FCCM 2023

OFS on Intel Stratix 10

John Tio
Application Engineer

intel.

Agenda

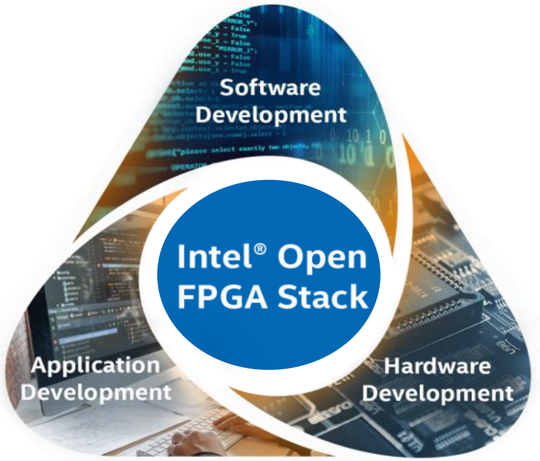
- Overview
- BMC
- FIM
- OPAE-SDK
- OFS Development
- AFU Development
- AFU Simulation Environment
- Summary

What is OFS?

OFS is a **software and hardware infrastructure** providing an efficient path to develop a custom FPGA-based platform or workload using an Intel, 3rd party, or custom board.

OFS Value Propositions:

- Scalable, source-accessible hardware and software framework delivered through Git repositories
- Reduce development time with modular and composable source code used as-is or easily customized
- Upstreamed Linux kernel drivers are being adopted by leading OS and orchestration vendors
- Maximize ROI for workload developers with standard hardware and software interfaces and by deploying across multiple OFS-based platforms
- Growing ecosystem of OFS-enabled boards, workloads, and OS distributions



OFS Deliverables

Hardware

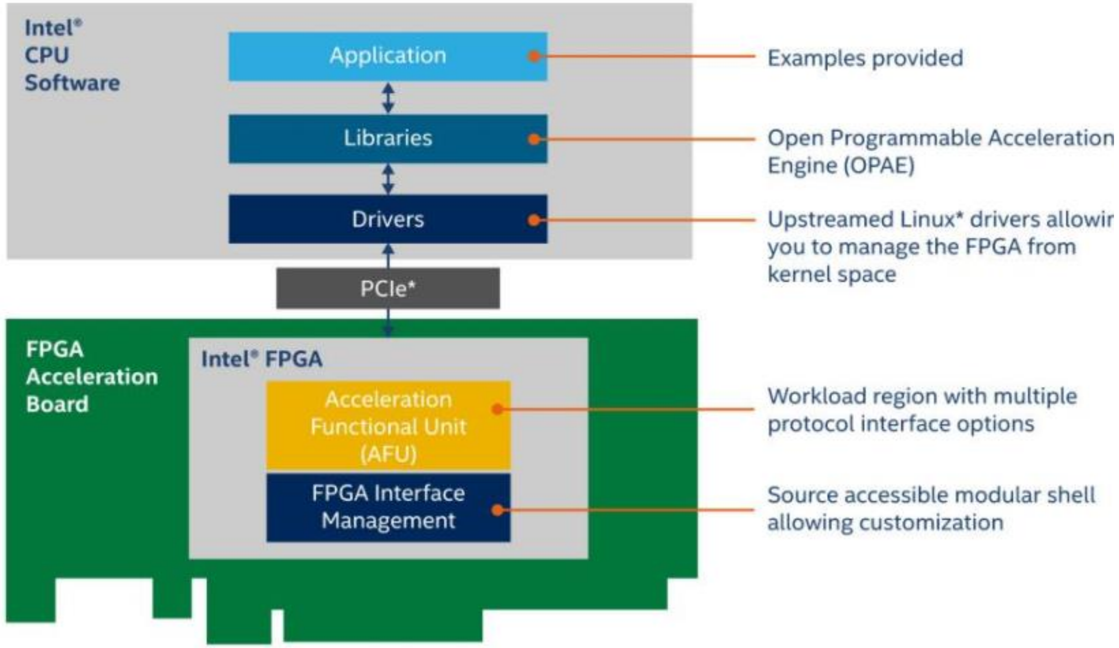
- Acceleration Functional Unit (AFU) Region for Workload Development with Sample AFUs
- FPGA Interface Manager (FIM)
- Board Management Controller (BMC)
- HLD enablement

Software

- Upstreamed, open-source kernel drivers
- OPAE libraries, tools and APIs
- Example Applications

Verification Environment

- UVM Verification environment provided through Git repositories



Examples provided

Open Programmable Acceleration Engine (OPAE)

Upstreamed Linux* drivers allowir you to manage the FPGA from kernel space

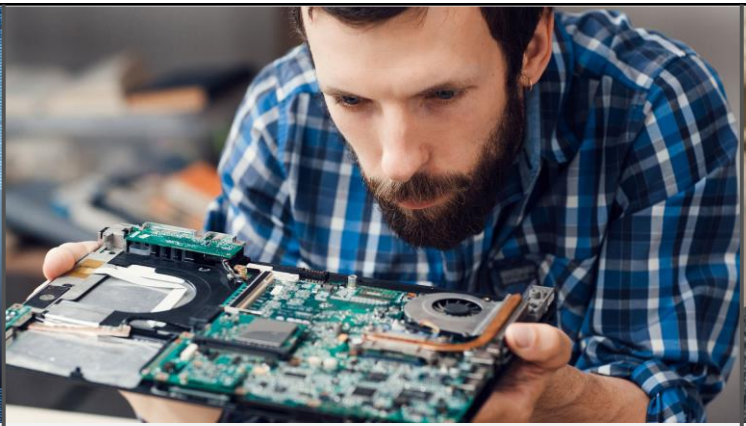
Workload region with multiple protocol interface options

Source accessible modular shell allowing customization

Developer Benefits

Software Developer

- Utilize open-sourced and upstreamed kernel drivers and user space
- Flexibility to target OS distributions of choice *or*
- Native support from leading vendor OS distributions



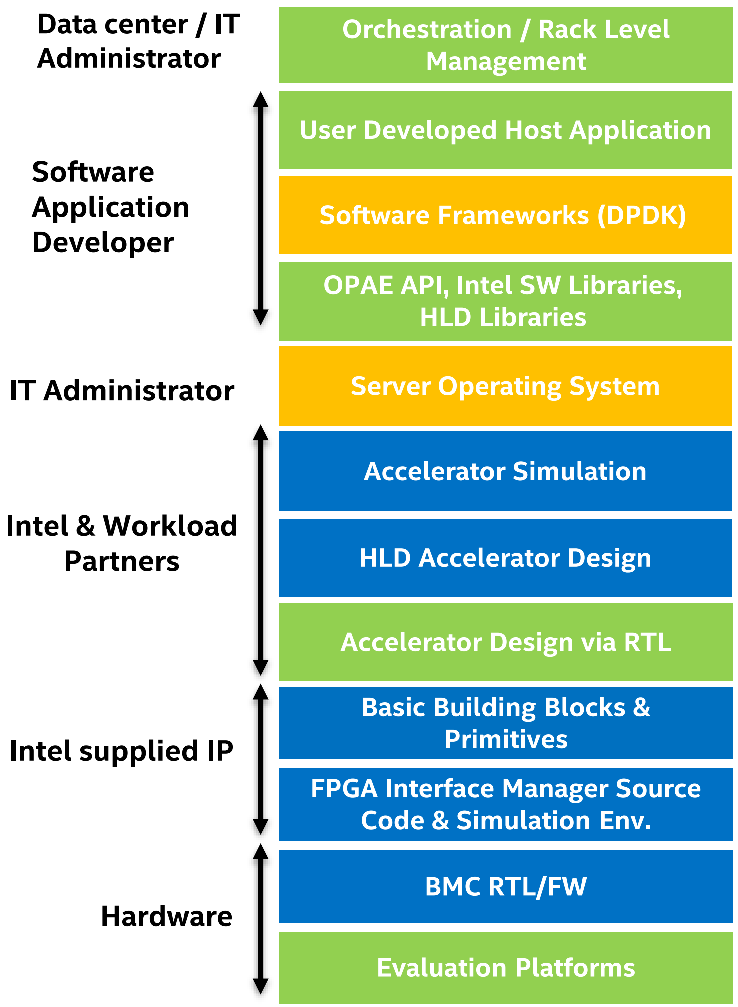
Hardware/Board Developer

- Leverage source-accessible, modular RTL source code
- Easily modify or use as-is in the provided framework to save development time

Application Developer

- Develop applications in RTL or HLD (OpenCL/oneAPI) for increased portability and ROI across OFS-based platforms
- Leverage a growing ecosystem of OFS-based platforms from Intel and third-parties

OFS Enables Scale and Deployment


Data center / IT Administrator
Orchestration / Rack Level Management
Software Application Developer
User Developed Host Application
Software Frameworks (DPDK)
OPAE API, Intel SW Libraries, HLD Libraries
IT Administrator
Server Operating System
Intel & Workload Partners
Accelerator Simulation
HLD Accelerator Design
Accelerator Design via RTL
Intel supplied IP
Basic Building Blocks & Primitives
FPGA Interface Manager Source Code & Simulation Env.
Hardware
BMC RTL/FW
Evaluation Platforms

Scalable infrastructure

- Addressing needs of board, software and application developers
- Source-accessible hardware and software code for BMC, FIM and related IP blocks

Customize your platform

- Modular and composable source code
- Take and tailor Intel supplied IP to meet platform needs

Adoption of upstreamed kernel code

- Leading OS and orchestration vendor support

Growing ecosystem

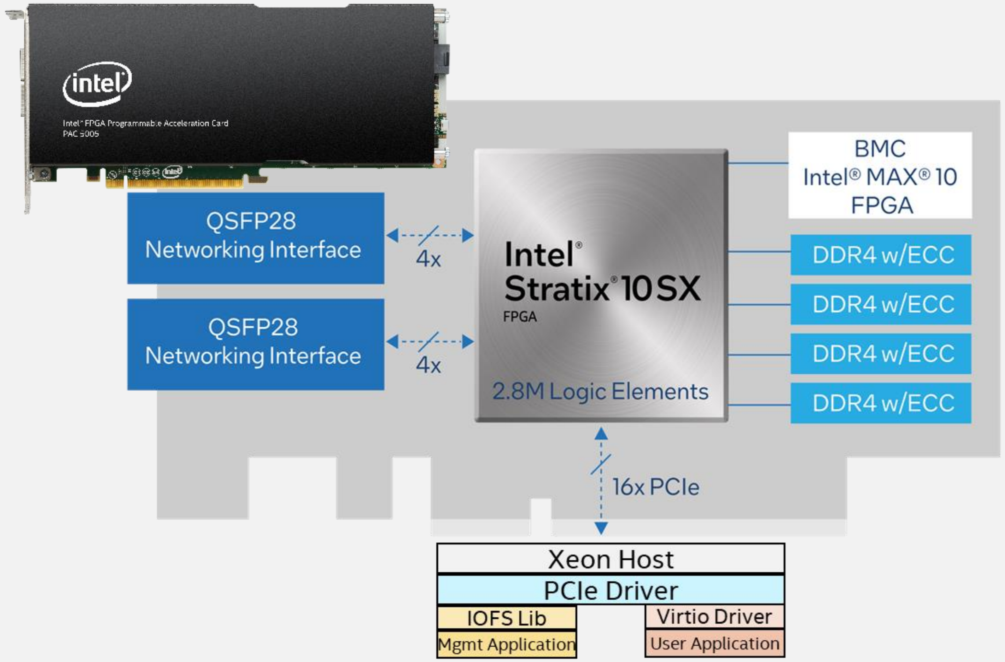
- Intel® Open FPGA Stack-based boards from Intel and third parties
- Portable RTL and HLD-based accelerator workloads
- Flexibility for OS distributions and orchestration frameworks

🟦 OFS Deliverable
🟩 Intel or Third Party
🟨 Third Party

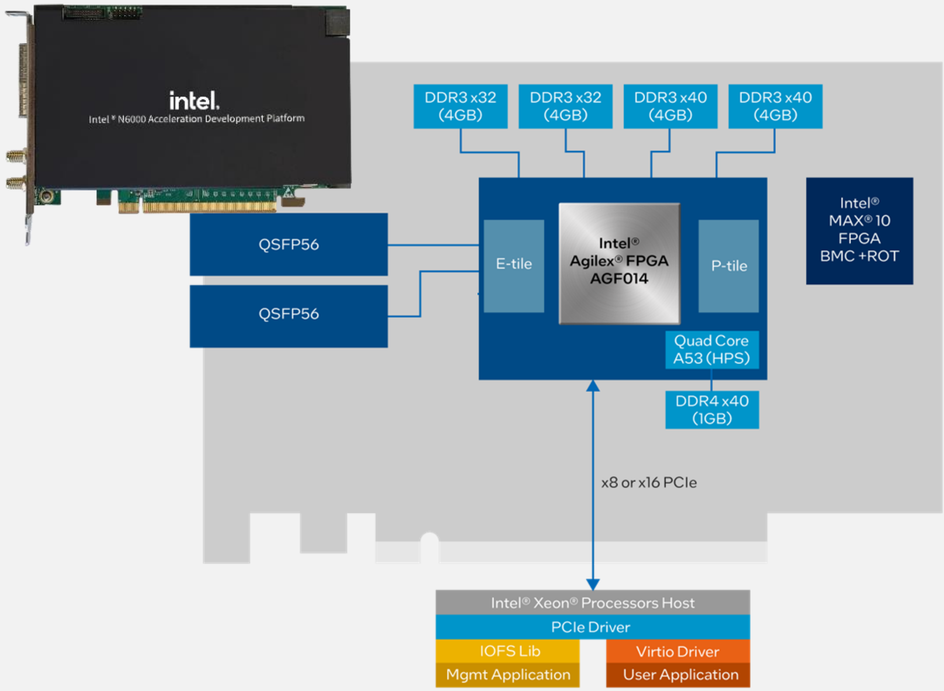
OFS Reference Platforms

Leverage hardware reference platforms as a starting point for evaluating OFS

Intel Stratix 10: D5005 Programmable Acceleration Platform (PAC)

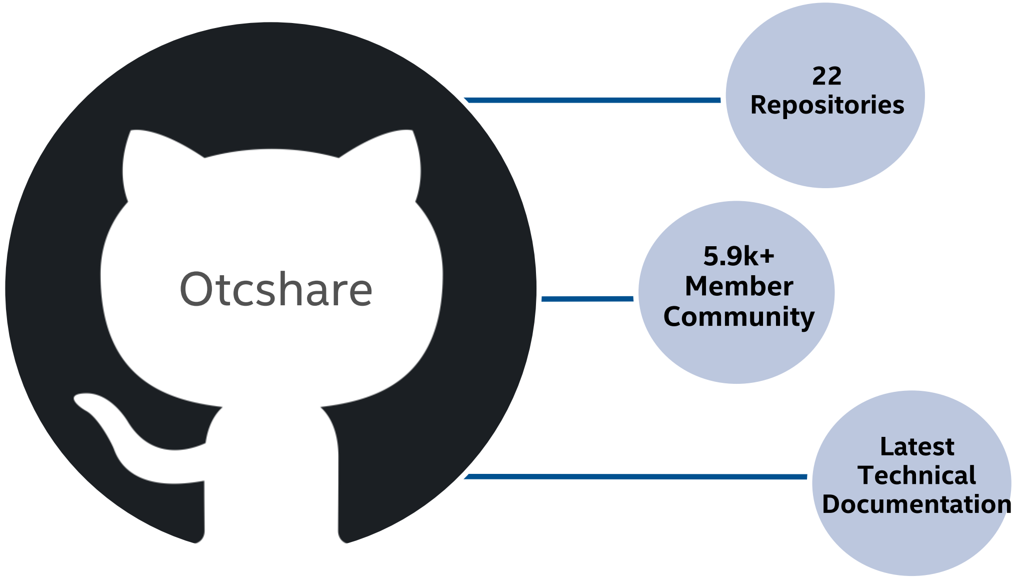


Agilex:N600x Acceleration Development Platform (ADP)

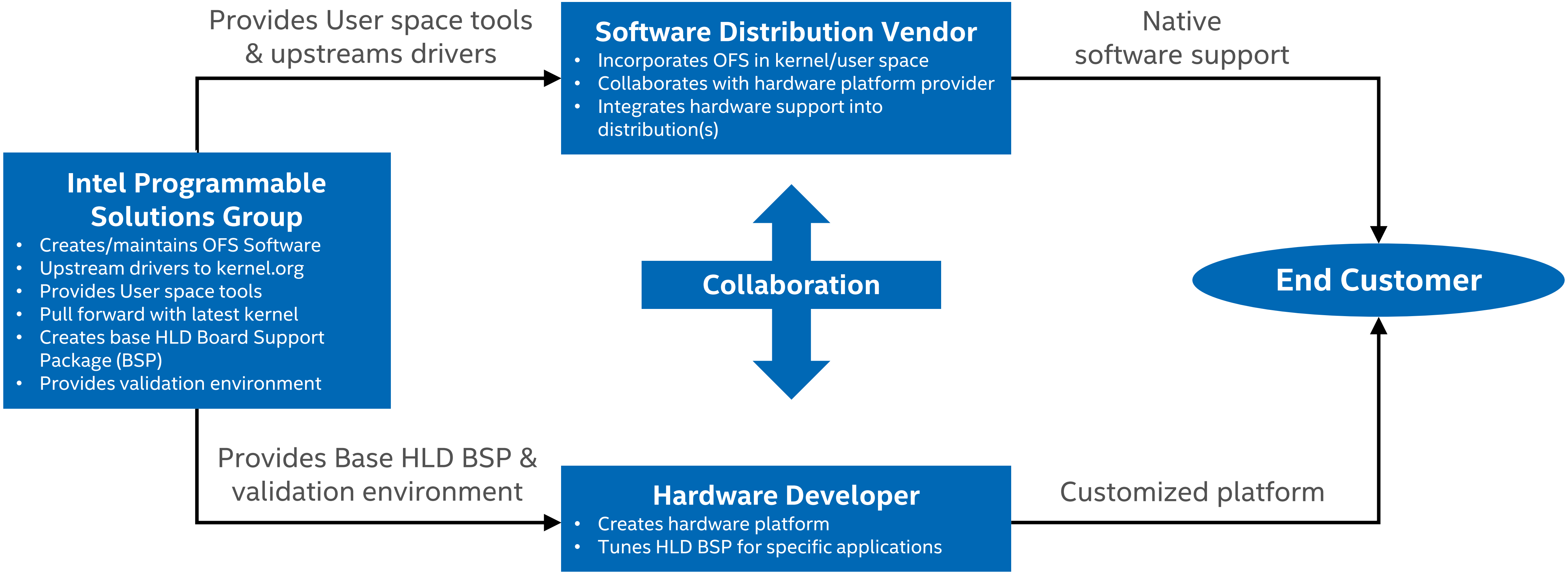


Open-Source Development Methodology

- Source-accessible hardware code for Stratix 10 and Agilex delivered to community through Git repositories
  - Hardware source code access must be requested through an Intel representative
  - Working towards a fully open-sourced model incorporating code contributions
  - Intel Stratix 10 code (https://github.com/OFS) is available today in Github
- Open-source software code (https://github.com/OPAE) is available today on Github
  - Kernel drivers are currently being upstreamed to kernel.org with patches



Software Collaboration Model



Intel Programmable Solutions Group

- Creates/maintains OFS Software
- Upstream drivers to kernel.org
- Provides User space tools
- Pull forward with latest kernel
- Creates base HLD Board Support Package (BSP)
- Provides validation environment

Provides User space tools & upstreams drivers

Software Distribution Vendor

- Incorporates OFS in kernel/user space
- Collaborates with hardware platform provider
- Integrates hardware support into distribution(s)

