

# Multi-Channel PCIe QDMA Subsystem

## User Guide

**V1.2**

July 31, 2022

### Revision History

The following table shows the revision history for this document.

Data	Version	Revision
2021.10.23	1.0	Initial Release
2022.7.5	1.1	增加C2H和H2C DMA的逻辑复位
2022.7.31	1.2	增加Display的FPS Timing控制和寄存器

## 目 录

1	介绍.....	1
1.1	特性.....	1
1.2	应用.....	1
2	概述.....	2
2.1	特性概要.....	3
3	产品规格.....	3
3.1	性能.....	4
3.2	资源.....	4
3.3	内核 COMPONENT .....	5
3.3.1	<i>Target Bridge</i> .....	5
3.3.2	<i>H2C 通道</i> .....	6
3.3.3	<i>C2H 通道</i> .....	6
3.3.4	<i>AXI4-Lite Master</i> .....	6
3.3.5	<i>IRQ Module</i> .....	6
3.3.6	<i>DMA 操作</i> .....	6
3.4	端口描述.....	7
3.5	寄存器空间.....	12
3.5.1	<i>PCIe to AXI4-Lite Master (BAR1) 地址映射</i> .....	12
3.5.2	<i>PCIe to DMA (BAR0) 地址映射</i> .....	12
3.6	DMA FLOW .....	22
3.6.1	<i>Global Reset</i> .....	22
3.6.2	<i>CH#i H2C Flow for SGDMA</i> .....	23
3.6.3	<i>CH#i C2H Flow for SGDMA</i> .....	24
3.6.4	<i>CH#i H2C Flow for CDMA</i> .....	24
3.6.5	<i>CH#i C2H Flow for CDMA</i> .....	25
3.7	设备驱动.....	26
3.7.1	<i>Windows WDF (Queue or Non-Queue)</i> .....	26

3.7.2	<i>Linux (Queue or Non-Queue)</i> .....	26
3.7.3	<i>V4L2</i> .....	26

# 1 介绍

基于 PCI Express Integrated Block, Multi-Channel PCIe QDMA Subsystem 实现了使用 DMA 地址队列的独立多通道、高性能 Continuous 或 Scatter Gather DMA, 提供 FIFO/AXI4-Stream 用户接口。

## 1.1 特性

- 支持 Ultrascale+, Ultrascale, 7 Series 的 PCI Express Integrated Block
- 支持 64, 128, 256, 512-bit 数据路径
- 64-bit 源地址, 目的地址, 和描述符地址
- 多达 16 个独立的 host-to-card (H2C/Read) 数据通道或 H2C DMA
- 多达 16 个独立的 card-to-host (C2H/Write) 数据通道或 C2H DMA
- AXI4-Stream/FIFO 用户接口 (每个通道都有自己的 AXI4-Stream/FIFO 接口)
  
- 每个 DMA 引擎支持 DMA 地址队列, 队列深度可达 32
- AXI4-Lite Master 接口允许 PCIe 通信绕过 DMA 引擎
- Scatter Gather 描述符列表支持无限列表大小
- 每个描述符的最大传输长度为 4GB
- MSI 中断
- 连续描述符的块获取
- 中断或查询模式

## 1.2 应用

本内核体系结构支持广泛的计算和通信目标程序应用, 强调性能、成本、可扩展性、功能可扩展性和关键任务可靠性。典型应用包括:

- 数据通信网络
- 电信网络
- 宽带有线和无线应用
- 网络接口卡
- 用于各种应用程序的服务器 add-in card

典型应用如下图所示:

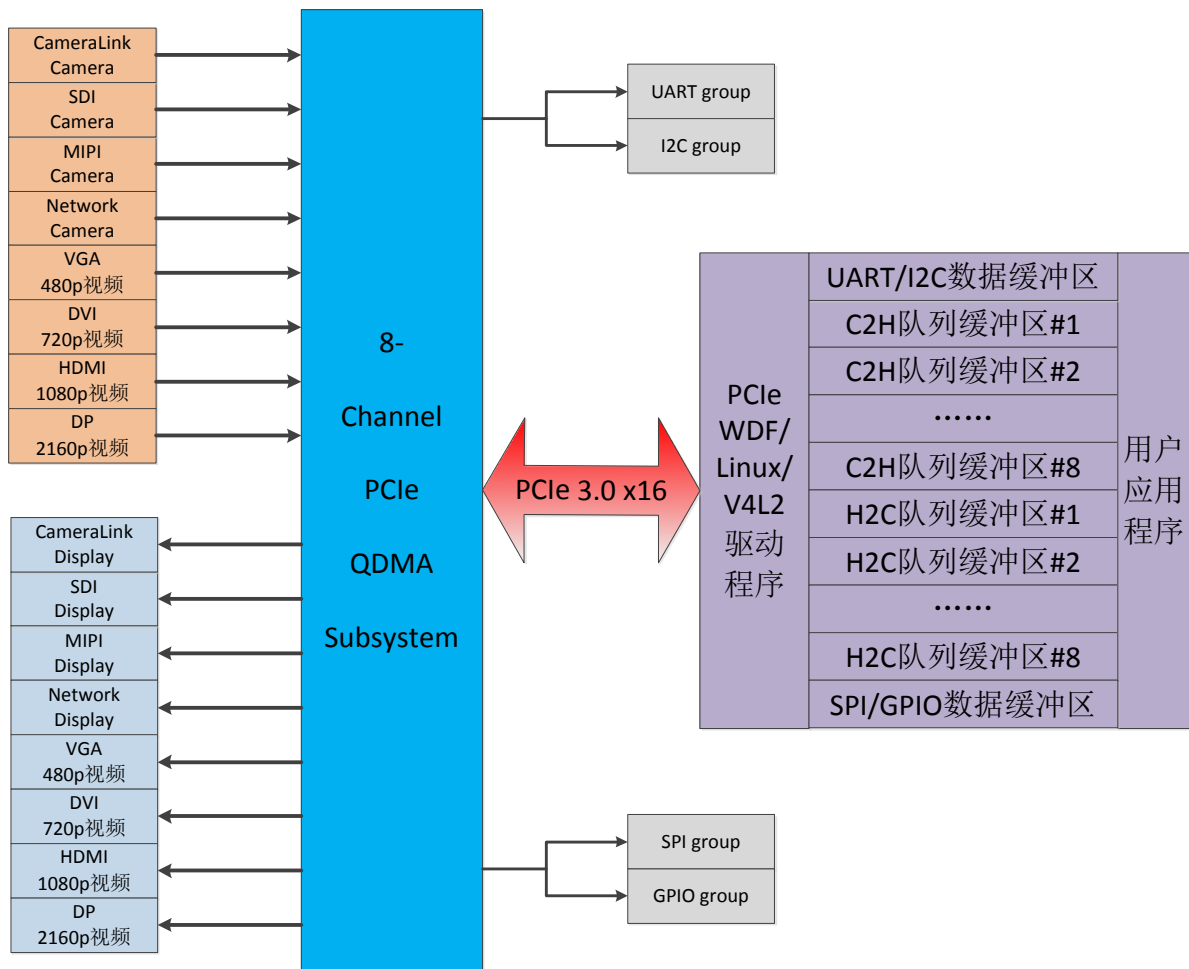


图 1 Multi-Channel PCIe QDMA Subsystem 典型应用：8 通道视频采集和视频显示

## 2 概述

Multi-Channel PCIe QDMA Subsystem 作为一个高性能 DMA 数据搬运器，内核通过 AXI4-Stream/FIFO 接口直接连接 RTL 逻辑。使用提供的字符驱动程序，AXI4-Stream/FIFO 接口可用于 PCIe 地址空间和 AXI 地址空间之间的高性能数据搬运。除了基本的 DMA 功能，DMA 支持多达 16 个独立的 upstream 和 downstream 通道，每个通道支持深度为 32 的 DMA 地址队列，另外还允许 PCIe 通信绕过 DMA 引擎。

