

Enabling Consistent Platform-Level Services for Tightly Coupled Accelerators

White Paper

Intel® QuickAssist
Technology Accelerator
Abstraction Layer

Computationally-Intensive
Applications

New software abstraction layer simplifies
the use and deployment of accelerators

Table of Contents

Executive Summary.....	3
Accelerator Deployment Challenges.....	4
Insulating Application Software	4
An Accelerator Abstraction Layer Solution.....	5
AAL Supports Platform-Level Services.....	6
FSB-FPGA Accelerator Modules.....	7
FSB-FPGA Accelerator Module System Architecture.....	8
AAL Common Programming Model	10
Benefits from the Intel Accelerator Application Layer	11

Executive Summary

Industries, such as financial services, energy, manufacturing and chemical production, are exploring the use of Field Programmable Gate Array (FPGA)-based accelerators for computation-intensive algorithms. FPGA accelerators can speed up domain specific algorithms by implementing these algorithms directly in hardware.

Typically, accelerators are programmed via a “direct” API that essentially wraps a driver interface. The programming paradigm offered by a direct API is of a device tightly bound to an individual application. However, this paradigm lacks the flexibility to easily accelerate functions across multiple users or abstract the accelerator interconnect specifics, which is needed to avoid software changes when accelerators with different busses are deployed. In today’s data center environment, where scarce resources are virtualized and accelerator technology advances at a rapid rate, a direct API doesn’t offer sufficient agility. Further, as the underlying hardware technology of the accelerator changes (e.g., different FPGAs, capacities, properties and attach technologies), necessary modifications inevitably “leak through” the API to the end user, requiring application software revisions.

Easing accelerator system integration, the Intel® QuickAssist Technology Accelerator Abstraction Layer (AAL) solves the management and virtualization problem, providing a uniform way for applications to communicate with accelerators and manage them as resources. This paper outlines some of the platform-level service needs of accelerator users and how introducing a new software framework for acceleration increases platform agility.

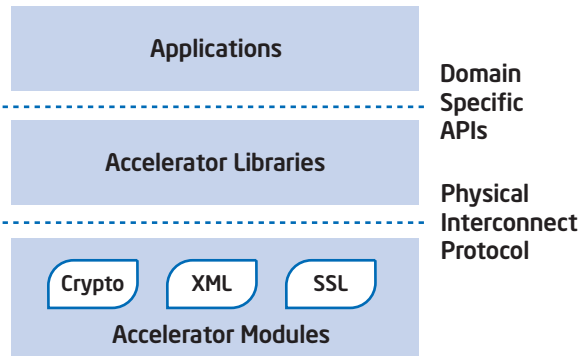


Figure 1. System Block Diagram

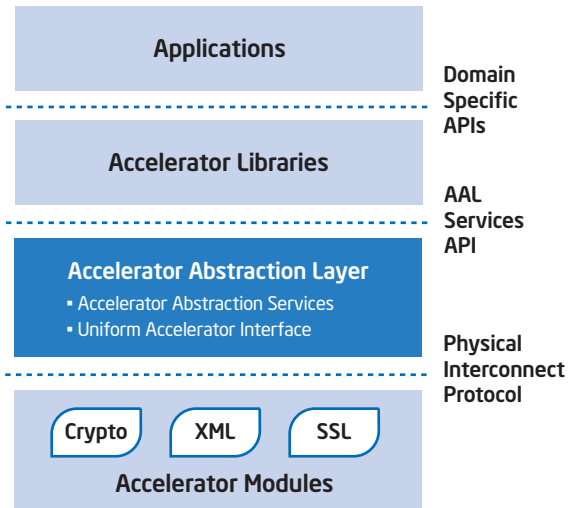


Figure 2. Intel® QuickAssist Technology Accelerator Abstraction Layer

Accelerator Deployment Challenges

There will always be classes of workloads and algorithms that push the capabilities of traditional compute platforms. Appropriately partitioning a problem between a general-purpose processor and one or more accelerators is often an approach used to meet the demands of these algorithms. For example, financial services firms may benefit from coding the core algorithms of financial analytics applications, such as Monte Carlo-based algorithms, onto FPGAs to speed them up. Likewise, many security appliances deploy accelerator modules that execute cryptology, XML and SSL algorithms, as illustrated in Figure 1.

