "Getting Started with Examples."

Step 1: Preparing and Setting up the SDAccel Environment

In this step, you will setup SDx™ to run in command line, and clone the Github repository for SDAccel™.

1. Launch a terminal and source the settings scripts found in the SDx environment using the command:

   ```
   source <SDx_install_location>/<version>/settings64.csh
   ```

   or

   ```
   source <SDx_install_location>/<version>/settings64.sh
   ```

   This allows you to run the SDx command lines without the need to use the GUI.

2. Clone the design from the Github repository to a workspace of your choice by using the command:

   ```
   git clone https://github.com/Xilinx/SDAccel_Examples <workspace>/
   examples
   ```

***Note*:** This Github repository totals to be around 400MB in size. Make sure you have sufficient space on a local or remote disk to ensure that it can be completely downloaded.

3. Once the download is complete navigate to the `vadd` directory by using the following command:

   ```
   cd <workspace>/examples/getting_started/misc/vadd
   ```

   Once in this directory, run the `ls` command and view the files. You should get something like this in return:

   ```
   [sdaccel@localhost vadd ]$ ls
   Makefile    README.md    description.json src
   ```

   If you run the `ls` on the `src` directory, you should get the following:

   ```
   [sdaccel@localhost vadd ]$ ls src
   host.cpp    krnl_vadd.cl    vadd.h
   ```

Step 2: Initial Design and Makefile Exploration

1. The `vadd` directory contains the `Makefile` file, which you will use to compile the design in both Hardware and Software Emulation, as well as generate a System Run.
2. Open the Makefile in a text editor. View the content and become familiar with how it is written. Makefiles are written in a bash style syntax.

***Note*:** The file itself makes references to generic makefiles that are used by all the Github example designs.

3. The first few lines contain `include` statements for other generic makefiles that are used by all the examples.

```
COMMON_REPO:=../../../
include $(COMMON_REPO)/utility/boards.mk
include $(COMMON_REPO)/libs/xcl2/xcl2.mk
include $(COMMON_REPO)/libs/opencl/opencl.mk
```

4. Open the `../../../utility/boards.mk`. This makefile contains the flags and command line compiler info needed to build the host and source code.

```
# By Default report is set to none, so report will be generated
# 'estimate' for estimate report generation
# 'system' for system report generation
REPORT:=none

# Default C++ Compiler Flags and xocc compiler flags
CXXFLAGS:=-Wall -O0 -g
CLFLAGS:= --xp "param:compiler.preserveHlsOutput=1" --xp
"param:compiler.generateExtraRunData=true" -s

ifneq ($(REPORT),none)
CLFLAGS += --report $(REPORT)
endif
```

`REPORT` is an input flag (parameter) for the `make` command in the terminal. Notice that the `CLFLAGS` is building a long list of `xocc` command line flags to be used.

5. Scroll down to line 45 and you will see:

```
# By default build for hardware can be set to
#   hw_emu for hardware emulation
#   sw_emu for software emulation
#   or a collection of all or none of these
TARGETS:=hw

# By default only have one device in the system
NUM_DEVICES:=1
```

Here, `TARGETS` defines what default build to have (if not specified in the makefile command line), by default it is set to `hw` (System build). You will be setting this value as desired when working on your own design. Lastly, you can define the number of devices the machine uses that contain the board you selected. Generally, one device is fine to start, but you can change this if your design requires more.

6. Close the `boards.mk` file and refocus on the `Makefile`. Looking at line 9 and beyond, notice that this file handles the majority of where the source code is located, and naming the kernel and application executables.
7. Finally, open the `../../../utility/rules.mk` file. This file is where all the setup items from the previous makefiles are handled into creating the `xocc` and the `xcpp` (`gcc`) command line arguments. Explore this file until you feel comfortable with what it does. Key areas to focus are labeled with `define make_exe` (line 34) and `define make_xclbin` (line 107).

Step 3: Running Software Emulation

Now that you understand parts of the makefile construction, it is time to compile the code to run Software Emulation.

