Data Plane Development Kit Reference Manual

Intel FPGA Programmable Acceleration Card N3000

Online Version
Send Feedback

MNL-1107

ID: 683512
Version: 2019.12.06

Contents

1. About this Document

1.1. About this Document

This reference manual introduces the IFPGA Rawdev Driver (`ifpga_rawdev`) that is available for the Intel® FPGA PAC N3000. It introduces the data structures and API that the Open Programmable Acceleration Engine (OPAE) User mode driver (UMD) provides for FPGA management.

The intended audience for this reference manual is software engineers interested in using and customizing this IFPGA Rawdev Driver and the API that the OPAE user mode driver provides. Refer to the existing DPDK documentation for information about other Data Plane Development Kit (DPDK) functionality.

1.2. Acronym List

Acronym	Expansion	Description
Intel FPGA PAC	Intel FPGA Programmable Acceleration Card	Intel FPGA PAC N3000 is a full-duplex 100 Gbps in-system re-programmable acceleration card for multiworkload networking application acceleration.
AFU	Accelerator Functional Unit	Hardware Accelerator implemented in FPGA logic which offloads a computational operation for an application from the CPU to improve performance.
AF	Acceleration Function	Compiled Hardware Accelerator image implemented in FPGA logic that accelerates an application.
API	Application Programming Interface	A set of subroutine definitions, protocols, and tools for building software applications.
DPDK	Data Plane Development Kit	The Data Plane Development Kit consists of libraries to accelerate packet processing workloads running on many CPU architectures, including x86, POWER and ARM processors. DPDK runs mostly on Linux with a FreeBSD port available for a subset of DPDK features. The Open Source BSD License DPDK licenses DPDK.
FIU	FPGA Interface Unit	FIU is a platform interface layer that acts as a bridge between platform interfaces like PCIe* and AFU-side interfaces such as CCI-P.
OPAE	Open Programmable Acceleration Engine	The OPAE is a set of drivers, utilities, and API's for managing and accessing AFs.

2. Data Plane Development Kit Overview

This reference manual describes the DPDK software components and API that support the Intel FPGA Programmable Acceleration Card N3000 and Intel Arria® 10 FPGA. These software components build on the existing DPDK software.

The framework creates a set of libraries that your application can link to. Here are some examples of the available libraries:

- A packet forwarding algorithm that uses a hash-based approach to a longest prefix match (LPM) library
- A network buffer management (Mbuf library)
- A traffic management API

DPDK also provides APIs for the following network functions:

- Ethernet API to receive and transmit packets and configure network interfaces
- Baseband API for virtual radio access network (vRAN)
- Quality of service API configuration
- Metering API for counter support
- Additional basic functions such as locking, memory allocation, and CPU core synchronization

The Poll Mode Driver (IPN3KE) and `testpmd` application that support the Intel FPGA PAC N3000 platform use the Ethernet device DPDK API. Refer to DPDK 19.08.0 Functions for information about Ethernet receive and transmit packets.

The DPDK API functions provide a hardware abstraction layer so that you can easily port your application to different hardware platforms. If you create your own custom driver, you must provide functions that override DPDK API interfaces. If the required functionality or hardware acceleration cannot override DPDK API, you can add a new DPDK API functions.

Related Information

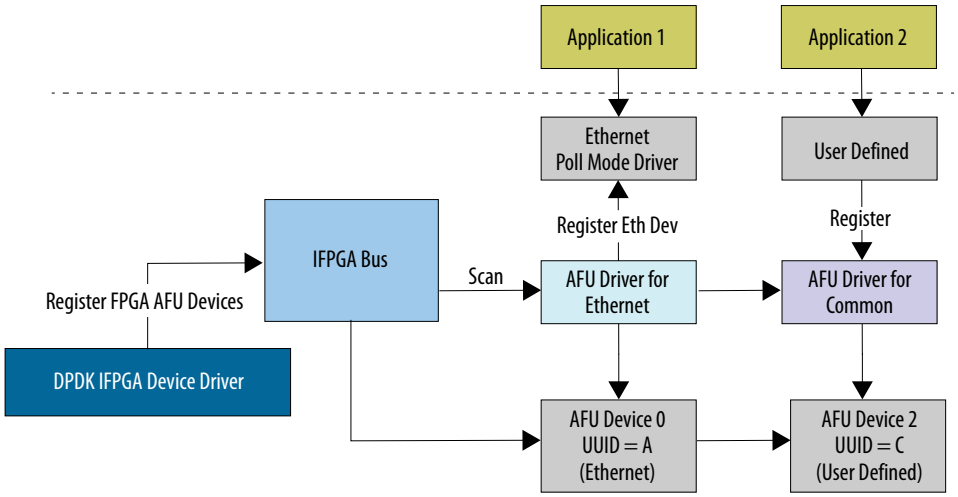
- Intel Acceleration Stack User Guide: Intel FPGA Programmable Acceleration Card N3000
  For information about using the Intel FPGA Programmable Acceleration Card N3000, including operating system support, hardware installation, software installation, sample testing, installing drivers, and a network loopback test.
- DPDK Documentation
  For comprehensive information about DPDK, including sample applications, a programmer's guide, tools guide, and device drivers.

2.1. DPDK Intel FPGA PAC N3000 Software Overview

The DPDK Intel FPGA PAC N3000 includes the following components:

- The Poll Mode Driver (IPN3KE) is a user-mode driver for the Intel Ethernet Controller XL710 and Intel Arria 10 FPGA. This driver implements the `rte_ethdev` API. This Poll Mode Driver works with the Intel-provided FPGA factory image.
- The FPGA Device Driver (IFPGA Rawdev) binds to the Intel Arria 10 FPGA on the Intel FPGA PAC N3000. The IFPGA driver calls API functions that the Open Programmable Acceleration Engine (OPAE) User Mode Driver (UMD) exports to enumerate and discover the features of the FPGA and Accelerator Functional Unit (AFU). In addition, the `IFPGA_Rawdev` driver works in coordination with the Open Programmable Acceleration Engine (OPAE) share code to provide the following common FPGA management functions:
  - AFU identification
  - Thermal management
  - Retimer link status and statistics
- The Intel FPGA PAC N3000 supports the following Ethernet configurations:
  - 8x10 (4 lanes in each quad small form factor pluggable (QSFP), 40 (GbE))
  - 2x2x25 (2 lanes in each QSFP, 25 GbE)
  - 4x25 (4 lanes in one of the QSFP, 25 GbE)
- The OPAE UMD provides interfaces for user space applications to configure, enumerate, open, and access Intel FPGA AFUs.
- The IFPGA bus provides a mechanism for AFU devices to register on the bus. The IFPGA bus logic compares the AFU universal unique identifier (UUID) to the UUIDs of registered AFU drivers. If a driver matches the AFU UUID, the AFU driver probe routine runs. The AFU driver for the default Intel-provided factory image loaded on the FPGA is IPN3KE PMD.

