

SDAccel Environment User Guide

UG1023 (v2017.1) June 20, 2017

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
06/20/17	2017.1	<ul style="list-style-type: none">Revised and reorganized Kernel Optimization Support chapter.New section for OpenCL Installable Client Drive (ICD) Loader.New Getting Started with Examples chapter.New XP Parameters table in Commpliation Flow chapter.Revised xocc Options table in Compilation chatper.Updates to accommodate 2017.1 SDx software release.

Table of Contents

Introduction

Understanding the OpenCL Platform and Memory Model

OpenCL Platform Model	6
OpenCL Devices and FPGAs	8
OpenCL Memory Model	9
OpenCL Installable Client Driver Loader.....	12
Recommended Libraries	12

Kernel Language Support

Expressing a Kernel in RTL	14
Expressing a Kernel in OpenCL C	26
Expressing a Kernel in C/C++	26

Compilation Flow

Xilinx OpenCL Compiler	29
Compiling Your OpenCL Kernel Using the Xilinx OpenCL Compiler (xocc).....	32

Getting Started with Examples

Installed Examples.....	45
GitHub Examples	46

Estimating Performance

Generating the System Performance Estimate Report.....	48
Analyzing the Performance Estimate Report.....	48

Application Profiling in the SDAccel Environment

Kernel Synthesis Report.....	53
Profiling Summary Report	55
Application Timeline	62
Device Hardware Transaction View.....	65
Detailed Kernel Trace	71

Debugging Applications in the SDAccel Environment

Preparing the Host Application for Debug.....	73
Preparing Kernel Code for Debug in CPU Emulation Flow.....	73
Launching GDB Standalone	74
Application Debug.....	75
Kernel Debug	76

SDAccel Environment Supported Devices

OpenCL Built-In Functions Support in the SDAccel Environment

xbinst Command Reference

Xilinx Board Swiss Army Knife Utility

xbsak Commands and Options	93
----------------------------------	----

Using the Runtime Initialization File

Converting Tcl Compilation Flow to XOCC

SDAccel System Info Checker Utility

Launching System Info Checker Utility	106
---	-----

Board Installations

Installing the Alpha Data ADM-PCIE-KU3 Card	108
Installing the Alpha Data ADM-PCIE-7V3 Card	117
Installing the Alpha Data ADM-PCIE-8K5 Card.....	126
Installing the Xilinx XIL-ACCEL-RD-KU115 Card.....	134

Additional Resources and Legal Notices

References	144
Please Read: Important Legal Notices.....	145

Introduction

Software is at the foundation of application specification and development. Whether the end application is targeted towards entertainment, gaming, or medicine, most products available today began as a software model or prototype that needed to be accelerated and executed on a hardware device. From this starting point, the software engineer is tasked with determining the execution device to get a solution to market and to achieve the highest possible degree of acceleration possible.

One traditional approach to accomplish this task has been to rely on processor clock frequency scaling. On its own, this approach has entered a state of diminishing returns, which has in turn led to the development of multi-core and heterogeneous computing devices. These architectures provide the software engineer with the possibility to more effectively trade-off performance and power for different form factors and computational loads. The one challenge in using these new computing architectures is the programming model of each device. At a fundamental level, all multi-core and heterogeneous computing devices require that the programmer rethink the problem to be solved in terms of explicit parallelism.

Recognizing the programming challenge of multi-core and heterogeneous compute devices, the [Khronos Group](#) industry consortium has developed the OpenCL™ programming standard. The OpenCL specification for multi-core and heterogeneous compute devices defines a single consistent programming model and system level abstraction for all hardware devices that support the standard. For a software engineer this means a single programming model to learn what can be directly used on devices from multiple vendors.

As specified by the OpenCL standard, any code that complies with the OpenCL specification is functionally portable and will execute on any computing device that supports the standard. Therefore, any code change is for performance optimization. The degree to which an OpenCL program needs to be modified for performance depends on the quality of the starting source code and the execution environment for the application.

Xilinx is an active member of the [Khronos Group](#), collaborating on the OpenCL specification, and supports the compilation of OpenCL programs for Xilinx® FPGAs. The Xilinx SDAccel™ development environment is used for compiling OpenCL programs to execute on a Xilinx FPGA.

There are some differences between compiling a program for execution in an FPGA and a CPU/GPU environment. The following chapters in this guide describe how to use the SDAccel development environment to compile an OpenCL program for a Xilinx FPGA. This book is intended to document the features and usages of the SDAccel development environment. It is assumed that the user already has a working knowledge of OpenCL API. Though it includes some high level OpenCL concepts, it is not intended as an exhaustive technical guide on the OpenCL API. For more information on the OpenCL API, see the OpenCL specification available from the Khronos Group, and the OpenCL API introductory videos available on the Xilinx website.

Understanding the OpenCL Platform and Memory Model

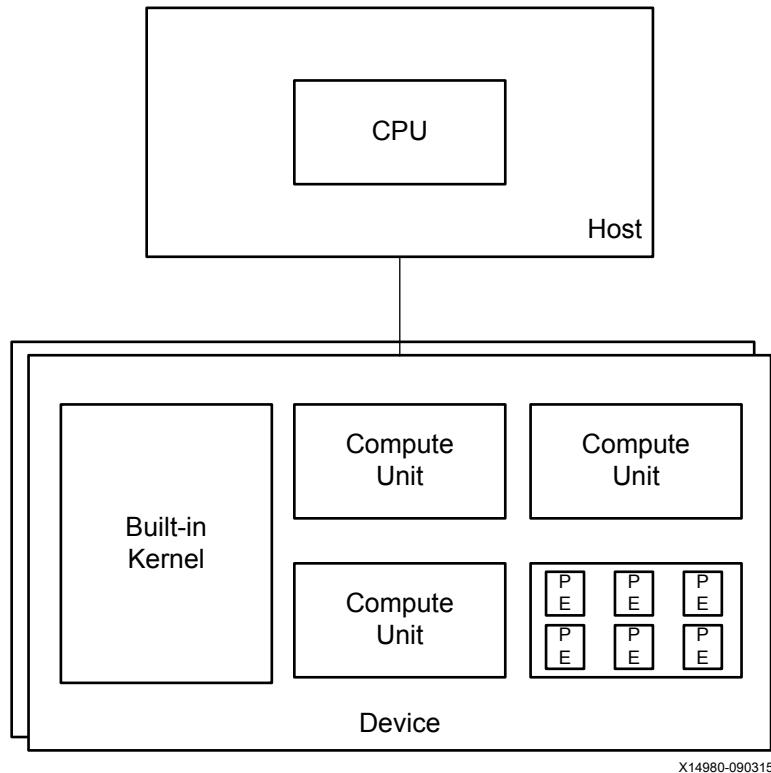
The OpenCL™ standard describes all hardware compute resources capable of executing OpenCL applications using a common abstraction for defining a platform and the memory hierarchy. The platform is a logical abstraction model for the hardware executing the OpenCL application code. This model, which is common to all vendors implementing this standard, provides the application programmer with a unified view from which to analyze and understand how an application is mapped into hardware. Understanding how these concepts translate into physical implementations on the FPGA is necessary for application optimization.

This chapter provides a review of the OpenCL platform model and its extensions to FPGAs. It explains the mapping of the OpenCL platform and memory model into an SDAccel™ development environment-generated implementation.

OpenCL Platform Model

The OpenCL™ platform model defines the logical representation of all hardware capable of executing an OpenCL program. At the most fundamental level all platforms are defined by the grouping of a processor and one or more devices. The host processor, which runs the OS for the system, is also responsible for the general bookkeeping and task launch duties associated with the execution of parallel programs such as OpenCL applications. The device is the element in the system on which the kernels of the application are executed. The device is further divided into a set of compute units. The number of compute units depends on the target hardware for a specific application. A compute unit is defined as the element in the hardware device onto which a work group of a kernel is executed. This device is responsible for executing the operations of the assigned work group to completion. In accordance to the OpenCL standard division of work groups into work items, a compute unit is further subdivided into processing elements. A processing element is the data path in the compute unit, which is responsible for executing the operations of one work item. A conceptual view of this model is shown in the following figure.

Figure 1: OpenCL Platform Model



X14980-090315

An OpenCL platform always starts with a host processor. For the case of platforms created with Xilinx® devices, the host processor is an x86 or Power8 based processor communicating to the devices using a PCIe™ solution. The host processor has the following responsibilities:

- Manage the operating system and enable drivers for all devices.
- Execute the application host program.
- Set up all global memory buffers and manage data transfer between the host and the device.
- Monitor the status of all compute units in the system.

In all OpenCL platforms, the host processor tasks are executed using a common set of API functions. The implementation of the OpenCL API functions is provided by the hardware vendor and is referred to as the runtime library.

The OpenCL runtime library, which is provided by the hardware vendor, is the logical layer in a platform that is responsible for translating user commands described by the OpenCL API into hardware specific commands for a given device. For example, when the application programmer allocates a memory buffer using the `clCreateBuffer` API call, it is the responsibility of the runtime library to keep track of where the allocated buffer physically resides in the system, and of the mechanism required for buffer access. It is important for the application programmer to keep in mind that the OpenCL API is portable across vendors, but the runtime library provided by a vendor is not. Therefore, OpenCL applications have to be linked at compile time with the runtime library that is paired with the target execution device.

The other component of a platform is the device. A device in the context of an OpenCL API is the physical collection of hardware resources onto which the application kernels are executed. A platform must have at least one device available for the execution of kernels. Also, per the OpenCL platform model, all devices in a platform do not have to be of identical type.

OpenCL Devices and FPGAs

In the context of CPU and GPU devices, the attributes of a device are fixed and the programmer has very little influence on what the device looks like. On the other hand, this characteristic of CPU/GPU systems makes it relatively easy to obtain an off-the-shelf board. The one major limitation of this style of device is that there is no direct connection between system I/O and the OpenCL™ kernels. All transactions of data are through memory-based transfers.

An OpenCL device for an FPGA is not limited by the constraints of a CPU/GPU device. By taking advantage of the fact that the FPGA starts off as a blank computational canvas, the user can decide the level of device customization that is appropriate to support a single application or a class of applications. In determining the level of customization in a device, the programmer needs to keep in mind that kernel compute units are not placed in isolation within the FPGA fabric.

FPGAs capable of supporting OpenCL programs consist of the following:

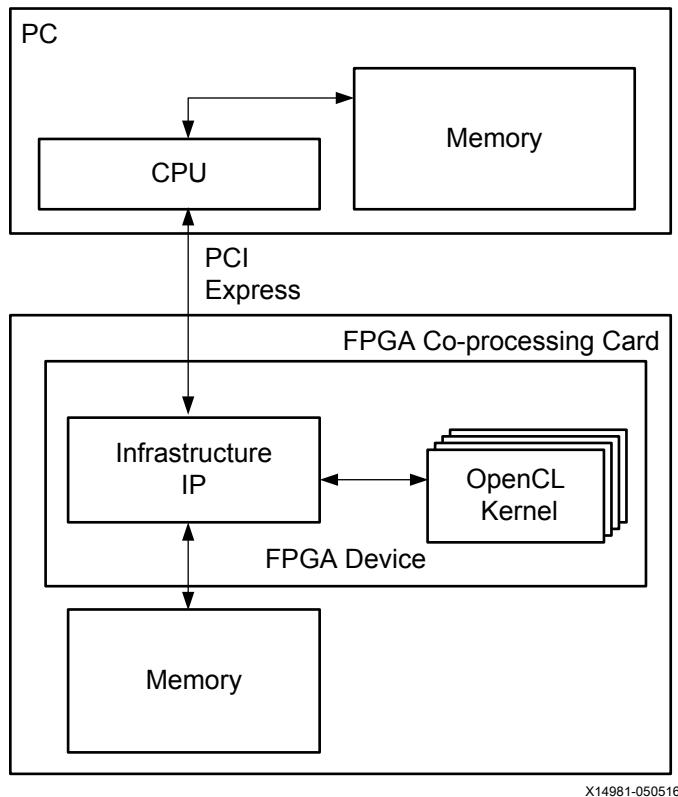
- Connection to the host processor
- I/O peripherals
- Memory controllers
- Interconnect
- Kernel region

The creation of FPGAs requires FPGA design knowledge and is beyond the scope of capabilities for the SDAccel™ development environment. Devices for the SDAccel environment are created using the Xilinx Vivado® Design Suite for FPGA designers. The SDAccel environment provides pre-defined devices and allows users to augment the tool with third party created devices. A methodology guide describing how to create a device for the SDAccel development environment is available upon request from [Xilinx](#).

The devices available in the SDAccel environment are for Virtex®-7, Kintex®-7, and UltraScale™ FPGAs. These devices are available in a PCIe form factor. The PCIe form factor for Virtex-7, Kintex-7, and UltraScale devices assumes that the host processor is an x86 or Power8 based processor and that the FPGA is used for the implementation of compute units.

Using a PCIe Reference Device

The PCIe™ base device has a distributed memory architecture, which is also found in GPU accelerated compute devices. This means that the host and the kernels access data from separate physical memory domains. Therefore, the developer has to be aware that passing buffers between the host and a device triggers memory data copies between the physical memories of the host and the device. The data transfer time must be accounted for when determining the best optimization strategy for a given application. A representative example of this type of device is shown in the following figure.

Figure 2: PCIe Base Device

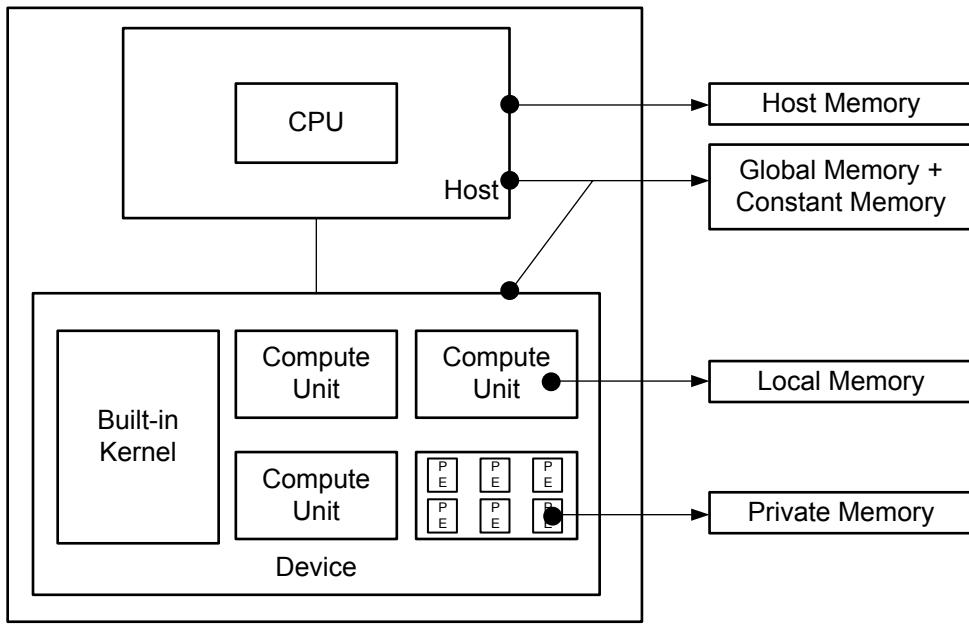
X14981-050516

The main characteristics of devices with a PCIe form factor are as follows:

- The x86 or Power8 processor in the PC is the host processor for the OpenCL™ application.
- The infrastructure IP provided as part of the device is needed for communication to the host over the PCIe core and to access the DDR memories on the board.
- Connecting OpenCL kernels to IP other than infrastructure IP or blocks generated by the SDAccel™ development environment is not supported.
- Kernels work on data in the DDR memory attached to the FPGA.

OpenCL Memory Model

The OpenCL™ API defines the memory model to be used by all applications that comply with the standard. This hierarchical representation of memory is common across all vendors and can be applied to any OpenCL application. The vendor is responsible for defining how the OpenCL memory model maps to specific hardware. The OpenCL memory model is shown overlaid onto the OpenCL device model in the following figure.

Figure 3: OpenCL Memory Model

X14982-090315

The memory hierarchy defined in the OpenCL specification has the following levels:

- Host Memory
- Global Memory
- Constant Global Memory
- Local Memory
- Private Memory

Host Memory

The host memory is defined as the region of system memory that is only visible and accessible to the host processor. The host processor has full control of this memory space and can read and write from this space without any restrictions. Kernels cannot access data located in this space. Any data needed by a kernel must be transferred into global memory so that it is accessible by a compute unit.

Global Memory

The global memory is defined as the region of system memory that is accessible to both the OpenCL™ host and device. The host is responsible for the allocation and deallocation of buffers in this memory space. There is a handshake between host and device over control of the data stored in this memory. The host processor transfers data from the host memory space into the global memory space. Then, when a kernel is launched to process the data, the host loses access rights to the buffer in global memory. The device takes over and is capable of reading and writing from the global memory until the kernel execution is complete. Upon completion of the operations associated with a kernel, the device turns control of the global memory buffer back to the host processor. Once it has regained control of a buffer, the host processor can read and write data to the buffer, transfer data back to the host memory, and deallocate the buffer.

Constant Global Memory

Constant global memory is defined as the region of system memory that is accessible with read and write access for the OpenCL™ host and with read-only access for the OpenCL device. As the name implies, the typical use for this memory is to transfer constant data needed by kernel computation from the host to the device.

Local Memory

Local memory is defined as the region of system memory that is only accessible to the OpenCL™ device. The host processor has no visibility and no control on the operations that occur in this memory space. This memory space allows read and write operations by the work items within the same compute unit. This level of memory is typically used to store and transfer data that must be shared by multiple work items.

Private Memory

Private memory is the region of system memory that is only accessible by a processing element within an OpenCL™ device. This memory space can be read from and written to by a single work item.

For devices using an FPGA, the physical mapping of the OpenCL memory model is the following:

- Host memory is any memory connected to the host processor only.
- Global and constant memories are any memory that is connected to the FPGA. These are usually memory chips that are physically connected to the FPGA. The host processor has access to these memory banks through infrastructure in the FPGA base device.
- Local memory is memory inside of the FPGA. This memory is typically implemented using block RAM elements in the FPGA fabric.
- Private memory is memory inside of the FPGA. This memory is typically implemented using registers in the FPGA fabric in order to minimize latency to the compute data path in the processing element.

OpenCL Installable Client Driver Loader

SDAccel™ supports the OpenCL™ ICD extension (`cl_khr_icd`). The OpenCL ICD Loader Library allows multiple implementations of OpenCL to co-exist on the same system. Applications may choose a platform from the list of installed platforms and hence dispatch OpenCL API calls to the correct underlying implementation.

Xilinx® does not provide the OpenCL ICD library and the following should be used to install the library on your preferred system:

On Ubuntu ICD the library is packaged with the distribution, install the following packages:

- `ocl-icd-libopencl1`
- `opencl-headers`
- `ocl-icd-opencl-dev`

For RHEL/CentOS 7.X use EPEL 7, install the following packages:

- `ocl-icd`
- `ocl-icd-devel`
- `opencl-headers`

For RHEL/CentOS 6.X the following procedure may be used:

1. Install prerequisites.

```
linux> yum install libICE libSM libXext libpng libXrender  
fontconfig numactl redhat-lsb-core
```

2. Install the OpenCL SDK RPMs.

```
linux> rpm -i /proj/picasso/centos/OpenCL-RPMS/*.rpm
```

3. Install `xilinx.icd`.

```
linux> cat <<EOF > /etc/OpenCL/vendors/xilinx.icd  
linux> libxilinxopencl.so  
linux> EOF
```

Recommended Libraries

Xilinx recommends that you install the following libraries on your operating system.

- Independent JPEG Group's JPEG runtime library (version 6.2)
- `%sudo apt-get install libjpeg62 libjpeg62-dev`

Xilinx recommends the following packages should be installed on CentOS 7.x

- PNG reference library.
- %sudo yum install libpng12
- The Linux Standards Base (LSB) library. The redhat-lsb package provides utilities needed for LSB Compliant Applications.
- %sudo yum install redhat-lsb
- The libtiff3 package, an older version of libtiff library for manipulating TIFF (Tagged Image File Format) image format files.
- %sudo yum install redhat-lsb

Kernel Language Support

The SDAccel™ environment supports kernels expressed in OpenCL™ C, C/C++ and RTL (Verilog or VHDL). You can use different kernel types in the same application.

Expressing a Kernel in RTL

A kernel can be implemented in RTL and developed using the Vivado® IDE tool suite. RTL kernels offer potentially higher performance with lower area and power, but require development using RTL coding, tools, and verification methodologies. Existing RTL based IP and algorithms can be wrapped and migrated to the SDAccel™ framework enabling those HDL based algorithms to be callable by the runtime and application program. RTL kernels must use the correct interfaces, protocols, and packaging to be recognized by the SDAccel tool flow and runtime library. The following section describes how to implement RTL kernels.

RTL kernels should be written, designed, and tested using the recommendations in the *UltraFast Design Methodology Guide for the Vivado Design Suite*, ([UG949](#)).

Interface Requirements

The following signals and interfaces are required on the top level of an RTL block.

- Clock.
- Active Low reset.
- 1 or more AXI4 memory mapped (MM) master interfaces for global memory. All AXI MM master interfaces must have 64-bit addresses.
 - You are responsible for partitioning global memory spaces. Each partition in the global memory becomes a kernel argument. The memory offset for each partition must be set by a control register programmable via the AXI4 MM Slave Lite interface.
- One and only one AXI4 MM slave lite I/F for control interface. The AXI Lite interface name must be S_AXI_CONTROL.
 - Offset 0 of the AXI4 MM slave lite must have the following signals:
 - Bit 0: start signal - The kernel starts processing data when this bit is set.
 - Bit 1: done signal - The kernel asserts this signal when the processing is done.
 - Bit 2: idle signal - The kernel asserts this signal when it is not processing any data.
- One or more AXI4-Stream interfaces for streaming data between kernels.

Programming Paradigm

Software Function Model

RTL kernels are modeled in software as functions with a void return value similar to the software interface model used in HLS based kernels. This means that RTL kernels can only be passed scalars for input arguments and memory pointer addresses for data to be exchanged with the host application.

In the host application, the RTL kernel can be invoked in a similar manner as HLS kernels with a function signature such as:

```
void mmult(int *a, int *b, int *output)  
  
void mmult(unsigned int length, int *a, int *b, int *output)
```

If the underlying logic requires a different control or software model then logic shall be added to fit this shape.

Interface Requirements for Integration Into the Platform

The RTL kernel integrates into a platform using a slave AXI4-Lite interface for control register access (to pass kernel arguments and to start/stop the kernel). The RTL kernel can also have AXI4 master interfaces to talk to memory.

The following signals and interfaces are required on the top level of an RTL block.

- Primary clock input port named `ap_clk`.
 - This clock is used by the AXI interfaces of the kernel.
 - There can be a secondary optional clock input named `ap_clk_2`.
- Primary active low reset input port named `ap_rst_n`.
 - This signal should be internally pipelined to improve timing.
 - This signal is driven by a synchronous reset in the `ap_clk` clock domain.
- There can be a secondary optional active low reset input `ap_rst_n_2`.
 - This signal should be internally pipelined to improve timing.
 - This signal is driven by a synchronous reset in the `ap_clk_2` clock domain.

- One and only one AXI4-Lite slave control interface.
 - Offset 0 of the AXI4-Lite slave interface must have the following signals:
 - Bit 0: start signal - The kernel starts processing data when this bit is set.
 - Bit 1: done signal - The kernel asserts this signal when the processing is done. This bit is clear on read.
 - Bit 2: idle signal - The kernel asserts this signal when it is not processing any data. The transition from low to high should occur synchronously with assertion of done signal.
 - The host typically writes to 0x00000001 to the offset 0 control register which sets Bit 0, clears Bits 1 and 2, and polls on reading done signal until it is a "1".
- 0 or more AXI4 memory mapped (MM) master interfaces for global memory.
 - All AXI4 MM master interfaces must have 64-bit addresses.
 - The kernel developer is responsible for partitioning global memory spaces. Each partition in the global memory becomes a kernel argument. The memory offset for each partition must be set by a control register programmable via the AXI4-Lite Slave interface.
 - AXI4 masters must not use Wrap or Fixed burst types, and must not use narrow (sub-size) bursts meaning AxSIZE should match the width of the AXI data bus.
- 0 or more AXI4-Stream interfaces for streaming data between kernels. AXI4-Stream interfaces can only be used to connect between kernels.
 - Interfaces must have TDATA, TREADY, and TVALID.
 - The TDATA width should match the width of the other kernel to be connected to.
- A kernel must have at least one AXI4 MM interface or at least one AXI4-Stream Interface

Any user logic or RTL code that does not conform to the requirements above, must be wrapped or bridged to satisfy these requirements.

SDAccel Tool Flow for RTL Kernels

There are three steps to packaging an RTL block as an RTL kernel for SDAccel™ applications:

1. Package the RTL block as Vivado® IP.
2. Create a kernel description XML file.
3. Package the RTL kernel into a Xilinx Object (XO) file.

A fully packaged RTL Kernel is delivered as an XO file which has a file extension of .xo. This file is a container encapsulating the Vivado IP object (including source files) and kernel XML file. The XO file can be compiled into the platform and run in hardware, or hardware emulation flows.

Packaging RTL block as a Vivado IP

RTL Kernels must be packaged as a Vivado® IP suitable for use in IP Integrator. See *Vivado Design Suite User Guide: Creating and Packaging Custom IP* ([UG1118](#)) for details on IP packaging in Vivado.

The following interface packaging is required for the RTL Kernel:

- The AXI4-Lite interface name must be packaged as `S_AXI_CONTROL`, but the underlying AXI ports can be named differently.
- The AXI4 MM interfaces must be packaged as AXI4 master endpoints with 64 bit address support.
 - Xilinx strongly recommends that AXI4 MM interfaces be packaged with AXI meta data `HAS_BURST=0` and `SUPPORTS_NARROW_BURST=0`. These properties can be set in an IP level `bd.tcl` file. This indicates wrap and fixed burst type is not used and narrow (sub-size burst) is not used.
- `ap_clk` and `ap_clk_2` must be packaged as clock interfaces.
- `ap_rst_n` and `ap_rst_n_2` must be packaged as active low reset interfaces.
- `ap_clk` must be packaged to be associated with all AXI4-Lite, AXI4 MM, and AXI4-Stream interfaces.

To test if the RTL kernel is packaged correctly for IP Integrator, try to instantiate the packaged kernel in IP Integrator. In the GUI it should show up as having interfaces for clock, reset, AXI4-Lite slave, AXI4 MM master, and AXI4-Slave only. No other ports should be present in the canvas view. The properties of the AXI interface can be viewed by selecting the interface on the canvas. Then in the **Block Interface Properties** window, select the **Properties** tab and expand the `CONFIG` table entry. If an interface is to be read-only or write-only then the unused AXI channels can be removed and the `READ_WRITE_MODE` will be set to read-only or write-only.

Create Kernel Description XML File

A kernel description XML file needs to be manually created for the RTL IP to be used as an RTL kernel in SDAccel environment. The following is an example of the kernel XML file:

```
<?xml version="1.0" encoding="UTF-8"?>
<root versionMajor="1" versionMinor="0">
<kernel name="input_stage" language="ip"
vlnv="xilinx.com:hls:input_stage:1.0" attributes=""
preferredWorkGroupSizeMultiple="0" workGroupSize="1">
<ports>
<port name="M_AXI_GMEM" mode="master" range="0x3FFFFFFF" dataWidth="32"
portType="addressable" base="0x0"/>
<port name="S_AXI_CONTROL" mode="slave" range="0x1000" dataWidth="32"
portType="addressable" base="0x0"/>
<port name="AXIS_P0" mode="write_only" dataWidth="32" portType="stream"/>
</ports>
<args>
<arg name="input" addressQualifier="1" id="0" port="M_AXI_GMEM"
size="0x4" offset="0x10" hostOffset="0x0" hostSize="0x4" type="int*" />
<arg name="__xcl_gv_p0" addressQualifier="4" id="" port="AXIS_P0"
size="0x4" offset="0x18" hostOffset="0x0" hostSize="0x4" type=""
memSize="0x800"/>
</args>
</kernel>
<pipe name="xcl_pipe_p0" width="0x4" depth="0x200" linkage="internal"/>
<connection srcInst="input_stage" srcPort="p0" dstInst="xcl_pipe_p0"
```

```
dstPort="S_AXIS"/>
</root>
```

The following table describes the format of the kernel XML in detail:

Table 1: Kernel XML Format

Tag	Attribute	Description
<root>	versionMajor	Set to 1 for the current release of SDAccel
	versionMinor	Set to 0 for the current release of SDAccel
<kernel>	name	Kernel name
	language	Always set it to "ip" for RTL kernels
	vlnv	Must match the vendor, library, name, and version attributes in the component.xml of an IP. For example, If component.xml has the following tags: <spirit:vendor>xilinx.com</spirit:vendor> <spirit:library>hls</spirit:library> <spirit:name>test_sincos</spirit:name> <spirit:version>1.0</spirit:version> the vlnv attribute in kernel XML will need to be set to: xilinx.com:hls:test_sincos:1.0
	attributes	Reserved. Set it to empty string.
	preferredWorkGroupSizeMultiple	Reserved. Set it to 0.
	workGroupSize	Reserved. Set it to 1.
	name	Port name. At least an AXI4 master port and an AXI4-Lite slave port are required. AXI stream port can be optionally specified to stream data between kernels. The AXI4-Lite interface name must be S_AXI_CONTROL.
	mode	<ul style="list-style-type: none"> For AXI-4master port, set it to "master". For AXI-4 slave port, set it to "slave". For AXI Stream master port, set it to "write_only". For AXI Stream slave port, set it "read_only".
<port>	range	The range of the address space for the port.
	dataWidth	The width of the data that goes through the port, default is 32 bits.

Tag	Attribute	Description
	portType	Indicate whether or not the port is addressable or streaming. <ul style="list-style-type: none">• For AXI4 master and slave ports, set it to "addressable".• For AXI4-Stream ports, set it to "stream".
	base	For AXI4 master and slave ports, set to "0x0". This tag is not applicable to AXI4-Stream ports.
<arg>	name	Kernel argument name.
	addressQualifier	Valid values: 0: Scalar kernel input argument 1: global memory 2: local memory 3: constant memory 4: pipe
	id	Only applicable for AXI4 master and slave ports. The ID needs to be sequential. It is used to determine the order of kernel arguments. Not applicable for AXI4-Stream ports.
	port	Indicates the port that the arg is connected to.
	size	Size of the argument. The default is 4 bytes.
	offset	Indicates the register memory address.
	type	The C data type for the argument. E.g. int*, float*
	hostOffset	Reserved. Set to 0x0.
	hostSize	Size of the argument. The default is 4 bytes.
	memSize	Not applicable to AXI4 master and slave ports. For AXI4-Stream ports, memSize sets the depth of the FIFO created for the AXI stream ports.

The following tags specify additional information for AXI4-Stream ports. They are not applicable to AXI4 master or slave ports.

<pipe>	For each pipe in the compute unit, the compiler inserts a FIFO for buffering the data. The pipe tag describes configuration of the FIFO.
name	This specifies the name for the FIFO inserted for the AXI4-Stream port. This name must be unique among all pipes used in the same compute unit.
width	This specifies the width of FIFO in bytes. For example, 0x4 for 32-bit FIFO.
depth	This specifies the depth of the FIFO in number of words.
linkage	Always set to "internal".

Tag	Attribute	Description
<connection>	The connection tag describes the actual connection in hardware either from the kernel to the FIFO inserted for the PIPE or from the FIFO to the kernel.	
	srcInst	Specifies the source instance of the connection.
	srcPort	Specifies the port on the source instance for the connection.
	dstInst	Specifies the destination instance of the connection.
	dstPort	Specifies the port on the destination instance of the connection.

Package RTL Kernel into Xilinx Object File

The final step is to package the RTL IP and the kernel XML together into a Xilinx object file (.xo) so it can be used by the SDAccel compiler. The following is the command example in Vivado 2017.1_sda. The final RTL kernel is in the test.xo file.

```
package_xo -xo_path test.xo -kernel_name test_sincos -kernel_xml
kernel.xml -ip_directory ./ip/
```

RTL Kernel Wizard

The RTL Kernel Wizard walks you through the process of specifying your desired software function model and interface model for their RTL Kernel. The RTL Kernel Wizard generates an example project design and a set of scripts to build that example design into an XO file. This example design can then serve as a framework for the user to integrate new or existing RTL code into. The example project is a normal Vivado® project with sample RTL that conforms to the interface requirements of RTL Kernels. The provided scripts package the RTL into a Vivado IP with the necessary interface definitions and meta data. A `kernel.xml` file is also generated to match the software function prototype and behavior specified in the wizard. The benefit of the wizard is that allowable selections serve to guide you to understand the supported features of an RTL kernel.

Launching RTL Kernel Wizard

The RTL Kernel Wizard can be launched with two different methods: from the SDx™ Development Environment or from the Vivado® IDE. The SDx Development Environment provides a more seamless experience, but IP management is limited. The Vivado IDE is recommended if multiple kernels are going to be generated and allows for better re-entrant workflows.

Launching the RTL Kernel Wizard from the SDx Development Environment:

1. Launch SDx Development Environment.

2. Create Project (SDAccel Product Type).
3. Click **Xilinx > Create RTL Kernel....**

Launching the RTL Kernel Wizard from Vivado IDE:

1. Create a new Vivado project choosing the same device as exists on the platform you intend to target. If you do not know your target device, choose the default part.
2. Go to the IP catalog by clicking on the **IP Catalog** button.
3. Type `wizard` in the IP Catalog search box.
4. Double-click **SDx Kernel Wizard** to launch the wizard.

NOTE: Use Vivado from the SDx install so the tool versions are the same.

RTL Kernel Wizard Settings

The wizard is organized into pages that break down the process of creating a kernel into smaller steps. To navigate between pages, use the **Next** and **Back** buttons. To finalize the kernel and build a project based on the wizards inputs, click the **OK** button. Each of the following sections describes each page and its input options.

Welcome to SDx Kernel Wizard

This page gives an abbreviated version of the workflow and the steps for following the wizard.

RTL Kernel Wizard General Settings

Kernel Identification

- **Kernel name** -The kernel name. This will be the name of the IP, top level module name, kernel, and C function call. It should be unique. Must conform to C and Verilog identifier naming rules. It must also conform to Vivado® IP Integrator naming rules, which prohibits underscores except when placed in between alphanumeric characters.
- **Kernel vendor** - The name of the vendor. Used in the VLNV.
- **Kernel library** - The name of the library. Used in the VLNV. Must conform to same identifier rules **Kernel name**.

Clocking Options

Number of clocks - Sets the number of clocks used by the Kernel. Every kernel has a primary clock called `ap_clk` that all input/output ports on the kernel are synchronous to. When **Number of clocks** is set to **2**, a secondary clock and related reset are provided to be used by the kernel internally. The secondary clock and reset are called `ap_clk_2` and `ap_rst_n_2`, respectively. This secondary clock supports independent frequency scaling and is independent from the primary clock. The secondary clock is useful if the kernel clock needs to run at a faster/slower rate than the AXI4 interface, which must be clocked on the primary clock. When designing with multiple clocks, proper clock domain crossing techniques must be used to ensure data integrity across all clock frequency scenarios.

Scalars Arguments

Scalar arguments are used to pass control type of information to the kernels. Scalar arguments can not be read back from the host. For each argument that is specified a corresponding control register is created to facilitate passing the argument from software to hardware.

- **Number of scalar kernel input arguments** - Specifies the number of scalar input arguments to pass to the kernel. For each of the number specified, a table row is generated that allows customization of the argument name and argument type. There is no required minimum number of scalars and the maximum is limited to 16.

Scalar Input Argument Definition

- **Argument name** - The argument name is used in the generated Verilog control register module as an output signal. Each argument is assigned an ID value. This ID value is used to access the argument from the host software. The ID value assignments can be found on the summary page of this wizard. To ensure maximum compatibility, the argument name follows the same identifier rules as the Kernel name.
- **Argument type** - Specifies the the data type of the argument. This will affect the width of the control register in the generated Verilog module. The data types available are limited to the ones specified by the OpenCL version 2.0 specification. Data types that represent a bit width greater than 32 bits will require two write operations to the control registers.

Global Memory

Global memory is accessed by the kernel through AXI4 master interfaces. Each AXI4 interface operates independently of each other. Each AXI4 interface may be connected to one or more memory controllers to off chip memory such as DDR4. Global memory is primarily used to pass large data sets to and from the kernel from the host. It can also be used to pass data between kernels. See the [Memory Performance Optimizations for AXI4 Interface](#) section for recommendations on how to design these interfaces for optimal performance. For each interface, example AXI master logic is generated in the RTL kernel to provide a starting point and can be discarded if not used.

- **Number of AXI master interfaces** - Specifies the number of AXI master interfaces present on the Kernel. A maximum of 16 interfaces may be specified. For each interface, an interface name, data width, and number of associated arguments can be customized. Each interface contains all read/write channels.

AXI Master Definition (table columns)

Interface Name - Specifies the name of the interface. To ensure maximum compatibility, the argument name follows the same identifier rules as the Kernel name.

Width (bytes) - Specifies the data width of the AXI data channels. It is recommended this is matched to the native data width of the memory controller AXI4 slave interface. The memory controller slave interface is typically 64 bytes (512 bits) wide.

