

SDAccel Environment Tutorial

Introduction

UG1021 (v2016.4) March 9, 2017

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
03/09/17	2016.4	Tutorial validated for SDx™ IDE 2016.4.
11/30/16	2016.3	Initial documentation release for SDx IDE 2016.3, which includes both the SDSoc™ Environment and the SDAccel™ Environment. Due to this major change in tool architecture, this document has undergone substantial changes in structure and content since the previous release.

Table of Contents

Introduction

Flow Overview

Lab 1: Introduction to the SDAccel Development Environment	5
--	---

Additional Resources and Legal Notices

References	24
Please Read: Important Legal Notices	25

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++ 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 Smith-Waterman algorithm, which is a database search algorithm developed by T.F. Smith and M.S. Waterman. It is based on the earlier Needleman and Wunsch algorithm.

The design targets the Alpha Data Kintex® Ultrascale™ PCIe® board utilizing the xcku060 device.



TIP: Although this tutorial design targets a xcku060 Kintex Ultrascale board, you can choose any other board, such as the xc7k690t Kintex-7 board. Tutorial results should be similar.

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.

Locating Tutorial Design Files

You can find the files for this tutorial in the SDx Suite examples directory at the following location:

```
<SDx_install_area>/<version>/examples
```

NOTE: SDAccel is only available on Linux operating systems that support the GLIBC 2.9 or higher compiled library.

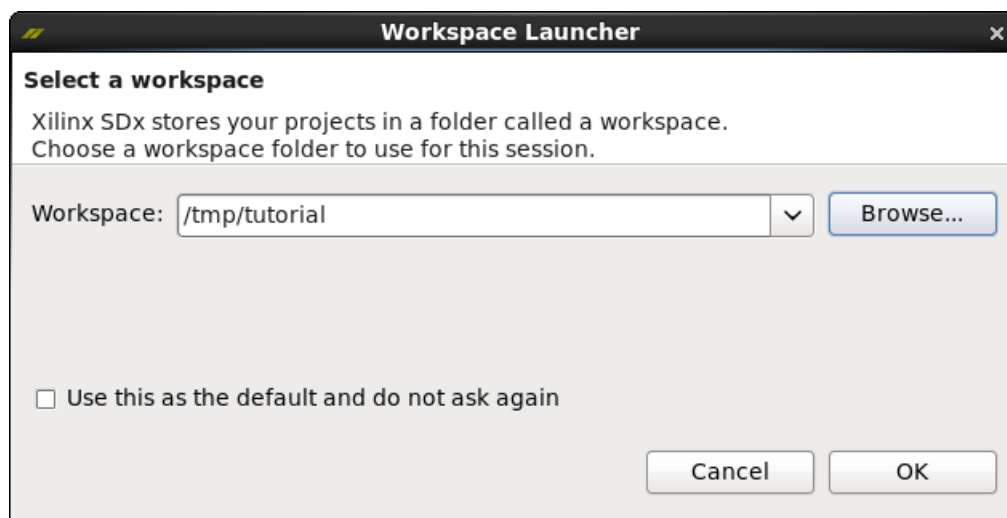
Flow Overview

Lab 1: Introduction to the SDAccel Development Environment

Step 1: Creating SDAccel Project

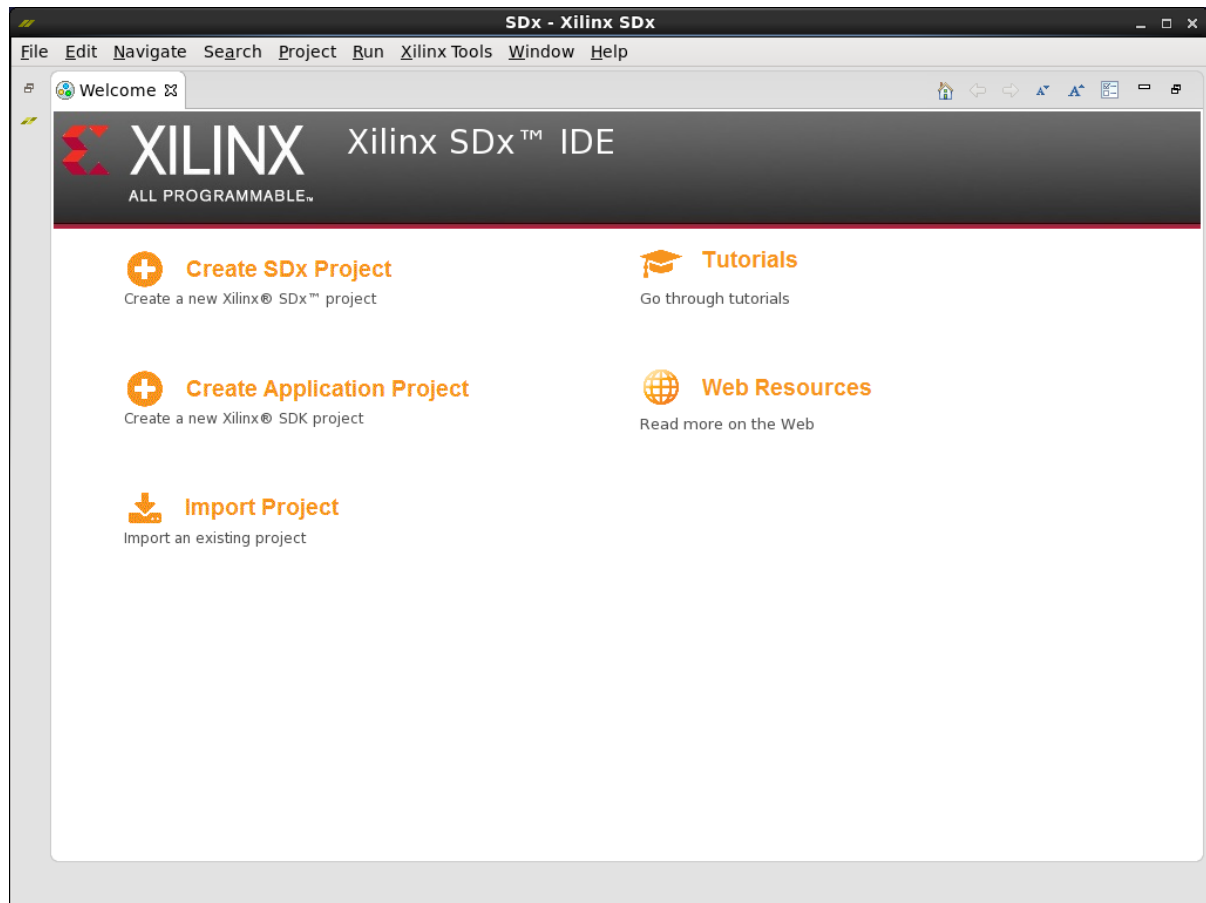
1. Launch SDx with the command `sdx` and you will see the **Workspace Launcher** window. Select a location for your workspace, this is where the project will reside.