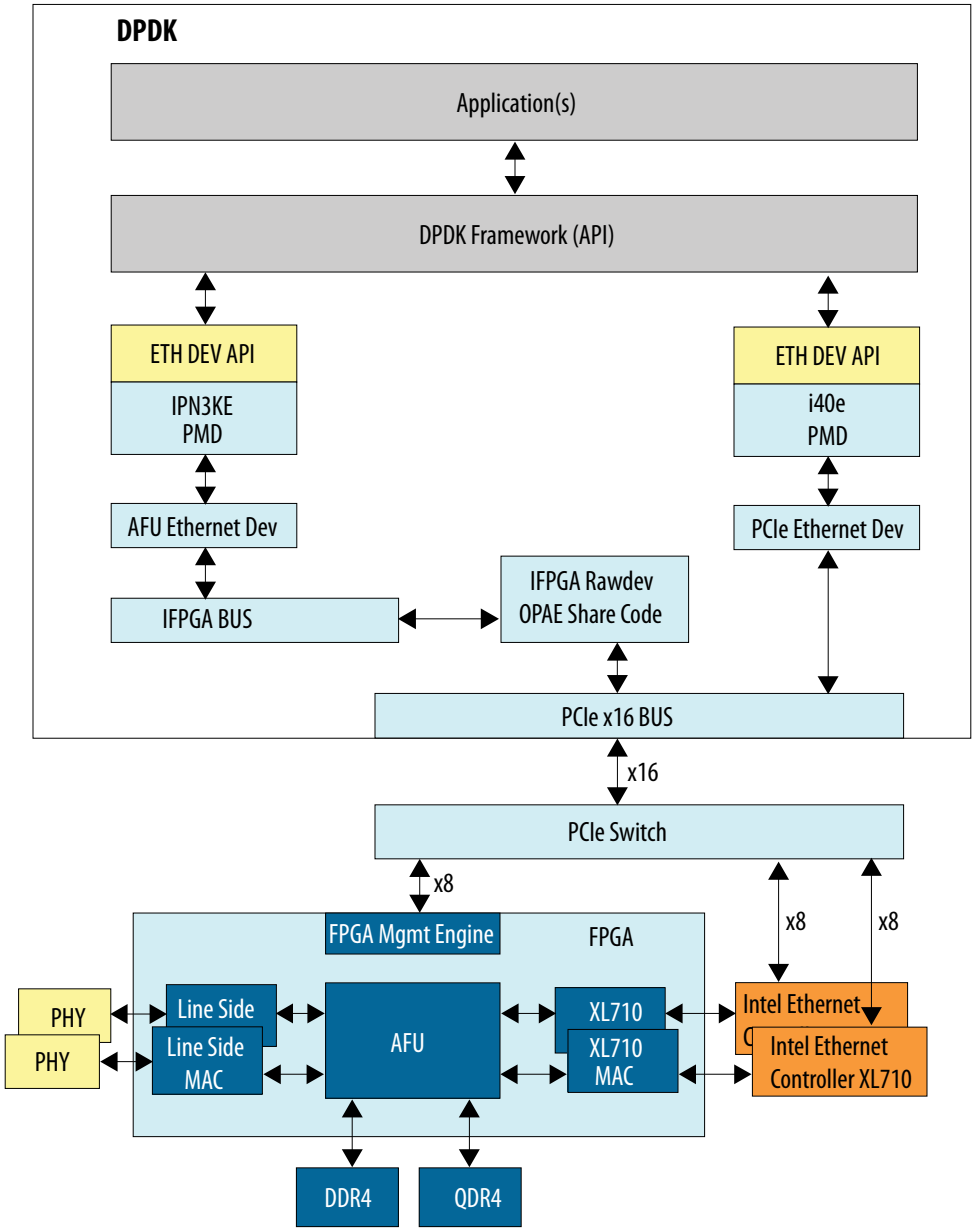
Figure 1. DPDK IFPGA Bus Scan



When the DPDK application runs, this application performs two scans:

- PCIe scan—A kernel module, such as `vfio-pci` driver, performs this scan. This scan is a legacy DPDK PCIe device scan which discovers the Intel Arria 10 FPGA as a PCIe device. This scan uses the `/sys` kernel interface to read the address space and map to the application space.
- AFU scan—This scan adds the AFU discovered to IFPGA bus. The `ifpga bus` scans the AFU device, identifies and then calls the corresponding AFU driver probe routine.

Figure 2. DPDK Line Side and Host Side Status Representation



The MACs inside the Intel Arria 10 FPGA connect to both line side and host side Ethernet ports. The line side port performs retiming and connects to the Ethernet network backbone. The host side port is the Intel Ethernet Controller XL710. DPDK software provides line side statistics and link status from the IPN3KE driver and gets the host side statistics from the XL710 Ethernet controller. The Ii40e PMD identifies the available XL710 ports and provides statistics by reading the XL710 MAC. The IPN3KE PMD probes the corresponding line side ports to get link status and statistics. The IPN3KE PMD probes the line side ports as port representors.

For a general overview of port-representors refer to Port Representors in *Data Plane Development Kit 18.05.1 Release*.

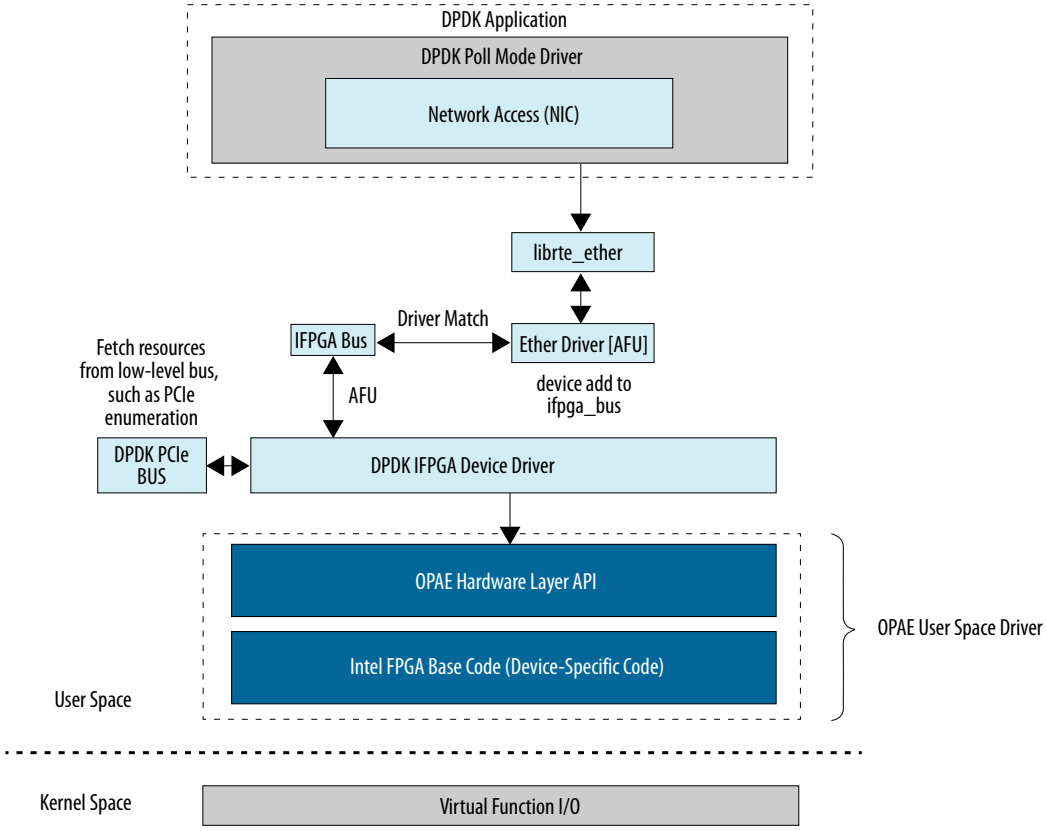
Related Information

- Intel FPGA Programmable Acceleration Card N3000: Overview
  For an overview the Intel FPGA PAC N3000 hardware and supported network function virtualization (NFV) applications.
- Intel FPGA Programmable Acceleration Card N3000: Documentation
  For a product brief, information on 5G requirements and solutions, and building a proof of concept segment router.
- IPN3KE Poll Mode Driver
  For information about the Intel Ethernet Controller XL710 and the Intel Arria 10 FPGA on the Intel FPGA PAC N3000.
- IFPGA Rawdev Driver
  For an overview of the IFPGA Rawdev Driver for the Intel Arria 10 FPGA.
- DPDK 17.02.1
  For information about the IFPGA bus interface.
- DPDK - Contribute
  For DPDK usage discussions, development discussions, and a bug archive.
- DPDK Documentation
  For comprehensive information about DPDK, including sample applications, programmer's guide, tools guide, and device drivers.

