WHITE PAPER

Intel® FPGA

# Simplify Software Integration for FPGA Accelerators with OPAE

## Cross-Platform FPGA Programming Layer for Application Developers

**Authors**

**Enno Luebbers**
Senior Software Engineer
Intel® Corporation

**Song Liu**
Senior Software Engineering Manager
Intel Corporation

**Michael Chu**
FPGA Marketing Senior Manager
Intel Corporation

**Table of Contents**

## Abstract

The rising scales of FPGA deployments especially in cloud environments scaling Acceleration as-a-Service or for datacenter-wide workload acceleration have created an urgent need for a unified and simple software interface to discover, access, and manage reconfigurable resources. Current system design methodologies either abstract FPGA access and resource management to provide a domain-specific interface, or generate register transfer level (RTL) outputs that leave the actual integration to the developer, providing very limited system debug and deployment support.

The Open Programmable Acceleration Engine (OPAE) is an open community effort started by Intel to simplify and streamline the integration of various FPGA acceleration devices into software applications and environments. The OPAE currently consists of several software components and encompasses drivers as well as user-space application programming interfaces (APIs). It presents a unified, layered software access model for reconfigurable accelerators that provides common and extensible methods for discovery, allocation, access, and management of accelerator resources, while providing access to the software stack at different layers to aid in debug, bring-up, and deployment. The OPAE is designed to be light-weight and easy to integrate into existing software frameworks, from domain-specific libraries to cloud orchestration frameworks. At the user API level, it provides abstractions to simplify resource access and management without significantly impacting performance, thus relieving system integrators, software developers, and accelerator designers from having to re-implement basic FPGA infrastructure components for register access, shared memory, synchronization, and reconfiguration. The OPAE allows you to select the level of abstraction and control by providing application interfaces throughout the software stack.

## Introduction

The predominant model for the design and implementation of FPGA accelerators is one of custom vertical integration of application, infrastructure, and RTL development. In systems aiming at accelerating complex sequences of algorithms, developers usually need to either create the physical and logical communication layers between the FPGA implementation and the software executing on the processor from scratch, or rely on standard IP libraries providing varying levels of control and still involving heavy integration work and IP vendor specific software libraries and conventions. That, of course, is on top of identifying, designing, and implementing in RTL the portion of the algorithm to be accelerated.

With the rise of at-scale deployments of FPGAs in cloud environments and continuing integration of FPGA accelerators into domain frameworks such as machine learning, the task of providing a standardized way of accessing FPGA accelerators and integrating them into existing software environments is becoming just as important as the creation of the accelerator function itself. Efficient discovery, allocation, and management of FPGA resources is a prerequisite of large-scale deployments of reconfigurable technology. At the same time, developers of accelerator logic also need access at different levels of the software stack to aid in development, debug, and bring-up of single-nodes and at-scale installations alike.

The OPAE provides a layered software access model for enumerating, accessing, and managing FPGA resources with the goal of laying the groundwork for a unified API for FPGA development that can span horizontally across different domains and platforms. With very light-weight support from the underlying FPGA-implemented infrastructure to identify and enumerate hardware capabilities and features, the OPAE eliminates the need to reimplement standard hardware communication primitives, such as accessing control registers or allocating shared memory, without limiting the scope of interconnect mechanisms or impacting accelerator performance. In addition, the OPAE API model and driver framework is designed to be extensible to also support unique hardware characteristics.

## Acceleration Stack for Intel® Xeon® CPU with FPGAs

To provide a layered access model for FPGA accelerators that is applicable across devices, operating systems, and application domains, we need to address the entire system stack from the FPGA implementation through drivers, user-space APIs, application-specific libraries, and frameworks as depicted in Figure 1.