Native software support

Collaboration

Provides Base HLD BSP & validation environment

Hardware Developer

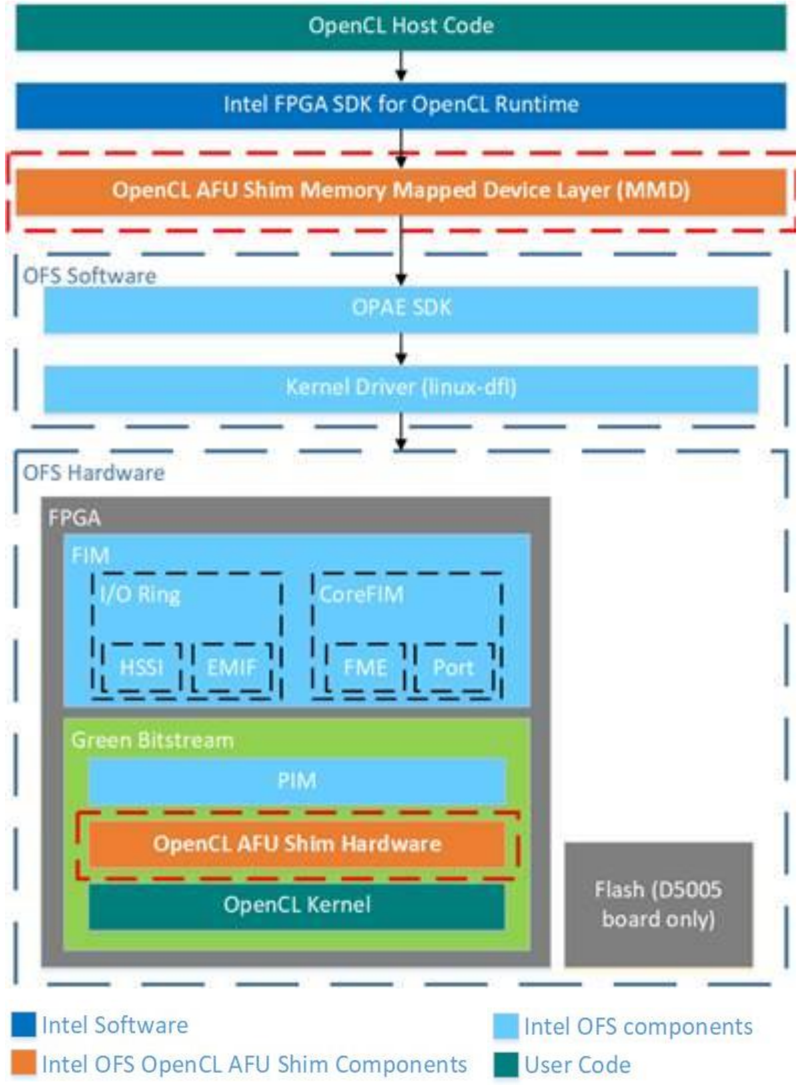
- Creates hardware platform
- Tunes HLD BSP for specific applications

Customized platform

End Customer

Develop HLD Applications using OpenCL & oneAPI

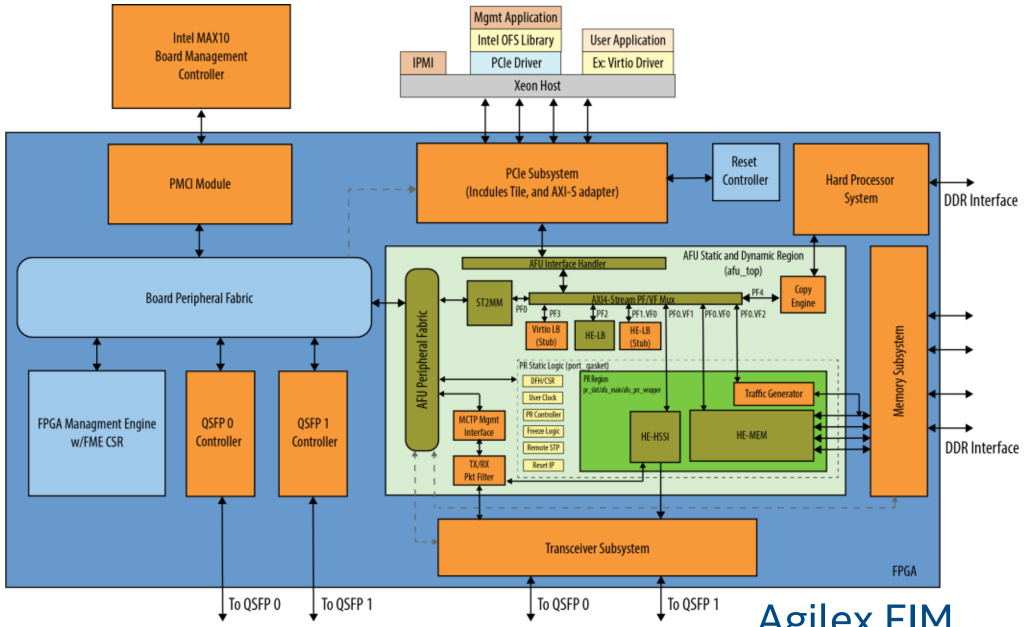
- OFS HLD Shim enables compilation/running of OpenCL kernels on OFS platforms
- Compatible with existing high-performance languages
- Leverage familiar programming languages to improve ramp-up and debug time
- OpenCL & oneAPI development on OFS enables **portability** across architectures and board vendors



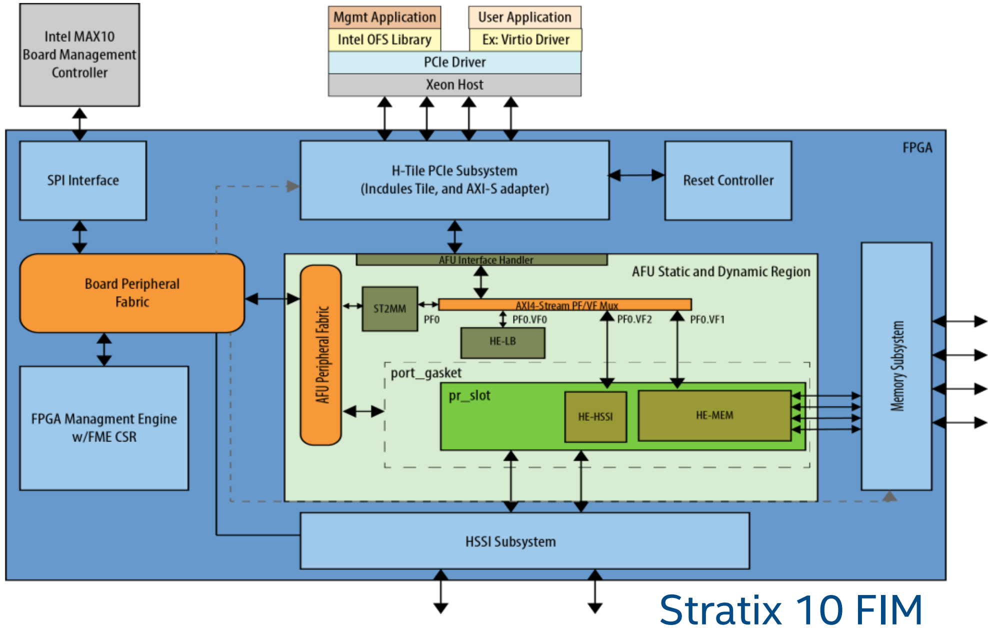
OFS Base FIMs

Leverage the base FIM as-is or modify it to reduce development time

Key Feature	Stratix 10 Reference FIM	Agilex Reference FIM
FPGA	Intel® Stratix® 10 SX FPGA	Intel® Agilex™ FPGA
Processor	Device contains HPS with Arm® but is not pinned out on card	HPS with Arm® Cortex A-53
Ethernet Configuration	1x10 GbE	2x4x25 GbE
PCIe	Gen 3x16	Gen 4x16
EMIF	Up to 4 DDR4 channels	Up to 4 DDR4 channels and 1 HPS Channel: 1 x40, 2GB
PF/VF	1 PF / 3 VFs	5 PFs / 4 VFs
Management	FPGA Management Engine (FME) with FIM management registers	
Interface	AXI4	
HLD Support	oneAPI*	
Board Management Controller	MAX 10 BMC	New, Enhanced MAX 10 BMC Architecture
UVM	UVM Support for FIM and AFU	
Software	Kernel code upstreamed to Linux	



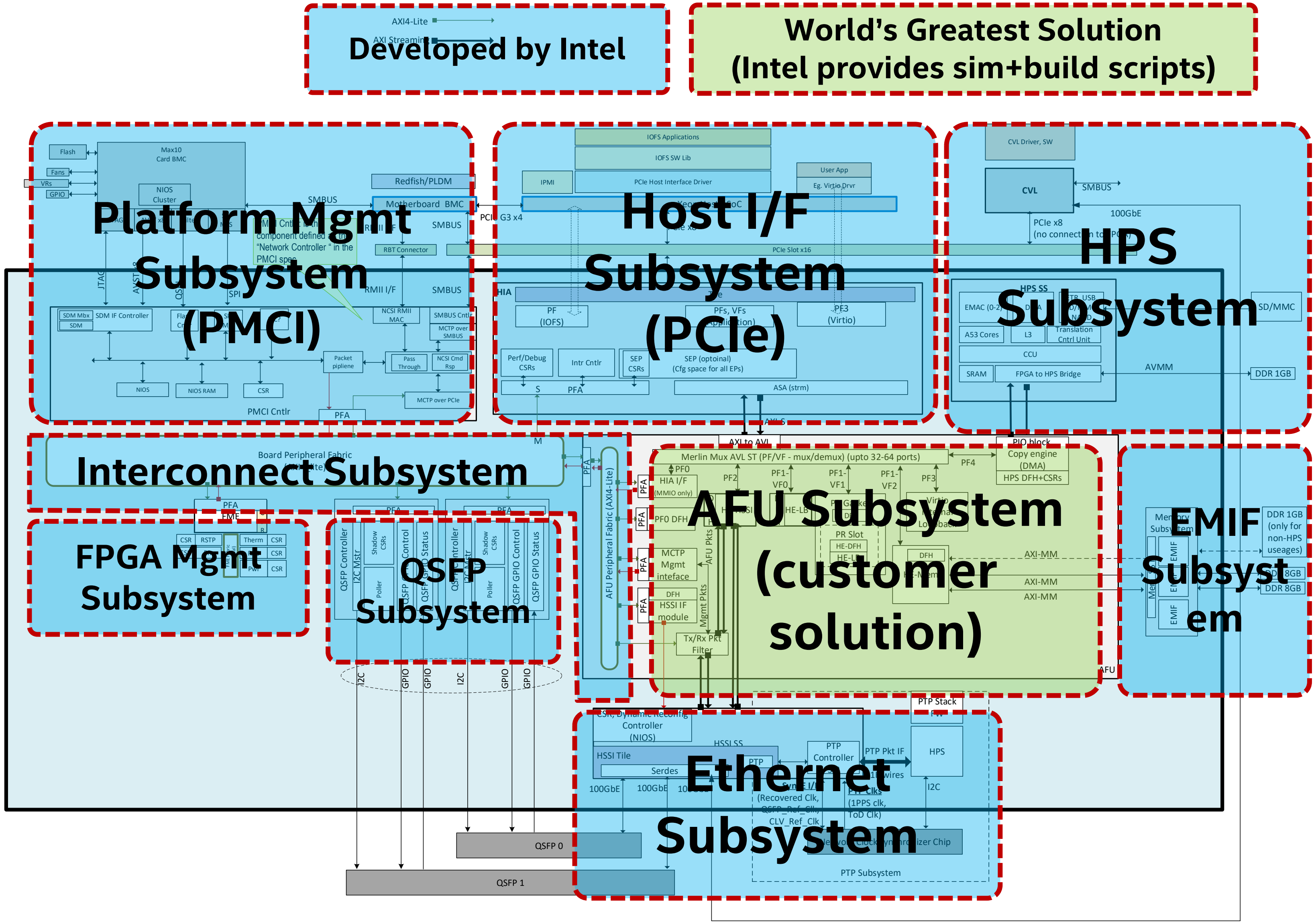
Agilex FIM



Stratix 10 FIM

AFU Development

- Focused on the algorithm
- Can be PR
- Scripts provided for SW/HW co-simulation and synthesis
- Uses PR Template generated by FIM build script
    - Developer specifies environment variable to point to required FIM qdb files



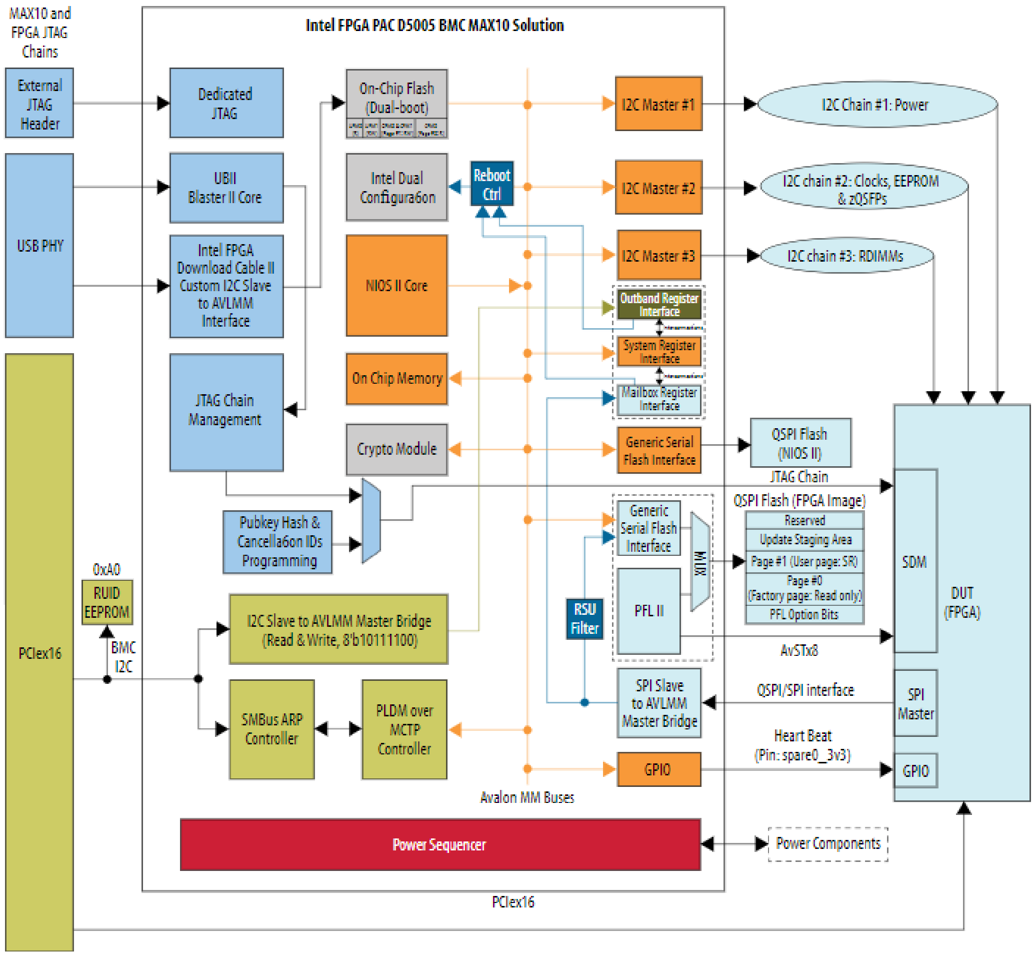
Repository Folder

Repository Folder	Description	Hardware or Software Repository
linux-dfl	This repository is a mirror of the linux.org Git site and contains the most up-to-date drivers that are being developed and upstreamed for OFS platforms.	Software
opae-sdk	Contains the ingredients to build the OFS Open Programmable Acceleration Engine (OPAE) Software Development Kit which provides APIs and userspace tools for OFS FPGA management.	Software
ofs.github.io	Contains the hardware and software collateral that surfaces on the OFS website: https://ofs.github.io	Markdown/HTML
ofs-d5005	Provides RTL, unit tests, and build scripts to create Intel Stratix 10 FIM and is leveraged as a starting point for a custom design. The reference FIM targets an Intel FPGA PAC D5005 development board.	Hardware
ofs-fim-common	Provides RTL components that are shared among all new platforms that are introduced in OFS. This folder is a subumodule in each platform repository folder.	Hardware
ofs-platform-afu-bbb	Contains the hardware and software code used to build the host interface for the FIM and provides test examples.	Hardware/Software
linux-dfl-backport	A place for finding and leveraging out-of-tree backported drivers for older OS versions .	Software
examples-afu	Provides standard AFU examples you can use as a template for starting your own workload design.	Software
opae-legacy	Supports OFS platforms built on the legacy version of OPAE software. Not used in current OFS designs	Software
opae-sim	This repository is used to build the AFU Hardware/Software Co-Simulation Environment workload developers can use to ensure their AFU can work with the OFS software stack.	Hardrware/Software

OFS-BMC

Overview

- An Intel® MAX® 10 FPGA contains the Board Management Controller (BMC) for the Intel FPGA Programmable Acceleration Card D5005.
- The BMC acts as Root of Trust (RoT) on the Intel® FPGA PAC D5005.
- The Intel® FPGA PAC D5005 BMC supports features
  - Power sequence management
  - Board monitoring through sensors.
  - Secure remote system update for Nios firmware, Intel® MAX® 10 image and FPGA Interface Manager (FIM) image updates.



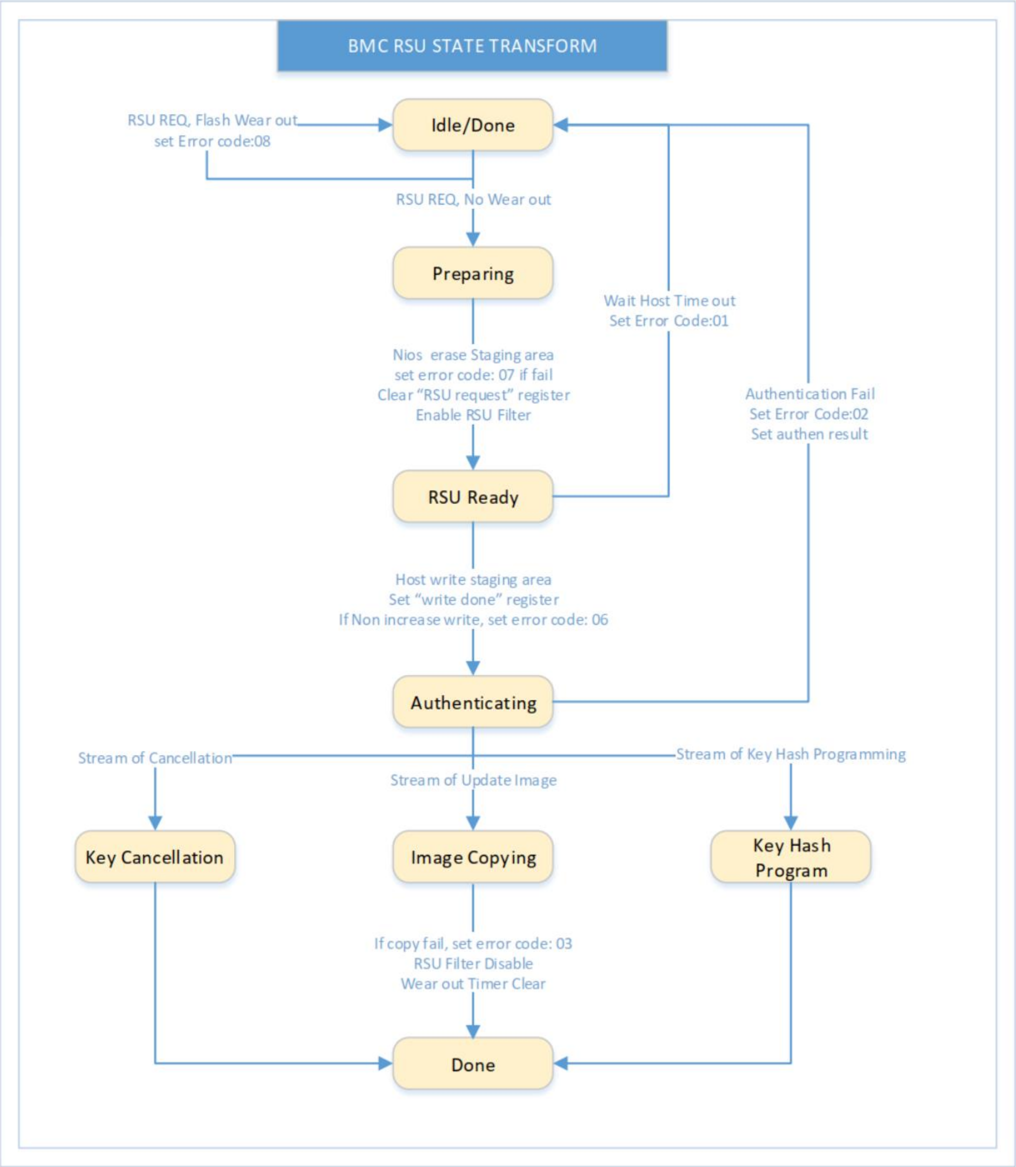
MAX10 and FPGA JTAG Chains
Intel FPGA PAC D5005 BMC MAX10 Solution
External JTAG Header
USB PHY
PCIex16
Dedicated JTAG
UBII Blaster II Core
Intel FPGA Download Cable II Custom I2C Slave to AVLMM Interface
JTAG Chain Management
On-Chip Flash (Dual-boot)
Intel Dual Configura6on
Reboot Ctrl
NIOS II Core
On Chip Memory
Crypto Module
Pubkey Hash & Cancella6on IDs Programming
0xA0
RUID EEPROM
BMC I2C
I2C Slave to AVLMM Master Bridge (Read & Write, 8'b10111100)
SMBus ARP Controller
PLDM over MCTP Controller
Avalon MM Buses
I2C Master #1
I2C Master #2
I2C Master #3
Outband Register Interface
System Register Interface
Mailbox Register Interface
Generic Serial Flash Interface
Generic Serial Flash Interface
PFL II
MUX
RSU Filter
SPI Slave to AVLMM Master Bridge
GPIO
Power Sequencer
PCIex16
I2C Chain #1: Power
I2C chain #2: Clocks, EEPROM & zQSFPs
I2C chain #3: RDIMMs
QSPI Flash (NIOS II)
JTAG Chain
QSPI Flash (FPGA Image)
Reserved
Update Staging Area
Page #1 (User page: SR)
Page #0 (Factory page: Read only)
PFL Option Bits
AvSTx8
QSPI/SPI interface
Heart Beat (Pin: spare0_3v3)
Power Components
SDM
SPI Master
GPIO
DUT (FPGA)

Root of Trust (RoT)

The Intel® MAX® 10 BMC acts as a Root of Trust (RoT) and enables the secure remote system update feature of the Intel® FPGA PAC D5005. The RoT includes features that may help prevent the following:

- Loading unauthorized bitstreams.
- Disruptive operations attempted by unprivileged software, privileged software, or the host BMC.
- Unintended execution of older designs with known bugs or vulnerabilities by enabling the BMC to revoke authorization.

