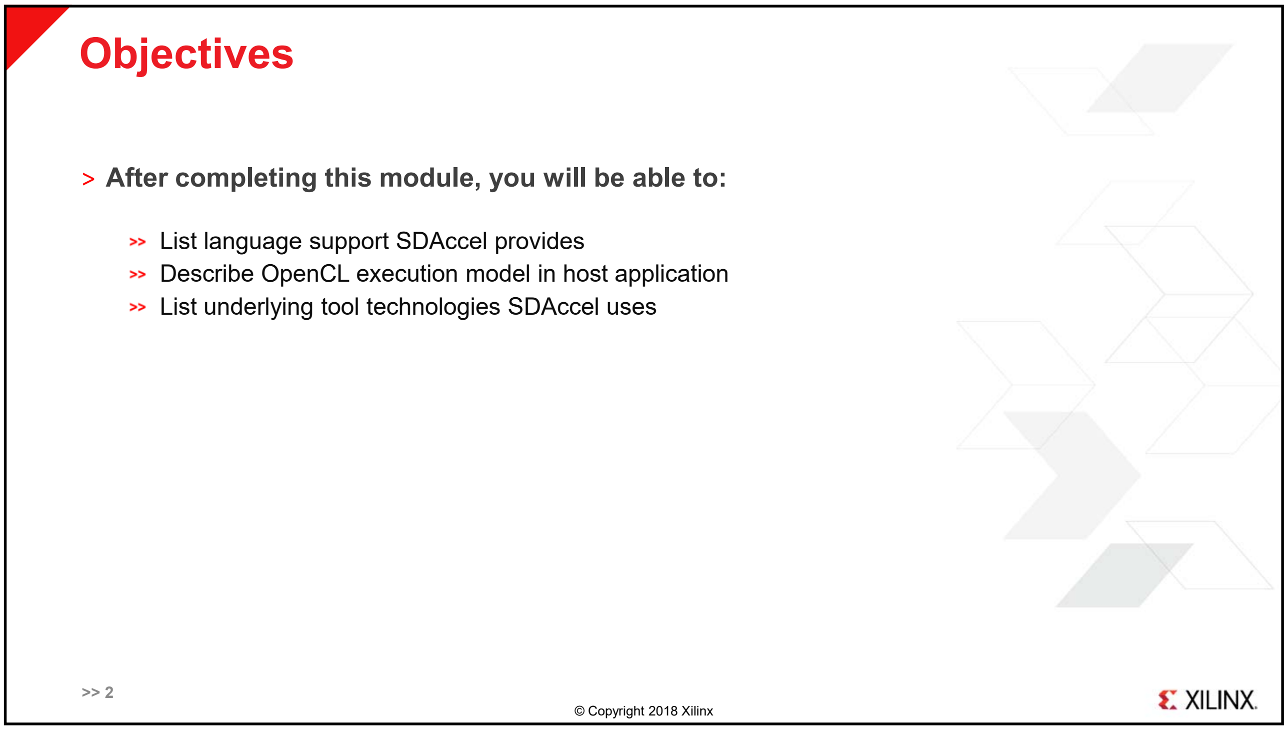
SDAccel Tool Overview

SDX 2018.2

XILINX.



Objectives

> After completing this module, you will be able to:

- List language support SDAccel provides
- Describe OpenCL execution model in host application
- List underlying tool technologies SDAccel uses

XILINX.