Number of Arguments - Specifies the number of arguments to associate with this interface. Each argument represents a data pointer to global memory that the kernel can access.

Argument Definition

Interface - Specifies the name of the AXI Interface that the corresponding columns in the current row are associated to. This value is not directly modifiable, it is copied from the interface name defined in the previous table.

Argument name - The argument name is used in the generated Verilog control register module as an output signal. Each argument is assigned an ID value. This ID value is used to access the argument from the host software. The ID value assignments can be found on the summary page of this wizard. To ensure maximum compatibility, the argument name follows the same identifier rules as the Kernel name.

Summary

This page gives a summary of the VLN, software function prototype and hardware control registers created from options selected in the previous pages. The function prototype conveys what a kernel call would like if it was a C function. See the host code generated example of how to set the kernel arguments for the kernel call. The register map shows the relationship between host software ID, argument name, hardware register offset, type, and associated interface. Review this page to for correctness before proceeding to generating the kernel.

Finalizing and Generating the Kernel from the RTL Wizard

If the RTL Kernel Wizard was launched from SDx™, after clicking **OK**, the example Vivado® project opens.

If the RTL Kernel Wizard was launched from Vivado, after clicking **OK** do the following:

1. When the **Generate Output Products** window appears, select **Global** synthesis options and click **Generate**, then **OK**.
2. Right-click on the `.xci` file in the **Design Sources View**, and select **Open IP Example Design....**
3. In the open example design window, select an output directory (or accept default) and click **OK**.
4. This opens a new Vivado® project with the example design in it. You can now close the current Vivado project that the RTL Kernel Wizard was invoked from.

XO Generation Scripts and Generated Files

In the example design project, review the HDL files in the design sources view and review the generated `kernel.xml` file.

To package the RTL code as a Vivado® IP and generate the XO:

1. In the Vivado IDE, click the **Generate RTL Kernel** button located at the top of the screen. Alternatively, in the **Tcl Console**, type `source imports/package_kernel.tcl` and review the **Tcl Console** log files. Notice that a package kernel XO file is now created in the `sdx_imports` directory along with host file called `main.c`.
2. The generated XO file and `main.c` host code can be imported into SDx or used with `xocc` to run the hardware emulation or HW (system) workflows.. Note that host code needs to be written to invoke the kernel.

After running and reviewing the example, the user can modify the design to add their own RTL code and adapt it to the top level module definition of the example project. After any RTL changes use the Vivado tools to test synthesis and run DRCs. Each time the source code is changed, rerun the `package_kernel.tcl` script.

Designing RTL Recommendations

Memory Performance Optimizations for AXI4 Interface

The AXI4 MM interfaces typically connect to DDR memory controllers in the platform. For optimal frequency and resource usage it is recommended that one interface is used per memory controller. For best performance from the memory controller, the following is the recommended AXI interface behavior:

1. Use an AXI data width that matches the native memory controller AXI data width, typically 512 bits.

2. Do not use WRAP, FIXED, or sub-sized bursts.
3. Use burst transfer as large as possible (up to 4KByte AXI4 protocol limit).
4. Avoid use of de-asserted write strobes. De-asserted write strobes can cause ECC logic in the DDR memory controller to perform read-modify-write operations.
5. Use pipelined AXI transactions
6. Avoid using threads if an AXI interface is only connected to one DDR controller.
7. Avoid generating write address commands if the kernel does not have the ability to deliver the full write transaction (non-blocking write requests).
8. Avoid generating read address commands if the kernel does not have the capacity to accept all the read data without backpressure (non-blocking read requests).
9. If a read-only or write-only interfaces are desired, then the ports of the unused channels can be commented out in the top level RTL file before the project is packaged into a kernel.
10. Using multiple threads can cause larger resource requirements in the infrastructure IP between the kernel and the memory controllers.

Quality of Results Considerations

The following recommendations help improve results for timing and area:

1. Pipeline all reset inputs and internally distribute resets avoiding high fanout nets.
2. Reset only essential control logic FFs.
3. Consider registering input and output signals to the extent possible.
4. Understand the size of the kernel relative to the capacity of the target platforms to ensure fit, especially if multiple kernels will be instantiated.
5. Recognize platforms that use Stack Silicon Interconnect (SSI) Technology. These devices have multiple dice and any logic that must cross between them should be Flip Flop (FF) to FF timing paths.

Debug and Verification Considerations

1. RTL kernels should be verified in their own test bench using advanced verification techniques including Verification components, randomization, and protocol checkers. The AXI Verification IP (AXI VIP) is available in the Vivado® IP catalog and can help with verification of AXI interfaces. The RTL kernel example designs contain an AXI VIP based test bench with sample stimulus files.
2. The hardware emulation flow should not be used for functional verification because it does not accurately represent the range of possible protocol signalling conditions that real AXI traffic in hardware may incur. Hardware emulation should be used to test the host code software integration or to view the interaction between multiple kernels.

Using RTL Kernels

RTL kernels can be created in an SDAccel™ application by using the commands below. Note the type for `create_kernel` command is `ip` and the file name for kernel source is the `.xo` file generated from the Package RTL Kernel step. RTL kernels are not supported in CPU emulation flow. They are supported in hardware emulation flow and build system flow.

```
create_kernel test_sincos -type ip add_files -kernel
[get_kernels test_sincos] "test.xo"
```

Expressing a Kernel in OpenCL C

The SDAccel™ environment supports the OpenCL™ language constructs and built in functions from the OpenCL 1.0 embedded profile. The following is an example of an OpenCL kernel for matrix multiplication that can be compiled with the SDAccel environment.

```
kernel __attribute__ ((reqd_work_group_size(16,16,1)))
void mult(__global int* a, __global int* b, __global int* output)
{
    int r = get_local_id(0);
    int c = get_local_id(1);
    int rank = get_local_size(0);
    int running = 0;
    for(int index = 0; index < 16; index++) {
        int aIndex = r*rank + index;
        int bIndex = index*rank + c;
        running += a[aIndex] * b[bIndex];
    }
    output[r*rank + c] = running;
    return;
}
```



IMPORTANT: Standard C libraries such as `math.h` can not be used in OpenCL C kernel. Use OpenCL built-in C functions instead.

Expressing a Kernel in C/C++

The kernel for matrix multiplication can be expressed in C/C++ code that can be synthesized by the Vivado® HLS tool. For kernels captured in this way, the SDAccel™ development environment supports all of the optimization techniques available in Vivado HLS. The only thing that the user has to keep in mind is that expressing kernels in this way requires compliance with a specific function signature style.

```
void mmult(int *a, int *b, int *output)
{
```

```

#pragma HLS INTERFACE m_axi port=a offset=slave bundle=gmem
#pragma HLS INTERFACE m_axi port=b offset=slave bundle=gmem
#pragma HLS INTERFACE m_axi port=output offset=slave bundle=gmem
#pragma HLS INTERFACE s_axilite port=a bundle=control
#pragma HLS INTERFACE s_axilite port=b bundle=control
#pragma HLS INTERFACE s_axilite port=output bundle=control
#pragma HLS INTERFACE s_axilite port=return bundle=control

const int rank = 16;
int running = 0;
int bufa[256];
int bufb[256];
int bufc[256];
memcpy(bufa, (int *) a, 256*4);
memcpy(bufb, (int *) b, 256*4);

for (unsigned int c=0;c<rank;c++) {
    for (unsigned int r=0;r<rank;r++) {
        running=0;
        for (int index=0; index<rank; index++) {
#pragma HLS pipeline
            int aIndex = r*rank + index;
            int bIndex = index*rank + c;
            running += bufa[aIndex] * bufb[bIndex];
        }
        bufc[r*rank + c] = running;
    }
}

memcpy((int *) output, bufc, 256*4);
return;
}

```

The preceding code example is the matrix multiplication kernel expressed in C/C++ for Vivado HLS. The first thing to notice about this code is the function signature.

```
void mmult(int *a, int *b, int *output)
```

This function signature is almost identical to the signature of the kernel expressed in OpenCL C. It is important to keep in mind that by default, kernels captured in C/C++ for HLS do not have any inherent assumptions on the physical interfaces that will be used to transport the function parameter data. HLS uses pragmas embedded in the code to direct the compiler as to which physical interface to generate for a function port. For the function to be treated as a valid OpenCL kernel, the ports on the C/C++ function must be reflected on the memory and control interface pragmas for HLS.

The memory interface specification is

```
#pragma HLS INTERFACE m_axi port=<variable name> offset=slave
bundle=<interface name>
```

A separate AXI4 master interface in the name of `M_AXI_<interface name>` is created for each unique bundle. For example, if `bundle` is set to `bank0`, the name for the AXI4 interface is set to `M_AXI_BANK0`. This interface name is required for some advanced options like `map_connect` to make particular physical connections in the FPGA design.

The control interface specification is

```
#pragma HLS INTERFACE s_axilite port=<variable name> bundle=<interface name>
```

Detailed information on how these pragmas are used is available in the *Vivado Design Suite User Guide: High-Level Synthesis* ([UG902](#)).



IMPORTANT: *Global variables in HLS C/C++ kernels are not supported.*

NOTE: *C++ arbitrary precision data types can be used for global memory pointers on a kernel. They are not supported for scalar kernel inputs that are passed by value.*

Compilation Flow

The Xilinx® SDAccel™ development environment is used for creating and compiling OpenCL™ applications onto a Xilinx® FPGA. This tool suite provides a software development environment for algorithm development and emulation on x86 based workstations, as well as deployment mechanisms for Xilinx FPGAs.

The compilation of OpenCL applications into binaries for execution on an FPGA does not assume nor require FPGA design knowledge. A basic understanding of the capabilities of an FPGA is necessary during application optimization in order to maximize performance. The SDAccel environment handles the low-level details of program compilation and optimization during the generation of application specific compute units for an FPGA fabric. Therefore, using the SDAccel environment to compile an OpenCL program does not place any additional requirements on the user beyond what is expected for compilation towards a CPU or GPU target.

An OpenCL program can be compiled using standalone command line compilers(`xcpp` for host code and `xocc` for kernels) in the SDAccel development environments. The compilation flow is described in detail in this chapter.



IMPORTANT: Starting with the SDAccel 2016.3 release, Tcl-based access is no longer supported. Direct command line access via XOCC (and MakeFile) is the main entry point to use SDAccel services. You must convert all Tcl scripts to the MakeFile based setup which will call XOcc directly or the scripts will no longer function. Refer to examples in the SDAccel installation on usages of Makefile/XOCC for compiling SDAccel applications. See [Converting Tcl Compilation Flow to XOcc](#) for more information.

Xilinx OpenCL Compiler

Creating the Xilinx OpenCL Compute Unit Binary Container

The main difference between targeting an OpenCL™ application to a CPU/GPU and targeting an FPGA is the source of the compiled kernel. Both CPUs and GPUs have a fixed computing architecture onto which the kernel code is mapped by a compiler. Therefore, OpenCL programs targeted for either kind of device invokes just-in-time compilation of kernel source files from the host code. The API for invoking just-in-time kernel compilation is as follows:

```
clCreateProgramWithSource (...)
```



IMPORTANT: `ClCreateProgramWithSource` function is not supported for kernels targeting FPGA.

In contrast to a CPU or a GPU, an FPGA can be thought of as a blank computing canvas onto which a compiler generates an optimized computing architecture for each kernel in the system. This inherent flexibility of the FPGA allows the developer to explore different kernel optimizations and compute unit combinations that are beyond what is possible with a fixed architecture. The only drawback to this flexibility is that the generation of a kernel specific optimized compute architecture takes a longer time than what is acceptable for just-in-time compilation. The OpenCL standard addresses this fundamental difference between devices by allowing for an offline compilation flow. This allows the user to generate libraries of kernels that can be loaded and executed by the host program. The OpenCL APIs for supporting kernels generated in an offline compilation flow are as follows:

```
// An offline binary container may be loaded with
clCreateProgramWithBinary()
cl_program p = clCreateProgramWithBinary(binary1);
clBuildProgram(p);

// Use the program and run kernels...

// Release the binary before the next call to clCreateProgramWithBinary()
clReleaseProgram(p);

// A second binary may then be loaded
p = clCreateProgramWithBinary(binary2);
clBuildProgram(p);

// Use the program and run kernels...

clReleaseProgram(p);
```

The SDAccel™ development environment uses an offline compilation flow to generate kernel binaries. To maximize efficiency in the host program and allow the simultaneous instantiation of kernels that cooperate in the computation of a portion of an application, Xilinx has defined the Xilinx® OpenCL Compute Unit Binary format .xclbin. The xclbin file is a binary library of kernel compute units that will be loaded together into an OpenCL context for a specific device. This format can hold either programming files for the FPGA fabric or shared libraries for the processor. It also contains library descriptive metadata, which is used by the Xilinx OpenCL runtime library during program execution.

The library metadata included in the xclbin file is automatically generated by the SDAccel environment and does not require user intervention. This data is composed of compute unit descriptive information that is automatically generated during compute unit synthesis and used by the runtime to understand the contents of an xclbin file.

The xclbin file is created using the SDx IDE. See *SDAccel Environment Tutorial: Introduction* ([UG1021](#)) or the Xilinx OpenCL Compiler (xocc) command line utility discussed in this chapter. It provides a mechanism for command line users to compile their kernels, which is ideal for compiling host applications and kernels using a makefile.

The first step in building any system is to select an acceleration device supported by Xilinx and third-party platform providers.

```
xocc --platform <arg> <input_file>
```

A complete list of supported devices is included in the *SDx Environments Release Notes, Installation, and Licensing Guide*, ([UG1238](#)).

Building the System

Compilation of an application for execution on a Xilinx™ enabled OpenCL™ device through the SDAccel™ development environment is performed by selecting the target. This step goes beyond compilation of host and kernel code and is also responsible for the generation of custom compute units for all of the binary containers in the solution.

The following is the command for this step in the flow:

```
xocc --target sw_emu|hw_emu|hw ...
```

The recommended flow is:

1. Perform software emulation (sw_emu) to confirm the functionality.
2. Perform hardware emulation (hw_emu) to create custom hardware and review the performance of the kernel.
3. Perform a build of the hardware (hw) system to implement the custom hardware.

The compilation method used in each step is dependent on the user selected kernel execution target. The `xocc --target` option invokes different flows for kernels targeted at a processor and kernels targeted at the FPGA fabric.

Using CPU Emulation

In the context of the SDAccel™ development environment, application emulation on a CPU is the same as the iterative development process that is typical of CPU/GPU programming. In this type of development style, a programmer continuously compiles and runs an application as it is being developed.

The main goal of CPU emulation is to ensure functional correctness and to partition the application into kernels. Although partitioning and optimizing an application into kernels is integral to OpenCL™ development, performance is not the main goal at this stage of application development in the SDAccel environment.

For CPU-based emulation, both the host code and the kernel code are compiled to run on an x86 processor. The programmer model of iterative algorithm refinement through fast compile and run loops is preserved with speeds that are the same as a CPU compile and run cycle.

Using Hardware Emulation

The SDAccel™ development environment generates at least one custom compute unit for each kernel in an application. This means that while the CPU emulation flow is a good measure of functional correctness, it does not guarantee correctness on the FPGA execution target. Before deployment, the application programmer should also check that the custom compute units generated by the tool are producing the correct results.

The SDAccel environment has a hardware emulation flow, which enables the programmer to check the correctness of the logic generated for the custom compute units. This emulation flow invokes the hardware simulator in the SDAccel environment to test the functionality of the logic that will be executed on the FPGA compute fabric.

Using the Build Flow for Compute Units Targeting the FPGA Fabric

The SDAccel™ development environment generates custom logic for every compute unit in the binary container. Therefore, it is normal for this build step to run for a longer period of time than the other steps in the SDAccel application compilation flow.

The steps in compiling compute units targeting the FPGA fabric are as follows:

1. Generate a custom compute unit for a specific kernel.
2. Instantiate the compute units in the OpenCL™ binary container.
3. Connect the compute units to memory and infrastructure elements of the target device.
4. Generate the FPGA programming file.

The generation of custom compute units for any given kernel code uses the production proven capabilities of the Xilinx® Vivado® High-Level Synthesis (HLS) tool, which is the compute unit generator in the SDAccel environment. Based on the characteristics of the target device in the solution, the SDAccel environment invokes the compute unit compiler to generate custom logic that maximizes performance while at the same time minimizing compute resource consumption on the FPGA fabric. Automatic optimization of a compute unit for maximum performance is not possible for all coding styles without additional user input to the compiler. [Kernel Optimization](#) discusses the additional user input that can be provided to the SDAccel environment to optimize the implementation of kernel operations into a custom compute unit.

After all compute units have been generated, these units are connected to the infrastructure elements provided by the target device in the solution. The infrastructure elements in a device are all of the memory, control, and I/O data planes which the device developer has defined to support an OpenCL application. The SDAccel environment combines the custom compute units and the base device infrastructure to generate an FPGA binary which is used to program the Xilinx device during application execution.



IMPORTANT: *The SDAccel environment always generates a valid FPGA hardware design, but does not generate an optimal allocation of the available bandwidth in the control and memory data planes. The user can manually optimize the data bandwidth usage by selecting connection points into the memory and control data planes per compute unit.*

Compiling Your OpenCL Kernel Using the Xilinx OpenCL Compiler (xocc)

The Xilinx® OpenCL™ Compiler (xocc) is a standalone command line utility for compiling an OpenCL kernel supporting all flows in the SDAccel™ environment. It provides a mechanism for command line users to compile their kernels, which is ideal for compiling host applications and kernels using a makefile.

Following are details of xocc command line format and options.

Syntax:

```
xocc [options] <input_file>
```

Table 2: XOCC Options

Option	Valid Values	Description
--platform <arg>	Supported acceleration platforms by Xilinx and third-party board partners	Required Set target Xilinx device. See <i>SDx Environments Release Notes, Installation, and Licensing Guide (UG1238)</i> for all supported devices.
--list_xdevices	N/A	Lists the supported devices.
--target <arg>	[sw_emu hw_emu hw]	Specify a compile target. <ul style="list-style-type: none"> • sw_emu: CPU emulation • hw_emu: Hardware emulation • hw: Hardware Default: hw NOTE: Without the -c or -l option, xocc is run in build mode, an .xclbin file is generated.
--compile	N/A	Optional Run xocc in compile mode, generate .xo file.
--link	N/A	Optional Run xocc in link mode, link .xo input files, generate .xclbin file.
--kernel <arg>	Kernel to be compiled from the input .cl or .c/.cpp kernel source code	Required for C/C++ kernels Optional for OpenCL kernels Compile/build only the specified kernel from the input file. Only one -k option is allowed per command.

Option	Valid Values	Description
		NOTE: When an OpenCL kernel is compiled without the -k option, all the kernels in the input file are compiled.
--output <arg>	File name with .xo or .xclbin extension depending on mode	Optional Set output file name. Default: a.xo for compile mode a.xclbin for link and build mode
--version	N/A	Prints the version and build information.
--help	N/A	Print help.
--define <arg>	Valid macro name and definition pair <i><name>=<definition></i>	Predefine name as a macro with definition. This option is passed to the openCL preprocessor.
--include <arg>	Directory name that includes required header files	Add the directory to the list of directories to be searched for header files. This option is passed to the SDAccel compiler preprocessor.
--kernel_frequency	Frequency (MHz) of the kernel.	Sets a user defined clock frequency in MHz for a the kernel overriding a default value from the DSA.
--nk <arg>	<p style="background-color: #f0f0f0; padding: 5px;"> <i><kernel_name>: <compute_units></i> </p> <p>(for example, foo:2)</p>	N/A in compile mode Optional in link mode Instantiate the specified number of compute units for the given kernel in the .xclbin file. Default: One compute unit per kernel.
--pk <arg>	[kernel_name all] : [none stream pipe memory]	Optional Set a stall profile type for the given kernel(s) Default: none

Option	Valid Values	Description
--max_memory_ports <arg>	[all <kernel_name>]	<p>Optional</p> <p>Set the maximum memory port property for all kernels or a given kernel.</p>
--memory_port_data_width <arg>	[all <kernel_name>]:<width>	<p>Set the specified memory port data width for all kernels or a given kernel. Valid width values are 32, 64, 128, 256, and 512.</p>
--optimize<arg>	<p>Valid optimization levels: 0, 1, 2, 3, s, quick</p> <p>example: --optimize2</p>	<p>These options control the default optimizations performed by the Vivado® hardware synthesis engine.</p> <p>NOTE: Familiarity with the Vivado tool suite is recommended in order to make the most use of these settings.</p> <ul style="list-style-type: none"> • 0: Default optimization. Reduce compilation time and make debugging produce the expected results. • 1: Optimize to reduce power consumption. This takes more time to compile the design. • 2: Optimize to increase kernel speed. This option increases both compilation time and the performance of the generated code. • 3: This is the highest level of optimization. This option provides the highest level performance in the generated code, but compilation time may increase considerably. • s: Optimize for size. This reduces the logic resources for the kernel

Option	Valid Values	Description
		<ul style="list-style-type: none"> • quick: Quick compilation for fast run time. This may result in reduced performance and a greater use of resources in the hardware implementation.
--xp	Refer to the following table, XP Parameters.	Specify detailed parameter and property settings in the Vivado tool suite used to implement the FPGA hardware. NOTE: Familiarity with the Vivado tool suite is recommended in order to make the most use of these parameters.
--debug	N/A	Generate code for debugging.
--log	N/A	Creates a log for in the current working directory.
--message-rules <arg>	Message rule file name	Optional - Specify a message rule file with message controlling rules. See <i>Using the Message Rule File</i> chapter for more details.
--report <arg>	Generate [estimate system] reports	Generate a report type specified by <arg>. estimate: Generate estimate report in <code>report_estimate.xtext</code> system: Generate the estimate report and detailed hardware reports in report directory
--save-temp	N/A	Save intermediate files/directories created during the compilation and build process.
--report_dir <arg>	Directory	Specify a report directory. If the --report option is specified, the default is to generate all reports in the current working directory (cwd).

Option	Valid Values	Description
--log_dir <arg>	Directory	Specify a log directory. If the --log option is specified, the default is to generate the log file in the current working directory (cwd).
--temp_dir <arg>	Directory	Specify a log directory. If the --save-temp option is specified, the default is to create the temporary compilation and build files in the current working directory (cwd).
--export_script	N/A	<p>This option allows detailed control of the Vivado tool suite used to implement the FPGA hardware.</p> <p>NOTE: Familiarity with the Vivado tool suite is recommended in order to make the most use of the <i>Tcl</i> file generated by this option.</p> <p>Generates the <i>Tcl</i> script used to execute Vivado HLS <i><kernel_name>.tcl</i> but halts before Vivado HLS starts. The expectation is for the script to be modified and used with the --custom_script option.</p> <p>Not supported for -t sw_emu with OpenCL kernels.</p>
--custom_script	<kernel_name>:<path to kernel Tcl file>	<p>Intended for use with the <i><kernel_name>.tcl</i> file generated with option --export_script.</p> <p>This option allows you to customize the <i>Tcl</i> file used to create the kernel and execute using the customize version of the script.</p>
--jobs <arg>	Number of parallel jobs	Optional

Option	Valid Values	Description
		<p>This option allows detailed control of the Vivado tool suite used to implement the FPGA hardware.</p> <p>NOTE: Familiarity with the Vivado tool suite is recommended in order to make the most use of the Tcl file generated by this option.</p> <p>Specify the number of parallel jobs to be passed to the Vivado tool suite for implementation. Increasing the number of jobs allows the hardware implementation step to spawn more parallel processes and complete faster.</p>
--lsv <arg>	bsub command line to pass to LSF cluster NOTE: This argument is required.	<p>Optional</p> <p>Use IBM Platform Load Sharing Facility (LSF) for Vivado implementation.</p>
input file	OpenCL or C/C++ kernel source file	Compile kernels into a .xo or .xclbin file depending on the xocc mode.



IMPORTANT: All examples in the SDAccel installation use Makefile to compile OpenCL applications with gcc and xocc commands, which can be used as references for compiling user applications using xocc.

XP Parameters

Use the --xp switch to specify parameter values in SDAccel™. These parameters allow fine grain control over the hardware generated by SDAccel and the hardware emulation process.



IMPORTANT: Familiarity with the Vivado™ tool suite is recommended in order to make the most use of these parameters.

Parameters are specified as `parm:<parameter>=<value>`. For example:

```
xocc -xp param:compiler.enableDSAIntegrityCheck=true
-xp param:prop:kernel.foo.kernel_flags="-std=c++0x"
```

The `-xp` command option may be specified multiple times in a single `xocc` invocation, or the value(s) may be specified in a `xocc.ini` file with each option specified on a separate line (without `--xp` switch).

```
param:prop:solution.device_repo_paths=../dsa
param:compiler.preserveHlsOutput=1
```

Upon invocation, xocc first looks for an `xocc.ini` file in the `$HOME/.Xilinx/sdx` directory. If the file does not exist there, xocc will then look for it in the current working directory. If the same `--xp` parameter value is specified in both the command line and `xocc.ini` file, the command line value will be used.

The following table lists the `-xp` parameters and their values.

Table 3: XP Parameters

Parameter Name	Type	Default Value	Description
param:compiler.enableDSAIntegrityCheck	Boolean	False	Enables the DSA Integrity Check. If this value is set to True, and SDAccel™ detects a DSA which has been modified outside the of the Vivado® tool suite SDAccel halts operation.
param:compiler.errorOnHoldViolation	Boolean	True	Error out if there is hold violation.
param:compiler.maxComputeUnits	Int	-1	The maximum compute units allowed in the system. Any positive value will overwrite the numComputeUnits setting in the DSA.
param:hw_em.debugLevel	String	OFF	The debug level of the simulator. Option OFF is used for optimized run times, BATCH is for batch runs and GUI for use in GUI-mode
param:hw_em.enableProtocolChecker	Boolean	False	Enables the AXI protocol checker during HW emulation. This is used to confirm the accuracy of any AXI interfaces in the design.
param:compiler.interfaceLatency	Int	-1	This option specifies the expected latency on the kernel AXI bus, the number of clock cycles from when bus access is requested until it is granted.
param:compiler.xclDataflowFifoDepth	Int	-1	Specifies the depth of FIFOs used in kernel dataflow region.
param:compiler.interfaceWrOutstanding	Int Range	0	Specifies how many outstanding writes to buffer are on the kernel AXI interface. Values are 1 through 256.
param:compiler.interfaceRdOutstanding	Int Range	0	Specifies how many outstanding reads to buffer are on the kernel AXI interface. Values are 1 through 256.
param:compiler.interfaceWrBurstLen	Int Range	0	Specifies the expected length of AXI write bursts on the kernel AXI interface.

Parameter Name	Type	Default Value	Description
			This is used with option compiler.interfaceWrOutstanding to determine the hardware buffer sizes. Values are 1 through 256.
param:compiler.interfaceRdBurstLen	Int Range	0	Specifies the expected length of AXI read bursts on the kernel AXI interface. This is used with option compiler.interfaceRdOutstanding to determine the hardware buffer sizes. Values are 1 through 256.
misc:map_connect=<type>.kernel.<kernael_name>. <kernel_AXI_interface>.core.OCL_REGION_0.<dest_port>	String	<empty>	Used to map AXI interfaces from a kernel to DDR memory banks. <ul style="list-style-type: none"> • <type> is add or remove. • <kernael_name> is the name of the kernel. • <dest_port> is DDR memory bank M00_AXI, M01_AXI, M02_AXI or M03_AXI.
prop:kernel.<kernel_name>.kernel_flags	String	<empty>	Sets specific compile flags on kernel <kernelk_name>. e.g.
prop:solution.device_repo_path	String	<empty>	Specifies the path to the DSA repository.
prop:solution.hls_pre_tcl	String	<empty>	Specifies the path to a Vivado HLS Tcl file, which is executed before the C code is synthesized. This allows Vivado HLS configuration settings to be applied prior to synthesis.
prop:solution.hls_post_tcl	String	<empty>	Specifies the path to a Vivado HLS Tcl file, which is executed after the C code is synthesized.
prop:solution.kernel_compiler_margin	Float	12.5% of the kernel clock period.	The clock margin in ns for the kernel. This value is substracted from the kernel clock period prior to synthesis to provide some margin for P&R delays.
vivado_prop:<object_type>. <object_name>. <prop_name>	Various	Various	This allows you to specify any property used in the Vivado hardware compilation flow. Object_type is run fileset file project

Parameter Name	Type	Default Value	Description
			<p>The <code>object_name</code> and <code>prop_name</code> values are described in <i>Vivado Design Suite Properties Reference Guide</i>, (UG912)</p> <p>Examples:</p> <pre>vivado_prop:run.impl_1. {STEPS.PLACE_DESIGN.ARGS.MORE OPTIONS}={-fanout_opt}</pre> <pre>vivado_prop:fileset. current.top=foo</pre> <p>NOTE: For <code>object_type file</code>, <code>current</code> is not supported</p> <p>NOTE: For object type <code>run</code> the special value of <code>__KERNEL__</code> can be used to specify run optimization settings for ALL kernels, instead of having to specify them one by one</p>

Running Software and Hardware Emulation in XOCC Flow

In the XOCC/Makefile flow, users manage compilation and execution of host code and kernels outside the Xilinx® SDAccel™ development environment. Follow the steps below to run software and hardware emulation:

1. Create the emulation configuration file.

For software or hardware emulation, the runtime library needs to know what devices and how many to emulate. This information is provided to the runtime library by an emulation configuration file. SDAccel provides a utility, `emconfigutil` to automate creation of the emulation configuration file. The following are details of the `emconfigutil` command line format and options:

Option	Valid Values	Description
<code>--xdevice</code>	Target device	Required: Set target device. Check Appendix B for all supported devices
<code>--nd</code>	Any positive integer	Optional: Number of devices. Default is 1.
<code>--od</code>	Valid directory	Optional: Output directory, <code>emconfig.json</code> file must be in the same directory as the host executable.

Option	Valid Values	Description
--xp	Valid Xilinx parameters and properties	<p>Optional: Specify additional parameters and properties. For example:</p> <pre>--xp prop:solution.device_repo_paths=my_dsa_path</pre> <p>Sets the search path for the device specified in --xdevice option.</p>
-h	NA	Print help messages

The `emconfigutil` command creates the configuration file `emconfig.json` in the output directory.

The `emconfig.json` file must be in the same directory as the host executable.

The following example creates a configuration file targeting two `xilinx:adm-pcie-7v3:1ddr:3.0` devices.

```
$emconfigutil --xdevice xilinx:adm-pcie-7v3:1ddr:3.0 --nd 2
```

2. Set `XILINX_SDX` environment variable

The `XILINX_SDX` environment needs to be set and pointed to the SDAccel installation path for the emulation to work. Below are examples assuming SDAccel is installed in `/opt/Xilinx/SDx/2017.1`.

C Shell:

```
setenv XILINX_SDX /opt/Xilinx/SDx/2017.1
```

Bash:

```
export XILINX_SDX=/opt/Xilinx/SDx/2017.1
```

3. Set emulation mode

Setting `XCL_EMULATION_MODE` environment variable to `1` or `true` changes the application execution to emulation mode so that the runtime looks for the file `emconfig.json` in the same directory as the host executable and reads in the target configuration for the emulation runs.

C Shell:

```
setenv XCL_EMULATION_MODE true
```

Bash:

```
export XCL_EMULATION_MODE=true
```

Setting the `XCL_EMULATION_MODE` environment variable to `0` or `false`, or unsetting it will turn off the emulation mode.

4. Run CPU and hardware emulation

With the configuration file `emconfig.json` and `XCL_EMULATION_MODE` set to `true`, execute the host application with proper arguments to run CPU and hardware emulation:

```
$ ./host.exe kernel.xclbin
```

Running Application on FPGA

Use the following steps to run an application on an FPGA with host executable and kernel binaries compiled in the XOCC flow.

1. Source the `setup.sh` (Bash) or `setup.csh` (Csh/Tcsh) script file generated by the `xbinst` utility. See *Board Installation* chapter for more details.
2. Run the application: `$./host.exe kernel.xclbin`

Using the Message Rule File

XOCC executes various Xilinx tools during kernel compilation. These tools generate many messages that provide compilation status to the users. These messages may or may not be relevant to all users depending on users' focus and design phases. A Message Rule file can be used to better manage these messages. It provides commands to promote important messages to the terminal or suppress unimportant ones. This will help users better understand the kernel compilation result and explore ways to optimize the kernel.

The Message Rule file (`.mrf`) is a text file consisting of comments and supported commands. Only one command is allowed on each line.

Comment

Any line with “#” as the first non-whitespace character is a comment.

Supported Commands

By default, `xocc` recursively scans the entire working directory and promotes all error messages to the `xocc` output. The `promote` and `suppress` commands below provide more control on the `xocc` output.

`promote`

This command indicates that matching messages should be promoted to the `xocc` output.

`suppress`

This command indicates that matching messages should be suppressed or filtered from the `xocc` output. Note that errors can not be suppressed.

Command Options

The Message Rule file can have multiple promote and suppress commands. Each command can have one and only one of the options below. The options are case sensitive.

- -id [message_id]

All messages matching the specified message ID are promoted or suppressed. The message ID is in format of nnn-mmm. As an example, the following is a warning message from HLS. The message ID in this case is 204-68.

```
WARNING: [XOCC 204-68] Unable to enforce a carried dependence
constraint (II = 1, distance = 1, offset = 1)
between bus request on port 'gmem'
(/matrix_multiply_cl_kernel/mmult1.cl:57) and bus request on port
'gmem'-severity [severity_level]
```

- -severity [severity_level]

The following are valid values for the severity level. All messages matching the specified severity level will be promoted or suppressed.

- info
- warning
- critical_warning

Precedence of Message Rules

suppress rules take precedence over promote rules. If the same message ID or severity level is passed to both promote and suppress commands in the Message Rule file, the matching messages are suppressed and not displayed.

Example of Message Rule File

The following is an example of a valid message rule file:

```
# promote all warning, critical warning
promote -severity warning
promote -severity critical_warning
# suppress the critical warning message with id 19-2342
suppress -id 19-2342
```

Getting Started with Examples

All Xilinx SDx™ Environments are provided with examples designs:

- To help you quickly get started.
- To demonstrate useful coding styles.
- To highlight important optimization techniques.

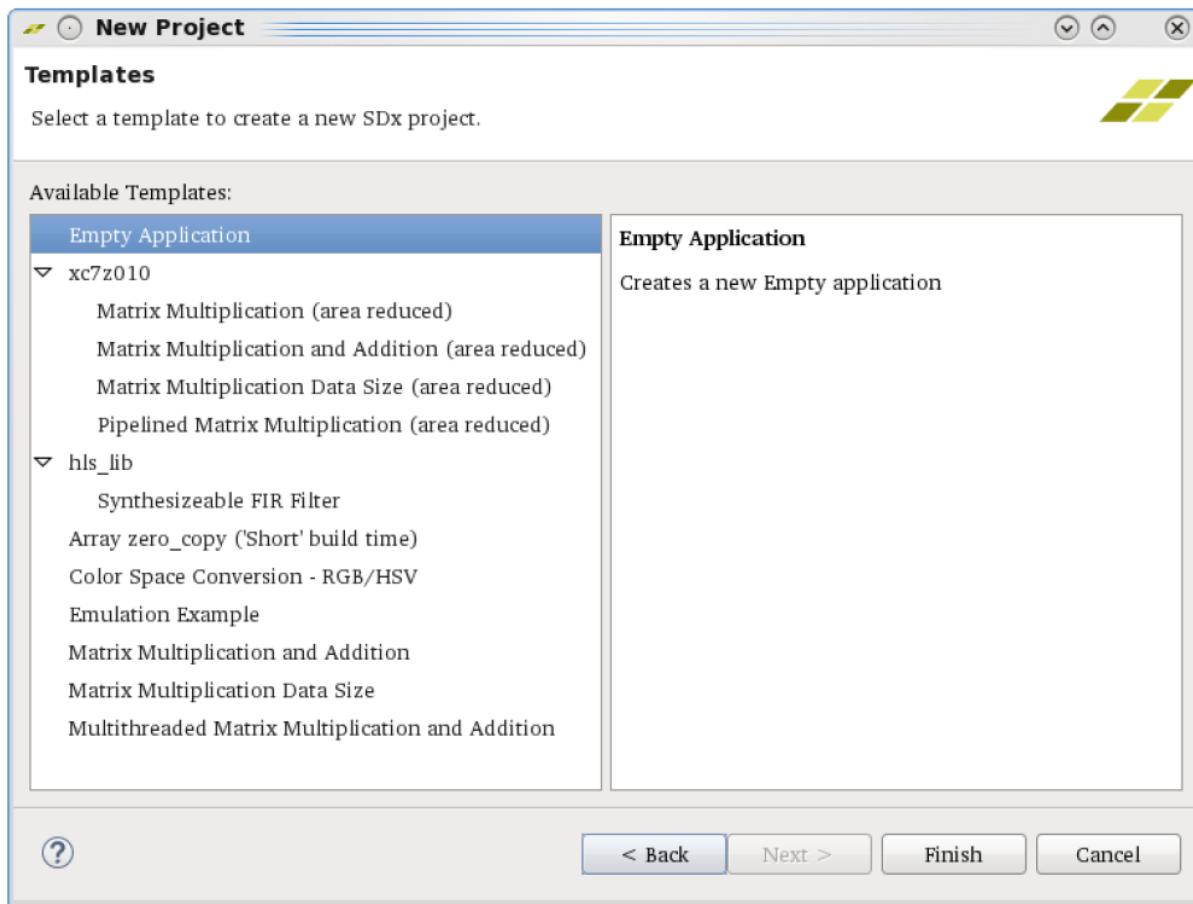
Example designs are provided with the tool installation and additional examples may be downloaded from the Xilinx® GitHub repository.

