



Petalinux and OpenCL

Adam Taylor

Session 1

Introduction to
Xilinx FPGA

Session 2

Processing in
Xilinx FPGA



Session 3

Embedded
Linux

Session 4

Accelerating
Solutions

History of PetaLinux

Linux was created as a free version of Unix for the x386 Intel CPU by Linus Torvalds in 1991

Linux was ported to ARM in 1994 on the Acorn processor which was not embedded

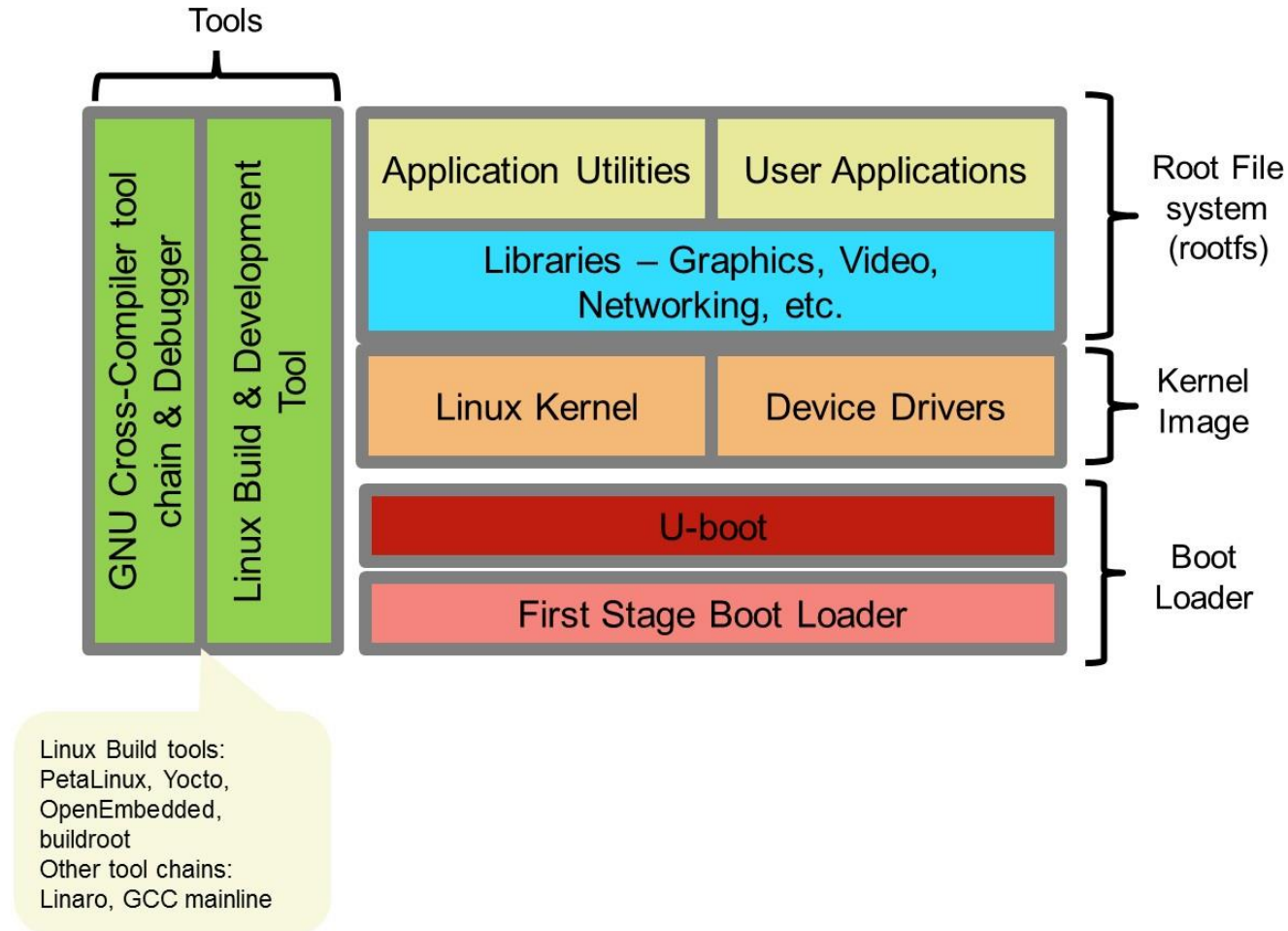
The first embedded project that Linux was ported to is unclear, it is believed to have been an x86 variant in 1997

2012 Xilinx acquires embedded Linux company: PetaLogix

PetaLinux first public release in 2013

Yocto build system utilized since version 2016.3

Linux Elements



What is PetaLinux? (a set of tools)

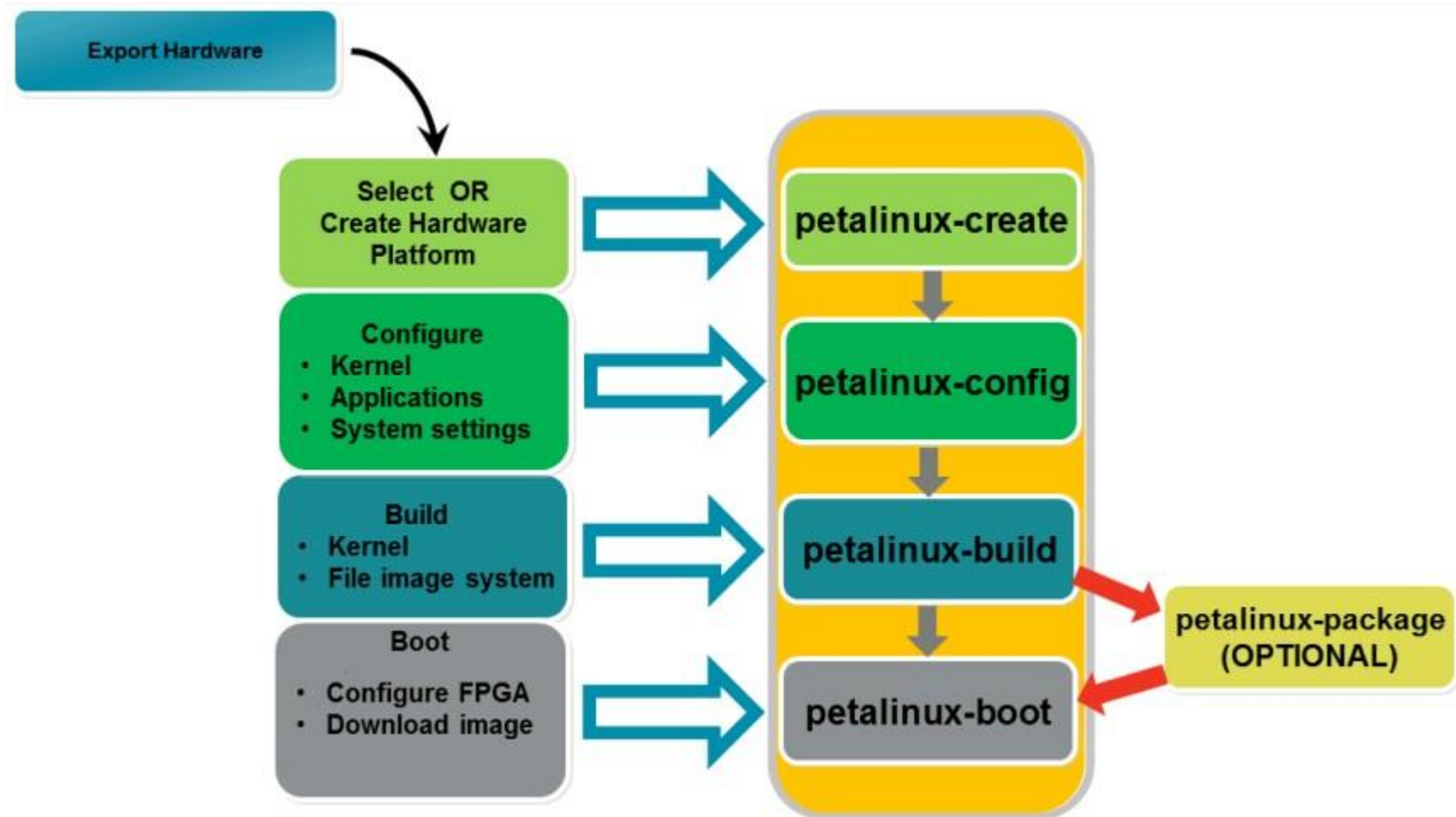
Petalinux Tools enable, build and customization of

- » First Stage Boot Loader
- » U-Boot (Second Stage Boot loader)
- » Linux Kernel and Device Tree
- » Root File System – User Applications, Libraries and Kernel Modules

PetaLinux is a build tool, all component can be built without PetaLinux if sufficiently experienced

Components from other build processes can be integrated

PetaLinux Tools Flow



Petalinux Multitasking Capabilities

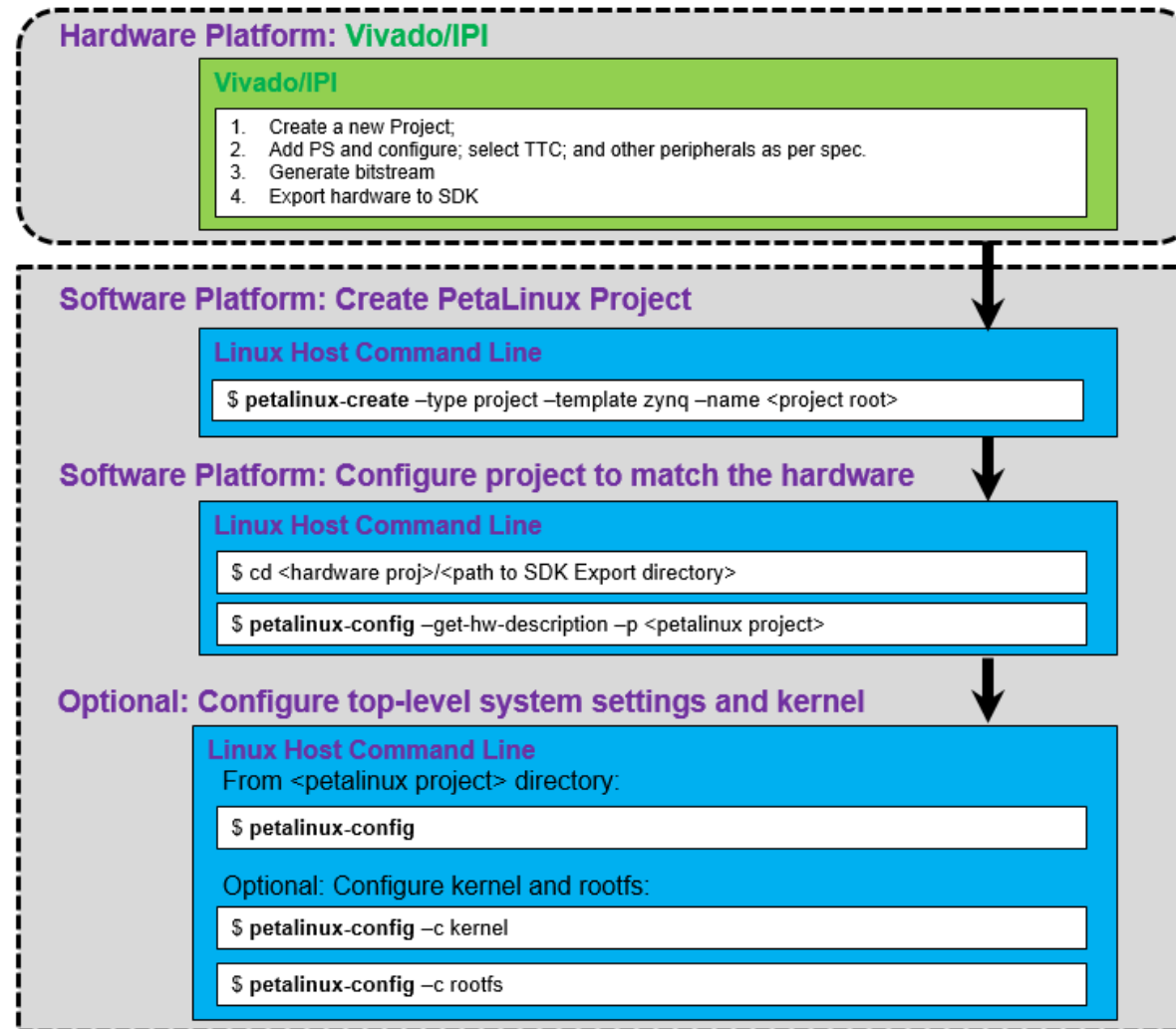
Embedded Linux (like Linux) is a multi-tasking multi-user operating system