Outline

> SDAccel Overview

> Host code: OpenCL execution model

> Makefile Flow

> Summary

> Lab Intro

SDAccel Development Environment

SDAccel supports OpenCL 1.0 Embedded Profile with various 1.2 / 2.0 features

Host code

– OpenCL APIs

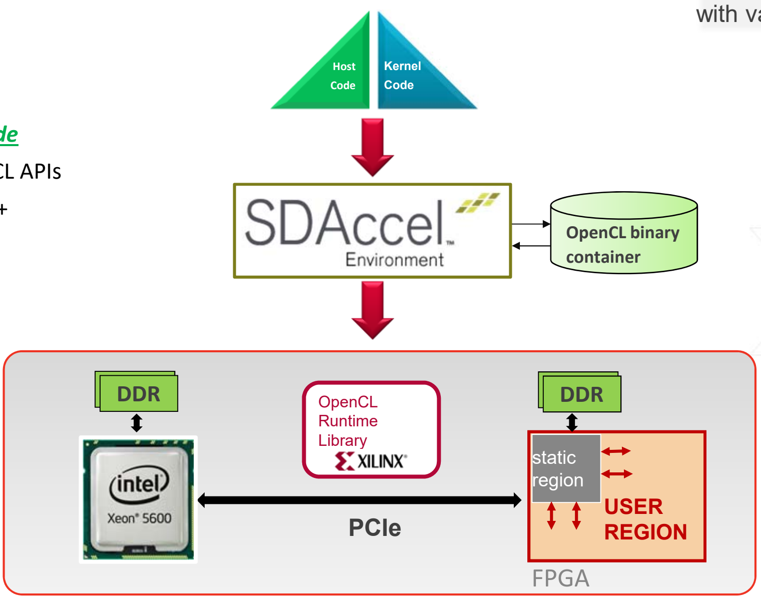
– C / C++

Kernel code

– OpenCL kernel code

– C/C++

– RTL IP



OpenCL - Open Computing Language

> **An open industry standard**
>> for **parallel** computing
>> of **heterogeneous** systems

> **Enables cross-platform functional portability**
>> No code changes *(note: embedded profile)*
>> Portable across CPU, GPU, FPGA, DSP etc.
– Can run on: cell phones, laptops, supercomputers
>> Important: No Performance Portability

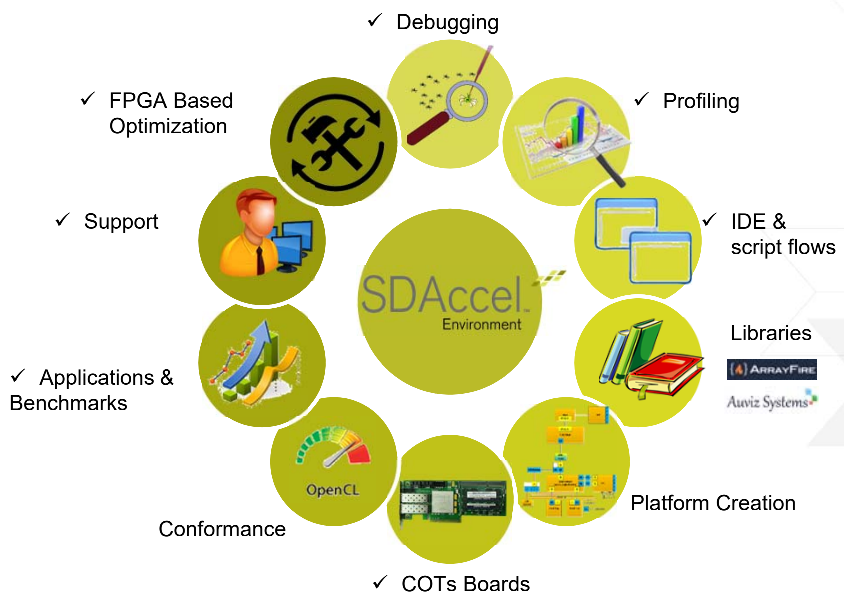
> **Wide market adoption**
>> Support implemented by:
– Apple, AMD, **Xilinx**, Intel, ARM, Nvidia, Qualcomm, …
>> Many companies developing applications
– Image, video, audio processing, scientific calculations, medical imaging, …

