ADIUVO

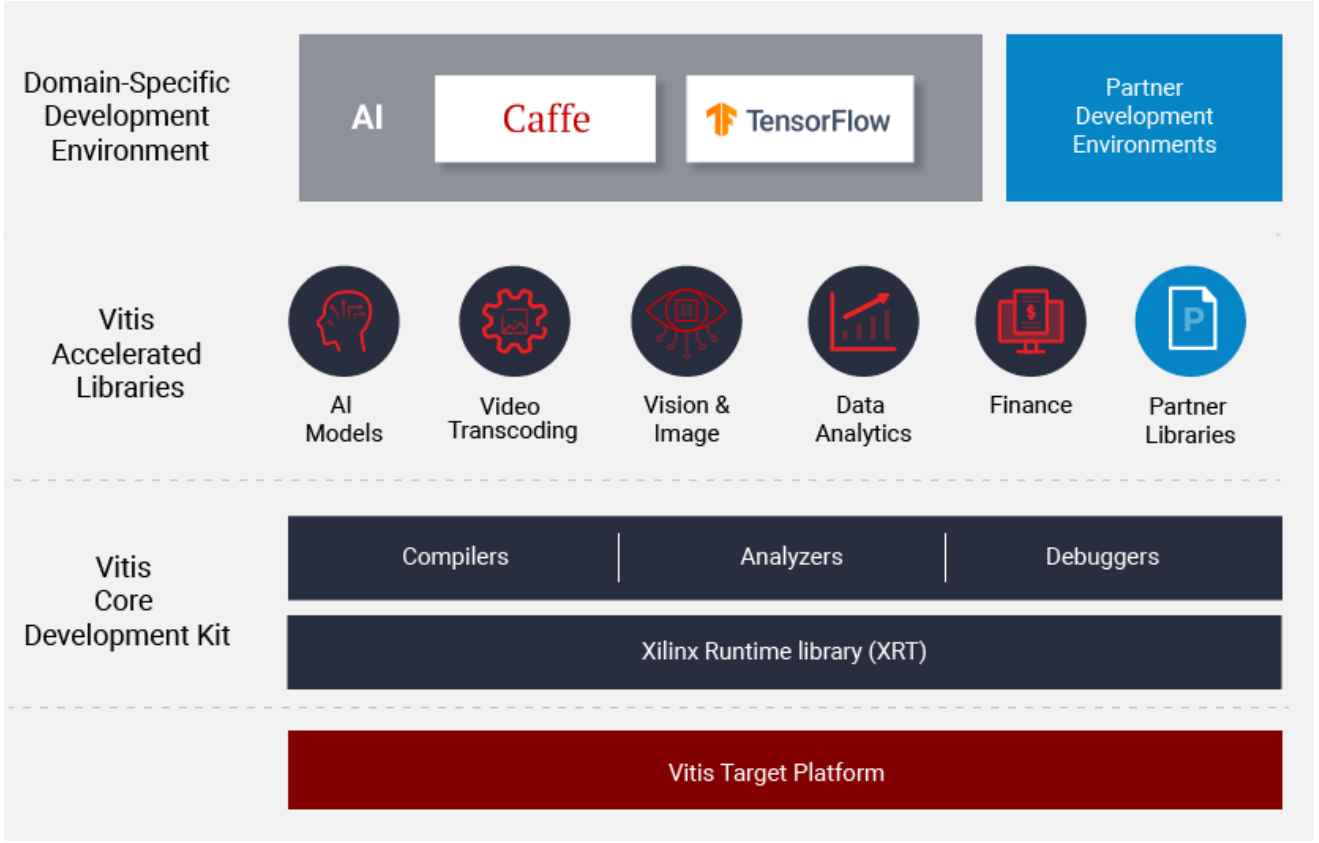
ENGINEERING AND TRAINING, LTD.

Building Accelerated Applications with Vitis

Sponsored by

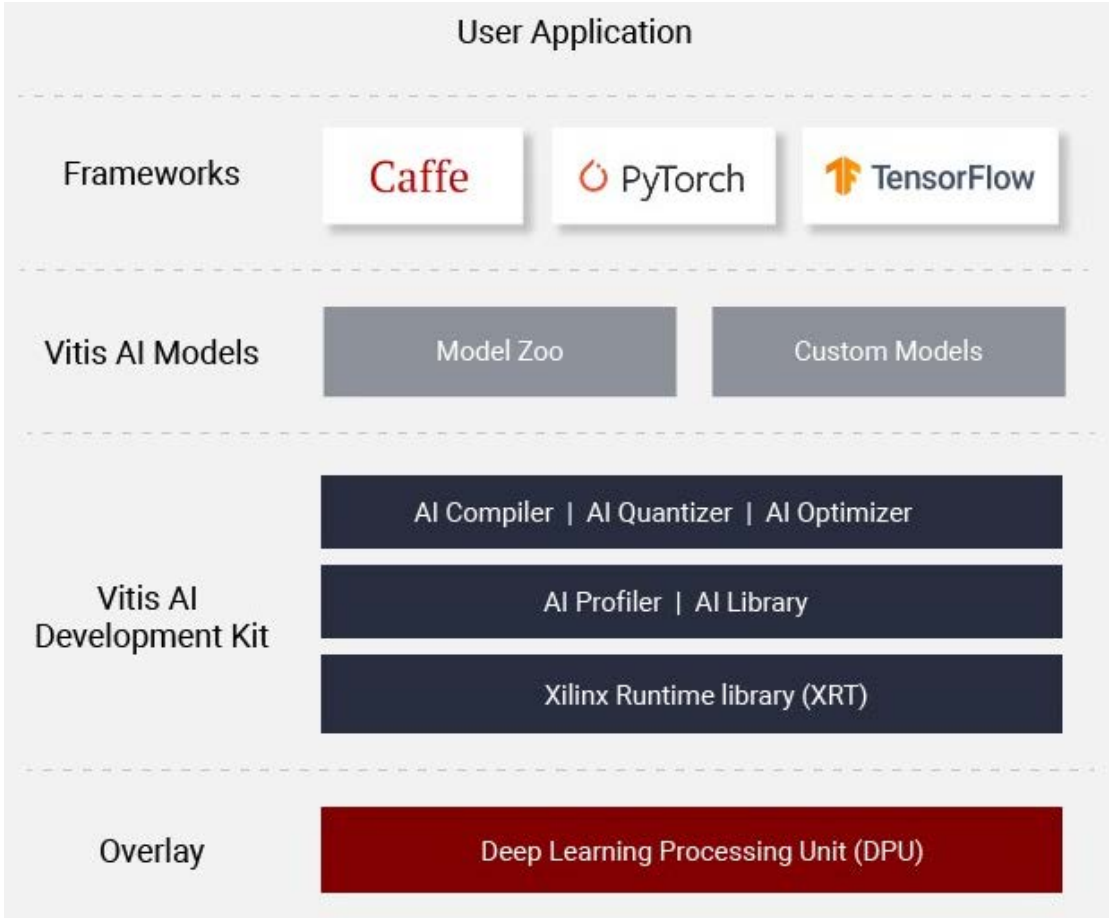
What is Vitis?

- Vitis is unified software development environment from Xilinx
- Edge and cloud development methodologies
- Support embedded and accelerated flows



