# COMPUTE OFFLOAD ACCELERATION WITH FPGA

IXPUG 2019 – Tutorial
CERN September, 27th 2019

francisco.perez@intel.com

# Agenda

Introduction to FPGA Acceleration Stack

Software and board installation

- Walkthrough on bring-up test on the server

What is an Accelerator Functional Unit (AFU)

Application Development using the Acceleration Stack

AFU development using High Level Synthesis (C/C++)

- Introduction to HLS tools
- HLS Interfaces
- HLS AFU development flow

# INTRODUCTION TO INTEL® PROGRAMABLE ACCELERATION STACK

# The Big Data Problem

HOW'S THE BIG DATA PROJECT COMING ALONG, HOSKINS?
© D.Fletcher for CloudTweaks.com

We are generating data at a faster rate than our ability to analyze, understand, transmit, secure and reconstruct in real-time

Not enough compute power, storage or infrastructure to compute in real time with a reasonable TCO

This creates an immense demand for compute architectures that can scale up and out exponentially

# Focused investments to accelerate HPC & AI

## ADVANCED ARCHITECTURES

intel XEON PLATINUM inside — SCALAR

X<sup>e</sup> — VECTOR

intel NERVANA inside · intel MOVIDIUS inside — MATRIX

intel STRATIX inside · intel AGILEX inside — SPATIAL

+ Quantum & Neuromorphic

COMPUTE ARCHITECTURES
FOR ALL YOUR WORKLOADS

## INTELLIGENT INTERCONNECT

OPEN STANDARDS

CXL Compute Express Link

ADVANCED HIGH
PERFORMANT FABRICS

INTERCONNECT BEYOND "I/O"

## SIMPLIFIED PROGRAMMING

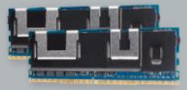
OPEN STANDARDS

oneAPI
Services and Tools

UNIFIED SINGLE SOFTWARE ABSTRACTION
AND DOMAIN-SPECIFIC LIBRARIES

## TRANSFORMING MEMORY & STORAGE

OPEN STANDARDS



OPTANE

DAOS

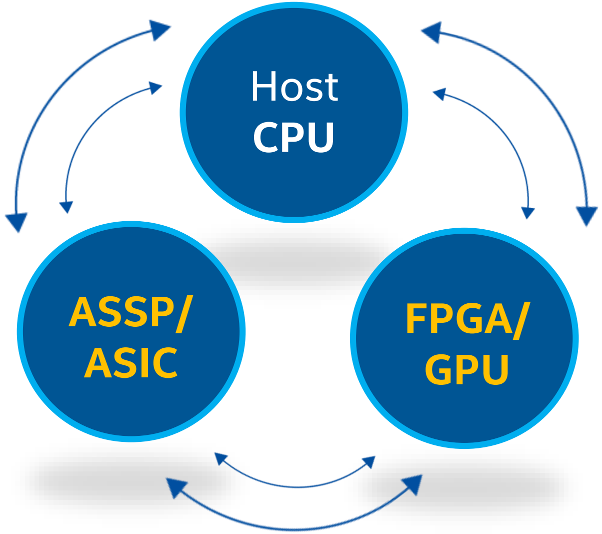
RE-ARCHITECTING THE MEMORY
HIERARCHY AND FILE SYSTEMS

**INTEL IS BUILDING THE HARDWARE, SOFTWARE, INTERCONNECT, MEMORY AND SECURITY
ARCHITECTURES NEEDED TO ENABLE YOUR TOMORROW'S APPLICATIONS**

# Acceleration Choices

**CPUs to become more powerful and efficient but certain applications will still require a hardware accelerator.**

**DEDICATED ACCELERATORS** for maximum compute efficiency of specific, fixed functions



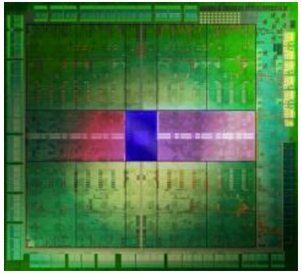
**VERSATILE ACCELERATORS** for customized and changing workloads in networking, storage, and compute

# The Intel Vision

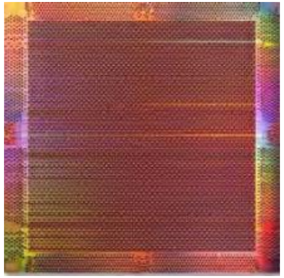
## Heterogeneous Systems:

- Span from CPU to GPU to FPGA to dedicated devices with consistent programming models, languages, and tools

CPUs



GPUs



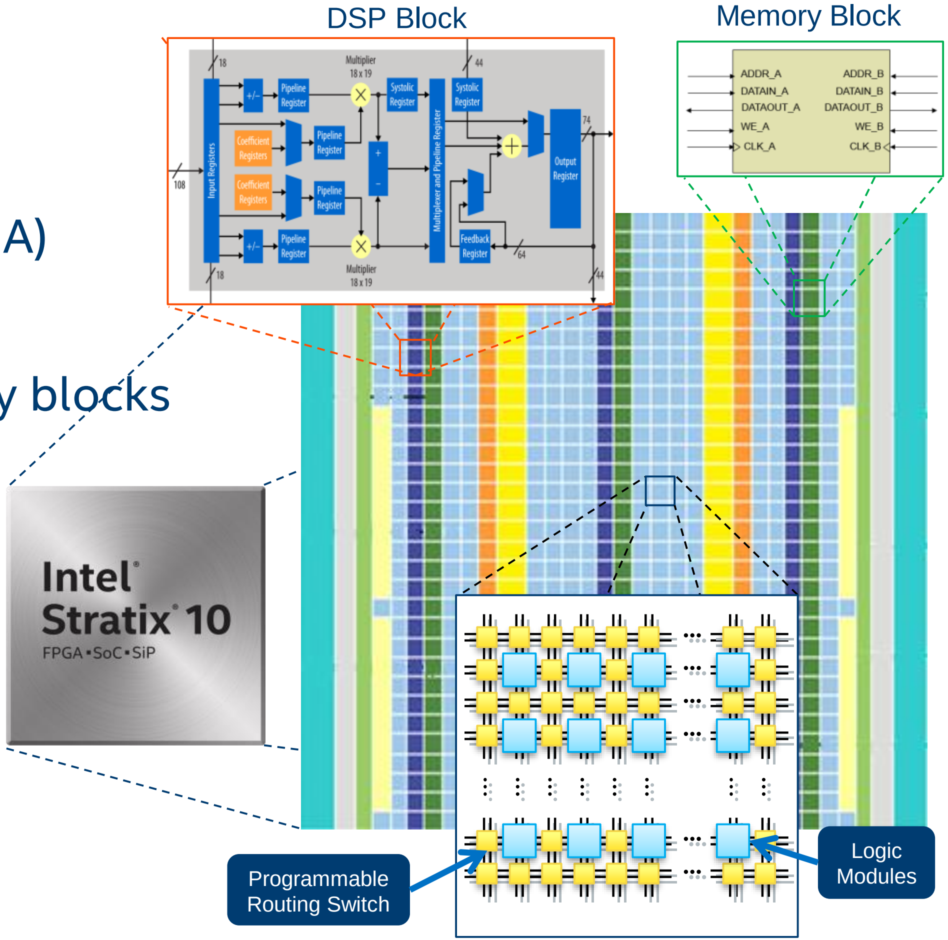
FPGAs

ASSP

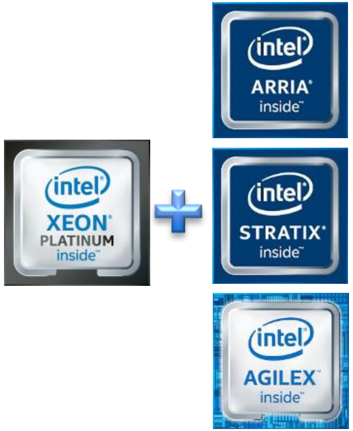
**FPGAs are the focus of today**

# What is a FPGA?

- Field Programmable Gate Array (FPGA)
  - Millions of logic elements
  - Thousands of embedded memory blocks
  - Thousands of DSP blocks
  - Programmable routing
  - High speed transceivers
  - Various built-in hardened IP
- Programmable interconnect
- Used to create **Custom Hardware!**



# How Do Intel® FPGAs Help to Solve the Problem?



## Workload Optimization:

ensure Xeon cores serve their highest value processing
FPGA focus on intensive tasks

Workload 1
Workload 2
Workload N

## Efficient Performance:

improve performance/watt
Custom hardware tailored

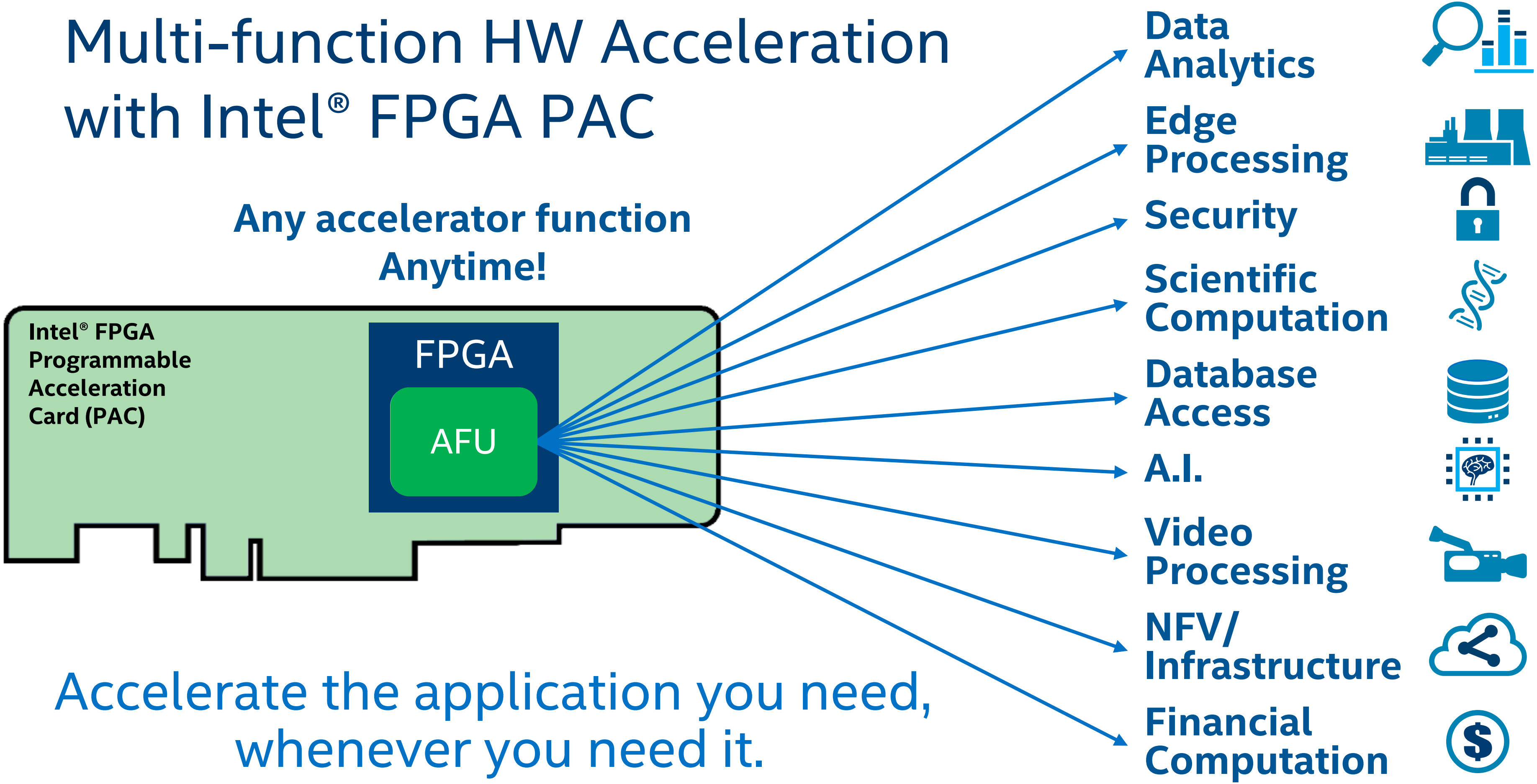
## Real-Time:

high bandwidth connectivity and low-latency parallel processing
In-line data streaming

Milliseconds

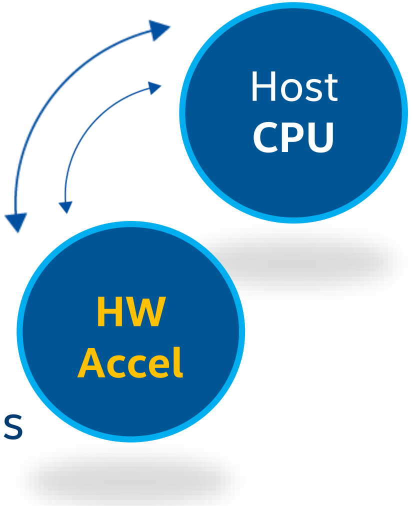
# Multi-function HW Acceleration with Intel® FPGA PAC

**Any accelerator function Anytime!**



Accelerate the application you need, whenever you need it.

# Separation of concerns



Two groups of developers:

- Domain experts concerned with getting a result
  - Host application developers leveraging optimized libraries
- Tuning experts concerned with performance
  - Typical FPGA developers that create optimized libraries

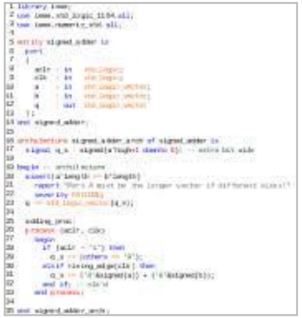
Intel® Math Kernel Library a simple example of raising the level of abstraction to the math operations

- Domain experts focus on formulating their problems
- Tuning experts focus on vectorization and parallelization

# Traditional FPGA Design and Use is *"Difficult"*

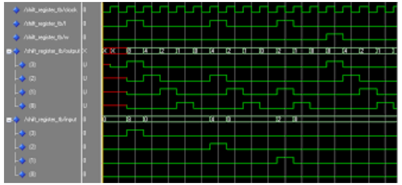
## Low level hardware design requires complicated, long, time-consuming efforts

Hardware Description Languages

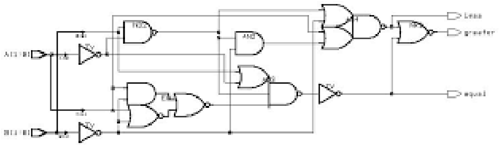


```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity signed_adder is
  port
  (
    aclr : in   std_logic;
    clk  : in   std_logic;
    a    : in   std_logic_vector;
    b    : in   std_logic_vector;
    q    : out  std_logic_vector
  );
end signed_adder;

architecture signed_adder_arch of signed_adder is
  signal q_s : signed(a'high+1 downto 0); -- extra bit wide

begin -- architecture
  assert(a'length >= b'length)
    report "Port A must be the longer vector if different sizes!"
    severity FAILURE;
  q <= std_logic_vector(q_s);

  adding_proc:
  process (aclr, clk)
    begin
      if (aclr = '1') then
        q_s <= (others => '0');
      elsif rising_edge(clk) then
        q_s <= ('0'&signed(a)) + ('0'&signed(b));
      end if; -- clk'd
    end process;

end signed_adder_arch;
```

Behavioral Simulation



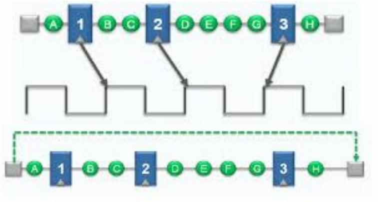
Synthesis



Place & Route / Timing Analysis / Timing Closure

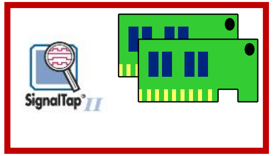
Intel® Quartus® Prime

Design Software



Intel STRATIX® inside™

Board Simulation & Test



# Software Developers are the New FPGA Developers

"I don't speak FPGA!

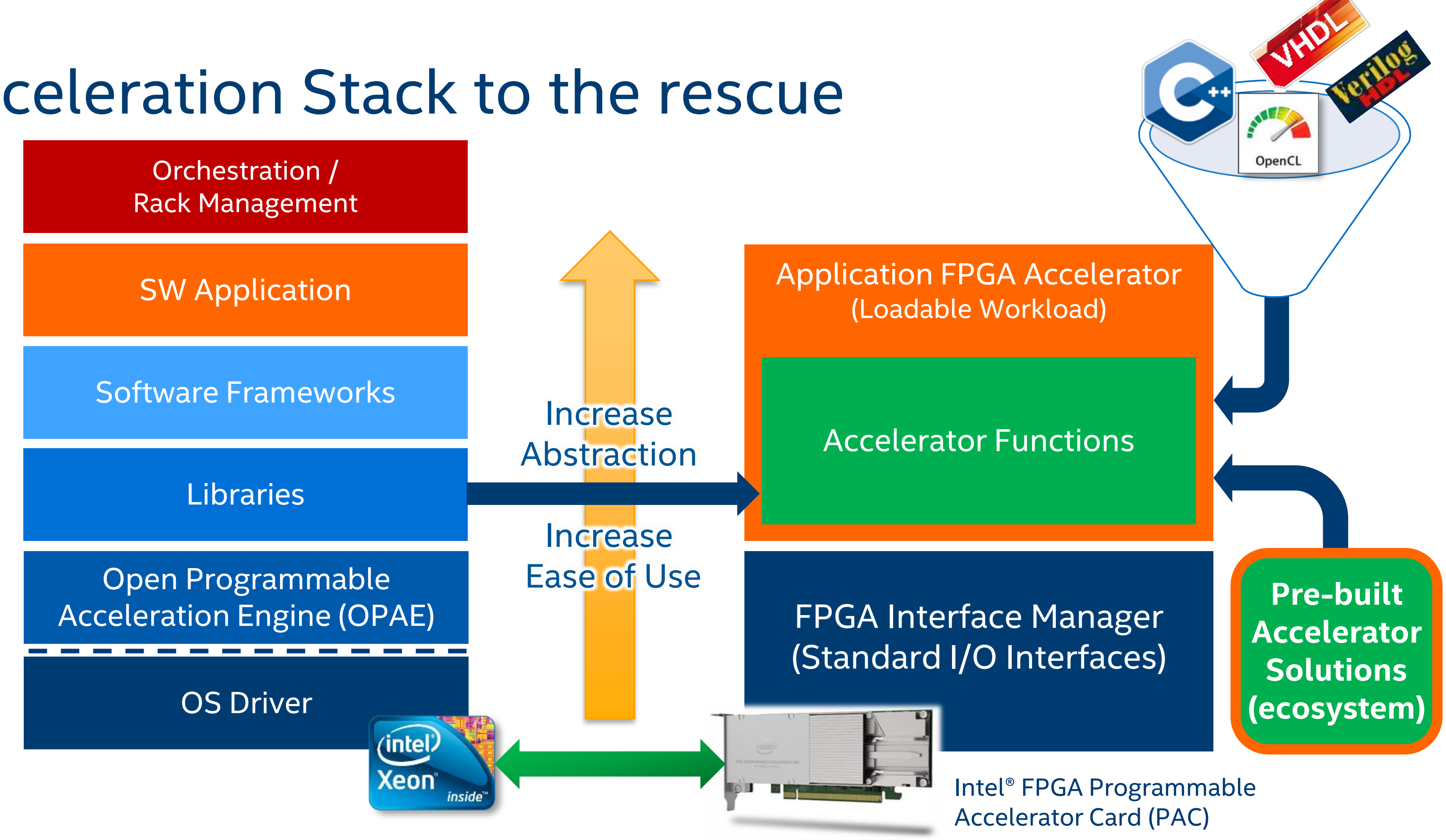
What is the programming model, and where are the compilers, libraries and tools I am used to?"

New use case of FPGAs as software-defined hardware and the benefits as accelerators

→

Opens up the usage for a much larger developer base

# Acceleration Stack to the rescue



Orchestration / Rack Management

SW Application

Software Frameworks

Libraries

Open Programmable Acceleration Engine (OPAE)

OS Driver

Increase Abstraction

Increase Ease of Use

Application FPGA Accelerator (Loadable Workload)

Accelerator Functions

FPGA Interface Manager (Standard I/O Interfaces)

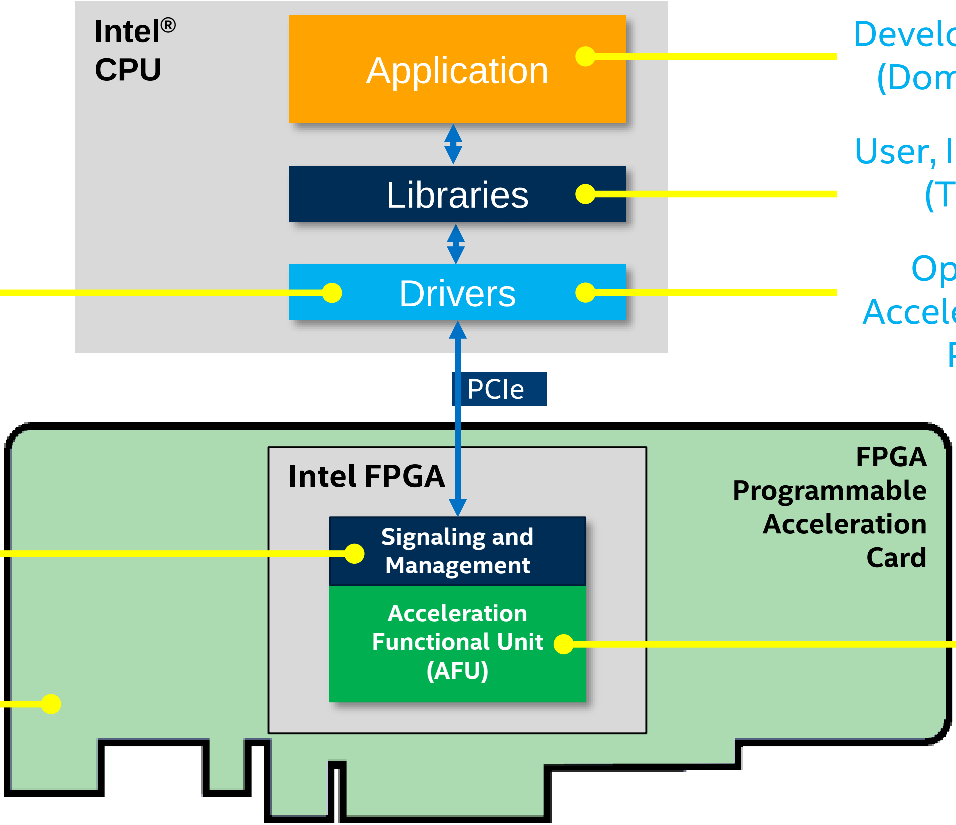
**Pre-built Accelerator Solutions (ecosystem)**

Intel® FPGA Programmable Accelerator Card (PAC)

* Other names and brands may be claimed as the property of others.

OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission by Khronos

# Components of Acceleration Stack: Overview



Intel® CPU

Application

Libraries

Drivers

PCIe

Intel FPGA

Signaling and Management

Acceleration Functional Unit (AFU)

FPGA Programmable Acceleration Card

Developed by User
(Domain Expert)

User, Intel, and 3rd Party
(Tuning Expert)

Open Programmable
Acceleration Engine (OPAE)
Provided by Intel

PCIe* Drivers
Provided by Intel

*FPGA Interface Manager*
Provided by Intel

User, Intel, or 3rd-Party IP
Plugs into AFU Slot
(Tuning Expert)

*Qualified and Validated for volume deployment*
Provided by OEMs

# The Challenge: Enabling the Performance & Capabilities of FPGA for Everyone

**Board Design & Qualification**

**Software Development**

**FPGA Accelerator Development**

**Intel® Investment in All These Areas Democratizes FPGA Acceleration**

# FPGA Acceleration Cards for datacenters

## Intel® FPGA Programmable Acceleration Cards for Application Acceleration

### Intel® FPGA PAC with Arria® 10 GX

***Broad deployment at low power***

*½ length, ½ height, 1 PCIe slot card*

*70W TDP*

### Intel® FPGA PAC with Stratix® 10 GX

***Enabling high throughput***

*¾ length, full height, dual PCIe slot card*

*225W TDP*

# Intel® FPGA Programmable Acceleration Card

## with Intel® Arria® 10 GX FPGA

Low power programmable acceleration platform with data center-grade software stack enabling in-line processing and memory intensive applications.

### Features

- *1.15 million logic elements*
- *DDR4 memory, 2 banks 4GB @2133Mbps*
- *53Mbit embedded memory*
- *4x10G / 1x40G QSFP*
- *PCIe* Gen 3 x8 (x16 mechanical)*
- *BMC for monitoring and control (PLDM)*
- *½ length, ½ height, 1slot PCIe* card*
- *70W TDP, 45W FPGA*
- *Acceleration Stack for Intel® Xeon® CPU with FPGAs*

* Other names and brands may be claimed as the property of others.

# Intel® FPGA Programmable Acceleration Card

## with Intel® Stratix® 10 GX FPGA

### Features

- *2.8 million logic elements*
- *32 Gb DDR4 DIMM memory (4x8GB, 2133Mbps)*
- *229 Mbit embedded memory*
- *2x 100G (4x25Gb) QSFP*
- *PCIe* Gen3 x16*
- *BMC for monitoring and control (PLDM)*
- *¾ length, full height, dual slot card*
- *225W TDP, 150W FPGA*
- *Acceleration Stack for Intel® Xeon® CPU with FPGAs*

intel
Intel® Programmable Acceleration Card
with Intel Stratix® 10 SX SoC FPGA

High bandwidth programmable acceleration platform with data center-grade software stack enabling in-line processing and memory intensive applications.

* Other names and brands may be claimed as the property of others.

Specifications preliminary and are subject to change

# Open Programmable Acceleration Engine (OPAE)

**Consistent API across product generations and platforms**
Abstraction for hardware specific FPGA resource details

**Designed for minimal software overhead and latency**
Lightweight user-space library *(libfpga)*

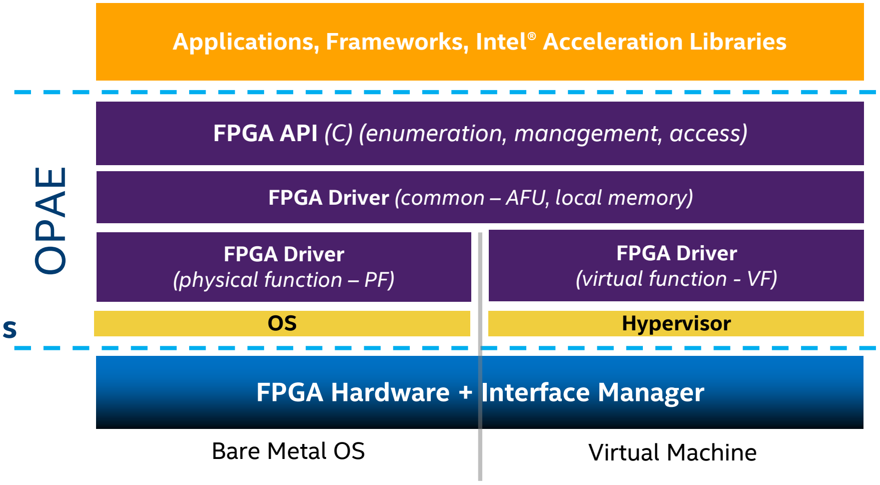
**Open ecosystem for industry and developer community**
FPGA driver being upstreamed into Linux kernel

**Supports both virtual machines and bare metal platforms**

**Faster development and debugging of Accelerator Functions with the included AFU Simulation Environment (ASE)**

**Includes guides, command-line utilities and sample code**

## Simplified FPGA Programming Model for Application Developers



Applications, Frameworks, Intel® Acceleration Libraries
OPAE
FPGA API (C) (enumeration, management, access)
FPGA Driver (common – AFU, local memory)
FPGA Driver (physical function – PF)
FPGA Driver (virtual function - VF)
OS
Hypervisor
FPGA Hardware + Interface Manager
Bare Metal OS
Virtual Machine

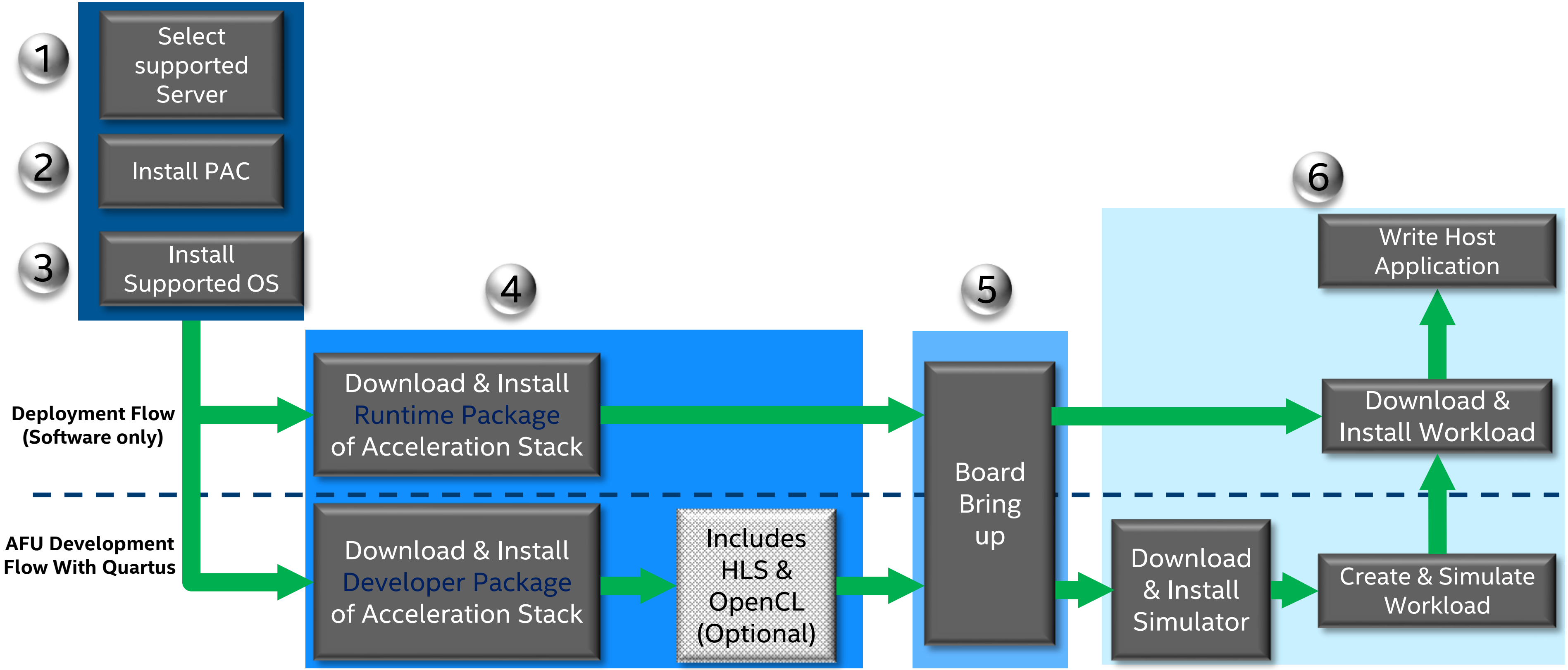
CentOS redhat ubuntu

Start developing for Intel FPGAs with OPAE today: http://01.org/OPAE

# INTEL® PROGRAMABLE ACCELERATION CARD AND STACK INSTALLATION

Step Guide

# Out-of-Box User Flow for Acceleration Stack


1
Select supported Server
2
Install PAC
3
Install Supported OS
4
5
6
Deployment Flow (Software only)
AFU Development Flow With Quartus
Download & Install Runtime Package of Acceleration Stack
Download & Install Developer Package of Acceleration Stack
Includes HLS & OpenCL (Optional)
Board Bring up
Download & Install Workload
Write Host Application
Download & Install Simulator
Create & Simulate Workload

Runtime package includes only OPAE drivers and sample AFU, Developer package includes Quartus + IP Lic + drivers

1

# Select Supported Server

| OEM | Dell | Fujitsu | HPE | Inspur | Quanta | Kontron | Supermicro |
|---|---|---|---|---|---|---|---|
| **Status** | **Qualified** | **Qualified** | **Qualified** | **Qualified*** | **Qualified** | **Ongoing** | **Qualified*** |
| **Servers Supported** | R640<br>R740<br>R740xd<br>R840<br>R940xa<br> | RX2540<br>TX2550<br> | ProLiant<br>DL360<br>DL380<br> | 5280M5<br> | QuantaGrid<br>D52BQ-1U<br>D52BQ-2U<br>QuantaVault<br>JG4080<br> | Symkloud<br>MS2900<br> | Sys-1029U<br>Sys-2029U<br>Sys-6019U<br>Sys-6029U<br> |

Customers can deploy on their servers of choice following:

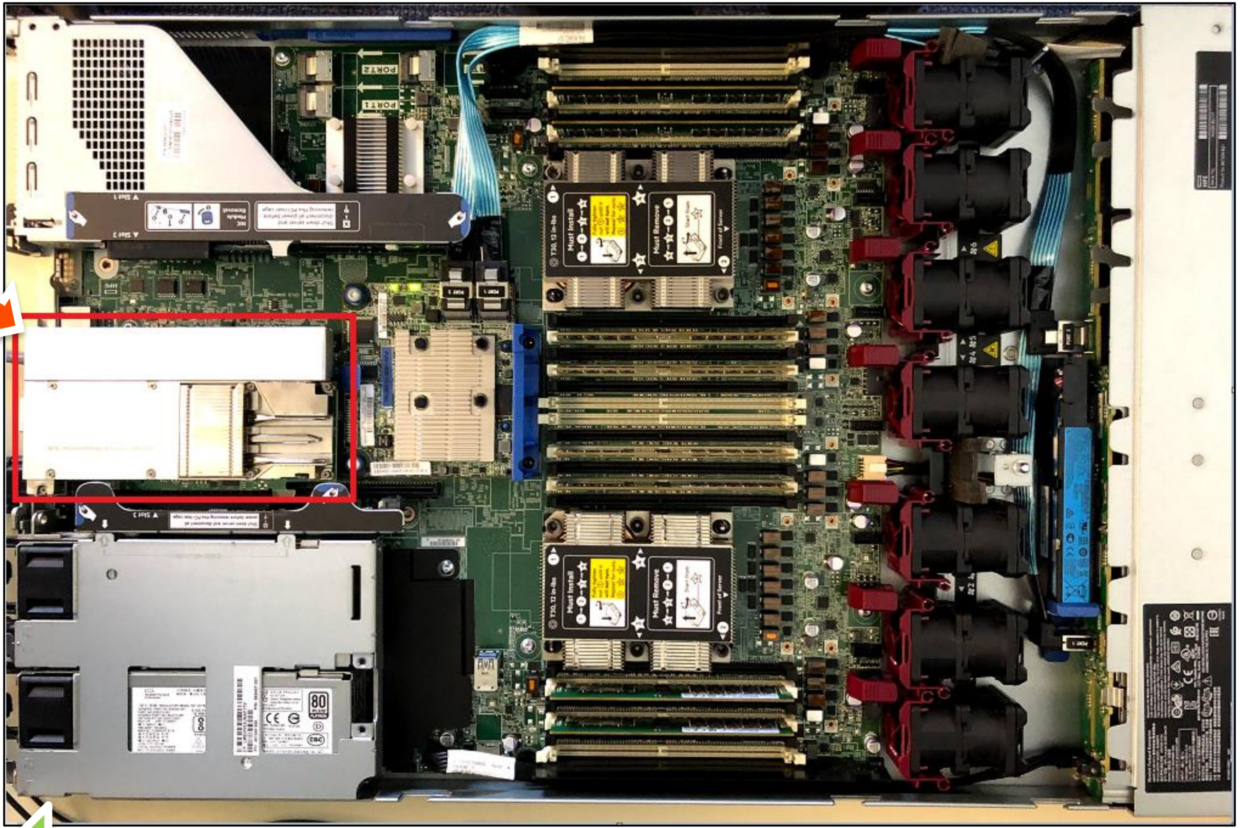
**Intel Programmable Acceleration Card Platform Qualification Guidelines***

*Available on request

2

# Install PAC – Arria10 PAC in HPE DL360

Back



Front

Air Flow

3

# Install Supported OS

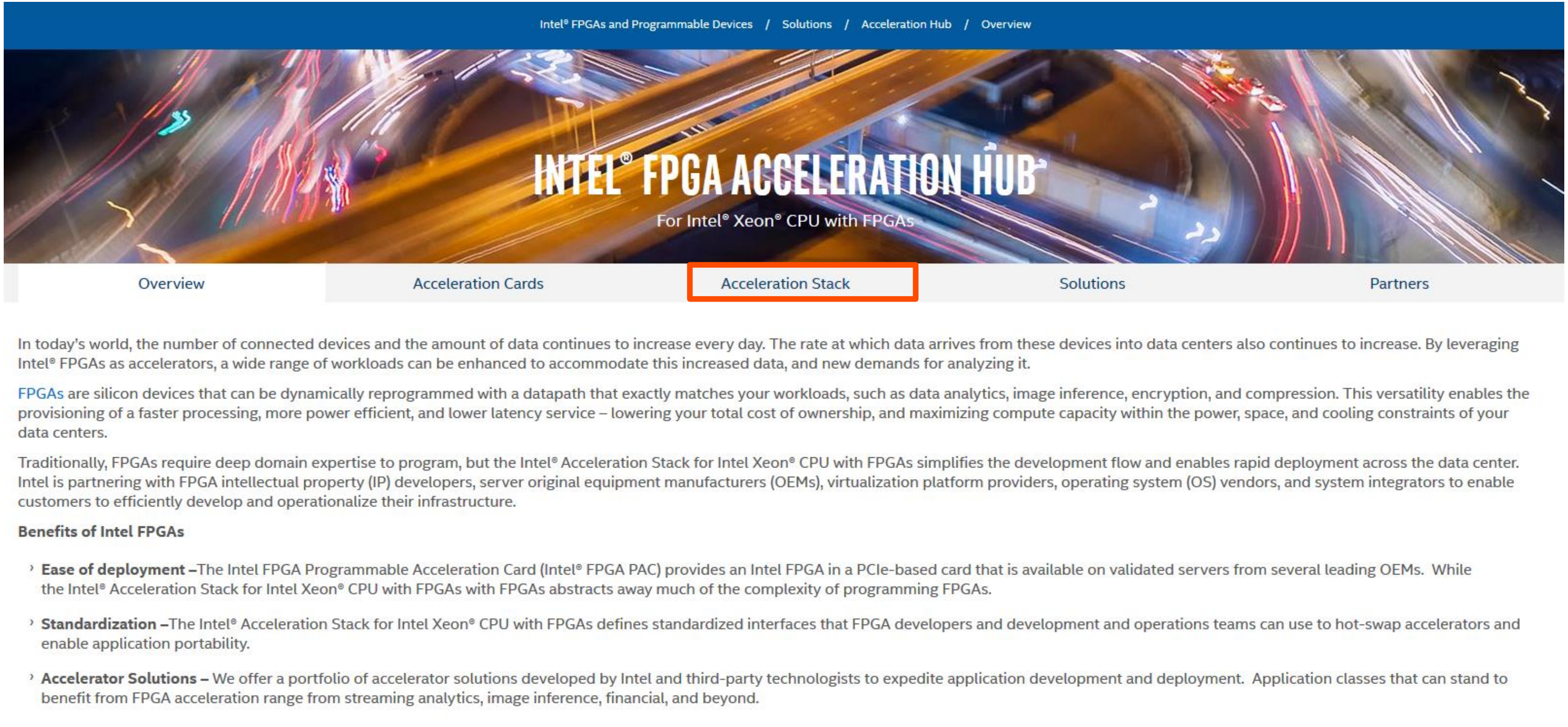
## Acceleration Stack v1.2 validated OS

- RHEL kernel 3.10 (v7.4 & 7.6)
- CentOS kernel 3.10 (v7.4 & 7.6)
- Ubuntu kernel 4.4 (v16.04)

4

# Download Acceleration Stack

https://www.intel.com/content/www/us/en/programmable/solutions/acceleration-hub/overview.html


Intel® FPGAs and Programmable Devices / Solutions / Acceleration Hub / Overview
INTEL® FPGA ACCELERATION HUB
For Intel® Xeon® CPU with FPGAs
Overview
Acceleration Cards
Acceleration Stack
Solutions
Partners

In today's world, the number of connected devices and the amount of data continues to increase every day. The rate at which data arrives from these devices into data centers also continues to increase. By leveraging Intel® FPGAs as accelerators, a wide range of workloads can be enhanced to accommodate this increased data, and new demands for analyzing it.

FPGAs are silicon devices that can be dynamically reprogrammed with a datapath that exactly matches your workloads, such as data analytics, image inference, encryption, and compression. This versatility enables the provisioning of a faster processing, more power efficient, and lower latency service – lowering your total cost of ownership, and maximizing compute capacity within the power, space, and cooling constraints of your data centers.

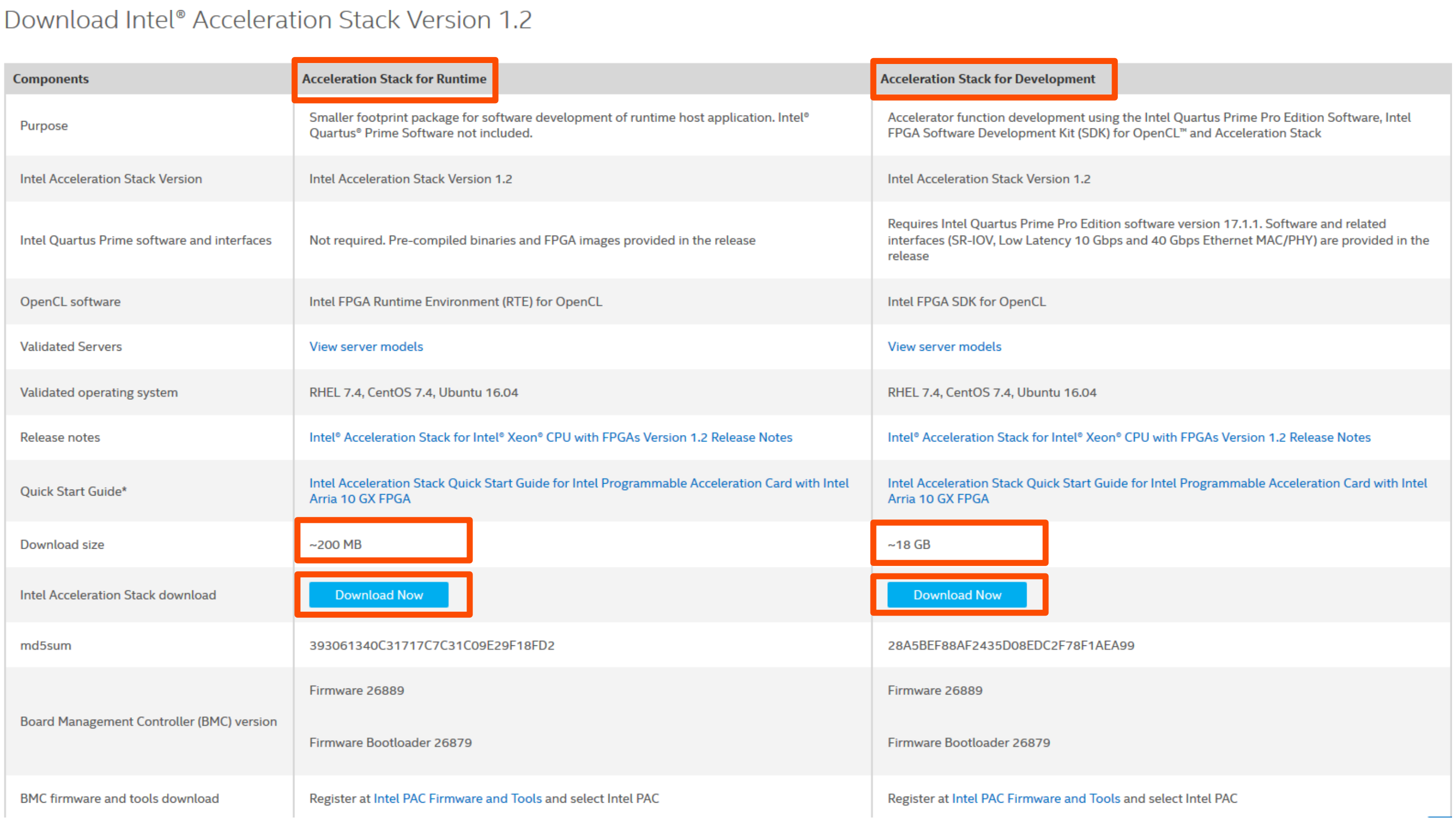
Traditionally, FPGAs require deep domain expertise to program, but the Intel® Acceleration Stack for Intel Xeon® CPU with FPGAs simplifies the development flow and enables rapid deployment across the data center. Intel is partnering with FPGA intellectual property (IP) developers, server original equipment manufacturers (OEMs), virtualization platform providers, operating system (OS) vendors, and system integrators to enable customers to efficiently develop and operationalize their infrastructure.

**Benefits of Intel FPGAs**

- **Ease of deployment –**The Intel FPGA Programmable Acceleration Card (Intel® FPGA PAC) provides an Intel FPGA in a PCIe-based card that is available on validated servers from several leading OEMs. While the Intel® Acceleration Stack for Intel Xeon® CPU with FPGAs with FPGAs abstracts away much of the complexity of programming FPGAs.
- **Standardization –**The Intel® Acceleration Stack for Intel Xeon® CPU with FPGAs defines standardized interfaces that FPGA developers and development and operations teams can use to hot-swap accelerators and enable application portability.
- **Accelerator Solutions –** We offer a portfolio of accelerator solutions developed by Intel and third-party technologists to expedite application development and deployment. Application classes that can stand to benefit from FPGA acceleration range from streaming analytics, image inference, financial, and beyond.