- » On the Arty Z7 there are two cores, on Gensys ZU there are four APU cores

Petalinux multitasking means:

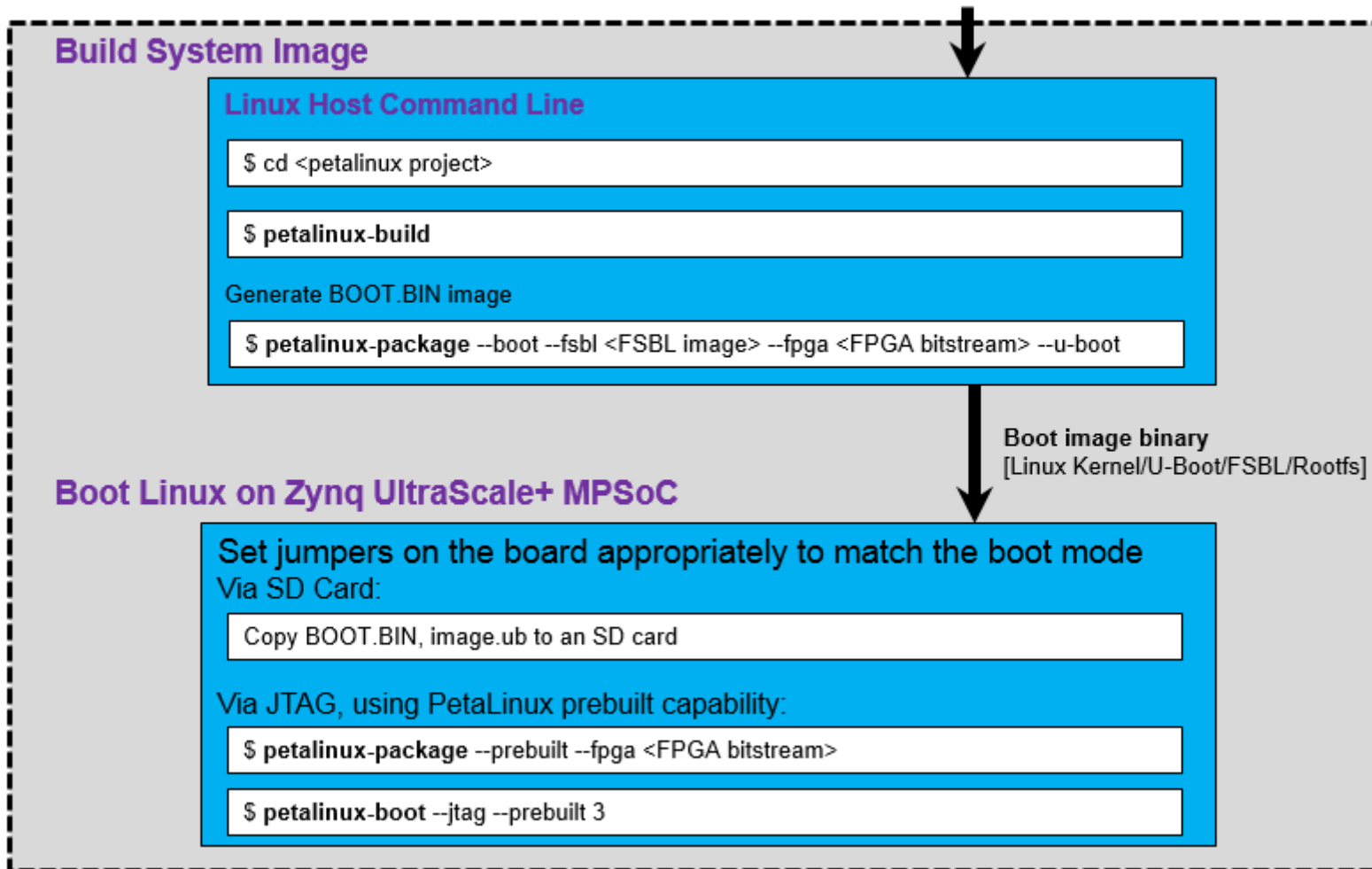
- » Fast (Two, 32-bit or Four 64-bit cores) but not a real-time OS (RTOS)
- » If your application requires real-time, use the MicroBlaze or R5 cores. These core cannot run PetaLinux but can run FreeRTOS etc.
- » Communication between the A9, A53 (PetaLinux) and MicroBlaze, R5 (FreeRTOS etc.) can use OpenAMP
- » Even though PetaLinux is not a pure RTOS it can still be used for many applications (with careful system architecture, design and expectations)

Creating a PetaLinux Project



Select BSP or import from Vivado

Configuring a PetaLinux Project



Configure

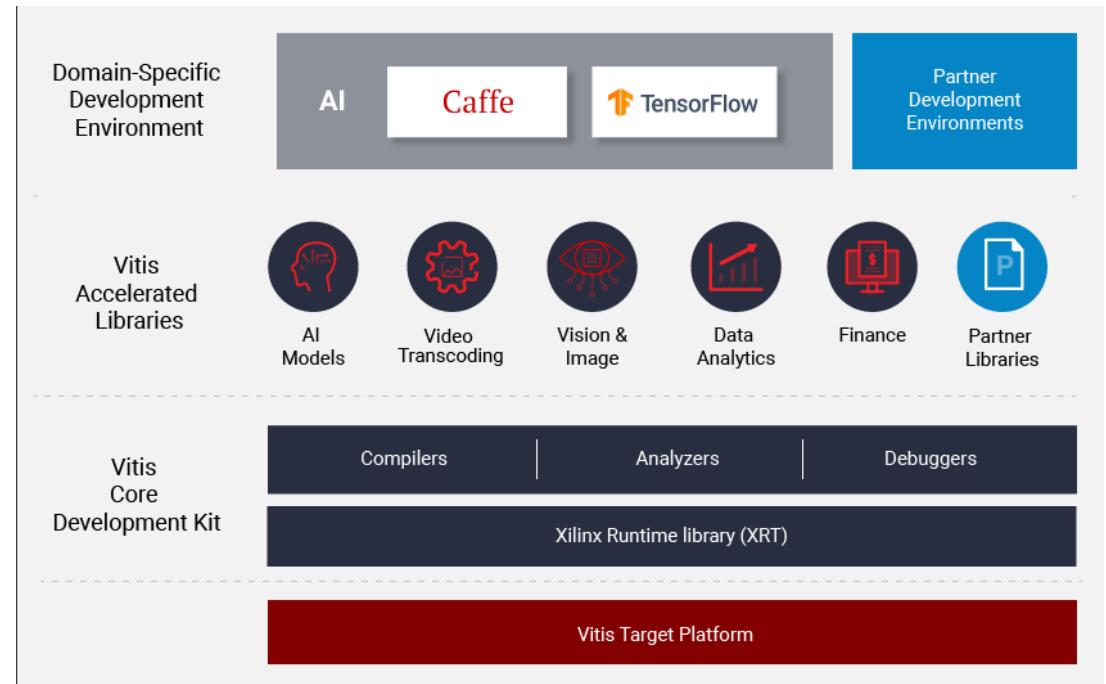
- Kernel
- Applications
- System settings



Vitis

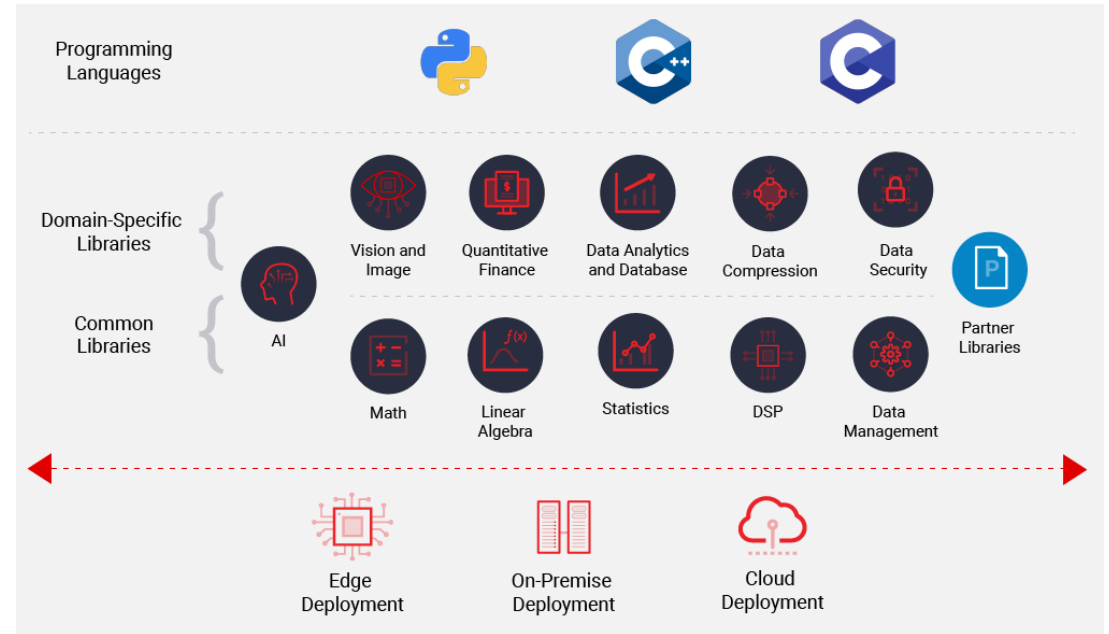
What is Vitis

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



Vitis Accelerated Libraries

- Several 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 & Command line tools for compilation, debug and analysis of C, C++ and OpenCL designs.