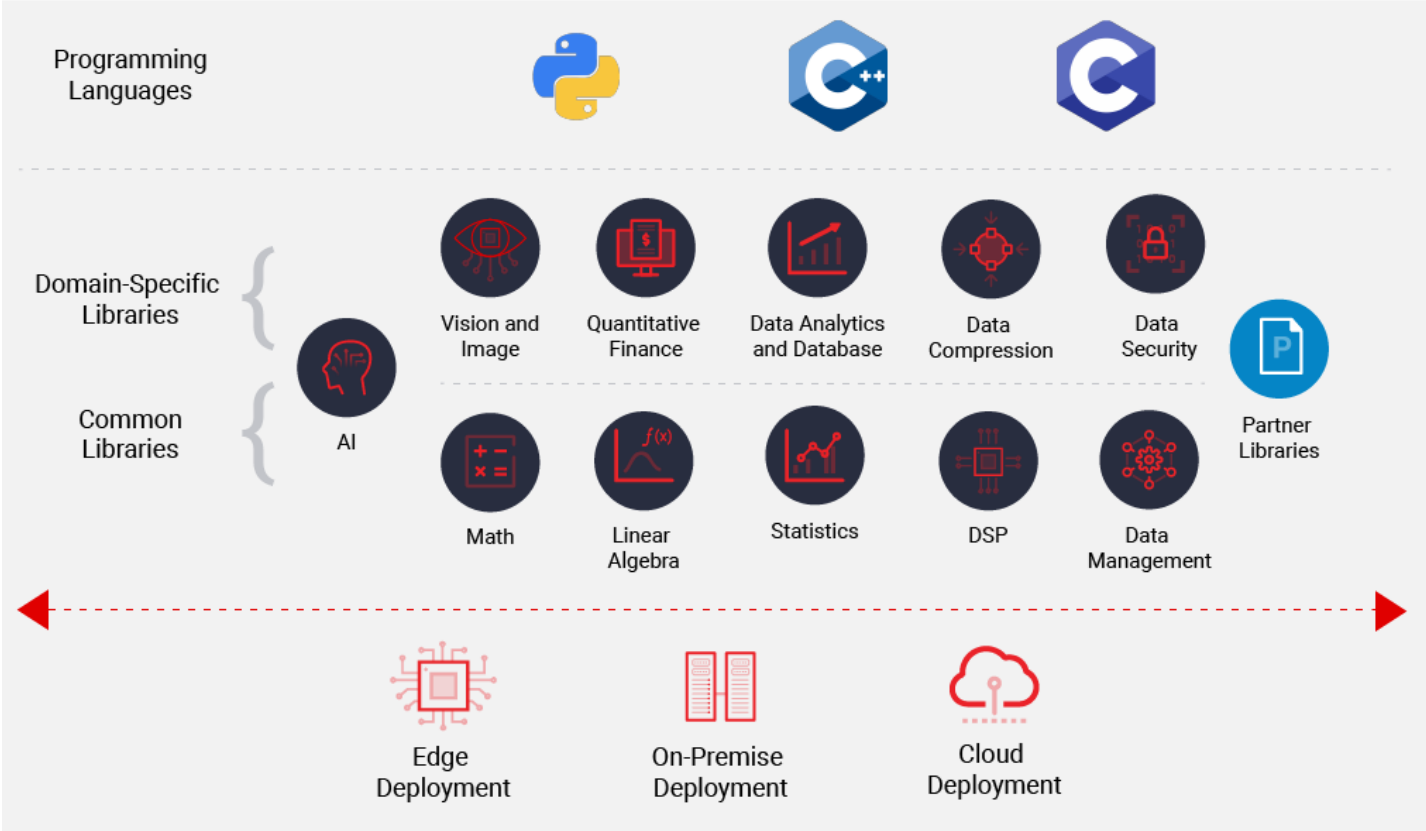
Vitis AI Development Environment

- Vitis AI enables acceleration of AI inference at edge and cloud
- Supports leading frameworks such as TensorFlow, Caffe and Pytorch
- Works with fixed-point representation and Xilinx Deep Learning Processor Unit (DPU)



Vitis Accelerated Libraries

- Open Source acceleration ready libraries
- Common libraries offer a set of common functionality
- Domain-specific libraries offer out of the box functions for specific domains (e.g., vision)



Vitis Core Development Kit

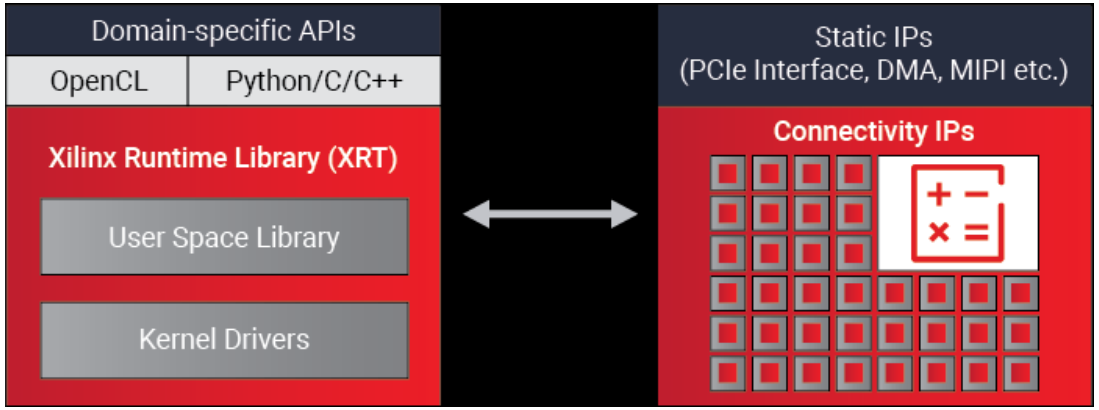
- GUI and command line tools for compilation, debug and analysis of C, C++ and OpenCL designs
- Use preferred GUI or integrated GUI
- Supports embedded and accelerated flows

Vitis Target Platforms

- Embedded
  - SoC: MPSoC, RFSoC, Zynq
  - FPGA: MicroBlaze
- Cloud
  - Alveo
  - AWS F1 Instance
- Embedded SoC and cloud applications can use acceleration flow
- All required files and boot elements are generated

Xilinx Runtime library

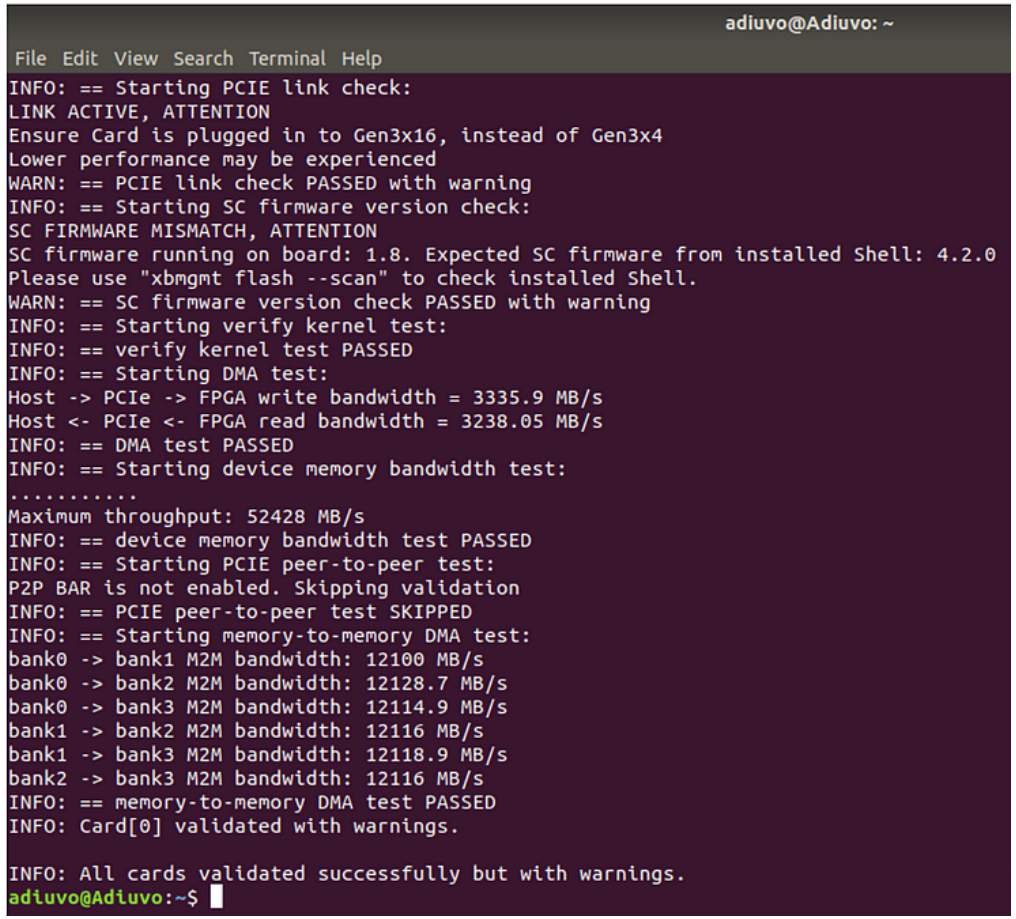
- Xilinx Runtime library (XRT) enables communication between the host and accelerator
- Cloud-based – Host x86
- Embedded – Arm Cortex-A9 or A53
- Provides all libraries, APIs, drivers and utilities.



