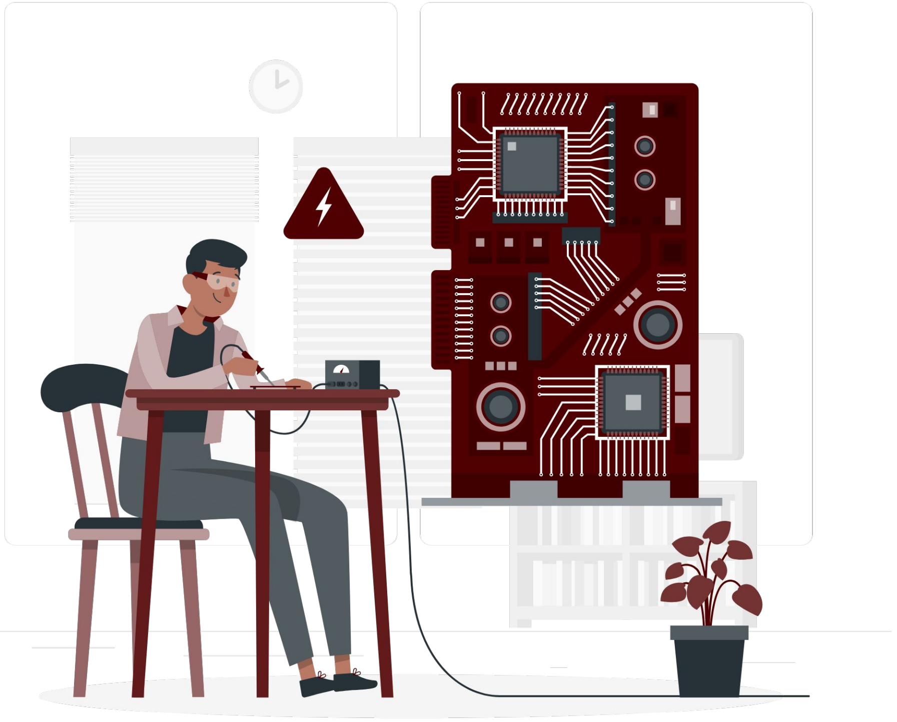


ACES Short Course Series

Introduction to Intel FPGAs

Abhinand Nasari & Shaina Le

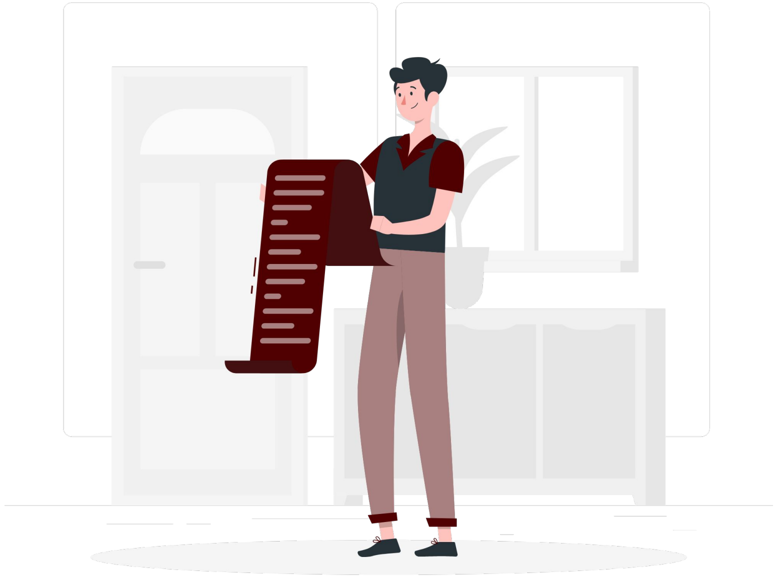
High Performance
Research Computing
DIVISION OF RESEARCH

hprc.tamu.edu

Rev 1.0.0
29 Nov 2022

Outline

- Introduction to FPGAs
  - Course Objectives
  - Terminology
  - What are FPGAs?
  - Applications of FPGAs
  - Developing on FPGAs vs Other Platforms
- Break
- Getting Started
  - Requirements
  - Accessing the FPGAs
  - Notes on FPGA Usage
- FPGA Development Flow
- Break
- Demonstration
- Getting Support & Other Resources



Course Objectives

- complete a brief overview of FPGAs
- learn some terminology related to the FPGAs
- learn about the tools needed to work with FPGAs
- learn about the availability of tools for working with FPGAs
- learn how to access the FPGAs
- complete a brief overview of how to use the FPGAs
- learn where to get help regarding FPGAs

ℹ️ These objectives are to be interpreted within the context of FPGAs on TAMU HPRC systems.