4

# Download Intel® Acceleration Stack



## Download Intel® Acceleration Stack Version 1.2

| Components | Acceleration Stack for Runtime | Acceleration Stack for Development |
|---|---|---|
| Purpose | Smaller footprint package for software development of runtime host application. Intel® Quartus® Prime Software not included. | Accelerator function development using the Intel Quartus Prime Pro Edition Software, Intel FPGA Software Development Kit (SDK) for OpenCL™ and Acceleration Stack |
| Intel Acceleration Stack Version | Intel Acceleration Stack Version 1.2 | Intel Acceleration Stack Version 1.2 |
| Intel Quartus Prime software and interfaces | Not required. Pre-compiled binaries and FPGA images provided in the release | Requires Intel Quartus Prime Pro Edition software version 17.1.1. Software and related interfaces (SR-IOV, Low Latency 10 Gbps and 40 Gbps Ethernet MAC/PHY) are provided in the release |
| OpenCL software | Intel FPGA Runtime Environment (RTE) for OpenCL | Intel FPGA SDK for OpenCL |
| Validated Servers | View server models | View server models |
| Validated operating system | RHEL 7.4, CentOS 7.4, Ubuntu 16.04 | RHEL 7.4, CentOS 7.4, Ubuntu 16.04 |
| Release notes | Intel® Acceleration Stack for Intel® Xeon® CPU with FPGAs Version 1.2 Release Notes | Intel® Acceleration Stack for Intel® Xeon® CPU with FPGAs Version 1.2 Release Notes |
| Quick Start Guide* | Intel Acceleration Stack Quick Start Guide for Intel Programmable Acceleration Card with Intel Arria 10 GX FPGA | Intel Acceleration Stack Quick Start Guide for Intel Programmable Acceleration Card with Intel Arria 10 GX FPGA |
| Download size | ~200 MB | ~18 GB |
| Intel Acceleration Stack download | Download Now | Download Now |
| md5sum | 393061340C31717C7C31C09E29F18FD2 | 28A5BEF88AF2435D08EDC2F78F1AEA99 |
| Board Management Controller (BMC) version | Firmware 26889<br>Firmware Bootloader 26879 | Firmware 26889<br>Firmware Bootloader 26879 |
| BMC firmware and tools download | Register at Intel PAC Firmware and Tools and select Intel PAC | Register at Intel PAC Firmware and Tools and select Intel PAC |

# AFU Development Software Requirements

Acceleration Stack SDK (all licenses included in development package)

- Quartus Prime Pro Software 17.1.1 for v1.2 Acceleration Stack, 18.0.1 for v2.0
- IP-PCIE/SRIOV License
- Low Latency 10Gbps Ethernet MAC(6AF7-0119) license
- Low Latency 40Gbps Ethernet MAC and PHY(6AF7-011B) license

python2-jsonschema package from the epel repository (version 2.7 or higher)

GCC – C compiler version 4.7 or greater

RTL Simulator

- Synopsys VCS-MX version 2016.06-SP2-1
- 64-bit ModelSim SE or QuestaSim version 10.5c or higher

4

From the Quick Start Guide

# Installing the Intel® Acceleration Stack

1. Extract the archive file:

```
tar xvf *rte_installer.tar.gz
```

or

```
tar xvf *dev_installer.tar.gz
```

2. Change to the installation directory.

```
cd *rte_installer
```

or

```
cd *dev_installer
```

3. Install Extra packages for Enterprise Linux (EPEL) for RHEL 7.4 only

```
sudo yum install https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
sudo subscription-manager repos --enable "rhel-*-optional-rpms“ --enable "rhel-*-extras-rpms“
```

4. Run setup script

```
./setup.sh
```

5. Run the initialization script from the installation directory to setup environment variables

```
source /home/<username>/intelrtestack/init_env.sh
```

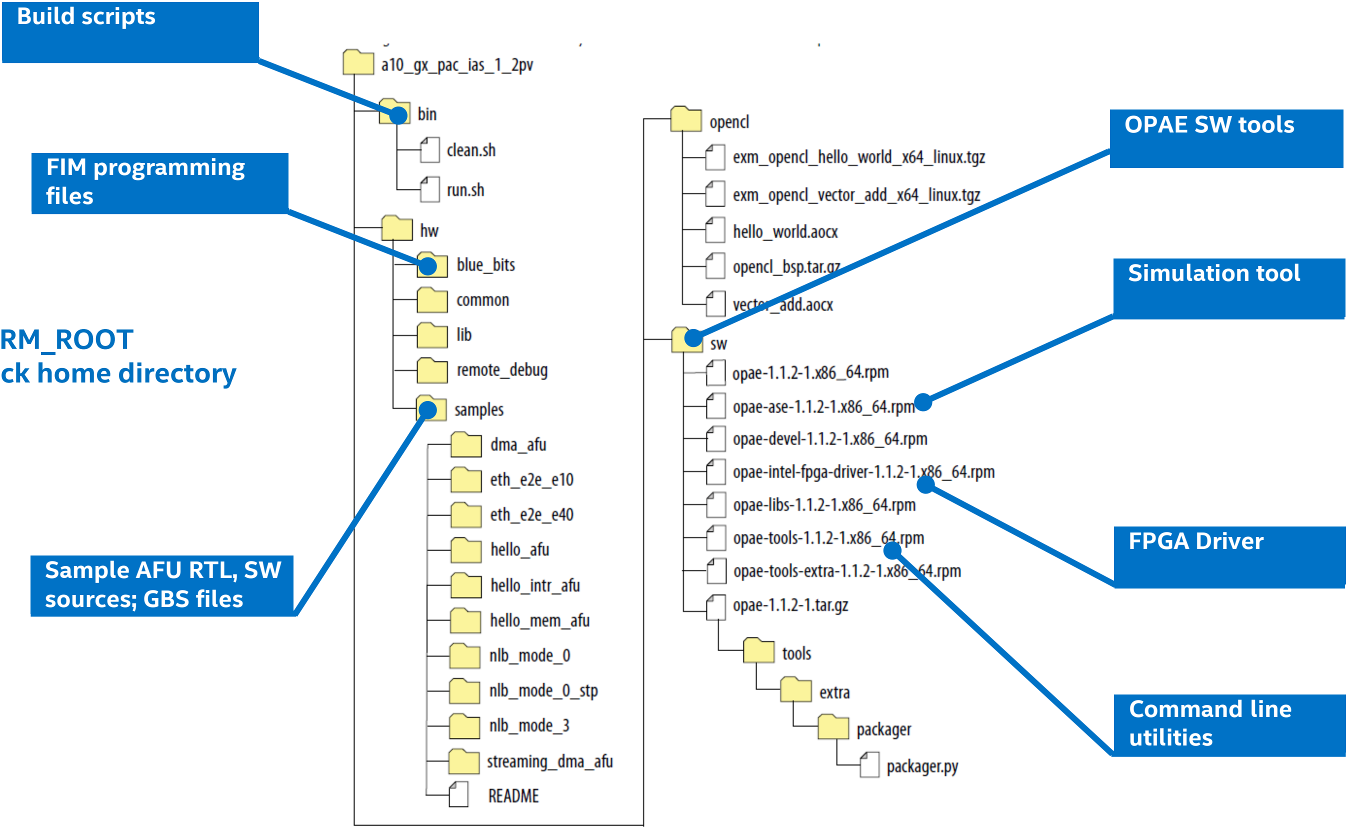
or

```
source /home/<username>/inteldevstack/init_env.sh
```

4

# Intel® Acceleration Stack Directory Structure

$OPAE_PLATFORM_ROOT
Acceleration Stack home directory



Build scripts

FIM programming files

Sample AFU RTL, SW sources; GBS files

OPAE SW tools

Simulation tool

FPGA Driver

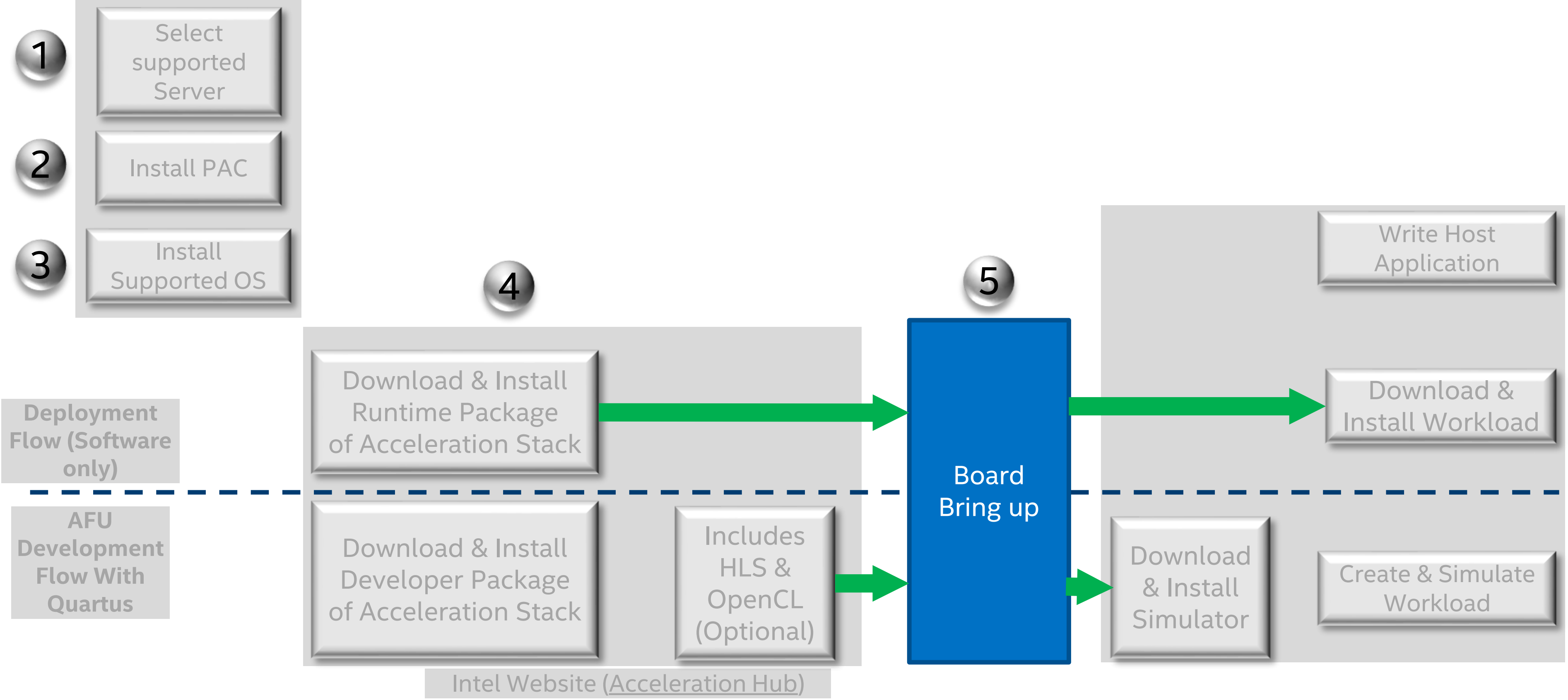
Command line utilities

```
a10_gx_pac_ias_1_2pv
├── bin
│   ├── clean.sh
│   └── run.sh
├── hw
│   ├── blue_bits
│   ├── common
│   ├── lib
│   ├── remote_debug
│   └── samples
│       ├── dma_afu
│       ├── eth_e2e_e10
│       ├── eth_e2e_e40
│       ├── hello_afu
│       ├── hello_intr_afu
│       ├── hello_mem_afu
│       ├── nlb_mode_0
│       ├── nlb_mode_0_stp
│       ├── nlb_mode_3
│       ├── streaming_dma_afu
│       └── README
├── opencl
│   ├── exm_opencl_hello_world_x64_linux.tgz
│   ├── exm_opencl_vector_add_x64_linux.tgz
│   ├── hello_world.aocx
│   ├── opencl_bsp.tar.gz
│   └── vector_add.aocx
└── sw
    ├── opae-1.1.2-1.x86_64.rpm
    ├── opae-ase-1.1.2-1.x86_64.rpm
    ├── opae-devel-1.1.2-1.x86_64.rpm
    ├── opae-intel-fpga-driver-1.1.2-1.x86_64.rpm
    ├── opae-libs-1.1.2-1.x86_64.rpm
    ├── opae-tools-1.1.2-1.x86_64.rpm
    ├── opae-tools-extra-1.1.2-1.x86_64.rpm
    └── opae-1.1.2-1.tar.gz
        └── tools
            └── extra
                └── packager
                    └── packager.py
```

5

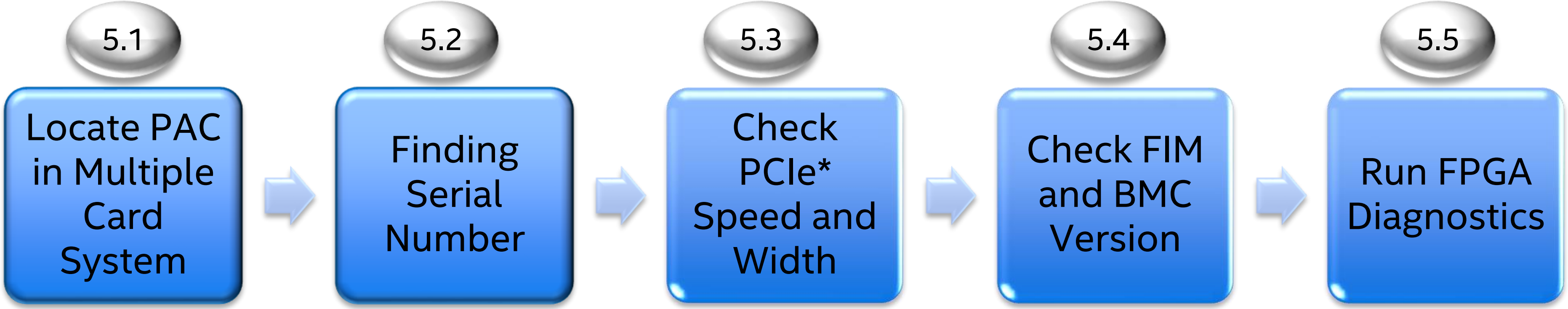
# BOARD BRING-UP

# Out-of-Box User Flow for Acceleration Stack


1
Select supported Server
2
Install PAC
3
Install Supported OS
4
5
Deployment Flow (Software only)
AFU Development Flow With Quartus
Download & Install Runtime Package of Acceleration Stack
Download & Install Developer Package of Acceleration Stack
Includes HLS & OpenCL (Optional)
Board Bring up
Write Host Application
Download & Install Workload
Download & Install Simulator
Create & Simulate Workload
Intel Website (Acceleration Hub)

Runtime package includes only OPAE drivers and sample AFU
Developer package includes Quartus + IP Lic + drivers

# Board Bring up steps



5.1 Locate PAC in Multiple Card System

5.2 Finding Serial Number

5.3 Check PCIe* Speed and Width

5.4 Check FIM and BMC Version

5.5 Run FPGA Diagnostics

5.1

# Locating PAC in Multi-Card System: SYSFS Entry

## To list all SYSFS entries in a multi-PAC system

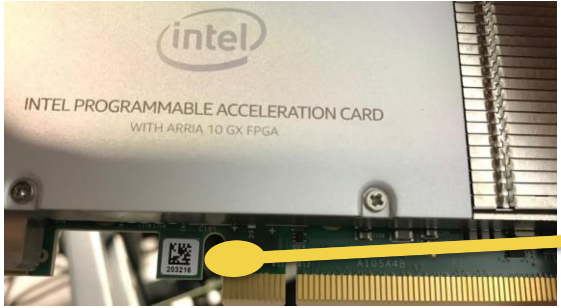
```
$ ls -l /sys/class/fpga/intel-fpga-dev.?/device
```

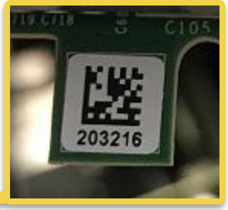
```
lrwxrwxrwx. 1 root root 0 Oct 25 12:34 /sys/class/fpga/intel-fpga-dev.0/device -> ../../../0000:3b:00.0
lrwxrwxrwx. 1 root root 0 Oct 25 12:34 /sys/class/fpga/intel-fpga-dev.1/device -> ../../../0000:86:00.0
lrwxrwxrwx. 1 root root 0 Oct 25 12:34 /sys/class/fpga/intel-fpga-dev.2/device -> ../../../0000:87:00.0
```

5.2

# Finding Board Serial Number

## To view serial number for a particular SYSFS entry





**Find serial number on front bottom of Arria® 10 GX PAC**

```
$ hexdump -C /sys/class/fpga/intel-fpga-dev.2/intel-fpga-fme.2/intel-pac-hssi.?.auto/hssi_mgmt/eeprom
```

```
00000000  4d 41 43 3d 30 30 3a 30  62 3a 33 65 3a 30 31 3a  |MAC=00:0b:3e:01:|
00000010  65 65 3a 66 38 0a 53 4e  3d 32 30 33 32 31 36 0a  |ee:f8.SN=203216.|
00000020  50 43 3d 41 31 30 53 41  34 2d 30 55 2d 42 31 31  |PC=A10SA4-0U-B11|
00000030  35 58 32 45 32 51 2d 32  32 2d 49 34 30 31 34 30  |5X2E2Q-22-I40140|
00000040  54 2d 36 0a 52 45 56 3d  31 2e 31 32 2e 30 2e 30  |T-6.REV=1.12.0.0|
00000050  2e 30 0a 0a ff ff ff ff  ff ff ff ff ff ff ff ff  |.0..............|
***
00000200
```

5.3

# Check PCIe Speed and Width

```
$ sudo lspci -d 8086:09c4 -vvv
```

```
05:00.0 Processing accelerators: Intel Corporation Device 09c4
        Subsystem: Intel Corporation Device 0000
        Physical Slot: 2
        Control: I/O- Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr- Stepping- SERR+ FastB2B- DisINTx+
        Status: Cap+ 66MHz- UDF- FastB2B- ParErr- DEVSEL=fast >TAbort- <TAbort- <MAbort- >SERR- <PERR- INTx-
        Latency: 0, Cache Line Size: 64 bytes
        Interrupt: pin A routed to IRQ 25
        Region 0: Memory at eab00000 (64-bit, prefetchable) [size=512K]
        Region 2: Memory at eaa00000 (64-bit, prefetchable) [size=1M]
        Capabilities: [68] MSI-X: Enable+ Count=7 Masked-
                Vector table: BAR=0 offset=00009000
                PBA: BAR=0 offset=0000a000
        Capabilities: [78] Power Management version 3
                Flags: PMEClk- DSI- D1- D2- AuxCurrent=0mA PME(D0-,D1-,D2-,D3hot-,D3cold-)
                Status: D0 NoSoftRst+ PME-Enable- DSel=0 DScale=0 PME-
        Capabilities: [80] Express (v2) Endpoint, MSI 00
                DevCap: MaxPayload 256 bytes, PhantFunc 0, Latency L0s <64ns, L1 <1us
                        ExtTag+ AttnBtn- AttnInd- PwrInd- RBE+ FLReset+
                DevCtl: Report errors: Correctable+ Non-Fatal+ Fatal+ Unsupported+
                        RlxdOrd- ExtTag+ PhantFunc- AuxPwr- NoSnoop+ FLReset-
                        MaxPayload 256 bytes, MaxReadReq 1024 bytes
                DevSta: CorrErr- UncorrErr- FatalErr- UnsuppReq- AuxPwr- TransPend-
                LnkCap: Port #0, Speed 8GT/s, Width x8, ASPM not supported, Exit Latency L0s <4us, L1 <1us
                        ClockPM- Surprise- LLActRep- BwNot-
                LnkCtl: ASPM Disabled; RCB 64 bytes Disabled- CommClk+
                        ExtSynch- ClockPM- AutWidDis- BWInt- AutBWInt-
                LnkSta: Speed 8GT/s, Width x8, TrErr- Train- SlotClk+ DLActive- BWMgmt- ABWMgmt-
(truncated)
```

# Useful OPAE Command-line Utilities For Board Management

| Commands | Description |
|---|---|
| fpgainfo | User can read the Board Telemetry data. For example temperature or Voltages. |
| fpgabist | Performs self-diagnostic test: measure bandwidth between local DDR4 memory and system memory |
| fpgaconf | Configure Acceleration Function Unit (AFU) into FPGA;<br>Check compatibility with targeted FPGA and FIM |
| fpgaflash | Updates FPGA Interface Manager (FIM) image (.rpd file) being stored in flash;<br>Updates BMC firmware. |

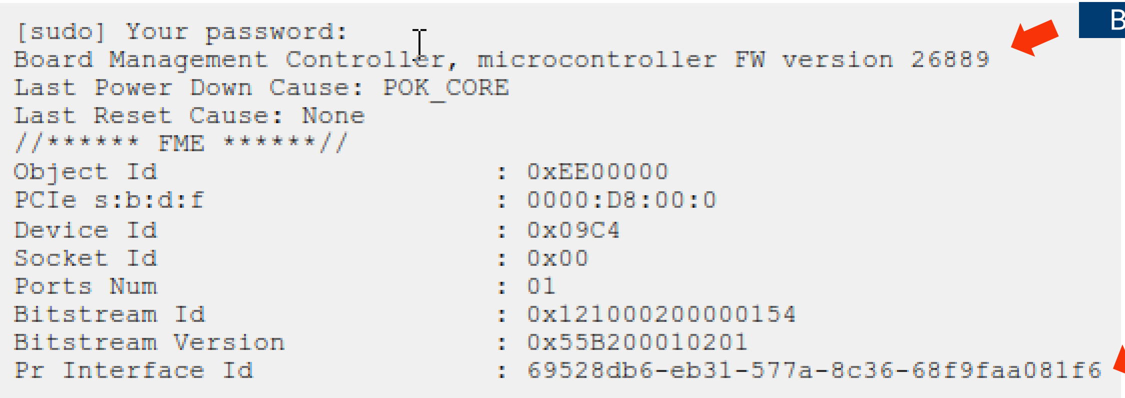
5.4

# Checking FIM and BMC Version

- Board needs active PCIe* link to check FIM version
- Use OPAE tool **fpgainfo** to check PAC's FIM and BMC version

```
$ sudo fpgainfo fme
```

- Sample output



```
[sudo] Your password:
Board Management Controller, microcontroller FW version 26889
Last Power Down Cause: POK_CORE
Last Reset Cause: None
//****** FME ******//
Object Id                     : 0xEE00000
PCIe s:b:d:f                  : 0000:D8:00:0
Device Id                     : 0x09C4
Socket Id                     : 0x00
Ports Num                     : 01
Bitstream Id                  : 0x121000200000154
Bitstream Version             : 0x55B200010201
Pr Interface Id               : 69528db6-eb31-577a-8c36-68f9faa081f6
```

BMC Version

FIM Version

5.4

# Version Table

| Acceleration Stack Version | FIM Version (PR Interface ID) | OPAE Version | BMC Version |
|---|---|---|---|
| 1.2 Production | 69528db6-eb31-577a-8c36-68f9faa081f6 | 1.1.2-1 | 26889 |
| 1.2 Alpha | 93abeb6a-30c8-5f77-8172-d828c3a699ca | 1.1.1-1 | 26889 |
| 1.1 Production | 9926ab6d-6c92-5a68-aabc-a7d84c545738 | 1.0.2 | 26822 |

5.4

# Ensure PAC Is Visible In-System

If FIM is loaded correctly, PAC should show up as PCIe* endpoint, and can be seen from ‘lspci’ (Linux command).

```
$ lspci | grep 09c4
```

- OS on host CPU will discover PAC cards as PCIe device 8086:09c4

```
04:00.0 Processing accelerators [1200]: Intel Corporation Device [8086:09c4]
```

5.4

# Checking OPAE Software Version

## Follow Quick Start Guide to check OPAE version

- For example, in CentOS/RHEL, run the following to check OPAE version:

```
$ rpm -qa | grep opae
```

- Sample output

```
opae-tools-1.1.2-1.x86_64
opae-devel-1.1.2-1.x86_64
opae-libs-1.1.2-1.x86_64
opae-1.1.2-1.x86_64
```

# What is Diagnostic test

The fpgabist tool performs self-diagnostic tests on supported FPGA platforms.

Tests PCIe, DMA from CPU DDR to Device DDR and memory access bandwidth

Currently, fpgabist accepts the following AFs:

1. **nlb_mode_3:** The native loopback (NLB) test implements a loopback from TX to RX. Use it to verify basic functionality and to measure bandwidth.
2. **dma_afu:** The direct memory access (DMA) AFU test transfers data from host memory to FPGA-attached local memory.

5.5

# Run FPGA Diagnostics

Configure the number of system hugepages the fpgadiag utility requires

```
sudo sh -c "echo 20 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages"
```

Configure and run diagnostics with NLB_3 AFU Image

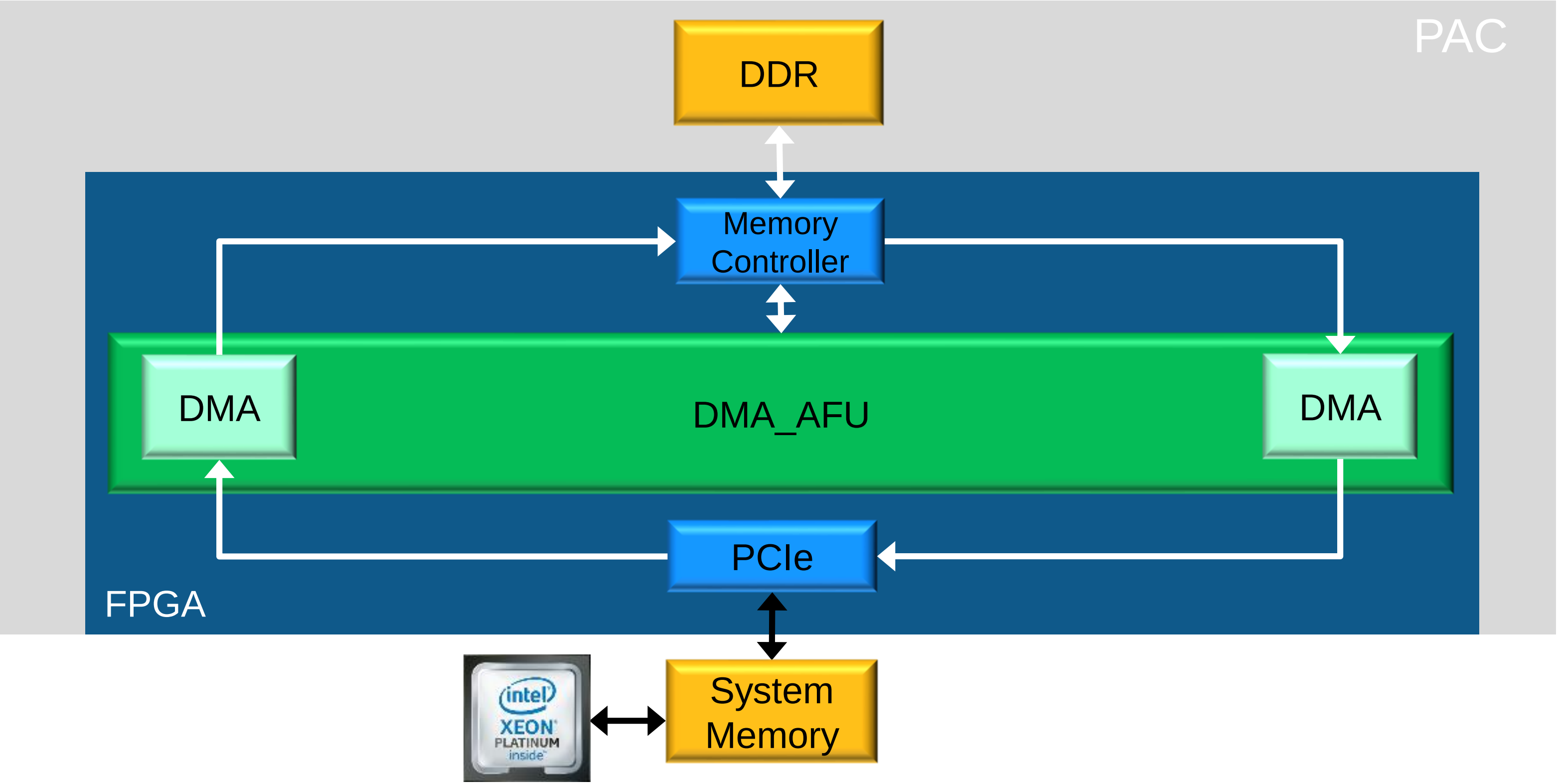
```
sudo fpgabist $OPAE_PLATFORM_ROOT/hw/samples/nlb_mode_3/bin/nlb_mode_3.gbs
```

Configure and run diagnostics with DMA AFU Image

```
sudo fpgabist $OPAE_PLATFORM_ROOT/hw/samples/dma_afu/bin/dma_afu.gbs
```

5.5

# DMA AFU: Built-In Self Test (fpgabist)



5.5

# DMA Built-in Self Test Output

`fpgainfo` Tool output (FME, TEMP, POWER, PORT)

FME and PORT error status registers (for AFU developer and user to debug)

Partial Reconfiguration messages (loading AFU)

DMA bandwidth report

# DMA BIST Output  1/5

```
==========================================================

Beginning FPGA Built-In Self-Test

==========================================================
Device: bus = 04, device = 00, func = 0
Board Management Controller, microcontroller FW version 26889
Last Power Down Cause: POK_CORE
Last Reset Cause: None
//****** FME ******//
Object Id                     : 0xF300000
PCIe s:b:d:f                  : 0000:04:00:0
Device Id                     : 0x09C4
Socket Id                     : 0x00
Ports Num                     : 01
Bitstream Id                  : 0x121000200000161
Bitstream Version             : 0x10201
Pr Interface Id               : 93abeb6a-30c8-5f77-8172-d828c3a699ca
Board Management Controller, microcontroller FW version 26889
Last Power Down Cause: POK_CORE
Last Reset Cause: None
//****** PORT ******//
Object Id                     : 0xF200000
PCIe s:b:d:f                  : 0000:04:00:0
Device Id                     : 0x09C4
Socket Id                     : 0x00
Ports Num                     : 01
Bitstream Id                  : 0x121000200000161
Bitstream Version             : 0x10201
Pr Interface Id               : 93abeb6a-30c8-5f77-8172-d828c3a699ca
Accelerator Id                : 331db30c-9885-41ea-9081-f88b8f655caa
Board Management Controller, microcontroller FW version 26889
Last Power Down Cause: POK_CORE
Last Reset Cause: None
```

**Output of “fpgainfo fme” command**

**Output of “fpgainfo port” command**

# DMA BIST Output 2/5

```
//****** TEMP ******//
Object Id                      : 0xF300000
PCIe s:b:d:f                   : 0000:04:00:0
Device Id                      : 0x09C4
Socket Id                      : 0x00
Ports Num                      : 01
Bitstream Id                   : 0x121000200000161
Bitstream Version              : 0x10201
Pr Interface Id                : 93abeb6a-30c8-5f77-8172-d828c3a699ca
(11) FPGA Core TEMP            : 73.00 °C
(12) Board TEMP                : 47.00 °C
(14) QSFP TEMP                 : No reading (reading state unavailable)
(15) Core Supply Temp          : 75.96 °C
Board Management Controller, microcontroller FW version 26889
Last Power Down Cause: POK_CORE
Last Reset Cause: None
//****** POWER ******//
Object Id                      : 0xF300000
PCIe s:b:d:f                   : 0000:04:00:0
Device Id                      : 0x09C4
Socket Id                      : 0x00
Ports Num                      : 01
Bitstream Id                   : 0x121000200000161
Bitstream Version              : 0x10201
Pr Interface Id                : 93abeb6a-30c8-5f77-8172-d828c3a699ca
( 0) Total Input Power         : 23.50 Watts
( 1) PCIe 12V Current          : 1.96 Amps
( 2) PCIe 12V Voltage          : 11.60 Volts
( 3) 1.2V Voltage              : 1.22 Volts
( 4) 1.2V Current              : 2.66 Amps
( 5) 1.8V Voltage              : 1.83 Volts
( 6) 1.8V Current              : 2.91 Amps
( 7) 3.3V Mgmt Voltage         : 3.36 Volts
( 8) 3.3V Current              : 0.72 Amps
( 9) FPGA Core Voltage         : 0.90 Volts
(10) FPGA Core Current         : 8.02 Amps
```

Output of “fpgainfo temp” command

Output of “fpgainfo power” command

# DMA BIST Output 3/5

```
//****** FME ERRORS ******//
Object Id                     : 0xF300000
PCIe s:b:d:f                  : 0000:04:00:0
Device Id                     : 0x09C4
Socket Id                     : 0x00
Ports Num                     : 01
Bitstream Id                  : 0x121000200000161
Bitstream Version             : 0x7FFD00010201
Pr Interface Id               : 93abeb6a-30c8-5f77-8172-d828c3a699ca
First Error                   : 0x0
Next Error                    : 0x0
Errors                        : 0x0
PCIe1 Errors                  : 0x0
Nonfatal Errors               : 0x0
Inject Error                  : 0x0
Catfatal Errors               : 0x0
PCIe0 Errors                  : 0x0
```

**Output of “fpgainfo error” command**

# DMA BIST Output 4/5

Loading DMA AFU (FPGA partial reconfiguration)

AFU will “find slot” if AFU version matched FIM version

```
Running mode: dma_afu
Attempting Partial Reconfiguration:
Reading bitstream
Looking for slot
Found slot
Programming bitstream
Writing bitstream
Done
```

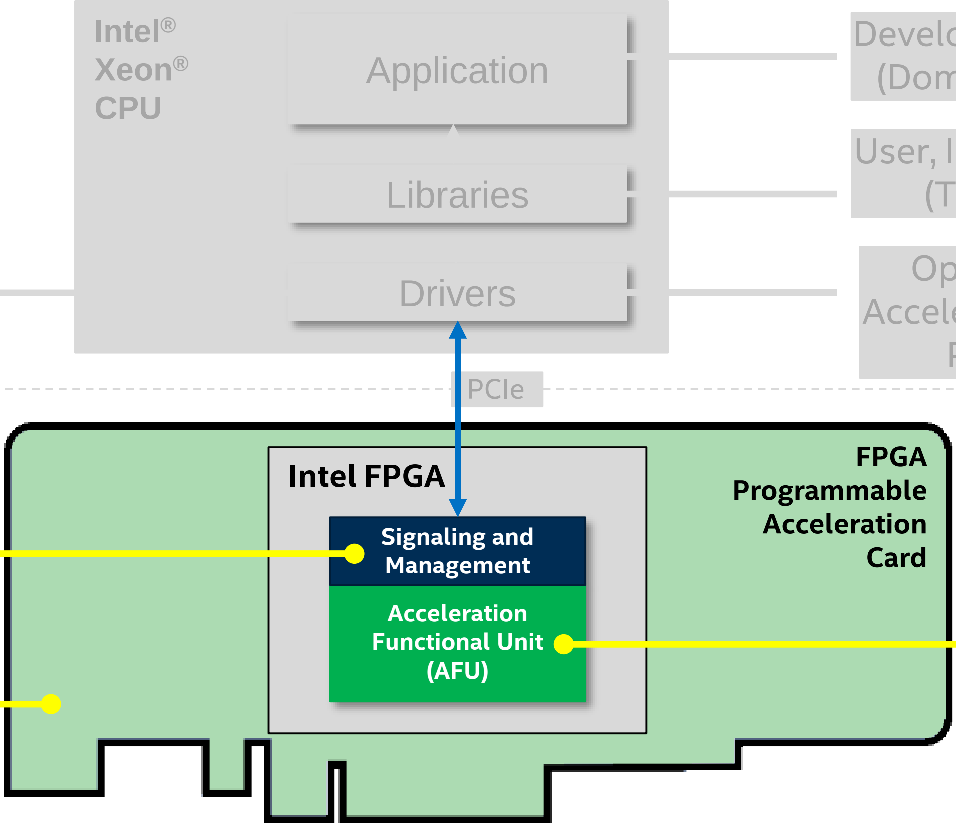
# DMA BIST Output 5/5

```
Running fpga_dma_test test...

Running test in HW mode
Buffer Verification Success!
Buffer Verification Success!
Running DDR sweep test
Buffer pointer = 0x7f1b68982000, size = 0x100000000 (0x7f1b68982000 through 0x7f1c68982000)
Allocated test buffer
Fill test buffer
DDR Sweep Host to FPGA
Measured bandwidth = 6810.764668 Megabytes/sec
Clear buffer
DDR Sweep FPGA to Host
Measured bandwidth = 6917.527127 Megabytes/sec
Verifying buffer..
Buffer Verification Success!
DDR sweep with unaligned pointer and size
Buffer pointer = 0x7f1b6938303d, size = 0xffffffbe (0x7f1b6938303d through 0x7f1c69382ffb)
…
…
Buffer pointer = 0x7f1b69383000, size = 0xfffffff9 (0x7f1b69383000 through 0x7f1c69382ff9)
Allocated test buffer
Fill test buffer
DDR Sweep Host to FPGA
Measured bandwidth = 6813.543883 Megabytes/sec
Clear buffer Clear buffer
DDR Sweep FPGA to Host
Measured bandwidth = 6926.264906 Megabytes/sec
Verifying buffer..
Buffer Verification Success!
Finished Executing DMA Tests


Built-in Self-Test Completed.
```

**Measured bandwidth for each direction**

# ACCELERATOR FUNCTIONAL UNIT (AFU)

# FPGA Interface Manager (FIM) + AFU



Developed by User (Domain Expert)

User, Intel, and 3rd Party (Tuning Expert)

Open Programmable Acceleration Engine (OPAE) Provided by Intel

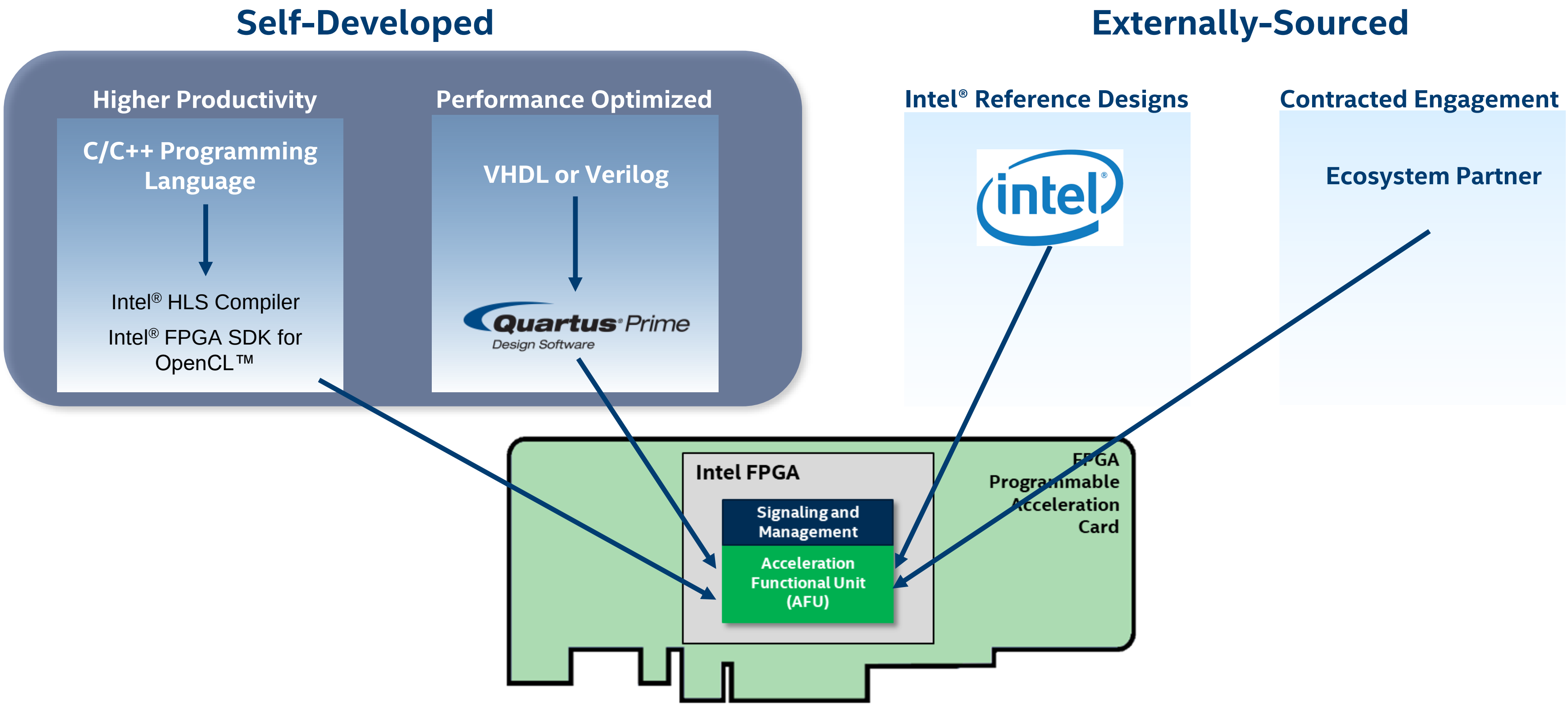
PCIe* Drivers Provided by Intel

*FPGA Interface Manager* Provided by Intel

*Qualified and Validated for volume deployment* Provided by OEMs

User, Intel, or 3rd-Party IP Plugs into AFU Slot (Tuning Expert)

# How Can FPGA Accelerators Be Created?



## Self-Developed

### Higher Productivity

C/C++ Programming Language

Intel® HLS Compiler

Intel® FPGA SDK for OpenCL™

### Performance Optimized

VHDL or Verilog

Quartus Prime
Design Software

## Externally-Sourced

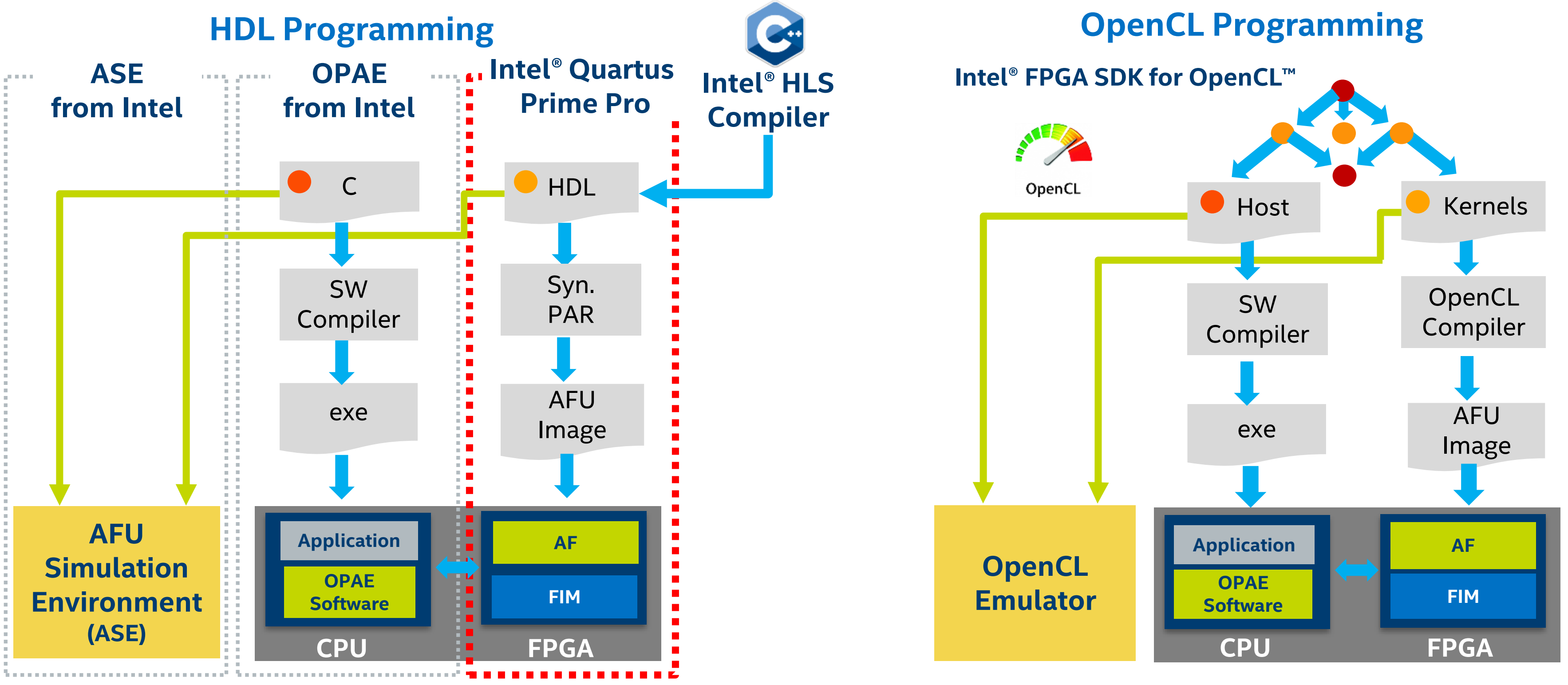
### Intel® Reference Designs

### Contracted Engagement

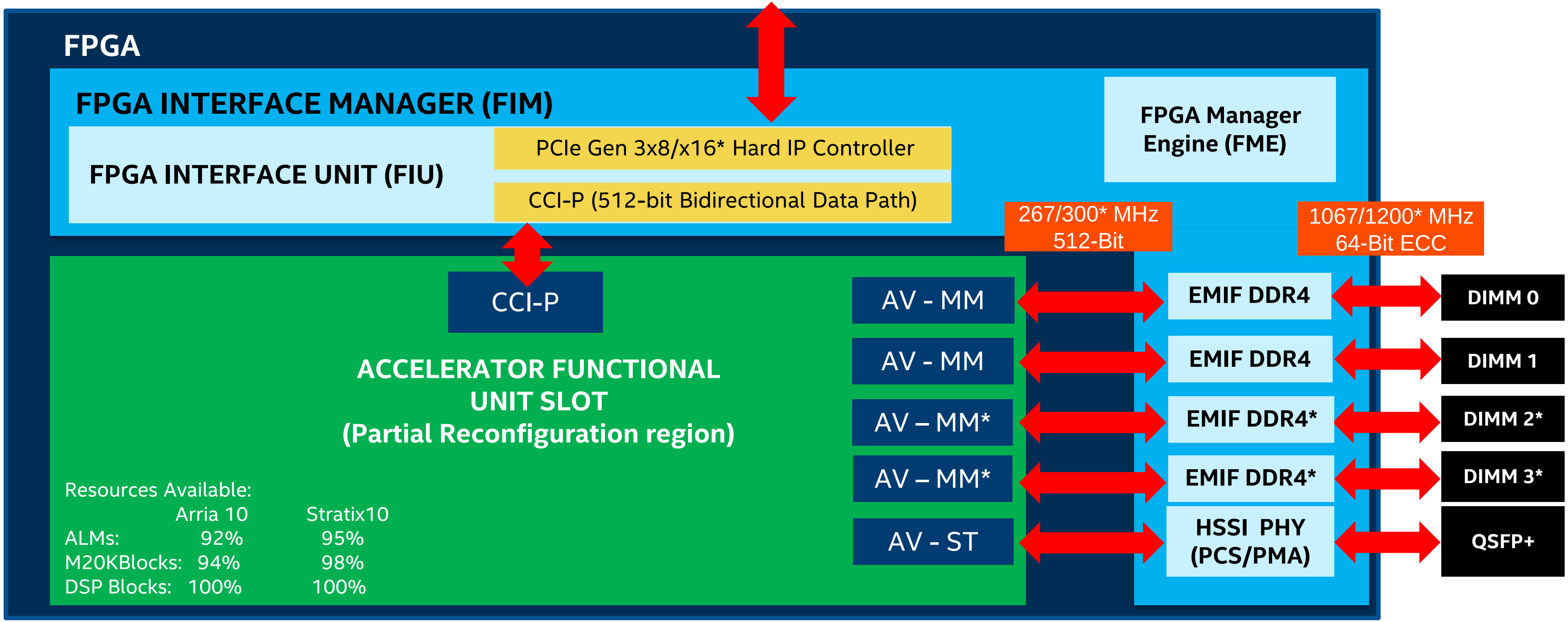
Ecosystem Partner

Intel FPGA
Signaling and Management
Acceleration Functional Unit (AFU)
FPGA Programmable Acceleration Card