To communicate with these accelerator modules, software applications use domain specific APIs to call routines in accelerator libraries. However, this software framework may be susceptible to portability, performance and system issues, as described in Table 1. It is the purpose of AAL to resolve these issues.

Insulating Application Software

AAL, shown in Figure 2, resides between the accelerator libraries and the accelerator modules. This software framework is accelerator technology agnostic and performs key services on behalf of the application, thereby reducing the total cost of ownership/execution (TCO/TCE) for accelerator-enabled platforms.

AAL serves a wide variety of use cases and workloads and is designed to be compatible with the firmware and software environments used by developers today. By facilitating communications between the host CPU and the accelerator, this software framework can simplify the deployment of the platform services listed in Table 2. The table describes how AAL framework features can help minimize application code changes, lower TCO and increase memory efficiency, agility and system responsiveness.

Table 1. Possible Application Software Issues

Issues	Resolutions
Portability	The software calls are closely tied to hardware drivers for network adapters, impacting portability.
Performance	The accelerator and the operating system may inefficiently copy data, affecting performance.
Complexity	Management and resource-sharing functions, such as resource balancing and multi-tasking, may need to be implemented by application software, increasing complexity.
Churn	The memory model may require modification when the operating system, CPU or system fabric changes, necessitating a new software release.

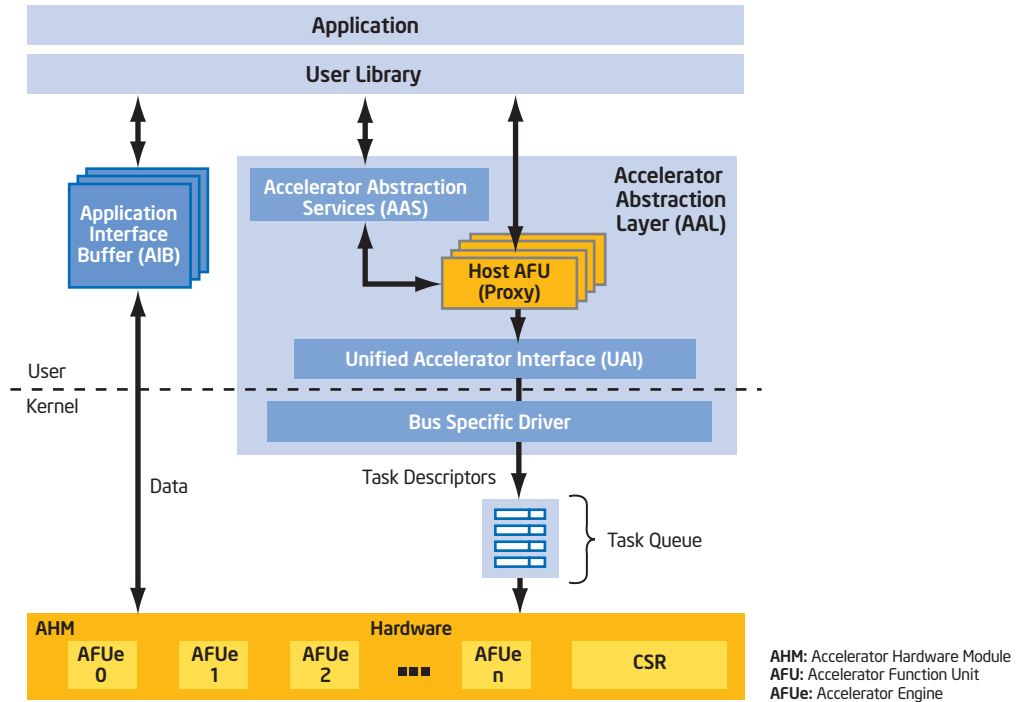


Figure 3. System Architecture

An Accelerator Abstraction Layer Solution

Through a variety of initiatives under the banner of Intel® QuickAssist Technology, Intel will support accelerator innovation across functionalities, markets and interconnects. This technology includes the Intel QuickAssist Technology Accelerator Abstraction Layer (AAL) that supports a set of software services between applications and accelerator modules deployed on Intel® processor-based platforms, such as FPGA-based accelerator modules.