2.2. DPDK Intel FPGA PAC N3000 Software Components

The kernel space uses virtual function I/O (VFIO) to report PCIe resources to the user space. The VFIO driver advertises direct device access to the user space. Direct access to user space significantly reduces latency, increases bandwidth, and supports direct use of the BareMetal device driver.

Figure 3. DPDK for Intel FPGA PAC N3000 Software Architecture



The following components make up the DPDK Intel FPGA PAC N3000 stack. The code path is relative to $RTE_SDK which is the extracted DPDK 19.08 with the patch directory. Refer to the Intel Acceleration Stack User GuideIntel FPGA Programmable Acceleration Card N3000 for instructions on installing DPDK with the patch.

- The DPDK PCIe scan enumerates the PCIe bus during initialization.
- The OPAE hardware layer is a common hardware base abstraction layer. This abstraction layer provides hardware function calls and data structures for the FPGA, such as device feature list enumeration.
- The IFPGA base code specifies the FPGA hardware capability, including FPGA enumeration, and FPGA register access. The code is here: $RTE_SDK/`drivers/raw/ifpga/base/`
- The IFPGA bus is new in the DPDK. The DPDK software registers all AFU devices on this bus. The code is here: `$RTE_SDK/drivers/bus/ifpga/`

- The Rawdev library defines the raw device type for acceleration devices. For FPGAs the IFPGA Rawdev driver FPGA calls the API that the IFPGA exports to enumerate and discover the management engine (FME) and AFU region. The implementation is here: `$RTE_SDK/drivers/raw/ifpga/ifpga_rawdev.c`.
- The `afu_device` defines a data structure to describe the AFU device, providing access to functions such as memory-mapped I/O (MMIO) read and write.
- The IPN3KE PMD provides poll mode driver support for the Intel FPGA PAC N3000. The IPN3KE PMD implements functions that gather the XL710 host side link statistics by making function calls to I40e driver and gathers line side link status and statistics using IFPGA Rawdev driver. This driver is the Ether (Ethernet) driver shown in Figure 3 on page 8.

Related Information

- IFPGA Rawdev Driver
- I40E Poll Mode Driver
- Intel Acceleration Stack User GuideIntel FPGA Programmable Acceleration Card N3000

2.3. FPGA Scan and Enumeration

The following sections describe the Intel FPGA PAC N3000 factory image layout and the functions that the OPAE implements in the user mode driver to abstract the features and functionality of the Intel Arria 10 FPGA.

The static region of the FPGA factory image contains a device feature list within the MMIO space. The software walks the feature list to enumerate the following resources and features.

- FME device
- AFU

The `ifpga` base code calls the `ifpga_bus_enumerate()` function to walk the device feature list. After enumeration completes, `ifpga_hw` data structure contains the FME device and device port information.

2.4. OPAE User Mode Driver

The `ifpga rawdev` driver, `$RTE_SDK/drivers/raw/ifpga/ifpga_rawdev.c`, defines the FPGA. This driver uses all the API functions described in this section to retrieve information from FPGA components.

The OPAE Hardware Layer API uses the IFPGA base code data structures defined in table below. These data structures abstract away the detailed hardware implementation to allow communication between IPN3KE User Mode Driver and FPGA base code. DPDK PMD or application can use the OPAE hardware layer API to communicate with the FPGA.

Figure 4. DPDK Software Applications Layered on OPAE Software and Hardware

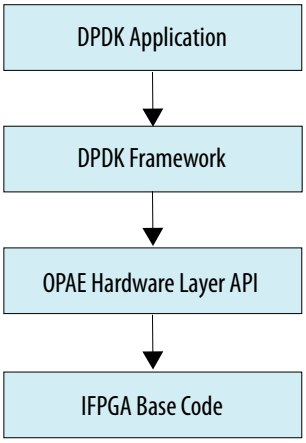
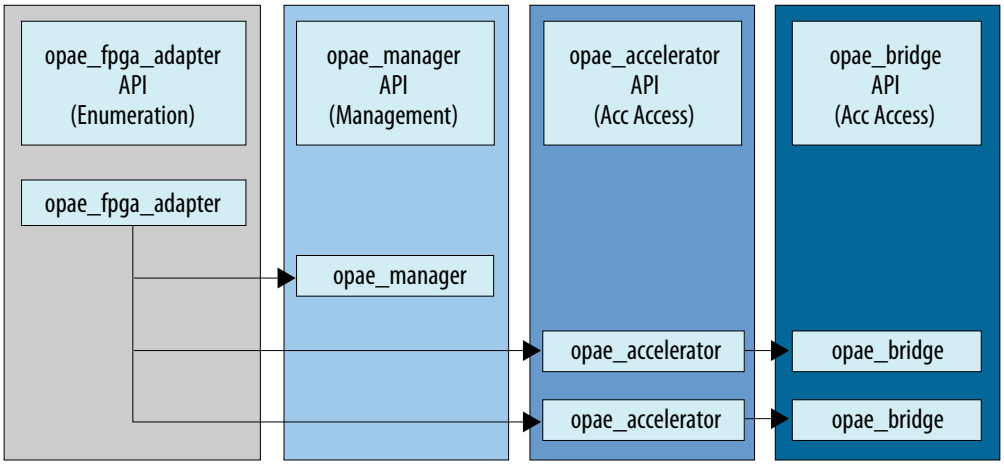


Table 1. Intel FPGA Base Code Data Structures

Data Structure Name	Description
`opae_fpga_adapter`	Represents an FPGA device for user mode device drivers. The DPDK software uses this structure for FPGA enumeration.
`opae_manager`	Represents an FPGA management unit such as the FME. Stores information about FPGA functions such as thermal management, error reporting, and retimer status reporting.
`opae_accelerator`	Represents an Accelerator Function (AF).
`opae_bridge`	Provides a link between the FPGA Management Unit and AF.

Figure 5. OPAE Hardware Layer API Architecture



2.4.1. OPAE FPGA Adapter

The `opae_fpga_adapter` uses the following data structures and defines the function calls described below.

The `$RTE_SDK/drivers/raw/ifpga_rawdev/base/opae_hw_api.h` header file defines the following data structures.

```
struct opae_adapter {
   const char *name;
   struct opae_manager *mgr;
   struct opae_accelerator_list acc_list;
   struct opae_adapter_ops *ops;
   void *data;
```

Table 2. opae_adapter Struct Field Definitions

Data Structure Field Name	Description
`name`	A pointer to the name of the `opae_adapter`.
`mgr`	Pointer to the `opae_manager` data structure.
`acc_list`	A list of AFs.

`ops`	A pointer to struct `opae_adapter_ops` which defines operations that the OPAE adapter provides. The list of operations is dynamic.
	`struct opae_adapter_ops {`
	`  int (*enumerate)(struct opae_adapter *adapter);`
	`  void (*destroy)(struct opae_adapter *adapter);`
	`};`

`data`	A pointer to private data of the `opae_adapter`.

