



PCIe SGDMA Reference Design

上海安路信息科技股份有限公司

APUG069 (v3.0) 2023 年 12 月



目 录

目 录	1
1 概要	1
2 整体功能	2
2.1 系统框图	2
2.2 特性说明	3
2.3 接口性能	5
2.4 资源使用与时序信息	6
3 文件目录说明	7
4 设计接口与功能介绍	8
4.1 顶层模块接口	8
4.2 顶层模块时序	14
4.3 H2C 功能流程	14
4.4 C2H 功能流程	15
4.5 寄存器空间	16
4.6 描述符	43
4.7 其它功能	45
5 功能仿真验证	49
5.1 系统仿真框架	49
5.2 仿真流程	50
5.3 测试激励介绍	51
5.4 仿真波形分析	52



6 移植与上板验证	56
6.1 软硬件环境准备	56
6.2 SGDMA 工程移植	56
6.3 Linux 驱动移植	60
6.4 Linux 上板验证	60
6.5 Windows 驱动介绍	62
6.5.1 文件目录	62
6.5.2 文件编译	62
6.5.3 上板验证	62
6.5.4 驱动安装	63
6.5.5 APP 测试	68
7 参考文档	69
8 版本信息	71
免责声明	71



1 概要

安路科技提供的 SGDMA IP，可作为一个 PCIe2AXI4 系列接口的桥接或者一个高性能 DMA 使用。SGDMA IP 支持嵌入式标准协议 AMBA4，提供 AXI4-MM 接口、AXI4-Stream 接口、AXI4-Lite 接口、CFG_MGMT 接口等总线接口。

SGDMA 核心架构广泛应用于数据通信网络、电信网络、宽带有线和无线应用、网络接口卡、芯片到芯片和背板接口卡等领域，适用于各种应用程序的服务器附加卡。具有高性能、低成本、可扩展性和可靠性等特点。

本文档在 PH1A 系列、PH1A100 系列和 PH2A 系列器件的基础上介绍了 SGDMA IP 参考设计，方便用户快速掌握其功能和使用方法。

2 整体功能

PCI Express (以下简称 PCIe) DMA 桥接子系统实现了高性能、可配置的 Scatter Gather DMA。下文从系统构造、IP 属性、接口性能三个方向介绍 SGDMA IP 的整体功能。

2.1 系统框图

PCIe 的 DMA 桥接子系统可以配置为高性能直接内存访问 (DMA) 数据移动器或 PCIe 到用户之间的桥梁。

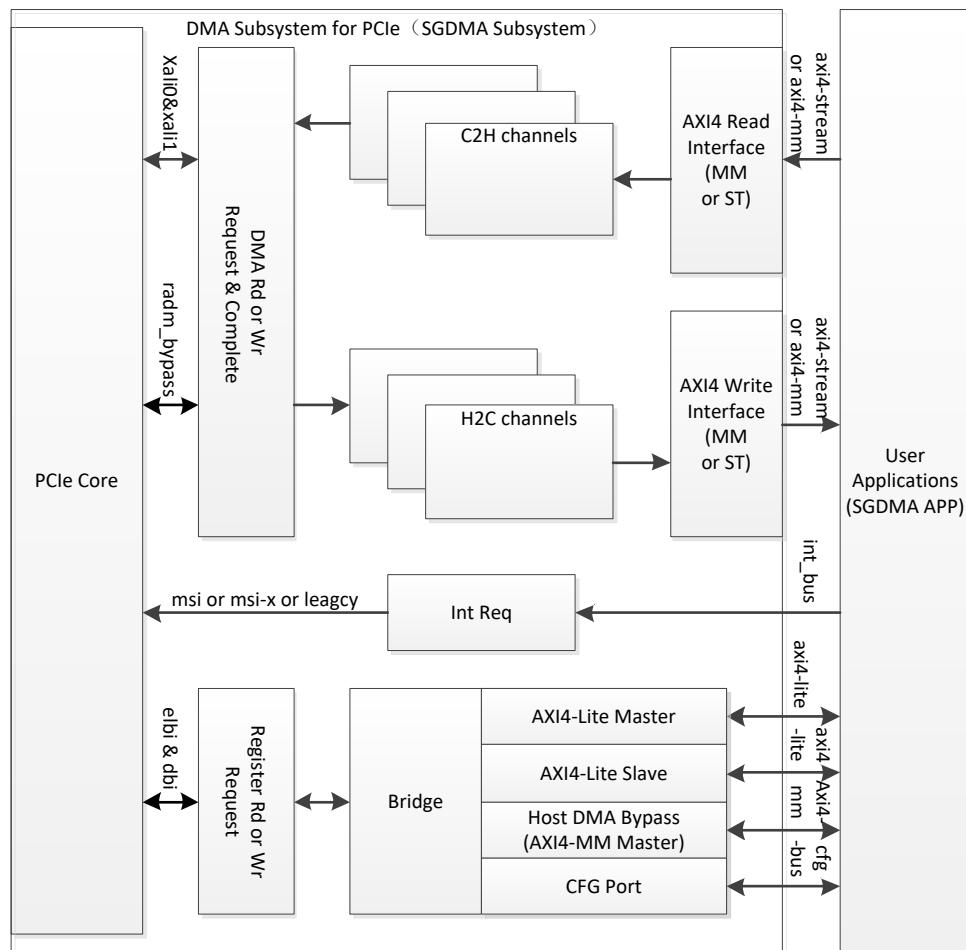


图 2-1 DMA 子系统框图

上图展示了两类总线: 高速总线 axi4-mm 或 axi4-stream, 低速总线 axi4-lite 或 cfg_bus 等, 这两类总线分别完成数据的移动或桥接, 和配置寄存器的读写。

DMA 数据移动器: 作为 DMA 子系统可以配置成 AXI4-MM 接口或 AXI4-Stream 接口以允许直接连接到 RTL 逻辑。在使用安路科技提供的驱动后, 其中任一接口都可用于块状数据在系统内存空间到 AXI4 地址空间的高效传输。除了基本的 DMA 功能外, DMA 子系统后续版本还将支持高达 N(资源允许的最大值)个 C2H 和 H2C 通道; PCIe 流量绕过 DMA 引擎; 以及一个可选的描述符旁路, 用于管理来自 FPGA 架构的描述符, 用于需要最高性能和最低延迟的应用程序。