OpenCL

***Khronos** Group*
www.khronos.org

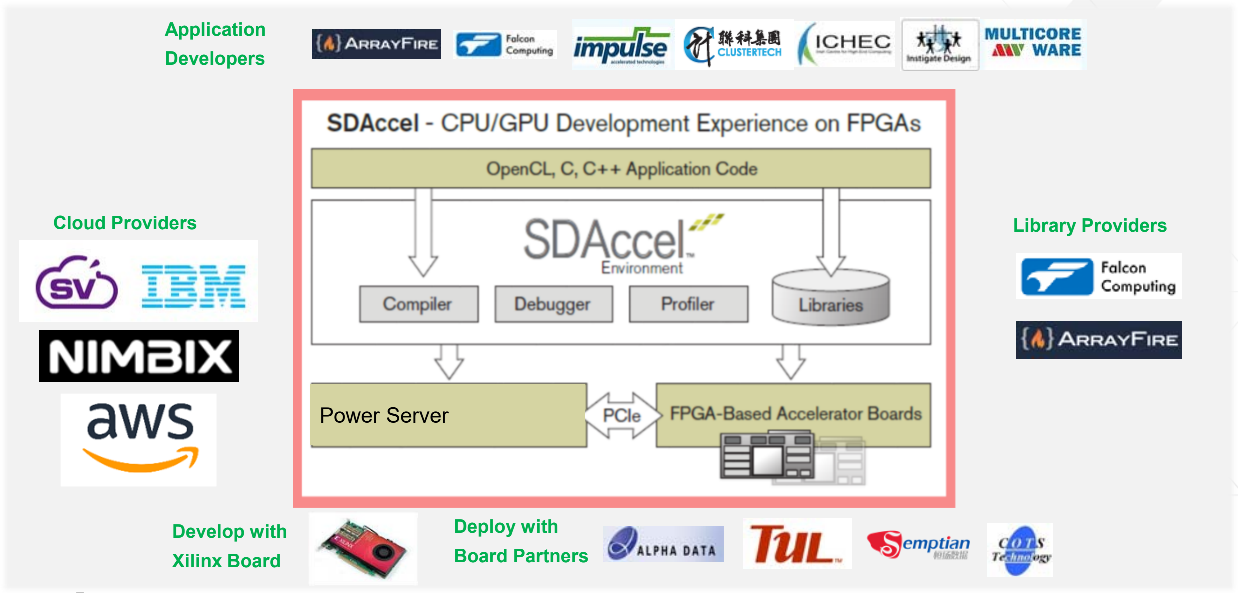
XILINX.

SDAccel – The Complete Development Environment



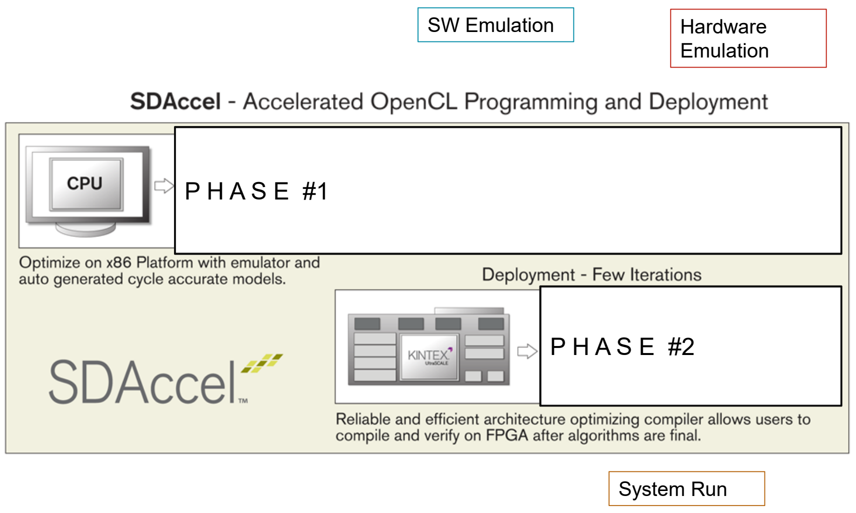
XILINX.

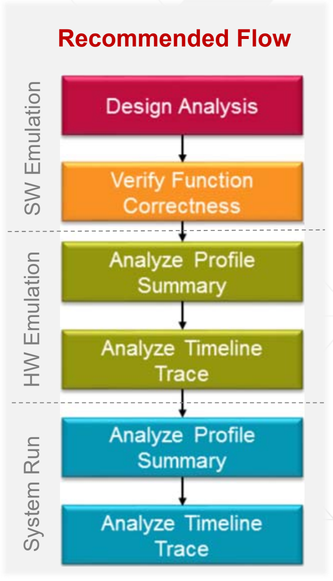
SDAccel Environment and Ecosystem



XILINX.

SDAccel Flow


SW Emulation
Hardware Emulation
SDAccel - Accelerated OpenCL Programming and Deployment
CPU
PHASE #1
Optimize on x86 Platform with emulator and auto generated cycle accurate models.
Deployment - Few Iterations
SDAccel
KINTEX
PHASE #2
Reliable and efficient architecture optimizing compiler allows users to compile and verify on FPGA after algorithms are final.
System Run



*Note: SDAccel can be run using **Makefile**, **GUI***

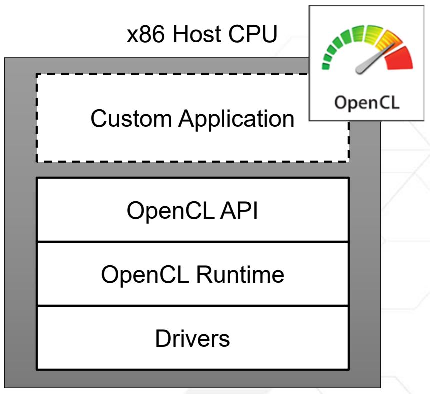
XILINX.