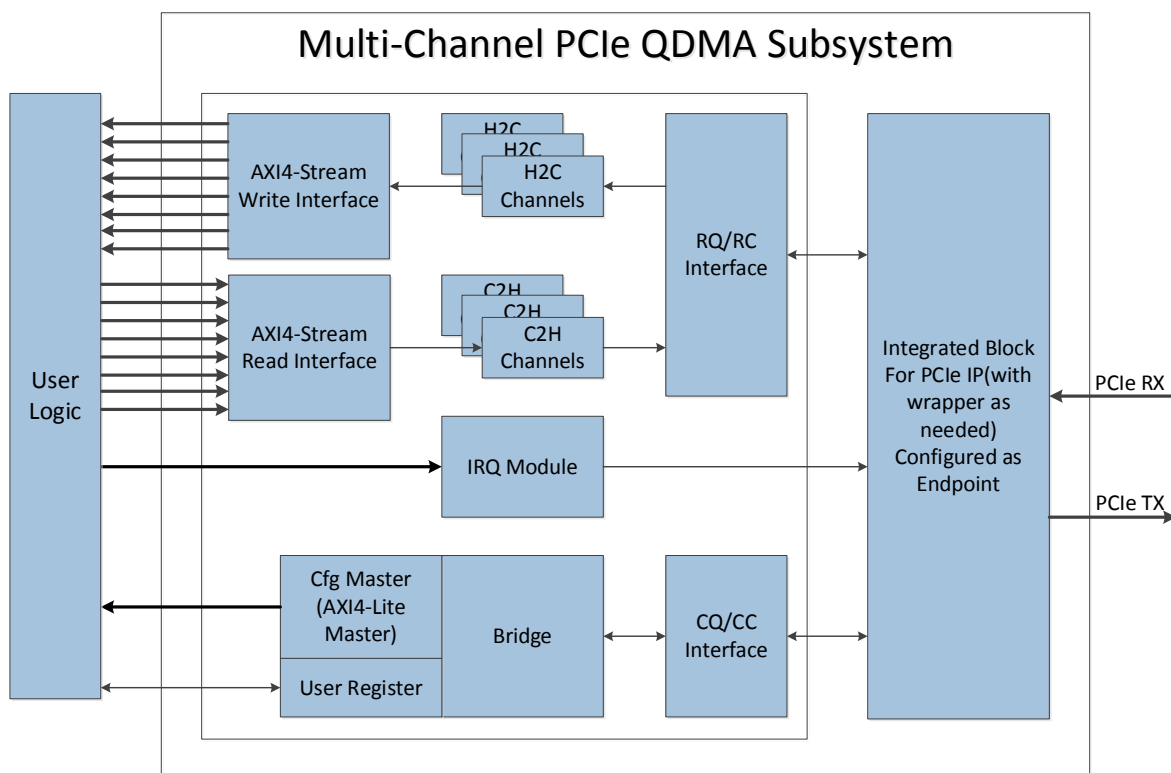


图 2 Multi-Channel PCIe QDMA Subsystem 概述

## 2.1 特性概要

基于描述符提供的信息：源地址，目的地址和传输数据长度，Multi-Channel PCIe QDMA Subsystem 实现 Host 存储器和 PCIe DMA 子系统之间的数据搬移。这些 DMA 可以同时是 Host to Card (H2C) 和 Card to Host (C2H) 传输。每个 DMA 通道对应各自的 AX4-Stream/FIFO 接口，DMA 从 Host 存储器获取并解析描述符链表，基于描述符链表信息完成自己通道的数据传输，然后使用 MSI 中断发出描述符完成或错误的信令。内核也提供多达 16 个输出到 Host 的用户中断信号。

主机可以通过以下 2 个接口访问用户逻辑：

- AXI4-Lite Master 配置接口：这个接口是一个固定的 32-bit 端口，用于对性能要求不高的用户配置和状态寄存器的访问
- User Register：这个接口是多个 32-bit 向量信号和 1-bit 信号，这些信号来自对应 DMA 通道数据搬移过程中产生的控制或状态信号

## 3 产品规格

结合 Integrated Block for PCI Express IP, Multi-Channel PCIe QDMA Subsystem 为 PCIe 提供了一个高性能的 DMA 解决方案。

## 3.1 性能

Endpoint 配置参数: Max Payload Size=256-byte, Max Read Request Size=512-byte

8-Channel PCIe-SGQDMA Subsystem, DMA Transfer Length = 4MB

表 1 PCIe 3.0 x16 C2H DMA 速率

	DMA0	DMA1	DMA2	DMA3	DMA4	DMA5	DMA6	DMA7
速率	1660MB/s	1660MB/s	1660MB/s	1660MB/s	1660MB/s	1660MB/s	1660MB/s	1660MB/s

表 2 PCIe 3.0 x16 H2C DMA 速率

	DMA0	DMA1	DMA2	DMA3	DMA4	DMA5	DMA6	DMA7
速率	1670MB/s	1670MB/s	1670MB/s	1670MB/s	1670MB/s	1670MB/s	1670MB/s	1670MB/s

表 3 PCIe 3.0 x8 C2H DMA 速率

	DMA0	DMA1	DMA2	DMA3	DMA4	DMA5	DMA6	DMA7
速率	880MB/s	880MB/s	880MB/s	880MB/s	880MB/s	880MB/s	880MB/s	880MB/s

表 4 PCIe 3.0 x8 H2C DMA 速率

	DMA0	DMA1	DMA2	DMA3	DMA4	DMA5	DMA6	DMA7
速率	890MB/s	890MB/s	890MB/s	890MB/s	890MB/s	890MB/s	890MB/s	890MB/s

表 5 PCIe 2.0 x8 C2H DMA 速率

	DMA0	DMA1	DMA2	DMA3	DMA4	DMA5	DMA6	DMA7
速率	450MB/s	450MB/s	450MB/s	450MB/s	450MB/s	450MB/s	450MB/s	450MB/s

表 6 PCIe 2.0 x8 H2C DMA 速率

	DMA0	DMA1	DMA2	DMA3	DMA4	DMA5	DMA6	DMA7
速率	455MB/s	455MB/s	455MB/s	455MB/s	455MB/s	455MB/s	455MB/s	455MB/s

## 3.2 资源

8-Channel PCIe-SGQDMA Subsystem

表 7 PCIe 3.0 x16 DMA Subsystem 资源

	LUTs	FFs	BRAMs	PCIe
资源	46985	101938	150	1

表 8 PCIe 3.0 x8 DMA Subsystem 资源

	LUTs	FFs	BRAMs	PCIe

资源	26388	51935	78	1
----	-------	-------	----	---

表 9 PCIe 2.0 x8 DMA Subsystem 资源

	LUTs	FFs	BRAMs	PCIe
资源	26945	38687	55	1

### 8-Channel PCIe-CQDMA Subsystem

表 10 PCIe 3.0 x16 DMA Subsystem 资源

	LUTs	FFs	BRAMs	PCIe
资源	34976	75994	150	1

表 11 PCIe 3.0 x8 DMA Subsystem 资源

	LUTs	FFs	BRAMs	PCIe
资源	19364	37487	78	1

表 12 PCIe 2.0 x8 DMA Subsystem 资源

	LUTs	FFs	BRAMs	PCIe
资源	20973	29963	55	1

## 3.3 内核 Component

内核内部实现多达 32 个独立的物理 DMA 引擎（多达 16 个 C2H 和 16 个 H2C）。这些 DMA 引擎被映射到各自的 AXI4-Stream/FIFO 用户应用接口。AXI4-Stream/FIFO 接口只传递数据。

通道的类型配置决定了在哪个总线上进行事务传输。

- Host-to-Card (H2C) 通道向 PCIe 产生读请求，然后给用户应用提供数据
- 同样，Card-to-Host (C2H) 通道等待用户侧的数据，然后向 PCIe 产生写请求（包含接收到的数据）

Host 通过 AXI4-Lite Master 接口访问用户逻辑的配置和状态寄存器。这些请求是 32-bit 的读或写。

### 3.3.1 Target Bridge

Target bridge 从 Host 接收请求。基于 BARs，这些请求通过 AXI4-Lite Master 接口传递到内部的目标用户。下行用户逻辑返回 non-post 请求的数据后，target bridge



产生一个读返回 TLP，然后通过 CC 总线发送给 PCIe IP。

PCIe BARs 配置如下表所示。

表 13 32-bit BARs

BAR0(32-bit)	BAR1(32-bit)
DMA	PCIe to AXI4-Lite Master

### 3.3.2 H2C 通道

H2C 通道处理 host-to-card 的 DMA 传输。根据最大读请求大小和可用的内部资源，H2C 通道负责拆分这些读请求。读取请求的每个拆分（如果有）都会消耗一条额外的读取请求条目。DMA 通道向 PCIe RQ 模块发出读请求后，直到按顺序接收到用户接口的写完成确认，这个读请求才算完成。在数据传输完成后，DMA 通道向 Host 发出一个中断。

H2C 通道会根据配置的最大读请求大小和数据 FIFO 可用空间来拆分 Host 读接口上的数据传输。从 PCIe RC 模块输出的读请求返回完成数据包会存储到已分配的数据 FIFO 中。为了减少数据传输延时，一旦接收到任何一个完成数据包，H2C 通道就开始把读取的数据输出到 AXI4-Stream/FIFO Write 接口。