AAL consists of components that run in user space (a.k.a. ring 3) and kernel space (ring 0) and balance performance and reliability, as shown in Figure 3. Together, these components provide services, such as accelerator virtualization and resource management. Libraries are provided that act as proxies to the accelerator hardware by implementing technology-specific message formatting and APIs. Accelerator resources are exposed to the application as an object called an Accelerator Function Unit (AFU). The AFU is made up of a collection of hardware and software components abstracted through a single object.

Table 2. Platform Service Examples

Platform Services	Features	Benefits
Manage Accelerators	Maintain a database of acceleration functions (e.g., encryption, random number generation and differential equation solvers).	Organizing core algorithms in libraries is a standard programming practice and helps minimize application code changes.
Optimize Memory Usage	Orchestrate the memory interaction between accelerators and the operating system.	Maximizing memory efficiency increases performance.
Accelerator Virtualization	Virtualize the accelerators so multiple calls to the accelerators from different applications operate seamlessly.	Sharing accelerator resources lowers TCO.
Workload Distribution	Enable the ability to distribute the workload or even prioritize certain calls or operations, such as load balancing or fault resilience.	Addressing critical demand enables the system to be more responsive.
Reconfigure FPGA	Manage the loading of reconfigurable FPGA as needed by the workload distribution.	Reconfiguring FPGAs on-the-fly increases system agility.

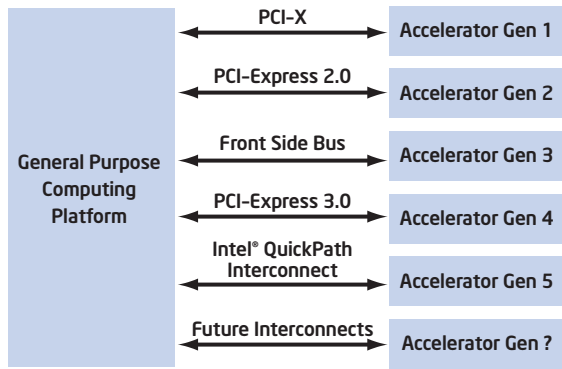


Figure 4. Many Interconnect Options

These components may include any hardware (gates) embedded in an FPGA, as well as the host-side software that exposes it. An AFU will typically have a hardware component in an Accelerator Hardware Module (AHM), but it could also be implemented entirely in software. Very often an AFU will implement parts of its algorithm on the IA host and parts on the dedicated accelerator hardware. AAL allows AFU developers to implement their algorithm on the compute engine that best serves their needs.

The IA host and the AFU typically use a block of memory to share data, such as matrices and variables. AAL manages the shared memory block, mapping it into user space and making it accessible to the accelerator module.

The AAL kernel space code interacts with the device driver across many possible bus interfaces, such as PCI Express*, the CPU Front Side Bus (FSB) and future interconnects, as shown in Figure 4. AAL specifies a common presentation of the services, and it provides a programming model and a set of commands for interacting with various accelerators.

AAL Supports Platform-Level Services

The AFU Package is a bundle of software that implements the AFU and other supporting software. This includes the code run by the host CPU to configure and communicate with the AFU as well as any firmware, FPGA bit files and configuration information necessary to manage the AFU.

AAL provides a service infrastructure and a collection of common platform services known as the Accelerator Abstraction Services (AAS). Through these services, AAL implements a lightweight Service Oriented Architecture (SOA). The AAS provides AFU Package registration, resource management and communications services, including the event and threading models. AAS services perform uniform accelerator discovery, configuration and messaging services. They employ a common installer interface for applications and libraries to register and manage the AFU Packages using functions such as query, enumerate, find and load packages.

This AAL software framework establishes a flexible communications interface between applications, accelerators and a variety of support services. It is the foundation for many platform-level capabilities, including:

- Asynchronous programming model with event dispatcher and delivery that is optimized for concurrent task processing.
- Protection of shared resources allowing multiple applications to share a single accelerator by supporting multiple threads.
- An SOA that enables dynamic binding to acceleration packages, providing maximum flexibility with minimal application impact.