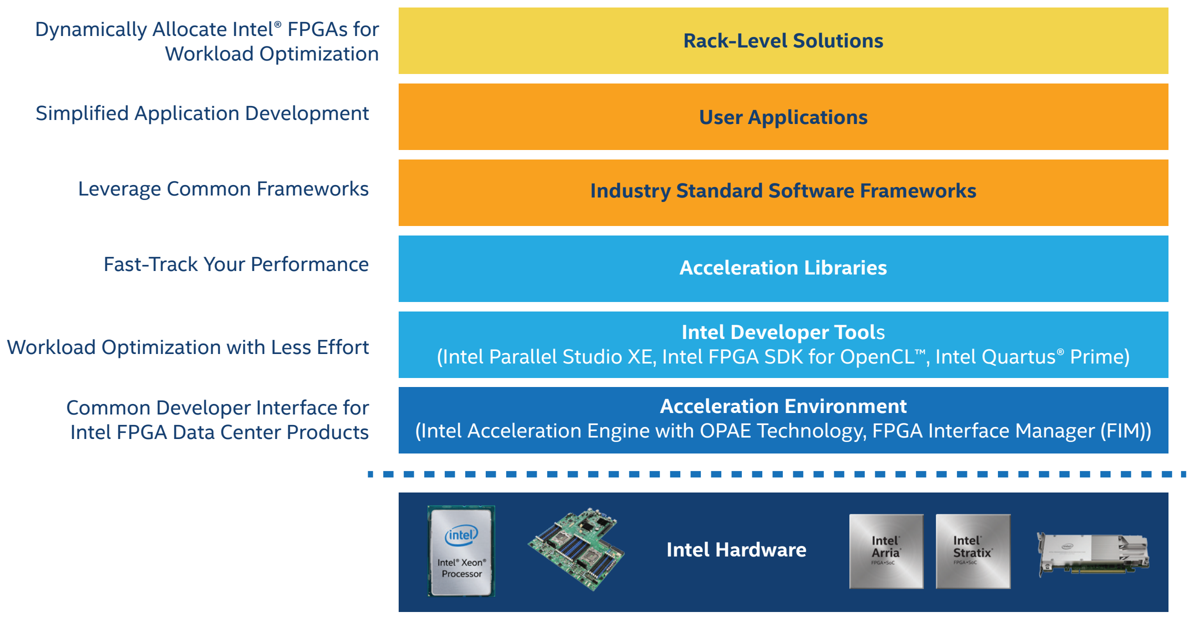
Dynamically Allocate Intel® FPGAs for Workload Optimization
Rack-Level Solutions
Simplified Application Development
User Applications
Leverage Common Frameworks
Industry Standard Software Frameworks
Fast-Track Your Performance
Acceleration Libraries
Workload Optimization with Less Effort
Intel Developer Tools
(Intel Parallel Studio XE, Intel FPGA SDK for OpenCL™, Intel Quartus® Prime)
Common Developer Interface for Intel FPGA Data Center Products
Acceleration Environment
(Intel Acceleration Engine with OPAE Technology, FPGA Interface Manager (FIM))
Intel® Xeon® Processor
Intel Hardware
Intel Arria
Intel Stratix

**Figure 1.** Acceleration Stack for Intel Xeon CPUs with FPGAs

The actual accelerator hardware resources comprised of FPGA devices, interconnects, and infrastructure logic, form the bottom layer and usually connect to the processor through standard system buses involving address translation logic, and caching hierarchies. The OPAE does not impose a specific interconnect technology or topology. However, it does require that the programmable logic exposes software-accessible data structures to identify and enumerate hardware components and capabilities. The physical access interface of the reconfigurable hardware resources is usually not exposed directly to applications.