### 3.3.3 C2H 通道

C2H 通道处理 card-to-host 的 DMA 传输。在 AXI4-Stream/FIFO Read 接口接收数据之前，C2H 通道首先接收 DMA 描述符，建立 DMA 传输的环境，然后在准备好请求 ID 和启用 C2H 通道后，通道的 AXI4-Stream/FIFO 接口可以接收数据并对主机执行 DMA。在数据传输完成后，DMA 通道向 Host 发出一个中断。

### 3.3.4 AXI4-Lite Master

Host 使用这个接口向用户逻辑发出 32-bit 读和 32-bit 写请求。通过 PCIe 总线接收读或者写请求，路由到 AXI4-Lite Master BAR 中，target bridge 通过 PCIe IP CC 总线返回读取的完成数据。

### 3.3.5 IRQ Module

IRQ module 接收来自用户逻辑中断请求和每个 DMA 通道的独立中断线。这个模块通过 PCIe 向 Host 产生 MSI 中断。

### 3.3.6 DMA 操作

从 DMA 的原理出发，PCIe DMA 引擎通常在 Host 内存和驻留在 FPGA 中的内存之间搬移数据，FPGA 的内存通常（但不总是）位于 add-in 板卡。当数据从 Host

内存搬移到 FPGA 内存时，称为 Host to Card (H2C) 传输。相反，当数据从 FPGA 内存搬移到 Host 内存，称为 Card to Host (C2H) 传输。

在通常的操作中，Host 中的应用程序必须在 FPGA 和 Host 内存之间搬移数据。为了完成此次传输，Host 在系统内存中设置缓冲区空间，并创建 DMA 引擎用于搬移数据的描述符。

Multi-Channel PCIe QDMA Subsystem 使用连续的描述符链表，单个描述符指定了地址和 DMA 传输长度。Host 驱动创建描述符链表，并存储在 Host 内存。Host 驱动只需要少量控制寄存器就可以初始化 DMA 通道，接着该 DMA 通道就开始去获取描述符链表并执行 DMA 操作。

描述符描述了 Multi-Channel PCIe QDMA Subsystem 执行的内存传输。每个 DMA 通道有自己的描述符链表。Host 驱动通过硬件寄存器初始化每个 DMA 通道描述符链表的起始地址。在启动 DMA 通道后，该 DMA 通道开始从起始地址获取描述符。

EOP 控制比特显示描述符链表的终止。当 DMA 通道检测到某个描述符的 EOP 控制比特后，会停止获取该描述符链表。EOP 控制比特只能在描述符链表的最后一个描述符中被设置。

描述符格式如下表所示。

表 14 描述符格式

Offset	Fields
0x0	Bit 31: EOP Bit 30~16: 保留 Bit 15~0: Magic, 常数 16'hAD4B
0x4	DMA Transfer Length
0x8	Address[31:0]
0xC	Address[63:32]

Address: 64-bit, Destination address for C2H 或 Source address for H2C

EOP: 1-bit, End of packet for stream data

DMA Transfer Length: 32-bit, Length of data in bytes

### 3.4 端口描述

Multi-Channel PCIe QDMA Subsystem 直接和 integrated block for PCIe 连接。和 PCIe integrated block IP 数据路径接口的宽度是 64, 128, 256 或 512-bit, 时钟频率最高可达 250MHz。除了 AXI4-Lite Master 接口, 数据路径宽度适用于所有数据接口。AXI4-Lite Master 接口的宽度固定为 32-bit。

以下列表描述了这个 IP 的端口 (默认数据路径接口宽度是 512-bit, PCIe 接口是 x16)。

表 15 参数定义

参数名称	描述	默认值
CNUM	H2C 和 C2H 的通道数量	8
C2H_BUF_BRAM_CASCADE_DEPTH	C2H 数据缓冲区的 BRAM 级联深度 0: BRAM 深度= $2^9=512$ 1: BRAM 深度= $2^{9+1}=1024$ ..... N: BRAM 深度= $2^{9+N}$	0
C2H_BUF_USE_URAM	C2H 数据缓冲区是否使用 URAM 0: 不使用 URAM 1: 使用 URAM	0
C2H_BUF_URAM_CASCADE_DEPTH	H2C 数据缓冲区的 URAM 级联深度 0: URAM 深度= $2^{12}=4096$ 1: URAM 深度= $2^{12+1}=8192$ ..... N: URAM 深度= $2^{12+N}$	0
H2C_BUF_BRAM_CASCADE_DEPTH	H2C 数据缓冲区的 BRAM 级联深度 0: BRAM 深度= $2^9=512$ 1: BRAM 深度= $2^{9+1}=1024$ ..... N: BRAM 深度= $2^{9+N}$	0
H2C_BUF_USE_URAM	H2C 数据缓冲区是否使用 URAM 0: 不使用 URAM	0

	1: 使用 URAM	
H2C_BUF_URAM_CASCADE_DEPTH	C2H 数据缓冲区的 URAM 级联深度 0: URAM 深度= $2^{12}=4096$ 1: URAM 深度= $2^{12+1}=8192$ ..... N: URAM 深度= $2^{12+N}$	0

表 16 顶层接口信号

信号名称	输入输出	描述
pcie_trn_clk	输出	PCI Express Transaction 接口时钟
pcie_trn_reset_n	输出	PCI Express Transaction 复位, 低有效
trn_lnk_up	输出	PCI Express Transaction Link Up 信号, 高有效

表 17 PCIe 接口信号

信号名称	输入输出	描述
pcie_refclk_p	输入	PCI Express 接口参考时钟+
pcie_refclk_n	输入	PCI Express 接口参考时钟-
pcie_perst_n	输入	PCI Express 接口基本复位, 低有效
pci_exp_txp[15:0]	输出	PCI Express 串行差分输出+, 16 通道
pci_exp_txn[15:0]	输出	PCI Express 串行差分输出-, 16 通道
pci_exp_rxp[15:0]	输入	PCI Express 串行差分输入+, 16 通道
pci_exp_rxn[15:0]	输入	PCI Express 串行差分输入-, 16 通道

表 18 H2C 通道 0-CNUM-1 FIFO (FWFT) 接口信号

信号名称	输入输出	描述
fifo_rdclk_disp [CNUM-1:0]	输入	FIFO 读时钟 Bit i, 表示 H2C 通道 i 的读时钟
fifo_rdrstn_disp [CNUM-1:0]	输入	FIFO 读复位, 低有效 Bit i, 表示 H2C 通道 i 的读复位
fifo_rdreq_disp [CNUM-1:0]	输入	FIFO 读使能, 高有效 Bit i, 表示 H2C 通道 i 的读使能
fifo_q_disp [512*CNUM-1:0]	输出	FIFO 读数据 Bit 512*(i+1)-1~512*i, 表示 H2C 通道 i 的读数据
fifo_empty_disp [CNUM-1:0]	输出	FIFO 空, 高有效 Bit i, 表示 H2C 通道 i 的缓存空信号
fifo_prog_empty_disp [CNUM-1:0]	输出	FIFO 可编程空 (阈值等于 16), 高有效 Bit i, 表示 H2C 通道 i 的缓存可编程空信号

表 19 C2H 通道 0-CNUM-1 FIFO 接口信号

信号名称	输入输出	描述
fifo_wrclk_acq [CNUM-1:0]	输入	FIFO 写时钟 Bit i, 表示 C2H 通道 i 的写时钟
fifo_wrrstn_acq [CNUM-1:0]	输入	FIFO 写复位, 低有效 Bit i, 表示 C2H 通道 i 的写复位
fifo_wrreq_acq [CNUM-1:0]	输入	FIFO 写使能, 高有效 Bit i, 表示 C2H 通道 i 的写使能
fifo_data_acq [512*CNUM-1:0]	输入	FIFO 写数据 Bit 512*(i+1)-1~512*i, 表示 C2H 通道 i 的写数据
fifo_prog_full_acq [CNUM-1:0]	输出	FIFO 可编程满 (阈值等于深度-16), 高有效 Bit i, 表示 C2H 通道 i 的缓存可编程满信号

