ACCELERATION STACK FOR INTEL® XEON® CPU WITH FPGA

Bill Jenkins
Intel Programmable Solutions Group
Objectives

Describe high-level parallel computing concepts and challenges

Understand the advantages of using the acceleration stack with Intel® FPGAs

Write host software applications that can transparently access Intel® FPGAs

Understand the design flows and options for creating workloads for the FPGA

Build and simulate accelerator workloads for the FPGA
Agenda

Introduction and acceleration stack overview

Getting Started with the Acceleration Stack

Developing a SW host application
  – Lab 1

Introduction to Accelerator Functional Unit (AFU)

Creating an Accelerator Functional Unit (AFU)

Co-simulation using AFU Simulation Environment (ASE)

Compiling the Accelerator Function Unit into an Accelerator Function (AF)

Debugging an Accelerator Function
  – Lab 2

Conclusion
Better Computation Enables Innovation and Discovery

- Astrophysics
- Genomics
- Artificial Intelligence
- Manufacturing
- Data Analytics
- Financial
- Weather & Climate
- Cyber Security
50+ Years of Moore’s Law
Computing has Changed...
The Urgency of Parallel Computing

If engineers keep building processors the way we do now, CPUs will get even faster but they’ll require so much power that they won’t be usable.

—Patrick Gelsinger, former Intel Chief Technology Officer, February 7, 2001

Source: http://www.cnn.com/2001/tech/ptech/02/07/hot.chips.idg/
Implications to High Performance Computing

- 1.5x from transistor
- 670x from parallelism

- 8x from transistor
- 128x from parallelism

- 32x from transistor
- 32x from parallelism

- 50 GFLOPS/W
- ~100MW

- 1986: Giga
- 1996: Tera
- 2006: Peta
- 2016: Exa
- 2022

Graph showing relative transistor performance over time.
Challenges Scaling Systems to Higher Performance

CPU Intensive

Result: Excessive power requirements

Memory Intensive

Result: Slow Performance

IO Intensive

Result: Slow Performance (high latency)

Need to think about Compute Offload as well as Ingress/Egress Processing
Diverse Application Demands

Accelerators can increase Performance at lower TCO for targeted workloads

Intel estimates; bubble size is relative CPU intensity
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
Heterogeneous Computing Systems

Modern systems contain more than one kind of processor

- Applications exhibit different behaviors:
  - Control intensive (Searching, parsing, etc...)
  - Data intensive (Image processing, data mining, etc...)
  - Compute intensive (Iterative methods, financial modeling, etc...)

- Gain performance by using specialized capabilities of different types of processors
Separation of Concerns

Two groups of developers:

- Domain experts concerned with getting a result
  - Host application developers leverage 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
**FPGA Enabled Performance and Agility**

**Workload Optimization:**
ensure Xeon cores serve their highest value processing

**Efficient Performance:**
improve performance/watt

**Real-Time:**
high bandwidth connectivity and low-latency parallel processing

**Developer Advantage:**
code re-use across Intel FPGA data center products

FPGAs enhance CPU-based processing by *accelerating algorithms* and *minimizing bottlenecks*.
Using FPGAs Just Got Easier

- Orchestration / Rack Management
- User Application
- Software Frameworks
- Libraries
- Open Programmable Acceleration Engine (OPAE)
- OS Driver

![Diagram showing layers and interfaces related to FPGAs.]

- Increase Abstraction
- Increase User Base

- User Design
  - Accelerator Function (Loadable Workload)
  - FPGA Interface Manager (Standard I/O Interfaces)
  - Core Cache Interface

Programmable Accelerator Card
WHAT ACCELERATION STACK ENABLES

ECOSYSTEM OF FPGA WORKLOADS

APPLICATION & FPGA DEVELOPMENT

FPGA DEPLOYMENT & MANAGEMENT

Enabled by
## INTRODUCING THE ACCELERATION STACK FOR INTEL® XEON® CPU WITH FPGA

### Applications/Orchestration

<table>
<thead>
<tr>
<th>Orchestration / Rack Level Management</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Applications</td>
</tr>
<tr>
<td>Intel Xeon FPGA Acceleration Libraries</td>
</tr>
</tbody>
</table>

### Vertical Software Frameworks/Libs

(DL, Networking, Genomics, etc.)

<table>
<thead>
<tr>
<th>Frameworks/Libs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® DAAL</td>
</tr>
<tr>
<td>Intel® MKL</td>
</tr>
<tr>
<td>Intel® MKL-DNN</td>
</tr>
</tbody>
</table>

### Common Infrastructure

✓ *Simplify FPGA programming model*

<table>
<thead>
<tr>
<th>FPGA HW &amp; SW Tool Chains</th>
<th>Open Programmable Acceleration Engine (OPAE Software API)</th>
<th>Operating Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drivers, virtualization, API's, acceleration engine</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intel FPGA SDK for OpenCL™, Intel Quartus® Prime</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### FPGA Images

<table>
<thead>
<tr>
<th>FPGA Interface Manager (FIM)</th>
<th>Loadable AF image (.gbs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP Libraries: DLA, GEMM, VirtIO, pHMM Compression, Encryption, etc.</td>
<td></td>
</tr>
</tbody>
</table>

### Hardware

<table>
<thead>
<tr>
<th>FPGA Platforms (Programmable Acceleration Cards)</th>
</tr>
</thead>
</table>

### Hardware

FPGA Platforms (Programmable Acceleration Cards)
Orchestrating FPGA-accelerated applications

Orchestration Software (FPGA Enabled)

Public and Private Cloud Users

Launch Workload

Place Workload

Resource Pool

Storage  Network  Compute

Static/Dynamic FPGA Programming

Software-Defined Infrastructure

IP Repository

End User Developed IP

Intel Developed IP

3rd-Party Developed IP

Workload Accelerators

Workload

Workload 1

Workload 2

Workload N
Components of Acceleration Stack: Overview

- **Application**
  - Developed by User (Domain Expert)
  - Intel® Xeon® Software

- **Libraries**
  - User, Intel, and 3rd Party (Tuning Expert)

- **Drivers**
  - PCIe* Drivers Provided by Intel
  - Open Programmable Acceleration Engine (OPAE) Provided by Intel

- **Signal Bridge and Management**
  - FPGA Interface Manager Provided by Intel
  - User, Intel, or 3rd-Party IP Plugs into AF Slot (Tuning Expert)

- **Accelerator Functional Unit (AFU)**
  - User, Intel, or 3rd-Party IP Plugs into AF Slot (Tuning Expert)
  - Qualified and Validated for volume deployment Provided by OEMs

- **FPGA Platforms (Programmable Acceleration Cards)**
  - Provided by OEMs
Programmable Acceleration Card with Arria 10 FPGA

- Low-profile (half-length, half height) PCIe® slot card
- 168 mm × 56 mm
- Maximum component height: 14.47 mm
- PCIe × 16 mechanical

- 128 MB Flash
- For storage of FPGA configuration
- Board Management Controller (BMC)
- Server class monitor system
- Accessed via USB or PCIe