Installed Examples

The installed examples are provided through the Create SDx™ Project wizard. Select **Create SDx Project** from the SDx Development Environment Welcome page to open the new project wizard. After selecting your hardware platform and software platform, the final page of the wizard lists the available templates.

NOTE: *Not all available platforms have an installed example.*

You may select examples from the Templates page, as shown below.

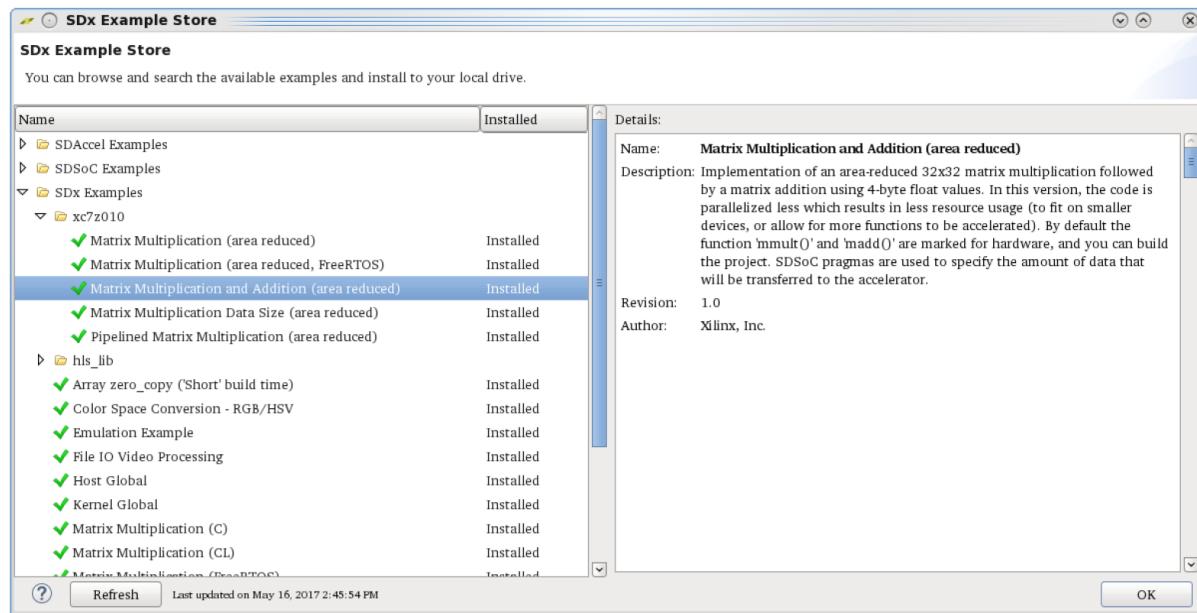
Figure 4: Templates Page

After selecting **Finish**, the example is copied into the local workspace and can be used.

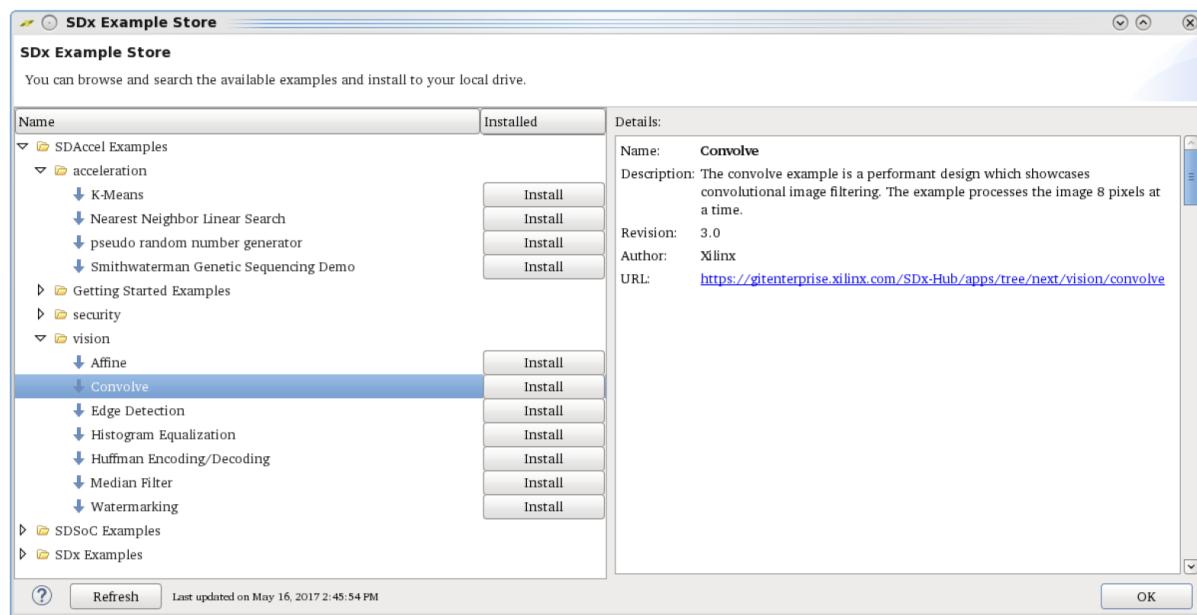
GitHub Examples

The GitHub examples may be accessed from the menu **Xilinx > Open SDx Example Store**. When the **SDx Example Store** dialog box opens it lists all the available examples and indicates if the examples are already installed or not.

The SDx Examples folder lists all installed examples and shows they are already installed.

Figure 5: SDx Examples Folder

The **SDx Example Store** dialog also lists the GitHub examples provided for specific SDx Environments. For example, the **SDAccel** folder lists all GitHub examples for the SDAccel Environment.

Figure 6: SDx Examples for SDAccel

The GitHub examples also indicate if they are installed or not. Use the **Refresh** button to ensure you have the latest update from the repository. Click **Install** to download and install the example design.

Once the example design has been installed, it may be accessed during new project creation in the same manner as the installed examples.

Estimating Performance

The generation of FPGA programming files is the step in the SDAccel™ development environment with the longest execution time. It is also the step in which the execution time is most affected by the target hardware device and the number of compute units placed on the FPGA fabric. Therefore, it is essential for the application programmer to have a quicker way to understand the performance of the application before running it on the target device so they can spend more time iterating and optimizing their applications instead of waiting for the FPGA programming file to be generated.

Generating the System Performance Estimate Report

The system performance estimate in the SDAccel™ development environment takes into account the target hardware device and each compute unit in the application. Although an exact performance metric can only be measured on the target device, the estimation report in the SDAccel environment provides an accurate representation of the expected behavior. The command below generates the system performance estimate report `system_estimate.xtext` for all kernels in `kernel.cl`:

```
xocc -c -t hw_emu --platform xilinx:adm-pcie-7v3:1ddr:3.0 --report estimate kernel.cl
```

Analyzing the Performance Estimate Report

The performance estimate report generated by the `xocc -report estimate` option provides information on every binary container in the application, as well as every compute unit in the design. The structure of the report is:

- Target device information
- Summary of every kernel in the application
- Detailed information on every binary container in the solution

The following example report file represents the information that is generated for the estimate report.

```
-----  
Design Name:          _xocc_compile_kernel_bin.dir  
Target Device:       xilinx:adm-pcie-ku3:2ddr-xpr:3.3  
Target Clock:        200MHz
```

```
Total number of kernels: 1
-----
Kernel Summary
Kernel Name      Type   Target          OpenCL Library  Compute Units
-----  -----  -----
smithwaterman    clc    fpga0:OCL_REGION_0  xcl_xocc       1
-----
OpenCL Binary:      xcl_xocc
Kernels mapped to: clc_region

Timing Information (MHz)
Compute Unit      Kernel Name      Module Name      Target Frequency
-----  -----  -----
smithwaterman_1   smithwaterman  smithwaterman  200

Estimated Frequency
-----
202.020203

Latency Information (clock cycles)
Compute Unit      Kernel Name      Module Name      Start Interval
-----  -----  -----
smithwaterman_1   smithwaterman  smithwaterman  29468

Best Case  Avg Case  Worst Case
-----  -----  -----
29467     29467     29467

Area Information
Compute Unit      Kernel Name      Module Name      FF      LUT      DSP      BRAM
-----  -----  -----
smithwaterman_1   smithwaterman  smithwaterman  2925    4304    1        10
-----
```

Design and Target Device Summary

All design estimate reports begin with an application summary and information about the target hardware. Device information is provided by the following section of the report:

```
-----
Design Name:          _xocc_compile_kernel_bin.dir
Target Device:        xilinx:adm-pcie-ku3:2ddr-xpr:3.3
Target Clock:         200MHz
Total number of kernels: 1
-----
```

For the design summary, the only information you provide is the design name and the selection of the target device. The other information provided in this section is the target board and the clock frequency.

The target board is the name of the board that runs the application compiled by the SDAccel™ development environment. The clock frequency defines how fast the logic runs for compute units mapped to the FPGA fabric. Both of these parameters are fixed by the device developer. These parameters cannot be modified from within the SDAccel environment.

Kernel Summary

The kernel summary section lists all of the kernels defined for the current SDAccel™ solution. Following is an example kernel summary:

Kernel Summary				
Kernel Name	Type	Target	OpenCL Library	Compute Units
smithwaterman	clc	fpga0:OCL_REGION_0	xcl_xocc	1

Along with the kernel name, the summary provides the execution target and the OpenCL™ binary container where the compute unit of the kernel is stored. Also, because there is a difference in compilation and optimization methodology for OpenCL C and C/C++ source files, the type of kernel source file is specified.

The kernel summary section is the last summary information in the report. From here, detailed information on each compute unit binary container is presented.

Timing Information

The detail section for each binary container begins with the execution target of all compute units. It also provides timing information for every compute unit. As a general rule, an estimated frequency that is higher than that of the device target means that the compute unit will run in hardware. If the estimated frequency is below the target frequency, the kernel code for the compute unit needs to be further optimized for the compute unit to run correctly on the FPGA fabric. Following is an example of this information:

OpenCL Binary:	xcl_xocc		
Kernels mapped to:	clc_region		
 Timing Information (MHz)			
Compute Unit	Kernel Name	Module Name	Target Frequency
-----	-----	-----	-----
smithwaterman_1	smithwaterman	smithwaterman	200
 Estimated Frequency			
-----	-----		
202.020203			

The importance of the timing information is the difference between the target and the estimated frequencies. As stated in [Understanding the OpenCL Platform and Memory Model](#), compute units are not placed in isolation into the FPGA fabric. Compute units are placed as part of a valid FPGA design that can include other components defined by the device developer to support a class of applications.

Because the compute unit custom logic is generated one kernel at a time, an estimated frequency that is higher than the device target provides confidence to the developer using the SDAccel™ environment that there will not be any problems during the creation of the FPGA programming files.

Latency Information

The latency information presents the execution profile of each compute unit in the binary container. When analyzing this data, it is important to keep in mind that all values are measured from the compute unit boundary through the custom logic. In-system latencies associated with data transfers to global memory are not reported as part of these values. Also, the latency numbers reported are only for compute units targeted at the FPGA fabric. Following is an example of the latency report:

Latency Information (clock cycles)					
Compute Unit	Kernel Name	Module Name	Start Interval	Best Case	
smithwaterman_1	smithwaterman	smithwaterman	29468	29467	
Avg Case Worst Case					
29467	29467				

The latency report is divided into the following fields:

- Start interval
- Best case latency
- Average case latency

The start interval defines the amount of time that has to pass between invocations of a compute unit for a given kernel. This number sets the limit as to how fast the runtime can issue application ND range data tiles to a compute unit.

The best and average case latency numbers refer to how much time it takes the compute unit to generate the results of one ND range data tile for the kernel. For cases where the kernel does not have data dependent computation loops, the latency values will be the same. Data dependent execution of loops introduces data specific latency variation that is captured by the latency report.

The interval or latency numbers will be reported as "undef" for kernels with one or more conditions listed below:

- Do not have an explicit reqd_work_group_size(x,y,z)
- Have loops with variable bounds

Area Information

Although the FPGA can be thought of as a blank computational canvas, there are a limited number of fundamental building blocks available in each FPGA. These fundamental blocks (FF, LUT, DSP, block RAM) are used by the SDAccel™ development environment to generate the custom logic for each compute unit in the design. The number of each fundamental resource needed to implement the custom logic in a compute unit determines how many compute units can be simultaneously loaded into the FPGA fabric. Following is an example of the area information reported for a compute unit:

Area Information						
Compute Unit	Kernel Name	Module Name	FF	LUT	DSP	BRAM
smithwaterman_1	smithwaterman	smithwaterman	2925	4304	1	10

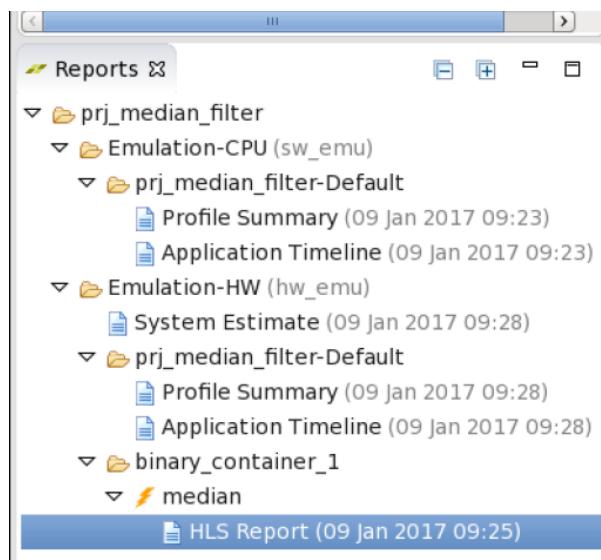
Application Profiling in the SDAccel Environment

The SDAccel™ Environment generates various reports on the kernel resource and performance during compilation. It also collects profiling data during application execution in emulation mode, and on the FPGA acceleration card. The reports and profiling data provide you with information on performance bottlenecks in the application, and optimization techniques that can be used to improve performance. This chapter describes how to generate the reports and collect, display, and interpret profiling results in the SDAccel Environment.

Kernel Synthesis Report

After compiling a kernel on SDx GUI, the HLS (High Level Synthesis) report is presented to the user under the binary container in the **Reports** window. The HLS report includes lots of details about the performance and logic usage of the custom-generated hardware logic from user kernel code. These details provide advanced users many insights into the kernel compilation results to guide kernel optimization.

Figure 7: Reports Window



The **HLS Report** window consists of six views for various aspects of the kernel synthesis result. Each view can be accessed by clicking on the tab for the view at the bottom of the HLS Report window as shown in the Figure below.

Figure 8: HLS Report Window



Synthesis View

The Synthesis View shows the general information, estimated timing and latency summary and details, estimated utilization summary and details, and interface summary.

Performance View

The Performance View shows how the operations in the selected block are scheduled into clock cycles.

Resource View

The Resource View shows the control state of the operations in the selected hierarchy.

Performance Profile View

The Performance Profile View provides details on the performance of the selected hierarchy.

Resource Profile View

The Resource Profile View shows the resources used in the selected hierarchy.

Dataflow View

The Dataflow View is available for kernels using DATAFLOW directive or attribute. It shows the structure of the design and how data are passed from one task to the next.

Profiling Summary Report

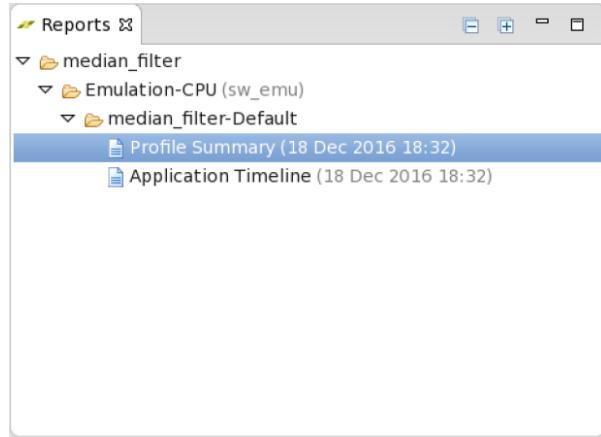
The SDAccel runtime automatically collects profiling data on host applications. After the application finishes execution, the profile summary is saved in HTML, .csv, and Google Protocol Buffer formats in the solution report directory or working directory. These reports can be reviewed in a web browser, spreadsheet viewer, or the integrated Profile Summary Viewer in SDAccel. The profile reports are generated in both SDAccel GUI and XOCC/Makefile flows.

If you review the Profile Summary, Profile Rule Checks (PRC) are also provided to help interpret profiling results and suggest areas for performance improvements. These PRCs operate on the results in the profile summary .csv file and are reported in the Google Protocol Buffer file.

GUI Flow

When you compile and execute an application from SDx™ environment, the profile summary is automatically generated and placed in the **Reports** window. Double-click the report to open it.

Figure 9: Profiling in SDAccel GUI Flow



XOCC/Makefile Flow

XOCC/Makefile users execute applications standalone outside the SDx™ environment. The following profile summary reports are generated in the directory where the application is executed:

```
<working_directory>/sdaccel_profile_summary.html  
<working_directory>/sdaccel_profile_summary.csv
```

The .csv file needs to be manually converted to Google Protocol Buffer format (.xprf) before the profiling result can be viewed in the integrated Profile Summary Viewer. The following is a command line example that generates an .xprf file from the .csv input file:

```
$sda2protobuf sdaccel_profile_summary.csv
```

Displaying the Profile Summary

Use the following methods to display the profile summary.

Web Browser

The HTML profile summary can be displayed in a Web Browser. The following figure shows the profiling result from a system run on the FPGA.

SDAccel Profile Summary

Generated on: 2016-12-18 18:37:34

Profiled application: host

Target platform: Xilinx

Target devices: xilinx:adm-pcie-ku3:2ddr:3.2-0

Flow mode: Hardware Emulation

Tool version: 2016.3

OpenCL API Calls

API Name	Number Of Calls	Total Time (ms)	Minimum Time (ms)	Average Time (ms)	Maximum Time (ms)
clCreateProgramWithBinary	1	25821.3	25821.3	25821.3	25821.3
clFinish	2	874.418	0.024556	437.209	874.394
clEnqueueReadBuffer	1	3.72858	3.72858	3.72858	3.72858
clEnqueueWriteBuffer	2	2.84012	1.29159	1.42006	1.54854
clCreateBuffer	3	0.768503	0.048409	0.256168	0.551973
clEnqueueNDRangeKernel	1	0.300828	0.300828	0.300828	0.300828
clSetKernelArg	4	0.131731	0.015843	0.0329328	0.075294
clGetPlatformIDs	3	0.086898	0.009262	0.028966	0.067877
clReleaseKernel	1	0.068589	0.068589	0.068589	0.068589
clReleaseMemObject	3	0.062009	0.015689	0.0206697	0.026926
clCreateContext	1	0.038972	0.038972	0.038972	0.038972
clCreateKernel	1	0.037861	0.037861	0.037861	0.037861
clGetEventProfilingInfo	2	0.036608	0.01591	0.018304	0.020698
clBuildProgram	1	0.029705	0.029705	0.029705	0.029705
clGetDeviceInfo	2	0.02339	0.0094	0.011695	0.01399
clGetPlatformInfo	2	0.020604	0.009065	0.010302	0.011539
clReleaseCommandQueue	1	0.017273	0.017273	0.017273	0.017273
clReleaseContext	1	0.017117	0.017117	0.017117	0.017117
clCreateCommandQueue	1	0.011907	0.011907	0.011907	0.011907
clGetDeviceIDs	1	0.010666	0.010666	0.010666	0.010666

Kernel Execution (includes estimated device times)

Kernel	Number Of Enqueues	Total Time (ms)	Minimum Time (ms)	Average Time (ms)	Maximum Time (ms)
vadd	1	0.006795	0.006795	0.006795	0.006795

Compute Unit Utilization (includes estimated device times)

Device	Compute Unit	Kernel	Global Work Size	Local Work Size	Number Of Calls	Total Time (ms)	Minimum Time (ms)	Average Time (ms)	Maximum Time (ms)
xilinx.adm-pcie-ku3:2ddr:3.2-0	vadd_1	vadd	1:1:1	1:1:1	1	0.006795	0.006795	0.006795	0.006795

Data Transfer: Host and Global Memory

Context: Number of Devices	Transfer Type	Number Of Transfers	Transfer Rate (MB/s)	Average Bandwidth Utilization (%)	Average Size (KB)	Total Time (ms)	Average Time (ms)
context0:1	READ	1	N/A	N/A	16.384	N/A	N/A
context0:1	WRITE	2	N/A	N/A	16.384	N/A	N/A

Data Transfer: Kernels and Global Memory

Device	Transfer Type	Number Of Transfers	Transfer Rate (MB/s)	Average Bandwidth Utilization (%)	Average Size (KB)	Average Time (ns)
xilinx.adm-pcie-ku3:2ddr:3.2-0	READ	32	4822.37	41.8608	1.024	463.125
xilinx.adm-pcie-ku3:2ddr:3.2-0	WRITE	16	2411.18	20.9304	1.024	133.125

Top Data Transfer: Kernels and Global Memory

Device	Kernel Name	Number of Transfers	Average Bytes per Transfer	Transfer Efficiency (%)	Total Data Transfer (MB)	Total Write (MB)	Total Read (MB)	Transfer Rate (MB/s)	Average Bandwidth Utilization (%)
xilinx.adm-pcie-ku3:2ddr:3.2-0	ALL	48	1024	25	0.049152	0.016384	0.032768	3616.78	31.3956

Profile Summary Viewer

Use the integrated “Profile Summary Viewer” to display profile summary generated from the SDAccel GUI or XOCC/Makefile flow.

For SDAccel GUI users, double click **Profile Summary** in the Reports window to open the Application Timeline window.

For XOCC/Makefile users, follow these steps to open the profile summary in the Profile Summary Viewer:

1. Start SDAccel GUI by running the `sdx` command:

```
$ sdx
```

2. Choose the default workspace when prompted.
3. Select **File→Open File**, browse to and then open the `.xprf` file created by the `sda2protobuf` command from data generated during an emulation or system run.

Below is a snapshot of the Profile Summary window that displays OpenCL API calls, kernel executions, data transfers, and profile rule checks (PRCs).

Figure 13: Profile Summary Window

The screenshot shows the Xilinx SDAccel Profile Summary window with the following sections:

- Report Information:** Report name: Profile Summary (sdaccel_profile_summary), Project name: median_filter, Created: 18 Dec 2016 19:05, Target: Hardware Emulation, Build configuration: Emulation-HW, Launch configuration: median_filter-Default.
- Top Operations:** Kernels & Compute Units, Data Transfers, OpenCL APIs.
- Top Data Transfer: Kernels and Global Memory:**

Device	Kernel Name	Number Of Transfers	Average Bytes per Transfer	Transfer Efficiency (%)	Total Data Transfer (MB)	Total Write (MB)	Total Read (MB)	Transfer Rate (MB/s)	Average Bandwidth Utilization (%)
xilinx:adm-pcie-ku3:2:ddr:3.2-0	ALL	257	512.000	12.500	0.132	0.066	0.066	389.405	3.380
- Top Kernel Execution:**

Kernel Instance Address	Kernel	Context ID	Command Queue ID	Device Name	Start Time (ms)	Duration (ms)	Global Work Size	Local Work Size
0x102f2d8	median	0	0	xilinx:adm-pcie-ku3:2:ddr:3.2-0	5.2E-5	0.169	1:1:1	1:1:1
- Top Memory Writes: Host and Device Global Memory:**

Buffer Address	Context ID	Command Queue ID	Start Time (ms)	Duration (ms)	Buffer Size (KB)	Writing Rate (MB/s)
0x10465c0	0	0	810.579	N/A	65.536	N/A
- Top Memory Reads: Host and Device Global Memory:**

Buffer Address	Context ID	Command Queue ID	Start Time (ms)	Duration (ms)	Buffer Size (KB)	Reading Rate (MB/s)
0x103beb0	0	0	19218.400	N/A	65.536	N/A
- Profile Rule Checks (11 met, 2 warnings):**

Rule	Threshold Value	Actual Value	Conclusion	Details	Guidance
Kernel Data Transfer (4 met, 2 warnings)	> 0.512000	0.512	Met	Average kernel read transfer size is adequate.	
	> 0.512000	0.512	Met	Average kernel write transfer size is adequate.	
	> 10.000000	3.393	Not Met	Kernel read utilization of 3.393% on xilinx:adm-pcie-ku3:2:ddr:3.2-0... Kernel write utilization of 3.367% on xilinx:adm-pcie-ku3:2:ddr:3.2-0... Improve kernel data path or memory read efficiency. Improve kernel data path or memory write efficiency.	
	> 10.000000	3.367	Not Met		
	> 0.250000	1.008	Met	Compute units on all devices use adequate data from host.	
	< 2.000000	1.008	Met	No data re-use issues were found between host and compute units.	
Host Data Transfer (2 met, 0 warnings)	> 4.096000	65.536	Met	Host read transfers are efficient from off-chip global memory.	
	> 4.096000	65.536	Met	Host write transfers are efficient to off-chip global memory.	
Resource Usage (5 met, 0 warnings)	> 0	1	Met	All available compute units were used.	
	< 1000	1	Met	Active compute units were used an adequate amount.	
	> 10.000000	100.000	Met	Active compute units have sufficient utilization.	

Profile Summary Descriptions

The profile summary includes a number of useful statistics for your OpenCL application. This can provide you with a general idea of the functional bottlenecks in your application. The profile summary consists of the following tables:

- OpenCL API Calls** - This table displays the profile data for all OpenCL host API function calls executed in the host application.
- Kernel Execution** - This table displays the profile data for all kernel functions scheduled and executed.
- Compute Unit Utilization** - This table displays the profile data for all compute units on the FPGA.
 - Data Transfer: Host and Global Memory** - This table displays the profile data for all read and write transfers between the host and device memory via PCIe® link.
 - Number of Transfers:** Number of host data transfers (Note: May contain printf transfers)
 - Transfer Rate (MB/s):** (Total Bytes Sent)/(Total Time in uSec)
 - Average Bandwidth Utilization (%):** Transfer Rate / (Max. Transfer Rate) where Max. Transfer Rate = 5.0 Gbps
 - Average Size (KB):** (Total KB sent) / (number of transfers)
 - Total Time (ms):** Total Time (ms)

- **Data Transfer: Kernels and Global Memory** - This table displays the profile data for all read and write transfers between the FPGA and device memory.
 - **Number of Transfers:** Number of transactions monitored on device (Note: May contain printf transfers)
 - **Transfer Rate (MB/s):** (Total Bytes Sent) / (Device Execution Time)
where Total Bytes Sent is sum of bytes across all transactions
 - **Device Execution Time** = End of last kernel execution - Start of first kernel execution
 - **Average Bandwidth Utilization (%)**: (Transfer Rate) / (Max. Transfer Rate)
where Max. Transfer Rate = $0.6 * 10.7 \text{ GBps} = 6.4 \text{ GBps}$
 - **Average Size (KB):** (Total KB sent) / (number of transactions)
 - **Average Time (ms):** (Total latency of all transaction) / (number of transactions)
- **Top Data Transfer: Kernels and Global Memory** - This table displays the profile data for top data transfers between FPGA and device memory.
 - **Average Bytes per Transfer:** (Total Read Bytes + Total Write Bytes) / (Total Read Transactions + Total Write Transactions)
 - **Transfer Efficiency (%)**: (Average Bytes per Transfer) / min(4K, (Memory Bit Width/8 * 256))
AXI4 specification limits the max burst length to 256 and max burst size to 4K bytes.
 - **Transfer Rate (MB/s):** (Total Data Transfer) / (Device Execution Time)
 - **Average Bandwidth Utilization (%)**: (Transfer Rate) / (0.6 * Max. Theoretical Rate)

Profile Rule Checks

Profile Rule Checks (PRCs) are integrated with the Profile Summary Viewer and interpret profiling results so users know exactly where to focus on to improve the performance of a kernel. PRCs highlight certain profile results, inform users known issues, and provide improvement recommendations. PRCs work for both hardware emulation and system runs on the FPGA.

The PRC analyses are displayed in a tabular format with the following columns:

Rule

The Rule column displays the rule name. The following are the current rule set:

- **Kernel Data Transfer**
 - Average Read Size (KB)
 - Average Write Size (KB)
 - Read Bandwidth (%)
 - Write Bandwidth (%)
 - Read Amount - Minimum (MB)
 - Read Amount - Maximum (MB)

- **Host Data Transfer**
 - Average Read Size (KB)
 - Average Write Size (KB)
- **Resource Usage**
 - Compute Unit Calls - Minimum
 - Compute Unit Calls - Maximum
 - Compute Unit Utilization (%)
 - Kernel Utilization (%)
 - Device Utilization (ms)

Threshold Value

The **Threshold Value** column displays the values used by the PRCs to determine whether or not a rule is met. The threshold values are collected from many applications that follow good design and coding practices in the SDAccel development environment.

Actual Value

The **Actual Value** column displays the values in the profiling report from the hardware emulation or system run. This value is compared against the threshold value to see if the rule is met.

Conclusion

The **Conclusion** column displays the current status of the rule check: Met or Not Met.

Details

The **Details** column provides additional explanation on the current rule.

Guidance

The **Guidance** column provides recommendations on ways to improve the kernel in order to meet the current rule. Clicking the text brings up a popup window with tips and code snippets that you can apply to your kernel.

Application Timeline

Application Timeline collects and displays host and device events on a common timeline to help you understand and visualize the overall health and performance of your systems. These events include:

- OpenCL API calls from the host code.
- Device trace data including AXI transaction start/stop, kernel start/stop, etc.

Collecting Timeline and Device Trace Data

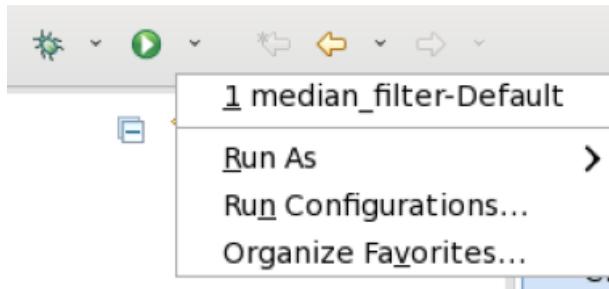
Timeline and device trace data are not collected by default because the runtime needs to periodically unload the trace data from the FPGA, which can add additional time to the overall application execution. However, the device data are collected with dedicated hardware inside the FPGA, so the data collection does not affect kernel functionality on the FPGA. The following sections describe setups required to enable time and device data collection.

Turning on device profiling is intrusive and can negatively affect overall performance. This feature should be used for system performance debugging only.

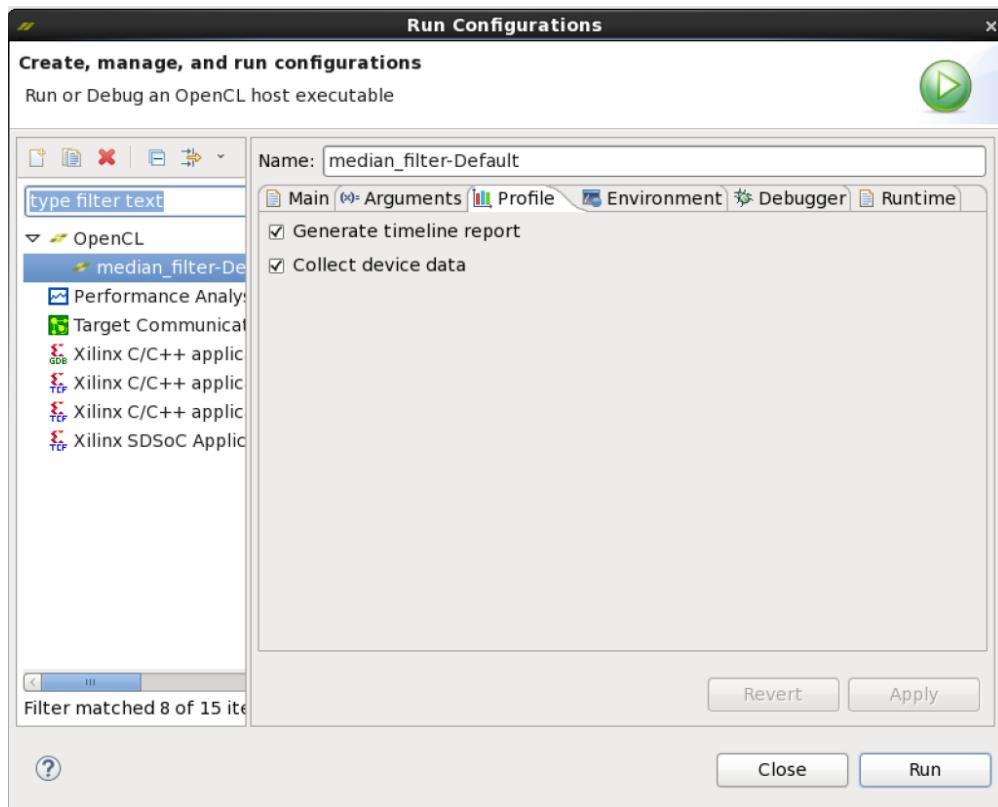
GUI Flow for Collecting Timeline and Device Trace Data

Timeline and device trace data collection is part of run configuration for an SDAccel™ project created from the SDAccel integrated environment. Follow the steps below to enable it:

1. Click the down arrow next to the debug or run button and then select **Run Configurations** to open the **Run Configurations** window.



2. On the **Run Configurations** window, click the **Profile** tab and check both the **Generate Timeline Report** and the **Collect Device Data** checkboxes.



You can have multiple run configurations for the same project and the profile settings need to be changed for each run configuration.

XOCC/Makefile Flow for Collecting Timeline and Device Trace Data

Follow the instructions below to enable timeline and device trace data collection in the XOCC/Makefile flow:

1. Turn on debug code generation during kernel compilation.

```
xocc --debug
```

2. Create an `sdaccel.ini` file in the same directory as the host executable with the contents below:

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

3. Execute hardware emulation or system run as normal. The timeline reports are generated after the application completes.

```
sdaccel_timeline_trace.csv
sdaccel_timeline_trace.html
```

4. Convert the CSV report to the Application Timeline format using the "sda2wdb" utility before it can be opened and displayed on SDAccel GUI..

```
sda2wdb sdaccel_timeline_trace.csv
```

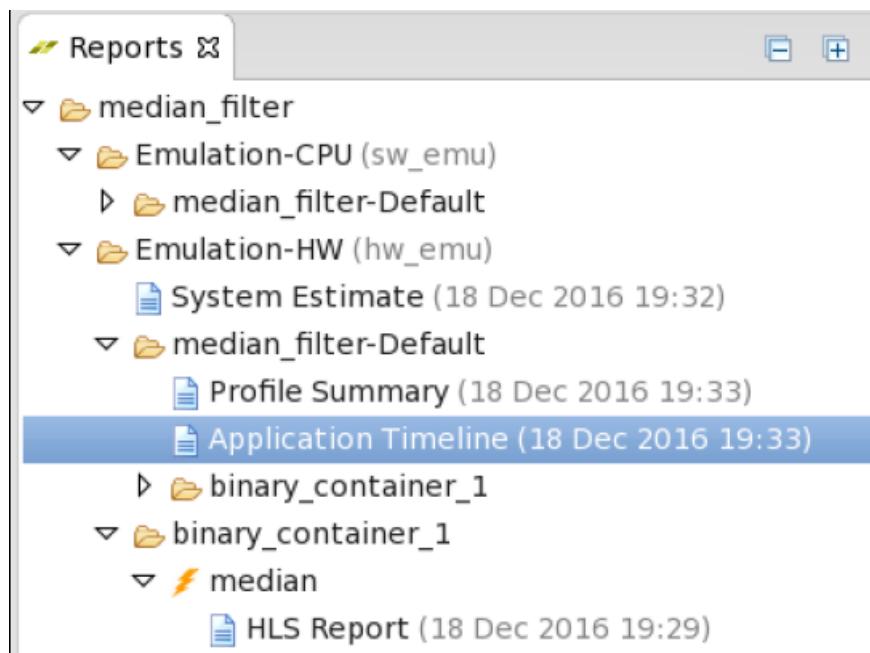
This command creates the following two files in the current working directory:

```
sdaccel_timeline_trace.wcfg  
sdaccel_timeline_trace.wdb
```

Displaying Timeline and Device Trace Data

For SDx™ Development Environment users, double-click **Application Timeline** in the **Reports** window to open the **Application Timeline** window.

Figure 14: Reports Window



For XOCC/Makefile users, follow the steps below to open the timeline report to visualize host and device events during application execution.