Can use preferred GUI or integrated GUI

Supports embedded and accelerated flows



Vitis Target Platforms

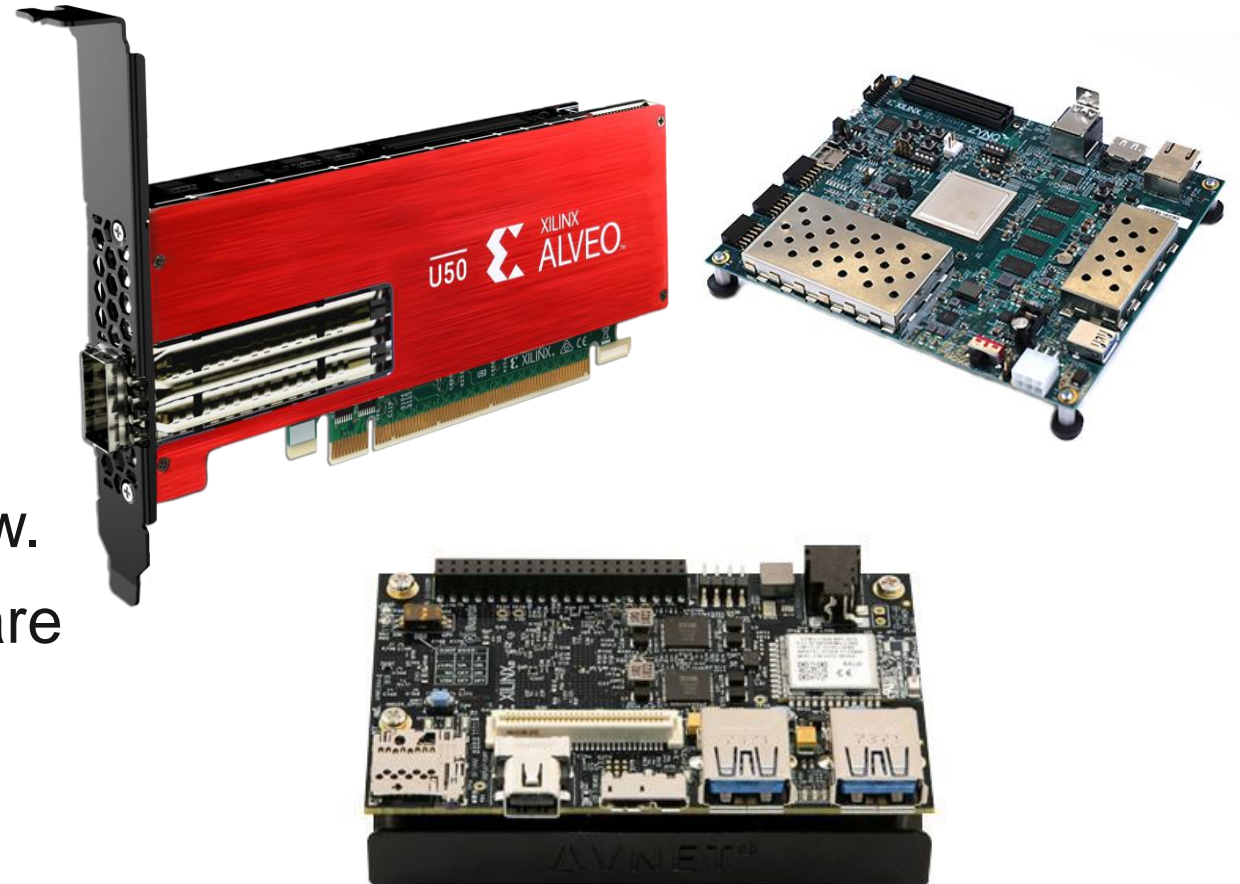
Embedded

- » 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



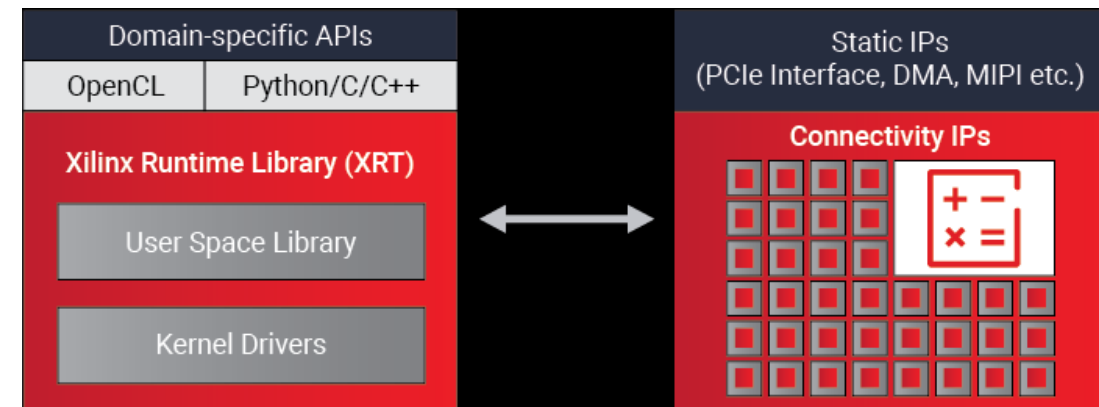
Xilinx Runtime library

Xilinx Runtime library (XRT) enables communication between the host and accelerator

Cloud based – Host x86

Embedded – Arm 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

```
adiuvo@Adiuvo: ~  
File Edit View Search Terminal Help  
INFO: == Starting PCIE link check:  
LINK ACTIVE, ATTENTION  
Ensure Card is plugged in to Gen3x16, instead of Gen3x4  
Lower performance may be experienced  
WARN: == PCIE link check PASSED with warning  
INFO: == Starting SC firmware version check:  
SC FIRMWARE MISMATCH, ATTENTION  
SC firmware running on board: 1.8. Expected SC firmware from installed Shell: 4.2.0  
Please use "xbmgmt flash --scan" to check installed Shell.  
WARN: == SC firmware version check PASSED with warning  
INFO: == Starting verify kernel test:  
INFO: == verify kernel test PASSED  
INFO: == Starting DMA test:  
Host -> PCIE -> FPGA write bandwidth = 3335.9 MB/s  
Host <- PCIE <- FPGA read bandwidth = 3238.05 MB/s  
INFO: == DMA test PASSED  
INFO: == Starting device memory bandwidth test:  
.....  
Maximum throughput: 52428 MB/s  
INFO: == device memory bandwidth test PASSED  
INFO: == Starting PCIE peer-to-peer test:  
P2P BAR is not enabled. Skipping validation  
INFO: == PCIE peer-to-peer test SKIPPED  
INFO: == Starting memory-to-memory DMA test:  
bank0 -> bank1 M2M bandwidth: 12100 MB/s  
bank0 -> bank2 M2M bandwidth: 12128.7 MB/s  
bank0 -> bank3 M2M bandwidth: 12114.9 MB/s  
bank1 -> bank2 M2M bandwidth: 12116 MB/s  
bank1 -> bank3 M2M bandwidth: 12118.9 MB/s  
bank2 -> bank3 M2M bandwidth: 12116 MB/s  
INFO: == memory-to-memory DMA test PASSED  
INFO: Card[0] validated with warnings.  
  
INFO: All cards validated successfully but with warnings.  
adiuvo@Adiuvo:~$
```


Element of Vitis

All projects required 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

Platform
Choose a platform for your project. You can also create an application from XSA through the 'Create a new platform from hardware (XSA)' tab.

Select a platform from repository | Create a new platform from hardware (XSA)

Find:

Name	Board	Flow	Vendor	Path
xilinx_zcu104_base_202010_1	zcu104	Embedded Accel	xilinx	/opt/xilinx/platforms/xilinx_zcu104_base_202010_1/xilinx_zcu104_base_202010_1.xpfm
xilinx_u200_xdma_201830_2	u200	DataCenter Accel	xilinx	/opt/xilinx/platforms/xilinx_u200_xdma_201830_2/xilinx_u200_xdma_201830_2.xpfm

Platform Info

General Info

Name:

Part:

Family:

Description:

A basic static platform targeting the ZCU104 evaluation board, which includes 2GB DDR4, GEM, USB, SDIO interface and UART of the Processing System. It reserves most of the PL resources for user to add acceleration

Acceleration Resources

Clock Frequencies	
Clock	Frequency (MHz)
CPU	1200.000000
PL 0	150.000000
PL 1	300.000000
PL 2	75.000000
PL 3	100.000000
PL 4	200.000000
PL 5	400.000000
PL 6	600.000000

Domain Details

Domains	Domain name	Details
xrt		CPU: cortex-a53 OS: linux

Vitis Output

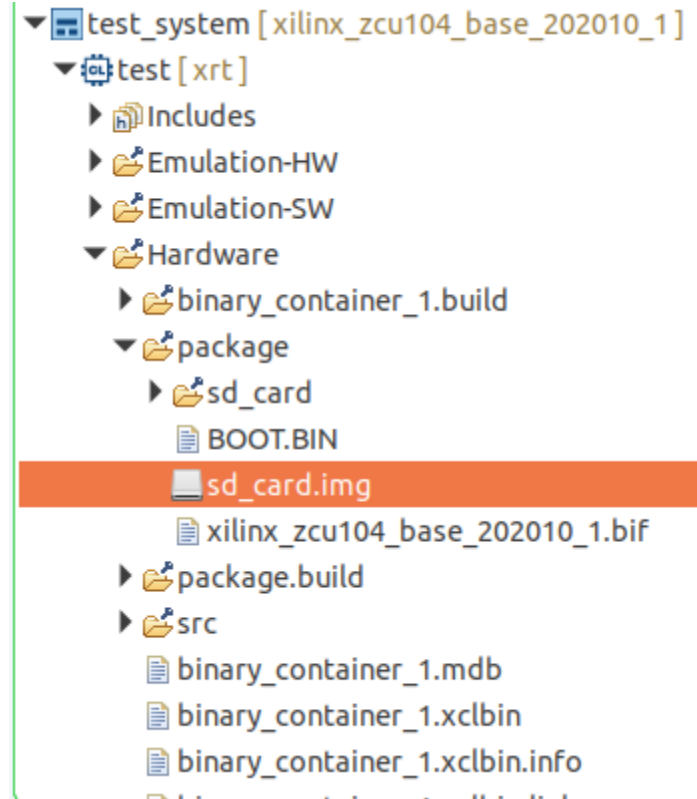
Compiled binary (host) and XCLbin (accelerator)