Host Code
OpenCL Execution Model

XILINX

Host Framework and Execution Stack

- An open industry standard for parallel computing
  - OpenCL language for expressing kernels
  - OpenCL API for host code interaction with kernels
  - OpenCL runtime to manage kernel scheduling during execution
- Standard maintained by Khronos Group
  - khronos.org
- Host application submits work to FPGA kernels using standard OpenCL API
- Separates application logic from performance code
  - Can swap kernels dynamically
  - Naturally captures inherent parallelism in algorithms
- Xilinx's OpenCL runtime and drivers are managing the communication with the hardware



XILINX

OpenCL Execution Model

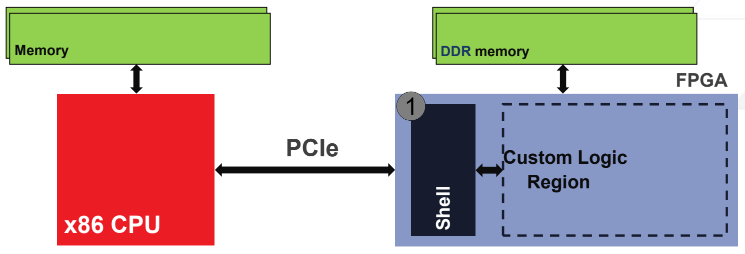
> OpenCL API provides a high-level description of the key steps of an application executing on F1

1. Powering-Up
2. Runtime Initialization
3. Device Configuration
4. Buffer Allocation
5. Writing Buffers to FPGA Memory
6. Running the Accelerators
7. Reading Buffers from FPGA Mem

XILINX.

1. Powering-Up

> On power-up, the FPGA is initialized
> At this stage, the only logic in the FPGA is the Shell
> The Shell will be managing the communications with the host



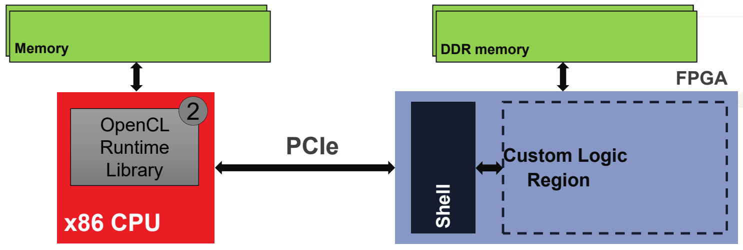
XILINX.

2. Runtime Initialization

> Creation of OpenCL context and device
  >> Context → Platform
  >> Device → FPGA
> Creation of OpenCL command queues used to send commands to the FPGA

```
context = clCreateContextFromType(…);
clGetDeviceIDs(…, &device_id, …);

queue = clCreateCommandQueue(context, device_id, …);
```

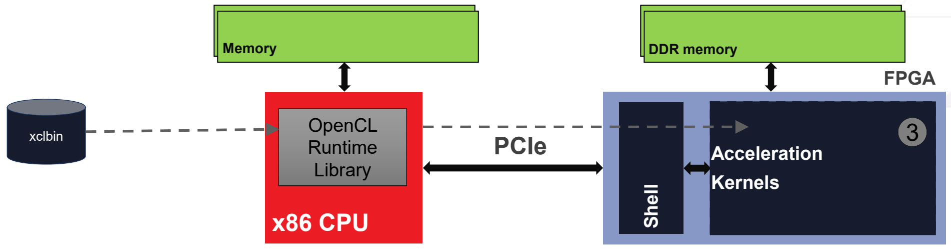


XILINX.

3. Device Configuration

> Host programs the FPGA by calling clCreateProgramWithBinary
  >> Loads the .xclbin file

```
program = clCreateProgramWithBinary(…);
```

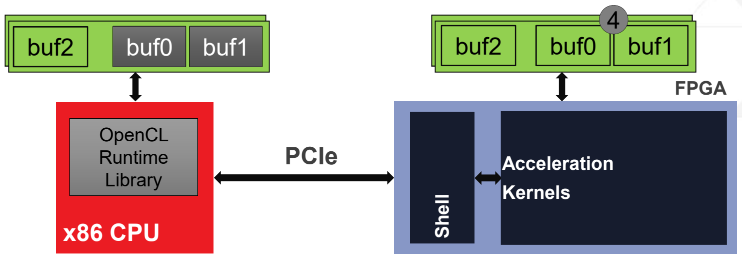


XILINX.