The OPAE PCIe adapter defines the following data structure.

```
struct opae_adapter_data_pci {
    enum opae_adapter_type type;
    u16 device_id;
    u16 vendor_id;
    struct opae_reg_region region[PCI_MAX_RESOURCE];
    int vfio_dev_fd; /* VFIO device file descriptor */
```

The opae_fpga_adapter implements the following API functions:

2.4.1.1. opae_adapter_init()

Prototype:	`int opae_adapter_init(struct opae_adapter *adapter, const char *name, void *data);`
Arguments:	`opae_adapter`: Pointer to the OPAE adapter.
	`name`: Pointer to a name.
	`data`: Pointer to associated data.
Returns:	Int.
Description:	Initializes an OPAE adapter instance.
Related Defines	`#define opae_adapter_free(adapter) opae_free(adapter)`

2.4.1.2. opae_adapter_data_alloc()

Prototype:	`void *opae_adapter_data_alloc(enum opae_adapter_type type);`
Arguments:	`opae_adapter_type`: Specifies the adapter type. PCI is the only valid `opae_adapter_type`.
Returns:	Void.
Description:	Allocates `opae_adapter_data` and passes `opae_adapter_data` to `opae_adapter_init()`.
Related Defines	`#define opae_adapter_data_free(data) opae_free(data)`

2.4.1.3. opae_adapter_enumerate()

Prototype:	`int opae_adapter_enumerate(struct opae_adapter *adapter);`
Arguments:	`opae_adapter`: Pointer to the OPAE adapter.
Returns:	`Int`.
Description:	Reads the FPGA device feature list. When complete, the DPDK software has a list of the available features.
Related Defines	None.

2.4.1.4. opae_adapter_destroy()

Prototype:	`int opae_adapter_destroy(struct opae_adapter *adapter);`
Arguments:	`opae_adapter`: Pointer to the OPAE adapter.
Returns:	Void
Description:	Destroys the OPAE adapter.
Defines	None.

2.4.2. OPAE Manager

OPAE Data Structures

The `$RTE_SDK/drivers/raw/ifpga_rawdev/base/opae_hw_api.h` header file defines the following data structures.

```
struct opae_manager {
   const char *name;
   struct opae_adapter *adapter;
   struct opae_manager_ops *ops;
   struct opae_manager_networking_ops *network_ops;
   void *data;
```

Table 3. opae_manager Struct Field Definitions

Data Structure Field Names	Description
`name`	A pointer to the name of the `opae_manager`.
`adapter`	A pointer to the `opae_adapter` data structure.

continued...

Data Structure Field Names	Description

ops	A pointer to the opae_manager_ops data structure which defines the operations that the OPAE manager provides:
	`struct opae_manager_ops {`
	`int (*flash)(struct opae_manager *mgr, int id,`
	`const char *buffer, u32 size, u64 *status);`
	`int (*get_eth_group_region_info)`
	`(struct opae_manager *mgr,`
	`struct opae_eth_group_region_info *info);`
	`int (*get_sensor_value)(struct opae_manager *mgr,`
	`struct opae_sensor_info *sensor,`
	`unsigned int *value);`
	`int (*get_board_info)(struct opae_manager *mgr,`
	`truct opae_board_info **info);`
	`};`

opae_manager_networking_ops	A pointer to the opae_manager_networking_ops data structure which defines the networking functions required for networking on the Intel FPGA PAC N3000:
	`struct opae_manager_networking_ops {`
	`int (*read_mac_rom)(struct opae_manager *mgr, int offset,`
	`void *buf,int size);`
	`int (*write_mac_rom)(struct opae_manager *mgr, int offset,`
	`void *buf, int size);`
	`int (*get_eth_group_nums)(struct opae_manager *mgr);`
	`int (*get_eth_group_info)(struct opae_manager *mgr,`
	`u8 group_id, struct opae_eth_group_info *info);`
	`int (*eth_group_reg_read)(struct opae_manager *mgr, u8 group_id,`
	`u8 type, u8 index, u16 addr, u32 *data);`
	`int (*eth_group_reg_write)(struct opae_manager *mgr, u8 group_id,`
	`u8 type, u8 index, u16 addr, u32 data);`
	`int (*get_retimer_info)(struct opae_manager *mgr,`
	`struct opae_retimer_info *info);`
	`int (*get_retimer_status)(struct opae_manager *mgr,`
	`struct opae_retimer_status *status);`
	`};`

data	A pointer to opae_manager private data.

The OPAE manager exports the following API functions:

2.4.2.1. opae_manager_alloc()

Prototype:	`opae_manager_alloc(const char *name, struct opae_manager_ops *ops, void *data);`
Arguments:	name: A pointer an identifying name.

	opae_manager_ops: A pointer the opae_manager_ops data structure which defines operations that the OPAE manager provides:
	`struct opae_manager_ops {`
	`int (*flash)(struct opae_manager *mgr, int id,`
	`const char *buffer, u32 size, u64 *status);`
	`int (*get_eth_group_region_info)`
	`(struct opae_manager *mgr,`
	`struct opae_eth_group_region_info *info);`
	`int (*get_sensor_value)(struct opae_manager *mgr,`
	`struct opae_sensor_info *sensor,`
	`unsigned int *value);`
	`int (*get_board_info)(struct opae_manager *mgr,`
	`truct opae_board_info **info);`

	data: A pointer to private data.
Returns:	opae_manager.
Description:	Initializes an OPAE manager instance.
Related Defines	`#define opae_manager_free(mgr) opae_free(mgr)`

2.4.2.2. opae_manager_flash()

Prototype:	`int opae_manager_flash(struct opae_manager *mgr, int id, void *buf, u32 size, u64 *status);`
Arguments:	`name`: A pointer to the `opae_manager`
	`id`: An ID.
	`buf`: A pointer to a buffer.
	`size`: The size of the buffer which is the size of the configuration bitstream.
	`status`: The hardware status, including the PR error code when the return value is -EIO.
Returns:	`Int`.
Description:	Store a configuration bitstream for FPGA reconfiguration in flash memory using the `opae_manager`. The Intel FPGA PAC N3000 does not support partial reconfiguration (PR).
Related Defines	`None`.

2.4.2.3. opae_manager_get_eth_group_region_info()

Prototype:	`int opae_manager_get_eth_group_region_info(struct opae_manager *mgr, u8 group_id, struct opae_eth_group_region_info *info);`
Arguments:	`name`: A pointer to the `opae_manager`.
	`group_id`: An ID of for the Ethernet group.

	`opae_eth_group_region_info`: A pointer to the data structure containing Ethernet sensor information.
	`get_eth_group_region_info = ifpga_mgr_get_eth_group_region_info,`
	`.get_sensor_value = ifpga_mgr_get_sensor_value,`
	`};`

Returns:	`Int`.
Description:	Get information about Ethernet groups.
Related Defines	`None`.

2.4.2.4. Sensor Information

The Intel MAX® 10 Board Management Controller monitors approximately 19 sensors. The OPAE Manager enumerates all the sensors and adds the sensors to a global list. The FPGA reads this information through its SPI interface.

The following data structure describes the sensor info:

```
struct opae_sensor_info {
        TAILQ_ENTRY(opae_sensor_info) node;
        const char *name;
        const char *type;
        unsigned int id;
        unsigned int high_fatal;
        unsigned int high_warn;
        unsigned int low_fatal;
        unsigned int low_warn;
        unsigned int hysteresis;
        unsigned int multiplier;
        unsigned int flags;
```

```
        unsigned int value;
        unsigned int value_reg;
};
```

Table 4. opae_sensor_info Fields

Data Structure Field Name	Description
name	A pointer to the name of the sensor.
type	The sensor type, such as temperature sensor or voltage sensor.
id	The Intel FPGA PAC N3000 hardware ID for this sensor.
high_fatal	Specifies a fatal threshold for a sensor.
high_warn	Specifies a warning threshold for a sensor.
low_fatal	Specifies the low fatal threshold for a sensor.
low_warn	Specifies the low warning threshold for a sensor.
hysteresis	Specifies the hysteresis of a sensor.
multiplier	Specifies the multiplier for a sensor.
flags	Specifies sensor flags.
value	Specifies sensor data.
value_reg	Specifies the sensor register in the Intel MAX 10 device.

