|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Change Request** | | | | | | | |
| **Document** | **ORAN-WG6.AAL-GAnP** | **ver** | **00.01.01** | **CR** | **NVD-004** | **rev** | 3 |

|  |  |  |  |
| --- | --- | --- | --- |
| ***Title:*** | AAL GAnP Chapter 4 Revision – to align with Cat 2 Greenfield Agreements | | |
| ***Source to WG:*** | NVIDIA | | |
| ***Target WG :*** | **WG6** | | |
| ***Category:*** | **B** | ***CR Creation Date*** | November 1, 2021 |
|  | *Use one of the following* ***categories****:* ***A*** *(mirror corresponding to a change in an earlier release)* ***B*** *(addition of feature),* ***C*** *(functional modification of feature)* ***D*** *(editorial modification)* ***F*** *(correction)*  Detailed explanations of the above categories can be found in 3GPP [TR 21.900](http://www.3gpp.org/ftp/Specs/html-info/21900.htm). | | |

|  |  |
| --- | --- |
| ***Reason for Change:*** | To align AAL GAnP Chapter 2 with Cat 2 Greenfield Agreements |
| ***Summary of change:*** | New text is proposed and can be reviewed by track change in the text below |
| ***Consequences if not aproved:*** | If not included, AAL GAnP Chapter 2 will not be consistent with AAL Greenfield Agreements |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***Clauses affected:*** | <list specific document sections impacted by the CR> | | | | |
|  | **Y** | **N** |  | |  |
| ***Other specs*** |  | **X** | Other core specifications: | <fill in related CRs if “Y”> | |
| ***affected:*** |  | **x** | Test specifications: | <fill in related CRs if “Y”> | |
| ***(show related CRs)*** |  | **X** | O&M Specifications: | <fill in related CRs if “Y”> | |
| ***Supporting material:***  ***Other comments:*** | <provide file name or URL of any material supporting this CR> | | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***Status:*** |  | ***CR Closed Date:*** |  |
| ***Outcome:*** |  | ***Duplication:*** |  |
| ***Outcome explanation:*** |  | | |

The proposed changes are indicated by Track Changes in the text below.

# AALI Configuration and Management Principles

The AALI API has two distinct parts, the first part corresponds to a set of common APIs (‘AAL Common API’) between application(s) and underlying AALI implementation(s) within an O-cloud platform, which allows an application to query and identify the capabilities of an AALI implementation and configure the associated AAL-LPU(s) accordingly for application’s use. A candidate set of functionalities supported by AAL common API(s) potentially includes (but not limited to) the following:

1. initialization of AAL-LPU(s) assigned to an application.
2. configuration of the state of these AAL-LPU(s) (for example, start, stop, or reset of an AAL-LPU).
3. configuration of various counters and resources associated with AAL-LPU(s) (for example, performance measurements/indicators, performance monitoring metrics, events, faults etc.).
4. discovery of AAL-profile(s) supported by these AAL-LPU(s) and associated configurations etc.
5. abstraction of transport mechanism between the application and AAI implementation

The second part of AALI corresponds to a set of AAL profile specific APIs (‘AAL Profile API’) which is specific to each defined AAL profile. The AAL profile shall be common across the AALI implementations accelerating the same set of AFs. It enables the applications to be able to efficiently offload AAL profile workload to AALI implementations in a consistent way without requiring them to know every single detail of the underlying HW implementation. Figure 4.1 shows examples of the AALI APIs presented to an application in three different scenarios.

Diagram, text

Description automatically generated

Figure 4.1 AALI Common and profile APIs

## AALI Common Functions

The AAL specification consists of a common initialization and configuration section and multiple profile API specifications.

Note: A Network Function may choose to use one or more AAL profiles as part of its implementation.

The AALI initialization and configuration are done on a per AAL-LPU basis. That is, for each AAL-LPU that is assigned to the NF, the NF shall initialize and configure each AAL-LPU that is assigned to it.

### AAL Initialization and Configuration Procedures

#### AAL-LPU Management

This section discusses about AAL-LPU(s), which are logical abstractions of an AALI implementation, presented to applications using the AAL interface. An AAL-LPU should not be confused with a physical HW accelerator.

#### AAL-LPU Identification & Representation

Within a process address space each AAL-LPU can be abstracted using a generalized object handle.

