



# 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

# What is PetaLinux? (a set of tools)

## A build system:

- » A set of open-source cross-platform development command line and menu tools for use running under an x86 Linux Host OS
- » Just six PetaLinux commands (with many options):  
`petalinux-create`, `petalinux-config`, `petalinux-build`, `petalinux-boot`, `petalinux-package`, `petalinux-util`
- » The build system tailors embedded Linux to run on many different Xilinx platforms: Zynq MPSoC, Zynq and MicroBlaze. See: <https://www.xilinx.com/support/answers/71653.html>
- » PetaLinux tools now use the open-source community Yocto system since v2016.3
  - This is very useful for embedded Linux developers!
- » Also includes an emulator (QEMU) that can emulate PetaLinux under x86 Linux
  - Very useful for Kernel and some device driver development

# What is PetaLinux? (what comes with it)

## Source code, libraries, applications and Yocto recipes

- » Mostly open-source based with a few exceptions for some hardware drivers
- » PetaLinux includes and manages the Linux kernel sources and libraries
- » Various hardware drivers and modules
- » Traditional Linux applications and utilities

Through Yocto, PetaLinux can make use of additional embedded Linux applications beyond what Xilinx supplies.

- » More on that later!

A more complete list of what is included by Xilinx can be found here:

<https://www.xilinx.com/support/answers/71653.html>

# What is PetaLinux? (what goes in the firmware)

## Includes Bootloaders, Trusted Platforms

- » U-Boot, FSBL, ARM Trusted Firmware,

## Vivado Hardware Designs

- » PetaLinux device drivers work with Vivado IP blocks
- » PetaLinux imports hardware designs into the PL (FPGA)

## A Linux OS that runs on Xilinx Zynq MPSoC, Zynq (or MicroBlaze) CPUs

- » 2018.3 uses Linux Kernel version 4.14
- » For single or multiple CPUs

Linux applications, device drivers and lots of configuration files

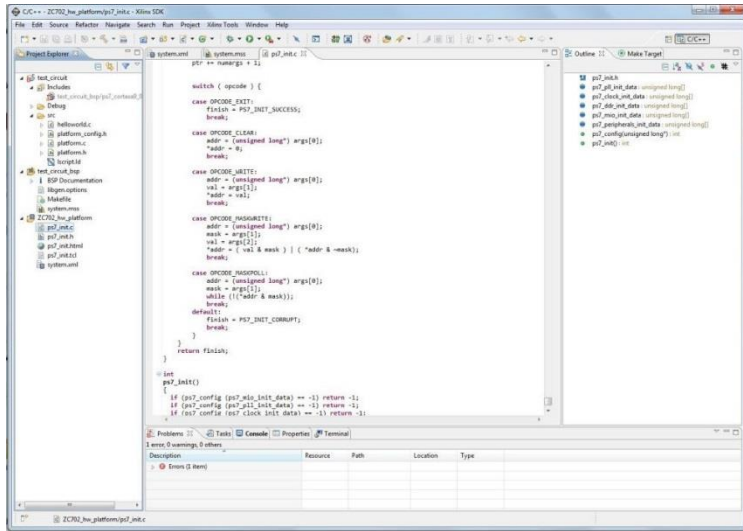
Detailed hardware and memory requirements can be found here: <http://www.xilinx.com/cgi-bin/docs/rdoc?v=latest;d=ug1144-petalinux-tools-reference-guide.pdf>

This is User Guide 1144: it is THE master reference book for PetaLinux!

# What is PetaLinux? (part of an ecosystem)

Xilinx SDK and PetaLinux work together

- » SDK is a complete editing, compiling and debugging environment based on Eclipse

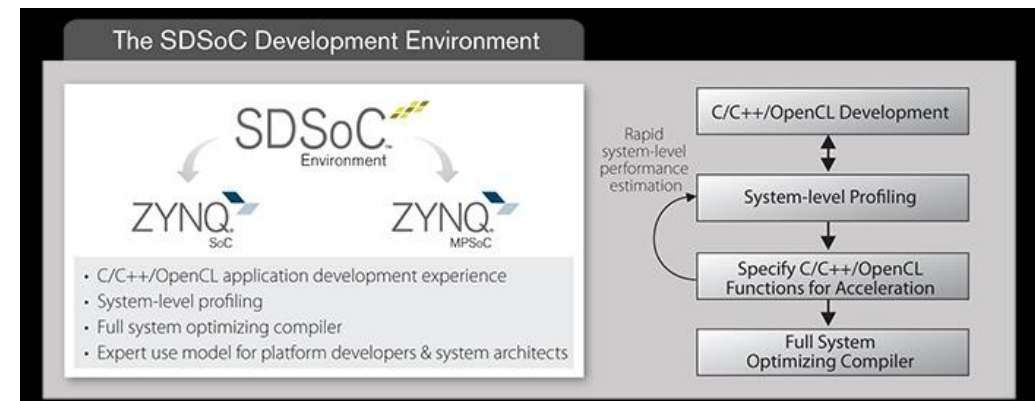


The PetaLinux tools, packaging, version management and source are all funded and supported directly by Xilinx

- Vivado and PetaLinux work together



- SDSoC and PetaLinux work together

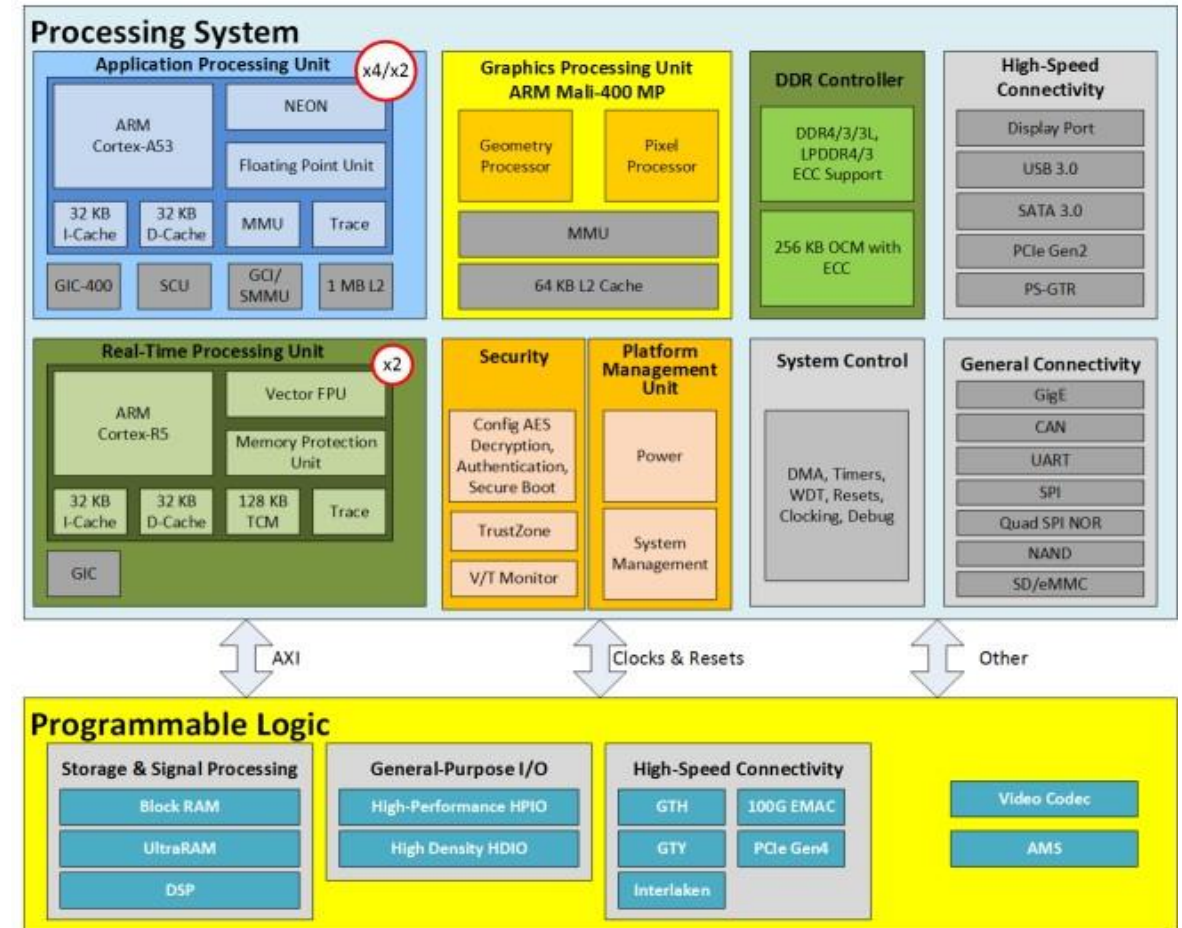
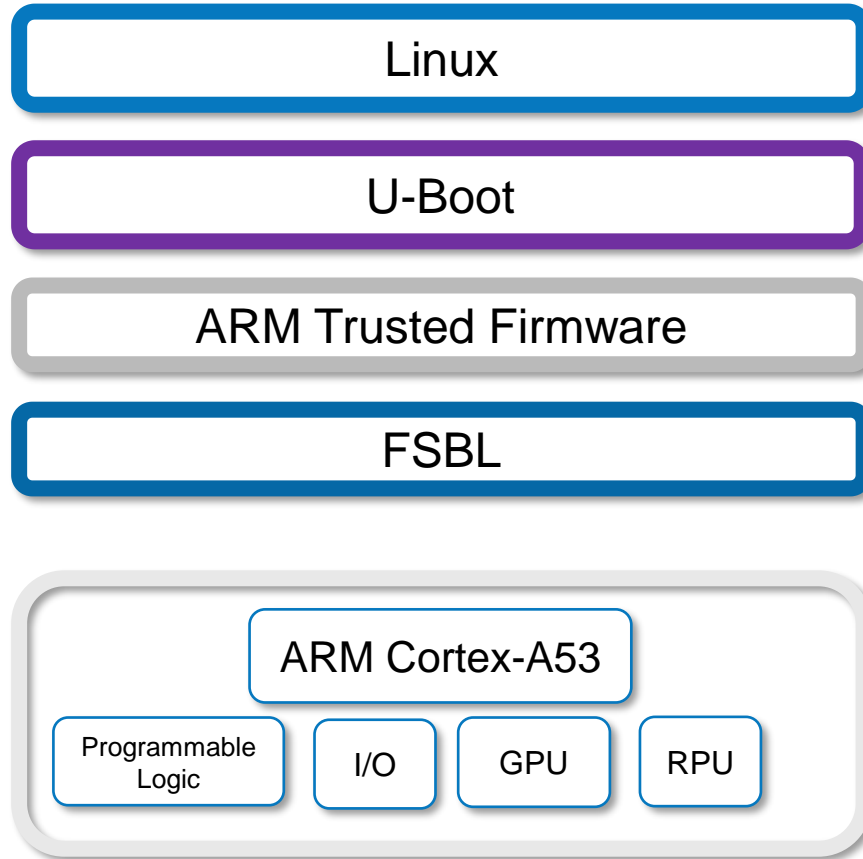


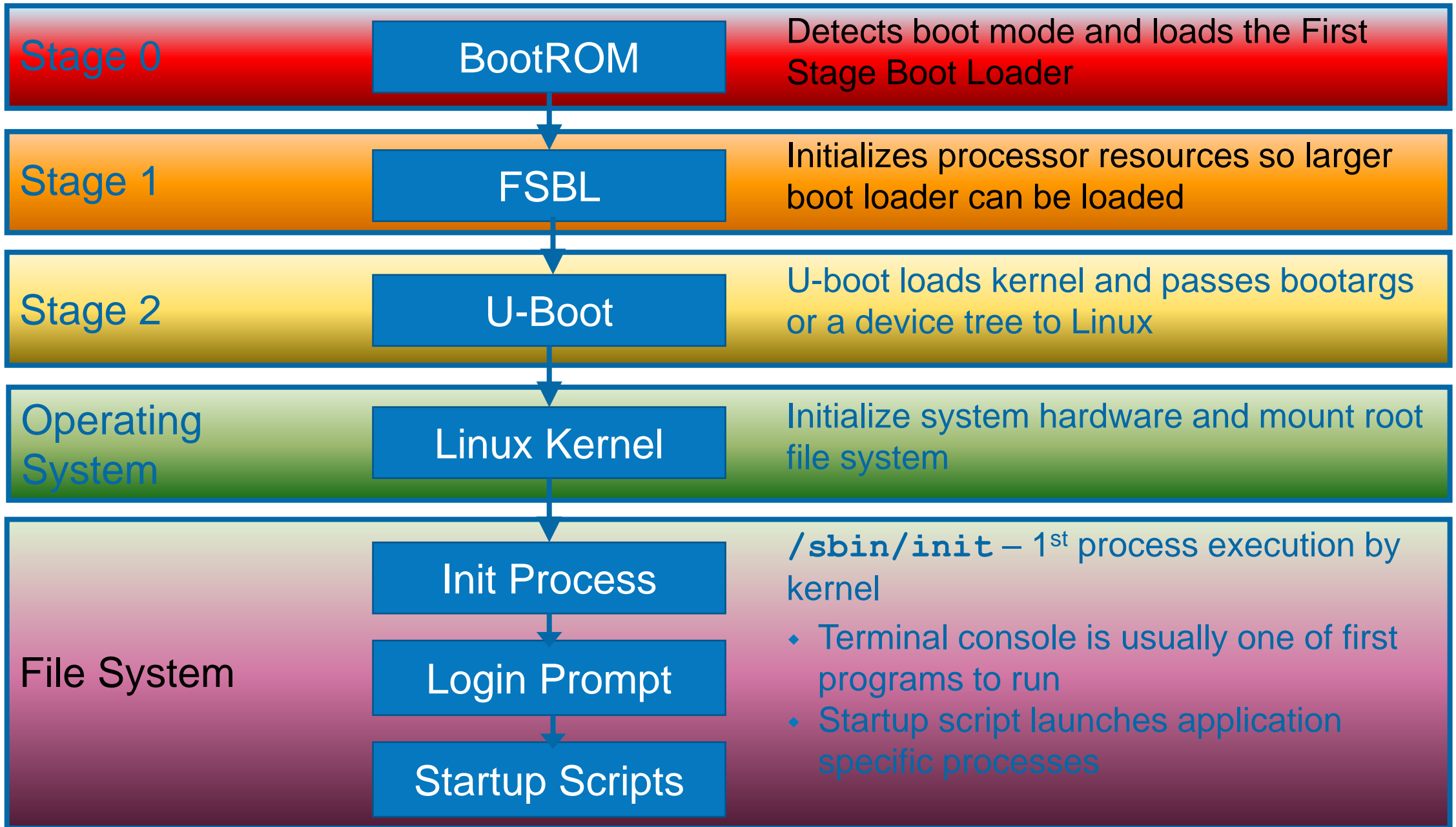
# PetaLinux Tools Testing, Release, and Documentation