PCIe 到 AXI4 系列接口的桥梁：作为桥梁，子系统可以完成 PCIe 高速本地总线到 AXI4-MM 或 AXI4-Stream 接口转换。配置寄存器的读写：AXI4-Lite Master 完成低速本地总线到 axi4-lite 总线的转换，主要用作 User 寄存器的读写；AXI4-Lite Slave 完成 axi4-lite 到 DMA 引擎内部私有总线的转换，主要用于 User 读取 DMA 引擎寄存器；Bypass Descriptor Port 完成 bypass_bus 到 DMA 引擎内部又一私有总线的转换，主要用于当 CPU 不清楚 DDR 地址空间的状态时从 User 获取描述符信息；CFG Port 完成 PCIe 配置空间读写总线 dbi 到 cfg_bus 的转换，主要用于用户读写 PCIe Core 的配置空间。

上图 2-1 为 DMA 子系统简略框图，当用户配置成 AXI4-MM 总线模式时，要外接 DDR 或 BRAM，由于 DDR 和 SGDMA 处在两个不同的时钟域，中间需要添加 AXI-Connect 做时钟域转换，再与 AXI4-MM 总线的 DDR 控制器相连接，简略的子系统框图如下图 2-2 所示。如果选择外接 BRAM，DMA Subsystem 直接外接 AXI4-MM 接口的 BRAM 即可。

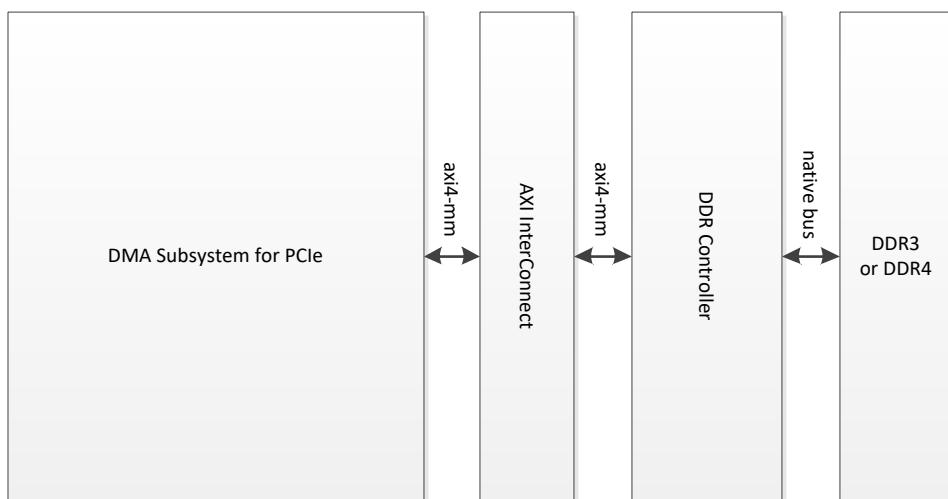


图 2-2 AXI4-MM 接口 DMA 子系统框图

2.2 特性说明

如上文所述，目前 PH1A 系列和 PH1A100 系列的 SGDMA IP 特性如下所示：

- 64bit (对应 PH1A100) 和 128bit (对应 PH1A400、PH1A90、PH1A180) 数据位宽
- 32bit 或 64bit 源地址、目的地址和描述符地址
- 支持单通道 C2H，后续版本支持 N 通道 C2H
- 支持单通道 H2C，后续版本支持 N 通道 H2C
- 单通道 AXI4-Stream 接口或者单通道 AXI4-MM 接口
- AXI4-Lite Master 读写用户和 DMA 寄存器支持
- AXI4-Lite Slave 读 DMA 状态寄存器支持
- 支持 Scatter Gather 描述符列表和环形描述符列表，列表大小无限制



- 单个描述符大小最大 256MB，支持链式和环形描述符
- 目前支持 MSI 中断，后续支持 Legacy 和 MSI-X 中断
- AXI4-Stream 地址支持 Byte 对齐，长度 64byte 对齐；AXI4-MM 地址和长度都为 byte 对齐
- 异常操作或错误，RUN 下降沿后重启能恢复正常
- 支持任意的源地址和目的地址
- C2H 支持 Poll mode 和 Stream Writeback，H2C 支持 Poll mode

PH2A 系列（500K）的 SGDMA IP 特性如下所示：

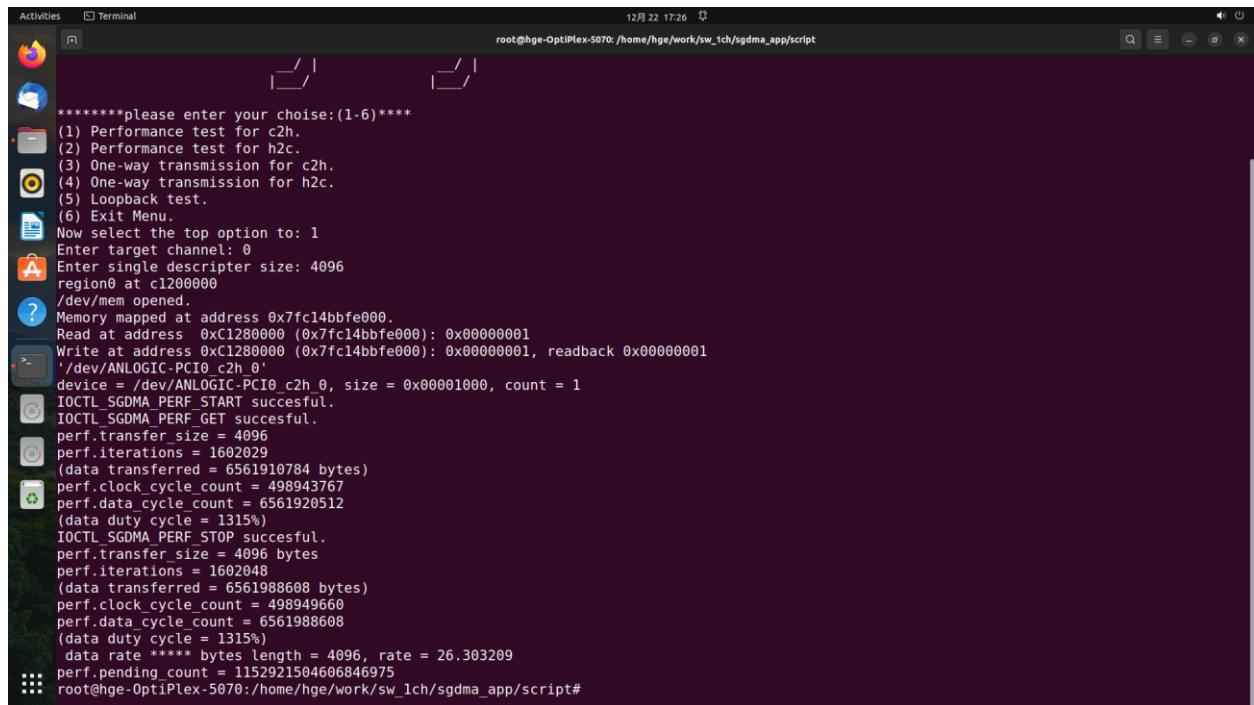
- PCIe Basic Mode, GEN1X1~GEN4X8
- Address width 64bit 或 32bit, Data Width 512bit, 提供 AXI-Width-Converter 模块
- AXI4-MM 接口, AXI4 Clock 62.5MHz~300MHz, 目前支持读写 BRAM 和 DDR4 测试
- 包长和地址都为 byte 对齐
- 支持 PCIe to DMA Bypass Interface
- 支持一个 C2H 通道，一个 H2C 通道，读请求 ID 为 32 个，写请求 ID 为 16 个
- 支持 AXI4-Lite Slave Interface
- 支持 MSI 中断，一个中断矢量，后续支持 Legacy 和 MSI-X 中断
- 支持 Extended Tag Field，最多 256 个 Tag
- 支持 Configuration Management interface
- 支持使能槽时钟配置
- 输出 DMA Status Ports，后续支持
- User 中断与引擎协同，后续支持

