|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | | |  |
|  | | |
|  | | |
| ½Ô²穙k1 | | | |
|  | **FACS**  User Development Guide | | ¸½¼þ1-16K |
|  | |
| **Issue** | **01** |
| **Release Date** | **2017-01-17** |
| **Huawei Technologies Co., Ltd.** | |

|  |
| --- |
| Copyright © Huawei Technologies Co., Ltd. 2017. All rights reserved.  No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Huawei Technologies Co., Ltd.  Trademarks and Permissions  附件3-图 and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.  All other trademarks and trade names mentioned in this document are the property of their respective holders.  Notice  The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.  The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied. |

|  |  |
| --- | --- |
| Huawei Technologies Co., Ltd. | |
| Address: | Huawei Industrial Base  Bantian, Longgang  Shenzhen 518129  People's Republic of China |
| Website: | <http://e.huawei.com> |
| Email: | [support@huawei.com](mailto:Support@huawei.com) |

Contents

[1 Introduction to FACS Interfaces 1](#_Toc511120940)

[1.1 Overview 1](#_Toc511120941)

[1.2 Terminology 1](#_Toc511120942)

[1.3 Specifications 2](#_Toc511120943)

[1.3.1 PCIe Interfaces 2](#_Toc511120944)

[1.3.2 DDR Interfaces 2](#_Toc511120945)

[1.3.3 Queues 2](#_Toc511120946)

[1.3.4 BAR Space 2](#_Toc511120947)

[1.4 Hardware Description 2](#_Toc511120948)

[1.4.1 Hardware Resources 2](#_Toc511120949)

[1.4.2 Dynamic and Static Interfaces 3](#_Toc511120950)

[1.4.3 Interface Signals 4](#_Toc511120951)

[1.4.4 Interface Timing 11](#_Toc511120952)

[1.5 Interaction Mode Descriptions 11](#_Toc511120953)

[1.5.1 TX Interaction 12](#_Toc511120954)

[1.5.2 RX Interaction 13](#_Toc511120955)

[2 FACS High Performance Interface (DPDK) Description 14](#_Toc511120956)

[2.1 Shell Interface Description 14](#_Toc511120957)

[2.1.1 Overview 14](#_Toc511120958)

[2.1.2 Shell-to-UL BD Interface 14](#_Toc511120959)

[2.1.3 UL-to-Shell BD Interface 16](#_Toc511120960)

[2.1.4 Shell-to-UL Data Interface 17](#_Toc511120961)

[2.1.5 UL-to-Shell Data Interface 18](#_Toc511120962)

[2.1.6 AXI-Lite Interface 20](#_Toc511120963)

[2.1.7 HardAcc Description 23](#_Toc511120964)

[2.2 SDK Interface Description 23](#_Toc511120965)

[2.2.1 Common DPDK Interfaces for VF Devices 24](#_Toc511120966)

[2.2.2 rte\_eal\_init Failure 33](#_Toc511120967)

[2.3 Example 2 Detailed Design 35](#_Toc511120968)

[2.3.1 Overview 35](#_Toc511120969)

[2.3.2 Features 36](#_Toc511120970)

[2.3.3 HDK Building Description 36](#_Toc511120971)

[2.3.4 Terminology 37](#_Toc511120972)

[2.3.5 Design Constraints 37](#_Toc511120973)

[3 FACS General-Purpose Interface (SDAccel) Description 38](#_Toc511120974)

[3.1 FACS SDAccel Overview 38](#_Toc511120975)

[3.2 Terminology 38](#_Toc511120976)

[3.3 FACS SDAccel Implementation Scheme Description 38](#_Toc511120977)

[3.4 FACS SDAccel Development Description 41](#_Toc511120978)

[3.4.1 Host Code Development 41](#_Toc511120979)

[3.4.2 Kernel Code Development 41](#_Toc511120980)

[3.5 Example Design Description 41](#_Toc511120981)

# Introduction to FACS Interfaces

## Overview

This chapter describes acceleration design, including interfaces, syntax, usage examples, and hardware resources and interfaces available for Huawei FACS users.

## Terminology

FACS: indicates FPGA Accelerated Cloud Server, and FPGA is short for Field Programmable Gate Array.

HDK: indicates hardware develop kit, consisting of coding environment, simulation platform, automated compilation tools, and code encryption and debugging tools.

SDK: indicates software develop kit, consisting of application examples, hardware abstract interfaces, accelerator abstract interfaces, an accelerator driver, runtime, and a version management tool.

Shell: indicates static logic, which is provided by Huawei, including external interfaces such as PCIe and DDR4.

UL: indicates user logic, which is provided by a developer.

PF: indicates physical function.

VF: indicates virtual function.

DMA: indicates direct memory access.

HPI: indicates high performance interface.

BD: indicates buffer description.

PR: indicates partial reconfiguration.

AE: indicates acceleration engine.

BAR: indicates base address register.

AXI4: indicates ARM Advanced eXtensible Interface.

AXI4-Stream: indicates ARM Advanced eXtensible Stream Interface.

AXI4-Lite: indicates ARM Advanced eXtensible Lite Interface.

M-: indicates the master end of the AXI bus.

S-: indicates the slave end of the AXI bus.

DPDK: indicates data plane development kit.

PMD: indicates poll mode drivers.

## Specifications

### PCIe Interfaces

Each FPGA supports a PCIe x16 interface, with a maximum bandwidth of 110 Gbit/s.

### DDR Interfaces

Each FPGA supports four DDR4 channels. The single-channel capacity of a DDR4 is 16 GB, and the total capacity is 64 GB.

### Queues

One PF.

One VF.

Each VF supports eight queues.

### BAR Space

UL VF BAR space specifications:

BAR2 is 64-bit, and the space is 16 Mbytes.

BAR4 is 64-bit, and the space is 512 Mbytes.

## Hardware Description

### Hardware Resources



The following resources are available:

* One PCIe Gen3 x16 interface;
* Four DDR4 DIMM interfaces;
* Nine ETH300G interfaces (100 Gbit/s).

DDR partitioning:

* One DDR controller is placed in the static logic partition.
* Three DDR controllers are placed in the UL partition.
* A maximum of four DDR controllers can be used.

### Dynamic and Static Interfaces

The following figure shows the interfaces available for FPGA users:

****

The AXI4-Lite interface is used, and the data bit width is 32 bits. This interface implements register access and BAR2 space mapping functions.

The AXI4-Lite interface is used, and the data bit width is 32 bits. This interface implements register access and BAR4 space mapping functions. (This interface is reserved.)

The AXI4-Stream interface is used, and the data bit width is 256 bits. This interface implements the DMA BD function.

The AXI4-Stream interface is used, and the data bit width is 512 bits. This interface implements the DMA data function.

The AXI4 interface is used, and the data bit width is 64 bits. This interface implements the SH DDR read/write access function.

BSCAN and register interfaces are used. With a bit width of 1 bit and 16 bits respectively, DEBUG implements the XVC function while VLED implements the virtual LED function.

The AXI4 interface is used, and the data bit width is 64 bits. This interface implements the UL DDRA/DDRB/DDRD read/write access function.

The SERDES interface is used, and the data bit width can be customized. This interface implements the serial-to-parallel conversion function.