- 2 – Banks of DDR4-2133 SDRAM, 4 GB each
- 64 bit data, 8 bit ECC
- Total 8 GB

- USB 2.0 port for board firmware update and FIM image recovery

Intel® Arria™ 10 Accelerator Card

- QSFP 4x 10GbE Networking Interface
- Powered from PCIe+12V rail
- 70 W total board power
- 45 W FPGA power
- PCIe X 8 Gen3 connectivity to Xeon host

- 4GB DDR4 w/ECC
- 4GB DDR4 w/ECC
Current Capabilities of Arria 10 PAC

Dual SDRAM interfaces
- AFU exposed a two 512-bit interface operating at 267MHz offering ~34GB/s

Multiple FIM driven clocks
- Fixed 400/200/100 MHz frequencies
- Programmable user and user/2 frequencies (defined in .json file or at runtime)

Core Cache Interface providing host → FPGA and FPGA → host connectivity
- Includes five channels all operating at 400MHz
- Allows for 51GB/s peak bandwidth but PCIe will limit sustained throughput to ~6.4GB/s
COMPONENTS OF **ACCELERATION STACK:** OPEN PROGRAMMABLE ACCELERATION ENGINE

Developed by User (Domain Expert)

Software

PCle* Drivers Provided by Intel

Intel® Xeon®

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

Open Programmable Acceleration Engine (OPAE) Provided by Intel

Hardware

FPGA

FPGA Interface Manager Provided by Intel

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

Simplifies the use of FPGAs

FPGA Platforms (Programmable Acceleration Cards)

Application

Libraries

Drivers

Accelerator Functional Unit (AFU)

Signal Bridge and Management

Provided by Intel

PCIe®* Drivers

Provided by Intel

Open Programmable Acceleration Engine (OPAE)

Provided by Intel

Libraries

FPGA

User, Intel, and 3rd Party

(Tuning Expert)

User, Intel, or 3rd-Party IP

Plugs into AFU Slot

(Tuning Expert)
OPAE: SIMPLIFIED FPGA PROGRAMMING MODEL FOR APPLICATION DEVELOPERS

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

Start developing for Intel FPGAs with OPAE today: http://01.org/OPAE
From the OS’s point of view:

- FPGA hardware appears as a regular PCIe device
- FPGA accelerator appears as a set of features accessible by software programs running on host

Unified C API model:

- Supports different kinds of FPGA integration and deployment. (E.g: A single application can use the FPGA to accelerate certain algorithms)
- Resource management and orchestration services in a data center use to discover and select the FPGA resources and organize them to be used by the workloads
Accessing FPGA Accelerators as Physical or Virtual Functions

Architecture supports Single Root I/O Virtualization (SROIV) PCIe extension enabling host software to access the accelerator:

- Via a hypervisor/VMM (Virtual Function)
- Bypassing the VMM/Hypervisor Physical Functions

PCIe SR-IOV makes one physical device appear as multiple virtual devices

- The physical device is referred to as Physical Function (PF)
- Fully featured PCIe functions can be discovered, managed, and manipulated like any other PCIe device
- Creates Virtual Functions (VFs) which can be used to assign individual accelerators to virtual machines
FPGA Driver Architecture

FME: FPGA Management Engine Driver
- Static circuits for power/thermal management, reconfiguration, debugging, error reporting, performance counters, etc.

Port:
- Interface between the static (FIM) and the reconfigurable Acceleration Functional Unit (AFU) region
- Controls communication from software to the accelerator
- Expose features such as reset and debug
- There may be multiple ports exposed through a VF

AFU: Accelerator Functional Unit Driver
- Exposes a 256KB region as control registers through Port
- Reconfigurable circuits for application specific functions
- User process can share memory buffers with AFU
**HOW HOST APPLICATIONS ENUMERATE THE FPGA DEVICE: SYSFS**

Ex:

2 Intel(R) FPGA devices are installed in the host

Each FPGA device has one FME and one Port (AFU)
SOFTWARE APPLICATION DEVELOPMENT FLOW FOR ACCELERATION STACK
Software Application Development

**HDL Programming**
- ASE from Intel
- OPAE from Intel
- Intel® Quartus Prime Pro
- Intel® HLS Compiler
- C
- HDL
- Syn. PAR
- AFU Image
- SW Compiler
- exe
- AFU
- Simulation Environment (ASE)
- CPU
- FPGA

**OpenCL Programming**
- Intel® FPGA SDK for OpenCL™
- Host
- Kernels
- SW Compiler
- exe
- AFU Image
- OpenCL Emulator
- Application
- OPAE Software
- FIM
- CPU
- FPGA
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.
Application Development with OPAE

User application
myapp.c

links against

OPAE C library
(or, ASE OPEA C library)

interacts with

OPAE Intel FPGA driver
(or, RTL simulator)

includes

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
The OPAE Library Programming Model

Properties Object → Discover / search resource

Object model → Token Object → Acquire ownership of resource

→ Handle Object → Map AFU registers to user space

→ Allocate / define shared memory space → Start / stop computation on AFU and wait for result

→ Deallocate shared memory

→ Unmap MMIO

→ Relinquish ownership

→ Reconfigure AFU
The OPAE Object Model

- **Properties**
  - fpga_properties
  - describes a resource
  - query information about a resource

- **Token**
  - fpga_token
  - identifies a resource
  - enumerate resources based on criteria

- **Handle**
  - fpga_handle
  - signifies ownership
  - acquire ownership of a resource
fpga_properties Object

An opaque type for a properties object used by application to query and search for appropriate resources

2 Object types for FPGA resources

- **FPGA_DEVICE**
  - Corresponds to physical FPGA device
  - Can invoke management functions