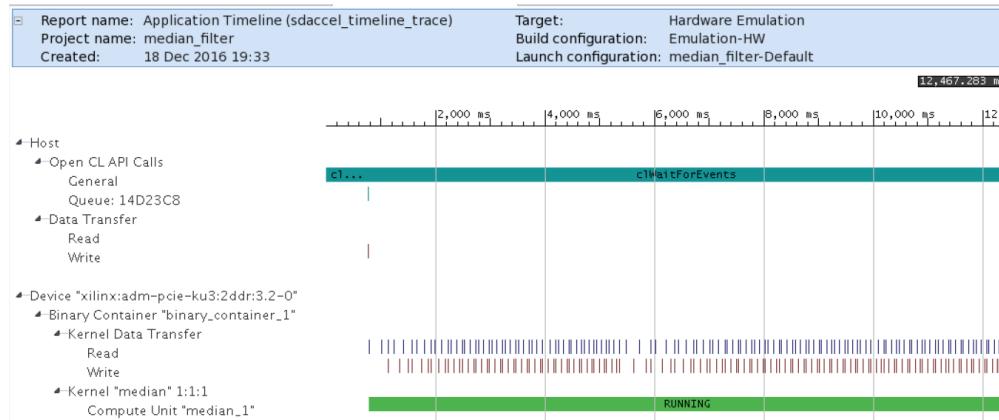
1. Start the SDx environment IDE by running the command:

```
$sdx
```

2. Choose a workspace when prompted.
3. Select **File→Open File**, browse to the `.wdb` file generated during hardware emulation or system run, and open it.

Below is a snapshot of the **Application Timeline** window that displays host and device events on a common Timeline. This information helps you to understand details of application execution and identify potential areas for improvements.

Figure 15: Application Timeline Window



Device Hardware Transaction View

SDAccel™ Environment can generate a Device Hardware Transaction View when running hardware emulation. It displays in-depth details on the emulation results at system level, compute unit (CU) level, and at function level. The details include data transfers between the kernel and global memory, data flow via inter-kernel pipes as well as data flow via intra-kernel pipes. They provide many insights into the performance bottleneck from the system level down to individual function call to help developers optimize their applications.

Collecting Data for Device Hardware Transaction View

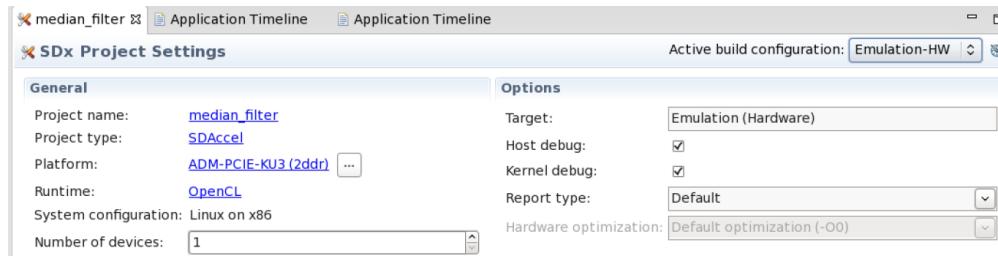
Hardware Transaction View data are not collected by default because it requires the runtime to generate simulation waveform during hardware emulation, which consumes more time and disk space. The following sections describe setups required to enable data collection for the Device Hardware Transaction view.

GUI Flow for Collecting Hardware Transaction Data

Follow the steps below to enable Hardware Transaction data collection:

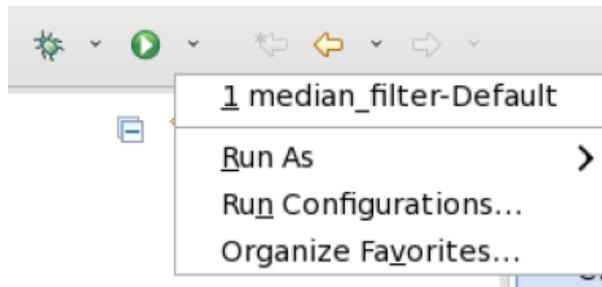
1. Open **Project Settings** window and check **Kernel debug** checkbox.

Figure 16: Project Settings Window



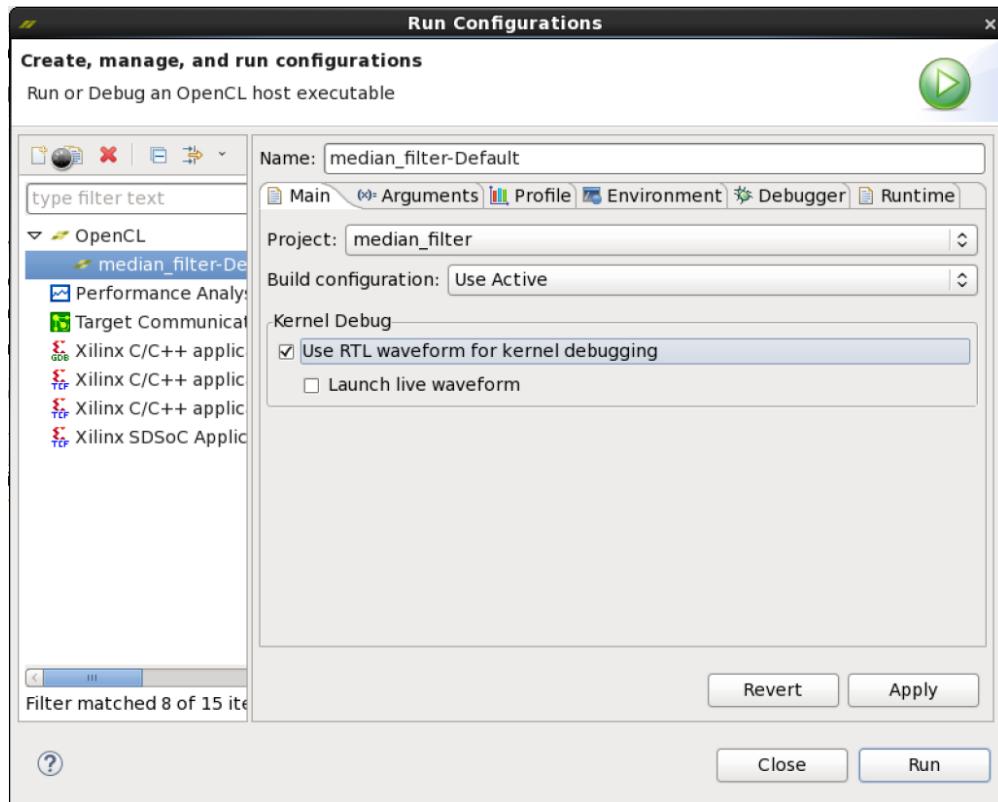
2. Click the down arrow next to the debug or run button and then select **Run Configurations** to open the **Run Configurations** window.

Figure 17: Run Configurations



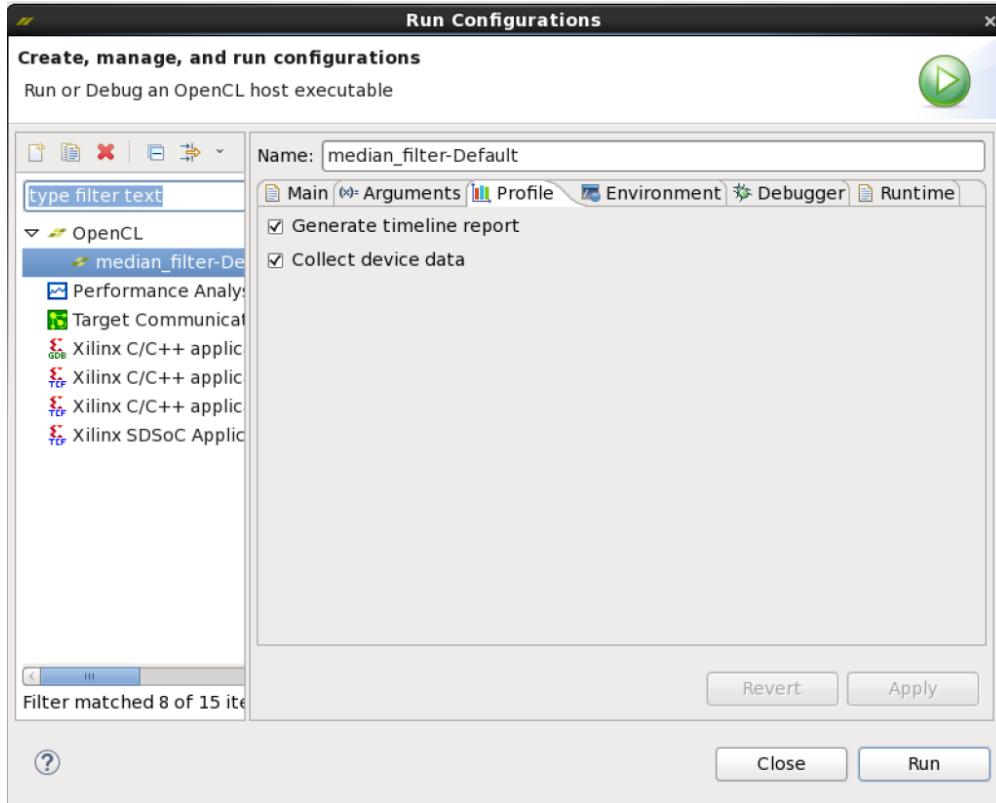
3. On the **Run Configurations** window, click the **Main** tab and check the **Use RTL waveform for kernel debugging** checkbox. **Launch live waveform** can be optionally checked to bring up the **Simulation** window to view the waveform while the hardware emulation is running.

Figure 18: Run Configurations, Main Tab



4. On the **Run Configurations** window, click the **Profile** tab and check both the **Generate Timeline Report** and the **Collect Device Data** checkboxes.

Figure 19: Run Configurations, Profile Tab



You can have multiple run configurations for the same project and the profile settings need to be changed for each run configuration.

XOCC/Makefile Flow for Collecting Hardware Transaction Data

Follow the instructions below to enable hardware transaction data collection in the XOCC/Makefile flow:

1. Turn on debug code generation during kernel compilation.

```
xocc -g
```

2. Create an `sdaccel.ini` file in the same directory as the host executable with the contents below:

```
[Debug]
timeline_trace=true
device_profile=true
[Emulation]
launch_waveform=batch
```

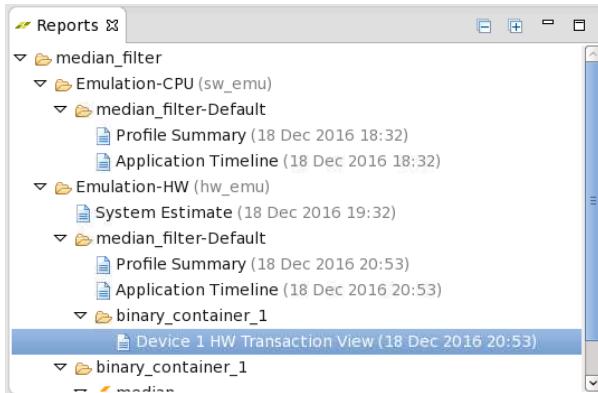
3. Execute hardware emulation. The hardware transaction data will be collected in file <hardware_platform>.-<deice_id>-<xclbin_name>.wdb. For example, the hardware transaction data file below is generated after running hardware emulation with vadd.xclbin targeting xilinx:adm-pcie-ku3:2ddr platform:

```
xilinx:adm-pcie-ku3:2ddr:3.3-0-vadd.wdb
```

Displaying Device Hardware Transaction View

For SDx™ GUI users, double-click **Device 1 HW Transaction View** in the **Reports** window to open the **Device Hardware Transaction** view window.

Figure 20: Reports Window



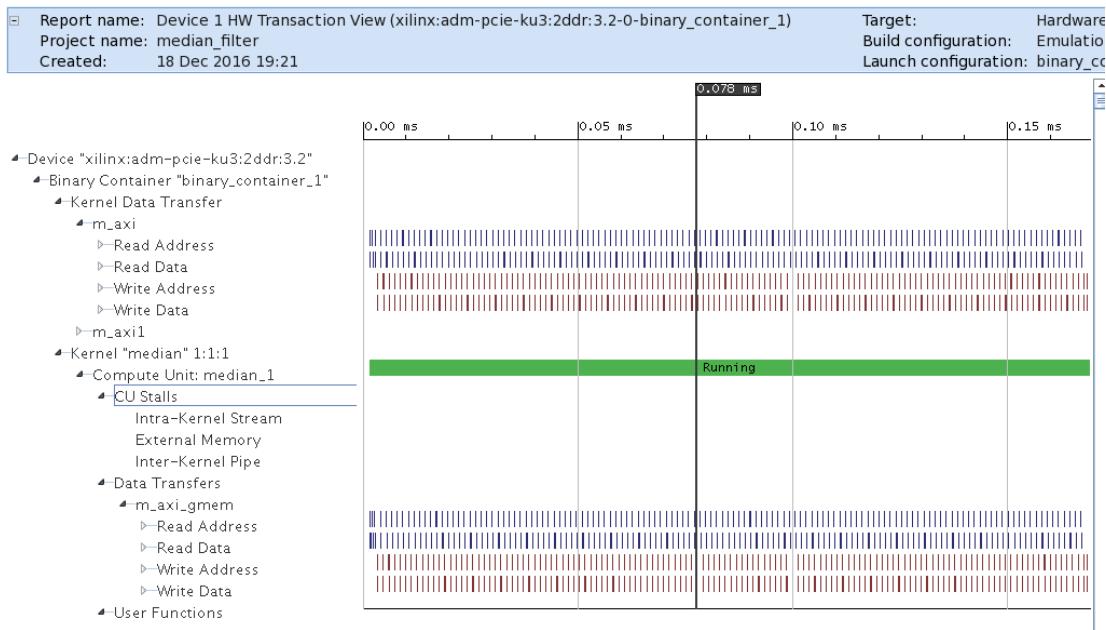
For X_OCC/Makefile users, follow the steps below to open the Hardware Transaction View:

1. Start the SDx GUI by running the command:

```
$sdxl s
```

2. Choose a workspace when prompted.
3. Select **File->Open File**, browse to the .wdb file generated during hardware emulation.

Below is a snapshot of the **Device Hardware Transaction** view for the median filter example:

Figure 21: Device Hardware Transaction View


The Device Hardware Transaction View is organized hierarchically for easy navigation. Below are the hierarchy tree and descriptions:

```

Device xilinx:adm-pcie-ku3:2ddr:3.3: target device name. This device has
two memory channels.

Binary Container binary_container_1: binary container name

Kernel Data Transfer: data transfers for all kernels
    m_axi: data transfers on memory channel 0
        Read
        Write
    m_axil: data transfers on memory channel 1
        Read
        Write

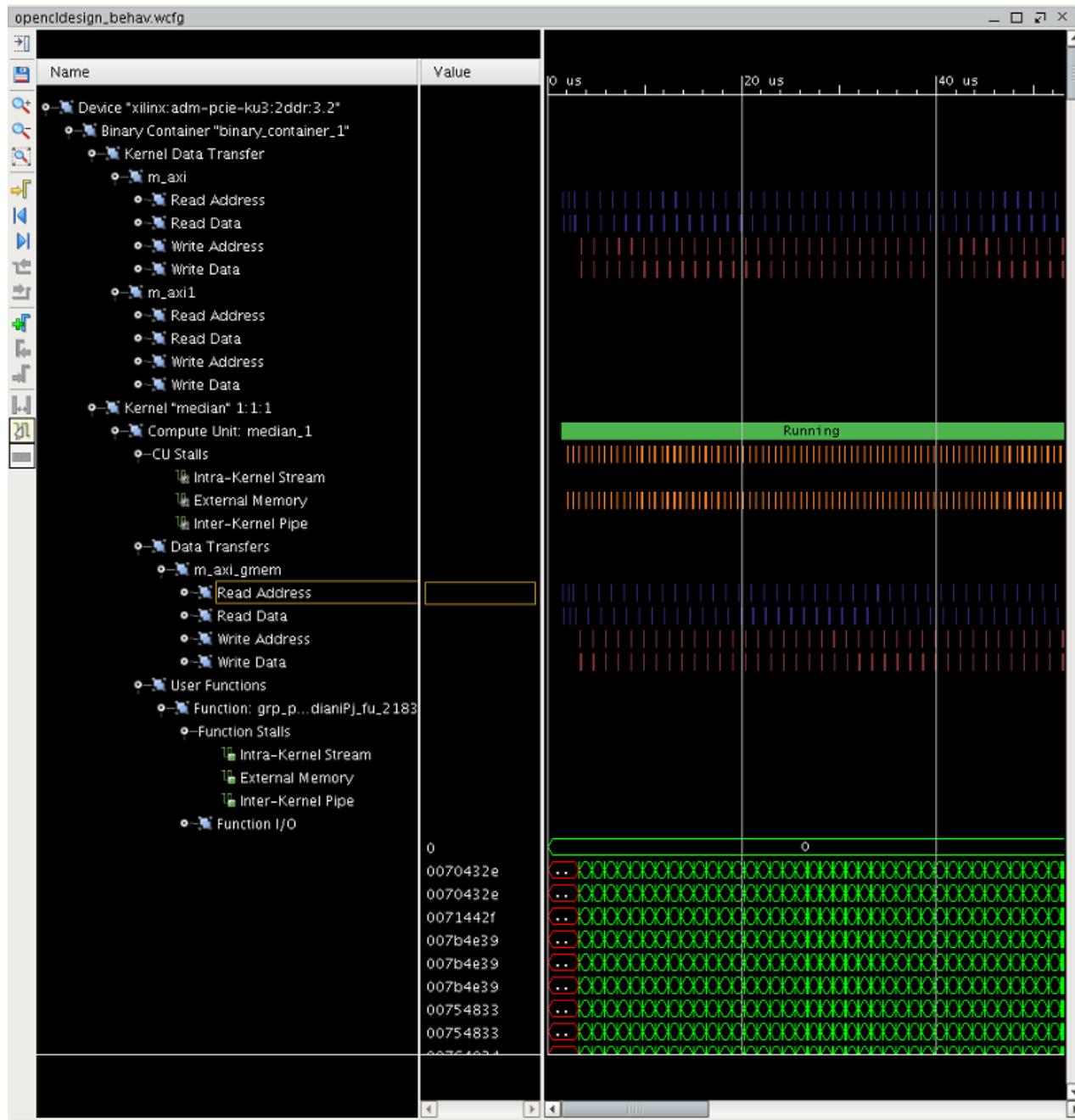
Kernel <kernel> <local_size>: kernel name in the binary container
    Compute Unit: <cul> : compute unit name
    CU Stalls: stall information on the compute unit
    Data Transfers: data transfers on the compute unit
        Read
        Write

User Functions: user functions in the CU
    <func1>: user function name
        Function Stalls: stall information on the function
        Function I/O: data transfers on the function
    
```

Detailed Kernel Trace

SDAccel Environment can generate a detailed kernel trace view when running hardware emulation. It displays in-depth details on the emulation results at system level, compute unit (CU) level, and at function level. The details include data transfers between the kernel and global memory, data flow via inter-kernel pipes as well as data flow via intra-kernel pipes. They provide many insights into the performance bottleneck from the system level down to individual function call to help developers optimize their applications.

Below is a snapshot of the detailed kernel trace view from running hardware emulation of the median filter example.



The detailed kernel trace view is organized hierarchically for easy navigation. Below is the hierarchy tree and descriptions:

```
Device xilinx:adm-pcie-ku3:2ddr:3.3: target device name. This device has
two memory channels.
  Binary Container binary_container_1: binary container name
    Kernel Data Transfer: data transfers for all kernels
      m_axi: data transfers on memory channel 0
        Read
        Write
      m_axi1: data transfers on memory channel 0
        Read
        Write
    Kernel <kernel> <local_size>: kernel name in the binary container
      Compute Unit: <cu1> : compute unit name
      CU Stalls: stall information on the compute unit
      Data Transfers: data transfers on the compute unit
        Read
        Write
      User Functions: user functions in the CU
        <func1>: user function name
        Function Stalls: stall information on the function
        Function I/O: data transfers on the function
```

Debugging Applications in the SDAccel Environment

The SDAccel™ Environment provides Application level debug features which allow the host code, the kernel code, and the interactions between them to be debugged. There are three steps to debugging applications::

1. Prepare the Host Code for Debug.
2. Prepare Kernel Code for Debug.
3. Launch GDB Standalone to Debug.

The SDAccel Environment supports host program debugging in all flows and kernel debugging in the CPU emulation flow with integrated gdb.

Preparing the Host Application for Debug

The host program needs to be compiled with debugging information generated in the executable by adding the `--debug` option to the `gcc` command line option.

```
gcc --debug ...
```

Preparing Kernel Code for Debug in CPU Emulation Flow

Kernel code can be debugged together with the host program in the CPU emulation flow. Debugging information needs to be generated first in the binary container by passing the `--debug` option to the `xocc` command line option before it can be debugged.

```
xocc -g -t sw_emu ...
```

Launching GDB Standalone

You can launch GDB standalone to debug the application if the host program and kernel were built with debug information (built with the **--debug** flag).

Below are the instructions:

1. Set up your SDAccel Environment by following instructions in the *SDx Environments Release Notes, Installation, and Licensing Guide (UG1238)*.
2. Set up your SDAccel Environment by following instructions in the [../../ug1238-sdx-rnil-HELP.ditamap](#).
3. Launch GDB on the host program as in the following example:

```
gdb --args host.exe test.xclbin
```

Recommended Debug Flow

The debug flow for Applications may be summarised as:

1. Perform Software Emulation

Verify both the host and kernel code are functionally correct by running software or CPU emulation. It is recommended to iterate in software emulation, which takes little compile time and executes quickly, until the application is functioning correctly in all modes of operation.

2. Perform Hardware Emulation

Verify the host code and the kernel hardware implementation is correct by running Hardware emulation on a small data set. Hardware Emulation performs detailed verification using an accurate model of the hardware. The execution time for hardware emulation takes more time than software emulation and Xilinx recommends that you use small data sets. It is also in this stage that you may optionally modify the kernel code to improve performance. Iterate in Hardware emulation until the functionality is correct and the estimated performance is sufficient.

3. Verify that host code and kernel hardware implementation is correct on board with System flow.

This confirms the kernel executes correctly on the FPGA hardware.

In addition to providing debug at different levels of abstraction the SDAccel Environment provides Application Debug features, which allow debug to be performed in the interaction of the compiled host and kernel code, no matter what level of abstraction.

If running CPU emulation or Hardware emulation from the command line, the following environment variables must be set:

Environment Variable	Value
XCL_EMULATION_MODE	true
XILINX_SDACCEL	Path to the installed SDx
XILINX_OPENCL	Path to the installed SDx (same as \${XILINX_SDX})
LD_LIBRARY_PATH	\${LD_LIBRARY_PATH} : \${XILINX_SDX}/lib/ lnx64.o : \${XILINX_SDX}/runtime/lib/x86_64

Application Debug

The Application Debug feature in the SDAccel™ Environment provides tools to debug the OpenCL™ application running in all modes (Software Emulation, Hardware Emulation, or Hardware). It is referred to as Application Debug because it mainly deals with debugging on the host side including the associated OpenCL APIs and is in contrast to pure Kernel Debug which involves debugging OpenCL kernels (ie. the accelerated portion).

Application Debug introduces new GDB commands, provided by Xilinx, that give visibility into the host application and into the OpenCL runtime data structures (`cl_command_queue`, `cl_event`, and `cl_mem`):

- `xprint queue <cl_command_queue>`
- `xprint event <cl_event>`
- `xprint mem <cl_mem>`

The GDB command allows the interaction between the host code and the kernel, which is managed through the OpenCL APIs, to be analyzed during Debug.

To be able to debug the host application, it must be compiled with the `--debug` switch (it is not necessary to have a debug runtime library). To use the commands, the application debug feature must be enabled at run time using an attribute in the `sdaccel.ini` file. Create an `sdaccel.ini` file in the same directory as your host executable and include the following lines.

```
[Debug]
app_debug=true
```

The GDB commands may then be accessed by sourcing a python script in the GDB console that loads the Application Debug commands into the GDB:

```
gdb> source ${XILINX_SDACCEL}/scripts/appdebug.py
```

The flow for Application Debug is:

1. Compile the OpenCL application (Host and Kernel) code using the `--debug` debug flag.
2. Launch the application in `gdb/ddd` and source the `appdebug` script. This imports the new Xilinx commands into the GDB that you can use to print the contents of OpenCL objects, namely `command_queue`, `cl_events`, and `cl_mem`.
 - a. If the problem you are seeing is a hang, then the host application is likely waiting for the command queue to finish or waiting on an event list.
 - b. Printing the command queue using `xprint` commands can tell you what events are unfinished and you can analyze the dependencies between the events.
3. You can step through the code under GDB and examine the OpenCL objects to verify that their contents are indeed as expected at any point in the code, similar to the way one debugs c/c++ code.

Kernel Debug

Kernels may be debugged using the `printf()` command in all three flows. Each flow also provides unique features and debug opportunities.

Using `printf()` to Debug Kernels

The SDAccel™ development environment supports OpenCL™ `printf()` kernel built-in function in all development flows: CPU emulation, hardware emulation, and running kernel in actual hardware.



IMPORTANT: `printf()` is not supported in C/C++ kernels.

Below is a kernel example of using `printf()` and the output when the kernel is executed with global size of 8:

```
__kernel __attribute__((reqd_work_group_size(1, 1, 1)))
void hello_world(__global int *a)
{
    int idx = get_global_id(0);

    printf("Hello world from work item %d\n", idx);
    a[idx] = idx;
}
```

Output is as follows:

```
Hello world from work item 0
Hello world from work item 1
Hello world from work item 2
Hello world from work item 3
Hello world from work item 4
```

```
Hello world from work item 5
Hello world from work item 6
Hello world from work item 7
```



IMPORTANT: *printf()* messages are buffered in the global memory and unloaded when kernel execution is completed. If *printf()* is used in multiple kernels, there is no guarantee that the order the messages from each kernel will be displayed on the host terminal.

Software Emulation Kernel Debug

To better mimic the hardware being emulated, software emulation kernels are spawned off as separate processes. If you are using GDB to debug the host code, breakpoints set on kernel lines will not be hit as the kernel code is not run within that process. To support the concurrent debugging of the host code as well as the kernel code, the SDAccel™ Environment provides a mechanism to attach to spawned kernels through the use of `sdx_server`.

In a Linux terminal, set up your environment to run emulation executables by setting all environment variables what is specified in the Emulation Environment table.

Before launching your host code, you must start the `sdx_server` with the appropriate options. In a separate Linux terminal, run the following command:

```
 ${XILINX_SDX}/Vivado/bin/sdx_server --sdx-url
```

After the `sdx_server` has started, run your host code as normal in a different Linux terminal. At this point, the `sdx_server` should specify a `gdb` listener port on standard out. Keep track of the number specified by the `sdx_server` as the `gdb` listener port is what will be used by GDB to debug the kernel process. When the `gdb` listener port is output, the spawned kernel process has attached to the `sdx_server` and is waiting for commands from you. To control this process, you must start a new instance of GDB and connect to the `sdx_server`.



IMPORTANT: If the `sdx_server` is running, then all spawned processes compiled for debug will connect and wait for control from you. If no GDB ever attaches or provides commands, the kernel code appears to hang.

In a third terminal, run the following command:

```
 ${XILINX_SDX}/lnx64/tools/gdb/gdb-7.9.1/bin/gdb
```

At the GDB prompt, run the following commands:

```
 ${XILINX_SDX}/data/emulation/cpu_em/generic_pcie/model/genericpciemodel
target remote :xxx
```

Where `xxx` is the number specified by the `sdx` server as the `gdb` listener port.

Once these two commands are executed, you can set breakpoints on your kernels as needed, run the `continue` command, and debug your kernel code. When the all kernel invocations have finished, the host code should continue, and the connection will be dropped.

When debugging CPU emulation kernels in the SDAccel Environment, these steps are handled automatically and the kernel process is automatically attached, providing multiple contexts to debug both the host code and kernel code simultaneously.

HW Emulation Kernel Debug Waveforms

Hardware Emulation can be used to analyze the performance of your system before compiling and running on a board. SDx™ provides two features that can enhance this performance analysis:

- **Waveform Debug:** A live simulation waveform is provided to show the details of kernel operations and data transfers.
- **Data Mining:** The simulation results are mined for critical results and shown in the profile summary under a **Kernel Internals** tab

Steps to Run:

1. Compile your host code.
2. Compile your kernels for Hardware emulation using the --debug option.
3. Create an `sdaccel.ini` file in the same directory of your host executable.
 - a. Use batch mode for data mining or use the SDx Environment for waveform debug + data mining.
 - b. Add the following to the `sdaccel.ini` file:

```
[Emulation]
launch_waveform = <batch | gui>
```

4. View the results in the Waveform Viewer.
5. If you would like to view the results in the SDx Environment,
 - a. Convert your files to compatible formats:
 - **sda2protobuf:** `sdaccel_profile_summary.csv`, `sdaccel_profile_summary.xprf`, `sdaccel_profile_kernels.csv`
 - **sda2wdb:** `sdaccel_timeline_trace.csv`
 - b. In the SDx Environment, select **File > Import** and navigate to the `.xprf` and `.wdb` files created above.
6. If you would like to view the waveform database after a run is over, run the following command:

```
xsim --gui <device>-<design>.wdb
```

For a given device and binary container, there exists the following hierarchy in a waveform debug view:

- Kernel Data Transfer: This section shows AXI transfers at the OCL masters to the DDR. Data transfers from all compute units funnel through these interfaces. Note that these interfaces may be a different bit width than the compute units. If so, then the burst lengths would be different (e.g., a burst of sixteen 32-bit words at a compute unit would be a burst of one 512-bit word at the OCL master).
- Kernel *<kernel name> <workgroup size>* Compute Unit *<CU name>*:
 - CU Stalls (%): This section shows a summary of stalls for the entire compute unit (CU). A bus of all lowest-level stall signals is created, and the bus is represented in the waveform as a percentage (%) of those signals that are active at any point in time.
 - Loop Pipeline Activity: This section shows the top-level loop pipeline signals for a CU. This section is only populated for flat CUs.
 - Data Transfers: This section shows the data transfers for all AXI masters on the CU.
 - User Functions: This section lists all of the functions within the hierarchy of the CU.
 - Function: *<function name>*
 - Function Stalls: This section lists the three stall signals within this function.
 - Loop Pipeline Activity: This section shows the function-level loop pipeline signals for a CU. The number of pipeline regions is dictated by Vivado® HLS and can typically be identified as compute sections of loops.
 - Function I/O: This section lists the I/O for the function. These I/O are of protocol `m_axi`, `ap_fifo`, `ap_memory`, or `ap_none`.

SDAccel Environment Supported Devices

SDAccel™ solutions are compiled against a target device. A device is the combination of board and infrastructure components on which the kernels of an application is executed. Applications compiled against a target device can only be executed on the board associated with the target device. Devices can be provided by SDAccel ecosystem partners, FPGA design teams, and Xilinx. Contact your Xilinx representative for more information about adding third-party devices in the SDAccel environment.

Refer to the *SDx Environments Release Notes, Installation, and Licensing Guide* ([UG1238](#)) for the most up-to-date list of supported devices.

OpenCL Built-In Functions Support in the SDAccel Environment

The OpenCL™ C programming language provides a rich set of built-in functions for scalar and vector operations. Many of these functions are similar to the function names provided in common C libraries but they support scalar and vector argument types. The SDAccel™ development environment is OpenCL 1.0 embedded profile compliant. The following tables show descriptions of built-in functions in OpenCL 1.0 embedded profile and their support status in the SDAccel environment.

Work-Item Functions

Function	Description	Supported
get_global_size	Number of global work items	Yes
get_global_id	Global work item ID value	Yes
get_local_size	Number of local work items	Yes
get_local_id	Local work item ID	Yes
get_num_groups	Number of work groups	Yes
get_group_id	Work group ID	Yes
get_work_dim	Number of dimensions in use	Yes

Math Functions

Function	Description	Supported
acos	Arc Cosine function	Yes
acosh	Inverse Hyperbolic Cosine function	Yes
acospi	acos(x)/PI	Yes
asin	Arc Cosine function	Yes
asinh	Inverse Hyperbolic Cosine function	Yes
asinpi	Computes acos (x) / pi	Yes
atan	Arc Tangent function	Yes
atan2(y, x)	Arc Tangent of y / x	Yes
atanh	Hyperbolic Arc Tangent function	Yes

Function	Description	Supported
atanpi	Computes atan (x) / pi	Yes
atan2pi	Computes atan2 (y, x) / pi	Yes
cbrt	Compute cube-root	Yes
ceil	Round to integral value using the round to +ve infinity rounding mode.	Yes
copysign(x, y)	Returns x with its sign changed to match the sign of y.	Yes
cos	Cosine function	Yes
cosh	Hyperbolic Cosine function	Yes
cospi	Computes cos (x * pi)	Yes
erf	The error function encountered in integrating the normal distribution	Yes
erfc	Complementary Error function	Yes
exp	base- e exponential of x	Yes
exp2	Exponential base 2 function	Yes
exp10	Exponential base 10 function	Yes
expm1	$\exp(x) - 1.0$	Yes
fabs	Absolute value of a floating-point number	Yes
fdim(x, y)	$x - y$ if $x > y$, $+0$ if x is less than or equal to y .	Yes
floor	Round to integral value using the round to -ve infinity rounding mode.	Yes
fma(a, b, c)	Returns the correctly rounded floating-point representation of the sum of c with the infinitely precise product of a and b. Rounding of intermediate products shall not occur. Edge case behavior is per the IEEE 754-2008 standard.	Yes
fmax(x, y) fmax(x, float y)	Returns y if x is less than y, otherwise it returns x. If one argument is a NaN, fmax() returns the other argument. If both arguments are NaNs, fmax() returns a NaN	Yes
fmin(x, y) fmin(x, float y)	Returns y if y less than x, otherwise it returns x. If one argument is a NaN, fmax() returns the other argument. If both arguments are NaNs, fmax() returns a NaN.	Yes
fmod	Modulus. Returns $x - y * \text{trunc}(x/y)$	Yes
fract	Returns $fmin(x - \text{floor}(x), 0x1.fffffep-1f)$. floor(x) is returned in iptr	Yes
frexp	Extract mantissa and exponent from x. For each component the mantissa returned is a float with magnitude in the interval [1/2, 1] or 0. Each component of x equals mantissa returned * 2^{exp} .	Yes

Function	Description	Supported
hypot	Computes the value of the square root of $x^2 + y^2$ without undue overflow or underflow.	Yes
ilogb	Returns the exponent as an integer value.	Yes
ldexp	Multiply x by 2 to the power n.	Yes
lgamma	Returns the natural logarithm of the absolute value of the gamma function. The sign of the gamma function is returned in the signp argument of lgamma_r.	Yes
lgamma_r	Returns the natural logarithm of the absolute value of the gamma function. The sign of the gamma function is returned in the signp argument of lgamma_r.	Yes
log	Computes natural logarithm.	Yes
log2	Computes a base 2 logarithm	Yes
log10	Computes a base 10 logarithm	Yes
log1p	$\text{log}(1.0+x)$	Yes
logb	Computes the exponent of x, which is the integral part of $\text{log} x $.	Yes
mad	Approximates $a * b + c$. Whether or how the product of $a * b$ is rounded and how supernormal or subnormal intermediate products are handled is not defined. mad is intended to be used where speed is preferred over accuracy30	Yes
modf	Decompose a floating-point number. The modf function breaks the argument x into integral and fractional parts, each of which has the same sign as the argument. It stores the integral part in the object pointed to by iptr.	Yes
nan	Returns a quiet NaN. The nancode may be placed in the significand of the resulting NaN.	Yes
nextafter	Next representable floating-point value following x in the direction of y	Yes
pow	Computes x to the power of y	Yes
pown	Computes x to the power of y, where y is an integer.	Yes
powr	Computes x to the power of y, where x is greater than or equal to 0.	Yes
remainder	Computes the value r such that $r = x - n*y$, where n is the integer nearest the exact value of x/y . If there are two integers closest to x/y , n shall be the even one. If r is zero, it is given the same sign as x.	Yes
remquo	Floating point remainder and quotient function.	Yes
rint	Round to integral value (using round to nearest even rounding mode) in floating-point format.	Yes
rootn	Compute x to the power $1/y$.	Yes
round	Return the integral value nearest to x rounding halfway cases away from zero, regardless of the current rounding direction.	Yes
rsqrt	Inverse Square Root	Yes
sin	Computes the sine	Yes