### Interface Signals

|  |  |  |  |
| --- | --- | --- | --- |
| Signal | Bit Width | I/O | Description |
| Global signals | | | |
| clk\_200m | 1 | I | Indicates the system working clock (200 MHz) |
| clk\_a | 1 | I | Indicates user working clock a (200 MHz) |
| clk\_b | 1 | I | Indicates user working clock b (200 MHz) |
| rst\_200m | 1 | I | Indicates the reset signal corresponding to the 200 MHz system working clock |
| rst\_a | 1 | I | Indicates the reset signal corresponding to user working clock a |
| rst\_b | 1 | I | Indicates the reset signal corresponding to user working clock b |
| AXI-S interface for UL-to-Shell read packet request | | | |
| ul2sh\_dmas2\_tlast | 1 | O | The TLAST sent from UL to static logic indicates the last cycle of the packet. |
| ul2sh\_dmas2\_tdata | 256 | O | Indicates the read packet request sent from UL to static logic. |
| ul2sh\_dmas2\_tkeep | 32 | O | The TKEEP byte modifier sent from UL to static logic indicates whether the contents of TDATA are processed as part of the data flow. |
| ul2sh\_dmas2\_tvalid | 1 | O | Indicates that dynamic logic transmits valid data. |
| sh2ul\_dmas2\_tready | 1 | I | Indicates that static logic is ready to receive data from UL. |
| AXI-S interface for UL-to-Shell write packet request | | | |
| ul2sh\_dmas3\_tlast | 1 | O | The TLAST sent from UL to static logic indicates the last cycle of the packet. |
| ul2sh\_dmas3\_tdata | 512 | O | Indicates the write packet request sent from UL to static logic. |
| ul2sh\_dmas3\_tkeep | 64 | O | The TKEEP byte modifier sent from UL to static logic indicates whether the contents of TDATA are processed as part of the data flow. |
| ul2sh\_dmas3\_tvalid | 1 | O | Indicates that dynamic logic transmits valid data. |
| sh2ul\_dmas3\_tready | 1 | I | Indicates that static logic is ready to receive data from UL. |
| AXI-S interface for Shell-to-UL BD returning | | | |
| sh2ul\_dmam0\_tlast | 1 | I | The TLAST sent from static logic to UL indicates the last cycle of the packet. |
| sh2ul\_dmam0\_tdata | 256 | I | Indicates the BDs returned from static logic to UL. |
| sh2ul\_dmam0\_tkeep | 32 | I | The TKEEP byte modifier sent from static logic to UL indicates whether the contents of TDATA are processed as part of the data flow. |
| sh2ul\_dmam0\_tvalid | 1 | I | Indicates that static logic transmits valid data. |
| ul2sh\_dmam0\_tready | 1 | O | Indicates that static logic is ready to receive data from UL. |
| AXI-S interface for Shell-to-UL data returning | | | |
| sh2ul\_dmam1\_tlast | 1 | I | The TLAST sent from static logic to UL indicates the last cycle of the packet. |
| sh2ul\_dmam1\_tdata | 512 | I | Indicates the data returned from static logic to UL. |
| sh2ul\_dmam1\_tkeep | 64 | I | The TKEEP byte modifier sent from static logic to UL indicates whether the contents of TDATA are processed as part of the data flow. |
| sh2ul\_dmam1\_tvalid | 1 | I | Indicates that static logic transmits valid data. |
| ul2sh\_dmam1\_tready | 1 | O | Indicates that static logic is ready to receive data from UL. |
| AXI4 interface between UL and DDRC | | | |
| ul2sh\_ddr\_awid | 4 | O | Indicates that dynamic logic accesses the ID of the write address channel of static logic DDRC. |
| ul2sh\_ddr\_awaddr | 64 | O | Indicates that dynamic logic accesses the address of the write address channel of static logic DDRC. |
| ul2sh\_ddr\_awlen | 8 | O | Indicates that dynamic logic accesses the valid cycles of the write address channel of static logic DDRC. |
| ul2sh\_ddr\_awsize | 3 | O | Indicates that dynamic logic accesses the valid bytes of each cycle of the write address channel of static logic DDRC. |
| ul2sh\_ddr\_awvalid | 1 | O | Indicates that dynamic logic's access to the write address channel of static logic DDRC is valid. |
| sh2ul\_ddr\_awready | 1 | I | Indicates that static logic is ready to access the write address channel of dynamic logic DDRC. |
| ul2sh\_ddr\_wid | 4 | O | Indicates that dynamic logic accesses the ID of the write data channel of static logic DDRC. |
| ul2sh\_ddr\_wdata | 512 | O | Indicates that dynamic logic accesses the address of the write data channel of static logic DDRC. |
| ul2sh\_ddr\_wstrb | 64 | O | Indicates that dynamic logic accesses the valid cycles of the write data channel of static logic DDRC. |
| ul2sh\_ddr\_wlast | 1 | O | Indicates that dynamic logic accesses the valid bytes of each cycle of the write data channel of static logic DDRC. |
| ul2sh\_ddr\_wvalid | 1 | O | Indicates that dynamic logic's access to the write data channel of static logic DDRC is valid. |
| sh2ul\_ddr\_wready | 1 | I | Indicates that static logic is ready to access the write data channel of dynamic logic DDRC. |
| sh2ul\_ddr\_bid | 4 | I | Indicates the ID of the write response channel between static logic and dynamic logic DDRC. |
| sh2ul\_ddr\_bresp | 2 | I | Indicates the response of the write response channel between static logic and dynamic logic DDRC. |
| sh2ul\_ddr\_bvalid | 1 | I | Indicates that the write response channel between static logic and dynamic logic DDRC is valid. |
| ul2sh\_ddr\_bready | 1 | O | Indicates that the write response channel between dynamic logic and static logic DDRC is ready. |
| ul2sh\_ddr\_arid | 4 | O | Indicates that dynamic logic accesses the ID of the read address channel of static logic DDRC. |
| ul2sh\_ddr\_araddr | 64 | O | Indicates that dynamic logic accesses the address of the read address channel of static logic DDRC. |
| ul2sh\_ddr\_arlen | 8 | O | Indicates that dynamic logic accesses the valid cycles of the read address channel of static logic DDRC. |
| ul2sh\_ddr\_arsize | 3 | O | Indicates that dynamic logic accesses the valid bytes of each cycle of the read address channel of static logic DDRC. |
| ul2sh\_ddr\_arvalid | 1 | O | Indicates that dynamic logic's access to the read address channel of static logic DDRC is valid. |
| sh2ul\_ddr\_arready | 1 | I | Indicates that static logic is ready to access the read address channel of dynamic logic DDRC. |
| sh2ul\_ddr\_rid | 4 | I | Indicates that dynamic logic accesses the ID of the read data channel of static logic DDRC. |
| sh2ul\_ddr\_rdata | 512 | I | Indicates that dynamic logic accesses the address of the read data channel of static logic DDRC. |
| sh2ul\_ddr\_rresp | 2 | I | Indicates that dynamic logic accesses the valid cycles of the read data channel of static logic DDRC. |
| sh2ul\_ddr\_rlast | 1 | I | Indicates that dynamic logic accesses the valid bytes of each cycle of the read data channel of static logic DDRC. |
| sh2ul\_ddr\_rvalid | 1 | I | Indicates that dynamic logic's access to the read data channel of static logic DDRC is valid. |
| ul2sh\_ddr\_rready | 1 | O | Indicates that static logic is ready to access the read data channel of dynamic logic DDRC. |
| AXI-Lite interface of BAR channel | | | |
| sh2bar1\_awvalid | 1 | I | Corresponds to the BAR2 channel of VF and indicates the write valid signal of the BAR address channel. |
| sh2bar1\_awaddr | 32 | I | Corresponds to the BAR2 channel of VF and indicates the write valid signal of the BAR address channel. |
| bar12sh\_awready | 1 | O | Corresponds to the BAR2 channel of VF and indicates the write ready signal of the BAR address channel. |
| sh2bar1\_wvalid | 1 | I | Corresponds to the BAR2 channel of VF and indicates the write valid signal of the BAR data channel. |
| sh2bar1\_wdata | 32 | I | Corresponds to the BAR2 channel of VF and indicates the write data signal of the BAR data channel. |
| sh2bar1\_wstrb | 4 | I | Corresponds to the BAR2 channel of VF and indicates the write strb signal of the BAR data channel. |
| bar12sh\_wready | 1 | O | Corresponds to the BAR2 channel of VF and indicates the write ready signal of the BAR data channel. |
| bar12sh\_bvalid | 1 | O | Corresponds to the BAR2 channel of VF and indicates the valid signal of the BAR response channel. |
| bar12sh\_bresp | 2 | O | Corresponds to the BAR2 channel of VF and indicates the valid signal of the BAR response channel. |
| sh2bar1\_bready | 1 | I | Corresponds to the BAR2 channel of VF and indicates the ready signal of the BAR response channel. |
| sh2bar1\_arvalid | 1 | I | Corresponds to the BAR2 channel of VF and indicates the read valid signal of the BAR address channel. |
| sh2bar1\_araddr | 32 | I | Corresponds to the BAR2 channel of VF and indicates the read valid signal of the BAR address channel. |
| bar12sh\_arready | 1 | O | Corresponds to the BAR2 channel of VF and indicates the read ready signal of the BAR address channel. |
| bar12sh\_rvalid | 1 | O | Corresponds to the BAR2 channel of VF and indicates the read valid signal of the BAR data channel. |
| bar12sh\_rdata | 32 | O | Corresponds to the BAR2 channel of VF and indicates the read data signal of the BAR data channel. |
| bar12sh\_rresp | 1 | O | Corresponds to the BAR2 channel of VF and indicates the read strb signal of the BAR data channel. |
| sh2bar1\_rready | 1 | I | Corresponds to the BAR2 channel of VF and indicates the read ready signal of the BAR data channel. |
| AXI-Lite interface of BAR5 channel | | | |
| sh2bar5\_awvalid | 1 | I | Corresponds to the BAR4 channel of VF and indicates the write valid signal of the BAR address channel. |
| sh2bar5\_awaddr | 32 | I | Corresponds to the BAR4 channel of VF and indicates the write valid signal of the BAR address channel. |
| bar52sh\_awready | 1 | O | Corresponds to the BAR4 channel of VF and indicates the write ready signal of the BAR address channel. |
| sh2bar5\_wvalid | 1 | I | Corresponds to the BAR4 channel of VF and indicates the write valid signal of the BAR data channel. |
| sh2bar5\_wdata | 32 | I | Corresponds to the BAR4 channel of VF and indicates the write data signal of the BAR data channel. |
| sh2bar5\_wstrb | 4 | I | Corresponds to the BAR4 channel of VF and indicates the write strb signal of the BAR data channel. |
| bar52sh\_wready | 1 | O | Corresponds to the BAR4 channel of VF and indicates the write ready signal of the BAR data channel. |
| bar52sh\_bvalid | 1 | O | Corresponds to the BAR4 channel of VF and indicates the valid signal of the BAR response channel. |
| bar52sh\_bresp | 2 | O | Corresponds to the BAR4 channel of VF and indicates the valid signal of the BAR response channel. |
| sh2bar5\_bready | 1 | I | Corresponds to the BAR4 channel of VF and indicates the ready signal of the BAR response channel. |
| sh2bar5\_arvalid | 1 | I | Corresponds to the BAR4 channel of VF and indicates the read valid signal of the BAR address channel. |
| sh2bar5\_araddr | 32 | I | Corresponds to the BAR4 channel of VF and indicates the read valid signal of the BAR address channel. |
| bar52sh\_arready | 1 | O | Corresponds to the BAR4 channel of VF and indicates the read ready signal of the BAR address channel. |
| bar52sh\_rvalid | 1 | O | Corresponds to the BAR4 channel of VF and indicates the read valid signal of the BAR data channel. |
| bar52sh\_rdata | 32 | O | Corresponds to the BAR4 channel of VF and indicates the read data signal of the BAR data channel. |
| bar52sh\_rresp | 1 | O | Corresponds to the BAR4 channel of VF and indicates the read strb signal of the BAR data channel. |
| sh2bar5\_rready | 1 | I | Corresponds to the BAR4 channel of VF and indicates the read ready signal of the BAR data channel. |
| Interface signals of DDRA, DDRB, and DDRD | | | |
| ddra\_100m\_ref\_clk\_p | 1 | I | Indicates the p end of the DDRA 100 MHz differential reference clock. |
| ddra\_100m\_ref\_clk\_n | 1 | I | Indicates the n end of the DDRA 100 MHz differential reference clock. |
| ddrb\_100m\_ref\_clk\_p | 1 | I | Indicates the p end of the DDRB 100 MHz differential reference clock. |
| ddrb\_100m\_ref\_clk\_n | 1 | I | Indicates the n end of the DDRB 100 MHz differential reference clock. |
| ddrd\_100m\_ref\_clk\_p | 1 | I | Indicates the p end of the DDRD 100 MHz differential reference clock. |
| ddrd\_100m\_ref\_clk\_n | 1 | I | Indicates the n end of the DDRD 100 MHz differential reference clock. |
| ddra\_72b\_act\_n | 1 | O | Indicates the act\_n signal of DDRA. |
| ddra\_72b\_addr | 17 | O | Indicates the addr signal of DDRA. |
| ddra\_72b\_ba | 2 | O | Indicates the ba signal of DDRA. |
| ddra\_72b\_bg | 2 | O | Indicates the bd signal of DDRA. |
| ddra\_72b\_cke | 2 | O | Indicates the cke signal of DDRA. |
| ddra\_72b\_odt | 2 | O | Indicates the odt signal of DDRA. |
| ddra\_72b\_cs\_n | 2 | O | Indicates the cs\_n signal of DDRA. |
| ddra\_72b\_ck\_t | 1 | O | Indicates the ck\_t signal of DDRA. |
| ddra\_72b\_ ck\_c | 1 | O | Indicates the ck\_c signal of DDRA. |
| ddra\_72b\_rst\_n | 1 | O | Indicates the rst\_n signal of DDRA. |
| ddra\_72b\_par | 1 | O | Indicates the par signal of DDRA. |
| ddra\_72b\_dq | 72 | I/O | Indicates the dq signal of DDRA. |
| ddra\_72b\_dqs\_t | 9 | I/O | Indicates the dqs\_t signal of DDRA. |
| ddra\_72b\_dqs\_c | 9 | I/O | Indicates the dqs\_c signal of DDRA. |
| ddra\_72b\_dm\_dbi\_n | 9 | I/O | Indicates the dm\_dbi\_n signal of DDRA. |
| ddrb\_72b\_act\_n | 1 | O | Indicates the act\_n signal of DDRB. |
| ddrb\_72b\_addr | 17 | O | Indicates the addr signal of DDRB. |
| ddrb\_72b\_ba | 2 | O | Indicates the ba signal of DDRB. |
| ddrb\_72b\_bg | 2 | O | Indicates the bd signal of DDRB. |
| ddrb\_72b\_cke | 2 | O | Indicates the cke signal of DDRB. |
| ddrb\_72b\_odt | 2 | O | Indicates the odt signal of DDRB. |
| ddrb\_72b\_cs\_n | 2 | O | Indicates the cs\_n signal of DDRB. |
| ddrb\_72b\_ck\_t | 1 | O | Indicates the ck\_t signal of DDRB. |
| ddrb\_72b\_ ck\_c | 1 | O | Indicates the ck\_c signal of DDRB. |
| ddrb\_72b\_rst\_n | 1 | O | Indicates the rst\_n signal of DDRB. |
| ddrb\_72b\_par | 1 | O | Indicates the par signal of DDRB. |
| ddrb\_72b\_dq | 72 | I/O | Indicates the dq signal of DDRB. |
| ddrb\_72b\_dqs\_t | 9 | I/O | Indicates the dqs\_t signal of DDRB. |
| ddrb\_72b\_dqs\_c | 9 | I/O | Indicates the dqs\_c signal of DDRB. |
| ddrb\_72b\_dm\_dbi\_n | 9 | I/O | Indicates the dm\_dbi\_n signal of DDRB. |
| ddrd\_72b\_act\_n | 1 | O | Indicates the act\_n signal of DDRD. |
| ddrd\_72b\_addr | 17 | O | Indicates the addr signal of DDRD. |
| ddrd\_72b\_ba | 2 | O | Indicates the ba signal of DDRD. |
| ddrd\_72b\_bg | 2 | O | Indicates the bd signal of DDRD. |
| ddrd\_72b\_cke | 2 | O | Indicates the cke signal of DDRD. |
| ddrd\_72b\_odt | 2 | O | Indicates the odt signal of DDRD. |
| ddrd\_72b\_cs\_n | 2 | O | Indicates the cs\_n signal of DDRD. |
| ddrd\_72b\_ck\_t | 1 | O | Indicates the ck\_t signal of DDRD. |
| ddrd\_72b\_ ck\_c | 1 | O | Indicates the ck\_c signal of DDRD. |
| ddrd\_72b\_rst\_n | 1 | O | Indicates the rst\_n signal of DDRD. |
| ddrd\_72b\_par | 1 | O | Indicates the par signal of DDRD. |
| ddrd\_72b\_dq | 72 | I/O | Indicates the dq signal of DDRD. |
| ddrd\_72b\_dqs\_t | 9 | I/O | Indicates the dqs\_t signal of DDRD. |
| ddrd\_72b\_dqs\_c | 9 | I/O | Indicates the dqs\_c signal of DDRD. |
| ddrd\_72b\_dm\_dbi\_n | 9 | I/O | Indicates the dm\_dbi\_n signal of DDRD. |
| BSCAN interface | | | |
| S\_BSCAN\_drck | 1 | I | Indicates the drck signal of the BSCAN interface. |
| S\_BSCAN\_shift | 1 | I | Indicates the shift signal of the BSCAN interface. |
| S\_BSCAN\_tdi | 1 | I | Indicates the tdi signal of the BSCAN interface. |
| S\_BSCAN\_update | 1 | I | Indicates the update signal of the BSCAN interface. |
| S\_BSCAN\_sel | 1 | I | Indicates the sel signal of the BSCAN interface. |
| S\_BSCAN\_tdo | 1 | O | Indicates the tdo signal of the BSCAN interface. |
| S\_BSCAN\_tms | 1 | I | Indicates the tms signal of the BSCAN interface. |
| S\_BSCAN\_tck | 1 | I | Indicates the tck signal of the BSCAN interface. |
| S\_BSCAN\_runtest | 1 | I | Indicates the runtest signal of the BSCAN interface. |
| S\_BSCAN\_reset | 1 | I | Indicates the reset signal of the BSCAN interface. |
| S\_BSCAN\_capture | 1 | I | Indicates the capture signal of the BSCAN interface. |
| S\_BSCAN\_bscanid\_en | 1 | I | Indicates the bscanid signal of the BSCAN interface. |