- u-boot, Linux kernel, Xen hypervisor
  - Sources:
    - Kernel.org + Xilinx patches
  - Testing:
    - Xilinx silicon and Xilinx reference board features
  - Documentation:
    - Source code
    - Open source communities/forums
    - Answer Records for select interim issues
- Linux file system
  - Sources:
    - Yocto community code base
  - Testing:
    - Build verification
    - Testing of Xilinx-unique software and features
  - Release strategy:
    - Bug fixes with next Yocto release integration
  - Documentation:
    - Source code
    - Open source communities/forums



# What is PetaLinux for Ultra96?





# PetaLinux Multitasking Capabilities

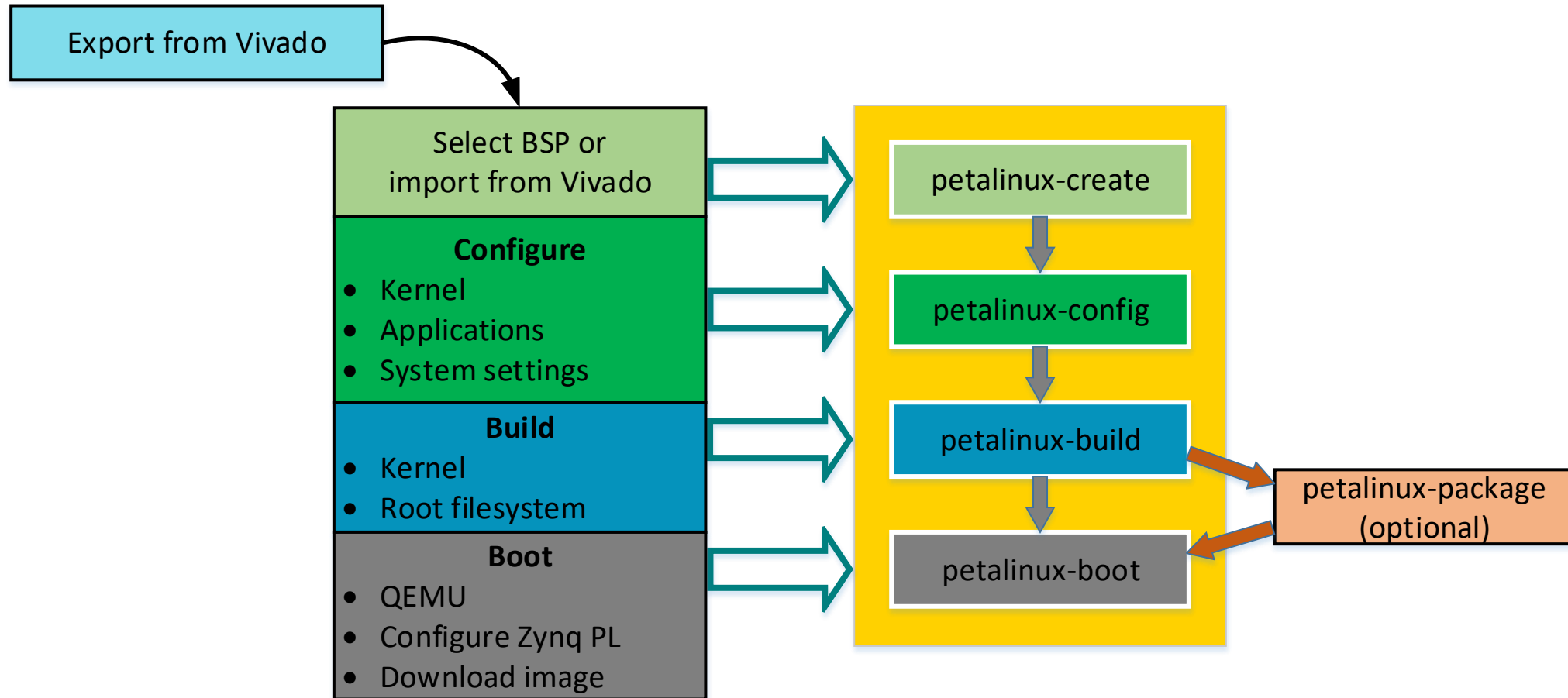
Embedded Linux (like Linux) is a multi-tasking multi-user operating system

- » On the Ultra96 there are four APU cores

PetaLinux multitasking means:

- » Fast (Four 64-bit cores) but not a real-time OS (RTOS)
- » If your application requires real-time, use the R5 cores. These core cannot run PetaLinux but can run FreeRTOS etc.
- » Communication between the A53 (PetaLinux) and 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)

# PetaLinux Tools Flow



# Creating a PetaLinux Project

Select BSP or  
import from Vivado

## petalinux-create

» Creates a new PetaLinux project or component

» Usage

**petalinux-create** [options] \

**-t|--type** <TYPE> \

**-n|--name** <COMPONENT\_NAME>

Required/Optional	Available Type
-t, --type <TYPE>	project - Create a PetaLinux project apps - Create an application libs - Create a library modules - Create a module
-n, --name <COMPONENT_NAME>	Specify name for the component/project Not required if BSP is specified

# Configuring a PetaLinux Project

## `petalinux-config`

- » Configures the PetaLinux project or component
- » Usage

```
petalinux-config [options] {--component <COMPONENT> | \  
--get-hw-description[=SRC] | \  
--searchpath <--ACTION> [VALUE]}
```

Required/Optional	Available Type
<code>--get-hw-description[=SRC]</code>	Get hardware description Look in the location of Vivado SDK export
<code>-p, --project &lt;PROJECT&gt;</code>	Path to PetaLinux project

`--get-hw-description` this is the same HDF file we used in the Avnet TTC HW and SW courses.

**Configure**

- Kernel
- Applications
- System settings

# Building a PetaLinux Project

- Build**
  - Kernel
  - Root filesystem

## `petalinux-build`

- » Builds the PetaLinux project or component
- » Usage

`petalinux-build [options]`

Required/Optional	Available Type
<code>-p, --project &lt;PROJECT&gt;</code>	Path to PetaLinux project
<code>-c, --component &lt;COMPONENT&gt;</code>	Specify the component If specified, e.g.: <code>-c rootfs</code> <code>-c rootfs/myapp</code>

# What is The Yocto Project?

The Yocto Project is an open source collaboration project that provides templates, tools and methods to help you create custom Linux-based systems for embedded products regardless of the hardware architecture. It was founded in 2010 as a collaboration among many hardware manufacturers, open-source operating systems vendors, and electronics companies to bring some order to the chaos of embedded Linux development.

## It's a framework, not a distribution

- » Allows users to create ***their own*** distributions
- » Contains reference distributions

## Commonly used by several organizations:

- » Wind River Linux
- » Mentor Embedded Linux
- » Just about everyone else...

## Uses a build tool called **bitbake**

PetaLinux 2018.3 abstracts most of the bitbake details, commands, etc.





# Why Yocto?

## The challenge...

- » Linux is becoming increasingly popular for embedded systems
- » Non-commercial and commercial embedded Linux has many distros

## Leads to the result...

- » Developers spend lots of time porting or making build systems
- » Leaves less time/money to develop interesting software features
- » The industry needs a common build system and core technology
- » Industry leaders joined together in 2010 to form the Yocto Project

## With these benefits...

- » Less time spent on things which don't add value (build system, core Linux components)
- » Linux grows more in embedded
- » More time spent on product differentiation and interesting software

# Yocto Layers, Recipes, and Bitbake

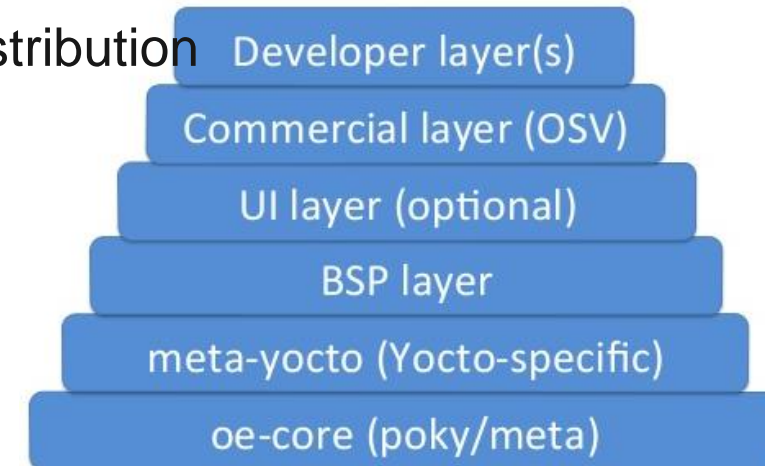
Yocto builds and assembles Linux images based on *layers*



- » Layers are named **meta-something**, such as:
  - **meta-yocto**, **meta-xilinx**, **meta-xilinx-tools**, etc.

## Layers consist of *recipes*

- » Recipes describe individual components of a target Linux distribution
- » Can be written to do all sorts of things:
  - Apply source code patches to the kernel
  - Copy files to a target root filesystem
  - Build / install user applications
  - *Build and install kernel drivers for wireless modules*



## PetaLinux 2018.3 specifics

- » Bitbake is abstracted using familiar commands, e.g. petalinux-build, etc.
- » Layer set aside for user customization: <project>/project-spec/meta-user

# Source Fetching

Recipes call out location of all sources

- » Can be local or remote (internet)

(look for SRC\_URI in \*.bb files, more about this in Lab 6)

Bitbake can get sources from git, svn, bsr, from tarballs, and many more

Versions of packages can be fixed or updated automatically

# PetaLinux and Yocto



PetaLinux supports and is supported by the Yocto ecosystem

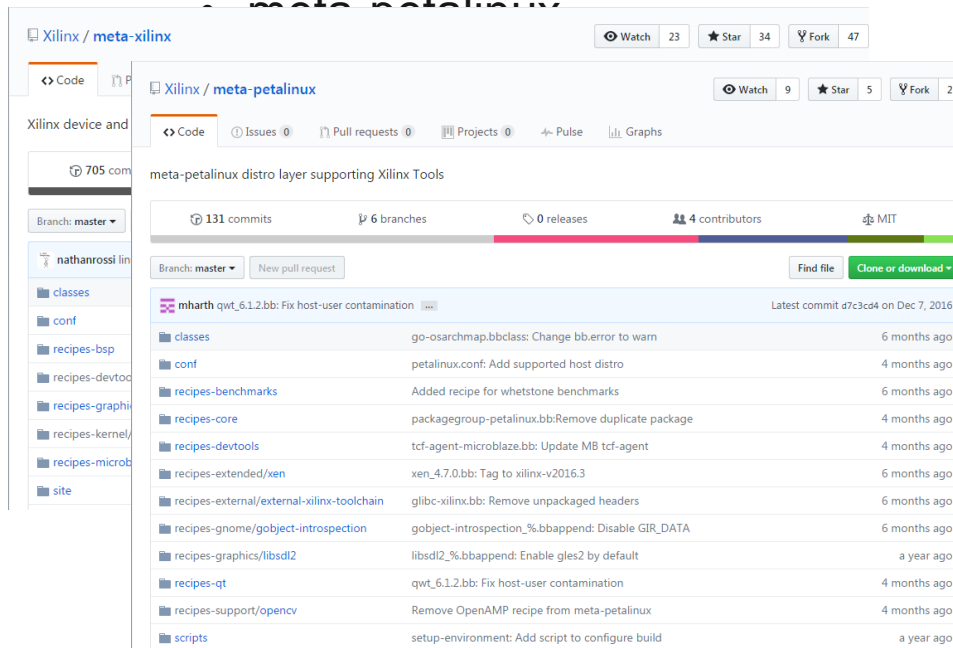
- » PetaLinux tools have the ability to integrate Yocto-generated RPM feeds into the native configured Linux filesystem
  - Allows designs to fully leverage in-house and Yocto-community software
- » Yocto build system can generate the complete PetaLinux filesystem
  - Ensures that all Yocto-built software is library-compatible with Xilinx-provided software
  - Allows comprehensive revision and source control
- » Xilinx GIT provides Yocto recipes for all PetaLinux components
  - ARM Trusted Firmware
  - U-Boot
  - Kernel
  - Root filesystem

# PetaLinux Yocto Recipes

Xilinx Yocto recipes available now!