1. To compile the application for Software Emulation, run the following commands:

```
make all REPORT=estimate TARGETS=sw_emu
DEVICES=xilinx_kcu1500_dynamic_5_0
```

   The three files that should be generated are `vadd` (host executable), `xclbin/krnl_vadd.sw_emu.xilinx_kcu1500_dynamic.xclbin` (binary container), and a system estimate report. To double check, run an `ls` command in the directory and you should get the follow:

```
[sdaccel@localhost vadd]$ ls
Makefile
README.md
_xocc_compile_krnl_vadd_krnl_vadd.sw_emu.xilinx_kcu1500_dynamic.dir
_xocc_link_krnl_vadd.sw_emu.xilinx_kcu1500_dynamic_krnl_vadd.sw_emu.xili
nx_kcu1500_dynamic.dir
description.json
src
system_estimate.xtxt
vadd
xclbin
[sdaccel@localhost vadd ]$ ls xclbin/
krnl_vadd.sw_emu.xilinx_kcu1500_dynamic.xclbin
krnl_vadd.sw_emu.xilinx_kcu1500_dynamic.xo
```

2. To run the application in emulation, run the following command:

```
emconfigutil --platform xilinx_kcu1500_dynamic_5_0 --nd 1
```

   The `emconfigutil` tool generates a `emconfig.json` file which contains the information about the target device. However, from the Github repository, the makefile is how you will run it. Run this command:

```
make check TARGETS=sw_emu DEVICES=xilinx_kcu1500_dynamic_5_0
```

Note: Make sure that the `DEVICES` used in the original make is the same as what was used for compilation

   In this flow, this will run the previous command, and also run the application.

3. If the application runs successfully, the following messages appear in the terminal:

```
[sdaccel@localhost vadd]$ make check TARGETS=sw_emu
DEVICES=xilinx_kcu1500_dynamic_5_0
<install location>/SDx/2017.4/bin/emconfigutil --platform
xilinx_kcu1500_dynamic_5_0 --nd 1

****** configutil v2017.4 (64-bit)
  **** SW Build 2064444 on Sun Nov 19 18:07:27 MST 2017
    ** Copyright 1986-2017 Xilinx, Inc. All Rights Reserved.

INFO: [ConfigUtil 60-895]    Target platform: <install location>/SDx/
2017.4/platforms/xilinx_kcu1500_dynamic_5_0/
xilinx_kcu1500_dynamic_5_0.xpfm
emulation configuration file `emconfig.json` is created in current
working directory

...
platform Name: Intel(R) OpenCL
Vendor Name : Xilinx
platform Name: Xilinx
Vendor Name : Xilinx
Found Platform
XCLBIN File Name: krnl_vadd
INFO: Importing xclbin/krnl_vadd.sw_emu.xilinx_kcu1500_dynamic.xclbin
Loading: 'xclbin/krnl_vadd.sw_emu.xilinx_kcu1500_dynamic.xclbin'
Result Match: i = 0 CPU result = 0 Krnl Result = 0
Result Match: i = 1 CPU result = 3 Krnl Result = 3
Result Match: i = 2 CPU result = 6 Krnl Result = 6
Result Match: i = 3 CPU result = 9 Krnl Result = 9
Result Match: i = 4 CPU result = 12 Krnl Result = 12
Result Match: i = 5 CPU result = 15 Krnl Result = 15
...
Result Match: i = 1018 CPU result = 3054 Krnl Result = 3054
Result Match: i = 1019 CPU result = 3057 Krnl Result = 3057
Result Match: i = 1020 CPU result = 3060 Krnl Result = 3060
Result Match: i = 1021 CPU result = 3063 Krnl Result = 3063
Result Match: i = 1022 CPU result = 3066 Krnl Result = 3066
Result Match: i = 1023 CPU result = 3069 Krnl Result = 3069
TEST PASSED
```