Figure 1: Workspace Launcher



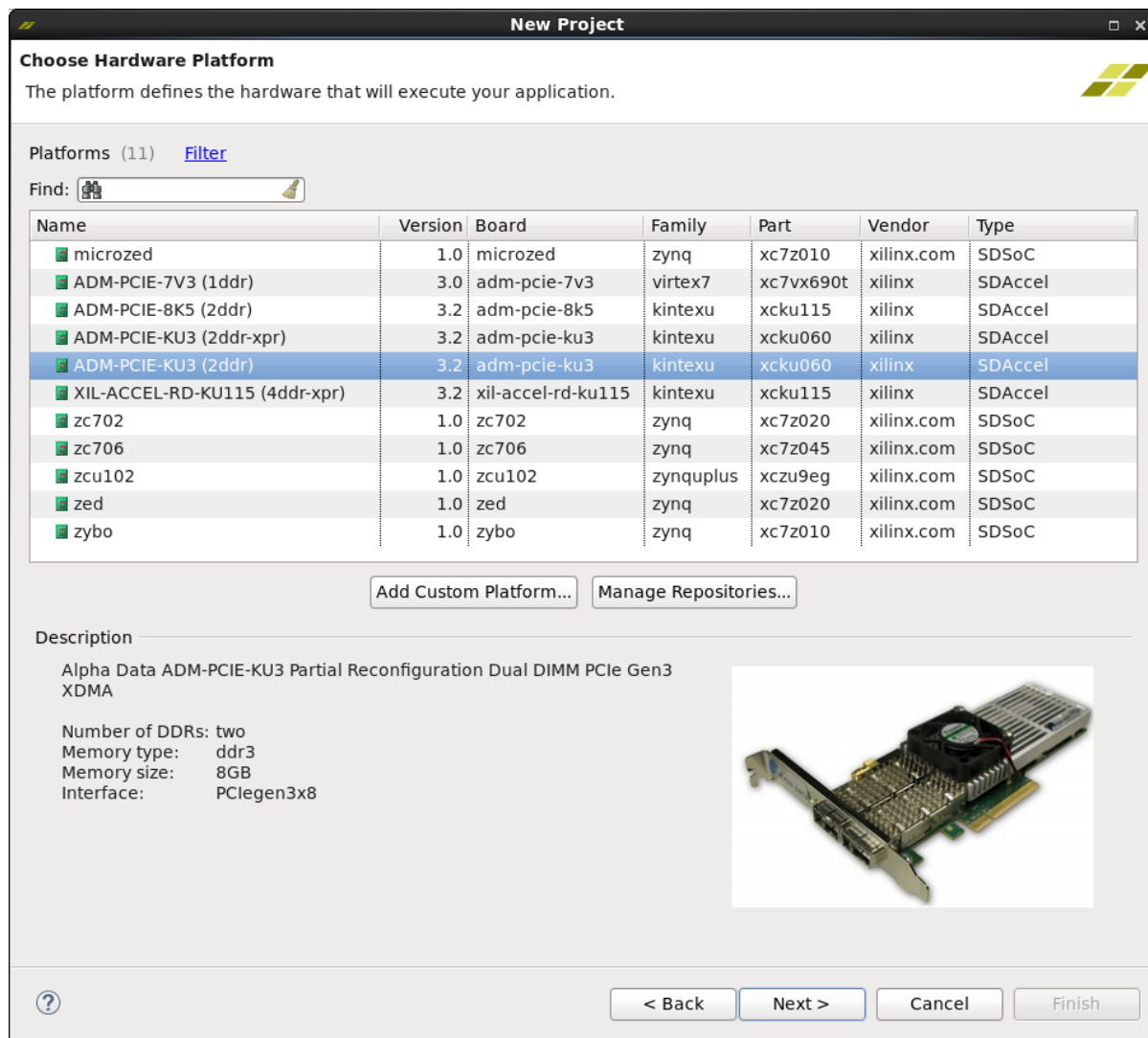
2. In the the **SDx Welcome** window, click **Create SDx Project**.

Figure 2: SDx Welcome Window



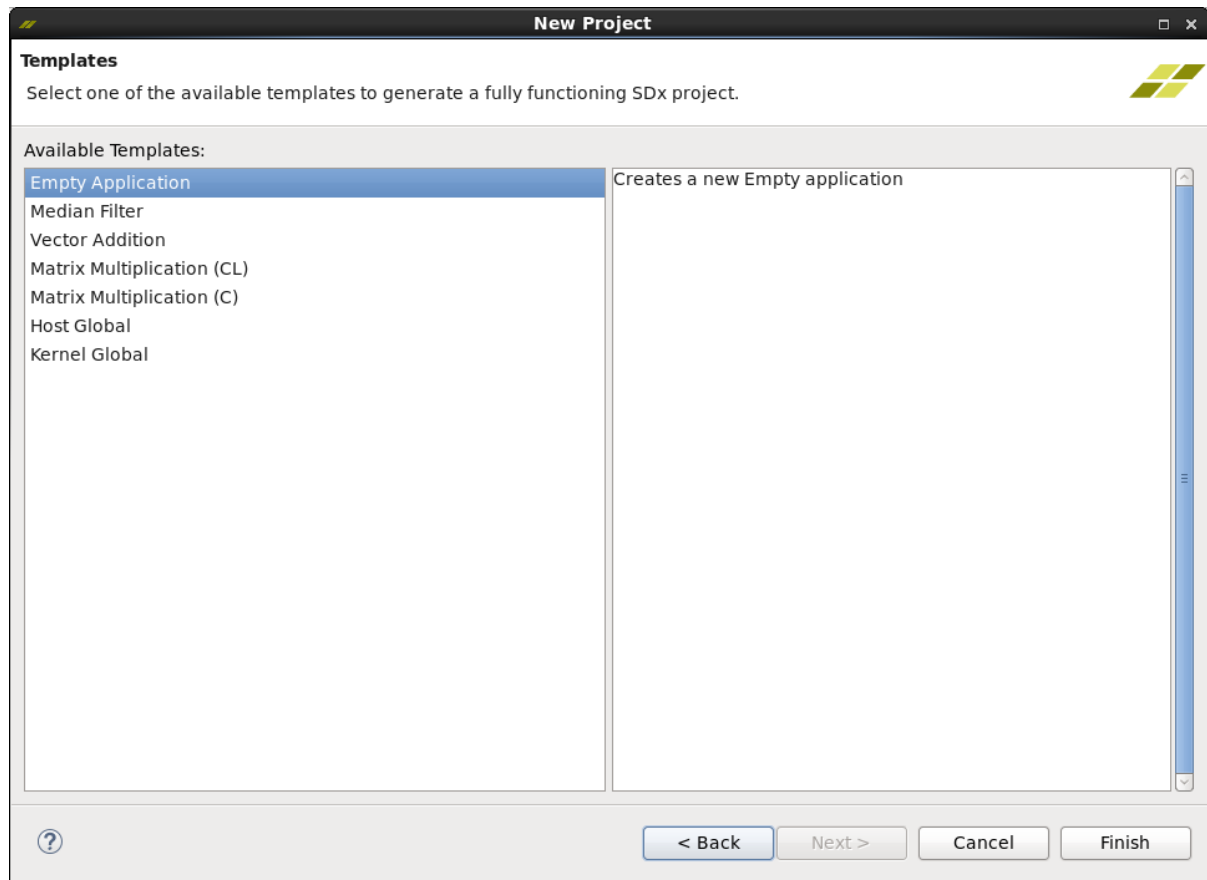
3. In the **Create a New SDx project** window in the **Project name** field, type **smithwaterman** and click **Next**.
4. In the **Choose Hardware Platform** window choose the **ADM-PCIE-KU3 (2ddr)** platform and click **Next**.

Figure 3: Hardware Platform



5. The **Software Platform** window will only have **Linux on x86** as a valid option, click **Next**.
6. The **Templates** window has a list of possible templates that you can use to get started in building an SDAccel project. For this tutorial, select **Empty Application** and click **Finish**.