Xilinx Runtime library

Key functions of the Runtime include:

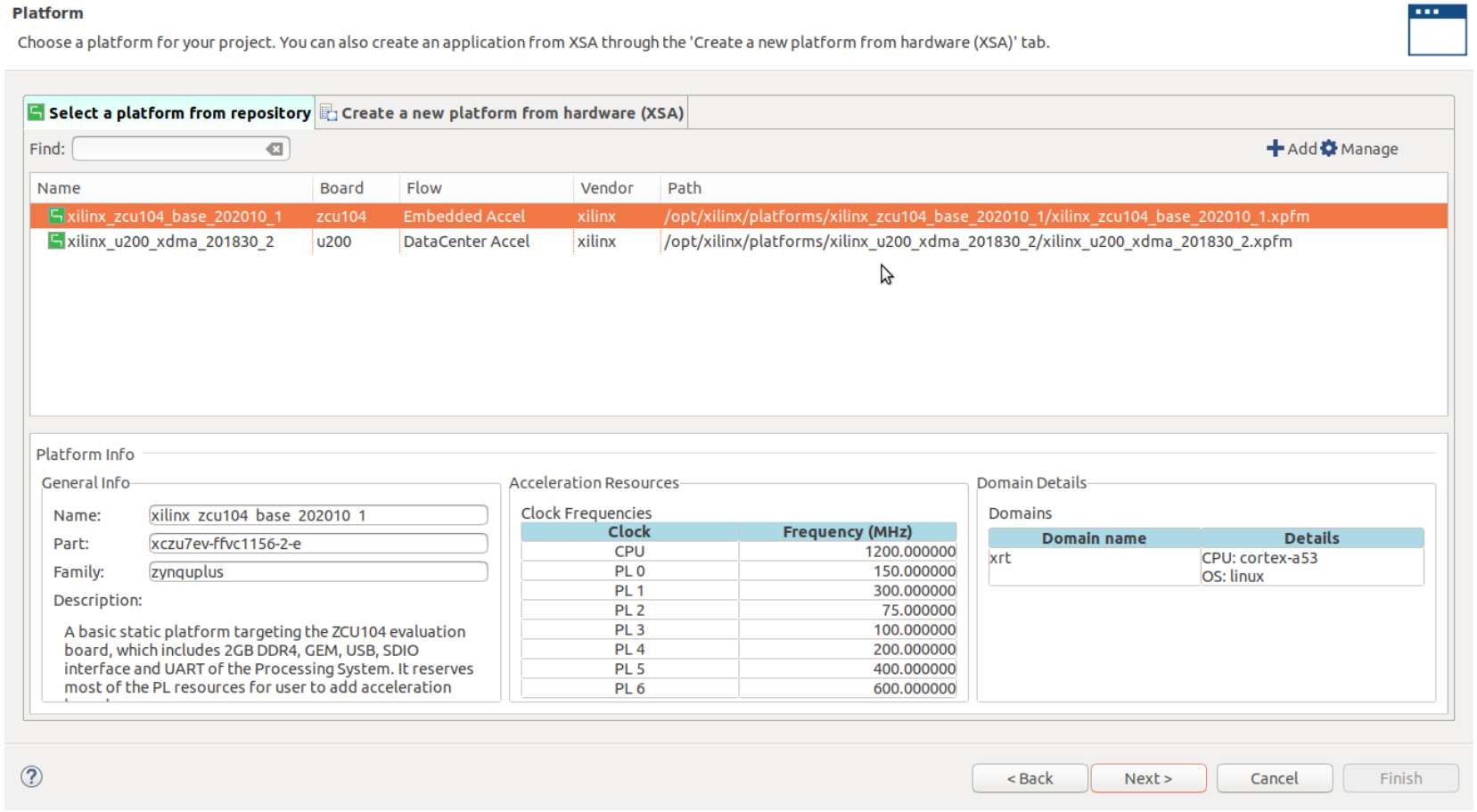
- Downloading the FPGA binary
- Memory management between host and accelerator
- Execution management
- Board management



Element of Vitis

All projects require a platform

- Hardware element – makes available AXI connections, clocks and Interrupts in the PL to Vitis Compiler
- Software element – provides boot, XRT and QEMU support
- Linux element – FS, Image and SysRoot



ADIUVO ENGINEERING AND TRAINING, LTD.

Vitis Output

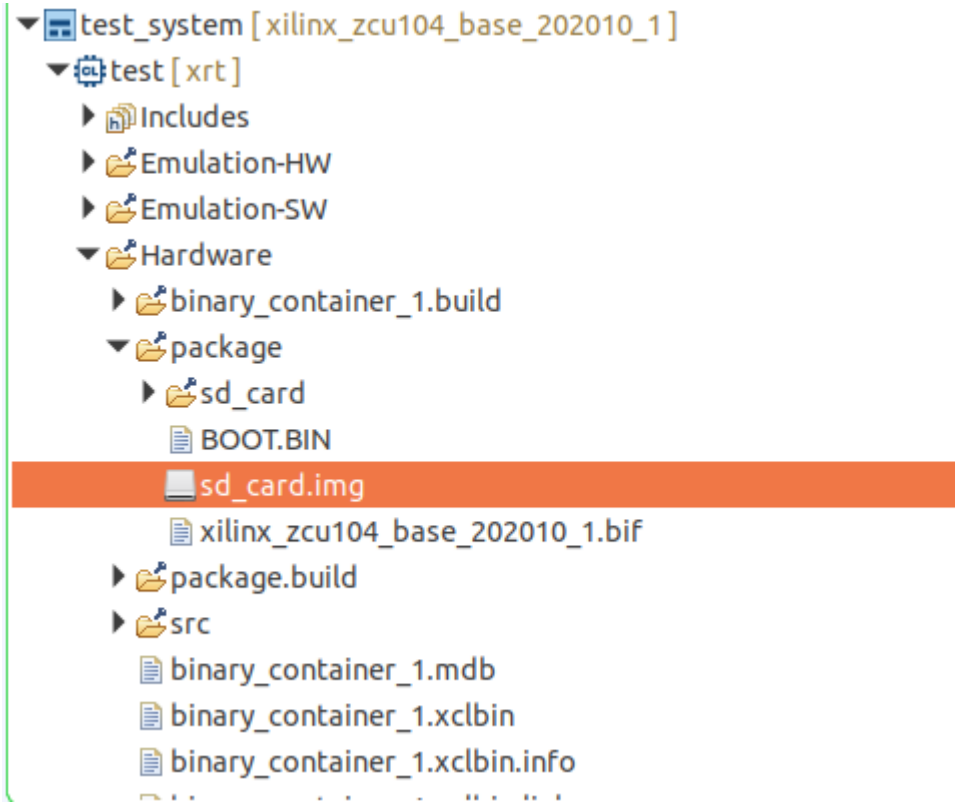
Compiled binary (host) and XCLbin (accelerator)

Embedded System Output

- SD Card Image
  - Image
  - File System
  - Binary and XCLBin

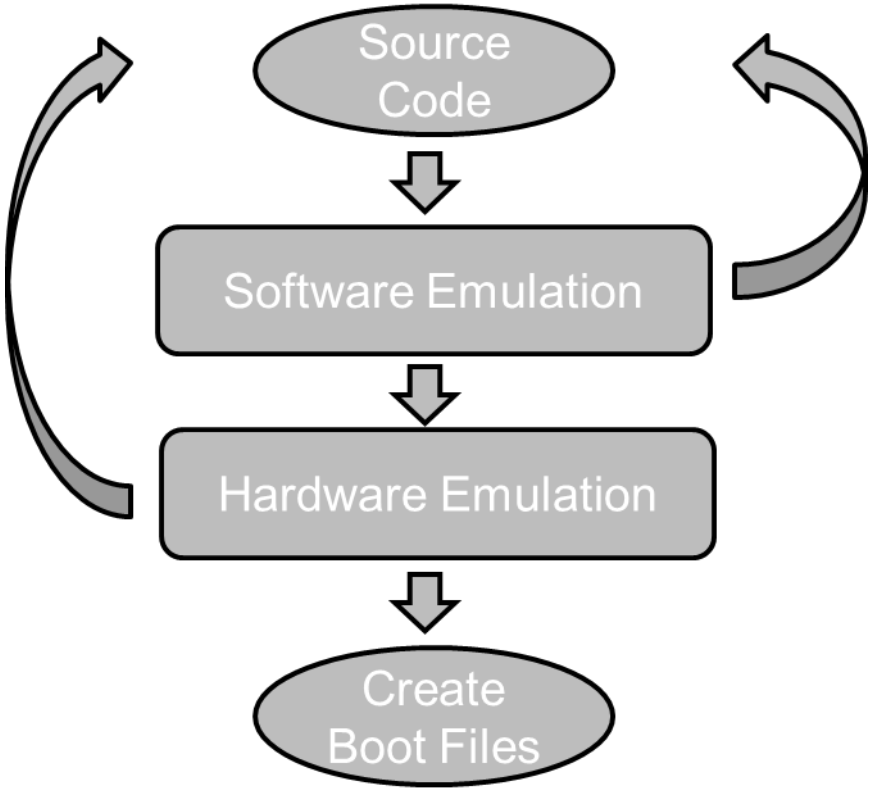
Cloud output

- Binary and XCLBin



Vitis Development Flow

- Software Emulation – Syntax errors and algorithm verification
- Hardware Emulation – Optimize performance, interfacing and resources