» [https://github.com/xilinx](https://github.com/xilinx/meta-petalinux)

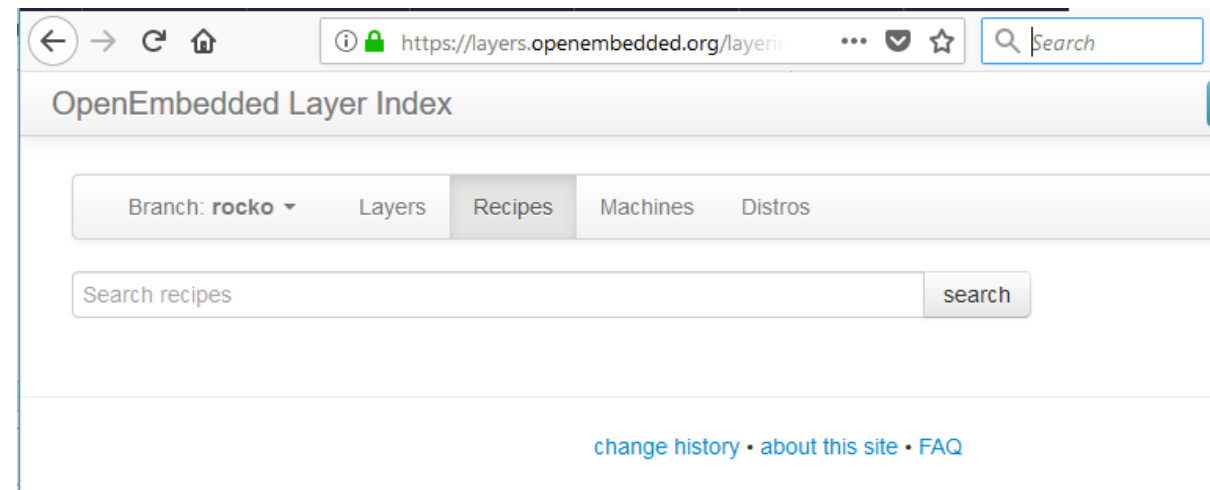
- meta-xilinx



- 3<sup>rd</sup> party recipes (use at own risk)

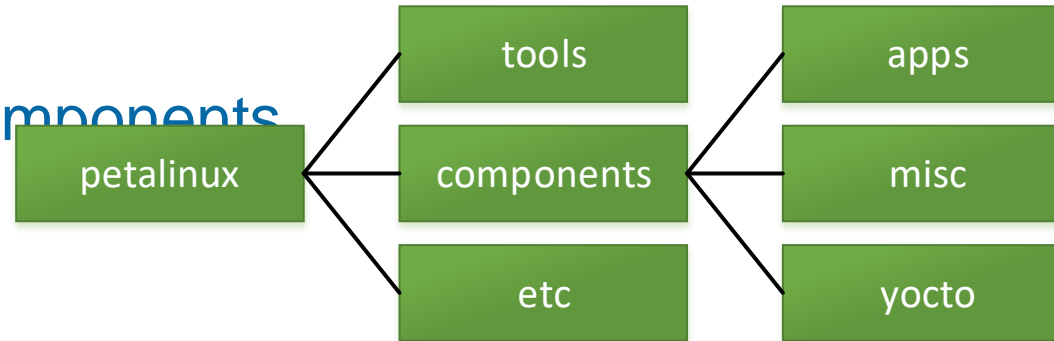
– PetaLinux 2018.3 aka rocko

<https://layers.openembedded.org/layerindex/branch/rocko/recipes/>



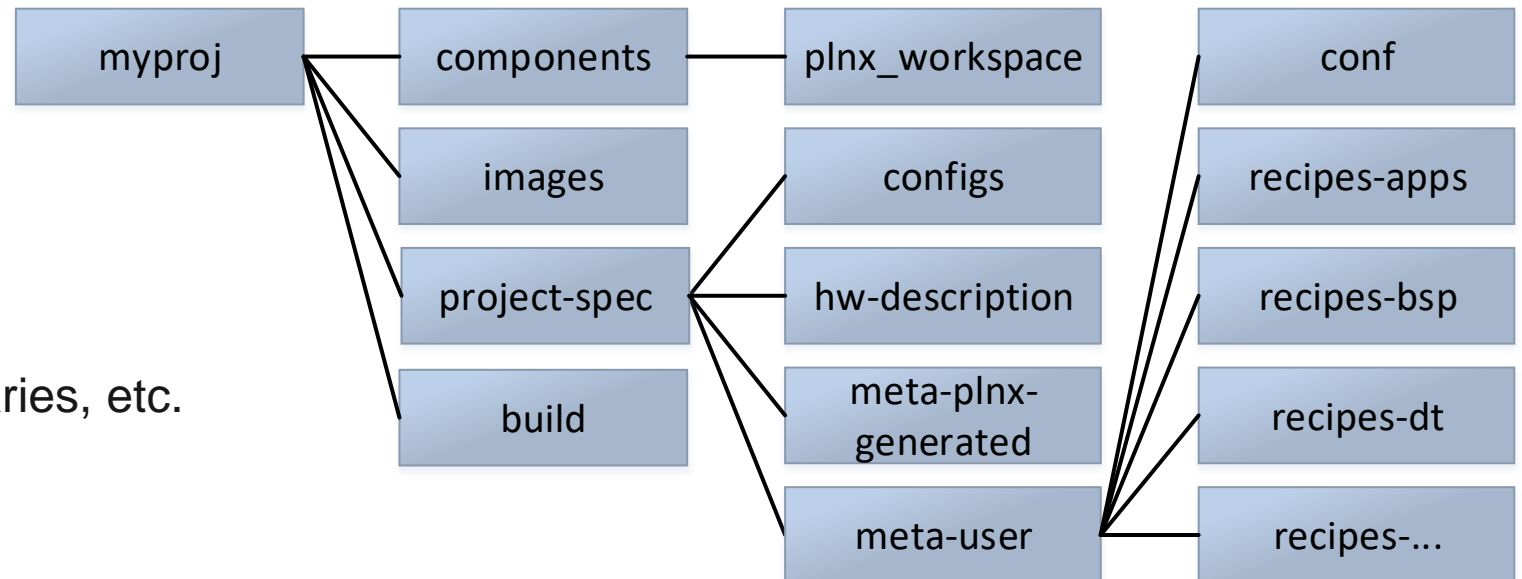
# PetaLinux Install and Project Taxonomy

Tools and default source components  
in install area



## Per-project directory

- » Configuration files
- » Custom recipes
  - Applications, scripts, libraries, etc.
- » Build products



# PetaLinux Project Example

```

.
├── conf
├── recipes-apps
│   ├── gpio-demo
│   │   └── files
│   └── peekpoke
│       └── files
├── recipes-bsp
│   ├── custom
│   │   └── files
│   ├── device-tree
│   │   └── files
│   ├── minized-firmware
│   │   └── files
│   ├── minized-misc
│   │   └── files
│   └── u-boot
│       └── files
├── recipes-connectivity
│   ├── bluez5
│   │   └── bluez5
│   └── wpa-supPLICANT
│       └── files
├── recipes-core
│   └── images
├── recipes-kernel
│   ├── linux
│   │   └── linux-xlnx
├── recipes-modules
│   └── lis2ds
│       └── files
├── recipes-multimedia
│   ├── pulseaudio
│   │   └── files
│   └── pulseaudio-service
│       └── files

```

meta-user Sub-folder Name	Purpose
	Custom user applications
	Customizations to the u-boot bootloader, device tree, hardware platform and user file additions to the root file system
	Networking protocols and support files
	Configuration files to tell PetaLinux what recipes to add to the system
	Kernel patches to correct, customize or enhance baseline kernel functionality
	Custom device drivers
	Custom application and support files for multimedia functionality

# Yocto References and Resources

## Online trainings

- » <https://www.doulos.com/content/events/LinuxZynqYocto.php>
- » <https://www.yoctoproject.org/tools-resources/videos/getting-started-yocto-project-new-developer-screencast-tutorial>

## References

- » <https://wiki.yoctoproject.org/wiki/FAQ>
- » <https://www.yoctoproject.org/>
- » <http://www.wiki.xilinx.com/Yocto>

## Documentation

- » <https://www.yoctoproject.org/documentation>
- » [https://www.xilinx.com/support/documentation/sw\\_manuals/xilinx2018\\_3/ug1144-petalinux-tools-reference-guide.pdf](https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_3/ug1144-petalinux-tools-reference-guide.pdf)



# PetaLinux Platform Configuration

In a Zynq MPSoC device, every PS and PL configuration can be different

- » Address map, memory size
- » IRQ structures, CPU, and device configuration

How does the kernel know what the hardware looks like?

- » Hard coded values or #defines? Definitely not!
- » A special data structure called the Device Tree

## Device Tree

- » A Zynq MPSoC Linux system can be viewed as a tree structure
  - CPU is the root node
  - Buses are branches
  - Devices are nodes
  - Memory, peripherals
- » Each node has properties
  - Address range
  - Device properties
  - Interrupt connection
  - Etc...

# Device Tree

## DTS file

- » Device Tree Source
- » Textual description of system device tree

## DTB

- » Device Tree Blob
- » Compiled, binary representation of the DTS

## DTC

- » Device Tree Compiler
- » Provided in PetaLinux installation
- » Converts the DTS to a DTB
- » Can de-compile a DTB back to a DTS

# Device Tree Structure

A DTB is provided to the kernel at boot time

- » Kernel startup code parses the tree
- » Instantiates and initializes kernel appropriately
  - System memory location/size
  - Device types, address mappings, configuration options

Kernel infrastructure binds drivers to devices

- » Matched against the "compatible" property in the device node
- » Device driver declares known "compatible" devices

Greatly simplifies the problem of  
device ↔ driver binding

Device Tree Source Example (shown on right)

```
40 &sdhci0 {
41     status = "okay";
42     bus-width = <4>;
43     xlnx,has-cd = <0x0>;
44     xlnx,has-power = <0x0>;
45     xlnx,has-wp = <0x0>;
46     non-removeable;
47     wifi-host; /* adding for compatibility w/ bcmhd driver */
48 };
49
50 /* Define the external I2C devices and their names and addresses */
51 &axi_iic_0 {
52     #address-cells = <0x1>;
53     #size-cells = <0x0>;
54     compatible = "xlnx,xps-iic-2.00.a";
55     i2c5@1d {
56         #address-cells = <0x1>;
57         #size-cells = <0x0>;
58         compatible = "stm,lis2ds12";
59         reg = <0x1d>;
60     };
61 };
62
63 /* QSPI addresses are defined with petalinux-config, but here they are overwritten so that one can
   program the flash internally */
64 &qspi {
65     #address-cells = <1>;
66     #size-cells = <0>;
67     flash0: flash@0 {
68         compatible = "micron,m25p80";
69         reg = <0x0>;
70         #address-cells = <1>;
71         #size-cells = <1>;
72         spi-max-frequency = <50000000>;
73         partition@0x00000000 {
74             label = "boot";
75             reg = <0x00000000 0x00ff0000>;
76         };
77     };
78 };
```

# Device Tree Creation

PetaLinux includes an automated device tree generator

- » Generates device tree based on hardware design
- » All IP cores and properties exported in DTS

Kernel build process compiles DTS → DTB and links into kernel image

- » Auto-generated DTS files in  
<petalinux\_project>/components/plnx\_workspace/device-tree-generation/
  - Do not edit! Auto-generated on import of hardware platform from Vivado
- » User-editable system-top.dts DTS file in  
<petalinux\_project>/project-spec/meta-user/recipes-dt/device-tree/files/
  - User overwrite of default parameters and addition of new parameters
  - Changes to the system-top.dts will not be overwritten by subsequent petalinux-config

# PetaLinux Device Tree Inspection

By inspecting `/proc/device-tree` you can look at what PetaLinux has detected in the FPGA:

```
root@plnx_arm:/proc/device-tree# ls -l
total 0
-r--r--r-- 1 root root 4 Sep 18 11:13 #address-cells
-r--r--r-- 1 root root 4 Sep 18 11:13 #size-cells
drwxr-xr-x 2 root root 0 Sep 18 11:13 aliases
drwxr-xr-x 33 root root 0 Sep 18 11:13 amba
drwxr-xr-x 9 root root 0 Sep 18 11:13 amba_pl
drwxr-xr-x 2 root root 0 Sep 18 11:13 chosen
-r--r--r-- 1 root root 15 Sep 18 11:13 compatible
drwxr-xr-x 3 root root 0 Sep 18 11:13 cpus
drwxr-xr-x 2 root root 0 Sep 18 11:13 fixedregulator
drwxr-xr-x 2 root root 0 Sep 18 11:13 fpga-full
drwxr-xr-x 2 root root 0 Sep 18 11:13 memory
-r--r--r-- 1 root root 1 Sep 18 11:13 name
drwxr-xr-x 2 root root 0 Sep 18 11:13 pmu@f8891000
drwxr-xr-x 2 root root 0 Sep 18 11:13 usb_phy@0
drwxr-xr-x 2 root root 0 Sep 18 11:13 wlreg-on
```

```
root@plnx_arm:~# hexdump -C '/proc/device-tree/amba_pl/i2c@41610000/compatible'
00000000 78 6c 6e 78 2c 78 70 73 2d 69 69 63 2d 32 2e 30 |xltx,xps-iic-2.0|
00000010 30 2e 61 00                                     |0.a.|
00000014
root@plnx_arm:~#
```

Inspect attribute name>

# Key Concept: Memory Management Unit (MMU)

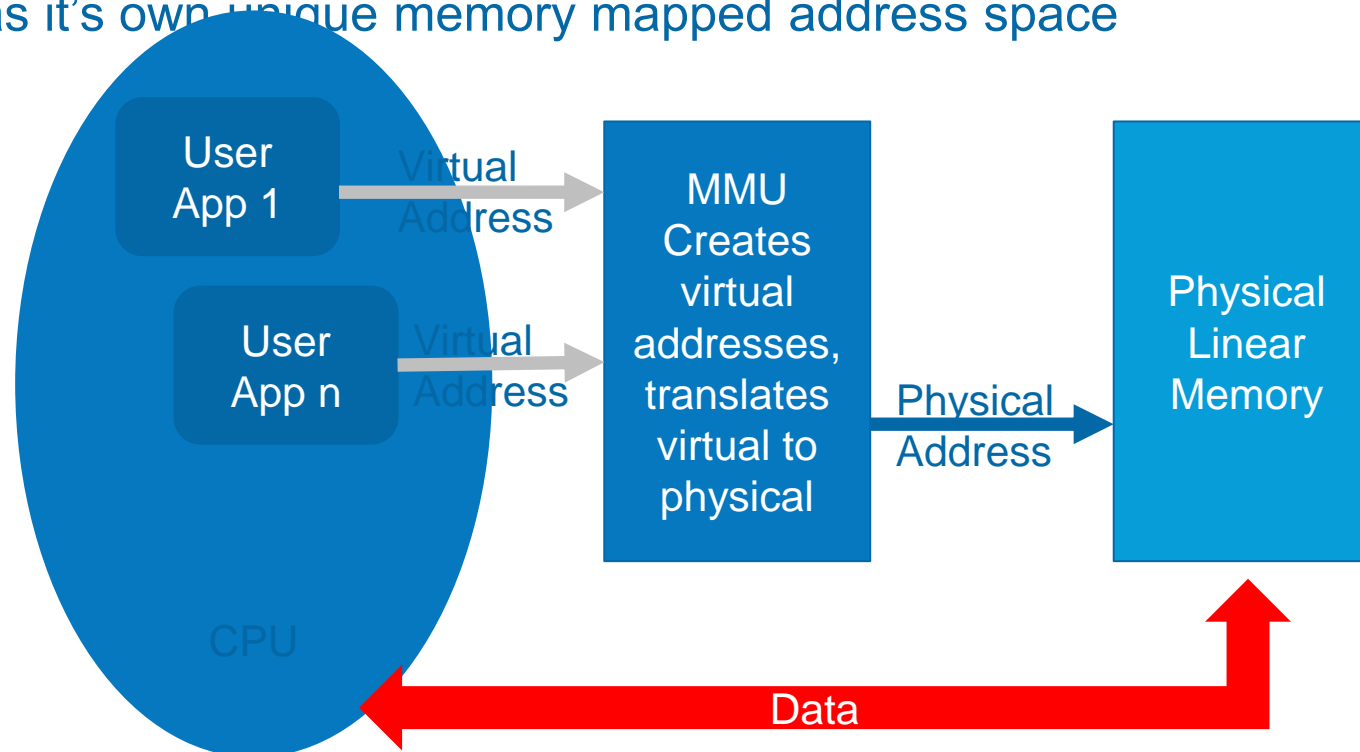
User applications access program and data from virtualized (pseudo) addresses

