|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | | |  |
|  | | |
|  | | |
| 方圆ok1 | | | |
|  | **FACS接口手册** | | 附件1-16K |
|  | |
| **文档版本** | **01** |
| **发布日期** | **2017-07-30** |
| **华为技术有限公司** | |

|  |
| --- |
| 版权所有 © 华为技术有限公司2017。 保留一切权利。  非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。  商标声明  附件3-版权声明页图和其他华为商标均为华为技术有限公司的商标。  本文档提及的其他所有商标或注册商标，由各自的所有人拥有。  注意  您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或默示的声明或保证。  由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。 |

|  |  |
| --- | --- |
| 华为技术有限公司 | |
| 地址： | 深圳市龙岗区坂田华为总部办公楼 邮编：518129 |
| 网址： | <http://e.huawei.com> |
|  |  |
|  |  |

目 录

[目 录 iii](#_Toc514178950)

[1 FACS 接口手册简介 5](#_Toc514178951)

[1.1 概要 5](#_Toc514178952)

[1.2 术语介绍 5](#_Toc514178953)

[1.3 规格说明 6](#_Toc514178954)

[1.3.1 PCIe接口规格 6](#_Toc514178955)

[1.3.2 DDR接口规格 6](#_Toc514178956)

[1.3.3 队列规格 6](#_Toc514178957)

[1.3.4 BAR空间规格 6](#_Toc514178958)

[1.4 硬件说明 7](#_Toc514178959)

[1.4.1 硬件资源介绍 7](#_Toc514178960)

[1.4.2 动静态接口概述 7](#_Toc514178961)

[1.4.3 接口信号列表 8](#_Toc514178962)

[1.4.4 接口时序 15](#_Toc514178963)

[1.4.5 TX方向交互 16](#_Toc514178964)

[1.4.6 RX方向交互 17](#_Toc514178965)

[2 FACS高性能型(DPDK)接口说明 18](#_Toc514178966)

[2.1 SHELL接口说明 18](#_Toc514178967)

[2.1.1 简介 18](#_Toc514178968)

[2.1.2 Shell2UL BD接口 18](#_Toc514178969)

[2.1.3 UL2Shell BD接口 20](#_Toc514178970)

[2.1.4 Shell2UL data接口 21](#_Toc514178971)

[2.1.5 UL2Shell data接口 22](#_Toc514178972)

[2.1.6 AXI-Lite接口说明 24](#_Toc514178973)

[2.2 SDK接口说明 26](#_Toc514178974)

[2.2.1 针对VF设备的常用DPDK接口 26](#_Toc514178975)

[2.3 EXAMPLE2详细设计参考 37](#_Toc514178976)

[2.3.1 概要 37](#_Toc514178977)

[2.3.2 特性介绍 37](#_Toc514178978)

[2.3.3 HDK构建说明 38](#_Toc514178979)

[2.3.4 术语介绍 38](#_Toc514178980)

[2.3.5 设计约束 39](#_Toc514178981)

[2.4 EXAMPLE3详细设计参考 39](#_Toc514178982)

[2.4.1 概要 39](#_Toc514178983)

[2.4.2 特性介绍 39](#_Toc514178984)

[2.4.3 hardacc说明 40](#_Toc514178985)

[2.4.4 HDK构建说明 40](#_Toc514178986)

[2.4.5 术语介绍 41](#_Toc514178987)

[3 FACS通用型(SDAccel)接口说明 41](#_Toc514178988)

[3.1 FACS SDAccel概要说明 41](#_Toc514178989)

[3.2 术语介绍 41](#_Toc514178990)

[3.3 FACS SDAccel方案说明 42](#_Toc514178991)

[3.4 FACS SDAccel开发说明 44](#_Toc514178992)

[3.4.1 Host Code开发 44](#_Toc514178993)

[3.4.2 Kernel Code开发 44](#_Toc514178994)

[3.5 Example设计说明 44](#_Toc514178995)

# FACS 接口手册简介

## 概要

FACS接口手册为华为fpga云服务器用户在实现加速设计的时候提供统一的设计说明，包括用户可以使用的硬件资源、接口说明、语法和使用示例。

## 术语介绍

FACS：FPGA Accelerated Cloud Server，现场可编程门阵列加速云服务。

HDK：Hardware Develop Kit，硬件开发套件。包括编码环境、仿真平台、自动化编译工具、代码加密和调试工具等。

SDK：Software Develop Kit，软件开发套件。包括应用示例、硬件抽象接口、加速器抽象接口、加速器驱动和runtime、版本管理工具等。

SHELL：静态逻辑，由华为对外提供。包括PCIe、DDR4等外部接口。

UL：User Logic，用户逻辑，由开发者提供。

PF：Phsical Function。

VF：Virtual Function。

DMA：Direct Memory Access，直接存储器访问。

HPI：High Performance Interface，高性能接口。

BD：Buffer Description，缓存描述符。

PR：Partial Reconfigration，部分重加载技术。

AE：Acceleration Engine，加速引擎。

Bar：BAR Base Address Register，基地址寄存器。

AXI-4 ：ARM Advanced eXtensible Interface。

AXI-4 Stream：ARM Advanced eXtensible Stream Interface。

AXI-4 Lite ：ARM Advanced eXtensible Lite Interface。

M-指：AXI总线Master端。

S-指：AXI总线Slave端。

DPDK：Data Plane Development Kit，数据平面开发套件。

PMD：Poll Mode Drivers，轮询模式驱动。

## 规格说明

### PCIe接口规格

单个FPGA支持一个PCIe X16接口，最大支持110Gbps带宽。

### DDR接口规格

单个FPGA支持4个DDR4通道，单通道DDR4容量为16GB，共64GB。

### 队列规格

* 1个PF。
* 1个VF。
* 每个VF支持8个队列。

### BAR空间规格

UL的VF Bar空间使用规格：

* BAR2为64-bit BAR，大小为16MiByte。
* BAR4为64-bit BAR，大小为512MiByte。

## 硬件说明

### 硬件资源介绍



用户可以使用的资源如下：

* 1个PCIe Gen3 x16接口。
* 4个DDR4 DIMM接口。
* 9个100Gbps的ETH300G接口。

DDR区域划分：

* 1个DDR控制器放置在静态逻辑部分。
* 3个DDR控制器放置在用户逻辑部分。
* 支持用户最多使用4个DDR控制器。

### 动静态接口概述

每个FPGA用户可以使用的接口如下图所示：

****

1. 使用的是AXI4-Lite接口，数据位宽为32bit，主要实现寄存器访问和bar2空间映射功能。
2. 使用的是AXI4-Lite接口，数据位宽为32bit，主要实现寄存器访问和bar4空间映射功能，目前保留，用户不可使用该接口。
3. 使用的是AXI4-Stream接口，数据位宽为256bit，主要实现DMA BD功能。
4. 使用的是AXI4-Stream接口，数据位宽为512bit，主要实现DMA数据功能。
5. 使用的是AXI4接口，数据位宽为64bit，主要实现对SH的DDR读写访问功能。
6. 使用的是BSACN接口和寄存器接口，DEBUG的数据位宽为1bit，主要实现XVC功能。VLED的数据位宽为16bit，主要实现虚拟点灯功能。
7. 使用的是AXI4接口，数据位宽为64bit，主要实现对UL的DDR4\_A/B/D的读写访问功能。
8. 使用的是SERDES接口，数据位宽由用户自定义，主要实现串并转换等功能。