OpenCL Framework

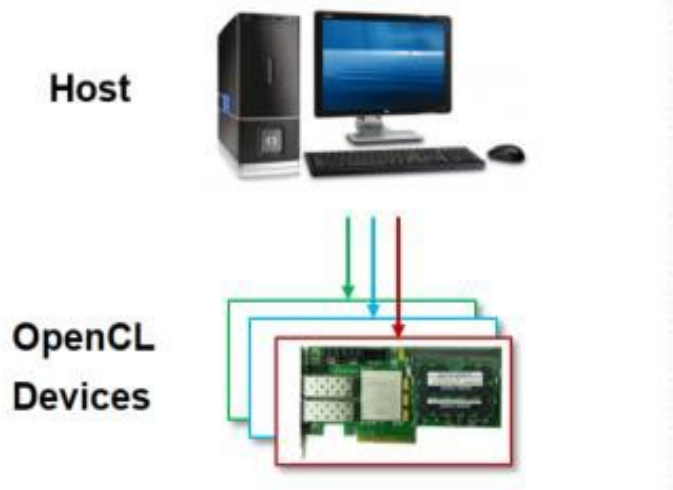
- An open industry standard
  - For parallel computing
  - Of heterogeneous systems
- Enables cross-platform functional portability
  - No code changes
  - Portable across CPU, GPU, FPGA, DSP, etc.
  - Important: No performance portability
- Wide market adoption
  - Support implemented by technology leaders
  - Many companies developing applications

OpenCL

Khronos Group

www.khronos.org

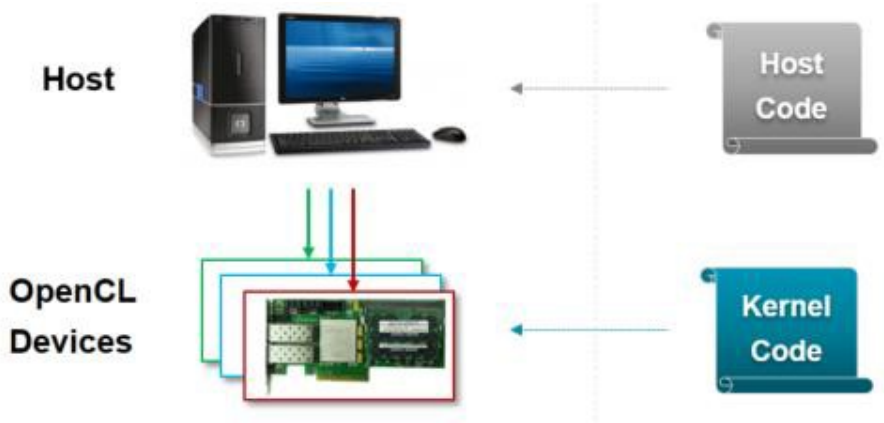
OpenCL Framework



Platform model

- Defines representation of ANY platform
- Contains
  - Single host
  - One or more OpenCL devices (compute device)

OpenCL Framework



Platform model

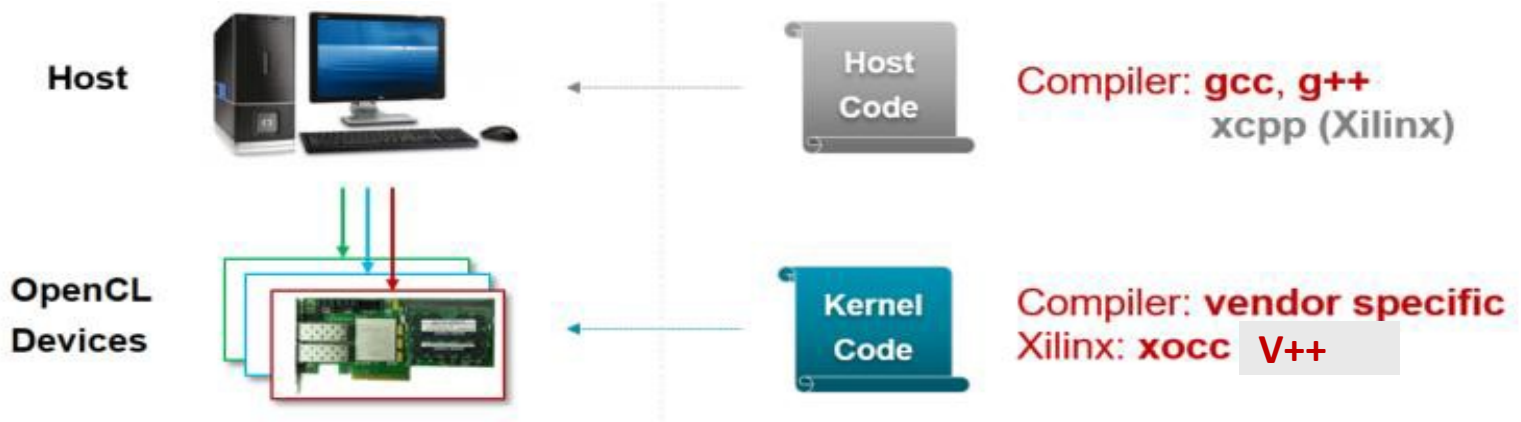
- Defines representation of ANY platform
- Contains
  - Single host
  - One or more OpenCL devices (compute device)

Execution model

OpenCL application (two parts)

- Host program
  - Manages the entire application: OpenCL APIs
- Kernels (OpenCL C language)
  -Functions to accelerate, run on OpenCL devices

OpenCL Framework



Platform model

- Defines representation of ANY platform
- Contains
  - Single host
  - One or more OpenCL devices (compute device)

Execution model

OpenCL application (two parts)

- Host program
  - Manages the entire application: OpenCL APIs
- Kernels (OpenCL C language)
  - Functions to accelerate: run on OpenCL devices

Execution Model – Command Queues

Interaction between host and device occurs via command queues

- Created by host
- Attached to a single device

Note: Multiple command queues can be active within context

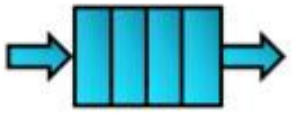
Three command types:

- Kernel execution commands
- Memory commands
  - Transfer data between host and different memory objects
- Synchronization commands
  - Put constraints on in the order in which commands are executed

Memory Model

Three types of memory objects:

- Buffer objects
  - Contiguous block of memory
  - Available to kernels for read/write
  - Programmer can write data to buffers
  - Access to data via pointers
- Image objects (not a part of embedded profile)
  - Hold images only
  - Storage/format can be optimized for specific OpenCL device
  - OpenCL framework provides functions to manipulate images
- Pipes
  - Data organized as FIFO
  - Accessed (read/write) via built in
  - Pipe not accessible from the host