# Accelerator Function Development



# FPGA INTERFACE MANAGER (FIM): Under the hood


FPGA
FPGA INTERFACE MANAGER (FIM)
FPGA INTERFACE UNIT (FIU)
PCIe Gen 3x8/x16* Hard IP Controller
CCI-P (512-bit Bidirectional Data Path)
FPGA Manager Engine (FME)
267/300* MHz
512-Bit
1067/1200* MHz
64-Bit ECC
CCI-P
ACCELERATOR FUNCTIONAL UNIT SLOT
(Partial Reconfiguration region)
AV - MM
AV - MM
AV – MM*
AV – MM*
AV - ST
EMIF DDR4
EMIF DDR4
EMIF DDR4*
EMIF DDR4*
HSSI PHY (PCS/PMA)
DIMM 0
DIMM 1
DIMM 2*
DIMM 3*
QSFP+

Resources Available:

| | Arria 10 | Stratix10 |
|---|---|---|
| ALMs: | 92% | 95% |
| M20KBlocks: | 94% | 98% |
| DSP Blocks: | 100% | 100% |

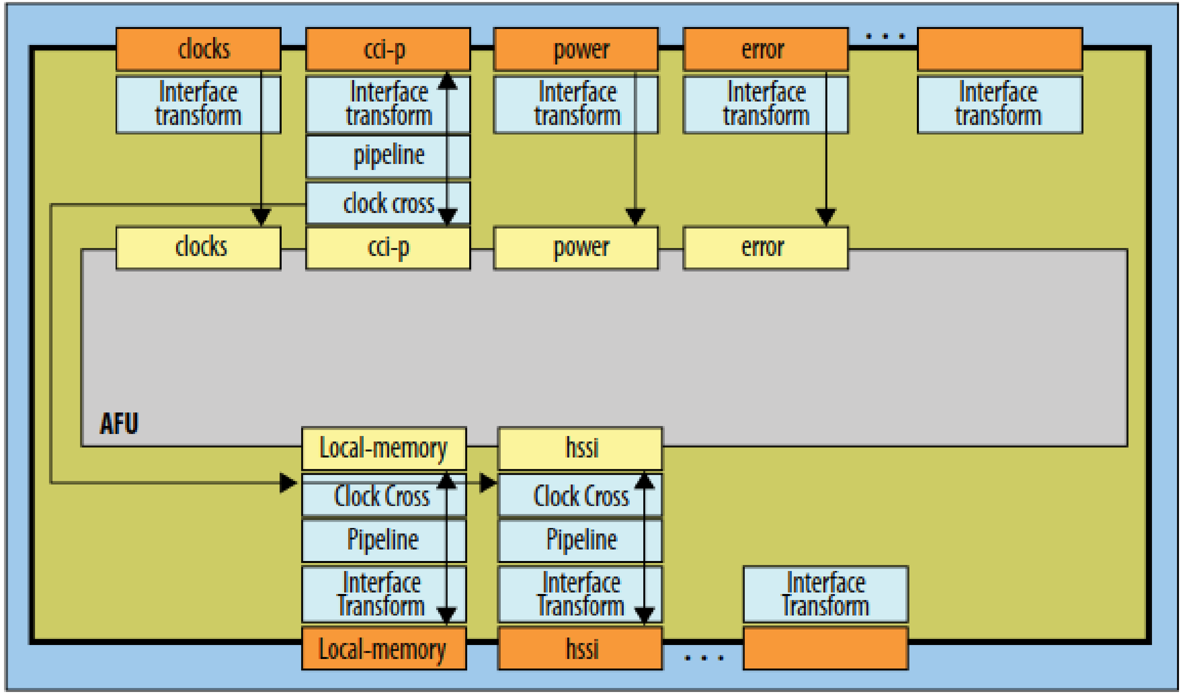
* Stratix 10 PAC only

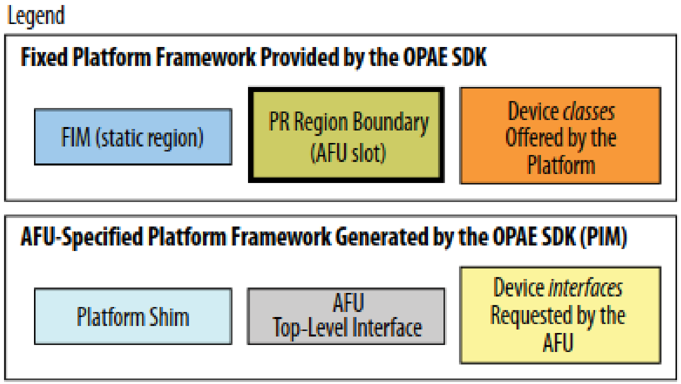
# Overview of OPAE Platform for AFUs

Platform Interface Manager (PIM) defines a generic OPAE platform for which AFU top-levels should be designed

- The AFU requests the device interfaces and properties it needs from the PIM using a platform configuration file specification (.json)
- Generates a shim that translates hardware platform-specific device interfaces to the OPAE Platform’s generic device interfaces used by the AFU
- Shim inserted between platforms PR region and the AFU providing top level module interface for the AFU

# OPAE Platform Device Classes




Legend
Fixed Platform Framework Provided by the OPAE SDK
FIM (static region)
PR Region Boundary (AFU slot)
Device *classes* Offered by the Platform
AFU-Specified Platform Framework Generated by the OPAE SDK (PIM)
Platform Shim
AFU Top-Level Interface
Device *interfaces* Requested by the AFU

Clocks

CCI-P

Power

Error

HSSI

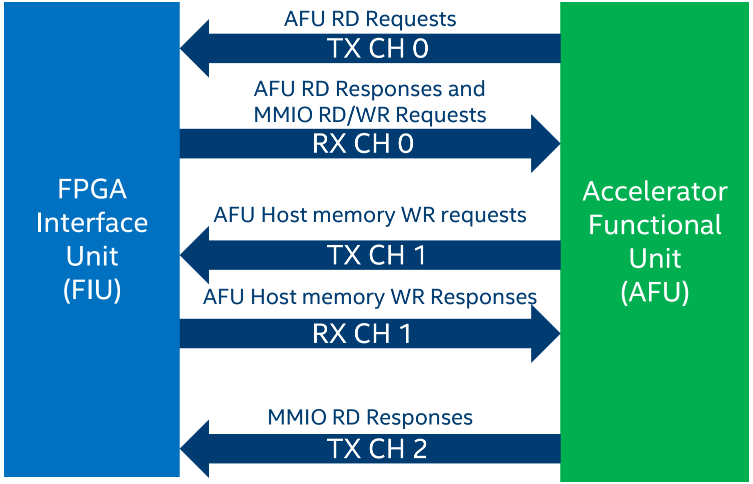
Local Memory

# Core Cache Interface: Overview

CCI abstracts AFU from lower level PCIe protocol

Enables AFU to access host memory and respond to MMIO requests

Composed of 3 command and response channels


FPGA Interface Unit (FIU)
AFU RD Requests
TX CH 0
AFU RD Responses and MMIO RD/WR Requests
RX CH 0
AFU Host memory WR requests
TX CH 1
AFU Host memory WR Responses
RX CH 1
MMIO RD Responses
TX CH 2
Accelerator Functional Unit (AFU)

- Supports bidirectional 512-bit data operating at 400MHz pClk domain
- Host memory accesses are on 64Byte Cache Line (CL) basis
  - Supports Multi-CL bursts of 2 or 4
  - Supports write fence mechanism to support synchronizing shared host memory accesses between AFU and Host SW application

https://www.intel.com/content/dam/altera-www/global/en_US/pdfs/literature/manual/mnl-ias-ccip.pdf

# Intel FPGA Basic Building Blocks (BBB)

Suite of RTL shims for transforming the CCI interface

Memory Properties Factory (MPF)

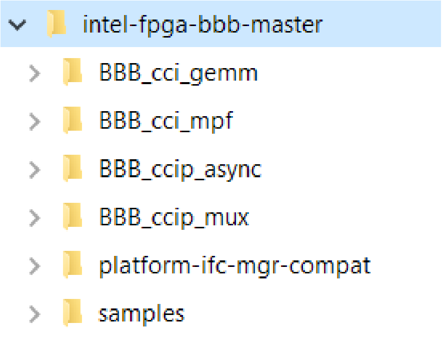
- Adds features to the base CCI memory interface

CCI Async-shim

- Clock crossing shim for slower-running accelerators

CCI Multiplexer

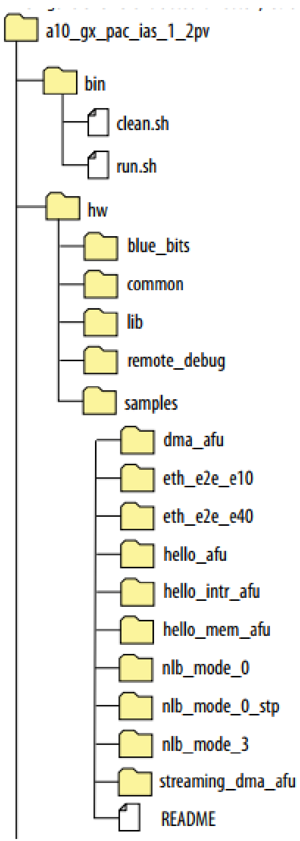
- Allows multiple agents to share a single CCI-P interface



```
intel-fpga-bbb-master
    BBB_cci_gemm
    BBB_cci_mpf
    BBB_ccip_async
    BBB_ccip_mux
    platform-ifc-mgr-compat
    samples
```

```
$ git clone https://github.com/OPAE/intel-fpga-bbb
```

# Example Designs to Get Started

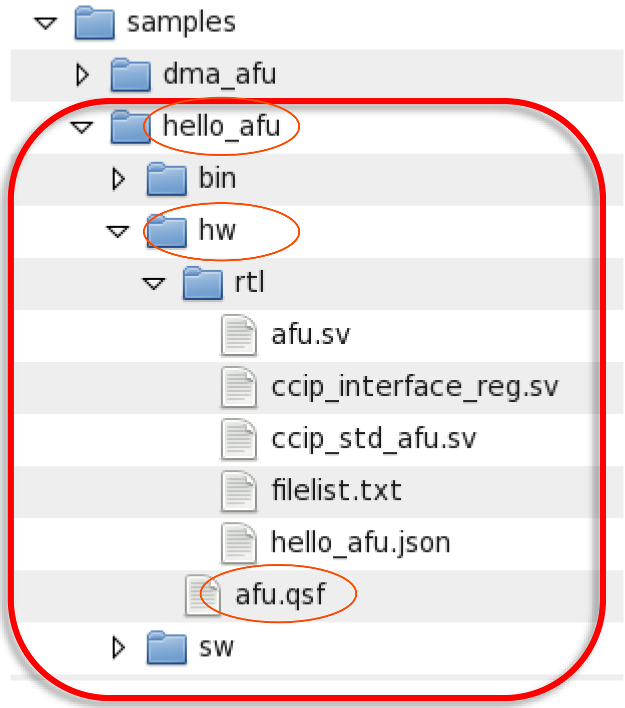


```
a10_gx_pac_ias_1_2pv
├─ bin
│  ├─ clean.sh
│  └─ run.sh
└─ hw
   ├─ blue_bits
   ├─ common
   ├─ lib
   ├─ remote_debug
   └─ samples
      ├─ dma_afu
      ├─ eth_e2e_e10
      ├─ eth_e2e_e40
      ├─ hello_afu
      ├─ hello_intr_afu
      ├─ hello_mem_afu
      ├─ nlb_mode_0
      ├─ nlb_mode_0_stp
      ├─ nlb_mode_3
      ├─ streaming_dma_afu
      └─ README
```

| Example | Description |
|---|---|
| Hello AFU | Simple AFU with direct CCI connection for MMIO access |
| Hello Intr AFU | Example use of user interrupts |
| Hello Mem AFU | Example showing using USR Clock to auto close timing in the AFU |
| DMA AFU | Example DMA AFU to move data between host memory and local FPGA memory. Uses BBB and bridges Avalon to CCI |
| Streaming DMA AFU | Example DMA AFU to move data between host memory and the AFU directly as a streaming packet |
| Eth e2e e10 | 10Gb Ethernet loopback design |
| Eth e2e e40 | 40Gb Ethernet loopback  design |
| NLB mode 0 | Native LoopBack adaptor (rd/wr) with more features |
| NLB mode 0 stp | Native LoopBack adaptor with SignalTap remote debug |
| NLB mode 3 | Native LoopBack adaptor (rd/wr) |

# AF Project Structure

## Overview of hello_afu example AFU



```
samples
  dma_afu
  hello_afu
    bin
    hw
      rtl
        afu.sv
        ccip_interface_reg.sv
        ccip_std_afu.sv
        filelist.txt
        hello_afu.json
      afu.qsf
    sw
```

Start with existing design and modify for your needs

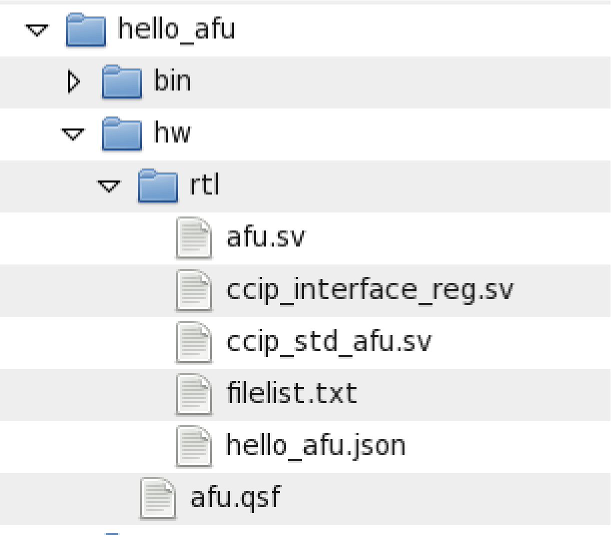
- The ./*hw* directory provides an example file structure for the AFU's design source and build structure
- Host OPAE software application source in the ./*sw* directory
  - To perform the co-simulation environment

Project directory typically contains :

- AFU's Quartus settings file (./hw/afu.qsf)
- AFU's RTL
- AFU's Quartus PR *build* directory (./build) with project files and compiled AF image (.gbs)
- Platform configuration file (.json)
- Build configuration file (.txt)

# AFU RTL Source

## Mandatory Source Files and Hierarchical Structure



```
hello_afu
    bin
    hw
        rtl
            afu.sv
            ccip_interface_reg.sv
            ccip_std_afu.sv
            filelist.txt
            hello_afu.json
        afu.qsf
```

afu.sv

- AFU top-level RTL source file describing accelerator
- Can have any name, but the top-level module within must be named “afu”

ccip_std_afu.sv

- Mandatory top level wrapper RTL file that instantiates the AFU module described in afu.sv
- Instantiates mandatory ccip_interface_reg module described in the mandatory ccip_interface_reg.sv source file

The .json file is the platform configuration file describing the devices classes required by AFU

The filelist.txt file specifies the build configuration (including source files and .json file)

# Platform Configuration File (.json)

## Specify the AFU's UUID

- uuidgen To generate

## Request a top-level interfaces

- ccip_std_afu, ccip_std_afu_avalon_mm and optional HSSI device interfaces

## Request pipelining on device interfaces

- Adds user defined number of pipeline register stages to cci or local memory interfaces

## Request clock crossing on device interfaces

- Inserts clock crossing bridge to synchronize cci and local memory to a clock

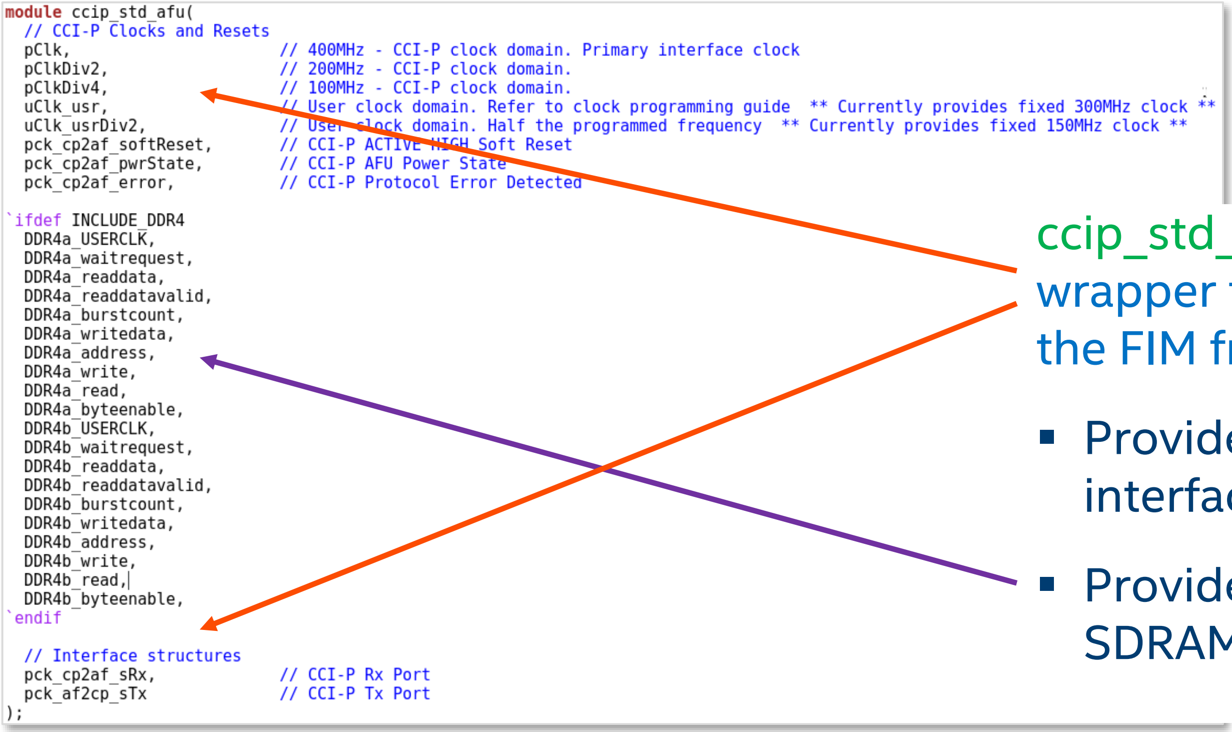
## Specify a requested device interface as optional

## Specify AFU user clock timing

- Close timing using user clock frequency range defined here

# AFU RTL Source

## ccip_std_afu.sv Source File (1/2)



```
module ccip_std_afu(
  // CCI-P Clocks and Resets
  pClk,                       // 400MHz - CCI-P clock domain. Primary interface clock
  pClkDiv2,                   // 200MHz - CCI-P clock domain.
  pClkDiv4,                   // 100MHz - CCI-P clock domain.
  uClk_usr,                   // User clock domain. Refer to clock programming guide  ** Currently provides fixed 300MHz clock **
  uClk_usrDiv2,               // User clock domain. Half the programmed frequency  ** Currently provides fixed 150MHz clock **
  pck_cp2af_softReset,        // CCI-P ACTIVE HIGH Soft Reset
  pck_cp2af_pwrState,         // CCI-P AFU Power State
  pck_cp2af_error,            // CCI-P Protocol Error Detected

`ifdef INCLUDE_DDR4
  DDR4a_USERCLK,
  DDR4a_waitrequest,
  DDR4a_readdata,
  DDR4a_readdatavalid,
  DDR4a_burstcount,
  DDR4a_writedata,
  DDR4a_address,
  DDR4a_write,
  DDR4a_read,
  DDR4a_byteenable,
  DDR4b_USERCLK,
  DDR4b_waitrequest,
  DDR4b_readdata,
  DDR4b_readdatavalid,
  DDR4b_burstcount,
  DDR4b_writedata,
  DDR4b_address,
  DDR4b_write,
  DDR4b_read,
  DDR4b_byteenable,
`endif

  // Interface structures
  pck_cp2af_sRx,              // CCI-P Rx Port
  pck_af2cp_sTx               // CCI-P Tx Port
);
```

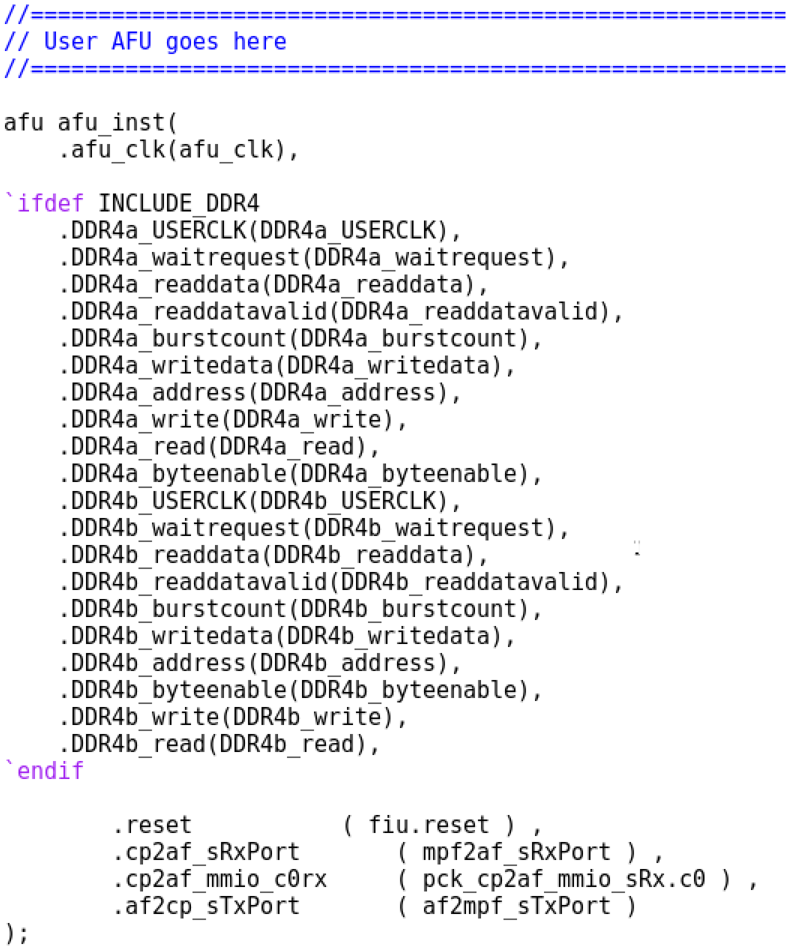
ccip_std_afu Module provides the wrapper for instantiating the AF into the FIM framework

- Provides access to the FIU host interface
- Provides access to the local DDR4 SDRAM banks

# AFU RTL Source

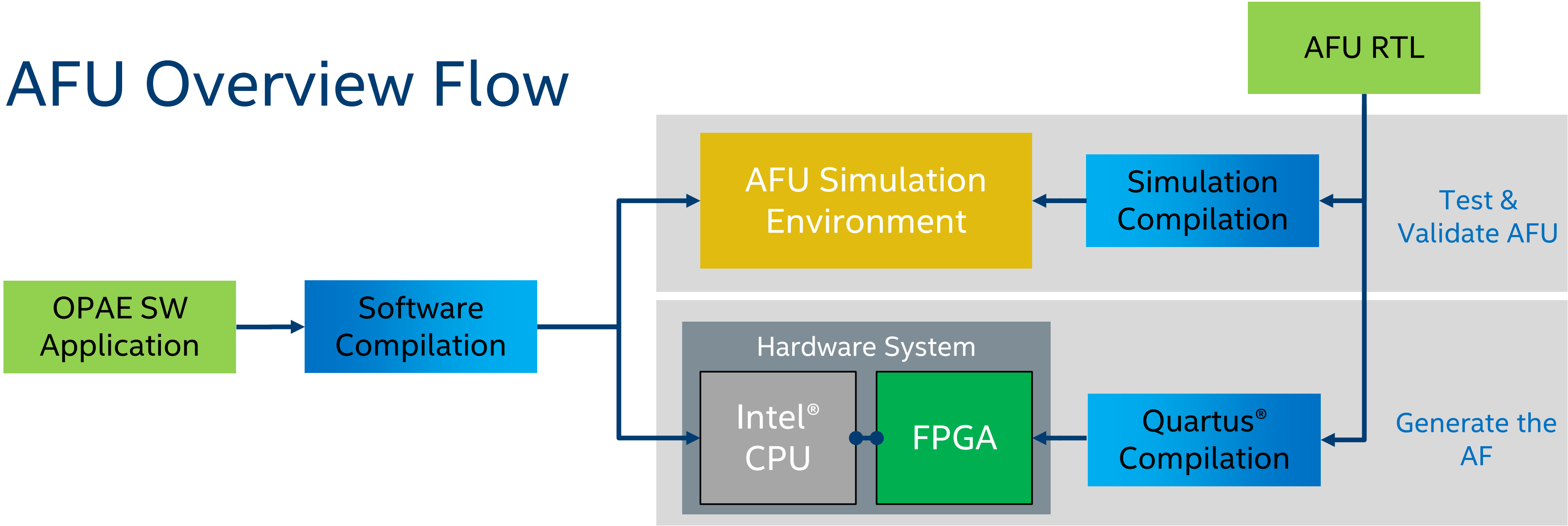
## ccip_std_afu.sv Source File (2/2)

Your AFU goes here



```
//=============================================================
// User AFU goes here
//=============================================================

afu afu_inst(
    .afu_clk(afu_clk),

`ifdef INCLUDE_DDR4
    .DDR4a_USERCLK(DDR4a_USERCLK),
    .DDR4a_waitrequest(DDR4a_waitrequest),
    .DDR4a_readdata(DDR4a_readdata),
    .DDR4a_readdatavalid(DDR4a_readdatavalid),
    .DDR4a_burstcount(DDR4a_burstcount),
    .DDR4a_writedata(DDR4a_writedata),
    .DDR4a_address(DDR4a_address),
    .DDR4a_write(DDR4a_write),
    .DDR4a_read(DDR4a_read),
    .DDR4a_byteenable(DDR4a_byteenable),
    .DDR4b_USERCLK(DDR4b_USERCLK),
    .DDR4b_waitrequest(DDR4b_waitrequest),
    .DDR4b_readdata(DDR4b_readdata),
    .DDR4b_readdatavalid(DDR4b_readdatavalid),
    .DDR4b_burstcount(DDR4b_burstcount),
    .DDR4b_writedata(DDR4b_writedata),
    .DDR4b_address(DDR4b_address),
    .DDR4b_byteenable(DDR4b_byteenable),
    .DDR4b_write(DDR4b_write),
    .DDR4b_read(DDR4b_read),
`endif

        .reset           ( fiu.reset ) ,
        .cp2af_sRxPort       ( mpf2af_sRxPort ) ,
        .cp2af_mmio_c0rx     ( pck_cp2af_mmio_sRx.c0 ) ,
        .af2cp_sTxPort       ( af2mpf_sTxPort )
);
```

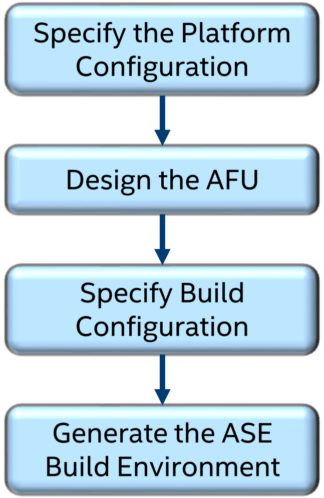
# AFU Overview Flow



## AF Simulation Environment (ASE) enables seamless portability to real HW

- Allows fast verification of OPAE software together with AF RTL without HW
  - SW Application loads ASE library and connects to RTL simulation
- For execution on HW, application loads Runtime library and RTL is compiled by Intel® Quartus into FPGA bitstream

# AFU Development Flow Using OPAE SDK


Specify the Platform Configuration
Design the AFU
Specify Build Configuration
Generate the ASE Build Environment

AFU requests the ccip_std_afu top level interface classes

- *$OPAE_PLATFORM_ROOT/hw/samples/hello_afu/hw/rtl/hello_afu.json*

AFU RTL files implementing accelerated function

- *$OPAE_PLATFORM_ROOT/hw/samples/hello_afu/hw/rtl/afu.sv*

List all source files and platform configuration file

- *$OPAE_PLATFORM_ROOT/hw/samples/hello_afu/hw/rtl/filelist.txt*

In terminal window, enter these commands:

- *cd $OPAE_PLATFORM_ROOT/hw/samples/hello_afu*
- *afu_sim_setup --source hw/rtl/filelist.txt build_sim*

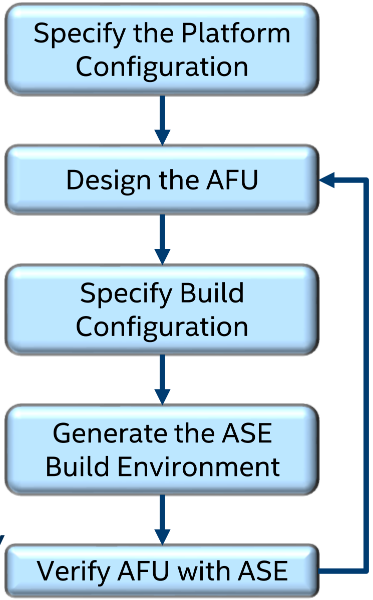
# AFU Development Flow Using OPAE SDK

## Compile AFU and platform simulation models and start simulation server process

- *cd build_sim*
- *make*
- *make sim*

## In 2nd terminal window compile the host application and start the client process

- *Export ASE_WORKDIR= $OPAE_PLATFORM_ROOT/hw/samples/hello_afu/build_sim/work*
- *cd $OPAE_PLATFORM_ROOT/hw/samples/hello_afu/sw*
- ***make USE_ASE=1***
- *./hello_afu*


Specify the Platform Configuration
Design the AFU
Specify Build Configuration
Generate the ASE Build Environment
Verify AFU with ASE

# AFU Simulation Environment (ASE)

Hardware software co-simulation environment

Uses simulator Direct Programming Interface (DPI) for HW/SW connectivity

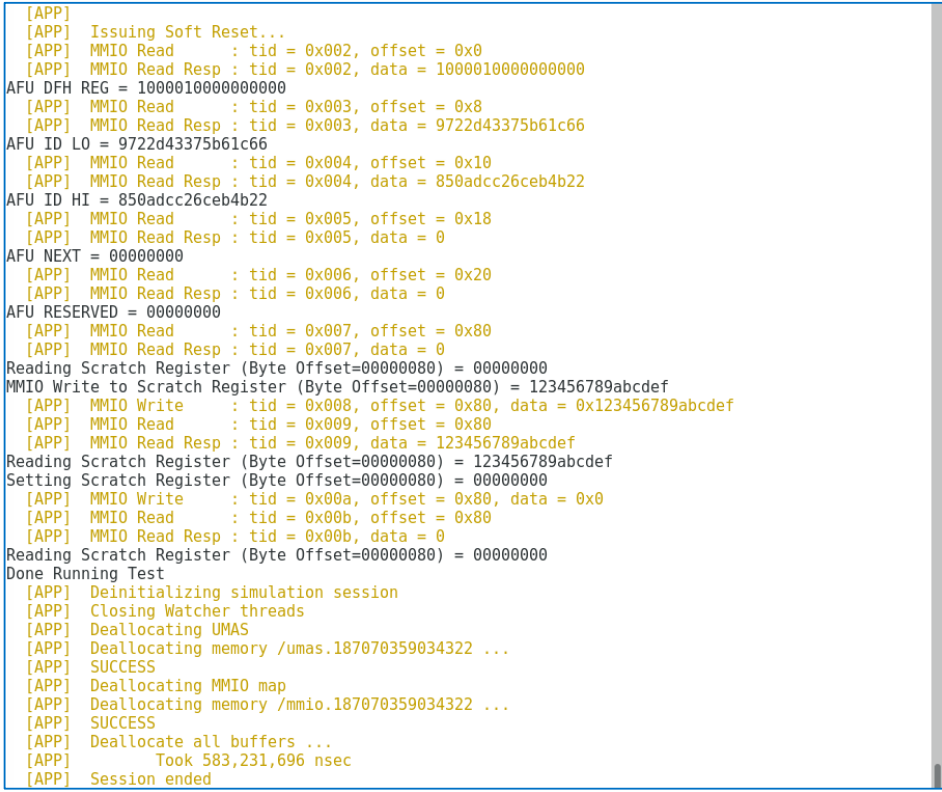
- Not cycle accurate (used for functional correctness)
- Converts SW API to CCI transactions

Provides transactional model for the Core Cache Interface (CCI-P) protocol and memory model for the FPGA-attached local memory

Validates compliance to

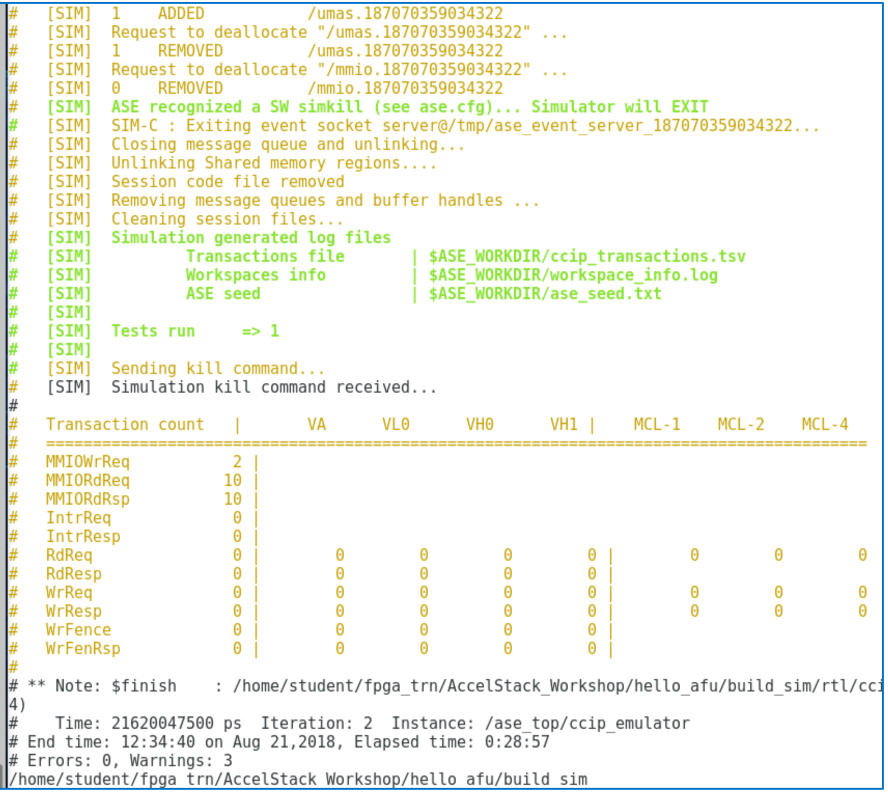
- CCI-P protocol specification
- Avalon® Memory Mapped (Avalon-MM) Interface Specification
- Open Programmable Acceleration Engine

# Simulation Complete



```
  [APP]
  [APP]  Issuing Soft Reset...
  [APP]  MMIO Read      : tid = 0x002, offset = 0x0
  [APP]  MMIO Read Resp : tid = 0x002, data = 1000010000000000
AFU DFH REG = 1000010000000000
  [APP]  MMIO Read      : tid = 0x003, offset = 0x8
  [APP]  MMIO Read Resp : tid = 0x003, data = 9722d43375b61c66
AFU ID LO = 9722d43375b61c66
  [APP]  MMIO Read      : tid = 0x004, offset = 0x10
  [APP]  MMIO Read Resp : tid = 0x004, data = 850adcc26ceb4b22
AFU ID HI = 850adcc26ceb4b22
  [APP]  MMIO Read      : tid = 0x005, offset = 0x18
  [APP]  MMIO Read Resp : tid = 0x005, data = 0
AFU NEXT = 00000000
  [APP]  MMIO Read      : tid = 0x006, offset = 0x20
  [APP]  MMIO Read Resp : tid = 0x006, data = 0
AFU RESERVED = 00000000
  [APP]  MMIO Read      : tid = 0x007, offset = 0x80
  [APP]  MMIO Read Resp : tid = 0x007, data = 0
Reading Scratch Register (Byte Offset=00000080) = 00000000
MMIO Write to Scratch Register (Byte Offset=00000080) = 123456789abcdef
  [APP]  MMIO Write     : tid = 0x008, offset = 0x80, data = 0x123456789abcdef
  [APP]  MMIO Read      : tid = 0x009, offset = 0x80
  [APP]  MMIO Read Resp : tid = 0x009, data = 123456789abcdef
Reading Scratch Register (Byte Offset=00000080) = 123456789abcdef
Setting Scratch Register (Byte Offset=00000080) = 00000000
  [APP]  MMIO Write     : tid = 0x00a, offset = 0x80, data = 0x0
  [APP]  MMIO Read      : tid = 0x00b, offset = 0x80
  [APP]  MMIO Read Resp : tid = 0x00b, data = 0
Reading Scratch Register (Byte Offset=00000080) = 00000000
Done Running Test
  [APP]  Deinitializing simulation session
  [APP]  Closing Watcher threads
  [APP]  Deallocating UMAS
  [APP]  Deallocating memory /umas.187070359034322 ...
  [APP]  SUCCESS
  [APP]  Deallocating MMIO map
  [APP]  Deallocating memory /mmio.187070359034322 ...
  [APP]  SUCCESS
  [APP]  Deallocate all buffers ...
  [APP]         Took 583,231,696 nsec
  [APP]  Session ended
```

Application SW Window (client)



```
#    [SIM]  1    ADDED          /umas.187070359034322
#    [SIM]  Request to deallocate "/umas.187070359034322" ...
#    [SIM]  1    REMOVED        /umas.187070359034322
#    [SIM]  Request to deallocate "/mmio.187070359034322" ...
#    [SIM]  0    REMOVED        /mmio.187070359034322
#    [SIM]  ASE recognized a SW simkill (see ase.cfg)... Simulator will EXIT
#    [SIM]  SIM-C : Exiting event socket server@/tmp/ase_event_server_187070359034322...
#    [SIM]  Closing message queue and unlinking...
#    [SIM]  Unlinking Shared memory regions....
#    [SIM]  Session code file removed
#    [SIM]  Removing message queues and buffer handles ...
#    [SIM]  Cleaning session files...
#    [SIM]  Simulation generated log files
#    [SIM]          Transactions file       | $ASE_WORKDIR/ccip_transactions.tsv
#    [SIM]          Workspaces info         | $ASE_WORKDIR/workspace_info.log
#    [SIM]          ASE seed                | $ASE_WORKDIR/ase_seed.txt
#    [SIM]
#    [SIM]  Tests run     => 1
#    [SIM]
#    [SIM]  Sending kill command...
#    [SIM]  Simulation kill command received...
#
#    Transaction count   |       VA       VL0       VH0       VH1 |    MCL-1    MCL-2    MCL-4
#    ==========================================================================================
#    MMIOWrReq          2 |
#    MMIORdReq         10 |
#    MMIORdRsp         10 |
#    IntrReq            0 |
#    IntrResp           0 |
#    RdReq              0 |        0         0         0         0 |        0        0        0
#    RdResp             0 |        0         0         0         0 |
#    WrReq              0 |        0         0         0         0 |        0        0        0
#    WrResp             0 |        0         0         0         0 |        0        0        0
#    WrFence            0 |        0         0         0         0 |
#    WrFenRsp           0 |        0         0         0         0 |
#
# ** Note: $finish    : /home/student/fpga_trn/AccelStack_Workshop/hello_afu/build_sim/rtl/cci
4)
#    Time: 21620047500 ps  Iteration: 2  Instance: /ase_top/ccip_emulator
# End time: 12:34:40 on Aug 21,2018, Elapsed time: 0:28:57
# Errors: 0, Warnings: 3
/home/student/fpga trn/AccelStack Workshop/hello afu/build sim
```

AFU Simulator Window (server)

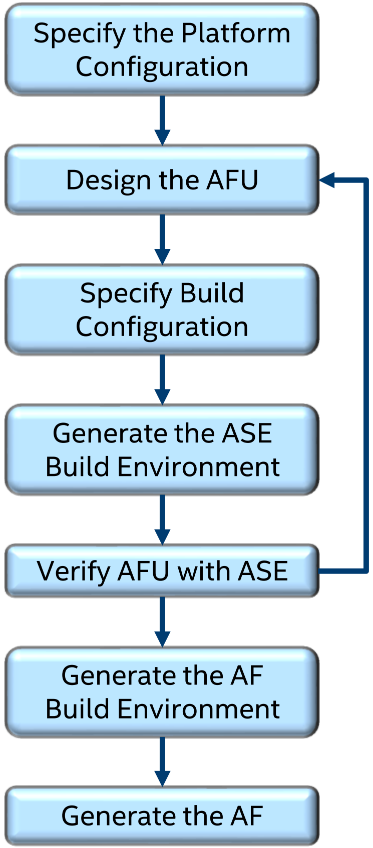
# AFU Development Flow Using OPAE SDK

## Generate the AF build environment:

- *cd $OPAE_PLATFORM_ROOT/hw/samples/hello_afu*
- *afu_synth_setup --source hw/rtl/filelist.txt build_synth*

## Generate the AF

- *cd build_synth*
- *$OPAE_PLATFORM_ROOT/bin/run.sh*



Specify the Platform Configuration
Design the AFU
Specify Build Configuration
Generate the ASE Build Environment
Verify AFU with ASE
Generate the AF Build Environment
Generate the AF

# Using the Quartus GUI

Compiling the AFU uses a command line-driven PR compilation flow

- Builds PR region AF as a .gbs file to be loaded into OPAE hardware platform

Can use the Quartus GUI for the following types of work:

- Viewing compilation reports
- Interactive Timing Analysis
- Adding SignalTap instances and nodes
  - For on-board debugging

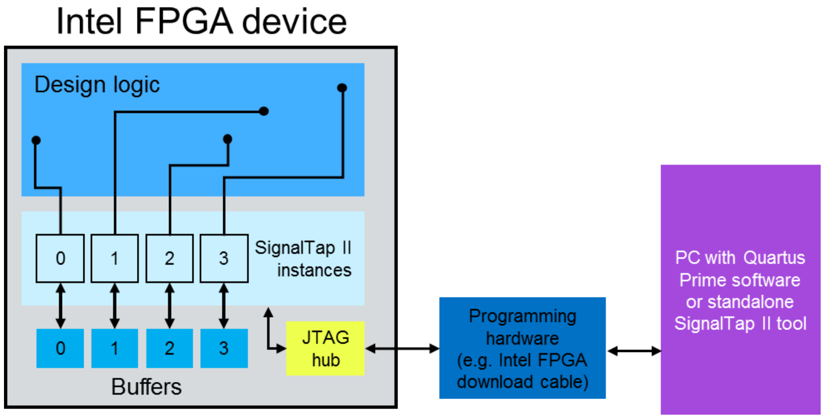
# AFU Debug with Remote SignalTap

## Introduction

Remote SignalTap enables in-system debug of AFUs on PAC installations with limited physical access

Remote debug capability in OPAE supports the following in-system debug tools included with Quartus Prime Pro:

- In-system sources and probes
- In-system memory content editor
- Signal Probe
- System Console

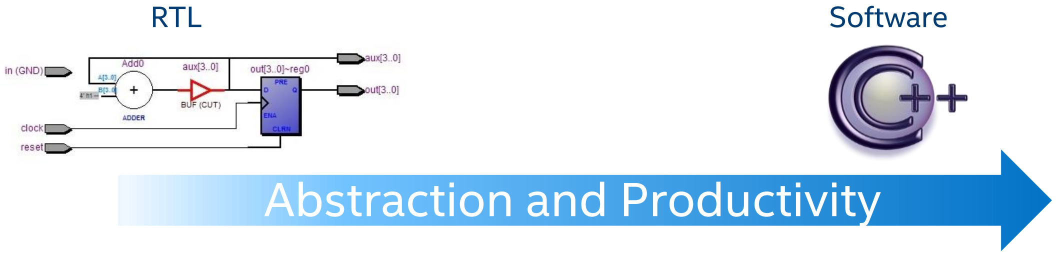

Intel FPGA device
Design logic
0
1
2
3
SignalTap II instances
0
1
2
3
Buffers
JTAG hub
Programming hardware (e.g. Intel FPGA download cable)
PC with Quartus Prime software or standalone SignalTap II tool