由于逻辑部分收敛时序的限制，本设计方案最大支持 100K 器件的 Gen2 X2, 400K、90K 和 180K 器件的 Gen3 X4。如有 PCIe 硬核支持 GEN3 的需求，应用程序中需要注意逻辑级数不要超过 5 级。PH2A 系列没有上述限制。

PCIe 硬核提供了一个轻量级数据总线来配置寄存器，这个总线掩码到 BAR0，所以设计中的所有配置都通过操作 BAR0 的地址来实现，目前 BAR0 开设了 1MByte 的空间，低 512KByte 分配给 DMA 控制器，另外一半预留给 User。

2.3 接口性能

安路科技提供的 SGDMA IP 性能如下图 2-3 和图 2-4 所示，测试在 PH1A 系列(400K)开发板上进行，DMA 收发的文件大小为 1MByte。其它配置还包括：主机和开发板协商到 Gen3 X4，数据位宽 128bit，1 个 H2C 和 C2H 通道，MPS 128 字节，MRSS 512 字节，操作系统 Ubuntu20.4 或 CENTOS 7.4，PolIMode 模式。测试结果 H2C 的速率已经达到 22Gbit/s, C2H 的速率为 26Gbit/s。



The screenshot shows a terminal window titled 'Terminal' with the command 'root@hge-OptiPlex-5070: /home/hge/work/sw_1ch/sgdma_app/script'. The window displays a menu of options for testing SGDMA performance:

```
*****please enter your choise:(1-6)*****
(1) Performance test for c2h.
(2) Performance test for h2c.
(3) One-way transmission for c2h.
(4) One-way transmission for h2c.
(5) Loopback test.
(6) Exit Menu.
```

Then, the user selects option 1 and enters target channel 0. The script then performs a series of operations to measure performance, including mapping memory at address 0x7fc14bbfe000, reading from address 0xC1280000, writing to address 0xC1280000, and performing ioctl operations for performance monitoring. The final output shows a data rate of approximately 26.303209 Gbit/s.

```
Now select the top option to: 1
Enter target channel: 0
Enter single descriptor size: 4096
region0 at c1280000
/dev/mem opened.
Memory mapped at address 0x7fc14bbfe000.
Read at address 0xC1280000 (0x7fc14bbfe000): 0x00000001
Write at address 0xC1280000 (0x7fc14bbfe000): 0x00000001, readback 0x00000001
'/dev/ANLOGIC-PCIO_c2h_0'
device = /dev/ANLOGIC-PCIO_c2h_0, size = 0x00001000, count = 1
IOCTL_SGDMA_PERF_START succesful.
IOCTL_SGDMA_PERF_GET succesful.
perf.transfer_size = 4096
perf.iterations = 1602029
(data transferred = 6561910784 bytes)
perf.clock_cycle_count = 498943767
perf.data_cycle_count = 6561920512
(data duty cycle = 1315%)
IOCTL_SGDMA_PERF_STOP succesful.
perf.transfer_size = 4096 bytes
perf.iterations = 1602048
(data transferred = 6561988608 bytes)
perf.clock_cycle_count = 498949660
perf.data_cycle_count = 6561988608
(data duty cycle = 1315%)
data rate ***** bytes length = 4096, rate = 26.303209
perf.pending_count = 1152921504606846975
root@hge-OptiPlex-5070:/home/hge/work/sw_1ch/sgdma_app/script#
```

图 2-3 单通道 C2H 测试性能



```
Activities Terminal 12月 22 17:27 root@hge-OptiPlex-5070:/home/hge/work/sw_1ch/sgdma_app/script
*****please enter your choise:(1-6)*****
(1) Performance test for c2h.
(2) Performance test for h2c.
(3) One-way transmission for c2h.
(4) One-way transmission for h2c.
(5) Loopback test.
(6) Exit Menu.
Now select the top option to: 2
Enter target channel: 0
Enter single descriptor size: 4096
region0 at c1200000
/dev/mem opened.
Memory mapped at address 0x7feba91c6000.
Read at address 0xc1280000 (0x7feba91c6000): 0x00000001
Write at address 0xc1280000 (0x7feba91c6000): 0x00000002, readback 0x00000002
'/dev/ANLOGIC-PCI0_h2c_0'
device = /dev/ANLOGIC-PCI0_h2c_0, size = 0x00001000, count = 1
IOCTL_SGDMA_PERF_START successful.
IOCTL_SGDMA_PERF_GET successful.
perf.transfer_size = 4096
perf.iterations = 1349550
(data transferred = 5527756800 bytes)
perf.clock_cycle_count = 498914734
perf.data_cycle_count = 5527766528
(data duty cycle = 1107%)
IOCTL_SGDMA_PERF_STOP successful.
perf.transfer_size = 4096 bytes
perf.iterations = 1349565
(data transferred = 5527818240 bytes)
perf.clock_cycle_count = 498920539
perf.data_cycle_count = 5527822848
(data duty cycle = 1107%)
data rate ***** bytes length = 4096, rate = 22.159131
perf.pending_count = 1152921504606846975
root@hge-OptiPlex-5070:/home/hge/work/sw_1ch/sgdma_app/script#
```