Embedded System Output

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

Cloud output

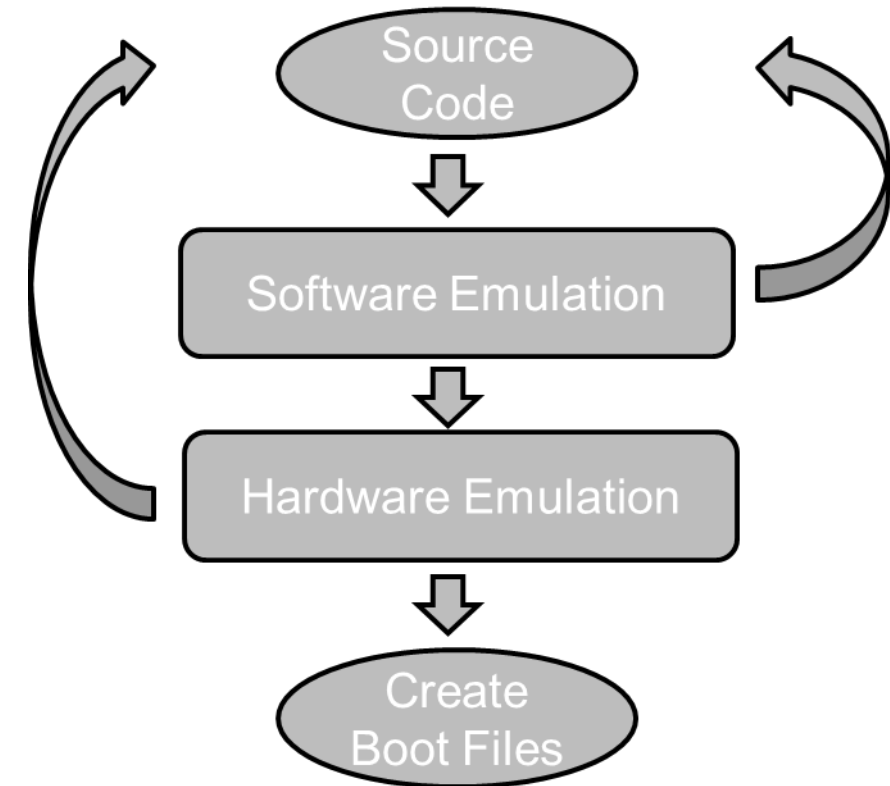
- » Binary and XCLBin



Vitis Development Flow

Software Emulation – Syntax errors & algorithm verification

Hardware Emulation – Optimize Performance, Interfacing & Resources





OPENCL

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.
 - Can run on cell phones, laptops, super computers
- Important: No performance portability

Wide market adoption

- Support implemented by
 - Apple, AMD, Xilinx, Intel, ARM, Nvidia, Qualcomm, etc.
- Many companies developing applications
 - Image, video, audio processing, scientific calculations, medical imaging, and more

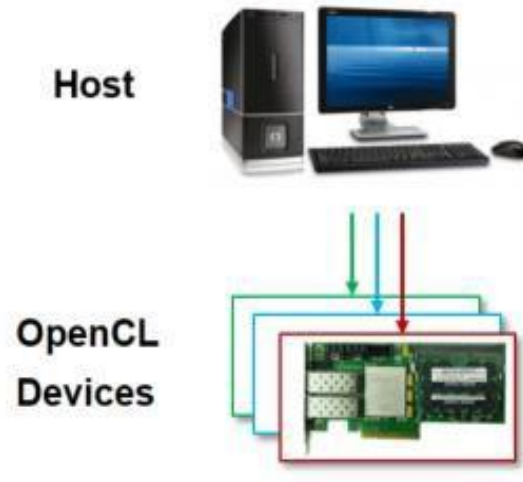


OpenCL

Khronos Group

www.khronos.org

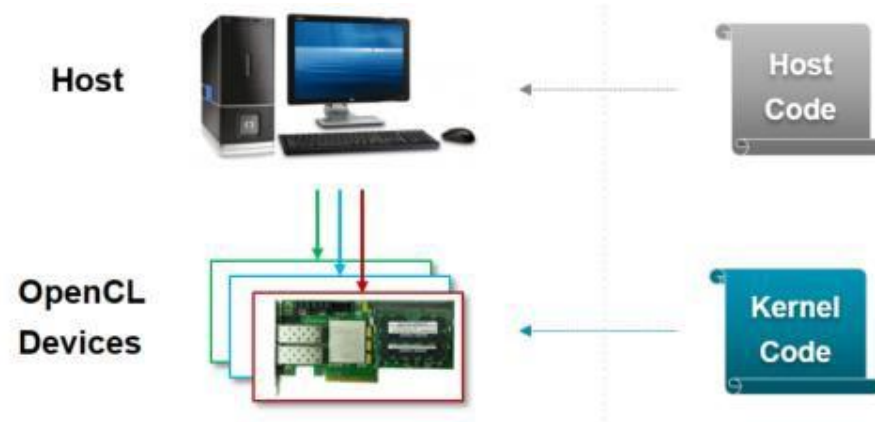
OpenCL Framework



Platform model

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

OpenCL Framework



Platform model

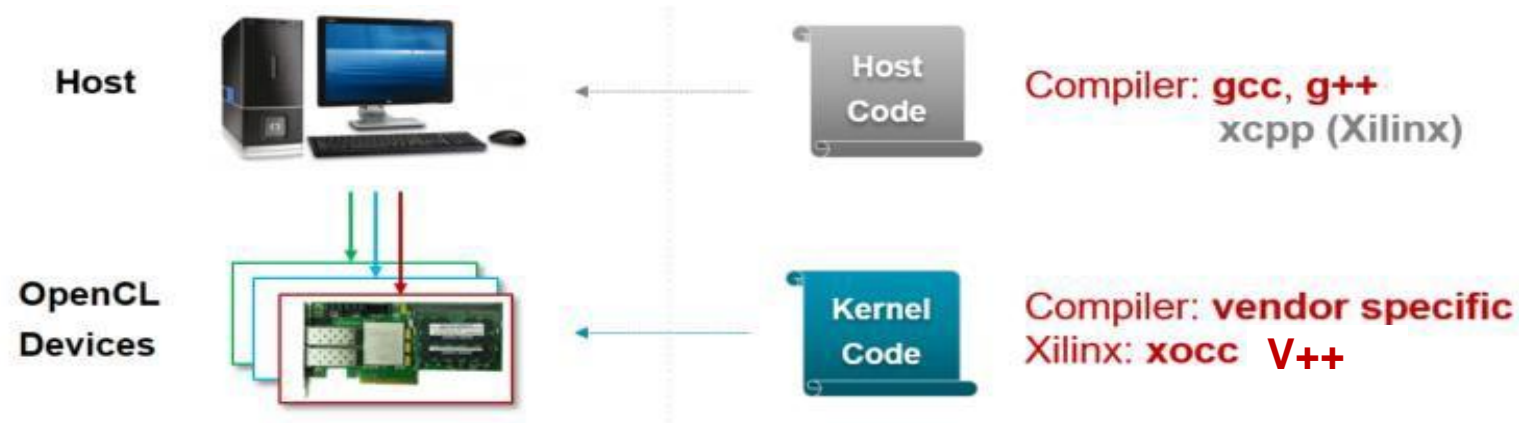
- Defines representation of ANY platform
- Contains
 - Single host
 - One or more OpenCL devices (compute device)

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

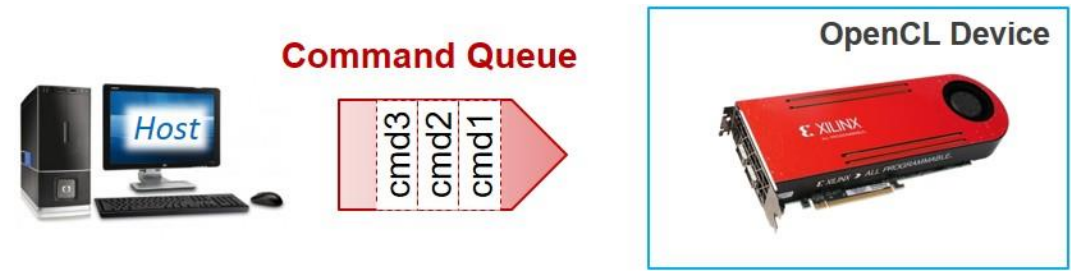
OpenCL application: Two parts

- 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

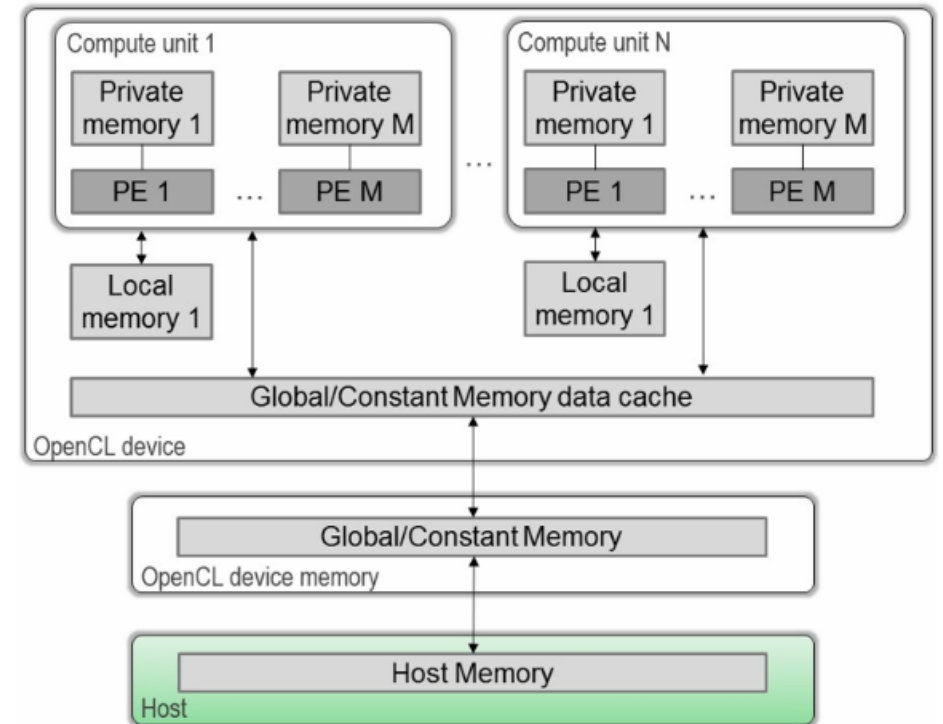
- Region of global memory
- Work items – read access only

Local memory

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

Private memory

- Accessible by a work-item





High Level Synthesis

Amdahl's law

- S: overall performance improvement
 - Alpha: percentage of the algorithm that can be sped up with hardware acceleration
 - 1-alpha: percentage of the algorithm that cannot be improved.
 - p: is the speedup due to acceleration (%).
-
- Set Alpha to 0.1 and select speed up - even with large acceleration P defined, speed up is close to 1
 - Set Alpha to 0.5 and select same speed up – close to factor of two improvement.

$$S = \frac{1}{(1 - \alpha) + \alpha/p}$$

Getting the best from HLS

Functions we accelerate into logic often need optimising

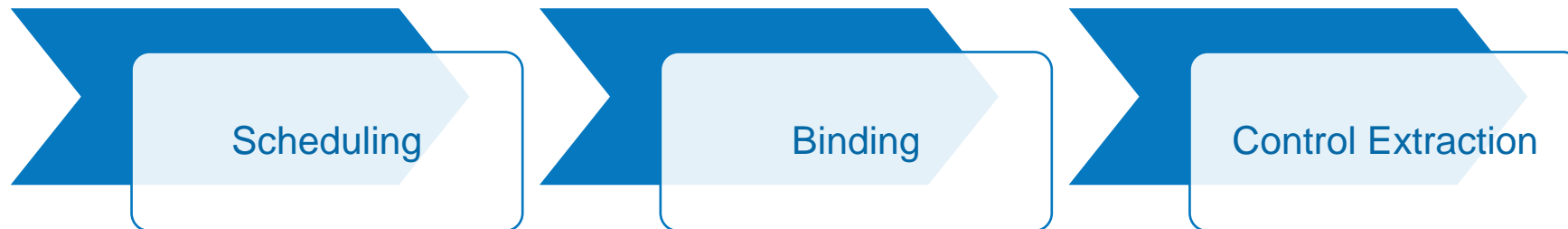
- » Loops need unrolling
- » Memory Structures need optimising
- » Resource allocation

HLS controlled via `#pragma` in the accelerated function

Who has Used HLS before ?