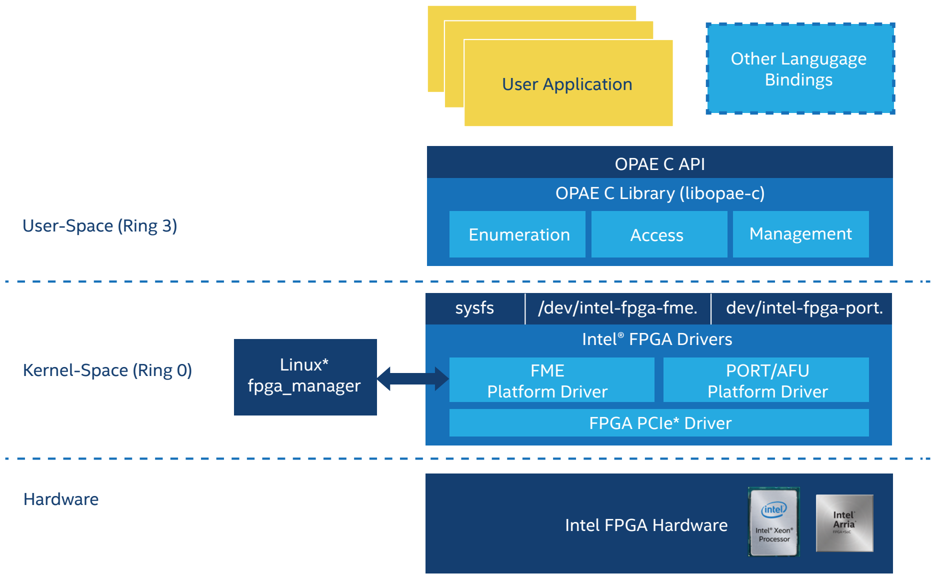

**Figure 2.** OPAE Layers and Components

On top of the programmable hardware resources and system interconnect, drivers connect and integrate the accelerator resources into the operating system's (OS) device management and expose their fundamental access methods to user-space applications. An important task of the driver layer is to enforce basic access policies to ensure system integrity, as well as perform critical management tasks such as error, power, and thermal management in the background. The user-space driver API is still OS and device-specific. For example, Linux* OS consists of a combination of device nodes, `ioctl()` calls and `sysfs` interfaces.

To enable the benefits of portability across devices, platforms, and operating systems, but at the same time maintain fine-grained control over individual resources, the OPAE provides a C API layer implemented in a thin user-space library (libopae-c), which interfaces with the device driver APIs to create a low-level abstraction of accelerator resources and allows enumeration, access, and management of these resources. It is still somewhat FPGA technology specific, as evident by the components of the abstraction model it provides, but enables the desired level of expressiveness to cover a wide range of accelerator resources, platforms, and deployment domains.

By integrating the FPGA software stack into standard software libraries and frameworks, (for example, linear algebra, deep learning, cryptography, compression, or other common functions) it is then possible to transparently accelerate larger numbers of applications and simplify larger scale deployments because individual applications do not need to deal with individual accelerator instances, let alone low-level communication primitives. Optimization efforts within the FPGA API library and driver stack are focused on improving management functions such as allocation or reconfiguration.

## FPGA Accelerator Hardware

The fundamental task of accelerating a particular workload relies on the specific capabilities of the underlying FPGA hardware platform. In order to be used with the OPAE, the platform needs to provide a minimum level of light-weight infrastructure for discovering, allocating, and accessing reconfigurable hardware resources. The particular hardware specifics, such as the number and type of communication links, the management and reconfiguration features are handled and abstracted in the OPAE driver and API library layers so that applications developed for it can be ported with minimal effort to other platforms exposing the same access model.

## FPGA Driver

The FPGA driver architecture defines individual platform drivers for management functions, such as reconfiguration and accelerator access. The former attaches to FPGA management logic while the latter is used to access generic methods to communicate with an accelerator programmed into a slot of the FPGA.