The Intel® FPGA PAC D5005 BMC also enforces several other security policies relating to access through various interfaces, as well as protecting the on-board flash through write rate limitation.



BMC RSU STATE TRANSFORM

Idle/Done

RSU REQ, Flash Wear out
set Error code:08

RSU REQ, No Wear out

Preparing

Wait Host Time out
Set Error Code:01

Nios erase Staging area
set error code: 07 if fail
Clear "RSU request" register
Enable RSU Filter

Authentication Fail
Set Error Code:02
Set authen result

RSU Ready

Host write staging area
Set "write done" register
If Non increase write, set error code: 06

Authenticating

Stream of Cancellation

Stream of Update Image

Stream of Key Hash Programming

Key Cancellation

Image Copying

Key Hash Program

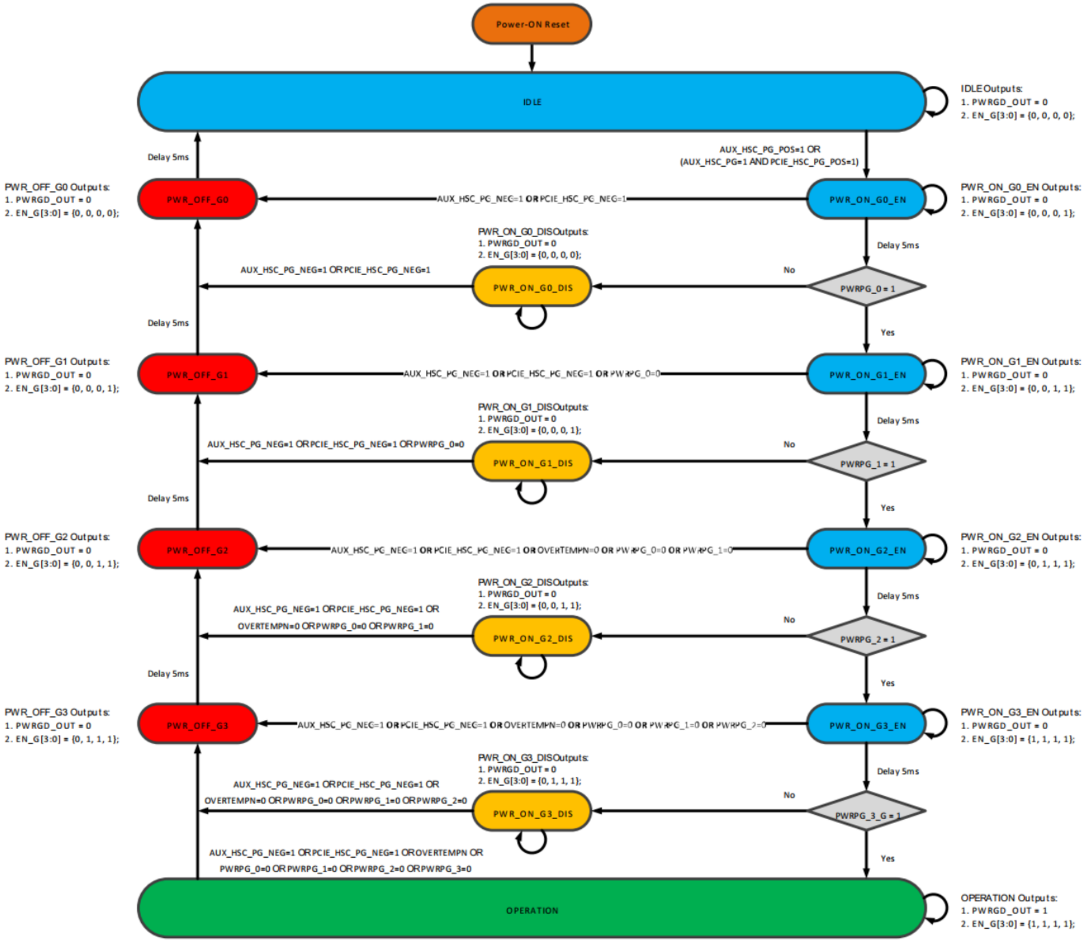
If copy fail, set error code: 03
RSU Filter Disable
Wear out Timer Clear

Done

Power Sequence Management

- The BMC Power sequencer state machine is used to manage programmable acceleration card power-on and power-off sequences, to handle different corner cases during power-on process or normal operation.
- Intel® MAX® 10 power-up flow covers the entire process including Intel® MAX® 10 boot-up, Nios® II boot-up, and power sequence management for FPGA configuration.
- The host needs to check the build versions of both Intel® MAX® 10 and FPGA, and the Nios® II status every time after a power-cycle, and then takes corresponding actions in case the Intel® FPGA PAC D5005 runs into corner cases such as a Intel® MAX® 10/FPGA factory build load failure or Nios® II boot up failure.

Intel FPGA PAC D5005 Power Sequencer

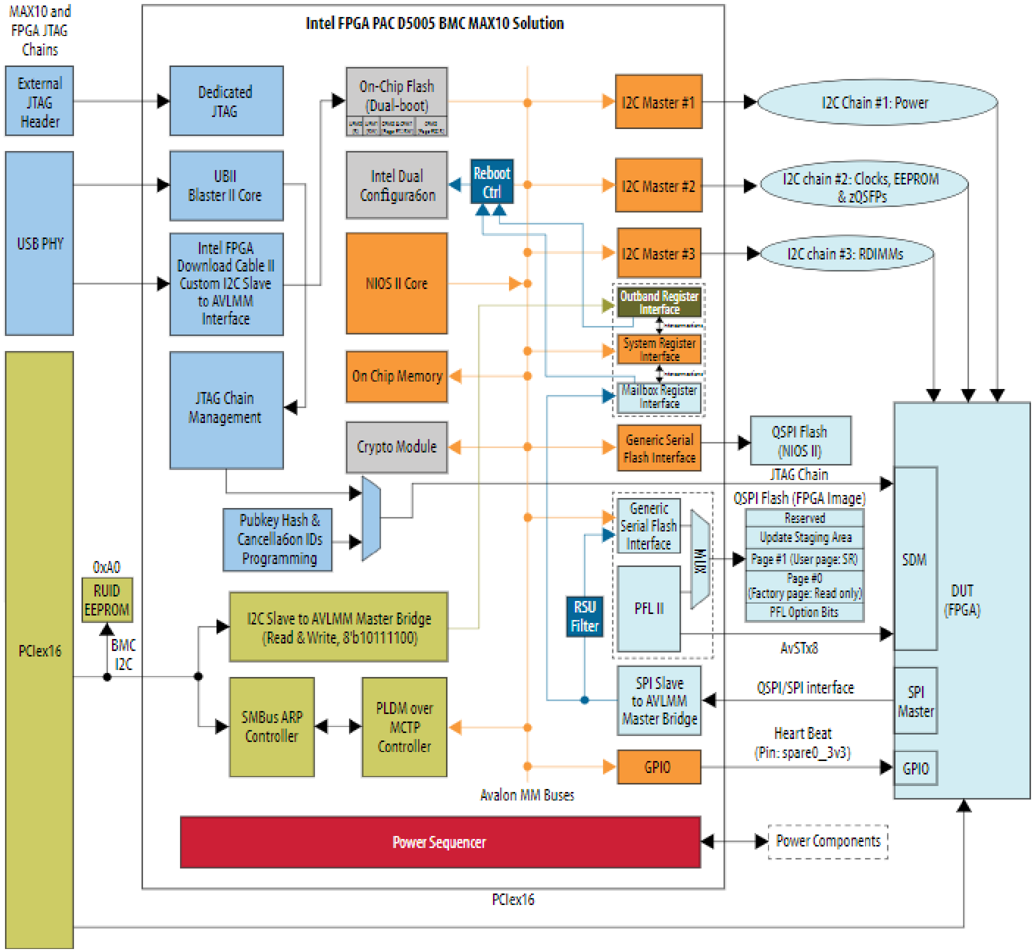


Power-ON Reset
IDLE
IDLE Outputs:
1. PWRGD_OUT = 0
2. EN_G[3:0] = {0, 0, 0, 0};
Delay 5ms
AUX_HSC_PG_POS=1 OR
(AUX_HSC_PG=1 AND PCIE_HSC_PG_POS=1)
PWR_OFF_G0 Outputs:
1. PWRGD_OUT = 0
2. EN_G[3:0] = {0, 0, 0, 0};
PWR_OFF_G0
AUX_HSC_PG_NEG=1 OR PCIE_HSC_PG_NEG=1
PWR_ON_G0_EN
PWR_ON_G0_EN Outputs:
1. PWRGD_OUT = 0
2. EN_G[3:0] = {0, 0, 0, 1};
PWR_ON_G0_DIS Outputs:
1. PWRGD_OUT = 0
2. EN_G[3:0] = {0, 0, 0, 0};
Delay 5ms
AUX_HSC_PG_NEG=1 OR PCIE_HSC_PG_NEG=1
No
PWR_ON_G0_DIS
PWRPG_0 = 1
Delay 5ms
Yes
PWR_OFF_G1 Outputs:
1. PWRGD_OUT = 0
2. EN_G[3:0] = {0, 0, 0, 1};
PWR_OFF_G1
AUX_HSC_PG_NEG=1 OR PCIE_HSC_PG_NEG=1 OR PWRPG_0=0
PWR_ON_G1_EN
PWR_ON_G1_EN Outputs:
1. PWRGD_OUT = 0
2. EN_G[3:0] = {0, 0, 1, 1};
PWR_ON_G1_DIS Outputs:
1. PWRGD_OUT = 0
2. EN_G[3:0] = {0, 0, 0, 1};
Delay 5ms
AUX_HSC_PG_NEG=1 OR PCIE_HSC_PG_NEG=1 OR PWRPG_0=0
No
PWR_ON_G1_DIS
PWRPG_1 = 1
Delay 5ms
Yes
PWR_OFF_G2 Outputs:
1. PWRGD_OUT = 0
2. EN_G[3:0] = {0, 0, 1, 1};
PWR_OFF_G2
AUX_HSC_PG_NEG=1 OR PCIE_HSC_PG_NEG=1 OR OVERTEMPN=0 OR PWRPG_0=0 OR PWRPG_1=0
PWR_ON_G2_EN
PWR_ON_G2_EN Outputs:
1. PWRGD_OUT = 0
2. EN_G[3:0] = {0, 1, 1, 1};
PWR_ON_G2_DIS Outputs:
1. PWRGD_OUT = 0
2. EN_G[3:0] = {0, 0, 1, 1};
Delay 5ms
AUX_HSC_PG_NEG=1 OR PCIE_HSC_PG_NEG=1 OR
OVERTEMPN=0 OR PWRPG_0=0 OR PWRPG_1=0
No
PWR_ON_G2_DIS
PWRPG_2 = 1
Delay 5ms
Yes
PWR_OFF_G3 Outputs:
1. PWRGD_OUT = 0
2. EN_G[3:0] = {0, 1, 1, 1};
PWR_OFF_G3
AUX_HSC_PG_NEG=1 OR PCIE_HSC_PG_NEG=1 OR OVERTEMPN=0 OR PWRPG_0=0 OR PWRPG_1=0 OR PWRPG_2=0
PWR_ON_G3_EN
PWR_ON_G3_EN Outputs:
1. PWRGD_OUT = 0
2. EN_G[3:0] = {1, 1, 1, 1};
PWR_ON_G3_DIS Outputs:
1. PWRGD_OUT = 0
2. EN_G[3:0] = {0, 1, 1, 1};
Delay 5ms
AUX_HSC_PG_NEG=1 OR PCIE_HSC_PG_NEG=1 OR
OVERTEMPN=0 OR PWRPG_0=0 OR PWRPG_1=0 OR PWRPG_2=0
No
PWR_ON_G3_DIS
PWRPG_3_G = 1
AUX_HSC_PG_NEG=1 OR PCIE_HSC_PG_NEG=1 OR OVERTEMPN OR
PWRPG_0=0 OR PWRPG_1=0 OR PWRPG_2=0 OR PWRPG_3=0
Yes
OPERATION
OPERATION Outputs:
1. PWRGD_OUT = 1
2. EN_G[3:0] = {1, 1, 1, 1};

1. OVERTEMPN= DIODE0_THERM_N & DIODE1_THERM_N & TWARN_0V89_FPGA
2. PWRPG_3_G= PWRPG_3 & pg_ddr 4_ldo

Board Monitoring through Sensor

- The Intel® MAX® 10 BMC monitors voltage, current and temperature of various components on the Intel® FPGA PAC D5005.
- Host BMC can access the telemetry data through PCIe SMBus.
- The PCIe SMBus between the host BMC and Intel® FPGA PAC D5005 Intel® MAX® 10 BMC is shared by both:
  - PLDM over MCTP SMBus endpoint
  - I2C slave to Avalon-MM interface (read-only).

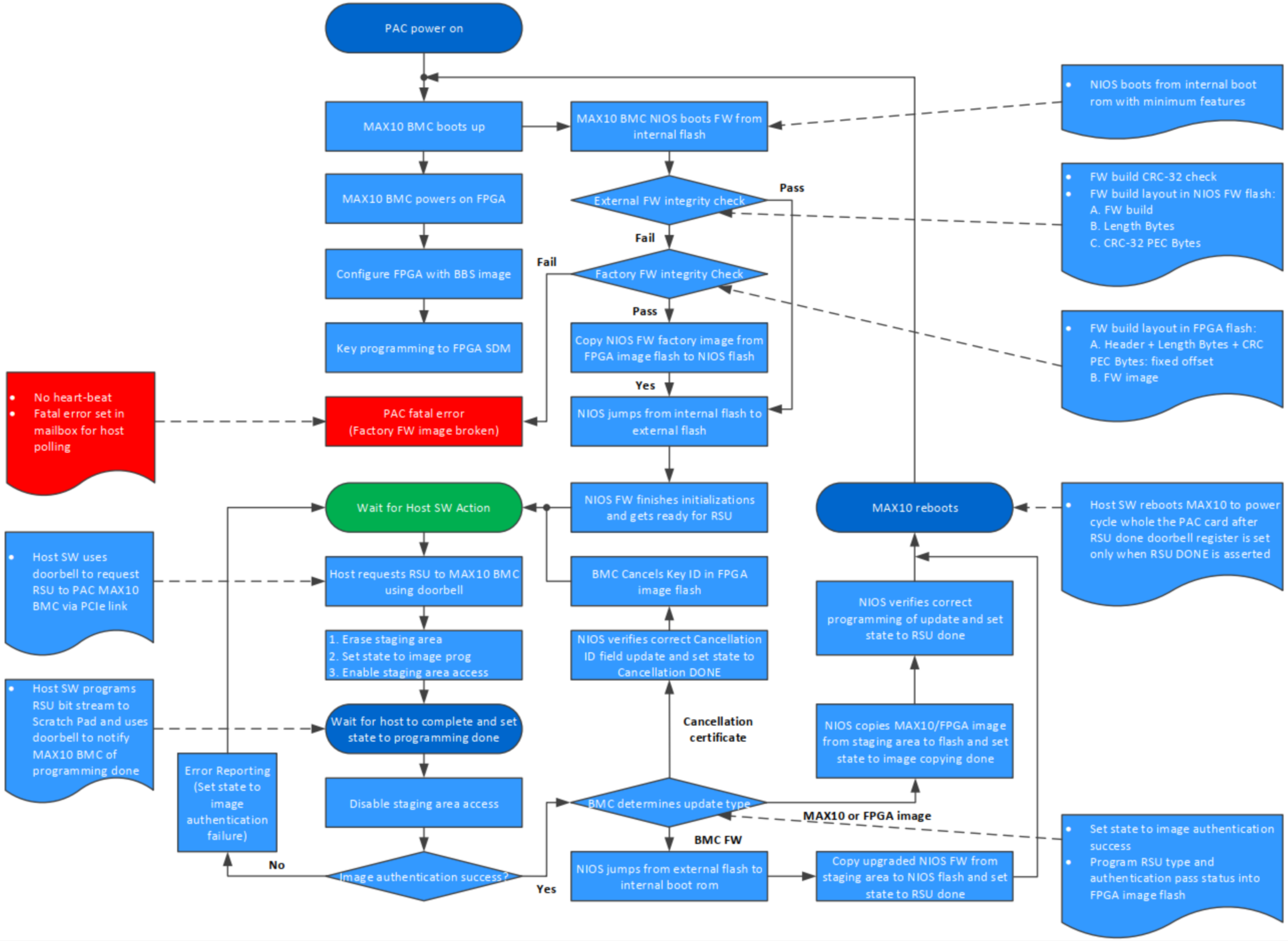


MAX10 and FPGA JTAG Chains
Intel FPGA PAC D5005 BMC MAX10 Solution
External JTAG Header
USB PHY
PCIex16
Dedicated JTAG
UBII Blaster II Core
Intel FPGA Download Cable II Custom I2C Slave to AVLMM Interface
JTAG Chain Management
Pubkey Hash & Cancella6on IDs Programming
On-Chip Flash (Dual-boot)
Intel Dual Configura6on
Reboot Ctrl
NIOS II Core
On Chip Memory
Crypto Module
I2C Master #1
I2C Master #2
I2C Master #3
Outband Register Interface
System Register Interface
Mailbox Register Interface
Generic Serial Flash Interface
Generic Serial Flash Interface
MUX
PFL II
RSU Filter
SPI Slave to AVLMM Master Bridge
GPIO
I2C Chain #1: Power
I2C chain #2: Clocks, EEPROM & zQSFPs
I2C chain #3: RDIMMs
QSPI Flash (NIOS II)
JTAG Chain
QSPI Flash (FPGA Image)
Reserved
Update Staging Area
Page #1 (User page: SR)
Page #0 (Factory page: Read only)
PFL Option Bits
AvSTx8
QSPI/SPI interface
Heart Beat (Pin: spare0_3v3)
SDM
DUT (FPGA)
SPI Master
GPIO
0xA0
RUID EEPROM
BMC I2C
I2C Slave to AVLMM Master Bridge (Read & Write, 8'b10111100)
SMBus ARP Controller
PLDM over MCTP Controller
Avalon MM Buses
Power Sequencer
Power Components
PCIex16

Secure Remote System Update

- The Intel® FPGA PAC D5005 provides a mechanism to securely update Nios® II firmware, Intel® MAX® 10 image, or Intel® Stratix® 10 FPGA image over PCIe interface from the host called Remote System Update (RSU). RSU can be used for the following three updates:
  - Intel MAX 10 image update
  - Nios® II firmware image update
  - FPGA Interface Manager (FIM) image update
- The Nios® II firmware is in charge of authenticating the image during the update process. The updates are pushed over the PCIe interface to the Intel® Stratix® 10 SX FPGA, which in turn writes to the Intel® Stratix® 10 FPGA SPI master and finally to the Intel® MAX® 10 FPGA SPI slave. A temporary flash area called staging area stores any type of authentication bitstream through SPI interface.
- BMC RoT contains the cryptographic module which implements SHA2-256 bit Hash verification function to authenticate the keys and ECDSA-256P-256 signature verification to authenticate your AFU.
- Nios® II Firmware uses the cryptographic module to authenticate the user signed image in the staging area, if authentication passes, Nios® II Firmware copies the user image to user flash area. If the authentication fails Nios® II Firmware reports an error.

Remote System Upgrade Flow



PAC power on

MAX10 BMC boots up

MAX10 BMC powers on FPGA

Configure FPGA with BBS image

Key programming to FPGA SDM

PAC fatal error (Factory FW image broken)

- No heart-beat
- Fatal error set in mailbox for host polling

MAX10 BMC NIOS boots FW from internal flash

- NIOS boots from internal boot rom with minimum features

External FW integrity check

Pass

Fail

- FW build CRC-32 check
- FW build layout in NIOS FW flash:
  A. FW build
  B. Length Bytes
  C. CRC-32 PEC Bytes

Factory FW integrity Check

Fail

Pass

- FW build layout in FPGA flash:
  A. Header + Length Bytes + CRC PEC Bytes: fixed offset
  B. FW image

Copy NIOS FW factory image from FPGA image flash to NIOS flash

Yes

NIOS jumps from internal flash to external flash

NIOS FW finishes initializations and gets ready for RSU

Wait for Host SW Action

Host requests RSU to MAX10 BMC using doorbell

- Host SW uses doorbell to request RSU to PAC MAX10 BMC via PCIe link

1. Erase staging area
2. Set state to image prog
3. Enable staging area access

Wait for host to complete and set state to programming done

- Host SW programs RSU bit stream to Scratch Pad and uses doorbell to notify MAX10 BMC of programming done

Disable staging area access

Image authentication success?