### 接口信号列表

| 信号名 | 位宽 | I/O | 描述 |
| --- | --- | --- | --- |
| **全局信号** | | | |
| clk\_200m | 1 | I | 系统工作时钟：200MHz |
| clk\_a | 1 | I | 用户工作时钟a：200MHz |
| clk\_b | 1 | I | 用户工作时钟b：200MHz |
| rst\_200m | 1 | I | 系统工作时钟200MHz对应的复位信号 |
| rst\_a | 1 | I | 用户工作时钟a对应的复位信号 |
| rst\_b | 1 | I | 用户工作时钟b对应的复位信号 |
| **UL和SHELL之间的读报文请求AXI-S接口（UL→SHELL）** | | | |
| ul2sh\_dmas2\_tlast | 1 | O | 用户逻辑给静态逻辑的TLAST，表明包的边界 |
| ul2sh\_dmas2\_tdata | 256 | O | 用户逻辑给静态逻辑的读报文请求 |
| ul2sh\_dmas2\_tkeep | 32 | O | 用户逻辑给静态逻辑的TKEEP字节修饰符，用来表明TDATA相关字节的内容是否作为数据流的一部分被处理 |
| ul2sh\_dmas2\_tvalid | 1 | O | 动态逻辑传输有效数据指示信号 |
| sh2ul\_dmas2\_tready | 1 | I | 静态逻辑给用户逻辑已经准备好接收数据信号 |
| **UL和SHELL之间的写报文请求AXI-S接口（UL→SHELL）** | | | |
| ul2sh\_dmas3\_tlast | 1 | O | 用户逻辑给静态逻辑的TLAST，表明包的边界 |
| ul2sh\_dmas3\_tdata | 512 | O | 用户逻辑给静态逻辑的写报文请求 |
| ul2sh\_dmas3\_tkeep | 64 | O | 用户逻辑给静态逻辑的TKEEP字节修饰符，用来表明TDATA相关字节的内容是否作为数据流的一部分被处理 |
| ul2sh\_dmas3\_tvalid | 1 | O | 动态逻辑传输有效数据指示信号 |
| sh2ul\_dmas3\_tready | 1 | I | 静态逻辑给用户逻辑已经准备好接收数据信号 |
| **UL和SHELL之间返回BD的AXI-S接口（SHELL→UL）** | | | |
| sh2ul\_dmam0\_tlast | 1 | I | 静态逻辑给用户逻辑的TLAST，表明包的边界 |
| sh2ul\_dmam0\_tdata | 256 | I | 静态逻辑给用户逻辑的返回BD |
| sh2ul\_dmam0\_tkeep | 32 | I | 静态逻辑给用户逻辑的TKEEP字节修饰符，用来表明TDATA相关字节的内容是否作为数据流的一部分被处理 |
| sh2ul\_dmam0\_tvalid | 1 | I | 静态逻辑传输有效数据指示信号 |
| ul2sh\_dmam0\_tready | 1 | O | 静态逻辑给用户逻辑已经准备好接收数据信号 |
| **UL和SHELL之间返回数据的AXI-S接口（SHELL→UL）** | | | |
| sh2ul\_dmam1\_tlast | 1 | I | 静态逻辑给用户逻辑的TLAST，表明包的边界 |
| sh2ul\_dmam1\_tdata | 512 | I | 静态逻辑给用户逻辑的返回数据 |
| sh2ul\_dmam1\_tkeep | 64 | I | 静态逻辑给用户逻辑的TKEEP字节修饰符，用来表明TDATA相关字节的内容是否作为数据流的一部分被处理 |
| sh2ul\_dmam1\_tvalid | 1 | I | 静态逻辑传输有效数据指示信号 |
| ul2sh\_dmam1\_tready | 1 | O | 静态逻辑给用户逻辑已经准备好接收数据信号 |
| **UL和DDRC之间的axi4接口** | | | |
| ul2sh\_ddr\_awid | 4 | O | 动态逻辑访问静态逻辑ddr写地址通道的ID信息 |
| ul2sh\_ddr\_awaddr | 64 | O | 动态逻辑访问静态逻辑ddr写地址通道的地址信息 |
| ul2sh\_ddr\_awlen | 8 | O | 动态逻辑访问静态逻辑ddr写地址通道的有效拍数信息 |
| ul2sh\_ddr\_awsize | 3 | O | 动态逻辑访问静态逻辑ddr写地址通道的地址每拍数据有效字节数信息 |
| ul2sh\_ddr\_awvalid | 1 | O | 动态逻辑访问静态逻辑ddr写地址通道的valid有效信息 |
| sh2ul\_ddr\_awready | 1 | I | 静态逻辑访问动态逻辑ddr写地址通道的ready有效信息 |
| ul2sh\_ddr\_wid | 4 | O | 动态逻辑访问静态逻辑ddr写数据通道的ID信息 |
| ul2sh\_ddr\_wdata | 512 | O | 动态逻辑访问静态逻辑ddr写数据通道的地址信息 |
| ul2sh\_ddr\_wstrb | 64 | O | 动态逻辑访问静态逻辑ddr写数据通道的有效拍数信息 |
| ul2sh\_ddr\_wlast | 1 | O | 动态逻辑访问静态逻辑ddr写数据通道的地址数据最后一拍 |
| ul2sh\_ddr\_wvalid | 1 | O | 动态逻辑访问静态逻辑ddr写数据通道的valid有效信息 |
| sh2ul\_ddr\_wready | 1 | I | 静态逻辑访问动态逻辑ddr写数据通道的ready有效信息 |
| sh2ul\_ddr\_bid | 4 | I | 静态逻辑和动态逻辑ddr写响应通道的ID信息 |
| sh2ul\_ddr\_bresp | 2 | I | 静态逻辑和动态逻辑ddr写响应通道的响应信息 |
| sh2ul\_ddr\_bvalid | 1 | I | 静态逻辑和动态逻辑ddr写响应通道的valid信息 |
| ul2sh\_ddr\_bready | 1 | O | 动态逻辑和静态逻辑ddr写响应通道的ready信息 |
| ul2sh\_ddr\_arid | 4 | O | 动态逻辑访问静态逻辑ddr读地址的通道ID信息 |
| ul2sh\_ddr\_araddr | 64 | O | 动态逻辑访问静态逻辑ddr读地址通道的地址信息 |
| ul2sh\_ddr\_arlen | 8 | O | 动态逻辑访问静态逻辑ddr读地址通道的有效拍数信息 |
| ul2sh\_ddr\_arsize | 3 | O | 动态逻辑访问静态逻辑ddr读地址通道的地址每拍数据有效字节数信息 |
| ul2sh\_ddr\_arvalid | 1 | O | 动态逻辑访问静态逻辑ddr读地址通道的valid有效信息 |
| sh2ul\_ddr\_arready | 1 | I | 静态逻辑访问动态逻辑ddr读地址通道的ready有效信息 |
| sh2ul\_ddr\_rid | 4 | I | 动态逻辑访问静态逻辑ddr读数据通道的ID信息 |
| sh2ul\_ddr\_rdata | 512 | I | 动态逻辑访问静态逻辑ddr读数据通道的地址信息 |
| sh2ul\_ddr\_rresp | 2 | I | 动态逻辑访问静态逻辑ddr读数据通道的有效拍数信息 |
| sh2ul\_ddr\_rlast | 1 | I | 动态逻辑访问静态逻辑ddr读数据通道的地址每拍数据最后一拍 |
| sh2ul\_ddr\_rvalid | 1 | I | 动态逻辑访问静态逻辑ddr读数据通道的valid有效信息 |
| ul2sh\_ddr\_rready | 1 | O | 静态逻辑访问动态逻辑ddr读数据通道的ready有效信息 |
| **BAR通道的AXI-L接口** | | | |
| sh2bar1\_awvalid | 1 | I | 对应VF的bar2通道，bar地址通道写有效指示 |
| sh2bar1\_awaddr | 32 | I | 对应VF的bar2通道，bar地址通道写有效指示 |
| bar12sh\_awready | 1 | O | 对应VF的bar2通道，bar地址通道写ready指示 |
| sh2bar1\_wvalid | 1 | I | 对应VF的bar2通道，bar数据通道写有效指示 |
| sh2bar1\_wdata | 32 | I | 对应VF的bar2通道，bar数据通道写数据指示 |
| sh2bar1\_wstrb | 4 | I | 对应VF的bar2通道，bar数据通道写strb指示 |
| bar12sh\_wready | 1 | O | 对应VF的bar2通道，bar数据通道写ready指示 |
| bar12sh\_bvalid | 1 | O | 对应VF的bar2通道，bar响应通道有效指示 |
| bar12sh\_bresp | 2 | O | 对应VF的bar2通道，bar响应通道有效信息 |
| sh2bar1\_bready | 1 | I | 对应VF的bar2通道，bar响应通道ready指示 |
| sh2bar1\_arvalid | 1 | I | 对应VF的bar2通道，bar地址通道读有效指示 |
| sh2bar1\_araddr | 32 | I | 对应VF的bar2通道，bar地址通道读有效指示 |
| bar12sh\_arready | 1 | O | 对应VF的bar2通道，bar地址通道读ready指示 |
| bar12sh\_rvalid | 1 | O | 对应VF的bar2通道，bar数据通道读有效指示 |
| bar12sh\_rdata | 32 | O | 对应VF的bar2通道，bar数据通道读数据指示 |
| bar12sh\_rresp | 1 | O | 对应VF的bar2通道，bar数据通道读strb指示 |
| sh2bar1\_rready | 1 | I | 对应VF的bar2通道，bar数据通道读ready指示 |
| **BAR4通道的AXI-L接口** | | | |
| sh2bar5\_awvalid | 1 | I | 对应VF的bar4通道，bar地址通道写有效指示 |
| sh2bar5\_awaddr | 32 | I | 对应VF的bar4通道，bar地址通道写有效指示 |
| bar52sh\_awready | 1 | O | 对应VF的bar4通道，bar地址通道写ready指示 |
| sh2bar5\_wvalid | 1 | I | 对应VF的bar4通道，bar数据通道写有效指示 |
| sh2bar5\_wdata | 32 | I | 对应VF的bar4通道，bar数据通道写数据指示 |
| sh2bar5\_wstrb | 4 | I | 对应VF的bar4通道，bar数据通道写strb指示 |
| bar52sh\_wready | 1 | O | 对应VF的bar4通道，bar数据通道写ready指示 |
| bar52sh\_bvalid | 1 | O | 对应VF的bar4通道，bar响应通道有效指示 |
| bar52sh\_bresp | 2 | O | 对应VF的bar4通道，bar响应通道有效信息 |
| sh2bar5\_bready | 1 | I | 对应VF的bar4通道，bar响应通道ready指示 |
| sh2bar5\_arvalid | 1 | I | 对应VF的bar4通道，bar地址通道读有效指示 |
| sh2bar5\_araddr | 32 | I | 对应VF的bar4通道，bar地址通道读有效指示 |
| bar52sh\_arready | 1 | O | 对应VF的bar4通道，bar地址通道读ready指示 |
| bar52sh\_rvalid | 1 | O | 对应VF的bar4通道，bar数据通道读有效指示 |
| bar52sh\_rdata | 32 | O | 对应VF的bar4通道，bar数据通道读数据指示 |
| bar52sh\_rresp | 1 | O | 对应VF的bar4通道，bar数据通道读strb指示 |
| sh2bar5\_rready | 1 | I | 对应VF的bar4通道，bar数据通道读ready指示 |
| **DDRA、DDRB、DDRD接口信号** | | | |
| ddra\_100m\_ref\_clk\_p | 1 | I | ddra 100MHz差分参考时钟p端 |
| ddra\_100m\_ref\_clk\_n | 1 | I | ddra 100MHz差分参考时钟n端 |
| ddrb\_100m\_ref\_clk\_p | 1 | I | ddrb 100MHz差分参考时钟p端 |
| ddrb\_100m\_ref\_clk\_n | 1 | I | ddrb 100MHz差分参考时钟n端 |
| ddrd\_100m\_ref\_clk\_p | 1 | I | ddrd 100MHz差分参考时钟p端 |
| ddrd\_100m\_ref\_clk\_n | 1 | I | ddrd 100MHz差分参考时钟n端 |
| ddra\_72b\_act\_n | 1 | O | ddra的act\_n信号 |
| ddra\_72b\_addr | 17 | O | ddra的addr信号 |
| ddra\_72b\_ba | 2 | O | ddra的ba信号 |
| ddra\_72b\_bg | 2 | O | ddra的bd信号 |
| ddra\_72b\_cke | 2 | O | ddra的cke信号 |
| ddra\_72b\_odt | 2 | O | ddra的odt信号 |
| ddra\_72b\_cs\_n | 2 | O | ddra的cs\_n信号 |
| ddra\_72b\_ck\_t | 1 | O | ddra的ck\_t信号 |
| ddra\_72b\_ ck\_c | 1 | O | ddra的ck\_c信号 |
| ddra\_72b\_rst\_n | 1 | O | ddra的rst\_n信号 |
| ddra\_72b\_par | 1 | O | ddra的par信号 |
| ddra\_72b\_dq | 72 | IO | ddra的dq信号 |
| ddra\_72b\_dqs\_t | 9 | IO | ddra的dqs\_t信号 |
| ddra\_72b\_dqs\_c | 9 | IO | ddra的dqs\_c信号 |
| ddra\_72b\_dm\_dbi\_n | 9 | IO | ddra的dm\_dbi\_n信号 |
| ddrb\_72b\_act\_n | 1 | O | ddrb的act\_n信号 |
| ddrb\_72b\_addr | 17 | O | ddrb的addr信号 |
| ddrb\_72b\_ba | 2 | O | ddrb的ba信号 |
| ddrb\_72b\_bg | 2 | O | ddrb的bd信号 |
| ddrb\_72b\_cke | 2 | O | ddrb的cke信号 |
| ddrb\_72b\_odt | 2 | O | ddrb的odt信号 |
| ddrb\_72b\_cs\_n | 2 | O | ddrb的cs\_n信号 |
| ddrb\_72b\_ck\_t | 1 | O | ddrb的ck\_t信号 |
| ddrb\_72b\_ ck\_c | 1 | O | ddrb的ck\_c信号 |
| ddrb\_72b\_rst\_n | 1 | O | ddrb的rst\_n信号 |
| ddrb\_72b\_par | 1 | O | ddrb的par信号 |
| ddrb\_72b\_dq | 72 | IO | ddrb的dq信号 |
| ddrb\_72b\_dqs\_t | 9 | IO | ddrb的dqs\_t信号 |
| ddrb\_72b\_dqs\_c | 9 | IO | ddrb的dqs\_c信号 |
| ddrb\_72b\_dm\_dbi\_n | 9 | IO | ddrb的dm\_dbi\_n信号 |
| ddrd\_72b\_act\_n | 1 | O | ddrd的act\_n信号 |
| ddrd\_72b\_addr | 17 | O | ddrd的addr信号 |
| ddrd\_72b\_ba | 2 | O | ddrd的ba信号 |
| ddrd\_72b\_bg | 2 | O | ddrd的bd信号 |
| ddrd\_72b\_cke | 2 | O | ddrd的cke信号 |
| ddrd\_72b\_odt | 2 | O | ddrd的odt信号 |
| ddrd\_72b\_cs\_n | 2 | O | ddrd的cs\_n信号 |
| ddrd\_72b\_ck\_t | 1 | O | ddrd的ck\_t信号 |
| ddrd\_72b\_ ck\_c | 1 | O | ddrd的ck\_c信号 |
| ddrd\_72b\_rst\_n | 1 | O | ddrd的rst\_n信号 |
| ddrd\_72b\_par | 1 | O | ddrd的par信号 |
| ddrd\_72b\_dq | 72 | IO | ddrd的dq信号 |
| ddrd\_72b\_dqs\_t | 9 | IO | ddrd的dqs\_t信号 |
| ddrd\_72b\_dqs\_c | 9 | IO | ddrd的dqs\_c信号 |
| ddrd\_72b\_dm\_dbi\_n | 9 | IO | ddrd的dm\_dbi\_n信号 |
| **BSCAN接口** | | | |
| S\_BSCAN\_drck | 1 | I | BSACN接口的drck信号 |
| S\_BSCAN\_shift | 1 | I | BSACN接口的shift信号 |
| S\_BSCAN\_tdi | 1 | I | BSACN接口的tdi信号 |
| S\_BSCAN\_update | 1 | I | BSACN接口的update信号 |
| S\_BSCAN\_sel | 1 | I | BSACN接口的sel信号 |
| S\_BSCAN\_tdo | 1 | O | BSACN接口的tdo信号 |
| S\_BSCAN\_tms | 1 | I | BSACN接口的tms信号 |
| S\_BSCAN\_tck | 1 | I | BSACN接口的tck信号 |
| S\_BSCAN\_runtest | 1 | I | BSACN接口的runtest信号 |
| S\_BSCAN\_reset | 1 | I | BSACN接口的reset信号 |
| S\_BSCAN\_capture | 1 | I | BSACN接口的capture信号 |
| S\_BSCAN\_bscanid\_en | 1 | I | BSACN接口的bscanid信号 |