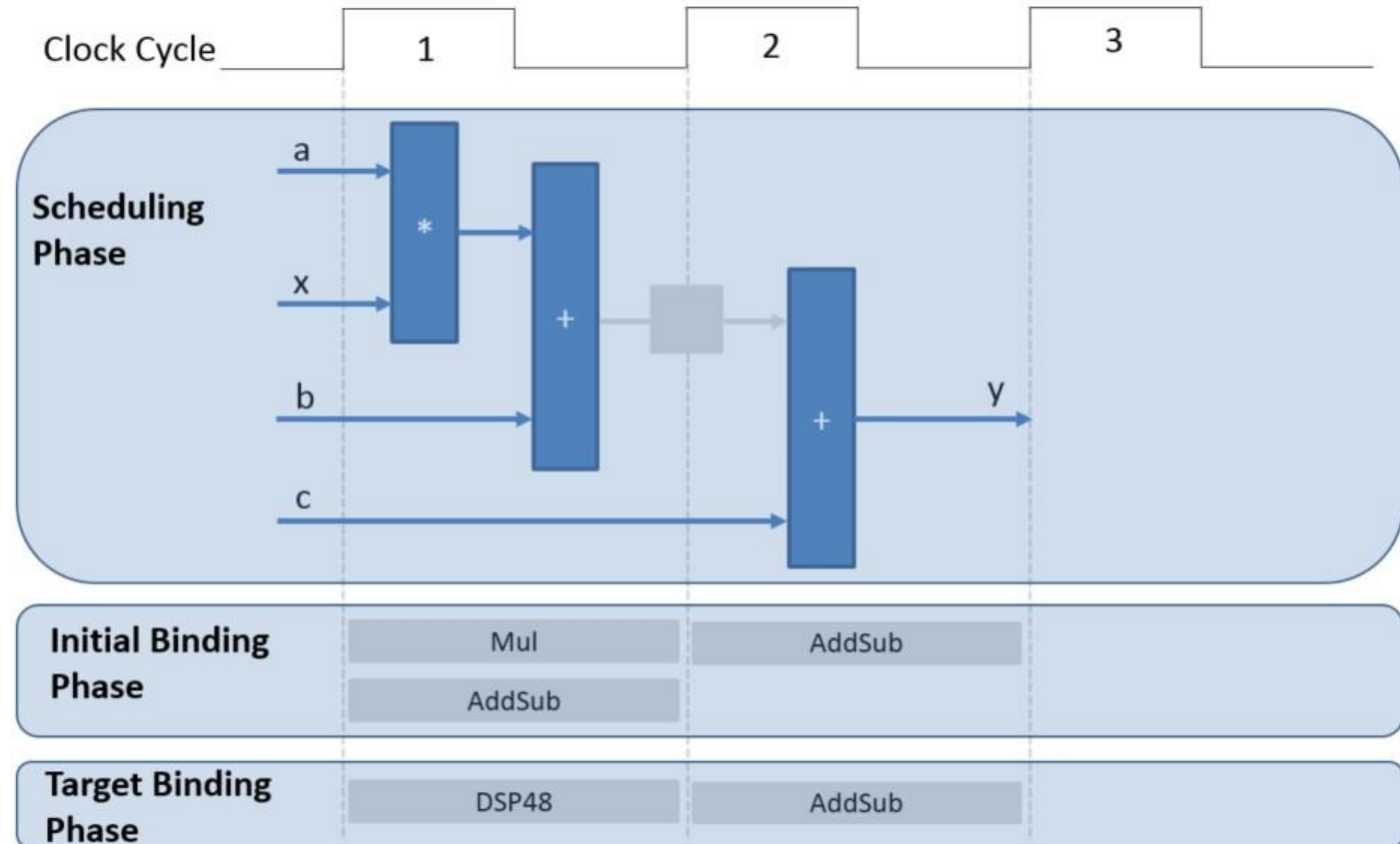
HLS came of age over the last 5 years

HLS is excellent for data flow acceleration e.g. signal processing, image processing, Artificial Intelligence and Machine Learning



Example of HLS

```
int foo(char x, char a, char b, char c) {
  char y;
  y = x*a+b+c;
  return y
}
```



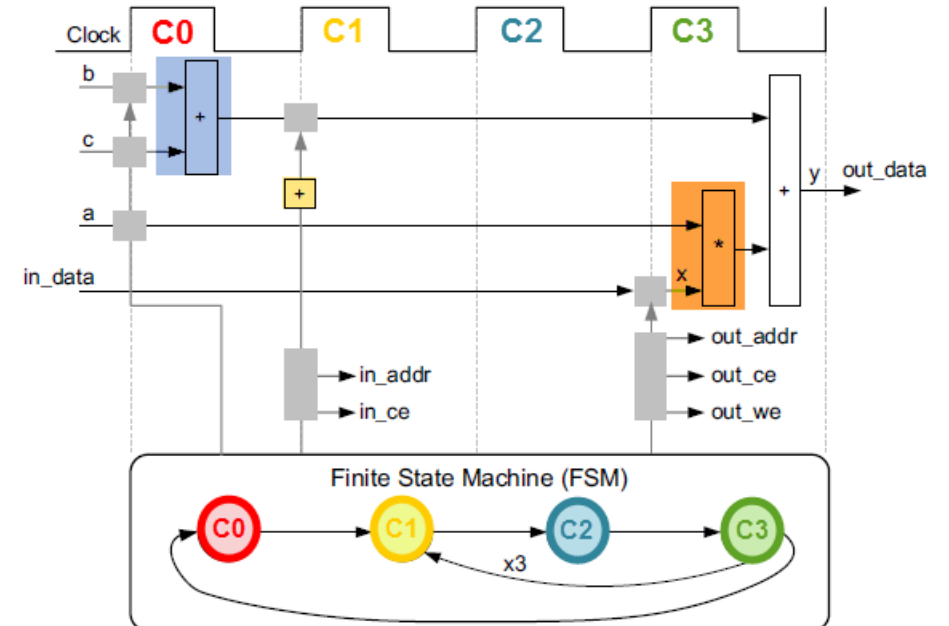
Looking a little deeper

```

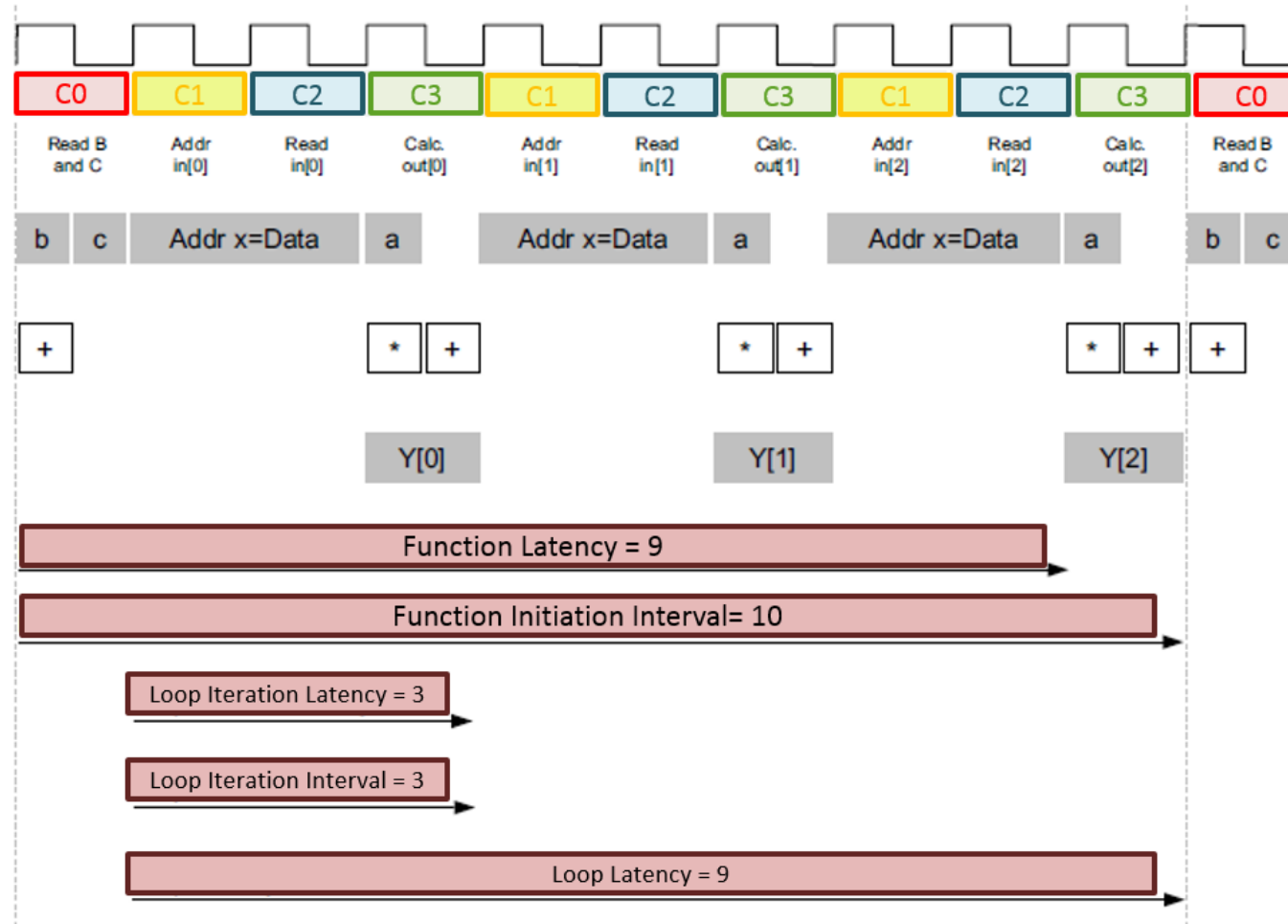
void foo(int in[3], char a, char b, char c, int out[3]) {
  int x,y;
  for(int i = 0; i < 3; i++) {
    x = in[i];
    y = a*x + b + c;
    out[i] = y;
  }
}

```

- C0** - adder b + c
- C1** - generates the address for *in* and *adder* to increment to count how many times design iterates (C1, C2 and C3)
- C2** - Block RAM returns the data for *in* and stores it as variable *x*
- C3** - Reads the data from port *a* with other values to perform the calculation and generates the first *y* output