- **FPGA_ACCELERATOR**
  - Represents an instance of an AFU

Defined in `types.h` file

<table>
<thead>
<tr>
<th>Property</th>
<th>FPGA*</th>
<th>Accelerator*</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent</td>
<td>No</td>
<td>Yes</td>
<td><code>fpga_token</code> of the parent object</td>
</tr>
<tr>
<td>ObjectType</td>
<td>Yes</td>
<td>Yes</td>
<td>The type of the resource: either FPGA_DEVICE or FPGA_ACCELERATOR</td>
</tr>
<tr>
<td>Bus</td>
<td>Yes</td>
<td>Yes</td>
<td>The bus number</td>
</tr>
<tr>
<td>Device</td>
<td>Yes</td>
<td>Yes</td>
<td>The PCI device number</td>
</tr>
<tr>
<td>Function</td>
<td>Yes</td>
<td>Yes</td>
<td>The PCI function number</td>
</tr>
<tr>
<td>SocketId</td>
<td>Yes</td>
<td>Yes</td>
<td>The socket ID</td>
</tr>
<tr>
<td>DeviceId</td>
<td>Yes</td>
<td>Yes</td>
<td>The device ID</td>
</tr>
<tr>
<td>NumSlots</td>
<td>Yes</td>
<td>No</td>
<td>Number of AFU slots available on an FPGA_DEVICE resource</td>
</tr>
<tr>
<td>BBSID</td>
<td>Yes</td>
<td>No</td>
<td>The FPGA Interface Manager (FIM) ID of an FPGA_DEVICE resource</td>
</tr>
<tr>
<td>BBSVersion</td>
<td>Yes</td>
<td>No</td>
<td>The FIM version of an FPGA_DEVICE resource</td>
</tr>
<tr>
<td>VendorId</td>
<td>Yes</td>
<td>No</td>
<td>The vendor ID of an FPGADEVICE resource</td>
</tr>
<tr>
<td>Model</td>
<td>Yes</td>
<td>No</td>
<td>The model of an FPGA_DEVICE resource</td>
</tr>
<tr>
<td>LocalMemorySize</td>
<td>Yes</td>
<td>No</td>
<td>The local memory size of an FPGA_DEVICE resource</td>
</tr>
<tr>
<td>Capabilities</td>
<td>Yes</td>
<td>No</td>
<td>The capabilities of an FPGA_DEVICE resource</td>
</tr>
<tr>
<td>GUID</td>
<td>Yes</td>
<td>Yes</td>
<td>The Global Unique Identifier of an FPGA_DEVICE or FPGA_ACCELERATOR resource</td>
</tr>
<tr>
<td>NumMMIO</td>
<td>No</td>
<td>Yes</td>
<td>The number of MMIO space of an FPGA_ACCELERATOR resource</td>
</tr>
<tr>
<td>NumInterrupts</td>
<td>No</td>
<td>Yes</td>
<td>The number of interrupts of an FPGA_ACCELERATOR resource</td>
</tr>
<tr>
<td>Accelerator State</td>
<td>No</td>
<td>Yes</td>
<td>The state of an FPGA_ACCELERATOR resource: either FPGA_ACCELERATOR_ASSIGNED or FPGA_ACCELERATOR_UNASSIGNED</td>
</tr>
</tbody>
</table>
Creating `fpga_properties` object

Initializing memory pointed at by `prop` to represent properties object

Populates with properties of the resource referred to by `token`

- Passing NULL `token` creates empty properties object which would match all FPGA resources in the enumeration query
- Refine query criteria using `fpgaPropertiesSet*()` functions

Individual properties can be queried using `fpgaPropertiesGet*()` accessor functions

Destroy `fpga_properties` object using `fpgaDestroyProperties()` function

Located in `properties.h` file

* Target property from `fpga_properties` object (i.e. `fpgaSetObjectType()`)
Function that searches for FPGA resources in system that match criteria (may be more than one)

- All accelerators assigned to a host interface, all FPGAs of a specific type, etc.

Creates fpga_token objects and populates the array with these tokens

- Number of tokens in the returned tokens array, either max_tokens or num_matches whichever is smaller

Free the memory with tokens no longer needed using the fpgaDestroyToken() function

Located in enum.h file
Receive **fpga_handle**

**fpga_result fpgaOpen**(*fpga_token token, fpga_handle *handle, int flags*)

- Acquires ownership of FPGA resource referenced by *token*
- Ownership required to interact with accelerator function
- Remains open until *fpga_Close()* function called or process terminates
  - Can also reset accelerator using *fpga_Reset()* function

*Located in access.h file*
Programming Solutions Group

**Enumeration and Discovery**

```c
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

```c
fpga_token token;
// ... enumeration ...

fpga_handle handle;

fpgaOpen(token, &handle, 0);
```

```
fpgaClose(handle);
```
Memory-Mapped I/O

fpgaMapMMIO(..., &mmio_ptr)

fpgaReadMMIO()
fpgaWriteMMIO()
Mapping MMIO Space

Provides access to control registers through memory mappable address spaces (MMIO spaces)

Returns `mmio_ptr` to specified MMIO space of target object in process virtual memory

- Setting to `mmio_ptr` to NULL implies access will be performed through `fpgaReadMMIO*()` and `fpgaWriteMMIO*()`
  - Only supported mode by AFU Simulation Environment (ASE)
- After mapping can access through direct pointer operations

Unmap with `fpgaUnmapMMIO()` function

Located in `mmio.h` file
MMIO Read/Write

Performs 64bit MMIO space write of value to specified byte offset
- Also supports 32 bit MMIO writes using `fpgaWriteMMIO32()` function

Reads use `fpgaReadMMIO64()` or `fpgaReadMMIO32()` functions

Located in `mmio.h` file
Shared Memory

Available on Intel® Xeon®+FPGA multi-chip packages with UPI interface

FPGA_DEVICE

FPGA_ACCELERATOR

AFU_ID: 0xabcdef

DDR Memory

shared buffer (“workspace”)

Physically contiguous!

fpgaPrepareBuffer(..., len, ..., 0); // Allocated by lib
fpgaPrepareBuffer(..., len, ..., FPGA_BUF_PREALLOCATED); Pre-allocated
Sharing System Memory with an Accelerator

Prepares a memory buffer for shared access between accelerator and calling process

- Buffer can be pre-allocated or created dynamically

When finished with buffers release with `fpgaReleaseBuffer()` function

Located in `buffer.h` file
Getting Base IO Address of Buffer

- **Objective**: Acquire the physical base address for a shared buffer identified by the workspace ID.

- **Function**: `fpgaGetIOAddress`

  ```c
  fpga_result fpgaGetIOAddress(fpga_handle handle, uint64_t wsid, uint64_t *ioaddr)
  ```

- **Parameters**:
  - `fpga_result` represents the error code.
  - `handle`: Handle of previously opened accelerator resource.
  - `wsid`: Buffer Workspace ID.
  - `ioaddr`: Pointer to memory where IO Address will be returned.

**Usage**: Used to acquire the physical base address for a shared buffer identified by the workspace ID.

- **IO Virtual Address (IOVA)**

Located in `buffer.h` file.
Shared Memory

```c
fpga_handle handle;        /* handle to accelerator */
void  *buf_ptr;
uint64_t io_addr;
uint64_t wsid;
uint64_t length = 1 * 1024 * 1024; /* 1 MiB */

/* Allocate and share buffer */
fpgaPrepareBuffer(handle, length, &buf_ptr, &wsid, 0);

/* Get IO address to be used by accelerator (share via MMIO) */
fpgaGetIOAddress(handle, wsid, &io_addr);

/* ... */

/* Release and deallocate shared buffer */
fpgaReleaseBuffer(handle, wsid);
```
Shared Memory (Pre-allocated)

```c
fpga_handle handle;     /* handle to accelerator */
void   *buf_ptr;
uint64_t io_addr;
uint64_t wsid;
uint64_t length = 1 * 1024 * 1024; /* 1 MiB */

/* Pre-allocate buffer */
buf_ptr = allocate_your_own_buffer(length); /* must be physically contiguous! */

fpgaPrepareBuffer(handle, length, &buf_ptr, &wsid, FPGA_BUF_PREALLOCATED);

/* Get IO address to be used by accelerator (share via MMIO) */
fpgaGetIOAddress(handle, wsid, &io_addr);