Function	Description	Supported
sincos	Computes sine and cosine of x. The computed sine is the return value and computed cosine is returned in cosval.	Yes
sinh	Computes the hyperbolic sine	Yes
sinpi	Computes sin (pi * x).	Yes
sqrt	Computes square root.	Yes
tan	Computes the tangent.	Yes
tanh	Computes hyperbolic tangent.	Yes
tanpi	Computes tan(pi * x).	Yes
tgamma	Computes the gamma.	Yes
trunc	Round to integral value using the round to zero rounding mode.	Yes
half_cos	Computes cosine. x must be in the range -216... +216. This function is implemented with a minimum of 10-bits of accuracy	Yes
half_divide	Computes x / y. This function is implemented with a minimum of 10-bits of accuracy	Yes
half_exp	Computes the base- e exponential of x. implemented with a minimum of 10-bits of accuracy	Yes
half_exp2	The base- 2 exponential of x. implemented with a minimum of 10-bits of accuracy	Yes
half_exp10	The base- 10 exponential of x. implemented with a minimum of 10-bits of accuracy	Yes
half_log	Natural logarithm. implemented with a minimum of 10-bits of accuracy	Yes
half_log10	Base 10 logarithm. implemented with a minimum of 10-bits of accuracy	Yes
half_log2	Base 2 logarithm. implemented with a minimum of 10-bits of accuracy	Yes
half_powr	x to the power of y, where x is greater than or equal to 0.	Yes
half_recip	Reciprocal. Implemented with a minimum of 10-bits of accuracy	Yes
half_rsqrt	Inverse Square Root. Implemented with a minimum of 10-bits of accuracy	Yes
half_sin	Computes sine. x must be in the range -2^16... +2^16. implemented with a minimum of 10-bits of accuracy	Yes
half_sqrt	Inverse Square Root. Implemented with a minimum of 10-bits of accuracy	Yes
half_tan	The Tangent. Implemented with a minimum of 10-bits of accuracy	Yes
native_cos	Computes cosine over an implementation-defined range. The maximum error is implementation-defined.	Yes
native_divide	Computes x / y over an implementation-defined range. The maximum error is implementation-defined	Yes

Function	Description	Supported
native_exp	Computes the base- e exponential of x over an implementation-defined range. The maximum error is implementation-defined.	Yes
native_exp2	Computes the base- 2 exponential of x over an implementation-defined range. The maximum error is implementation-defined.	No
native_exp10	Computes the base- 10 exponential of x over an implementation-defined range. The maximum error is implementation-defined.	No
native_log	Computes natural logarithm over an implementation-defined range. The maximum error is implementation-defined.	Yes
native_log10	Computes a base 10 logarithm over an implementation-defined range. The maximum error is implementation-defined.	No
native_log2	Computes a base 2 logarithm over an implementation-defined range.	No
native_powr	Computes x to the power of y, where x is greater than or equal to 0. The range of x and y are implementation-defined. The maximum error is implementation-defined.	No
native_recip	Computes reciprocal over an implementation-defined range. The maximum error is implementation-defined.	No
native_rsqrt	Computes inverse square root over an implementation-defined range. The maximum error is implementation-defined.	No
native_sin	Computes sine over an implementation-defined range. The maximum error is implementation-defined.	Yes
native_sqrt	Computes inverse square root over an implementation-defined range. The maximum error is implementation-defined.	No
native_tan	Computes tangent over an implementation-defined range. The maximum error is implementation-defined	Yes

Integer Functions

Function	Description	Supported
abs	$ x $	Yes
abs-diff	$ x-y $ without modulo overflow	Yes
add_sat	$x+y$ and saturate result	Yes
hadd	$(x+y) \gg 1$ without modulo overflow	Yes
rhadd	$(x+y+1) \gg 1$. The intermediate sum does not modulo overflow.	Yes
clz	Number of leading 0-bits in x	Yes
mad_hi	$\text{mul_hi}(a,b)+c$	Yes
mad24	(Fast integer function.) Multiply 24-bit integer then add the 32-bit result to 32-bit integer	Yes
mad_sat	$a*b+c$ and saturate the result	Yes

Function	Description	Supported
max	The greater of x or y	Yes
min	The lesser of x or y	Yes
mul_hi	High half of the product of x and y	Yes
mul24	(Fast integer function.) Multiply 24-bit integer values a and b	Yes
rotate	result[<i>indx</i>]= <i>v</i> [<i>indx</i>]<< <i>i</i> [<i>indx</i>]	Yes
sub_sat	x - y and saturate the result	Yes
upsample	result[i] = ((gentype)hi[i] << 8 16 32) lo[i]	Yes

Common Functions

Function	Description	Supported
clamp	Clamp x to range given by min, max	Yes
degrees	radians to degrees	Yes
max	Maximum of x and y	Yes
min	Minimum of x and y	Yes
mix	Linear blend of x and y	Yes
radians	degrees to radians	Yes
sign	Sign of x	Yes
smoothstep	Step and interpolate	Yes
step	0.0 if x < edge, else 1.0	Yes

Geometric Functions

Function	Description	Supported
clamp	Clamp x to range given by min, max	Yes
degrees	radians to degrees	Yes
cross	Cross product	Yes
dot	Dot product only float, double, half data types	Yes
dstance	Vector distance	Yes
length	Vector length	Yes
normalize	Normal vector length 1	Yes
fast_distance	Vector distance	Yes
fast_length	Vector length	Yes
fast_normalize	Normal vector length 1	Yes

Relational Functions

Function	Description	Supported
isequal	Compare of $x == y$.	Yes
isnotequal	Compare of $x != y$.	Yes
isgreater	Compare of $x > y$.	Yes
isgreaterequal	Compare of $x >= y$.	Yes
isless	Compare of $x < y$.	Yes
islessequal	Compare of $x <= y$.	Yes
islessgreater	Compare of $(x < y) (x > y)$.	Yes
isfinite	Test for finite value.	Yes
isinf	Test for +ve or -ve infinity.	Yes
isnan	Test for a NaN.	Yes
isnormal	Test for a normal value.	Yes
isordered	Test if arguments are ordered.	Yes
isunordered	Test if arguments are unordered.	Yes
signbit	Test for sign bit.	Yes
any	1 if MSB in any component of x is set; else 0.	Yes
all	1 if MSB in all components of x is set; else 0.	Yes
bitselect	Each bit of result is corresponding bit of a if corresponding bit of c is 0.	Yes
select	For each component of a vector type, $\text{result}[i] = \text{if MSB of } c[i] \text{ is set ? } b[i] : a[i]$ For scalar type, $\text{result} = c ? b : a$.	Yes

Vector Data Load and Store Functions

Function	Description	Supported
vloadn	Read vectors from a pointer to memory.	Yes
vstoren	Write a vector to a pointer to memory.	Yes
vload_half	Read a half float from a pointer to memory.	Yes
vload_halfn	Read a half float vector from a pointer to memory.	Yes
vstore_half	Convert float to half and write to a pointer to memory.	Yes
vstore_halfn	Convert float vector to half vector and write to a pointer to memory.	Yes
vloada_halfn	Read half float vector from a pointer to memory.	Yes
vstorea_halfn	Convert float vector to half vector and write to a pointer to memory.	Yes

Synchronization Functions

Function	Description	Supported
barrier	All work-items in a work-group executing the kernel on a processor must execute this function before any are allowed to continue execution beyond the barrier.	Yes

Explicit Memory Fence Functions

Function	Description	Supported
mem_fence	Orders loads and stores of a work-item executing a kernel	Yes
read_mem_fence	Read memory barrier that orders only loads	Yes
write_mem_fence	Write memory barrier that orders only stores	Yes

Async Copies from Global to Local Memory, Local to Global Memory Functions

Function	Description	Supported
async_work_group_copy	Must be encountered by all work-items in a workgroup executing the kernel with the same argument values; otherwise the results are undefined.	Yes
wait_group_events	Wait for events that identify the async_work_group_copy operations to complete.	Yes
prefetch	Prefetch bytes into the global cache.	No

PIPE Functions

Function	Description	Supported
read_pipe	Read packet from pipe	Yes
write_pipe	Write packet to pipe	Yes
reserve_read_pipe	Reserve entries for reading from pipe	No
reserve_write_pipe	Reserve entries for writing to pipe	No
commit_read_pipe	Indicates that all reads associated with a reservation are completed	No
commit_write_pipe	Indicates that all writes associated with a reservation are completed	No
is_valid_reserve_id	Test for a valid reservation ID	No

Function	Description	Supported
work_group_reserve_read_pipe	Reserve entries for reading from pipe	No
work_group_reserve_write_pipe	Reserve entries for writing to pipe	No
work_group_commit_read_pipe	Indicates that all reads associated with a reservation are completed	No
work_group_commit_write_pipe	Indicates that all writes associated with a reservation are completed	No
get_pipe_num_packets	Returns the number of available entries in the pipe	Yes
get_pipe_max_packets	Returns the maximum number of packets specified when pipe was created	Yes

Pipe Functions enabled by the cl_khr_subgroups extension

Function	Description	Supported
sub_group_reserve_read_pipe	Reserve entries for reading from a pipe	No
sub_group_reserve_write_pipe	Reserve entries for writing to a pipe	No
sub_group_commit_read_pipe	Indicates that all reads associated with a reservation are completed	No
sub_group_commit_write_pipe	Indicates that all writes associated with a reservation are completed	No

OpenCL 2.0 Image Objects

Table 4: OpenCL 2.0 Image Options

Function	Description	Supported
clCreateImage	Create an image object for a 1D image, 1D image buffer, 1D image array, 2D image, 2D image array or 3D image.	Yes
clGetSupportedImageFormats	Get the list of image formats supported by an OpenCL implementation when the Context, Image type (1D, 2D, or 3D image, 1D image buffer, 1D or 2D image array) and Image object allocation information of the image memory object is specified.	Yes
clEnqueueReadImage	Enqueue commands to read from an image or image array object to host memory.	Yes
clEnqueueWriteImage	Enqueue commands to write to an image or image array object from host memory.	Yes

Function	Description	Supported
clEnqueueFillImage	Enqueues a command to fill an image object with a specified color.	No
clEnqueueCopyImageToBuffer	Enqueues a command to copy an image object to a buffer object.	No
clEnqueueMapImage	Enqueues a command to map a region in an image object into the host address space and returns a pointer to this mapped region.	No
clGetImageInfo	Obtain information specific to an image object created with clCreateImage. To get information that is common to all memory objects, use the clGetMemObjectInfo function.	Yes

xbinst Command Reference

The FPGA acceleration card plugged into the host machine needs to have the associated Linux kernel driver, firmware and runtime libraries installed before it can be used for running user applications. SDAccel™ provides a Xilinx board installation utility, `xbinst` to generate all necessary files for the platform support package for the FPGA card. It also generates an installation script to compile and install the driver, firmware and runtime libraries.

The `xbinst` utility requires superuser privileges on the host machine to run. The supported options are listed below:

Table 5: xbinst Options

Short Option	Long Option	Valid Values	Description
-h	--help	NA	Print Usage Message
-f <arg>	--platform <arg>	Supported platform from SDAccel installation or the full path to platform definition file for custom platforms	Required. All installed platforms are listed in the <i>SDx Release Notes, Installation, and Licensing Guide (UG1238)</i> .
-d <arg>	--destination <arg>	Valid path on the file system.	Required destination directory for driver and firmware for the specified platform

Follow the instructions below to install the driver and firmware for the FPGA card on the host machine. The ADM-PCIE-7V3 DSA is used as an example. Replace it with the DSA for the actual card plugged into the system.

All commands need to be run with superuser privileges.

1. Create a board installation directory.

```
$sudo mkdir 7v3_dsa
```

2. Run xbinst to generate all necessary files.

```
$ sudo xbinst -f xilinx:adm-pcie-7v3:1ddr:3.0 -d 7v3_dsa
***** xbinst v2017.1_sdx
**** Copyright 1986-2017 Xilinx, Inc. All Rights Reserved.

INFO: [XBINST 60-267] Packaging for PCIe...
INFO: [XBINST 60-268] Packaging for PCIe...COMPLETE
INFO: [XBINST 60-667] xbinst has successfully created a board
installation directory
at /opt/dsa/7v3_dsa
```

If you installed a custom platform, the full path to the platform package file needs to be provided to the xbinst command as shown in the following example:

```
$ sudo xbinst -f /platform/repo/vendor_board_name_version.xpfm -d
custom_platform
```

3. Install the driver, firmware, and runtime libraries.

```
$ cd 7v3_dsa/xbinst/pkg/pcie/
$ sudo ./install.sh
```

This will do the following:

- Compile and install Linux kernel device drivers.
- Install the firmware files to the Linux firmware area.
- Generate a `setup.sh` (Bash) or `setup.csh` (for csh/tcsh) to set up the runtime environment. Users must source the setup script before running any application on the target FPGA card.

Xilinx Board Swiss Army Knife Utility

Xilinx Board Swiss Army Knife (xbsak) utility is a standalone command line utility that can perform the following board administration and debug tasks independent of SDAccel runtime library:

- Board administration tasks:
 - Flash PROM.
 - Reboot boards without rebooting the host.
 - Reset hung boards.
 - Query board status, sensors and PCIe AER registers.
- Debug operations:
 - Download SDAccel binary (.xclbin) to FPGA.
 - DMA test for PCIe bandwidth.
 - Show status of compute units.

The xbsak utility is automatically added to your path by using the `setup.csh` file created by the `xbinst` utility. For example:

```
$XILINX_SDX/bin/xbinst -f xilinx:adm-pcie-7v3:1ddr:3.0 -d .
cd xbinst
source setup.csh
xbsak -help
```

xbsak Commands and Options

The following are xbsak command line format, and details of commands and options.

Syntax:

```
xbsak <command> [options]
```

Commands and Options

- `help` - Print help messages.

- `list` - List all supported devices installed on the server in the format of [`device_id`] : `device_name`.

The following is an example output where the device ID is 0 and the device name is `xilinx:xil-accel-rd-ku115:4ddr-xpr:3.3`.

```
[0] xilinx:xil-accel-rd-ku115:4ddr-xpr:3.3
```

- `query [-d device_id] [-r region_id]`

Query the specified device and programmable region on the device to get detailed status information.

 - `-d device_id` - Specify the target device. Optional. Default=0 if not specified.

 - `-r region_id` - Specify the target region. Optional. Default=0 if not specified.

- `boot [-d device_id]`

- `clock [-d device_id] [-r region_id] -f clock1_freq [-g clock2_freq]`

Set frequencies of clocks driving the computing units.

 - `-d device_id` - Specify the target device. Optional. Default=0 if not specified.

 - `-r region_id` - Specify the target region. Optional. Default=0 if not specified.

 - `-f clock1_freq` - Specify clock frequency in MHz for the first clock. Required. All platforms have this clock.

 - `-g clock2_freq` - Specify clock frequency in MHz for the second clock. Optional: Some platforms have this clock to support IP based kernels.

- `dmatest [-d device_id] [-b blocksize]`

Test throughput of data transfer between the host machine and global memory on the device.

 - `-d device_id` - Specify the target device. Optional. Default=0 if not specified.

 - `-b blocksize` - Specify the test block size in KB. Optional: Default=65536 or 64MB if not specified. The block size can be specified in both decimal or hexadecimal formats. e.g. Both `-b 1024` and `-b 0x400` set the block size to 1024KB or 1MB.

- `flash [-d device_id] -m primary_mcs [-n secondary_mcs]`

Program PROMs on the device with specified configuration files.

- `-d device_id` - Specify the target device. Optional. Default=0 if not specified.
- `-m primary_mcs` - Specify the primary configuration file. Required. All platforms have at least one PROM.
- `-n secondary_mcs` - Specify the secondary configuration file. Optional. Some platform have two PROMs and the secondary configuration file is required for the second PROM.



IMPORTANT:

The flash programming function requires certain hardware features in the platform to work, so it only works with devices already programmed with the supported firmware:

Board	Platform Firmware
ADM-PCIE-7V3	<code>xilinx:adm-pcie-7v3:1ddr:3.0 or newer</code>
ADM-PCIE-KU3	<ul style="list-style-type: none"> ◦ <code>xilinx:adm-pcie-ku3:2ddr-xpr:3.3 or newer</code> ◦ <code>xilinx:adm-pcie-ku3:2ddr:3.3 or newer</code> ◦ <code>xilinx:adm-pcie-ku3:1ddr:3.3 or newer</code>
XIL-ACCEL-RD-KU115	<code>xilinx:xil-accel-rd-ku115:4ddr-xpr:3.2 or newer</code>

- `program [-d device_id] [-r region_id] -p xclbin`

Download the OpenCL binary to the programmable region on the device.

- `-d device_id` - Specify the target device. Optional. Default=0 if not specified.
- `-r region_id` - Specify the target region. Optional. Default=0 if not specified.
- `-p xclbin` - Specify the OpenCL binary file. Required.

- `reset [-d device_id] [-r region_id]`

Reset the programmable region on the device. All running compute units in the region will be stopped and reset.

- `-d device_id` - Specify the target device. Optional. Default=0 if not specified.
- `-r region_id` - Specify the target region. Optional. Default=0 if not specified.

- status [--apm | --lapc]

Displays the status of any AXI Performance Monitor (apm) or Lightweight AXI Protocol Checkers (lapc) that are available in the base platform.

- --apm - Returns the value of the AXI Performance Monitor (apm) counters. This option is only applicable if one or more AXI Performance Monitors are available in the base platform.
- --lapc - Returns the values of the violations codes detected by the Lightweight AXI Protocol Checkers (lapc). This option is only applicable if one or more Lightweight AXI Protocol Checkers are available in the base platform.

- - scan

Scans the system and displays any Xilinx PCIe devices, associated drivers and pertinent system information.

Using the Runtime Initialization File

SDAccel™ runtime library uses various parameters to control debug, profiling, and message logging during host application and kernel execution in CPU emulation, hardware emulation, and system run on the acceleration board. These control parameters are specified in a runtime initialization file.

For command line users, the runtime initialization file needs to be created manually. The file must be named `sdaccel.ini` and saved in the same directory as the host executable.

For SDx GUI users, the project manager creates `sdaccel.ini` file automatically based on users run configuration and saves it next to the host executable.

The runtime library will check if `sdaccel.ini` exists in the same directory as the host executable and automatically read the parameters from the file during start-up if it finds it.

Runtime Initialization File Format

The runtime initialization file is a text file with groups of keys and their values. Any line beginning with semicolon (;) or hash (#) is a comment. The group names, keys, and key values are all case sensitive.

The following is a simple example that turns on profile timeline trace and sends the runtime log messages to console.

```
#Start of Debug group
[Debug]
timeline_trace = true

#Start of Runtime group
[Runtime]
runtime_log = console
```

The following table lists all supported groups, keys, valid key values and short descriptions on the function of the keys.

Key	Valid Values	Descriptions
[Debug] Group		

Key	Valid Values	Descriptions
debug	[true false]	Enable or disable kernel debug. <ul style="list-style-type: none"> • true: enable • false: disable • Default: false
profile	[true false]	Enable or disable OpenCL code profiling. <ul style="list-style-type: none"> • true: enable • false: disable • Default: true
timeline_trace	[true false]	Enable or disable profile timeline trace <ul style="list-style-type: none"> • true: enable • false: disable • Default: false
device_profile	[true false]	Enable or disable device profiling. <ul style="list-style-type: none"> • true: enable • false: disable • Default: false
[Runtime] Group		
api_checks	[true false]	Enable or disable OpenCL API checks. <ul style="list-style-type: none"> • true: enable • false: disable • Default: true
runtime_log	null console syslog filename	Specify where the runtime logs are printed <ul style="list-style-type: none"> • null: Do not print any logs. • console: Print logs to <code>stdout</code> • syslog: Print logs to Linux syslog • filename: Print logs to the specified file. e.g. <code>runtime_log=my_run.log</code> • Default: null

Key	Valid Values	Descriptions
polling_throttle	An integer	<p>Specify the time interval in microseconds that the runtime library polls the device status.</p> <ul style="list-style-type: none"> • Default: 0
[Emulation] Group		
aliveness_message_interval	Any integer	<p>Specify the interval in seconds that aliveness messages need to be printed</p> <ul style="list-style-type: none"> • Default 300
print_infos_in_console	[true false]	<p>Controls the printing of emulation info messages to users console. Emulation info messages are always logged into a file called <code>emulation_debug.log</code></p> <ul style="list-style-type: none"> • true = print in users console • false = don't print in users console • Default: true
print_warnings_in_console	[true false]	<p>Controls the printing emulation warning messages to users console. Emulation warning messages are always logged into a file called <code>emulation_debug.log</code>.</p> <ul style="list-style-type: none"> • true = print in users console • false = do not print in users console • Default: true
print_errors_in_console	[true false]	<p>Controls printing emulation error messages in users console. Emulation error messages are always logged into file called <code>emulation_debug.log</code>.</p> <ul style="list-style-type: none"> • true = print in users console • false = don't print in users console • Default: true

Key	Valid Values	Descriptions
enable_oob	[true false]	Enable or disable diagnostics of out of bound access during emulation. A warning is reported if there is any out of bound access. <ul style="list-style-type: none"> • true: enable • false: disable • Default: false
launch_waveform	[off batch gui]	Specify how the waveform is saved and displayed during emulation. <ul style="list-style-type: none"> • off: Do not launch simulator waveform GUI, and do not save <code>wdb</code> file • batch: Do not launch simulator waveform GUI, but save <code>wdb</code> file • gui: Launch simulator waveform GUI, and save <code>wdb</code> file • Default: off <p>NOTE: The kernel needs to be compiled with debug enabled (<code>xocc -g</code>) for the waveform to be saved and displayed in the simulator GUI.</p>

Converting Tcl Compilation Flow to XOCC

Starting with the 2016.3 release of SDAccel™, Tcl based compilation flow is no longer supported. Direct command line access via XOCC (and MakeFile) is the main entry point to use SDAccel services. This appendix provides guidance on how to convert Tcl based compilation script to XOCC based command line options. All examples in the SDAccel installation use Makefile/XCOCC for compilation and can be used as additional reference.

The following sections list Tcl commands and their equivalent XOCC options. All XOCC options need to be provided to the XCOCC command in a single command line.

Solution

- Tcl

```
create_solution -name example_alpha -dir . -force
```

- XOCC

XOCC does not require a solution to be explicitly created.

Host Code Management and Compilation

- Tcl

```
add_files "test-cl.c"
```

- XOCC

In XOCC flow, the host code is managed by the user and compiled with xcpp command line or Makefile to generate host executable.

```
xcpp -g -Wall -DFPGA_DEVICE -I/opt/SDx/2017.1/runtime/include/1_2 -c
test-cl.cpp -o test-cl.o
xcpp -L/opt/SDx/2017.1/runtime/lib/x86_64 -lxilinxopencl -lilmx6.0
-lstdc++ test-cl.o -o mmult_ex
```

Device

- Tcl

```
add_device -vbnv xilinx:adm-pcie-7v3:1ddr:3.0
set_property device_repo_paths /path/to/custom_dsa [current_solution]
```

- XOCC

```
--platform xilinx:adm-pcie-7v3:1ddr:3.0
--xp prop:solution.device_repo_paths=/path/to/custom_dsa
```

Kernel Definition

- Tcl

```
create_kernel mmult -type clc
add_files -kernel [get_kernels mmult] "mmult1.cl"
```

- XOCC

```
--kernel mmult mmult1.cl
```

Setting Kernel Compile Flags

- Tcl

```
set_property kernel_flags "-DUSE2DDR=1" [get_kernels mmult]
```

- XOCC

```
-DUSE2DDR=1
```

Binary Container Definition

- Tcl

```
create_opencl_binary bin_mmult
set_property region "OCL_REGION_0" [get_opencl_binary bin_mmult]
create_compute_unit -opencl_binary [get_opencl_binary bin_mmult]
-kernel [get_kernels mmult] -name k1
create_compute_unit -opencl_binary [get_opencl_binary bin_mmult]
-kernel [get_kernels mmult] -name k2
```

- XOCC

```
--output bin_mmult.xclbin --nk mmult:2:k1.k2
```

Compile for CPU Emulation

- Tcl

```
compile_emulation -flow cpu -opencl_binary [get_opencl_binary bin_mmult]
```

- XOCC

```
--target sw_emu
```

Compile for Hardware Emulation

- Tcl

```
compile_emulation -flow hardware -opencl_binary [get_opencl_binary bin_mmult]
```

- XOCC

```
--target hw_emu
```

Run CPU and Hardware emulation Emulation

- Tcl

```
run_emulation -flow cpu -args "bin_mmult.xclbin"  
run_emulation -flow hardware -args "bin_mmult.xclbin"
```

- XOCC

Refer to [Running Software and Hardware Emulation in XOCC Flow](#) for details.

Build System

- Tcl

```
build_system
```

- XOCC

```
--target hw
```

Report Estimate

- Tcl

```
report_estimate
```

- XOCC

```
--report estimate
```

Package System

- Tcl

```
package_system
```

- XOCC

XOCC does not have an equivalent option for `package_system`. Packaging of libraries and drivers for deployment is provided by another command, `xbinst`. Below is a simple command line example.

Refer to the *SDx Environments Release Notes, Installation, and Licensing Guide* ([UG1238](#)).

```
xbinst -f xilinx:adm-pcie-7v3:1ddr:3.0 -d 7v3
```

Putting All Together

- **Tcl**

```
create_solution -name example_alpha -dir . -force
add_device -vbnv xilinx:adm-pcie-7v3:1ddr:3.0

create_kernel mmult -type clc
add_files -kernel [get_kernels mmult] "mmult1.cl"
set_property kernel_flags "-DUSE2DDR=1" [get_kernels mmult]

create_opencl_binary bin_mmult
set_property region "OCL_REGION_0" [get_opencl_binary bin_mmult]
create_compute_unit -opencl_binary [get_opencl_binary bin_mmult]
-kernel [get_kernels mmult] -name k1
create_compute_unit -opencl_binary [get_opencl_binary bin_mmult]
-kernel [get_kernels mmult] -name k2

compile_emulation -flow cpu -opencl_binary [get_opencl_binary bin_mmult]
run_emulation -flow cpu -args "bin_mmult.xclbin"

compile_emulation -flow hardware -opencl_binary [get_opencl_binary
bin_mmult]
run_emulation -flow cpu -args "bin_mmult.xclbin"

build_system
```

- **XOCC**

The following is the equivalent XOCC command line options of the Tcl commands above for compilation of CPU emulation. Change `--target sw_emu` to `--target hw_emu` for compilation for hardware emulation or `--target hw` for compilation for system run.

```
xocc --target sw_emu --platform xilinx:adm-pcie-7v3:1ddr:3.0 \
-DUSE2DDR=1 \
--output bin_mmult.xclbin --nk mmult1:2:k1.k2
--kernel mmult mmult1.c
```

SDAccel System Info Checker Utility

The SDAccel™ System Info Checker utility (`sdxsyschk`) performs analysis of system and hardware setup on the PCIe® acceleration cards that are supported by the Xilinx® SDAccel integrated environment.

When you launch the utility, it compiles and generates an informative status report. It also provides some basic debug and troubleshooting capabilities by issuing messages describing any problems.

The following is a list of analyses the utility performs:

- System and Environment Diagnosis
- Python Version Check
- Linux OS System Check
- 64-bit Architecture Check
- Environment Variables Check
- Motherboard System Info
- PCIe Diagnosis
- Xilinx PCIe Device Check
- Device Link Speed Check
- Root Port Speed Check
- Xilinx Kernel Driver Check
- Xilinx DSA-Device Matching Check

Launching System Info Checker Utility

The following requirements must be met before running the Info Checker Utility:

- You MUST have SDAccel™ installed and environment set up to run SDAccel tools.
- You MUST have superuser privileges (sudo or root access).
- You MUST have `lspci` package installed either under the `/sbin` or `/usr/bin` directory and the path to the `lspci` command added to your `PATH` environment variable.

Run the following command to launch the utility to query the system and get detailed PCIe® platform diagnosis.

```
$ sudo sdxsyschk
```

The following are a few examples showing how the System Info Checker Utility can be used.

- To query the system and basic platform detection status:

```
$sdxsyschk
```

- To query the system and detailed PCIe platform diagnosis:

```
$sudo sdxsyschk
```

- To query detailed analysis with verbose (optional) outputs:

- \$sudo sdxsyschk -v

- To query status and redirect the output to a text file specified with an argument:

```
$sudo sdxsyschk > <path_and_file_name>
```

- To query basic system status plus environment variables information.

This must run without sudo access.

```
$sdxsyschk -e
```

Board Installations

SDAccel™ applications are executed in hardware using one of the supported FPGA cards or boards listed below. You may develop and run applications on a single host computer, or you may develop applications on one host computer and run applications on another host computer. In all cases, if the application is to be executed on hardware, a board must be installed on the host computer and SDAccel must be installed on the host computer.

The procedure for installing a hardware board on a host computer is provided here and includes instructions for using the xbinst board installation utility and the Vivado® Design Suite to install the board. The xbinst utility, the Vivado Design Suite, and any required cable drivers are included when SDAccel is installed on the host computer.

Follow the installation instructions appropriate for your the acceleration card:

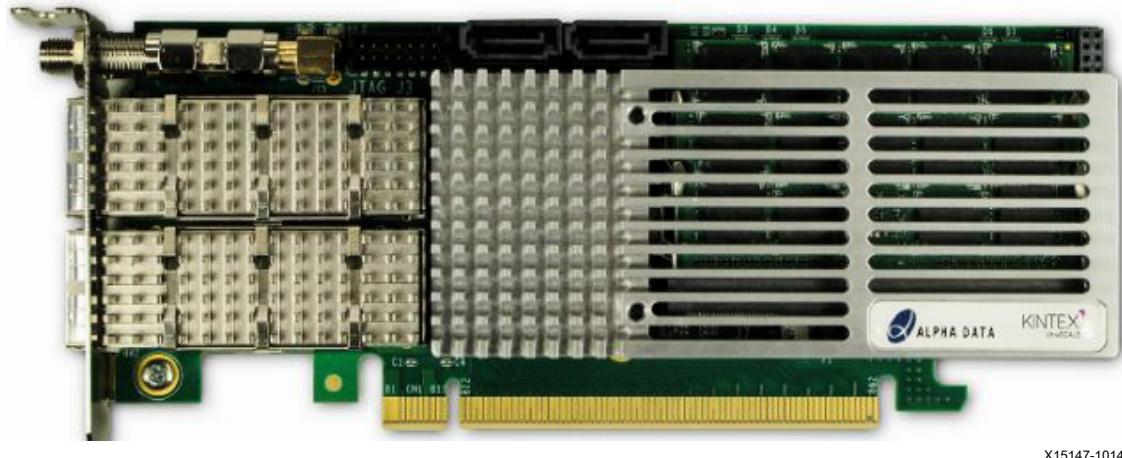
- [Installing the Alpha Data ADM-PCIE-KU3 Card](#)
- [Installing the Alpha Data ADM-PCIE-7V3 Card](#)
- [Installing the Alpha Data ADM-PCIE-8K5 Card](#)
- [Installing the Xilinx XIL-ACCEL-RD-KU115 Card](#)

Installing the Alpha Data ADM-PCIE-KU3 Card

The ADM-PCIE-KU3 card is a high-performance reconfigurable computing card for data center applications. It features:

- Kintex® UltraScale™ XCKU060T-2FFVA1156E FPGA
- 16 GB of DDR3 memory

Figure 22: ADM-PCIE-KU3/KU060E card



Step 1: Prepare Board Installation Files

1. SDAccel™ provides a utility, `xbinst`, that generates firmware and driver files for the target board plugged into the host computer. Run the commands below to prepare files for ADM-PCIE-KU3 card installation. See [xbinst Command Reference](#) for more details on the `xbinst` utility.

All commands must be run with `root` or `sudo` privilege.

```
$ sudo mkdir ku3_dsa
$ sudo xbinst -f xilinx:adm-pcie-ku3:2ddr:3.3 -d ku3_dsa
***** xbinst v2017.1
**** Copyright 1986-2017 Xilinx, Inc. All Rights Reserved.
INFO: [XBINST 60-267] Packaging for PCIe...
INFO: [XBINST 60-268] Packaging for PCIe...COMPLETE
INFO: [XBINST 60-667] xbinst has successfully created a board installation
directory
      at /opt/dsa/ku3_dsa
```

Make a note of the board installation directory. This procedure uses `/opt/dsa/ku3_dsa` as an example in this chapter.

Copy the following file path to the programming computer:

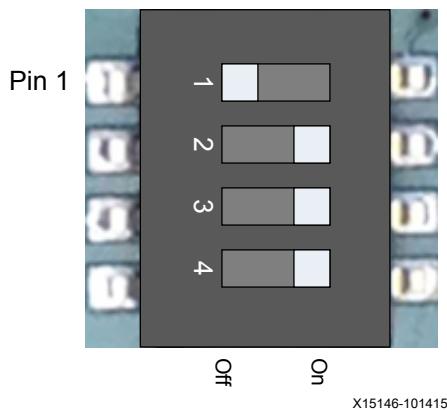
```
/opt/dsa/ku3_dsa/xbinst/pkg/pcie/firmware/xilinx_adm-pcie-ku3_2ddr_3_3.mcs
```

Make a note of the file location on the programming computer as it will be required for programming the configuration memory in a later step.

Step 2: Setting up the Card and Computer

1. Make sure the host computer is completely turned off.

- Locate DIP switch SW1 on the back side of the board near the center of top edge and set it as shown here:



- Install the ADM-PCIE-KU3 card into an open PCIe® slot in the host computer.

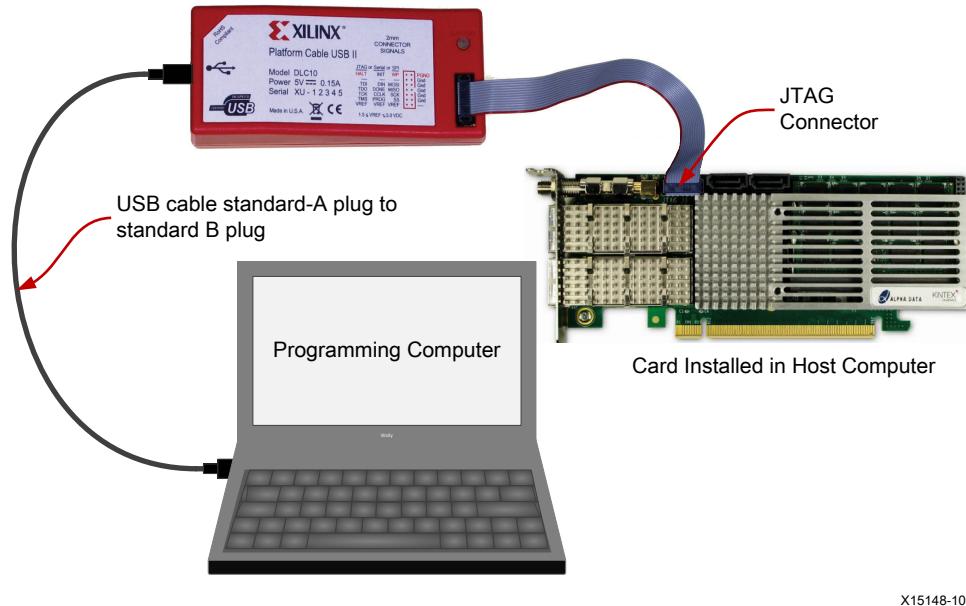
NOTE: Follow host computer manufacturer recommendations to ensure proper mounting and adequate cooling.

- Turn on the host computer.

Step 3: Programming the Base Platform

All applications compiled by the SDAccel™ compiler for the Alpha Data card are compiled against a specific device. A device is a combination of interfaces and infrastructure components on the card, which are required for proper execution of the user program. The base device program or firmware is different for all devices. This program must be loaded onto the FPGA before the user application is loaded. To program the firmware program do the following:

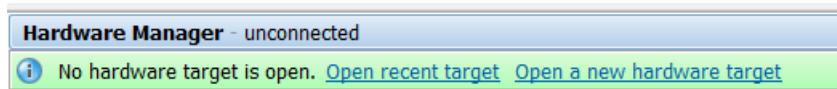
- Connect the Alpha Data ADM-PCIE-KU3 card to the programming computer with an installation of Vivado® Design Suite as shown here:



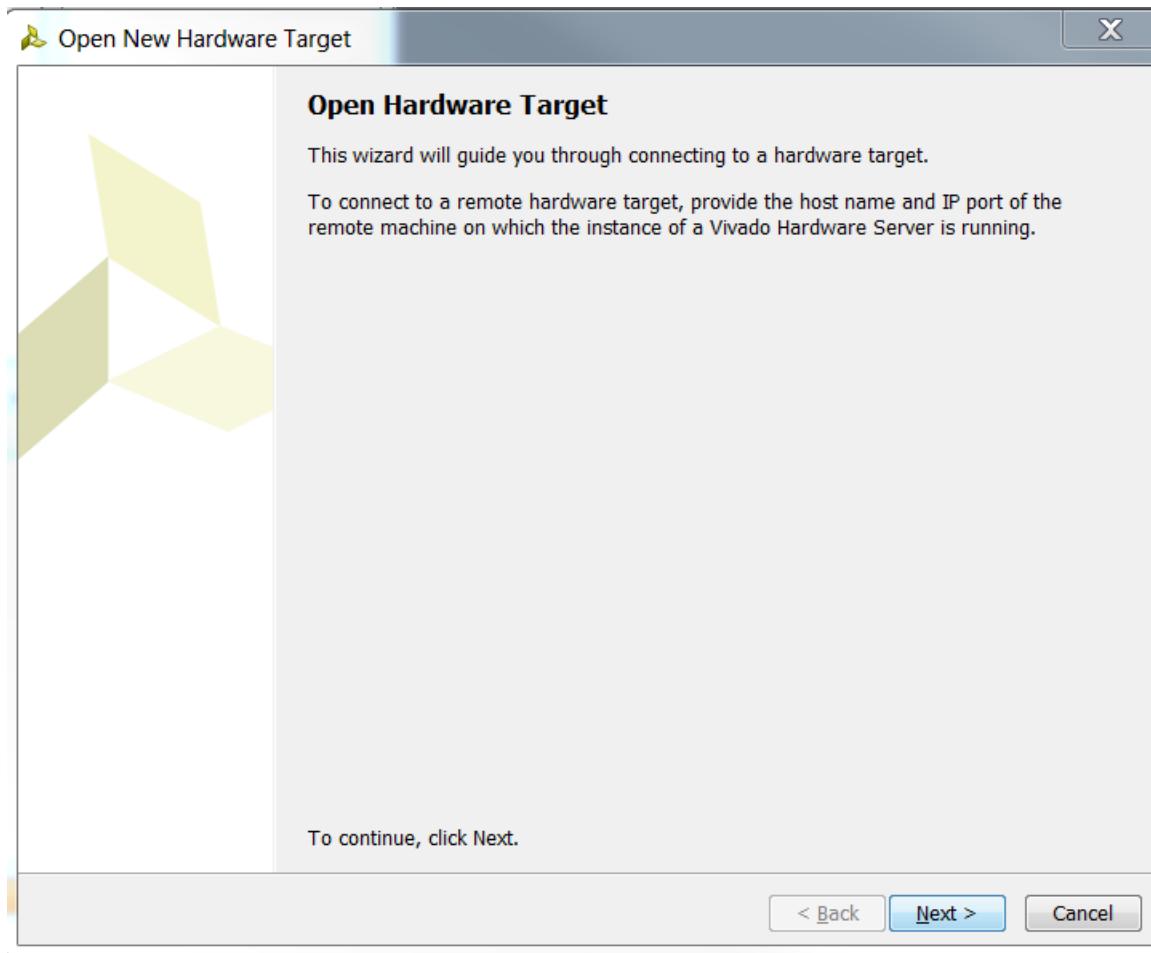
2. On the programming computer, start the Vivado Design Suite then on the **Welcome** page, select **Open Hardware Manager**.