What are FPGAs?

- “Field-Programmable Gate Array”
- Devices that can be reprogrammed for desired applications or functionality after manufacturing
  - Contrast to ASIC (application-specific integrated circuit), whose functionality *cannot* be changed after manufacture)
- Often referred to as “spatial” accelerators
  - Computation is distributed across discrete processing elements spread within physical space

Terminology

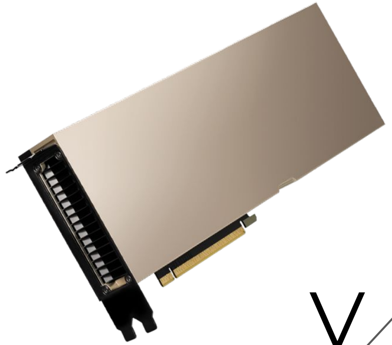
- binary
  - single file containing host code and device code
  - also known as device image
- bitstream
  - FPGA configuration file containing programming information for the FPGA
  - loaded into an FPGA when ready for execution
- board support package (BSP)
  - consists of software layers and an FPGA hardware scaffold design that makes it possible to target the FPGA
  - analogous to firmware
- board variant
  - FPGA board firmware supporting different functional capabilities of the FPGA
  - one FPGA model can have multiple board variants
- device
  - identifies the platform the compiled image will support
  - e.g. emulator or hardware

Terminology

- DPC++ (Data Parallel C++)
  - a high-level language designed for parallel programming productivity and based on the C++ language for broad compatibility
- emulator image
  - device image resulting from compiling for the FPGA emulator
  - runs on the CPU
- hardware image
  - device image resulting from compiling for the FPGA hardware
  - runs on the FPGA
- host
  - a CPU-based system (computer) that executes the primary portion of a program, specifically the application scope and command group scope
- initialize
  - prepare the board for use by programming it with a default image so it enters a “clean” state
- reconfigure
  - program the FPGA with an image to change its functionality

Why use FPGAs?

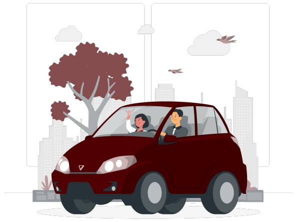
- Lower power consumption compared to a GPU
- Ability to fine-tune the hardware for specific applications, as opposed to a general purpose accelerator
- Lower latency compared to GPUs
- Increasing support of FPGA development tools