### Interface Timing

For details about the AXI interface timing, visit <http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.set.amba/index.html>.

## Interaction Mode Descriptions

Huawei FPGA Accelerated Cloud Server (FACS) supports a shell framework driven by DPDK PMD, providing high-performance, low-latency, and high-reliability PCIe DMA channels.

Definition:

TX is the data transmission direction from the CPU memory to UL, while RX from UL to the CPU memory.

### TX Interaction



TX:

The application obtains the mbufs through DPDK interfaces and stores the data to be processed in the mbufs. After the mbufs are obtained, the data address information is assigned, which will be used subsequently.

The application sends the information returned by the mbufs and the information of data length to the DPDK PMD, which will build BDs based on the information.

The DPDK PMD sends the BD address to shell logic through BAR channels.

Shell logic obtains the BDs based on the address and length, and then sends the BDs to UL by the DMA read operation.

Shell logic sends the BD information to UL through AXI interfaces.

UL generates a DMA read operation command and sends it to shell logic.

Shell logic obtains the data to be processed from the CPU memory by the DMA read operation.

Shell logic sends the data to be processed to UL logic.

### RX Interaction



RX:

The application uses the DPDK PMD to enable the RX direction.

The DPDK PMD allocates idle BDs to shell logic through BAR channels.

Shell logic obtains and caches idle BDs.

UL sends the processed packets to shell logic.

Shell logic allocates idle BD information based on queues.

Shell logic writes back the processed data to the CPU memory through the DMA write operation.

Shell logic writes back the processed response result by the DMA write operation, and the information such as the address and length of the processed data is stored in the response result.