### 接口时序

AXI详细接口时序请参见。

<http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.set.amba/index.html>

交互方式说明：华为FPGA云服务器提供基于DPDK PMD驱动的Shell框架，提供了高性能、低时延、高可靠性的PCIe DMA通路。

定义：CPU内存数据到UL为发送方向（Transmit，下文简称TX），UL到CPU内存为接收方向(Receive，下文简称RX)。

### TX方向交互



TX方向：

1. 应用（Application APP）通过DPDK接口申请mbuf（包括BD的跟报文的，共两份），并将待处理数据及构造好的BD（Buffer Description）填充到各自的mbuf地址域。
2. APP调用DPDK的TX接口，DPDK再调用PMD接口，PMD将APP缓存的BD信息复制到与Shell逻辑约定的BD队列中。
3. DPDK PMD驱动通过BAR通路将BD地址信息发送给Shell逻辑。
4. Shell逻辑根据BD地址、长度信息通过DMA读操作获取BD。
5. Shell逻辑将BD信息通过AXI接口发送给UL逻辑。
6. UL逻辑产生DMA读命令操作并发送给Shell逻辑。
7. Shell逻辑通过DMA读操作从CPU内存获取待处理数据。
8. Shell逻辑将待处理数据发送给UL逻辑。

### RX方向交互



RX方向：

1. APP调用DPDK PMD驱动使能接收方向。
2. DPDK PMD驱动通过BAR通路向Shell逻辑分配空闲BD地址。
3. Shell逻辑获取并缓存空闲BD地址。
4. UL逻辑将处理完成的报文发送给Shell逻辑。
5. Shell逻辑通过DMA写操作向CPU内存回写处理完成的数据。
6. Shell逻辑按队列分配空闲BD地址信息。
7. Shell逻辑通过DMA写操作回写处理响应的结果，这里存储处理后数据的地址、长度等信息。
8. APP通过DPDK PMD驱动获取处理完成的报文以及处理响应的结果。

# FACS高性能型(DPDK)接口说明

## SHELL接口说明

### 简介

Shell/UL接口说明章节主要描述Shell逻辑与用户逻辑（User Logic）之间的每个接口的详细说明。

****

### Shell2UL BD接口

功能介绍

该接口用于Shell静态逻辑发送BD给UL用户逻辑。

image013

用户在APP层准备好待处理数据，驱动将创建与之关联的BD（具体格式见BD格式），并下发给Shell静态逻辑，Shell静态逻辑收到BD后，通过AXI-S接口转发给UL用户逻辑。

Shell发送给UL的BD数据格式表

Shell2UL BD Data Format

| 名称 | bit 位置 | 描述 |
| --- | --- | --- |
| odd\_even | [255:248] | [255]:[247:224]偶校验值  [254]:[223:192]偶校验值  [253]:[191:160]偶校验值  [252]:[159:128]偶校验值  [251]:[127:96]偶校验值  [250]:[95:64]偶校验值  [249]:[63:32]偶校验值  [248]:[31:0]偶校验值 |
| bd\_code | [247:240] | BD校验码，固定为8’h5a |
| rsv | [239:234] | 保留，固定为0 |
| thread\_id | [233:226] | 线程id号（此域仅对example3有效） |
| opcode | [225:224] | 2'd1:只读操作 2'd2:只写操作  2'd0，2'd3:环回处理操作  （此域仅对example3有效） |
| rsv | [223:208] | 保留，固定为0 |
| sh\_info | [207:160] | [207:192]VM\_id  [191:176]PF/VF id  [175:160]单VF/PF下挂的队列ID |
| length | [159:128] | 该BD所带需要UL处理的数据总长度 |
| des\_addr | [127:64] | 处理完成信息的目的物理地址（驱动预先为RX方向UL回写报文申请的地址起始值） |
| src\_addr | [63:0] | 待处理信息的源地址 |

1. Shell信息（sh\_info）：VM\_id表示该BD来自于哪个虚拟机。PF/VF id表示该BD来自PCIE的哪个PF/VF。队列ID表示该BD来自于哪个队列。
2. 源地址（src\_addr）：该BD关联的待处理数据存放的物理地址。
3. 长度（length）：该BD关联的待处理数据的长度。