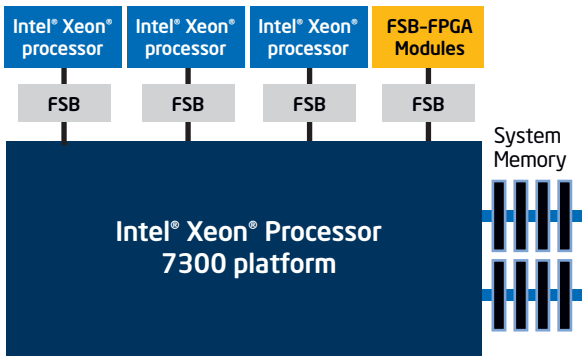


Figure 5. Accelerator Hardware Module Interfaces to the Front Side Bus

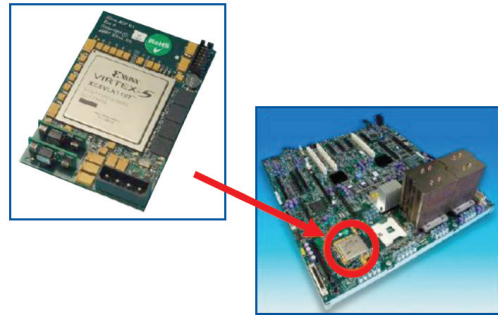


Figure 6. FSB-FPGA Accelerator Module

FSB-FPGA Accelerator Modules

AAL manages the Accelerator Hardware Modules (AHMs) that attach to the processor’s Front Side Bus (FSB). This is illustrated in Figure 5 where the FSB supported by the Intel® 7300 chipset interfaces to four components: an FSB-FPGA accelerator module and three Intel® Xeon® processors.

Intel provides FPGA accelerator module vendors with FSB Register Transfer Logic (RTL) for integration into their designs. This capability allows accelerator modules to benefit from a low latency interconnect, communicating over the FSB to system memory. The FSB interface gives accelerators high-bandwidth access to the host CPU, often speeding up overall acceleration

several times over. An FSB-FPGA accelerator module based on Xilinx FPGAs is shown mounted on a four-socket system in Figure 6; and an FSB-FPGA accelerator module from XtremeData, Inc. based on Altera FPGAs is shown mounted on a two-socket system in Figure 7.

It’s also possible to partition complex algorithms across multiple FSB-FPGA accelerator modules, as shown in Figure 8. In addition to communicating across the FSB, the accelerator modules can deploy serial busses to share data between themselves and achieve a multiprocessing system. AAL supports multiple AHMs from different vendors.