/* ... */

/* Release and deallocate shared buffer */
fpgaReleaseBuffer(handle, wsid);
```
Management and Reconfiguration

SW application (with admin privilege)

fpgaReconfigureSlot(..., buf, len, 0)

libopae-c

Partial configuration

GBS file

xyz.gbs

interface_id

AFU_id

Storage

GBS metadata

Programmable Solutions Group
Management and Reconfiguration

```
fpga_handle handle;      /* handle to device */
FILE   *gbs_file;
void    *gbs_ptr;
size_t  gbs_size;

/* Read bitstream file */
gbs_ptr = malloc(gbs_size);
fread(gbs_ptr, 1, gbs_len, gbs_file);

/* Program GBS to FPGA */
fpgaReconfigureSlot(handle, 0, gbs_ptr, gbs_size, 0);
/* ... */
```
A Code Example - Put Everything Together

“Loopback” is the default AFU on each port of an Intel® FPGA
  – Copy memory content between host and FPGA

The hello_afu.c code in the $OPAEPLATFORM_ROOT/hw/samples directory of the OPAE library
  – Demonstrates all OPAE API functions discussed in this presentation
  – The same flow can be used to access and exercise any other AFUs

To compile source code run appropriate gcc/make commands
EXERCISE 1

Compiling Host Application
OPAE Tools

- **fpgaconf**
  - A tool for partial reconfiguration: take an acceleration configuration bitstream and use it to overwrite a reconfigurable region of the FPGA

- **fpgadiag**
  - Several diagnostic tests, such as loopback, bandwidth, latency, etc

- **fpgainfo**
  - Shows the current status of FPGA hardware, such as power, temp, and errors

- **fpgad**
  - A daemon to monitor FPGA drivers' error status; report errors as events to OPAE

- **fpgaflash**
  - Updates the static FIM image loaded from flash at power on
OpenCL Development Approach

OpenCL™ Programming

Standard C/C++ Compiler linked with
- Intel® FPGA OpenCL Library
- OpenCL Support Package for Acceleration Stack Library
- OPAE Library

Intel® FPGA Kernel Compiler (aoc)
- Compiles OpenCL kernel code
- Generates RTL
- Invokes Intel® Quartus Prime Design Software

FPGA Bitstream(<file>.aocx)
- Contains AF .gbs file
- And other OpenCL kernel related information
OpenCL™ Programming Model: Host + Accelerators

Host program on CPU

Functions to be accelerated (kernels) offloaded onto accelerator devices (FPGAs, GPUs, etc)

```c
main()
{
    read_data_from_file( ... );
    manipulate_data( ... );

    clEnqueueWriteBuffer( ... );
    clEnqueueNDRange( ..., sum, ... );
    clEnqueueReadBuffer( ... );