4. The application generates a profiling summary report called `sdaccel_profile_summary` in the format of CSV.

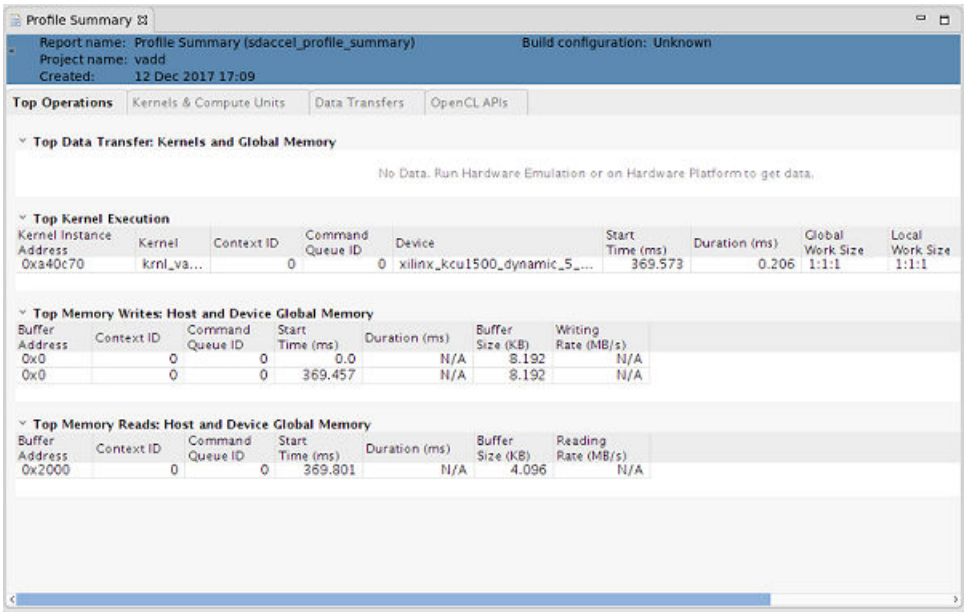
   You can convert this into a report shown in the Lab 1 profile summary and explore it in the SDx™ IDE. To do this, run the following command:

```
[sdaccel@localhost vadd]$ sdx_analyze profile
sdaccel_profile_summary.csv
```

   This generates an `sdaccel_profile_summary.xprf` file. To view this report, open the SDx IDE, select **File** > **Open File** and click the file from the menu. The report is shown below.

Note: For viewing these reports, you do not need to use the workspace you previously used in Lab 1. You can use this command to create a workspace locally for viewing these reports: `sdx -workspace ./lab2`. You may also need to close the **Welcome Window** to view the report.

Figure 22: **SDx Profile Summary in SDx GUI**



***Note*:** Software Emulation does not provide all the profiling information (data transfer between kernel and global memory). This information will be available in Hardware Emulation and System.

5. The System Estimate report (`system_estimate.xtxt`) is also generated. This is from the `--report` switch used when compiling using the `xocc` command.

Figure 23: **SDx Makefile System Estimate**

```
================================================================================
Version:    xocc v2017.4 (64-bit)
Build:      SW Build 2081761 on Mon Dec 11 18:13:45 MST 2017
Copyright:  Copyright 1986-2017 Xilinx, Inc. All Rights Reserved.
Created:    Tue Dec 12 17:07:43 2017
================================================================================

--------------------------------------------------------------------------------
Design Name:             krnl_vadd.sw_emu.xilinx_kcu1500_dynamic
Target Device:           xilinx:kcu1500:dynamic:5.0
Target Clock:
Total number of kernels: 1
--------------------------------------------------------------------------------

Kernel Summary
Kernel Name  Type  Target              OpenCL Library                          Compute Units
-----------  ----  ------------------  --------------------------------------  -------------
krnl_vadd    clc   fpga0:OCL_REGION_0  krnl_vadd.sw_emu.xilinx_kcu1500_dynamic  1


--------------------------------------------------------------------------------
OpenCL Binary:     krnl_vadd.sw_emu.xilinx_kcu1500_dynamic
Kernels mapped to: clc_region

Timing Information (MHz)
Compute Unit  Kernel Name  Module Name  Target Frequency  Estimated Frequency
------------  -----------  -----------  ----------------  -------------------
krnl_vadd_1   krnl_vadd    krnl_vadd    300.300293        411.522614

Latency Information (clock cycles)
Compute Unit  Kernel Name  Module Name  Start Interval  Best Case  Avg Case  Worst Case
------------  -----------  -----------  --------------  ---------  --------  ----------
krnl_vadd_1   krnl_vadd    krnl_vadd    undef           undef      undef     undef

Area Information
Compute Unit  Kernel Name  Module Name  FF    LUT   DSP  BRAM
------------  -----------  -----------  ----  ----  ---  ----
krnl_vadd_1   krnl_vadd    krnl_vadd    4774  5907  0    32
--------------------------------------------------------------------------------
```