No

Yes

Error Reporting (Set state to image authentication failure)

BMC determines update type

Cancellation certificate

BMC FW

MAX10 or FPGA image

- Set state to image authentication success
- Program RSU type and authentication pass status into FPGA image flash

NIOS verifies correct Cancellation ID field update and set state to Cancellation DONE

BMC Cancels Key ID in FPGA image flash

NIOS jumps from external flash to internal boot rom

Copy upgraded NIOS FW from staging area to NIOS flash and set state to RSU done

NIOS copies MAX10/FPGA image from staging area to flash and set state to image copying done

NIOS verifies correct programming of update and set state to RSU done

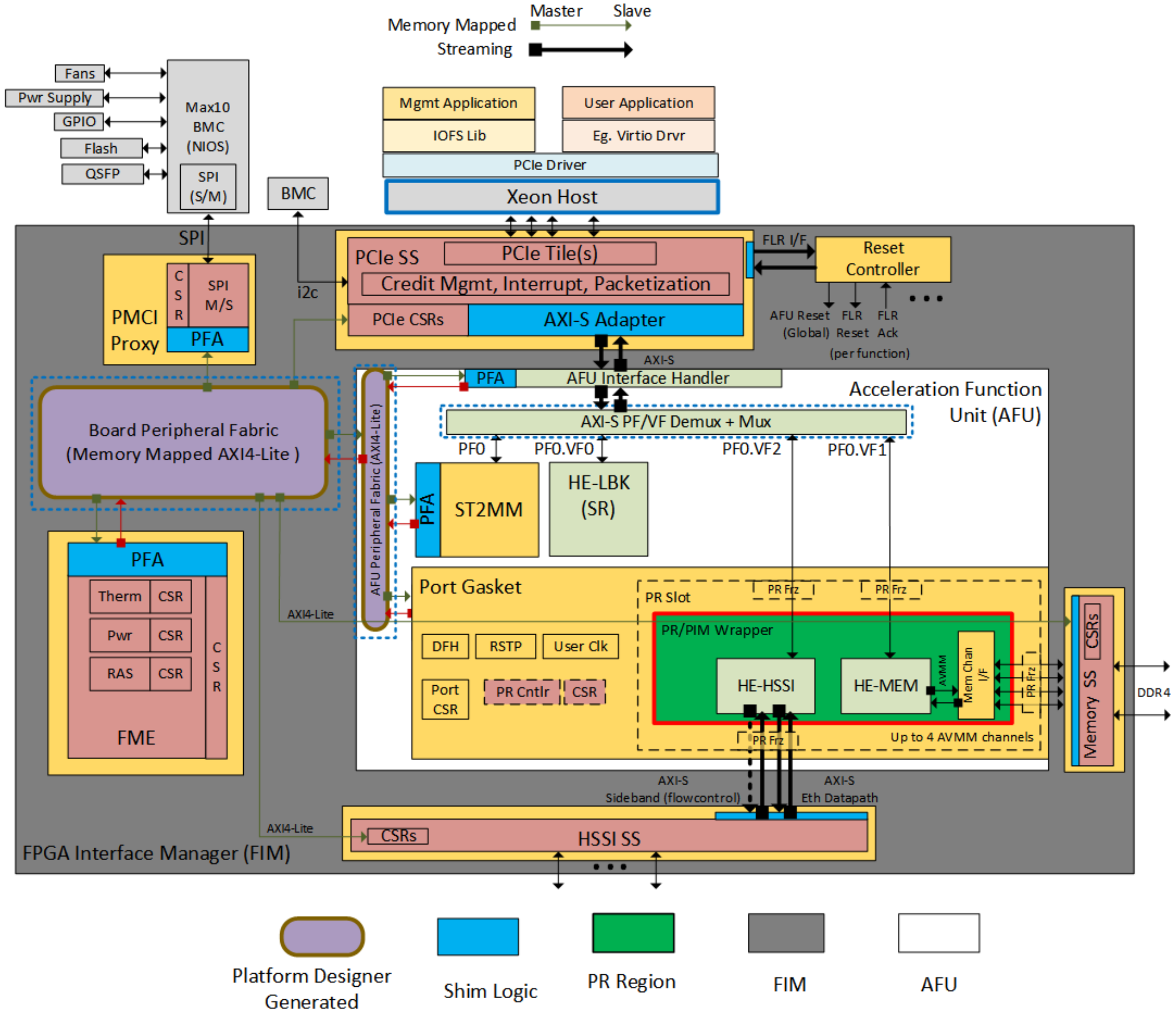
MAX10 reboots

- Host SW reboots MAX10 to power cycle whole the PAC card after RSU done doorbell register is set only when RSU DONE is asserted

OFS-FIM

OFS Features

- FPGA Interface Manager (FIM)
  - PCIe Subsystem
  - HSSI Subsystem
  - Memory Subsystem
  - Reset Controller
  - FPGA Management Engine
  - AFU Peripheral Fabric for AFU accesses to other interface peripherals
  - Board Peripheral Fabric for master to slave CSR accesses from Host or AFU
  - SPI Interface to BMC controller
- Accelerator Functional Unit (AFU)
  - Custom workloads and contains both static and partial reconfiguration regions.



OFS Features table

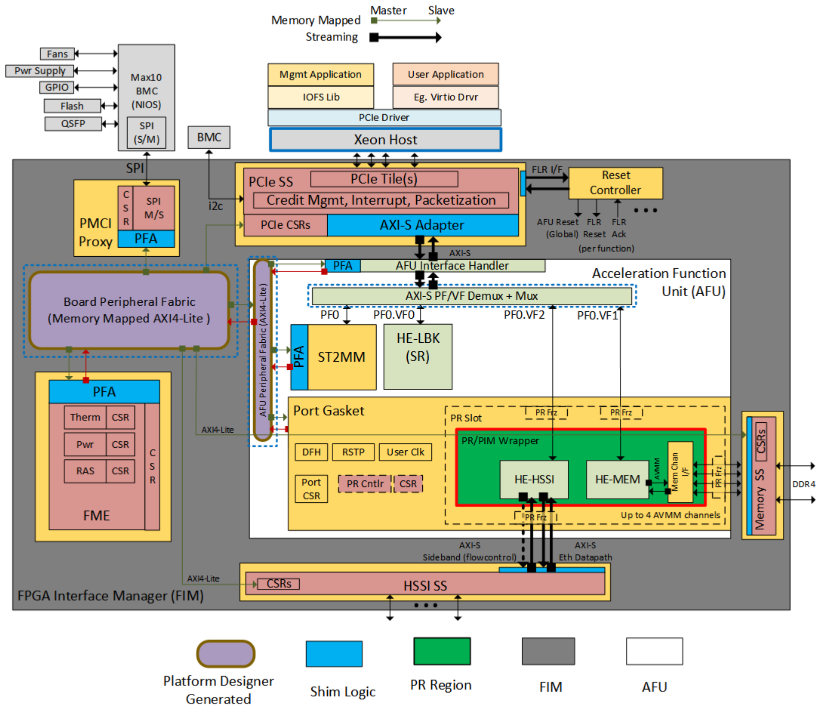
Key Feature	**Description**
PCIe	H-tile PCIe Gen3x16 Interface
Memory	Two Avalon Memory Mapped channels provided as default with capability to compile design with four channels support.
HSSI	1 Arm* AMBA* 4 AXI4-Stream channel of 10G Ethernet, using the low latency Ethernet 10G MAC Intel FPGA IP interfacing to an E-tile PHY.
Manageability	SPI interface to Board Management Controller targeting Intel FPGA PAC D5005
CoreFIM	Flexible configuration support using Arm* AMBA* 4 AXI4-Stream Physical Function/Virtual Function (PF/VF) Demux/Mux and AFU Peripheral Fabric (APF) and Board Peripheral (BPF) Fabric Interconnects.
Physical Function/Virtual	1 PF/3VF configuration is provided as an example but the architecture now supports full virtualization with the ability to expand to whatever the PCIe tile supports.
Partial Reconfiguration	1 Partial Reconfiguration region supported in hardware and software
Sample test PR AFUs	Host exerciser modules provided to exercise interfaces. These modules are provided in both the flat and PR AFU examples.
OneAPI	Yes
Software Support	OFS software stack with support for full virtualization.

FPGA MANAGEMENT ENGINE (FME)

The FIM contains only one FME, regardless of the number of host interfaces to the FIM. The FME provides management features for the platform and controls reset and loading of the AFU into the partial reconfiguration region of the FPGA.

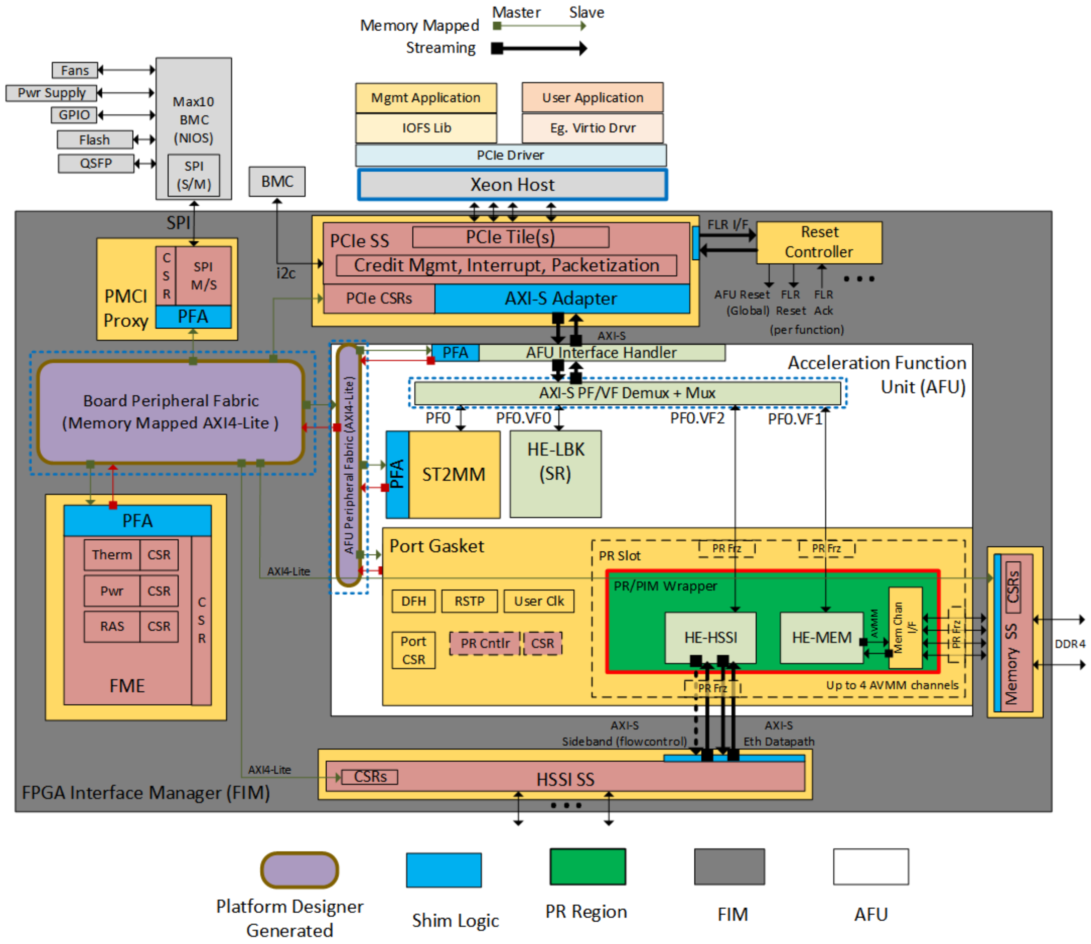
Each FME feature exposes its capability to host software drivers through a device feature header (DFH) register found at the beginning of its control status register (CSR) space. The FME CSR maps to physical function 0 (PF0) Base address register 0 (BAR0) so that software can access it through a single PCIe link. For more information about DFHs, refer to the [Device Feature Header (DFH) structure].

- **STREAMING DATAPATH**
  - The FIM implements an AXI4-Stream bus protocol for data transfer in the FIM. AXI4-Stream channels send data packets to and from the host channel IP without data abstraction. Memory-mapped I/O (MMIO) CSR accesses are routed to the ST2MM module which converts the AXI4-Stream to an AXI4 memory mapped protocol.
- **VIRTUALIZATION**
  - This design supports virtualization by making use of the virtualization functionality in the PCIe Hard IP and mapping packets to the appropriate physical or virtual function through a PF/VF multiplexer. This reference FIM supports 1 PF and 3 VFs as an example; however, you may extend your configuration to whatever the PCIe Hard IP can support or what your application requires.



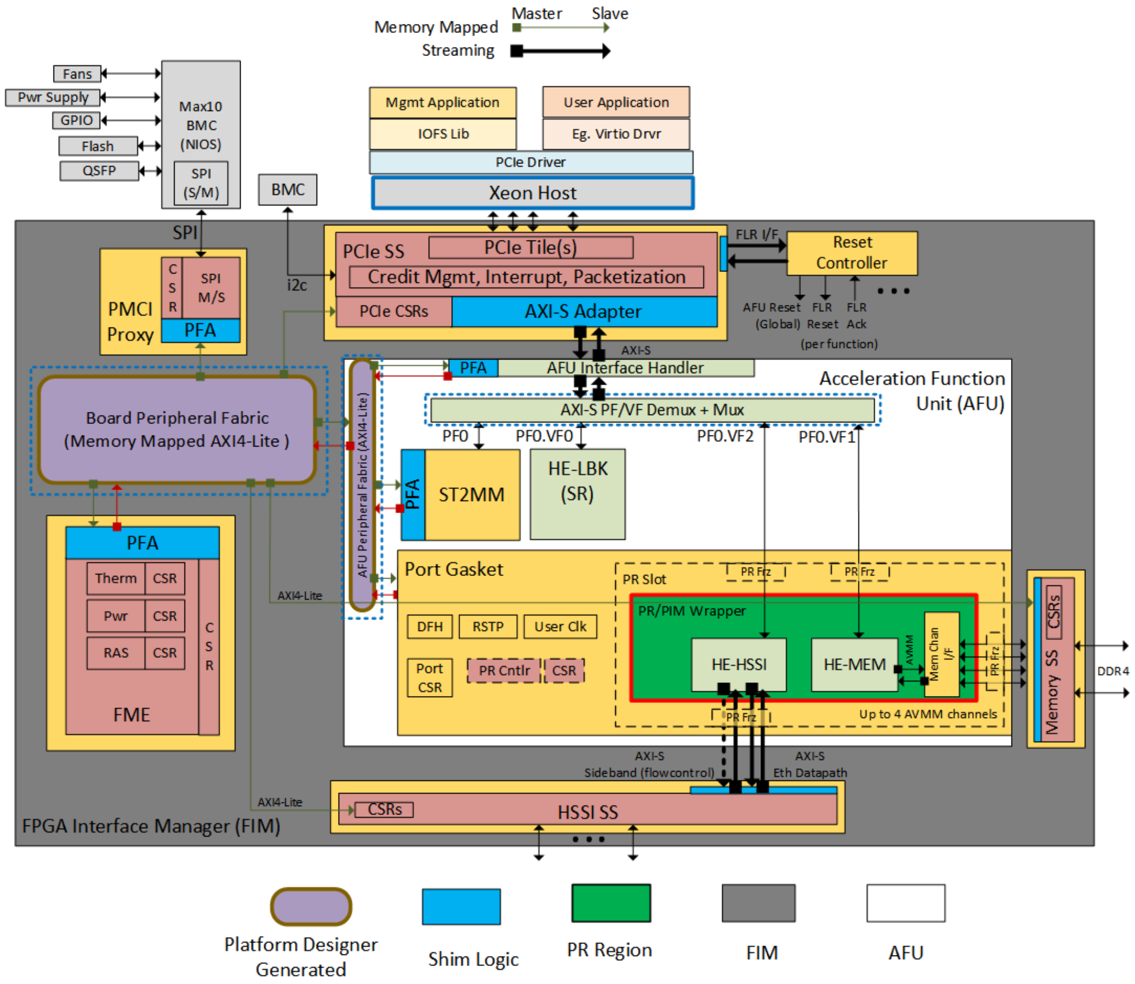
Accelerator Functional Unit (AFU)

- An AFU is an acceleration workload that interfaces to the FIM.
- The AFU boundary in this design comprises both static and partial reconfiguration (PR) regions. You can compile your design in one of the following ways:
  - Your entire AFU resides in a partial reconfiguration region of the FPGA
  - The AFU is part of the static region and is compiled a flat design



Platform Interface Manager

- The PIM provides a way to abstract the AXI4-Stream interface to the AFU by providing a library of shims that convert the host channel native packet into other protocols such as
  - CCI-P
  - AXI4 memory-mapped
  - Avalon® streaming (Avalon-ST)
  - Avalon® memory-mapped (Avalon-MM).
- If you expose the raw AXI4-Stream interface of the FIM, workload developers also have the option to convert to a desired protocol using the PIM resources as well.



FIM Simulation

OFS provides a UVM environment for the FIM and a framework for new feature verification. UVM provides a modular, reusable, and scalable testbench structure by providing an API framework that can be deployed across multiple projects. The FIM testbench is UVM compliant and integrates third-party verification IPs from Synopsys that require license to use. Verification components include:

- FIM monitor to detect correct design behavior
- FIM assertions for signal level integrity testing
- Arm AMBA AXI4 scoreboards to check data integrity
- FIM coverage to collect functional data

OFS Datapath

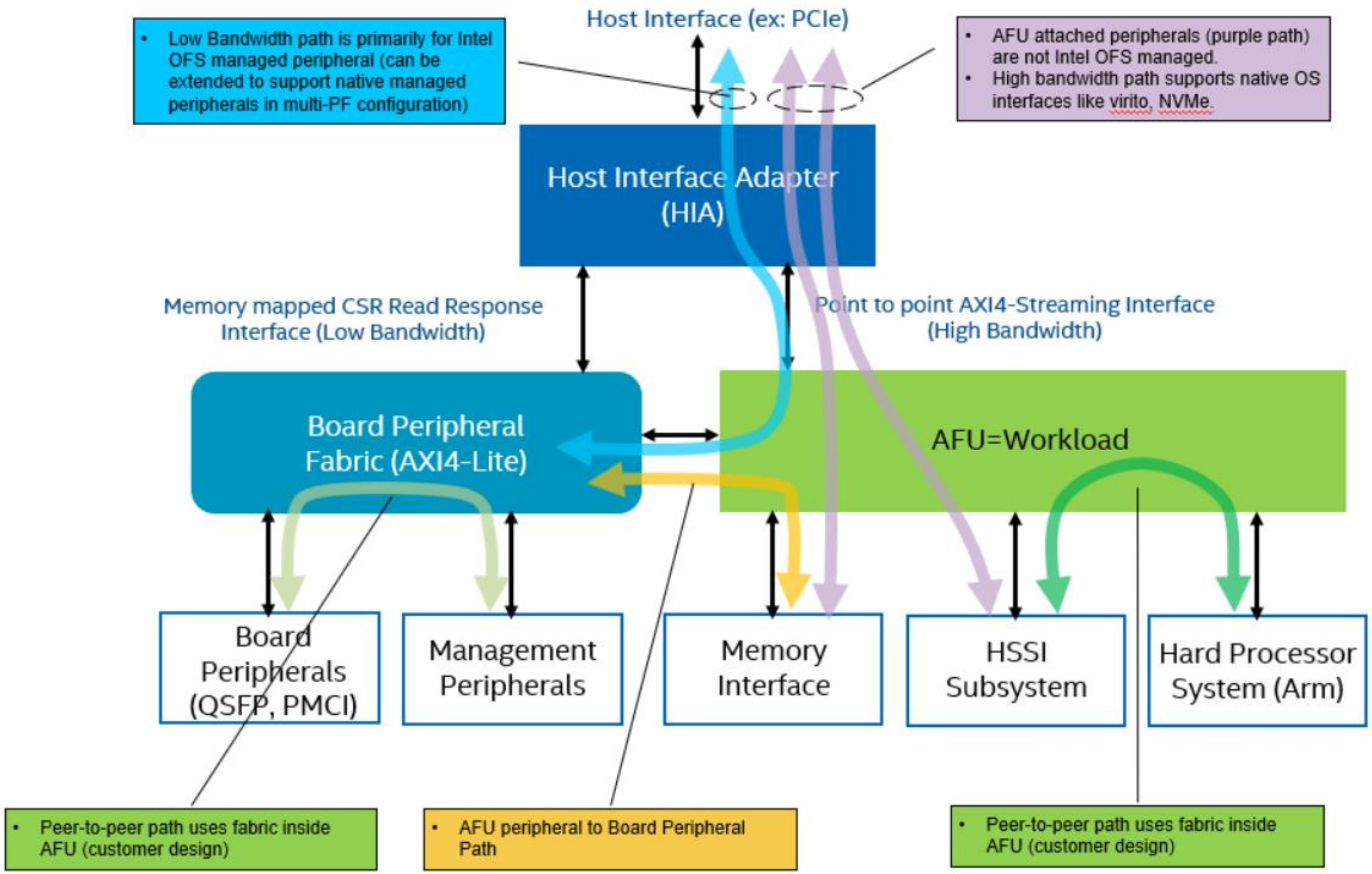
OFS provides distinct datapaths that simplifies the design and integration process for add or for removing interface modules:

- **High Bandwidth datapath** for AFU-attached high performance peripherals (HSSI, Memory, HPS, workload).
- **Low Bandwidth datapath** for OFS management and slow peripheral components (JTAG, I2C, SMBus).
- **AFU Peripheral Fabric (APF) to Board Peripheral Fabric (BPF) path** to communicate with interface control and status registers (CSRs) and board components.
- **Peer-to-peer datapath** between AFU components.
- **Peer-to-peer datapath** between BPF components.