Five Sub-regions of Memory Objects

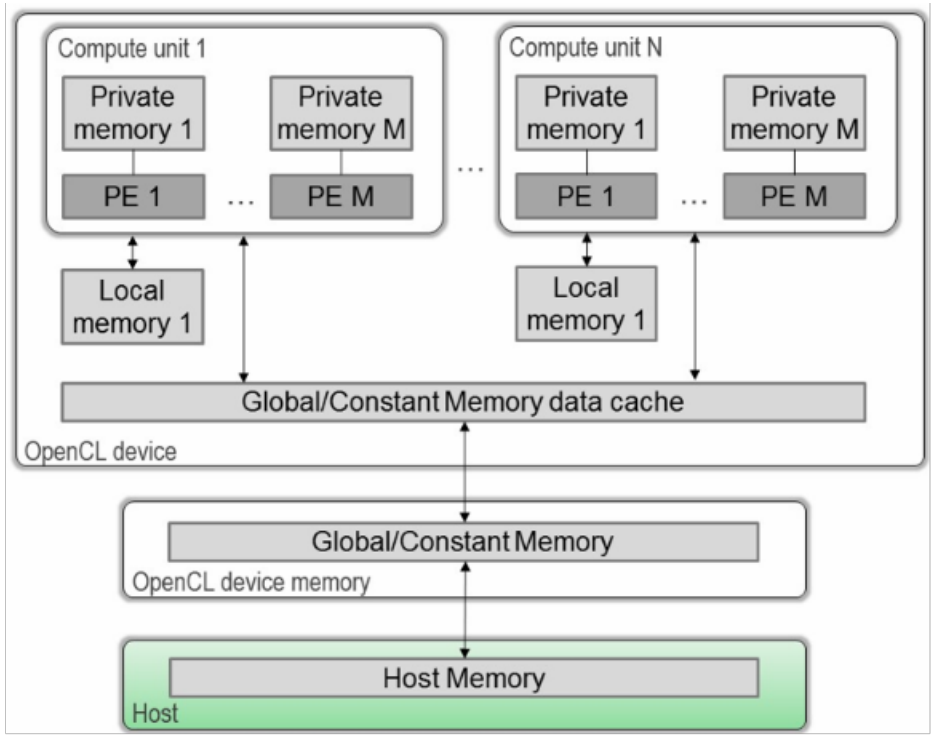
Host memory
- Visible to host only
- OpenCL framework only defines how host memory interacts with OpenCL objects

Global memory
- Visible to host and device
- All work items in all workgroups can read/write there
- Global on-chip memory – visible to device only

Constant memory
- Region of global memory
- Work items – read access only

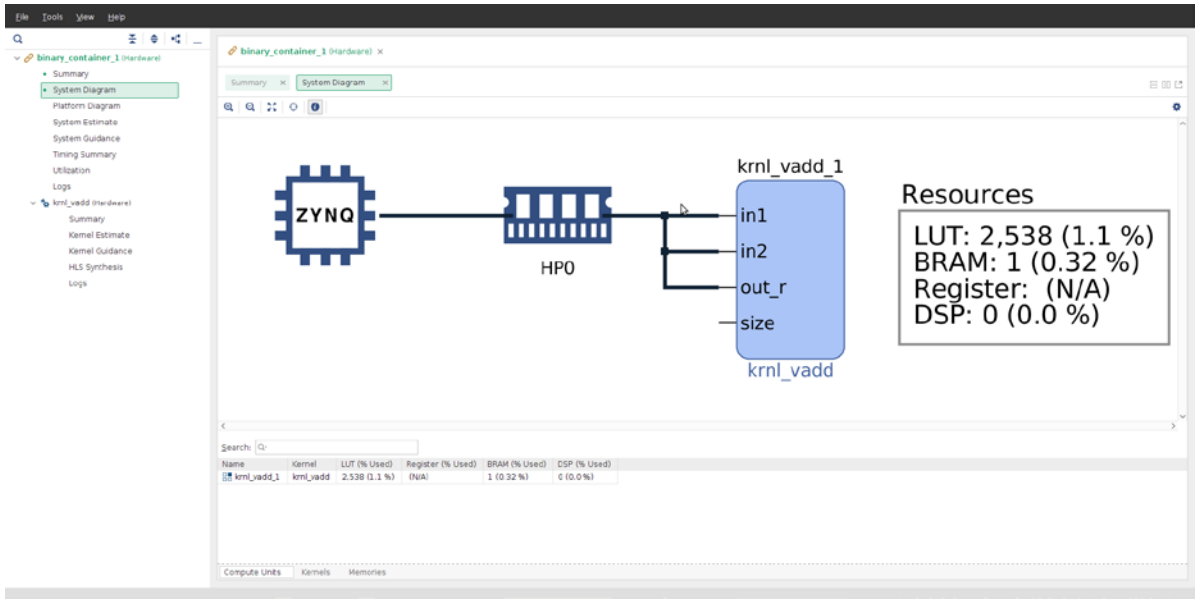
Local memory
- Local to workgroup (shared by all work-items in a group)

Private memory
- Accessible by a work-item



Optimization

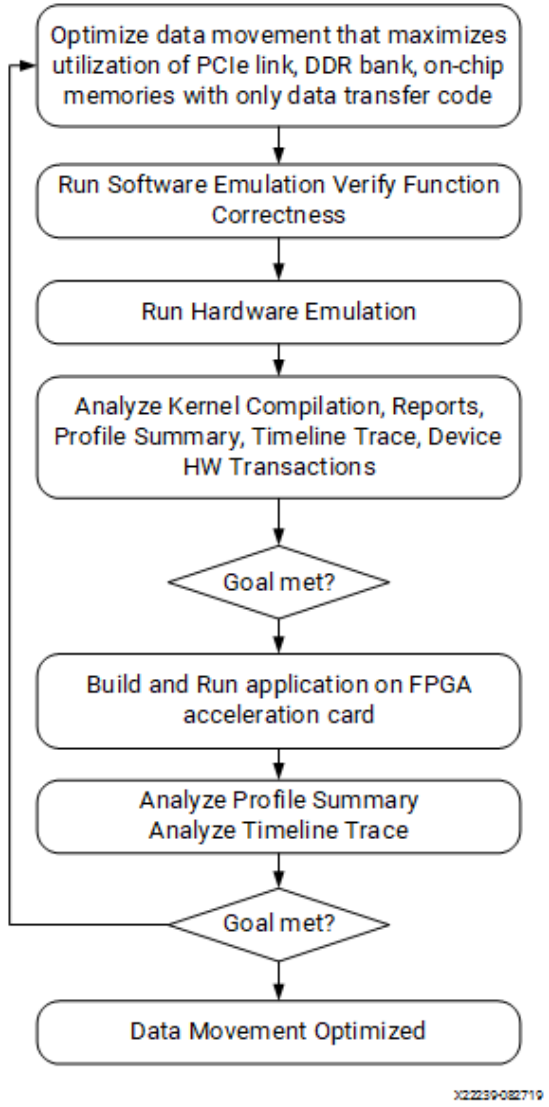
- Optimization possible at both host and kernel
- Enables most responsive solution
- Host optimization
- Kernel optimization possible in OpenCL and C/C++
  - Optimization syntax differs



Host Optimization

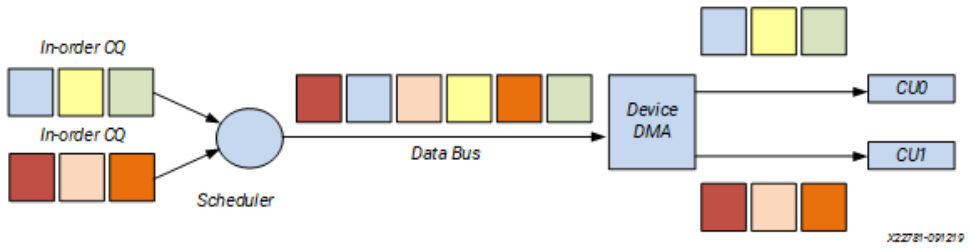
- Optimize the data movement in the application before optimizing computation

- Compute Unit Scheduling
  - Multiple In-Order Command Queues
  - Single Out-of-Order Command Queue