表 20 Config AXI4-Lite Master 接口信号

信号名称	输入输出	描述
m_axil_awaddr[31:0]	输出	write address
m_axil_awprot[2:0]	输出	write protection type
m_axil_awvalid	输出	write address valid
m_axil_awready	输入	write address ready
m_axil_wdata[31:0]	输出	write data
m_axil_wstrb[3:0]	输入	write strobes
m_axil_wvalid	输出	write valid
m_axil_wready	输入	write ready
m_axil_bresp[1:0]	输入	write response
m_axil_bvalid	输入	write response valid
m_axil_bready	输出	response ready
m_axil_araddr[31:0]	输出	read address
m_axil_arprot[2:0]	输出	read protection type
m_axil_arvalid	输出	read address valid
m_axil_arready	输入	read address ready
m_axil_rdata[31:0]	输入	read data
m_axil_rresp[1:0]	输入	read response
m_axil_rvalid	输入	read valid
m_axil_rready	输出	read ready

表 21 中断接口信号

信号名称	输入输出	描述
usr_intr_pos[15:0]	输入	用户中断输入。 Bit i: 用户中断#i 输入, 上升沿有效。

表 22 软复位接口信号

信号名称	输入输出	描述
c2h_dma_grst_n	输出	C2H DMA 全局复位输出，低有效。
c2h_dma_fsm_srst_n [CNUM-1:0]	输出	C2H 通道 i 的 DMA FSM 复位输出，低有效。
c2h_dma_buf_srst_n [CNUM-1:0]	输出	C2H 通道 i 的 DMA Buffer 复位输出，低有效。
h2c_dma_grst_n	输出	H2C DMA 全局复位输出，低有效。
h2c_dma_fsm_srst_n [CNUM-1:0]	输出	H2C 通道 i 的 DMA FSM 复位输出，低有效。
h2c_dma_buf_srst_n [CNUM-1:0]	输出	H2C 通道 i 的 DMA Buffer 复位输出，低有效。

表 23 用户寄存器接口信号

信号名称	输入输出	描述
acquisition_stat [32*CNUM-1:0]	输入	CNUM 个 C2H 通道状态 Bit 32*(i+1)-1~32*i, 表示 C2H 通道 i 的状态
display_stat [32*CNUM-1:0]	输入	CNUM 个 H2C 通道状态 Bit 32*(i+1)-1~32*i, 表示 H2C 通道 i 的状态
acquisition_res_vld [CNUM-1:0]	输出	CNUM 个 C2H 通道分辨率有效信号 Bit i, 表示 C2H 通道 i 分辨率的有效信号
display_res_vld [CNUM-1:0]	输出	CNUM 个 H2C 通道分辨率有效信号 Bit i, 表示 H2C 通道 i 分辨率的有效信号
acquisition_fps_vld [CNUM-1:0]	输出	CNUM 个 C2H 通道帧率有效信号 Bit i, 表示 C2H 通道 i 帧率的有效信号
display_fps_vld [CNUM-1:0]	输出	CNUM 个 H2C 通道帧率有效信号 Bit i, 表示 H2C 通道 i 帧率的有效信号
acquisition_xlen_vld [CNUM-1:0]	输出	CNUM 个 C2H 通道传输长度有效信号 Bit i, 表示 C2H 通道 i 传输长度的有效信号
display_xlen_vld [CNUM-1:0]	输出	CNUM 个 H2C 通道传输长度有效信号 Bit i, 表示 H2C 通道 i 传输长度的有效信号
acquisition_res [32*CNUM-1:0]	输出	CNUM 个 C2H 通道的分辨率设置 Bit 32*(i+1)-1~32*i, 表示 C2H 通道 i 的分辨率
display_res [32*CNUM-1:0]	输出	CNUM 个 H2C 通道的分辨率设置 Bit 32*(i+1)-1~32*i, 表示 H2C 通道 i 的分辨率
acquisition_fps [32*CNUM-1:0]	输出	CNUM 个 C2H 通道的帧率设置 Bit 32*(i+1)-1~32*i, 表示 C2H 通道 i 的帧率
display_fps [32*CNUM-1:0]	输出	CNUM 个 H2C 通道的帧率设置 Bit 32*(i+1)-1~32*i, 表示 H2C 通道 i 的帧率
acquisition_xlen [32*CNUM-1:0]	输出	CNUM 个 C2H 通道的传输长度设置 Bit 32*(i+1)-1~32*i, 表示 C2H 通道 i 的传输长

		度
display_xlen [32*CNUM-1:0]	输出	CNUM 个 H2C 通道的传输长度设置 Bit $32*(i+1)-1\sim 32*i$ , 表示 H2C 通道 i 的传输长度
acquisition_enable [CNUM-1:0]	输出	CNUM 个 C2H 通道的采集使能, 高有效 Bit $i = 1$ , 表示使能 C2H 通道 i 采集
display_enable [CNUM-1:0]	输出	CNUM 个 H2C 通道的显示使能, 高有效 Bit $i = 1$ , 表示使能 H2C 通道 i 显示
display_timing_enable [CNUM-1:0]	输出	CNUM 个 H2C 通道的显示定时使能, 高有效 Bit $i = 1$ , 表示使能 H2C 通道 i 显示定时
acq_usr_reset[CNUM-1:0]	输出	CNUM 个 C2H 通道的用户复位信号 Bit $i$ , 表示 C2H 通道 i 用户复位信号
disp_usr_reset[CNUM-1:0]	输出	CNUM 个 H2C 通道的用户复位信号 Bit $i$ , 表示 H2C 通道 i 用户复位信号
usr_ctrl[31:0]	输出	用户控制信号输出
usr_ctrl2[31:0]	输出	用户控制#2 信号输出
usr_stat[31:0]	输入	用户状态信号输入

表 24 Display FPS Timing 通道 0-CNUM-1 接口信号

信号名称	输入输出	描述
fifo_wrclk_disp_fps [CNUM-1:0]	输入	FIFO 写时钟 Bit $i$ , 表示 Display FPS Timing 通道 i 的写时钟
fifo_wrrstn_disp_fps [CNUM-1:0]	输入	FIFO 写复位, 低有效 Bit $i$ , 表示 Display FPS Timing 通道 i 的写复位
fifo_wrreq_disp_fps [CNUM-1:0]	输入	FIFO 写使能, 高有效 Bit $i$ , 表示 Display FPS Timing 通道 i 的写使能
fifo_prog_full_disp_fps [CNUM-1:0]	输出	FIFO 可编程满, 高有效 Bit $i$ , 表示 Display FPS Timing 通道 i 的缓存可编程满信号

### 3.5 寄存器空间

Host 可以通过将读或写请求映射到 BAR 方式访问 Multi-Channel PCIe QDMA Subsystem 内部的配置和状态寄存器以及用户逻辑中的配置和状态寄存器。根据 BAR 的命中, 读写请求被路由到相应的位置上。对于 PCIe BAR 分配, 请参考 Target Bridge。

### 3.5.1 PCIe to AXI4-Lite Master (BAR1) 地址映射

命中 PCIe to AXI4-Lite Master 空间的事务被路由到 AXI4-Lite Memory Mapped 用户接口上。这个接口支持 32-bit 地址空间和 32-bit 读写请求。PCIe to AXI4-Lite Master 的地址映射由用户逻辑定义。

### 3.5.2 PCIe to DMA (BAR0) 地址映射

命中 PCIe to DMA 空间的事务被路由到 Multi-Channel PCIe QDMA Subsystem 内部的配置寄存器总线。这个接口支持 32-bit 地址空间和 32-bit 读写请求。这些寄存器应用于 DMA 编程和状态检查。