The application obtains the processed packets and response result through the DPDK PMD.

# FACS High Performance Interface (DPDK) Description

## Shell Interface Description

### Overview

This section describes the interfaces between shell logic and UL.

****

### Shell-to-UL BD Interface

* **Function**

This interface transmits BDs sent from static shell logic to UL.

note

After the data to be processed is prepared at the application layer, the driver will create BDs and send them to static shell logic. After the BDs are received, static shell logic forwards the BDs to UL through the AXI-S interface.

The following table describes the BD data format.

* **Shell-to-UL BD data format**

| Name | Bit Position | Description |
| --- | --- | --- |
| odd\_even | [255:248] | [255]:[247:224] Even parity check value  [254]:[223:192] Even parity check value  [253]:[191:160] Even parity check value  [252]:[159:128] Even parity check value  [251]:[127:96] Even parity check value  [250]:[95:64] Even parity check value  [249]:[63:32] Even parity check value  [248]:[31:0] Even parity check value |
| bd\_code | [247:240] | BD verification code. The default value is **8'h5a**. |
| rsv | [239:208] | Reserved. The default value is **0**. |
| sh\_info | [207:160] | [207:192] VM\_id  [191:176] PF/VF id  [175:160] Queue ID of a single VF/PF |
| length | [159:128] | The total length of BD data that needs to be processed by UL. |
| des\_addr | [127:64] | The destination physical address of processed information. |
| src\_addr | [63:0] | The source address of the information to be processed. |

Shell information (**sh\_info**): **VM\_id** indicates the VM to which the BD belongs. **PF/VF id** indicates the PCIe PF/VF from which the BD comes. **Queue ID** indicates the queue to which the BD belongs.

Source address (**src\_addr**): indicates the physical address for storing to-be-processed data of the BD.

Length (**length**): indicates the length of to-be-processed data of the BD.

* **Interface signals**

| Name | Bit Width | I/O | Description |
| --- | --- | --- | --- |
| sh2ul\_dmam0\_tvalid | 1 | I | Indicates the axi-s master valid signal. |
| ul2sh\_dmam0\_tready | 1 | O | Indicates the axi-s slave ready signal. |
| sh2ul\_dmam0\_tdata | 256 | I | Indicates the axi-s master data. |
| sh2ul\_dmam0\_tlast | 1 | I | Indicates the last cycle of **sh2ul\_dmam0\_tdata**. |
| sh2ul\_dmam0\_tkeep | 32 | I | Indicates the axi-s keep signal. |

### UL-to-Shell BD Interface

* **Function**

This interface transmits read data commands sent from UL to static shell logic.

note

UL needs to build a read data command according to the information of the received shell BDs and send it to shell logic through the AXI-S interface. Shell logic then obtains required data from the memory according to the command.

The following table describes the BD data format.

* **UL-to-Shell BD data format**

| Name | Bit Position | Description |
| --- | --- | --- |
| ul\_info | [255:224] | UL private information, which needs to be returned in shell read packets. |
| rsv | [223:208] | Reserved. The default value is **0**. |
| sh\_info | [207:160] | [207:192] VM\_id  [191:176] PF/VF id  [175:160] Queue ID of a single VF/PF |
| length | [159:128] | The total length of the BD packet that needs to be processed by UL. |
| rsv | [127:64] | Reserved. |
| src\_addr | [63:0] | The source physical address of the information to be processed. |

UL information (**ul\_info**): indicates private user information. Data is returned to UL by shell from the original path.

Shell information (**sh\_info**): obtained from shell BDs.

Source address (**src\_addr**): obtained from shell BDs.

Length (**length**): obtained from shell BDs. If the length is greater than 4 Kbytes, the user needs to cut it to less than or equal to 4 Kbytes.

* **Interface signals**

| Name | Bit Width | I/O | Description |
| --- | --- | --- | --- |
| **ul2sh\_dmas2\_tvalid** | 1 | O | Indicates the axi-s master valid signal. |
| **sh2ul\_dmas2\_tready** | 1 | I | Indicates the axi-s slave ready signal. |
| **ul2sh\_dmas2\_tdata** | 256 | O | Indicates the axi-s master data. |
| **ul2sh\_dmas2\_tlast** | 1 | O | Indicates the last cycle of **ul2sh\_dmas2\_tdata**. |
| **sh2ul\_dmas2\_tkeep** | 32 | O | Indicates the axi-s keep signal. |

### Shell-to-UL Data Interface

* **Function**

This interface transmits the to-be-processed original packet information sent from static shell logic to UL.

note

After UL sends a read packet request, static shell logic returns the to-be-processed data packets through the AXI-S interface. UL executes processing according to the returned packets.

The to-be-processed packets returned from shell logic to UL are multi-cycle data. The following table describes the data format.

* **Shell-to-UL packet data format**

| Name | Cycle Position | Bit Width | Description |
| --- | --- | --- | --- |
| pkt header | cycle 1 | 512 | Packet header. |
| pkt payload | cycle 2~cycle n | 512 | The packets to be accelerated. |

The following table describes the data format of the packet header.

* **Shell-to-UL packet header data format**

| Name | Bit position | Description |
| --- | --- | --- |
| rsv | [511:256] | Reserved. The value is **0**. |
| ul\_info | [255:224] | UL private information, which needs to be returned in shell read packets. |
| rsv | [223:216] | Reserved. |
| acc\_type | [215:208] | Reserved. |
| sh\_info | [207:160] | [207:192] VM\_id  [191:176] PF/VF id  [175:160] Queue ID of a single VF/PF |
| rsv | [159:128] | Reserved. |
| rsv | [127:64] | Reserved. |
| rsv | [63:0] | Reserved. |

1. UL information (**ul\_info**): indicates private user information. Data is returned to UL by shell from the original path.
2. Shell information (**sh\_info**): obtained from shell BDs.

* **Interface signals**

| Name | Bit Width | I/O | Description |
| --- | --- | --- | --- |
| **sh2ul\_dmam1\_tvalid** | 1 | I | Indicates the axi-s master valid signal. |
| **ul2sh\_dmam1\_tready** | 1 | O | Indicates the axi-s slave ready signal. |
| **sh2ul\_dmam1\_tdata** | 512 | I | Indicates the axi-s master data. |
| **sh2ul\_dmam1\_tlast** | 1 | I | Indicates the last cycle of **sh2ul\_dmam1\_tdata**. |
| **sh2ul\_dmam1\_tkeep** | 64 | I | Indicates the axi-s keep signal. |

### UL-to-Shell Data Interface

* **Function**

This interface transmits the processed packet information returned from UL to shell logic.

note

After processing the data, UL sends the data to shell logic through the AXI-M interface, and maintains and calculates the packet returning destination address **Des\_addr**. After all packets are transmitted, UL sends the interaction complete information, including **done\_flg**, **src\_addr**, and **des\_addr**.

Note: A maximum of 4 Kbyte interaction is supported.

The packets returned from UL to shell logic are multi-cycle data. The following table describes the data format.

* **UL-to-Shell packet data format**

| Name | Cycle Position | Bit Width | Description |
| --- | --- | --- | --- |
| pkt header | cycle 1 | 512 | Packet header. |
| pkt payload | cycle 2~cycle n | 512 | The packets to be accelerated. |

The following table describes the data format of the packet header.

* **UL-to-Shell packet header data format**

| Name | Position | Description |
| --- | --- | --- |
| Rsv | [255:217] | Reserved. |
| done\_flg | [216] | **done\_flg** flag. |
| Acc\_type | [215:208] | Reserved. |
| sh\_info | [207:160] | [207:192] VM\_id  [191:176] PF/VF id  [175:160] Queue ID of a single VF/PF |
| Rsv | [159:128] | Reserved. |
| Des\_addr | [127:64] | The destination physical address of processed information. |
| Src\_addr | [63:0] | The source physical address of processed information. |

Acceleration complete flag (**done\_flag**): indicates that the packet acceleration is complete. When the position of the flag is **1**, the packet processing is complete, and the CPU can capture the packets.

Shell information (**sh\_info**): obtained from shell BDs.

Destination address (**Des\_addr**): indicates the physical address in the CPU memory to which the accelerated packets are returned.

Source address (**src\_addr**): obtained from shell BDs.

* **Interface signals**

| Name | Bit Width | I/O | Description |
| --- | --- | --- | --- |
| **ul2sh\_dmas3\_tvalid** | 1 | O | Indicates the axi-s master valid signal. |
| **sh2ul\_dmas3\_tready** | 1 | I | Indicates the axi-s slave ready signal. |
| **ul2sh\_dmas3\_tdata** | 512 | O | Indicates the axi-s master data. |
| **ul2sh\_dmas3\_tlast** | 1 | O | Indicates the last cycle of **ul2sh\_dmas3\_tdata**. |
| **ul2sh\_dmas3\_tkeep** | 64 | O | Indicates the axi-s keep signal. |

* **Example**

For example, shell logic sends to UL the BDs whose **Src\_addr** is **0x10000000**, **Len** is **11520**, and **Des\_addr** is **0x20000000**. UL divides the packet BDs into three parts, and the length of the processed packet is the same as that of the source packet. The corresponding BD information (the structure of **src\_addr len**) is as follows:

BD0: SRC\_addr = 0x10000000 LEN=0x1000;

BD1: SRC\_addr = 0x10001000 LEN=0x1000;

BD2: SRC\_addr = 0x10002000 LEN=0xD00;

The structure of the packet header is as follows:

pkt header 0: done\_flag=0 Des\_addr=0x20000000 SRC\_addr = 0x10000000;

pkt header 1: done\_flag=0 Des\_addr=0x20001000 SRC\_addr = 0x10001000;

pkt header2: done\_flag=0 Des\_addr=0x20002000 SRC\_addr = 0x10002000;

pkt header3: done\_flag=1 Des\_addr=0x20000000 SRC\_addr = 0x10000000;

### AXI-Lite Interface

* **Function**

This interface allows applications to access internal registers of dynamic logic. Dynamic logic is connected to two groups of standard AXI-Lite interface signals to access different BARs (BAR2 and BAR4).