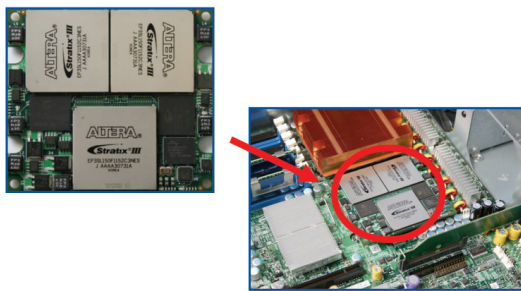


Figure 7. FSB-FPGA Accelerator Module

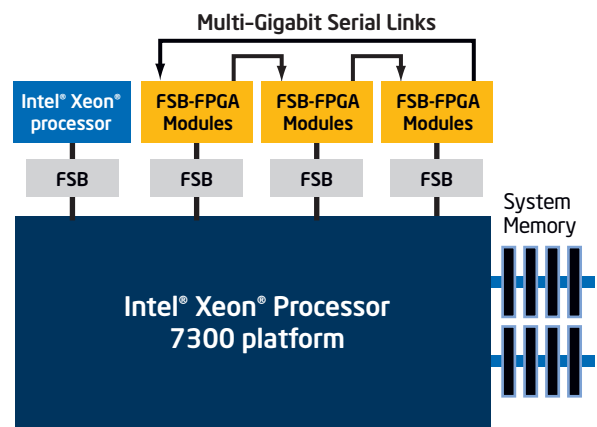


Figure 8. Multiple FSB-FPGA Accelerator Modules

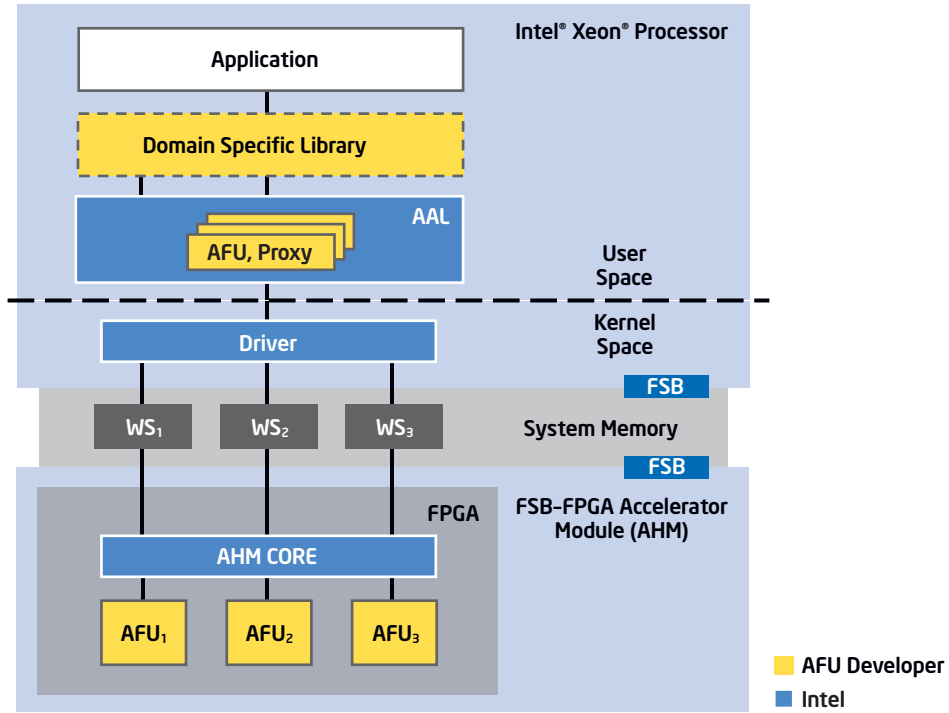


Figure 9. FSB-FPGA System Architecture

FSB-FPGA Accelerator Module System Architecture

Figure 9 illustrates the overall FSB-FPGA accelerator module system architecture. The components of this system consist of both hardware and software, and are located on the host, on system memory attached to the host via the FSB, and on AHMs also attached to the FSB. Communication between host and AHM components is performed via FSB transactions and interrupts, which are handled by the AHM device driver on the host and by the AHM core on the AHM.

The Application consists of the collection of software threads on the Host that use the AAL framework to invoke accelerator operations. In Figure 9, the framework is represented by the box labeled AAL and by the components within that box labeled AAL Proxy.

The operations are carried out on the AHM by hardware components called Accelerator Function Unit Engines, or AFU Engines. These are hardware designs synthesized for the FPGAs on the AHM. Because the operational details of the AFU engine are not only vendor-specific, but potentially algorithm-specific, the AFU engine is associated with a Proxy AFU component on the Host in which those details are encapsulated. The Application uses the AFU Proxies and the services provided by AAL to send requests to AFU engines and to receive responses back from the AFU engines. Together, the request and response are referred to as an AAL Transaction, and the AFU Proxy/AFU Engine pair is referred to as an AFU.

A third-party developer may provide a Domain Specific Library, as shown in Figure 9, to improve the usability of the AAL environment for an application domain. For example, in financial analytics, the Application may use “price model evaluation engines” that are implemented on AFUs and provisioned by a class library.

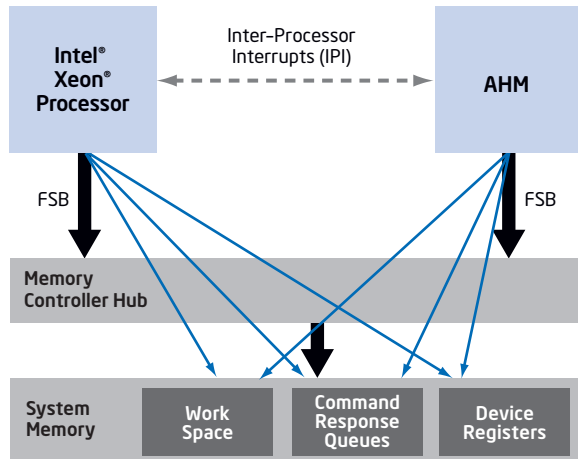


Figure 10. Host CPU and Accelerator Module Share Memory

The memory bandwidth and latency improvements offered by the FSB-FPGA system architecture are due to its attachment to the FSB, which allows bulk data to be stored in system memory and accessed by the AHM. This is accomplished by defining shared memory workspaces in system memory, created by the AAL framework at the behest of the application, as shown in Figure 10. Bulk data movement is done by writing the data into a workspace and setting up a DMA transfer request.