# AFU Design Using High Level Synthesis (HLS)

## Leverage GNU compatible HLS compiler to produce verified RTL

## Designing at a higher level of abstraction = increase productivity

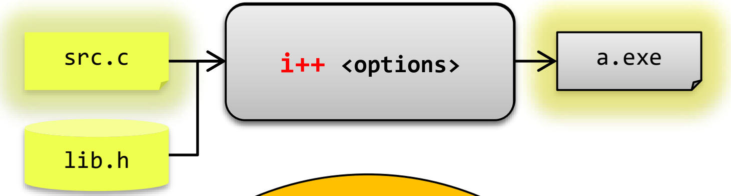
- Debugging software is much faster than hardware
- Easier to specify functions in software
- Simulation of RTL takes thousands times longer than software
- Easier to modify C/C++ source than RTL

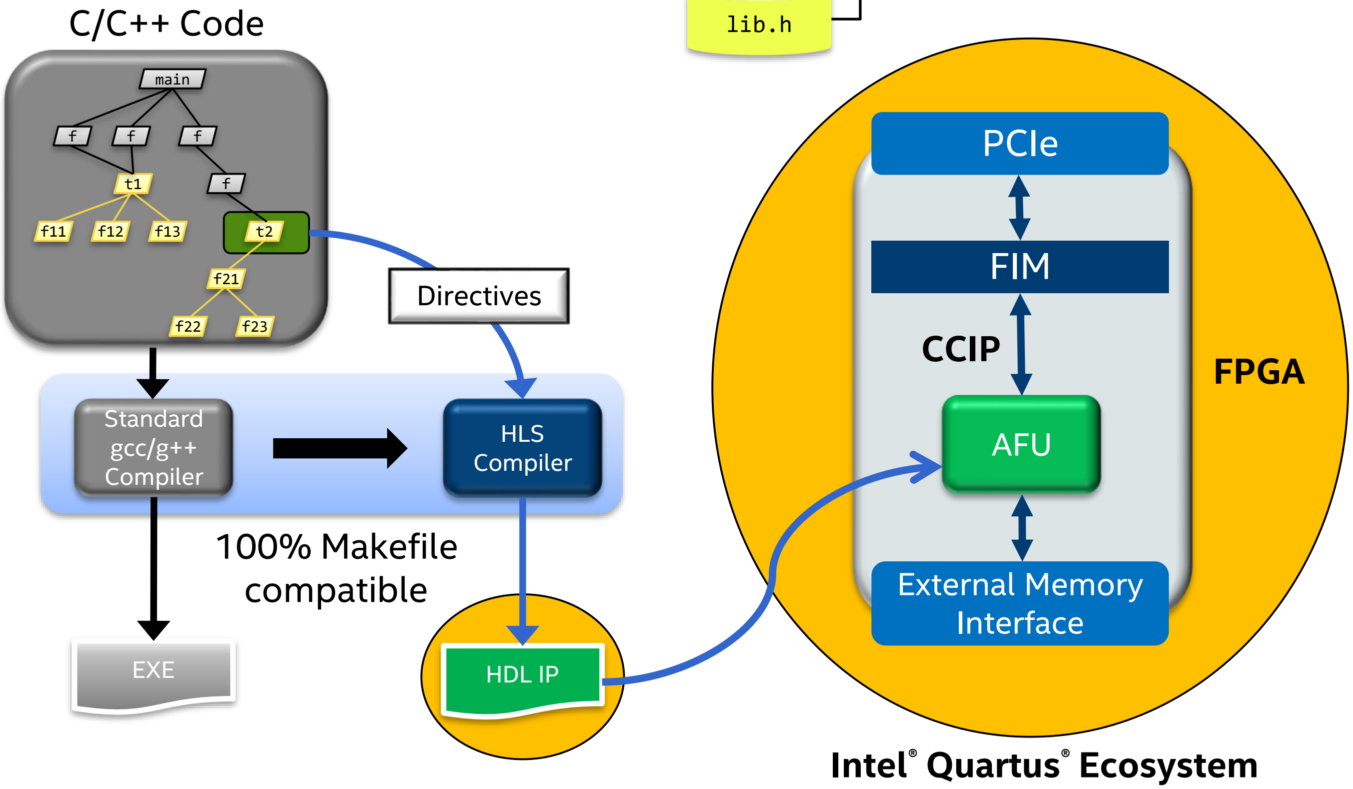

RTL

Software

Abstraction and Productivity

# HLS Use Model



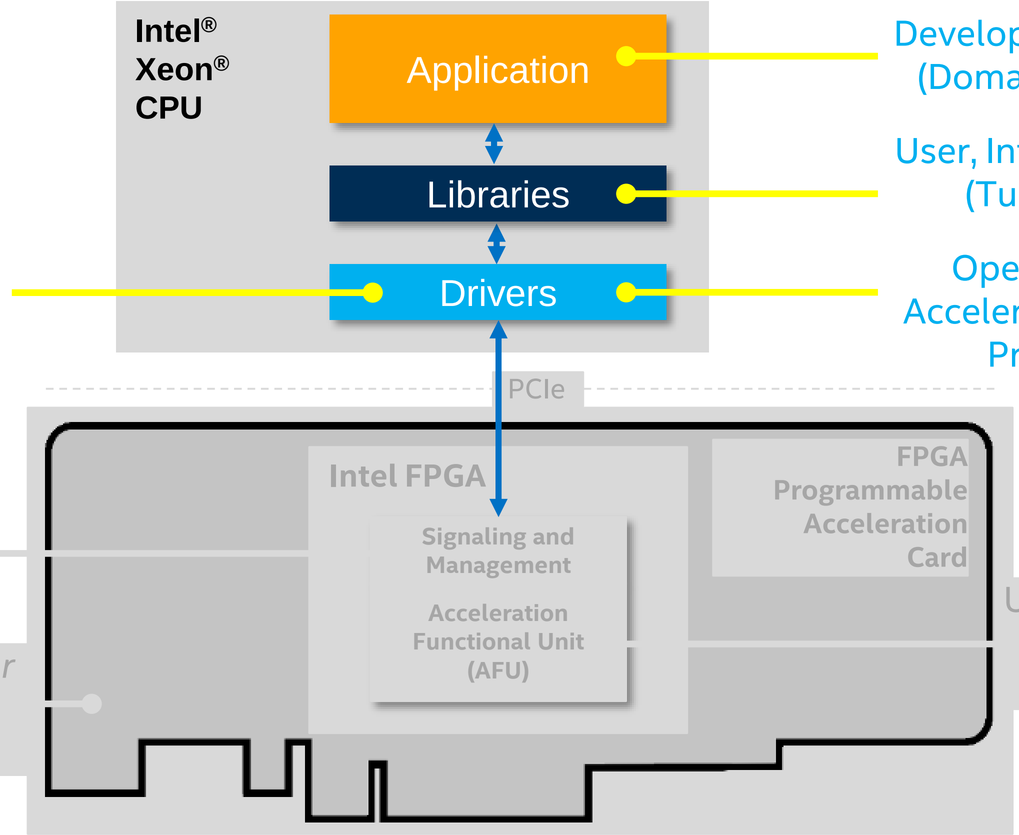


Intel® Quartus® Ecosystem

https://www.intel.com/content/www/us/en/programmable/documentation/div1537518568620.html

# APPLICATION DEVELOPMENT ON THE ACCELERATION STACK

# Components of Acceleration Stack: Overview



Developed by User (Domain Expert)

User, Intel, and 3rd Party (Tuning Expert)

Open Programmable Acceleration Engine (OPAE) Provided by Intel

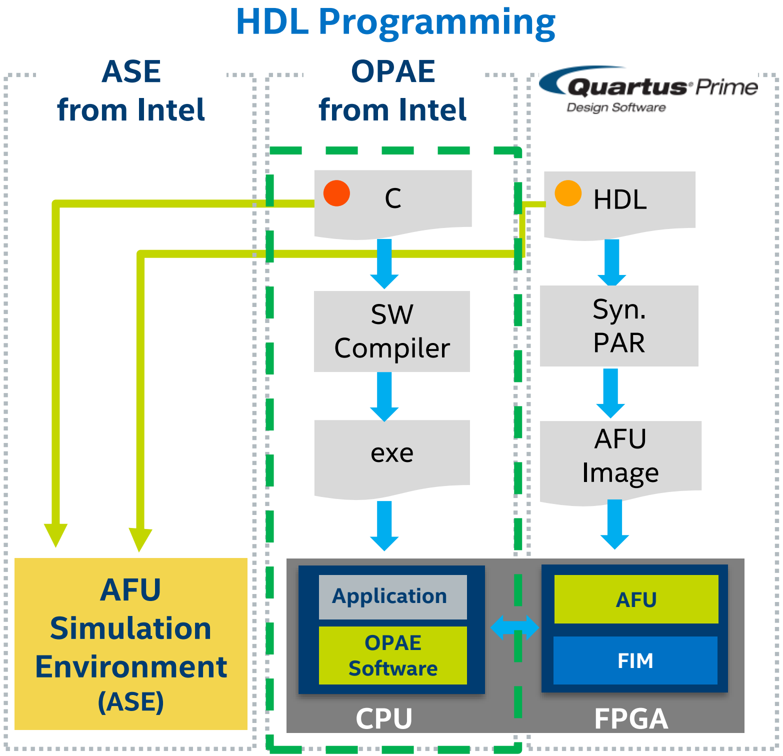
PCIe* Drivers Provided by Intel

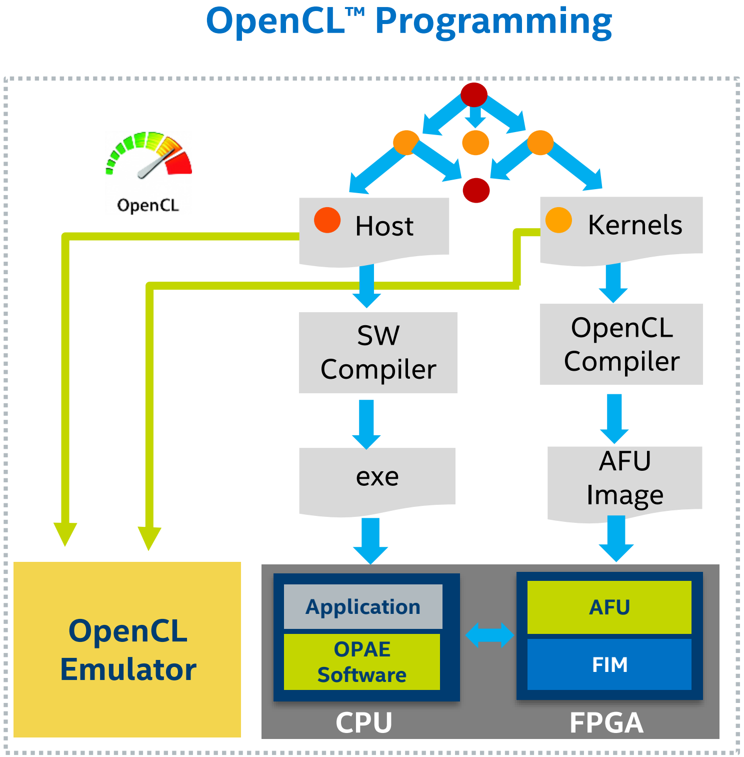
*FPGA Interface Manager* Provided by Intel

*Qualified and Validated for volume deployment* Provided by OEMs

User, Intel, or 3rd-Party IP Plugs into AFU Slot (Tuning Expert)

# Co-Design for HW and SW





# Open Programmable Acceleration Engine (OPAE)

**Consistent API across product generations and platforms**

- Abstraction for hardware specific FPGA resource details

**Designed for minimal software overhead and latency**

- Lightweight user-space library *(libfpga)*

**Open ecosystem for industry and developer community**

- License: FPGA API (BSD), FPGA driver (GPLv2)

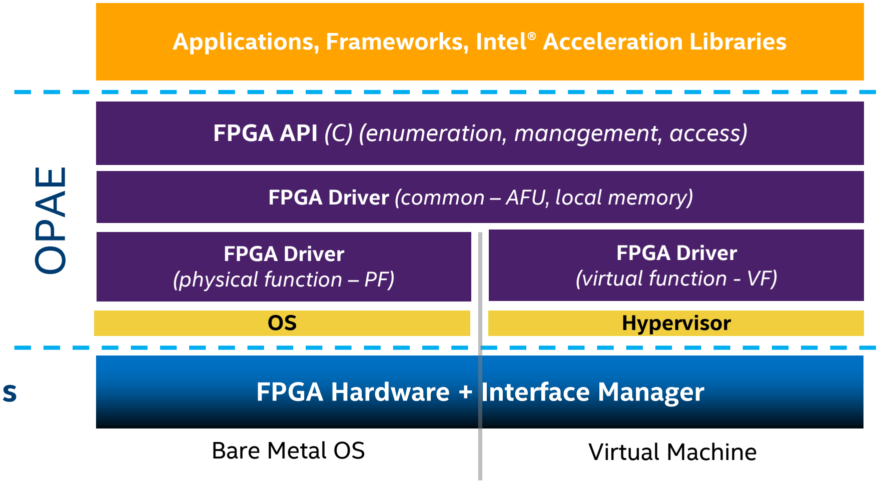
**FPGA driver being upstreamed into Linux kernel**

**Supports both virtual machines and bare metal platforms**

**Faster development and debugging of Accelerator Functions with the included AFU Simulation Environment (ASE)**

**Includes guides, command-line utilities and sample code**

Simplified FPGA Programming Model for Application Developers


Applications, Frameworks, Intel® Acceleration Libraries
OPAE
FPGA API (C) (enumeration, management, access)
FPGA Driver (common – AFU, local memory)
FPGA Driver (physical function – PF)
FPGA Driver (virtual function - VF)
OS
Hypervisor
FPGA Hardware + Interface Manager
Bare Metal OS
Virtual Machine

CentOS redhat ubuntu

Start developing for Intel FPGAs with OPAE today: http://01.org/OPAE

# The OPAE Library at a Glance

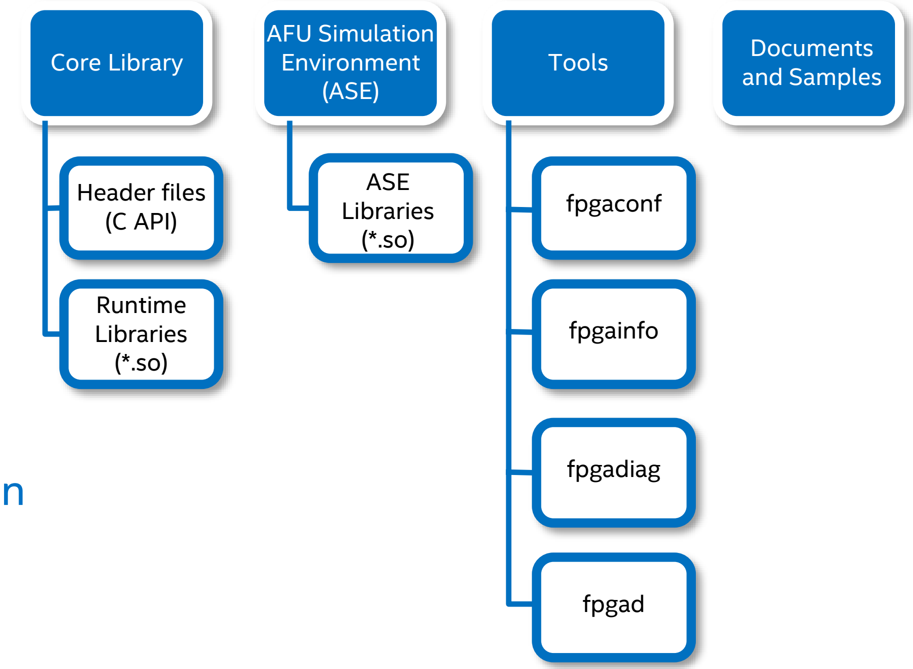
Enumerate, access, and manage FPGA resources through API objects

A common interface across different FPGA form factors

C API designed for extensibility

AFU Simulation Environment (ASE) allows developing and debugging accelerator functions and software applications without an FPGA

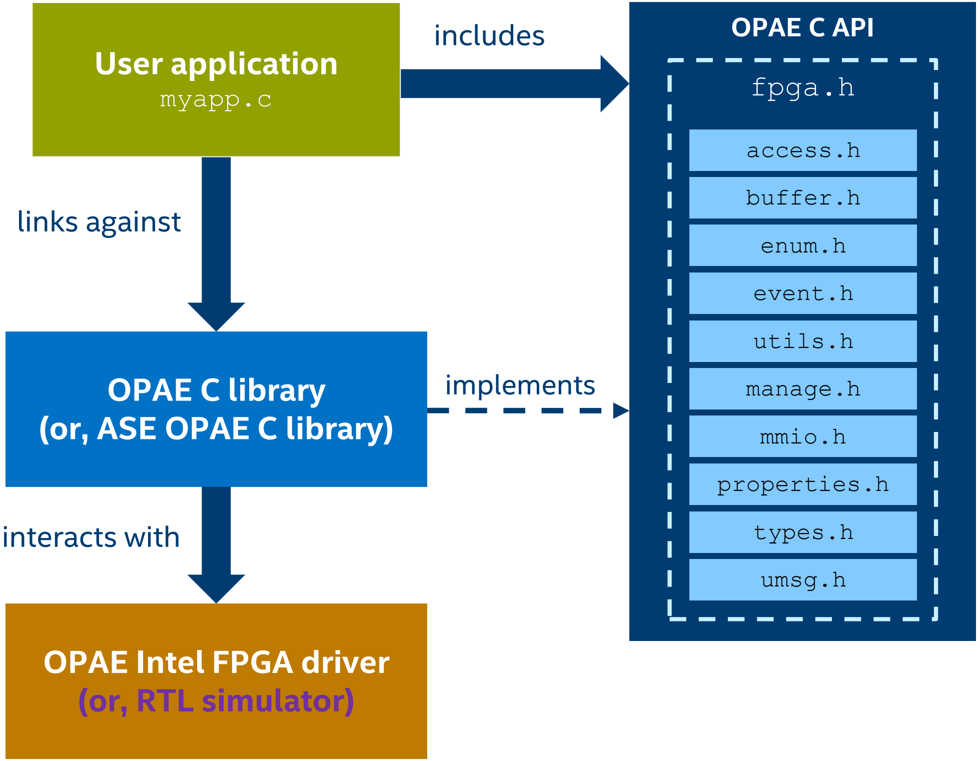
Tools for partial reconfiguration, FPGA hardware information, error reporting, etc.



# Useful OPAE Command-line Utilities For Board Management

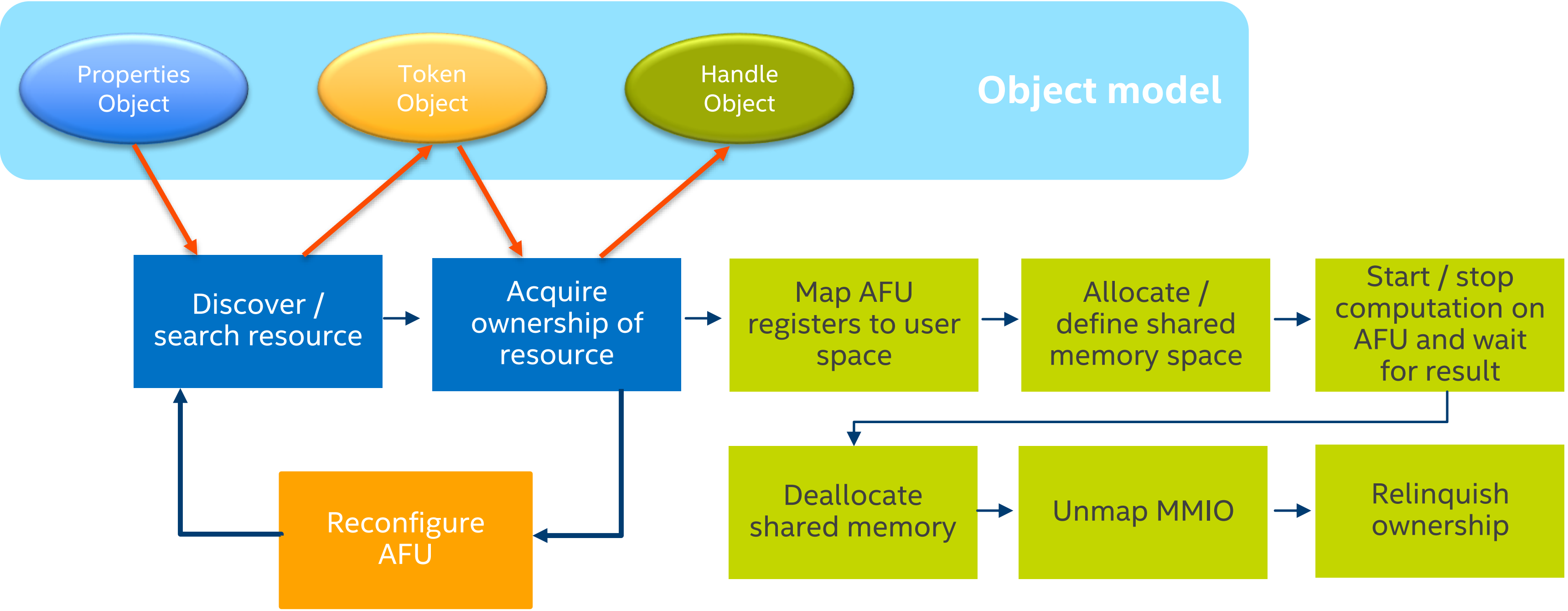
| Commands | Description |
|---|---|
| fpgainfo | User can read the Board Telemetry data. For example temperature or Voltages. |
| fpgabist | Performs self-diagnostic test: measure bandwidth between local DDR4 memory and system memory |
| fpgaconf | Configure Acceleration Function Unit (AFU) into FPGA;<br>Check compatibility with targeted FPGA and FIM |
| fpgaflash | Updates FPGA Interface Manager (FIM) image (.rpd file) being stored in flash;<br>Updates BMC firmware. |
| fpgad | A daemon to monitor FPGA drivers' error status; report errors as events to OPAE |

# Application Development with OPAE

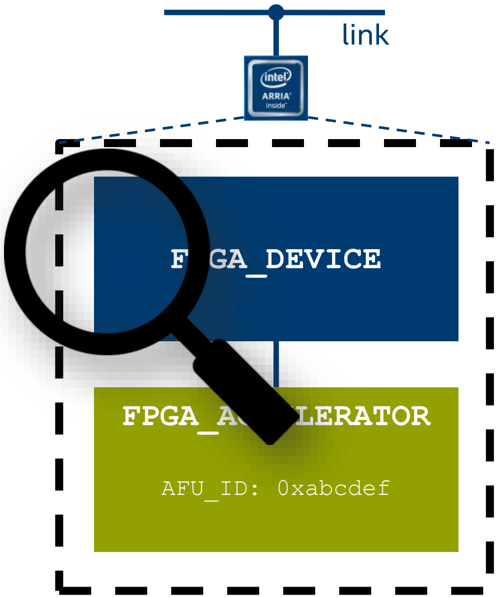


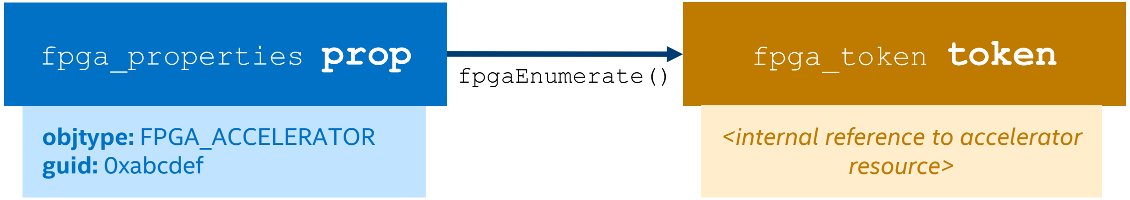
User application
myapp.c
includes
links against
OPAE C API
fpga.h
access.h
buffer.h
enum.h
event.h
utils.h
manage.h
mmio.h
properties.h
types.h
umsg.h
OPAE C library
(or, ASE OPAE C library)
implements
interacts with
OPAE Intel FPGA driver
(or, RTL simulator)

# The OPAE Library Programming Model


Properties Object
Token Object
Handle Object
Object model
Discover / search resource
Acquire ownership of resource
Map AFU registers to user space
Allocate / define shared memory space
Start / stop computation on AFU and wait for result
Reconfigure AFU
Deallocate shared memory
Unmap MMIO
Relinquish ownership

# Enumeration and Discovery





fpga_properties **prop**

**objtype:** FPGA_ACCELERATOR
**guid:** 0xabcdef

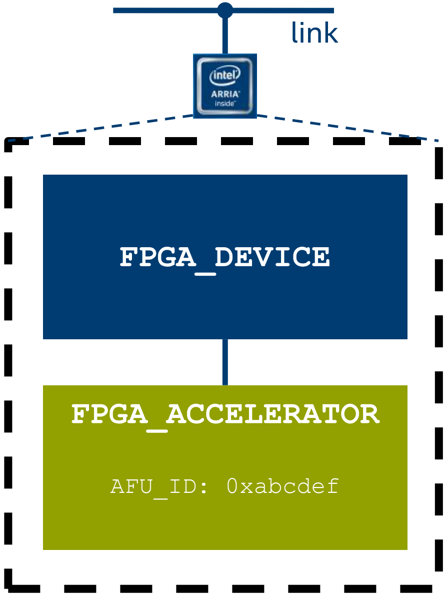
fpgaEnumerate()

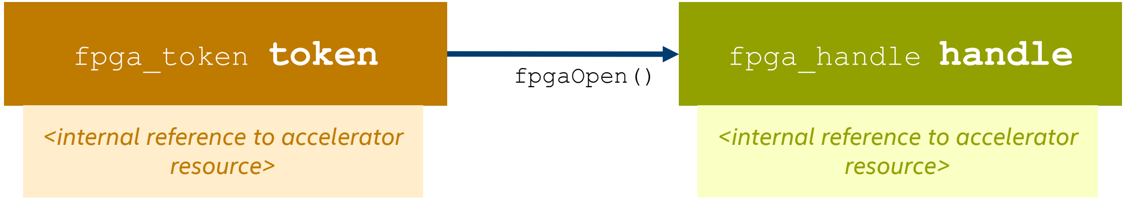
fpga_token **token**

*<internal reference to accelerator resource>*

```
fpga_properties prop;
fpga_token      token;
fpga_guid       myguid;   /* 0xabcdef */

fpgaGetProperties(NULL, &prop);

fpgaPropertiesSetObjectType(prop, FPGA_ACCELERATOR);
fpgaPropertiesSetGUID(prop, myguid);

fpgaEnumerate(&prop, 1, &token, 1, &n);

fpgaDestroyProperties(&prop);
```

# Acquire and Release Accelerator Resource





fpga_token **token**

*<internal reference to accelerator resource>*

fpgaOpen()

fpga_handle **handle**

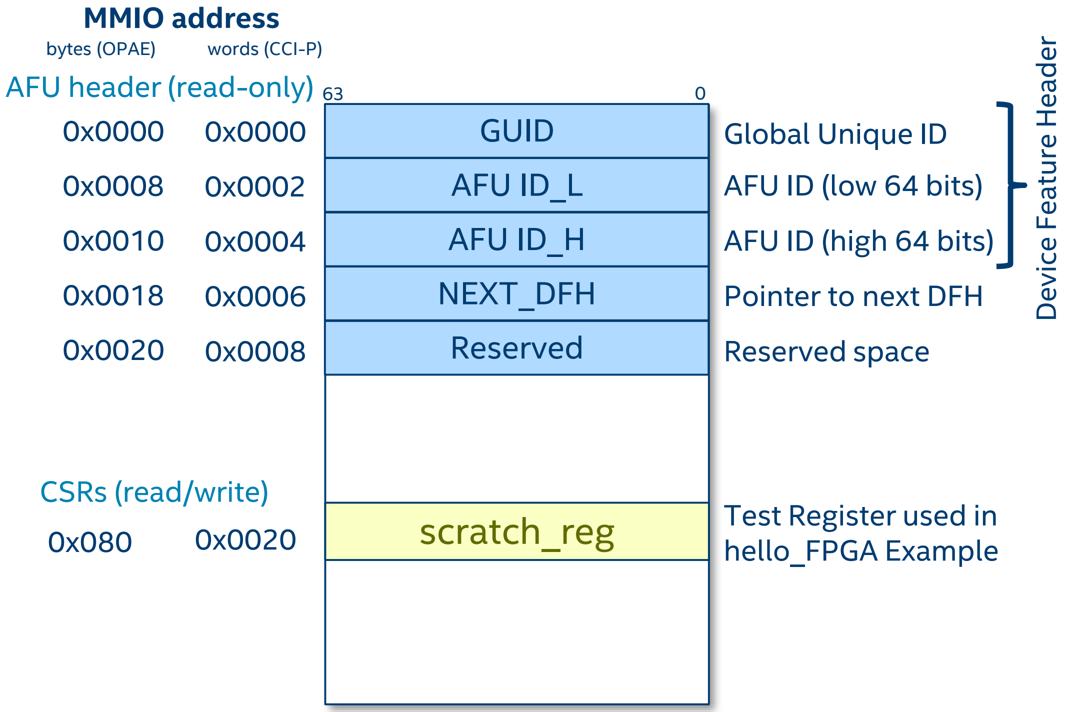
*<internal reference to accelerator resource>*

```
fpga_token token;
// ... enumeration ...
fpga_handle handle;

fpgaOpen(token, &handle, 0);
.
.(operations...)
.(operations...)
.
.
.
fpgaClose(handle);
```

# Software Developer Needs AFU Specification

## Memory mapped register space

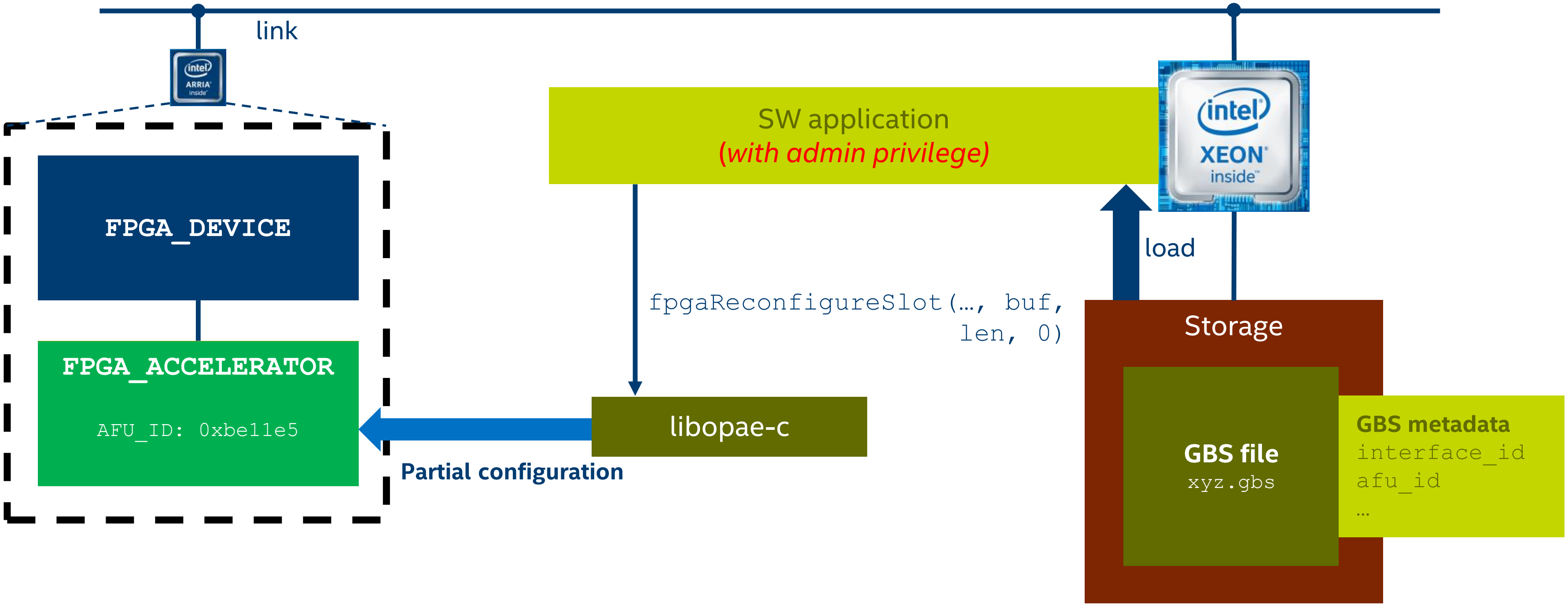
- Software uses to discover, control and communicate with FPGA accelerator
  - Report status flags
  - Configure AFU settings
  - Start/Stop control of acceleration workload



**MMIO address**

| bytes (OPAE) | words (CCI-P) | 63 … 0 | |
|---|---|---|---|
| AFU header (read-only) | | | |
| 0x0000 | 0x0000 | GUID | Global Unique ID |
| 0x0008 | 0x0002 | AFU ID_L | AFU ID (low 64 bits) |
| 0x0010 | 0x0004 | AFU ID_H | AFU ID (high 64 bits) |
| 0x0018 | 0x0006 | NEXT_DFH | Pointer to next DFH |
| 0x0020 | 0x0008 | Reserved | Reserved space |
| CSRs (read/write) | | | |
| 0x080 | 0x0020 | scratch_reg | Test Register used in hello_FPGA Example |

# Management and Reconfiguration



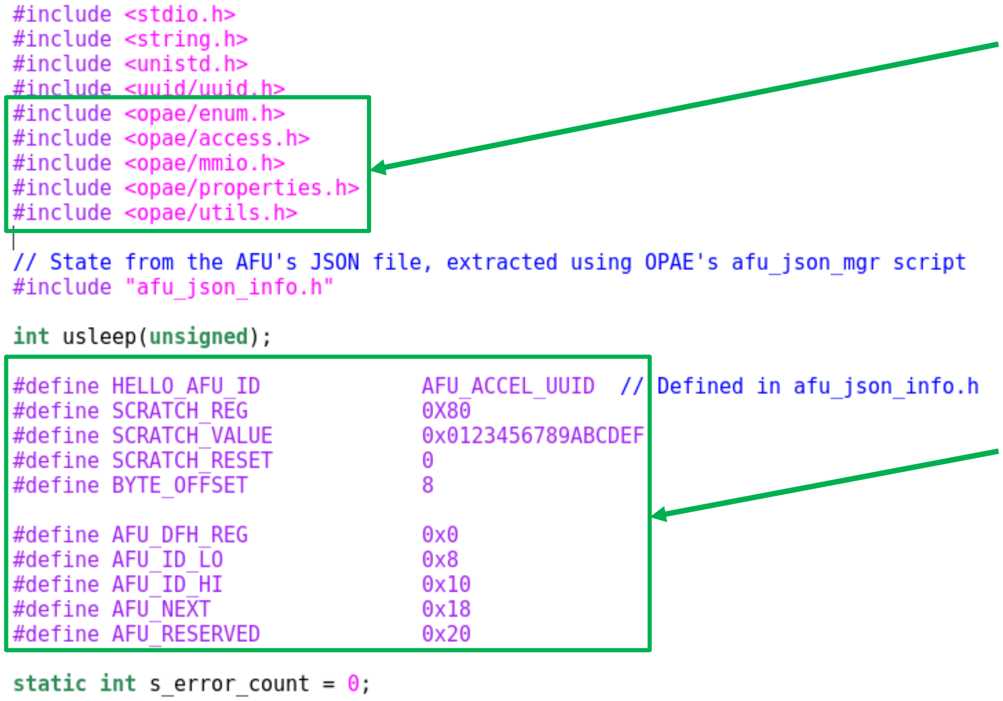
# A Code Example - Put Everything Together

The hello_afu.c code in the $OPAE_PLATFORM_ROOT/hw/samples directory of the OPAE library

- Demonstrates all OPAE API functions discussed in this presentation
- Write and read configuration registers from the host to the FPGA to show basic configuration accesses are done
- The same flow can be used to access and exercise any other AFUs

To compile source code run appropriate gcc/make commands

# Deeper Look into *hello_fpga.c*

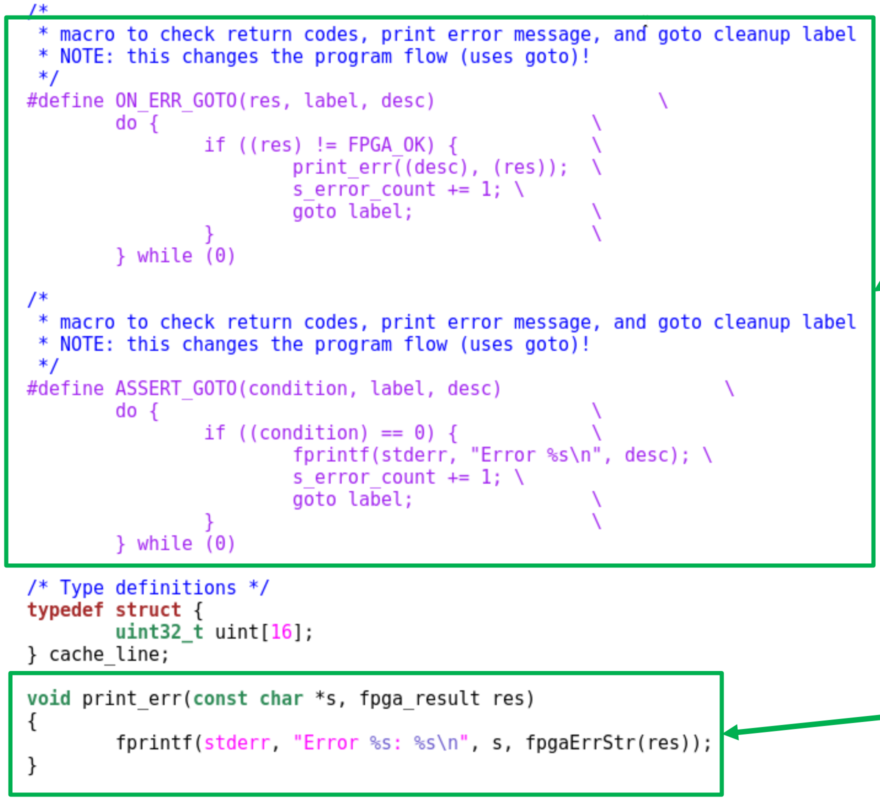


```c
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <uuid/uuid.h>
#include <opae/enum.h>
#include <opae/access.h>
#include <opae/mmio.h>
#include <opae/properties.h>
#include <opae/utils.h>

// State from the AFU's JSON file, extracted using OPAE's afu_json_mgr script
#include "afu_json_info.h"

int usleep(unsigned);

#define HELLO_AFU_ID              AFU_ACCEL_UUID  // Defined in afu_json_info.h
#define SCRATCH_REG               0X80
#define SCRATCH_VALUE             0x0123456789ABCDEF
#define SCRATCH_RESET             0
#define BYTE_OFFSET               8

#define AFU_DFH_REG               0x0
#define AFU_ID_LO                 0x8
#define AFU_ID_HI                 0x10
#define AFU_NEXT                  0x18
#define AFU_RESERVED              0x20

static int s_error_count = 0;
```

Include OPAE header files

Define constants that will be used when communicating with FPGA accelerator

# Deeper Look into *hello_fpga.c*

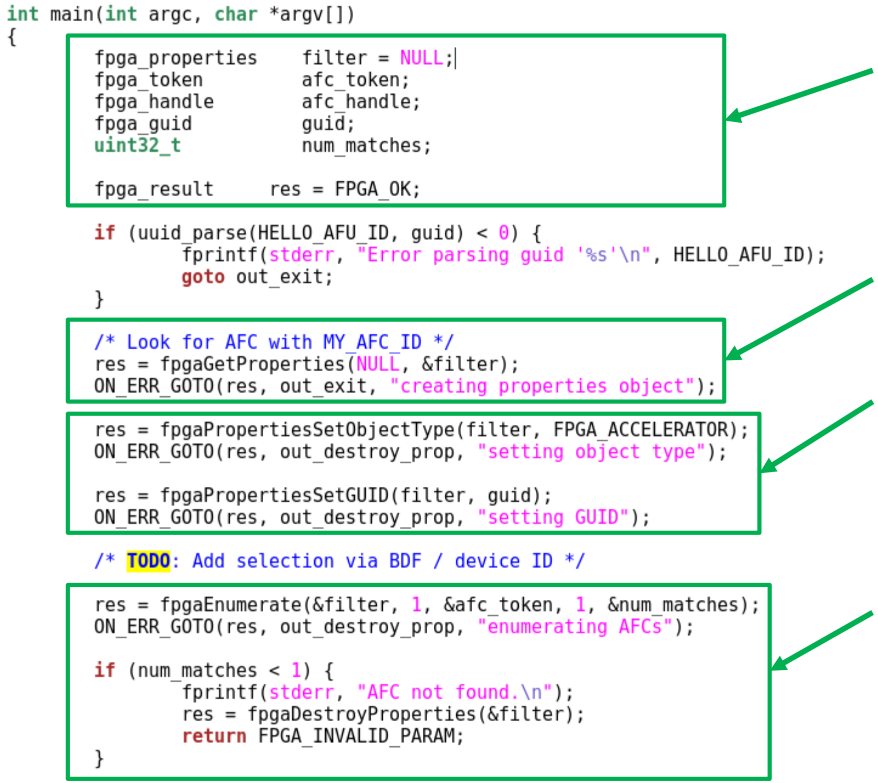


```
/*
 * macro to check return codes, print error message, and goto cleanup label
 * NOTE: this changes the program flow (uses goto)!
 */
#define ON_ERR_GOTO(res, label, desc)                    \
        do {                                            \
                if ((res) != FPGA_OK) {                 \
                        print_err((desc), (res));       \
                        s_error_count += 1; \
                        goto label;                     \
                }                                       \
        } while (0)

/*
 * macro to check return codes, print error message, and goto cleanup label
 * NOTE: this changes the program flow (uses goto)!
 */
#define ASSERT_GOTO(condition, label, desc)                     \
        do {                                            \
                if ((condition) == 0) {                 \
                        fprintf(stderr, "Error %s\n", desc); \
                        s_error_count += 1; \
                        goto label;                     \
                }                                       \
        } while (0)

/* Type definitions */
typedef struct {
        uint32_t uint[16];
} cache_line;

void print_err(const char *s, fpga_result res)
{
        fprintf(stderr, "Error %s: %s\n", s, fpgaErrStr(res));
}
```

Error handling macro evaluates fpga_result object, *res*

Error printing function

# Deeper Look into *hello_fpga.c*



```
int main(int argc, char *argv[])
{
    fpga_properties    filter = NULL;
    fpga_token         afc_token;
    fpga_handle        afc_handle;
    fpga_guid          guid;
    uint32_t           num_matches;

    fpga_result     res = FPGA_OK;

    if (uuid_parse(HELLO_AFU_ID, guid) < 0) {
            fprintf(stderr, "Error parsing guid '%s'\n", HELLO_AFU_ID);
            goto out_exit;
    }

    /* Look for AFC with MY_AFC_ID */
    res = fpgaGetProperties(NULL, &filter);
    ON_ERR_GOTO(res, out_exit, "creating properties object");

    res = fpgaPropertiesSetObjectType(filter, FPGA_ACCELERATOR);
    ON_ERR_GOTO(res, out_destroy_prop, "setting object type");

    res = fpgaPropertiesSetGUID(filter, guid);
    ON_ERR_GOTO(res, out_destroy_prop, "setting GUID");

    /* TODO: Add selection via BDF / device ID */

    res = fpgaEnumerate(&filter, 1, &afc_token, 1, &num_matches);
    ON_ERR_GOTO(res, out_destroy_prop, "enumerating AFCs");

    if (num_matches < 1) {
            fprintf(stderr, "AFC not found.\n");
            res = fpgaDestroyProperties(&filter);
            return FPGA_INVALID_PARAM;
    }
```

Create variables and objects that will be used when communicating with FPGA accelerator

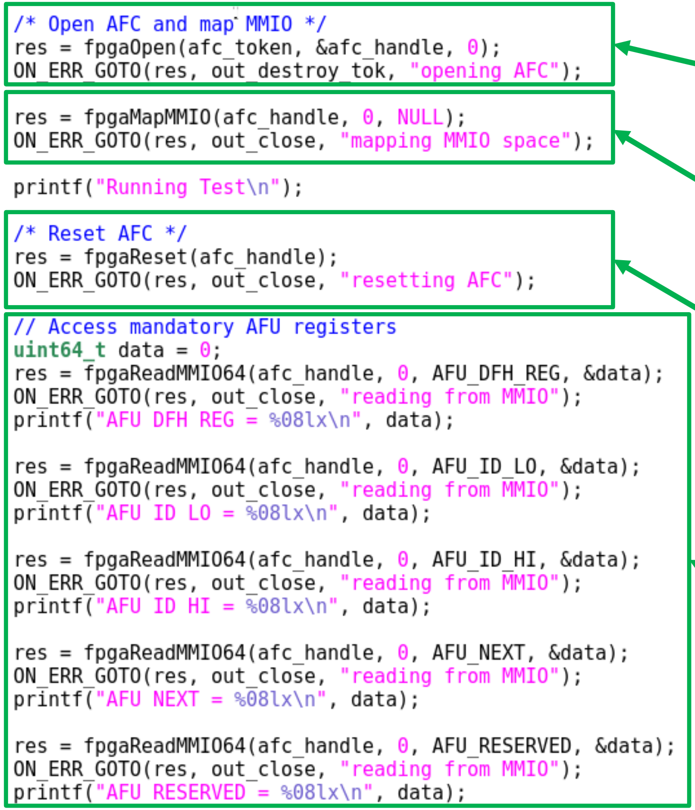
Create an empty FPGA properties object

Populate the opaque FPGA properties object with desired search parameters

Search for matching FPGA resources using *fpga_Enumerate()* which returns the list of matches to the fpga_token, *afc_token*