V S

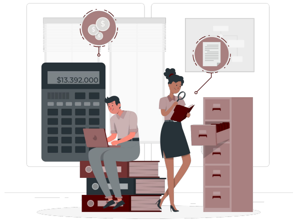
Applications of FPGAs

Artificial Intelligence

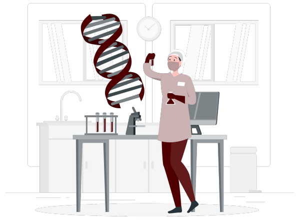


Automotive

Data Analytics

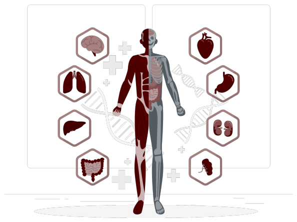


Financial Workloads

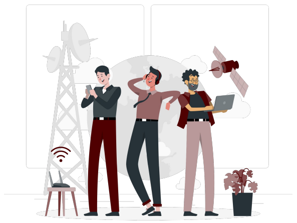


Genomics

Industrial Processes

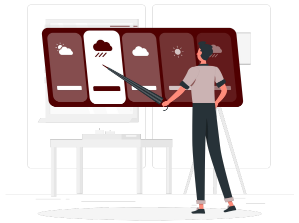


Medical Research



Network Transformation

Prototyping



Weather & Climate

Additional Reading: https://hardwarebee.com/fpga-common-applications/

High Performance
Research Computing
DIVISION OF RESEARCH

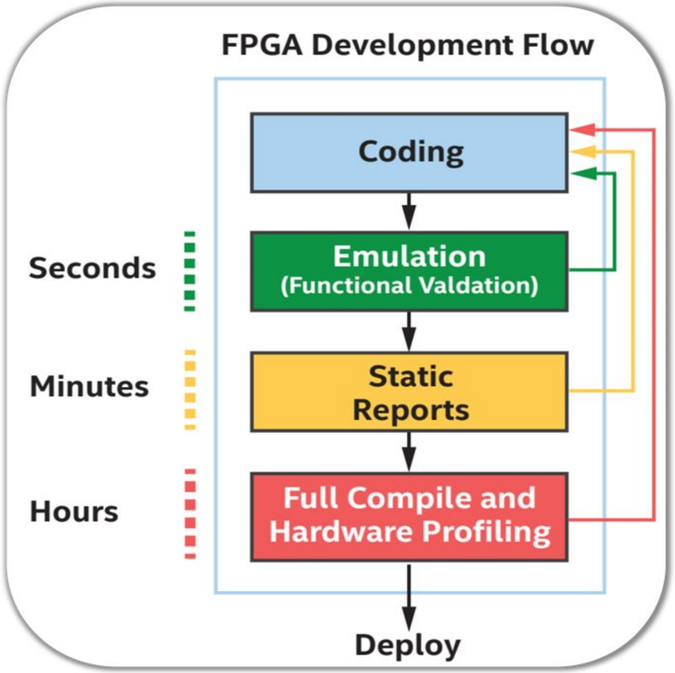
Developing on FPGAs

- Programming was traditionally done in an HDL (hardware description language) such as VHDL or Verilog.
  - Typically a steep learning curve
- Introduction of Intel oneAPI tools allow for more widespread adoption because of its foundation on C++
  - Basis in OpenCL, write applications in DPC++, and call the SYCL SDK + FPGA SDK libraries
  - CUDA to DPC++ migration tool available for compatibility of GPU code on FPGAs
    - Available here: https://www.intel.com/content/www/us/en/developer/tools/oneapi/dpc-compatibility-tool.html
  - Syclomatic by Codeplay
    - Mostly automated CUDA -> DPC++ code migration

FPGA Development: High Level Synthesis Tools

- Code written in high level language C++ to translated to RTL
- Verification can be done by execution of the code in CPU
- Enables a faster development iteration
- Example : Intel’s HLS compiler is packaged with Quartus
  - https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/hls-compiler.html

FPGA Development Flow



1. Emulation/Functional Validation
    a. code is checked for correctness using a test bench
    b. code targeting the FPGA is compiled and executed on CPU
    c. faster turnaround time for resolving bugs
2. Static Performance Analysis
    a. compiler generated reports
    b. reports include all the information required for identifying bottlenecks in the design, and suggestions for optimization techniques to resolve the bottlenecks.
3. Full Compile
    a. compiler can insert on request profiling logic into the generated hardware
    b. profiling logic generates dynamic profiling data that can later be used for identifying data pattern dependent bottlenecks that cannot be spotted in any other way

CPU vs. GPU vs. FPGA Development

	CPU	GPU	FPGA
Architecture	● Fixed architecture	● Fixed architecture	● Compiled instructions become hardware components on the board
Execution	● Instructions from the software are fetched and executed on the fixed architecture	● Instructions from the software are fetched and executed on the fixed architecture	● Software "execution" is data flowing through the deep pipelines that match the instructions from software
Software Development	● C/ C++ and other general programming languages	● CUDA, OpenCL	● RTL (Register Transfer Level)languages like VHDL or Verilog were used to synthesize the board ● High Level Synthesis tools enable design in language like C++

Break

Getting Started

Requirements

- TAMU HPRC account/ACCESS account
- Access to a terminal with SSH access
- Access to the TAMU campus network (on-campus or off-campus with VPN)
- Familiarity with job submission (interactive/batch) on TAMU HPRC clusters
- Knowledge of basic terminal usage
- Knowledge of C++ code development

Overview

This section will walk students through the necessary steps for:

- environment set-up
- device identification
- device initialization

and demonstrate:

- execution of FPGA code on an emulator
- execution of FPGA code on the FPGA device itself
- compilation of FPGA code on a CPU-only node

prior to the hands-on portion of the course.

Accessing the Cluster

Log into the FASTER cluster:

```
# TAMU Users
ssh NetID@faster.hprc.tamu.edu
# ACCESS/XSEDE Users
ssh -J UserID@faster-jump.hprc.tamu.edu:8822 UserID@login.faster.hprc.tamu.edu
```

ACCESS/XSEDE Users: A few additional steps are required prior to logging in for the first time. Please see this page for more information: https://hprc.tamu.edu/wiki/ACES#ACES_Phase_I

Accessing the Nodes - Interactive

1. Log into the FASTER cluster via a TAMU HPRC account or an ACCESS account.
2. Request an interactive session on the cluster nodes:
   a. CPU Nodes (For Code Compilation)

   ```
   srun --partition=cpu --time=12:00:00 --nodes=1 --ntasks-per-node=16 --mem=64G --pty bash -i
   ```

   b. FPGA Nodes (For Code Execution)

   ```
   srun --partition=fpga --time=24:00:00 --nodes=1 --exclusive --pty bash -i
   ```

   ℹ️ Students will not be using the FPGA nodes during the course session. There are some technical limitations that will be discussed shortly.

3. Wait for your request to be allocated.
4. Once allocated, you will be placed in an interactive session on the requested node.