接口信号列表

| 名称 | 位宽 | IO | 描述 |
| --- | --- | --- | --- |
| sh2ul\_dmam0\_tvalid | 1 | I | axi-s master valid信号 |
| ul2sh\_dmam0\_tready | 1 | O | axi-s slave ready信号 |
| sh2ul\_dmam0\_tdata | 256 | I | axi-s master写数据 |
| sh2ul\_dmam0\_tlast | 1 | I | axi-s写数据包尾（最后一拍） |
| sh2ul\_dmam0\_tkeep | 32 | I | axi-s keep信号 |

### UL2Shell BD接口

功能介绍

该接口用于UL用户逻辑发送读数据命令给Shell静态逻辑。

image013

UL用户逻辑需要根据接收到的Shell BD中的信息，构造出用户读数据命令（具体格式见BD格式）。通过AXI-S接口发送给Shell逻辑，Shell逻辑将根据该读数据命令，从内存取回用户所需要的数据。

UL发送给Shell逻辑的BD数据格式表

UL2Shell BD Data Format

| 名称 | bit 位置 | 描述 |
| --- | --- | --- |
| ul\_info | [255:224] | UL私有域信息，需要Shell读报文返回携带 |
| rsv | [223:208] | 保留，固定填0 |
| sh\_info | [207:160] | [207:192]VM\_id  [191:176]PF/VF id  [175:160]单VF/PF下挂的队列ID  Shell关注，UL透传 |
| length | [159:128] | 该BD所带需要UL处理的报文总长度 |
| rsv | [127:64] | 保留 |
| src\_addr | [63:0] | 待处理信息的源物理地址,TX BD中得到，UL根据读取长度进行相应调整 |

1. UL信息（ul\_info）：用户私有信息，Shell返回数据给UL用户逻辑时，会原路带回。
2. Shell信息（sh\_info）：从Shell BD直接获得。
3. 源地址（src\_addr）：从Shell BD直接获得。
4. 长度（length）：从Shell BD直接获得，如果大于4K长度，需要用户自行切割到小于等于4K。

接口信号列表

| 名称 | 位宽 | IO | 描述 |
| --- | --- | --- | --- |
| ul2sh\_dmas2\_tvalid | 1 | O | axi-s master valid信号 |
| sh2ul\_dmas2\_tready | 1 | I | axi-s slave ready信号 |
| ul2sh\_dmas2\_tdata | 256 | O | axi-s master写数据 |
| ul2sh\_dmas2\_tlast | 1 | O | axi-s写数据包尾（最后一拍） |
| sh2ul\_dmas2\_tkeep | 32 | O | axi-s keep信号 |

### Shell2UL data接口

功能介绍

该接口用于Shell静态逻辑向UL发送待处理的原始报文信息。

image013

UL用户逻辑发送读报文请求后，Shell静态逻辑返回待处理的数据报文信息。通过AXI-S接口发送给UL逻辑，UL根据返回的报文进行相应的处理。

Shell逻辑返回给UL逻辑的待处理报文

Shell逻辑返回给UL逻辑的待处理报文为多拍数据，多拍数据如下表。

Shell2UL Pkt Data Format

| 名称 | 拍位置 | 位宽 | 描述 |
| --- | --- | --- | --- |
| pkt header | cycle 1 | 512 | 报文头信息 |
| pkt payload | cycle 2~cycle n | 512 | 待加速报文 |

其中首拍pkt header的数据格式如下表：

Shell2UL Pkt Header Data Format

| 名称 | bit位置 | 描述 |
| --- | --- | --- |
| rsv | [511:256] | 保留域固定为0 |
| ul\_info | [255:224] | UL私有域信息，需要Shell读报文返回携带 |
| rsv | [223:216] | 保留域 |
| acc\_type | [215:208] | 保留域 |
| sh\_info | [207:160] | [207:192]VM\_id  [191:176]PF/VF id  [175:160]单VF/PF下挂的队列ID |
| rsv | [159:128] | 保留域 |
| rsv | [127:64] | 保留域 |
| rsv | [63:0] | 保留域 |

1. UL信息（ul\_info）：用户私有信息，Shell返回数据给UL用户逻辑时，会原路带回。
2. Shell信息（sh\_info）：从Shell BD直接获得。

接口信号列表

| 名称 | 位宽 | IO | 描述 |
| --- | --- | --- | --- |
| sh2ul\_dmam1\_tvalid | 1 | I | axi-s master valid信号 |
| ul2sh\_dmam1\_tready | 1 | O | axi-s slave ready信号 |
| sh2ul\_dmam1\_tdata | 512 | I | axi-s master写数据 |
| sh2ul\_dmam1\_tlast | 1 | I | axi-s写数据包尾（最后一拍） |
| sh2ul\_dmam1\_tkeep | 64 | I | axi-s keep信号 |

### UL2Shell data接口

功能介绍

该接口用于UL逻辑向Shell逻辑返回处理后的报文信息。

image013

UL用户逻辑处理完成数据后通过AXI-M接口发送给Shell逻辑，UL逻辑需要维护计算报文返回目的地址Des\_addr，并且在报文全部上送完成后上送交互完成信息，其中包含报文done\_flg标志，src\_addr以及des\_addr。



最大支持4Kbyte交互。

Shell逻辑返回给UL逻辑的待处理报文

UL逻辑返回给SHELL逻辑的待处理报文为多拍数据，多拍数据如下表。

UL2Shell Pkt Data Format

| **名称** | **拍位置** | **位宽** | **描述** |
| --- | --- | --- | --- |
| pkt header | cycle 1 | 512 | 报文头信息 |
| pkt payload | cycle 2~cycle n | 512 | 待加速报文 |

其中首拍pkt header的数据格式如下表：

UL2Shell Pkt Header Data Format

| **名称** | **位置** | **描述** |
| --- | --- | --- |
| Rsv | [255:217] | 保留域 |
| done\_flg | [216] | done\_flg标记，由UL填写 |
| Acc\_type | [215:208] | 保留域 |
| sh\_info | [207:160] | [207:192]VM\_id  [191:176]PF/VF id  [175:160]单VF/PF下挂的队列ID填写  此域从TX BD中得到，需要UL透传，不可更改。 |
| Rsv | [159:128] | 保留域 |
| Des\_addr | [127:64] | 处理完成信息的目的物理地址 |
| Src\_addr | [63:0] | 处理完成信息的原始物理地址 |

1. 加速完成标志（done\_flag）：报文加速完成标志，当此标志位1时，指示本次报文处理完成，可以通知软件驱动去取报文。
2. shell信息（sh\_info）：从shell TX BD直接获得。
3. 目的地址（Des\_addr）：加速完成后报文返回CPU内存物理地址。
4. 源地址（src\_addr）：从Shell TX BD直接获得。

接口信号列表

| **名称** | **位宽** | **IO** | **描述** |
| --- | --- | --- | --- |
| ul2sh\_dmas3\_tvalid | 1 | O | axi-s master valid信号 |
| sh2ul\_dmas3\_tready | 1 | I | axi-s slave ready信号 |
| ul2sh\_dmas3\_tdata | 512 | O | axi-s master写数据 |
| ul2sh\_dmas3\_tlast | 1 | O | axi-s写数据包尾（最后一拍） |
| ul2sh\_dmas3\_tkeep | 64 | O | axi-s keep信号 |

举例

Shell逻辑发送给UL BD中Src\_addr=0x10000000，Len=11520，Des\_addr=0x20000000。UL逻辑产生读取报文的BD需切分为3个，处理完成报文长度和原始报文长度一致，分别对应的BD信息如下。

* src\_addr len构造：

BD0：SRC\_addr=0x10000000，LEN=0x1000。

BD1：SRC\_addr=0x10001000，LEN=0x1000。

BD2：SRC\_addr=0x10002000，LEN=0xD00。

* pkt header构造：

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=1，Des\_addr=0x20002000 SRC\_addr=0x10002000。