The MMU unit of the Zynq MPSoC is the translator from virtual to physical addresses

Each user program runs as if it is the only program on the CPU

Each user program has it's own unique memory mapped address space

Simplified  
Operation:



# Example /dev/i2c

- This is the general approach from C code for using a device on the I2C bus
- Ultra96 has 1 I2C bus which is connected to a I2C switch which contains several sub buses. These include Two I2C buses for each of the high and low speed expansion ports, Power Management bus and Current Monitor
- Example code below (for reference only)

```
int num_read;  
Char read_bytes[2];  
int file = open("/dev/i2c-2", O_RDWR); // Get a file handle to the I2C bus  
ioctl(file, I2C_SLAVE, 0x40); // Set the I2C address to 0x40  
write(file, "\0x1\0x2",2); // write 2-byte values 1,2  
num_read = read(file,read_bytes,2); // Attempt to read back up to 2 bytes  
close(file);
```

# Customize PetaLinux to Add New Components

## Components can be

- » Software applications
- » Kernel modules (device drivers)
- » Shell scripts
- » Kernel patches

## The PetaLinux commands like `petalinux-create` perform a lot of hidden tasks

```
$ petalinux-create -t apps --template c++ --name lsm6dsl-sensor --enable
```

- » Creates a new application named '**lsm6dsl-sensor**'
- » Creates a new C++ source file that can be used as a starting point for development
- » Adds the new application to the root filesystem configuration menu and enables it
  - Subsequent runs of `petalinux-build` will automatically build and add the app to the rootfs
- » Creates a new bitbake recipe and Makefile

## New component additions placed in special folder in the PetaLinux project

- » Can be found in `project-spec/meta-user/recipes-<apps | kernel | modules>`



# PetaLinux and Yocto Bitbake

PetaLinux commands abstract the details of the bitbake commands

- » Creation of new software applications, kernel modules (drivers), scripts, etc.
- » Add new bitbake recipes
- » Additions to configuration menus
- » Build, clean, config
- » Installation onto the root file system

Easy edits to makefile and recipe as application complexity grows

- » Add source files, etc.

PetaLinux command to build the root file system becomes bitbake command...

```
$ petalinux-build -c rootfs
```

```
$ bitbake petalinux-user-image -c do_rootfs
```

# What is in a Bitbake Recipe?

Descriptive information about the package

The version of the recipe

Existing Dependencies

Where the source code resides


Whether the source code requires any patches

How to compile the source code

Where on the target machine to install the package being compiled

# Simple Bitbake Recipe

Open ▾



```
#
# This file is the lsm6dsl-sensor recipe.
#

SUMMARY = "Simple lsm6dsl-sensor application"
SECTION = "PETALINUX/apps"
LICENSE = "MIT"
LIC_FILES_CHKSUM = "file://${COMMON_LICENSE_DIR}/MIT;md5=0835ade698e0bcf8506ecda2f7b4f302"

SRC_URI = "file://main.c \
          file://Makefile \
          "

S = "${WORKDIR}"

FILES_${PN} += "/home/root/*"

do_compile() {
    oe_runmake
}

do_install() {
    install -d ${D}${bindir}
    install -m 0755 lsm6dsl-sensor ${D}/home/root/|
}
```

Location and MD5 checksum of the license file for the application

Location of the source file(s) and Makefile

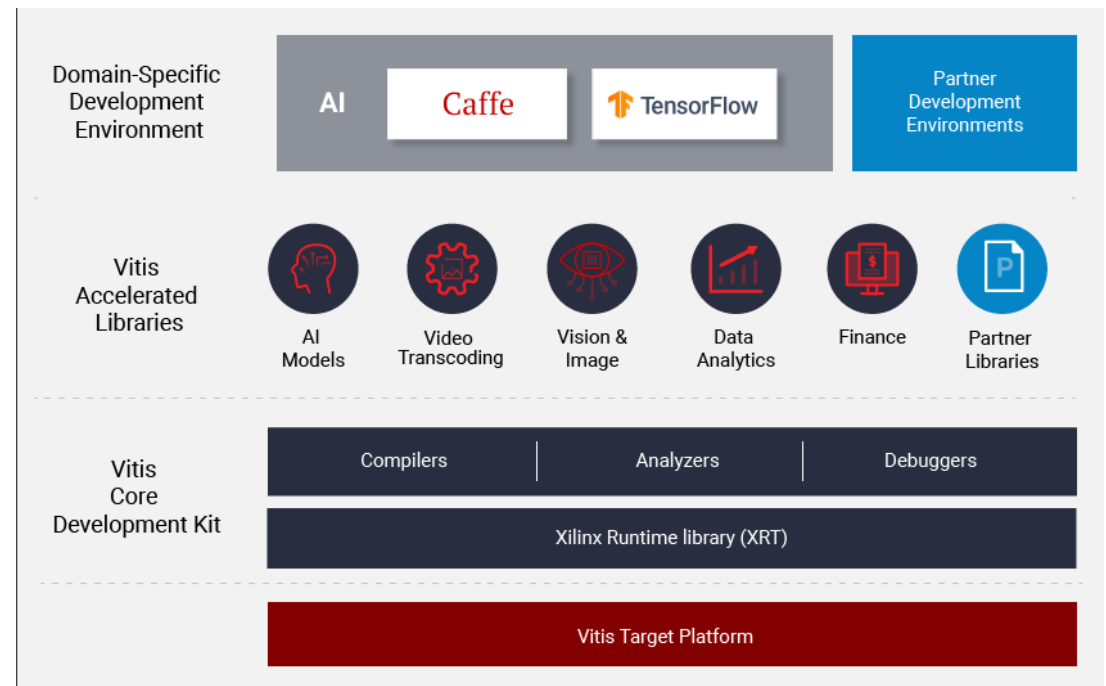
Recipe instructions

Specify where the application should be installed



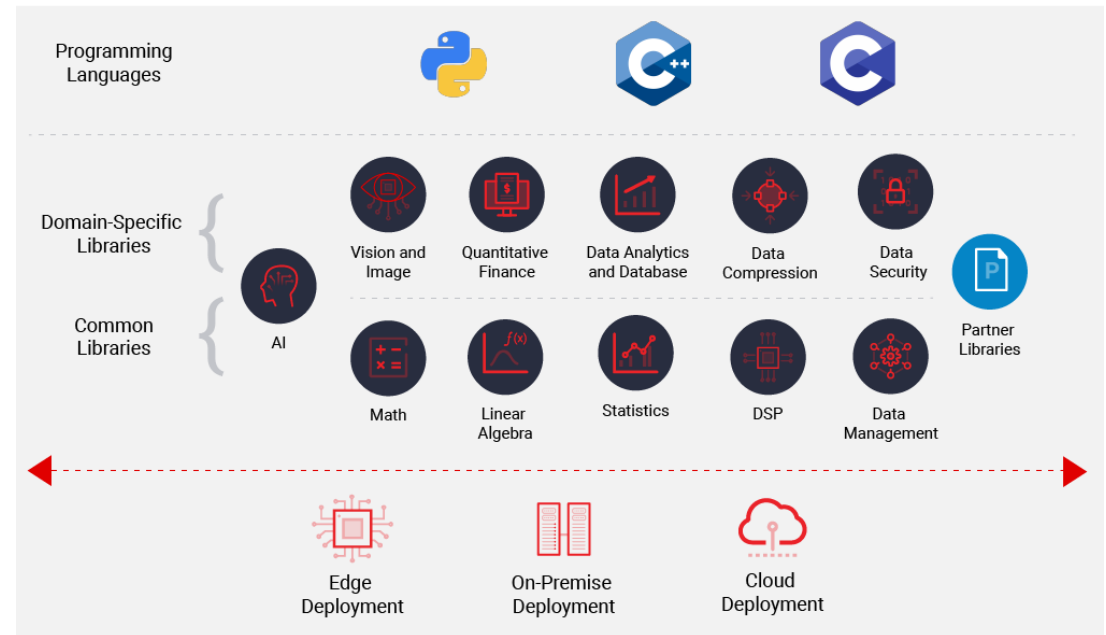
# What is Vitis

- 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 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