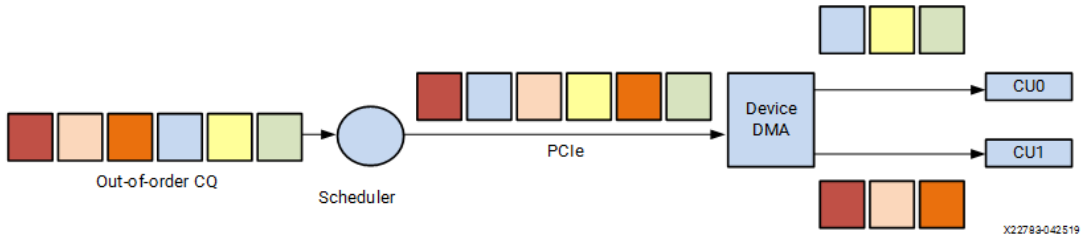


Host Optimization

Multiple In-Order Command Queues

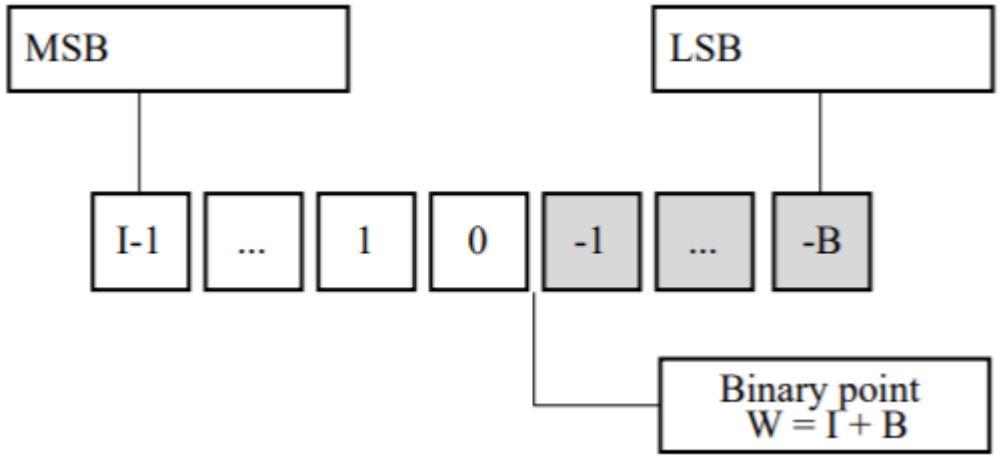


Single Out-of-Order Command Queue



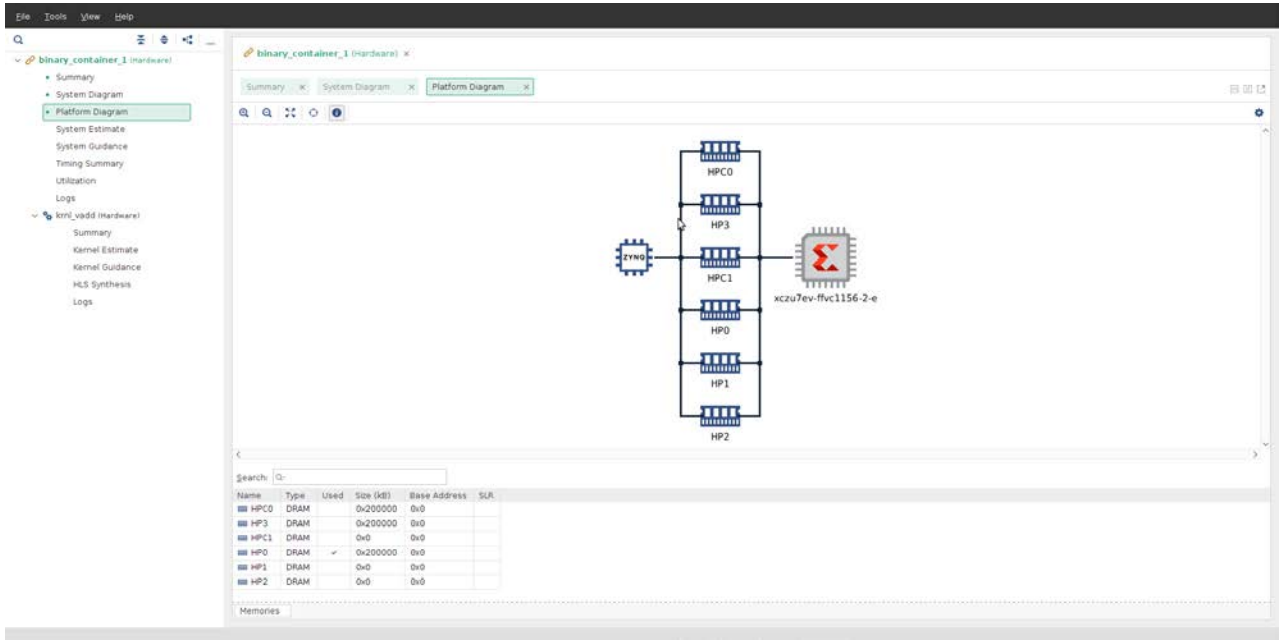
Kernel Optimization – Data Types

- Avoid native C data types (e.g., int, float, double)
- Best performance is using bit accurate types (C/C++ Kernels)
  - Arbitrary Precision Integer
  - Arbitrary Precision fixed point
- Enables smaller and faster logic implementations



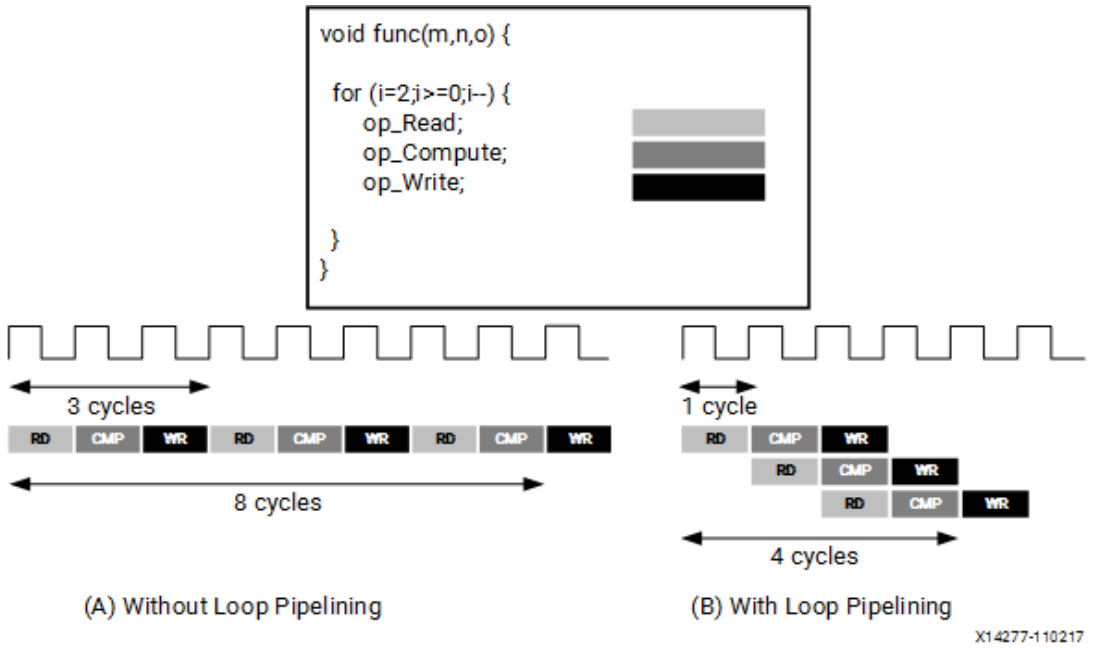
Kernel Optimization – Interfacing

- Two types of data transfer
  - Data Pointers via global memory (M_AXI)
  - Scalar direct to kernel (AXI_LITE)
- Vitis automatically selects interface type
- Max data width is 512 bits – maximum performance leverages this



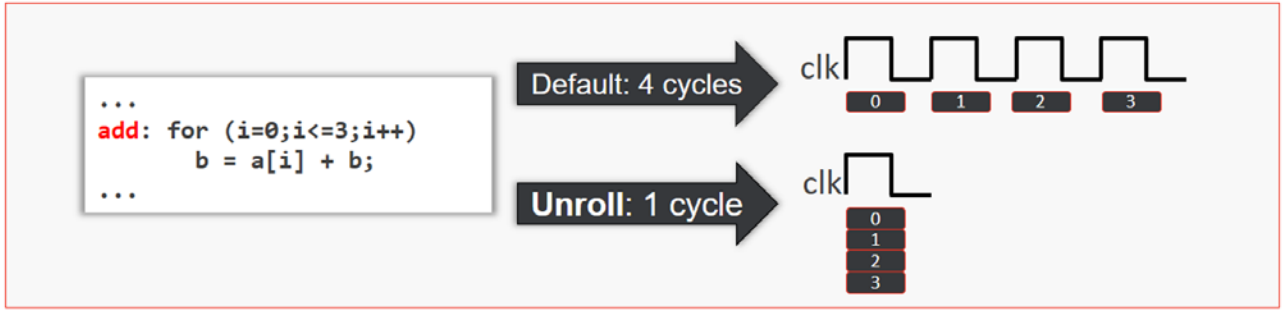
Kernel Optimization – Pipelining

- By default, every iteration of a loop only starts when the previous iteration has finished
- Pipelining the loop executes subsequent iterations in a pipelined manner