Depending on HW design and implementation choice, a HW Accelerator may want to accelerate multiple profiles or offer support for sharing HW Accelerator resources between multiple threads, processes, VMs, PODs. For this reason, a second abstract construct known as AAL Queue can be optionally used to

* distinguish between multiple supported AAL profiles per AAL-LPU
* prioritize access to AAL-LPU resources
* group operation requests
* Allow parallel access through AALI for multiple threads

As an abstract construct, an AAL Queue does not reflect a HW design specification or requirement but an AAL interface specification.

##### Example AAL-LPU Mapping

The following Section contains example deployments mapping AAL-LPUs to Applications / O-RAN Cloudified Network Functions. The labels ‘lpuId’ and ‘queueID’ in the following diagrams denote AAL-LPU object handle and identifier of an AAL queue respectively.

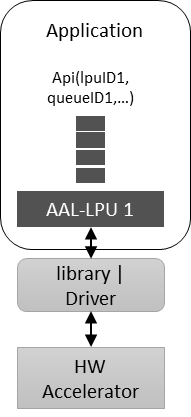


Figure 4.2 Basic mapping of AAL-LPU to O-RAN Cloudified NF

Figure 4.2 example shows a simple deployment with a HW Accelerator supporting a single AAL-LPU which exposes a single AAL Queue for the application to use.

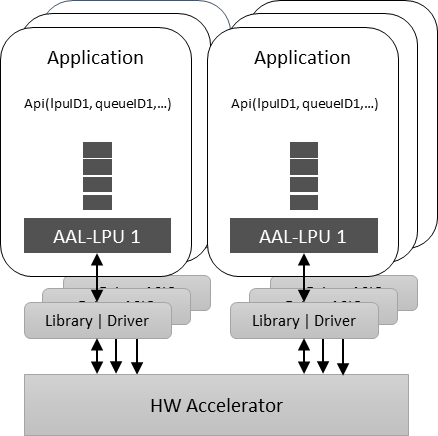


Figure 4.3 AAL-LPU mapping example showing multiple application support by a single HW accelerator

Figure 4.3 example shows the AALI supporting multiple applications with a single HW Accelerator. The HW Accelerator exposes multiple VFs through SRIOV. Each VF maps to an AAL-LPU. Each Application is assigned a single AAL-LPU (VF) and all AAL-LPUs share the resources of the underlying HW Accelerator.

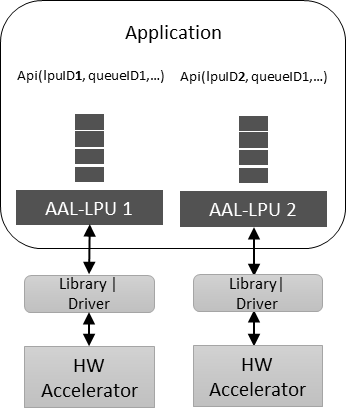


Figure 4.4 AAL-LPU mapping example showing multiple HW accelerators assigned to a single application

Figure 4.4 example shows the AALI supporting an application that is deployed with multiple HW Accelerators. In this case, a cloud platform contains multiple HW Accelerators, each HW Accelerator being exposed to the same application as an AAL-LPU.

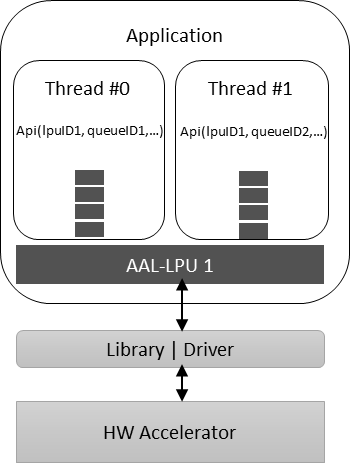


Figure 4.5 AAL-LPU mapping showing multiple AAL Queue support

Figure 4.5 example shows the AAL-LPU supporting multiple AAL Queues which are used by an application in multiple threads – allowing the application to avoid locking when exercising the AAL interface.