2.4.2.5. opae_mgr_get_sensor_value_by_name()

Prototype:	`int opae_mgr_get_sensor_value_by_name(struct opae_manager *mgr, const char *name, unsigned int *value);`
Arguments:	`mgr`: A pointer to the `opae_manager`.
	`name`: A pointer to the name of the sensor.
	`value`: A pointer to the value of the sensor.
Returns:	`Int`.
Description:	Gets the value of the specified sensor by looking up the name in the global sensor list.
Related Defines	None.

2.4.2.6. opae_mgr_get_sensor_value_by_id()

Prototype:	`int opae_mgr_get_sensor_value_by_id(struct opae_manager *mgr, unsigned int id, unsigned int *value);`
Arguments:	`name`: A pointer to the `opae_manager`.
	`name`: A pointer to the ID of the sensor.
	`value`: A pointer to the value of the sensor.
Returns:	`Int`.
Description:	Gets the value of the specified sensor by looking up the ID in the global sensor list.
Related Defines	None.

2.4.2.7. opae_mgr_get_sensor_value()

Prototype:	`int opae_mgr_get_sensor_value(struct opae_manager *mgr, struct opae_sensor_info *sensor, unsigned int *value);`
Arguments:	`mgr`: A pointer to the `opae_manager`.
	`sensor`: A pointer to the `opae_sensor_info` struct. `struct opae_sensor_info *opae_mgr_get_sensor_by_id(unsigned int id);`
	`value`: A pointer to the sensor value.
Returns:	`Int`.
Description:	Gets the value of the specified sensor using the `opae_sensor_info` struct.
Related Defines	`None`.

2.4.2.8. opae_manager_networking_ops

The `opae_manager_networking_ops` data structure includes all functions required for networking operations on the Intel FPGA PAC N3000, including the following operations:

- Reading Ethernet group registers
- Reading the link speed
- Reading the link status

The `opae_manager_networking_ops` incorporates the following API functions:

```
struct opae_manager_networking_ops {
int (*read_mac_rom)(struct opae_manager *mgr, int offset, void *buf, int size);
int (*write_mac_rom)(struct opae_manager *mgr, int offset, void *buf, int size);
int (*get_eth_group_nums)(struct opae_manager *mgr);
int (*get_eth_group_info)(struct opae_manager *mgr, u8 group_id,
    struct opae_eth_group_info *info);
int (*eth_group_reg_read)(struct opae_manager *mgr, u8 group_id,
    u8 type, u8 index, u16 addr, u32 *data);
int (*eth_group_reg_write)(struct opae_manager *mgr, u8 group_id,
    u8 type, u8 index, u16 addr, u32 data);
int (*get_retimer_info)(struct opae_manager *mgr,
    struct opae_retimer_info *info);
int (*get_retimer_status)(struct opae_manager *mgr,
    struct opae_retimer_status *status);
```

The following table summarizes the information that these API functions read or write.

Table 5. Networking API Functions

Function	Description
`read_mac_rom`	Read raw data from the I2C EEPROM.
`write_mac_rom`	Write raw data to the MAC ROM.
`get_eth_group_nums`	Get the numbers of the Ethernet group.
`get_eth_group_info`	Get the configuration information of an Ethernet group.
`eth_group_reg_read`	Read the register of the Ethernet group.

continued...

Function	Description
eth_group_reg_write	Write the register of the Ethernet group.
get_retimer_info	Get retimer information.
get_retimer_status	Get retimer link status information.

For more information about Ethernet groups, refer to *Ethernet Group and Retimer API*.

Related Information

Ethernet Group and Retimer API on page 25

2.4.3. OPAE Accelerator

The OPAE opae_accelerator uses the following data structures that $RTE_SDK/drivers/raw/ifpga_rawdev/base/opae_hw_api.h defines.

```
struct opae_accelerator {
    TAILQ_ENTRY(opae_accelerator) node;
    const char *name;
    int index;
    struct opae_bridge *br;
    struct opae_manager *mgr;
    struct opae_accelerator_ops *ops;
    void *data;
    };
```

Table 6. opae_accelerator Struct Field Definitions

Data Structure Field Name	Description
name	A pointer to the name of the opae_accelerator.
index	An index value specifying an Ethernet port.

br	A pointer to the opae_bridge data structure:
	`struct opae_bridge {`
	`  const char *name;`
	`  int id;`
	`  struct opae_accelerator *acc;`
	`  struct opae_bridge_ops *ops;`
	`  void *data;`
	`  };`

mgr	A pointer to the opae_manager data structure:
	`struct opae_manager {`
	`  const char *name;`
	`  struct opae_adapter *adapter;`
	`  struct opae_manager_ops *ops;`
	`  struct opae_manager_networking_ops *network_ops;`
	`  void *data;`
	`};`

ops	A pointer to the opae_accelerator_ops data structure which defines the operations that the OPAE accelerator provides:
	`struct opae_accelerator_ops {`
	`  int (*read)(struct opae_accelerator *acc, unsigned`
	`  int region_idx, u64 offset, unsigned int byte, void *data);`
	`  int (*write)(struct opae_accelerator *acc,`
	`  unsigned int region_idx, u64 offset, unsigned int byte,`
	`  void *data);`
	`  int (*get_info)(struct opae_accelerator *acc,`
	`  struct opae_acc_info *info);`
	`  int (*get_region_info)(struct opae_accelerator *acc,`

continued...

Data Structure Field Name	Description
	struct opae_acc_region_info *info); int (*get_uuid)(struct opae_accelerator *acc, struct uuid *uuid); };
data	A pointer to private data of the opae_accelerator.

The opae_accelerator implements the following API functions:

2.4.3.1. opae_accelerator_alloc()

Prototype:	opae_accelerator_alloc(const char *name, struct opae_accelerator_ops *ops, void *data);
Arguments:	name: A pointer to the name of this accelerator.

	opae_accelerator_ops: A pointer to the opae_accelerator_ops data structure which defines the operations that the OPAE provides:
	struct opae_accelerator_ops {
	int (*read)(struct opae_accelerator *acc, unsigned int region_idx,
	u64 offset, unsigned int byte, void *data);
	int (*write)(struct opae_accelerator *acc, unsigned int region_idx,
	u64 offset, unsigned int byte, void *data);
	int (*get_info)(struct opae_accelerator *acc,
	struct opae_acc_info *info);
	int (*get_region_info)(struct opae_accelerator *acc,
	struct opae_acc_region_info *info);
	int (*get_uuid)(struct opae_accelerator *acc,
	struct uuid *uuid);
	};

	data: A pointer to private data.
Returns:	Void.
Description:	Initializes an OPAE manager instance.
Related Defines	#define opae_accelerator_free(acc) opae_free(acc)

2.4.3.2. opae_acc_get_uuid()

Prototype:	int opae_acc_get_uuid(struct opae_accelerator *acc, struct uuid *uuid);

Arguments:	acc: A pointer to the opae_accelerator.
	struct opae_accelerator {
	TAILQ_ENTRY(opae_accelerator) node;
	const char *name;
	int index;
	struct opae_bridge *br;
	struct opae_manager *mgr;
	struct opae_accelerator_ops *ops;
	void *data;
	};

	uuid: A pointer to uuid struc:
	struct uuid {
	u8 b[16];
	};

Returns:	Int.
Description:	Get an accelerator UUID.