Kernel Optimization – Unrolling

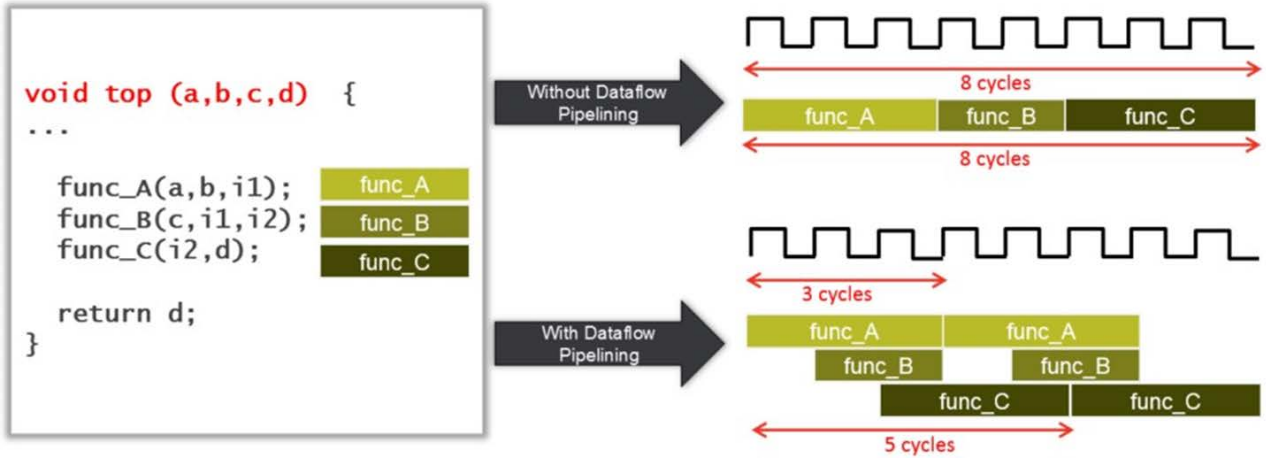
- Unrolling a loop enables the full parallelism
- Full or partial unroll
- Data dependencies in loops can impact the results of loop pipelining or unrolling



Kernel Optimization – DataFlow

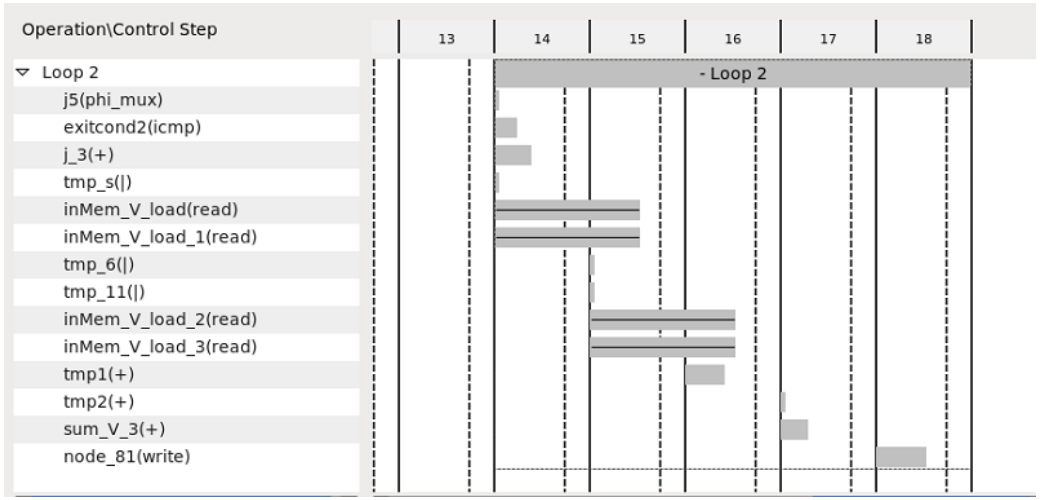
- Improve kernel performance by enabling task-level pipelining

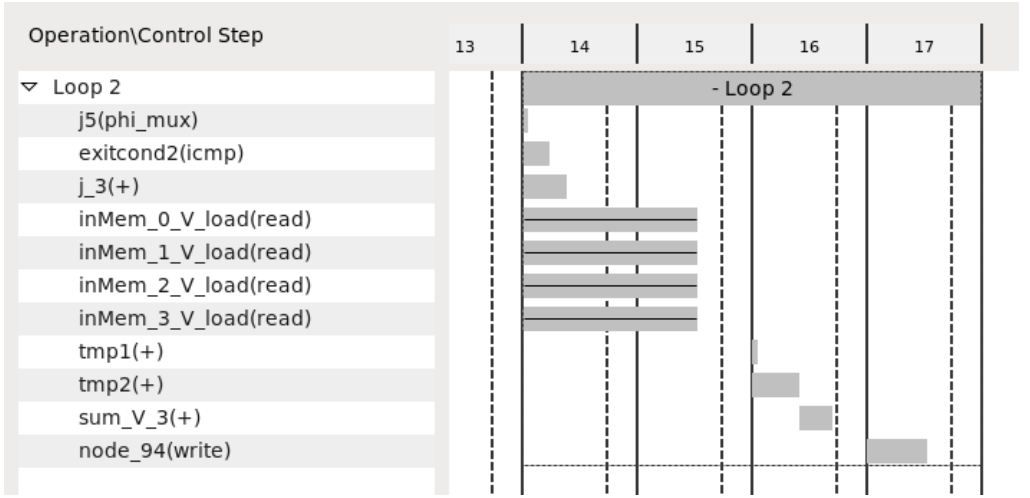
- Be careful of:
- Single producer-consumer violations.
- Bypassing tasks.
- Feedback between tasks.
- Conditional execution of tasks.
- Loops with multiple exit conditions or conditions defined within the loop



Kernel Optimization – Memory

- Limited BRAM access bandwidth, can heavily impact the overall performance
- Ability to partition and reshape arrays can increase bandwidth
- Partition – Separates into different BRAMS
- Reshape – Allows combination of words