Terminology



C to RTL

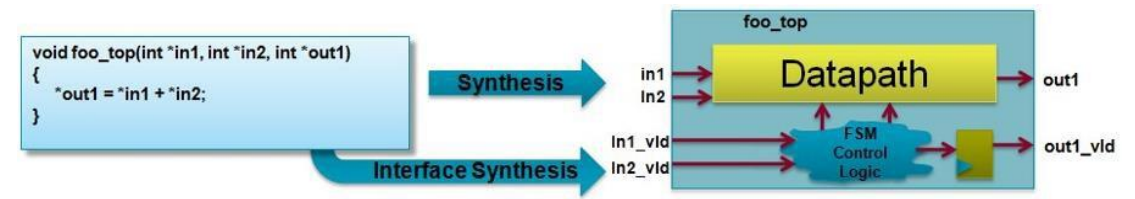
HLS synthesizes the C code in different ways

Top-level function arguments synthesize into RTL I/O ports

Loops in the C functions are kept *rolled* by default

Arrays in the C code synthesize into block RAM in the final design

C functions synthesize into blocks in the RTL hierarchy



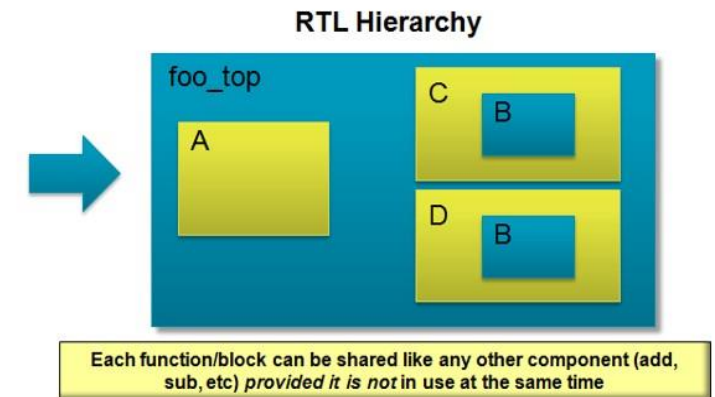
Source Code

```

void A() { ..body A.. }
void B() { ..body B.. }
void C() {
    B();
}
void D() {
    B();
}

void foo_top() {
    A(...);
    C(...);
    D(...);
}
    
```

my_code.c



Interfacing

	Argument Type	Scalar		Array			Pointer or Reference			HLS:: Stream
		Input	Return	I	I/O	O	I	I/O	O	I and O
Block-Level Protocol	ap_ctrl_none									
	ap_ctrl_hs		D							
	ap_ctrl_chain									
AXI Interface Protocol	axis									
	s_axilite									
	m_axi									
No I/O Protocol	ap_none	D					D			
	ap_stable									
Wire Handshake Protocol	ap_ack									
	ap_vld								D	
	ap_ovld							D		
	ap_hs									
Memory Interface Protocol: RAM : FIFO	ap_memory			D	D	D				
	bram									
	ap_fifo									D
Bus Protocol	ap_bus									

Supported
 D = Default Interface
 Not Supported



Optimization

Optimization

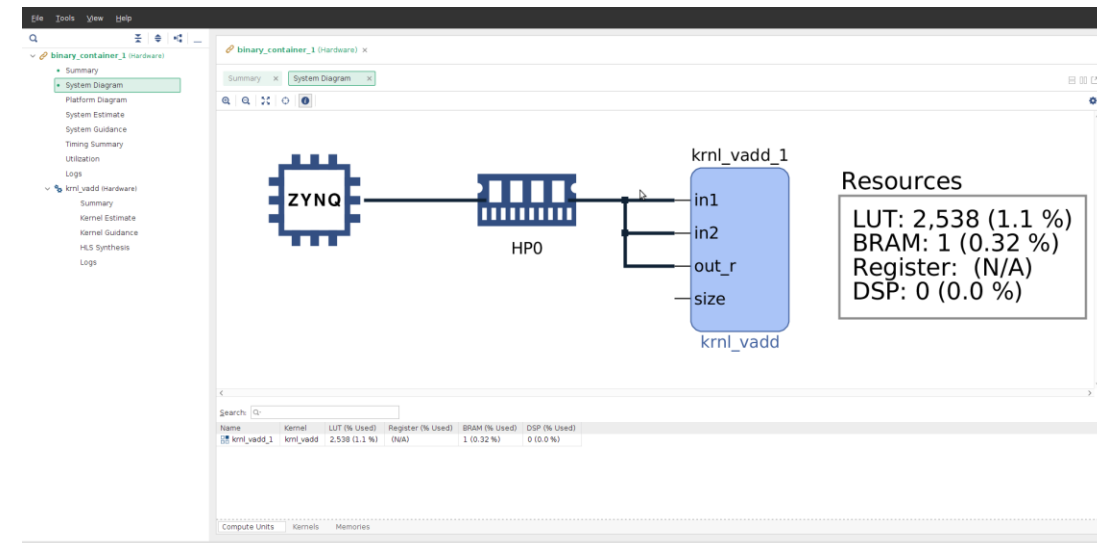
Optimization Possible at both Host and Kernel

Enables most responsive solution

Host optimization

Kernel optimization possible in OpenCL and C/C++

» Optimization Syntax differs

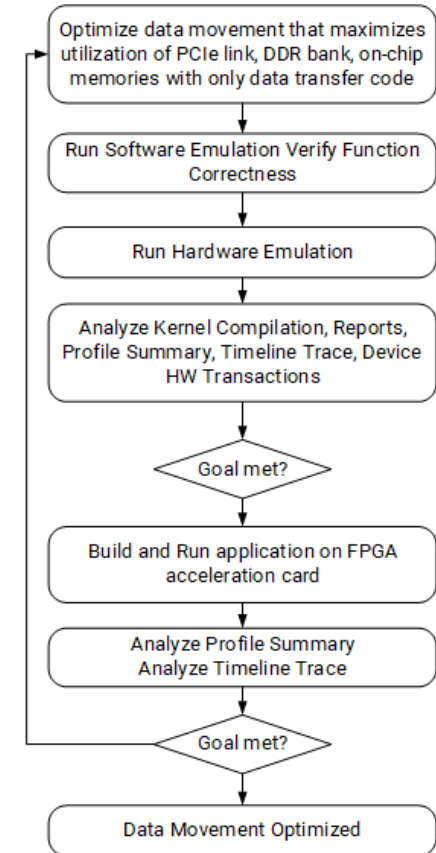


Host Optimization

Optimize the data movement in the application before optimizing computation

Compute Unit Scheduling

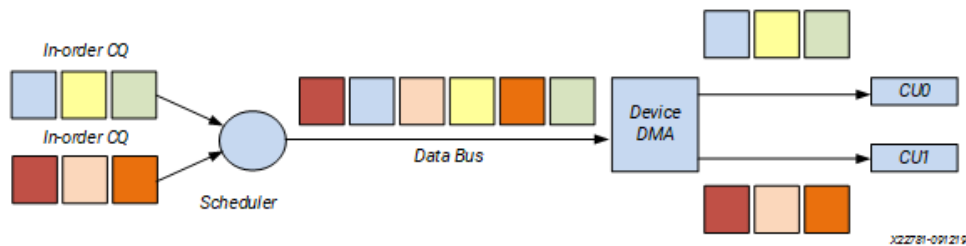
- » Multiple In-Order Command Queues
- » Single Out-of-Order Command Queue



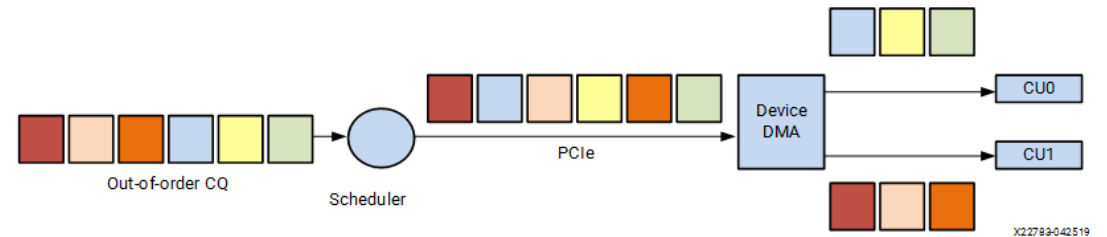
X122319-082719

Host Optimization

Multiple In-Order Command Queues



- Single Out-of-Order Command Queue



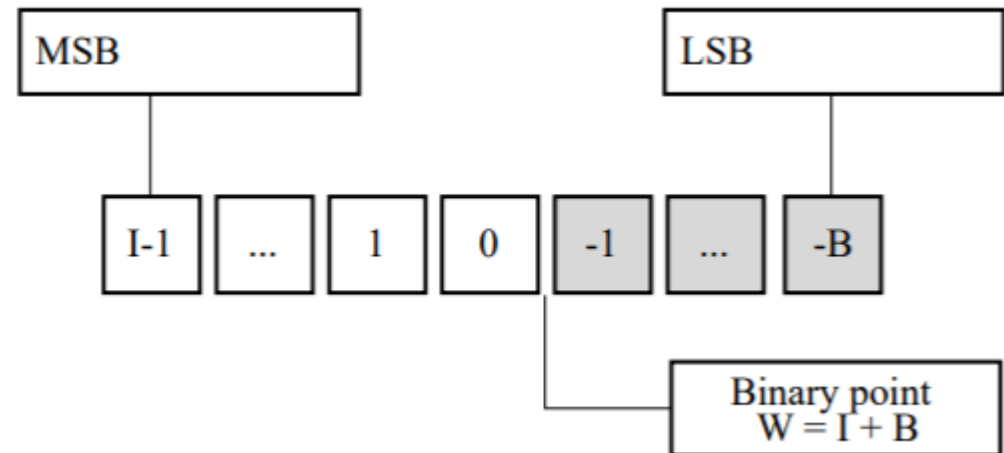
Kernel Optimization – Data Types

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 & faster logic implementations



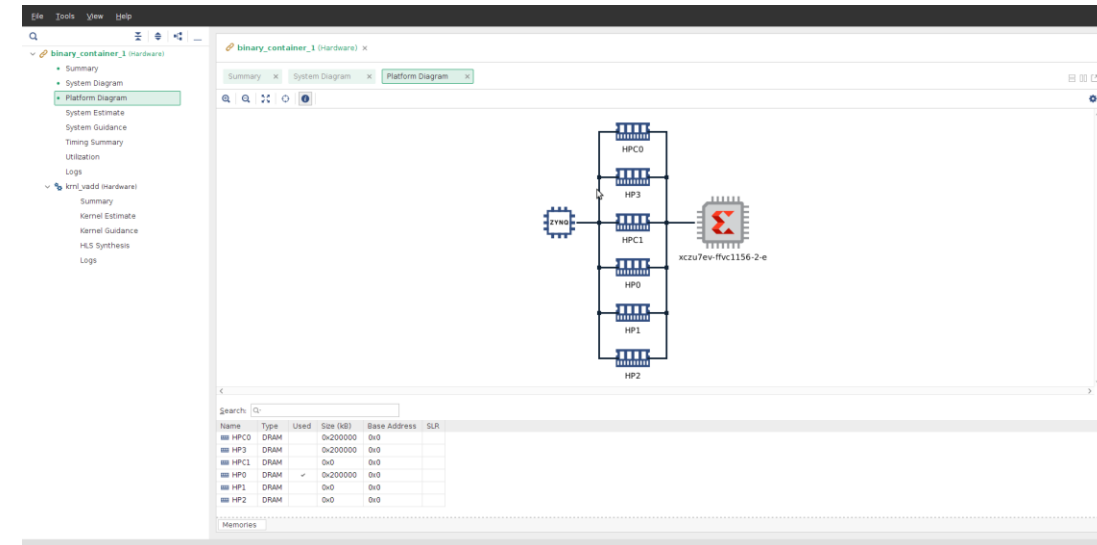
Kernel Optimization – Interfacing

Two types of data transfer

- » Data Pointers via global memory (M_AXI)
- » Scalar direct to kernel (AXI_LITE)