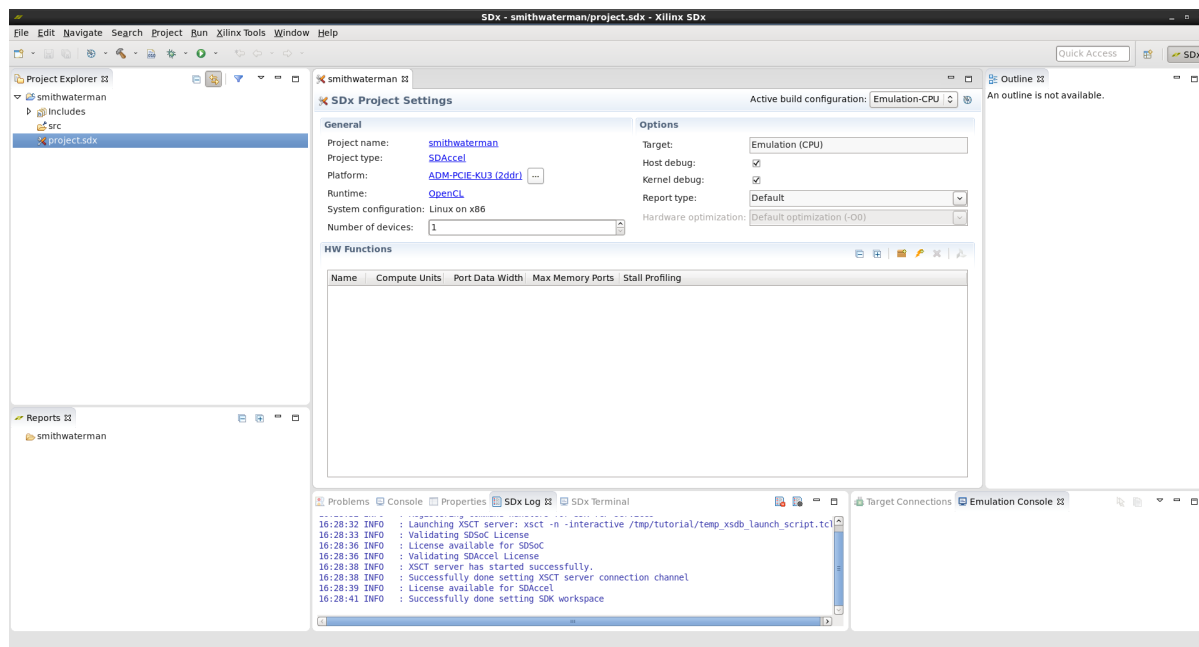
Figure 4: New Project



Step 2: Importing Design Files

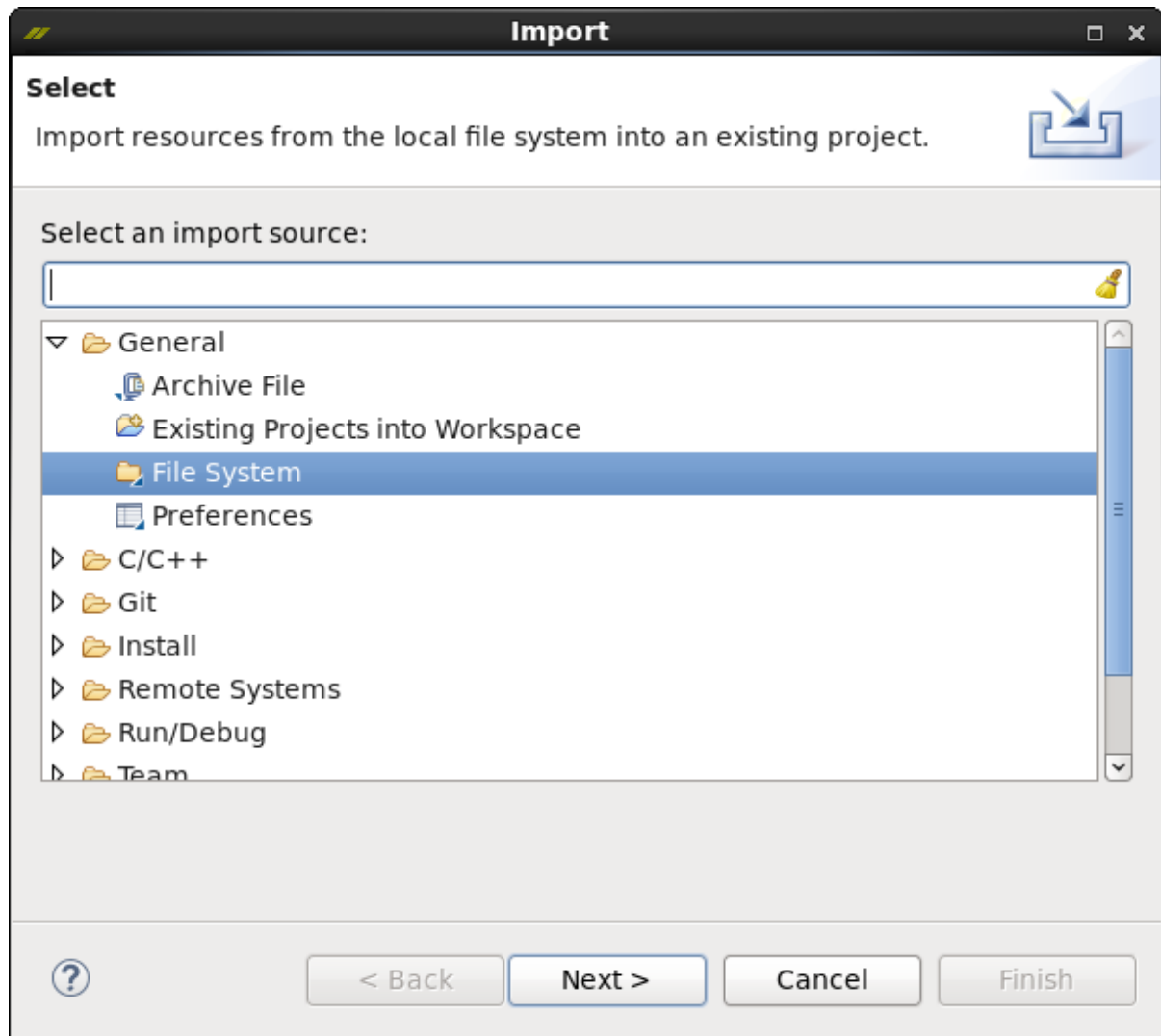
1. In the **Project Explorer** window, expand **smithwaterman** > **src**. Right-click **src** and select **Import**.

Figure 5: SDx Settings Overview



2. In the **Import** dialog box, under **General**, select **File System**, and click **Next**.

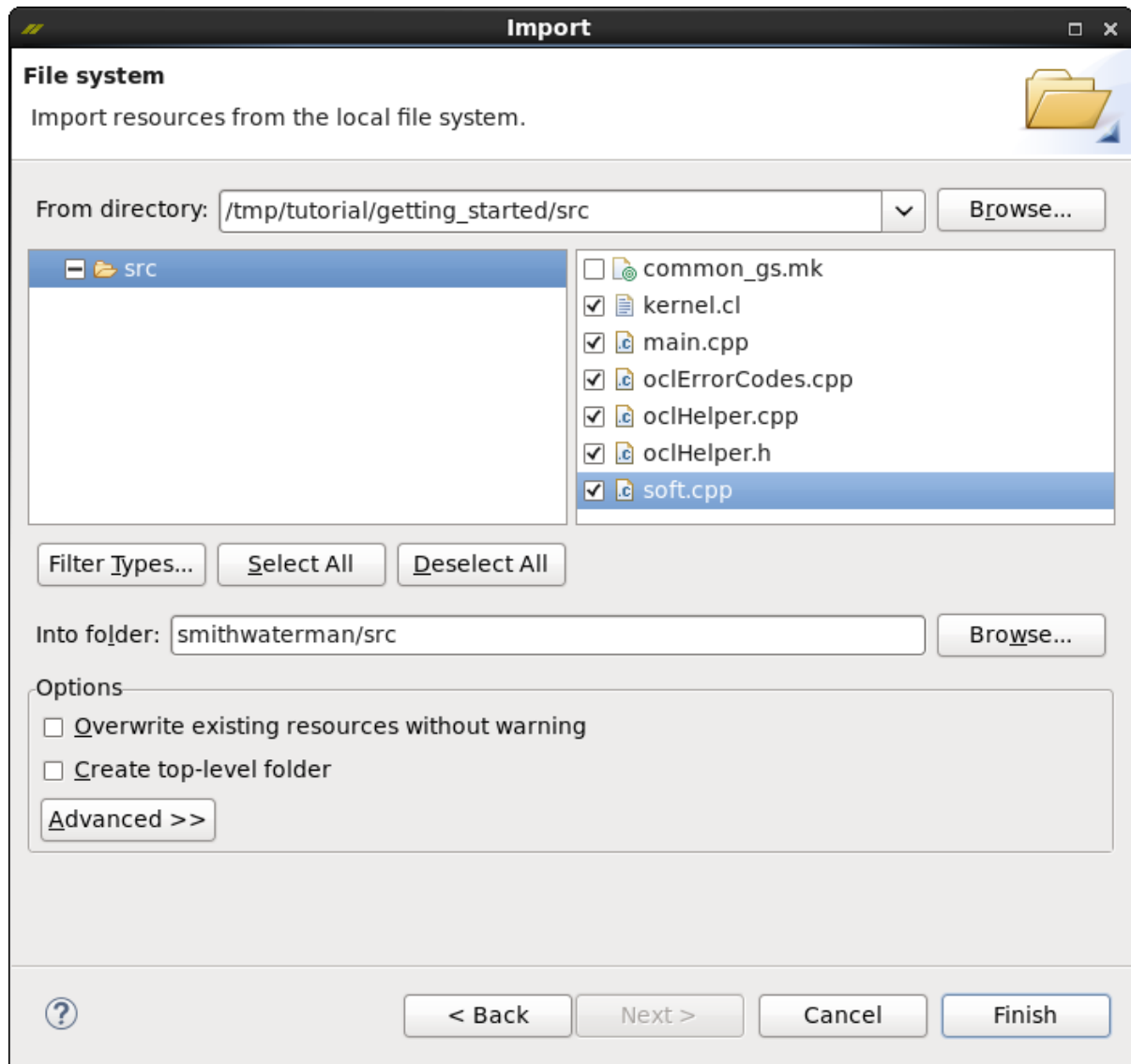
Figure 6: Eclipse Import Dialog



3. In the **Import** dialog and text box labeled **From directory**, navigate to the **getting_started** directory from the install directory (<install location>/examples), and select the **src** directory. Click **OK**.
4. In the **Import** dialog box, select the following files:
 - **kernel.cl**
 - **main.cpp**
 - **oclErrorCodes.cpp**
 - **oclHelper.cpp**
 - **oclHelper.h**
 - **soft.cpp**

Click **Finish**.