- Error and destroy object if none are found

# Deeper Look into *hello_fpga.c*



```
/* Open AFC and map MMIO */
res = fpgaOpen(afc_token, &afc_handle, 0);
ON_ERR_GOTO(res, out_destroy_tok, "opening AFC");

res = fpgaMapMMIO(afc_handle, 0, NULL);
ON_ERR_GOTO(res, out_close, "mapping MMIO space");

printf("Running Test\n");

/* Reset AFC */
res = fpgaReset(afc_handle);
ON_ERR_GOTO(res, out_close, "resetting AFC");

// Access mandatory AFU registers
uint64_t data = 0;
res = fpgaReadMMIO64(afc_handle, 0, AFU_DFH_REG, &data);
ON_ERR_GOTO(res, out_close, "reading from MMIO");
printf("AFU DFH REG = %08lx\n", data);

res = fpgaReadMMIO64(afc_handle, 0, AFU_ID_LO, &data);
ON_ERR_GOTO(res, out_close, "reading from MMIO");
printf("AFU ID LO = %08lx\n", data);

res = fpgaReadMMIO64(afc_handle, 0, AFU_ID_HI, &data);
ON_ERR_GOTO(res, out_close, "reading from MMIO");
printf("AFU ID HI = %08lx\n", data);

res = fpgaReadMMIO64(afc_handle, 0, AFU_NEXT, &data);
ON_ERR_GOTO(res, out_close, "reading from MMIO");
printf("AFU NEXT = %08lx\n", data);

res = fpgaReadMMIO64(afc_handle, 0, AFU_RESERVED, &data);
ON_ERR_GOTO(res, out_close, "reading from MMIO");
printf("AFU RESERVED = %08lx\n", data);
```

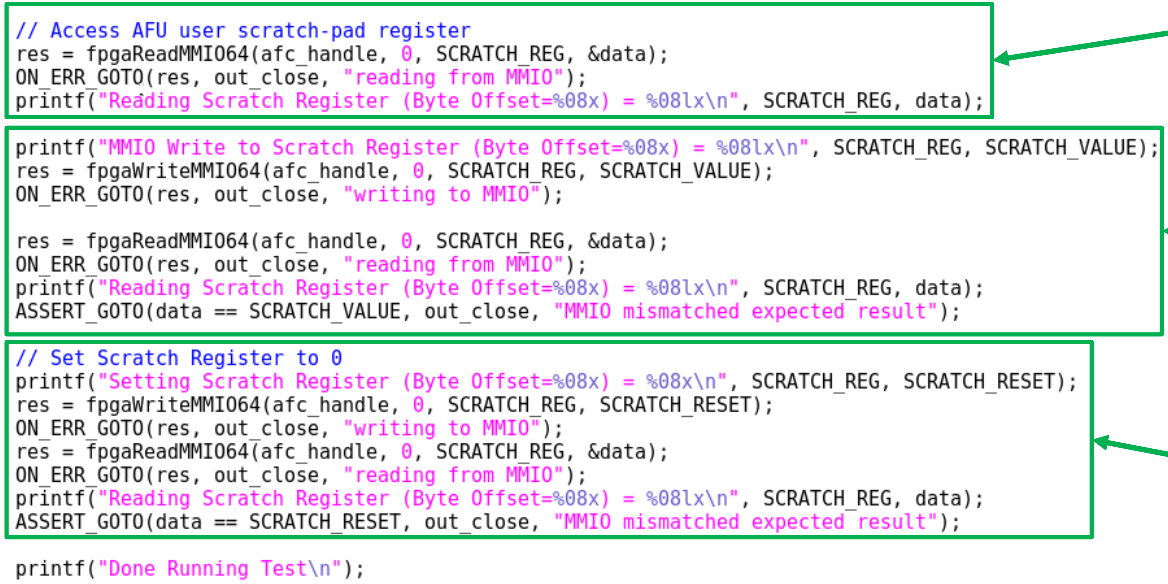
Acquire ownership of resource pointed to by *afc_token* using fpga_Open() receiving the fpga_handle, *afc_handle*

Map accelerator register space to user space

Reset the Accelerator Function using the *fpgaReset* API

Read the Device Feature Header registers from the AFU and print them to screen

# Deeper Look into *hello_fpga.c*



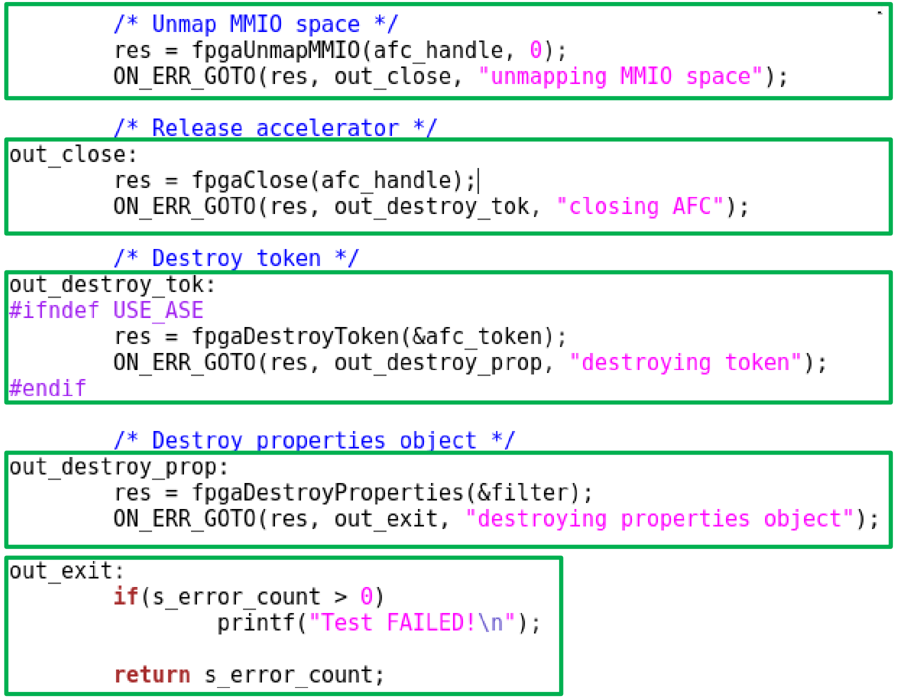
```
// Access AFU user scratch-pad register
res = fpgaReadMMIO64(afc_handle, 0, SCRATCH_REG, &data);
ON_ERR_GOTO(res, out_close, "reading from MMIO");
printf("Reading Scratch Register (Byte Offset=%08x) = %08lx\n", SCRATCH_REG, data);

printf("MMIO Write to Scratch Register (Byte Offset=%08x) = %08lx\n", SCRATCH_REG, SCRATCH_VALUE);
res = fpgaWriteMMIO64(afc_handle, 0, SCRATCH_REG, SCRATCH_VALUE);
ON_ERR_GOTO(res, out_close, "writing to MMIO");

res = fpgaReadMMIO64(afc_handle, 0, SCRATCH_REG, &data);
ON_ERR_GOTO(res, out_close, "reading from MMIO");
printf("Reading Scratch Register (Byte Offset=%08x) = %08lx\n", SCRATCH_REG, data);
ASSERT_GOTO(data == SCRATCH_VALUE, out_close, "MMIO mismatched expected result");

// Set Scratch Register to 0
printf("Setting Scratch Register (Byte Offset=%08x) = %08x\n", SCRATCH_REG, SCRATCH_RESET);
res = fpgaWriteMMIO64(afc_handle, 0, SCRATCH_REG, SCRATCH_RESET);
ON_ERR_GOTO(res, out_close, "writing to MMIO");
res = fpgaReadMMIO64(afc_handle, 0, SCRATCH_REG, &data);
ON_ERR_GOTO(res, out_close, "reading from MMIO");
printf("Reading Scratch Register (Byte Offset=%08x) = %08lx\n", SCRATCH_REG, data);
ASSERT_GOTO(data == SCRATCH_RESET, out_close, "MMIO mismatched expected result");

printf("Done Running Test\n");
```

Read the initial value of the scratch register

Write a new value to the scratch register and read back verify

Write configuration register to 0 and read back verify

# Deeper Look into *hello_fpga.c*



```
        /* Unmap MMIO space */
        res = fpgaUnmapMMIO(afc_handle, 0);
        ON_ERR_GOTO(res, out_close, "unmapping MMIO space");

        /* Release accelerator */
out_close:
        res = fpgaClose(afc_handle);
        ON_ERR_GOTO(res, out_destroy_tok, "closing AFC");

        /* Destroy token */
out_destroy_tok:
#ifndef USE_ASE
        res = fpgaDestroyToken(&afc_token);
        ON_ERR_GOTO(res, out_destroy_prop, "destroying token");
#endif

        /* Destroy properties object */
out_destroy_prop:
        res = fpgaDestroyProperties(&filter);
        ON_ERR_GOTO(res, out_exit, "destroying properties object");

out_exit:
        if(s_error_count > 0)
                printf("Test FAILED!\n");

        return s_error_count;
```

Unmap Register space

Release the accelerator for others to use

Destroy the token

Destroy the property object

If any errors occur during configuration register access, increase error count and print failure

# AFU DEVELOPMENT USING HLS

# AFU development using HLS - Agenda

- Introduction to High Level Synthesis
- HLS interfaces
- HLS AFU development flow

# Introduction to High-Level Synthesis

# Introduction to HLS - Agenda

- Introduction
- x86 Emulation
- Cosimulation
- Intel® Quartus® Software Integration

# High Level Synthesis

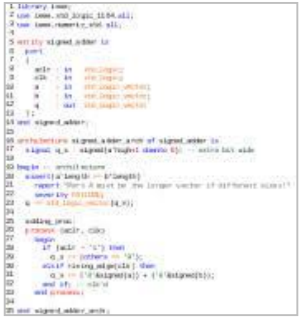
Synthesize a C/C++ function into an RTL implementation

- Develop the component in a software environment
- Verify the functionality of the component within a software environment
- Integrate it seamlessly with hardware simulation environment
- Optimize design using software-centric tools and reports
- Integrate generated IP easily within traditional FPGA design tools

# Traditional FPGA Design Process

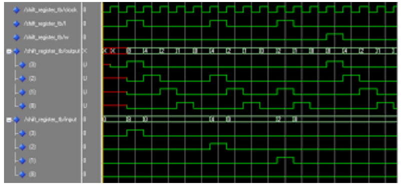
## Potentially Time-Consuming Effort

HDL

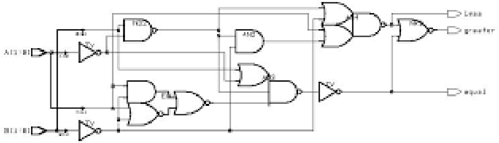


```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity signed_adder is
  port
  (
    aclr : in   std_logic;
    clk  : in   std_logic;
    a    : in   std_logic_vector;
    b    : in   std_logic_vector;
    q    : out  std_logic_vector
  );
end signed_adder;

architecture signed_adder_arch of signed_adder is
  signal q_s : signed(a'high+1 downto 0); -- extra bit wide

begin -- architecture
  assert(a'length >= b'length)
    report "Port A must be the longer vector if different sizes!"
    severity FAILURE;
  q <= std_logic_vector(q_s);

  adding_proc:
  process (aclr, clk)
    begin
      if (aclr = '1') then
        q_s <= (others => '0');
      elsif rising_edge(clk) then
        q_s <= ('0'&signed(a)) + ('0'&signed(b));
      end if; -- clk'd
    end process;

end signed_adder_arch;
```

Behavioral Simulation



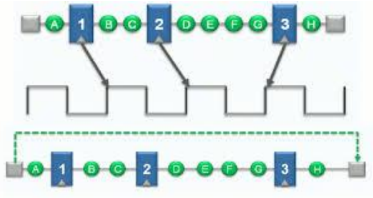
Synthesis



Place & Route / Timing Analysis / Timing Closure

**Intel® Quartus® Prime**

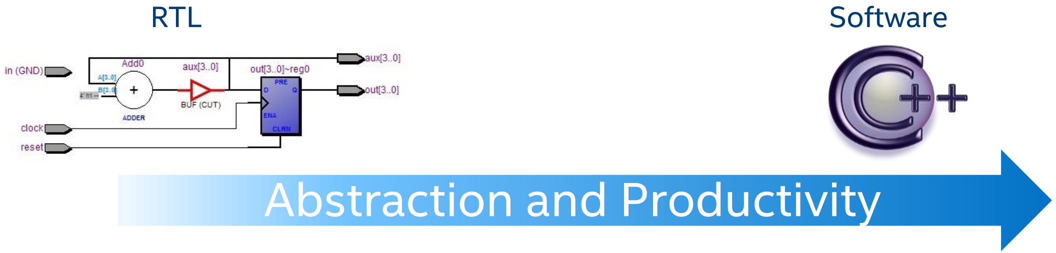
Design Software



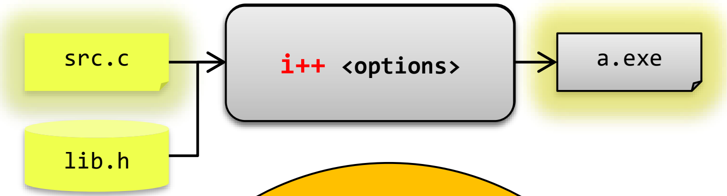
# Why HLS?

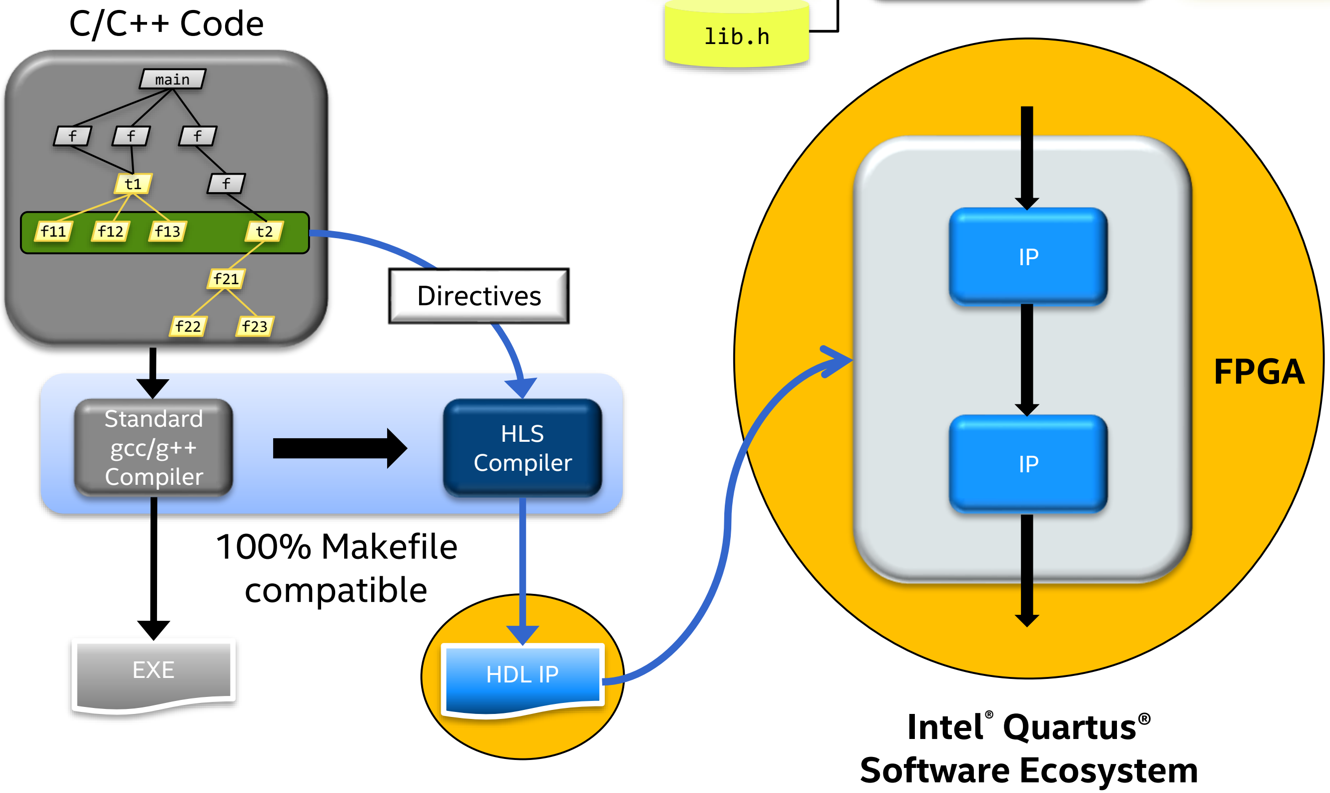
Designing at a higher level of abstraction = increase productivity

- Debugging software is much faster than hardware
- Easier to specify functions in software
- Simulation of RTL takes thousands times longer than software



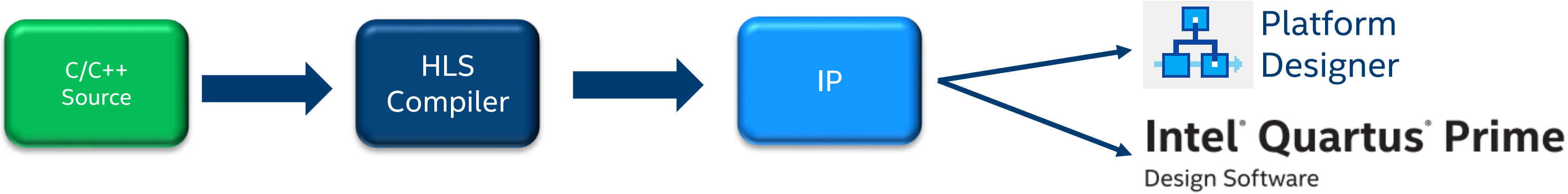
# HLS Use Model





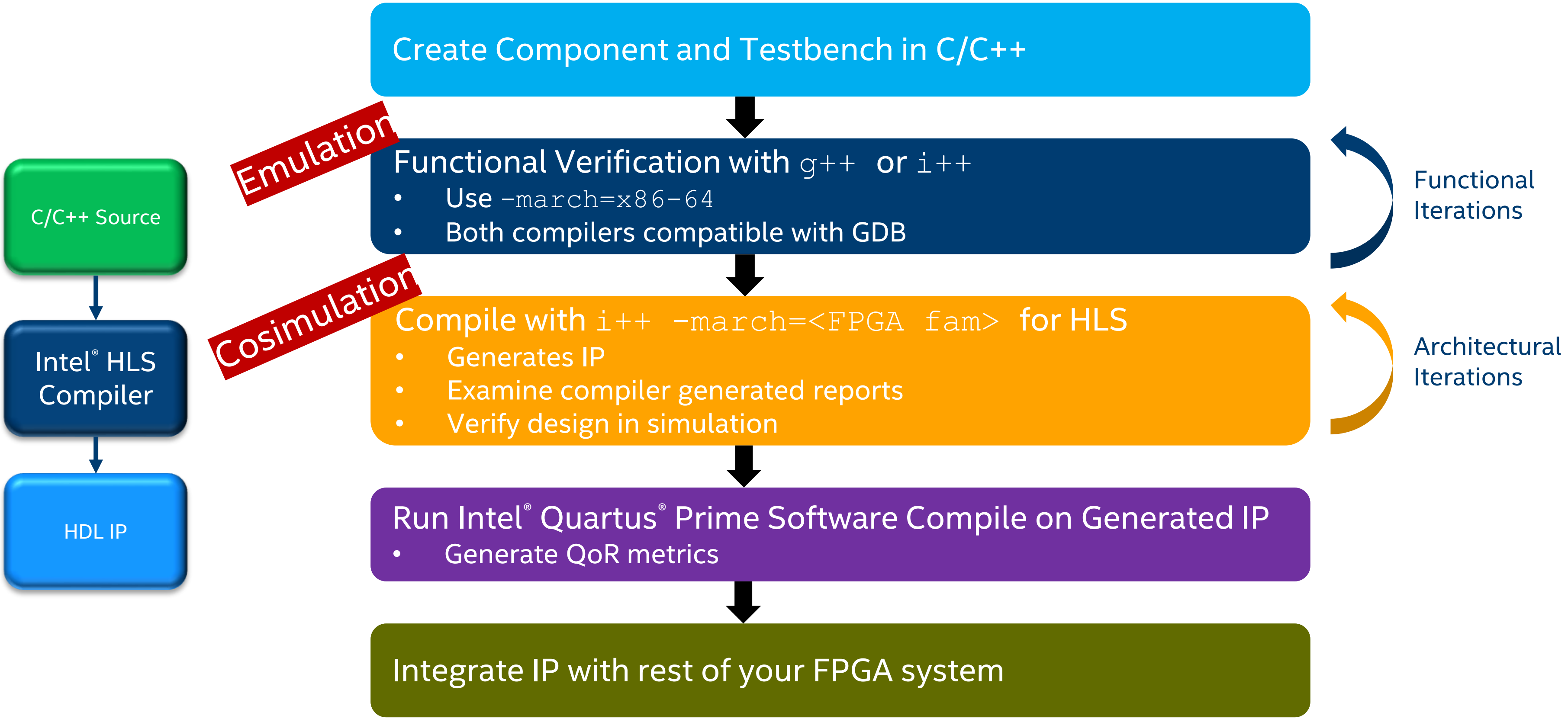
# Intel® HLS Compiler

- Targets Intel® FPGAs
- Command-line executable: `i++`
- Builds an IP block
  - To be integrated into a traditional FPGA design using FPGA tools



- Leverages standard C/C++ development environment
- Goal: Same performance as hand-coded RTL with 10-15% more resources

# HLS Procedure



C/C++ Source

Intel® HLS Compiler

HDL IP

Create Component and Testbench in C/C++

Emulation

Functional Verification with `g++` or `i++`

- Use `-march=x86-64`
- Both compilers compatible with GDB

Functional Iterations

Cosimulation

Compile with `i++ -march=<FPGA fam>` for HLS

- Generates IP
- Examine compiler generated reports
- Verify design in simulation

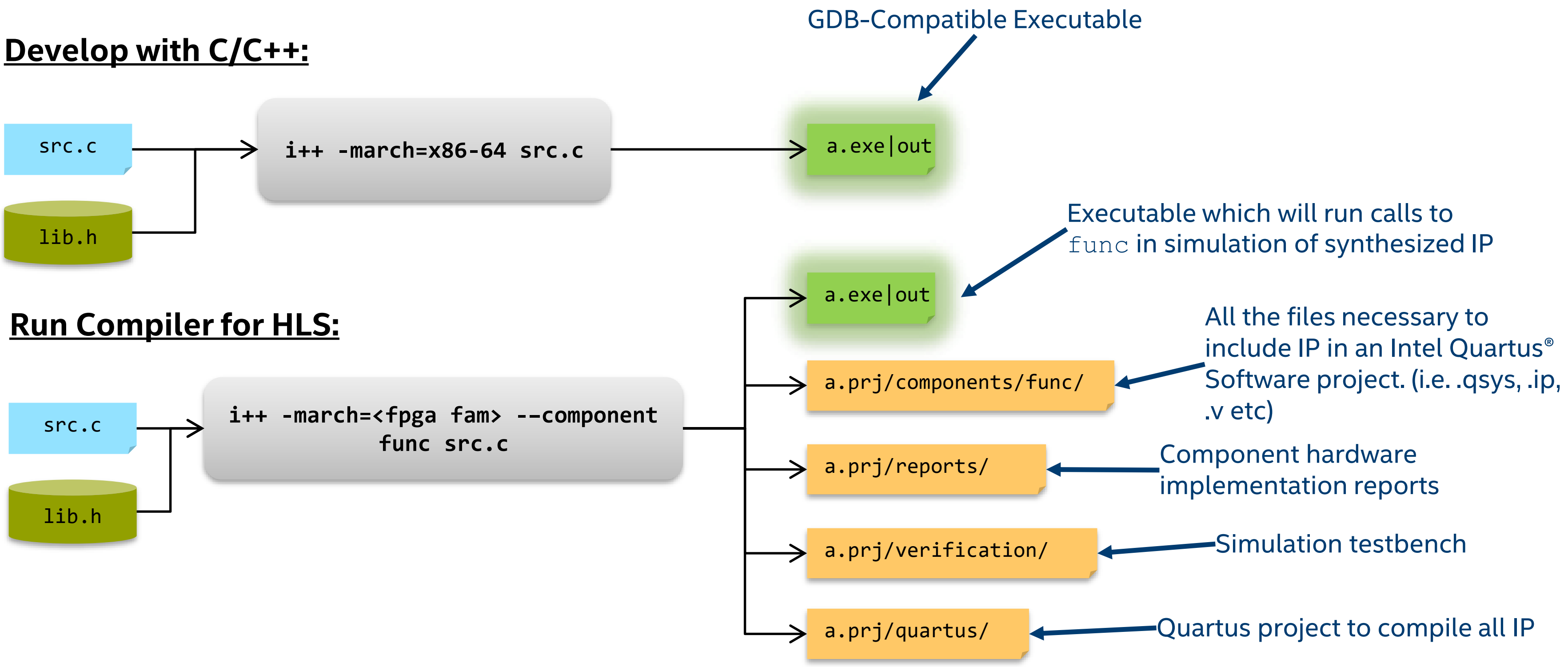
Architectural Iterations

Run Intel® Quartus® Prime Software Compile on Generated IP

- Generate QoR metrics

Integrate IP with rest of your FPGA system

# Intel® HLS Compiler Usage and Output



**Develop with C/C++:**

src.c

lib.h

```
i++ -march=x86-64 src.c
```

a.exe|out

GDB-Compatible Executable

**Run Compiler for HLS:**

src.c

lib.h

```
i++ -march=<fpga fam> --component
func src.c
```

a.exe|out

Executable which will run calls to `func` in simulation of synthesized IP

a.prj/components/func/

All the files necessary to include IP in an Intel Quartus® Software project. (i.e. .qsys, .ip, .v etc)

a.prj/reports/

Component hardware implementation reports

a.prj/verification/

Simulation testbench

a.prj/quartus/

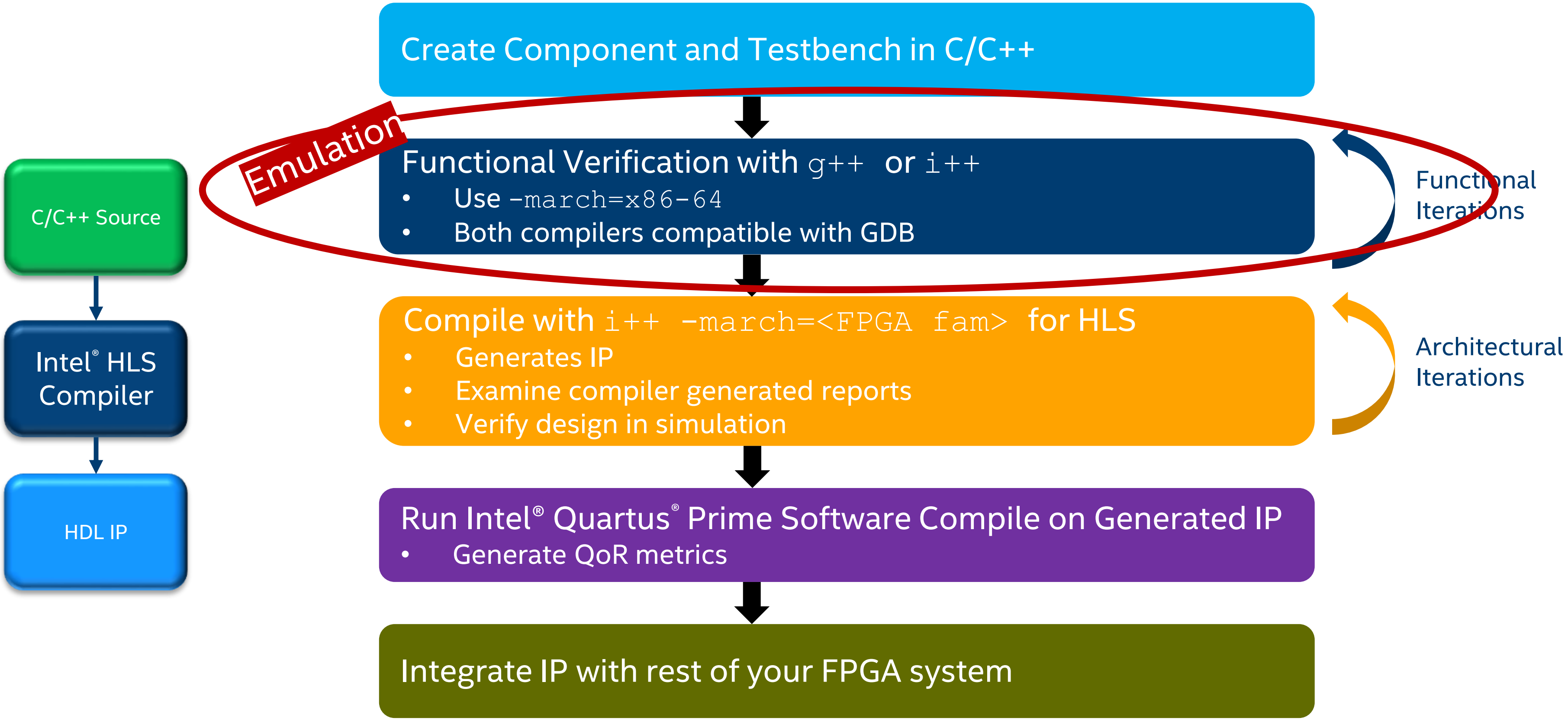
Quartus project to compile all IP

a is the default output name, -o option can be used to specify a non-default output name

# Introduction to HLS Agenda

- Introduction
- x86 Emulation
- Cosimulation
- Intel® Quartus® Software Integration

# HLS Procedure: x86 Emulation



C/C++ Source

Intel® HLS Compiler

HDL IP

Create Component and Testbench in C/C++

Emulation

Functional Verification with `g++` or `i++`

- Use `-march=x86-64`
- Both compilers compatible with GDB

Functional Iterations

Compile with `i++ -march=<FPGA fam>` for HLS

- Generates IP
- Examine compiler generated reports
- Verify design in simulation

Architectural Iterations

Run Intel® Quartus® Prime Software Compile on Generated IP

- Generate QoR metrics

Integrate IP with rest of your FPGA system

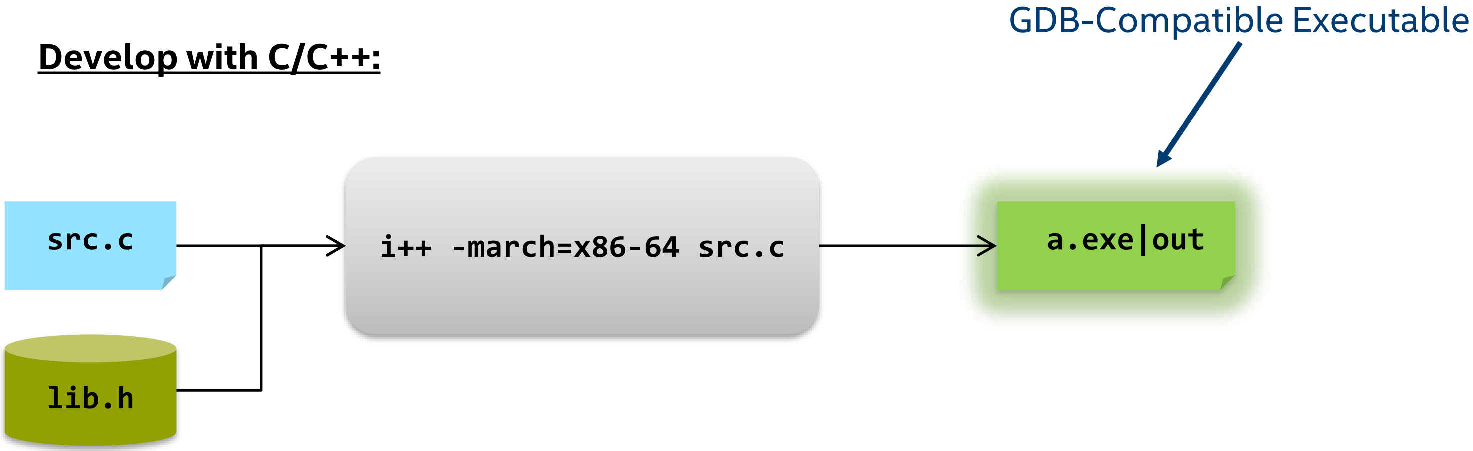
# g++ Compatibility

## Intel® HLS Compiler is command line compatible with g++

- Similar command-line flags, x86 behavior, and compilation flow
- Changing "`g++`" to "`i++`" should just work
  - `g++ <flags> <src>`
  - `i++ <flags> <src>`
- x86 behavior should match g++
- No source modifications required (for x86 mode)
- Support for GNU Makefiles

# x86 Debugging Tools

- printf/cout
- gdb
- Valgrind



# Using `printf()`

- Requires “`HLS/stdio.h`”
  - Maps to `<stdio.h>` when appropriate
- Can be included in the testbench or the component
  - Used with no limitations in the x86 emulation flow
- `printf` statements inside the **component** ignored for HDL generation
  - Ignored in the cosimulation flow with an HDL simulator

# Using printf(): Example

## Example Program

```cpp
// test.cpp
#include "HLS/stdio.h"

void say_hello() {
  printf("Hello from the component\n");
}

int main() {
  printf("Hello from the testbench\n");
  say_hello();
  return 0;
}
```

## Terminal Commands and output

```
$ i++ test.cpp
$ ./a.out
Hello from the testbench
Hello from the component
$
```

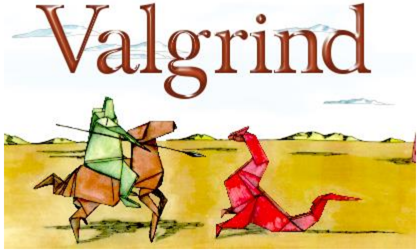
```
$ i++ test.cpp -march=Arria10 \
      --component say_hello
$ ./a.out
Hello from the testbench
$
```

# Debugging Using gdb

- i++ integrates well with GNU gdb
  - Debug data is generated by default
    - Unlike g++, -g enabled by default, use -g0 to turn off debug data
- `-march=x86-64` flow:
  - Can step through any part of the code (including the component)
- `-march=<fpga family>` flow:
  - Can step through testbench code
  - gdb does not see the component side execution (that runs in an HDL simulator)

# Debugging with Valgrind

The Valgrind tool suite provides a number of debugging and profiling tools that help you make your programs faster and more correct



http://www.valgrind.org/

Valgrind tools can detect:

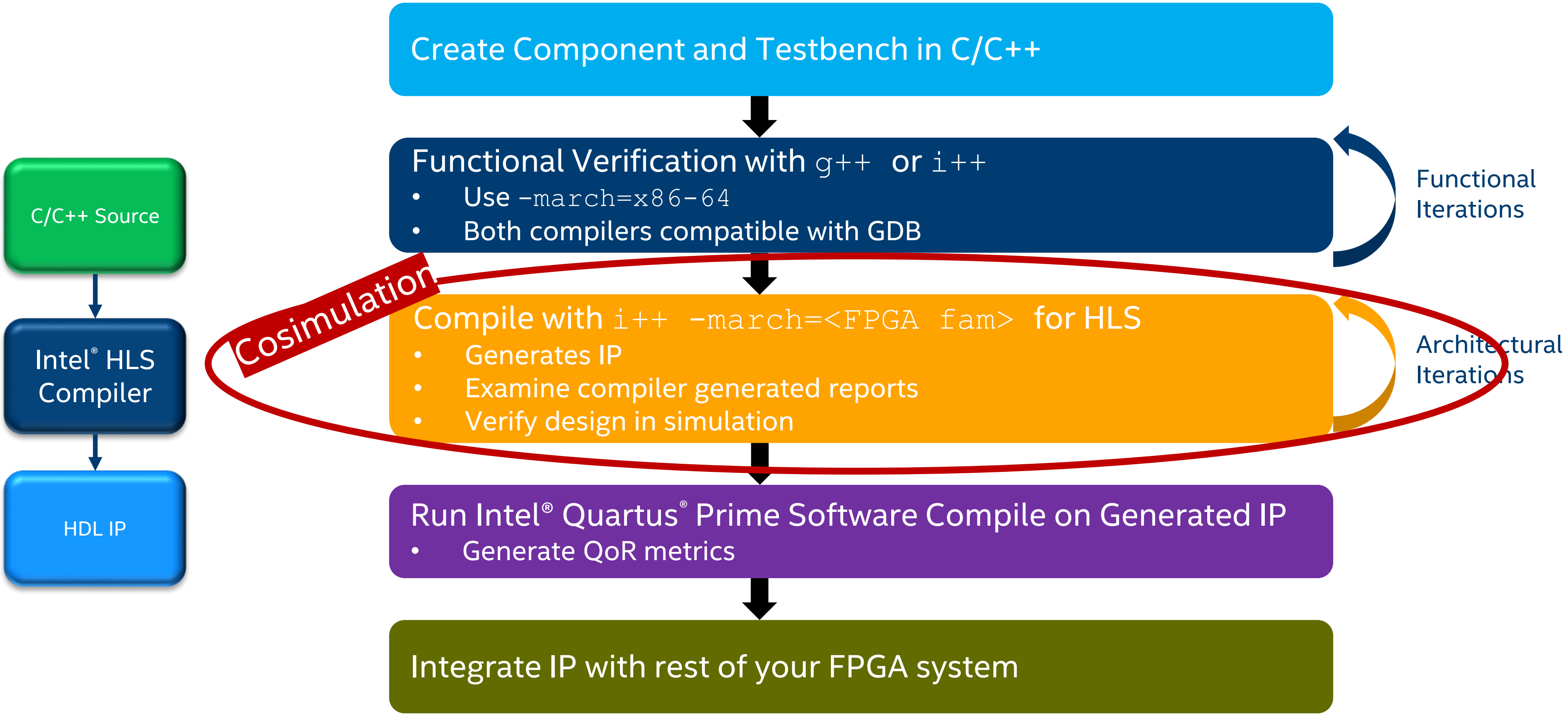
- Memory leaks
- Invalid pointer uses
- Use of uninitialized values
- Mismatched use of malloc/new vs free/delete
- Doubly freed memory

- Use to debug component and testbench in the x86 emulation flow

# Introduction to HLS Agenda

- Introduction
- x86 Emulation
- Cosimulation
- Intel® Quartus® Software Integration

# HLS Procedure: Cosimulation



C/C++ Source

Intel® HLS Compiler

HDL IP

Create Component and Testbench in C/C++

Functional Verification with `g++` or `i++`

- Use `-march=x86-64`
- Both compilers compatible with GDB

Functional Iterations

Cosimulation

Compile with `i++ -march=<FPGA fam>` for HLS

- Generates IP
- Examine compiler generated reports
- Verify design in simulation

Architectural Iterations

Run Intel® Quartus® Prime Software Compile on Generated IP

- Generate QoR metrics

Integrate IP with rest of your FPGA system

# Example Component/Testbench Source

```
i++ -march=<fpga family> --component accelerate mysource.cpp
```

```
#include "HLS/hls.h"
#include "assert.h"
#include "HLS/stdio.h"
#include "stdlib.h"

component int accelerate(int a, int b) {
      return a+b;
}

int main() {
      srand(0);
      for (int i=0; i<10; ++i) {
            int x=rand() % 10;
            int y=rand() % 10;
            int z=accelerate(x, y);
            printf("%d + %d = %d\n", x, y, z);
            assert(z == x + y);
      }
      return 0;
}
```

`accelerate()` becomes an FPGA component

- Use `--component` i++ argument or `component` attribute in source

`main()` becomes testbench for component `accelerate()`

# Translation from C function API to HDL module

- All component functions are synthesized to HDL
  - Each synthesized component is an independent HDL module
- Component functions can be declared:
  - Using `component` keyword in source
  - Specifying “`--component <component_name>`” in the command-line

# Cosimulation

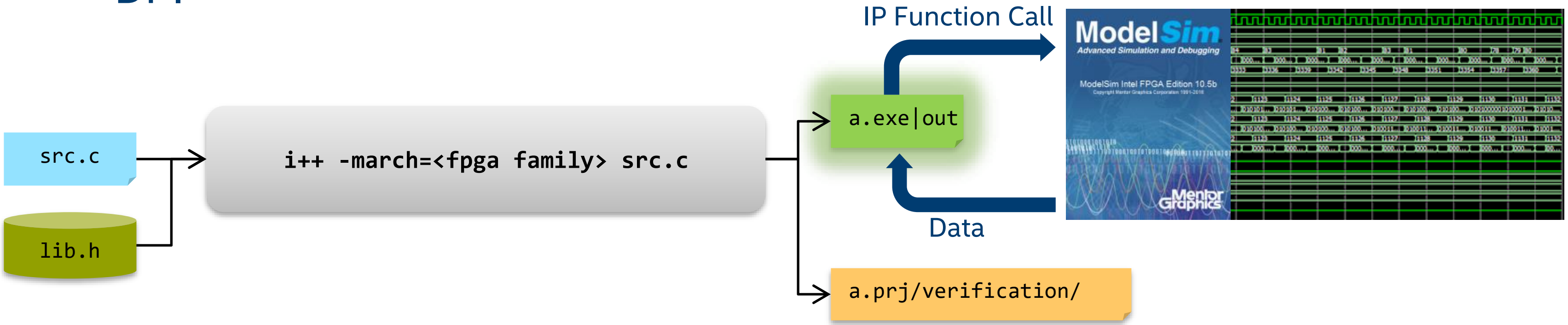
## Cosimulation: combines x86 testbench with RTL simulation

- HDL code for the component runs in an RTL Simulator
  - Verilog
  - RTL testbench automatically created from software
- `main()` and everything else called from `main` runs on x86 as the testbench
- Communication using SystemVerilog Direct Programming Interface (DPI)
  - Allows C/C++ to interface SystemVerilog
  - Inter-process communication (IPC) library used to pass testbench input data to RTL simulator, and returns the data back to the x86 testbench

# Cosimulation Verifying HLS IP

The Intel® HLS compiler automatically compiles and links C++ testbench with an instance of the component running in an RTL simulator

- To verify RTL behavior of IP, just run the executable generated by the HLS compiler targeting the FPGA architecture
  - Any calls to the component function becomes calls the simulator through DPI



src.c

lib.h

i++ -march=<fpga family> src.c

a.exe|out

IP Function Call

Data

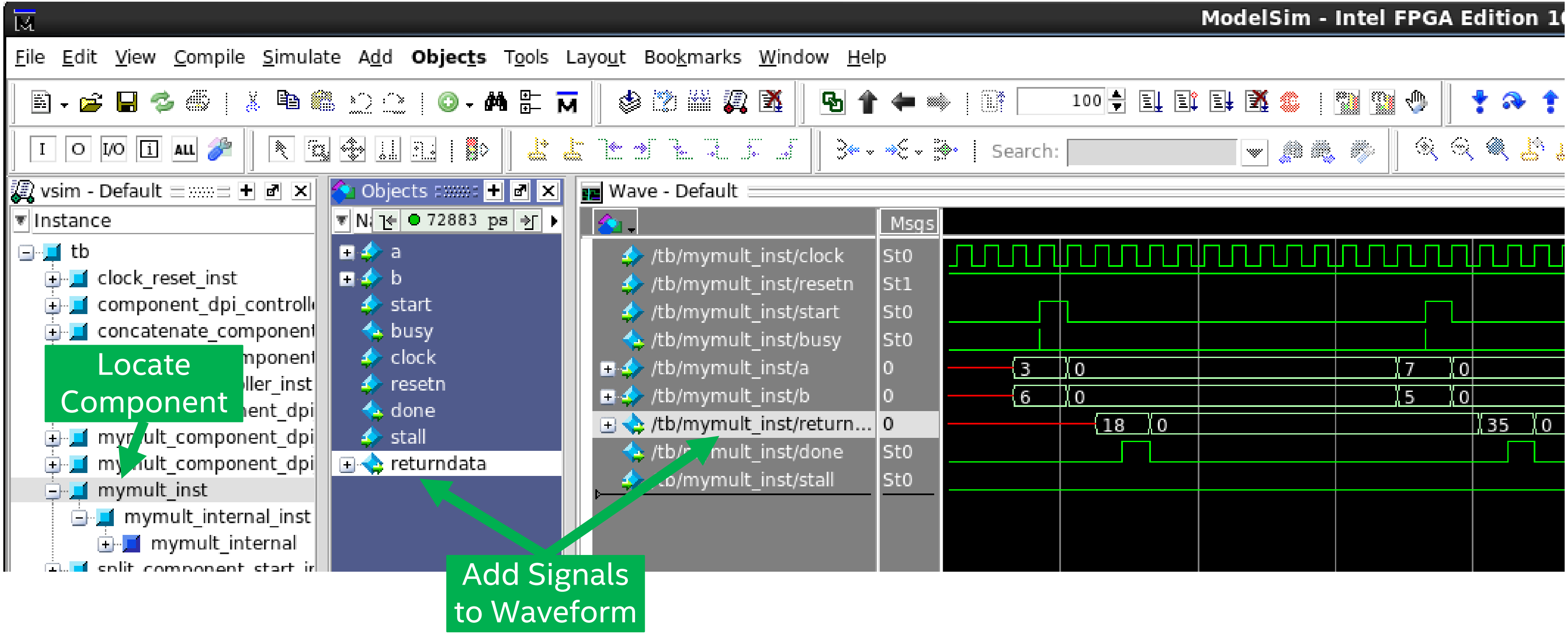
a.prj/verification/

ModelSim
Advanced Simulation and Debugging
ModelSim Intel FPGA Edition 10.5b
Mentor Graphics

# Viewing Component Waveforms

- Compile design with `i++ -ghdl` flag
  - Enable full visibility and logging of all HDL signals in simulation
- After cosimulation execution, waveform available at `a.prj/verification/vsim.wlf`
- Examine with the ModelSim* Simulator GUI:
  - `vsim a.prj/verification/vsim.wlf`