### AXI-Lite接口说明

功能介绍

本接口提供软件访问动态逻辑内部寄存器的功能。动态逻辑作为从机，接入了两组标准的AXI-Lite接口信号，用来访问不同的BAR（BAR2和BAR4）。

AXI-Lite接口信号列表

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名** | **位宽** | **IO** | **描述** |
| clk\_200m | 1 | I | 工作时钟，200MHz |
| rst\_200m | 1 | I | 复位信号，高电平有效 |
| sh2bar1\_awvalid | 1 | I | BAR2通道，写操作地址有效信号，主机通知从机写地址有效 |
| sh2bar1\_awaddr | 32 | I | BAR2通道，写操作地址信号 |
| bar12sh\_awready | 1 | O | BAR2通道，写操作地址通道准备就绪信号，从机通知主机已做好地址传输准备 |
| sh2bar1\_wvalid | 1 | I | BAR2通道，写操作数据有效信号，主机通知从机写数据有效 |
| sh2bar1\_wdata | 32 | I | BAR2通道，写操作数据信号 |
| sh2bar1\_wstrb | 4 | I | BAR2通道，写操作数据选通信号，1bit选通1Byte数据，当前固定输入4’hF，即写数据32bit都有效 |
| bar12sh\_wready | 1 | O | BAR2通道，写操作数据通道准备就绪信号，从机通知主机已做好数据传输准备 |
| bar12sh\_bvalid | 1 | O | BAR2通道，写操作应答状态有效信号 |
| bar12sh\_bresp | 2 | O | BAR2通道，写操作应答状态信号。2‘h0代表成功，其他为错误。 |
| sh2bar1\_bready | 1 | I | BAR2通道，写操作应答信号，主机通知从机已收到应答 |
| sh2bar1\_arvalid | 1 | I | BAR2通道，读操作地址有效信号，主机通知从机读操作地址有效 |
| sh2bar1\_araddr | 32 | I | BAR2通道，读操作地址信号 |
| bar12sh\_arreay | 1 | O | BAR2通道，读操作地址通道准备就绪信号，从机通知主机已做好地址传输准备 |
| bar12sh\_rvalid | 1 | O | BAR2通道，读操作数据有效信号，从机通知主机读数据有效 |
| bar12sh\_rdata | 32 | O | BAR2通道，读操作返回的数据信号 |
| sh2bar1\_rready | 1 | I | BAR2通道，读操作数据通道准备就绪信号，主机通知从机已做好数据传输准备 |
| bar12sh\_rresp | 2 | O | BAR2通道，读操作应答状态信号。2‘h0代表成功，其他为错误 |
| sh2bar5\_awvalid | 1 | I | BAR4通道，写操作地址有效信号，主机通知从机写地址有效 |
| sh2bar5\_awaddr | 32 | I | BAR4通道，写操作地址信号 |
| bar52sh\_awready | 1 | O | BAR4通道，写操作地址通道准备就绪信号，从机通知主机已做好地址传输准备 |
| sh2bar5\_wvalid | 1 | I | BAR4通道，写操作数据有效信号，主机通知从机写数据有效 |
| sh2bar5\_wdata | 32 | I | BAR4通道，写操作数据信号 |
| sh2bar5\_wstrb | 4 | I | BAR4通道，写操作数据选通信号，1bit选通1Byte数据，当前固定输入4’hF，即写数据32bit都有效 |
| bar52sh\_wready | 1 | O | BAR4通道，写操作数据通道准备就绪信号，从机通知主机已做好数据传输准备 |
| bar52sh\_bvalid | 1 | O | BAR4通道，写操作应答状态有效信号 |
| bar52sh\_bresp | 2 | O | BAR4通道，写操作应答状态信号。2‘h0代表成功，其他为错误 |
| sh2bar5\_bready | 1 | I | BAR4通道，写操作应答信号，主机通知从机已收到应答 |
| sh2bar5\_arvalid | 1 | I | BAR4通道，读操作地址有效信号，主机通知从机读操作地址有效 |
| sh2bar5\_araddr | 32 | I | BAR4通道，读操作地址信号 |
| bar52sh\_arreay | 1 | O | BAR4通道，读操作地址通道准备就绪信号，从机通知主机已做好地址传输准备 |
| bar52sh\_rvalid | 1 | O | BAR4通道，读操作数据有效信号，从机通知主机读数据有效 |
| bar52sh\_rdata | 32 | O | BAR4通道，读操作返回的数据信号 |
| sh2bar5\_rready | 1 | I | BAR4通道，读操作数据通道准备就绪信号，主机通知从机已做好数据传输准备 |
| bar52sh\_rresp | 2 | O | BAR4通道，读操作应答状态信号。2‘h0代表成功，其他为错误 |

## SDK接口说明

SDK采用由Intel主导的开源DPDK框架作为开发和部署基础。DPDK是用于快速数据包转发和处理的函数库与驱动集合，它具有以下典型特点：

1. 可以极大提高数据处理性能和吞吐量，提高数据平面应用程序的工作效率。
2. 用户界面标准化，开发流程统一化。
3. 在用户层开发PCIe设备的业务，降低用户开发难度。
4. 简易的内存控制方案，可直接在应用层操控物理地址。

该文档描述了在FPGA服务化项目中，DPDK的接口使用。

### 针对VF设备的常用DPDK接口

SDK基于DPDK-16.04，集成华为公司研发的针对本FPGA VF的高性能PMD模块，用户直接使用标准的DPDK接口进行数据包封装和处理。

#### DPDK接口与PMD的集成



SDK部分作为业务程序与底层逻辑的中间层，向上提供用户接口，向下提供逻辑驱动。

SDK基于DPDK框架进行设计，主要包括DPDK用户控制接口、DPDK用户内存管理接口和PMD三部分。SDK保留DPDK原始的内存管理模块和用户接口函数，底层设备控制信息交互在PMD模块内实现。

DPDK用户控制接口部分提供设备配置与数据交互服务，包括初始化DPDK环境、FPGA队列配置和业务RX/TX功能。

DPDK用户内存管理接口负责DPDK环境的内存管理，每个包数据放到DPDK的rte\_mbuf结构中，以内存池的方式进行管理。

PMD模块对应具体的设备操作，通过回调函数的形式挂载到DPDK接口，是CPU与底层逻辑进行控制信息交互的软件实体。

#### 数据流图



1. 应用程序通过DPDK mbuf接口，构建控制消息BD，通过rte\_eth\_tx\_burst()接口拷贝到BD队列中。
2. 将新的BD元素从BD队列拷贝到FPGA。
3. 报文下发到FPGA。
4. 处理结果返回输出报文空间。
5. 将响应BD写入到RX队列中。
6. 将响应BD结果队列返回应用程序。

#### 基本调用流程说明

用户使用DPDK标准接口可以方便的构建一个应用流程，以下是利用DPDK接口进行TX/RX行为的简单流程图。



以上的流程图并不是固定不变的，客户可以根据场景需要调整。比如，针对初始化和业务可以使用多进程的方式，针对Tx/Rx可以使用多个线程的方式，对动态IP的TX输入空间和RX输出空间不一定非要使用rte\_mbuf的方式承载，也可以使用memzone的方式等。

#### DPDK常用接口列表

|  |  |  |
| --- | --- | --- |
| 初始化、管理、配置接口 | rte\_eal\_init | DPDK环境初始化 |
| rte\_eth\_dev\_count | DPDK所管理的设备个数 |
| rte\_eth\_dev\_config | 配置设备参数 |
| rte\_eth\_rx\_queue\_setup | 配置设备RX队列参数 |
| rte\_eth\_tx\_queue\_setup | 配置设备TX队列参数 |
| rte\_eth\_dev\_start | 开启设备 |
| rte\_eth\_dev\_stop | 停止设备 |
| rte\_eth\_dev\_close | 关闭设备 |
| rte\_eth\_dev\_detach | 删除设备 |
| TX/RX接口 | rte\_eth\_rx\_burst | RX接口 |
| rte\_eth\_tx\_burst | TX接口 |
| 内存管理接口 | rte\_mempool\_create | 创建内存池 |
| rte\_pktmbuf\_alloc | 从内存池中申清mbuf |
| rte\_pktmbuf\_free | 释放mbuf |
| rte\_pktmbuf\_alloc\_bulk | 从内存池中申请多个mbuf |
| 线程控制接口 | rte\_eal\_remote\_launch | 指定一个CPU核执行线程 |
| rte\_eal\_wait\_lcore | 等待一个CPU核的线程执行完成 |

#### DPDK常用接口详细说明

1. rte\_eal\_init

|  |  |
| --- | --- |
| 函数原型 | int rte\_eal\_init(int argc,char \*\*argv) |
| 函数功能 | 初始化DPDK环境 |
| 输入说明 | argc：参数个数  argv：输入参数 |
| 输出说明 | NA |
| 返回值说明 | On success>=0  On failure, -1并且会对rte\_errno赋值，该值指示了错误原因 |
| 使用说明 | 该函数执行在MASTER lcore，尽可能在应用程序的main()函数中执行 |
| 约束 | 一个进程只能调用一个 |

1. rte\_eth\_dev\_count

|  |  |
| --- | --- |
| 函数原型 | uint8\_t rte\_eth\_dev\_count(void) |
| 函数功能 | 获取DPDK所管理的设备个数 |
| 输入说明 | NA |
| 输出说明 | NA |
| 返回值说明 | DPDK所管理设备的个数 |
| 使用说明 | DPDK所管理的设备个数是由系统本身的硬件资源和所加载的pmd驱动共同决定 |
| 约束 | 在使用rte\_eal\_init函数后调用 |

1. rte\_eth\_dev\_configure

|  |  |
| --- | --- |
| 函数原型 | 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) |
| 函数功能 | 配置某个设备 |
| 输入说明 | port\_id：要配置的设备id  nb\_rx\_queue：该设备的最大RX队列个数，本项目为8  nb\_tx\_queue：该设备的最大TX队列个数，本项目为8  eth\_conf：设备配置参数 |
| 输出说明 | NA |
| 返回值说明 | 0：成功  <0：失败 |
| 使用说明 | 在使用各个针对port id的接口之前，该函数必须先调用。对于本项目而言，eth\_conf的各个成员变量可以为0，但是eth\_conf不能为NULL |
| 约束 | NA |

1. rte\_eth\_rx\_queue\_setup

|  |  |
| --- | --- |
| 函数原型 | 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) |
| 函数功能 | 配置某个设备的某个RX队列 |
| 输入说明 | port\_id：要配置的设备id  rx\_queue\_id：要配置的RX队列id，该值范围为[0, ,nb\_rx\_queue-1]，nb\_rx\_queue就是rte\_eth\_config的rx队列参数  nb\_tx\_desc：RX队列的最大接收空间的大小，本项目为1024、2048、4096或8192  socket\_id：用于NUMA系统，如果是非NUMA系统，使用SOCKET\_ID\_ANY(-1)  rx\_conf：本项目使用NULL  mb\_pool：用于分配RX队列最大接收空间的内存池 |
| 输出说明 | NA |
| 返回值说明 | 0：成功  EINVAL：参数错误  ENOMEM：不能从内存池中分配足够的最大接收空间 |
| 使用说明 | NA |
| 约束 | NA |