* **AXI-Lite interface signals**

| Signal | Bit Width | I/O | Description |
| --- | --- | --- | --- |
| clk\_200m | 1 | I | Indicates the working clock (200 MHz). |
| rst\_200m | 1 | I | Indicates the high-level reset signal. |
| sh2bar1\_awvalid | 1 | I | Corresponds to the BAR2 channel and indicates that the write operation address is valid. The master informs the slave that the write address is valid. |
| sh2bar1\_awaddr | 32 | I | Corresponds to the BAR2 channel and indicates the write operation address. |
| bar12sh\_awready | 1 | O | Corresponds to the BAR2 channel and indicates that the write operation address channel is ready. The slave informs the master that the address transmission is ready. |
| sh2bar1\_wvalid | 1 | I | Corresponds to the BAR2 channel and indicates that the write operation data is valid. The master informs the slave that the write data is valid. |
| sh2bar1\_wdata | 32 | I | Corresponds to the BAR2 channel and indicates the write operation data. |
| sh2bar1\_wstrb | 4 | I | Corresponds to the BAR2 channel and indicates the write operation data gating. 1 bit selects 1 byte data. The default input value is **4'hF**, and all the 32-bit write data is valid. |
| bar12sh\_wready | 1 | O | Corresponds to the BAR2 channel and indicates that the write operation data channel is ready. The slave informs the master that the data transmission is ready. |
| bar12sh\_bvalid | 1 | O | Corresponds to the BAR2 channel and indicates that the response status of the write operation is valid. |
| bar12sh\_bresp | 2 | O | Corresponds to the BAR2 channel and indicates the response status of the write operation. **2'h0** indicates success, while other values indicate errors. |
| sh2bar1\_bready | 1 | I | Corresponds to the BAR2 channel and indicates the write operation response. The master informs the slave that the response has been received. |
| sh2bar1\_arvalid | 1 | I | Corresponds to the BAR2 channel and indicates that the read operation address is valid. The master informs the slave that the read operation address is valid. |
| sh2bar1\_araddr | 32 | I | Corresponds to the BAR2 channel and indicates the read operation address. |
| bar12sh\_arreay | 1 | O | Corresponds to the BAR2 channel and indicates that the read operation address channel is ready. The slave informs the master that the address transmission is ready. |
| bar12sh\_rvalid | 1 | O | Corresponds to the BAR2 channel and indicates that the read operation data is valid. The slave informs the master that the read data is valid. |
| bar12sh\_rdata | 32 | O | Corresponds to the BAR2 channel and indicates the data returned by the read operation. |
| sh2bar1\_rready | 1 | I | Corresponds to the BAR2 channel and indicates that the read operation data channel is ready. The master informs the slave that the data transmission is ready. |
| bar12sh\_rresp | 2 | O | Corresponds to the BAR2 channel and indicates the response status of the read operation. **2'h0** indicates success, while other values indicate errors. |
| sh2bar5\_awvalid | 1 | I | Corresponds to the BAR4 channel and indicates that the write operation address is valid. The master informs the slave that the write address is valid. |
| sh2bar5\_awaddr | 32 | I | Corresponds to the BAR4 channel and indicates the write operation address. |
| bar52sh\_awready | 1 | O | Corresponds to the BAR4 channel and indicates that the write operation address channel is ready. The slave informs the master that the address transmission is ready. |
| sh2bar5\_wvalid | 1 | I | Corresponds to the BAR4 channel and indicates that the write operation data is valid. The master informs the slave that the write data is valid. |
| sh2bar5\_wdata | 32 | I | Corresponds to the BAR4 channel and indicates the write operation data. |
| sh2bar5\_wstrb | 4 | I | Corresponds to the BAR4 channel and indicates the write operation data gating. 1 bit selects 1 byte data. The default input value is **4'hF**, and all the 32-bit write data is valid. |
| bar52sh\_wready | 1 | O | Corresponds to the BAR4 channel and indicates that the write operation data channel is ready. The slave informs the master that the data transmission is ready. |
| bar52sh\_bvalid | 1 | O | Corresponds to the BAR4 channel and indicates that the response status of the write operation is valid. |
| bar52sh\_bresp | 2 | O | Corresponds to the BAR4 channel and indicates the response status of the write operation. **2'h0** indicates success, while other values indicate errors. |
| sh2bar5\_bready | 1 | I | Corresponds to the BAR4 channel and indicates the write operation response. The master informs the slave that the response has been received. |
| sh2bar5\_arvalid | 1 | I | Corresponds to the BAR4 channel and indicates that the read operation address is valid. The master informs the slave that the read address is valid. |
| sh2bar5\_araddr | 32 | I | Corresponds to the BAR4 channel and indicates the read operation address. |
| bar52sh\_arreay | 1 | O | Corresponds to the BAR4 channel and indicates that the read operation address channel is ready. The slave informs the master that the address transmission is ready. |
| bar52sh\_rvalid | 1 | O | Corresponds to the BAR4 channel and indicates that the read operation data is valid. The slave informs the master that the read data is valid. |
| bar52sh\_rdata | 32 | O | Corresponds to the BAR4 channel and indicates the data returned by the read operation. |
| sh2bar5\_rready | 1 | I | Corresponds to the BAR4 channel and indicates that the read operation data channel is ready. The master informs the slave that the data transmission is ready. |
| bar52sh\_rresp | 2 | O | Corresponds to the BAR4 channel and indicates the response status of the read operation. **2'h0** indicates success, while other values indicate errors. |

### HardAcc Description

**Function**

HardAcc (length: 32 bytes) is used to cache physical addresses of the hardware logic DDRs. The physical addresses are written to the memory starting from **src\_addr** by the DPDK driver.

The following table describes the data format of HardAcc.

|  |  |  |
| --- | --- | --- |
| Name | Bit Position | Description |
| rsv | [255:72] | Reserved |
| ddr\_daddr | [71:36] | The parsing result after acceleration, which is written to the DDR start address |
| ddr\_saddr | [35:0] | Data to be accelerated, which is written to the DDR start address |

## SDK Interface Description

The SDK uses the Intel open-source DPDK framework as the basis for development and deployment. As a collection of function libraries and drivers used for fast data packet forwarding and processing, the DPDK:

Greatly improves data processing performance, throughput, and work efficiency of data plane applications.

Standardizes the user interface and unifies the development process.

Develops PCIe device services at the user layer to simplify user development.

Provides a simple memory control solution that directly controls physical addresses at the application layer.

This section describes how to use DPDK interfaces in service-oriented FPGA projects.

### Common DPDK Interfaces for VF Devices

Based on DPDK-16.04, the SDK integrates the high-performance PMD module developed by Huawei for the FPGA VF, allowing users to directly use standard DPDK interfaces to encapsulate and process data packets.

#### Integration of DPDK Interfaces and PMD



As the layer between the upper service program layer and the underlying logic, the SDK provides a user interface upwards and a logic driver downwards.

The SDK is designed based on the DPDK framework, including a user control interface, a user memory management interface, and PMD. The SDK reserves the original memory management module and user interface function of the DPDK. The control information interaction of the underlying devices is implemented in the PMD module.

The DPDK user control interface provides device configuration and data interaction services, including DPDK environment initialization, FPGA queue configuration, and service RX/TX functions.

The DPDK user memory management interface manages the memory of the DPDK environment. The data of each packet is stored in the memory pool in the rte\_mbuf structure.

Corresponding to specific devices and is mounted to DPDK interfaces as a callback function, the PMD module is a software entity for control information interaction between the CPU and the underlying logic.

#### Data Flow Diagram



The application program builds control message BDs through the DPDK mbuf interface and copies the BDs to the BD queue through the **rte\_eth\_tx\_burst()** interface.

Copy the new BDs from the queue to the FPGA.

Send packets to the FPGA.

Output the processing result to the packet space.

Write the response BDs to the RX queue.

Return the result queue of the response BDs to the application.

#### Basic Invocation Process

The standard DPDK interfaces allow users to build an application process easily. The following flowchart describes the TX/RX operations using the DPDK interfaces.



The preceding flowchart can be tailored by users as required. For example, multiple processes may be used for initialization and services. Multiple threads may be used for the TX/RX. For the TX input space and RX output space of a dynamic IP, memzones instead of rte\_mbufs may be used.

You can obtain a complete service case by referring to the program case code provided by the SDK.