Vitis automatically selects interface type

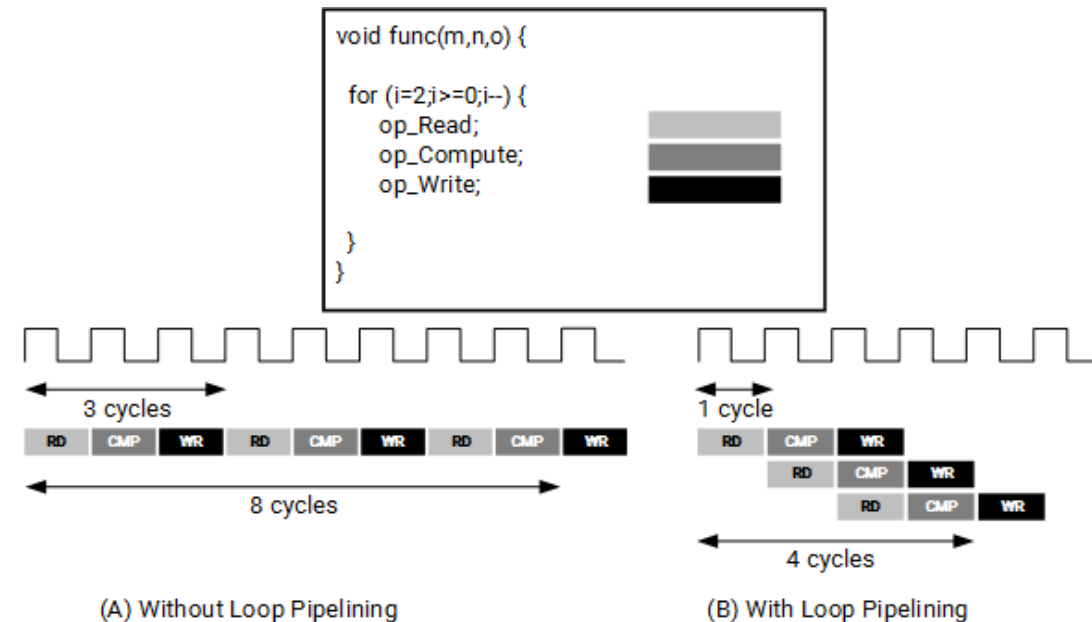
Max data width is 512 bits – maximum performance leverages this



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



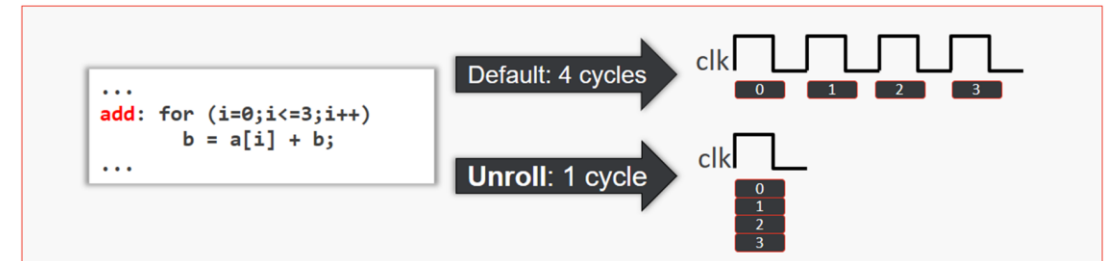
X14277-110217

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



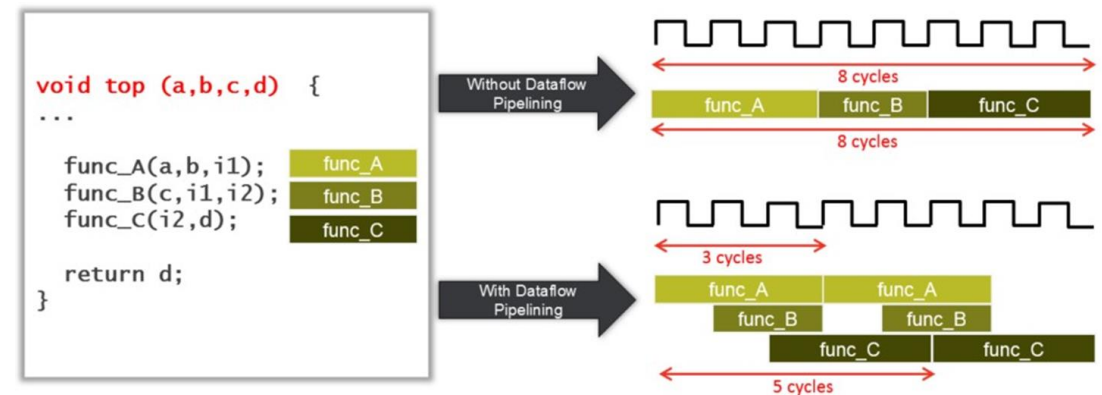
Kernel Optimization – DataFlow

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



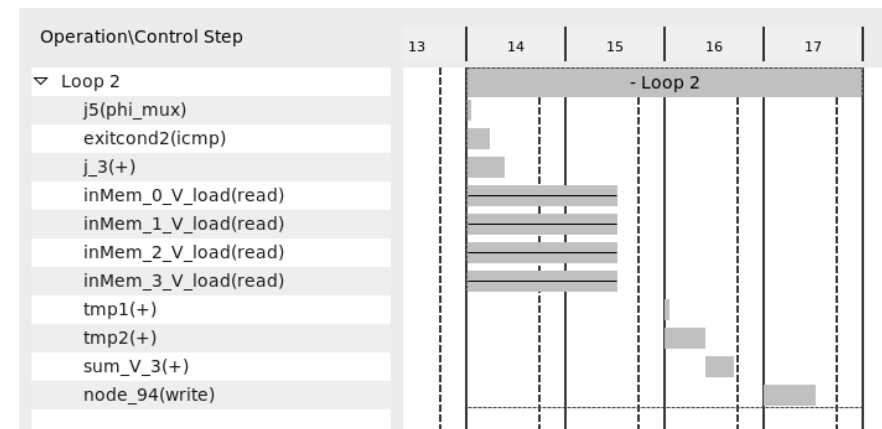
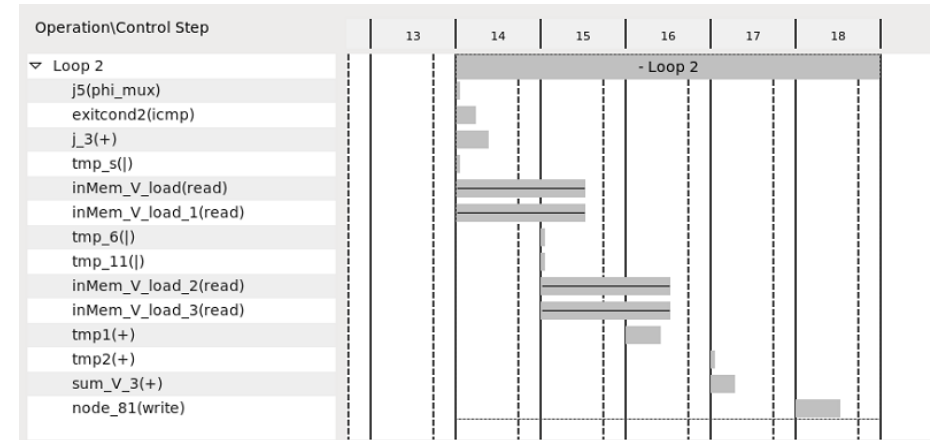
Kernel Optimization – Memory

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



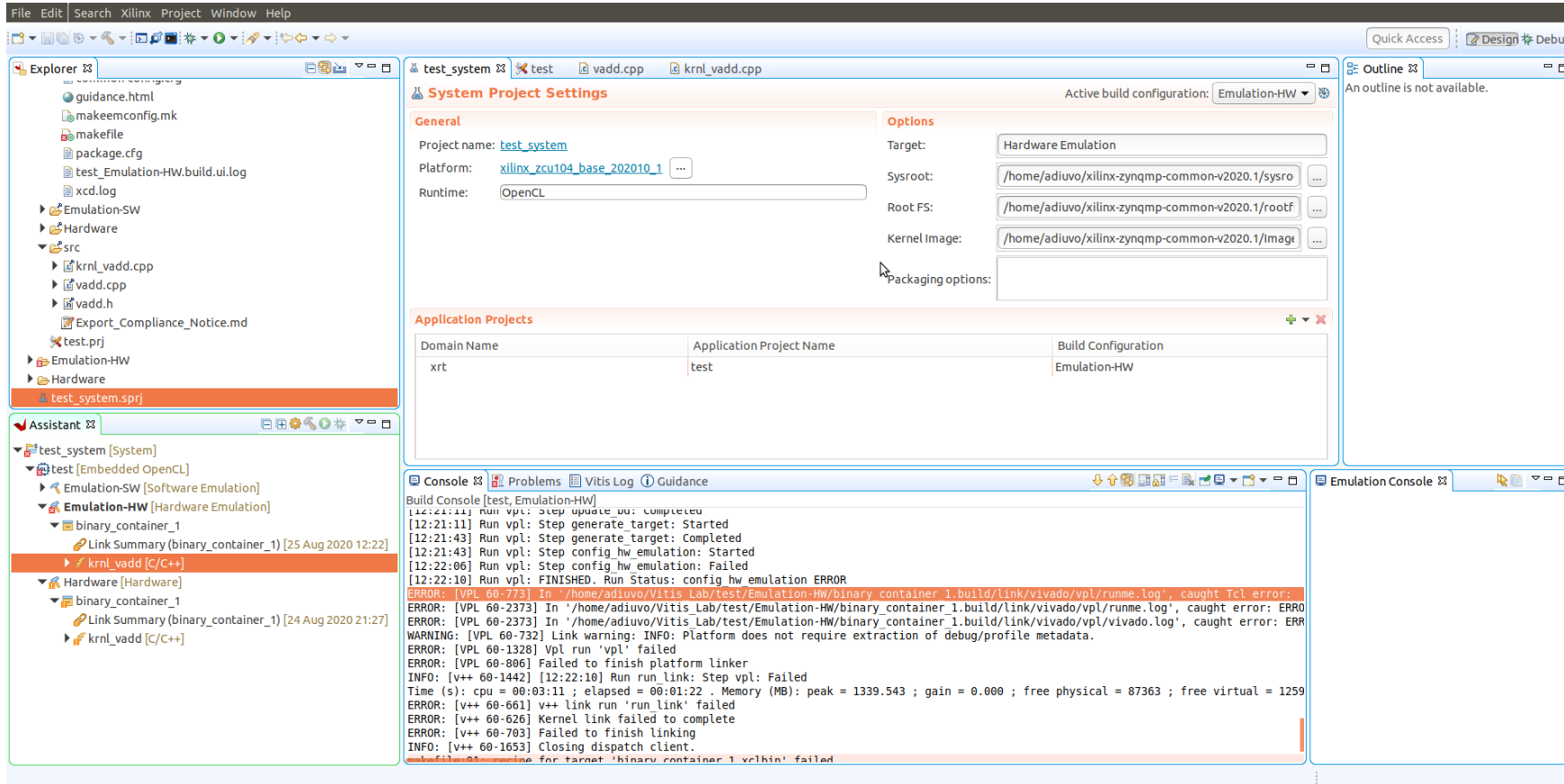
Kernel Optimization - Pragmas

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

Further information can be found at

https://www.xilinx.com/html_docs/xilinx2020_1/vitis_doc/optimizingperformance.html#the1553474153030