6. If you want to generate additional reports you will need to set either environment variables or create a file called `sdaccel.ini` with appropriate information and permissions.

   In this tutorial, you want to create the `sdaccel.ini` file in the `vadd` directory, and add the following contents:

```
[Debug]
timeline_trace = true
device_profile = true
```

   Run the same command as before:

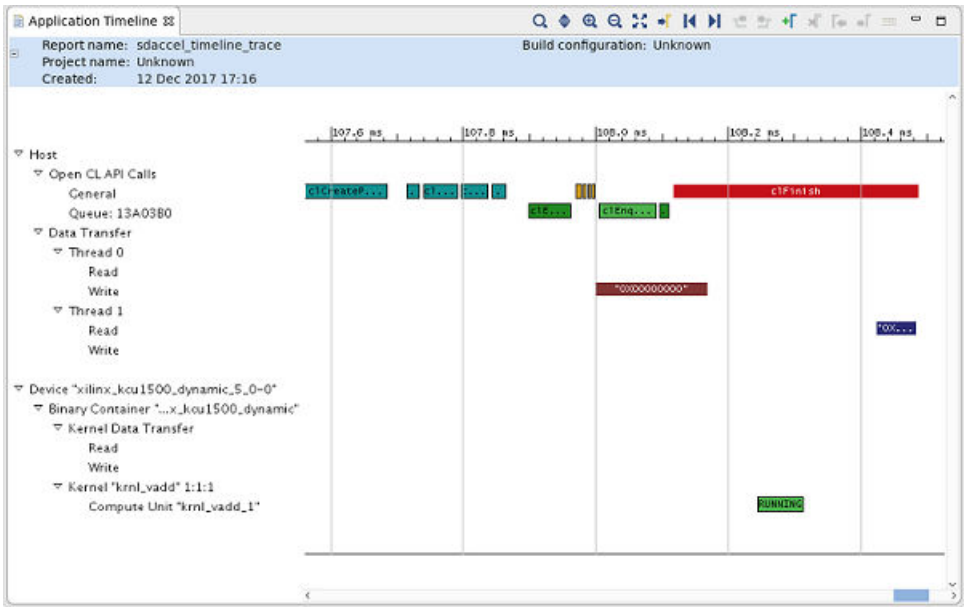
```
make check TARGETS=sw_emu DEVICES=xilinx_kcu1500_dynamic_5_0
```

After the application completes, there is an additional timeline trace file called `sdaccel_timeline_trace.csv`. To view this trace report in the GUI, you need to convert the CSV file into a WDB file using this command:

```
sdx_analyze trace sdaccel_timeline_trace.csv
```

As you did earlier, launch SDx IDE and using the **File** > **Open File**, locate the file: `sdaccel_timeline_trace.wdb`. You should get the report shown in the following figure:

Figure 24: **SDx Makefile Application Timeline GUI**



Step 4: Running Hardware Emulation

1. Now that Software Emulation is complete, you can run Hardware Emulation. To do this without changing the makefile, run the following command:

   ```
   make all REPORT=estimate TARGETS=hw_emu DEVICES=xilinx:kcu1500:dynamic
   ```

   By defining the `TARGETS` this way, it passes the value and overwrites the default that was set in the makefile that we saw in Step 2.