Depending on your design goals, you can present peripherals to software as:

- OFS managed peripherals with a device feature header that is part of a device feature list.
- Native driver managed peripherals that are exposed through an independent physical function or virtual function.

OFS Datapath



Host Interface (ex: PCIe)

- Low Bandwidth path is primarily for Intel OFS managed peripheral (can be extended to support native managed peripherals in multi-PF configuration)

- AFU attached peripherals (purple path) are not Intel OFS managed.
- High bandwidth path supports native OS interfaces like virito, NVMe.

Host Interface Adapter (HIA)

Memory mapped CSR Read Response Interface (Low Bandwidth)

Point to point AXI4-Streaming Interface (High Bandwidth)

Board Peripheral Fabric (AXI4-Lite)

AFU=Workload

Board Peripherals (QSFP, PMCI)

Management Peripherals

Memory Interface

HSSI Subsystem

Hard Processor System (Arm)

- Peer-to-peer path uses fabric inside AFU (customer design)

- AFU peripheral to Board Peripheral Path

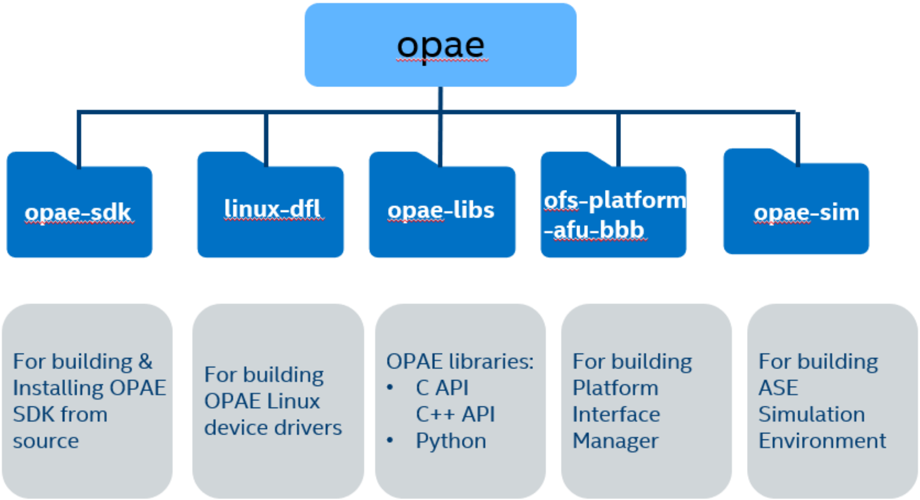
- Peer-to-peer path uses fabric inside AFU (customer design)

OPAE-SDK

OFS Software Stack Components

The OFS software stack (OPAE) is the lowest level of software that manages the FPGA card. Higher level software (OpenCL/oneAPI) can be built on top of OFS.

- Components—open source on Git
  - libOPAE (user-space library)
  - OPAE FPGA applications and tools (e.g. fpgainfo, fpgaconf, fpgasupdate, etc.)
  - AFU Simulation environment (ASE)
  - DFL Linux Kernel Driver

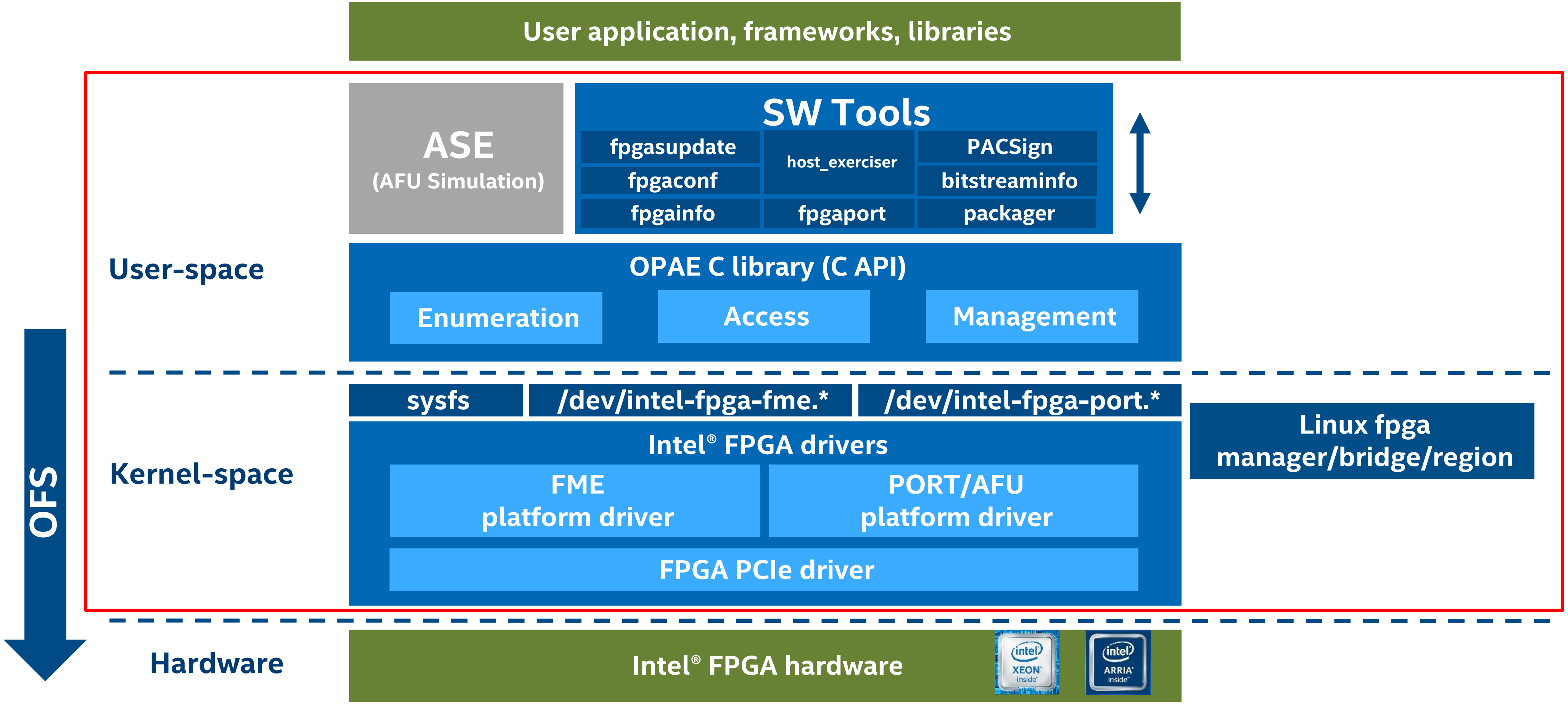


opae
opae-sdk
linux-dfl
opae-libs
ofs-platform-afu-bbb
opae-sim
For building & Installing OPAE SDK from source
For building OPAE Linux device drivers
OPAE libraries:
• C API
C++ API
• Python
For building Platform Interface Manager
For building ASE Simulation Environment

OPAE code is completely open source at: http://github.com/OFS/opae-sdk

Open Programmable Acceleration Engine (OPAE)

OPAE is a collection of APIs, libraries, and tools to provide FPGA abstraction



Device Feature List (DFL)

OFS uses a DFL structure to describe FPGA contents to software

- The DFL is a Linked list of Device Feature Headers (DFH)
- DFL framework abstracts low layer hardware details and provides a unified interface to userspace
- DFL Linux Driver “walks” the DFL attaching/binding software to enumerate each FPGA feature
- DFL driver source code available at: github.com/OPAE/linux-dfl
- In the process of being upstreamed to kernel.org

OPAE Tools Sample

OPAE Tool	Description
fpgaconf	Configures the FPGA with the accelerator function (AF). It also checks the AF for compatibility with the targeted FPGA and the FPGA Interface Manager (FIM).
fpgasupdate	Implements a secure firmware for FPGA or BMC
fpgainfo	Displays FPGA information derived from sysfs files
rsu	Allows the user to preform a remote system update (RSU) operation on a PAC device given its PCIe address
PACSign	Inserts authentication markers into bitstreams targeted for the Intel FPGA PAC D5005
bitstreaminfo	Displays authentication information contained with each provided FPGA bitstream binary file
hssi	Provides a means of interacting with the 10G HE-HSSI
opae.io	Provides user space access to PCIe devices via the vfio-pci driver

…and many more

fpgasupdate

- The fpgasupdate tool updates the Intel Max10 BMC image and firmware, root entry hash, and FPGA Static Region (SR) and user image (PR). The fpgasupdate will only accept images that have been formatted using PACsign. If a root entry hash has been programmed onto the board, then the image will also need to be signed using the correct keys.
- The Intel FPGA PAC ships with a factory and user programmed image for both the FIM and BMC FW and RTL on all cards.
- fpgasupdate [--log-level=<level>] file [bdf]

args (optional)	**Description**
--log-level	Specifies the log-level which is the level of information output to your command tool. The following seven levels are available: state, ioctl, debug, info, warning, error, critical. Setting --log-level=state provides the most verbose output. Setting --log-level=ioctl provides the second most information, and so on. The default level is info.
file	Specifies the secure update firmware file to be programmed. This file may be to program a static region (SR), programmable region (PR), root entry hash, key cancellation, or other device-specific firmware.
bdf	The PCIe address of the PAC to program. bdf is of the form [ssss:]bb:dd:f, corresponding to PCIe segment, bus, device, function. The segment is optional. If you do not specify a segment, the segment defaults to 0000. If the system has only one PAC you can omit the bdf and let fpgasupdate determine the address automatically.

fpgainfo

- fpgainfo [-h] [-S] [-B] [-D] [-F] [PCI_ADDR] {errors,power,temp,fme,port,bmc,mac,phy,security}

Command	args (optional)	Description
	--help, -h	Prints help information and exits.
	--version, -v	Prints version information and exits.
	--segment, -S	PCIe segment number of resource.
	--bus, -B	PCIe bus number of resource
	--device, -D	PCIe device number of resource.
	--function, -F	PCIe function number of resource.
errors	{fme, port, all} --clear, -c	First agument to the errors command specifies the resource type to display in human readable format. The second optional argument clears errors for the given FPGA resource.
power		Provides total power in watts that the FPGA hardware consumes
temp		Provides FPGA temperature values in degrees Celsius
port		Provides information about the port
fme		Provides information about the FME
bmc		Provides BMC sensors information
mac		Provides information about MAC ROM connected to FPGA
security		Provides information about the security keys, hashes, and flash count, if available.

rsu

- Performed remote system update operation on a device, given its PCIe address. A rsu operation sends an instruction to the device to trigger a power cycle of the card only. This will force reconfiguration from flash for either the BMC or FPGA.

- The Intel FPGA PAC contains a region of flash the user may store their FIM image. After an image has been programmed with fpgasupdate the user may choose to perform rsu to update the image on the device.

- bash session
  - rsu [-h] [-d] {bmc,bmcimg,retimer,sdm,fpgadefault} [PCIE_ADDR]

Opae.io

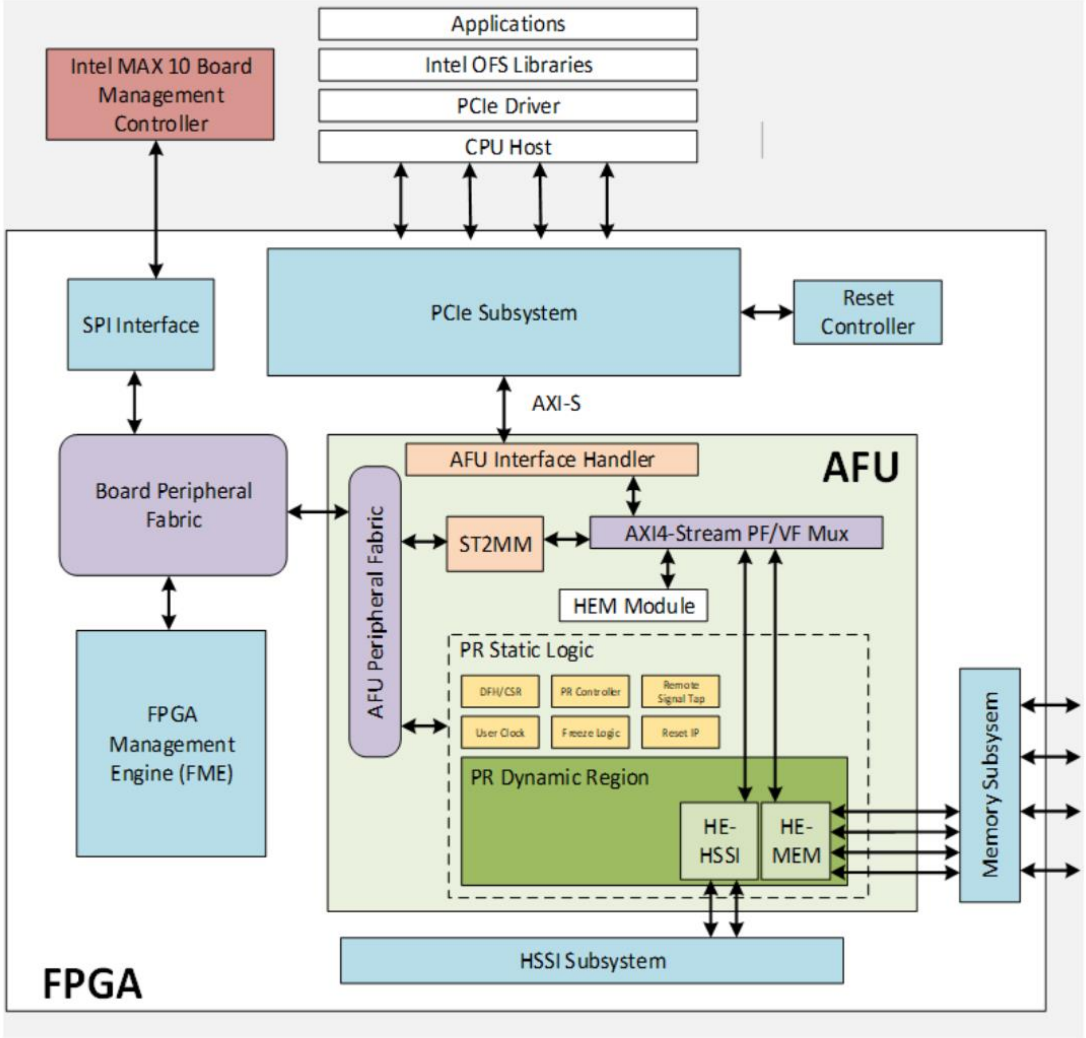
- Opae.io is an interactive Python environment which provides user space access to PCIe devices via vfio-pci driver.
- The main features of opae.io is its build-in Python command interpreter that provide a means to access Configuration and Status Registers (CSRs) that reside on the PCIe device
- Opae.io has 2 operating modes
    - Command line
    - Interactive mode

Opae.io example

- Opae.io ls [-v, --viddid VID:DID]
  - List each accelerator device along with the PCIe address, PCIe vendor/device ID, brief description and the driver to which the currently is bound
- Opae.io init [-d PCI_ADDR USER:[GROUP]]
  - Unbinds the specific device from its current driver and binds it to vfio-pci
- Opae.io release [-d PCI_ADDR]
  - Release the device form vfio-pci
- Opae.op walk [-d PCI_ADDR] [-r REGION] [OFFSET] [-u, --show-uuid]
  - Traverses and display the Device Features List of the given region
- Opae.io peek [-d PCI_ADDR] [-r REGION] OFFSET
  - Options reads and displays a CSR value
- Opae.io poke [-d PCI_ADDR [-r REGION] OFFSET VALUE
  - Writes a given value to a CSR

OFS Development Guide

Stratix 10 Architecture



Pre-requisites

OFS is an advanced application of FPGA technology. This guide assumes you have the following FPGA logic design-related knowledge and skills:

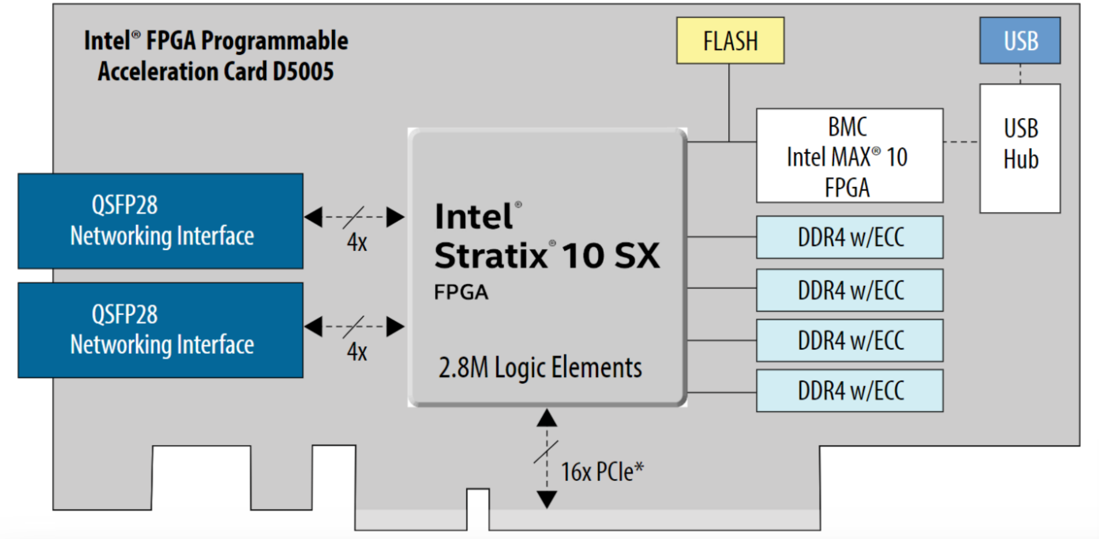
- FPGA compilation flows using Intel® Quartus® Prime Pro Edition design flow.
- Static Timing closure, including familiarity with the Timing Analyzer tool in Intel® Quartus® Prime Pro Edition, applying timing constraints, Synopsys* Design Constraints (.sdc) language and Tcl scripting, and design methods to close on timing critical paths.
- RTL and coding practices for FPGA implementation.
- RTL simulation tools.
- Intel® Quartus® Prime Pro Edition Signal Tap Logic Analyzer tool software.

Development Environment

Item	Version
Intel Quartus Prime Pro	Intel Quartus Prime Pro 22.3 (with license patch)
Target D5005 Sever Operating System	RHEL 8.2
OPAE SDK	2.3.0-1
Linux DFL	ofs-2022.3-2
Python	3.7.7
cmake	3.11.4
GCC	7.2.0
perl	5.8.8

Reference Board FIM

- Host interface
  - PCIe Gen3 x 16
- 2 - QSFP28 cages
- Current FIM supports 1 x 10 GbE, other interfaces can be created
- External Memory
- 2 or 4 channels of DDR4-2400 to RDIMM modules
- RDIMM modules = 8GB organized as 1 Gb X 72
- Board Management
- SPI interface
- FPGA configuration


Intel® FPGA Programmable Acceleration Card D5005
FLASH
USB
BMC Intel MAX® 10 FPGA
USB Hub
QSFP28 Networking Interface
4x
QSFP28 Networking Interface
4x
Intel® Stratix® 10 SX FPGA
2.8M Logic Elements
DDR4 w/ECC
DDR4 w/ECC
DDR4 w/ECC
DDR4 w/ECC
16x PCIe*

FIM FPGA Resource Usage

Summary	FPGA Resource Utilization
Logic utilization (in ALMs)	124,092 / 933,120 (13 %)
Total dedicated logic registers	282822
Total pins	630 / 912 (69 %)
Total block memory bits	3,425,120 / 240,046,080 (1 %)
Total RAM Blocks	661 / 11,721 (6 %)
Total DSP Blocks	0 / 5,760 (0 %)
Total eSRAMs	0 / 75 (0 %)
Total HSSI P-Tiles	17 / 48 (35 %)
Total HSSI E-Tile Channels	17 / 48 (35 %)
Total HSSI HPS	0 / 1 (0 %)
Total HSSI EHIPs	0 / 2 (0 %)
Total PLLs	36 / 104 (35 %)

OFS Contents

Eval Script

Contains scripts for evaluation of OFS for D5005 including compiling FIM/AFU from source, unit level test. Also includes resources to report and setup D5005 development environment

ipss

Contains the code and supporting files that define or set up the IP subsystems (HSSI, PCIe, memory, PMCI, SPI, etc…) contained in the D5005 FPGA Interface Manager (FIM).