    display_result( ... );
}

_kernel void
sum(__global float *a,
    __global float *b,
    __global float *y)
{
    int gid = get_global_id(0);
    y[gid] = a[gid] + b[gid];
}
```

OpenCL on Intel® FPGAs landing page
Software Application Flow using OpenCL™

No different from traditional OpenCL™ flow

- C based development and optimization flow to create AFUs and Host Application
- Standard OpenCL™ FPGA application using the Intel® FPGA SDK for OpenCL
  - FPGA OpenCL™ debug and profiling tools supported
- More information on using OpenCL with FPGAs

The Acceleration Stack abstracted away from user

- OPAE part of the Host Run-Time
  - Host does not need to interact with OPAE SW directly
- OpenCL™ Support Package(OSP) part of the FPGA Interface Manager
  - Kernel Avalon interface translated to CCI-P by the OSP

To learn more about using OpenCL with FPGAs, visit Intel FPGA Customer Training page
OpenCL™ Adds HW and SW Abstraction

OpenCL Host
- OpenCL Runtime
- OSP MMD
- OPAE
- CPU

Software Stack

OpenCL Interface Manager (BBS)

FPGA Interface Unit

PCIe*

CCI-P

External Memory Interface

DDR

AFU (GBS)

OpenCL Support Package HW

OpenCL Kernel

OPAE

OSP MMD

DDRR

Programmable Solutions Group
Software Stack and the OpenCL™ MMD Layer

Memory-Mapped Device (MMD) SW connects board driver to OpenCL runtime

- For the Acceleration Stack it links the OpenCL API to OPAE

Uses DMA BBB software driver unmodified and runs it in a separate thread.
OpenCL™ with Acceleration Stack Features

Standard OpenCL FPGA application can be used as is for the Acceleration Stack

- No code changes necessary, just switch the OSP in the compilation environment
  - Acceleration Stack functionality built into the OpenCL Support Package

Easily leverage the capabilities of the Acceleration Stack

- Allow OpenCL to be used with virtualization
- No need develop RTL conforming to CCI-P interface
- No need to code at the lower OPAE level

Switch between OpenCL and Non-OpenCL Acceleration Stack applications without rebooting

- FPGA Interface Manager for both are the same
OpenCL Usage Software Requirements

Intel® FPGA Runtime Environment for OpenCL™

- Download and install from Intel® FPGA Download Center
- For kernel development and compilation, the Intel® FPGA SDK for OpenCL and Intel® Quartus software is required

Acceleration Stack for the Intel® PAC

- Follow the Intel® Acceleration Stack Quick Start Guide for Intel® PAC
- Install OpenCL™ Support Package for the Intel® PAC
  - Part of the Acceleration Stack files
    - OpenCL™ on Intel Programmable Acceleration Card with Intel® Arria 10 GX FPGA Quick Start User Guide
ACCELERATOR FUNCTIONAL UNITS (AFU)
Components of Acceleration Stack: FPGA Interface Manager (FIM) + AFU

- **FPGA Interface Manager (FIM)** provided by Intel
- **AFU**
- **Applications**
- **Libraries**
- **Drivers**
- **Software**
- **FPGA**
- **Intel® Xeon®**
- **User, Intel, and 3rd Party (Tuning Expert)**
- **Open Programmable Acceleration Engine (OPAE)** provided by Intel
- **PCIe® Drivers** provided by Intel

- **Simplifies the use of FPGAs**

- **FPGA Platforms (Programmable Acceleration Cards)**

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

- **Developed by User (Domain Expert)**
- **Hardware**
- **PCle* Drivers Provided by Intel**
- **Libraries Provided by Intel**
- **user, Intel, and 3rd Party (Tuning Expert)**

- **Developed by User (Domain Expert)**
- **Software**
- **Intel® Xeon®**
- **Open Programmable Acceleration Engine (OPAE)** provided by Intel
- **PCIe® Drivers** provided by Intel
- **Libraries Provided by Intel**

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

- **Simplifies the use of FPGAs**

- **FPGA Platforms (Programmable Acceleration Cards)**

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

- **Open Programmable Acceleration Engine (OPAE)** provided by Intel
- **PCIe® Drivers** provided by Intel
- **Libraries Provided by Intel**
Where to Get AFUs?

Self-Developed

- Higher Productivity
- C/C++ Programming Language
  - Intel® HLS Compiler
  - Intel® FPGA SDK for OpenCL™
- Performance Optimized
  - VHDL or Verilog

Externally-Sourced

- Intel® Reference Designs
- Contracted Engagement
  - Ecosystem Partner
Growing List of Accelerator Solution Partners

Easing Development and Data Center Deployment of Intel FPGAs For Workload Optimization
GENOMICS SEQUENCING: ACCELERATING PERFORMANCE

50X PAIRHMM ALGORITHM SPEEDUP\(^1\)

1.2X OVERALL PIPELINE SPEEDUP\(^1\)

1. Test configuration: Intel® Xeon® processor E5-2699 v4 at 2.20 GHz, 2 sockets, 22 cores/socket, 256 GB RAM, 2 TB Intel SSD DC P3700, Intel Arria® 10 GX Development Kit compared to Intel Xeon processor E5-2699 v4 at 2.20 GHz with Intel® Advanced Vector Extensions (AVX), 2 sockets, 22 cores/socket, 256 GB RAM, 2 TB Intel SSD DC P3700.
SOLVING REAL-WORLD PROBLEMS: DATABASE ACCELERATION

10X+ FASTER REAL-TIME DATA ANALYTICS

2X+ TRADITIONAL DATA WAREHOUSING

3X+ STORAGE COMPRESSION

>40% TCO SAVINGS

1. Based on database queries run with SWARM64 acceleration vs. no acceleration. Testing performed by Swarm64. See System Configurations page for more details.
2. Data warehousing tested with queries and data taken from TPC-DS benchmark. Testing performed by Swarm64. See System Configurations for more details.
3. Based on database size run with SWARM64 acceleration vs. no acceleration. Testing performed by Swarm64. See System Configurations page for more details.
4. Projected Total Cost of Ownership savings for Swarm64DB over PostgreSQL database over a 3 year period. Swarm64 estimate. See System Configurations page for more details.
Key Value Store (KVS) associates values with keys. Algo-Logic's FPGA-accelerated KVS solution performs lookups with the lowest latency, highest throughput, and less processing energy than equivalent software.

- Ultra low latency (sub microsecond network read delay)
- Deterministic (Near-zero jitter)
- High Throughput (170M IOPs)
- Sub μ-Joule/lookup energy consumption

Software Controller API Options
- KVS client API compatible with C/C++, Java, Python, and other programming languages

† solutions compared were implementations of the same KVS protocol running in software in Linux on the same Intel i7 processor on the same machine and another running also in software but with the DPDK kernel bypass library. Details about the test appear in the whitepaper http://algo-logic.com/KVS-whitepaper. See system configuration slide for more detail on configuration and testing methodology.
IMAGE PROCESSING: ACCELERATING PERFORMANCE

4.9X FASTER JPEG TO WEBP

5X LOWER LATENCY

10X LOWER POWER THAN GPU

† Compared to E5-2630 v2 CPU, JPEG to WEBP. See System Configuration Slide for more details
CASE STUDY: 8X SPARK SQL / KAFKA PERFORMANCE INCREASE

**Customer Application:** Big Data Applications running on Spark/Kafka Platforms

**Current solution:** Run Spark/SQL on a cluster of CPUs

**Challenge:** For many applications in the FinServ/Genomics/Intelligence Agencies/etc. Spark performance does not meet customers SLA requirements, especially for delay sensitive streaming workloads

**Solution**

**Value Proposition**

Performance - Accelerate Spark SQL/Kafka by 8x
Ease of Use – Zero Code Change
Scalability - Hardware Agnostic
Lower TCO
CASE STUDY: 5X RISK ANALYTICS PERFORMANCE INCREASE

Customer Application: Risk Management acceleration framework (financial back-testing)

Current solution: Deploy a cluster of CPUs or GPUs with complex data access

Challenge: Traditional risk management methods are compute intensive, time consuming applications - > 10+ hours for financial back-testing

Solution Value Proposition

>5x Performance Improvement
Perform Risk and Pricing Calculations Simultaneously
Abstraction - Integrated Solution with Apache Spark, SSD Access and FPGA Implementation
Intel® FPGA Deep Learning Acceleration Suite

- CNN acceleration engine for common topologies executed in a graph loop architecture
  - AlexNet, GoogleNet, LeNet, SqueezeNet, VGG16, ResNet, Yolo, SSD, LSTM...

- Software Deployment
  - No FPGA compile required
  - Run-time reconfigurable

- Customized Hardware Development
  - Custom architecture creation w/ parameters
  - Custom primitives using OpenCL™ flow
Open Visual Inference & Neural network Optimization (OpenVINO™) toolkit

<table>
<thead>
<tr>
<th>Deep Learning Deployment Toolkit</th>
<th>OpenCV*</th>
<th>Optimized Libraries &amp; OpenVX*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cross-platform approach to deep learning inference</td>
<td>Optimized functions for Intel® Processors</td>
<td>Runtimes, emulator, kernels, workload samples</td>
</tr>
<tr>
<td>Model Optimizer: Convert &amp; optimize trained models</td>
<td>Inference Engine: Run optimized inferences</td>
<td>Create own customer kernels or use a library of functions</td>
</tr>
<tr>
<td>Hardware Support: GPU, FPGA, CPU</td>
<td></td>
<td>Enhanced, graphical development using Vision Algorithm Designer</td>
</tr>
</tbody>
</table>

Deep Learning Frameworks

Deep Learning Tools from Intel

Optimized Libraries & OpenVX*

OpenCL™ Driver for Intel® Architecture

OpenCV and the OpenVX logo are trademarks of the Khronos Group Inc. OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission by Khronos.
Intel® FPGA DLA Suite Usage

- Supports common software frameworks (Caffe, Tensorflow)
- Intel DL software stack provides graph optimizations
- Intel FPGA Deep Learning Acceleration Suite provides turn-key or customized CNN acceleration for common topologies
Machine Learning on Intel® FPGA Platform

Acceleration Stack Platform Solution

Software Stack
- Application
- DL Deployment Toolkit
- DLA Runtime Engine
- OpenCL™ Runtime
- Acceleration Stack

Hardware Platform & IP
- DLA Workload
- BBS

Intel® Xeon CPU

PAC Family Boards

ML Framework (Caffe*, TensorFlow*)
DEVELOPING, DEBUGGING AND SIMULATING AN ACCELERATOR FUNCTION
FPGA Components (Acceleration Stack v1.1)

- PCIe*
- FPGA Interface Unit (FIU)
- High Speed Serial Interface** (HSSI)
- Core Cache Interface (CCI)
- 10Gb/40Gb
- Accelerator Functional Unit (AFU)
- EMIF
- EMIF
- Local Memory Interfaces
- DDR4
- DDR4

- Could be other interfaces in the future (e.g. UPI)
- ** Available in v1.1 of Acceleration Stack
FPGA INTERFACE MANAGER (FIM): UNDER THE HOOD

Standard framework and abstraction layer for AFU integration with Acceleration Stack

FPGA INTERFACE MANAGER (FIM)

FPGA INTERFACE UNIT (FIU)

PCIe Gen 3x8 Hard IP Controller

CCI-P (512-bit Bidirectional Data Path)

ACCELERATOR FUNCTIONAL UNIT SLOT (Partial Reconfiguration region)

Resources Available:
ALMs: 92%
M20KBlocks: 94%
DSP Blocks: 100%

MAC IP
40Gb x1 or 10Gb x4

FPGA Manager Engine (FME)

SDRAM Bank 0 Interface

AV MM Master
AV MM Slave

267 MHz 512-Bit

1067 MHz 64-Bit ECC

SDRAM Bank 1 Interface

AV MM Master
AV MM Slave

267 MHz 512-Bit

1067 MHz 64-Bit ECC

HSSI PHY (PCS/PMA)

DIMM 0
DIMM 1
QSFP +
FPGA INTERFACE UNIT: THE LOGIC THAT ENABLES THE ACCELERATION STACK

FIU provides handling of protocol errors

- Bus timeouts, signaling violation, buffer overruns, etc...

FIU implements bus scheduling and synchronization

- FPGA write synchronization handled within FIU to ensure predictable ordering of writes to host memory

Programmable Accelerator Card with Arria 10 GX FPGA Host Interface Bandwidth (PCIe gen3 x8)

- Read from host memory: 6.911GB/sec
- Write to host memory: 6.955GB/sec
- Simultaneous read/write: 6.160GB/sec read, 6.263GB/sec write
How to program the FIM into the flash?

fpgaflash through PCIe