图 2-4 单通道 H2C 测试性能

2.4 资源使用与时序信息

sgdma_subsys_exam(单通道 AXI4-Stream)工程选择芯片 PH1A400, PCIe 配置成 Gen2X4, 在 TD5.6.4 编译后, 资源的使用情况如下表 2-1 所示:

表 2-1 sgdma_subsys_exam 工程资源使用与时序信息

名称	WNS	WHS	LUT6	REG	ERAM	DSP	PLL	GCLK	PCIe
数值	0.152	0.002	7399	4554	23	0	1	7	1

当总线接口选择 AXI4-MM, 芯片选择 PH1A400, PCIe 配置成 Gen2X4, 在 TD5.6.4 编译后, 资源的使用情况如下表 2-2 所所示:

表 2-2 sgdma_subsys_exam (AXI4-MM 接口) 工程资源使用与时序信息

名称	WNS	WHS	LUT6	REG	ERAM	DSP	PLL	GCLK	PCIe
数值	0.001	0.000	25044	45046	52	0	3	11	1

当总线接口选择 AXI4-MM, 芯片选择 PH2A106, PCIe 配置成 Gen1X1, 在 TD5.7.1_81009-64bit 编译后, 资源的使用情况如下表 2-3 所所示:

表 2-3 sgdma_subsys_exam (AXI4-MM 接口) 工程资源使用与时序信息

名称	WNS	WHS	LUT6	REG	ERAM	DSP	PLL	GCLK	PCIe
数值	0.001	0.000	27626	24749	140	0	2	/	1



3 文件目录说明

sgdma_prj (400K 版本, 其它版本与此版本类似) 工程文件目录包含四部分: doc 设计文档、impl 编译目录、src 源文件, sw 驱动文件和 tb 测试文件, 各目录下文件的详细介绍如下表 3-1 所示:

表 3-1 文件目录说明

目录	文件	文件说明
\impl\ep_prj	sgdmaXX_subsys_exam.al 或 sgdmaXXmm_subsys_exam.al	td 工程文件, 对应 AXI4-Stream 接口的工程或 AXI4-MM 接口的工程
\impl\ep_prj\constraint	sgdmaXX_subsys_exam.sdc	工程时序约束文件
	sgdmaXX_subsys_exam.adc	pin 约束及 pll 位置约束文件
\impl\rc_prj		暂未放 rc 的工程
\src\sgdma_app	\ip*.v	app 所用 ip 文件
	\src*.v	app 所用源文件
\src\sgdma_ip	\def*.vh	sgdma_ip 中用到的宏定义
	\ip*.v	sgdma_ip 中用到 ip 文件
	\src*.v 或 src_enc/*.v	sgdma_ip 中用到源文件
	\top*.v	sgdma_ip 顶层文件
\src\axi_connect	\src*.v	axi_connect 的源文件, 网表格式
\src\ddr3	\src*.v 或 \ip*.v 等	ddr 控制器的源文件和 IP 文件
\src\mc	\src*.v 或 \ip*.v 等	
\src\mcu	\src*.v 或 \ip*.v 等	
\sw	\sgdma_app\script*	驱动测试脚本文件
	\sgdma_app\src*	驱动 app 源文件
	\sgdma_drv\anlogic_pcie_drv*.c *.h *.sh	驱动源文件
\sw_win	AnlogicPcie.*	windows 驱动以及应用文件
\tb	\bfm*.v	rc 端接口仿真模型文件
	\data\case**.dat	tb 数据, 包括 golden 和 sim
	\def*.vh	tb 中用到的宏定义文件
	\qsim*.do *.f	tb 中的 modelsim 仿真文件
	\sim*.do *.f *.pl	tb 的自动化仿真文件
	\top*.v	tb 的顶层文件
\doc	*.docx	参考设计文档及调试记录

4 设计接口与功能介绍

本设计可实现两部分功能：DMA 控制器或 PCIe 本地总线到 AXI4 系列接口的转换，顶层接口如下图 4-1 所示。

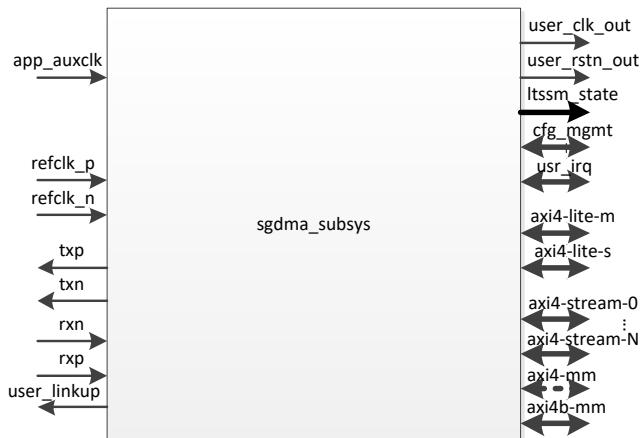


图 4-1 DMA/Bridge 子系统顶层结构图

4.1 顶层模块接口

顶层模块 `sgdma_subsys` 主要合并了 `pcie_phy` 和 `sgdma_ip` 两个子模块，对外接口包含 `sgdma_ip` 接口、PCIe 相关的部分输入输出接口及一些内部预留总线。用户可用`AXI4ST_BUS_EN`、`AXI4MM_BUS_EN` 或`DMA_BYPA_EN`、`AXI4L_MBUS_EN`、`AXI4L_SBUS_EN`、`USR_IRQ_EN`、`CFG_MGMT_EN` 这六类总线。接口信号功能描述如下表 4-1 所述：