License

License file for the Low Latency 10Gbps Ethernet MAC (6AF7 0119) IP core.

ofs-common

This directory contains resources that may be used across the board-specific repositories. This directory is referenced via a link within each of the FPGA-specific repositories.

Sim

Contains the testbenches and supporting code for all the unit test simulations.

Bus Functional Model code is contained here.

Scripts are included for automating a myriad of tasks.

All of the individual unit tests and their supporting code is also located here.

Src

SystemVerilog source and script files

Contains all of the structural and behavioral code for the FIM.

Scripts for generating the AXI buses for module interconnect.

Top-level RTL for synthesis is located in this directory.

Accelerated Functional Unit (AFU) infrastructure code is contained in this directory.

Syn

This directory contains all of the scripts, settings, and setup files for running synthesis on the FIM.

Linux OS package requirement

- RHEL 8.2 Library needed
  - sudo dnf install libnsl sudo dnf install ncurses-compat-libs
  - sudo ln -s /usr/bin/python3 /usr/bin/python

OFS Installation

- Install Quartus Prime Pro 22.3 Linux and setup environment
- Clone the github `ofs-d5005` repository
  - mkdir OFS_fim_build_root
  - cd OFS_fim_build_root
  - export OFS_BUILD_ROOT=$PWD
  - git clone --recurse-submodules https://github.com/OFS/ofs-d5005.git
  - cd ofs-d5005
  - git checkout tags/ofs-d5005-1.0.1
  - Install Quartus Patch
    - cd license
    - chmod +x quartus-0.0-0.01OFS-linux.run
    - sudo ./quartus-0.0-0.01OFS-linux.run
- Test installation by building the provided FIM

Setting up the environment

- cd $OFS_BUILD_ROOT/ofs-d5005
- export OFS_ROOTDIR=$PWD
    - OFS_ROOTDIR is the directory where you cloned the repo, e.g. /home/MyProject/ofs-d5005 *
- export WORKDIR=$OFS_ROOTDIR
- export VERDIR=$OFS_ROOTDIR/verification
- export QUARTUS_HOME=$QUARTUS_ROOTDIR
    - QUARTUS_ROOTDIR is your Quartus installation directory, e.g. $QUARTUS_ROOTDIR/bin contains Quartus executuable*
- export QUARTUS_INSTALL_DIR=$QUARTUS_ROOTDIR
- export IMPORT_IP_ROOTDIR=$QUARTUS_ROOTDIR/../ip
- export IP_ROOTDIR=$QUARTUS_ROOTDIR/../ip
- export OPAE_SDK_REPO_BRANCH=release/2.3.0

Compilation

- ofs-common/scripts/common/syn/build_top.sh [-p] target_configuration work_dir
  - Usage: ofs-common/scripts/common/syn/build_top.sh [-k] [-p] []
- target_configuration - Specifies the project

For example: d5005

- work_dir
  - Work Directory for this build in the form a directory name. It is created in the <local repo directory>/ofs-d5005/<work_dir> - NOTE:
  - The directory name must start with "work". If the work directory exists, then the script stops and asks if you want to overwrite the directory.
  - Example
    - ofs-common/scripts/common/syn/build_top.sh d5005 work_d5005
  - work directory as a name will be created in <local repo directory>/ofs-d5005/work_d5005
  - The obmission of <work_dir> results in a default work directory (<local repo directory>/ofs-d5005/work)
  - compile reports and artifacts (.rpt, .sof, etc) are stored in <work_dir>/syn/syn_top/output_files
  - There is a log file created in ofs-d5005 directory.

Generated

File	Description
d5005.sof	This is the Quartus generated programming file created by Quartus synthesis and place and route. This file can be used to programming the FPGA using a JTAG programmer. This file is used as the source file for the binary files used to program the FPGA flash
d5005.bin	This is an intermediate raw binary image of the FPGA
d5005_page1.bin	This is the binary file created from input file, d5005.sof. This file is used as the input file to the PACSign utility to generate “d5005_page1_unsigned.bin” binary image file.
d5005_page1_unsigned.bin	This is the unsigned PACSign output which can be programmed into the FPGA flash of an unsigned D5005 using the OPAE SDK utility “fpgasupdate”
mfg_d5005_reversed.bin	A special programming file for a third party programming device used in board manufacturing. This file is typically not used.

Enable PR

- cd $OFS_BUILD_ROOT
- git clone https://github.com/OPAE/ofs-platform-afu-bbb
- cd ofs-platform-afu-bbb
- export OFS_PLATFORM_AFU_BBB=$PWD
- cd $OFS_ROOTDIR
- syn/common/scripts/generate_pr_release.sh -t work_d5005/build_tree d5005 work_d5005

PR Build Tree

- Bin
    - Afu_synth
    - Build_env_config
    - Run.sh
    - Update_pim
- Hw
    - Blue_bbits
        - D5005_page1_unsigned.bin
        - D5005.sof
    - Lib
        - Build
        - Fme-ifc-id.txt
        - Fme-platform-class.txt
        - platform

Unit Level Simulation

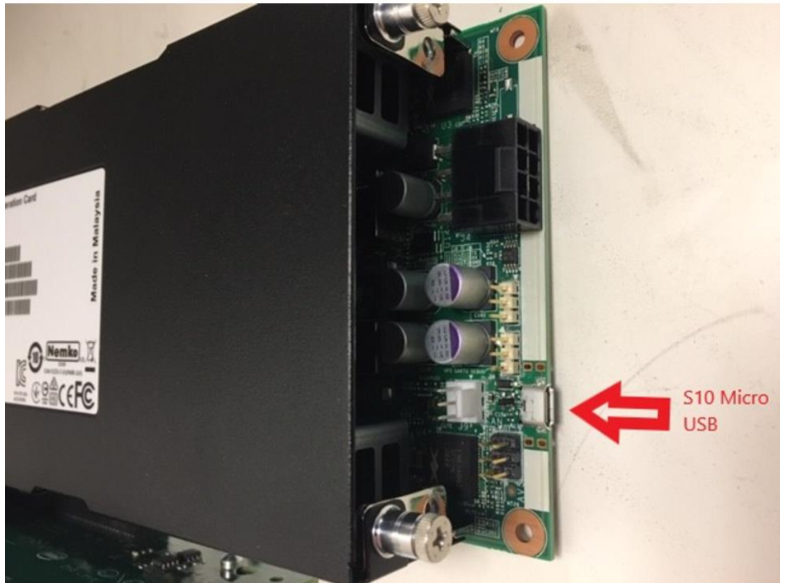
- Key components provided
  - HSSI
  - PCIe
  - External Memory
  - FIM management
- Simulation Requirement
  - Python
  - Quartus
  - VCS

Running simulation

- Compiling all IP
  - cd $OFS_ROOTDIR/ofs-common/scripts/common/sim
  - sh gen_sim_files.sh d5005
- IP Simulation filelist
  - $OFS_ROOTDIR/sim/scripts/ip_flist.f
- RTL file list for unit_test
  - $OFS_ROOTDIR/sim/scripts/rtl_comb.f
- IPs generated
  - $OFS_ROOTDIR/sim/scripts/qip_gen
- Running the simulation
  - $OFS_ROOTDIR/sim/unit_test/<Unit Test Name>/scripts
  - sh run_sim.sh VCS=1

OFS with SignalTap

- Perform a full compile using the script build_top.sh.
- Once the compile completes open the Quartus GUI using the FIM project. The Quartus project is named d5005 and is located in the work directory syn/syn_top/d5005.qpf. Once the project is loaded, go to Tools > Signal Tap Logic Analyzer to bring up the Signal Tap GUI.
- Add the signal needed for debug



AFU Developer Guide

intel®

OFS FIM

- OFS provides a build script with the following FPGA image creation options:
  - Flat compile, which combines the FIM and AFU into one FPGA image loaded into the entire FPGA device as static image
  - PR (Partial Reconfiguration) compile that creates an FPGA image consisting of the FIM that is loaded into the static region of the FPGA and a default AFU that is loaded into dynamic region.
- Build scripts included with OFS are verified to run in bash shell

Flat Compile

Hello FIM example (Top Level design)

If you intend to add a new module to the FIM area, then you will need to inform the host software of the new module. The FIM exposes its functionalities to host software through a set of CSR registers that are mapped to an MMIO region (Memory Mapped IO). This set of CSR registers and their operation is described in FIM MMIO Regions.

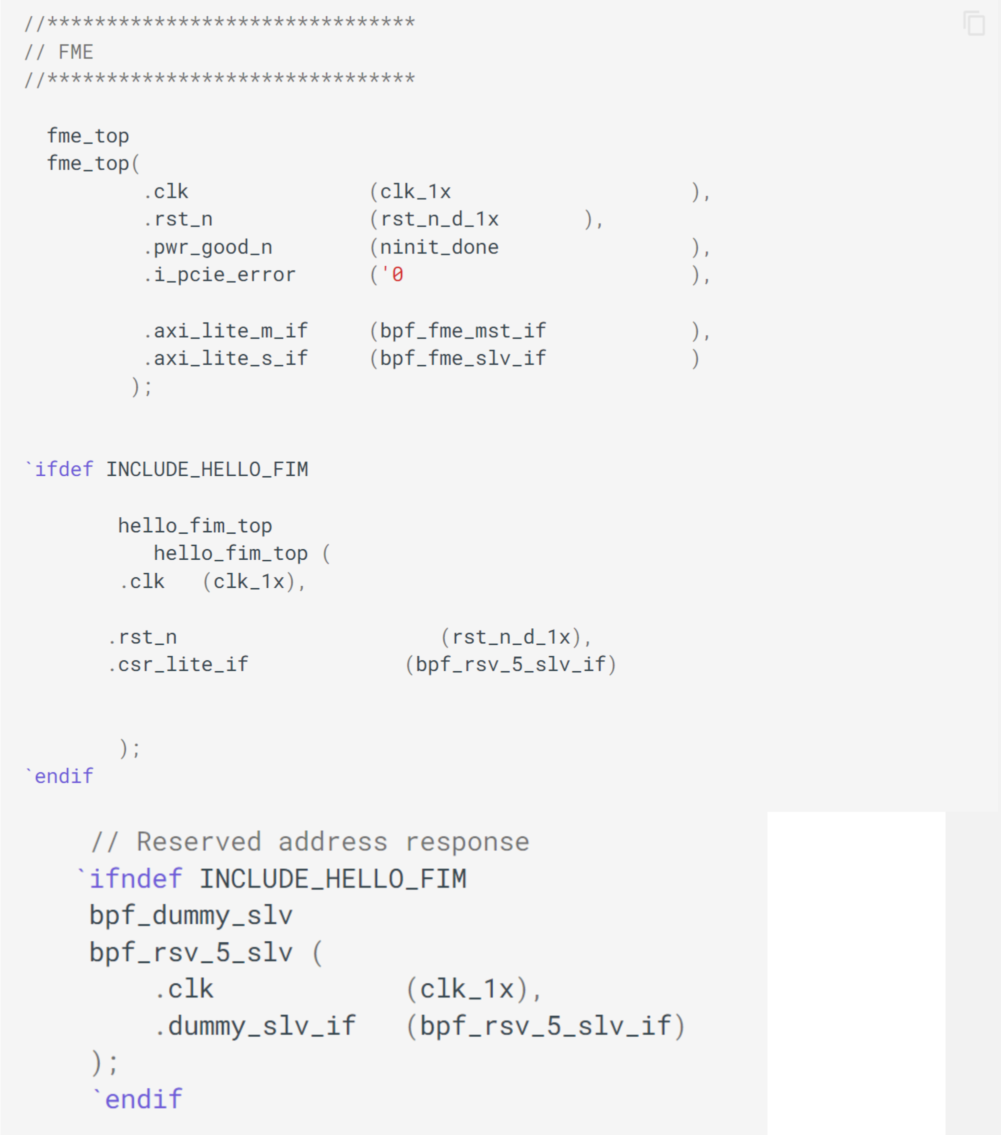
Step to add simple DFH register

1. Review current design documentation: OFS Tech Ref MMIO Regions
2. Understand FME and Port regions, DFH walking, DFH register structure
3. Run unit level simulations and review output: i. sim/unit_test/dfh_walker
4. Note DFH link list order, see DFH Walker Unit Level Simulation Output
5. Make code changes to top level FIM file to instantiate new DFH register
6. The DFH registers follow a link list. This example inserts the hello_fim DFH register after the EMIF DFH register, so the emif_csr.sv parameters are updated to insert the hello_fim DFH register as next register in the link list.
7. Create the new hello_fim SystemVerilog files.
8. Update and run the dfh_walker unit simulation files
9. Update synthesis files to include the new hello_fim source files
10. Build and test the new FIM

OFS Top Level

- src/top/iofs_top.sv

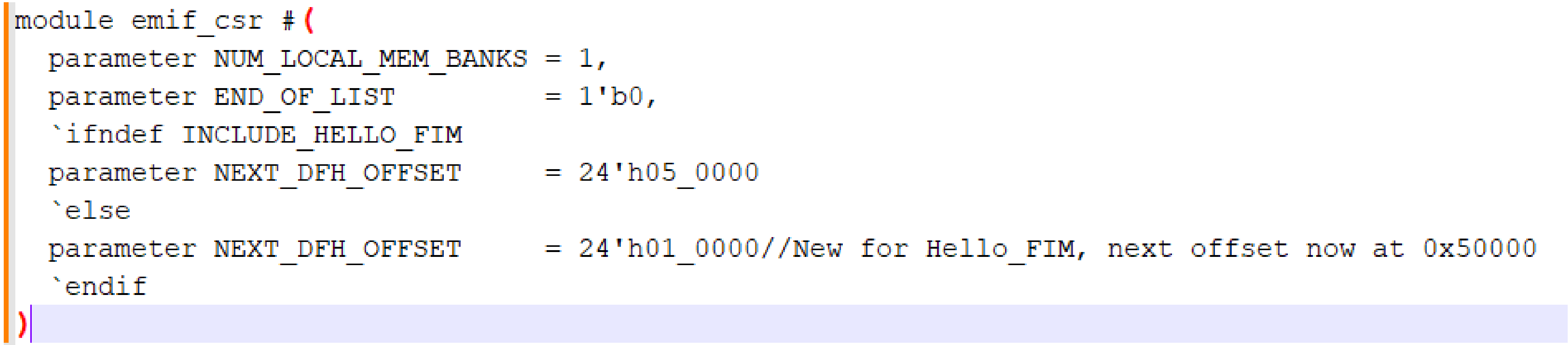
Address	Size (Byte)	Feature	Master
0x00000 – 0x0FFFF	64K	FME (FME, Error, etc)	Yes
0x10000 – 0x1FFFF	64K	PMCI Proxy (SPI Controller)	Yes
0x20000 – 0x2FFFF	64K	PCIe CSR	
0x30000 – 0x3FFFF	64K	HSSI CSR	
0x40000 – 0x4FFFF	64K	EMIF CSR	
0x50000 – 0x5FFFF	64K	Reserved	
0x60000 – 0x6FFFF	64K	Reserved	
0x70000 – 0x7FFFF	64K	Reserved	



```
//*******************************
// FME
//*******************************

  fme_top
  fme_top(
          .clk                (clk_1x                   ),
          .rst_n              (rst_n_d_1x        ),
          .pwr_good_n         (ninit_done               ),
          .i_pcie_error       ('0                       ),

          .axi_lite_m_if      (bpf_fme_mst_if           ),
          .axi_lite_s_if      (bpf_fme_slv_if           )
         );


`ifdef INCLUDE_HELLO_FIM

        hello_fim_top
           hello_fim_top (
        .clk    (clk_1x),

       .rst_n                      (rst_n_d_1x),
       .csr_lite_if             (bpf_rsv_5_slv_if)


        );
`endif

    // Reserved address response
   `ifndef INCLUDE_HELLO_FIM
    bpf_dummy_slv
    bpf_rsv_5_slv (
         .clk            (clk_1x),
         .dummy_slv_if   (bpf_rsv_5_slv_if)
    );
     `endif
```

EMIF CSR (ipss/mem/emif_csr.sv)

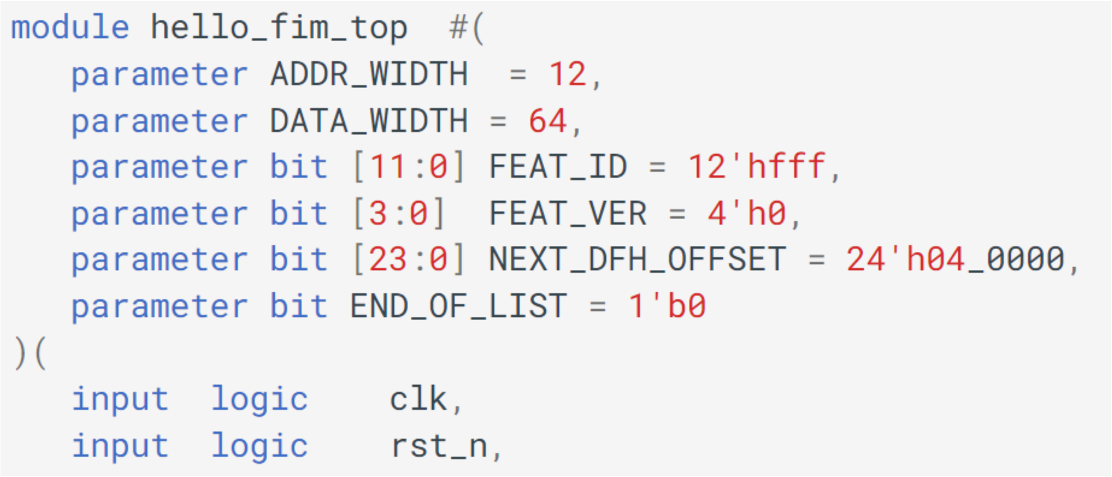
The Hello_FIM DFH is inserted in the DFH link list after the EMIF CSR DFH and before the FME_PR DFH. The file ipss/d5005/emif/emif_csr.sv contains a parameter defining the next address for the next DFH in in the link list chain. You will change the next address offset to be 0x10000 so the reserved BPF AXI lite link connected to the Hello_FIM DFH register is next in the DFH link list.



```
module emif_csr #(
  parameter NUM_LOCAL_MEM_BANKS = 1,
  parameter END_OF_LIST         = 1'b0,
  `ifndef INCLUDE_HELLO_FIM
  parameter NEXT_DFH_OFFSET     = 24'h05_0000
  `else
  parameter NEXT_DFH_OFFSET     = 24'h01_0000//New for Hello_FIM, next offset now at 0x50000
  `endif
)
```

Hello FIM TOP (src/hello_fim/hello_fim_top.sv)

- Create hello_fim_top.sv, and store it in src/hello_fim directory.
- The main purpose of this RTL is to convert AXI4-Lite interface to a simple interface to interface with the registers in hello_fim_com.sv.
- This register sets the DFH feature ID to 0xfff which is undefined.
- For test purposes, using an undefined feature ID will result in no driver being used.
- Normally, a defined feature ID will be used to associate a specific driver with the FPGA module.