Figure 7: Import Dialog



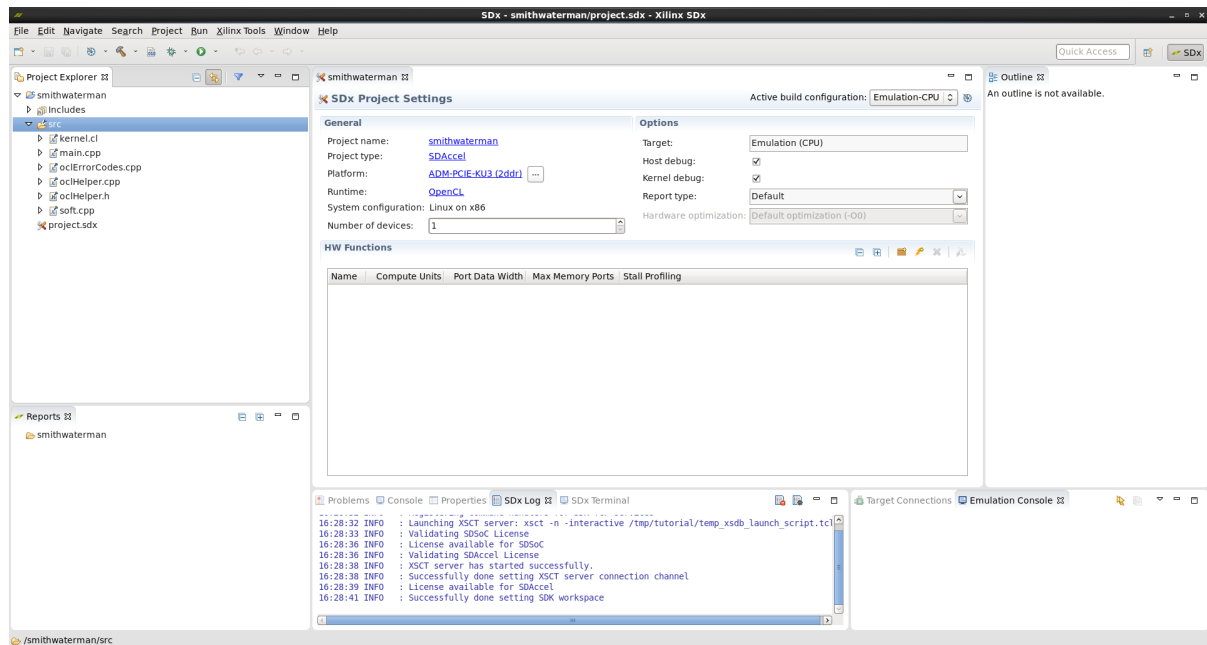
5. You can now expand the **src** directory to see that all the files are now populated in the project.

Step 3: Running CPU Emulation

This step shows you how to run CPU Emulation of a design, by setting Run Configuration settings, opening reports, and showing how to launch Debug. Details on reports and Debug can be found in the *SDAccel Environment User Guide*, ([UG1023](#)) .

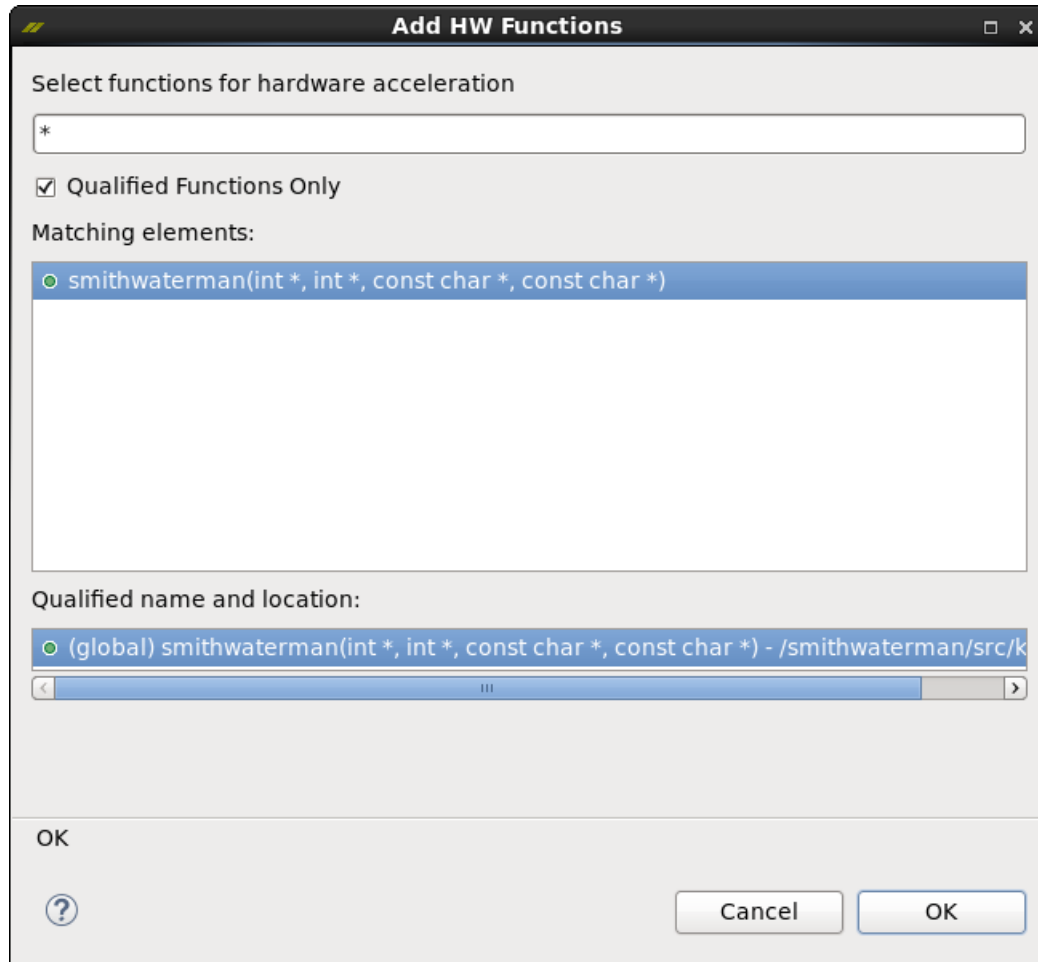
1. To run CPU Emulation, go to **SDx Project Settings** and ensure that **Active build configuration** is set to **Emulation-CPU**.


Figure 8: SDx Project Settings



2. Create an accelerator that will be the container for the kernel found in `kernel.cl`. In this view, click the lightning bolt button: . SDx analyzes the design for all possible kernels in the design (as well as the ability to filter the list if there are multiple kernels). For this design, only the **smithwaterman** exists. Ensure that the function name **smithwaterman** is selected and click **OK**. This creates a binary container for the kernel, which can be renamed if necessary.

Figure 9: Add HW Functions Dialog



3. Click the Green Arrow:  to run CPU Emulation. This builds the project before running the emulation.
4. Take note that the design successfully builds, but the emulation test fails. This can be viewed by looking for **FAILED TEST** in the **Console** window. Look at the `main.cpp` and notice that there are arguments that need to be provided. The **Run Configurations** field need to be adjusted to account for these arguments.
5. Go to the **Run** menu and to **Run Configurations...**

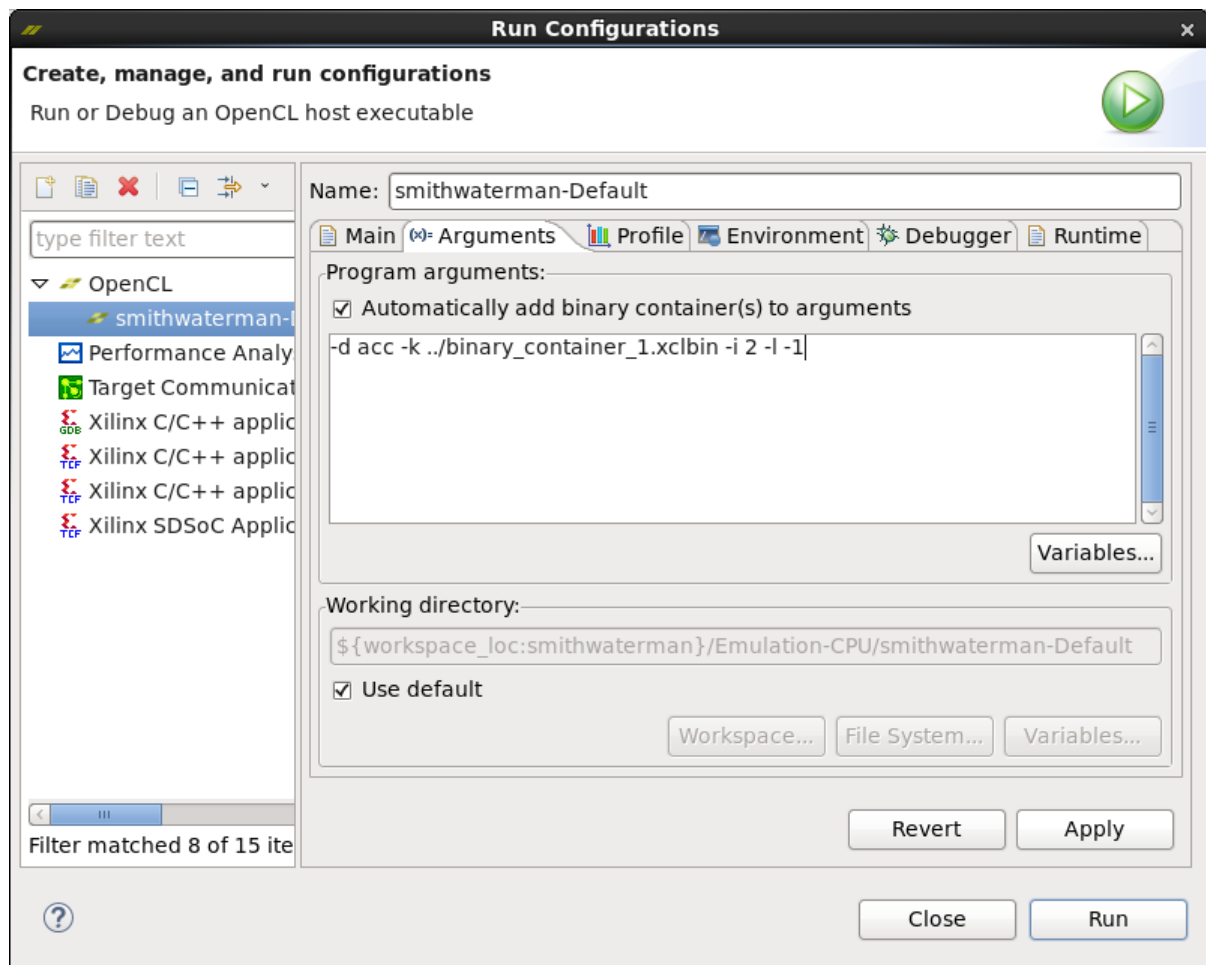
6. Under the **Arguments** tab we see only the container that holds the kernel as an argument. For this design, the following arguments need to be adjusted:

```
-d acc -k ../binary_container_1.xclbin -i 2 -l -1
```