4. Buffer Allocation

> Host allocates buffers in the device
> Buffers are used to transfer data from the CPU to the FPGA and back

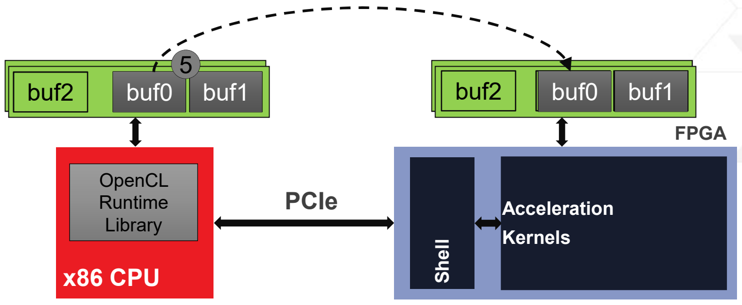
```
buf0 = clCreateBuffer(context, CL_MEM_READ_ONLY, …);
buf1 = clCreateBuffer(context, CL_MEM_READ_ONLY, …);
buf2 = clCreateBuffer(context, CL_MEM_WRITE_ONLY, …);
```



5. Writing Buffers to FPGA Memory

> Host copies data to be processed from local memory to the buffer in the FPGA DDR memory

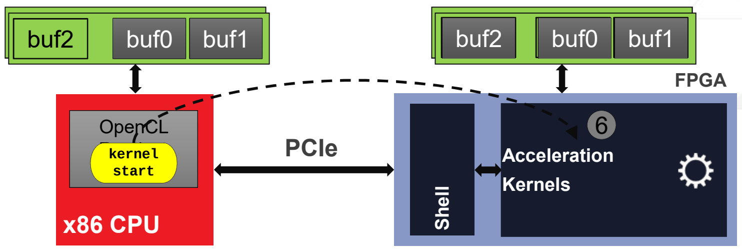
```
clEnqueueWriteBuffer(queue, buf0, …);
clEnqueueWriteBuffer(queue, buf1, …);
```



6. Running the Accelerators

- Host schedules execution of the desired kernel with clEnqueueTask
- Runtime starts the Kernel
- Kernel processes data previously copied to from host buffer to DDR

```
kernel = clCreateKernel(program, “mykernel", …);
clSetKernelArg(kernel, 0, sizeof(cl_mem), &buf0);
clSetKernelArg(kernel, 1, sizeof(cl_mem), &buf1);
clSetKernelArg(kernel, 2, sizeof(cl_mem), &buf2);
err = clEnqueueTask(queue, kernel, 0, NULL, NULL);
clFinish(queue);
```



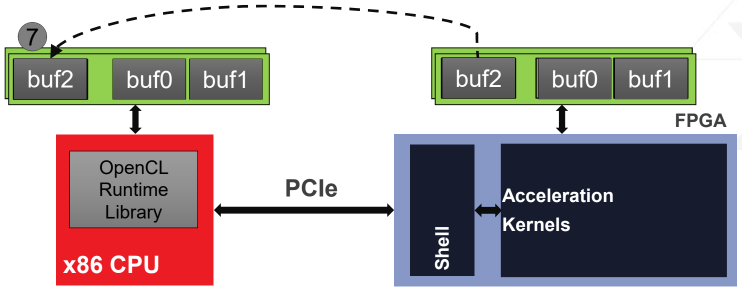
XILINX.

7. Reading Buffers from FPGA Mem

- The host retrieves the results by scheduling a copy of DDR content back to host memory

```
clEnqueueReadBuffer(queue, buf2, ..., &readevent );

clWaitForEvents(1, &readevent);
```



XILINX.