Vitis GUI – Project Settings



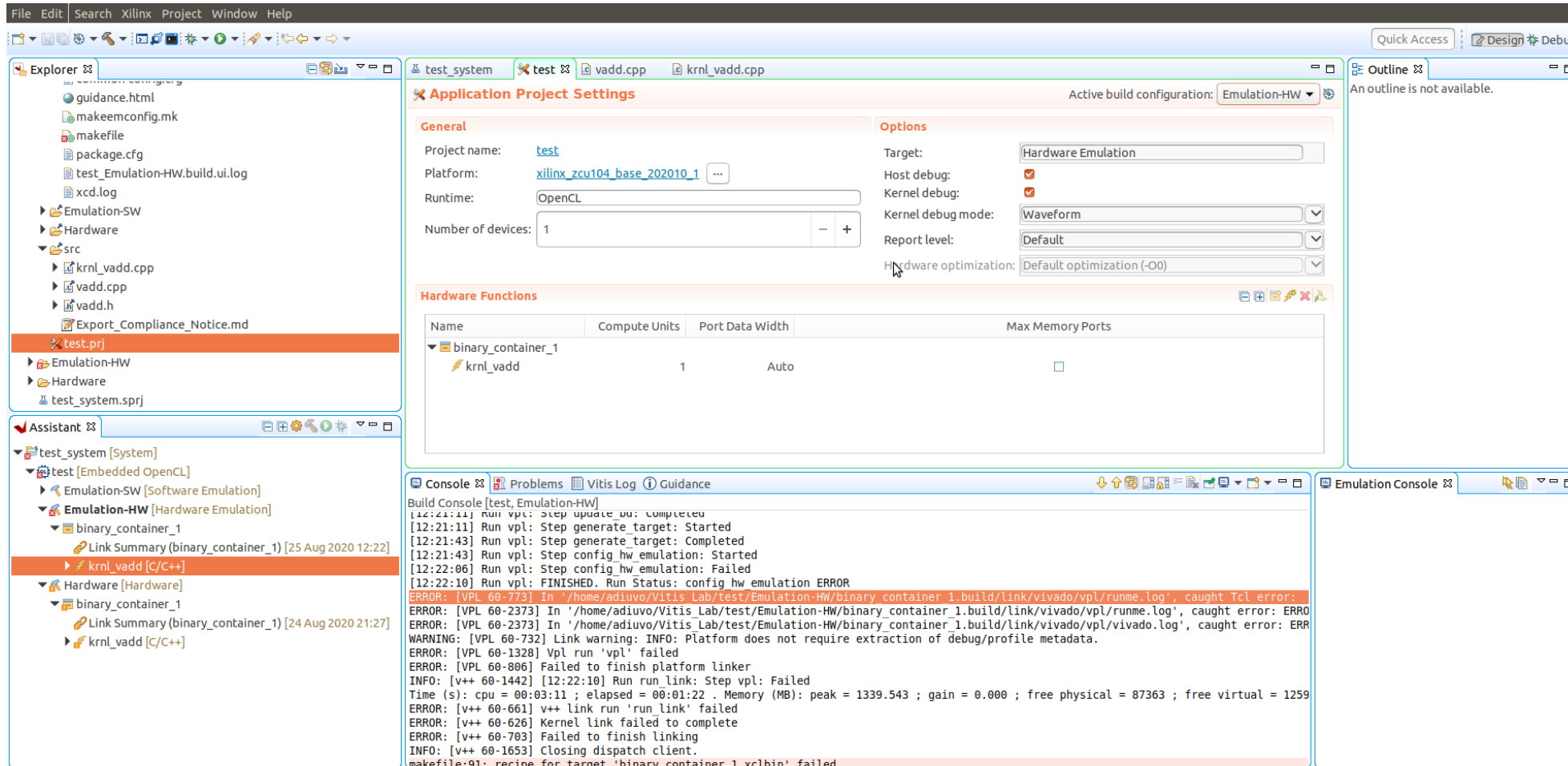
The screenshot displays the Vitis GUI interface for configuring a project named 'test_system'. The 'System Project Settings' window is open, showing the following configuration:

- General:**
 - Project name: test_system
 - Platform: xilinx_zcu104_base_20201_1
 - Runtime: OpenCL
- Options:**
 - Target: Hardware Emulation
 - Sysroot: /home/adiuvo/xilinx-zynqmp-common-v2020.1/sysro
 - Root FS: /home/adiuvo/xilinx-zynqmp-common-v2020.1/rootf
 - Kernel Image: /home/adiuvo/xilinx-zynqmp-common-v2020.1/imagt
 - Packaging options: (empty)
- Application Projects:**

Domain Name	Application Project Name	Build Configuration
xrt	test	Emulation-HW

The Assistant window shows the project hierarchy, including 'test_system [System]', 'test [Embedded OpenCL]', 'Emulation-SW [Software Emulation]', 'Emulation-HW [Hardware Emulation]', and 'Hardware [Hardware]'. The Console window displays the build process output, including errors and warnings related to the linker and platform configuration.

Vitis GUI – Project Setting



The screenshot displays the Vitis GUI interface for configuring a project named 'test'. The 'Application Project Settings' window is open, showing the following configuration:

- General:**
 - Project name: test
 - Platform: xilinx_zcu104_base_202010_1
 - Runtime: OpenCL
 - Number of devices: 1
- Options:**
 - Target: Hardware Emulation
 - Host debug:
 - Kernel debug:
 - Kernel debug mode: Waveform
 - Report level: Default
 - Hardware optimization: Default optimization (-O0)
- Hardware Functions:**

Name	Compute Units	Port Data Width	Max Memory Ports
binary_container_1			
krnl_vadd	1	Auto	<input type="checkbox"/>

The Assistant window shows the project structure under 'test_system' [System], including 'test' [Embedded OpenCL] and 'Emulation-HW' [Hardware Emulation]. The Console window displays the build process logs, indicating a failure in the 'vpl' step:

```
Build Console [test, Emulation-HW]
[12:21:11] Run vpl: Step update_db: completed
[12:21:11] Run vpl: Step generate_target: Started
[12:21:43] Run vpl: Step generate_target: Completed
[12:21:43] Run vpl: Step config_hw_emulation: Started
[12:22:06] Run vpl: Step config_hw_emulation: Failed
[12:22:10] Run vpl: FINISHED. Run Status: config hw emulation ERROR
ERROR: [VPL 60-773] In '/home/adiuvo/Vitis_Lab/test/Emulation-HW/binary_container_1.build/link/vivado/vpl/runme.log', caught Tcl error:
ERROR: [VPL 60-2373] In '/home/adiuvo/Vitis_Lab/test/Emulation-HW/binary_container_1.build/link/vivado/vpl/runme.log', caught error: ERROR:
ERROR: [VPL 60-2373] In '/home/adiuvo/Vitis_Lab/test/Emulation-HW/binary_container_1.build/link/vivado/vpl/vivado.log', caught error: ERROR:
WARNING: [VPL 60-732] Link warning: INFO: Platform does not require extraction of debug/profile metadata.
ERROR: [VPL 60-1328] Vpl run 'vpl' failed
ERROR: [VPL 60-806] Failed to finish platform linker
INFO: [v++ 60-1442] [12:22:10] Run run_link: Step vpl: Failed
Time (s): cpu = 00:03:11 ; elapsed = 00:01:22 . Memory (MB): peak = 1339.543 ; gain = 0.000 ; free physical = 87363 ; free virtual = 1259
ERROR: [v++ 60-661] v++ link run 'run_link' failed
ERROR: [v++ 60-626] Kernel link failed to complete
ERROR: [v++ 60-703] Failed to finish linking
INFO: [v++ 60-1653] Closing dispatch client.
makefile:91: recipe for target 'binary_container_1.xelbin' failed
```

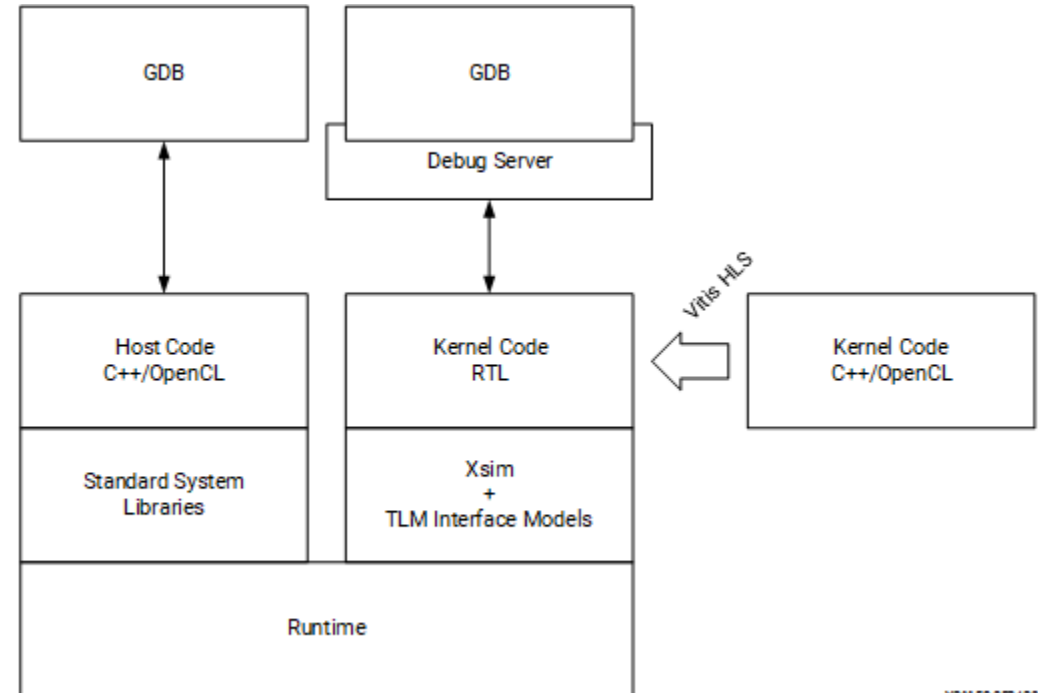
Vitis-Debug

Can Debug

- » Software Emulation
- » Hardware Emulation

Hardware flow insert ILA

Debugging will use QEMU and Logic Simulator



X21159052420



ADIUVO

ENGINEERING AND TRAINING, LTD.

www.adiuvoengineering.com



adam@adiuvoengineering.com