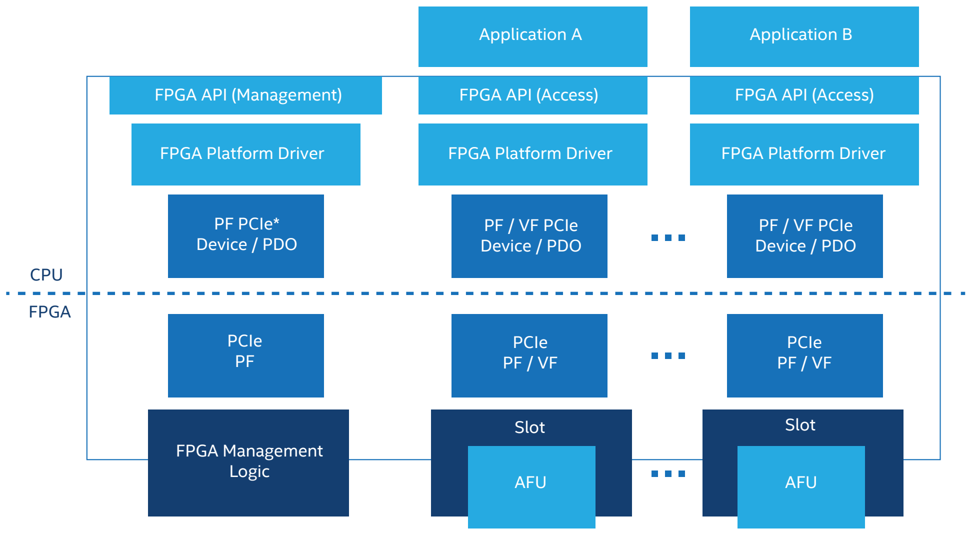

**Figure 3.** FPGA Driver

To enable a reasonable degree of flexibility in the implementation of the driver layer, which is responsible for encapsulating variations in the supported underlying hardware architectures, enabling multiple possible interconnection technologies, and maintain system integrity and stability, the driver is segmented into a class driver and several feature drivers. The class driver detects and discovers FPGA devices, and then instantiates individual feature drivers based on the top-level enumeration of the device features exposed by the discovered devices.

## FPGA Application Programming Interface (API)

As explained above, it is the goal of the user-space API to expose a low-level, but portable abstraction of accelerator resources to application software, diagnostic tools, and upper-level frameworks alike. It is designed to be as light-weight as possible. The API it provides follows an object-oriented pattern of modeling accelerator resources and associated operations; its implementation is, however, written in C to minimize the footprint and dependencies for applications using it.

The OPAE C API is modeled around a set of base objects used to describe, identify, and reference FPGA resources. It is worth emphasizing that the APIs presented here form the basic set that we believe are applicable to most FPGA accelerators interfacing with software systems; the API allows for device or platform-specific extensions to model unique features of specific target architectures. For example, we created a platform-specific API extension to expose a low-latency notification mechanism over the coherent memory interconnect of the Intel® Xeon® processor with integrated FPGA.

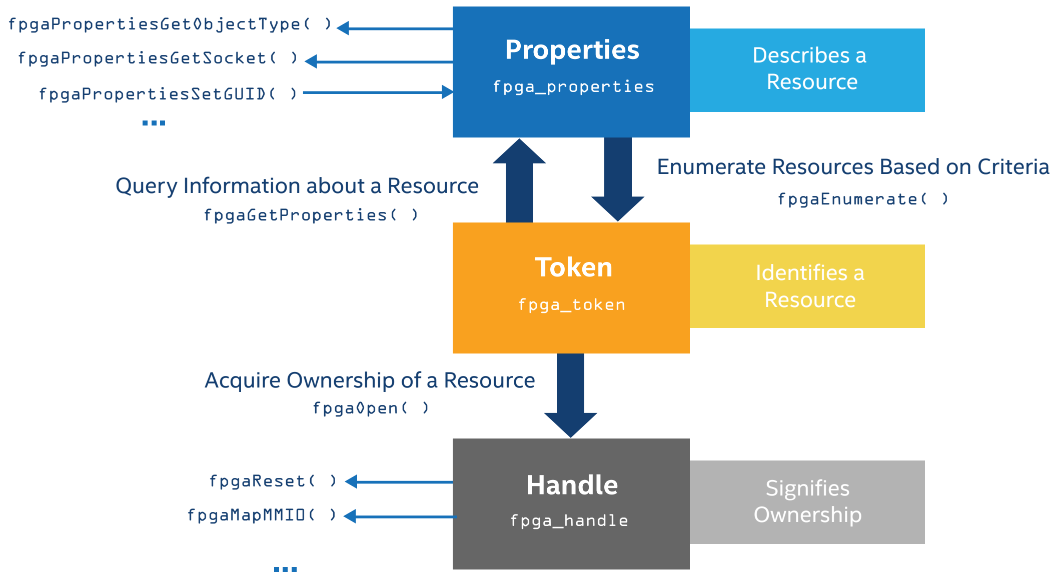

**Figure 4.** OPAE C API Objects and Control Flow

For an application that wants to acquire and access a specific accelerator, a typical flow through the API-provided functions is depicted in Figure 4. First, the application creates a properties object specifying object characteristics it is looking for; this properties object is passed into the `fpgaEnumerate()` call to yield a number of tokens identifying particular resource instances. Holding a token does not imply ownership of the associated resource. After selecting a token, the application calls `fpgaOpen()` to acquire ownership of the resource, which in turn yields a handle. This handle can be used to call the individual API functions to actually access the FPGA accelerator, such as resetting the accelerator, reading/writing control registers, or allocating shared memory. Finally, `fpgaClose()` closes a handle and releases ownership.