- `-d`, Is to say what type of device it is. In this case, `acc` specifies an accelerator.
- `-k`, Is to specify the kernel to use. If `-d` is set to `acc` then this must be a binary file.
- `-i`, The number of iterations to run.
- `-l`, The sequence length to be used in the algorithm. `-1` specifies the default length.

(To view more information about the argument list, use `-h` for the executable to see an entire list) Click **Run**.

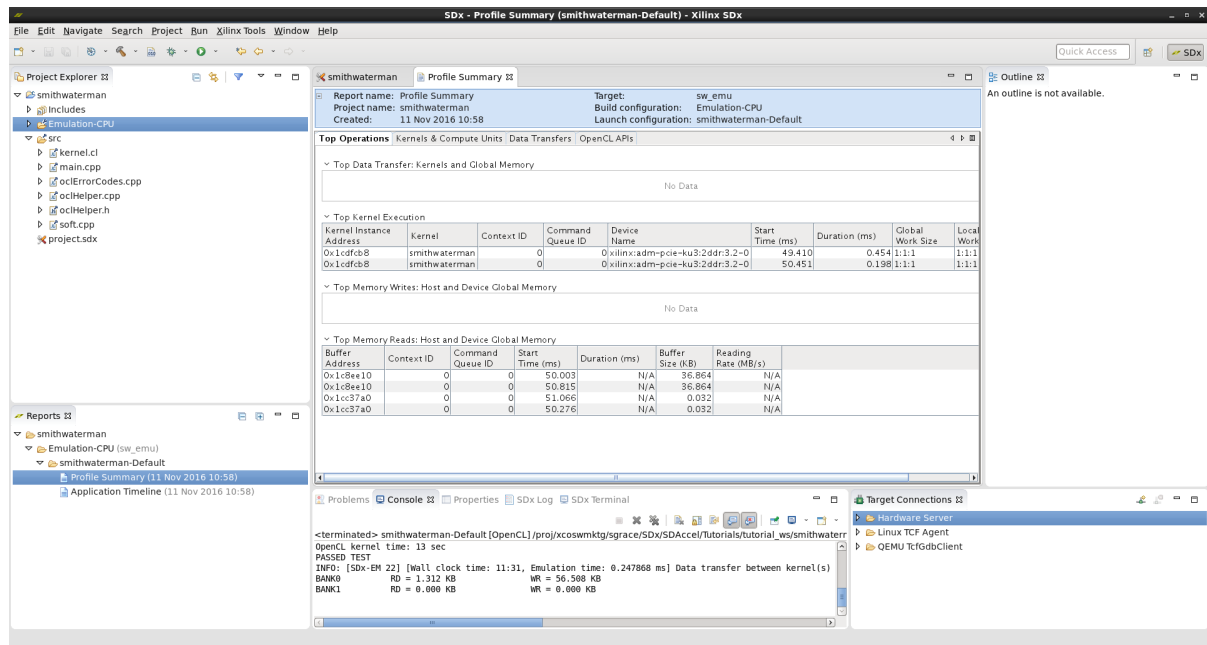
Figure 10: Run Configuration



7. The **Console** window should show **PASSED TEST**. If you want to see a verbose output of what the algorithm is doing, go back into **Run Configurations...** and add the `-v` to the arguments.

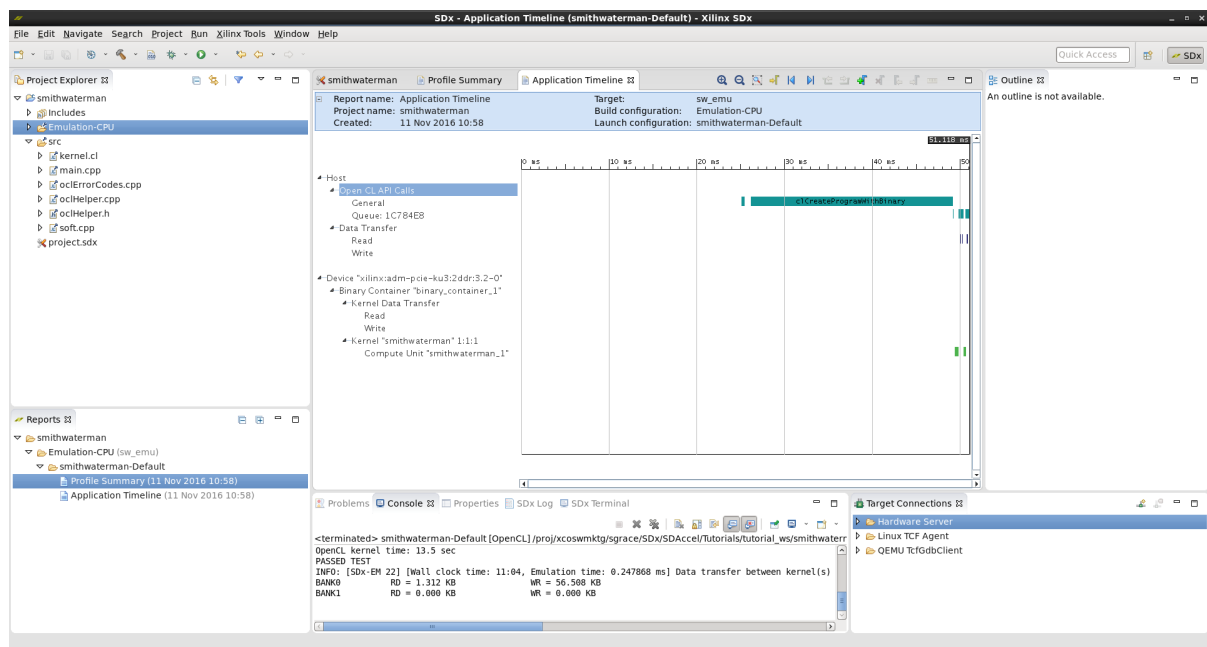
8. After the emulation run is complete, you can look at two reports to design details for further optimization. In the **Reports** window, double click **Profile Summary**. Here, you can view operations, execution time, bandwidth, and other useful data that can be used to optimize the design. Do note that the summary numbers may vary.