# Viewing Waveforms in the Modelsim* Simulator



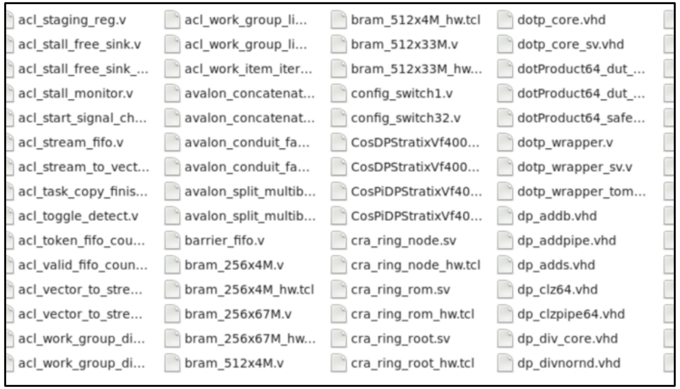
# Need for Cosimulation

- x86-emulation sufficient to functionally debug vast majority of issues
- Cosimulation used to test latency and performance of component
- Cosimulation used to catch hardware generation issues
  - Improper use of HLS compiler directives
    - e.g. `#pragmas`
  - Improper use of HLS compiler attributes
  - Improper use of HLS-specific constructs
  - Test component reset behavior
- Cosimulation should be done before integrating component with FPGA

# C/C++ Functions to Dataflow Circuits

## Each component function is converted into custom dataflow hardware

- Gain the benefits of Intel® FPGAs without the length design process
- Implement C/C++ operators as circuits
  - HDL code located in <HLS Installation>\ip
  - Load Store units to read/write memory
  - Arithmetic units to perform calculations
  - Flow control units
  - Connect circuits according to data flow in the function



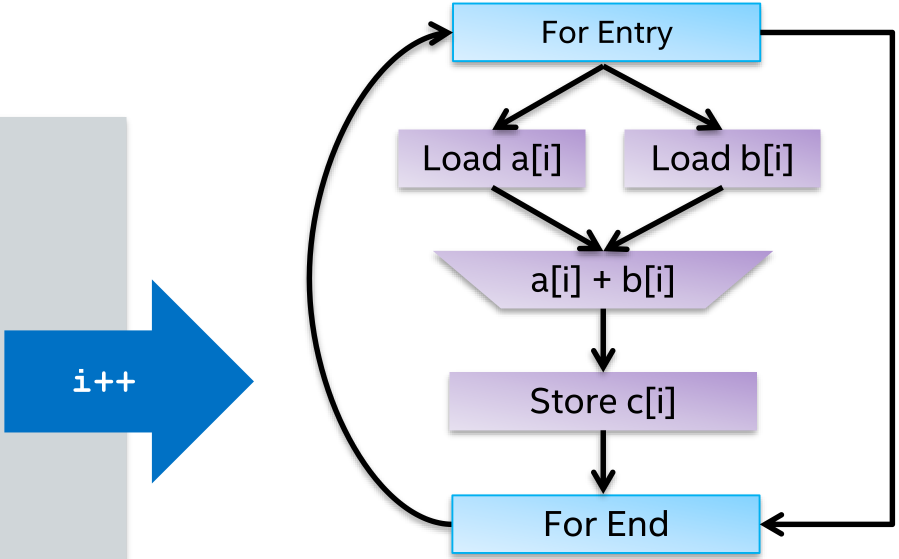
acl_staging_reg.v
acl_stall_free_sink.v
acl_stall_free_sink_...
acl_stall_monitor.v
acl_start_signal_ch...
acl_stream_fifo.v
acl_stream_to_vect...
acl_task_copy_finis...
acl_toggle_detect.v
acl_token_fifo_cou...
acl_valid_fifo_coun...
acl_vector_to_stre...
acl_vector_to_stre...
acl_work_group_di...
acl_work_group_di...
acl_work_group_li...
acl_work_group_li...
acl_work_item_iter...
avalon_concatenat...
avalon_concatenat...
avalon_conduit_fa...
avalon_conduit_fa...
avalon_split_multib...
avalon_split_multib...
barrier_fifo.v
bram_256x4M.v
bram_256x4M_hw.tcl
bram_256x67M.v
bram_256x67M_hw...
bram_512x4M.v
bram_512x4M_hw.tcl
bram_512x33M.v
bram_512x33M_hw...
config_switch1.v
config_switch32.v
CosDPStratixVf400...
CosDPStratixVf400...
CosPiDPStratixVf40...
CosPiDPStratixVf40...
cra_ring_node.sv
cra_ring_node_hw.tcl
cra_ring_rom.sv
cra_ring_rom_hw.tcl
cra_ring_root.sv
cra_ring_root_hw.tcl
dotp_core.vhd
dotp_core_sv.vhd
dotProduct64_dut_...
dotProduct64_dut_...
dotProduct64_safe...
dotp_wrapper.v
dotp_wrapper_sv.v
dotp_wrapper_tom...
dp_addb.vhd
dp_addpipe.vhd
dp_adds.vhd
dp_clz64.vhd
dp_clzpipe64.vhd
dp_div_core.vhd
dp_divnornd.vhd

# Compilation Example

## Software compiled into dataflow circuit with flow control

- Include branch and merge units

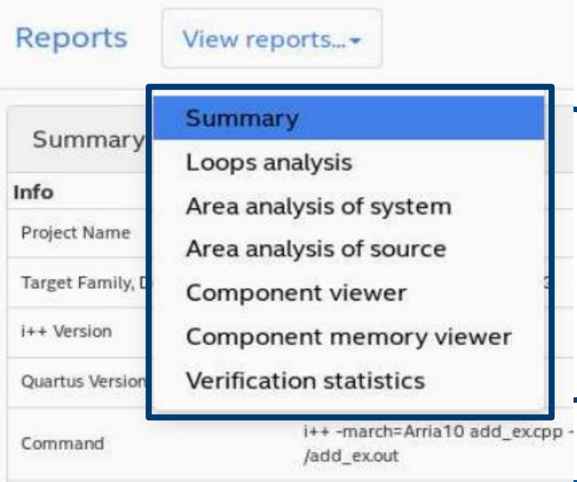
```
void my_component(    int *a,
                      int *b,
                      int *c,
                      int N)
{
    int i;
    for (i = 0; i < N; i++)
        c[i] = a[i] + b[i];
}
```



# Main HTML Report

The Intel® HLS Compiler automatically generates HTML report that analyzes various aspects of your function including area, loop structure, memory usage, and system data flow

- Located at `a.prj/reports/report.html`



Summary

Info

Project Name

Target Family, D

i++ Version

Quartus Version

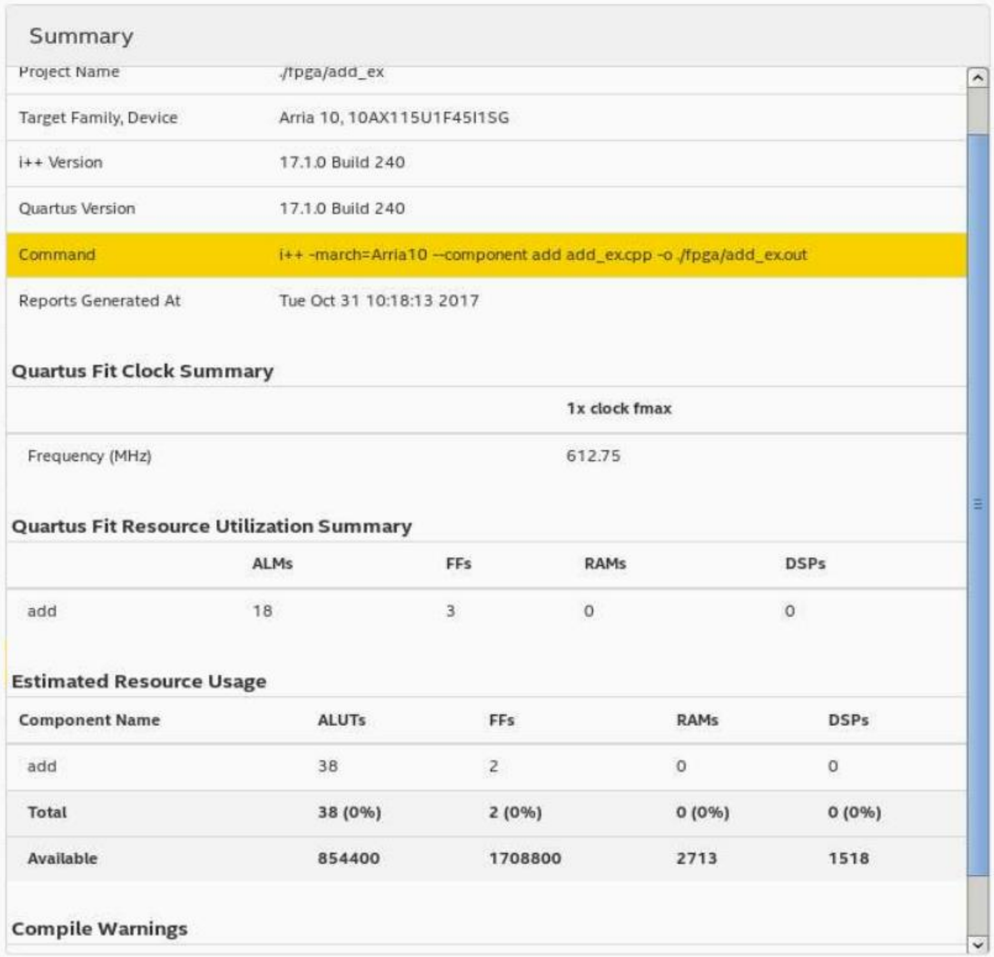
Command i++ -march=Arria10 add_ex.cpp - /add_ex.out

Many Types of Reports

# HTML Report: Summary

## Overall compile statics

- FPGA Resource Utilization
- Compile Warnings
- Intel® Quartus® Software fitter results
  - Available after compilation
- etc.



Summary

| | |
|---|---|
| Project Name | ./fpga/add_ex |
| Target Family, Device | Arria 10, 10AX115U1F45I1SG |
| i++ Version | 17.1.0 Build 240 |
| Quartus Version | 17.1.0 Build 240 |
| Command | i++ -march=Arria10 --component add add_ex.cpp -o ./fpga/add_ex.out |
| Reports Generated At | Tue Oct 31 10:18:13 2017 |

**Quartus Fit Clock Summary**

| | 1x clock fmax |
|---|---|
| Frequency (MHz) | 612.75 |

**Quartus Fit Resource Utilization Summary**

| | ALMs | FFs | RAMs | DSPs |
|---|---|---|---|---|
| add | 18 | 3 | 0 | 0 |

**Estimated Resource Usage**

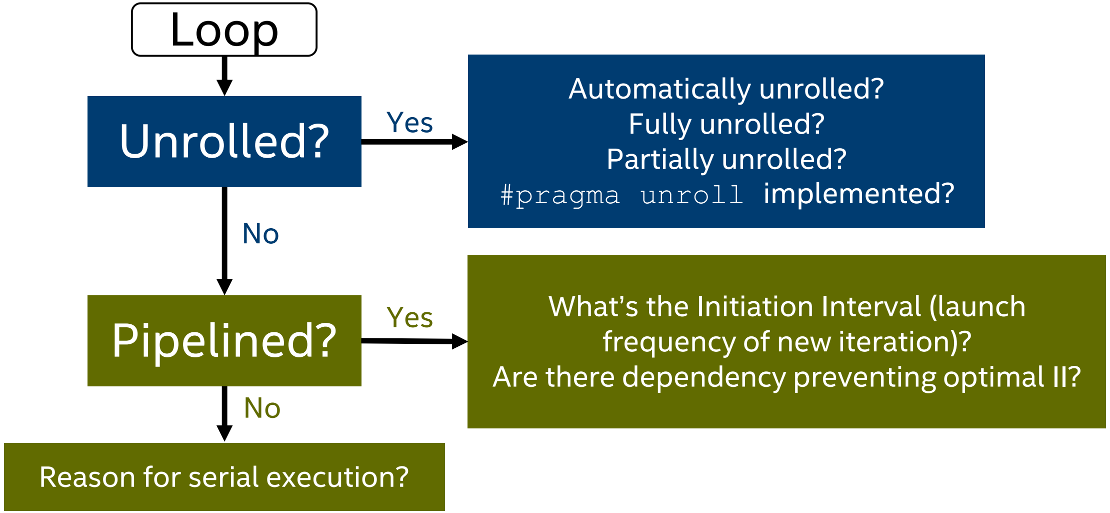
| Component Name | ALUTs | FFs | RAMs | DSPs |
|---|---|---|---|---|
| add | 38 | 2 | 0 | 0 |
| **Total** | **38 (0%)** | **2 (0%)** | **0 (0%)** | **0 (0%)** |
| **Available** | **854400** | **1708800** | **2713** | **1518** |

**Compile Warnings**

# HTML Report: Loops

## Serial loop execution hinders function dataflow circuit performance

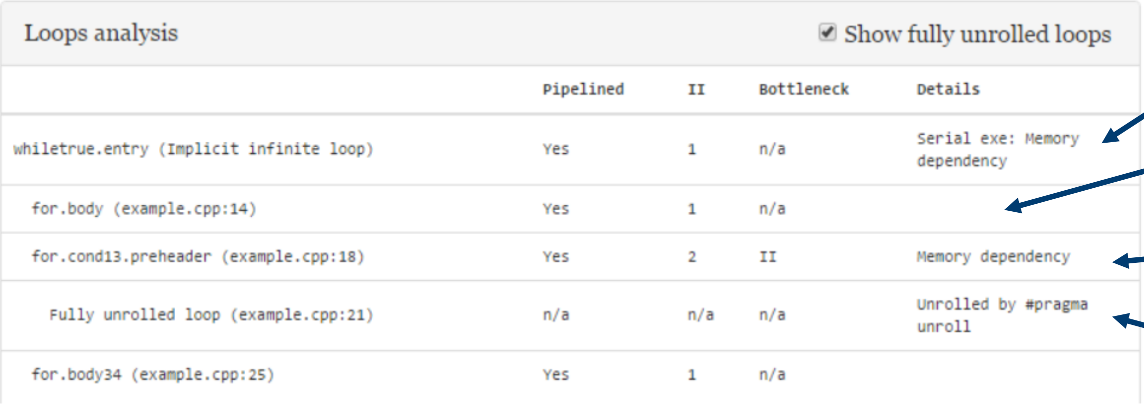
- Use Loop Analysis report to see if and how each loop is optimized
  - Helps identify component pipeline bottlenecks


Loop
Unrolled?
Yes
Automatically unrolled?
Fully unrolled?
Partially unrolled?
`#pragma unroll` implemented?
No
Pipelined?
Yes
What's the Initiation Interval (launch frequency of new iteration)?
Are there dependency preventing optimal II?
No
Reason for serial execution?

# HTML Report: Loop Analysis

Loop analysis shows how loops are implemented

- Ability to correlate with source code



Loops analysis

☑ Show fully unrolled loops

| | Pipelined | II | Bottleneck | Details |
|---|---|---|---|---|
| whiletrue.entry (Implicit infinite loop) | Yes | 1 | n/a | Serial exe: Memory dependency |
| for.body (example.cpp:14) | Yes | 1 | n/a | |
| for.cond13.preheader (example.cpp:18) | Yes | 2 | II | Memory dependency |
| Fully unrolled loop (example.cpp:21) | n/a | n/a | n/a | Unrolled by #pragma unroll |
| for.body34 (example.cpp:25) | Yes | 1 | n/a | |

Compiler-added loop, not in the code, implicit infinitely loop allowing the component to run continuously in pipelined fashion

Pipelined loop, II=1

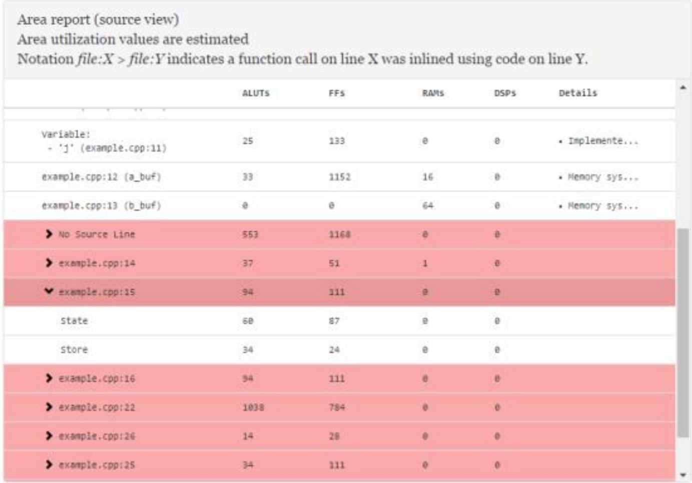
Pipelined loop, II=2 due to memory dependency

Fully unrolled loop, due to user `#pragma unroll`

# HTML Report: Area Analysis

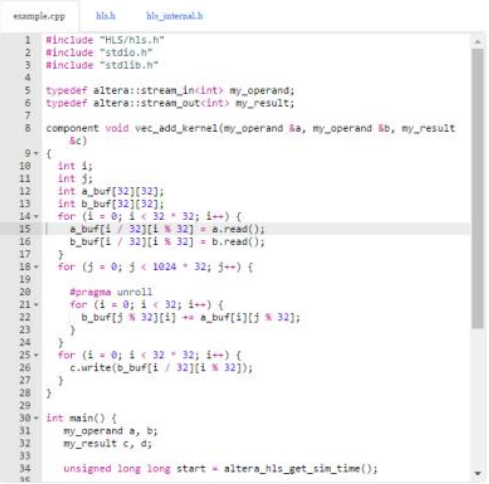
## View detailed estimated resource consumption by system or source line

- Analyze data control overhead
- View memory implementation
- Shows resource usage
  - ALUTs
  - FFs
  - RAMs
  - DSPs
- Identifies inefficient uses



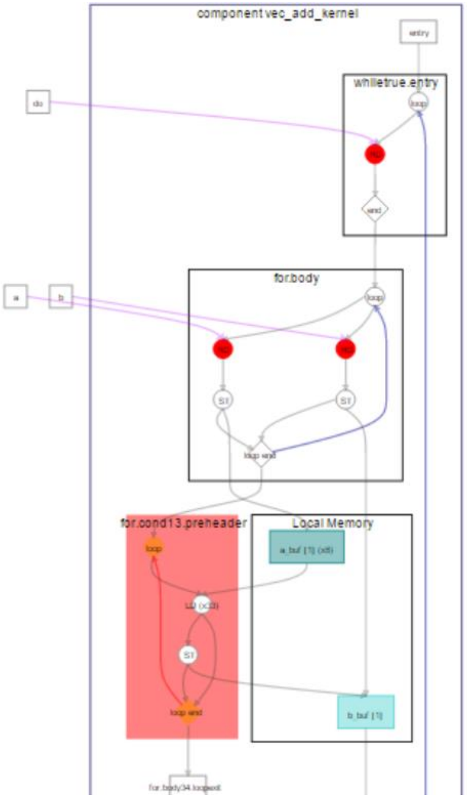
Area report (source view)
Area utilization values are estimated
Notation *file:X* > *file:Y* indicates a function call on line X was inlined using code on line Y.

| | ALUTs | FFs | RAMs | DSPs | Details |
|---|---|---|---|---|---|
| Variable:<br>- 'j' (example.cpp:11) | 25 | 133 | 0 | 0 | • Implemente... |
| example.cpp:12 (a_buf) | 33 | 1152 | 16 | 0 | • Memory sys... |
| example.cpp:13 (b_buf) | 0 | 0 | 64 | 0 | • Memory sys... |
| › No Source Line | 553 | 1168 | 0 | 0 | |
| › example.cpp:14 | 37 | 51 | 1 | 0 | |
| ⌄ example.cpp:15 | 94 | 111 | 0 | 0 | |
| State | 60 | 87 | 0 | 0 | |
| Store | 34 | 24 | 0 | 0 | |
| › example.cpp:16 | 94 | 111 | 0 | 0 | |
| › example.cpp:22 | 1038 | 784 | 0 | 0 | |
| › example.cpp:26 | 14 | 28 | 0 | 0 | |
| › example.cpp:25 | 34 | 111 | 0 | 0 | |



```
1   #include "HLS/hls.h"
2   #include "stdio.h"
3   #include "stdlib.h"
4
5   typedef altera::stream_in<int> my_operand;
6   typedef altera::stream_out<int> my_result;
7
8   component void vec_add_kernel(my_operand &a, my_operand &b, my_result
      &c)
9   {
10    int i;
11    int j;
12    int a_buf[32][32];
13    int b_buf[32][32];
14    for (i = 0; i < 32 * 32; i++) {
15      a_buf[i / 32][i % 32] = a.read();
16      b_buf[i / 32][i % 32] = b.read();
17    }
18    for (j = 0; j < 1024 * 32; j++) {
19
20      #pragma unroll
21      for (i = 0; i < 32; i++) {
22        b_buf[j % 32][i] += a_buf[i][j % 32];
23      }
24    }
25    for (i = 0; i < 32 * 32; i++) {
26      c.write(b_buf[i / 32][i % 32]);
27    }
28  }
29
30  int main() {
31     my_operand a, b;
32     my_result c, d;
33
34     unsigned long long start = altera_hls_get_sim_time();
```

# HTML Report: Component Viewer



## Displays abstracted netlist of the HW implementation

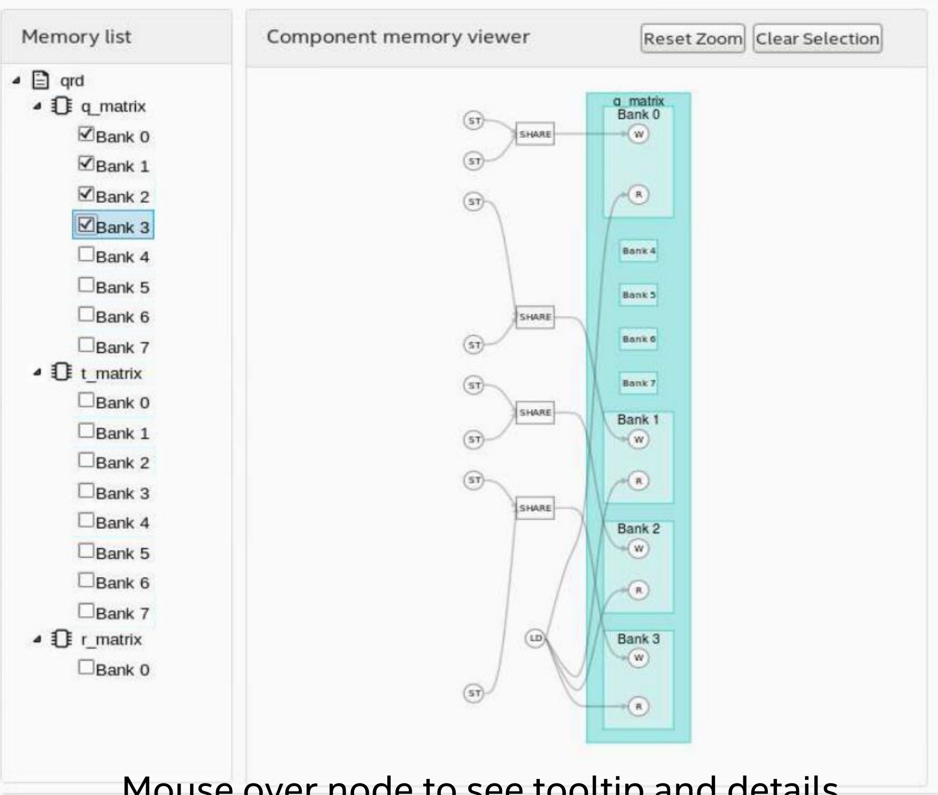
- View data flow pipeline
  - See loads and stores
  - Interfaces including stream reads and writes
  - Memory structure
  - Loop structure
  - Possible performance bottlenecks
    - Unpipelined loops are colored light red
    - Stallable points are red

Mouse over node to see tooltip and details. Correlates with source code.

# HTML Report: Memory Viewer

## Displays local memory implementation and accesses

- Visualize memory architecture
  - Banks, widths, replication, etc
- Visualize load-store units (LSUs)
  - Stall-free?
  - Arbitration
  - Red indicates stalled



Memory list

- qrd
  - q_matrix
    - ☑ Bank 0
    - ☑ Bank 1
    - ☑ Bank 2
    - ☑ Bank 3
    - □ Bank 4
    - □ Bank 5
    - □ Bank 6
    - □ Bank 7
  - t_matrix
    - □ Bank 0
    - □ Bank 1
    - □ Bank 2
    - □ Bank 3
    - □ Bank 4
    - □ Bank 5
    - □ Bank 6
    - □ Bank 7
  - r_matrix
    - □ Bank 0

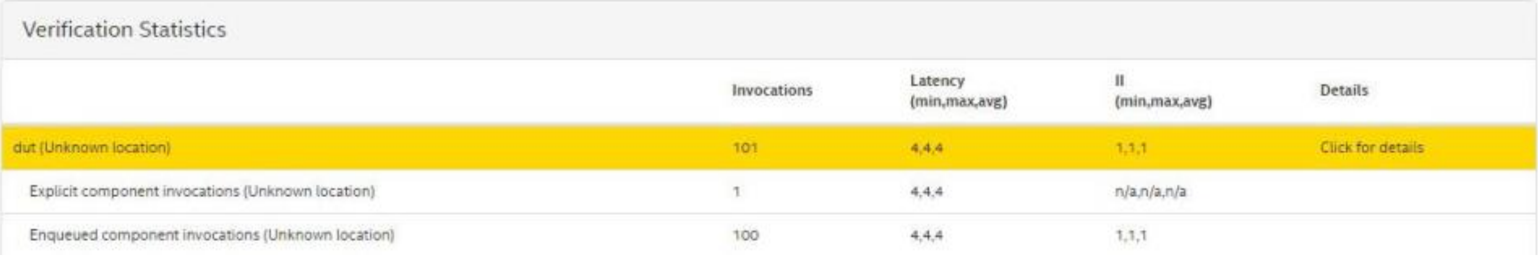
Mouse over node to see tooltip and details. Correlates with source code.

# HTML Report: Verification Statistics

Reports execution statics from testbench execution, available after component is simulated (testbench executable ran)

- Number and type of component invocation
- Latency of component
- Dynamic Initiation interval of Component
- Data rates of streams

Measurements based on latest execution of testbench



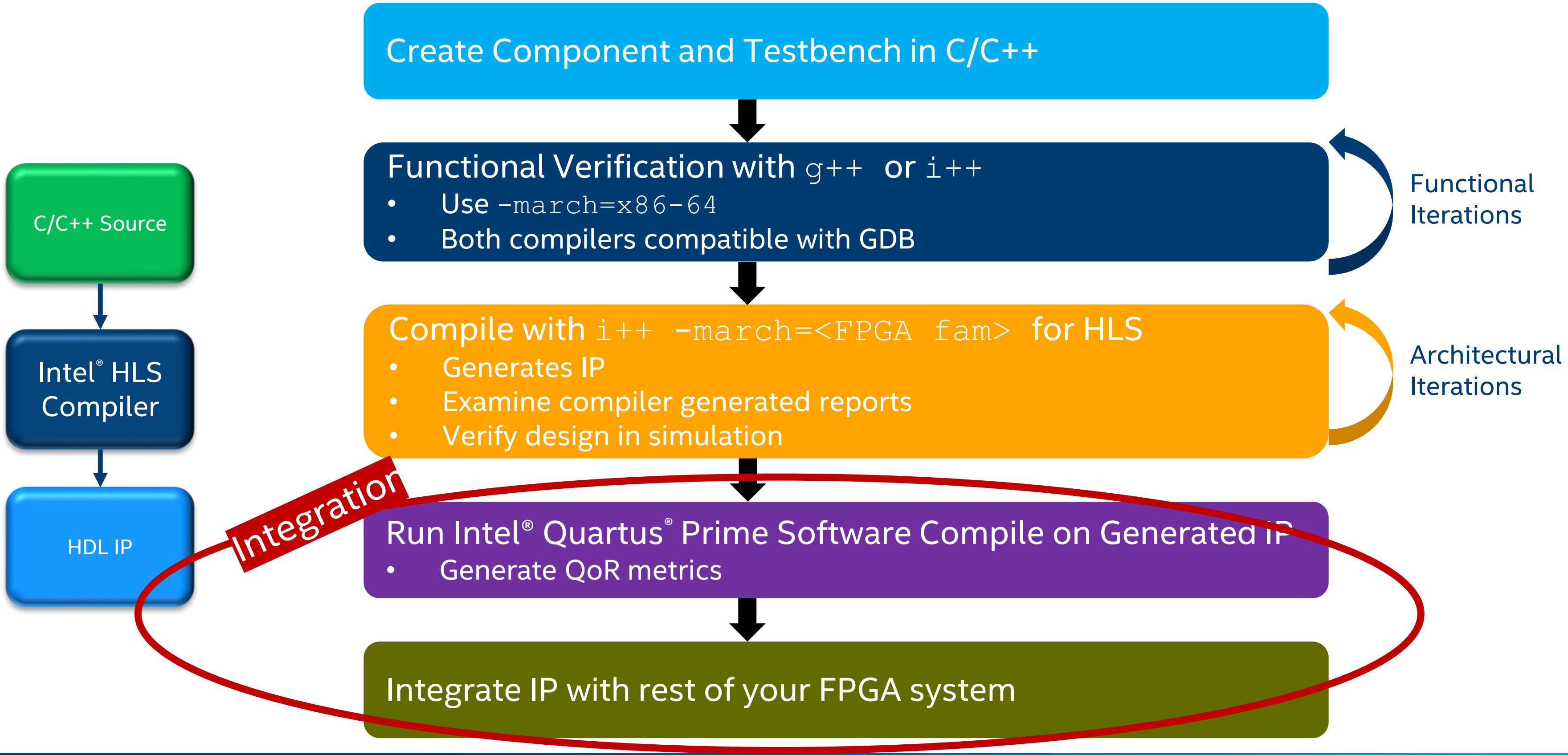
Verification Statistics

| | Invocations | Latency<br>(min,max,avg) | II<br>(min,max,avg) | Details |
|---|---|---|---|---|
| dut (Unknown location) | 101 | 4,4,4 | 1,1,1 | Click for details |
| Explicit component invocations (Unknown location) | 1 | 4,4,4 | n/a,n/a,n/a | |
| Enqueued component invocations (Unknown location) | 100 | 4,4,4 | 1,1,1 | |

# Introduction to HLS Agenda

- Introduction
- x86 emulation
- Cosimulation
- Intel® Quartus® Software Integration

# HLS Procedure: Integration



Create Component and Testbench in C/C++

Functional Verification with `g++` or `i++`

- Use `-march=x86-64`
- Both compilers compatible with GDB

Compile with `i++ -march=<FPGA fam>` for HLS

- Generates IP
- Examine compiler generated reports
- Verify design in simulation

Run Intel® Quartus® Prime Software Compile on Generated IP

- Generate QoR metrics

Integrate IP with rest of your FPGA system

# Intel Quartus® Software QoR Metrics for IP

## Use Intel® Quartus® Prime software to generate quality-of-result reports

- i++ creates the Quartus project in `a.prj/quartus`
- To generate QoR data (final resource utilization, fmax)
  - Run `quartus_sh --flow compile quartus_compile`
  - Or use `i++ --quartus-compile` option
- Report part of the HTML report
  - `a.prj/reports/report.html`
  - Summary page

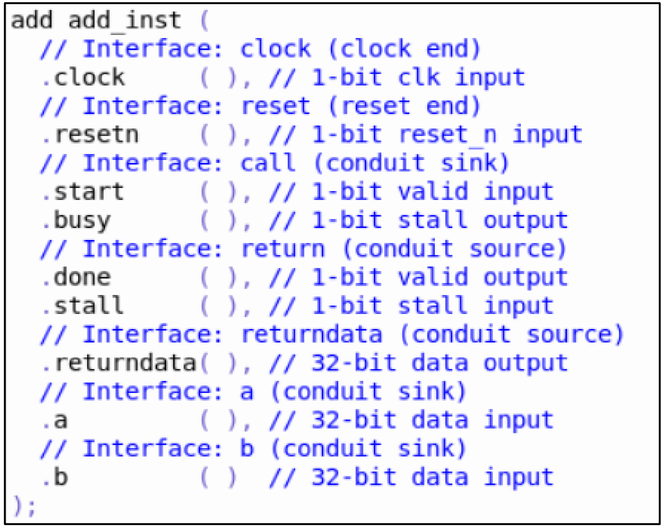
# Intel® Quartus® Software Integration

- `a.prj/components` directory contains all the files to integrate
  - One subdirectory for each component
    - Portable, can be moved to a different location if desire
- 2 use scenarios
  1. Instantiate in HDL
  2. Adding IP to a Platform Designer system

# HDL Instantiation

- Add Components to Intel® Quartus® Software Project
  - *<component>*`.qsys` to Standard Edition
  - *<component>*`.ip` to Pro Edition
- Instantiate component module in your design
  - Use template

```
a.prj/components/<component>/<component>_inst.v
```



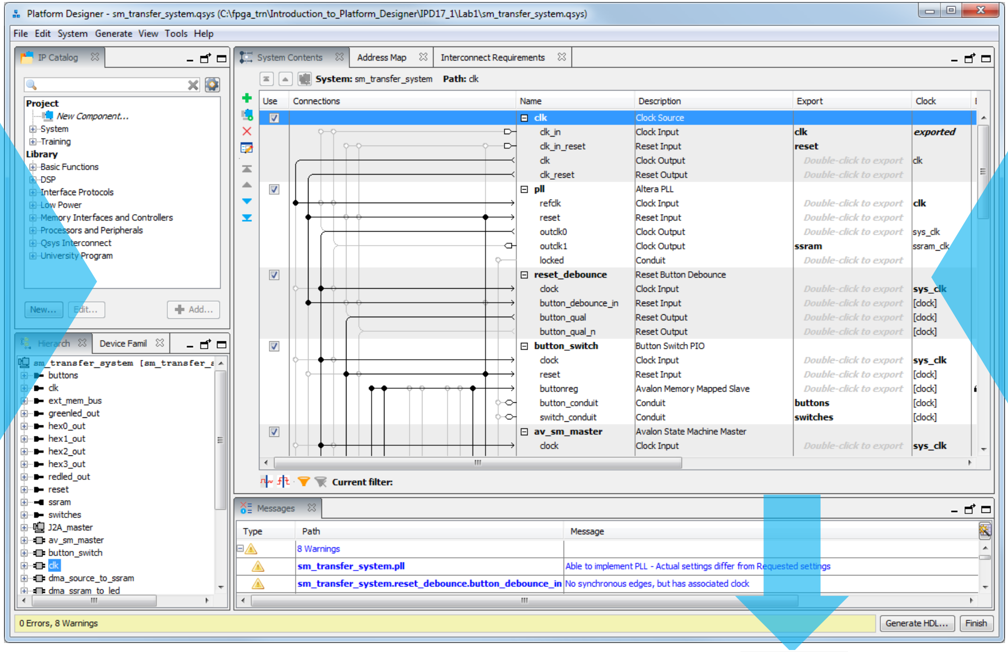
```
add add_inst (
  // Interface: clock (clock end)
  .clock     ( ), // 1-bit clk input
  // Interface: reset (reset end)
  .resetn    ( ), // 1-bit reset_n input
  // Interface: call (conduit sink)
  .start     ( ), // 1-bit valid input
  .busy      ( ), // 1-bit stall output
  // Interface: return (conduit source)
  .done      ( ), // 1-bit valid output
  .stall     ( ), // 1-bit stall input
  // Interface: returndata (conduit source)
  .returndata( ), // 32-bit data output
  // Interface: a (conduit sink)
  .a         ( ), // 32-bit data input
  // Interface: b (conduit sink)
  .b         ( )  // 32-bit data input
);
```

# Platform Designer System Integration Tool

## Catalog of available IP

- Interface protocols
- Memory
- DSP
- Embedded
- Bridges
- PLL
- **Custom Components**
- **Custom Systems**

Accelerate development



Platform Designer - sm_transfer_system.qsys (C:\fpga_trn\Introduction_to_Platform_Designer\IPD17_1\Lab1\sm_transfer_system.qsys)

File Edit System Generate View Tools Help

IP Catalog

**Project**
- *New Component...*
- System
- Training

**Library**
- Basic Functions
- DSP
- Interface Protocols
- Low Power
- Memory Interfaces and Controllers
- Processors and Peripherals
- Qsys Interconnect
- University Program

New... Edit... Add...

Hierarch | Device Famil

sm_transfer_system [sm_transfer_s
- buttons
- clk
- ext_mem_bus
- greenled_out
- hex0_out
- hex1_out
- hex2_out
- hex3_out
- redled_out
- reset
- ssram
- switches
- J2A_master
- av_sm_master
- button_switch
- clk
- dma_source_to_ssram
- dma_ssram_to_led

System Contents | Address Map | Interconnect Requirements

**System:** sm_transfer_system **Path:** clk

| Use | Connections | Name | Description | Export | Clock |
|---|---|---|---|---|---|
| ☑ | | **clk** | Clock Source | | |
| | | clk_in | Clock Input | **clk** | ***exported*** |
| | | clk_in_reset | Reset Input | **reset** | |
| | | clk | Clock Output | *Double-click to export* | clk |
| | | clk_reset | Reset Output | *Double-click to export* | |
| ☑ | | **pll** | Altera PLL | | |
| | | refclk | Clock Input | *Double-click to export* | **clk** |
| | | reset | Reset Input | *Double-click to export* | |
| | | outclk0 | Clock Output | *Double-click to export* | sys_clk |
| | | outclk1 | Clock Output | **ssram** | ssram_clk |
| | | locked | Conduit | *Double-click to export* | |
| ☑ | | **reset_debounce** | Reset Button Debounce | | |
| | | clock | Clock Input | *Double-click to export* | **sys_clk** |
| | | button_debounce_in | Reset Input | *Double-click to export* | [clock] |
| | | button_qual | Reset Output | *Double-click to export* | [clock] |
| | | button_qual_n | Reset Output | *Double-click to export* | [clock] |
| ☑ | | **button_switch** | Button Switch PIO | | |
| | | clock | Clock Input | *Double-click to export* | **sys_clk** |
| | | reset | Reset Input | *Double-click to export* | [clock] |
| | | buttonreg | Avalon Memory Mapped Slave | *Double-click to export* | [clock] |
| | | button_conduit | Conduit | **buttons** | [clock] |
| | | switch_conduit | Conduit | **switches** | [clock] |
| ☑ | | **av_sm_master** | Avalon State Machine Master | | |
| | | clock | Clock Input | *Double-click to export* | **sys_clk** |

**Current filter:**

Messages

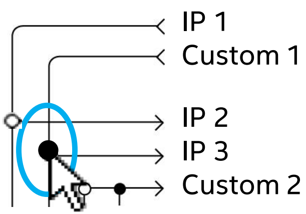
| Type | Path | Message |
|---|---|---|
| ⚠ | 8 Warnings | |
| ⚠ | **sm_transfer_system.pll** | Able to implement PLL - Actual settings differ from Requested settings |
| ⚠ | **sm_transfer_system.reset_debounce.button_debounce_in** | No synchronous edges, but has associated clock |

0 Errors, 8 Warnings

Generate HDL... Finish

*Automate integration tasks*

## Connect custom IP and systems



*Simplify integration*

# HLS Interfaces

How to integrate your component with the rest of the system

# HLS Interfaces Section - Agenda

- Avalon® Interfaces
- Default HLS Interfaces
- Memory Master Interfaces
- Explicit Streaming Interfaces
- Register Interfaces
- Memory Slave Interfaces

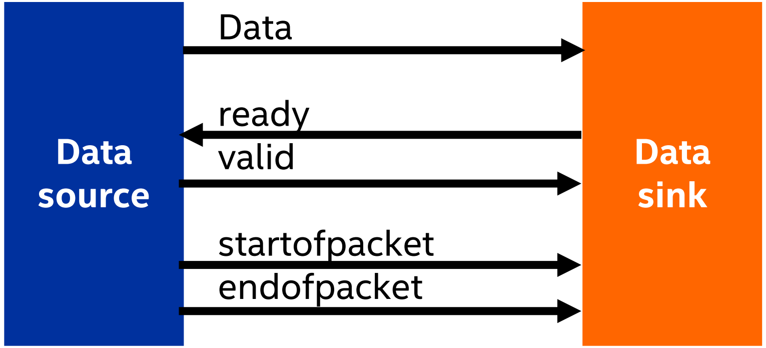
# Avalon® Interfaces

Easily connects components in an Intel® FPGA to simplify system design

- Standard interfaces design for interoperability
- HLS compiler generates Avalon® interfaces around HLS components
- Avalon Streaming Interface (Avalon-ST)
  - Unidirectional flow of data, simple flexible interface
- Avalon Memory Mapped Interface (Avalon-MM)
  - Address-based read/write interface typical of master-slave connections
- Other Interfaces
  - Conduit, Tri-State Conduit, Interrupt, Clock, Reset

# Avalon®-ST Interfaces

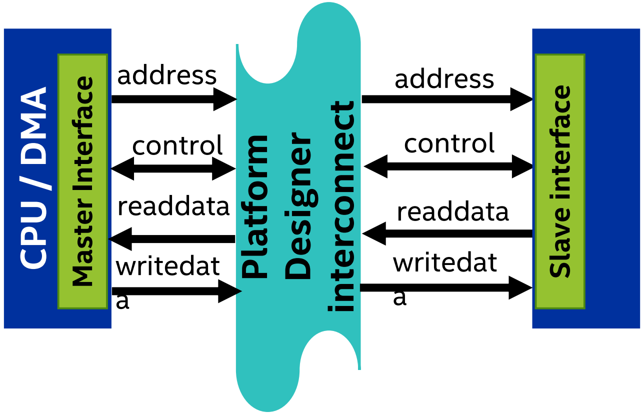
- Standard, flexible, and modular protocol for transfer of data
  - Unidirectional
  - Point-to-point connections
  - Fully synchronous
  - Supports simple and complex interface requirements



# Avalon®-MM Interfaces

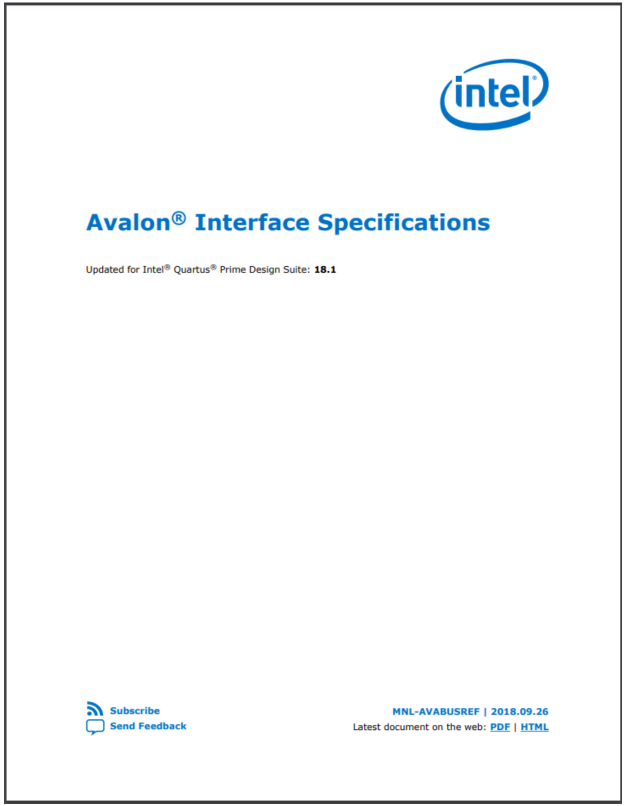
- Address-based (memory-mapped) protocol that allows components to communicate using read/write requests
- Master interface
  - Initiates read/write transfers targeting specific address
- Slave interface
  - Accepts and responds to transfer requests
- Interconnect handles decoding of master address request to actual slave interface, backpressure, clocking differences, etc.