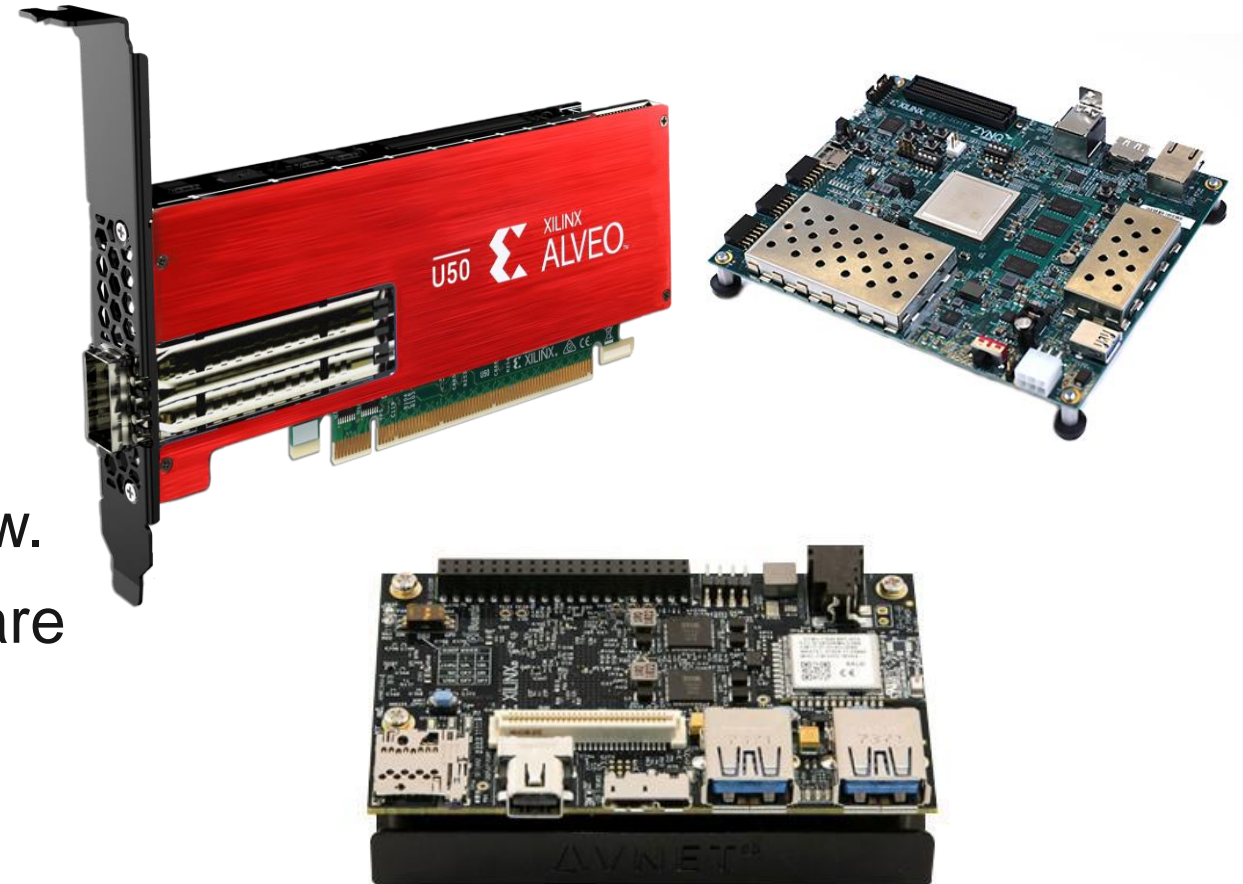
## Embedded

- » 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



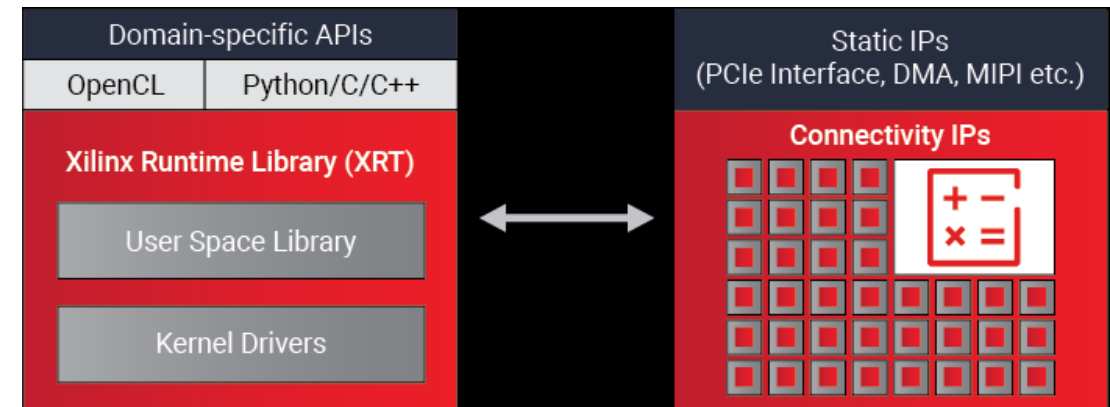
# Xilinx Runtime library

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

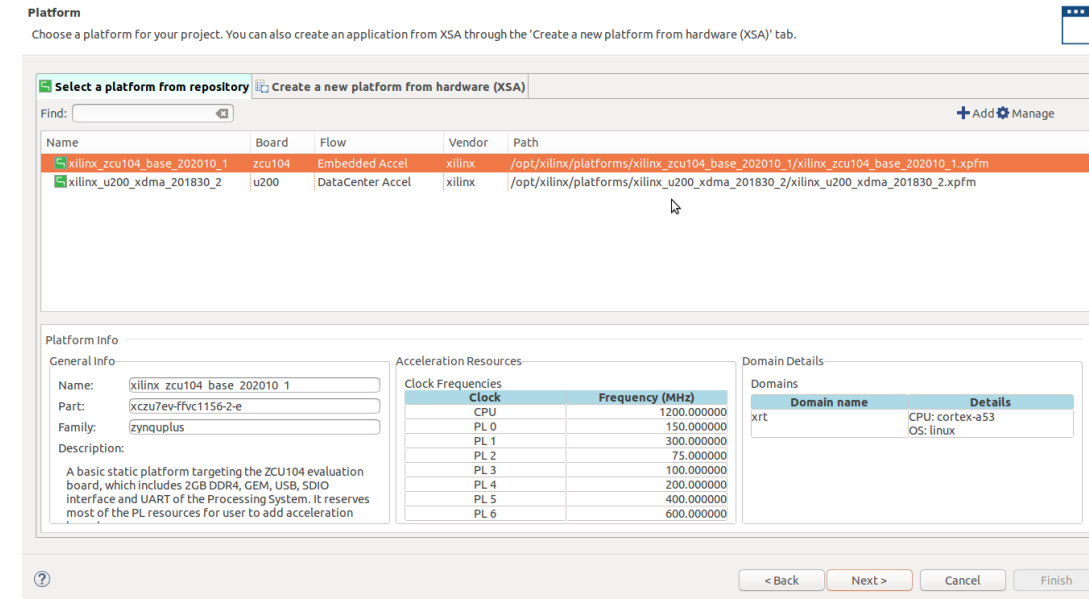
Board Management

```
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



# Vitis Output

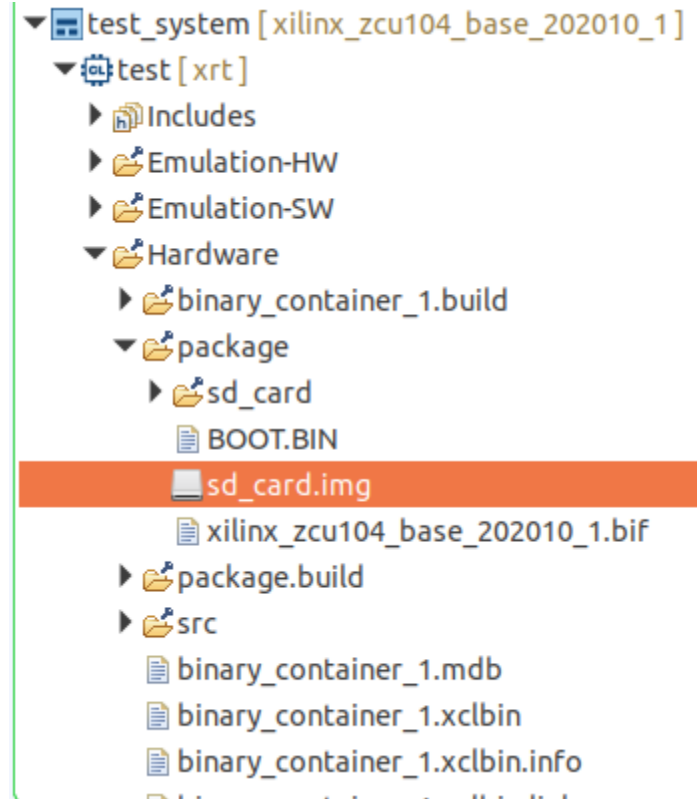
Compiled binary (host) and XCLbin (accelerator)

Embedded System Output

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

Cloud output

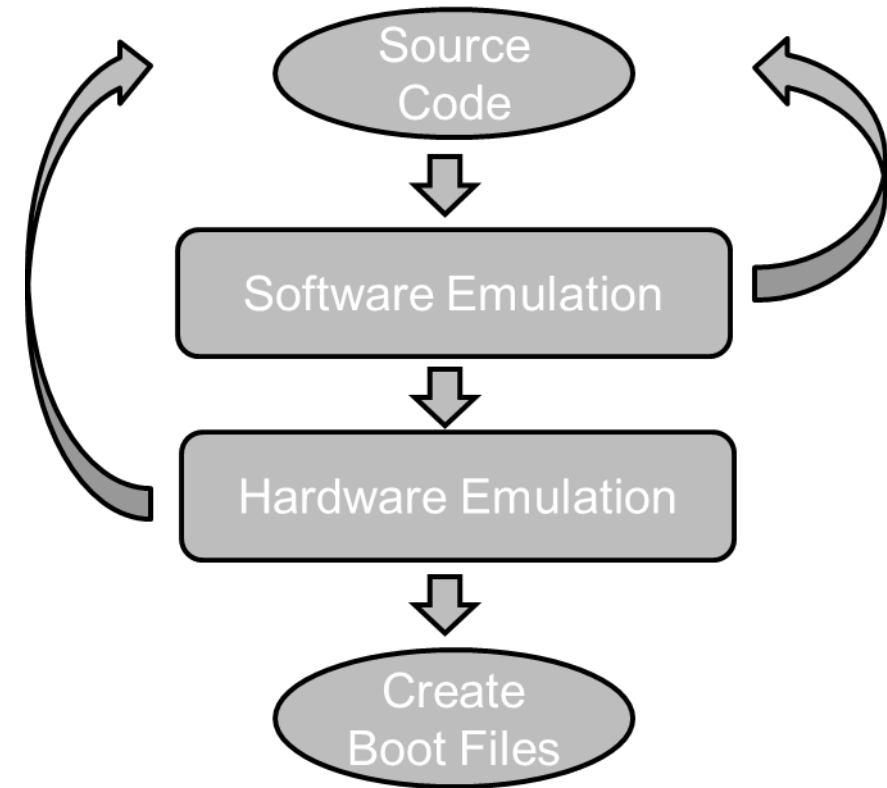
- » Binary and XCLBin



# Vitis Development Flow

Software Emulation – Syntax errors & algorithm verification

Hardware Emulation – Optimize Performance, Interfacing & Resources





# OpenCL 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

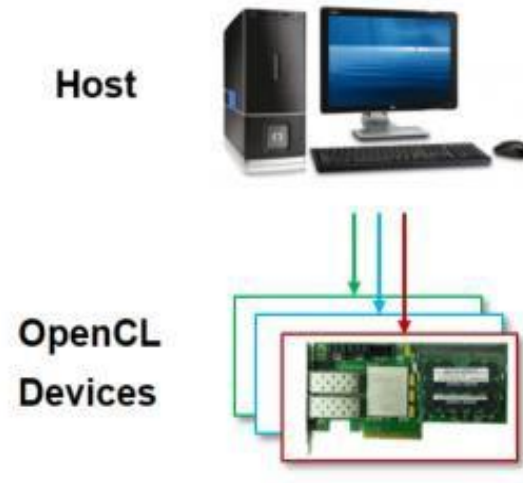


**OpenCL**

*Khronos Group*  
[www.khronos.org](http://www.khronos.org)



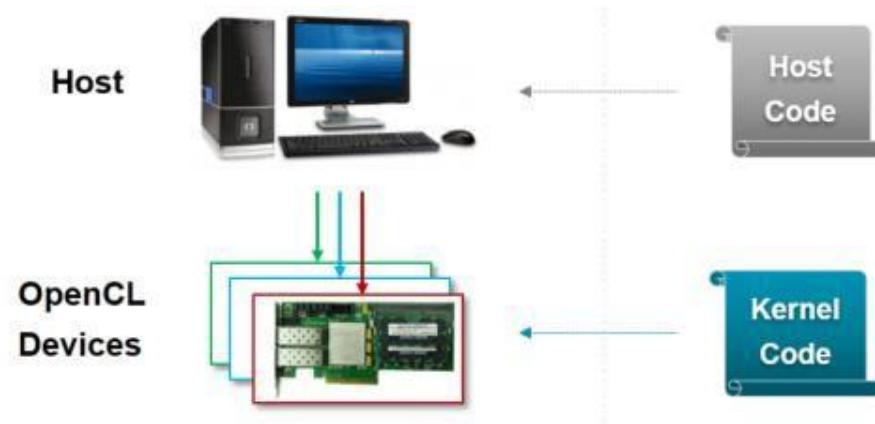
# OpenCL Framework



## Platform model

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

# OpenCL Framework



## Platform model

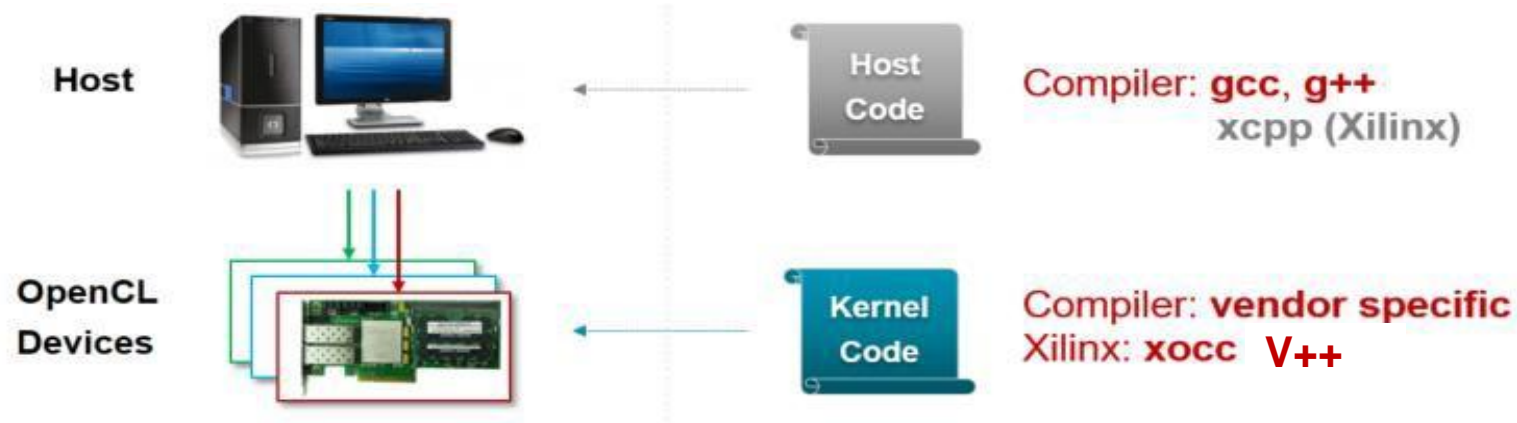
- Defines representation of ANY platform
- Contains
  - Single host
  - One or more OpenCL devices (compute device)

## 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

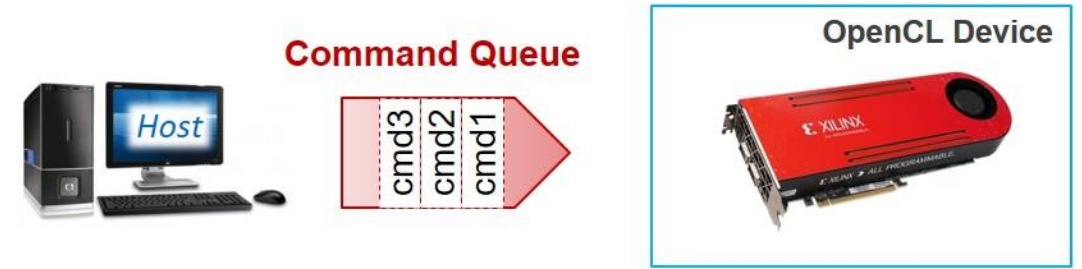
OpenCL application: Two parts

- 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

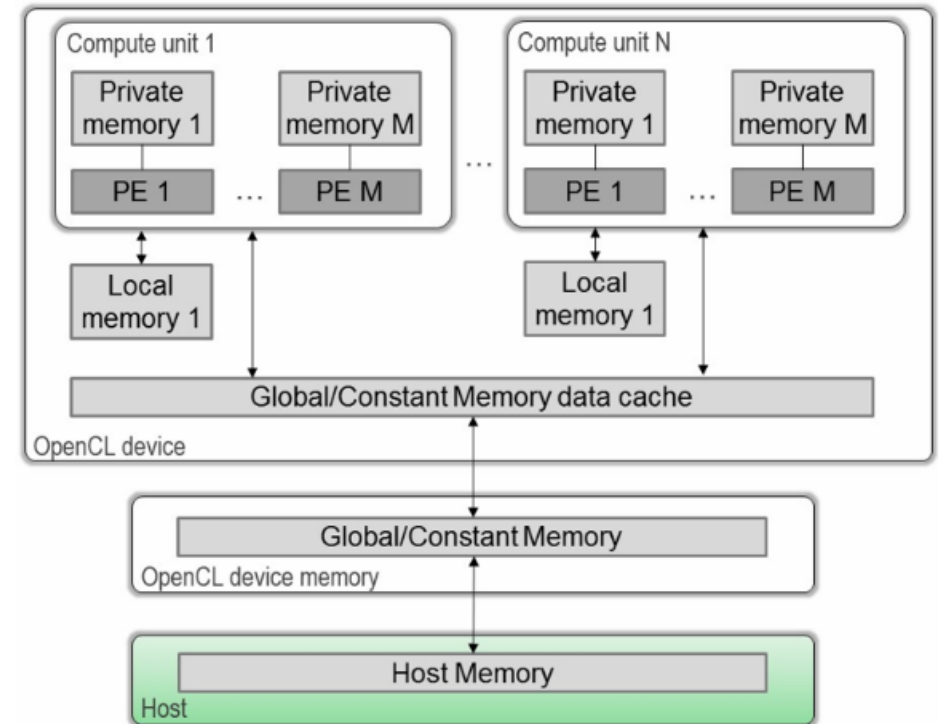
- Region of global memory
- Work items – read access only

## Local memory

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

## Private memory

- Accessible by a work-item





# 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

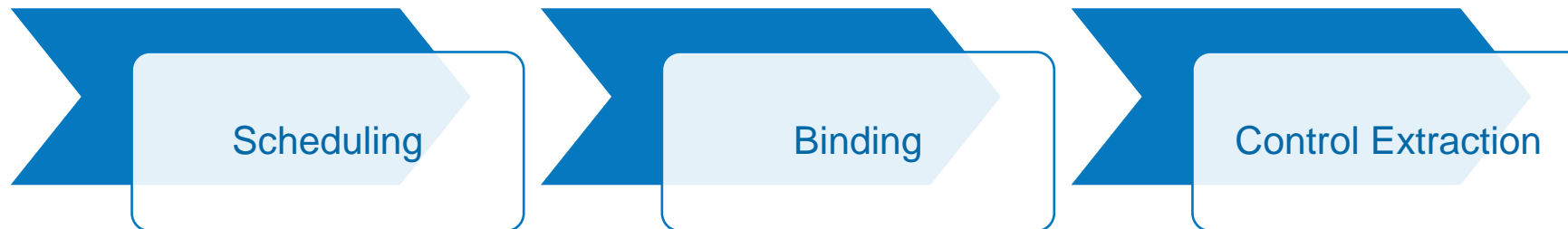
- » Loops need unrolling
- » Memory Structures need optimising
- » Resource allocation

HLS controlled via `#pragma` in the accelerated function

# Who has Used HLS before ?

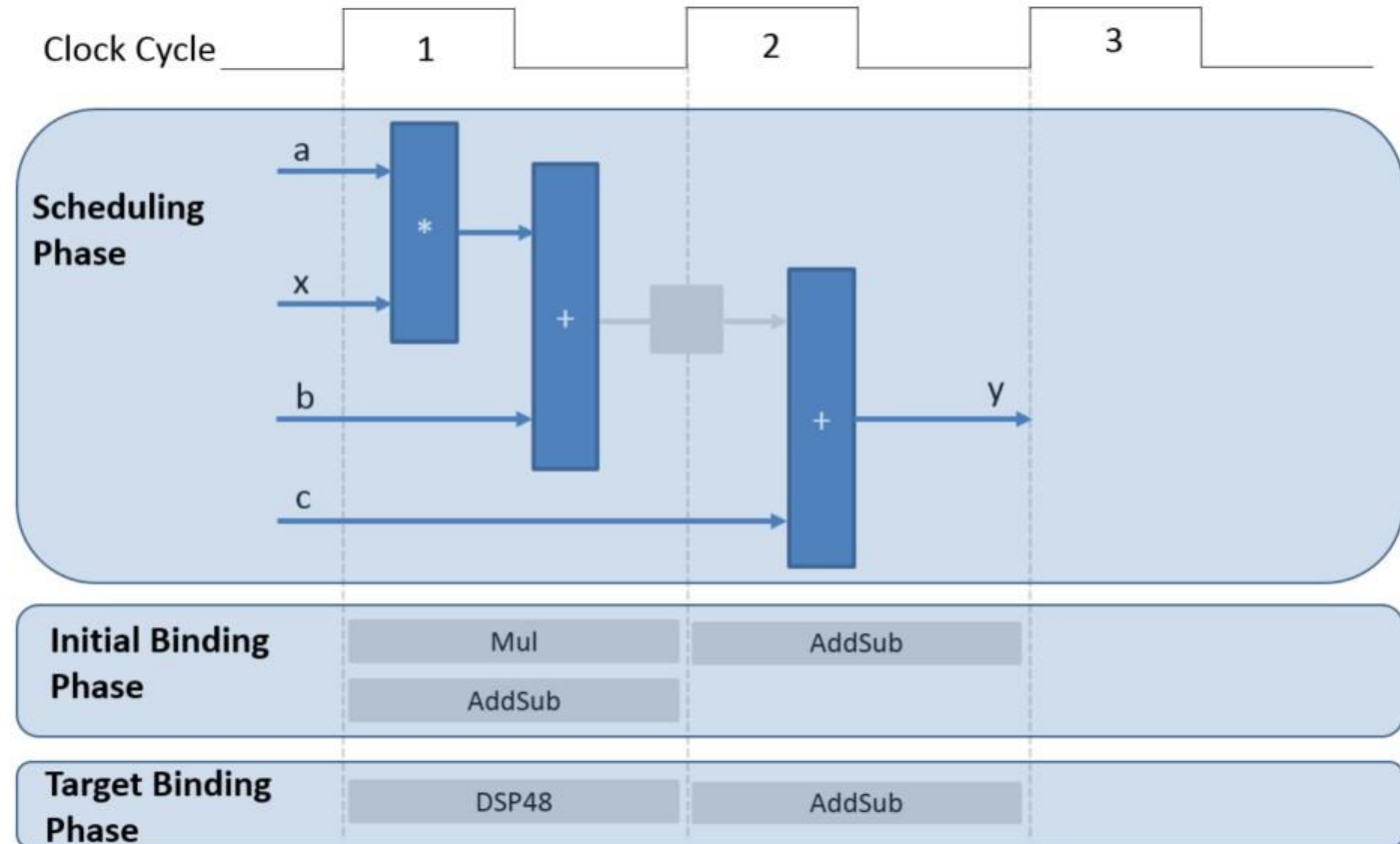
HLS came of age over the last 5 years

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



# Example of HLS

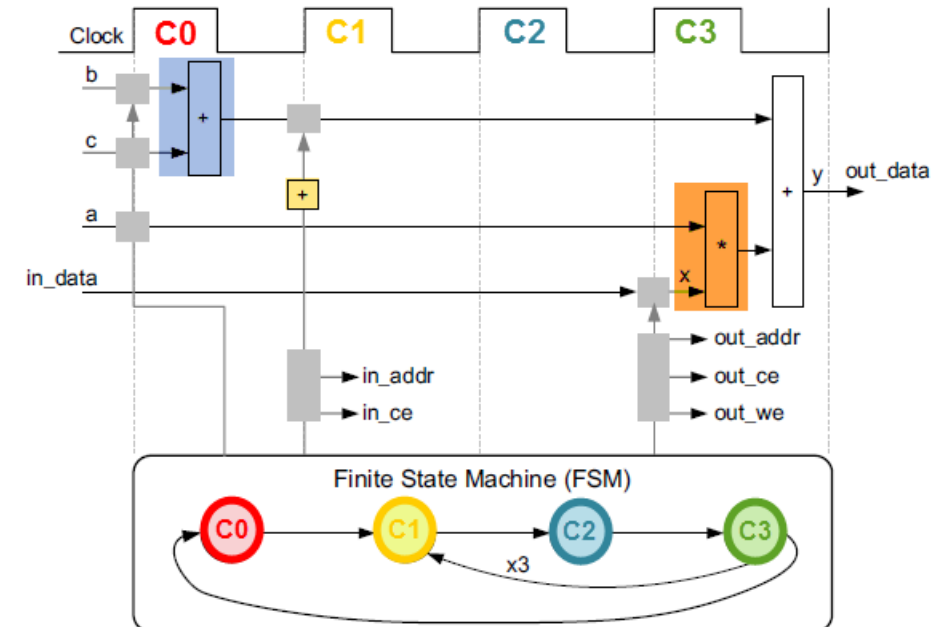
```
int foo(char x, char a, char b, char c) {
    char y;
    y = x*a+b+c;
    return y;
}
```



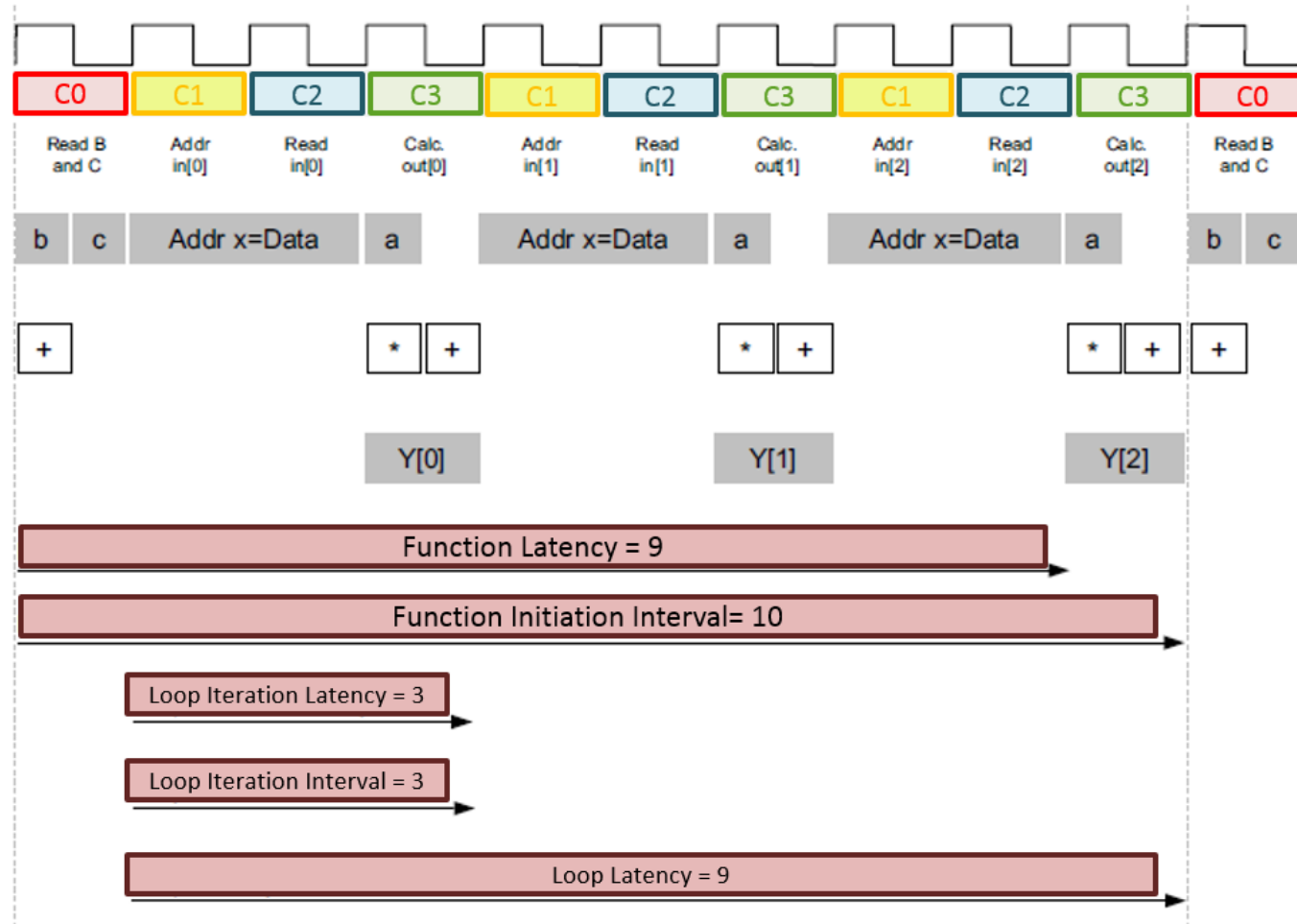
# Looking a little deeper