Figure 11: Emulation-CPU Profile Summary



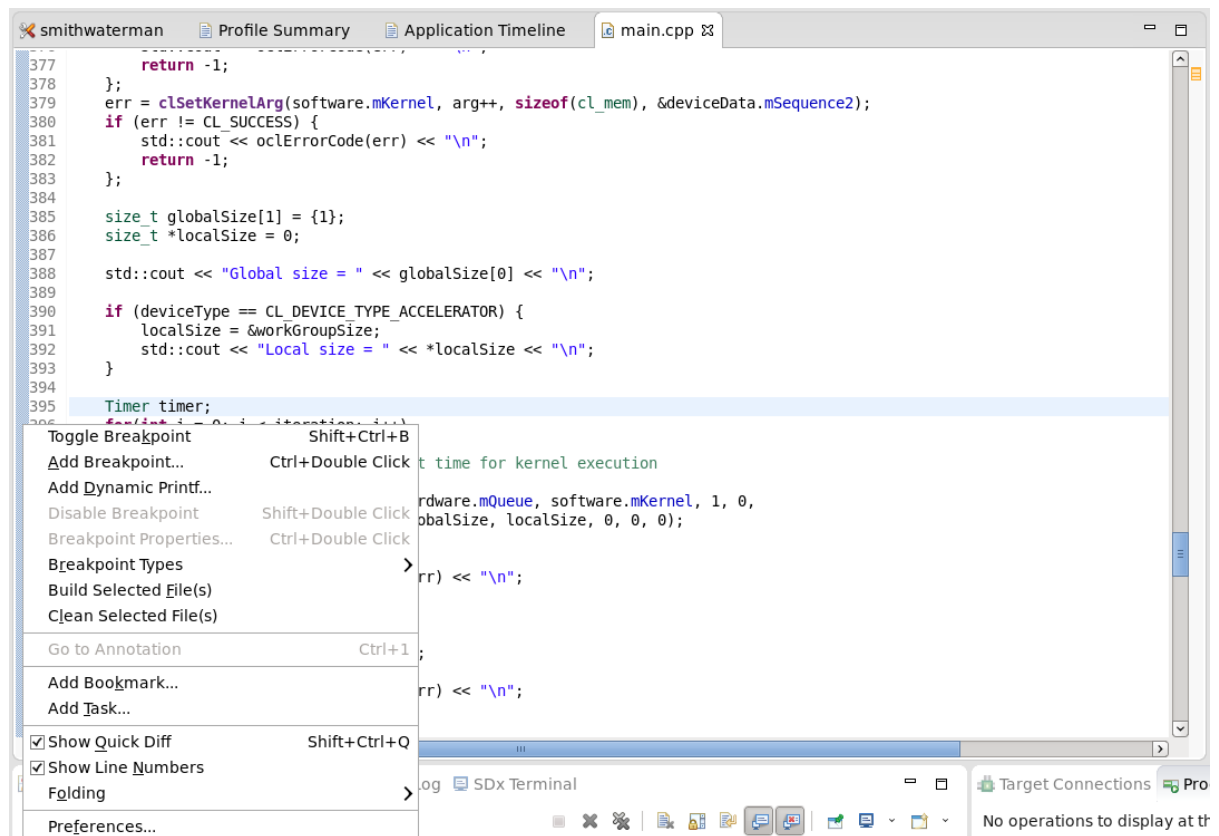
9. To view the **Application Timeline** report, in the **Reports** window, double click **Application Timeline**. This shows a breakdown of the host code and the kernel code, and execution time for each.

Figure 12: Application Timeline Report



10. From the **Profile Summary** and the **Application Timeline** you can see issues in how the host and kernel communicate with each other. Using the **Debug** feature can help pinpoint these issues.
11. To run in Debug, a breakpoint needs to be set. Setting breakpoints at key points in the execution helps identify problems. From the **Application Timeline**, notice that the API Calls are staggered around the end of the timeline. If you zoom in by clicking and dragging the mouse near the end of the timeline, you can see the fluctuation more easily. Hovering the mouse over the **Queue** line over the green boxes, you will notice that the tooltip shows that **clEnqueueReadBuffer** is being called several times. Set a breakpoint at the main for loop in `main.cpp` (line 396) by right-clicking the line number on the line and selecting **Toggle Breakpoint**. This is the iteration loop where **clEnqueueReadBuffer** is executed in the code.

Figure 13: Setting Breakpoint




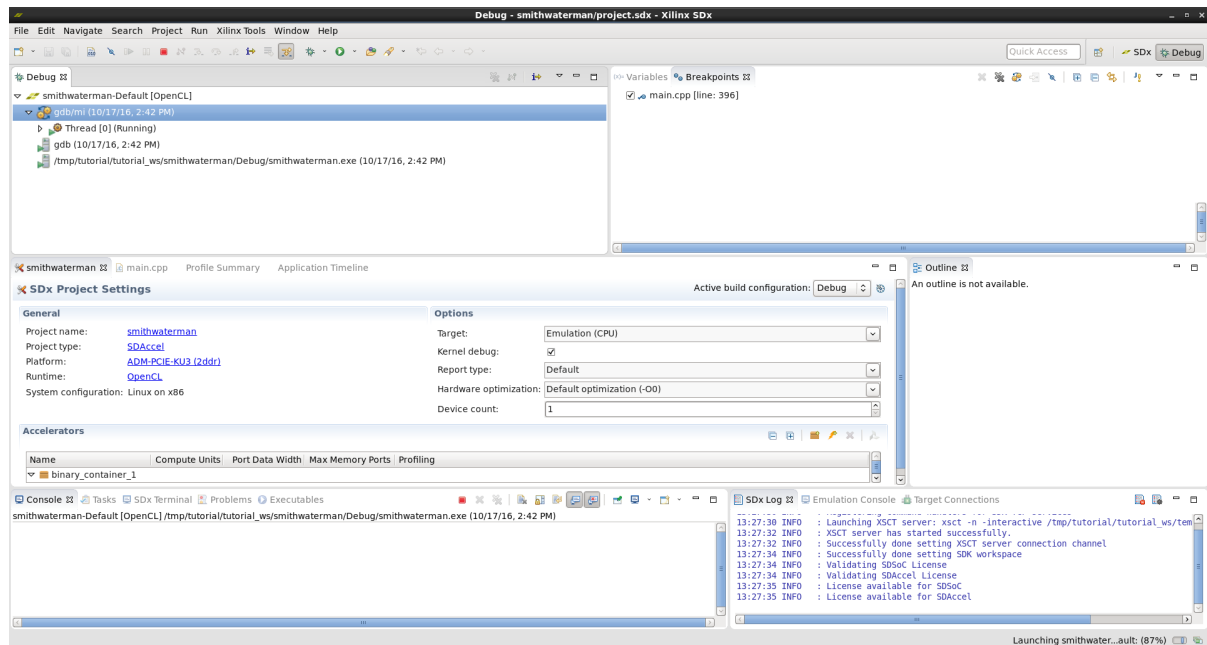
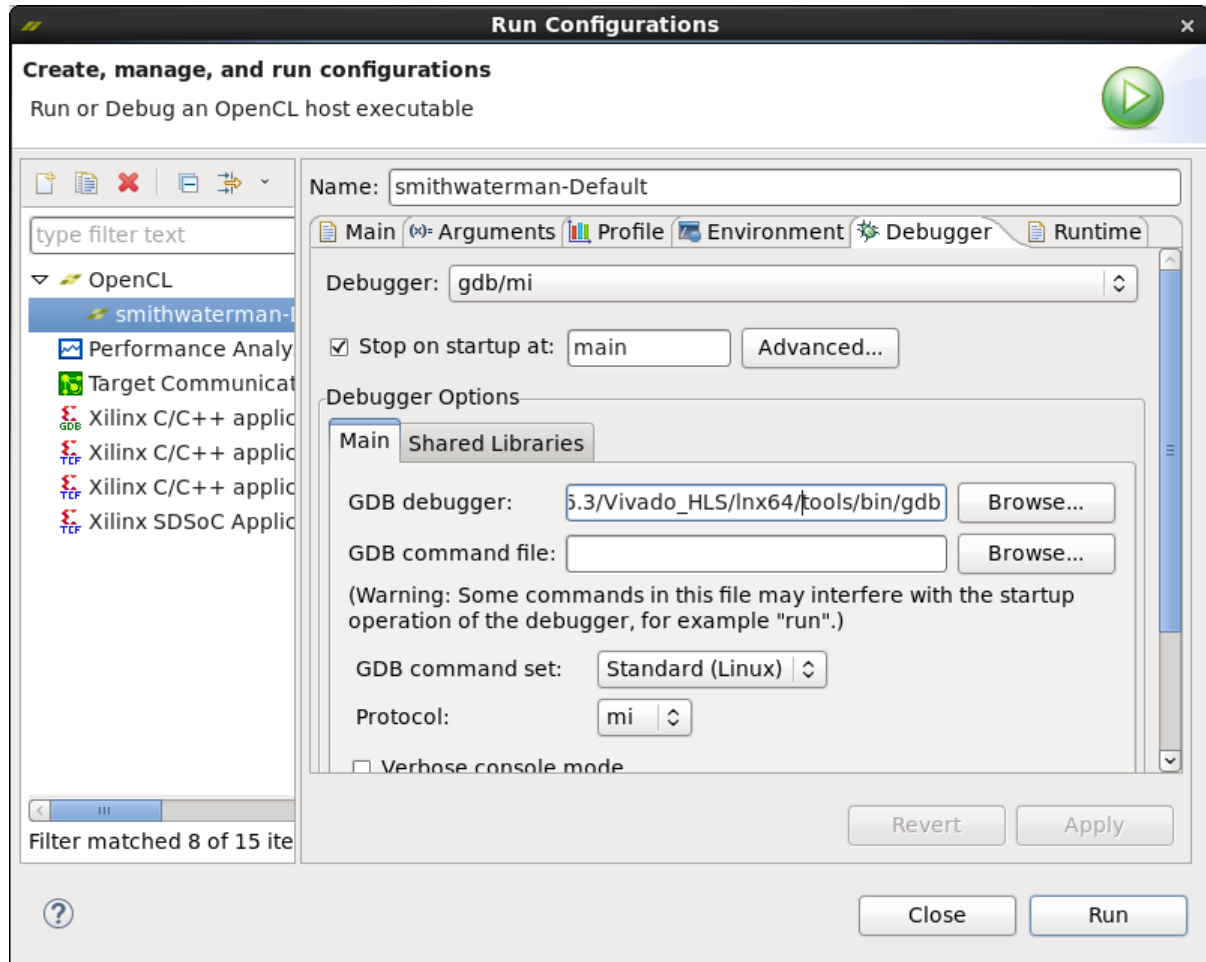
12. To run **Debug**, click on this icon: . A dialog box opens up asking you to switch to that perspective. Click **Yes**.
13. Using Eclipse debugging, the host and kernel code can be examined in more detail. All the controls with which to do step-by-step debugging are in the **Run** menu.