1. rte\_eth\_tx\_queue\_setup

|  |  |
| --- | --- |
| 函数原型 | 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) |
| 函数功能 | 配置某个设备的某个TX队列 |
| 输入说明 | port\_id：要配置的设备id  tx\_queue\_id：要配置的TX队列id，该值范围为[0, ,nb\_tx\_queue-1]，nb\_tx\_queue就是rte\_eth\_config的tx队列参数  nb\_tx\_desc：TX队列的最大接收空间的大小，本项目为1024、2048、4096或8192  socket\_id：用于NUMA系统，如果是非NUMA系统，使用SOCKET\_ID\_ANY(-1) |
| 输出说明 | NA |
| 返回值说明 | 0：成功  ENOMEM：不能分配足够的TX队列空间描述符 |
| 使用说明 | NA |
| 约束 | NA |

1. rte\_eth\_dev\_start

|  |  |
| --- | --- |
| 函数原型 | int rte\_eth\_dev\_start(uint8\_t port\_id) |
| 函数功能 | 开启设备 |
| 输入说明 | port\_id：设备id |
| 输出说明 | NA |
| 返回值说明 | 0：成功  <0：失败 |
| 使用说明 | 该函数是设备配置的最后一个接口，之后就可以调用TX/RX业务接口。在本项目中，该接口对设备要使用的每一个队列都进行了复位操作 |
| 约束 | 需要在tx/rx\_queue\_setup之后调用 |

1. rte\_eth\_dev\_stop

|  |  |
| --- | --- |
| 函数原型 | void rte\_eth\_dev\_stop(uint8\_t port\_id) |
| 函数功能 | 设备停止工作，本项目中，该接口会关闭设备的各个队列使能 |
| 输入说明 | port\_id：设备id |
| 输出说明 | NA |
| 返回值说明 | NA |
| 使用说明 | NA |
| 约束 | NA |

1. rte\_eth\_dev\_close

|  |  |
| --- | --- |
| 函数原型 | void rte\_eth\_dev\_close(uint8\_t port\_id) |
| 函数功能 | 关闭一个设备 |
| 输入说明 | port\_id：设备id |
| 输出说明 | NA |
| 返回值说明 | NA |
| 使用说明 | 关闭后设备不能再开启！该接口会释放大部分资源，如果要释放所有资源，还需要调用rte\_eth\_dev\_detach() |
| 约束 | NA |

1. rte\_eth\_dev\_detach

|  |  |
| --- | --- |
| 函数原型 | int rte\_eth\_dev\_detach(uint8\_t port\_id, char \*devname) |
| 函数功能 | 从DPDK中删除设备 |
| 输入说明 | port\_id：设备id |
| 输出说明 | devname：被删除设备的名字 |
| 返回值说明 | 0 on success and devname is filled, negative on error |
| 使用说明 | 该函数需要在close后被调用 |
| 约束 | NA |

1. rte\_eth\_tx\_burst

|  |  |
| --- | --- |
| 函数原型 | static uint16\_t rte\_eth\_tx\_burst(uint8\_t port\_id, uint16\_t queue\_id, struct rte\_mbuf\*\* tx\_pkts, uint16\_t nb\_pkts) |
| 函数功能 | Send a burst of output packets on a transmit queue of an Ethernet device |
| 输入说明 | port\_id：设备id  queue\_id：逻辑队列号  tx\_pkts：tx的输入数组  nb\_pkts：tx的输入元素个数 |
| 输出说明 | NA |
| 返回值说明 | 返回实际将tx\_pkts存放在内部TX队列描述法的个数 |
| 使用说明 | tx\_pkts数组是rte\_mbuf结构体指针，每一个都是从rte\_pktmbuf\_pool\_create()分配的内存池中申请 |
| 约束 | NA |

1. rte\_eth\_rx\_burst

|  |  |
| --- | --- |
| 函数原型 | static uint16\_t rte\_eth\_rx\_burst(uint8\_t port\_id, uint16\_t queue\_id, struct rte\_mbuf\*\* rx\_pkts, uint16\_t nb\_pkts) |
| 函数功能 | 返回Rx队列中的描述符 |
| 输入说明 | port\_id：设备id  queue\_id：逻辑队列id  nb\_pkts：最大接收个数 |
| 输出说明 | rx\_pkts：接收数组 |
| 返回值说明 | 返回实际接收个数 |
| 使用说明 | NA |
| 约束 | NA |

1. rte\_mempool\_create

|  |  |
| --- | --- |
| 函数原型 | 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) |
| 函数功能 | 创建一个命名为name的内存池，用于分配rte\_mbuf |
| 输入说明 | name：mempool的名字  n mempool中的元素(rte\_mbuf)个数，最优化的应该是n=(2^q-1)  elt\_size：每一个元素(rte\_mbuf)的大小  cache\_size：优化选项  private\_data\_size：mempool结构的私有数据大小，可以使用sizeof(rte\_pktmbuf\_pool\_private)  mp\_init：可以使用rte\_pktmbuf\_pool\_init  mp\_init\_arg：可以使用NULL  obj\_init：可以使用rte\_pktmbuf\_init  obj\_init\_arg：可以使用NULL  socket\_id：NUMA体系的numa id，对于非NUMA体系，可以使用SOCKET\_ID\_ANY  flags：可以使用MEMPOOL\_F\_SP\_PUT|MEMPOOL\_F\_SC\_GET |
| 输出说明 | NA |
| 返回值说明 | 成功：返回mempool的地址  错误：返回NULL |
| 使用说明 | NA |
| 约束 | NA |

1. rte\_pktmbuf\_alloc

|  |  |
| --- | --- |
| 函数原型 | static struct rte\_mbuf\* rte\_pktmbuf\_alloc(struct rte\_mempool \*mp) |
| 函数功能 | 从mempool中申请一个rte\_mbuf |
| 输入说明 | mp mempool地址 |
| 输出说明 | NA |
| 返回值说明 | 成功：返回rte\_mbuf地址  失败：返回NULL |
| 使用说明 | 该rte\_mbuf的数据区现在没有任何内容，length设定为0 |
| 约束 | NA |

建议：在调用rte\_eal\_init和rte\_eal\_remote\_launch时，要留意与执行环境的CPU核相关，如果使用的参数不符合执行环境的CPU核条件，会造成程序异常。

更多关于DPDK接口使用说明文档，请参见DPDK的官方文档：

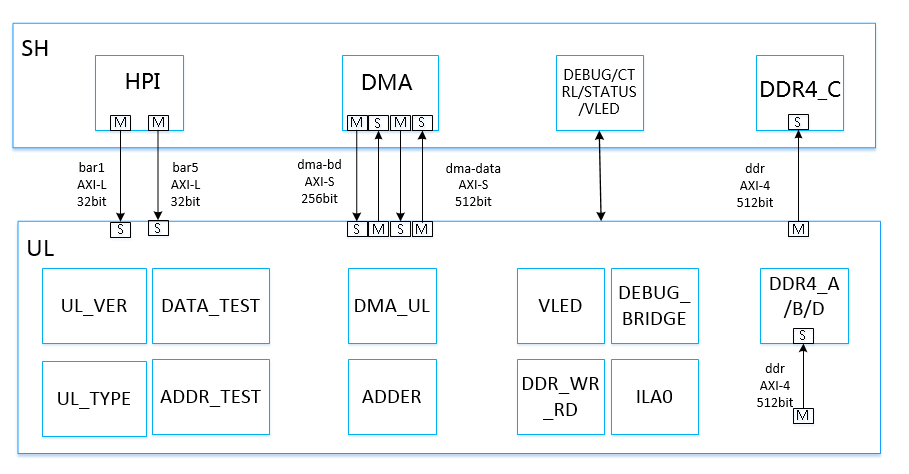
<http://www.dpdk.org/doc/archives>，请查看相对应的DPDK版本的文档。

## EXAMPLE2详细设计参考

### 概要

Example2主要实现ddr的读写访问和dma数据环回、加法器等操作，供用户参考所有涉及的接口和交互流程，使用户能快速的实现自己的加速设计。

### 特性介绍



UL\_VER例化ro\_reg\_inst CBB，版本号是示例一发布的时间信息。

UL\_TYPE例化ro\_reg\_inst CBB，类型是32'h00d10001。

DATA\_TEST例化ts\_reg\_inst CBB，实现对写入数据取反功能。

ADDR\_TEST例化ts\_addr\_reg\_inst CBB，实现对最近一次操作地址的取反功能。

DMA\_UL例化一个帧级fifo，实现host对用户逻辑的dma数据环回。

DDR\_WR\_RD实现对4组DDR的数据通道的读写访问功能。

VLED在PF下面访问，由静态提供管脚给动态用户使用，用户可以读写VLED，确保UL部分工作正常。VLED例化一组rw\_reg\_instCBB将输出结果链接到VLED。

ADDER是加法器模块，实现输入携带的加数和被加数的求和功能。

例化的DEBUG\_BRIDGE和ILA0，供用户调试使用。

### HDK构建说明

用户申请FPGA镜像后，登陆VM，HDK默认存放在/home/fpga\_design目录下。构建example2的步骤如下。

1. 设置EDA工具License。

用户打开“/home/fpga\_design/”路径下的“setup.cfg”文件，将文件中 XILINX\_LIC\_SETUP的值配置为:

华北：

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

华南：

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

华东：

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

image013

华为提供的Xilinx软件License仅限root账号使用。

1. 配置开发环境。

执行**/home/fpga\_design/setup.sh**完成硬件开发环境的配置。

$ cd /home/fpga\_design

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

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

1. 进入example2目录。

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

$ export UL2\_DIR=$(pwd)

1. 编译example2。

$ cd $UL2\_DIR/prj

$ sh ./build.sh

1. 仿真example2。

$ cd $UL2\_DIR/sim

$ make TC=sv\_demo\_001

### 术语介绍

FPGA HDK支持用户使用Verilog/VHDL语言进行FPGA开发，用户编写的源文件必须存放在用户工程目录<usr\_prj\_name>的src文件夹下。

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

### 设计约束

FPGA HDK提供了一套基于SystemVerilog-2012语法标准的通用FPGA仿真平台，支持业界主流仿真工具。为了帮助用户快速搭建自己的仿真平台，平台构架在设计之初考虑了业务解耦性设计，实现了Testbench与Testcase分离。Testbench位于“$HW\_FPGA\_DIR/hardware/vivado\_design/lib/sim”目录下。

## EXAMPLE3详细设计参考

### 概要

Example3的功能与Example2基本一致，也是实现数据的环回操作，区别是Example3的dma数据缓存在FPGA外挂的DDR中（Example2的dma数据缓存在FPGA内部的RAM中）。

### 特性介绍

说明：本节仅描述与Example2不同的地方，其他点请参考2.3章节。



为了实现将待加速数据写入与加速完成数据读出DDR操作，特对example3定义了hardacc域（参见2.4.3章节）。

四组DDR的地址由驱动控制，当有报文需要加速处理时，驱动会给待加速报文及加速完成的结果分别分配一个DDR地址，并将这两个DDR地址以hardacc格式缓存在BD中src\_addr起始的位置，并将待加速报文缓存在hardacc后面，供逻辑读取。

UL在收到BD之后，MMU\_UL模块会通过2.1.3章节描述的接口，发起读取hardacc以及待加速报文命令，待所读hardacc及待加速报文返回给MMU\_UL后，MMU\_UL模块根据hardacc中的ddr\_saddr将待加速报文写入DDR，之后MMU\_UL通知KERNEL\_UL加速报文，待KERNEL\_UL加速完成后将加速结果写入hardacc中的ddr\_daddr所指的DDR中，之后KERNEL\_UL通知MMU\_UL将加速结果读出，并返回给SH逻辑。

完整的Example3处理流程如下所示：



1. App申请mbuf给BD，src data, dst data；并构造BD写入驱动发送队列；
2. 驱动将BD信息通过门铃方式写入shell静态逻辑；
3. Shell静态逻辑获取并解析BD,通过AXI\_S接口传给UL的mmu\_tx模块；
4. UL的mmu\_tx模块从BD中提取opcode和线程号，判断操作类型；根据从BD中解析出的cpu\_src\_addr获取hardacc；
5. mmu\_tx模块解析到操作类型，根据操作类型做相应动作：

1）写操作：根据从BD中解析出的cpu\_src\_addr读hardacc 来获取fpga DDR源地址；读完hardacc后继续读取cpu内存数据，并将读取的内存数据写入fpga\_src\_ddr开始的DDR中，在写入最后一个分片到DDR时，通知mmu\_rx模块已经写入到DDR成功；

2）读操作：根据从BD中解析出的cpu\_src\_addr读hardacc来获取fpga DDR源地址；读完hardacc后，通知mmu\_rx模块已经需要从DDR的fpga\_dst\_ddr读取数据；

3）环回处理操作时，从BD中解析出的cpu\_src\_addr读hardacc 来获取fpga DDR源地址；读完hardacc后继续读取cpu内存数据，并将读取的内存数据写入fpga\_src\_ddr开始的DDR中，在写入最后一个分片到DDR时，通知kernel模块已经写入到DDR成功；

1. 待环回处理写DDR结束后，kernel从fpga\_src\_ddr开始的DDR中将数据搬回，并做相应处理，将处理完的数据，存入FPGA\_des\_ddr开始的DDR中，并在写DDR操作完成后构造信息通知mmu\_rx模块，通知信息包含kernel处理完成的数据存入DDR的起始位置，即FPGA\_des\_addr，和长度等信息；
2. mmu\_rx模块根据操作类型做相应处理：

1）写操作：构造写操作成功标识和hardacc信息（fpga\_src\_addr/fpga\_des\_addr/length/opcode/thread\_id）并送给shell的RX模块；

2)读操作：从DDR的fpga\_src\_ddr读取数据，并送给并送给shell的RX模块；在数据尾拍构造读数据完成标识和hardacc信息（fpga\_src\_addr/fpga\_des\_addr/length/opcode/thread\_id）送给shell的RX模块；

3）环回处理操作时，从DDR的fpga\_des\_ddr起始地址读取数据，并送给shell的RX模块；在数据尾拍构造读数据完成标识和hardacc信息（fpga\_src\_addr/fpga\_des\_addr/length/opcode/thread\_id）送给shell的RX模块；

1. shell逻辑RX模块识别数据是否是带有完成标识的尾片，如果是尾片则构造响应BD并发送到驱动软件的接收队列；
2. 驱动软件收到数据和响应BD后写入app；
3. App释放BD mbuf和数据mbuf；

### hardacc说明

* **功能介绍**

长度为32Byte，用来缓存硬件逻辑DDR的物理地址，由DPDK驱动填写在src\_addr起始的内存中，由UL在读取报文前首先读取该域。

hardacc域为专门针对example3的自定义格式，用户可对此域进行修改（同时需要对example3进行同步修改）。

UL获取的hardacc数据格式如下：

| 名称 | bit 位置 | 描述 |
| --- | --- | --- |
| rsv | [255:100] | 保留域 |
| ddr\_daddr | [99:64] | 加速完后解析结果写入DDR起始地址 |
| rsv | [63:36] | 保留域 |
| ddr\_saddr | [35:0] | 待加速数据写入DDR起始地址 |

UL返回给SHELL的hardacc数据格式如下：

| 名称 | bit 位置 | 描述 |
| --- | --- | --- |
| rsv | [255:176] | 保留域 |
| length | [175:144] | SHELL2UL的BD所带需要UL处理的数据总长度 |
| rsv | [143:138] | 保留域 |
| thread\_id | [137:130] | 线程id号 |
| opcode | [129:128] | 2'd1:只读操作 2'd2:只写操作  2'd0，2'd3:环回处理操作 |
| rsv | [127:100] | 保留域 |
| ddr\_daddr | [99:64] | 加速完后解析结果写入DDR起始地址 |
| rsv | [63:36] | 保留域 |
| ddr\_saddr | [35:0] | 待加速数据写入DDR起始地址 |

### HDK构建说明

用户申请FPGA镜像后，登陆VM，HDK默认存放在/home/fpga\_design目录下。构建example3的步骤如下。

1. 设置EDA工具License。

用户打开“/home/fpga\_design/”路径下的“setup.cfg”文件，将文件中 XILINX\_LIC\_SETUP的值配置为:

华北：

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

华南：

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

华东：

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

image013

华为提供的Xilinx软件License仅限root账号使用。

1. 配置开发环境。

执行**/home/fpga\_design/setup.sh**完成硬件开发环境的配置。

$ cd /home/fpga\_design

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

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

1. 进入example3目录。

$ cd $HW\_FPGA\_DIR/hardware/vivado\_design/examples/example3

$ export UL3\_DIR=$(pwd)

1. 编译example3。

$ cd $UL3\_DIR/prj

$ sh ./build.sh

1. 仿真example3。

$ cd $UL3\_DIR/sim

$ make TC=sv\_demo\_001

### SDK API列表

|  |  |  |
| --- | --- | --- |
| 初始化、管理、配置接口 | fddr\_access\_mode\_init | 资源初始化接口 |
| 线程id申请/释放接口 | alloc\_thread\_id | 申请线程id |
| free\_thread\_id | 释放线程id |
| 申请/释放内存接口 | memory\_manager\_alloc\_bulk | 内存申请接口 |
| memory\_manager\_free\_bulk | 内存释放接口 |
| 读、写、环回DDR内存接口 | read\_data\_from\_fddr | 从fpga ddr读取数据 |
| write\_data\_to\_fddr | 写数据到fpga ddr |
| process\_data\_with\_fpga | 环回模式 |
| 读写寄存器接口 | read\_register | 读取寄存器 |
| write\_register | 写入寄存器 |

### SDK API详细说明

1. fddr\_access\_mode\_init

|  |  |
| --- | --- |
| 函数原型 | int fddr\_access\_mode\_init(Callbackfunc callback) |
| 函数功能 | 资源初始化 |
| 输入说明 | callback：结果回调函数 |
| 输出说明 | NA |
| 返回值说明 | On success=0  On failure<0 |
| 使用说明 | 该函数在每个线程中均要被调用，但只生效一次。 |
| 约束 | NA |

1. alloc\_thread\_id

|  |  |
| --- | --- |
| 函数原型 | int alloc\_thread\_id(unsigned int \*thread\_id) |
| 函数功能 | 申请线程id |
| 输入说明 | NA |
| 输出说明 | thread\_id：线程id |
| 返回值说明 | On success=0  On failure<0 |
| 使用说明 | 每个线程申请一个唯一的id，下发读、写、环回DDR内存业务时作为参数传入。 |
| 约束 | 每个线程申请一次，使用完成后需要主动释放。 |