#### Common DPDK Interfaces



#### Description of Common DPDK Interfaces

rte\_eal\_init

|  |  |
| --- | --- |
| Function prototype | int rte\_eal\_init(int argc, char \*\*argv) |
| Function | Initialize the environment. |
| Input description | **argc**: indicates parameter number.  **argv**: indicates an input parameter. |
| Output description | N/A |
| Returned value description | ≥ **0**: indicates success.  **-1**: indicates failure. A value will be assigned to **rte\_errno**, which indicates the cause of error. |
| Operation instructions | This function is executed in MASTER lcore and is better to be executed in the **main()** function of the application. |
| Constraints | Only one function can be invoked in one process. |

rte\_eth\_dev\_count

|  |  |
| --- | --- |
| Function prototype | uint8\_t rte\_eth\_dev\_count(void) |
| Function | Query the number of devices managed by the DPDK. |
| Input description | N/A |
| Output description | N/A |
| Returned value description | The number of devices managed by the DPDK. |
| Operation instructions | The number of devices managed by the DPDK is determined by hardware resources of the system and the loaded PMD. |
| Constraints | This function is invoked after the **rte\_eal\_init** function is used. |

rte\_eth\_dev\_configure

|  |  |
| --- | --- |
| Function prototype | int rte\_eth\_dev\_configure(uint8\_t port\_id, uint16\_t nb\_rx\_queue, uint16\_t nb\_tx\_queue, const struct rte\_eth\_conf \*eth\_conf) |
| Function | Configure a device. |
| Input description | **port\_id**: indicates the ID of the device to be configured.  **nb\_rx\_queue**: indicates the maximum number of RX queues of the device. In this project, the value is **8**.  **nb\_tx\_queue**: indicates the maximum number of TX queues of the device. In this project, the value is **8**.  **eth\_conf**: indicates device configuration parameters. |
| Output description | N/A |
| Returned value description | **0**: indicates success.  <**0**: indicates failure. |
| Operation instructions | This function must be invoked before any **port id** interfaces are used. For this project, each member variable of **eth\_conf** can be set to **0**, but **eth\_conf** cannot be set to **NULL**. |
| Constraints | N/A |

rte\_eth\_rx\_queue\_setup

|  |  |
| --- | --- |
| Function prototype | int rte\_eth\_rx\_queue\_setup(uint8\_t port\_id, uint16\_t rx\_queue\_id, uint16\_t nb\_rx\_desc, unsigned int socket\_id, const struct rte\_eth\_rxconf\* rx\_conf, struct rte\_mempool\* mb\_pool) |
| Function | Configure an RX queue for a device. |
| Input description | **port\_id**: indicates the ID of the device to be configured.  **rx\_queue\_id**: indicates the ID of the RX queue to be configured. The value range is [0,nb\_rx\_queue-1]. **nb\_rx\_queue** is the RX queue parameter of **rte\_eth\_config**.  **nb\_tx\_desc**: indicates the maximum receiving space of the RX queue. The value can be set to **1024**, **2048**, **4096**, or **8192** in this project.  **socket\_id**: is used for the NUMA system. **SOCKET\_ID\_ANY (-1)** is used for a non-NUMA system.  **rx\_conf**: is set to **NULL** in this project.  **mb\_pool**: indicates the memory pool used to allocate the maximum receiving space of the RX queue. |
| Output description | N/A |
| Returned value description | **0**: indicates success.  **-EINVAL**: indicates parameter error.  **-ENOMEM**: indicates that the maximum receiving space is insufficient in the memory pool. |
| Operation instructions | N/A |
| Constraints | N/A |

rte\_eth\_tx\_queue\_setup

|  |  |
| --- | --- |
| Function prototype | int rte\_eth\_tx\_queue\_setup(uint8\_t port\_id, uint16\_t tx\_queue\_id, uint16\_t nb\_tx\_desc, unsigned int socket\_id, const struct rte\_eth\_txconf\* tx\_conf) |
| Function | Configure a TX queue for a device. |
| Input description | **port\_id**: indicates the ID of the device to be configured.  **tx\_queue\_id**: indicates the ID of the TX queue to be configured. The value range is [0, ,nb\_tx\_queue-1]. **nb\_tx\_queue** is the TX queue parameter of **rte\_eth\_config**.  **nb\_tx\_desc**: indicates the maximum receiving space of the TX queue. The value can be set to **1024**, **2048**, **4096**, or **8192** in this project.  **socket\_id**: is used for the NUMA system. **SOCKET\_ID\_ANY (-1)** is used for a non-NUMA system. |
| Output description | N/A |
| Returned value description | **0**: indicates success.  **-ENOMEM**: indicates that TX queue space descriptors are insufficient. |
| Operation instructions | N/A |
| Constraints | N/A |

rte\_eth\_dev\_start

|  |  |
| --- | --- |
| Function prototype | int rte\_eth\_dev\_start(uint8\_t port\_id) |
| Function | Start a device. |
| Input description | **port\_id**: indicates the device ID. |
| Output description | N/A |
| Returned value description | **0**: indicates success.  <**0**: indicates failure. |
| Operation instructions | This function is the last interface for configuring a device, after which the TX/RX service interface can be invoked. This interface resets each queue to be used by the device. |
| Constraints | This function needs to be invoked after **tx/rx\_queue\_setup**. |

rte\_eth\_dev\_stop

|  |  |
| --- | --- |
| Function prototype | void rte\_eth\_dev\_stop(uint8\_t port\_id) |
| Function | In this project, when a device stops working, this interface disables each queue of the device. |
| Input description | **port\_id**: indicates the device ID. |
| Output description | N/A |
| Returned value description | N/A |
| Operation instructions | N/A |
| Constraints | N/A |

rte\_eth\_dev\_close

|  |  |
| --- | --- |
| Function prototype | void rte\_eth\_dev\_close(uint8\_t port\_id) |
| Function | Close a device. |
| Input description | **port\_id**: indicates the device ID. |
| Output description | N/A |
| Returned value description | N/A |
| Operation instructions | A device cannot be restarted after it is closed, and this interface releases most resources. To release all resources, use **rte\_eth\_dev\_detach()**. |
| Constraints | N/A |

rte\_eth\_dev\_detach

|  |  |
| --- | --- |
| Function prototype | int rte\_eth\_dev\_detach(uint8\_t port\_id, char \*devname) |
| Function | Delete a device from the DPDK. |
| Input description | **port\_id**: indicates the device ID. |
| Output description | **devname**: indicates the name of the deleted device. |
| Returned value description | 0 on success and devname is filled, negative on error |
| Operation instructions | This function needs to be invoked after it is closed. |
| Constraints | N/A |

rte\_eth\_tx\_burst

|  |  |
| --- | --- |
| Function prototype | static uint16\_t rte\_eth\_tx\_burst(uint8\_t port\_id, uint16\_t queue\_id, struct rte\_mbuf\*\* tx\_pkts, uint16\_t nb\_pkts) |
| Function | Send a burst of output packets on a transmit queue of an Ethernet device. |
| Input description | **port\_id**: indicates the device ID.  **queue\_id**: indicates the logic queue ID.  **tx\_pkts**: indicates the TX input array.  **nb\_pkts**: indicates the number of TX input elements. |
| Output description | N/A |
| Returned value description | The actual number of **tx\_pkts** stored in the internal TX queue descriptors. |
| Operation instructions | The **tx\_pkts** array is a pointer to the rte\_mbuf structure. Each pointer is applied from the memory pool allocated by the **rte\_pktmbuf\_pool\_create()**. |
| Constraints | N/A |

rte\_eth\_rx\_burst

|  |  |
| --- | --- |
| Function prototype | static uint16\_t rte\_eth\_rx\_burst(uint8\_t port\_id, uint16\_t queue\_id, struct rte\_mbuf\*\* rx\_pkts, uint16\_t nb\_pkts) |
| Function | Return the packets in the RX queue. |
| Input description | **port\_id**: indicates the device ID.  **queue\_id**: indicates the logic queue ID.  **nb\_pkts**: indicates the maximum receiving number. |
| Output description | **rx\_pkts**: indicates the received array. |
| Returned value description | The actual number of received packets. |
| Operation instructions | N/A |
| Constraints | N/A |

rte\_mempool\_create

|  |  |
| --- | --- |
| Function prototype | struct rte\_mempool\* rte\_mempool\_create(const char \*name, unsigned n, unsigned elt\_size, unsigned elt\_size, unsigned cache\_size, unsigned private\_data\_size, rte\_mempool\_ctor\_t\* mp\_init,void \*mp\_init\_arg, rte\_mempool\_obj\_cb\_t \*obj\_init, void \*obj\_init\_arg, int socket\_id, unsigned flags) |
| Function | Create a memory pool named **name** to allocate rte\_mbufs. |
| Input description | **name**: indicates the name of the memory pool.  **n**: indicates the number of rte\_mbufs in the memory pool. The optimal value is **n=(2^q-1)**.  **elt\_size**: indicates the size of each rte\_mbuf.  **cache\_size**: indicates the optimized option.  **private\_data\_size**: indicates the private data size of the memory pool. The value can be set to **sizeof(rte\_pktmbuf\_pool\_private)**.  **mp\_init**: can be set to **rte\_pktmbuf\_pool\_init**.  **mp\_init\_arg**: can be set to **NULL**.  **obj\_init**: can be set to **rte\_pktmbuf\_init**.  **obj\_init\_arg**: can be set to **NULL**.  **socket\_id**: indicates the NUMA ID. **SOCKET\_ID\_ANY** is used for a non-NUMA system.  **flags**: can be set to **MEMPOOL\_F\_SP\_PUT | MEMPOOL\_F\_SC\_GET**. |
| Output description | N/A |
| Returned value description | The address of the memory pool is returned when the function is successfully executed.  **NULL** is returned when an error occurs. |
| Operation instructions | N/A |
| Constraints | N/A |

rte\_pktmbuf\_alloc

|  |  |
| --- | --- |
| Function prototype | static struct rte\_mbuf\* rte\_pktmbuf\_alloc(struct rte\_mempool \*mp) |
| Function | Apply for an rte\_mbuf from the memory pool. |
| Input description | **mp**: indicates the memory pool address. |
| Output description | N/A |
| Returned value description | The address of the rte\_mbuf is returned when the function is executed successfully.  **NULL** is returned when error occurs. |
| Operation instructions | There is no content in the data area of the **rte\_mbuf**. The length of data area is set to **0**. |
| Constraints | N/A |