表 4-1 `sgdma_subsys` 接口介绍

信号名称	方向	位宽	描述
<code>app_auxclk/tl_clk_user</code>	I	1	如果为 PH1A 系列或 PH1A100 系列芯片， <code>app_auxclk</code> 为 PCIe 硬核提供常在的辅助时钟；如果选择的是 PH2A 系列芯片， <code>tl_clk_user</code> 为用户提供给 PCIe 核的时钟。
<code>refclk_p</code>	I	1	PCIe 硬核参考时钟 100MHz 差分。
<code>refclk_n</code>	I	1	PCIe 硬核参考时钟 100MHz 差分。
<code>txp</code>	O	/	`LINK_WIDTH bit，在*.vh 中修改，PCIe 硬核输出 Serdes 信号差分。
<code>txn</code>	O	/	`LINK_WIDTH bit，在*.vh 中修改，PCIe 硬核输出 Serdes 信号差分。
<code>rxp</code>	I	/	`LINK_WIDTH bit，在*.vh 中修改，PCIe 硬核输入 Serdes 差分信号。
<code>rxn</code>	I	/	`LINK_WIDTH bit，在*.vh 中修改，PCIe 硬核输入 Serdes 差分信号。
<code>user_link_up</code>	O	1	



信号名称	方向	位宽	描述
			Data link 层链接指示信号，高有效。
user_clk_out	0	1	PCIe 硬核输出 core_clk 经 pll 处理后的输出时钟。
user_reset_out	0	1	PCIe 硬核输出 core_rst_n 经 pll 输出时钟处理后的输出时钟。
core_clk_led	0	1	PCIe Core clock 有效的指示灯
cfg_mgmt_addr	I	19	读写地址，配置空间 DW 对齐的地址。
cfg_mgmt_byte_enable	I	4	字节使能，写入数据的字节启用，其中 cfg_mgmt_byte_enable[0] 对应 cfg_mgmt_write_data[7:0]等。
cfg_mgmt_read_data	0	32	读出数据，读取的配置空间寄存器数据。
cfg_mgmt_read	I	1	读使能信号，为读取操作置位，高电平有效。
cfg_mgmt_read_write_done	0	1	读写操作完成指示信号，置位一个周期当操作完成，高电平有效。
cfg_mgmt_write_data	I	32	写数据，往配置空间寄存器写入数据。
cfg_mgmt_write	I	1	写使能信号，为写操作置位，高电平有效。
usr_irq_req	I	/	`DMA_USR_IRQ bit, 在*.vh 中修改，断言产生中断，维持中断直到握手成功。
usr_irq_ack	0	/	`DMA_USR_IRQ bit, 在*.vh 中修改，表示中断已在 PCIe 上发送，为传统中断生成两个周期 ack，为 MSI 中断生成一个周期 ack。
msi_enable	0	1	指示 MSI 中断是否是能，高有效。
msi_vector_width	0	3	表示 MSI 字段的大小 (MSI 向量的数量 分配给设备)。
s0_axis_c2h_RST	0	1	输出到*_app 模块的复位信号，C2H 引擎 RUN 下降沿后，随即会送出复位信号，高有效。
m0_axis_h2c_RST	0	1	输出到*_app 模块的复位信号，H2C 引擎 RUN 下降沿后，随即会送出复位信号，高有效
s0_axis_c2h_TREADY	0	1	此信号的断言表示 DMA 已准备好接受数据。当在同一周期内断言 s0_axis_TX_TREADY 和 s0_axis_c2h_TVALID 时，数据通过接口传输。如果 DMA 在有效信号为高时解除信号断言，则用户逻辑必须保持有效信号直到断言就绪。
s0_axis_c2h_TDATA	I	/	`DATA_WIDTH 数据宽度，PH1A100 为 64bit, PH1A400 为 128bit，将数据从用户逻辑传输到 DMA。
s0_axis_c2h_TKEEP	I	/	`KEEP_WIDTH 传输数据字节使能， s0_axis_c2h_TKEEP[0] =1' b1, s0_axis_c2h_TDATA[7:0]有效, s0_axis_c2h_TKEEP[7]=1' b0, s0_axis_c2h_TDATA[63:56]无效；当



信号名称	方向	位宽	描述
			s0_axis_c2h_tlast 没到时 s_axis_c2h_tkeep 必须全部为' hff。
s0_axis_c2h_tlast	I	1	用户逻辑断言该信号以指示 DMA 数据包的结束。
s0_axis_c2h_tvalid	I	1	每当用户逻辑在 s0_axis_c2h_tdata 上驱动有效数据时，就会断言这一点。
m0_axis_h2c_tdata	0	/	`DATA_WIDTH 数据宽度，将数据从 DMA 传输到用户逻辑
m0_axis_h2c_tkeep	0	1	接收数据字节使能，与 s0_axis_c2h_tkeep 类似。
m0_axis_h2c_tlast	0	1	DMA 的最后一拍中断言该信号，以指示数据包的结束。
m0_axis_h2c_tvalid	0	1	每当 DMA 在 m_axis_h2c_tdata 上驱动有效数据时，它都会断言这一点。
m0_axis_h2c_tready	I	1	此信号的断言表示 DMA 已准备好接受数据。当在同一周期内断言 m0_axis_h2c_tready 和 m0_axis_h2c_tvalid 时，数据通过接口传输。如果 DMA 在有效信号为高时解除信号断言，则用户逻辑必须保持有效信号直到断言就绪。
m_axil_awaddr	0	32	此信号是内存掩码的地址，从主机写入用户逻辑。
m_axil_awprot	0	3	3' d0
m_axil_awvalid	0	1	此信号的断言意味着存在对 m_axil_awaddr 上地址的有效写入请求。
m_axil_awready	I	1	主机写入地址就绪。
m_axil_wdata	0	32	主机写入数据。
m_axil_wstrb	0	4	主机写入选通。
m_axil_wvalid	0	1	主机写有效。
m_axil_wready	I	1	主机写数据就绪
m_axil_bvalid	I	1	主机响应有效
m_axil_bready	0	1	主机响应就绪。
m_axil_bresp	I	2	主机写响应。
m_axil_araddr	0	32	此信号是从用户逻辑读取到 DMA 的内存掩码地址。
m_axil_arprot	0	3	3' d0
m_axil_arvalid	0	1	此信号的断言意味着存在对 m_axil_araddr 上地址的有效读取请求。
m_axil_arready	I	1	主机读地址准备就绪。
m_axil_rdata	I	32	主机读到的数据。
m_axil_rrresp	I	2	主机读响应。
m_axil_rvalid	I	1	主机读有效。
m_axil_rready	0	1	主机响应就绪。