Example master/slave connections



# Avalon® Interface Specification

- Defines the entire Avalon interface standard, including all variations
- Provides reference information on additional transfer types
  - Use cases
  - Waveform diagrams
- https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/manual/mnl_avalon_spec.pdf



Avalon® Interface Specifications

Updated for Intel® Quartus® Prime Design Suite: **18.1**

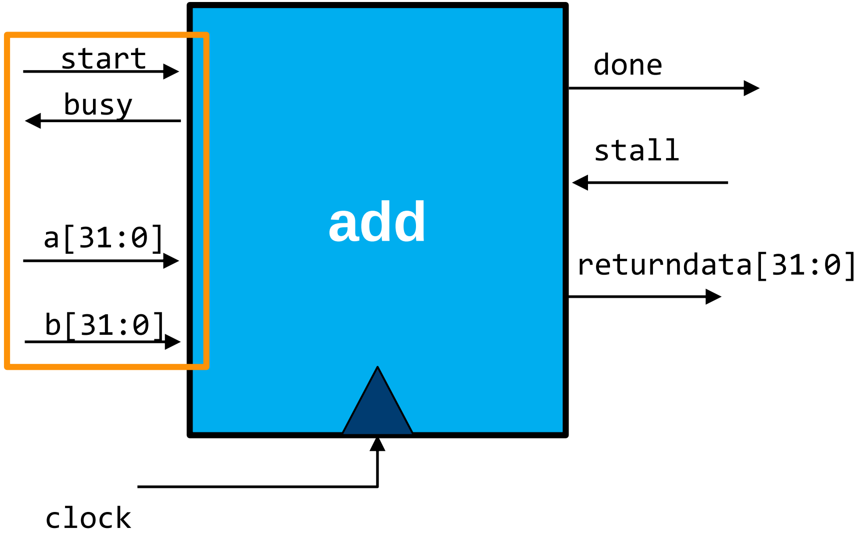
Subscribe
Send Feedback

MNL-AVABUSREF | 2018.09.26
Latest document on the web: PDF | HTML

# Default Interfaces for Scalars

- Scalar arguments results in an input conduit associated with `start` and `busy` signals

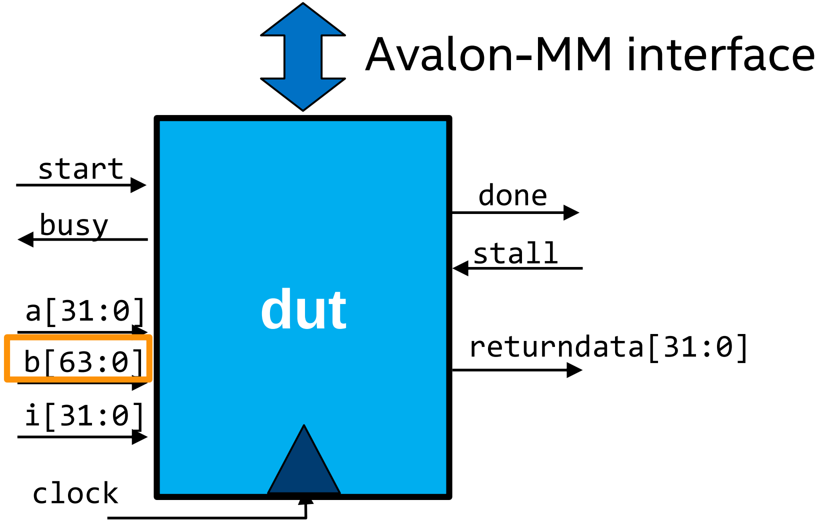
```
component int add(int a, int b ){
      return a+b;
}
```



# Pointers: Implicit Memory-Mapped Interface

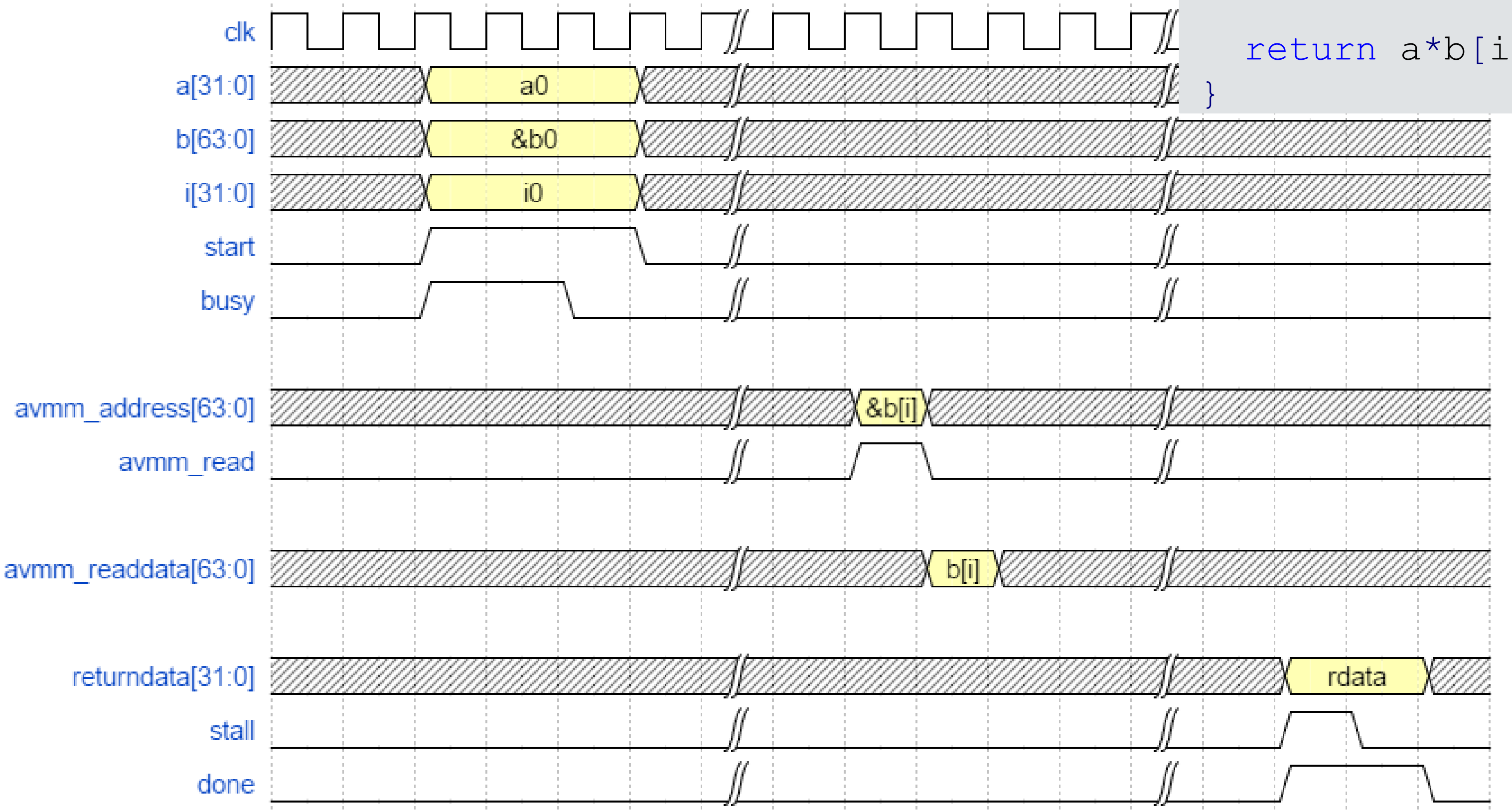
- All pointer or reference arguments becomes address input associated with `start` and `busy` signals
- Memory-mapped master interface automatically created
- Default 64bit address space

```
component int dut(int  a,
                  int *b,
                  int  i) {
  return a*b[i]; }
```



# Pointers – Waveform

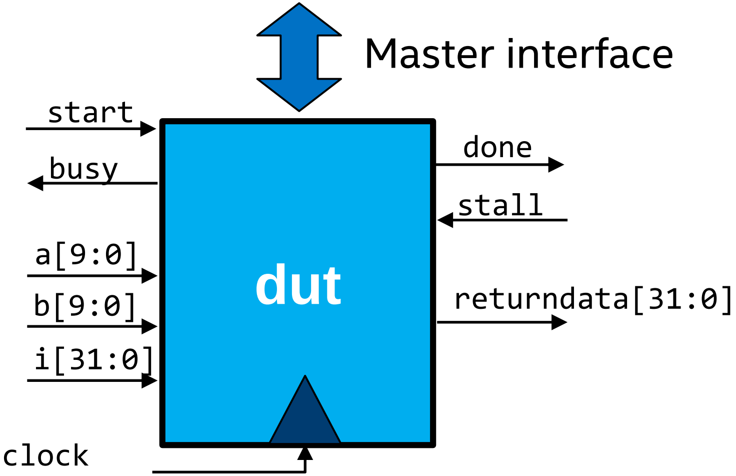
```
component int dut(int  a,
                  int *b,
                  int  i) {
  return a*b[i];
}
```



# Explicit MM Master Interface

```
component int dut(ihc::mm_master<int, ihc::aspace<2>, ihc::latency<0>,
                                    ihc::awidth<10>, ihc::dwidth<32> > &a,
                  ihc::mm_master<int, ihc::aspace<2>, ihc::latency<0>,
                                    ihc::awidth<10>, ihc::dwidth<32> > &b,
                  int i) {
    return a[i]*b[i];
}
```

- Explicitly declare Avalon-MM Master interfaces using `mm_master<>` class
  - Greater control over interface
  - Specify attributes through parameters



# `ihc::mm_master` Class Parameters

Usage: `ihc::mm_master<datatype, /*template arguments*/>`

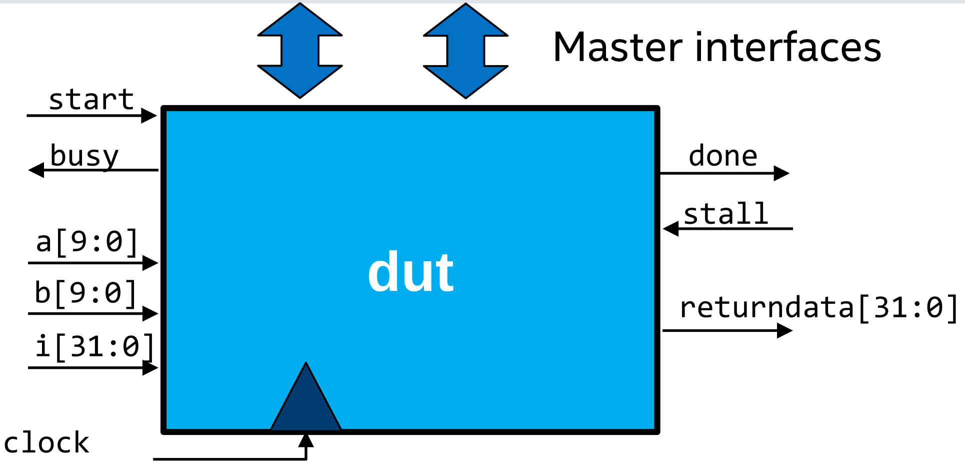
| Feature | Valid Values | Default | Description |
|---|---|---|---|
| ihc::dwidth | 8,16,32,…1024 | 64 | Width of data bus |
| ihc::awidth | 1-64 | 64 | Width of address bus (byte addressing) |
| ihc::aspace | >0 | 1 | Address space #, masters with the same address space are arbitrated |
| ihc::align | >default | type | Byte alignment of pointer address |
| ihc::latency | >=0 | 1 | Guaranteed latency from read to valid data, 0=variable latency |
| ihc::maxburst | 1-1024 | 1 | Max transfers associated with a read/write. For fixed latency interfaces, value must be 1 |

Other attributes including `readwrite_mode, and waitrequest` described in the HLS Compiler Reference Manual

# MM Master Address Spaces

```
component int dut(ihc::mm_master<int, ihc::aspace<1>, ihc::latency<3>,
                                    ihc::awidth<10>, ihc::dwidth<32> > &a,
                  ihc::mm_master<int, ihc::aspace<2>, ihc::latency<3>,
                                    ihc::awidth<10>, ihc::dwidth<32> > &b,
                  int i) {
    return a*b[i];
}
```

- Having multiple address spaces creates multiple MM Masters
  - Allows simultaneous multi-mastering over Platform Designer interconnect

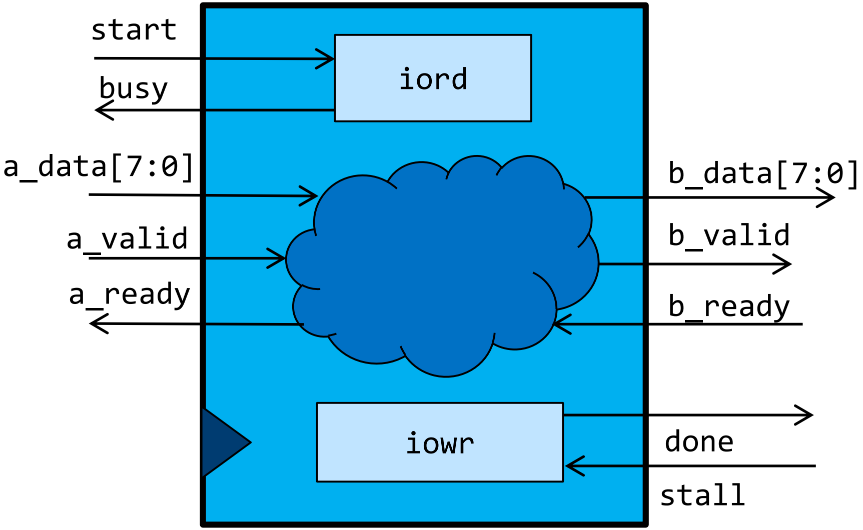


# Streaming Interfaces

- Scalar function arguments become pipelined input ports on the HDL module
  - Avalon Streaming interface associated with `start` and `busy` inputs
  - Implicit
- Explicit Streaming Interfaces
  - Use `ihc::stream_in<>` and `ihc::stream_out<>` template classes
    - Pass by reference
  - Creates Avalon Streaming interface with valid and ready signals
  - Explicit control over interface

# Explicit Streaming Interface Example

```
component
void dut(ihc::stream_in<unsigned char> &a,
         ihc::stream_out <unsigned char> &b)
{
    for (int i = 0; i < N; i++) {
        unsigned char input = a.read();
        input = 255 - input;
        b.write(input);
    }
}
```



# Explicit Streaming Interface Customizations

Usage: `ihc::stream_in<datatype, /*template arguments*/>`

| Feature | Valid Values | Description |
|---|---|---|
| ihc::buffer | Positive int | FIFO buffer capacity in words (for inputs) |
| ihc::usesPackets | true or false | Exposes `startofpacket` and `endofpacket` signals |
| ihc::usesValid | true or false | Whether a valid signal is present (for inputs) |
| ihc::usesReady | true or false | Whether a ready signal is present (for outputs) |

Other attributes including `bitsPerSymbol and readylatency` described in the HLS Compiler Reference Manual

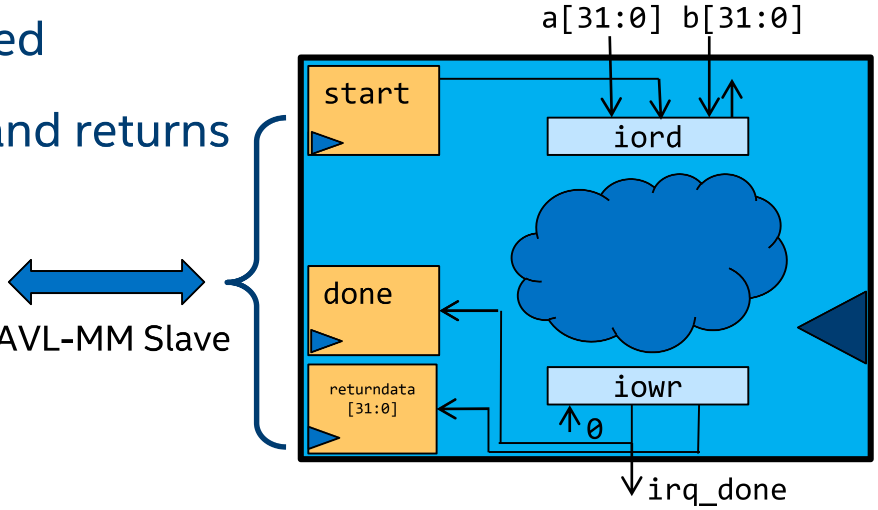
# Slaves Interfaces

- Component control and status register
  - In lieu of `start/busy/done/stall` signals
- Slave data registers
  - Ideal for smaller inputs
- Slave memories
  - For larger arrays

# MM Slave Component

- Component can have 1 CSR slave interface for function call and return
  - Shared with slave arguments
  - Address map described in generated <component_name>_csr.h
- irq_done signifies component is finished
- Used in place of default streaming calls and returns

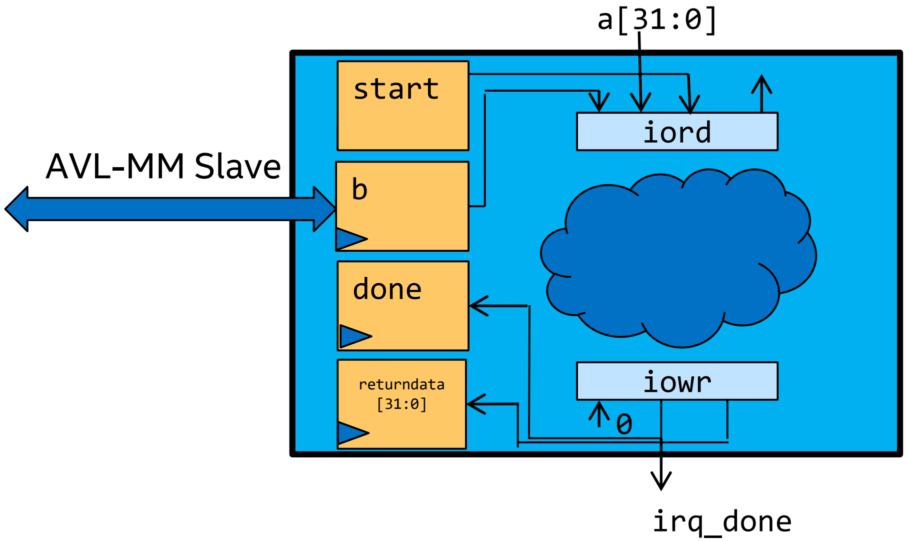
```
hls_avalon_slave_component
component int dut(…) {
    return result;
}
```



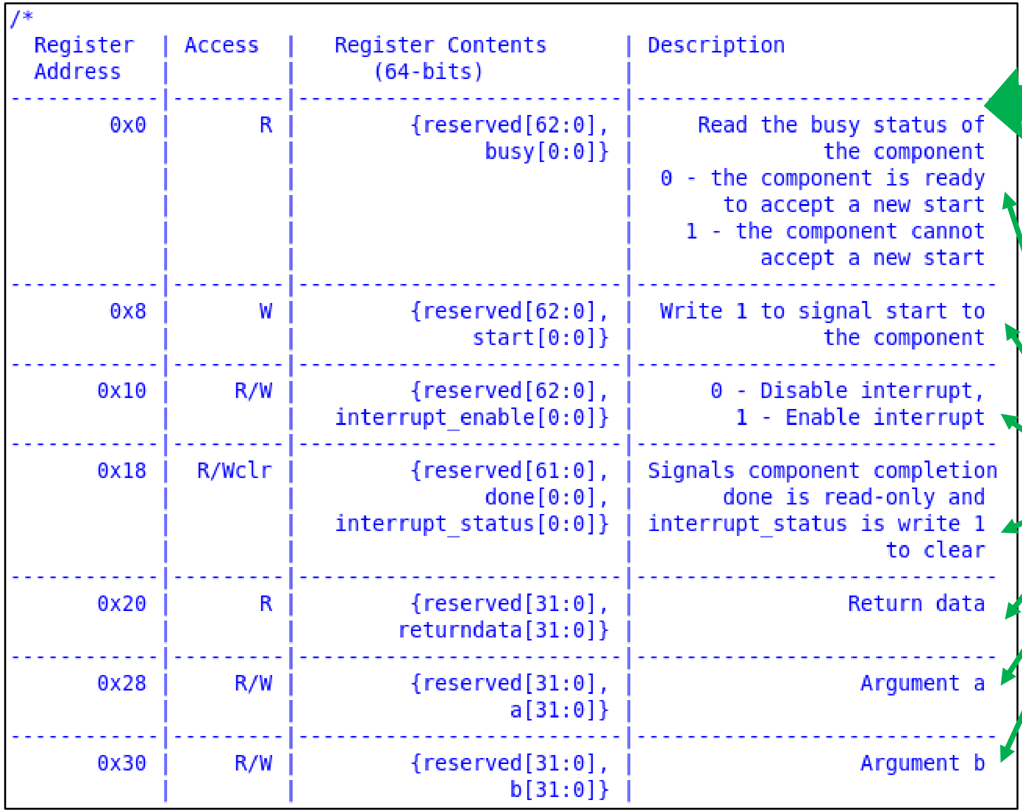
# MM Slave Register Argument

- Can be used independent of slave component
- Used in lieu of default conduit argument
- Ideal for smaller inputs

```
hls_avalon_slave_component component
int dut(int a,
        hls_avalon_slave_register_argument int b) {
    return a * b;
}
```



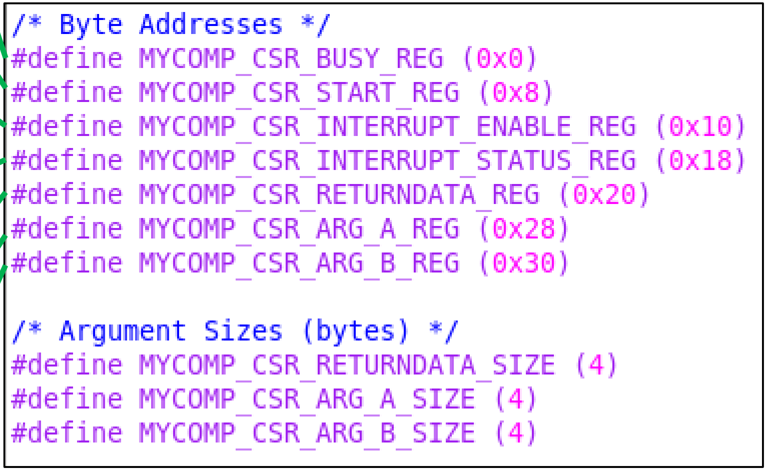
# Slave Component and Register Address Map



```
/*
  Register  | Access  |   Register Contents      | Description
  Address   |         |      (64-bits)           |
------------|---------|--------------------------|-----------------------------
        0x0 |       R |         {reserved[62:0], |     Read the busy status of
            |         |               busy[0:0]} |               the component
            |         |                          |  0 - the component is ready
            |         |                          |       to accept a new start
            |         |                          |    1 - the component cannot
            |         |                          |          accept a new start
------------|---------|--------------------------|-----------------------------
        0x8 |       W |         {reserved[62:0], |  Write 1 to signal start to
            |         |              start[0:0]} |               the component
------------|---------|--------------------------|-----------------------------
       0x10 |     R/W |         {reserved[62:0], |      0 - Disable interrupt,
            |         |   interrupt_enable[0:0]} |        1 - Enable interrupt
------------|---------|--------------------------|-----------------------------
       0x18 |  R/Wclr |         {reserved[61:0], | Signals component completion
            |         |               done[0:0], |       done is read-only and
            |         |   interrupt_status[0:0]} | interrupt_status is write 1
            |         |                          |                    to clear
------------|---------|--------------------------|-----------------------------
       0x20 |       R |         {reserved[31:0], |                 Return data
            |         |        returndata[31:0]} |
------------|---------|--------------------------|-----------------------------
       0x28 |     R/W |         {reserved[31:0], |                  Argument a
            |         |                 a[31:0]} |
------------|---------|--------------------------|-----------------------------
       0x30 |     R/W |         {reserved[31:0], |                  Argument b
            |         |                 b[31:0]} |
```

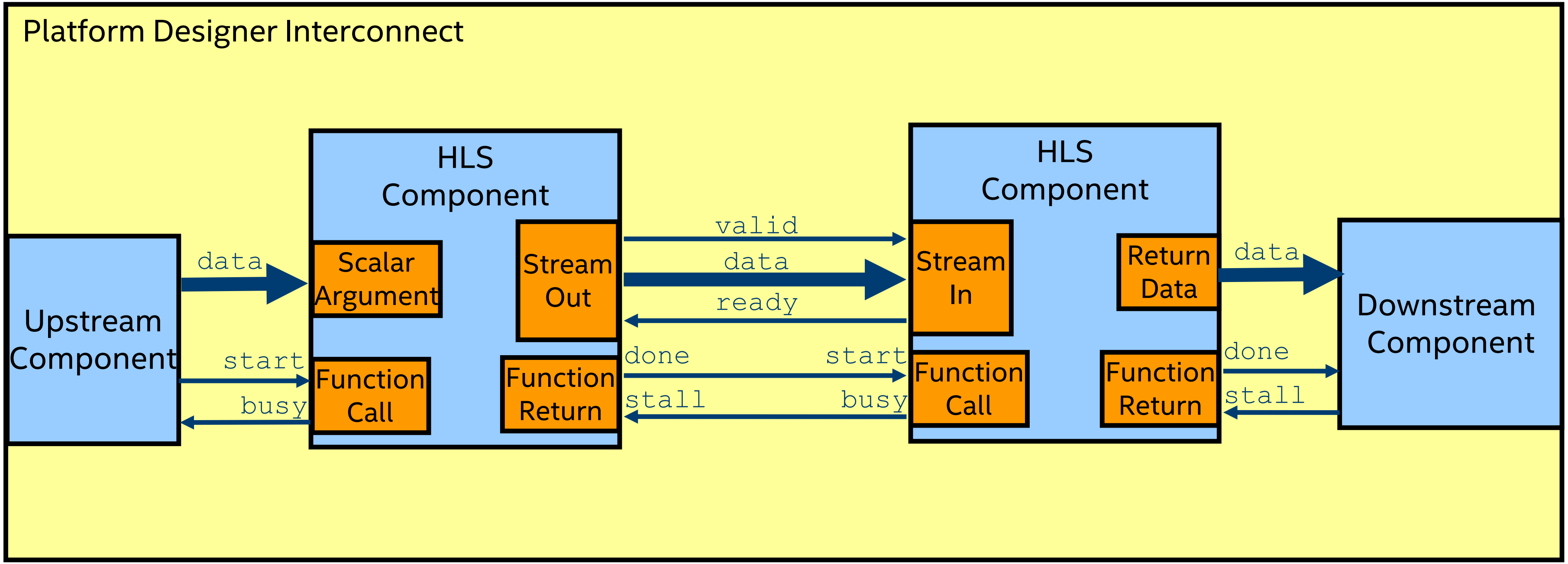
`<component>_csr.h` contains

- Address Map
- Macros created for register byte addresses and bit masks

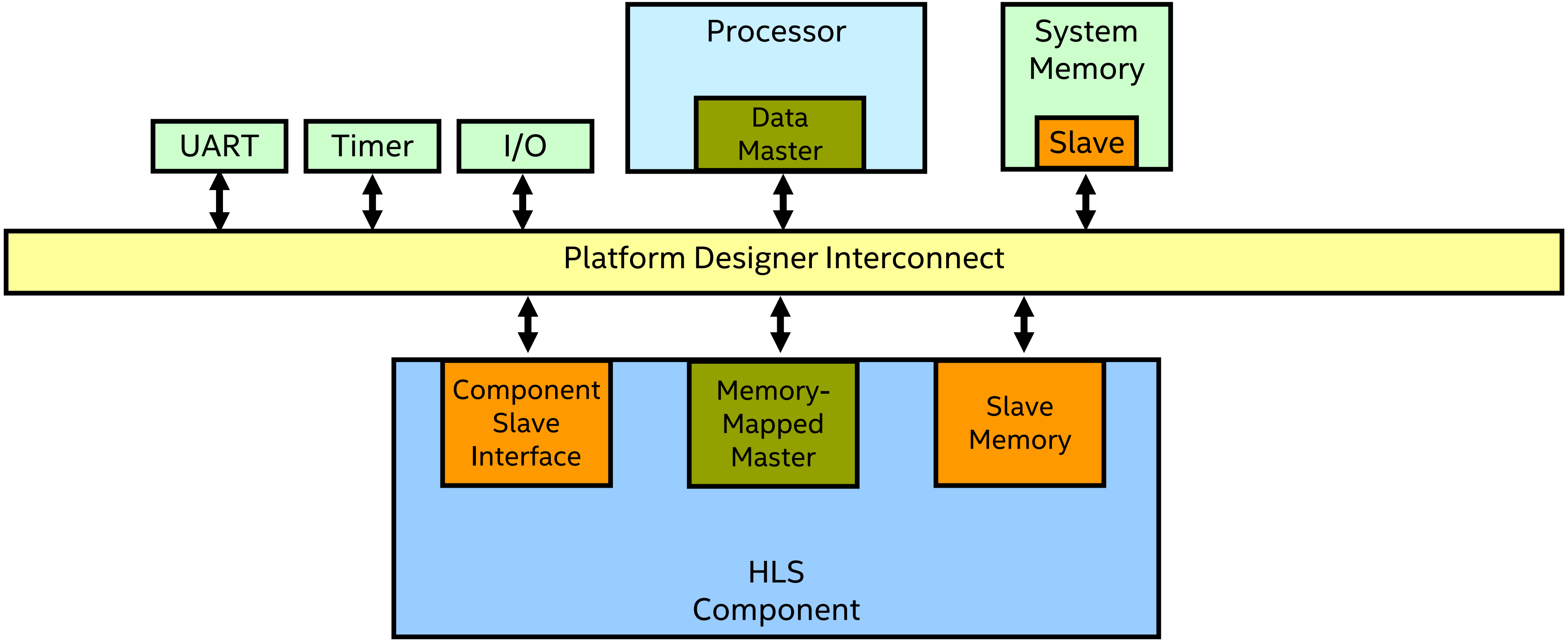


```
/* Byte Addresses */
#define MYCOMP_CSR_BUSY_REG (0x0)
#define MYCOMP_CSR_START_REG (0x8)
#define MYCOMP_CSR_INTERRUPT_ENABLE_REG (0x10)
#define MYCOMP_CSR_INTERRUPT_STATUS_REG (0x18)
#define MYCOMP_CSR_RETURNDATA_REG (0x20)
#define MYCOMP_CSR_ARG_A_REG (0x28)
#define MYCOMP_CSR_ARG_B_REG (0x30)

/* Argument Sizes (bytes) */
#define MYCOMP_CSR_RETURNDATA_SIZE (4)
#define MYCOMP_CSR_ARG_A_SIZE (4)
#define MYCOMP_CSR_ARG_B_SIZE (4)
```

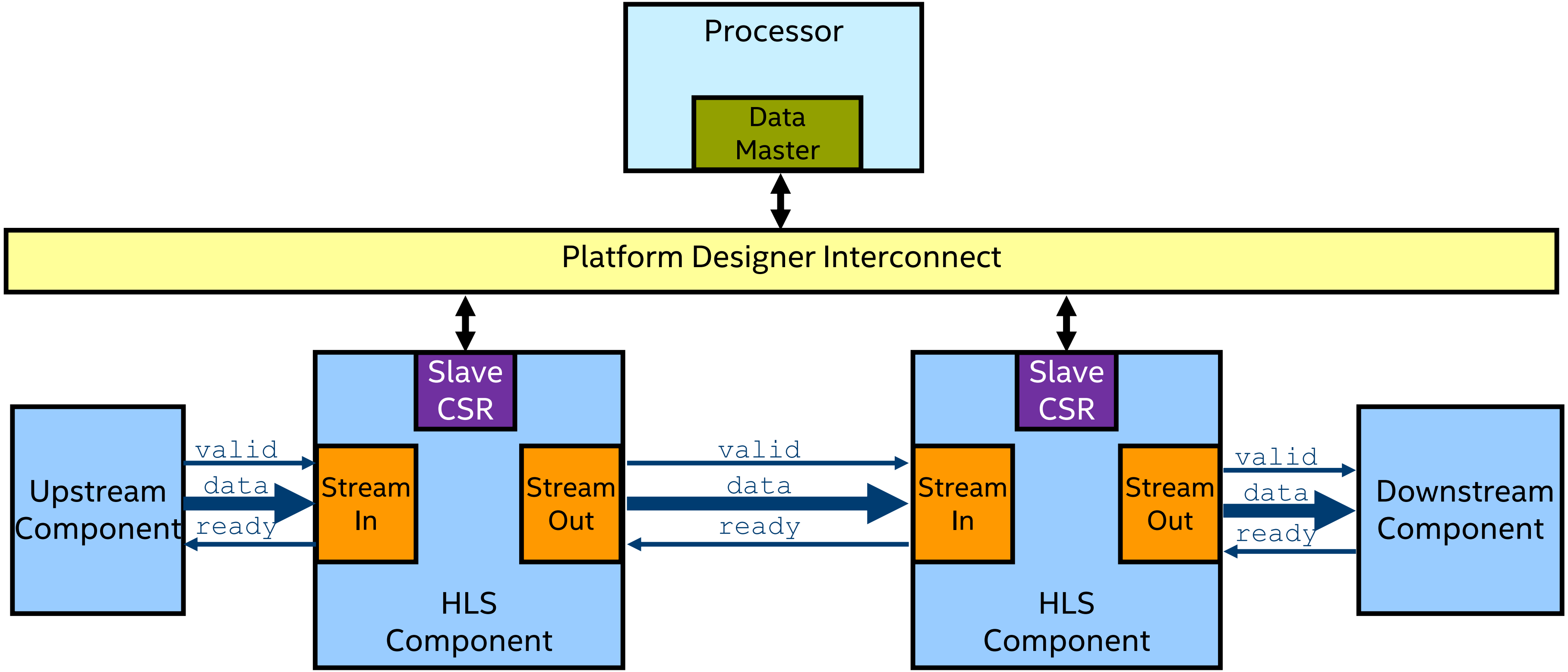
# Streaming HLS Component in a System



# Memory-Mapped HLS Component in a System



# MM HLS Component with Streaming Interfaces



# Interface Synthesis Tutorials

Located in `<hls_install_folder>/examples/tutorials/interfaces`

- explicit_streams_buffer
- explicit_streams_packets_ready_valid
- mm_master_testbench_operators
- mm_slaves
- multiple_stream_call_sites
- pointer_mm_master
- stable_arguments

# HLS AFU development flow

# HLS development flow

## Intel® High Level Synthesis Accelerator Functional Unit Design Example User Guide

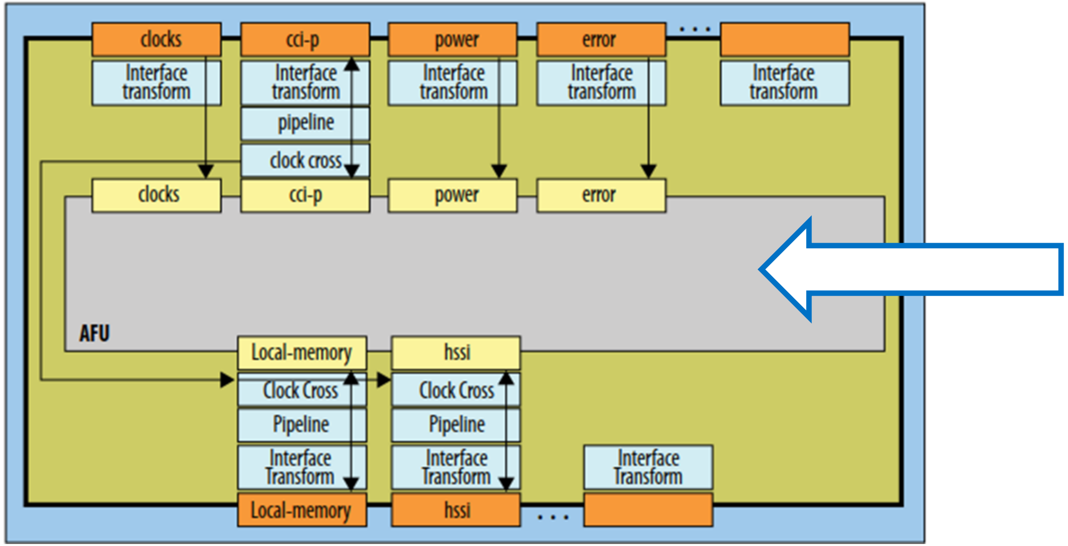
https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug-hls-afu.pdf

The Intel High Level Synthesis (HLS) Accelerator Functional Unit (AFU) design example shows how to create AFUs for the Intel® Acceleration Stack for Intel Xeon® CPU with FPGAs with with the Intel HLS.

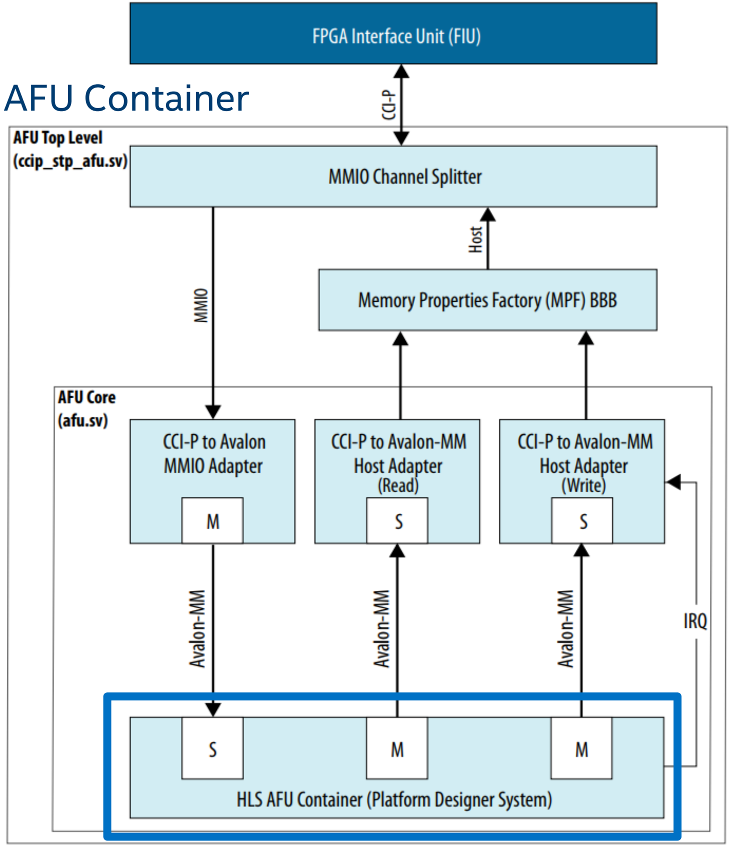
The package includes all the source code, scripts and makefile needed.

You can use this code as a model to create your own HLS AFUs if your AFUs use the same interfaces as the example design. Also, you might be able to convert your HLS application into an AFU by adding the required interfaces to the hardware design.

# HLS on Acceleration Stack

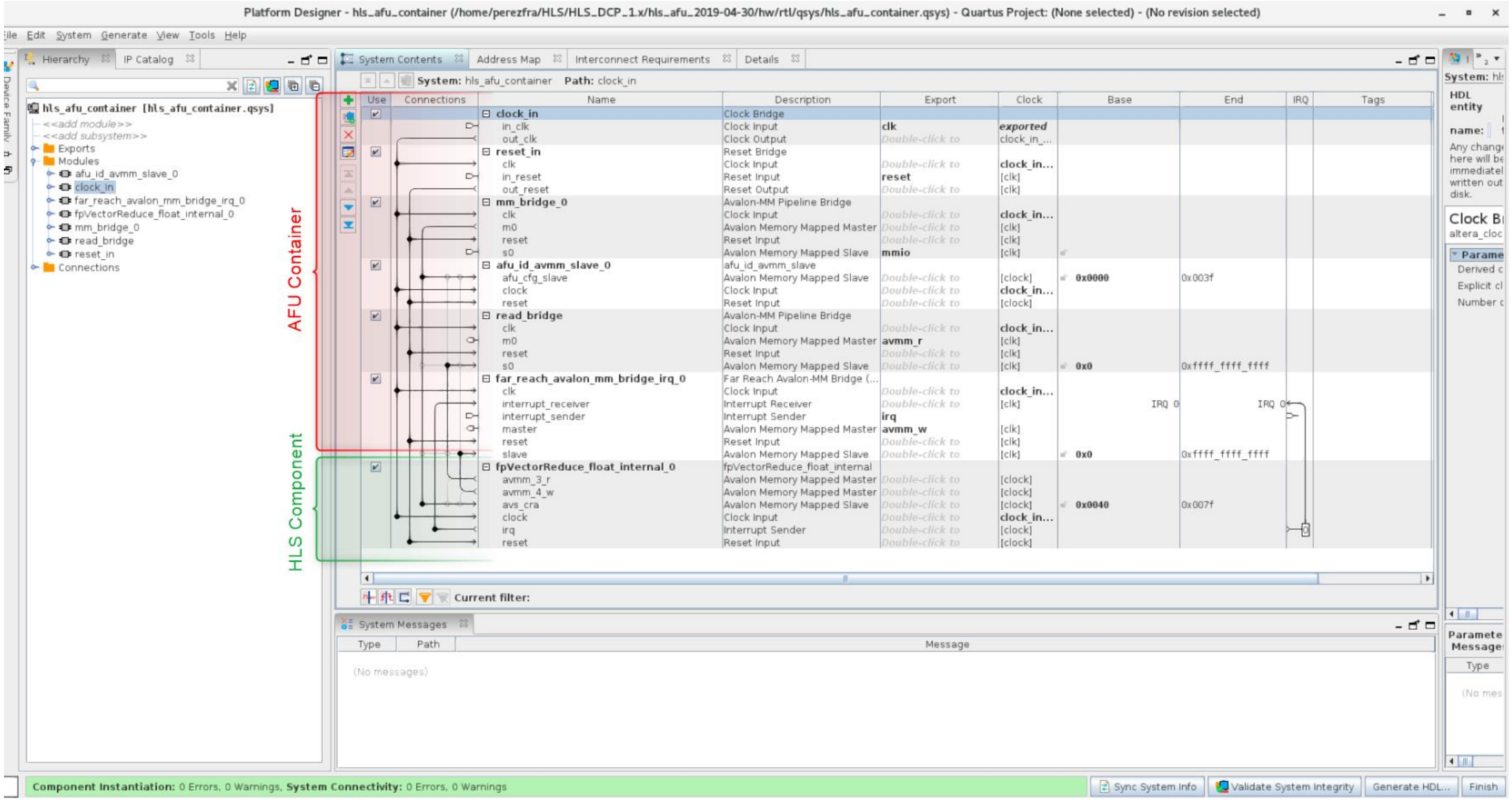


AFU Container


FPGA Interface Unit (FIU)
CCI-P
AFU Top Level
(ccip_stp_afu.sv)
MMIO Channel Splitter
Host
Memory Properties Factory (MPF) BBB
MMIO
AFU Core
(afu.sv)
CCI-P to Avalon MMIO Adapter
M
CCI-P to Avalon-MM Host Adapter (Read)
S
CCI-P to Avalon-MM Host Adapter (Write)
S
Avalon-MM
Avalon-MM
Avalon-MM
IRQ
S
M
M
HLS AFU Container (Platform Designer System)

HLS component to develop

# HLS AFU Container block diagram



Platform Designer - hls_afu_container (/home/perezfra/HLS/HLS_DCP_1.x/hls_afu_2019-04-30/hw/rtl/qsys/hls_afu_container.qsys) - Quartus Project: (None selected) - (No revision selected)

hls_afu_container [hls_afu_container.qsys]
- <<add module>>
- <<add subsystem>>
- Exports
- Modules
  - afu_id_avmm_slave_0
  - clock_in
  - far_reach_avalon_mm_bridge_irq_0
  - fpVectorReduce_float_internal_0
  - mm_bridge_0
  - read_bridge
  - reset_in
- Connections

**System:** hls_afu_container **Path:** clock_in

| Use | Connections | Name | Description | Export | Clock | Base | End | IRQ | Tags |
|---|---|---|---|---|---|---|---|---|---|
| ☑ | | ⊟ **clock_in** | Clock Bridge | | | | | | |
| | | in_clk | Clock Input | **clk** | ***exported*** | | | | |
| | | out_clk | Clock Output | *Double-click to* | clock_in_... | | | | |
| ☑ | | ⊟ **reset_in** | Reset Bridge | | | | | | |
| | | clk | Clock Input | *Double-click to* | **clock_in...** | | | | |
| | | in_reset | Reset Input | **reset** | [clk] | | | | |
| | | out_reset | Reset Output | *Double-click to* | [clk] | | | | |
| ☑ | | ⊟ **mm_bridge_0** | Avalon-MM Pipeline Bridge | | | | | | |
| | | clk | Clock Input | *Double-click to* | **clock_in...** | | | | |
| | | m0 | Avalon Memory Mapped Master | *Double-click to* | [clk] | | | | |
| | | reset | Reset Input | *Double-click to* | [clk] | | | | |
| | | s0 | Avalon Memory Mapped Slave | **mmio** | [clk] | | | | |
| ☑ | | ⊟ **afu_id_avmm_slave_0** | afu_id_avmm_slave | | | | | | |
| | | afu_cfg_slave | Avalon Memory Mapped Slave | *Double-click to* | [clock] | **0x0000** | 0x003f | | |
| | | clock | Clock Input | *Double-click to* | **clock_in...** | | | | |
| | | reset | Reset Input | *Double-click to* | [clock] | | | | |
| ☑ | | ⊟ **read_bridge** | Avalon-MM Pipeline Bridge | | | | | | |
| | | clk | Clock Input | *Double-click to* | **clock_in...** | | | | |
| | | m0 | Avalon Memory Mapped Master | **avmm_r** | [clk] | | | | |
| | | reset | Reset Input | *Double-click to* | [clk] | | | | |
| | | s0 | Avalon Memory Mapped Slave | *Double-click to* | [clk] | **0x0** | 0xffff_ffff_ffff | | |
| ☑ | | ⊟ **far_reach_avalon_mm_bridge_irq_0** | Far Reach Avalon-MM Bridge (... | | | | | | |
| | | clk | Clock Input | *Double-click to* | **clock_in...** | | | | |
| | | interrupt_receiver | Interrupt Receiver | *Double-click to* | [clk] | IRQ 0 | IRQ 0 | | |
| | | interrupt_sender | Interrupt Sender | **irq** | | | | | |
| | | master | Avalon Memory Mapped Master | **avmm_w** | [clk] | | | | |
| | | reset | Reset Input | *Double-click to* | [clk] | | | | |
| | | slave | Avalon Memory Mapped Slave | *Double-click to* | [clk] | **0x0** | 0xffff_ffff_ffff | | |
| ☑ | | ⊟ **fpVectorReduce_float_internal_0** | fpVectorReduce_float_internal | | | | | | |
| | | avmm_3_r | Avalon Memory Mapped Master | *Double-click to* | [clock] | | | | |
| | | avmm_4_w | Avalon Memory Mapped Master | *Double-click to* | [clock] | | | | |
| | | avs_cra | Avalon Memory Mapped Slave | *Double-click to* | [clock] | **0x0040** | 0x007f | | |
| | | clock | Clock Input | *Double-click to* | **clock_in...** | | | | |
| | | irq | Interrupt Sender | *Double-click to* | [clock] | | | 0 | |
| | | reset | Reset Input | *Double-click to* | [clock] | | | | |

**Current filter:**

| Type | Path | Message |
|---|---|---|
| (No messages) | | |

**System:** hls

**HDL entity**

**name:**

Any change here will be immediatel written out disk.