```
$ sudo fpgaflash user $OPAE_PLATFORM_ROOT/hw/blue_bits/*.rpd 04:00.0
```

Quartus Prime Programmer through JTAG

- Intel FPGA Download Cable is required

```
$QUARTUS_HOME/bin/quartus_pgm -m JTAG -o 'pvbi;dcp_1_1.jic'
```
How to find the version of FIM loaded?

```bash
$ sudo fpgainfo fme
```

//****** FME ******
Class Path: /sys/class/fpga/intel-fpga-dev.0/intel-fpga-fme.0
Device Path: /sys/devices/pci0000:00/0000:00:03.0/0000:04:00.0/fpga/intel-fpga-dev.0/intel-fpga-fme.0
Bus: 0x04
Device: 0x00
Function: 0x00
Device Id: 0x09C4
FIM Version: 1.1.3
Ports Num: 1
Socket Id: 0
Bitstream Id: 0x113000200000177
Bitstream Metadata: 0x18043013
Pr Interface Id: 9926ab6d-6c92-5a68-aabc-a7d84c545738
Object Id: 251658240 (FPGA DEVICE)
How to find the version of AFU loaded?

$ sudo fpgainfo port

****** PORT *****

Class Path : /sys/class/fpga/intel-fpga-dev.0/intel-fpga-port.0
Device Path : /sys/devices/pci0000:00/0000:00:03.0/0000:04:00.0/fpga/intel-fpga-dev.0/intel-fpga-port.0
Bus : 0x04
Device : 0x00
Function : 0x00
AFU Id : 850adcc2-6ceb-4b22-9722-d43375b61c66
Object Id : 0xf400000 FPGA_ACCELERATOR
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

Externally-Sourced

- Intel® Reference Designs
- Contracted Engagement
  - Ecosystem Partner
Accelerator Function Development

HDL Programming

- ASE from Intel
- OPAE from Intel
- Intel® Quartus Prime Pro
- Intel® HLS Compiler
- C
- HDL
- SW Compiler
- Syn. PAR
- AFU Image
- exe
- AFU Simulation Environment (ASE)
- CPU
- FPGA
- Application
- OPAE Software
- FIM

OpenCL Programming

- Intel® FPGA SDK for OpenCL™
- OPAE Software
- FIM
- Host
- Kernels
- SW Compiler
- OpenCL Compiler
- exe
- AFU Image
- OpenCL Emulator
- CPU
- FPGA
- Application
- OPAE Software
- FIM
AFU Development Software Requirements

Acceleration Stack SDK
- Quartus Prime Pro Software 17.1.1 for v1.1 Acceleration Stack (17.0 for v1.0)
- 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 version 10.5c or higher
- 64-bit QuestaSim version 10.5c or higher
Overview of OPAE Platform for AFUs

OPAE Platform is an abstraction of a hardware platform for which AFUs are designed

- Enables generating AFs from AFUs designed for generic OPAE Platform for any OPAE compliant hardware

Platform Interface Manager (PIM) defines a generic OPAE platform for which AFU top-levels should be designed to ensure provisioning on multiple hardware platforms

- 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

Clocks
CCI-P
Power
Error
HSSI
Local Memory
AFU and CCI

Develop AFU with standard FPGA development tools

Interface with the acceleration stack through Core Cache Interface (version P)

- Provides a base platform memory interface
  - Simple request/response interface (Simple cacheline Read/Write)
  - Physical addresses (unified with x86 CPU)
  - Split transactions (replies matched to request using a tag)
  - No order guarantees

- These minimal requirements satisfy major classes of algorithms, e.g.:
  - Double buffered kernels that read from and write to different buffers
  - Streaming kernels that read from one memory-mapped FIFO and write to another

- Standardized interface abstracts workload away form potential changing hardware interfaces saving months of work
Core Cache Interface: Overview

CCI abstracts AFU from lower level PCIe and UPI protocols

Enables AFU to access host memory and respond to MMIO requests

Composed of 3 command and response channels

- 3 TX and 2 RX independent channels
  - Only TX channels 0 and 1 capable of backpressure

- 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

- MMIO Addressing is D-word aligned (4-byte) and AFU must support 4 or 8-byte MMIO Accesses

https://www.altera.com/documentation/buf1506187769663.html
CCI-P Signal Mapping

CH 0 – host memory read requests and responses as well as receiving MMIO requests from host
CH 1 – Host memory WR requests and responses
CH2 – MMIO read responses back to host
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

$ git clone https://github.com/OPAE/intel-fpga-bbb
Memory Properties Factory (MPF)

Provides a common collection of memory semantic extensions to CCI-P

Applications instantiate only the semantics they require

Each MPF block is implemented as a CCI-P to CCI-P shim

- Consume CCI-P requests
- Implement some feature (e.g. translate virtual addresses to physical)
- Produce transformed CCI-P requests

Application-specific memory hierarchies are formed by composing MPF shims
MPF Composable Shims

All MPF shims may be enabled or disabled independently and run at full speed:

- **VTP**: Virtual to physical address translation
- **ROB**: Reorder buffer to sort read responses and return them in request order
- **VC Map**: Map requests to system memory channels explicitly
- **WRO**: Intra-line write/read ordering
- **PWRITE**: Partial (masked) write emulation using read-modify-write

*Note: Some shims depend on other shims, e.g.:*

- **WRO** on **VC Map**
- **PWRITE** on **WRO**
# Example Designs to Get Started

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

Overview of hello_afu example AFU

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
  - The location of RTL source, .sdc constraint files, SignalTap .stp files, etc is your choice
- Host OPAE software application source in the ./sw directory
  - To perform the co-simulation environment
  - Can be located elsewhere, but leverage scripts and directory structure

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)
  - Created at compile time by provided build flow scripts
- Platform configuration file (.json)
- Build configuration file (.txt)
AFU RTL Source
Mandatory Source Files and Hierarchical Structure

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 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` (see next slides)
- Optional HSSI device interfaces (see .json file from 10Gb or 40Gb design examples)

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
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)

---