```
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

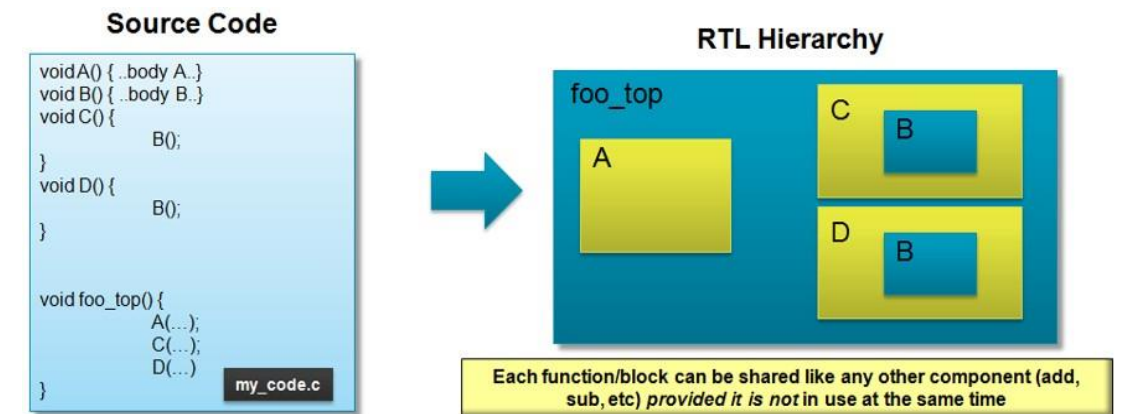
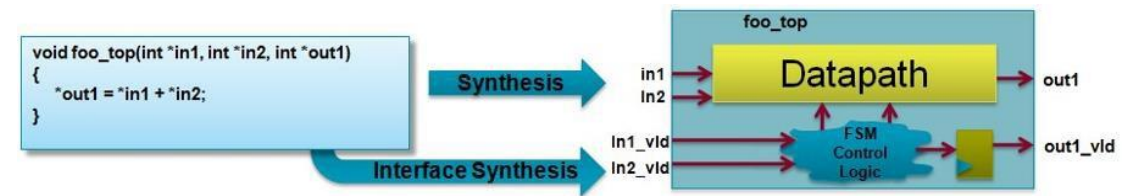
HLS synthesizes the C code in different ways

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



# Interfacing

	Argument Type	Scalar		Array			Pointer or Reference			HLS:: Stream
		Input	Return	I	I/O	O	I	I/O	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