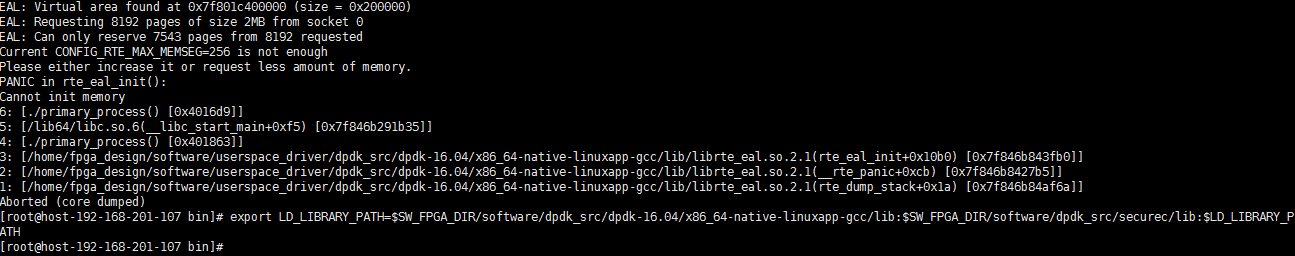
Note: When using **rte\_eal\_init** and **rte\_eal\_remote\_launch**, pay attention to the CPU cores of the execution environment. If the used parameters do not meet the requirements, program exceptions may occur.

For details about how to use DPDK interfaces, see the official documents at <http://www.dpdk.org/doc/archives>.

### rte\_eal\_init Failure

#### Huge Page Allocation Failure

**Symptom**



**Cause**

Excessive memory fragments result in excessive huge pages. As a result, the DPDK fails to initialize the huge pages.

**Solutions**

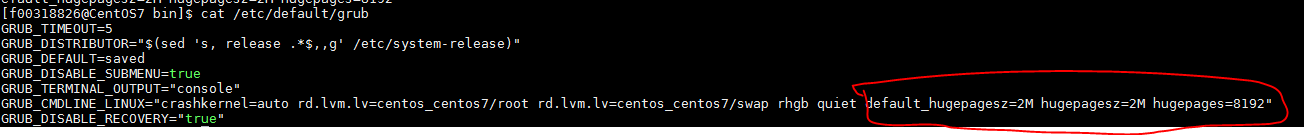
Run the **sysctl -w vm.nr\_hugepages=8192** command to configure huge pages when the system is started and few memory fragments exist, otherwise the DPDK fails to initialize huge pages due to excessive memory fragments.

Modify the startup parameters of the Linux system on a VM. Add the startup parameters **default\_hugepagesz=2M**, **hugepagesz=2M**, and **hugepages=8192** to the grub file. The procedure is as follows:

1. Edit the /**etc/default/grub** file.

Modify **GRUB\_CMDLINE\_LINUX="………."** into:

**GRUB\_CMDLINE\_LINUX ="…………default\_hugepagesz=2M hugepagesz=2M hugepages=8192"**. The following figure shows an example:



1. Run the following command on the VM to update the **grub.cfg** file:

grub2-mkconfig > /boot/grub2/grub.cfg

1. Restart the VM.

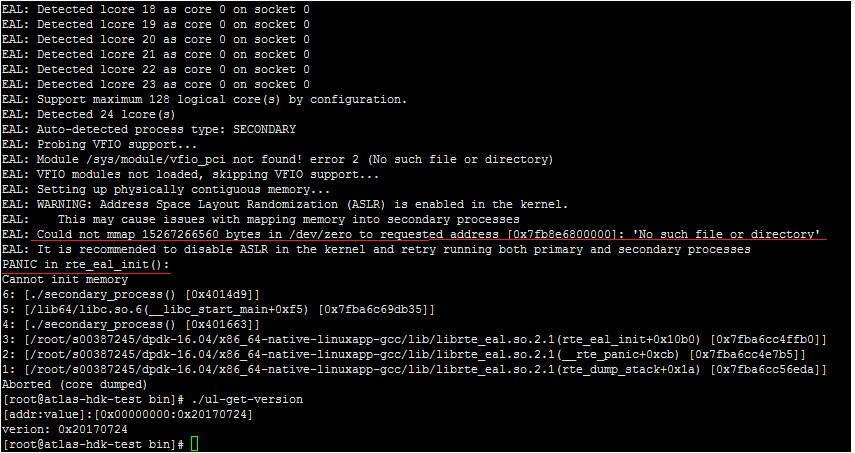
note

After method 2 is used, you do not need to run the **sysctl -w vm.nr\_hugepages=8192** command on the VM. Method 2 is essentially the same as method 1, reserving huge pages for the system when the system is started.

#### Huge Page Mapping Failure in the Secondary Process

**Symptom**

When the primary process starts and the secondary process invokes **rte\_eal\_init**, this problem may occur.



**Cause**

In the DPDK mechanism, the primary and secondary processes use the same virtual address space for huge page mapping. This problem occurs when the secondary process starts and occupies the virtual address space.

**Solutions**

Restart the secondary process. Currently, the problem does not occur for two consecutive times.

You are not advised to run multiple processes. This problem does not occur in single-process mode.

#### Enabling the Log Function

Create the **/var/log/fpga/** directory and the **dpdk.log** file.

Create a **dpdk.conf** file in **/etc/rsyslog.d**.

Write the following code to the **dpdk.conf** file:

if ($programname == 'packet\_process') then {

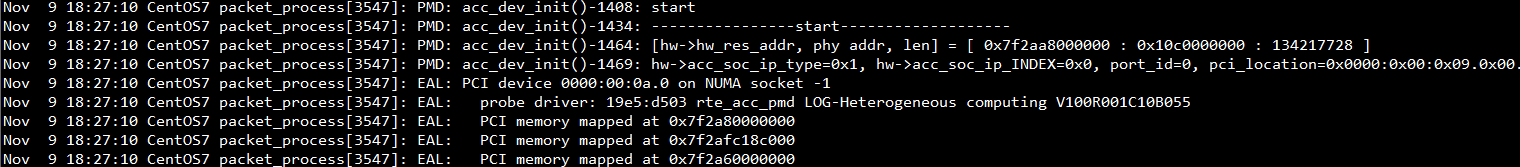
action(type="omfile" fileCreateMode="0600" file="/var/log/fpga/dpdk.log")

stop

}

Run the **service rsyslog restart** command to restart the rsyslog service.

Logs are automatically saved to the **/var/log/fpga/dpdk.log** file. Information similar to the following is recorded in the logs.

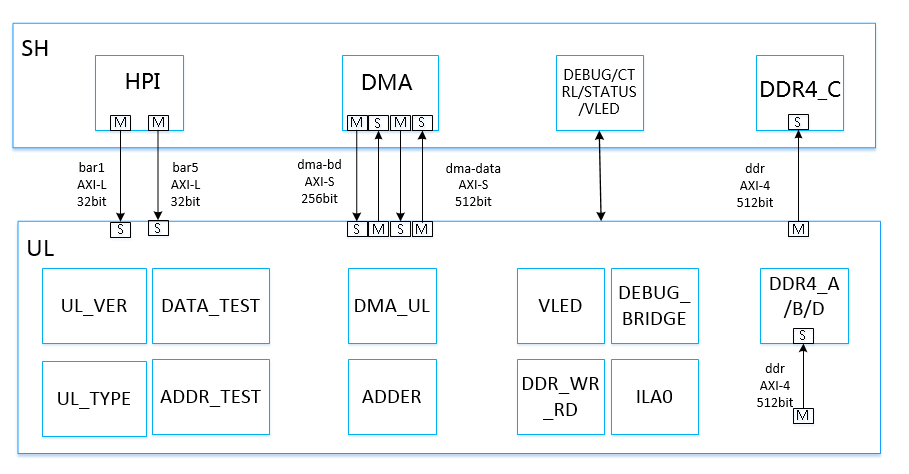


## Example 2 Detailed Design

### Overview

Example 2 implements DDR read/write access, DMA data loopback, and addition functions. You can quickly implement the acceleration design by referring to involved interfaces and interaction processes in this example.