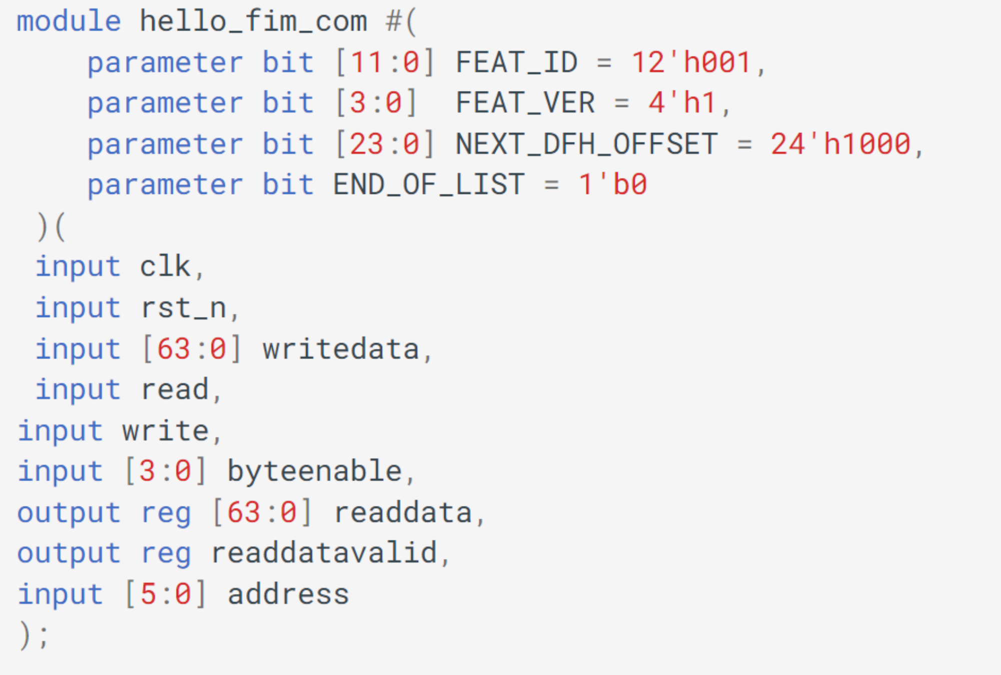


```
module hello_fim_top  #(
   parameter ADDR_WIDTH  = 12,
   parameter DATA_WIDTH = 64,
   parameter bit [11:0] FEAT_ID = 12'hfff,
   parameter bit [3:0]  FEAT_VER = 4'h0,
   parameter bit [23:0] NEXT_DFH_OFFSET = 24'h04_0000,
   parameter bit END_OF_LIST = 1'b0
)(
   input  logic    clk,
   input  logic    rst_n,
```

hello_fim_top.sv

Hello FIM COM (src/hello_fim/hello_fim_com.sv)

Create hello_fim_com.sv, and store it in src/hello_fim directory. This is the simple RTL to implement the Hello FIM registers. You may use this set of registers as the basis for your custom implementation.



```
module hello_fim_com #(
    parameter bit [11:0] FEAT_ID = 12'h001,
    parameter bit [3:0]  FEAT_VER = 4'h1,
    parameter bit [23:0] NEXT_DFH_OFFSET = 24'h1000,
    parameter bit END_OF_LIST = 1'b0
 )(
 input clk,
 input rst_n,
 input [63:0] writedata,
 input read,
input write,
input [3:0] byteenable,
output reg [63:0] readdata,
output reg readdatavalid,
input [5:0] address
);
```

hello_fim_com.sv

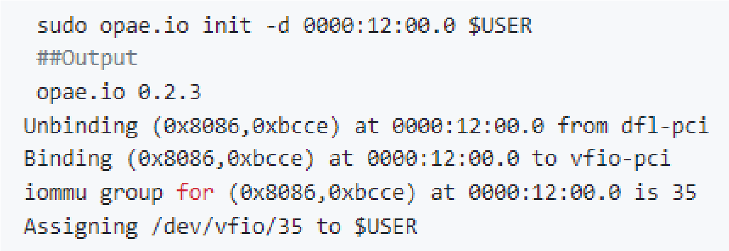
Update setting fiile

- Edit syn/syn_top/d5005.qsf
  - set_global_assignment -name VERILOG_MACRO INCLUDE_HELLO_FIM
  - set_global_assignment -name SOURCE_TCL_SCRIPT_FILE ../../../syn/setup/hello_fim_design_files.tcl

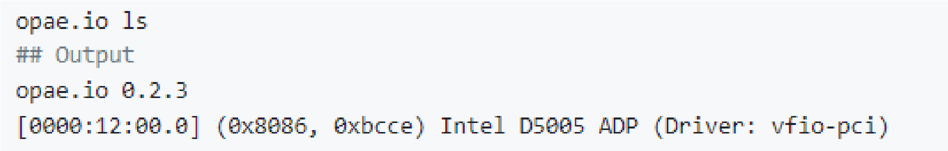
Build hello_fim example

- Execute command
  - cd $OFS_ROOTDIR
  - ofs-common/scripts/common/syn/build_top.sh d5005 work_d5005_hello_fim
- Updating the board
  - sudo fpgasupdate d5005_page1_unsigned.bin <D5005 PCIe B:D.F>
  - sudo rsu bmcimg <D5005 PCIe B:D.F>

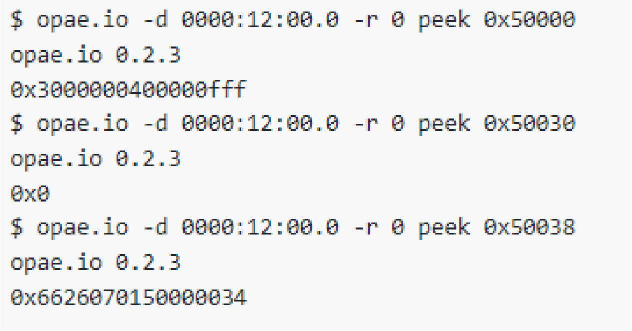
Running Hello World



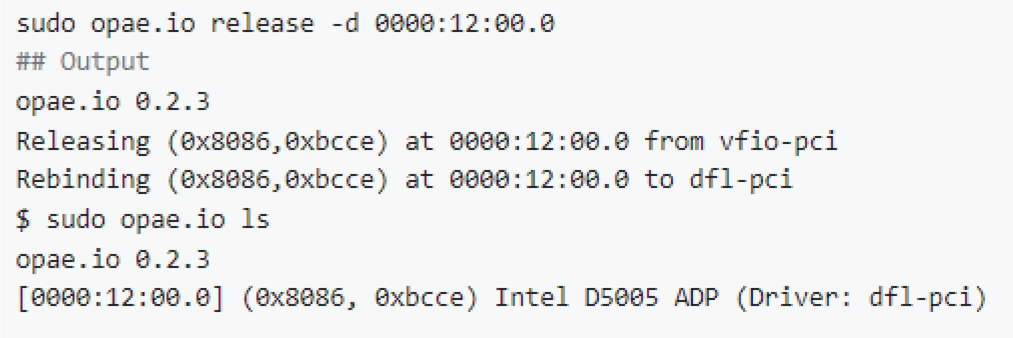
```
 sudo opae.io init -d 0000:12:00.0 $USER
 ##Output
 opae.io 0.2.3
Unbinding (0x8086,0xbcce) at 0000:12:00.0 from dfl-pci
Binding (0x8086,0xbcce) at 0000:12:00.0 to vfio-pci
iommu group for (0x8086,0xbcce) at 0000:12:00.0 is 35
Assigning /dev/vfio/35 to $USER
```



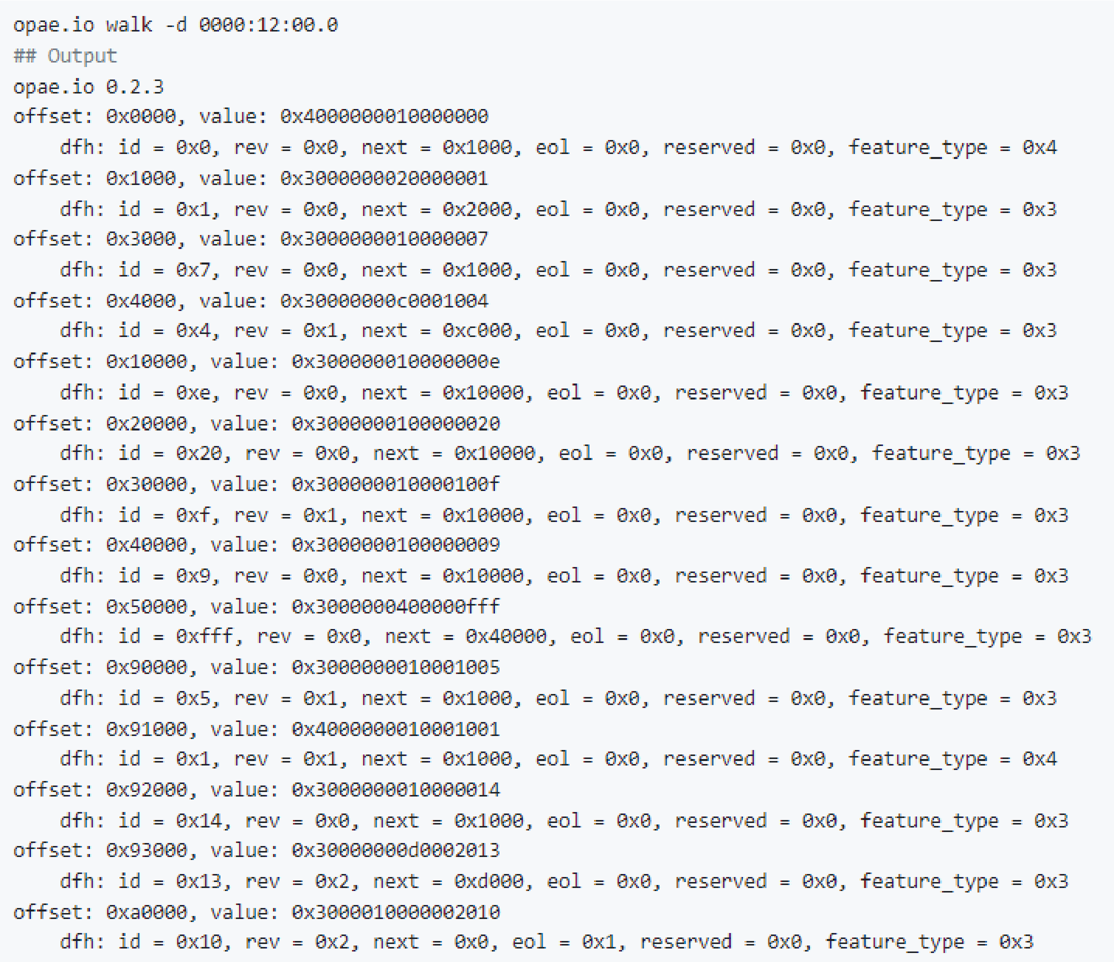
```
opae.io ls
## Output
opae.io 0.2.3
[0000:12:00.0] (0x8086, 0xbcce) Intel D5005 ADP (Driver: vfio-pci)
```



```
$ opae.io -d 0000:12:00.0 -r 0 peek 0x50000
opae.io 0.2.3
0x3000000400000fff
$ opae.io -d 0000:12:00.0 -r 0 peek 0x50030
opae.io 0.2.3
0x0
$ opae.io -d 0000:12:00.0 -r 0 peek 0x50038
opae.io 0.2.3
0x6626070150000034
```



```
sudo opae.io release -d 0000:12:00.0
## Output
opae.io 0.2.3
Releasing (0x8086,0xbcce) at 0000:12:00.0 from vfio-pci
Rebinding (0x8086,0xbcce) at 0000:12:00.0 to dfl-pci
$ sudo opae.io ls
opae.io 0.2.3
[0000:12:00.0] (0x8086, 0xbcce) Intel D5005 ADP (Driver: dfl-pci)
```

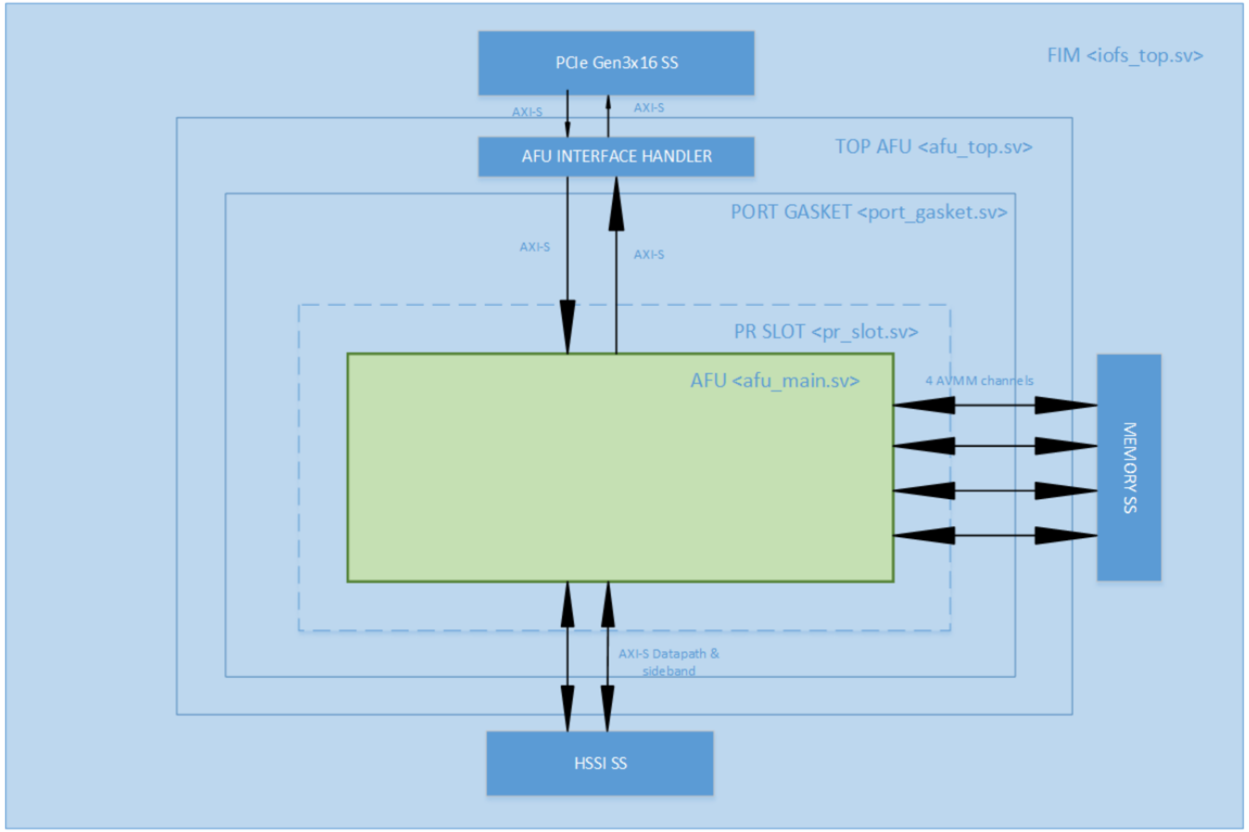


```
opae.io walk -d 0000:12:00.0
## Output
opae.io 0.2.3
offset: 0x0000, value: 0x4000000010000000
    dfh: id = 0x0, rev = 0x0, next = 0x1000, eol = 0x0, reserved = 0x0, feature_type = 0x4
offset: 0x1000, value: 0x3000000020000001
    dfh: id = 0x1, rev = 0x0, next = 0x2000, eol = 0x0, reserved = 0x0, feature_type = 0x3
offset: 0x3000, value: 0x3000000010000007
    dfh: id = 0x7, rev = 0x0, next = 0x1000, eol = 0x0, reserved = 0x0, feature_type = 0x3
offset: 0x4000, value: 0x30000000c0001004
    dfh: id = 0x4, rev = 0x1, next = 0xc000, eol = 0x0, reserved = 0x0, feature_type = 0x3
offset: 0x10000, value: 0x300000010000000e
    dfh: id = 0xe, rev = 0x0, next = 0x10000, eol = 0x0, reserved = 0x0, feature_type = 0x3
offset: 0x20000, value: 0x3000000100000020
    dfh: id = 0x20, rev = 0x0, next = 0x10000, eol = 0x0, reserved = 0x0, feature_type = 0x3
offset: 0x30000, value: 0x300000010000100f
    dfh: id = 0xf, rev = 0x1, next = 0x10000, eol = 0x0, reserved = 0x0, feature_type = 0x3
offset: 0x40000, value: 0x3000000100000009
    dfh: id = 0x9, rev = 0x0, next = 0x10000, eol = 0x0, reserved = 0x0, feature_type = 0x3
offset: 0x50000, value: 0x3000000400000fff
    dfh: id = 0xfff, rev = 0x0, next = 0x40000, eol = 0x0, reserved = 0x0, feature_type = 0x3
offset: 0x90000, value: 0x3000000010001005
    dfh: id = 0x5, rev = 0x1, next = 0x1000, eol = 0x0, reserved = 0x0, feature_type = 0x3
offset: 0x91000, value: 0x4000000010001001
    dfh: id = 0x1, rev = 0x1, next = 0x1000, eol = 0x0, reserved = 0x0, feature_type = 0x4
offset: 0x92000, value: 0x3000000010000014
    dfh: id = 0x14, rev = 0x0, next = 0x1000, eol = 0x0, reserved = 0x0, feature_type = 0x3
offset: 0x93000, value: 0x30000000d0002013
    dfh: id = 0x13, rev = 0x2, next = 0xd000, eol = 0x0, reserved = 0x0, feature_type = 0x3
offset: 0xa0000, value: 0x3000010000002010
    dfh: id = 0x10, rev = 0x2, next = 0x0, eol = 0x1, reserved = 0x0, feature_type = 0x3
```

PR compile

OFS FIM for PR Compile

- AXI Streaming (AXI-S) interface to Host via PCIe Gen3xx16
- Avalon Memory-Mapped Channels (4) to the DDR4 EMIF interface
- AXI Streaming (AXI-S) interface to the HSSI 10G Ethernet



Platform Interface Manager (PIM)

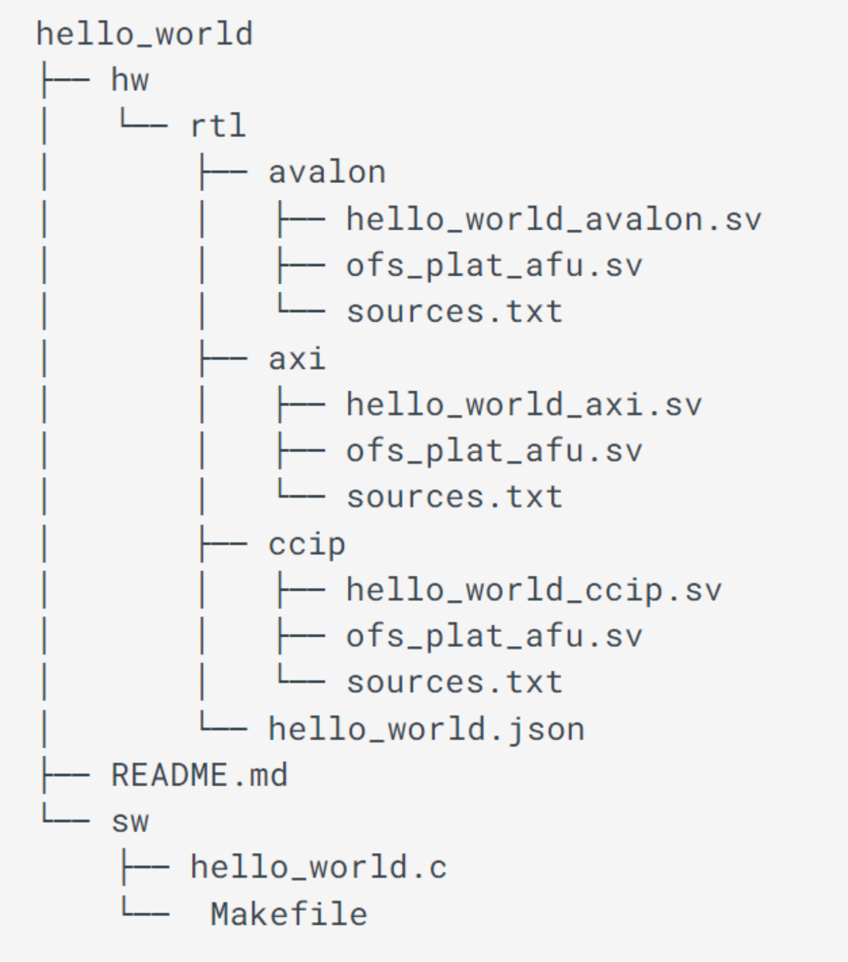
- PIM is a transformation layer between an AFU and raw FIM device interface.
  - Standard AFU-side SystemVerilog interfaces, both AXI and Avalon memory mapped and streaming.
  - PIM-provided modules that transform FIM interfaces to PIM interfaces. Transformations may be simple, such as mapping a FIM AXI-MM interface to the PIM's AXI-MM. Transformations may also be complex, such as mapping a PCIe TLP stream to AXI-MM, adding a clock crossing and sorting read responses.
  - A top-level module and interface bundle with consistent naming across all platforms, promoting AFU portability.
- Ofs-platform-afu-bbb repository contains the PIM files and example AFU.
  - git clone https://github.com/OFS/ofs-platform-afu-bbb.git

Top module

- If using PIM
  - ofs_plat_afu.sv
  - ex) https://github.com/OFS/examples-afu/blob/main/tutorial/afu_types/01_pim_ifc/hello_world/hw/rtl/avalon/ofs_plat_afu.sv
- If not using PIM
  - afu_main.sv
  - ex) https://github.com/OFS/examples-afu/blob/main/tutorial/afu_types/03_afu_main/hello_world/hw/rtl/afu_main.sv
  - Need to handle AXI streaming I/F which includes PCIE TLP packets by developer themselves

Example AFU (hello_world)

- Compiling and executing PIM-based hello_world AFU
  - hw directory - contains the RTL to implement the hardware functionality using CCIP, Avalon and AXI interface
  - Sw directory – contains source code of the host application that communicates with the AFU hardware
- git clone https://github.com/OFS/examples-afu.git



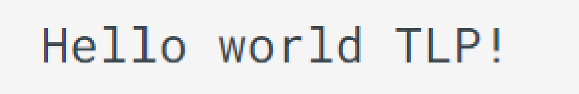
```
hello_world
├── hw
│   └── rtl
│       ├── avalon
│       │   ├── hello_world_avalon.sv
│       │   ├── ofs_plat_afu.sv
│       │   └── sources.txt
│       ├── axi
│       │   ├── hello_world_axi.sv
│       │   ├── ofs_plat_afu.sv
│       │   └── sources.txt
│       ├── ccip
│       │   ├── hello_world_ccip.sv
│       │   ├── ofs_plat_afu.sv
│       │   └── sources.txt
│       └── hello_world.json
├── README.md
└── sw
    ├── hello_world.c
    └──  Makefile
```