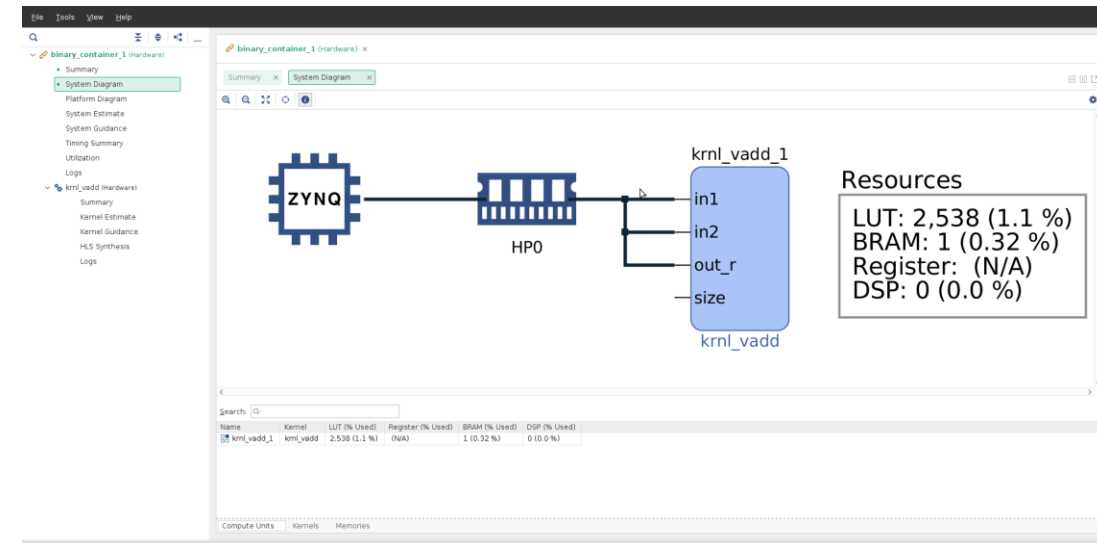
Optimization Possible at both Host and Kernel

Enables most responsive solution

Host optimization

Kernel optimization possible in OpenCL and C/C++

» Optimization Syntax differs

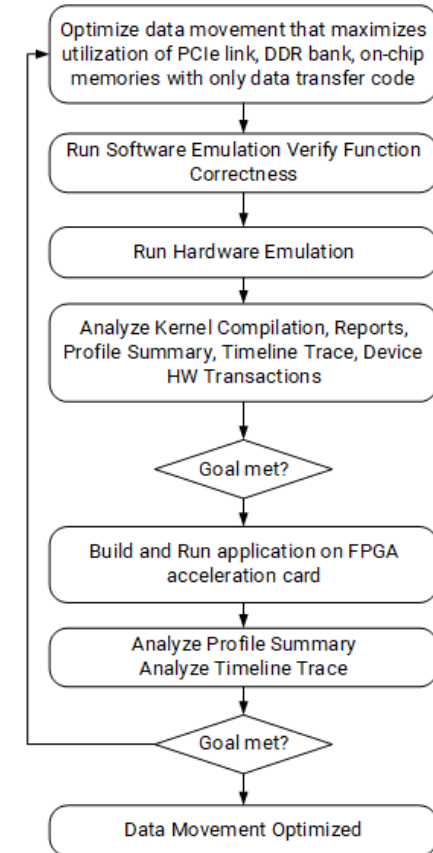


# Host Optimization

Optimize the data movement in the application before optimizing computation

## Compute Unit Scheduling

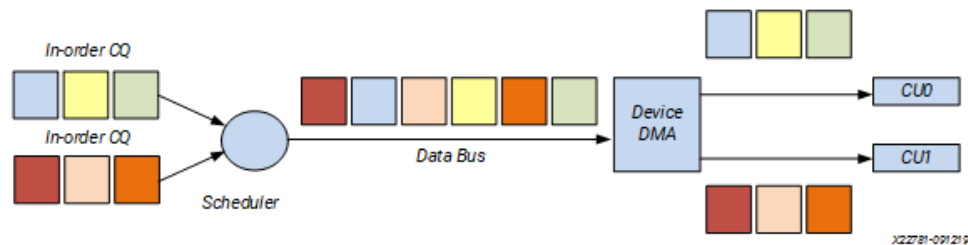
- » Multiple In-Order Command Queues
- » Single Out-of-Order Command Queue



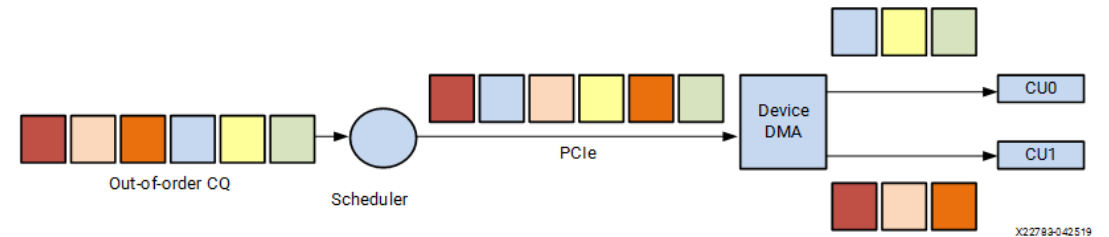
XZ2239-082719

# Host Optimization

## Multiple In-Order Command Queues



- Single Out-of-Order Command Queue



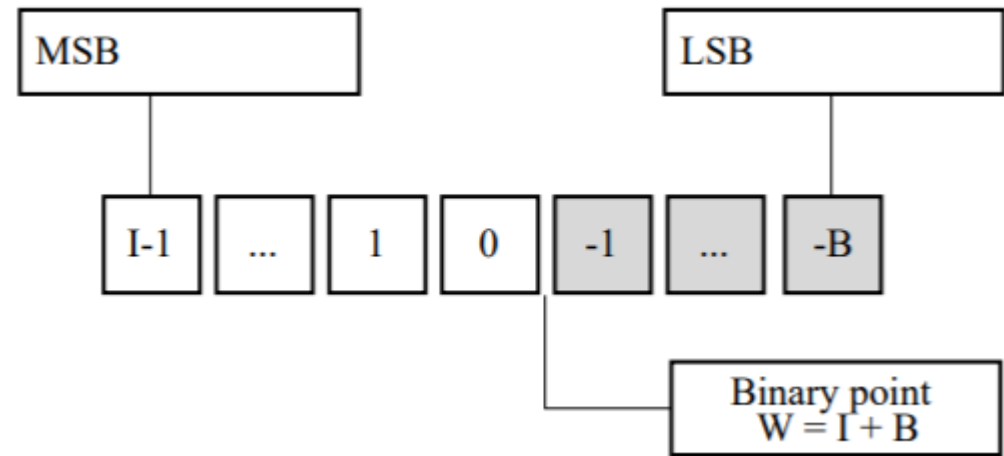
# Kernel Optimization – Data Types

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



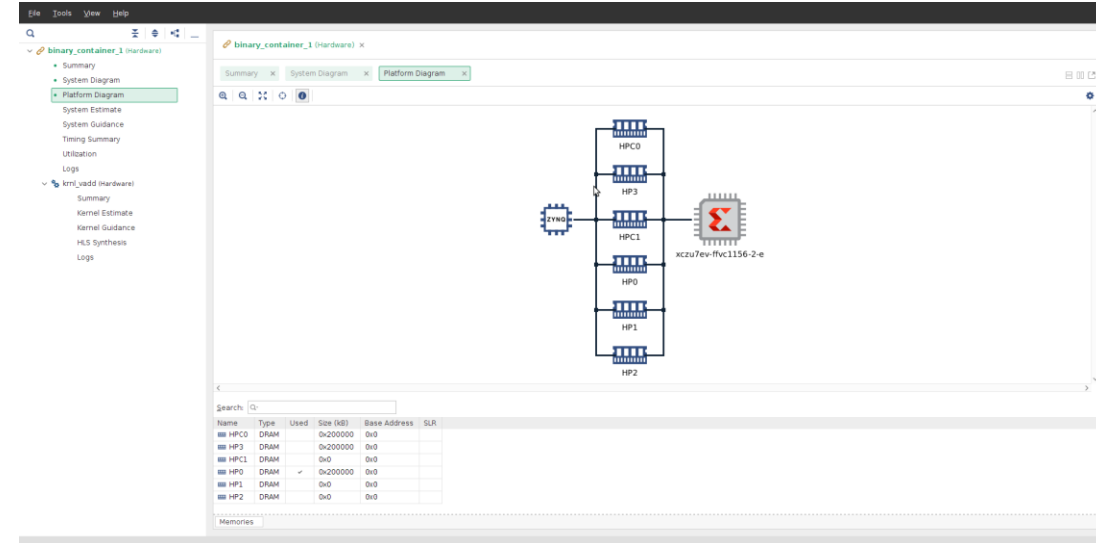
# Kernel Optimization – Interfacing

## Two types of data transfer

- » Data Pointers via global memory (M\_AXI)
- » Scalar direct to kernel (AXI\_LITE)

# Vitis automatically selects interface type

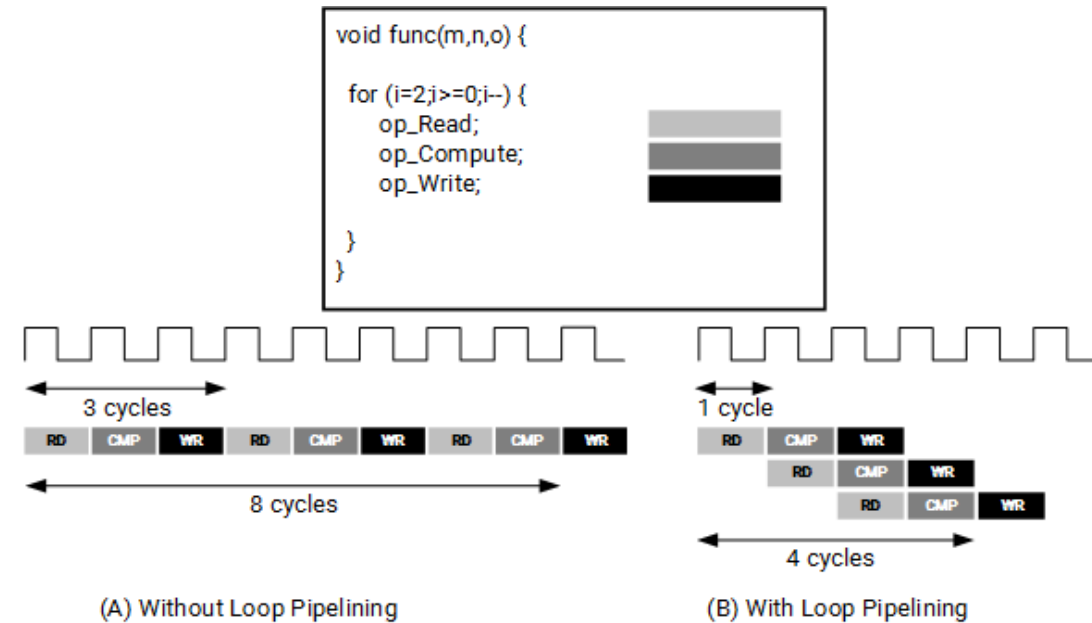
Max data width is 512 bits – maximum performance leverages this



# 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



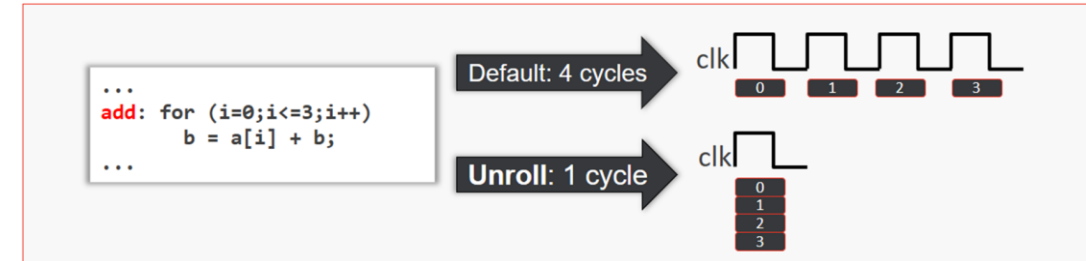
X14277-110217

# 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



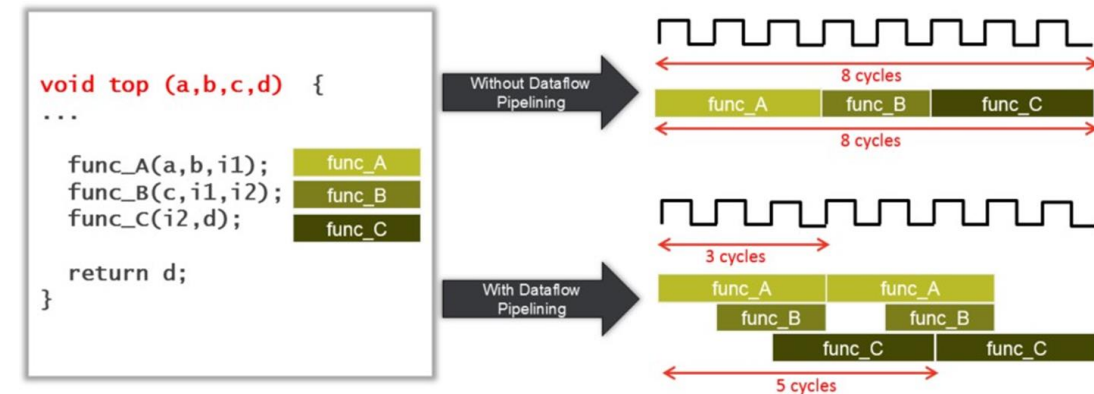
# Kernel Optimization – DataFlow

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





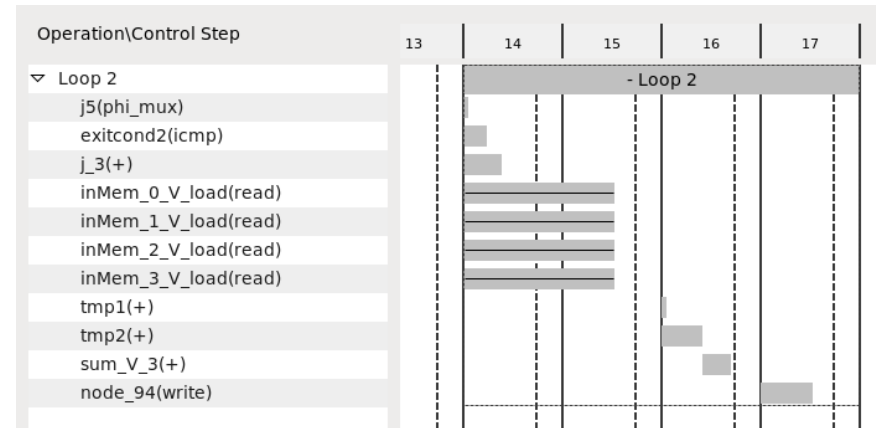
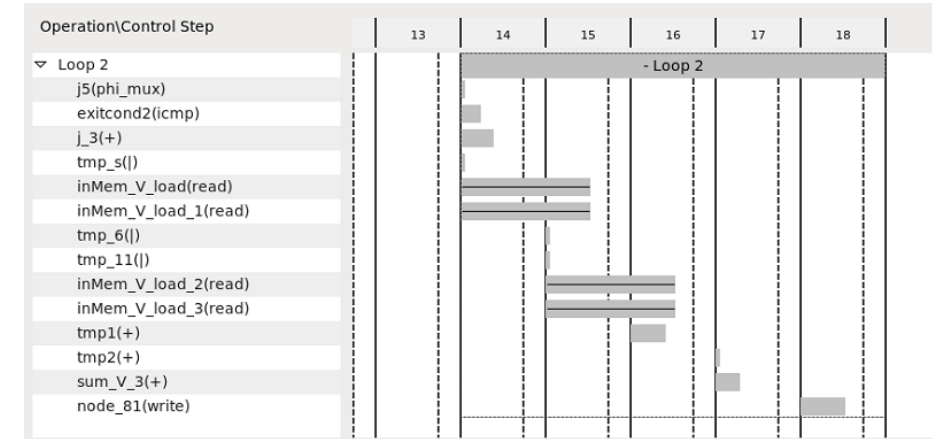
# Kernel Optimization – Memory