表 25 PCIe to DMA 地址映射

名称	位宽	偏移	描述	复位值
CH0_C2H_ADDR_L	32	0x000	通道 0 C2H 目的地址 Lower, RW。	0x00000000
CH1_C2H_ADDR_L	32	0x004	通道 1 C2H 目的地址 Lower, RW。	0x00000000
CH2_C2H_ADDR_L	32	0x008	通道 2 C2H 目的地址 Lower, RW。	0x00000000
CH3_C2H_ADDR_L	32	0x00C	通道 3 C2H 目的地址 Lower, RW。	0x00000000
CH4_C2H_ADDR_L	32	0x010	通道 4 C2H 目的地址 Lower, RW。	0x00000000
CH5_C2H_ADDR_L	32	0x014	通道 5 C2H 目的地址 Lower, RW。	0x00000000
CH6_C2H_ADDR_L	32	0x018	通道 6 C2H 目的地址 Lower, RW。	0x00000000
CH7_C2H_ADDR_L	32	0x01C	通道 7 C2H 目的地址 Lower, RW。	0x00000000
CH0_C2H_ADDR_U	32	0x020	通道 0 C2H 目的地址 Upper, RW。	0x00000000
CH1_C2H_ADDR_U	32	0x024	通道 1 C2H 目的地址 Upper, RW。	0x00000000
CH2_C2H_ADDR_U	32	0x028	通道 2 C2H 目的地址 Upper, RW。	0x00000000
CH3_C2H_ADDR_U	32	0x02C	通道 3 C2H 目的地址 Upper, RW。	0x00000000
CH4_C2H_ADDR_U	32	0x030	通道 4 C2H 目的地址 Upper, RW。	0x00000000
CH5_C2H_ADDR_U	32	0x034	通道 5 C2H 目的地址 Upper, RW。	0x00000000
CH6_C2H_ADDR_U	32	0x038	通道 6 C2H 目的地址 Upper, RW。	0x00000000
CH7_C2H_ADDR_U	32	0x03C	通道 7 C2H 目的地址 Upper, RW。	0x00000000
CH0_H2C_ADDR_L	32	0x040	通道 0 H2C 源地址 Lower, RW。	0x00000000
CH1_H2C_ADDR_L	32	0x044	通道 1 H2C 源地址 Lower, RW。	0x00000000
CH2_H2C_ADDR_L	32	0x048	通道 2 H2C 源地址 Lower, RW。	0x00000000
CH3_H2C_ADDR_L	32	0x04C	通道 3 H2C 源地址 Lower, RW。	0x00000000
CH4_H2C_ADDR_L	32	0x050	通道 4 H2C 源地址 Lower, RW。	0x00000000
CH5_H2C_ADDR_L	32	0x054	通道 5 H2C 源地址 Lower, RW。	0x00000000
CH6_H2C_ADDR_L	32	0x058	通道 6 H2C 源地址 Lower, RW。	0x00000000
CH7_H2C_ADDR_L	32	0x05C	通道 7 H2C 源地址 Lower, RW。	0x00000000
CH0_H2C_ADDR_U	32	0x060	通道 0 H2C 源地址 Upper, RW。	0x00000000
CH1_H2C_ADDR_U	32	0x064	通道 1 H2C 源地址 Upper, RW。	0x00000000
CH2_H2C_ADDR_U	32	0x068	通道 2 H2C 源地址 Upper, RW。	0x00000000
CH3_H2C_ADDR_U	32	0x06C	通道 3 H2C 源地址 Upper, RW。	0x00000000



CH4_H2C_ADDR_U	32	0x070	通道 4 H2C 源地址 Upper, RW。	0x00000000
CH5_H2C_ADDR_U	32	0x074	通道 5 H2C 源地址 Upper, RW。	0x00000000
CH6_H2C_ADDR_U	32	0x078	通道 6 H2C 源地址 Upper, RW。	0x00000000
CH7_H2C_ADDR_U	32	0x07C	通道 7 H2C 源地址 Upper, RW。	0x00000000
CH0_C2H_XFER_SIZE	32	0x080	通道 0 C2H 传输长度, RW。	0x00000000
CH1_C2H_XFER_SIZE	32	0x084	通道 1 C2H 传输长度, RW。	0x00000000
CH2_C2H_XFER_SIZE	32	0x088	通道 2 C2H 传输长度, RW。	0x00000000
CH3_C2H_XFER_SIZE	32	0x08C	通道 3 C2H 传输长度, RW。	0x00000000
CH4_C2H_XFER_SIZE	32	0x090	通道 4 C2H 传输长度, RW。	0x00000000
CH5_C2H_XFER_SIZE	32	0x094	通道 5 C2H 传输长度, RW。	0x00000000
CH6_C2H_XFER_SIZE	32	0x098	通道 6 C2H 传输长度, RW。	0x00000000
CH7_C2H_XFER_SIZE	32	0x09C	通道 7 C2H 传输长度, RW。	0x00000000
CH0_H2C_XFER_SIZE	32	0x0A0	通道 0 H2C 传输长度, RW。	0x00000000
CH1_H2C_XFER_SIZE	32	0x0A4	通道 1 H2C 传输长度, RW。	0x00000000
CH2_H2C_XFER_SIZE	32	0x0A8	通道 2 H2C 传输长度, RW。	0x00000000
CH3_H2C_XFER_SIZE	32	0x0AC	通道 3 H2C 传输长度, RW。	0x00000000
CH4_H2C_XFER_SIZE	32	0x0B0	通道 4 H2C 传输长度, RW。	0x00000000
CH5_H2C_XFER_SIZE	32	0x0B4	通道 5 H2C 传输长度, RW。	0x00000000
CH6_H2C_XFER_SIZE	32	0x0B8	通道 6 H2C 传输长度, RW。	0x00000000
CH7_H2C_XFER_SIZE	32	0x0BC	通道 7 H2C 传输长度, RW。	0x00000000
CH0_C2H_FPS	32	0x0C0	通道 0 C2H 帧率, RW。	0x00000000
CH1_C2H_FPS	32	0x0C4	通道 1 C2H 帧率, RW。	0x00000000
CH2_C2H_FPS	32	0x0C8	通道 2 C2H 帧率, RW。	0x00000000
CH3_C2H_FPS	32	0x0CC	通道 3 C2H 帧率, RW。	0x00000000
CH4_C2H_FPS	32	0x0D0	通道 4 C2H 帧率, RW。	0x00000000
CH5_C2H_FPS	32	0x0D4	通道 5 C2H 帧率, RW。	0x00000000
CH6_C2H_FPS	32	0x0D8	通道 6 C2H 帧率, RW。	0x00000000
CH7_C2H_FPS	32	0x0DC	通道 7 C2H 帧率, RW。	0x00000000
CH0_H2C_FPS	32	0x0E0	通道 0 H2C 帧率, RW。	0x00000000
CH1_H2C_FPS	32	0x0E4	通道 1 H2C 帧率, RW。	0x00000000
CH2_H2C_FPS	32	0x0E8	通道 2 H2C 帧率, RW。	0x00000000
CH3_H2C_FPS	32	0x0EC	通道 3 H2C 帧率, RW。	0x00000000
CH4_H2C_FPS	32	0x0F0	通道 4 H2C 帧率, RW。	0x00000000
CH5_H2C_FPS	32	0x0F4	通道 5 H2C 帧率, RW。	0x00000000
CH6_H2C_FPS	32	0x0F8	通道 6 H2C 帧率, RW。	0x00000000
CH7_H2C_FPS	32	0x0FC	通道 7 H2C 帧率, RW。	0x00000000
CH0_C2H_CTRL	32	0x100	通道 0 C2H 控制, RW。 Bit 0: 设置为 '1' 启动 DMA 引擎 Bit 1: 设置为 '1' 开始 C2H 传输 Bit 31: 设置为 '1' 复位 C2H DMA	0x00000000
CH1_C2H_CTRL	32	0x104	通道 1 C2H 控制, RW。	0x00000000
CH2_C2H_CTRL	32	0x108	通道 2 C2H 控制, RW。	0x00000000
CH3_C2H_CTRL	32	0x10C	通道 3 C2H 控制, RW。	0x00000000