Note: Like in Lab 1, Hardware Emulation takes longer to compile than the Software Emulation.

2. Next, you can re-run the application. You do not need to regenerate `emconfig.json` since the device information has not changed; however, the emulation needs to be set for Hardware Emulation. Re-run the application with the following command:

   ```
   make check TARGETS=hw_emu DEVICES=xilinx:kcu1500:dynamic
   ```

Note: The makefile sets the enviornment variable to `hw_emu`.

3. The output should be similar to the Software Emulation with the following output.

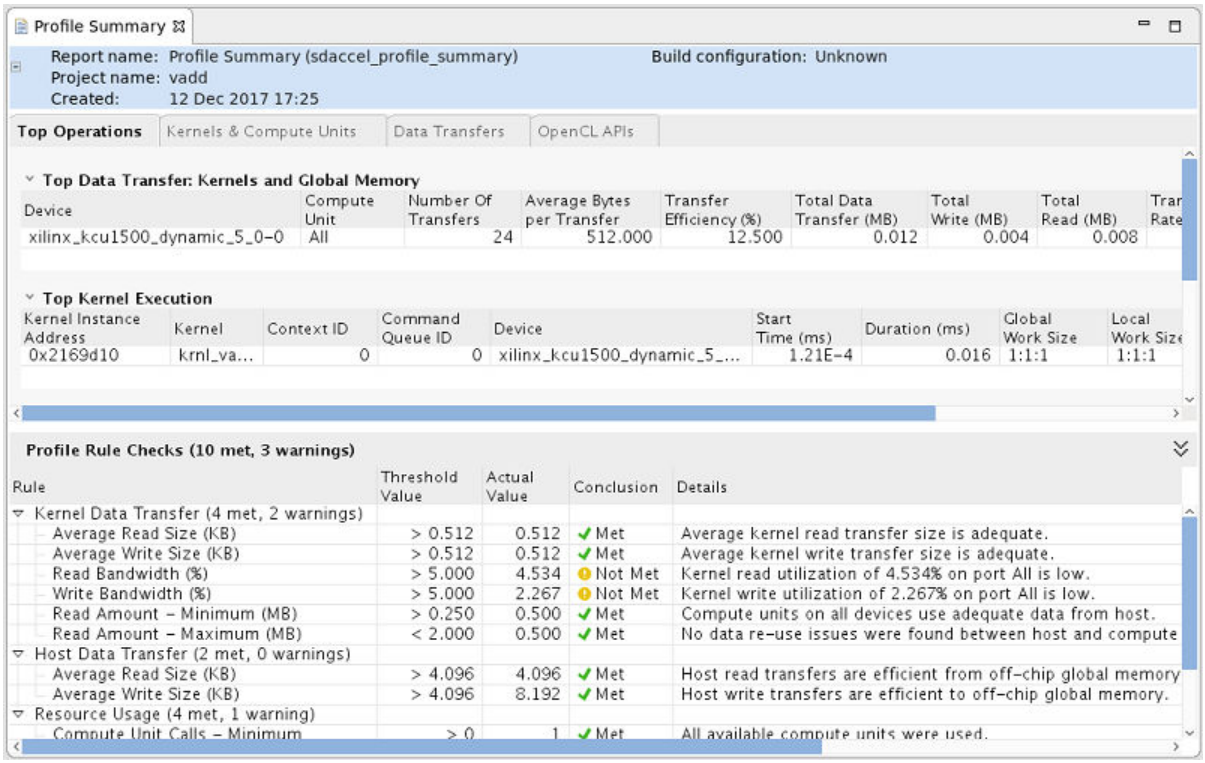
```
[sdaccel@localhost vadd]$ make check TARGETS=hw_emu
DEVICES=xilinx_kcu1500_dynamic_5_0
/<install location>/SDx/2017.4/bin/emconfigutil --platform
xilinx_kcu1500_dynamic_5_0 --nd 1
...
INFO: [ConfigUtil 60-895]    Target platform: <install location>/SDx/
2017.4/platforms/xilinx_kcu1500_dynamic_5_0/
xilinx_kcu1500_dynamic_5_0.xpfm
emulation configuration file `emconfig.json` is created in current
working directory
...
platform Name: Intel(R) OpenCL
Vendor Name : Xilinx
platform Name: Xilinx
Vendor Name : Xilinx
Found Platform
XCLBIN File Name: krnl_vadd
INFO: Importing xclbin/krnl_vadd.sw_emu.xilinx_kcu1500_dynamic.xclbin
Loading: 'xclbin/krnl_vadd.sw_emu.xilinx_kcu1500_dynamic.xclbin'
Result Match: i = 0 CPU result = 0 Krnl Result = 0
Result Match: i = 1 CPU result = 3 Krnl Result = 3
Result Match: i = 2 CPU result = 6 Krnl Result = 6
Result Match: i = 3 CPU result = 9 Krnl Result = 9
Result Match: i = 4 CPU result = 12 Krnl Result = 12
Result Match: i = 5 CPU result = 15 Krnl Result = 15
...
Result Match: i = 1018 CPU result = 3054 Krnl Result = 3054
Result Match: i = 1019 CPU result = 3057 Krnl Result = 3057
Result Match: i = 1020 CPU result = 3060 Krnl Result = 3060
Result Match: i = 1021 CPU result = 3063 Krnl Result = 3063
Result Match: i = 1022 CPU result = 3066 Krnl Result = 3066
Result Match: i = 1023 CPU result = 3069 Krnl Result = 3069
INFO: [SDx-EM 22] [Wall clock time: 10:42, Emulation time: 0.010001 ms]
Data transfer between kernel(s) and global memory(s)
BANK0          RD = 8.000 KB               WR = 4.000 KB
BANK1          RD = 0.000 KB               WR = 0.000 KB
BANK2          RD = 0.000 KB               WR = 0.000 KB
BANK3          RD = 0.000 KB               WR = 0.000 KB
TEST PASSED
```