2.4.3.3. opae_acc_get_region_info()

Prototype:	`int opae_acc_get_region_info(struct opae_accelerator *acc, struct opae_acc_info *info);`
Arguments:	`acc`: A pointer to the `opae_accelerator` struct. `struct opae_accelerator { TAILQ_ENTRY(opae_accelerator) node; const char *name; int index; struct opae_bridge *br; struct opae_manager *mgr; struct opae_accelerator_ops *ops; void *data; };`
	`info`: A pointer to `opae_acc_region_info` struct: `struct opae_acc_region_info { u32 flags; #define ACC_REGION_READ (1 << 0) #define ACC_REGION_WRITE (1 << 1) #define ACC_REGION_MMIO (1 << 2) u32 index; u64 phys_addr; u64 len; u8 *addr; };`
Returns:	`Int`.
Description:	Get accelerator memory region information that populates the `opae_acc_region_info` data structure.

2.4.3.4. opae_acc_get_info()

Prototype:	`int opae_acc_info(struct opae_accelerator *acc, struct opae_acc_info *info);`
Arguments:	`acc`: A pointer to the `opae_accelerator` struct. `struct opae_accelerator { TAILQ_ENTRY(opae_accelerator) node; const char *name; int index; struct opae_bridge *br; struct opae_manager *mgr; struct opae_accelerator_ops *ops; void *data; };`
	`info`: A pointer to the `opae_acc_info` data structure.: `struct opae_acc_info { unsigned int num_regions; unsigned int num_irqs; };`
Returns:	`Int`.
Description:	Get information about this accelerator. The `opae_acc_info` stores this information.

2.4.4. OPAE Bridge

The $RTE_SDK/drivers/raw/ifpga/base/opae_hw_api.h header file defines the following opae_bridge data structures.

```
struct opae_bridge {
    const char *name;
    int id;
    struct opae_accelerator *acc;
    struct opae_bridge_ops *ops;
    void *data;
};
```

Table 7. opae_bridge Struct Field Definitions

Data Structure Field Names	Description
name	A pointer to the opae_manager name.
id	The index of the port number.
acc	A pointer to the opae_accelerator data structure: `struct opae_bridge { const char *name; int id; struct opae_accelerator *acc; struct opae_bridge_ops *ops; void *data; };`
ops	A pointer to the opae_bridge_ops which defines operations that the OPAE bridge provides: `struct opae_bridge_ops { int (*reset)(struct opae_bridge *br); };`
data	A pointer to private data of the opae_bridge.

The OPAE bridge exports the following API.

2.4.4.1. opae_bridge_alloc()

Prototype:	`struct opae_bridge * opae_bridge_alloc(const char *name, struct opae_bridge_ops *ops, void *data);`
Arguments:	name: A pointer to a name.
	A pointer to the opae_bridge_ops which defines operations that the OPAE bridge provides: `struct opae_bridge_ops { int (*reset)(struct opae_bridge *br); };`
	data: A pointer to private data.
Returns:	opae_bridge.
Description:	Initializes an OPAE bridge instance.
Related Defines	`#define opae_bridge_free(br) opae_free(br)`

2.4.4.2. opae_bridge_reset()

Prototype:	`int opae_bridge_reset(struct opae_bridge *br);`
Arguments:	`br`: A pointer to the `opae_bridge` data structure: `struct opae_bridge { const char *name; int id; struct opae_accelerator *acc; struct opae_bridge_ops *ops; void *data; };`
Returns:	`Int`.
Description:	Resets the port and accelerator.
Related Defines	`#define opae_bridge_free(br) opae_free(br)`

2.5. OPAE User Mode Driver API Summary

Here is a summary of the OPAE User Mode Driver. The `drivers/raw/ifpga/base/opae_hw_api.h` header file defines this API.

OPAE Adapter API

```
opae_adapter_free(adapter)
```

```
int opae_adapter_enumerate(struct opae_adapter *adapter);
```

```
void opae_adapter_destroy(struct opae_adapter *adapter);
```

```
void *opae_adapter_data_alloc(enum opae_adapter_type type);
```

OPAE Manager API

```
struct opae_manager *opae_manager_alloc(const char *name,
            struct opae_manager_ops *ops,
            struct opae_manager_networking_ops *network_ops, void *data);
```

```
opae_manager_free(mgr)
```

```
int opae_manager_get_eth_group_region_info(struct opae_manager *mgr,
                u8 group_id, struct opae_eth_group_region_info *info);
```

```
struct opae_sensor_info *opae_mgr_get_sensor_by_name(struct opae_manager *mgr,
                const char *name);
```

```
struct opae_sensor_info *opae_mgr_get_sensor_by_id(struct opae_manager *mgr,
                unsigned int id);
```

```
int opae_mgr_get_sensor_value_by_name(struct opae_manager *mgr,
                const char *name, unsigned int *value);
```

```
int opae_mgr_get_sensor_value_by_id(struct opae_manager *mgr,
                unsigned int id, unsigned int *value);
```

```
int opae_mgr_get_sensor_value(struct opae_manager *mgr,
                struct opae_sensor_info *sensor,
                unsigned int *value);
```

OPAE Accelerator API

```
struct opae_accelerator *
opae_accelerator_alloc(const char *name, struct opae_accelerator_ops *ops,
               void *data);
```

```
opae_accelerator_free(acc)
```

```
int opae_acc_get_info(struct opae_accelerator *acc, struct opae_acc_info *info);
```

```
int opae_acc_get_region_info(struct opae_accelerator *acc,
                 struct opae_acc_region_info *info);
```

```
int opae_acc_get_uuid(struct opae_accelerator *acc,
              struct uuid *uuid);
```

Register Read and Write Accelerator API

```
opae_acc_reg_read64(acc, region, offset, data)
opae_acc_reg_write64(acc, region, offset, data)
opae_acc_reg_read32(acc, region, offset, data)
opae_acc_reg_write32(acc, region, offset, data)
opae_acc_reg_read16(acc, region, offset, data)
opae_acc_reg_write16(acc, region, offset, data)
opae_acc_reg_read8(acc, region, offset, data)
opae_acc_reg_write8(acc, region, offset, data)
```

OPAE Bridge

```
struct opae_bridge * opae_bridge_alloc(const char *name,
            struct opae_bridge_ops *ops, void *data)
int opae_bridge_reset(struct opae_bridge *br)
#define opae_bridge_free(br) opae_free(br)
```

2.5.1. Example Code

The IFPGA Rawdev Driver, `drivers/raw/ifpga/ifpga_rawdev.c`, demonstrates how the OPAE User Mode Driver to use the APIs.

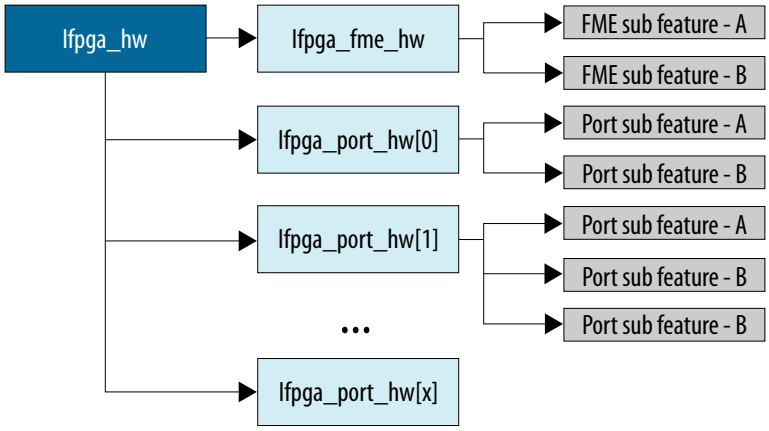
3. IFPGA Base Code Design

The IFPGA base driver provides an API for enumeration and a data structure for the enumeration result.