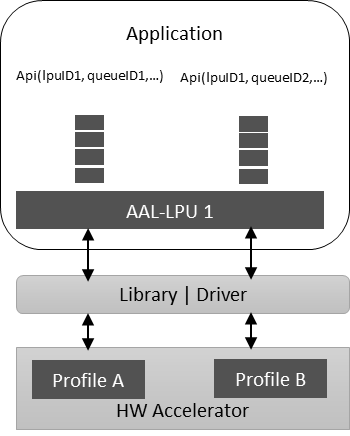


Figure 4.6 AAL-LPU Mapping example showing multi-function support

Figure 4.6 example shows the AAL accommodating a HW Accelerator that supports multiple AAL profiles. In this case application accesses the different profiles using the AAL Queue ID in the AALI.

#### AAL-LPU Configuration

Configuration of an AAL-LPU has two different levels: configuration that applies to the whole AAL-LPU, and configuration that applies to a single AAL Queue.

Note that, although all AAL Queues on an AAL-LPU support same capabilities, they can be configured differently and will then behave differently. This section details the AAL-LPU configuration which includes the following operations:

Allocation of resources, AAL Queues.

Resetting the AAL-LPU into a well-known default state.

Initialization of statistics counters.

The below sequence diagram shows the high-level initialization and configuration procedure for the AALI.

@startuml

Autonumber

Skinparam sequenceArrowThickness 2

skinparam ParticipantPadding 5

skinparam BoxPadding 10

Box “O-RAN” #NavajoWhite

Participant “O-DU” as ODU <<(Z,lime)O-RAN CLOUDIFIED NETWORK FUNCTION>>

end box

Box “Cloud Platform” #lightseagreen

participant “AAL” as AAL

End box

note over ODU, AAL

Assumed the O-DU is already instantiated on the O-Cloud

With desired HW accelerator resources assigned from O2 and

Configuration through O1 interface is complete

endnote

==AAL Initialization==

ODU -> AAL : Initialize

AAL --> ODU : Initialization Response

ODU -> AAL : Get Number of AAL-LPU

AAL --> ODU : Number of AAL LPUs available

ODU -> AAL : Get AAL LPU fine grain capabilities (AAL Profiles)

AAL --> ODU : AAL LPU capabilities

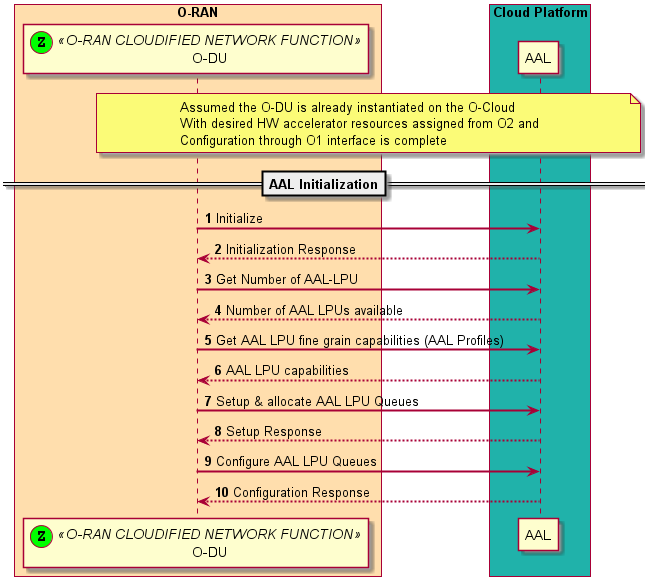
ODU -> AAL : Setup & allocate AAL LPU Queues

AAL --> ODU : Setup Response

ODU -> AAL : Configure AAL LPU Queues

AAL --> ODU : Configuration Response

@enduml



#### Device and Queue management

After initialization, AAL-LPUs are in a stopped state, so must be started by the application. If an application is finished using an AAL-LPU, it can close the AAL-LPU. Once closed, it cannot be restarted. By default, all AAL Queues are started when the AAL-LPU is started, but they can be stopped individually.