4. To view the profile summary and timeline trace, run the following commands to convert them for the SDx IDE to read and view the updated information below:

```
sdx_analyze profile sdaccel_profile_summary.csv
sdx_analyze trace sdaccel_timeline_trace.csv
```

For the Profile Summary you should get something similar in the following figure.

Figure 25: **SDx Makefile Hardware Emulation Profile Summary**



Profile Summary

Report name: Profile Summary (sdaccel_profile_summary) Build configuration: Unknown
Project name: vadd
Created: 12 Dec 2017 17:25

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

Top Data Transfer: Kernels and Global Memory

Device	Compute Unit	Number Of Transfers	Average Bytes per Transfer	Transfer Efficiency (%)	Total Data Transfer (MB)	Total Write (MB)	Total Read (MB)	Tran Rate
xilinx_kcu1500_dynamic_5_0-0	All	24	512.000	12.500	0.012	0.004	0.008	

Top Kernel Execution

Kernel Instance Address	Kernel	Context ID	Command Queue ID	Device	Start Time (ms)	Duration (ms)	Global Work Size	Local Work Size
0x2169d10	krnl_va...	0	0	xilinx_kcu1500_dynamic_5_...	1.21E-4	0.016	1:1:1	1:1:1

Profile Rule Checks (10 met, 3 warnings)