Control and Status Registers (CSR), which are defined by the AFU for application-specific purposes, are also implemented in shared memory. The protocol followed by the AAL device driver and the AHM core ensure that register contents are maintained coherently.

Workspaces and CSRs are stored in shared memory that is pinned in system memory in order to increase efficiency; with this convention in place, the AHM core can work with physical memory addresses. This system architecture is easy to maintain, debug and creates a straightforward standard communication scheme.

Accelerator hardware modules (AHMs) contain one or more FPGAs, SRAM, Flash memory and control logic, as shown in Figure 11. FPGAs implement core logic, AFU(s) and management functions such as FPGA reconfiguration. The AHM core logic provides the FSB interface logic, low-level AHM device interface and multiple AFU engine interfaces.

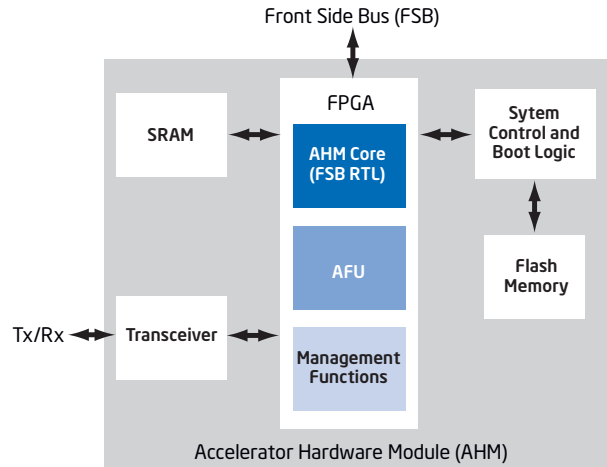


Figure 11. Accelerator Hardware Module Architecture

FSB-FPGA accelerator module vendors have access to a variety of development environments for device coding, simulation, synthesis and programming. They have a choice of HDL languages, such as VHDL and Verilog, and high-level languages, including C and MATLAB/Simulink, as illustrated by the design flow shown in Figure 12. Intel is working with a variety of third-party tool vendors to provide FPGA software development kits.