Makefile Flow

Environment Setup for SDAccel

> **Before you run the SDAccel tools, either in Makefile or GUI flow, you need to setup the PATH variable and other environment variables**

> **Execute the following commands to source the SDAccel and SDx setup scripts**

```
$ cd ~/aws-fpga
$ source sdaccel_setup.sh
$ source $XILINX_SDX/settings64.sh
```

SW Emulation of the *hello_world* Application

> **Validate and refine the application using software emulation**

- Validate correctness of the application
- Refine algorithm, if necessary
  - Fast compilation and run on a CPU

> **Execute the following commands to compile and run the application using the Makefile flow**

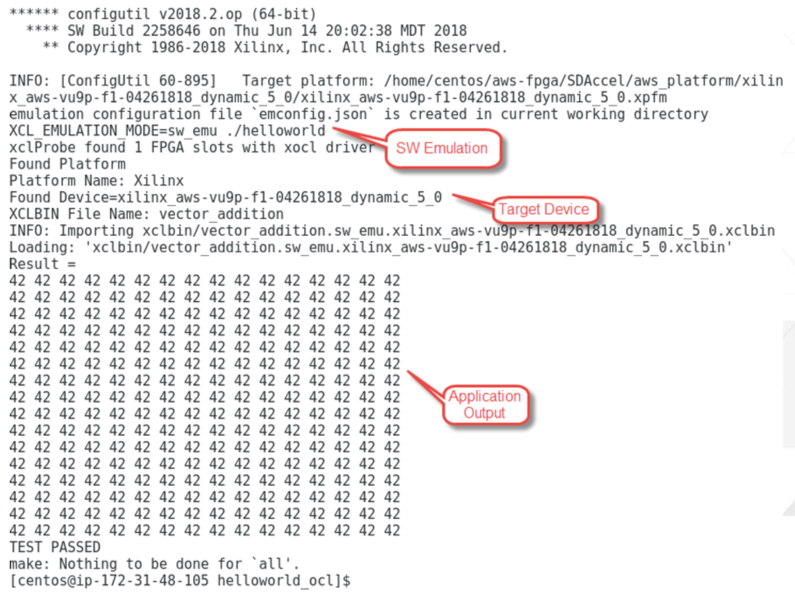
```
$ cd $SDACCEL_DIR/examples/xilinx/getting_started/host/helloworld_ocl/
$ make clean
$ make check TARGETS=sw_emu DEVICES=$AWS_PLATFORM all
```

- The application will be compiled, if it is not up to date, and executed on the x86 CPU (host machine) displaying the result (see next slide)

Note: In the above command if either TARGETS is misspelled or value is not sw_emu, hw_emu, or hw then the full hardware bitstream generation process will be done

XILINX.

Software Emulation Output



```
****** configutil v2018.2.op (64-bit)
  **** SW Build 2258646 on Thu Jun 14 20:02:38 MDT 2018
    ** Copyright 1986-2018 Xilinx, Inc. All Rights Reserved.

INFO: [ConfigUtil 60-895]   Target platform: /home/centos/aws-fpga/SDAccel/aws_platform/xilin
x_aws-vu9p-f1-04261818_dynamic_5_0/xilinx_aws-vu9p-f1-04261818_dynamic_5_0.xpfm
emulation configuration file `emconfig.json` is created in current working directory
XCL_EMULATION_MODE=sw_emu ./helloworld
xclProbe found 1 FPGA slots with xocl driver
Found Platform
Platform Name: Xilinx
Found Device=xilinx_aws-vu9p-f1-04261818_dynamic_5_0
XCLBIN File Name: vector_addition
INFO: Importing xclbin/vector_addition.sw_emu.xilinx_aws-vu9p-f1-04261818_dynamic_5_0.xclbin
Loading: 'xclbin/vector_addition.sw_emu.xilinx_aws-vu9p-f1-04261818_dynamic_5_0.xclbin'
Result =
42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42
42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42
42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42
42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42
42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42
42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42
42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42
42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42
42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42
42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42
42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42
42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42
42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42
42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42
42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42
42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42
TEST PASSED
make: Nothing to be done for `all'.
[centos@ip-172-31-48-105 helloworld_ocl]$
```

XILINX.

Summary

Summary

> SDAccel supports C, C++, and OpenCL languages
> SDAccel uses Vivado, Vivado HLS, OpenCL compilers
> OpenCL API are provided for application execution on F1
> SDAccel uses Eclipse environment and rich ecosystem

Lab Intro

XILINX

Lab Intro

> In this lab you will go through a Makefile flow to build an “hello world” example, which performs vector addition of 256 elements design for software emulation

> You then will modify the host code, using *gedit* editor, to change the values it adds and number of elements on which addition is performed. You will recompile the application and see the result

> You will use pre-compiled binary image and run it on the AWS F1 instance

XILINX

Adaptable.
Intelligent.

XILINX