A similar flow is used by management tools, for example to reconfigure an FPGA slot. These management-type API calls typically require a different privilege level than the accelerator access functions outlined above.

## Binding or Framework Integration

Although the basic resource abstractions provided by the API already ease portability of applications across platforms and operating systems, they still require application developers to understand the register maps, protocols, and flows for accessing resources, which may be device- and definitely are accelerator-specific. Considering what we currently have in the OPAE, one approach is to integrate FPGA accelerator resources into higher-level frameworks and libraries. They can then be used by applications agnostic of the accelerator-specific access protocols. In this way, transparent acceleration of a larger number of existing applications becomes possible. Examples of application domains that could benefit from transparent FPGA acceleration through higher-level libraries are image recognition, data analytics, and data compression. How this integration is carried out is very dependent on the upper framework and identified accelerator function; some frameworks may already be partitioned in a way amenable to replacing one implementation with another; others may require extensions to selectively utilize accelerated resources. The common access model and associated APIs simplifies the integration process and extends the benefits of portability provided by the API layer to the upper stack.

From a large-scale deployment point of view, the management APIs exposed by the FPGA API layer allow seamless discovery and allocation of FPGA resources in multi-node to data center installations.

## Application Example

To illustrate the basic usage of the API, the following listing contains a typical, simple application that enumerates an AFU, opens it, allocates a shared memory region, maps MMIO space, and interacts with control registers. Note that this example omits error checking for brevity - all API functions return error codes to signal success or error conditions.

```
#include <opae/fpga.h>

int main(int argc, char *argv[])

{

      fpga_properties    filter;
      fpga_token         afu_token;
      fpga_handle        afu_handle;
      fpga_guid          guid;
      uint32_t           num_matches = 1;

      volatile uint64_t  *mmio_ptr;
      volatile void      *buf_ptr;

      uint64_t           buf_handle;

      /*  Enumerate  */
      fpgaCreateProperties(&filter);
      fpgaPropertiesSetObjectType(filter, FPGA_AFU);

      /* (GUID 'guid' defined elsewhere) */
      fpgaPropertiesSetGUID(filter, guid);
      fpgaEnumerate(&filter, 1, &afu_token, &num_matches);
      fpgaDestroyProperties(&filter);

      /* Open and access */
      fpgaOpen(afu_token, &afu_handle, 0);
      fpgaMapMMIO(afu_handle, 0, &mmio_ptr);
      fpgaPrepareBuffer(afu_handle, BUF_SIZE, &buf_ptr,

      &buf_handle, 0);

      fpgaReset(afu_handle);

      fpgaWriteMMIO64(afu_handle, 0, CSR_BUF_ADDR, buf_ptr);
      fpgaWriteMMIO32(afu_handle, 0, CSR_CTL, 1); /* start */
```

```
        /* other accelerator logic */
        fpgaWriteMMIO32(afu _ handle, 0, CSR _ CTL, 7); /* stop */

        fpgaReleaseBuffer(afu _ handle, buf _ handle);
        fpgaClose(afu _ handle);

        return 0;
```

## Conclusion

FPGA technology has been there since 1980s. But programming and integrating FPGA with applications are still challenging and costly mainly due to the lack of a simple and common programming model that is accessible to everyday developers. The OPAE brings everyone in the community and in the industry together to work on such a programming model so that we can enable more developers to leverage FPGA acceleration and bring FPGA into the data center.

## References

- OPAE website: http://01.org/OPAE
- OPAE source code: http://github.com/OPAE
- FPGAs in the Data Center website: https://builders.intel.com/blog/fpga-in-the-data-center-programming-for-all/