Accessing the Nodes - Batch

```
#!/bin/bash

##NECESSARY JOB SPECIFICATIONS
#SBATCH --job-name=fpga_compilation
#SBATCH --time=12:00:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=16
#SBATCH --mem=64G
#SBATCH --output=%x.%j
#SBATCH --partition=fpga

module load oneAPI/2022.3.0
aocl initialize acl0 pac_s10
# commands to compile/run code
```

```
Indicate the file is a bash script


Set the job name to “fpga_compilation”
Set the wall clock limit to 12hrs
Request one (1) node
Request 16 tasks per node
Request 64GB per node
Redirect stdout/err to file called "[job-name].[jobid]"
Specify partition to submit job to

Set up the oneAPI environment variables
[FPGA partition only] Initialize the FPGA device

```

The contents shown in this example must be entered into a text file, then executed as such:

```
sbatch job.slurm
```

Interactive vs. Batch Job Methods

Interactive Jobs

- Pros
  - Better suited for shorter tasks
  - Good for testing a workflow before committing to a batch job
  - Good for graphical interfaces
- Cons
  - Exiting out of the session ends the job e.g. if you lose internet connection, your job terminates
  - Resources can remain idle if not attended to

Batch Jobs

- Pros
  - Better suited for longer tasks
  - Good for unattended workload processing
  - No need for manual intervention
  - Helps with reproducibility
  - Better suited for running multiple workloads simultaneously
- Cons
  - Not good if a workload requires frequent interaction

Notes on FPGA Usage

- Currently, there is only one (1) FPGA per node within the FPGA queue.
  - Due to device constraints, only one user may use an FPGA node at a time.
- FPGA emulator and hardware images can be compiled on a CPU-only node, but can only be run on a node with an FPGA installed.
  - This means users do not need to wait for an FPGA node to be available to compile their code, but will need to request one to run their code.

Load the oneAPI Module

```
# input commands
bash
module load oneAPI/2022.3.0
```

- If working interactively, it helps to launch another bash shell within your session to isolate your environment variables from your primary session.
    - Exiting out of the shell will return you to the shell you started in.
    - This is helpful for starting with a clean environment.

⚠️ The module is still a work in progress, please report any issues to help@hprc.tamu.edu

Device Identification

```
# input commands
aocl diagnose     # can also use ‘list-devices’
```

ℹ️ This can be run on either the CPU or FPGA nodes, but since standard CPU nodes will not have FPGAs installed, no device will be shown.

1. **Device Name:** used to reference the device with respect to the aocl tools
2. **Status**: indicates whether or not the device is ready to develop on. A value of “Uninitialized” means the device needs to be initialized with a compatible board variant image before use
3. **USM [not] supported**: indicates whether or not the device is currently configured to compile applications with USM capabilities

ℹ️ Additional subcommands for aocl can be viewed by running the following:

```
aocl help
```

```
# example output
⋮
--------------------------------------------------------------
BSP Diagnostics
--------------------------------------------------------------
--------------------------------------------------------------
Device Name:     1
acl0

BSP Install Location:
/opt/intel/oneapi/intelfpgadpcpp/2022.1.0/board/intel_s10sx_pac

Vendor: Intel Corp

Physical Dev Name   Status       Information
                           2
pac_f200000         Passed       Intel PAC Platform
(pac_f200000)
                                 PCIe 29:00.0
                                 USM not supported     3


DIAGNOSTIC_PASSED
--------------------------------------------------------------
⋮
```

Device Initialization

```
Syntax: aocl initialize <device_name> [board_variant]
```

```
# input commands
aocl initialize acl0 pac_s10    # or 'pac_s10_usm'
```

```
# example output: pac_s10
⋮
Physical Dev Name   Status            Information

pac_f200000         Passed            Intel PAC Platform (pac_f200000)
                                      PCIe 29:00.0
                                      USM not supported     1

⋮
```

```
# example output: pac_s10_usm
⋮
Physical Dev Name   Status            Information

pac_f200000         Passed            Intel PAC Platform (pac_f200000)
                                      PCIe 29:00.0
                                      USM supported         2

⋮
```

ℹ️ Specifying "board_variant" is optional. Without it, the default initialization will use "pac_s10".

ℹ️ This can be run on either the CPU or FPGA nodes, but since standard CPU nodes will not have FPGAs installed, no device will be detected as a valid target.