The Intel FPGA factory image implements a static region called the FPGA Interface Unit (FIU). This static region can have different features. To discover the FIU features, the IFPGA driver walks through the device feature list in the Intel Arria 10 FPGA device memory.

In the example below, after enumeration, the IFPGA code finds the FIU, port, accelerator and their sub features. The `ifpga_hw` data structure stores this information.

Figure 6. IFPGA Hardware Feature List



The `ifpga_hw` data structure stores this information.

```
struct ifpga_hw {
struct opae_adapter *adapter;
struct opae_adapter_data_pci *pci_data;
struct ifpga_fme_hw fme;
struct ifpga_port_hw port[MAX_FPGA_PORT_NUM];
};
```

3.1. FPGA Management

The FPGA management engine has different subfeatures to enable different functionality such as thermal monitoring and error reporting.

The `struct feature_driver fme_feature_drvs[]` in `$RTE_SDK/ifpga/base/ifpga_feature_dev.c` initializes all the subfeatures that FPGA management engine supports. Each of these subfeatures enables a set of operations. The operations have their own data structures defined in `$RTE_SDK/ifpga/base/ifpga_fme.c`.

The `ifpga_fme_hw` data structure stores this information.

```
struct ifpga_fme_hw {
    enum ifpga_fme_state state;
    struct ifpga_feature_list feature_list;
    spinlock_t lock; /* protect hardware access */
    void *parent; /* pointer to ifpga_hw */
    /* provided by HEADER feature */
    u32 port_num;
    struct uuid bitstream_id;
    u64 bitstream_md;
    size_t pr_bandwidth;
    u32 socket_id;
    u32 fabric_version_id;
    u32 cache_size;
    u32 capability;
    };
```

3.2. Accelerator Access and Control

The port hardware consists of an AFU subfeature to provide MMIO region access to the actual AFU.

The `ifpga_port_hw` data structure stores this information.

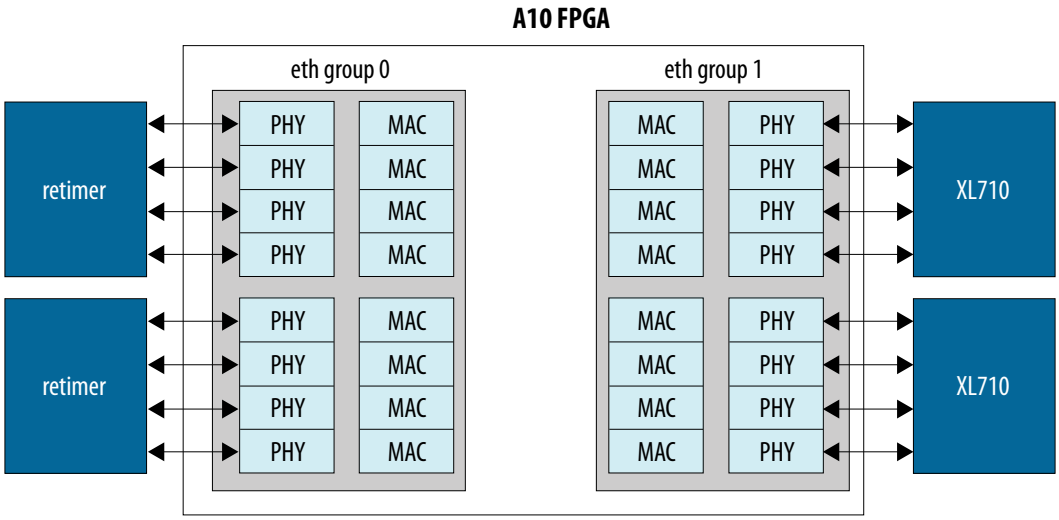
```
struct ifpga_port_hw {
enum ifpga_port_state state;
struct ifpga_feature_list feature_list;
spinlock_t lock; /* protect access to hw */
void *parent; /* pointer to ifpga_hw */
int port_id; /* from HEADER feature */
struct uuid afu_id; /* provied by User AFU feature */
unsigned int disable_count;
u32 capability;
u32 num_umsgs; /* The number of allocated umsgs */
u32 num_uafu_irqs; /* The number of uafu interrupts */
u8 *stp_addr;
u32 stp_size;
};
```

4. Ethernet Group and Retimer API

The Intel FPGA PAC N3000 defines two Ethernet groups. One group connects to the Intel Ethernet Controller XL710. The other group connects to the retimer. Each Ethernet group supports both the MAC and the PHY components.

Figure 7. Retimer and Line-Side Connectivity for 8x10 GbE

The following figure illustrates the high-level connectivity for retiming when you select 8x10 GbE mode.

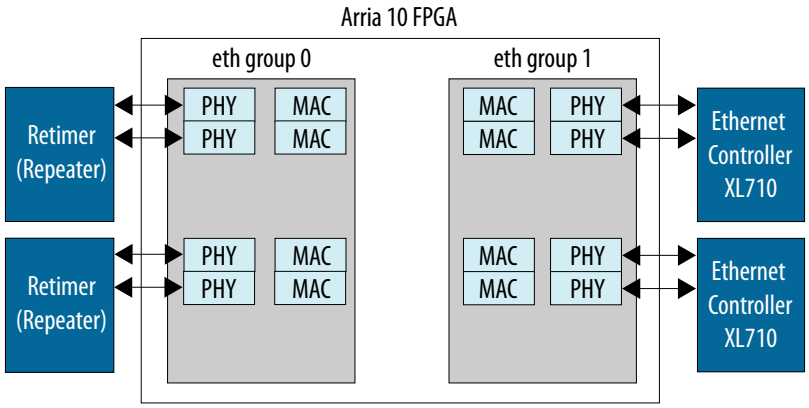


The 8x10G mode datapath includes the following components :

- Two retimers for 10 GbE each connected to 4 channels on the Intel Arria 10 FPGA
- Two Intel Ethernet Controller XL710s for 10 GbE, each connected to 4 channels on the Intel Arria 10 FPGA

The following figure illustrates the high-level connectivity for retiming when you select 2x2x25 GbE mode.

Figure 8. Retimer and Line-Side Connectivity for 2x2x25 GbE



The 2x2x25 GbE mode includes the following components for retiming:

- Two retimers for 25 GbE
- Two Intel Ethernet Controller XL710 for 2x40 GbE
- Two ports connected to each retimer
- Two ports connected to each Intel Ethernet Controller XL710

The following data structures store Ethernet group information:

```
struct opae_eth_group_info {
u8 group_id;
u8 speed;
u8 nums_of_phy;
u8 nums_of_mac;
};
```

Table 8. opae_eth_group_info Struct Field Definitions

Data Structure Field Name	Description
group_id	The group index id. The following values are valid for 8x10 GbE mode: • 0: The Ethernet group that connects to the retimers • 1: The Ethernet group that connects to the XL710.
speed	The supported speed of an Ethernet group. For example, on 2x2x25 GbE mode, group 0 supports 25 GbE and group 1 supports 40 GbE.
nums_of_phy	The number of PHYs in this group. For 8x10 GbE, there are 8 PHYs and MACs in each group. For 2x2x25 GbE, there are 4 PHYs and 4 MACs in each group.
nums_of_macs	The number of MACs in this group. For 8x10 GbE, there are 8 PHYs and MACs in each group. For 2x4x25 GbE, there are 4 PHYs and 4 MACs in each group.

```
struct opae_eth_group_region_info {
u8 group_id;
u64 phys_addr;
u64 len;
u8 *addr;
u8 mem_idx;
};
```

Table 9. opae_eth_group0_region_info Struct Field Definitions

Data Structure Field Name	Description
group_id	The group id for this register region.
phys_addr	The start physical address of this register region.
len	The length of this region.
addr	The virtual address of this region.
mem_idx	The memory index of the AFU.