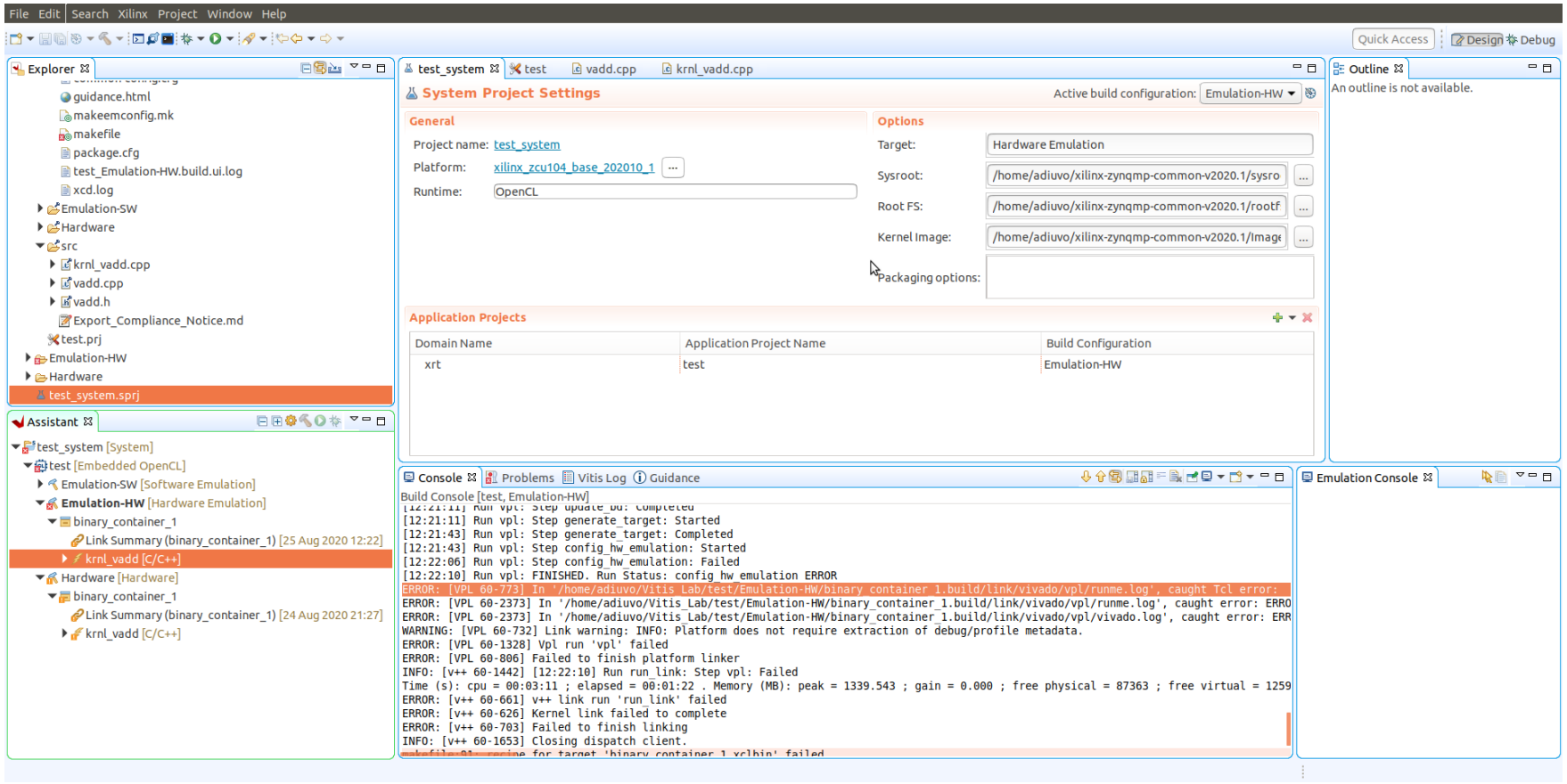


Kernel Optimization - Pragmas

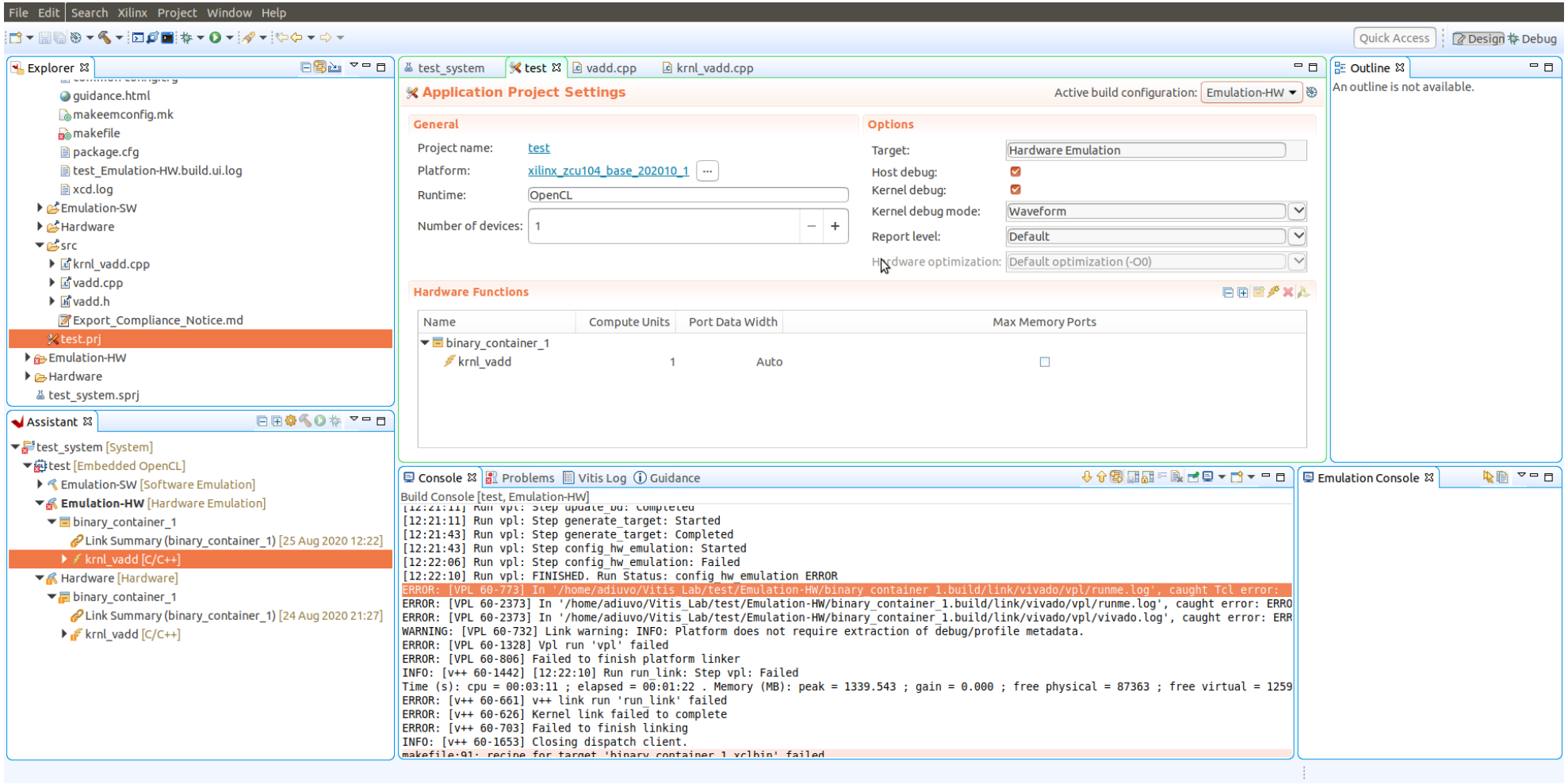
Optimization	C/C++	OpenCL
Pipeline	#pragma HLS PIPELINE	__attribute__((xcl_pipeline_loop))
Unroll	#pragma HLS UNROLL	__attribute__((opencl_unroll_hint))
DataFlow	#pragma HLS DATAFLOW	__attribute__ ((xcl_dataflow))
Memory	#pragma HLS ARRAY_PARTITION	

Further information can be found at
https://www.xilinx.com/html_docs/xilinx2020_1/vitis_doc/optimizingperformance.html#fhe1553474153030

Vitis GUI – Project Settings

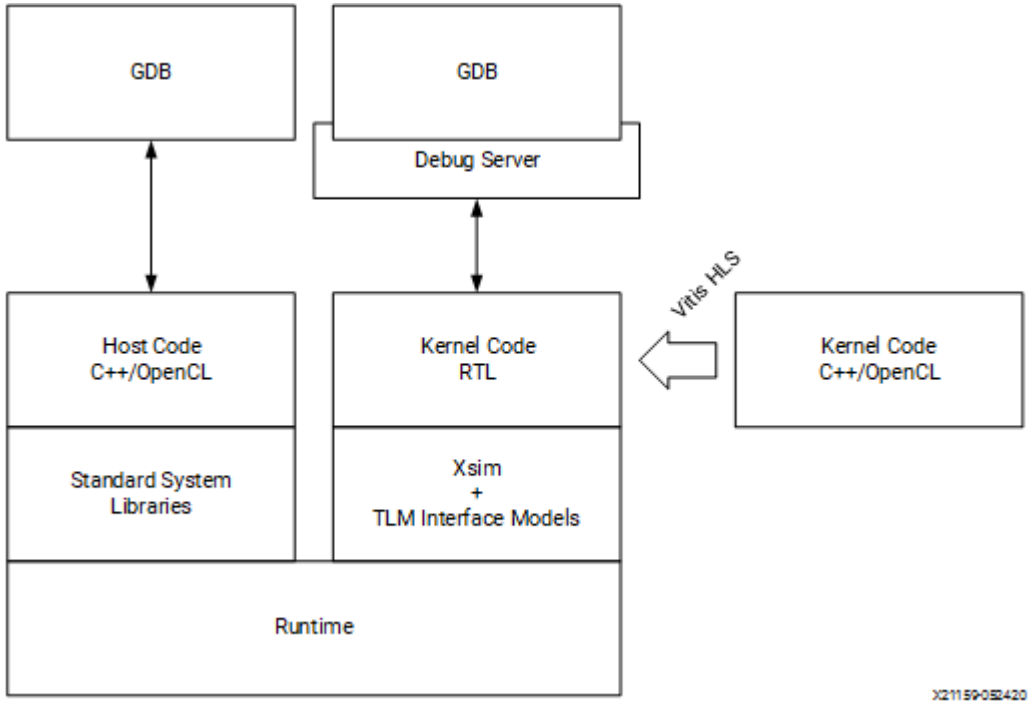


Vitis GUI – Project Setting



Vitis-Debug

- Can Debug
  - Software Emulation
  - Hardware Emulation
- Hardware flow insert ILA
- Debugging will use QEMU and logic simulator



Hands On Labs

ADIUVO

ENGINEERING AND TRAINING, LTD.

www.adiuvoengineering.com