1. free\_thread\_id

|  |  |
| --- | --- |
| 函数原型 | int free\_thread\_id(unsigned int thread\_id) |
| 函数功能 | 释放线程id |
| 输入说明 | thread\_id：线程id |
| 输出说明 | NA |
| 返回值说明 | On success=0  On failure<0 |
| 使用说明 | 线程退出之前进行调用，释放申请的线程id资源。 |
| 约束 | NA |

1. memory\_manager\_alloc\_bulk

|  |  |
| --- | --- |
| 函数原型 | void\* memory\_manager\_alloc\_bulk(unsigned int buff\_size) |
| 函数功能 | 申请内存 |
| 输入说明 | buff\_size：内存大小 |
| 输出说明 | NA |
| 返回值说明 | On success， 返回申请到的内存虚拟地址  On failure，返回NULL |
| 使用说明 | NA |
| 约束 | 需要根据现有内存池规格进行内存申请。 |

1. memory\_manager\_free\_bulk

|  |  |
| --- | --- |
| 函数原型 | int memory\_manager\_free\_bulk( void\* buff\_vaddr) |
| 函数功能 | 释放内存 |
| 输入说明 | buff\_vaddr：需要释放的内存虚拟地址 |
| 输出说明 | NA |
| 返回值说明 | On success=0  On failure<0 |
| 使用说明 | NA |
| 约束 | NA |

1. read\_data\_from\_fddr

|  |  |
| --- | --- |
| 函数原型 | int read\_data\_from\_fddr (unsigned int thread\_id, unsigned int slot\_id, rw\_ddr\_data rw\_data) |
| 函数功能 | 从FPGA DDR读取数据 |
| 输入说明 | thread\_id：线程id  slot\_id：槽位id  rw\_data：读写配置数据 |
| 输出说明 | NA |
| 返回值说明 | On success=0  On failure<0 |
| 使用说明 | NA |
| 约束 | NA |

1. write\_data\_to\_fddr

|  |  |
| --- | --- |
| 函数原型 | int write\_data\_to\_fddr(unsigned int thread\_id, unsigned int slot\_id, rw\_ddr\_data rw\_data) |
| 函数功能 | 写入FPGA DDR数据 |
| 输入说明 | thread\_id：线程id  slot\_id：槽位id  rw\_data：读写配置数据 |
| 输出说明 | NA |
| 返回值说明 | On success=0  On failure<0 |
| 使用说明 | NA |
| 约束 | NA |

1. process\_data\_with\_fpga

|  |  |
| --- | --- |
| 函数原型 | int process\_data\_with\_fpga(unsigned int thread\_id, unsigned int slot\_id, rw\_ddr\_data rw\_data) |
| 函数功能 | 环回模式 |
| 输入说明 | thread\_id：线程id  slot\_id：槽位id  rw\_data：读写配置数据 |
| 输出说明 | NA |
| 返回值说明 | On success=0  On failure<0 |
| 使用说明 | NA |
| 约束 | NA |

1. read\_register

|  |  |
| --- | --- |
| 函数原型 | int read\_register(unsigned int slot\_id, unsigned int addr, unsigned int \*value) |
| 函数功能 | 读取寄存器 |
| 输入说明 | slot\_id：槽位id  addr：寄存器地址 |
| 输出说明 | value：读取的结果数据 |
| 返回值说明 | On success=0  On failure<0 |
| 使用说明 | NA |
| 约束 | NA |

1. write\_register

|  |  |
| --- | --- |
| 函数原型 | int write\_register(unsigned int slot\_id ,unsigned int addr, unsigned int value) |
| 函数功能 | 读取寄存器 |
| 输入说明 | slot\_id：槽位id  addr：寄存器地址  value：需要写入的数据 |
| 输出说明 | NA |
| 返回值说明 | On success=0  On failure<0 |
| 使用说明 | NA |
| 约束 | NA |

### 术语介绍

FPGA HDK支持用户使用Verilog/VHDL语言进行FPGA开发，用户编写的源文件必须存放在用户工程目录<usr\_prj\_name>的src文件夹下。

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

# FACS通用型(SDAccel)接口说明

## FACS SDAccel概要说明

FACS SDAccel是以Xilinx SDAccel 2017.1版本为原型，支持用户在虚机上进行基于OpenCL的开发和应用，且在开发及应用方式上兼容Xilinx SDAccel 2017.1版本。本章节将简要介绍FACS的SDAccel方案，以及针对用户如何开发Host、Kenel做简单说明。

## 术语介绍

HAL：Hardware Abstraction Layer，硬件抽象层

XDMA：Xilinx Direct Memory Access，Xilinx直接存储器访问

Mgmt：Management

Drv：Driver，驱动

PF：Physical Function

OpenCL：Open Computing Language，开放运算语言

## FACS SDAccel方案说明

支持将SDAccel方案部署在虚机内，用户在虚机运行OpenCL Host/kernel code的开发、调试及应用。基于用户安全使用的考虑，FPGA的管理由Host来完成，因此用户在虚机内发起的部分Kernel的管理操作实际是在Host上完成，例如：xclbin加载。



FACS SDAccel方案框图

FACS SDAccel方案说明：

1. OpenCL Runtime：Xilinx OpenCL Runtime，对用户呈现OpenCL API
2. HAL：实现OpenCL Runtime和Kernel Driver的适配及Global Memory的地址管理
3. XDMA Drv：Xilinx DMA内核驱动
4. Fpga tool：提供基本的fpga管理工具，例如：fpga kernel加载命令，加载状态查询
5. Host Mgmt Drv：运行在Host的管理驱动，实现fpga kernel的加载
6. User PF：用户面PF接口，直通到虚机，为用户提供FPGA访问通道
7. Mgmt. PF：管理面PF接口，为主机访问FPGA的通道
8. Based Resgion：FPGA静态逻辑区域
9. Expanded Region：FPGA动态逻辑区域

由于User PF和Mgmt PF分别绑定到用户虚机和服务器主机，因此用户在虚机上直接执行管理相关的命令会受到限制：首先，FACS SDAccel在虚机上提供其管理工具，fpga\_tool，用户在虚机时，需要使用fpga\_tool工具进行管理相关的操作；其次，相较于在xilinx xbsak工具而言，fpga\_tool工具支持的命令有限，暂时还未支持所有的xbsak命令。因此HAL层中一些API用户将无法在虚机使用，FACS SDAccel目前支持的HAL的API如下：

|  |  |  |
| --- | --- | --- |
| **序号** | **API名称** | **备注** |
| 1 | xclOpen( ) | - |
| 2 | xclClose( ) | - |
| 3 | xclGetDeviceInfo2( ) | 支持，但有修改 |
| 4 | xclGetAXIErrorStatus( ) | - |
| 5 | xclLoadXclBin( ) | 支持，但有修改 |
| 6 | xclAllocDeviceBuffer( ) | - |
| 7 | xclAllocDeviceBuffer2( ) | - |
| 8 | xclFreeDeviceBuffer( ) | - |
| 9 | xclCopyBufferHost2Device( ) | - |
| 10 | xclCopyBufferDevice2Host( ) | - |
| 11 | xclWrite( ) | - |
| 12 | xclRead( ) | - |
| 13 | xclReClock2( ) | 支持，但有修改 |
| 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开发说明

### Host Code开发

完整支持Xilinx SDAccel 2017.1所支持的OpenCL API，具体的支持API见xilinx UG1023。用户可以基于xilinx SDx 2017.1版本支持的Host code开发。

UG1023链接如下。<https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_1/ug1023-sdaccel-user-guide.pdf>

### Kernel Code开发

完整支持Xilinx SDAccel 2017.1 OpenCL C的语法，具体的支持见UG1023、UG1207。用户基于xilinx SDx 2017.1版本支持的语法进行Kernel Code的开发。

UG1207链接如下。<https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_1/ug1207-sdaccel-optimization-guide.pdf>

## Example设计说明

FACS SDAccel提供了三个example，分别从三种不同的Kernel开发方式（即，RTL开发kernel，HLS开发kernel和OpenCL C开发kernel）上进行开发调试过程的说明。每个example分为两个部分：

1. kernel的开发及仿真在/home/fpga\_design/hardware/sdaccel\_design/examples目录下，并提供了对应的执行脚本。
2. Host code的开发及上板调试放在/home/fpga\_design/software/app/sdaccel\_app目录下，并提供了对应的执行脚本。

* example vadd\_cl

该example示例了以OpenCL C进行开发kernel的流程，其kernel源码来自于Xilinx SDx安装工具的<SDx\_install\_area>SDx/2017.1/examples/vadd。同时该example提供了kernel编译，仿真的脚本，如果用户选择OpenCL C的开发方式，用户需要替换掉kernel源代码，然后执行example提供的脚本进行编译及仿真调试或者上板调试。

* example mmult\_hls

该example是示例了以HLS进行开发kernel的流程，其kernel源码来自于Xilinx SDx安装工具的<SDx\_install\_area>SDx/2017.1/ samples/mmult。同时该example提供了kernel编译，仿真的脚本，如果用户选择HLS的开发方式，用户需要替换掉kernel源代码，然后执行example提供的脚本进行编译及仿真调试或者上板调试。

* example vadd\_rtl

该example示例了以RTL进行开发kernel的流程，其kernel源码来自于Xilinx github开源例子，链接为：<https://github.com/Xilinx/SDAccel_Examples/tree/master/getting_started/rtl_kernel>，该example提供了kernel编译，仿真的脚本，当用户选择RTL的开发方式，用户需要RTL kernel源代码及xml说明文件，然后执行example提供的脚本进行编译及仿真调试或者上板调试。

关于SDAccel的开发调试过程，用户也可以参见xilinx tutorial文档，链接如下。

<https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_1/ug1021-sdaccel-intro-tutorial.pdf>