Figure 14: Eclipse Debugging



14. After you start, Debug stops at line 441. This is the first line of `main` to be executed. 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 continue to the next breakpoint.

Figure 15: Run Configuration



15. The debugger is now at the `for` loop where you set the breakpoint. The run was configured to go through two iterations. Step through the loop while looking at the **Variables** window, and see the variables changing as the stepping occurs.
16. Close the **Debug Perspective** by going to the upper-right of the window where it shows **Debug**, right-click and select **Close**.

Step 4: Running Hardware Emulation

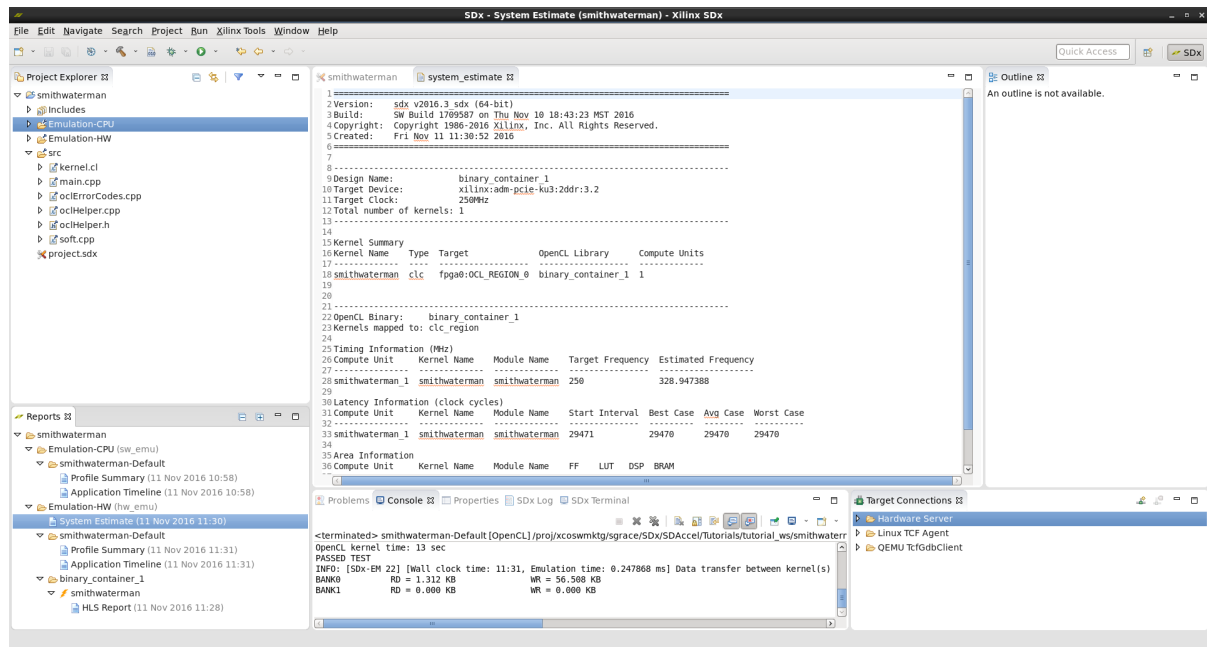
This step will cover running hardware emulation feature as well as looking at the basics of profiling and reports.

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

NOTE: The main difference between **Emulation-CPU** 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. The design also takes longer to compile.

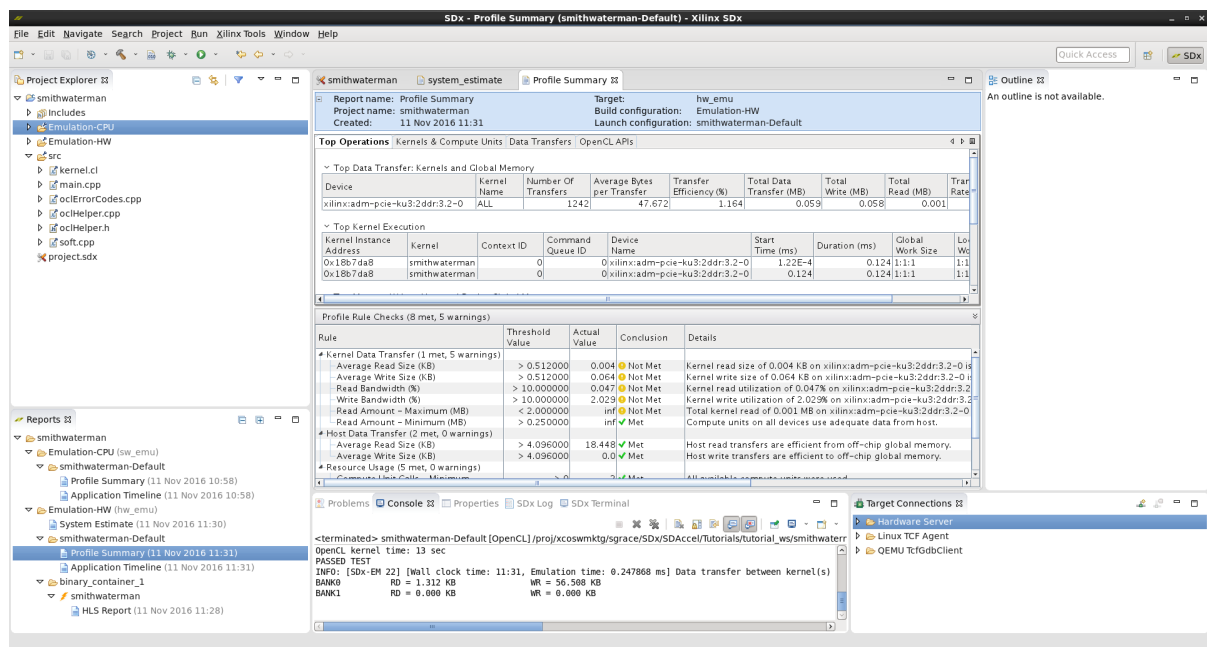
2. In the **Reports** tab, open **System Estimate**. This is a text report that provides information related to kernel information, timing about the design, clock cycles, and area used in the device.

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.