### Features



**UL\_VER** instantiates CBB **ro\_reg\_inst**. The version number is the release time of example 1.

**UL\_TYPE** instantiates CBB **ro\_reg\_inst**. The type is **32'h00d10001**.

**DATA\_TEST** instantiates CBB **ts\_reg\_inst**, which implements the input data inversion function.

**ADDR\_TEST** instantiates CBB **ts\_addr\_reg\_inst**, which implements the inversion of the last operation address.

**DMA\_UL** instantiates one frame FIFO, which implements UL DMA data loopback by the host.

**DDR\_WR\_RD** implements read/write access to the data channels of four DDRs.

VLED is accessed under PF and static logic provides pins for dynamic logic. Users read and write the VLED and ensure that UL works properly. VLED instantiates a group of CBB **rw\_reg\_inst**, which links the output result to VLED.

The adder implements the addition of carried addends and augends.

**DEBUG\_BRIDGE** and **ILA0** are instantiated for debugging.

### HDK Building Description

After applying for an FPGA image, log in to the VM. The HDK is stored in the **/home/fpga\_design** directory by default. To build example 2, perform the following steps:

Configure the license file of EDA.

Open the **setup.cfg** file in **/home/fpga\_design/** and set **XILINX\_LIC\_SETUP**:

CN North:

XILINX\_LIC\_SETUP="2100@100.125.1.240:2100@100.125.1.245"

CN South:

XILINX\_LIC\_SETUP="2100@100.125.16.137:2100@100.125.16.138"

CN East:

XILINX\_LIC\_SETUP="2100@100.125.17.108:2100@100.125.17.109"

note

Only user **root** has the right to use the Xilinx license file provided by Huawei.

Configure the development environment.

Run **/home/fpga\_design/setup.sh** to configure the hardware development environment.

$ cd /home/fpga\_design

$ export HW\_FPGA\_DIR=$(pwd)

$ source $HW\_FPGA\_DIR/setup.sh

Go to the **example2** directory.

$ cd $HW\_FPGA\_DIR/hardware/vivado\_design/examples/example2

$ export UL2\_DIR=$(pwd)

Compile example 2.

$ cd $UL2\_DIR/prj

$ sh ./build.sh

Simulate example 2.

$ cd $UL2\_DIR/sim

$ make TC=sv\_demo\_001

### Terminology

The FPGA HDK supports Verilog and VHDL for FPGA development. The source files must be stored in the **src** folder in the **usr\_prj\_name** directory.

$ cd $HW\_FPGA\_DIR/hardware/vivado\_design/user/<usr\_prj\_name>/src

### Design Constraints

The FPGA HDK provides a general-purpose FPGA simulation platform based on the SystemVerilog-2012 syntax standard. The platform supports mainstream simulation tools, and the platform architecture supports decoupling of Testbench and Testcase to enable quick simulation platform building. Testbench is located in the **$HW\_FPGA\_DIR/hardware/vivado\_design/lib/sim** directory.

# FACS General-Purpose Interface (SDAccel) Description

## FACS SDAccel Overview

Developed based on Xilinx SDAccel 2017.1, FACS SDAccel supports OpenCL development and application on VMs and is compatible with Xilinx SDAccel 2017.1. This chapter describes the implementation scheme of FACS SDAccel and how to develop host and kernel code.

## Terminology

HAL: indicates hardware abstraction layer.

XDMA: indicates Xilinx direct memory access.

Mgmt.: indicates management.

Drv.: indicates driver.

PF: indicates physical function.

OpenCL: indicates open computing language.

## FACS SDAccel Implementation Scheme Description

The SDAccel implementation scheme can be deployed on VMs, and you can develop, debug, and apply OpenCL host and kernel code on VMs. For security, the FPGA management is implemented by the host. Therefore, some kernel management initiated on VMs is implemented on the host, for example, loading an xclbin file.

FACS SDAccel implementation scheme



Figure description:

OpenCL runtime: indicates the runtime of Xilinx OpenCL and displays OpenCL APIs to users.

HAL: adapts OpenCL runtime and kernel driver and manages global memory addresses.

XDMA Drv.: indicates Xilinx DMA kernel driver.

fpga\_tool: provides basic FPGA management tools, such as the FPGA kernel loading command tool and loading status querying tool.

Host Mgmt. Drv.: indicates the management driver running on the host and implements FPGA kernel loading.

User PF: indicates the PF interface on the user plane and is directly connected to a VM, providing FPGA access channels for users.

Mgmt. PF: indicates the PF interface on the management plane. This interface is a channel for the host FPGA to access FPGA.

Based Region: indicates the area of FPGA static logic.

Expanded Region: indicates the area of FPGA dynamic logic.

User PF and Mgmt. PF are bound to user VM and server host respectively. Therefore, limited management commands can be executed directly on VMs. FACS SDAccel provides management tool fpga\_tool on VMs, and users need to use fpga\_tool for management operations on VMs. In addition, compared with Xilinx xbsak, fpga\_tool supports limited commands and does not support all xbsak commands. Therefore, some API users at the HAL layer cannot use fpga\_tool on VMs. The following table describes HAL APIs supported by FACS SDAccel.

| No. | API | Remarks |
| --- | --- | --- |
| 1 | xclOpen( ) |  |
| 2 | xclClose( ) |  |
| 3 | xclGetDeviceInfo2( ) | Support after modification. |
| 4 | xclGetAXIErrorStatus( ) |  |
| 5 | xclLoadXclBin( ) | Support after modification. |
| 6 | xclAllocDeviceBuffer( ) |  |
| 7 | xclAllocDeviceBuffer2( ) |  |
| 8 | xclFreeDeviceBuffer( ) |  |
| 9 | xclCopyBufferHost2Device( ) |  |
| 10 | xclCopyBufferDevice2Host( ) |  |
| 11 | xclWrite( ) |  |
| 12 | xclRead( ) |  |
| 13 | xclReClock2( ) | Support after modification. |
| 14 | xclProbe( ) |  |
| 15 | xclLockDevice( ) |  |
| 16 | xclWriteHostEvent( ) |  |
| 17 | xclGetDeviceTimestamp( ) |  |
| 18 | xclGetDeviceClockFreqMHz( ) |  |
| 19 | xclGetReadMaxBandwidthMBps( ) |  |
| 20 | xclGetWriteMaxBandwidthMBps( ) |  |
| 21 | xclSetOclRegionProfilingNumberSlots( ) |  |
| 22 | xclPerfMonClockTraining( ) |  |
| 23 | xclPerfMonStartCounters( ) |  |
| 24 | xclPerfMonStopCounters( ) |  |
| 25 | xclPerfMonReadCounters( ) |  |
| 26 | xclPerfMonStartTrace( ) |  |
| 27 | xclPerfMonStopTrace( ) |  |
| 28 | xclPerfMonGetTraceCount( ) |  |
| 29 | xclPerfMonReadTrace( ) |  |

## FACS SDAccel Development Description

### Host Code Development

Host code development supports all OpenCL APIs that are supported by Xilinx SDAccel 2017.1. For details, see Xilinx *UG1023*. You can develop host code based on the interfaces supported by Xilinx SDx 2017.1.

The following is the link of *UG1023*:

<https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_1/ug1023-sdaccel-user-guide.pdf>

### Kernel Code Development

Kernel code development supports OpenCL C syntax that is supported by Xilinx SDAccel 2017.1. For details, see *UG1023* and *UG1207*. Users can develop kernel code based on the syntax supported by Xilinx SDx 2017.1.

The following is the link of *UG1207*:

<https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_1/ug1207-sdaccel-optimization-guide.pdf>

## Example Design Description

FACS SDAccel provides three examples, which describe the process of development and debugging in three different kernel development modes, including RTL, HLS, and OpenCL C. Each example consists of two parts:

* The development and simulation paths of kernel are stored in the **/home/fpga\_design/hardware/sdaccel\_design/examples** directory, and corresponding execution scripts are provided.
* The development and on-board debugging paths of the host code are stored in the **/home/fpga\_design/software/app/sdaccel\_app** directory, and corresponding execution scripts are provided.
* example vadd\_cl

This example shows the kernel development process using OpenCL C. The kernel source code comes from **<SDx\_install\_area>SDx/2017.1/examples/vadd** of the Xilinx SDx. In addition, the example provides kernel compilation and simulation scripts. If you choose the OpenCL C development mode, replace the kernel source code with your own code, and then run the scripts provided by this example to perform compilation and simulation debugging, or on-board debugging.

* example mmult\_hls

This example shows the kernel development process using HLS. The kernel source code comes from **<SDx\_install\_area>SDx/2017.1/ samples/mmult** of the Xilinx SDx installation tool. In addition, the example provides kernel compilation and simulation scripts. If you choose the HLS development mode, replace the kernel source code with your own code, and then run the scripts provided by this example to perform compilation and simulation debugging, or on-board debugging.

* example vadd\_rtl

The example shows the kernel development process using HLS. The kernel source code comes from an open-source example on Xilinx github. For details, visit <https://github.com/Xilinx/SDAccel_Examples/tree/master/getting_started/rtl_kernel>. In addition, this example provides kernel compilation and simulation scripts. If you choose the RTL development mode, prepare the RTL kernel source code and XML description files, and then run the scripts provided by this example to perform compilation and simulation debugging, or on-board debugging.

For details about the SDAccel development and debugging process, visit <https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_1/ug1021-sdaccel-intro-tutorial.pdf>.