CH4_C2H_CTRL	32	0x110	通道 4 C2H 控制, RW。	0x00000000
CH5_C2H_CTRL	32	0x114	通道 5 C2H 控制, RW。	0x00000000
CH6_C2H_CTRL	32	0x118	通道 6 C2H 控制, RW。	0x00000000
CH7_C2H_CTRL	32	0x11C	通道 7 C2H 控制, RW。	0x00000000
CH0_H2C_CTRL	32	0x120	通道 0 H2C 控制, RW。 Bit 0: 设置为 ‘1’ 启动 DMA 引擎 Bit 1: 设置为 ‘1’ 开始 H2C 传输 Bit 2: 设置为 ‘1’ 使能显示定时 Bit 31: 设置为 ‘1’ 复位 H2C DMA	0x00000000
CH1_H2C_CTRL	32	0x124	通道 1 H2C 控制, RW。	0x00000000
CH2_H2C_CTRL	32	0x128	通道 2 H2C 控制, RW。	0x00000000
CH3_H2C_CTRL	32	0x12C	通道 3 H2C 控制, RW。	0x00000000
CH4_H2C_CTRL	32	0x130	通道 4 H2C 控制, RW。	0x00000000
CH5_H2C_CTRL	32	0x134	通道 5 H2C 控制, RW。	0x00000000
CH6_H2C_CTRL	32	0x138	通道 6 H2C 控制, RW。	0x00000000
CH7_H2C_CTRL	32	0x13C	通道 7 H2C 控制, RW。	0x00000000
CH0_C2H_STAT	32	0x140	通道 0 C2H 状态, RO。	0x00000000
CH1_C2H_STAT	32	0x144	通道 1 C2H 状态, RO。	0x00000000
CH2_C2H_STAT	32	0x148	通道 2 C2H 状态, RO。	0x00000000
CH3_C2H_STAT	32	0x14C	通道 3 C2H 状态, RO。	0x00000000
CH4_C2H_STAT	32	0x150	通道 4 C2H 状态, RO。	0x00000000
CH5_C2H_STAT	32	0x154	通道 5 C2H 状态, RO。	0x00000000
CH6_C2H_STAT	32	0x158	通道 6 C2H 状态, RO。	0x00000000
CH7_C2H_STAT	32	0x15C	通道 7 C2H 状态, RO。	0x00000000
CH0_H2C_STAT	32	0x160	通道 0 H2C 状态, RO。	0x00000000
CH1_H2C_STAT	32	0x164	通道 1 H2C 状态, RO。	0x00000000
CH2_H2C_STAT	32	0x168	通道 2 H2C 状态, RO。	0x00000000
CH3_H2C_STAT	32	0x16C	通道 3 H2C 状态, RO。	0x00000000
CH4_H2C_STAT	32	0x170	通道 4 H2C 状态, RO。	0x00000000
CH5_H2C_STAT	32	0x174	通道 5 H2C 状态, RO。	0x00000000
CH6_H2C_STAT	32	0x178	通道 6 H2C 状态, RO。	0x00000000
CH7_H2C_STAT	32	0x17C	通道 7 H2C 状态, RO。	0x00000000
CH0_C2H_RES	32	0x180	通道 0 C2H 分辨率, RW。	0x00000000
CH1_C2H_RES	32	0x184	通道 1 C2H 分辨率, RW。	0x00000000
CH2_C2H_RES	32	0x188	通道 2 C2H 分辨率, RW。	0x00000000
CH3_C2H_RES	32	0x18C	通道 3 C2H 分辨率, RW。	0x00000000
CH4_C2H_RES	32	0x190	通道 4 C2H 分辨率, RW。	0x00000000
CH5_C2H_RES	32	0x194	通道 5 C2H 分辨率, RW。	0x00000000
CH6_C2H_RES	32	0x198	通道 6 C2H 分辨率, RW。	0x00000000
CH7_C2H_RES	32	0x19C	通道 7 C2H 分辨率, RW。	0x00000000
CH0_H2C_RES	32	0x1A0	通道 0 H2C 分辨率, RW。	0x00000000
CH1_H2C_RES	32	0x1A4	通道 1 H2C 分辨率, RW。	0x00000000
CH2_H2C_RES	32	0x1A8	通道 2 H2C 分辨率, RW。	0x00000000

CH3_H2C_RES	32	0x1AC	通道 3 H2C 分辨率, RW。	0x00000000
CH4_H2C_RES	32	0x1B0	通道 4 H2C 分辨率, RW。	0x00000000
CH5_H2C_RES	32	0x1B4	通道 5 H2C 分辨率, RW。	0x00000000
CH6_H2C_RES	32	0x1B8	通道 6 H2C 分辨率, RW。	0x00000000
CH7_H2C_RES	32	0x1BC	通道 7 H2C 分辨率, RW。	0x00000000
CTRL	32	0x1E0	控制, RW。	0x00000000
CTRL2	32	0x1E4	控制 2, RW。	0x00000000
SRST	32	0x1E8	软复位, RW。 Bit 0: 设置为 ‘1’ 复位所有 C2H DMA Bit 1: 设置为 ‘1’ 复位所有 H2C DMA  Bit 16: 设置为 ‘1’ 复位 C2H#0 用户逻辑 Bit 17: 设置为 ‘1’ 复位 C2H#1 用户逻辑 Bit 18: 设置为 ‘1’ 复位 C2H#2 用户逻辑 Bit 19: 设置为 ‘1’ 复位 C2H#3 用户逻辑 Bit 20: 设置为 ‘1’ 复位 C2H#4 用户逻辑 Bit 21: 设置为 ‘1’ 复位 C2H#5 用户逻辑 Bit 22: 设置为 ‘1’ 复位 C2H#6 用户逻辑 Bit 23: 设置为 ‘1’ 复位 C2H#7 用户逻辑  Bit 24: 设置为 ‘1’ 复位 H2C#0 用户逻辑 Bit 25: 设置为 ‘1’ 复位 H2C#1 用户逻辑 Bit 26: 设置为 ‘1’ 复位 H2C#2 用户逻辑 Bit 27: 设置为 ‘1’ 复位 H2C#3 用户逻辑 Bit 28: 设置为 ‘1’ 复位 H2C#4 用户逻辑 Bit 29: 设置为 ‘1’ 复位 H2C#5 用户逻辑 Bit 31: 设置为 ‘1’ 复位 H2C#6 用户逻辑 Bit 32: 设置为 ‘1’ 复位 H2C#7 用户逻辑	0x00000000
INT_MASK	32	0x1EC	中断屏蔽, RW。 Bit 0: 设置为 ‘0’ 使能 CH0 C2H 中断 Bit 1: 设置为 ‘0’ 使能 CH1 C2H 中断 Bit 2: 设置为 ‘0’ 使能 CH2 C2H 中断 Bit 3: 设置为 ‘0’ 使能 CH3 C2H 中断 Bit 4: 设置为 ‘0’ 使能 CH4 C2H 中断 Bit 5: 设置为 ‘0’ 使能 CH5 C2H 中断 Bit 6: 设置为 ‘0’ 使能 CH6 C2H 中断 Bit 7: 设置为 ‘0’ 使能 CH7 C2H 中断 Bit 8: 设置为 ‘0’ 使能 CH0 H2C 中断 Bit 9: 设置为 ‘0’ 使能 CH1 H2C 中断 Bit 10: 设置为 ‘0’ 使能 CH2 H2C 中断 Bit 11: 设置为 ‘0’ 使能 CH3 H2C 中断 Bit 12: 设置为 ‘0’ 使能 CH4 H2C 中断 Bit 13: 设置为 ‘0’ 使能 CH5 H2C 中断 Bit 14: 设置为 ‘0’ 使能 CH6 H2C 中断	0xFFFFFFFF

			Bit 15: 设置为 ‘0’ 使能 CH7 H2C 中断 Bit 16: 设置为 ‘0’ 使能 User#0 中断 Bit 17: 设置为 ‘0’ 使能 User#1 中断 Bit 18: 设置为 ‘0’ 使能 User#2 中断 Bit 19: 设置为 ‘0’ 使能 User#3 中断 Bit 20: 设置为 ‘0’ 使能 User#4 中断 Bit 21: 设置为 ‘0’ 使能 User#5 中断 Bit 22: 设置为 ‘0’ 使能 User#6 中断 Bit 23: 设置为 ‘0’ 使能 User#7 中断 Bit 24: 设置为 ‘0’ 使能 User#8 中断 Bit 25: 设置为 ‘0’ 使能 User#9 中断 Bit 26: 设置为 ‘0’ 使能 User#10 中断 Bit 27: 设置为 ‘0’ 使能 User#11 中断 Bit 28: 设置为 ‘0’ 使能 User#12 中断 Bit 29: 设置为 ‘0’ 使能 User#13 中断 Bit 30: 设置为 ‘0’ 使能 User#14 中断 Bit 31: 设置为 ‘0’ 使能 User#15 中断	
INT_STAT	32	0x1F0	中断状态, RW1C。 Bit 0: ‘1’ 表示 CH0 C2H 中断 Bit 1: ‘1’ 表示 CH1 C2H 中断 Bit 2: ‘1’ 表示 CH2 C2H 中断 Bit 3: ‘1’ 表示 CH3 C2H 中断 Bit 4: ‘1’ 表示 CH4 C2H 中断 Bit 5: ‘1’ 表示 CH5 C2H 中断 Bit 6: ‘1’ 表示 CH6 C2H 中断 Bit 7: ‘1’ 表示 CH7 C2H 中断 Bit 8: ‘1’ 表示 CH0 H2C 中断 Bit 9: ‘1’ 表示 CH1 H2C 中断 Bit 10: ‘1’ 表示 CH2 H2C 中断 Bit 11: ‘1’ 表示 CH3 H2C 中断 Bit 12: ‘1’ 表示 CH4 H2C 中断 Bit 13: ‘1’ 表示 CH5 H2C 中断 Bit 14: ‘1’ 表示 CH6 H2C 中断 Bit 15: ‘1’ 表示 CH7 H2C 中断 Bit 16: ‘1’ 表示 User#0 中断 Bit 17: ‘1’ 表示 User#1 中断 Bit 18: ‘1’ 表示 User#2 中断 Bit 19: ‘1’ 表示 User#3 中断 Bit 20: ‘1’ 表示 User#4 中断 Bit 21: ‘1’ 表示 User#5 中断 Bit 22: ‘1’ 表示 User#6 中断 Bit 23: ‘1’ 表示 User#7 中断 Bit 24: ‘1’ 表示 User#8 中断 Bit 25: ‘1’ 表示 User#9 中断	0x00000000