Figure 17: Profile Summary Report



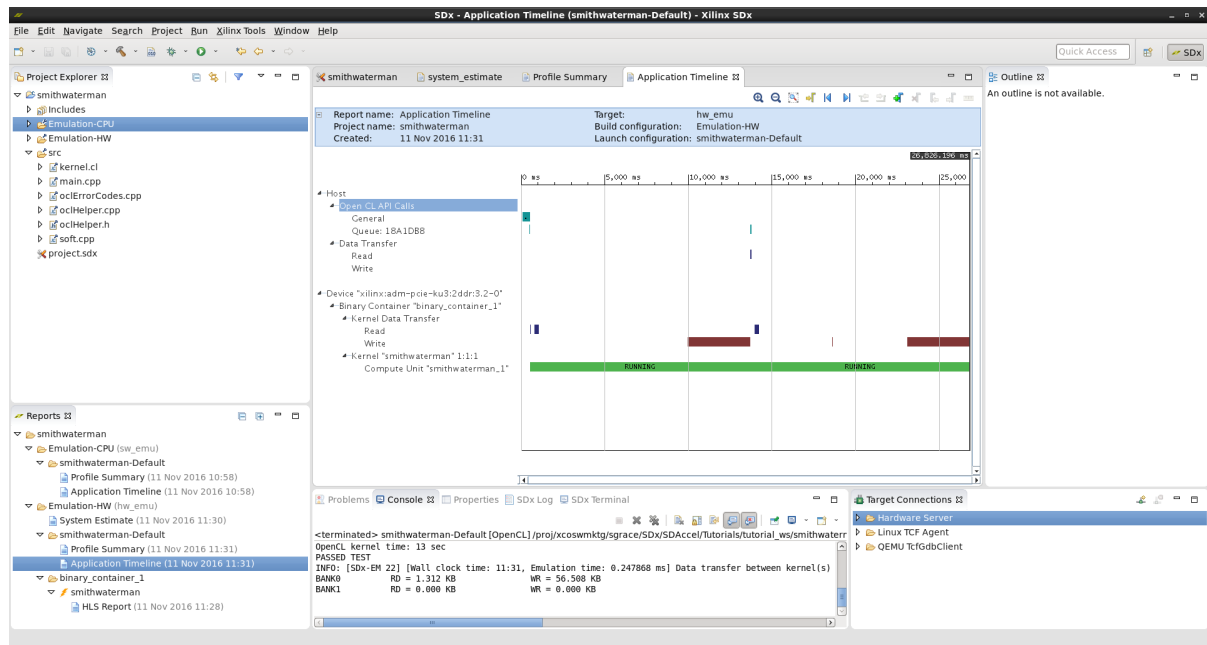
4. Scroll to the right in the **Profile Rule Checks** and look for the header column labeled **Guidance**. Here is where unmet checks provide some information on how to optimize the kernel.

Figure 18: Profile Rule Checks

NOTE: To see other performance optimization techniques and methodologies, please go to the *SDAccel Performance Optimization Methodology Guide* ([UG1207](#)).

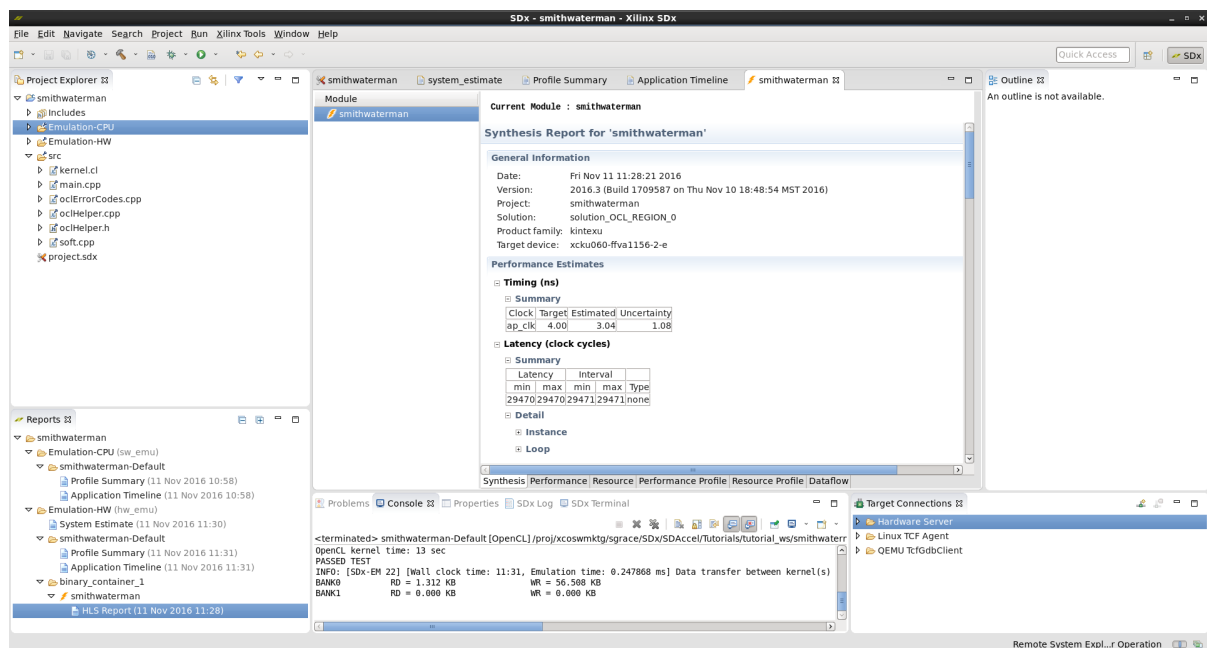
5. Open the **Application Timeline** report. This report shows the estimated time it takes for the host and kernel to complete the task and provides finer grained information on where bottlenecks can be. In this example, it is iterated twice and this timeline shows the kernel is run twice. Adding a marker,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

Figure 19: Application Timeline Report



- 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 be latency and clock period.

Figure 20: HLS Report

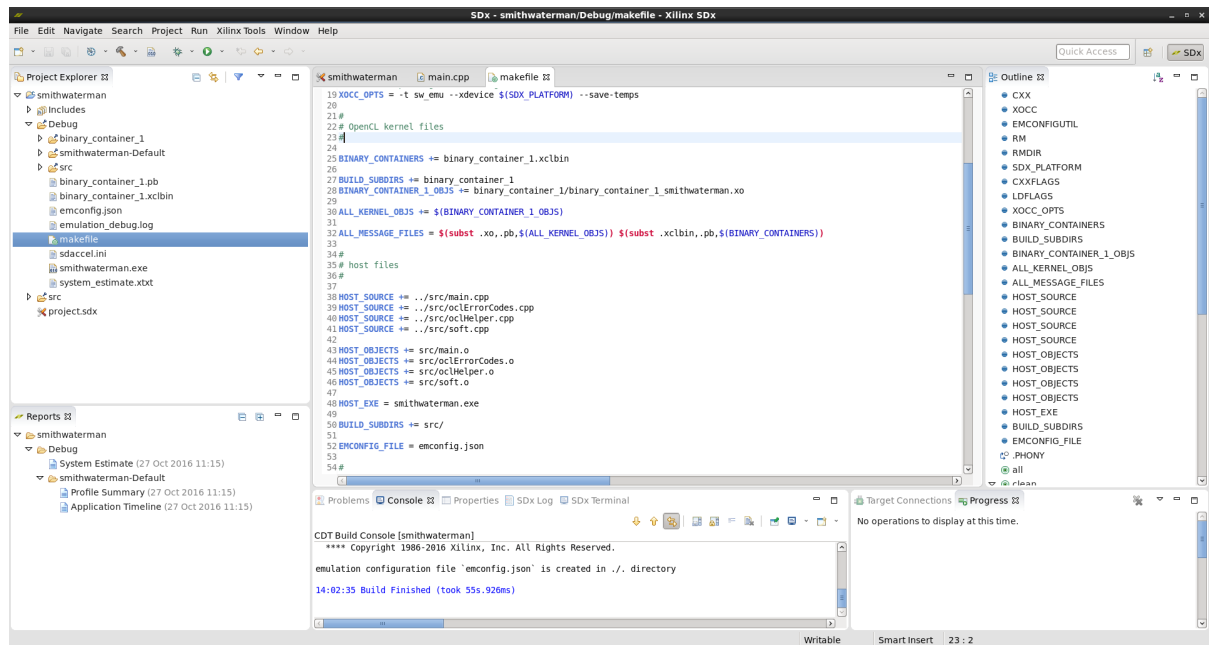


Step 5: Makefile Flow

This step explains the basics of the Makefile flow and how SDx uses it. The advantages to 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-CPU** 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.
 2. In the **Project Explorer**, navigate to the **Emulation-HW** directory and look for the **makefile** file. Open the file.
 3. While the project is building, go back into the **makefile** editor window and look at line 19 in both files. Notice that each one is set to either `hw_emu` or `sw_emu`.

Figure 21: Makefile Editor window



4. The makefile can also access SDx without using the GUI. Open up a new terminal session and navigate to where the workspace and navigate to the **Emulation-CPU** directory and type: **make incremental**. The process will produce a typical SDx log output.

Summary

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

- Create an SDAccel™ project and import the required design files.
- Create a binary container and accelerator for the design.
- Run CPU 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 CPU and Hardware Emulation reports.
- Read the project makefile and run the makefile command line.

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: Creating and Packaging Custom IP* ([UG1118](#))
9. *Vivado Design Suite User Guide: High level Synthesis* ([UG902](#))
10. [Khronos Group web page](#): Documentation for the OpenCL standard
11. [Alpha Data web page](#): Documentation for the ADM-PCIE-7V3 Card
12. [Pico Computing web page](#): Documentation for the M-505-K325T card and the EX400 Card

Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at www.xilinx.com/legal.htm#tos; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at www.xilinx.com/legal.htm#tos.

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

© Copyright 2017 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission by Khronos. PCI, PCIe and PCI Express are trademarks of PCI-SIG and used under license. All other trademarks are the property of their respective owners.