Clock Br

altera_cloc

Parame

Derived c

Explicit cl

Number c

Paramete Message

Type

(No mes

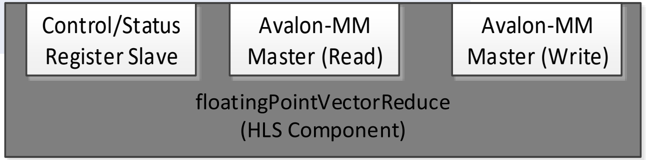
**Component Instantiation:** 0 Errors, 0 Warnings, **System Connectivity:** 0 Errors, 0 Warnings

# HLS on Acceleration Stack (basic vector reduce)

```
component
float floatingPointVectorReduce_basic(float *masterRead,
                                      float *masterWrite,
                                      int size)
{
  float sum = 0.0f;
  for (int idx = 0; idx < size; idx++)
  {
      float readVal = masterRead[idx];
      sum += readVal;

      masterWrite[idx] = readVal + 1.0f;
  }

  return sum;
}
```

# HLS on Acceleration Stack (HLS Signature)

```
typedef ihc::mm_master<float, ihc::dwidth<512>,
                       ihc::awidth<48>, ihc::latency<0>,
                       ihc::aspace<1>, ihc::readwrite_mode<readonly>,
                       ihc::waitrequest<true>, ihc::align<64>,
                       ihc::maxburst<4> > MasterReadFloat;

typedef ihc::mm_master<float, ihc::dwidth<512>,
                       ihc::awidth<48>, ihc::latency<0>,
                       ihc::aspace<2>, ihc::readwrite_mode<writeonly>,
                       ihc::waitrequest<true>, ihc::align<64>,
                       ihc::maxburst<4> > MasterWriteFloat;

component
hls_avalon_slave_component
float floatingPointVectorReduce_float (
      hls_avalon_slave_register_argument MasterReadFloat &masterRead,
      hls_avalon_slave_register_argument MasterWriteFloat &masterWrite,
      hls_avalon_slave_register_argument uint64_t size)

```


Control/Status Register Slave
Avalon-MM Master (Read)
Avalon-MM Master (Write)
floatingPointVectorReduce (HLS Component)

# HLS on Acceleration Stack (Code Body)

#define UNROLL_FACTOR 16

// 16 32-bit floats in 1 512-bit dword

#define FLOAT_BITS 32

// 32 bits in one float

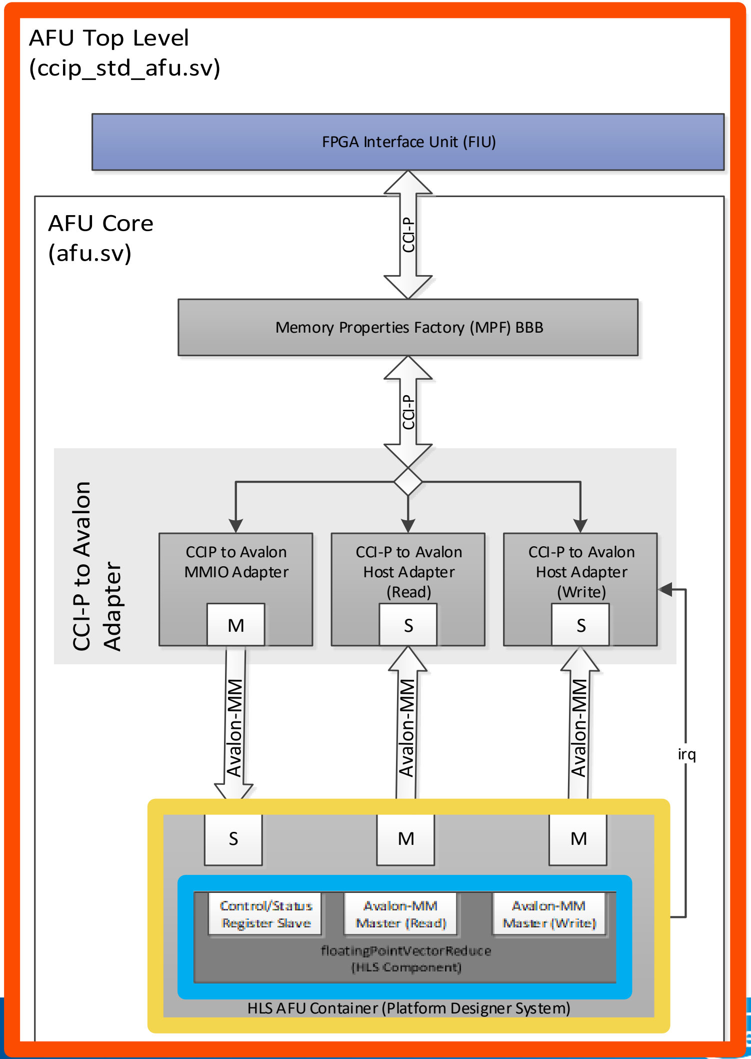
We can transfer up to 16x 32bit float in one 512 data bus cycle

We unroll to make it in parallel

```
{
 float sum = 0.0f;
 int iterations = 1 + ((size - 1) / UNROLL_FACTOR);
 for (int loop_idx = 0; loop_idx < iterations; loop_idx++)
 {
   float readSum = 0.0f;
#pragma unroll UNROLL_FACTOR
   for (int itr = 0; itr < UNROLL_FACTOR; itr++)
   {
     int idx = itr + (loop_idx * UNROLL_FACTOR);
     if (idx < size)
     {
       float readVal = masterRead[idx];
       readSum += readVal;
       masterWrite[idx] = readVal + 1.0f;
     }
   }
   sum += readSum;
 }
 return sum;
}
```

# HLS AFU Flow Overview

1. Build/Verify HLS code
2. Insert into Platform Designer
3. Build with Acceleration Stack tools
   - Either ASE, or AF Bitstream
4. Build and run host



AFU Top Level
(ccip_std_afu.sv)

FPGA Interface Unit (FIU)

CCI-P

AFU Core
(afu.sv)

Memory Properties Factory (MPF) BBB

CCI-P

CCI-P to Avalon Adapter

CCIP to Avalon MMIO Adapter

M

CCI-P to Avalon Host Adapter (Read)

S

CCI-P to Avalon Host Adapter (Write)

S

Avalon-MM

Avalon-MM

Avalon-MM

irq

S

M

M

Control/Status Register Slave

Avalon-MM Master (Read)

Avalon-MM Master (Write)

floatingPointVectorReduce
(HLS Component)

HLS AFU Container (Platform Designer System)

# Compiling and Simulating the HLS Component with the i++ Command

- We compile this example design using the included `makefile`
- Build and emulate the design using x86 instructions run these commands:

```
$ make test-x86-64
$ ./test-x86-64
```

- Generate RTL and simulate generated RTL with the ModelSim simulator:

```
$ make test-fpga
$ ./test-fpga
```

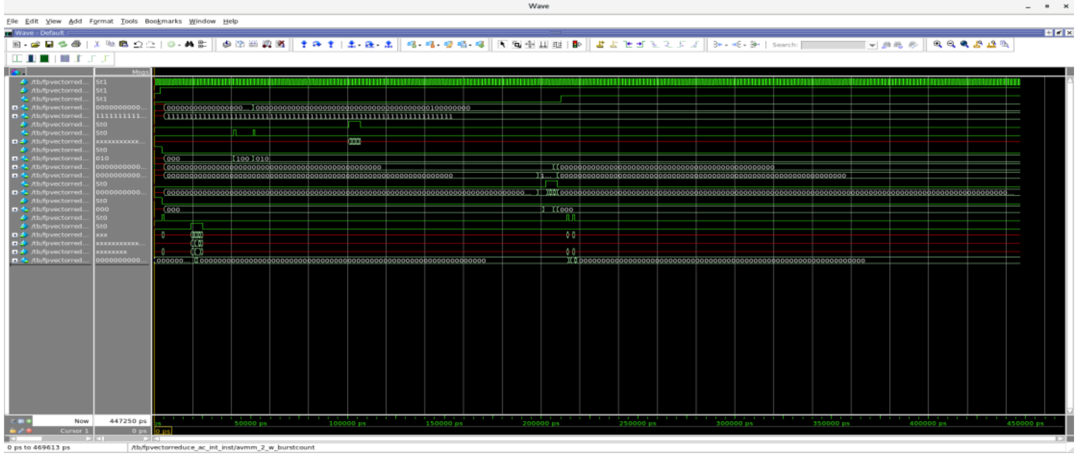
**Confirm that the outputs from the test-x86-64 the test-fpga command match.**
The test-x86-64 command runs C++ code on the processor, while the test-fpga command compiles the C++ source to Verilog RTL and then simulates the generate RTL using the testbench defined in the code.

# Viewing waves in simulator (opcional)

As we have built the component using -ghdl the ModelSim testbench generated will log all HDL signals in a wlf file

```
$ vsim fpga_ghdl.prj/verification/vsim.wlf
```

Add the desired signals to waveform viewer in the selected simulator



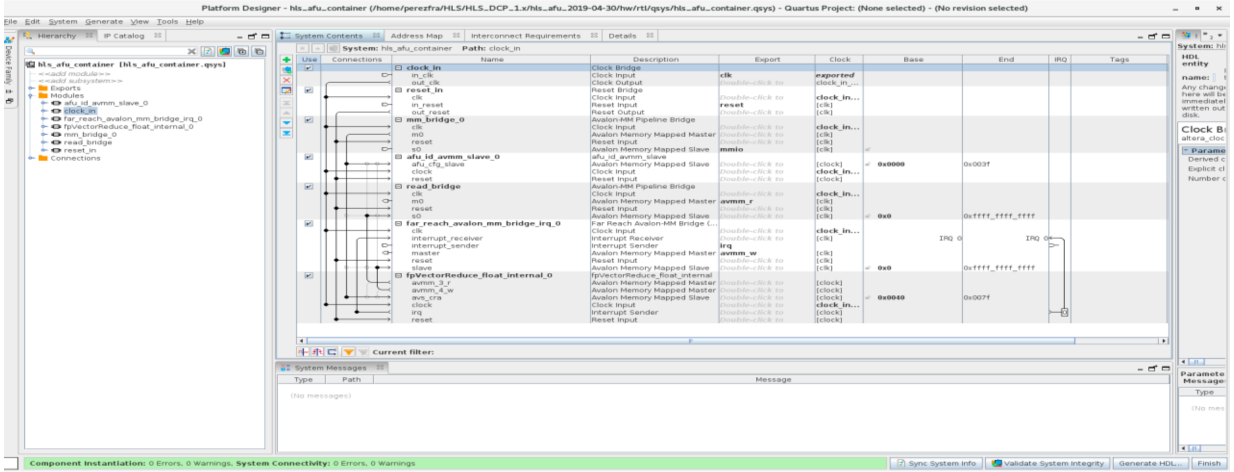
| | Msgs |
|---|---|
| /tb/fpvectorred... | St1 |
| /tb/fpvectorred... | St1 |
| /tb/fpvectorred... | St1 |
| /tb/fpvectorred... | 0000000000... |
| /tb/fpvectorred... | 1111111111... |
| /tb/fpvectorred... | St0 |
| /tb/fpvectorred... | St0 |
| /tb/fpvectorred... | xxxxxxxxxxx... |
| /tb/fpvectorred... | St0 |
| /tb/fpvectorred... | 010 |
| /tb/fpvectorred... | 0000000000... |
| /tb/fpvectorred... | 0000000000... |
| /tb/fpvectorred... | St0 |
| /tb/fpvectorred... | 0000000000... |
| /tb/fpvectorred... | St0 |
| /tb/fpvectorred... | 000 |
| /tb/fpvectorred... | St0 |
| /tb/fpvectorred... | St0 |
| /tb/fpvectorred... | xxx |
| /tb/fpvectorred... | xxxxxxxxxxx... |
| /tb/fpvectorred... | xxxxxxxx |
| /tb/fpvectorred... | 0000000000... |
| Now | 447250 ps |
| Cursor 1 | 0 ps |

0 ps to 469613 ps
/tb/fpvectorreduce_ac_int_inst/avmm_2_w_burstcount

# Generating a Platform Designer container for the HLS component

Use Platform Designer to integrate the HLS component into an AFU with the predesigned hardware interfaces available in the Acceleration Stack, and verify that all sources are linked correctly.

```
$ qsys-edit hls_afu_container.qsys
```



Platform Designer - hls_afu_container (/home/perezfra/HLS/HLS_DCP_1.x/hls_afu_2019-04-30/hw/rtl/qsys/hls_afu_container.qsys) - Quartus Project: (None selected) - (No revision selected)

File Edit System Generate View Tools Help
Hierarchy
IP Catalog
System Contents
Address Map
Interconnect Requirements
Details
Device Family

hls_afu_container [hls_afu_container.qsys]
- <<add module>>
- <<add subsystem>>
- Exports
- Modules
  - afu_id_avmm_slave_0
  - clock_in
  - far_reach_avalon_mm_bridge_irq_0
  - fpVectorReduce_float_internal_0
  - mm_bridge_0
  - read_bridge
  - reset_in
- Connections

System: hls_afu_container Path: clock_in

| Use | Name | Description | Export | Clock | Base | End | IRQ | Tags |
|---|---|---|---|---|---|---|---|---|
| ☑ | clock_in | Clock Bridge | | | | | | |
| | in_clk | Clock Input | clk | exported | | | | |
| | out_clk | Clock Output | Double-click to | clock_in_... | | | | |
| ☑ | reset_in | Reset Bridge | | | | | | |
| | clk | Clock Input | Double-click to | clock_in... | | | | |
| | in_reset | Reset Input | reset | [clk] | | | | |
| | out_reset | Reset Output | Double-click to | [clk] | | | | |
| ☑ | mm_bridge_0 | Avalon-MM Pipeline Bridge | | | | | | |
| | clk | Clock Input | Double-click to | clock_in... | | | | |
| | m0 | Avalon Memory Mapped Master | Double-click to | [clk] | | | | |
| | reset | Reset Input | Double-click to | [clk] | | | | |
| | s0 | Avalon Memory Mapped Slave | mmio | [clk] | | | | |
| ☑ | afu_id_avmm_slave_0 | afu_id_avmm_slave | | | | | | |
| | afu_cfg_slave | Avalon Memory Mapped Slave | Double-click to | [clock] | 0x0000 | 0x003f | | |
| | clock | Clock Input | Double-click to | clock_in... | | | | |
| | reset | Reset Input | Double-click to | [clock] | | | | |
| ☑ | read_bridge | Avalon-MM Pipeline Bridge | | | | | | |
| | clk | Clock Input | Double-click to | clock_in... | | | | |
| | m0 | Avalon Memory Mapped Master | avmm_r | [clk] | | | | |
| | reset | Reset Input | Double-click to | [clk] | | | | |
| | s0 | Avalon Memory Mapped Slave | Double-click to | [clk] | 0x0 | 0xffff_ffff_ffff | | |
| ☑ | far_reach_avalon_mm_bridge_irq_0 | Far Reach Avalon-MM Bridge (... | | | | | | |
| | clk | Clock Input | Double-click to | clock_in... | | | | |
| | interrupt_receiver | Interrupt Receiver | Double-click to | [clk] | IRQ 0 | IRQ 0 | | |
| | interrupt_sender | Interrupt Sender | irq | | | | | |
| | master | Avalon Memory Mapped Master | avmm_w | [clk] | | | | |
| | reset | Reset Input | Double-click to | [clk] | | | | |
| | slave | Avalon Memory Mapped Slave | Double-click to | [clk] | 0x0 | 0xffff_ffff_ffff | | |
| ☑ | fpVectorReduce_float_internal_0 | fpVectorReduce_float_internal | | | | | | |
| | avmm_3_r | Avalon Memory Mapped Master | Double-click to | [clock] | | | | |
| | avmm_4_w | Avalon Memory Mapped Master | Double-click to | [clock] | | | | |
| | avs_cra | Avalon Memory Mapped Slave | Double-click to | [clock] | 0x0040 | 0x007f | | |
| | clock | Clock Input | Double-click to | clock_in... | | | | |
| | irq | Interrupt Sender | Double-click to | [clock] | | | 0 | |
| | reset | Reset Input | Double-click to | [clock] | | | | |

Current filter:

System Messages

| Type | Path | Message |
|---|---|---|
| (No messages) | | |

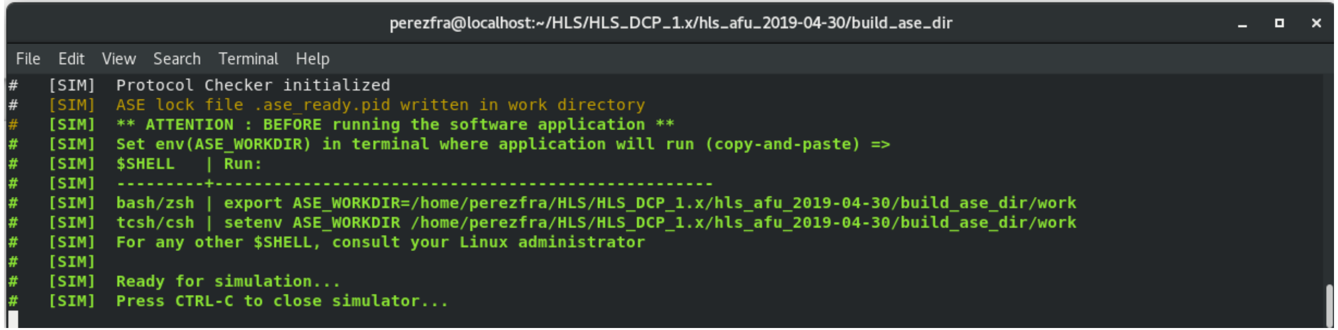
Component Instantiation: 0 Errors, 0 Warnings, System Connectivity: 0 Errors, 0 Warnings

# Cosimulation using ASE testbench

After integrating the HLS component into an AFU, you might want to cosimulate the AFU in the Intel AFU Simulation Environment (ASE), to quickly confirm the functionality of your HLS component within the AFU.

To simulate using ASE, navigate to the root of your project (the hls_afu directory) and run:

```
$ afu_sim_setup --source hw/rtl/filelist.txt build_ase_dir/
$ make
$ make sim
```


perezfra@localhost:~/HLS/HLS_DCP_1.x/hls_afu_2019-04-30/build_ase_dir
File Edit View Search Terminal Help

```
#   [SIM]  Protocol Checker initialized
#   [SIM]  ASE lock file .ase_ready.pid written in work directory
#   [SIM]  ** ATTENTION : BEFORE running the software application **
#   [SIM]  Set env(ASE_WORKDIR) in terminal where application will run (copy-and-paste) =>
#   [SIM]  $SHELL   | Run:
#   [SIM]  ---------+---------------------------------------------------
#   [SIM]  bash/zsh | export ASE_WORKDIR=/home/perezfra/HLS/HLS_DCP_1.x/hls_afu_2019-04-30/build_ase_dir/work
#   [SIM]  tcsh/csh | setenv ASE_WORKDIR /home/perezfra/HLS/HLS_DCP_1.x/hls_afu_2019-04-30/build_ase_dir/work
#   [SIM]  For any other $SHELL, consult your Linux administrator
#   [SIM]
#   [SIM]  Ready for simulation...
#   [SIM]  Press CTRL-C to close simulator...
```

# Cosimulation using ASE testbench

Open a new terminal window to compile the host application

Export the ASE_WORKDIR environment variable using the export command from the output of the make sim command in the ASE terminal window.

```
$ export ASE_WORKDIR=<path to work folder>
```

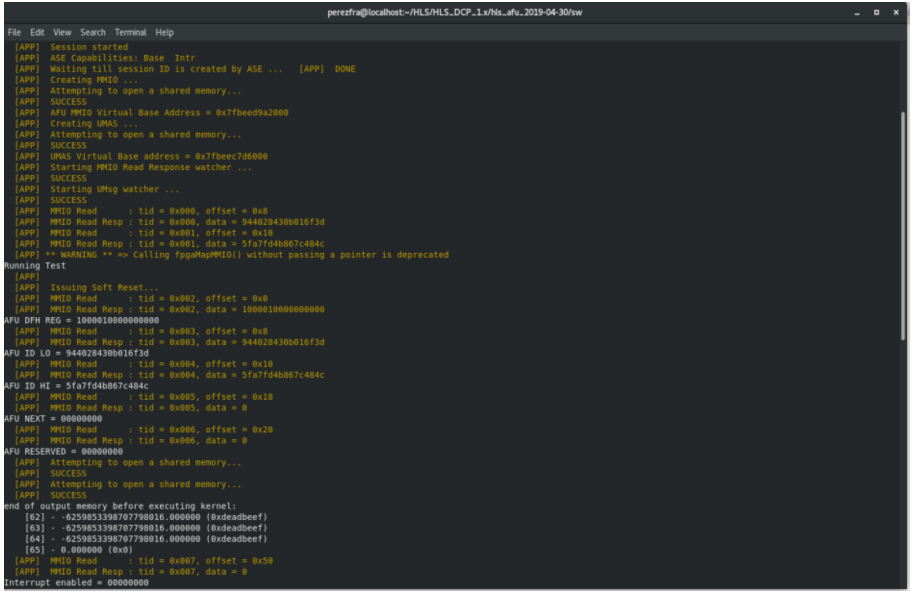
Build the host application with simulation support and run

```
$ make USE_ASE=1
```

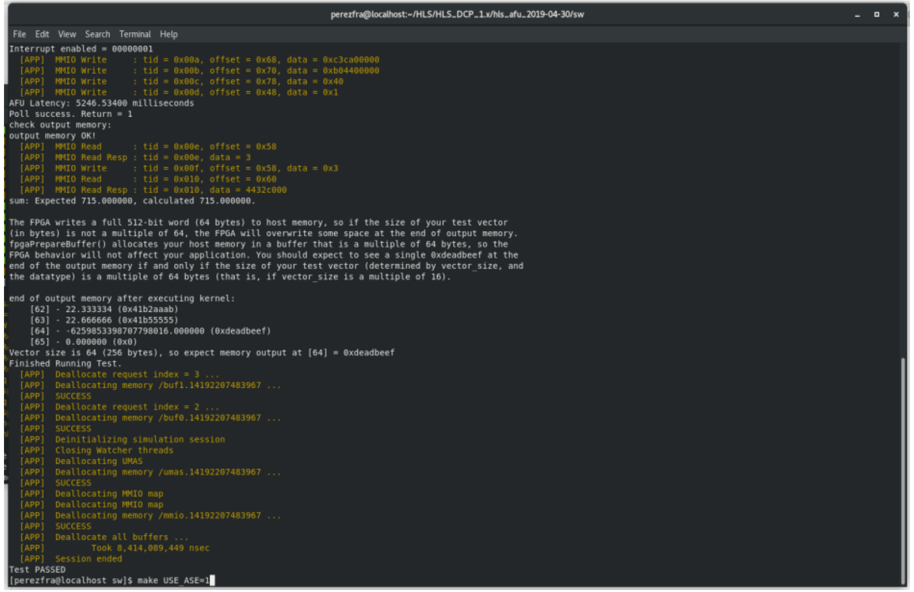
```
$ ./hls_afu_host
```

# Cosimulation using ASE testbench

## Host terminal window with all transactions



```
  [APP]  Session started
  [APP]  ASE Capabilities: Base  Intr
  [APP]  Waiting till session ID is created by ASE ...   [APP]  DONE
  [APP]  Creating MMIO ...
  [APP]  Attempting to open a shared memory...
  [APP]  SUCCESS
  [APP]  AFU MMIO Virtual Base Address = 0x7fbeed9a2000
  [APP]  Creating UMAS ...
  [APP]  Attempting to open a shared memory...
  [APP]  SUCCESS
  [APP]  UMAS Virtual Base address = 0x7fbeec7d6000
  [APP]  Starting MMIO Read Response watcher ...
  [APP]  SUCCESS
  [APP]  Starting UMsg watcher ...
  [APP]  SUCCESS
  [APP]  MMIO Read      : tid = 0x000, offset = 0x8
  [APP]  MMIO Read Resp : tid = 0x000, data = 944028430b016f3d
  [APP]  MMIO Read      : tid = 0x001, offset = 0x10
  [APP]  MMIO Read Resp : tid = 0x001, data = 5fa7fd4b867c484c
  [APP] ** WARNING ** => Calling fpgaMapMMIO() without passing a pointer is deprecated
Running Test
  [APP]
  [APP]  Issuing Soft Reset...
  [APP]  MMIO Read      : tid = 0x002, offset = 0x0
  [APP]  MMIO Read Resp : tid = 0x002, data = 1000010000000000
AFU DFH REG = 1000010000000000
  [APP]  MMIO Read      : tid = 0x003, offset = 0x8
  [APP]  MMIO Read Resp : tid = 0x003, data = 944028430b016f3d
AFU ID LO = 944028430b016f3d
  [APP]  MMIO Read      : tid = 0x004, offset = 0x10
  [APP]  MMIO Read Resp : tid = 0x004, data = 5fa7fd4b867c484c
AFU ID HI = 5fa7fd4b867c484c
  [APP]  MMIO Read      : tid = 0x005, offset = 0x18
  [APP]  MMIO Read Resp : tid = 0x005, data = 0
AFU NEXT = 00000000
  [APP]  MMIO Read      : tid = 0x006, offset = 0x20
  [APP]  MMIO Read Resp : tid = 0x006, data = 0
AFU RESERVED = 00000000
  [APP]  Attempting to open a shared memory...
  [APP]  SUCCESS
  [APP]  Attempting to open a shared memory...
  [APP]  SUCCESS
end of output memory before executing kernel:
    [62] - -6259853398707798016.000000 (0xdeadbeef)
    [63] - -6259853398707798016.000000 (0xdeadbeef)
    [64] - -6259853398707798016.000000 (0xdeadbeef)
    [65] - 0.000000 (0x0)
  [APP]  MMIO Read      : tid = 0x007, offset = 0x50
  [APP]  MMIO Read Resp : tid = 0x007, data = 0
Interrupt enabled = 00000000
```

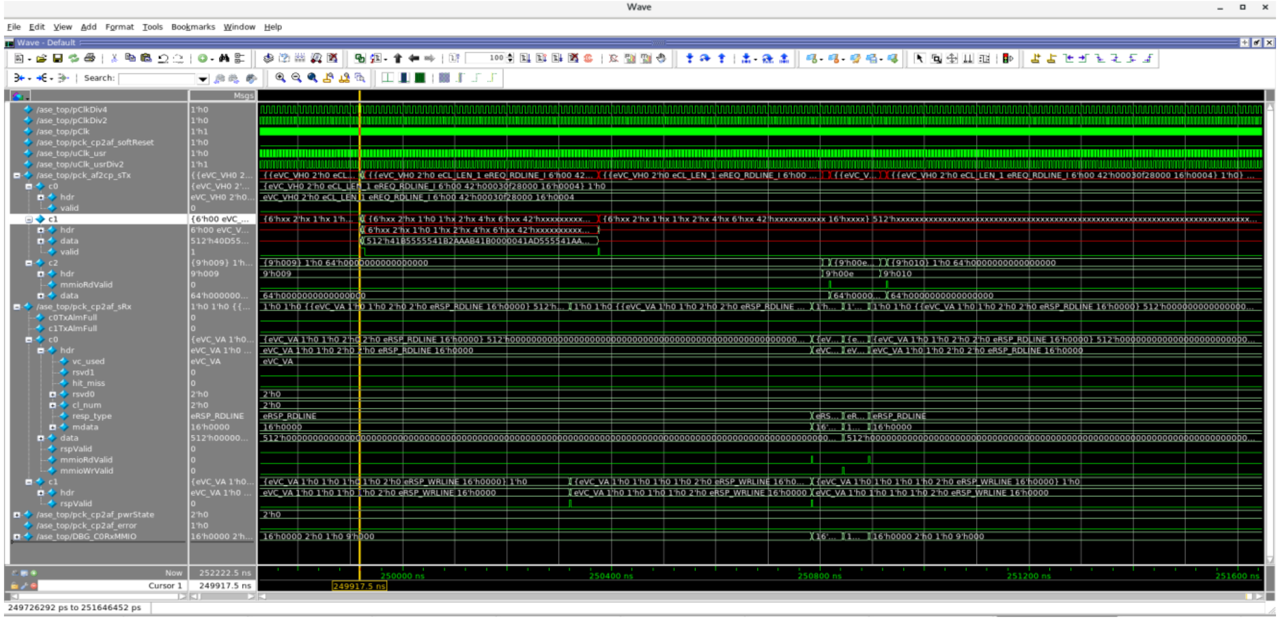


```
Interrupt enabled = 00000001
  [APP]  MMIO Write     : tid = 0x00a, offset = 0x68, data = 0xc3ca00000
  [APP]  MMIO Write     : tid = 0x00b, offset = 0x70, data = 0xb04400000
  [APP]  MMIO Write     : tid = 0x00c, offset = 0x78, data = 0x40
  [APP]  MMIO Write     : tid = 0x00d, offset = 0x48, data = 0x1
AFU Latency: 5246.53400 milliseconds
Poll success. Return = 1
check output memory:
output memory OK!
  [APP]  MMIO Read      : tid = 0x00e, offset = 0x58
  [APP]  MMIO Read Resp : tid = 0x00e, data = 3
  [APP]  MMIO Write     : tid = 0x00f, offset = 0x58, data = 0x3
  [APP]  MMIO Read      : tid = 0x010, offset = 0x60
  [APP]  MMIO Read Resp : tid = 0x010, data = 4432c000
sum: Expected 715.000000, calculated 715.000000.

The FPGA writes a full 512-bit word (64 bytes) to host memory, so if the size of your test vector
(in bytes) is not a multiple of 64, the FPGA will overwrite some space at the end of output memory.
fpgaPrepareBuffer() allocates your host memory in a buffer that is a multiple of 64 bytes, so the
FPGA behavior will not affect your application. You should expect to see a single 0xdeadbeef at the
end of the output memory if and only if the size of your test vector (determined by vector_size, and
the datatype) is a multiple of 64 bytes (that is, if vector_size is a multiple of 16).

end of output memory after executing kernel:
    [62] - 22.333334 (0x41b2aaab)
    [63] - 22.666666 (0x41b55555)
    [64] - -6259853398707798016.000000 (0xdeadbeef)
    [65] - 0.000000 (0x0)
Vector size is 64 (256 bytes), so expect memory output at [64] = 0xdeadbeef
Finished Running Test.
  [APP]  Deallocate request index = 3 ...
  [APP]  Deallocating memory /buf1.14192207483967 ...
  [APP]  SUCCESS
  [APP]  Deallocate request index = 2 ...
  [APP]  Deallocating memory /buf0.14192207483967 ...
  [APP]  SUCCESS
  [APP]  Deinitializing simulation session
  [APP]  Closing Watcher threads
  [APP]  Deallocating UMAS
  [APP]  Deallocating memory /umas.14192207483967 ...
  [APP]  SUCCESS
  [APP]  Deallocating MMIO map
  [APP]  Deallocating MMIO map
  [APP]  Deallocating memory /mmio.14192207483967 ...
  [APP]  SUCCESS
  [APP]  Deallocate all buffers ...
  [APP]        Took 8,414,089,449 nsec
  [APP]  Session ended
Test PASSED
[perezfra@localhost sw]$ make USE_ASE=1
```

# Cosimulation using ASE testbench

The waveform, CCI-P transactions, and simulation log files are stored in the simulation work directory. To view the waveform database, type:

```
$ make wave
```



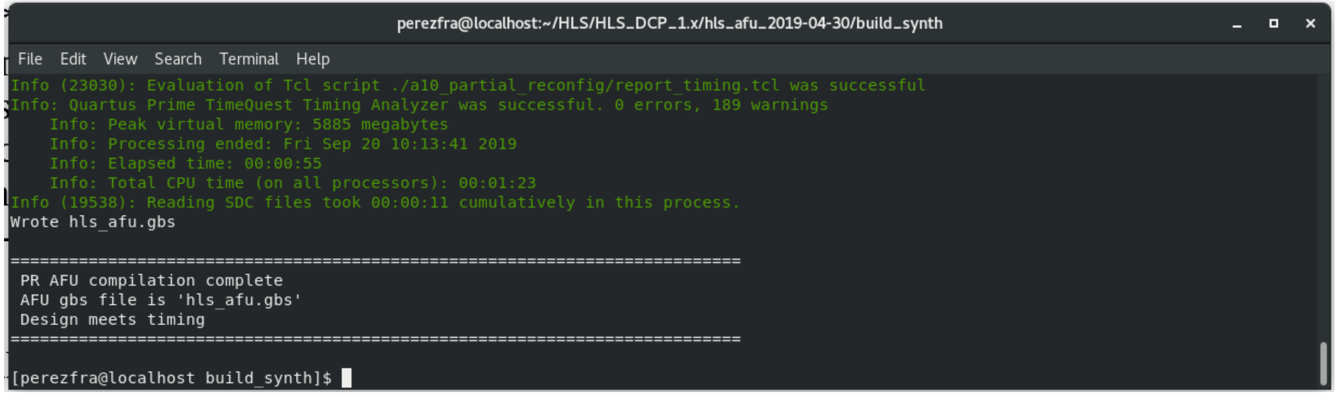
| | Msgs |
|---|---|
| /ase_top/pClkDiv4 | 1'h0 |
| /ase_top/pClkDiv2 | 1'h0 |
| /ase_top/pClk | 1'h1 |
| /ase_top/pck_cp2af_softReset | 1'h0 |
| /ase_top/uClk_usr | 1'h0 |
| /ase_top/uClk_usrDiv2 | 1'h1 |
| /ase_top/pck_af2cp_sTx | {{eVC_VH0 2... |
| c0 | {eVC_VH0 2'... |
| hdr | eVC_VH0 2'h0... |
| valid | 0 |
| c1 | {6'h00 eVC_... |
| hdr | 6'h00 eVC_V... |
| data | 512'h40D55... |
| valid | 1 |
| c2 | {9'h009} 1'h... |
| hdr | 9'h009 |
| mmioRdValid | 0 |
| data | 64'h000000... |
| /ase_top/pck_cp2af_sRx | 1'h0 1'h0 {{... |
| c0TxAlmFull | 0 |
| c1TxAlmFull | 0 |
| c0 | {eVC_VA 1'h0... |
| hdr | eVC_VA 1'h0 ... |
| vc_used | eVC_VA |
| rsvd1 | 0 |
| hit_miss | 0 |
| rsvd0 | 2'h0 |
| cl_num | 2'h0 |
| resp_type | eRSP_RDLINE |
| mdata | 16'h0000 |
| data | 512'h00000... |
| rspValid | 0 |
| mmioRdValid | 0 |
| mmioWrValid | 0 |
| c1 | {eVC_VA 1'h0... |
| hdr | eVC_VA 1'h0 ... |
| rspValid | 0 |
| /ase_top/pck_cp2af_pwrState | 2'h0 |
| /ase_top/pck_cp2af_error | 1'h0 |
| /ase_top/DBG_C0RxMMIO | 16'h0000 2'h... |
| Now | 252222.5 ns |
| Cursor 1 | 249917.5 ns |

# Synthesizing the AFU

Generate the AF build environment and create the AF (.gbs) image.

```
$ afu_synth_setup --source hw/rtl/filelist.txt build_synth
$ cd build_synth
$ $OPAE_PLATFORM_ROOT/bin/run.sh
```

When the AFU is created successfully, you get the following message:

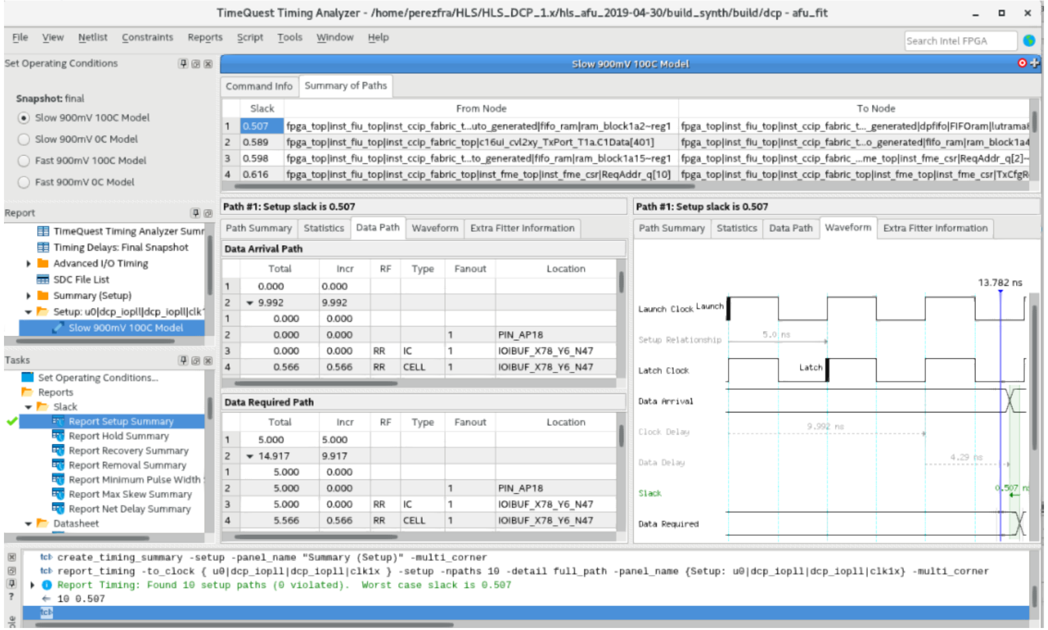

perezfra@localhost:~/HLS/HLS_DCP_1.x/hls_afu_2019-04-30/build_synth
File Edit View Search Terminal Help

```
Info (23030): Evaluation of Tcl script ./a10_partial_reconfig/report_timing.tcl was successful
Info: Quartus Prime TimeQuest Timing Analyzer was successful. 0 errors, 189 warnings
    Info: Peak virtual memory: 5885 megabytes
    Info: Processing ended: Fri Sep 20 10:13:41 2019
    Info: Elapsed time: 00:00:55
    Info: Total CPU time (on all processors): 00:01:23
Info (19538): Reading SDC files took 00:00:11 cumulatively in this process.
Wrote hls_afu.gbs

===========================================================================
 PR AFU compilation complete
 AFU gbs file is 'hls_afu.gbs'
 Design meets timing
===========================================================================

[perezfra@localhost build_synth]$ 
```

# Synthesizing the AFU

The run.sh script indicates the status of timing closure – make sure the generated AF has no hardware timing violations.

Optional:
Open the dcp.qpf Quartus project file in the Quartus Prime Pro GUI with the synthesis build project's afu_fit revision to view the details of the timing report and perform interactive timing analysis.



|  | Slack | From Node | To Node |
|---|---|---|---|
| 1 | 0.507 | fpga_top\|inst_fiu_top\|inst_ccip_fabric_t...uto_generated\|fifo_ram\|ram_block1a2~reg1 | fpga_top\|inst_fiu_top\|inst_ccip_fabric_t..._generated\|dpfifo\|FIFOram\|lutrama |
| 2 | 0.589 | fpga_top\|inst_fiu_top\|inst_ccip_fabric_top\|c16ui_cvl2xy_TxPort_T1a.C1Data[401] | fpga_top\|inst_fiu_top\|inst_ccip_fabric_t...o_generated\|fifo_ram\|ram_block1a4 |
| 3 | 0.598 | fpga_top\|inst_fiu_top\|inst_ccip_fabric_t...to_generated\|fifo_ram\|ram_block1a15~reg1 | fpga_top\|inst_fiu_top\|inst_ccip_fabric_...me_top\|inst_fme_csr\|ReqAddr_q[2]~ |
| 4 | 0.616 | fpga_top\|inst_fiu_top\|inst_ccip_fabric_top\|inst_fme_top\|inst_fme_csr\|ReqAddr_q[10] | fpga_top\|inst_fiu_top\|inst_ccip_fabric_top\|inst_fme_top\|inst_fme_csr\|TxCfgR |

Data Arrival Path

|  | Total | Incr | RF | Type | Fanout | Location |
|---|---|---|---|---|---|---|
| 1 | 0.000 | 0.000 |  |  |  |  |
| 2 | ▾ 9.992 | 9.992 |  |  |  |  |
| 1 | 0.000 | 0.000 |  |  |  |  |
| 2 | 0.000 | 0.000 |  |  | 1 | PIN_AP18 |
| 3 | 0.000 | 0.000 | RR | IC | 1 | IOIBUF_X78_Y6_N47 |
| 4 | 0.566 | 0.566 | RR | CELL | 1 | IOIBUF_X78_Y6_N47 |

Data Required Path

|  | Total | Incr | RF | Type | Fanout | Location |
|---|---|---|---|---|---|---|
| 1 | 5.000 | 5.000 |  |  |  |  |
| 2 | ▾ 14.917 | 9.917 |  |  |  |  |
| 1 | 5.000 | 0.000 |  |  |  |  |
| 2 | 5.000 | 0.000 |  |  | 1 | PIN_AP18 |
| 3 | 5.000 | 0.000 | RR | IC | 1 | IOIBUF_X78_Y6_N47 |
| 4 | 5.566 | 0.566 | RR | CELL | 1 | IOIBUF_X78_Y6_N47 |

```
tcl> create_timing_summary -setup -panel_name "Summary (Setup)" -multi_corner
tcl> report_timing -to_clock { u0|dcp_iopll|dcp_iopll|clk1x } -setup -npaths 10 -detail full_path -panel_name {Setup: u0|dcp_iopll|dcp_iopll|clk1x} -multi_corner
▸ ⓘ Report Timing: Found 10 setup paths (0 violated).  Worst case slack is 0.507
← 10 0.507
tcl>
```

# Running the AFU

To run the bitstream, ensure that your host system contains an Intel FPGA PAC and that you have Acceleration Stack (including OPAE) installed and configured.

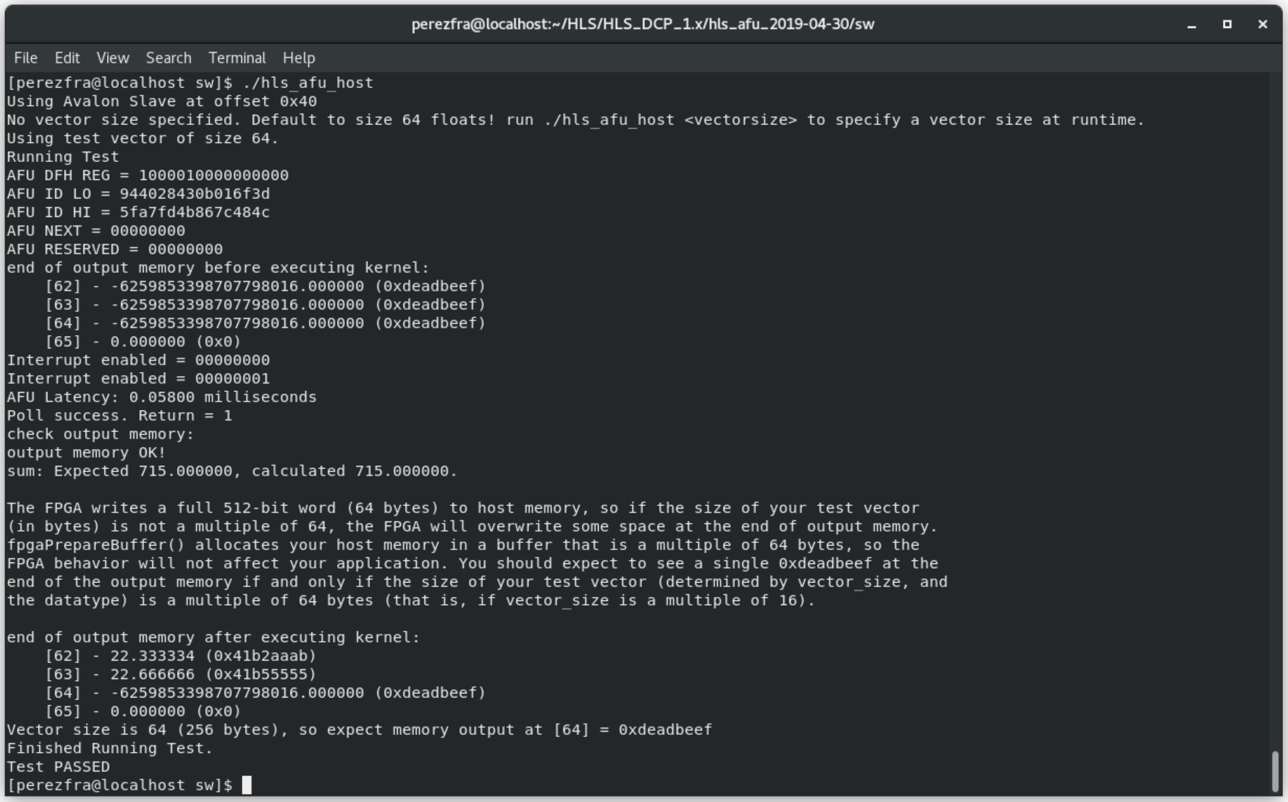
Load the AF into the FPGA

```
$ fpgaconf hls_afu.gbs
```

Navigate to the hls_afu/sw directory. Build and run the host application (do not specify USE_ASE=1)

```
$ make
$ ./hls_afu_host
```

# Running the AFU



perezfra@localhost:~/HLS/HLS_DCP_1.x/hls_afu_2019-04-30/sw
File Edit View Search Terminal Help

```
[perezfra@localhost sw]$ ./hls_afu_host
Using Avalon Slave at offset 0x40
No vector size specified. Default to size 64 floats! run ./hls_afu_host <vectorsize> to specify a vector size at runtime.
Using test vector of size 64.
Running Test
AFU DFH REG = 1000010000000000
AFU ID LO = 944028430b016f3d
AFU ID HI = 5fa7fd4b867c484c
AFU NEXT = 00000000
AFU RESERVED = 00000000
end of output memory before executing kernel:
    [62] - -6259853398707798016.000000 (0xdeadbeef)
    [63] - -6259853398707798016.000000 (0xdeadbeef)
    [64] - -6259853398707798016.000000 (0xdeadbeef)
    [65] - 0.000000 (0x0)
Interrupt enabled = 00000000
Interrupt enabled = 00000001
AFU Latency: 0.05800 milliseconds
Poll success. Return = 1
check output memory:
output memory OK!
sum: Expected 715.000000, calculated 715.000000.

The FPGA writes a full 512-bit word (64 bytes) to host memory, so if the size of your test vector
(in bytes) is not a multiple of 64, the FPGA will overwrite some space at the end of output memory.
fpgaPrepareBuffer() allocates your host memory in a buffer that is a multiple of 64 bytes, so the
FPGA behavior will not affect your application. You should expect to see a single 0xdeadbeef at the
end of the output memory if and only if the size of your test vector (determined by vector_size, and
the datatype) is a multiple of 64 bytes (that is, if vector_size is a multiple of 16).

end of output memory after executing kernel:
    [62] - 22.333334 (0x41b2aaab)
    [63] - 22.666666 (0x41b55555)
    [64] - -6259853398707798016.000000 (0xdeadbeef)
    [65] - 0.000000 (0x0)
Vector size is 64 (256 bytes), so expect memory output at [64] = 0xdeadbeef
Finished Running Test.
Test PASSED
[perezfra@localhost sw]$ 
```

# Summary

## What is the Acceleration Stack for Intel® Xeon® CPU with FPGAs

- Robust collection of software, firmware, and tools
- Makes it easy to develop and deploy Intel FPGAs in the data center
- Supports both RTL and HLS development flows
- Intel FPGA Acceleration Hub for more information

## How to develop an AFU using HLS

- Introduction to HLS
- Integration of HLS component, simulation & synthesis flows
- Developing a host application and run your accelerator

intel®

experience
what’s inside™