			Bit 26: ‘1’ 表示 User#10 中断 Bit 27: ‘1’ 表示 User#11 中断 Bit 28: ‘1’ 表示 User#12 中断 Bit 29: ‘1’ 表示 User#13 中断 Bit 30: ‘1’ 表示 User#14 中断 Bit 31: ‘1’ 表示 User#15 中断	
INT_DLY	32	0x1F4	中断延时, RW。单位: 4ns。	0x00000000
STAT	32	0x1F8	状态, RO。	
EPS for VU+	32	0x1FC	Endpoint 状态寄存器, RO。 Bit 0: Link Down 0b: Link is Up, 1b: Link is Down  Bit 2~1: Link Status 00b: No receivers detected 01b: Link training in progress 10b: Link up, DL initialization in progress 11b: Link up, DL initialization completed  Bit 4~3 : Current Link Speed 00b: 2.5 GT/s PCI Express Link 01b: 5.0 GT/s PCI Express Link 10b: 8.0 GT/s PCI Express Link 11b: Reserved  Bit 7~5 : Negotiated Link Width 000b: x1 001b: x2 010b: x4 011b: x8 100b: x16 Other values are reserved  Bit 9~8 : Max Payload Size。 00b: 128 byte 01b: 256 byte 10b: 512 byte 11b: 1024 byte  Bit 12~10 : Max Read Request Size 000b: 128 byte 001b: 256 byte 010b: 512 byte 011b: 1024 byte 100b: 2048 byte	

			<p>101b: 4096 byte Other values are reserved</p> <p>Bit 13 : IO 地址空间译码使能。 0b: 禁止 IO 译码 1b: 使能 IO 译码</p> <p>Bit 14 : Memory 地址空间译码使能。 0b: 禁止 Memory 译码 1b: 使能 Memory 译码</p> <p>Bit 15 : Master 使能。 0b: 禁止 Master 1b: 使能 Master</p> <p>Bit 16 : INTx 中断消息禁止。 0b: 使能 INTx 中断消息 1b: 禁止 INTx 中断消息</p> <p>Bit 22~17 : Current LTSSM State</p> <p>Bit 23: RCB Status 0b: an RCB of 64 bytes 1b: an RCB of 128 bytes</p> <p>Bit 24 : MSI 中断使能。 0b: 禁止 MSI 中断 1b: 使能 MSI 中断</p> <p>Bit 31: Link Up 1b: Link is Up, 0b: Link is Down</p>	
EPS for KU and V7	32	0x1FC	<p>Endpoint 状态寄存器, RO。</p> <p>Bit 0: Link Down 0b: Link is Up, 1b: Link is Down</p> <p>Bit 2~1: Link Status 00b: No receivers detected 01b: Link training in progress 10b: Link up, DL initialization in progress 11b: Link up, DL initialization completed</p> <p>Bit 5~3 : Current Link Speed 001b: 2.5 GT/s PCI Express Link 010b: 5.0 GT/s PCI Express Link</p>	

			<p>100b: 8.0 GT/s PCI Express Link Other values are reserved</p> <p>Bit 9~6 : Negotiated Link Width 0001b: x1 0010b: x2 0100b: x4 1000b: x8 Other values are reserved</p> <p>Bit 12~10 : Max Payload Size。 000b: 128 byte 001b: 256 byte 010b: 512 byte 011b: 1024 byte Other values are reserved</p> <p>Bit 15~13 : Max Read Request Size 000b: 128 byte 001b: 256 byte 010b: 512 byte 011b: 1024 byte 100b: 2048 byte 101b: 4096 byte Other values are reserved</p> <p>Bit 16 : IO 地址空间译码使能。 0b: 禁止 IO 译码 1b: 使能 IO 译码</p> <p>Bit 17 : Memory 地址空间译码使能。 0b: 禁止 Memory 译码 1b: 使能 Memory 译码</p> <p>Bit 18 : Master 使能。 0b: 禁止 Master 1b: 使能 Master</p> <p>Bit 19 : INTx 中断消息禁止。 0b: 使能 INTx 中断消息 1b: 禁止 INTx 中断消息</p> <p>Bit 25~20 : Current LTSSM State</p>	
--	--	--	--	--

			<p>Bit 26: RCB Status  0b: an RCB of 64 bytes  1b: an RCB of 128 bytes</p> <p>Bit 27 : MSI 中断使能。  0b: 禁止 MSI 中断  1b: 使能 MSI 中断</p> <p>Bit 31: Link Up  1b: Link is Up, 0b: Link is Down</p>	
EPS for K7	32	0x1FC	<p>Endpoint 状态寄存器, RO。</p> <p>Bit 0: Link Up  1b: Link is Up, 0b: Link is Down</p> <p>Bit 3~1: Link Status  000b: L0  001b: PPM L1  010b: PPM L2/L3 Ready  011b: PM_PME  100b: in or transitioning to/from ASPM L0s  101b: transitioning to/from PPM L1  110b: transition to PPM L2/L3 Ready  111b: Reserved</p> <p>Bit 5~4 : Current Link Speed  01b: 2.5 GT/s PCI Express Link  10b: 5.0 GT/s PCI Express Link  Other values are reserved</p> <p>Bit 9~6 : Negotiated Link Width  0001b: x1  0010b: x2  0100b: x4  1000b: x8  Other values are reserved</p> <p>Bit 12~10 : Max Payload Size。  000b: 128 byte  001b: 256 byte  010b: 512 byte  011b: 1024 byte  Other values are reserved</p> <p>Bit 15~13 : Max Read Request Size</p>	



			<p>000b: 128 byte  001b: 256 byte  010b: 512 byte  011b: 1024 byte  100b: 2048 byte  101b: 4096 byte  Other values are reserved</p> <p>Bit 16 : IO 地址空间译码使能。  0b: 禁止 IO 译码  1b: 使能 IO 译码</p> <p>Bit 17 : Memory 地址空间译码使能。  0b: 禁止 Memory 译码  1b: 使能 Memory 译码</p> <p>Bit 18 : Master 使能。  0b: 禁止 Master  1b: 使能 Master</p> <p>Bit 19 : INTx 中断消息禁止。  0b: 使能 INTx 中断消息  1b: 禁止 INTx 中断消息</p> <p>Bit 25~20 : Current LTSSM State</p> <p>Bit 26: RCB Status  0b: an RCB of 64 bytes  1b: an RCB of 128 bytes</p> <p>Bit 27 : MSI 中断使能。  0b: 禁止 MSI 中断  1b: 使能 MSI 中断</p> <p>Bit 31: Link Up  1b: Link is Up, 0b: Link is Down</p>	
--	--	--	--	--

## 3.6 DMA Flow

### 3.6.1 Global Reset

#### 3.6.1.1 Global Reset for SGDMA

- a. 停止 CH#i H2C DMA: 0 写入 CH#i\_H2C\_CTRL, 遍历 i
- b. 停止 CH#i C2H DMA: 0 写入 CH#i\_C2H\_CTRL, 遍历 i

- c. 施加 C2H 和 H2C DMA 总复位：SRST 设置为 3，延时 10us
- d. 释放 C2H 和 H2C DMA 总复位：SRST 设置为 0，延时 10us

### 3.6.1.2 Global Reset for CDMA-C2H

- a. 停止 CH#i C2H DMA：0 写入 CH#i\_C2H\_CTRL，遍历 i
- b. 施加 C2H DMA 总复位：SRST 设置为 1，延时 10us
- c. 释放 C2H DMA 总复位：SRST 设置为 0，延时 10us

### 3.6.1.3 Global Reset for CDMA-H2C

- a. 停止 CH#i H2C DMA：0 写入 CH#i\_H2C\_CTRL，遍历 i
- b. 施加 H2C DMA 总复位：SRST 设置为 2，延时 10us
- c. 释放 H2C DMA 总复位：SRST 设置为 0，延时 10us