Compiling hello_world sample AFU

- Environment setup
  - export FPGA_BBB_CCI_SRC=$OFS_BUILD_ROOT/examples-afu
  - export OPAE_PLATFORM_ROOT=$OFS_ROOTDIR/work_d5005/build_tree
- Execute
  - ofs-common/scripts/common/syn/generate_pr_release.sh -t work_d5005/build_tree d5005 work_d5005
  - afu_synth_setup -s $FPGA_BBB_CCI_SRC/tutorial/afu_types/01_pim_ifc/hello_world/hw/rtl/axi/sources.txt hello_world_synth
  - cd hello_world_synth
  - ${OPAE_PLATFORM_ROOT}/bin/afu_synth
- “hello_world.gbs” bitstream will be created

Running the AFU

- Update PR image
  - sudo fpgasupdate hello_world.gbs 3b:00.0
- Create virtual function (VFs) and bond VFs to VFIO Driver
  - sudo pci_device 3b:00.0 vf 3
  - sudo opae.io init -d 0000:3b:00.1 <Your username>
  - sudo opae.io init -d 0000:3b:00.2 <Your username>
  - sudo opae.io init -d 0000:3b:00.3 <Your username>
- Host Application
  - cd $FPGA_BBB_CCI_SRC/tutorial/afu_types/01_pim_ifc/hello_world/sw/
  - make
  - ./hello_world

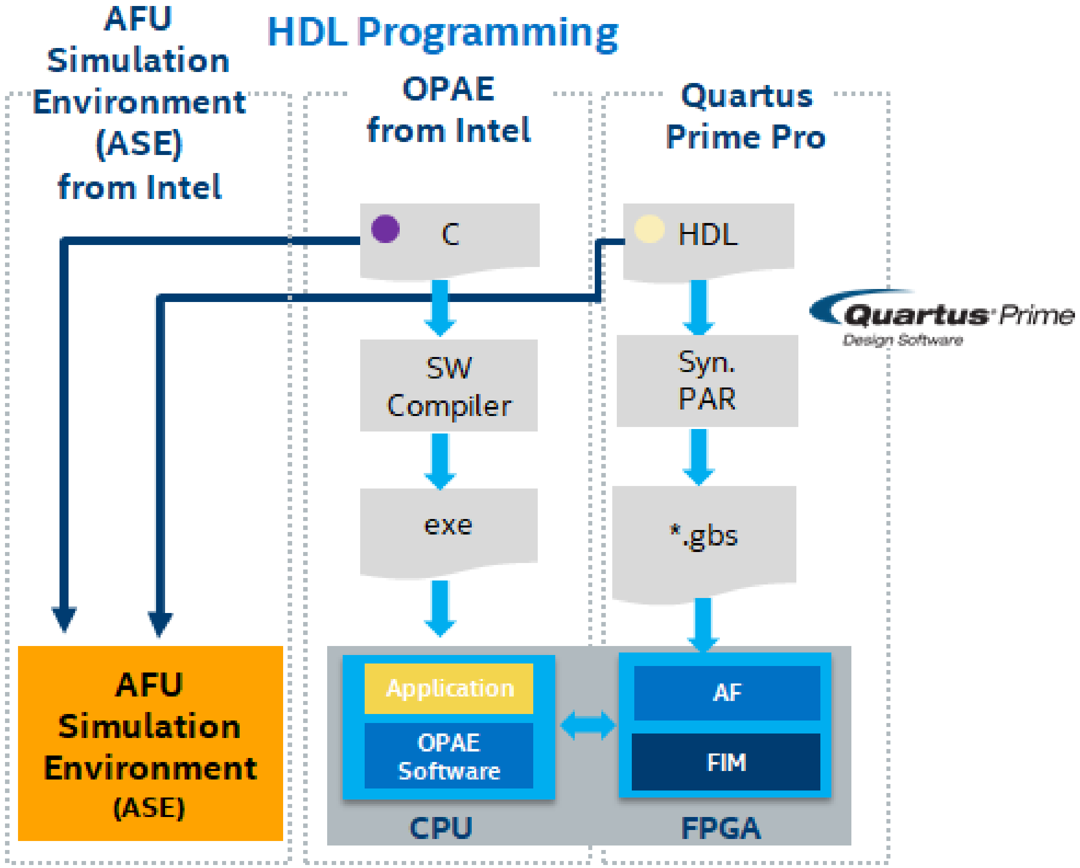


```
Hello world TLP!
```

AFU Simulation Environment (ASE)

Overview

- ASE is a hardware/software co-simulation environment for AFU
- ASE uses the simulator Direct Programming Interface (DPI) to provide HW/SW connectivity.
- The PCIe connection is emulated with transactional model

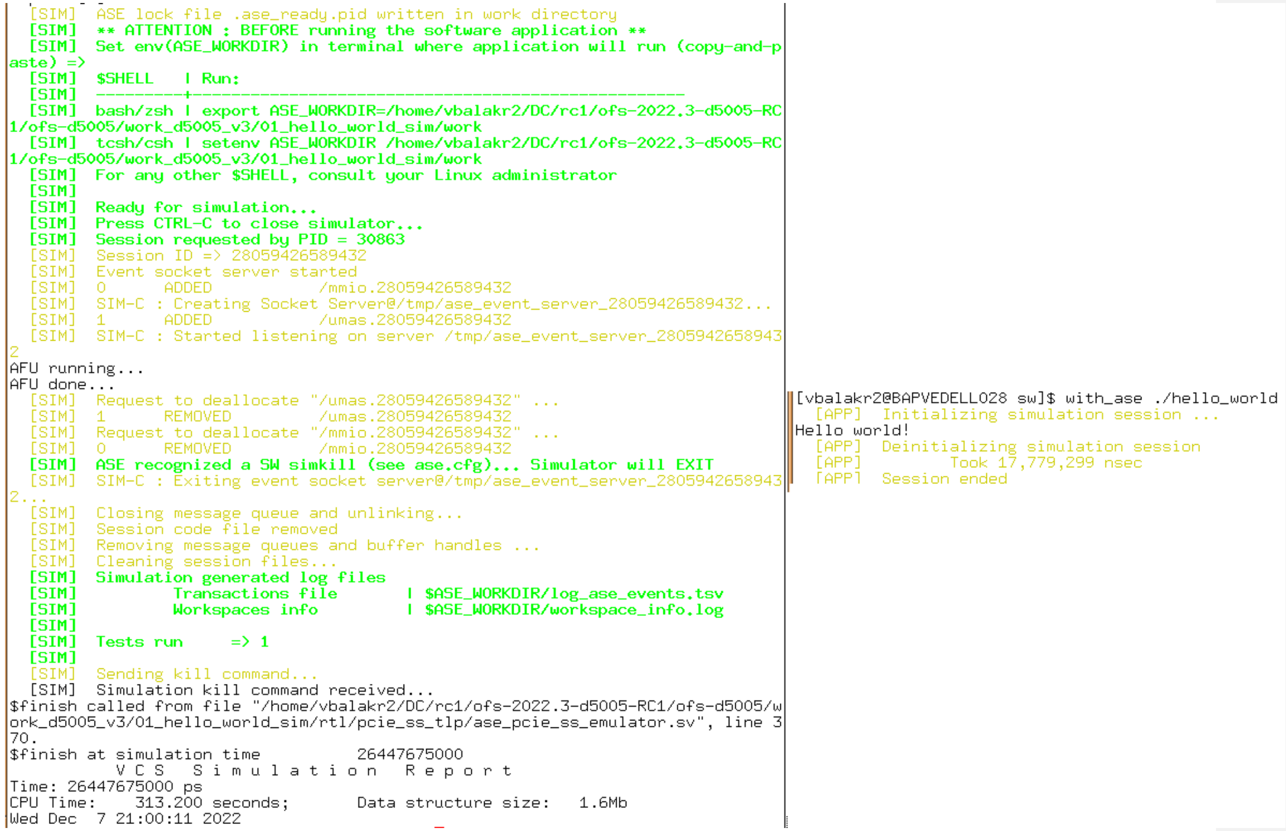


ASE Operation

- Attempts to replicate the transactions that will be seen in real system.
- Provides a memory model to AFU, so illegal memory accesses can be identified early.
- Not a cache simulator.
- Does not guarantee synthesizability or timing closure.
- Does not model system latency.
- No administrator privileges are needed to run ASE. All code is user level.

Running ASE

- Set the environment variables
- Preparing hello_world AFU for SIM
  - afu_sim_setup -s $FPGA_BBB_CCI_SRC/tutorial/afu_types/01_pim_ifc/hello_world/hw/rtl/axi/sources.txt -t VCS hello_world_sim
- Build and execute the AFU RTL simulator
  - cd $OFS_ROOTDIR/work_d5005/hello_world_sim
  - make
  - make sim
- Open second shell for host software
  - export ASE_WORKDIR=$OFS_ROOTDIR/work_d5005/hello_world_sim/work
  - cd $FPGA_BBB_CCI_SRC/examples-afu/tutorial/afu_types/01_pim_ifc/hello_world/sw
  - make
  - with_ase ./hello_world



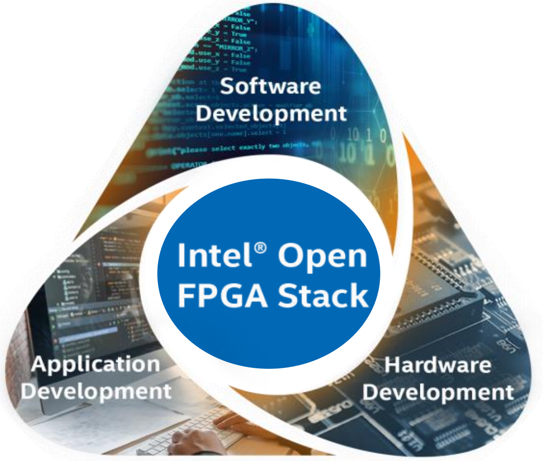
```
  [SIM]  ASE lock file .ase_ready.pid written in work directory
  [SIM]  ** ATTENTION : BEFORE running the software application **
  [SIM]  Set env(ASE_WORKDIR) in terminal where application will run (copy-and-p
aste) =>
  [SIM]  $SHELL   | Run:
  [SIM]  ---------+---------------------------------------------------
  [SIM]  bash/zsh | export ASE_WORKDIR=/home/vbalakr2/DC/rc1/ofs-2022.3-d5005-RC
1/ofs-d5005/work_d5005_v3/01_hello_world_sim/work
  [SIM]  tcsh/csh | setenv ASE_WORKDIR /home/vbalakr2/DC/rc1/ofs-2022.3-d5005-RC
1/ofs-d5005/work_d5005_v3/01_hello_world_sim/work
  [SIM]  For any other $SHELL, consult your Linux administrator
  [SIM]
  [SIM]  Ready for simulation...
  [SIM]  Press CTRL-C to close simulator...
  [SIM]  Session requested by PID = 30863
  [SIM]  Session ID => 28059426589432
  [SIM]  Event socket server started
  [SIM]  0      ADDED          /mmio.28059426589432
  [SIM]  SIM-C : Creating Socket Server@/tmp/ase_event_server_28059426589432...
  [SIM]  1      ADDED          /umas.28059426589432
  [SIM]  SIM-C : Started listening on server /tmp/ase_event_server_2805942658943
2
AFU running...
AFU done...
  [SIM]  Request to deallocate "/umas.28059426589432" ...
  [SIM]  1      REMOVED        /umas.28059426589432
  [SIM]  Request to deallocate "/mmio.28059426589432" ...
  [SIM]  0      REMOVED        /mmio.28059426589432
  [SIM]  ASE recognized a SW simkill (see ase.cfg)... Simulator will EXIT
  [SIM]  SIM-C : Exiting event socket server@/tmp/ase_event_server_2805942658943
2...
  [SIM]  Closing message queue and unlinking...
  [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/log_ase_events.tsv
  [SIM]          Workspaces info         | $ASE_WORKDIR/workspace_info.log
  [SIM]
  [SIM]  Tests run     => 1
  [SIM]
  [SIM]  Sending kill command...
  [SIM]  Simulation kill command received...
$finish called from file "/home/vbalakr2/DC/rc1/ofs-2022.3-d5005-RC1/ofs-d5005/w
ork_d5005_v3/01_hello_world_sim/rtl/pcie_ss_tlp/ase_pcie_ss_emulator.sv", line 3
70.
$finish at simulation time          26447675000
           V C S   S i m u l a t i o n   R e p o r t
Time: 26447675000 ps
CPU Time:    313.200 seconds;       Data structure size:   1.6Mb
Wed Dec  7 21:00:11 2022
```

```
[vbalakr2@BAPVEDELL028 sw]$ with_ase ./hello_world
  [APP]  Initializing simulation session ...
Hello world!
  [APP]  Deinitializing simulation session
  [APP]         Took 17,779,299 nsec
  [APP]  Session ended
```

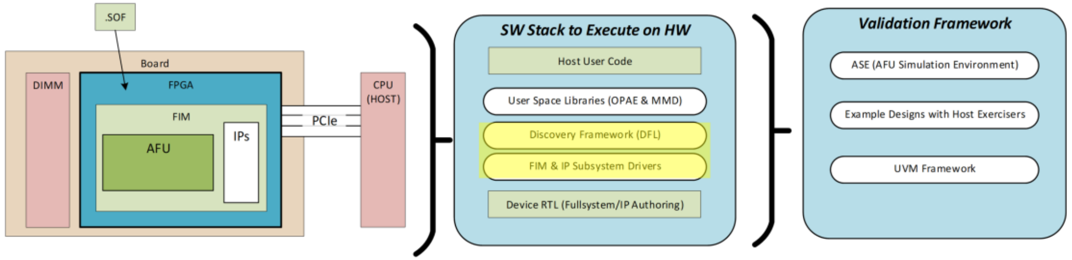
Summary

Why is OFS?

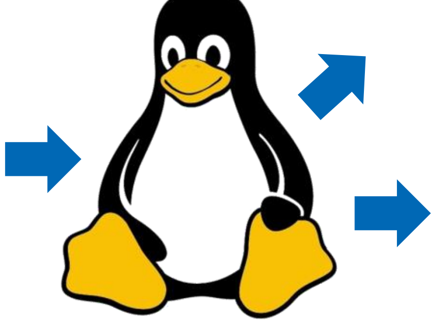
- **Use Standard Interfaces and Application Programming Interfaces (APIs)** to accelerate workload development and enable code reuse
- **Port existing workloads to the Acceleration Functional Unit (AFU) Region** and proliferate across OFS-based platforms for FPGA-based acceleration or CPU offload
- **Utilize a growing ecosystem** of OFS-enabled boards and workloads provided by Intel and third parties
- **Deploy Bare Metal, Virtualized, or Containerized** applications with data center class management for NFV, SmartNIC, VRAN, FSI, and more
- **Build Application Specific FPGA Interface Managers (FIMs)** from the provided reference FIMs using a modular, 'take and tailor' approach
- **Leverage Open-Sourced and Upstreamed Software Code** delivered via Git Repositories with native support from leading software vendors



Linux Upstreaming


.SOF
Board
DIMM
FPGA
FIM
AFU
IPs
PCIe
CPU (HOST)
SW Stack to Execute on HW
Host User Code
User Space Libraries (OPAE & MMD)
Discovery Framework (DFL)
FIM & IP Subsystem Drivers
Device RTL (Fullsystem/IP Authoring)
Validation Framework
ASE (AFU Simulation Environment)
Example Designs with Host Exercisers
UVM Framework

Contribute to Linux Kernel



Enable more developers

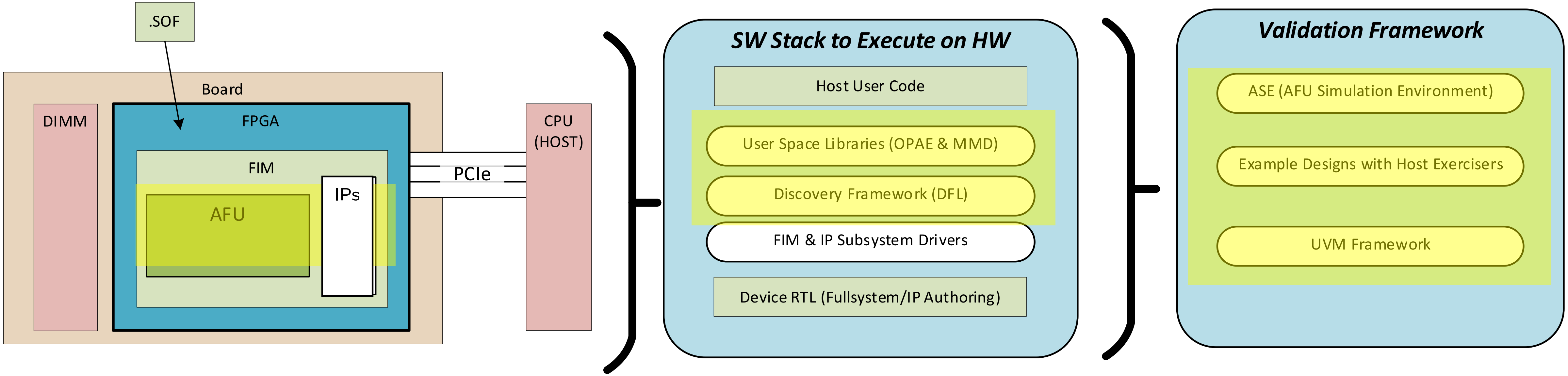
OS vendors distribute in their Linux based OS

Enable even more developers

- Reuse drivers for new cards & gens
- Get bug fixes from community
- Code reviewed by community

- Influence kernel development
- Avoid 3rd party impl. conflicts
- Enable more developers

Check out OFS: https://github.com/OFS!



.SOF

Board

DIMM

FPGA

FIM

AFU

IPs

PCIe

CPU (HOST)

SW Stack to Execute on HW

Host User Code

User Space Libraries (OPAE & MMD)

Discovery Framework (DFL)

FIM & IP Subsystem Drivers

Device RTL (Fullsystem/IP Authoring)

Validation Framework

ASE (AFU Simulation Environment)

Example Designs with Host Exercisers

UVM Framework

OFS BASE FIM

- FIM COMMON https://github.com/OFS/ofs-fim-common
- OFS-D5005 (Intel Stratix 10) https://github.com/OFS/ofs-d5005

Drivers and SW Tools

- LINUX-DFL https://github.com/OFS/linux-dfl
- OPAE-SDK https://github.com/OFS/opae-sdk

Validation Framework

- OPAE-SIM https://github.com/OFS/opae-sim

Term	Acronym	Description
FPGA Interface Manger	FIM	FIMs provide platform management, functionality, clocks, resets, and standard interfaces to the host and AFU. Reference FIMs are provided as working examples to help users develop their own custom platforms. Users can 'take and tailor' the reference FIMs.
Acceleration Functional Unit Region	AFU	This region is the designated area for custom workload development, containing both static and dynamic regions.
Partial Reconfiguration Region	PR	Region in the AFU for dynamically programming part or all of the workload.
FPGA Management Engine	FME	Contains FPGA management CSRs and logic for global control of the device.
Hard Processor System	HPS	The HPS in the Agilex architecture has a quad core ARM Cortex A-53 MPCore and integrates a wide set of peripherals to reduce board size and increase performance
Board Management Controller	BMC	Controls, monitors, and grants access to board features providing Root of Trust using a MAX 10 FPGA. The Agilex architecture includes an enhanced BMC architecture.
Subsystems		Memory, PCIe, and HSSI are built as modular subsystem to make them easier to modify.
Interconnect Fabrics		Route packets from PCIe subsystem to respective PF/VF functions.
Host Exerciser Modules	HE-LBK, HE-MEM, HE-HSSI	Host exercisers do full memory, PCIe, or ethernet loopbacks to demonstrate external interface capabilities. Host exercisers are evaluation tools and not meant to included in production designs.
AXI-Lite Interface		AXI is an industry-standard interface protocol. AXI provides a standardized IP protocol compatible with ARM and ARM partners.
Universal Verification Methodology Support	UVM	This industry standard verification environment is provided for the FIM & AFU.
Open Programmable Acceleration Engine Software Development Kit	OPAE SDK	A collection of libraries and tools to facilitate the development of software applications and accelerators using OPAE.
Platform Interface Manager	PIM	An interface manager that comprises two components: a configurable platform specific interface for board developers and a collection of shims that AFU developers can use to handle clock crossing, response sorting, buffering and different protocols.
Remote System Update	RSU	A remote system update operation sends an instruction to the device that triggers a power cycle of the card only, forcing reconfiguration
Device Feature List	DFL	A main component of the software infrastructure for OFS. The DFL driver provides support for FPGA devices that are designed to support the Device Feature List. The DFL, which is implemented in RTL, consists of a self-describing data structure in PCI BAR space that allows the DFL driver to automatically load the drivers required for a given FPGA configuration.
AFU Simulation Environment	ASE	A hardware/software co-simulation environment provided for the AFU as part of OPAE.

Reference

- Ofs.github.io
- https://github.com/OFS/examples-afu/
- https://github.com/OFS/ofs-platform-afu-bbb

intel®