1. Device is initialized with a Stratix 10 image **without USM** support, by specifying the "pac_s10" board variant.

2. Device is initialized with a Stratix 10 image **with USM** support, by specifying the "pac_s10_usm" board variant.

Demo 1: fpga_compile - emulator version

```
cp -R /scratch/training/aces-fpga $SCRATCH
cd $SCRATCH/aces-fpga/fpga_compile
mkdir build && cd build
aocl initialize acl0 pac_s10
cmake .. -DFPGA_DEVICE=intel_s10sx_pac:pac_s10
make fpga_emu
./fpga_compile.fpga_emu
```

```
Copy the training materials to your $SCRATCH directory
Navigate to the ‘fpga_compile’ example directory
Create and navigate to the ‘build’ directory
Ensure the FPGA device is configured correctly
Run ‘cmake’ to create input files for ‘make’
Build the emulator application
Run the emulator application
```

```
# output
Running on device: Intel(R) FPGA Emulation Device
PASSED: results are correct
```

Binary is running on the emulator on the CPU, instead of an FPGA device.

Demo 2: fpga_compile - hardware version

ℹ️ This slide serves only as an example demonstrating the execution of a binary compiled for one board variant on an FPGA currently using a different board variant.

```
cp -R /scratch/training/aces-fpga $SCRATCH
cd $SCRATCH/aces-fpga/fpga_compile
mkdir build && cd build
aocl initialize acl0 pac_s10
cmake .. -DFPGA_DEVICE=intel_s10sx_pac:pac_s10
make fpga
./fpga_compile.fpga
```

```
Copy the training materials to your $SCRATCH directory
Navigate to the ‘fpga_compile’ example directory
Create and navigate to the ‘build’ directory
Ensure the FPGA device is configured correctly
Run ‘cmake’ to create input files for ‘make’
Build the device application
Run the device application
```

```
# correct output
Running on device: pac_s10 : Intel PAC Platform (pac_f200000)
PASSED: results are correct
```

Binary is running on the FPGA device itself, instead of an emulator.

```
# incorrect output - pac_s10 (without USM) image run on USM-enabled platform (pac_s10_usm)
Running on device: pac_s10_usm : Intel PAC Platform (pac_f200000)
Caught a SYCL host exception:
Native API failed. Native API returns: -42 (CL_INVALID_BINARY) -42 (CL_INVALID_BINARY)
terminate called after throwing an instance of 'cl::sycl::runtime_error'
 what():  Native API failed. Native API returns: -42 (CL_INVALID_BINARY) -42 (CL_INVALID_BINARY)
Aborted (core dumped)
```

FPGA Development Flow on CPU

- Effectively the same as if compiling on the FPGA node, but running a requested CPU node.

```
# input commands
bash
module load oneAPI/2022.3.0
# navigate to project directory
make fpga
```

```
[ 50%] Building CXX object src/CMakeFiles/compute_units.fpga.dir/compute_units.cpp.o
[100%] Linking CXX executable ../compute_units.fpga
aoc: Compiling for FPGA. This process may take several hours to complete.  Prior to performing this compile, be sure to check the
reports to ensure the design will meet your performance targets.  If the reports indicate performance targets are not being met,
code edits may be required.  Please refer to the oneAPI FPGA Optimization Guide for information on performance tuning applications
for FPGAs.
[100%] Built target compute_units.fpga
[100%] Built target fpga
```

Break

Hands-On Demo

Overview

This section will walk students through the necessary steps for:

- creating a CPU node session and configuring the environment
- compile and run FPGA code examples on an emulator

Set-Up

Log-in to FASTER:

```
# TAMU Users
ssh NetID@faster.hprc.tamu.edu
# ACCESS/XSEDE Users
ssh -J UserID@faster-jump.hprc.tamu.edu:8822 UserID@login.faster.hprc.tamu.edu
```

Get a copy of the oneAPI FPGA course files:

```
cp -R /scratch/training/aces-fpga $SCRATCH
```

Create a CPU-only interactive session:

```
srun --reservation=aces-fpga --partition=cpu --time=12:00:00 --nodes=1 --mem=64G --pty bash -i
```

ℹ️ The reservation is only available for the duration of the short course, and for some time after its conclusion. Any submissions going forward should not include the –reservation=aces-fpga flag, and will be launched in the general CPU pool.