## 3.6.2 CH#i H2C Flow for SGDMA

### 3.6.2.1 初始化与启动 CH#i H2C

- a. 停止 CH#i H2C DMA：0 写入 CH#i\_H2C\_CTRL
- b. 施加 CH#i H2C DMA 复位：0x80000000 写入 CH#i\_H2C\_CTRL，延时 10us
- c. 释放 CH#i H2C DMA 复位：0 写入 CH#i\_H2C\_CTRL，延时 10us
- d. CH#i H2C 中断清除：1<<(i+8)写入 INT\_STAT
- e. CH#i H2C 中断使能：INT\_MASK[i+8]设置为 0
- f. Host 向 Card 的 CH#i H2C 地址队列写入一个或多个 DMA 内存地址：配置 CH#i\_H2C\_ADDR\_U 和 CH#i\_H2C\_ADDR\_L 寄存器
- g. Host 配置 CH#i 显示定时：如果使能显示定时 CH#i\_H2C\_CTRL[2]设置为 1；如果禁止显示定时 CH#i\_H2C\_CTRL[2]设置为 0
- h. Host 启动 Card 的 CH#i H2C DMA 引擎：CH#i\_H2C\_CTRL[0]设置为 1
- i. Host 配置 Card 的 CH#i H2C 参数：配置 CH#i\_H2C\_FPS 寄存器
- j. Host 启动 Card 的 CH#i H2C 工作：CH#i\_H2C\_CTRL[1]设置为 1

### 3.6.2.2 CH#i H2C DMA

- a. Card 完成 CH#i H2C DMA，Card 向 Host 发出 MSI 中断
- b. Host 读取中断状态寄存器：读取 INT\_STAT
- c. Host 清除该中断：回写 INT\_STAT
- d. Host 向 CH#i H2C 地址队列放入新的 DMA 内存地址：配置 CH#i\_H2C\_ADDR\_U 和 CH#i\_H2C\_ADDR\_L 寄存器

### 3.6.2.3 停止 CH#i H2C

- a. Host 关闭 Card 的 CH#i H2C DMA 引擎以及禁止 CH#i H2C 工作：0 写入 CH#i\_H2C\_CTRL
- b. CH#i H2C 中断禁止：INT\_MASK[i+8] 设置为 1

## 3.6.3 CH#i C2H Flow for SGDMA

### 3.6.3.1 初始化与启动 CH#i C2H

- a. 停止 CH#i C2H DMA：0 写入 CH#i\_C2H\_CTRL
- b. 施加 CH#i C2H DMA 复位：0x80000000 写入 CH#i\_C2H\_CTRL，延时 10us
- c. 释放 CH#i C2H DMA 复位：0 写入 CH#i\_C2H\_CTRL，延时 10us
- d. CH#i C2H 中断清除：1<<i 写入 INT\_STAT
- e. CH#i C2H 中断使能：INT\_MASK[i] 设置为 0
- f. Host 向 Card 的 CH#i C2H 地址队列写入一个或多个 DMA 内存地址：配置 CH#i\_C2H\_ADDR\_U 和 CH#i\_C2H\_ADDR\_L 寄存器
- g. Host 启动 Card 的 CH#i C2H DMA 引擎：CH#i\_C2H\_CTRL[0] 设置为 1
- h. Host 配置 Card 的 CH#i C2H 参数：配置 CH#i\_C2H\_FPS 寄存器
- i. Host 启动 Card 的 CH#i C2H 工作：CH#i\_C2H\_CTRL[1] 设置为 1

### 3.6.3.2 CH#i C2H DMA

- a. Card 完成 CH#i C2H DMA，Card 向 Host 发出 MSI 中断
- b. Host 读取中断状态寄存器：读取 INT\_STAT
- c. Host 清除该中断：回写 INT\_STAT
- d. Host 向 CH#i C2H 地址队列放入新的 DMA 内存地址：配置 CH#i\_C2H\_ADDR\_U 和 CH#i\_C2H\_ADDR\_L 寄存器

### 3.6.3.3 停止 CH#i C2H

- a. Host 关闭 Card 的 CH#i C2H DMA 引擎以及禁止 CH#i C2H 工作：0 写入 CH#i\_C2H\_CTRL
- b. CH#i C2H 中断禁止：INT\_MASK[i] 设置为 1

## 3.6.4 CH#i H2C Flow for CDMA

### 3.6.4.1 初始化与启动 CH#i H2C

- a. 停止 CH#i H2C DMA：0 写入 CH#i\_H2C\_CTRL
- b. 施加 CH#i H2C DMA 复位：0x80000000 写入 CH#i\_H2C\_CTRL，延时 10us

- c. 释放 CH#i H2C DMA 复位：0 写入 CH#i\_H2C\_CTRL，延时 10us
- d. CH#i H2C 中断清除：1<<(i+8)写入 INT\_STAT
- e. CH#i H2C 中断使能：INT\_MASK[i+8]设置为 0
- f. Host 向 Card 的 CH#i H2C 地址队列写入一个或多个 DMA 内存地址和 DMA 传输长度：配置 CH#i\_H2C\_ADDR\_U 和 CH#i\_H2C\_ADDR\_L 寄存器以及 CH#i\_H2C\_XFER\_SIZE 寄存器
- g. Host 启动 Card 的 CH#i H2C DMA 引擎：CH#i\_H2C\_CTRL[0]设置为 1
- h. Host 配置 Card 的 CH#i H2C 参数：配置 CH#i\_H2C\_FPS 寄存器
- i. Host 启动 Card 的 CH#i H2C 工作：CH#i\_H2C\_CTRL[1]设置为 1

#### 3.6.4.2 CH#i H2C DMA

- a. Card 完成 CH#i H2C DMA，Card 向 Host 发出 MSI 中断
- b. Host 读取中断状态寄存器：读取 INT\_STAT
- c. Host 清除该中断：回写 INT\_STAT
- d. Host 向 CH#i H2C 地址队列放入新的 DMA 内存地址和 DMA 传输长度：配置 CH#i\_H2C\_ADDR\_U 和 CH#i\_H2C\_ADDR\_L 寄存器以及 CH#i\_H2C\_XFER\_SIZE 寄存器

#### 3.6.4.3 停止 CH#i H2C

- a. Host 关闭 Card 的 CH#i H2C DMA 引擎以及禁止 CH#i H2C 工作：0 写入 CH#i\_H2C\_CTRL
- b. CH#i H2C 中断禁止：INT\_MASK[i+8]设置为 1

### 3.6.5 CH#i C2H Flow for CDMA

#### 3.6.5.1 初始化与启动 CH#i C2H

- a. 停止 CH#i C2H DMA：0 写入 CH#i\_C2H\_CTRL
- b. 施加 CH#i C2H DMA 复位：0x80000000 写入 CH#i\_C2H\_CTRL，延时 10us
- c. 释放 CH#i C2H DMA 复位：0 写入 CH#i\_C2H\_CTRL，延时 10us
- d. CH#i C2H 中断清除：1<<i 写入 INT\_STAT
- e. CH#i C2H 中断使能：INT\_MASK[i]设置为 0
- f. Host 向 Card 的 CH#i C2H 地址队列写入一个或多个 DMA 内存地址和 DMA 传输长度：配置 CH#i\_C2H\_ADDR\_U 和 CH#i\_C2H\_ADDR\_L 寄存器以及 CH#i\_C2H\_XFER\_SIZE 寄存器

- g. Host 启动 Card 的 CH#i C2H DMA 引擎：CH#i\_C2H\_CTRL[0]设置为 1
- h. Host 配置 Card 的 CH#i C2H 参数：配置 CH#i\_C2H\_FPS 寄存器
- i. Host 启动 Card 的 CH#i C2H 工作：CH#i\_C2H\_CTRL[1]设置为 1

#### 3.6.5.2 CH#i C2H DMA

- a. Card 完成 CH#i C2H DMA，Card 向 Host 发出 MSI 中断
- b. Host 读取中断状态寄存器：读取 INT\_STAT
- c. Host 清除该中断：回写 INT\_STAT
- d. Host 向 CH#i C2H 地址队列放入新的 DMA 内存地址和 DMA 传输长度：配置 CH#i\_C2H\_ADDR\_U 和 CH#i\_C2H\_ADDR\_L 寄存器以及 CH#i\_C2H\_XFER\_SIZE 寄存器

#### 3.6.5.3 停止 CH#i C2H

- a. Host 关闭 Card 的 CH#i C2H DMA 引擎以及禁止 CH#i C2H 工作：0 写入 CH#i\_C2H\_CTRL
- b. CH#i C2H 中断禁止：INT\_MASK[i]设置为 1

### 3.7 设备驱动

#### 3.7.1 Windows WDF (Queue or Non-Queue)

#### 3.7.2 Linux (Queue or Non-Queue)

#### 3.7.3 V4L2