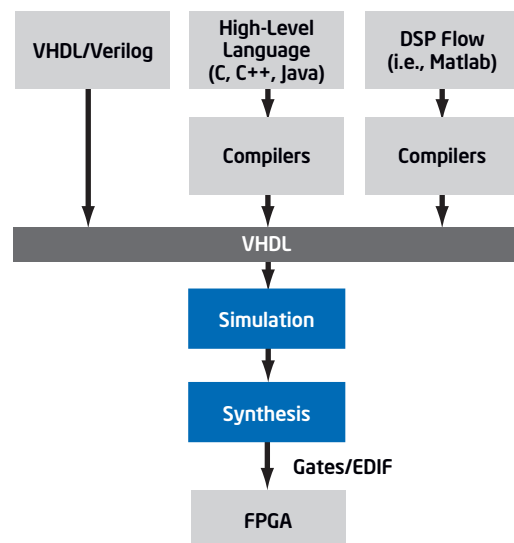


Figure 12. AFU Design Flow

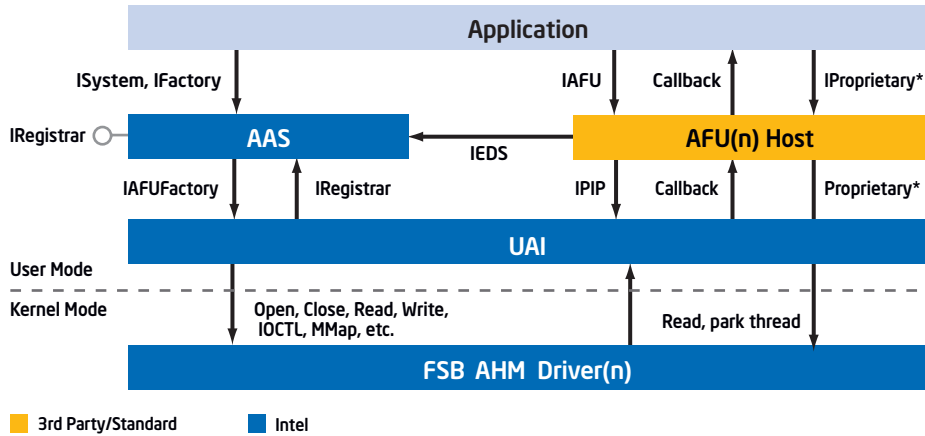


Figure 13. AAL Block Diagram

AAL Common Programming Model

AAL includes several interfaces which comprise a common programming model, as shown by the block diagram in Figure 13. At the top of Figure 13, there is an application and/or domain-specific library communicating with AAL services and objects. On the left, the Accelerator Abstraction Services (AAS) is depicted, and it contains the core set of AAL services which consist of:

- An IRegistrar holds installation and configuration data used by AAL services. The AAL Registrar is implemented on top of and uses the standard services defined by the platform (e.g., WMI for Windows* etc.). The Registrar implements and publishes a common platform independent API that is available to third-party utilities such as package install/configuration. AAL interacts with platform independent APIs to improve its portability across platforms.
- The ISystem interface initializes and configures global system parameters, such as how certain resources should be allocated or which program model is used by the application.
- AAL IFactory implements a variation of the well known Factory design pattern for service creation. A single virtual factory is used to delegate requests for resource (e.g., AFU) or service creation to specific concrete factories that are installed separately. This decouples AAL from the implementation of the services it supports, providing a great deal of flexibility and extensibility. New services or objects can be designed and installed without impacting AAL or other installed components.

- The Event Delivery Service (EDS) provides local dispatch services for events/messages being delivered from the AFU and other system services. The EDS implements the last hop of the message delivery and is the service that defines the callback model semantics.
- The Application Interface Adaptor (AIA) is the layer that enables the application to bind dynamically at runtime to the Accelerator Function Unit. The AIA defines a standard interface to AAL and abstracts the upper layers from the details of the underlying AFU implementation. The AIA is also responsible for marshaling messages to and from the remote accelerator (AFU engine) using the interconnect technology-specific physical interface protocol (PIP). This is typically implemented via a kernel driver module. Because the AIA implements to a "standard" PIP, it may be reused to communicate with any number of remote AFUs that implement PIP. For example, any FSB devices implementing the FSB-PIP protocol are supported by the single FSB-AIA concurrently. This would also be the case for other interfaces such as PCI Express*, the Intel® Common Systems Interconnect (CSI), InfiniBand* and Ethernet. The driver and PIP are optional components of an AIA. A special AIA, known as the NULL PIP AIA, is designed to allow software-only AFUs that use proprietary device drivers to integrate easily into AAL.

Benefits from the Intel Accelerator Application Layer

The Intel QuickAssist Technology Accelerator Abstraction Layer introduces a software framework for deploying platform-level services and abstracting the interconnect technology from the application code. This software abstraction layer provides resource management and provisioning services that enable more efficient use of costly accelerator resources by allowing them to be transparently shared amongst multiple workload clients. It manages the interaction with the operating system, so developers get zero-copy performance out of the box. Accelerator device drivers execute from user space, which eases code development and debug. Using a shared memory model, AAL runs concurrent tasks and can help introduce parallelism at the system-level. AAL can also be used to reconfigure FPGAs on-the-fly.

Intel is committed to working with hardware vendors who build FSB-attached accelerator modules, as well as providers of compilers for FPGAs, to integrate AAL into their offerings. This complements Intel's open approach to FSB-coupled accelerators, in which we provide the Register Transfer Logic (RTL) and drivers necessary to develop FSB solutions. Customers can leverage accelerator technology to take advantage of technology advances without the pain of rewriting and retesting application code. By introducing an accelerator abstraction layer, developers can reduce the amount of changes to application software and ease the adoption of new bus interface technologies.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request. Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order. Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or by visiting Intel's Web site at www.intel.com.

Copyright © 2008 Intel Corporation. All rights reserved. Intel, the Intel logo and Xeon are trademarks of Intel Corporation in the United States or other countries.

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