Rule	Threshold Value	Actual Value	Conclusion	Details
Kernel Data Transfer (4 met, 2 warnings)				
Average Read Size (KB)	> 0.512	0.512	Met	Average kernel read transfer size is adequate.
Average Write Size (KB)	> 0.512	0.512	Met	Average kernel write transfer size is adequate.
Read Bandwidth (%)	> 5.000	4.534	Not Met	Kernel read utilization of 4.534% on port All is low.
Write Bandwidth (%)	> 5.000	2.267	Not Met	Kernel write utilization of 2.267% on port All is low.
Read Amount – Minimum (MB)	> 0.250	0.500	Met	Compute units on all devices use adequate data from host.
Read Amount – Maximum (MB)	< 2.000	0.500	Met	No data re-use issues were found between host and compute
Host Data Transfer (2 met, 0 warnings)				
Average Read Size (KB)	> 4.096	4.096	Met	Host read transfers are efficient from off-chip global memory
Average Write Size (KB)	> 4.096	8.192	Met	Host write transfers are efficient to off-chip global memory.
Resource Usage (4 met, 1 warning)				
Compute Unit Calls – Minimum	> 0	1	Met	All available compute units were used.

Step 5: Running System Run

1. To compile for a System Run, run the following command:

```
make check TARGETS=hw_emu DEVICES=xilinx_kcu1500_dynamic_5_0
```

***Note*:** Building for System could take a long time depending on computer resources.

2. Once the build is complete, prepare the board installation by using the following command:

```
xbinst --platform xilinx_kcu1500_dynamic_5_0 -z -d .
```

Where:

```
--platform, the platform to be used by the design
-z, archive the board installation files for deployment
-d, the destination directory to use (Required)
```

3. Once complete, a folder called `xbinst` is created that contains all the files and scripts needed to deploy the design. To do this, run the `install.sh` script. The script installs the appropriate libraries, and firmware, and creates a `setup.sh` to be used to setup the runtime environment.

4. Run `setup.sh` to prepare the runtime environment.

***Note*:** Running `setup.sh` requires elevated permissions.

5. With the System Run completed, you can re-run this in emulation if desired. Re-run the follow command:

```
make check TARGETS=hw_emu DEVICES=xilinx_kcu1500_dynamic_5_0
```

Note: Running this command with the `TARGET` set to `hw` results in a runtime error on locating a platform.

As in the earlier step, the following reports are generated: profile summary, timeline trace, and system estimates.

Use the following commands to convert the profile summary and timeline trace into files that SDx can read:

```
sdx_analyze profile sdaccel_profile_summary.csv
sdx_analyze trace sdaccel_timeline_trace.csv
```

Summary

After completing this tutorial, you should be able to do the following:

- Set up the SDx™ enviornment to run all commands in a terminal.
- Cloning a Github repository.
- Be able to run `xcpp`, `xocc`, `emconfig`, `sdx_analyze profile`, `sdx_analyze trace` commands to generate the application, binary container, emulation model.
- Write a makefile to compile an OpenCL™ kernel and host code.
- View the generated files from emulation in a text editor or SDx IDE.
- Setup the environment and deploy the design to be used with the platform.

Appendix A

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

Solution Centers

See the Xilinx Solution Centers for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips

References

1. *SDx Environments Release Notes, Installation, and Licensing Guide* (UG1238)
2. *SDAccel Environment User Guide* (UG1023)
3. *SDAccel Environment Optimization Guide* (UG1207)
4. *SDAccel Environment Tutorial: Introduction* (UG1021)
5. *SDAccel Environment Platform Development Guide* (UG1164)
6. SDAccel Development Environment web page
7. Vivado® Design Suite Documentation
8. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
9. *Vivado Design Suite User Guide: Creating and Packaging Custom IP* (UG1118)
10. *Vivado Design Suite User Guide: Partial Reconfiguration* (UG909)
11. *Vivado Design Suite User Guide: High level Synthesis* (UG902)
12. *UltraFast Design Methodology Guide for the Vivado Design Suite* (UG949)
13. *Vivado Design Suite Properties Reference Guide* (UG912)
14. Khronos Group web page: Documentation for the OpenCL standard

15. Alpha Data web page: Documentation for the ADM-PCIE-7V3 Card
16. Pico Computing web page: Documentation for the M-505-K325T card and the EX400 Card

Please Read: Important Legal Notices

Copyright