The Ethernet group and retimer export the following API functions:

Related Information

opae_manager_networking_ops on page 16

4.1. opae_manager_get_eth_group_nums()

Prototype:	`int opae_manager_get_eth_group_nums(struct opae_manager *mgr, );`
Arguments:	`name`: A pointer to the `opae_manager`.
Returns:	`Int`.
Description:	Returns the number of Ethernet groups.
Related Defines	None.

4.2. opae_manager_get_eth_group_info()

Prototype:	`int opae_manager_get_eth_group_info(struct opae_manager *mgr, u8 group_id, struct opae_eth_group_info *info);`
Arguments:	`mgr`: A pointer to the `opae_manager` for the Ethernet group.
	`group_id`: The group ID index which is 0 for retimers and 1 for the XL710.

	A pointer to the `opae_eth_group_info`: data structure.
	`get_eth_group_info {`
	`u8 group_id;`
	`u8 speed;`
	`u8 nums_of_phy;`
	`u8 nums_of_mac;`
	`}`

Returns:	`Int`.
Description:	Get the configuration information for an Ethernet group.
Related Defines	None.

4.3. opae_manager_eth_group_write_reg()

Prototype:	`int opae_manager_eth_group_write_reg(struct opae_manager *mgr, u8 group_id, u8 type, u8 index, u16 addr, u32 *data); );`
Arguments:	`name`: A pointer to the `opae_manager` for the Ethernet group.
	`group_id`: The group ID index which is 0 for retimers and 1 for the XL710.
	`type`: The Ethernet type, MAC or PHY.
	`index`: Port index for the Ethernet group device.
	`addr`: Register address of the Ethernet group.
	`data`: The write data content.
Returns:	`Int`.
Description:	Writes registers in the PHY and MAC specified. This function is available to both AFU driver and client applications.
Related Defines	`None`.

4.4. opae_manager_eth_group_read_reg()

Prototype:	`int opae_manager_group_eth_read_reg(struct opae_manager *mgr, u8 group_id, u8 index, u8 type, u16 addr, u32 *data); );`
Arguments:	`name`: A pointer to the `opae_manager` for the Ethernet group.
	`group_id`: The group ID index which is 0 for retimers and 1 for the XL710.
	`type`: The Ethernet type, MAC or PHY.
	`index` : Port index for the Ethernet group device.
	`addr`: Register address of the Ethernet group.
	`data`: For reads, this is the buffer for the data read.
Returns:	`Int`.
Description:	Reads registers in the PHY and MAC specified. This function is available to both AFU driver and client applications.
Related Defines	`None`.

4.5. opae_manager_get_eth_group_region_info()

Prototype:	`int opae_manager_get_eth_group_region_info(struct opae_manager *mgr, u8 group_id, struct opae_eth_group_region_info *info); );`
Arguments:	`name`: A pointer to the `opae_manager` for flash memory.
	`group_id`: The Ethernet group ID.

	`opae_eth_group_region_info` : A pointer to the `opae_eth_group_region_info` data structure.
	`struct opae_eth_group_region_info {`
	`u8 group_id;`
	`u64 phys_addr;`
	`u64 len;`
	`u8 *addr;`
	`u8 mem_idx;`

Returns:	`Int`.
Description:	Returns register region information for specific Ethernet group.
Related Defines	None.

4.6. Data Structures for Retiming

The following data structures provide information about the Intel Ethernet Controller XL710 and retimers.

opae_retimer_info

```
/* retimer info */
struct opae_retimer_info {
unsigned int nums_retimer;
unsigned int ports_per_retimer;
unsigned int nums_fvl;
unsigned int ports_per_fvl;
enum retimer_speed support_speed;
};
```

Table 10. opae_retimer_info Struct Field Definitions

Data Structure Field Name	Description
`nums_retimer`	The number of retimers on the Intel FPGA PAC N3000. For 8x10 GbE mode this value is 2.
`ports_per_retimer`	The number of ports for each retimer. For8x10 GbE mode this value is 4.
`nums_fvl`	The number of Intel Ethernet Controller XL710 devices on the Intel FPGA PAC N3000. For 8x10 GbE mode this value is 2.
`ports_per_fv`	The number of ports on each Intel Ethernet Controller XL710. For 8x10 GbE mode, this value is 4.
`support_speed`	The speed of the retimers. For 8x10 GbE mode this value is 10 GbE.

retimer_speed enumeration

```
/* retimer speed */
enum retimer_speed {
MXD_1GB = 1,
MXD_2_5GB = 2,
MXD_5GB = 5,
MXD_10GB = 10,
```

```
MXD_25GB = 25,
MXD_40GB = 40,
MXD_100GB = 100,
MXD_SPEED_UNKNOWN,
};
```

opae_retimer_status

This data structure stores the link status of retimers.

```
/* retimer status*/
struct opae_retimer_status {
enum retimer_speed speed;
unsigned int line_link_bitmap;
/*
* retimer line link status bitmap:
* bit 0: Retimer0 Port0 link status
* bit 1: Retimer0 Port1 link status
* bit 2: Retimer0 Port2 link status
* bit 3: Retimer0 Port3 link status
* bit 4: Retimer1 Port0 link status
* bit 5: Retimer1 Port1 link status
* bit 6: Retimer1 Port2 link status
* bit 7: Retimer1 Port3 link status
*/
};
```

Table 11. opae_retimer_status Struct Field Definitions

Data Structure Field Name	Description
speed	Supported speed for retiming.
line_link_bitmap	The retiming line side link status. The line_link_bitmap is a bitmap variable.

4.7. opae_manager_get_retimer_info()

Prototype:	int opae_manager_get_retimer_info(struct opae_manager *mgr, u8 group_id, struct opae_retimer_info *info);
Arguments:	name: A pointer to the opae_manager for retiming.

	opae_retimer_status: A pointer to the opae_retimer_info data structure.
	struct opae_retimer_info {
	unsigned int nums_retimer;
	unsigned int ports_per_retimer;
	unsigned int nums_fvl;
	unsigned int ports_per_fvl;
	enum retimer_speed support_speed;
	};

Returns:	Int.
Description:	This function returns retimer information.
Related Defines	None.

4.8. opae_manager_get_retimer_status()

Prototype:	`int opae_manager_get_retimer_status(struct opae_manager *mgr, struct opae_retimer_status *status);`
Arguments:	`name`: A pointer to the `opae_manager` for retiming.

	`opae_retimer_status`: A pointer to the `opae_retimer_status` data structure:
	`struct opae_retimer_status {`
	`enum retimer_speed speed;`
	`unsigned int line_link_bitmap;`

Returns:	`Int`.
Description:	Returns information about the link status and speed of the retimer.
Related Defines	`None`.

5. Revision History for the Data Plane Development Kit Reference Manual: Intel FPGA PAC N3000

Document Version	Intel Acceleration Stack Version	Changes
2019.12.06	1.1	Initial release.