Set-Up

Navigate to the copied directory:

```
cd $SCRATCH/aces-fpga/fpga_lab
```

Open another shell, and load the oneAPI module:

```
bash
module load oneAPI/2022.3.0
```

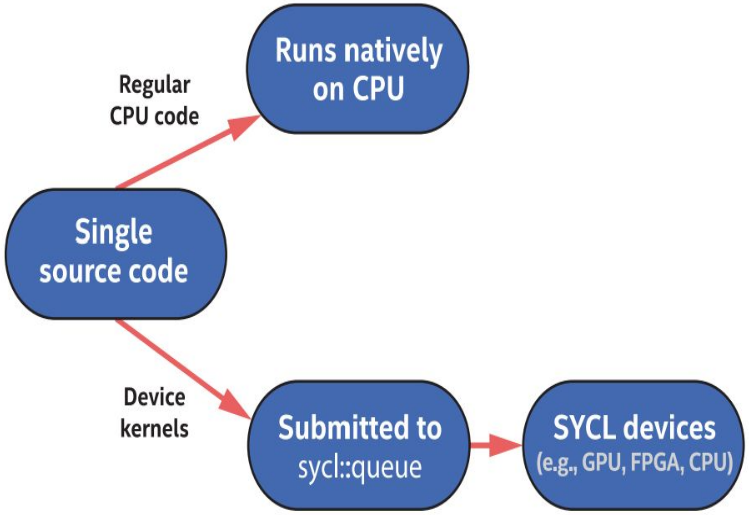
ℹ️ Please recall, if working interactively, it helps to launch another bash shell within your session to isolate your environment variables from your primary session due to the current format of the module. Exiting out of the shell will return you to the shell you started in.

⚠️ The module is still a work in progress, please report any issues to help@hprc.tamu.edu

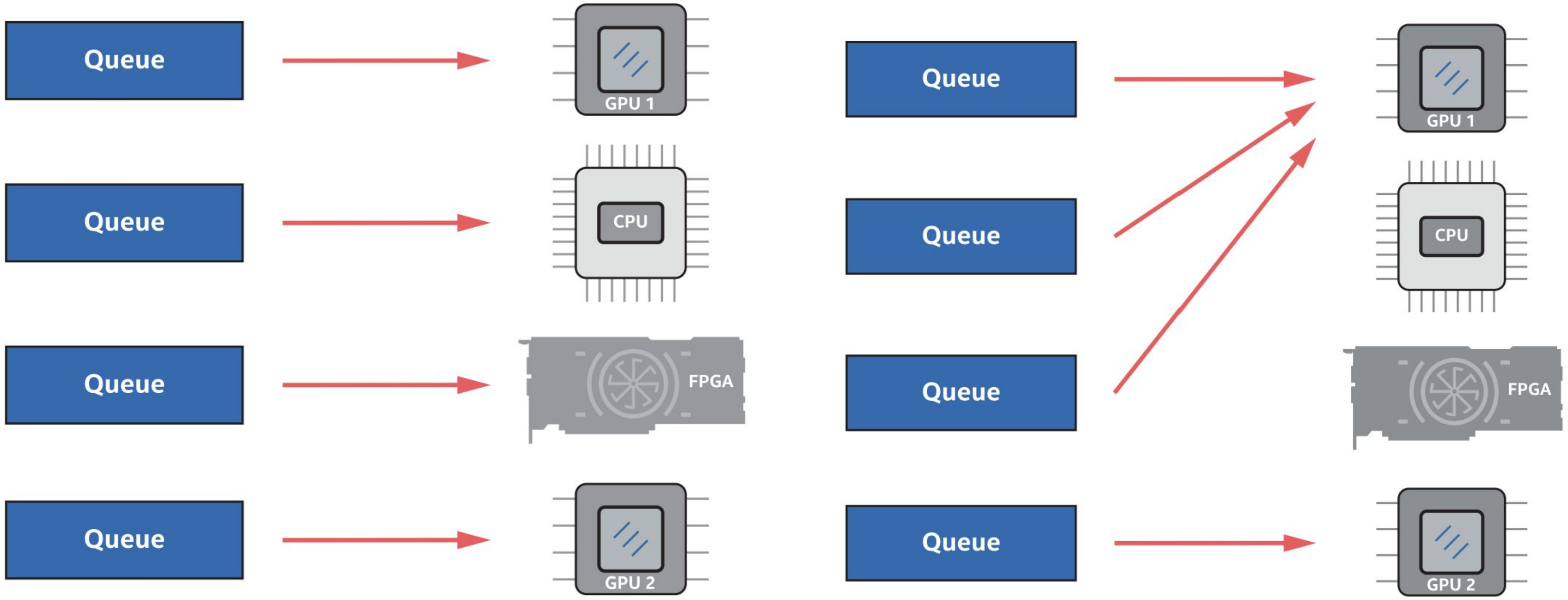
Exercise 0: Code Execution

```
dpcpp -fintelfpga -DFPGA_EMULATOR ex_0_sample.cpp -o ex_0_sample.fpga_emu
./ex_0_sample.fpga_emu
```