信号名称	方向	位宽	描述
s_axil_awaddr	I	32	此信号是从用户逻辑写入 DMA 的内存掩码地址。
s_axil_awprot	I	3	没有使用。
s_axil_awvalid	I	1	此信号的断言意味着存在对 s_axil_awaddr 上地址的有效写入请求。
s_axil_awready	O	1	从机写入地址就绪
s_axil_wdata	I	32	从机写数据
s_axil_wstrb	I	4	从机写入选通。
s_axil_wvalid	I	1	从机写有效。
s_axil_wready	O	1	从机写入数据就绪。
s_axil_bvalid	O	1	从机写响应有效。
s_axil_bready	I	1	从机写响应就绪。
s_axil_bresp	O	2	从机写响应。
s_axil_araddr	I	32	此信号是从用户逻辑读取到 DMA 的内存掩码地址。
s_axil_arprot	I	3	没有使用。
s_axil_arvalid	I	1	从机读地址有效。
s_axil_await	O	1	从机读地址准备好。
s_axil_rdata	O	32	从机读出数据。
s_axil_rresp	O	2	从机读响应。
s_axil_rvalid	O	1	从机读有效。
s_axil_rready	I	1	从机读准备就绪。
m_axi_awready	I	1	主写入地址就绪。
m_axi_awid	O	4	主写入地址 ID。
m_axi_awaddr	O	64	此信号为存储器映射写入地址 (从主机到用户逻辑)。
m_axi_awlen	O	8	主写入地址长度。
m_axi_awsize	O	3	主写入地址大小。
m_axi_awburst	O	2	主写入地址突发类型。
m_axi_awprot	O	3	3'h0
m_axi_awvalid	O	1	此信号断言有效即表示存在发射到 m_axib_araddr 上的地址的有效写入请求。
m_axi_awlock	O	1	1'b0
m_axi_awcache	O	4	4'h0
m_axi_wdata	O	/	'DATA_WIDTH 数据宽度, 主写入数据。
m_axi_wstrb	O	/	'KEEP_WIDTH, 主写入选通, 与 s0_axis_c2h_tkeep 类似
m_axi_wready	I	1	主写入就绪。
m_axi_wlast	O	1	主写入结束。



信号名称	方向	位宽	描述
m_axi_wvalid	0	1	主写入有效。
m_axi_bid	1	4	主响应 ID。
m_axi_bresp	1	2	主写入响应。
m_axi_bvalid	1	1	主写入响应有效。
m_axi_bready	0	1	主响应就绪。
m_axi_arready	1	1	主读取地址就绪。
m_axi_arid	0	4	主读取地址 ID。
m_axi_araddr	0	64	此信号为存储器映射读取地址（从主机到用户逻辑）。
m_axi_arlen	0	8	主读取地址长度。
m_axi_arsize	0	3	主读取地址大小。
m_axi_arburst	0	2	主读取突发类型。
m_axi_arprot	0	3	3'h0
m_axi_arvalid	0	1	此信号断言有效即表示存在发射到 m_axib_araddr 上的地址的有效读取请求。
m_axi_arlock	0	1	1'b0
m_axi_arcache	0	4	4'h0
m_axi_rid	1	4	主读取 ID。
m_axi_rdata	1	/	`DATA_WIDTH 数据宽度，主读取数据。
m_axi_rrresp	1	2	主读取响应。
m_axi_rlast	1	1	主读取结束。
m_axi_rvalid	1	1	主读取有效。
m_axi_rready	0	1	主读取就绪。
m_axib_awready	1	1	主写入地址就绪。
m_axib_awid	0	4	主写入地址 ID。
m_axib_awaddr	0	64	此信号为存储器映射写入地址（从主机到用户逻辑）。
m_axib_awlen	0	8	主写入地址长度。
m_axib_awsize	0	3	主写入地址大小。
m_axib_awburst	0	2	主写入地址突发类型。
m_axib_awprot	0	3	3'h0
m_axib_awvalid	0	1	此信号断言有效即表示存在发射到 m_axib_araddr 上的地址的有效写入请求。
m_axib_awlock	0	1	1'b0
m_axib_awcache	0	4	4'h0
m_axib_wdata	0	/	`DATA_WIDTH 数据宽度，主写入数据。
m_axib_wstrb	0	/	`KEEP_WIDTH, 主写入选通，与 s0_axis_c2h_tkeep 类似



信号名称	方向	位宽	描述
m_axib_wready	I	1	主写入就绪。
m_axib_wlast	O	1	主写入结束。
m_axib_wvalid	O	1	主写入有效。
m_axib_bid	I	4	主响应 ID。
m_axib_bresp	I	2	主写入响应。
m_axib_bvalid	I	1	主写入响应有效。
m_axib_bready	O	1	主响应就绪。
m_axi_arready	I	1	主读取地址就绪。
m_axib_arid	O	4	主读取地址 ID。
m_axib_araddr	O	64	此信号为存储器映射读取地址（从主机到用户逻辑）。
m_axib_arlen	O	8	主读取地址长度。
m_axib_arsize	O	3	主读取地址大小。
m_axib_arburst	O	2	主读取突发类型。
m_axib_arprot	O	3	3'h0
m_axib_arvalid	O	1	此信号断言有效即表示存在发射到 m_axib_araddr 上的地址的有效读取请求。
m_axib_arlock	O	1	1'b0
m_axib_arcache	O	4	4'h0
m_axib_rid	I	4	主读取 ID。
m_axib_rdata	I	/	`DATA_WIDTH 数据宽度，主读取数据。
m_axib_rrresp	I	2	主读取响应。
m_axib_rlast	I	1	主读取结束。
m_axib_rvalid	I	1	主读取有效。
m_axib_rready	O	1	主读取就绪。

4.2 顶层模块时序

上文讲到 `sgdma_subsys` 主要包含 5 类总线，其中的 AXI4 系列总线为 AMBA 提供的标准总线。主要介绍私有的 CFG_MGMT 总线和 IRQ 总线，其中的 CFG_MGMT 总线与 PCIe 硬核内部的 DBI 总线一致，完成 PCIe 内存空间的配置，配置时序如下图 4-2 所示。图中的 `cfg_addr` 相当于 DBI 总线的地址，`cfg_mgmt_write_data` 和 `cfg_mgmt_read_data` 分别为 DBI 总线的写读数据，`cfg_mgmt_read_done` 相当于 DBI 总线的 ack 信号。