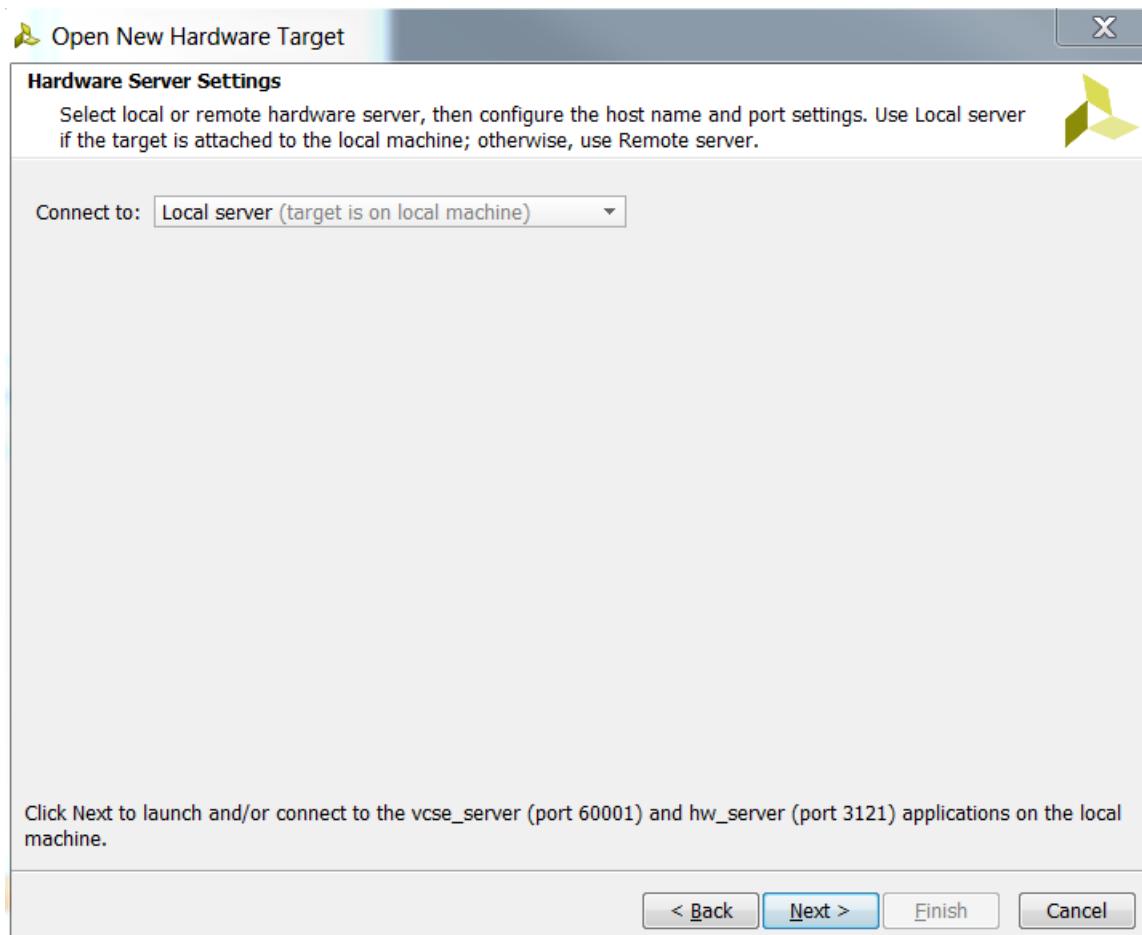
3. Select **Open a New Hardware Target**.



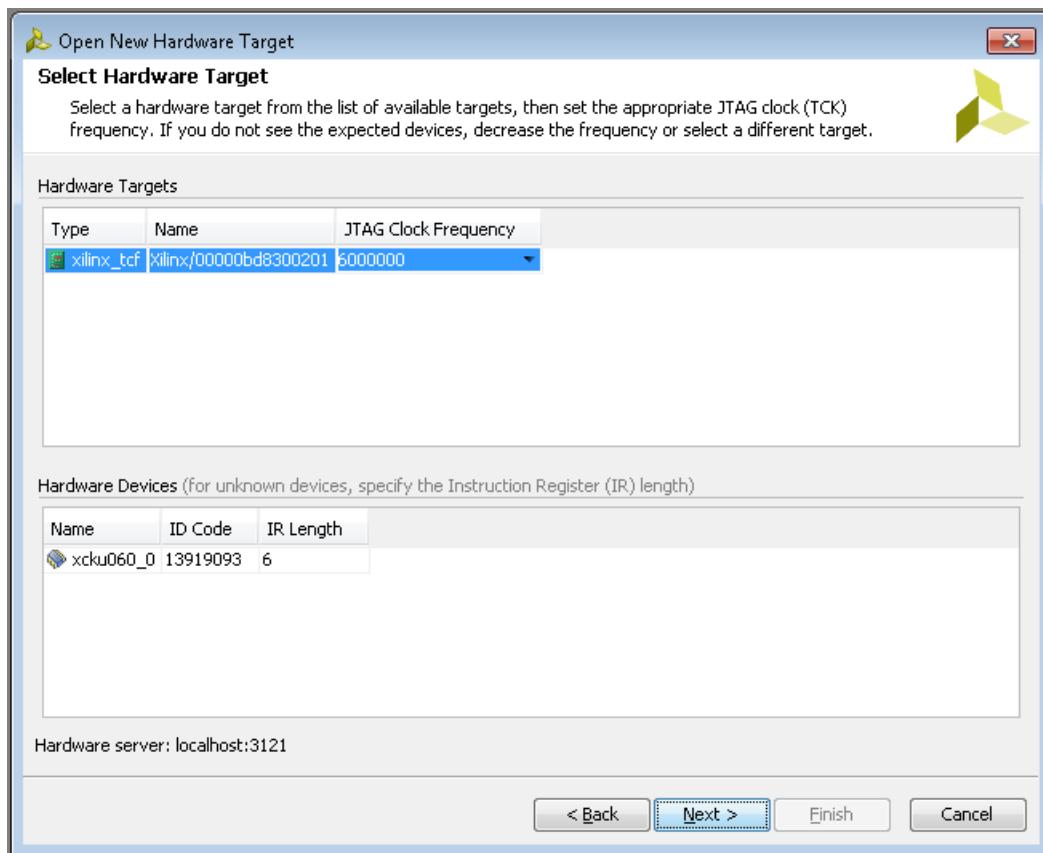
4. Click **Next** in the **Open Hardware Target** window.



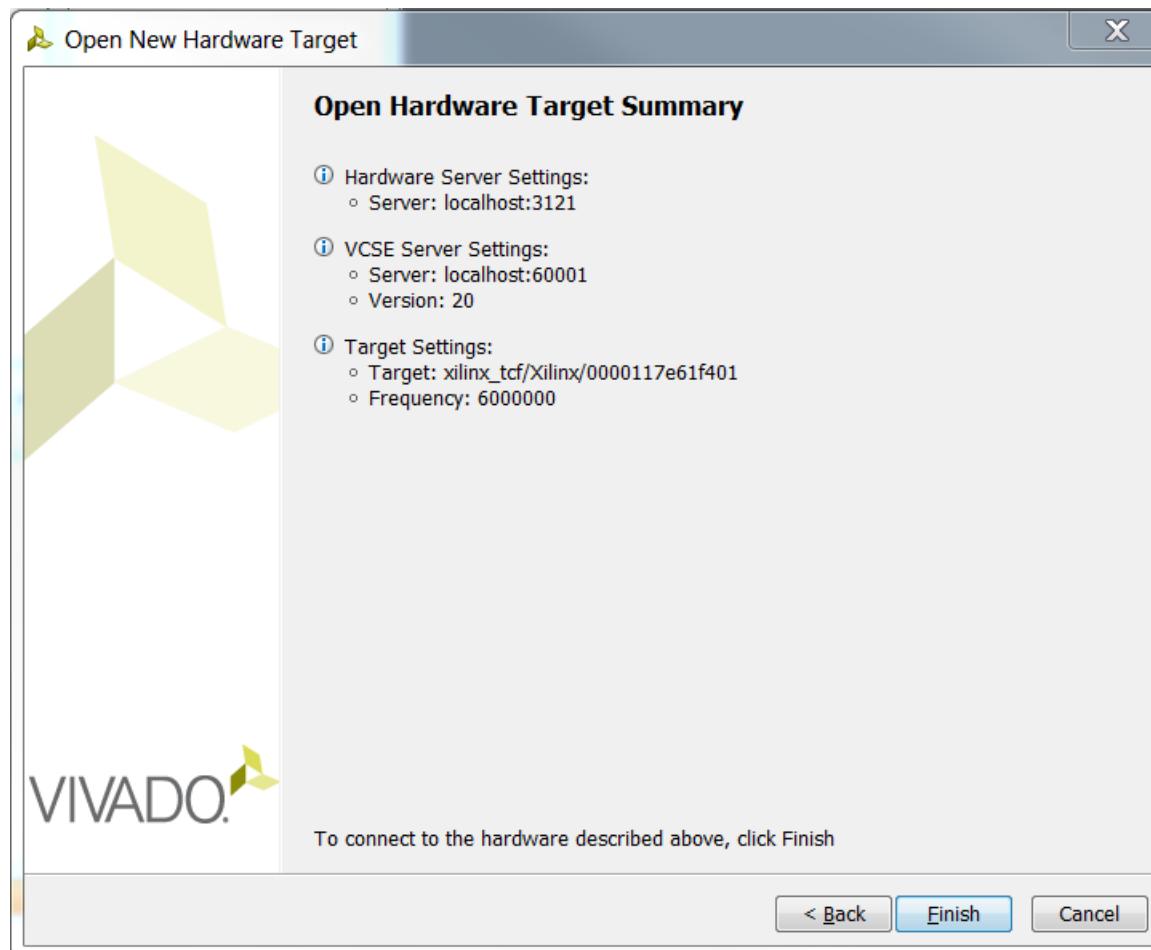
5. Select **Local server** in the **Connect to** field and click **Next**.



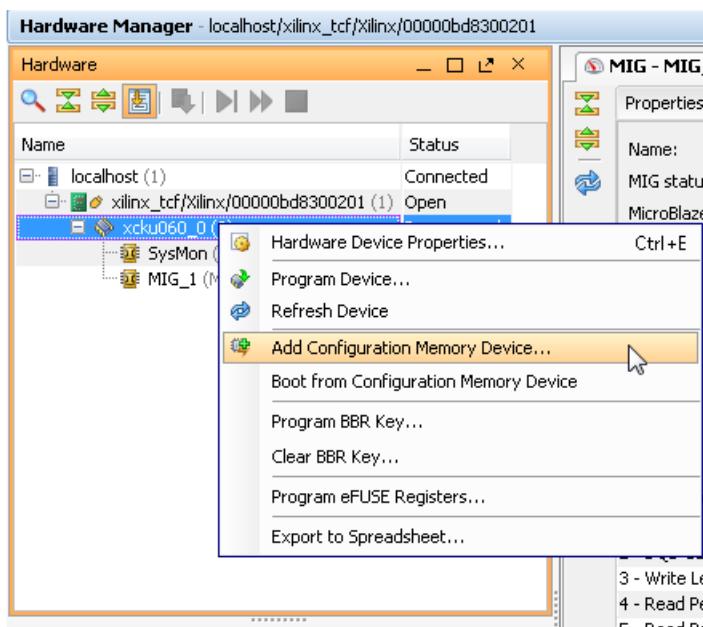
6. In the **Open New Hardware Target** window, select **xilinx_tcf** and click **Next**.



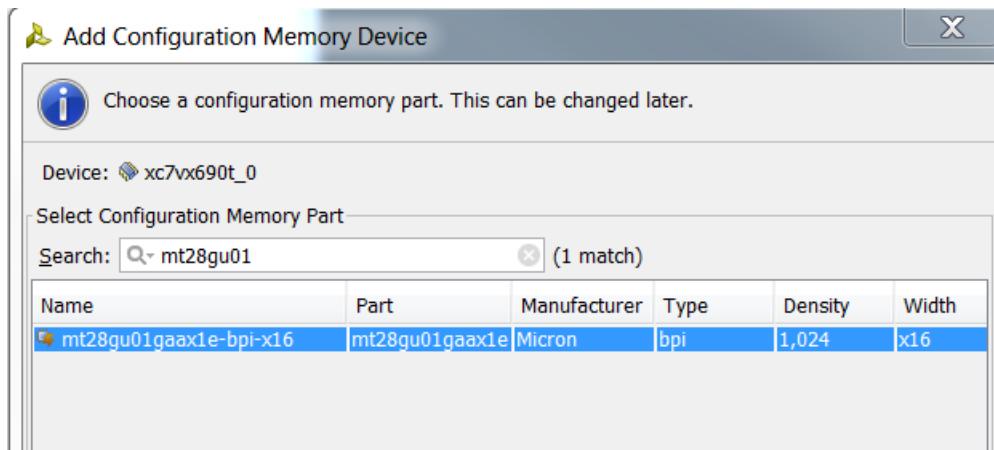
7. In the **Open Hardware Target Summary** window, click **Finish**.



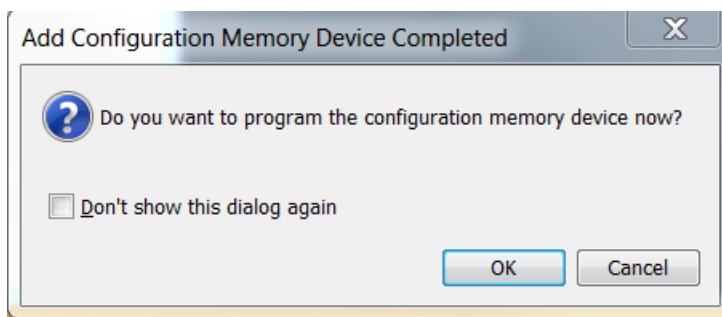
8. Right-click your FPGA (**xku060_0**), and select **Add Configuration Memory Device**.



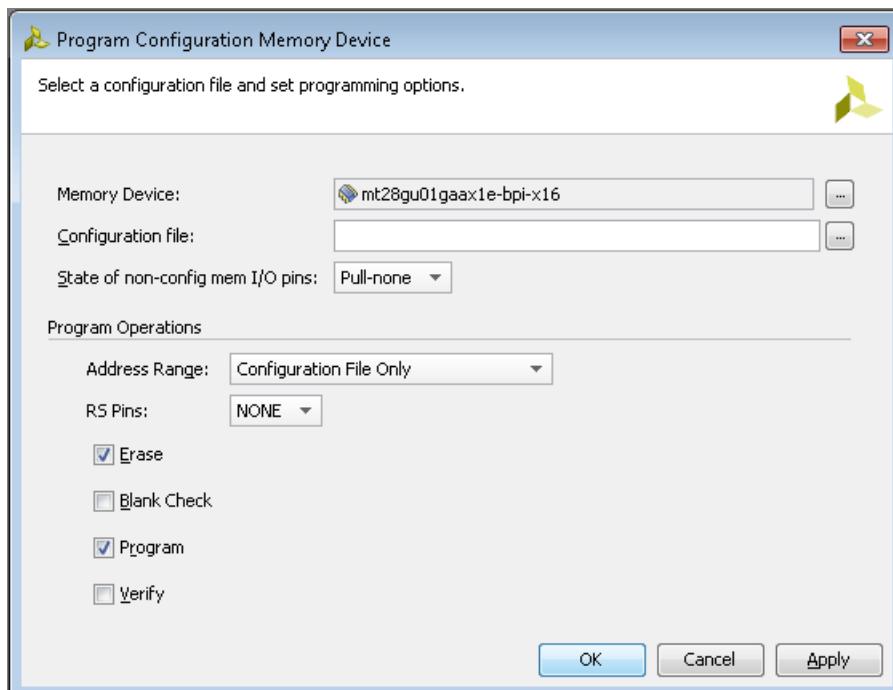
9. Select **mt28gu01gaax1e-bpi-x16** as the configuration memory.



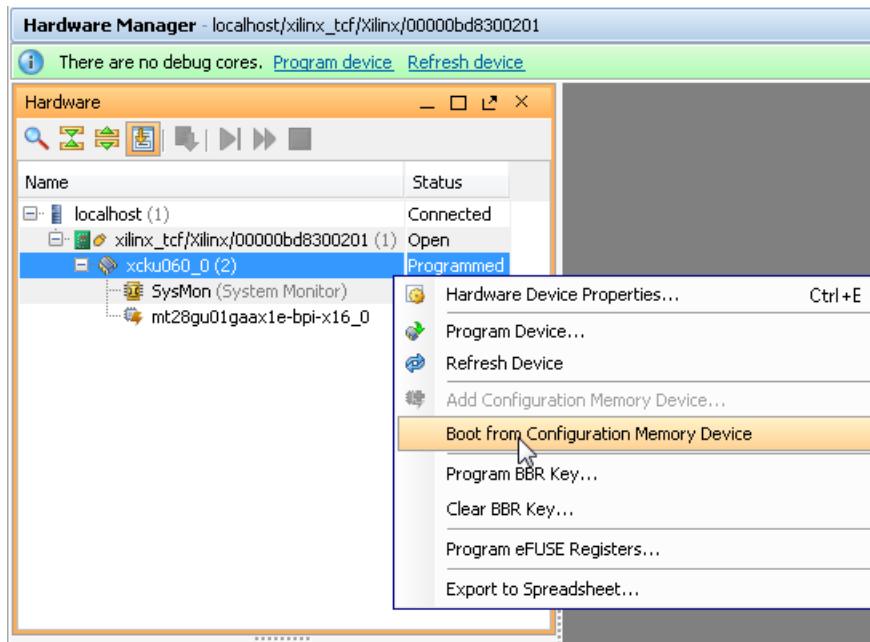
10. Click **OK** to program the configuration memory.



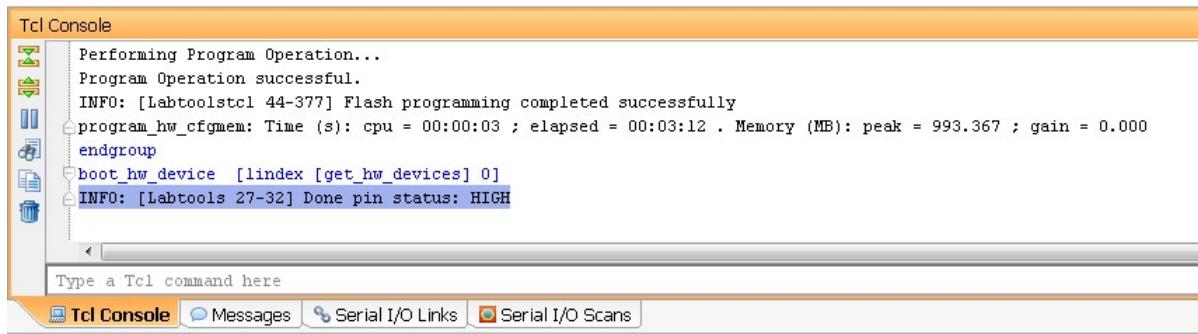
11. In the **Programming Configuration Memory Device** window, go to the **Configuration File** entry box, browse to and select the MCS file (**xilinx_adm-pcie-ku3_1ddr_3_0.mcs**) that you copied to the programming computer in Step 1. Verify all other settings as shown in the **Program Configuration Memory Device** window. Click **OK** to start programming the configuration memory.



12. After the memory has been configured, right-click the FPGA (**xku060_0**) and select **Boot from Configuration Memory Device**



The Tcl console displays Done pin status: HIGH after the FPGA device is booted successfully.



13. Reboot the host computer.

NOTE: Programming of the device firmware is required only once per device support archive (dsa). All applications targeting the same dsa can share a single programming instance of the card firmware.

Step 4: Installing Driver for the Card

You must install proper drivers for the card before you can use it to run SDAccel™ applications. Follow the instructions below to install the required drivers.

Change to the board installation directory generated in Step 1 and run installation script:

```
$cd /opt/dsa/ku3_dsa/xbininst/pkg/pcie
$sudo ./install.sh
```

This will do the following.

- Compile and install Linux kernel device drivers.
- Install the firmware to the Linux firmware area.
- Generate a `setup.sh` (Bash) or `setup.csh` (for csh/tcsh) to set up the runtime environment. You must source the setup script before running any application on the target FPGA card.
- Install Xilinx OpenCL Installable Client Driver (ICD) to `/etc/OpenCL/vendors`. The OpenCL ICD allows multiple implementations of OpenCL to co-exist on the same system. It allows applications to choose a platform from the list of installed platforms and dispatches OpenCL API calls to the underlying implementation.

Installing the Alpha Data ADM-PCIE-7V3 Card

The ADM-PCIE-7V3 card is a high-performance reconfigurable computing card for data center applications. It features:

- A Virtex®-7 XC7VX690T-2FFG1157C FPGA
- 16 GB of DDR3 memory

Figure 23: ADM-PCIE-7V3 card



X15029-091415

Step 1: Prepare Board Installation Files

SDAccel™ provides a utility, `xbinst`, that generates firmware and driver files for the target board plugged into the host computer. Run the commands below to prepare files for ADM-PCIE-7V3 card installation. See [xbinst Command Reference](#) for more details on `xbinst` utility.

All commands need to be run with `root` or `sudo` privilege.

```
$sudo mkdir 7v3_dsa
$ sudo xbinst -f xilinx:adm-pcie-7v3:1ddr:3.0 -d 7v3_dsa
***** xbinst v2017.1
***** Copyright 1986-2017 Xilinx, Inc. All Rights Reserved.
INFO: [XBINST 60-267] Packaging for PCIe...
INFO: [XBINST 60-268] Packaging for PCIe...COMPLETE
INFO: [XBINST 60-667] xbinst has successfully created a board installation
directory
at /opt/dsa/7v3_dsa
```

Make a note of the board installation directory. This procedure uses **/opt/dsa/7v3_dsa** in this chapter.

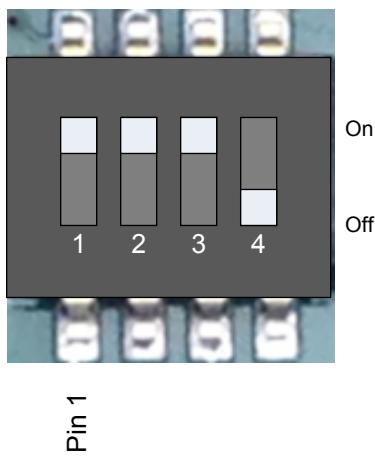
Copy the following file path to the programming computer:

```
/opt/dsa/7v3_dsa/xbinst/pkg/pcie/firmware/xilinx_adm-pcie-7v3_1ddr_3_0.mcs
```

Make a note the file location on the programming computer as it will be required for programming the configuration memory in later step.

Step 2. Setting Up the Card and Computer

1. Make sure the host computer is completely turned off.
2. Locate DIP switch SW1 on the back side of the board near the center of top edge and set it as shown here:


X15008-100615

3. Install the ADM-PCIE-7V3 card into an open PCIe slot in the host computer.

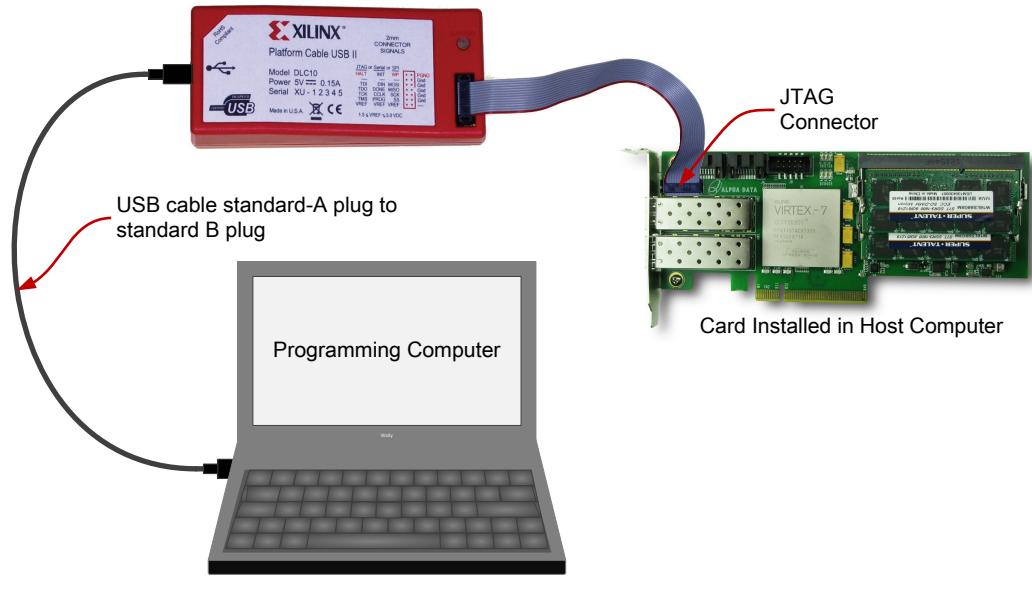
NOTE: Follow host computer manufacturer recommendations to ensure proper mounting and adequate cooling.

4. Turn on the host computer.

Step 3: Programming the Base Platform

All applications compiled by the SDAccel™ compiler for the Alpha Data card are compiled against a specific device. A device is a combination of interfaces and infrastructure components on the card, which are required for proper execution of the user program. The base device program or firmware is different for all devices. This program must be loaded onto the FPGA before the user application is loaded. To program the firmware program:

1. Connect the Alpha Data ADM-PCIE-7V3 card to the control computer having an installation of Vivado® Design Suite as shown here:



X15033-091415

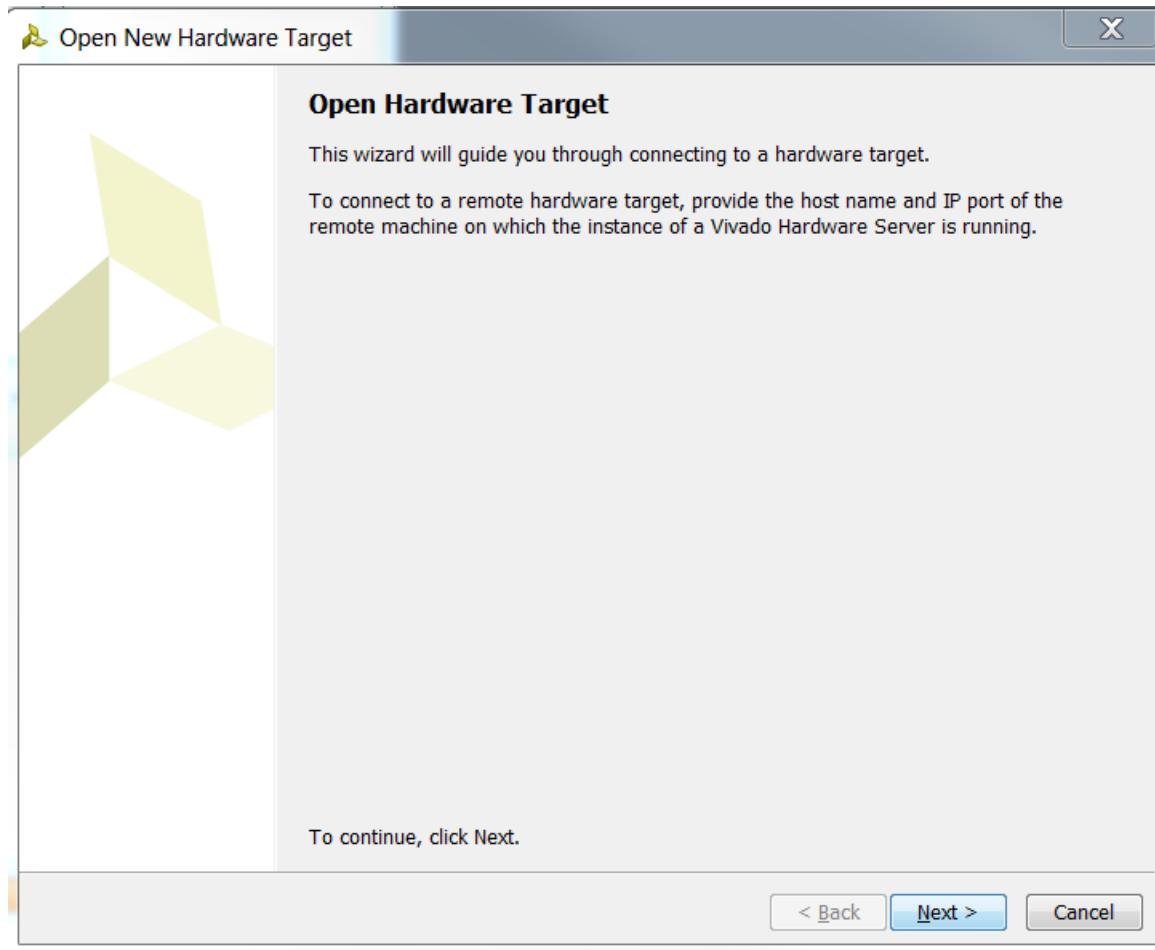
2. On the programming computer, **Open Hardware Manager**.



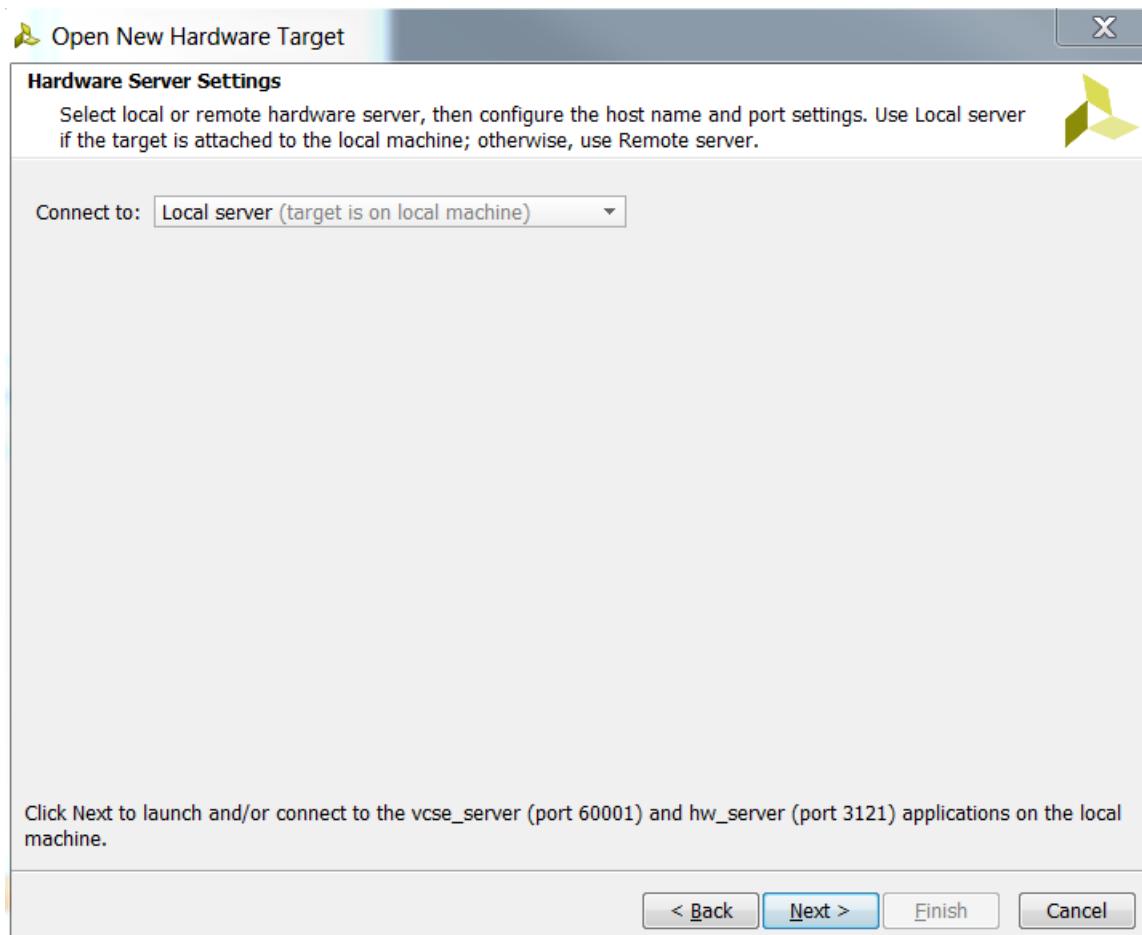
3. Select **Open a New Hardware Target**.



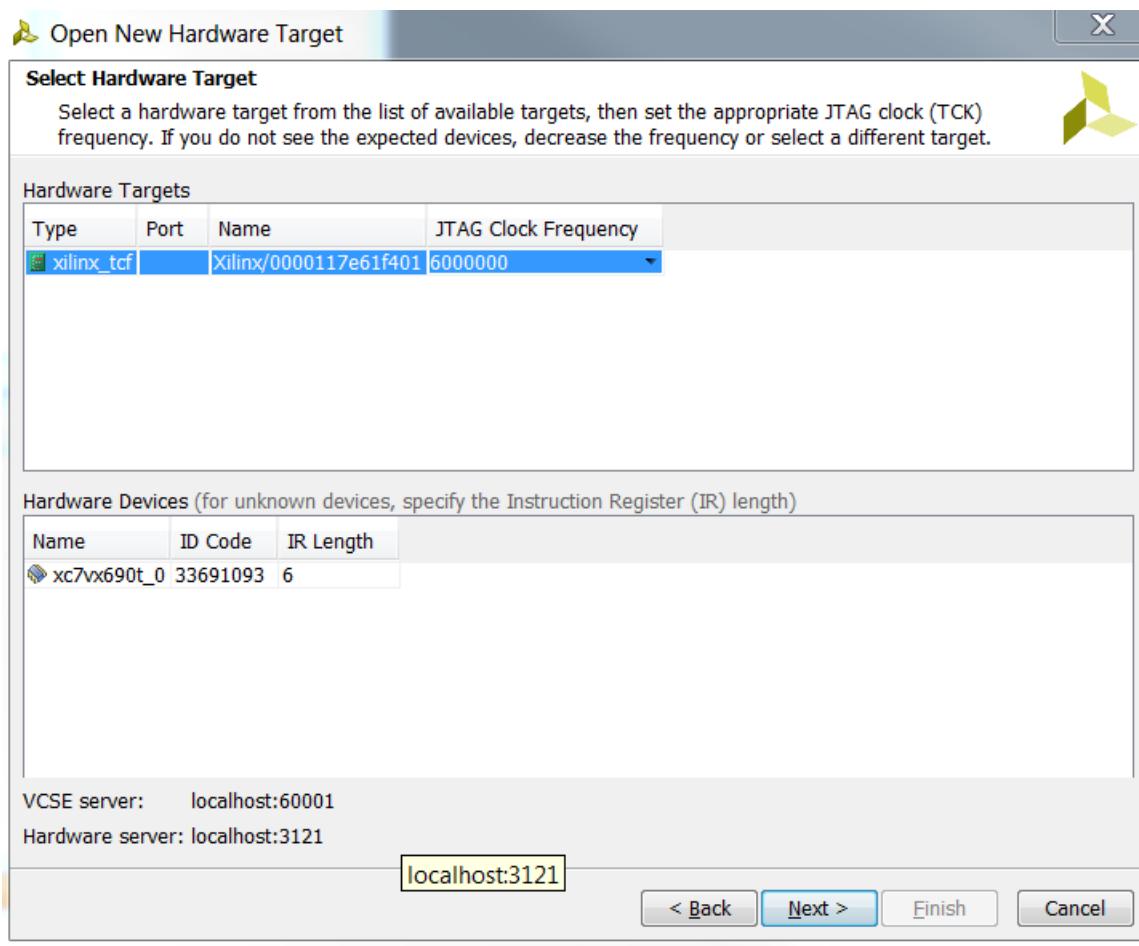
4. Click **Next** in the Open Hardware Target window.