@startuml

Autonumber

Skinparam sequenceArrowThickness 2

skinparam ParticipantPadding 5

skinparam BoxPadding 10

Box “O-RAN” #NavajoWhite

Participant “O-DU” as ODU <<(Z,lime)O-RAN CLOUDIFIED NETWORK FUNCTION>>

end box

Box “Cloud Platform” #lightseagreen

participant “AAL” as AAL

End box

note over ODU, AAL

Assumed the AAL is already initialized

endnote

==AAL Run Time==

ODU -> AAL : Start AAL LPU

AAL --> ODU : AAL LPU Start Response

ODU -> AAL : AAL Queue Start

AAL --> ODU : AAL Queue Start Response

loop Run Time

ODU -> AAL: Perform Operations

AAL --> ODU : Operation Response

end

ODU -> AAL : AAL Queue Stop

AAL --> ODU : AAL Queue Stop Response

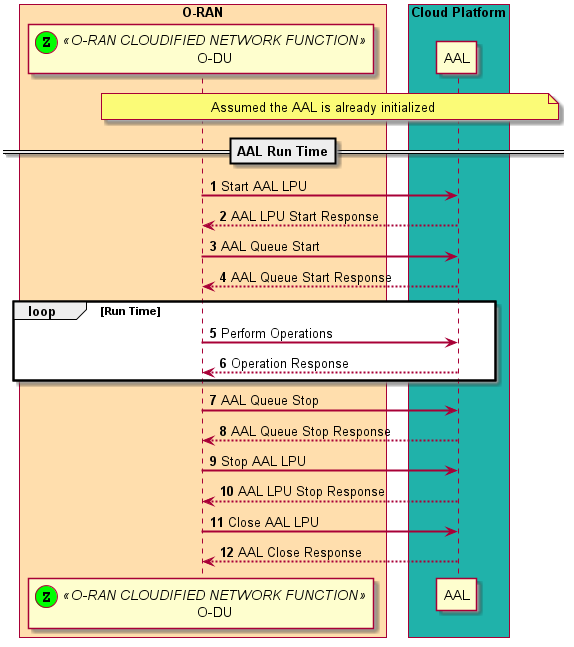
ODU -> AAL : Stop AAL LPU

AAL --> ODU : AAL LPU Stop Response

ODU -> AAL : Close AAL LPU

AAL --> ODU : AAL Close Response

@enduml



Note

The above sequence diagrams refer to the AAL-LPU and AAL queue operations. An AAL-LPU may not directly correspond to a HW Accelerator as an AAL-LPU may only represent a set of resources of the HW Accelerator. In this case an operation (start, stop etc..) on an AAL-LPU may not actually translate to a HW Accelerator operation – it is abstracted by the AAL and implementation specific.

#### Statistics

The AALI shall provide an O-RAN Cloudified NF with general statistics upon request. Statistics may include but not limited to operation counts and error counts.

#### Memory Management

O-RAN network functions (O-DU, O-CU, etc.) will be responsible for input, output and operation structure memory allocation and freeing, using AAL defined memory management functions. All other application memory is not required to use the AAL memory management functions.

Device Drivers are free to manage their own internal memory, DMA implementation as needed, the AAL specification does not add any memory requirements to device driver.

Each AAL Profile shall define its own memory requirements and implement its own memory backing if needed.

Each AAL Profile may define its own operation structure memory allocation if needed.

#### Run Time Configurations

Operations are requested to the AAL-LPU to perform specific HW Accelerated Function(s). Each operation shall be represented by an operation struct that shall define all necessary metadata, configurations and information required for the operation to be processed on an AAL-LPU. The operation structs shall define the operation type to be performed, including an operation status and reference to the AAL Profile specific operation data which can vary in size and content depending on the AAL profile. Each AAL profile shall define its own operation structure for its specific functions.

#### AAL profile(s) offload, processing status query and processed data retrieval

An application aggregates one or more than one AAL profile(s) and offload to the AALI implementation using a single AALI API invocation. As one example, for high-PHY AAL profiles defined in Chapter 5, multiple AAL profiles (where an AAL profile refers to a PHY channel/signal for one or more than one cell(s) and one or more than one UE(s)) scheduled within a slot can be aggregated and offloaded to an AAL-LPU by the application using a single AALI API invocation.

The processing status of offloaded/enqueued AAL profile(s) can be queried by the application in an ‘asynchronous’ manner, i.e., not necessarily in the same order in which the AAL profile(s) are offloaded. In case the application retrieves the post-processed data from the AALI implementation, a ‘processing status query’ request can be bundled with a ‘processed data retrieval/dequeue’ request. In general, status query and dequeue request corresponding to multiple enqueue requests can be bundled together by the application and invoked through a single AALI API function.