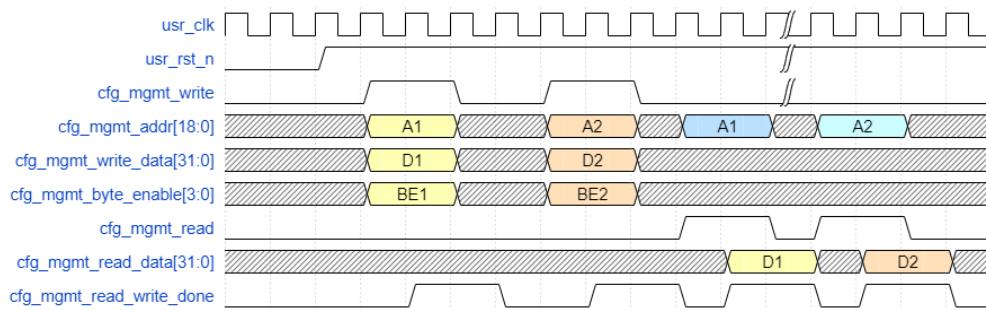


图 4-2 CFG_MGMT 总线时序

接着 IRQ 总线，它的时序如下图 4-3 所示，它对接的是 PCIe 硬核的中断总线，如 MSI 或 MSI-X 或 Legacy，由于当前设计只完成了 MSI 总线的调试，且 Linux 内核仅支持 MSI 的一个中断矢量，所以下图的中端矢量只能为 1。

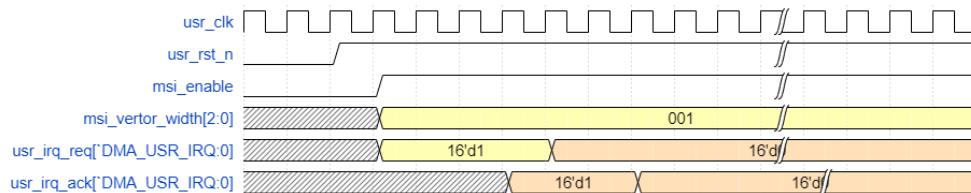


图 4-3 USR_IRQ 总线时序

4.3 H2C 功能流程

SGDMA 工作分为两个流程，H2C (Host to Card) 和 C2H (Card to Host)。H2C 的工作流程如下图 4-4 所示，初始时驱动程序通过 BAR0 完成 DMA 寄存器的配置，数据发送完成后通过 MSI 提交中断。图中黄色为驱动程序，橙色为应用程序，绿色为逻辑程序。