```vhdl
// User AFU goes here

afu afu_inst(
          .afu_clk(afu_clk),
          `ifndef INCLUDE_DDR4
            .DDR4 USERCLK(DDR4a_USERCLK),
            .DDR4_waitrequest(DDR4a_waitrequest),
            .DDR4_readdata(DDR4a_readdata),
            .DDR4_readdatavalid(DDR4a_readdatavalid),
            .DDR4_burstcount(DDR4a_burstcount),
            .DDR4_writedata(DDR4a_writedata),
            .DDR4_address(DDR4a_address),
            .DDR4_write(DDR4a_write),
            .DDR4_read(DDR4a_read),
            .DDR4_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_mmioc0Rx ( pck_cp2af_mmioc0Rx ),
          .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

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 2\textsuperscript{nd} 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 clean
- make USE_ASE=1
- ./hello_afu
AFU Simulation Environment (ASE)

Hardware software co-simulation environment for the Intel Xeon FPGA development

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

AFU Simulator Window (server)

Application SW Window (client)
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`
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
Provide Software Developer With a Specification

Memory mapped register space

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

<table>
<thead>
<tr>
<th>MMIO address</th>
<th>bytes (OPAE)</th>
<th>words (CCI-P)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFU header (read-only)</td>
<td>63</td>
<td>0</td>
</tr>
<tr>
<td>0x0000</td>
<td>0x0000</td>
<td>GUID</td>
</tr>
<tr>
<td>0x0008</td>
<td>0x0002</td>
<td>AFU ID_L</td>
</tr>
<tr>
<td>0x0010</td>
<td>0x0004</td>
<td>AFU ID_H</td>
</tr>
<tr>
<td>0x0018</td>
<td>0x0006</td>
<td>NEXT_DFH</td>
</tr>
<tr>
<td>0x0020</td>
<td>0x0008</td>
<td>Reserved</td>
</tr>
<tr>
<td>CSR (read/write)</td>
<td>0x080</td>
<td>0x0020</td>
</tr>
<tr>
<td></td>
<td>0x0000</td>
<td>Global Unique ID</td>
</tr>
<tr>
<td></td>
<td>0x0002</td>
<td>AFU ID (low 64 bits)</td>
</tr>
<tr>
<td></td>
<td>0x0004</td>
<td>AFU ID (high 64 bits)</td>
</tr>
<tr>
<td></td>
<td>0x0006</td>
<td>Pointer to next DFH</td>
</tr>
<tr>
<td></td>
<td>0x0008</td>
<td>Reserved space</td>
</tr>
</tbody>
</table>

Device Feature Header
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
Tune Results with Architectural Exploration

Intel® HLS Compiler optimization directives can be used to tune results.

**Design Tuning Knobs**

<table>
<thead>
<tr>
<th>Directive</th>
<th>Optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop Rolling</td>
<td>Area vs Performance</td>
</tr>
<tr>
<td>ii</td>
<td>Throughput</td>
</tr>
<tr>
<td>Clock</td>
<td>Fmax</td>
</tr>
<tr>
<td>Memory</td>
<td>Map arrays into device memories</td>
</tr>
<tr>
<td>Interface</td>
<td>Memory mapped, streaming, wire</td>
</tr>
</tbody>
</table>

Goal: Same performance as hand-coded RTL with 10-15% more resources.
HLS Use Model

C/C++ Code

Standard gcc/g++ Compiler

HLS Compiler

100% Makefile compatible

EXE

HDL IP

src.c

i++ <options>

a.exe

lib.h

Intel® Quartus® Ecosystem

PCle

FIM

CCIP

AFU

External Memory Interface
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
EXERCISE 2

Simulate AFU with Host Application and Generate AF
Out-of-Box Flow for Acceleration Stack

1. **Deployment Flow**
   - **Buy Server w/ PAC**
   - **Install Supported OS**
   - **Download & Install Deployment Package of Acceleration Stack**

2. **Development Flow**
   - **Download & Install Developer Package of Acceleration Stack**
   - **Download & HLS or OpenCL (Optional)**

3. **Vendor Website**
   - **Vendor Website (i.e. Red Hat)**

4. **Intel Website**
   - **Intel Website**

5. **Supported OEM**
   - **Supported OEM Server Website (i.e. Dell)**

6. **Write Host Application**
   - **Create & Simulate Workload**
   - **Download & Install Simulator**
   - **Download & Install Workload**
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 for, 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.
GET STARTED TODAY BY DOWNLOADING THE ACCELERATION STACK!

Software developers who develop and integrate their host application with accelerator functions can download the **Acceleration Stack for Runtime**.

Accelerator function developers who design using RTL and OpenCL™ can download the **Acceleration Stack for Development**.

<table>
<thead>
<tr>
<th>Components</th>
<th>Acceleration Stack for Runtime</th>
<th>Acceleration Stack for Development</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>Software development of runtime host application</td>
<td>Accelerator function development using the Intel® Quartus® Prime Pro Edition Software and Acceleration Stack</td>
</tr>
<tr>
<td>Intel Acceleration Stack</td>
<td>Acceleration Stack 1.1</td>
<td>Acceleration Stack 1.1</td>
</tr>
<tr>
<td>OpenCL Software</td>
<td>Intel FPGA Runtime Environment (RTE) for OpenCL</td>
<td>Intel FPGA Software Development Kit (SDK) for OpenCL</td>
</tr>
<tr>
<td>Download Size</td>
<td>619 MB</td>
<td>16.9 GB</td>
</tr>
</tbody>
</table>

Download Page for the Acceleration Stack
Announcing Our First OEM Partner For Intel PAC

DATA CENTER SERVERS WITH FPGA ACCELERATORS AVAILABLE NOW!

More OEM providers coming soon!
Best Known Configurations (BKC)