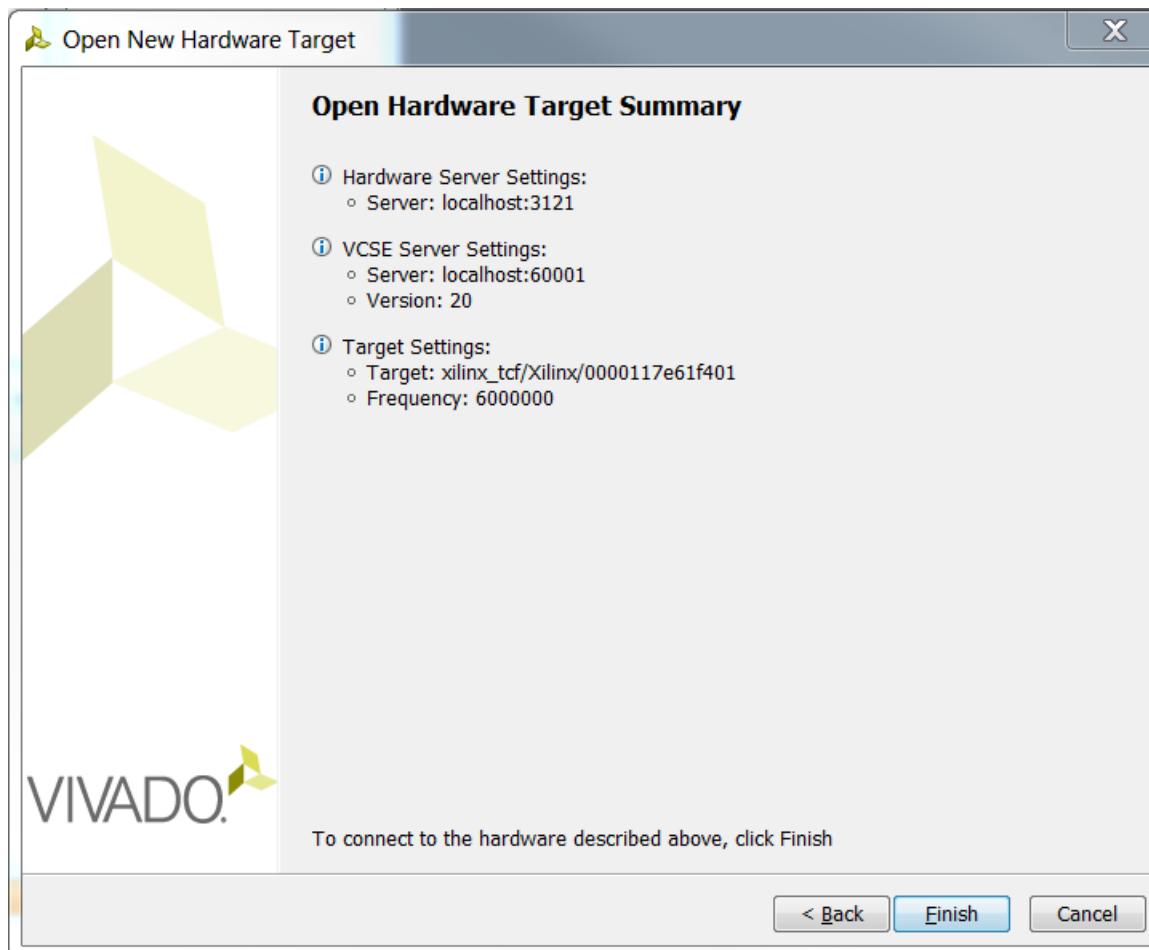
5. Select **Local server** in the **Connect to** field and click **Next**.



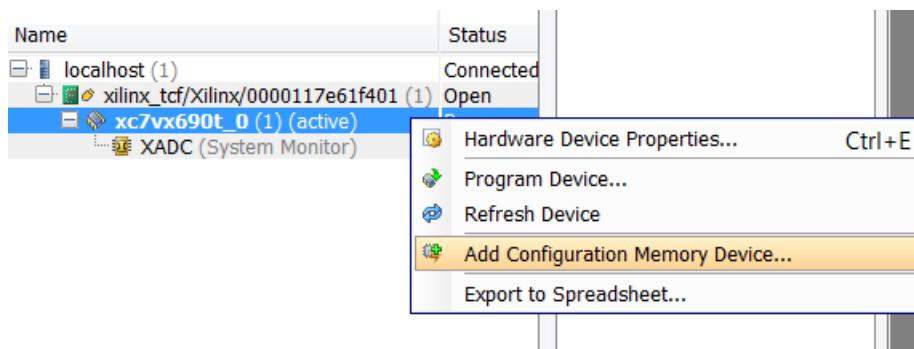
6. In the Open New Hardware Target window, select **xilinx_tcf** and click **Next**.



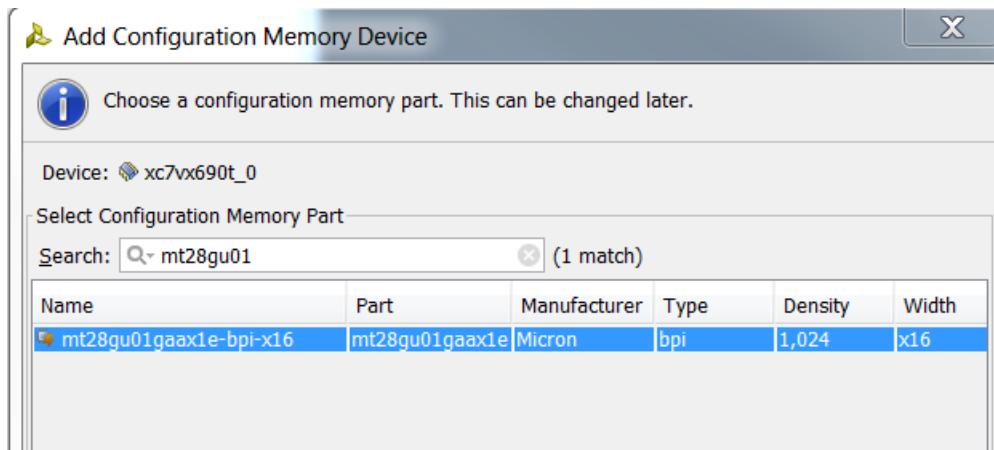
7. In the Open Hardware Target Summary window, click **Finish**.



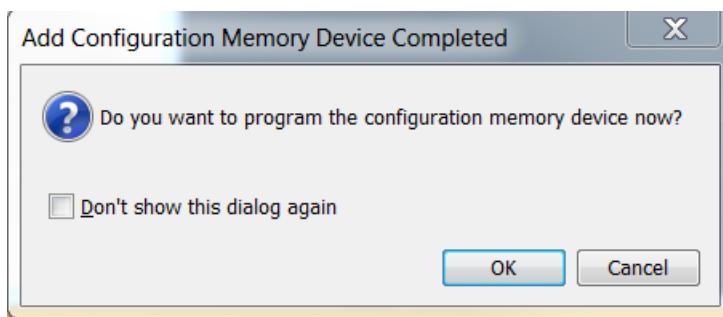
8. Right-click the FPGA (**xc7vx690t_0**) and select **Add Configuration Memory Device**.



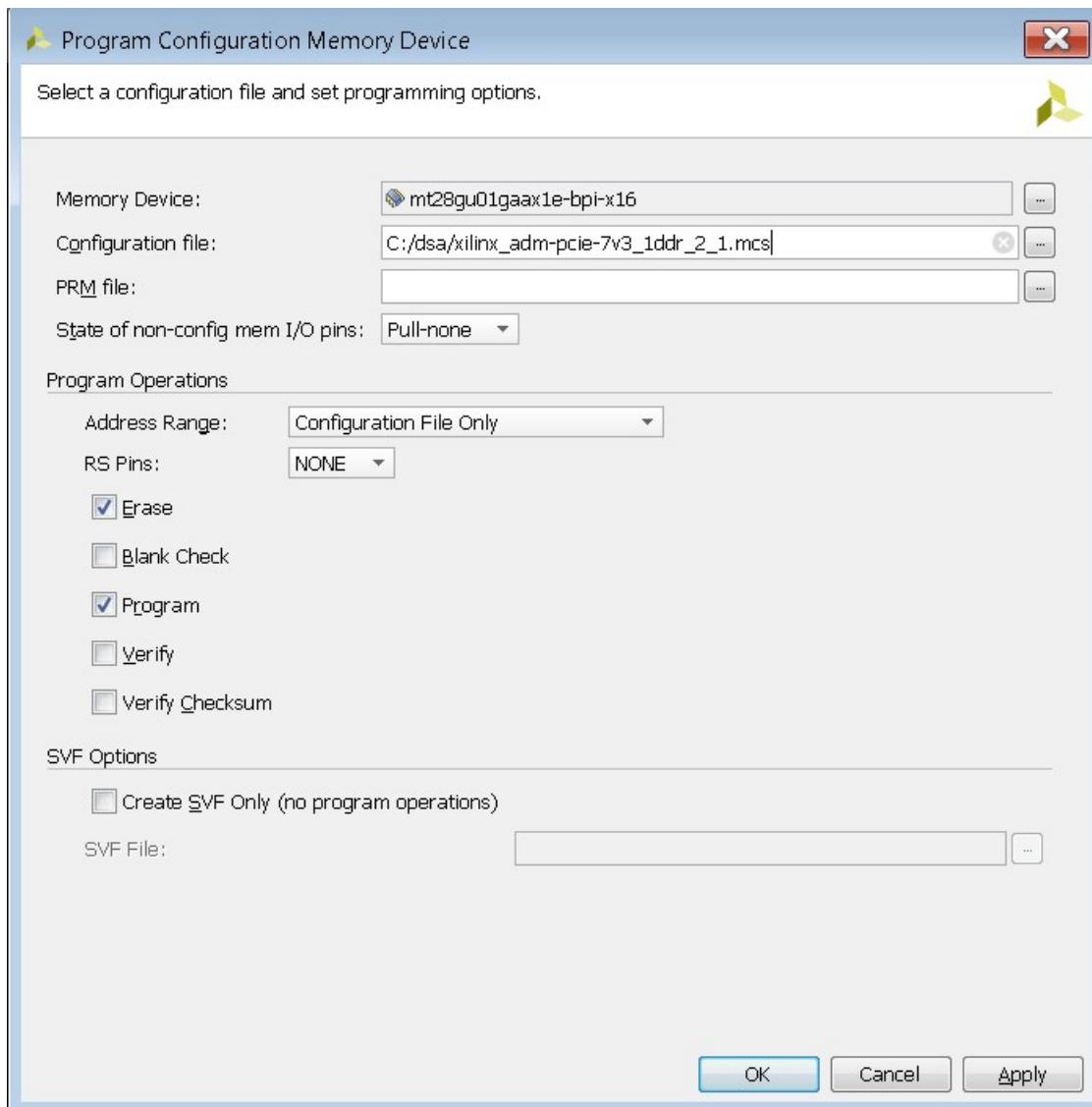
9. Select **mt28gu01gaax1e-bpi-x16** as the configuration memory.



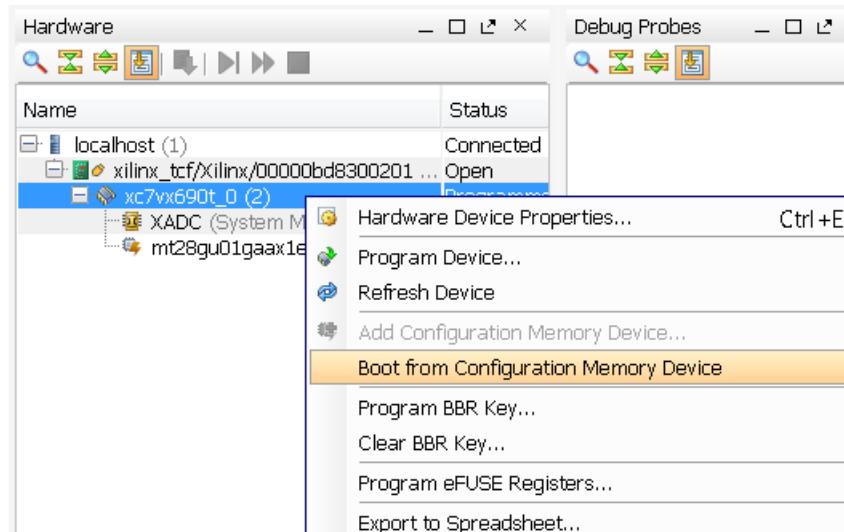
10. Click **OK** to program the configuration memory.



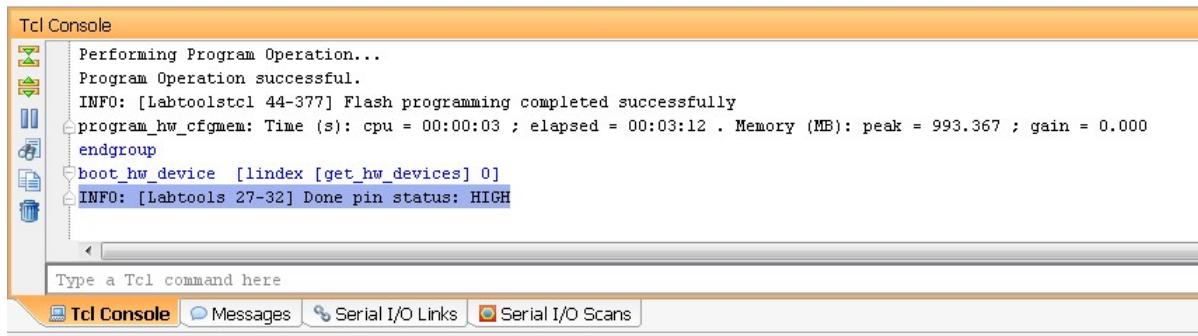
11. In the **Programming Configuration Memory Device** window, go to the **Configuration File** entry box, browse to, and select the MCS file (**xilinx_adm-pcie-7v3_1ddr_3_0.mcs**) that you copied to the programming computer in Step 1. Verify all other settings as shown in the **Program Configuration Memory Device** window. Click **OK** to start programming the configuration memory.



12. After the memory has been configured, right-click the FPGA (**xc7vx690t_0**) and select **Boot From Configuration Device**.



The Tcl console displays Done pin status: HIGH after the FPGA is booted successfully.



13. Reboot the host computer.

NOTE: Programming of the device firmware is required only once per device. All applications targeting the same device can share a single programming instance of the card firmware.

Step 4: Installing Driver for the Card

You must install proper drivers for the card before you can use it to run SDAccel™ applications. Follow the instructions below to install the required drivers.

Change to the board installation directory generated in Step 1, and run installation script:

```
$ cd /opt/dsa/7v3_dsa/xbininst/pkg/pcie
$ sudo ./install.sh
```

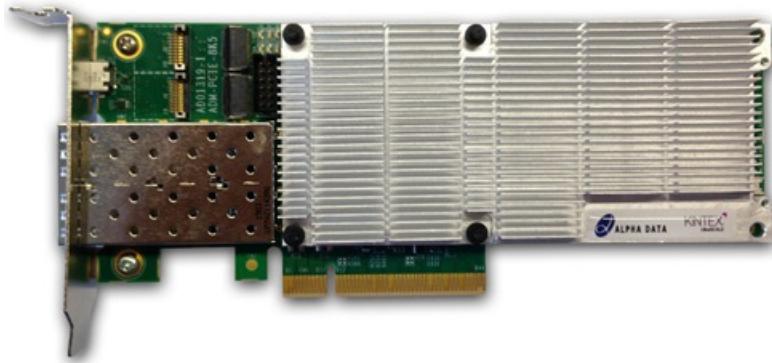
This will do the following:

- Compile and install Linux kernel device drivers.
- Install the firmware to the Linux firmware area.
- Generate a `setup.sh` (Bash) or `setup.csh` (for csh/tcsh) to set up the runtime environment. Users must source the setup script before running any application on the target FPGA card.
- Install Xilinx OpenCL Installable Client Driver (ICD) to `/etc/OpenCL/vendors`. The OpenCL ICD allows multiple implementations of OpenCL to co-exist on the same system. It allows applications to choose a platform from the list of installed platforms and dispatches OpenCL API calls to the underlying implementation.

Installing the Alpha Data ADM-PCIE-8K5 Card

The ADM-PCIE-8K5 card is a high-performance reconfigurable computing card for data center applications. It features:

- A Kintex® UltraScale™ XCKU115-2 FLVA1517E FPGA
- 16 GB of DDR3 memory

Figure 24: ADM-PCIE-8K5 Card

NOTE: The ADM-PCIE-8K5 is shipped with an optional blower. Xilinx highly recommends that you install the blower on the card before proceeding to the next step.

Step 1: Prepare Board Installation Files

SDAccel™ provides a utility, `xbinst`, that generates firmware and driver files for the target board plugged into the host computer. Run the commands below to prepare files for ADM-PCIE-8K5 card installation. See [xbinst Command Reference](#) for more details on `xbinst` utility.

All commands need to be run with `root` or `sudo` privilege.

```
$ sudo xbinst -f xilinx:adm-pcie-8k5:2ddr:3.3 -d 8k5_dsa
***** xbinst v2017.1_sdx
***** Copyright 1986-2017 Xilinx, Inc. All Rights Reserved.

INFO: [XBINST 60-267] Packaging for PCIe...
INFO: [XBINST 60-268] Packaging for PCIe...COMPLETE
INFO: [XBINST 60-667] xbinst has successfully created a board installation
directory at /opt/dsa/8k5_dsa.
```

Make a note of the board installation directory. This procedure uses `/opt/dsa/8k5_dsa` in this chapter.

Copy the following file path to the programming computer:

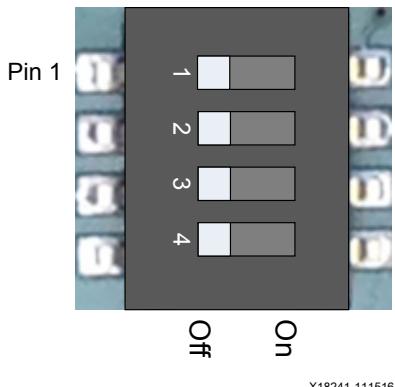
```
/opt/dsa/8k5_dsa/xbinst/pkg/pcie/firmware/ xilinx_adm-pcie-8k5_2ddr_3_3.mcs
```

Make a note the file location on the programming computer as it will be required for programming the configuration memory in later step.

Step 2. Setting Up the Card and Computer

1. Make sure the host computer is completely turned off.

- Locate DIP switch SW1 on the back side of the board near the side edge and set it as shown here:


X18241-111516

- Install the ADM-PCIE-8K5 card into an open PCIe slot in the host computer.

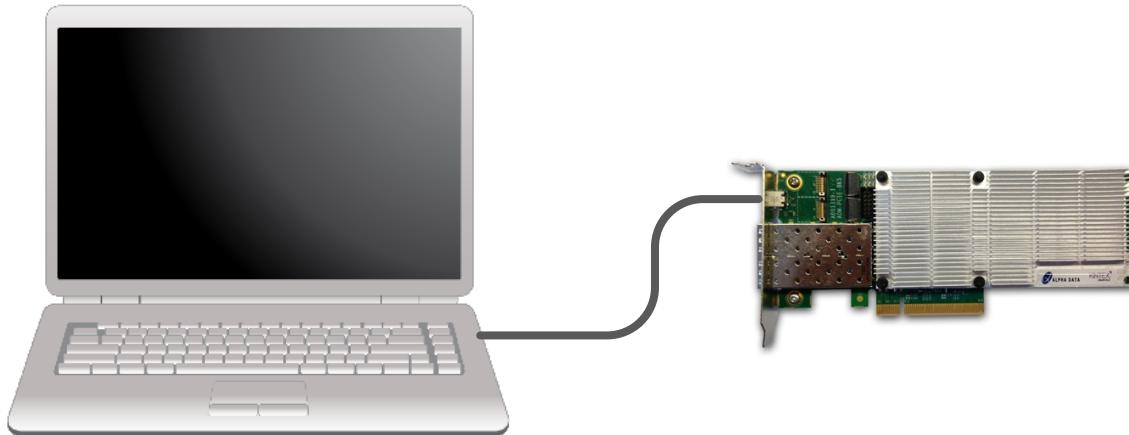
NOTE: Follow host computer manufacturer recommendations to ensure proper mounting and adequate cooling.

- Turn on the host computer.

Step 3: Programming the Base Platform

All applications compiled by the SDAccel™ compiler for the Alpha Data card are compiled against a specific device. A device is a combination of interfaces and infrastructure components on the card, which are required for proper execution of the user program. The base device program or firmware is different for all devices. This program must be loaded onto the FPGA before the user application is loaded. To program the firmware program:

- Connect the Alpha Data ADM-PCIE-8K5 card to the control computer having an installation of Vivado® Design Suite as shown here:


X18218-111116

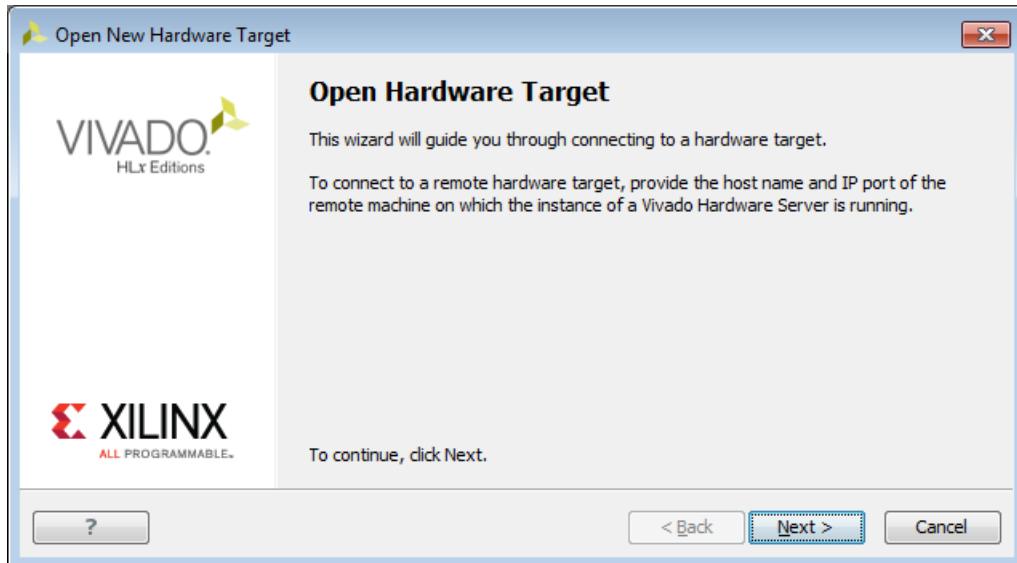
- On the programming computer, **Open Hardware Manager**.



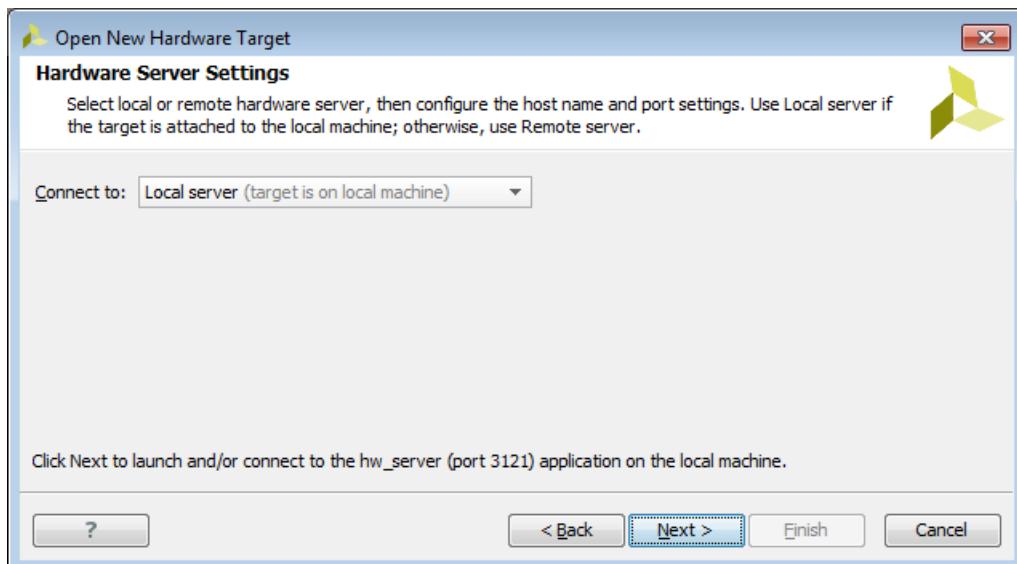
3. Select **Open a New Hardware Target**.



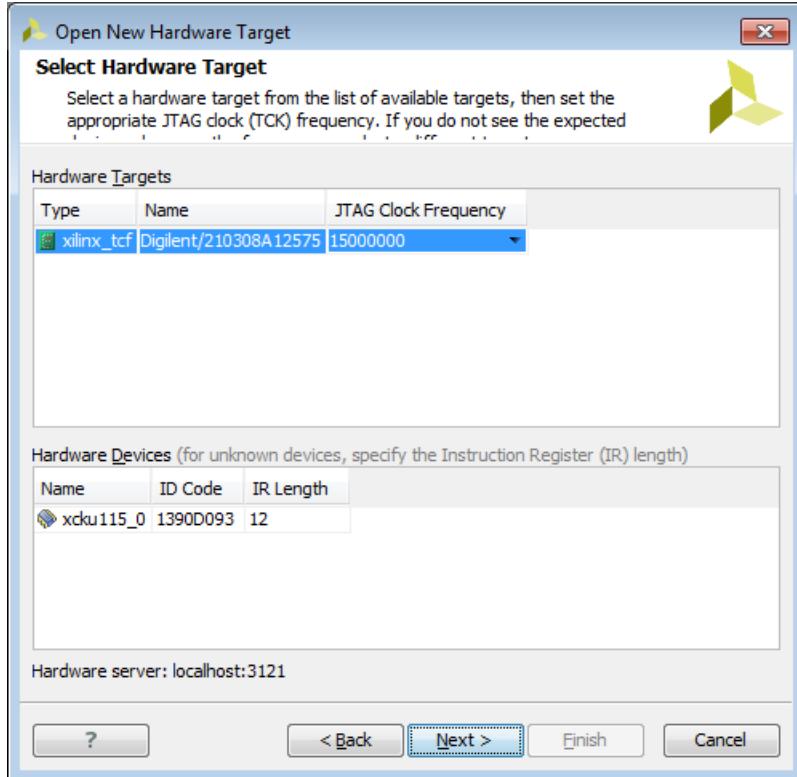
4. Click **Next** in the Open Hardware Target window.



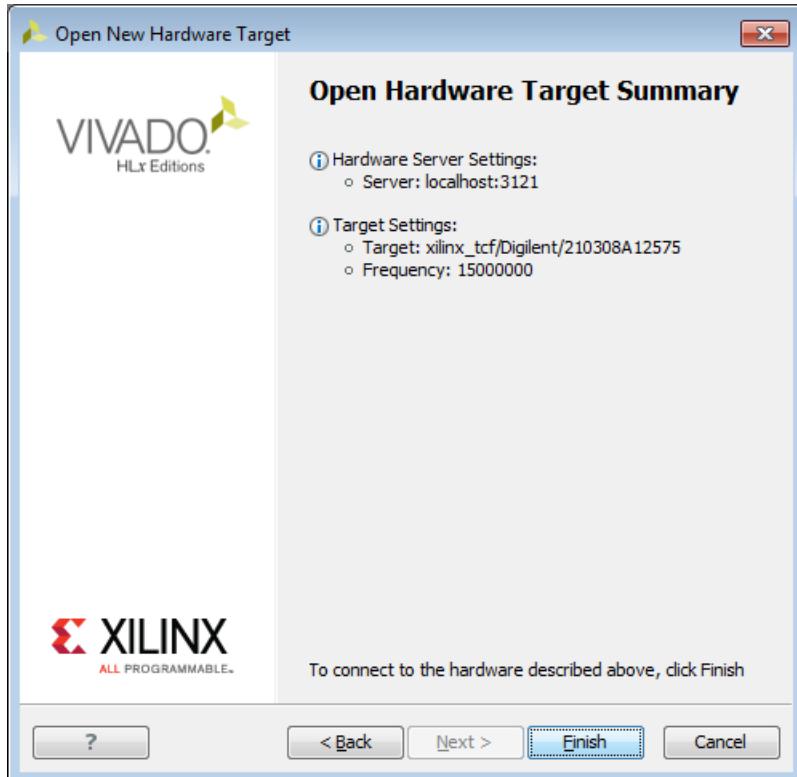
5. Select **Local server** in the **Connect to** field and click **Next**.



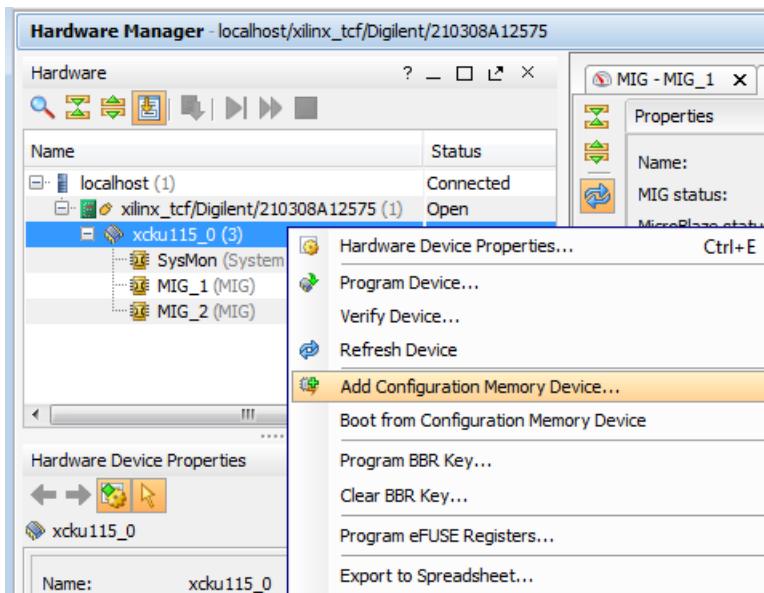
6. In the Open New Hardware Target window, select **xilinx_tcf** and click **Next**.



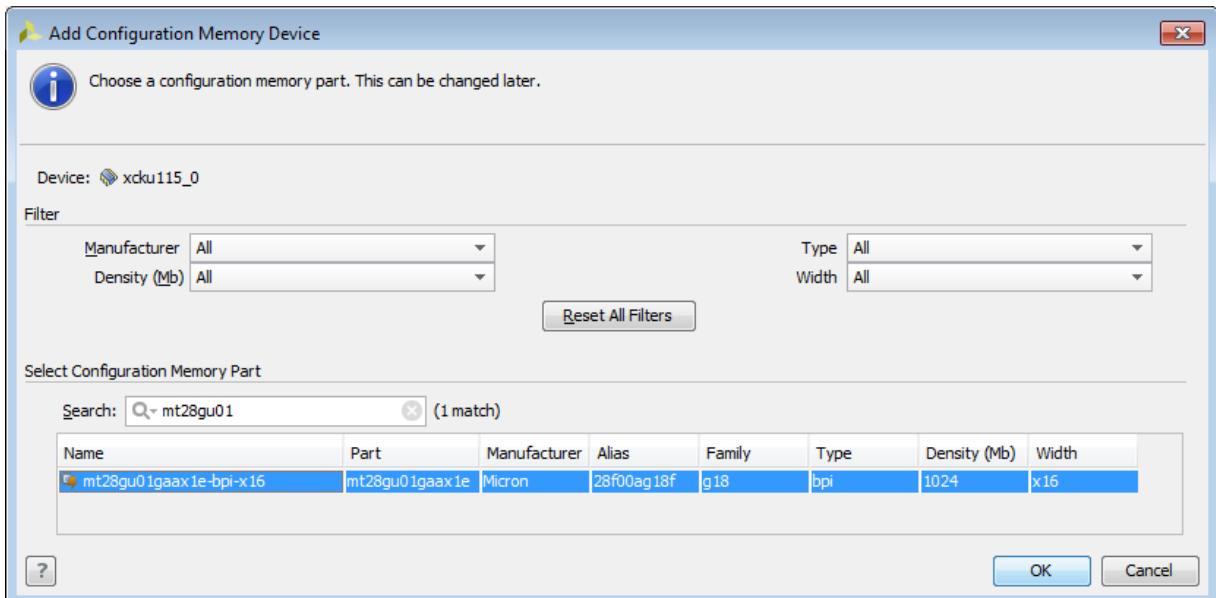
7. In the Open Hardware Target Summary window, click **Finish**.



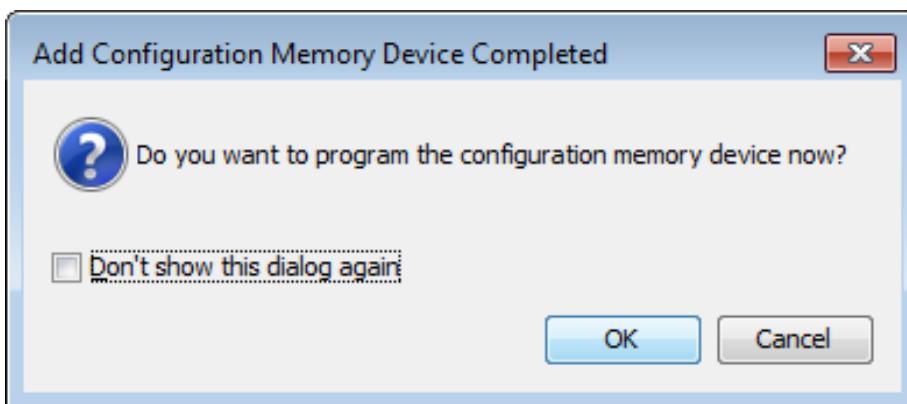
8. Right-click the FPGA (**kcku115_0**) and select **Add Configuration Memory Device**.



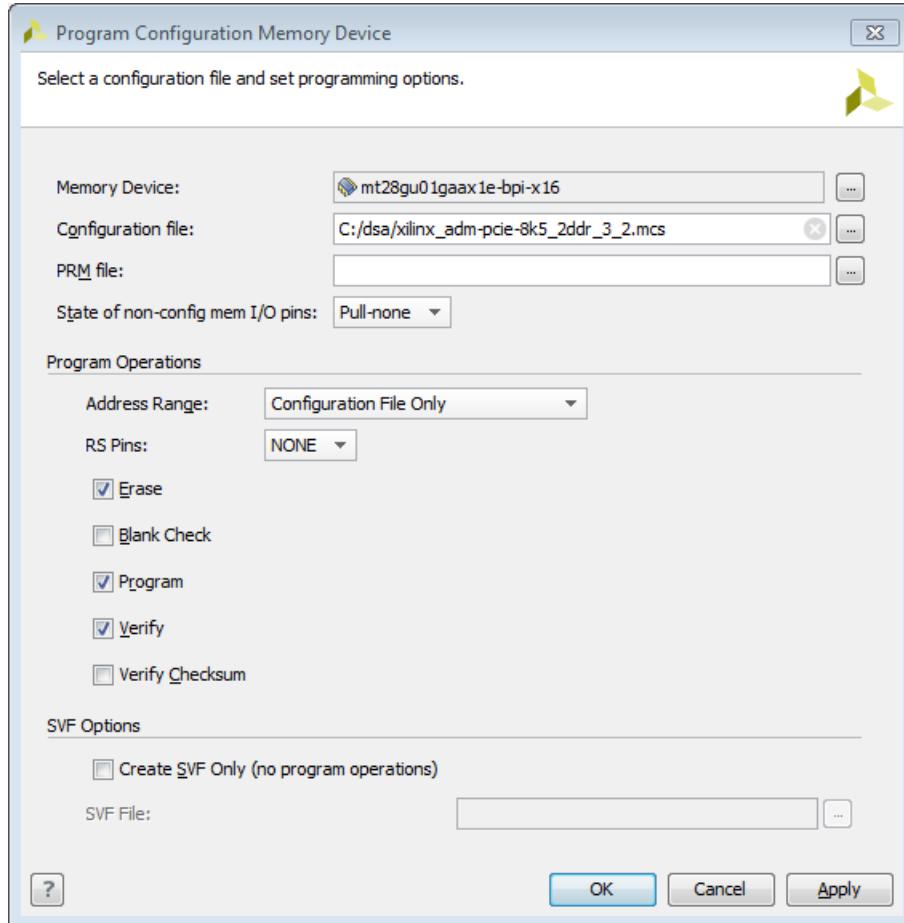
9. Select **mt28gu01gaax1e-bpi-x16** as the configuration memory.