Please execute the above instructions



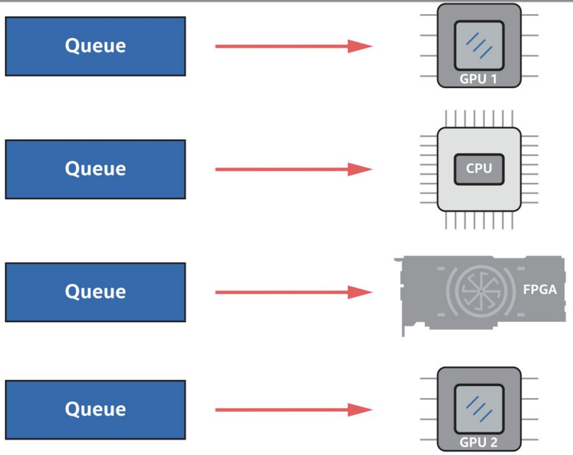
Exercise 1: Device Selection

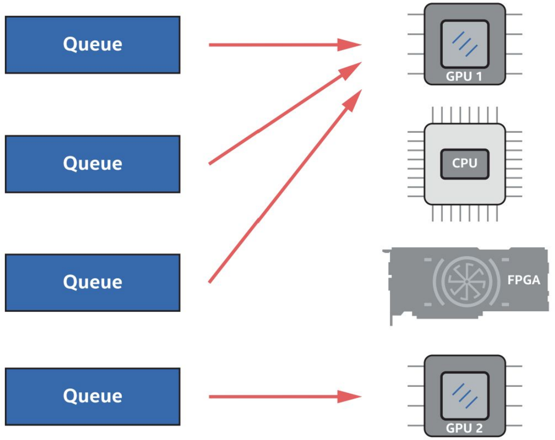


Exercise 1: Device Selection

```
dpcpp -fintelfpga -DFPGA_EMULATOR ex_1_deviceSelector.cpp -o
ex_1_deviceSelector.fpga_emu
./ex_1_deviceSelector.fpga_emu

# Modify ex_1_deviceSelector.cpp to use accelerator in the queue
```



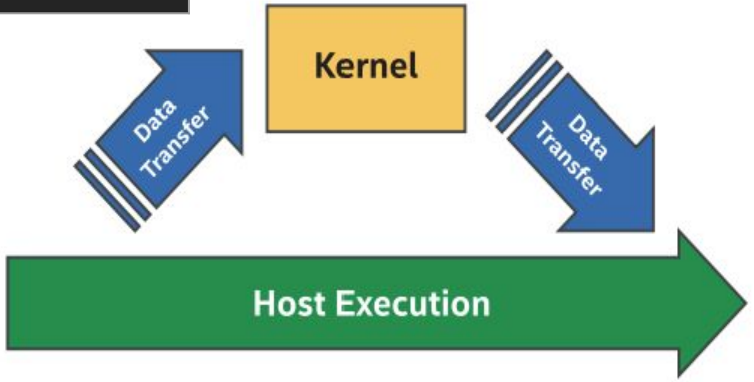


Ref: Data Parallel C++

Exercise 2-3: Data Management: USM and Buffers

```
# exercise-2
dpcpp -fintelfpga -DFPGA_EMULATOR ex_2_usm.cpp -o ex_2_usm.fpga_emu
./ex_2_usm.fpga_emu

# exercise-3
dpcpp -fintelfpga -DFPGA_EMULATOR ex_3_buffer_accessor.cpp -o
ex_3_buffer_accessor.fpga_emu
./ex_3_buffer_accessor.fpga_emu
```

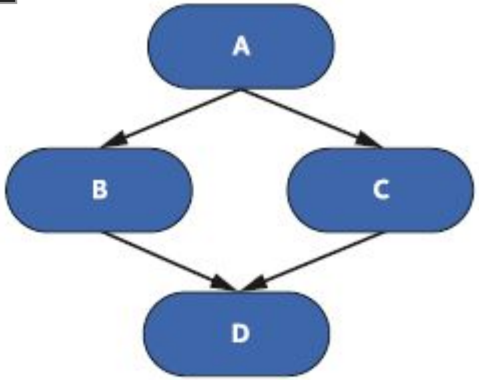


Exercise 4: Task Graph

```
dpcpp -fintelfpga -DFPGA_EMULATOR ex_4_matrixAdd.cpp -o
ex_4_matrixAdd.fpga_emu
./ex_4_matrixAdd.fpga_emu

# Modify the code to print the matrix with all 4’s
```

$$\begin{tikzcd} & A \arrow[dl] \arrow[dr] & \\ B \arrow[dr] & & C \arrow[dl] \\ & D & \end{tikzcd}$$