<table>
<thead>
<tr>
<th>Dell-EMC Servers</th>
<th>Base Configurations</th>
<th>Processor Thermal Configuration</th>
<th>Minimum Memory</th>
<th>PCIe Riser</th>
<th>GPU / FPGA Acceleration Cards</th>
</tr>
</thead>
<tbody>
<tr>
<td>R540</td>
<td>Chassis with up to 8 x 2.5&quot; SAS/SATA hard drives and two PCIe slots</td>
<td>Heat sink install kit for CPU config, no cable</td>
<td>8GB RDIMM, 2,666 MT/s, single rank (Qty 2)</td>
<td>Riser config 6, 5 x 8, 5 x 16 slots, single-wide CPU compatible</td>
<td>Intel PAC 80W FH</td>
</tr>
<tr>
<td>R740</td>
<td>Chassis with up to 8 x 2.5&quot; SAS/SATA hard drives for 2-CPU configuration</td>
<td>Heat sink install kit for GPU config, no cable</td>
<td>8GB RDIMM, 2,666 MT/s, single rank (Qty 2)</td>
<td>Riser config 6, 5 x 8, 5 x 16 slots, single-wide CPU compatible</td>
<td>Intel PAC 80W FH</td>
</tr>
<tr>
<td>R740xd</td>
<td>Chassis with up to 24 x 2.5&quot; NVMe Express (NVMe) drives (max), 2-CPU configuration, GPU-capable configuration</td>
<td>Heat sink install kit for CPU config, no cable</td>
<td>8GB RDIMM, 2,666 MT/s, single rank (Qty 2)</td>
<td>Riser config 9, 5 x 8, 5 x 16 slots, single-wide CPU compatible</td>
<td>Intel PAC 80W FH</td>
</tr>
<tr>
<td>R840</td>
<td>Any chassis configuration with accelerator or GPU support</td>
<td>Heat sink install kit for GPU config, no cable</td>
<td>8GB RDIMM, 2,666 MT/s, single rank (Qty 2)</td>
<td>Riser config 6, 5 x 8, 5 x 16 slots, single-wide CPU compatible</td>
<td>Intel PAC 80W FH</td>
</tr>
<tr>
<td>R940xa</td>
<td>Any chassis configuration with accelerator or GPU support</td>
<td>Heat sink install kit for GPU config, no cable</td>
<td>8GB RDIMM, 2,666 MT/s, single rank (Qty 2)</td>
<td>Riser config 6, 5 x 8, 5 x 16 slots, single-wide CPU compatible</td>
<td>Intel PAC 80W FH</td>
</tr>
</tbody>
</table>

Intel® Arria® 10 PAC board qualified at Dell for variety of Dell servers

Only available in certain configurations of these servers

Refer to BKC guide to know how to configure

Choosing a BKC Allows PAC to Appear
Broadening Customer Fit - Hardware

We provide **qualified** systems to ensure compatibility

<table>
<thead>
<tr>
<th>Component</th>
<th>Degree of Flexibility</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPGA</td>
<td>A10 1150 GX-2</td>
<td>Only FPGA offered on Arria 10 PAC</td>
</tr>
<tr>
<td>PCIe Card</td>
<td>Arria 10 PAC</td>
<td>Only card that has been qualified at this point</td>
</tr>
<tr>
<td>Processor</td>
<td>Intel Broadwell &amp; Skylake; potentially more</td>
<td>VT-d &amp; VT-x extensions required</td>
</tr>
<tr>
<td>Server Platform</td>
<td>Purley</td>
<td>potentially more</td>
</tr>
<tr>
<td>Server System</td>
<td>1RU, 2RU &amp; 4RU Rack Servers</td>
<td>Compatibility guidelines planned for Q2 2018</td>
</tr>
<tr>
<td>Repurpose card?</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Memory</td>
<td>48GB RAM suggested</td>
<td>16 GB is sufficient if using only pre-compiled FPGA binaries</td>
</tr>
</tbody>
</table>
### We embrace an open ecosystem

<table>
<thead>
<tr>
<th>Component</th>
<th>Degree of Flexibility</th>
<th>Comments</th>
</tr>
</thead>
</table>
| OS                    | RHEL & CentOS 7.4 (3.10 or 4.12 Kernel) | Upstreaming Linux kernel driver  
Intending to qualify with Ubuntu, SuSE  
FPGA driver can be loaded as kernel module |
| Hypervisor (VM)       | KVM                   | Partnering with VM Ware (ESXi hypervisor)                                |
| Container Format      | Untested at the moment | Linux driver architected to support containers                           |
| Bare Metal Support    | Yes                   | Linux driver architected to support bare metal                           |
| VM Management         | None.                 | Partnering with VM Ware (vSphere)                                       |
| Cloud Orchestration   | Open Stack            | PAC + Accel. Stack deployed with OpenStack  
Kubernetes support for FPGA Docker Containers (alpha) |
Recommended Getting Started Process

Install VM with Linux and install Acceleration Stack Development Flow

Run through quick start guide to validate environment

- Run ASE on Hello AFU
- Regenerate hello_afu GBS and application

If applicable, run the OpenCL example

Examine the DMA_AFU and/or Streaming_DMA_AFU example to understand how to move data using DMA with AFU

Run Signal Tap using n1b_mode_0_stp and hello_fpga

Create your own by modifying one of the example designs
INTEL® FPGA ACCELERATION HUB

A new collection of software, firmware, and tools that allows all developers to leverage the power of Intel® FPGAs.

Intel® portal for all things related to FPGA acceleration

• Acceleration Stack for Intel® Xeon® with FPGAs
• FPGA Acceleration Platforms
• Acceleration Solutions & Ecosystem
• Knowledge Center
• FPGA as a Service
• 01.org *

* 01.org is an open source community site
Follow-On Training:

**Online Training Course**

- Introduction to the Acceleration Stack for Intel® Xeon w/ FPGA
- OpenCL™ Development with the Acceleration Stack
- RTL development and acceleration with the Acceleration Stack
- Application Development on the Acceleration Stack for Intel® Xeon® CPU with FPGAs
- Introduction to High-Level Synthesis (7 courses)
- Introduction to Parallel Computing w/ OpenCL on FPGAs
- Deploying Intel FPGAs for Inferencing with OpenVINO Toolkit
- Programmers' Introduction to the Intel® FPGA Deep Learning Acceleration Suite

**Instructor Led Training Courses**

- Introduction to High-Level Synthesis with Intel® FPGAs
- High-Level Synthesis Advanced Optimization Techniques
- Introduction to OpenCL
- Optimizing OpenCL™ for Intel® FPGAs (16 Hours Course)

[https://www.altera.com/support/training/overview.html](https://www.altera.com/support/training/overview.html)
Exercise 3: Add and Test a Scratch Register

First add a scratch register to the AFU that you do something with (i.e. multiply by 2)

Then add the software capability in the host application to communicate with the new register and test it

**MMIO address**

<table>
<thead>
<tr>
<th>Address</th>
<th>Bytes (OPAE)</th>
<th>Words (CCI-P)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>0x0000</td>
<td></td>
<td>AFU header (read-only)</td>
</tr>
<tr>
<td>0x0008</td>
<td>0x0002</td>
<td></td>
<td>GUID</td>
</tr>
<tr>
<td>0x0010</td>
<td>0x0004</td>
<td></td>
<td>AFU ID_L</td>
</tr>
<tr>
<td>0x0018</td>
<td>0x0006</td>
<td></td>
<td>AFU ID_H</td>
</tr>
<tr>
<td>0x0020</td>
<td>0x0008</td>
<td></td>
<td>NEXT_DFH</td>
</tr>
<tr>
<td>0x0022</td>
<td>0x0008</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>0x080</td>
<td>0x0020</td>
<td></td>
<td>scratch_reg</td>
</tr>
<tr>
<td>0x088</td>
<td>0x0022</td>
<td></td>
<td>UserScratch_reg</td>
</tr>
</tbody>
</table>

**Device Feature Header**

- Global Unique ID
- AFU ID (low 64 bits)
- AFU ID (high 64 bits)
- Pointer to next DFH
- Reserved space