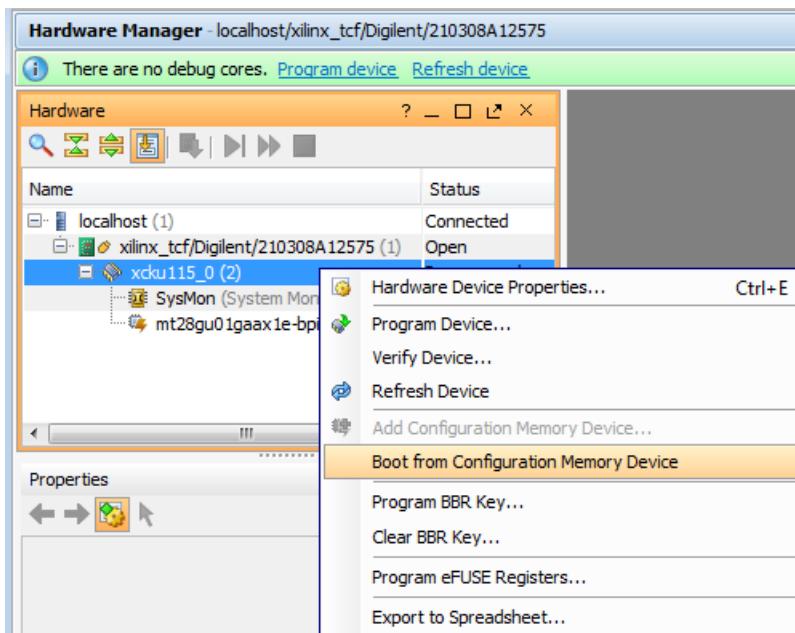
10. Click **OK** to program the configuration memory.



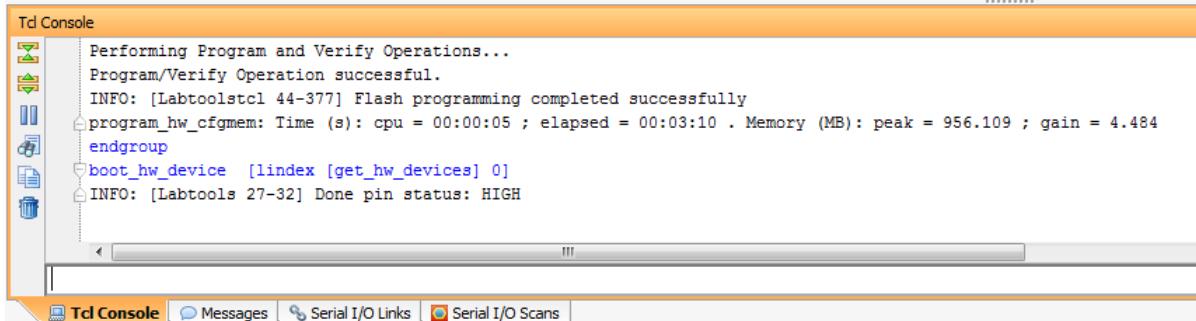
11. In the **Programming Configuration Memory Device** window, go to the **Configuration File** entry box, browse to, and select the MCS file (**xilinx_adm-pcie-8k5_2ddr_3_3.mcs**) that you copied to the programming computer in Step 1. Verify all other settings as shown in the **Program Configuration Memory Device** window. Click **OK** to start programming the configuration memory.



12. After the memory has been configured, right-click the FPGA (**xcku115_0**) and select **Boot From Configuration Device**.



The Tcl console displays Done pin status: HIGH after the FPGA is booted successfully.



13. Reboot the host computer.

NOTE: Programming of the device firmware is required only once per device. All applications targeting the same device can share a single programming instance of the card firmware.

Step 4: Installing Driver for the Card

You must install proper drivers for the card before you can use it to run SDAccel™ applications. Follow the instructions below to install the required drivers.

Change to the board installation directory generated in Step 1, and run installation script:

```
$ cd /opt/dsa/8K5_dsa/xbininst/pkg/pcie  
$ sudo ./install.sh
```

This will do the following:

- Compile and install Linux kernel device drivers.
- Install the firmware to the Linux firmware area.
- Generate a `setup.sh` (Bash) or `setup.csh` (for csh/tcsh) to set up the runtime environment. Users must source the setup script before running any application on the target FPGA card.

- Install Xilinx OpenCL Installable Client Driver (ICD) to `/etc/OpenCL/vendors`. The OpenCL ICD allows multiple implementations of OpenCL to co-exist on the same system. It allows applications to choose a platform from the list of installed platforms and dispatches OpenCL API calls to the underlying implementation.

Installing the Xilinx XIL-ACCEL-RD-KU115 Card

The XIL-ACCEL-RD-KU115 card is a high-performance reconfigurable computing card for data center applications. It features:

- Kintex® UltraScale™ XCKU115-FLVB2104-2-E FPGA
- Four 4GB DDR4 banks (16GB total)

Figure 25: XIL-ACCEL-RD-KU115 Card



Step 1: Prepare Board Installation Files

SDAccel™ provides a utility, `xbinst`, that generates firmware and driver files for the target board plugged into the host computer. Run the commands below to prepare files for XIL-ACCEL-RD-KU115 card installation. See [xbinst Command Reference](#) for more details on the `xbinst` utility.

All commands must be run with `root` or `sudo` privilege.

```
$sudo mkdir xil-accel-rd-ku115
$sudo xbinst -f xilinx:xil-accel-rd-ku115:4ddr-xpr:3.3 -d xil-accel-rd-ku115
***** xbinst v2017.1_sdx
**** Copyright 1986-2017 Xilinx, Inc. All Rights Reserved.

INFO: [XBINST 60-267] Packaging for PCIe...
INFO: [XBINST 60-268] Packaging for PCIe...COMPLETE
INFO: [XBINST 60-667] xbinst has successfully created a board installation
directory at /opt/xil-accel-rd-ku115.
```

Make a note of the board installation directory. This procedure uses /opt/xil-accel-rd-ku115 as an example in this chapter.

Copy the following files to the programming computer:

```
/opt/xil-accel-rd-ku115/xbinst/pkg/pcie/firmware/
xilinx_xil-accel-rd-ku115_4ddr-xpr_3_3_primary.mcs

/opt/xil-accel-rd-ku115/xbinst/pkg/pcie/firmware/
xilinx_xil-accel-rd-ku115_4ddr-xpr_3_3_secondary.mcs
```

Make a note of the file location on the programming computer as it will be required for programming the configuration memory in a later step.

Step 2: Setting up the Card and Computer

1. Make sure the host computer is completely turned off.
2. Install the XIL-ACCEL-RD-KU115 card into an open PCIe slot in the host computer.

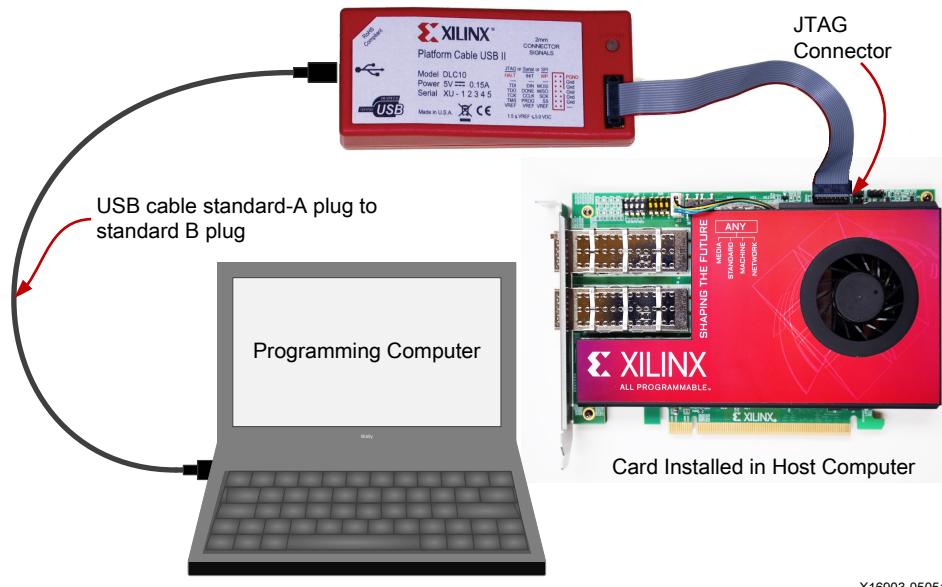
NOTE: *Follow host computer manufacturer recommendations to ensure proper mounting and adequate cooling.*

3. Turn on the host computer.

Step 3: Programming the Base Platform

All applications compiled by the SDAccel™ compiler for the Alpha Data card are compiled against a specific device. A device is a combination of interfaces and infrastructure components on the card, which are required for proper execution of the user program. The base device program or firmware is different for all devices. This program must be loaded onto the FPGA before the user application is loaded. To program the firmware program:

1. Connect the Xilinx® XIL-ACCEL-RD-KU115 card to the programming computer with an installation of Vivado® Design Suite as shown here:

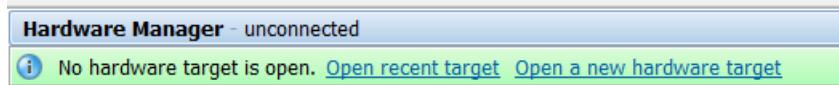


X16903-050516

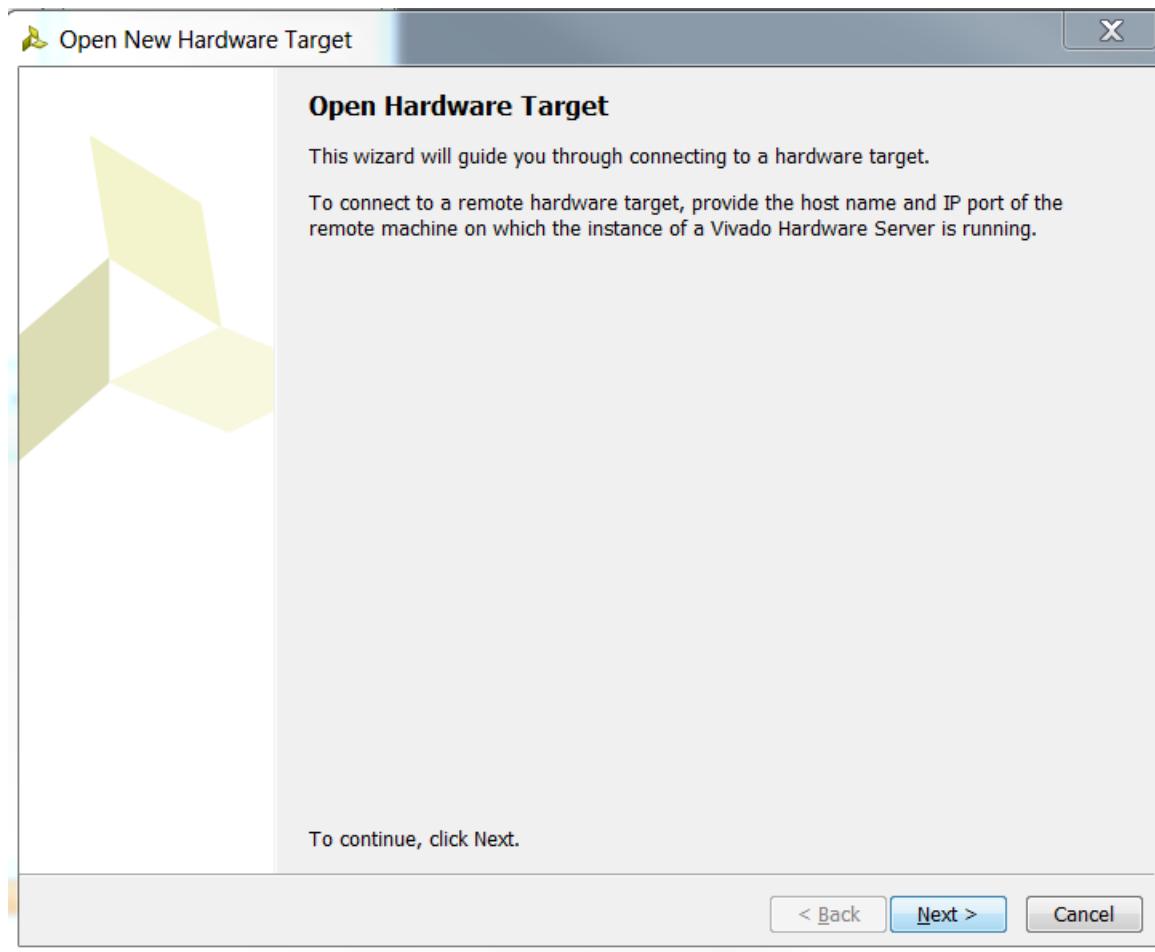
2. On the programming computer, start the Vivado Design Suite then on the Welcome page, select **Open Hardware Manager**.



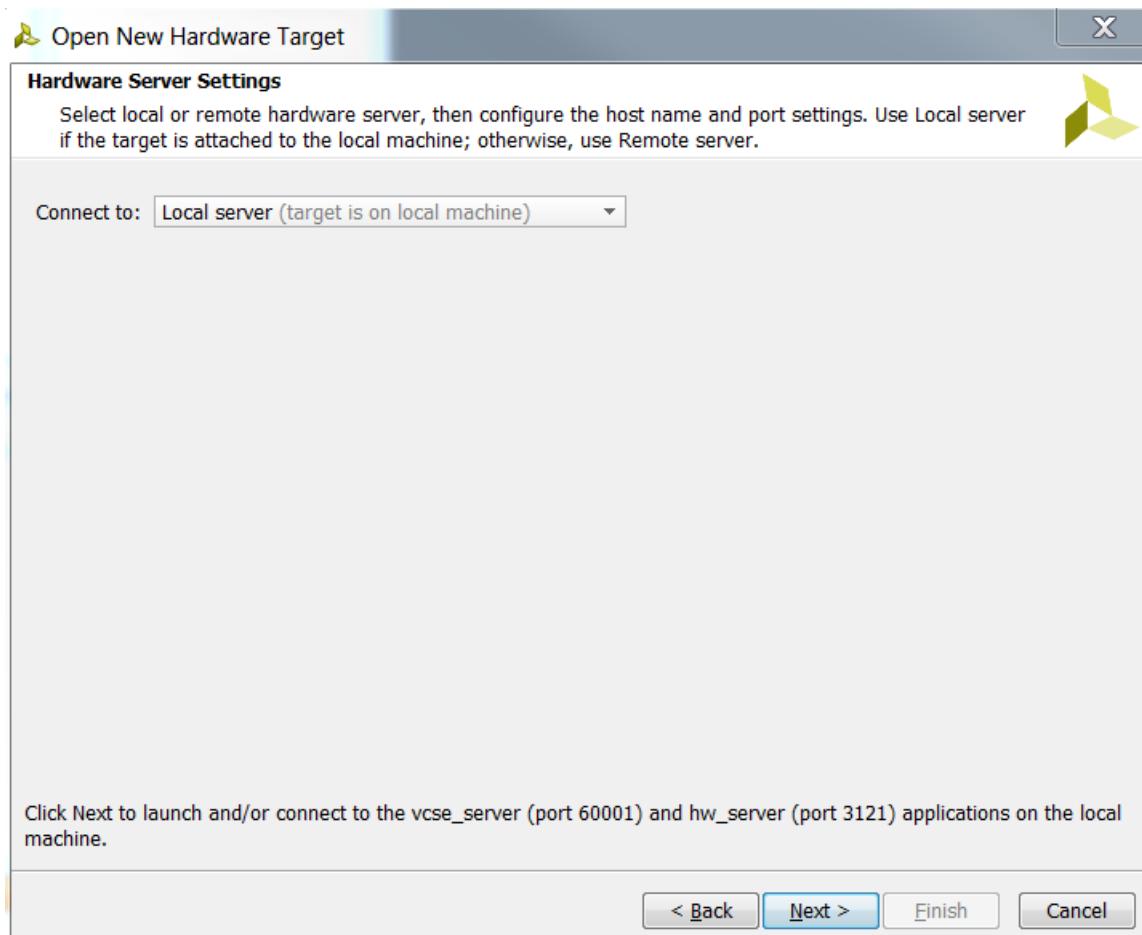
3. Select **Open a New Hardware Target**.



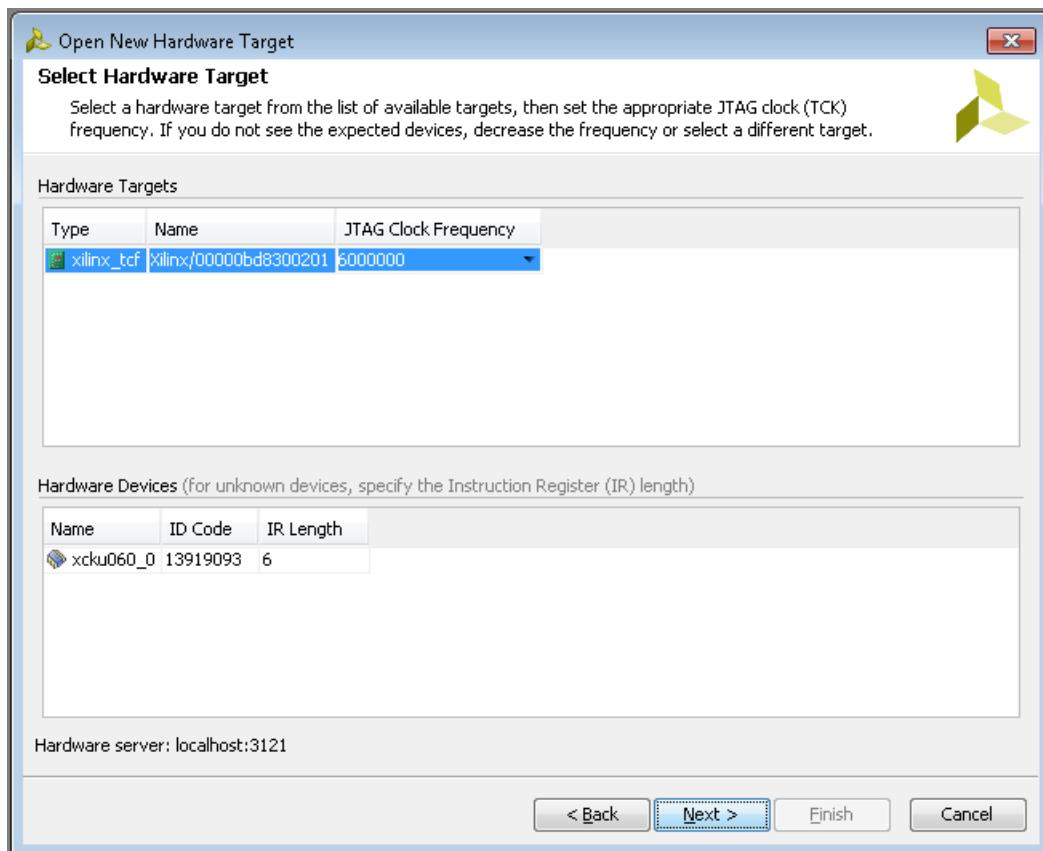
4. Click Next in the Open Hardware Target window.



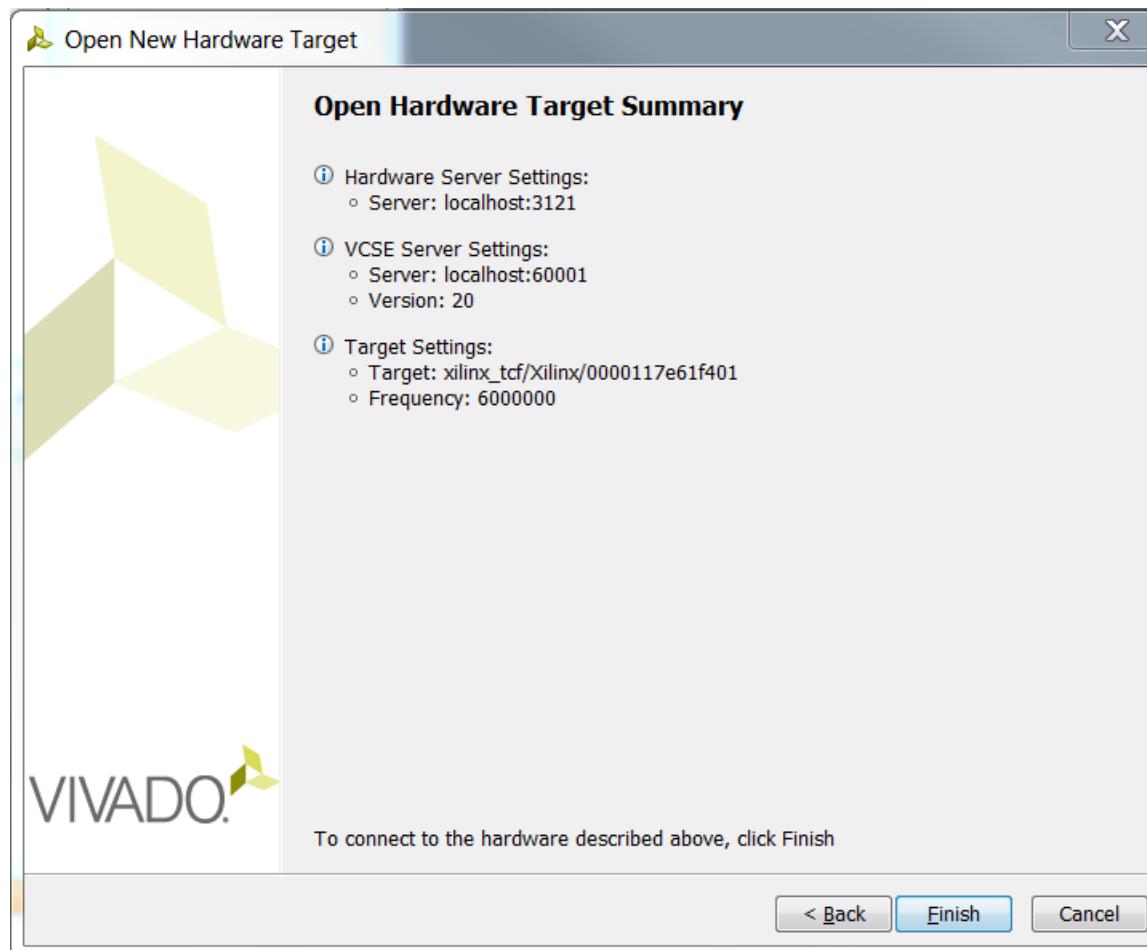
5. Select **Local server** in the **Connect to** field and click **Next**.



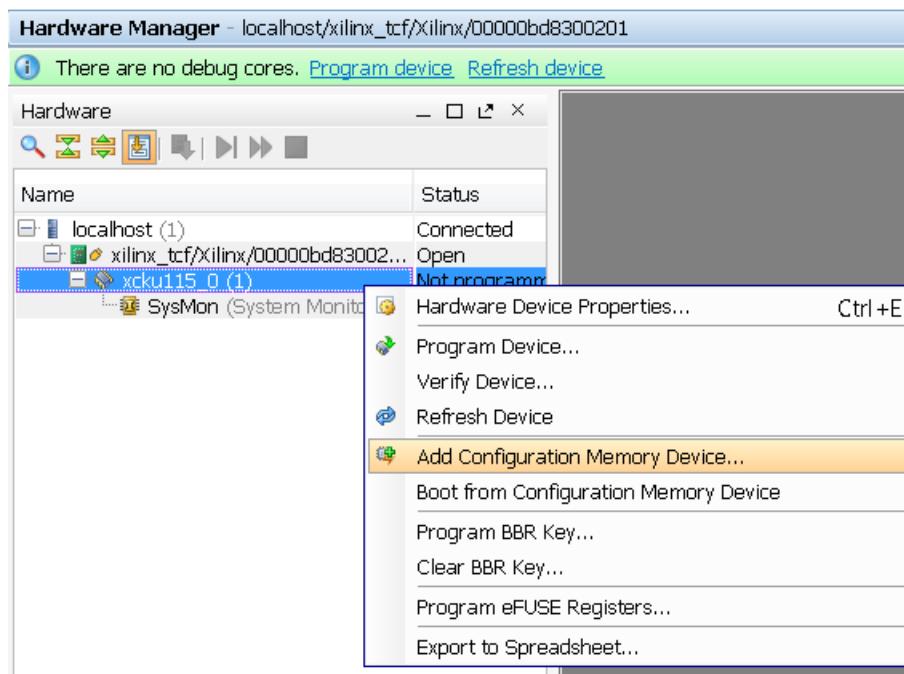
6. In the Open New Hardware Target window, select **xilinx_tcf** and click **Next**.



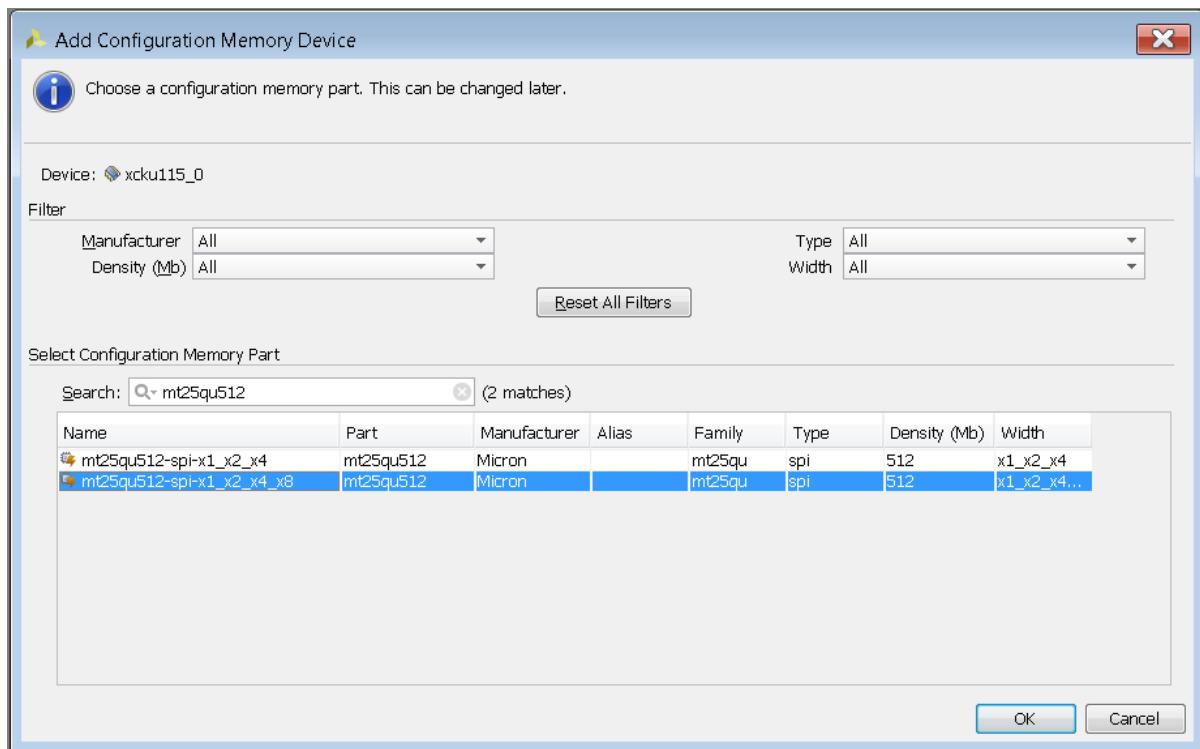
7. In the Open Hardware Target Summary window, click **Finish**.



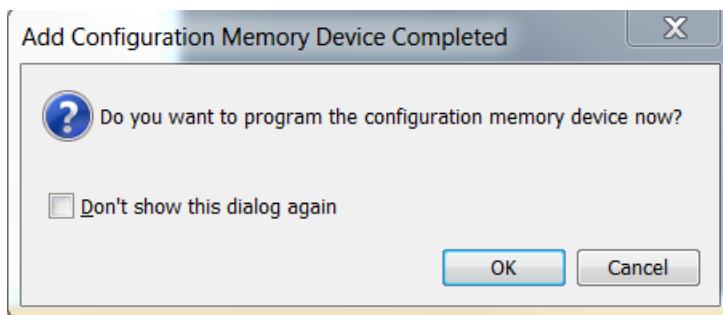
8. Right-click your FPGA (**xcku115_0**), and select **Add Configuration Memory Device**.



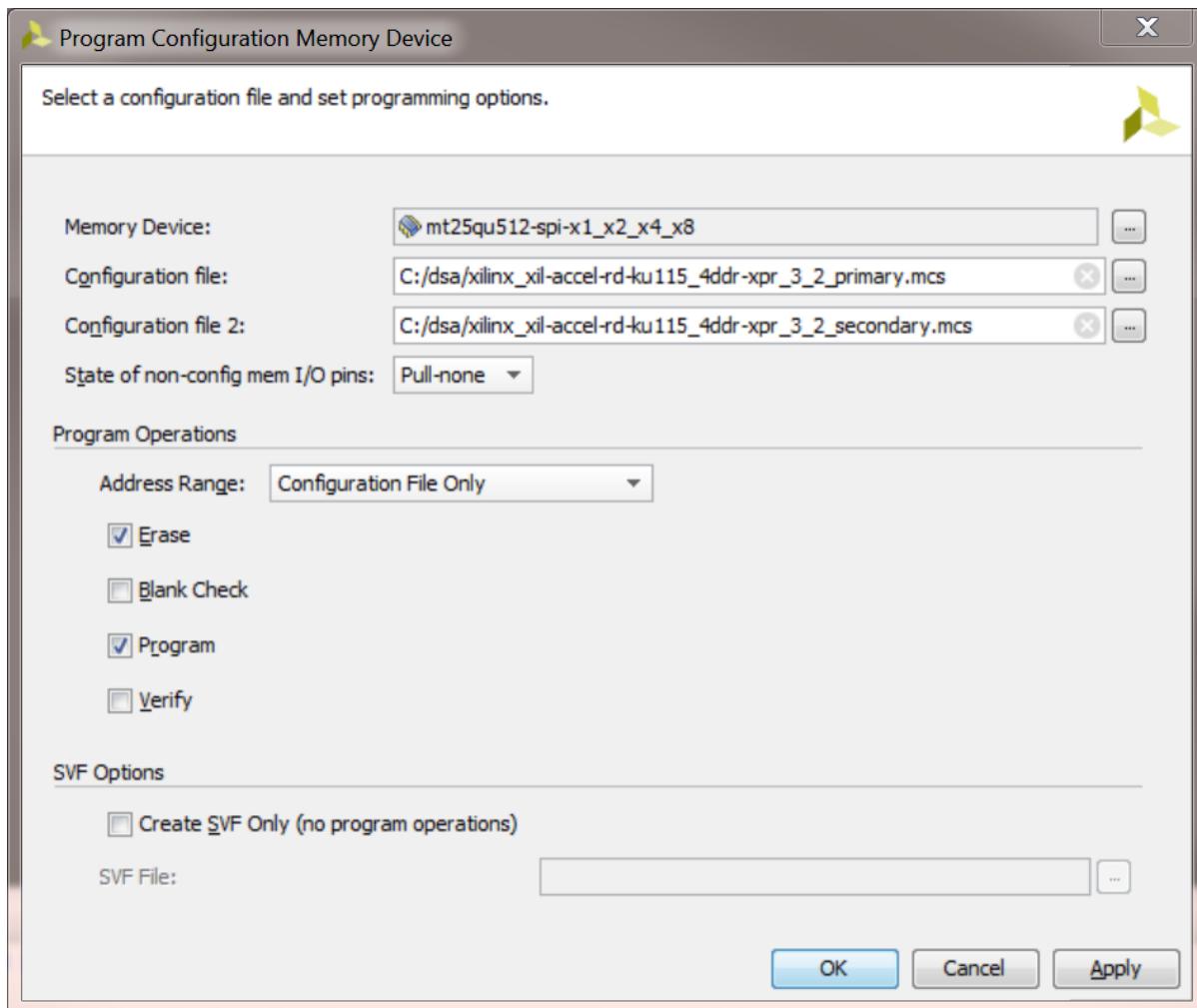
9. Select **mt25qu512-spi-x1_x2_x4_x8** as the configuration memory.



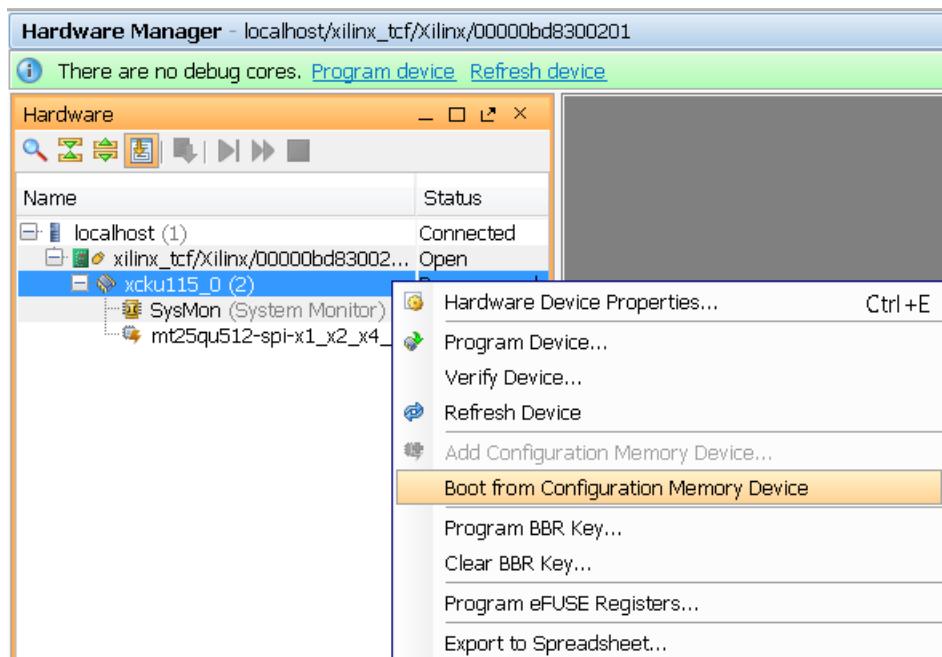
10. Click **OK** to program the configuration memory.



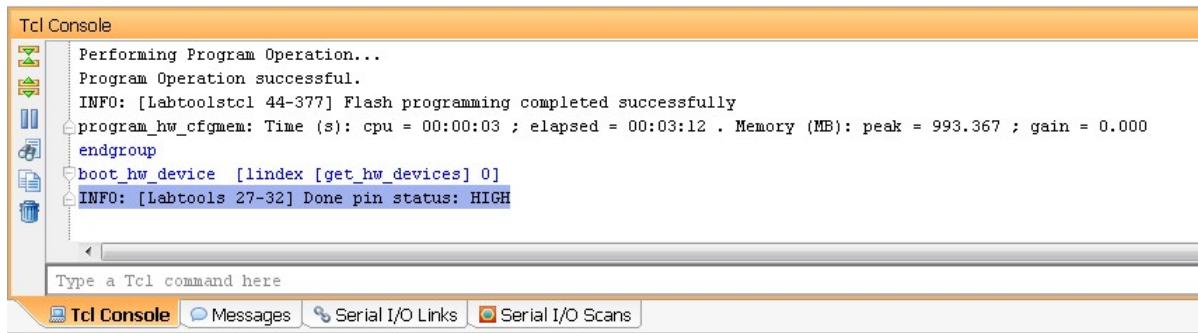
11. In the **Programming Configuration Memory Device** window, go to the **Configuration File** entry box, browse to and select the MCS file (**xilinx_xil-accel-rd-ku115_4ddr-xpr_3_2_primary.mcs**) that you copied to the programming computer in Step 1. Go to the **Configuration File 2** entry box, browse to and select the MCS file (**xilinx_xil-accel-rd-ku115_4ddr-xpr_3_3_secondary.mcs**). Verify all other settings as shown in the **Program Configuration Memory Device** window. Click **OK** to start programming the configuration memory.



12. After the memory has been configured, right-click the FPGA (**xcku115_0**) and select **Boot From Configuration Device**.



The Tcl console displays Done pin Status: HIGH after the FPGA is booted successfully.



13. Reboot the host computer.

NOTE: Programming of the device firmware is required only once per device support archive (dsa). All applications targeting the same dsa can share a single programming instance of the card firmware.

Step 4: Installing Driver for the Card

You must install proper drivers for the card before you can use it to run SDAccel™ applications. Follow the instructions below to install the required drivers.

Change to the board installation directory generated in Step 1 and run installation script:

```
$cd /opt/xil-accel-rd-ku115/xbinst/pkg/pcie
$sudo ./install.sh
```

This will do the following.

- Compile and install Linux kernel device drivers.
- Install the firmware to the Linux firmware area.
- Generate a `setup.sh` (Bash) or `setup.csh` (for csh/tcsh) to set up the runtime environment. You must source the `setup` script before running any application on the target FPGA card.
- Install Xilinx OpenCL™ Installable Client Driver (ICD) to `/etc/OpenCL/vendors`. The OpenCL ICD allows multiple implementations of OpenCL to co-exist on the same system. It allows applications to choose a platform from the list of installed platforms and dispatches OpenCL API calls to the underlying implementation.

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. *UltraFast Design Methodology Guide for the Vivado Design Suite*, ([UG949](#))
11. *Vivado Design Suite Properties Reference Guide*, ([UG912](#))
12. *Khronos Group web page*: Documentation for the OpenCL standard
13. *Alpha Data web page*: Documentation for the ADM-PCIE-7V3 Card
14. *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.