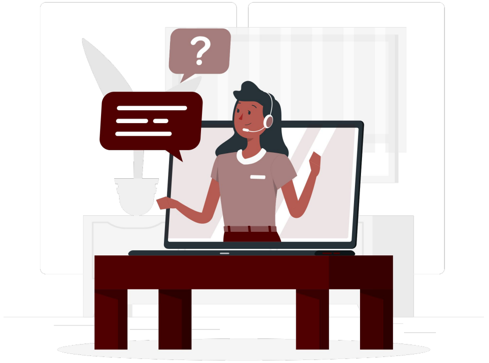
Best Practices

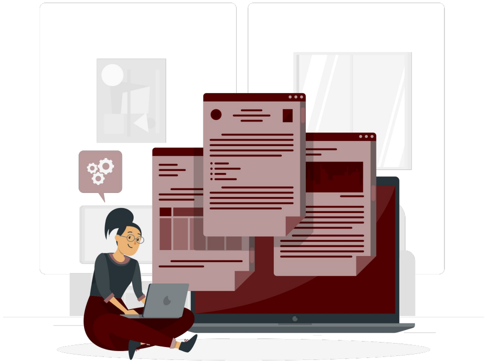
- **Check the initialization state of a card prior to running your executable.** It may have become uninitialized between users or runs.
- **Do not reinitialize devices too quickly between initializations and reconfigurations.** This may cause the card to become uninitialized or have other errors.
- **Compilation of either an emulator/device image for an application can take place on a CPU-only node.** This means you do not have to wait for an FPGA node to be available to compile your application.
- **Validate your design through the emulator on a CPU node prior to requesting an allocation for an FPGA node.** This allows other users who are further ahead in their development flow to test their design on the FPGA node when required.

Wrapping Up

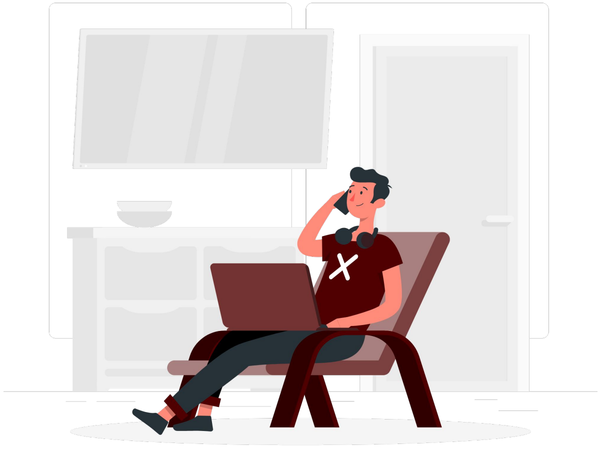
Resources

- Intel Documentation
    - https://www.intel.com/content/www/us/en/develop/documentation/oneapi-programming-guide/top.html
- Intel FPGA Technical Training
    - https://www.intel.com/content/www/us/en/support/programmable/support-resources/fpga-training/catalog.html?s=AtoZ
- Intel Learning
    - https://learning.intel.com/developer/learn
- oneAPI Samples on GitHub
    - https://github.com/oneapi-src/oneAPI-samples
- Intel Developer Cloud
    - https://www.intel.com/content/www/us/en/developer/tools/devcloud/overview.html
- Explore SYCL* Through Intel® FPGA Code Samples
    - https://www.intel.com/content/www/us/en/developer/articles/code-sample/explore-dpcpp-through-intel-fpga-code-samples.html





Support



help@hprc.tamu.edu

979-845-0219

https://hprc.tamu.edu/wiki

References

[1] *FPGA BSPS and boards*. Intel. (n.d.). Retrieved November 29, 2022, from https://www.intel.com/content/www/us/en/develop/documentation/oneapi-programming-guide/top/programming-interface/fpga-flow/fpga-bsps-and-boards.html

[2] *Intel® oneAPI Programming Guide - FPGA flow*. Intel. (n.d.). Retrieved November 29, 2022, from https://www.intel.com/content/www/us/en/develop/documentation/oneapi-programming-guide/top/programming-interface/fpga-flow.html

[3] *Intel® oneAPI Programming Guide - Glossary*. Intel. (n.d.). Retrieved November 29, 2022, from https://www.intel.com/content/www/us/en/develop/documentation/oneapi-programming-guide/top/glossary.html

[4] *oneAPI Software Tool Flow Frame for FPGAs by Intel – BittWare*. BittWare. (n.d.). Retrieved November 29, 2022, from https://www.bittware.com/ip-solutions/intel-oneapi/

[5] *Samples for Intel oneapi toolkits*. GitHub. Retrieved November 29, 2022, from https://github.com/oneapi-src/oneAPI-samples

[6] *Xilinx SDK Concepts: FPGA bitstream*. Xilinx. (n.d.). Retrieved November 29, 2022, from https://www.xilinx.com/htmldocs/xilinx2018_1/SDK_Doc/SDK_concepts/concept_fpgabitstream.html