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



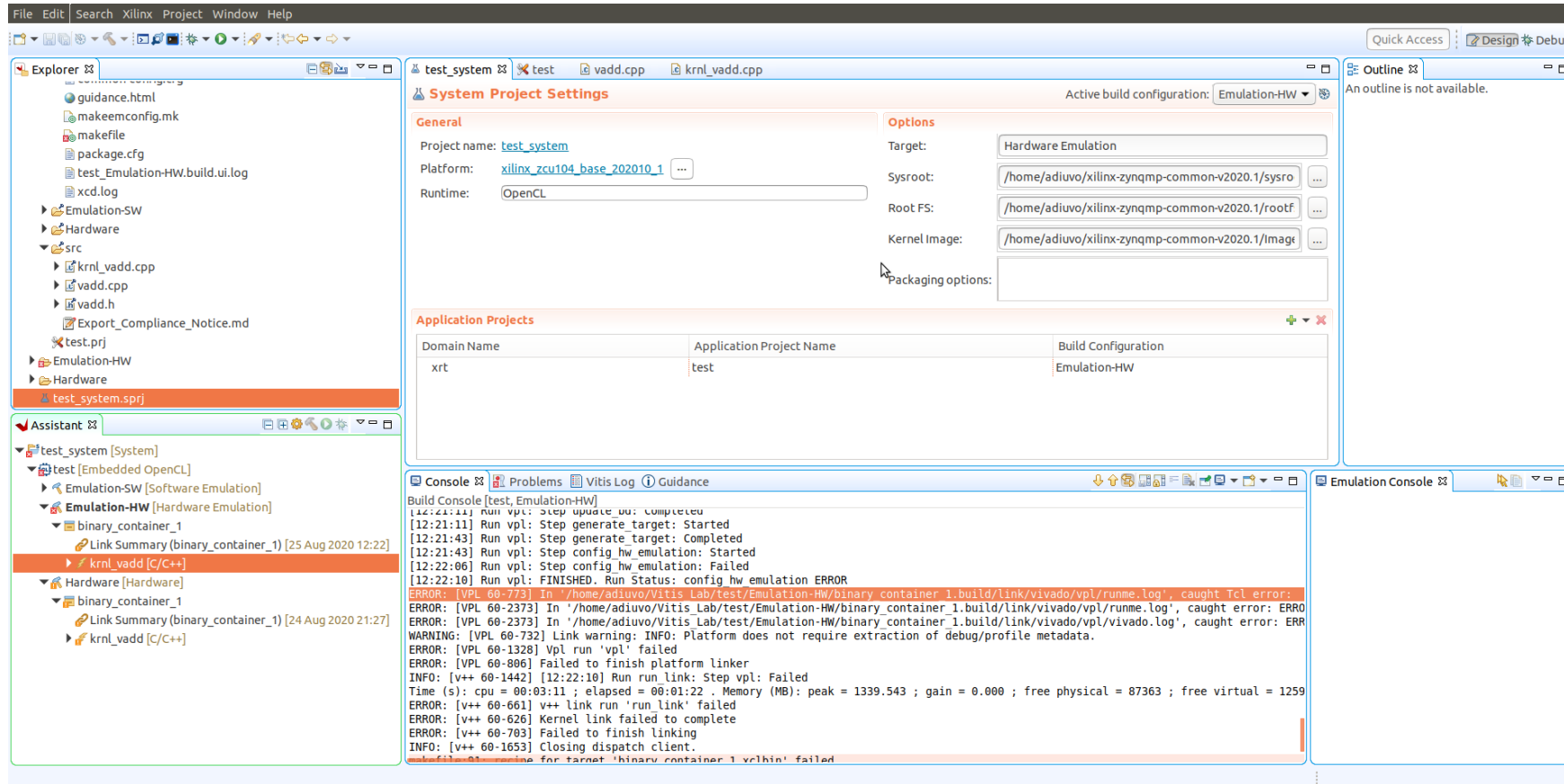
# Kernel Optimization - Pragmas

Optimization	C/C++	OpenCL
Pipeline	#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	

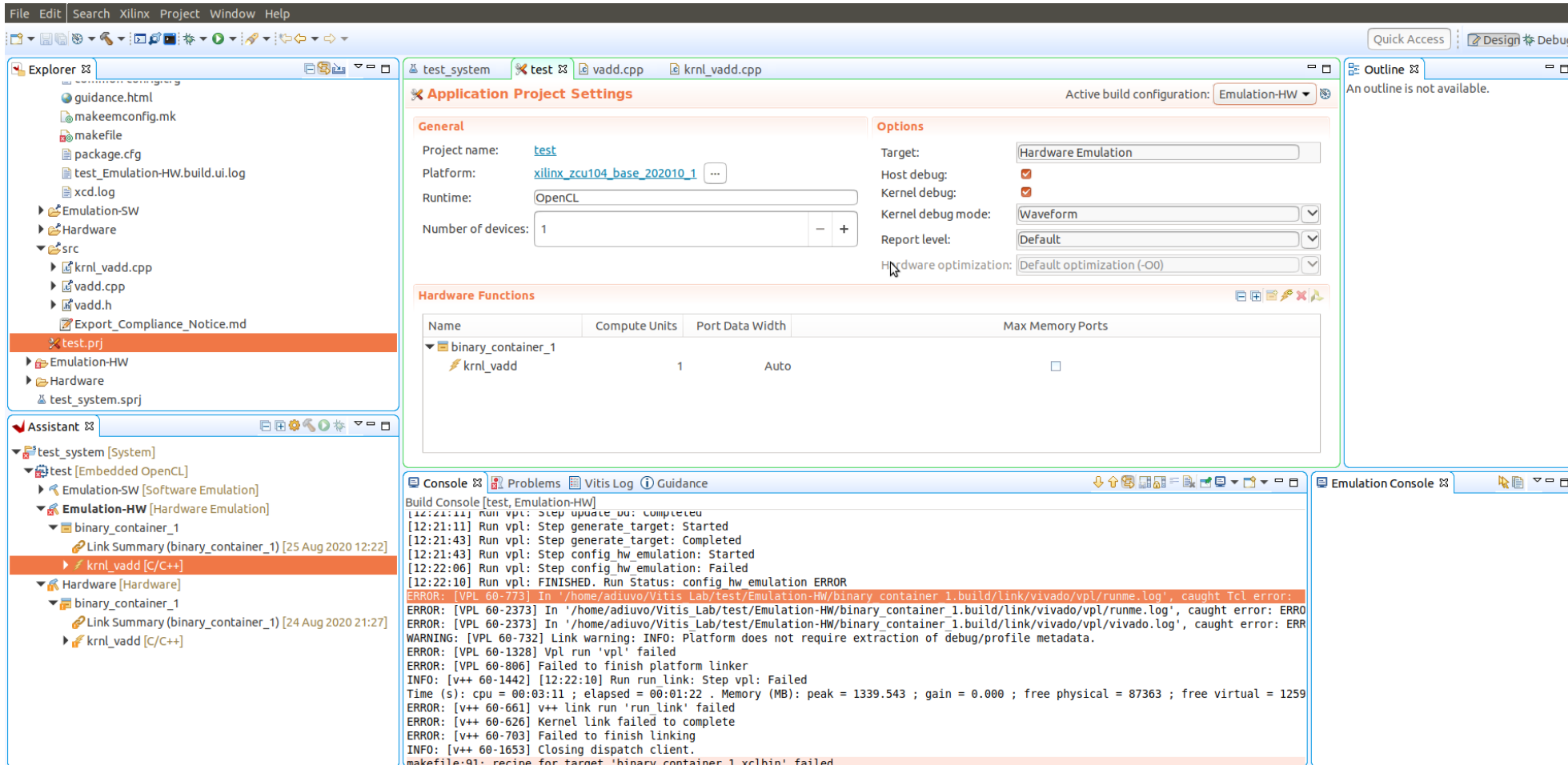
Further information can be found at

[https://www.xilinx.com/html\\_docs/xilinx2020\\_1/vitis\\_doc/optimizingperformance.html#the1553474153030](https://www.xilinx.com/html_docs/xilinx2020_1/vitis_doc/optimizingperformance.html#the1553474153030)

# Vitis GUI – Project Settings



# Vitis GUI – Project Setting



The screenshot displays the Vitis GUI interface for configuring a project named 'test'. The interface is divided into several panels:

- Explorer:** Shows the project file structure, including 'test.prj', 'Emulation-HW', 'Hardware', and 'test\_system.sprj'.
- Assistant:** Lists the project components, including 'test [Embedded OpenCL]', 'Emulation-SW [Software Emulation]', 'Emulation-HW [Hardware Emulation]', and 'binary\_container\_1'.
- Application Project Settings:**
  - General:** Project name: test, Platform: xilinx\_zcu104\_base\_202010\_1, Runtime: OpenCL, Number of devices: 1.
  - Options:** Target: Hardware Emulation, Host debug: checked, Kernel debug: checked, Kernel debug mode: Waveform, Report level: Default, Hardware optimization: Default optimization (-O0).
  - Hardware Functions:** A table showing the configuration for 'binary\_container\_1' and 'krnl\_vadd'.
- Console:** Displays the build log, showing the progress of the build process and various error messages.
- Emulation Console:** Shows the emulation status and output.

The console output includes the following messages:

```
Build Console [test, Emulation-HW]
[12:21:11] Run vpl: Step update_vpl: 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: ERR
ERROR: [VPL 60-2373] In '/home/adiuvo/Vitis_Lab/test/Emulation-HW/binary_container_1.build/link/vivado/vpl/vivado.log', caught error: ERR
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.xclbin' failed
```

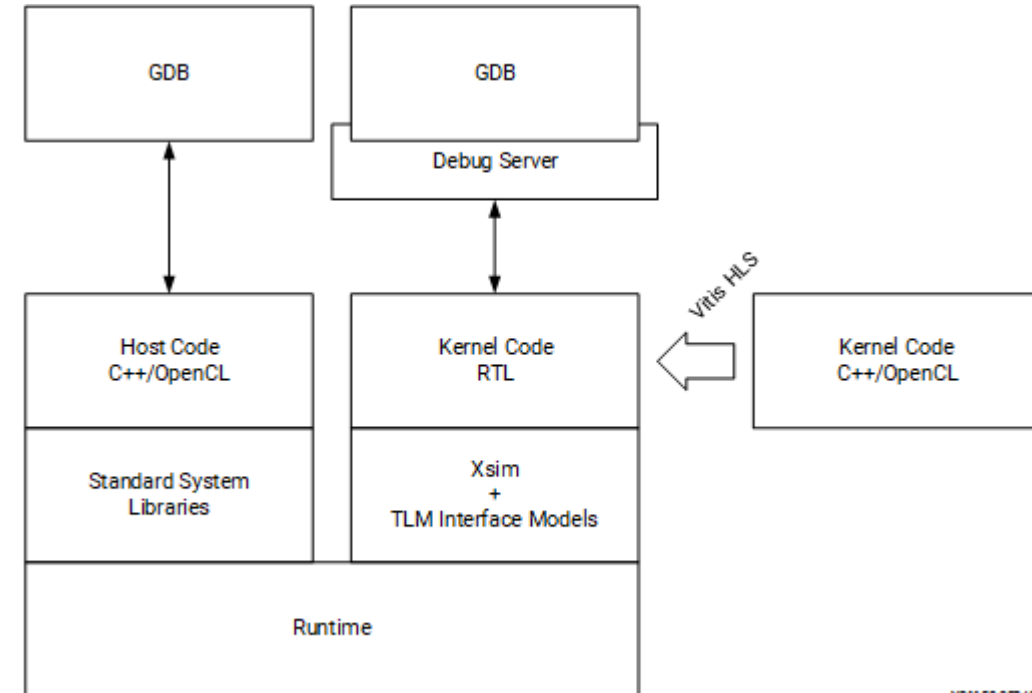
# Vitis-Debug

## Can Debug

- » Software Emulation
- » Hardware Emulation

Hardware flow insert ILA

Debugging will use QEMU and Logic Simulator





# ADIUVO

ENGINEERING AND TRAINING, LTD.

[www.adiuvoengineering.com](http://www.adiuvoengineering.com)



[adam@adiuvoengineering.com](mailto:adam@adiuvoengineering.com)