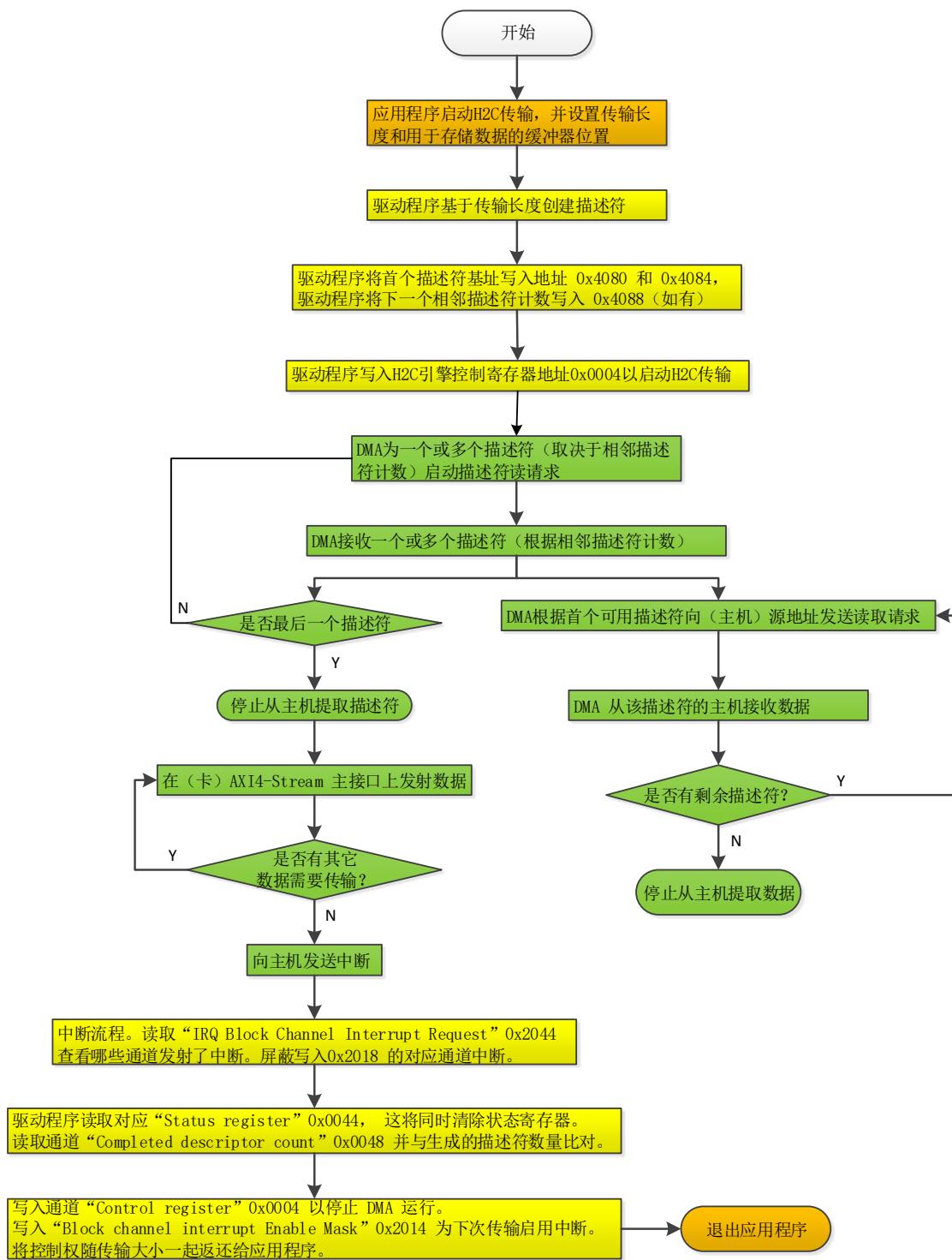


图 4-4 H2C 引擎工作流程图

4.4 C2H 功能流程

C2H 的流程与 H2C 的类似，主要的不同点为 C2H 的数据来自于用户，H2C 接收主机的数据，它的工作流程图如下图 4-5 所示。

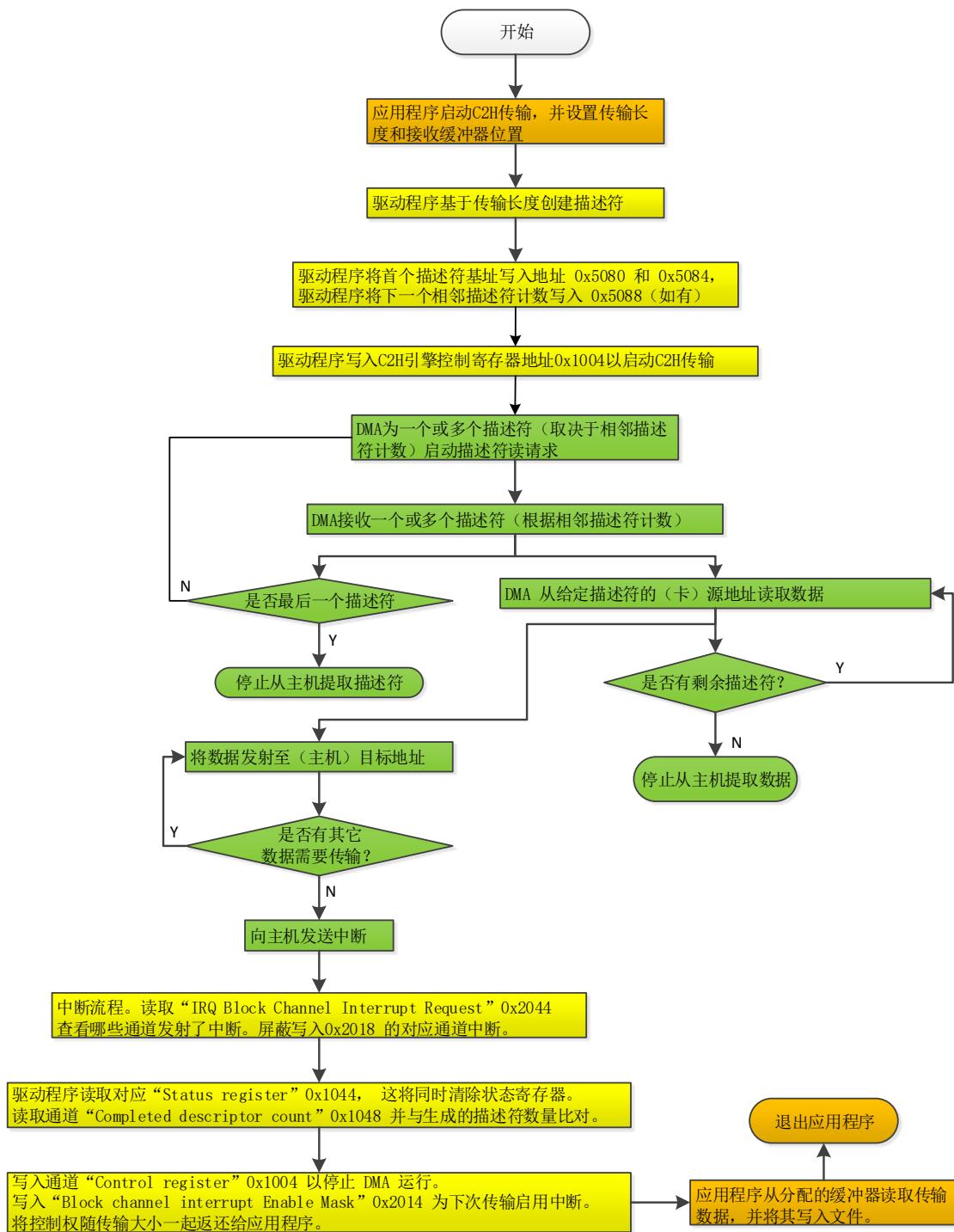


图 4-5 C2H 引擎工作流程图

4.5 寄存器空间

通过将读取请求或写入请求掩码写到基址寄存器(BAR0)，即可从主机访问 DMA 寄存器或用户应用程序寄存器。BAR0 寄存器大小默认为 1Mbyte 的空间，512K 用于 DMA 引擎，另一半预留给用户使用。PCIe 到 DMA 控制器核的传输事务用到了 ELBI 总线，该总线仅支持 32 位地址空间和 32bit 读写请求。



PCIe 到 DMA 地址格式如下表 4-2 所示, bit[15:2]对应的目标目前仅支持 MSI, 后续会增加 MSI-X 和 Legacy Interrupt。

表 4-2 PCIe 到 DMA 地址格式

位索引	描述
31:16	保留
15:12 (目标)	DMA 中的目标子模块 4'h0: H2C 通道 4'h1: C2H 通道 4'h2: IRQ 块 4'h3: Configure 暂未支持 4'h4: H2C SGDMA 4'h5: C2H SGDMA 4'h6: SGDMA 公用 4'h7 保留 4'h8: MSI-X 4'h9~4'h15: 保留
11:8 (通道)	该字段仅适用于 H2C 通道、C2H 通道、H2C SGDMA 和 C2H SGDMA 目标。该字段用于表示正在对哪个引擎进行寻址以供这些目标使用。对于所有其它目标, 该字段必须为 0。
7:0 (字节偏移)	将在目标内访问的寄存器的字节地址。位 [1:0] 必须为 0。

DMA 引擎内部有许多寄存器, 这些寄存器的属性的英文简写如下表 4-3 所示, 部分寄存器可通过不同属性来访问, 在此类情况下, 针对每个属性提供不同的寄存器偏移。未定义的 bit 位和地址空间都将保留。

表 4-3 配置寄存器属性定义

属性	描述
RV	保留
RW	读取/写入
RC	读取时清除
W1C	写 1 时清除
W1S	写 1 时设置
RO	只读
WO	只写

H2C 通道寄存器 (0x0), 本节描述了 H2C 通道寄存器空间。

H2C 通道标识符 (0x00)



表 4-4 H2C 通道标识符 (0x00)

位索引	默认值	访问类型	描述
31:20	12'h1fc	R0	子系统标识符
19:16	4'h0	R0	H2C 通道目标
15	1'b0	R0	数据流传输 1: AXI4-Stream 接口 0: AXI4 Memory Mapped 接口
14:12	0	R0	保留
11:8	可变	R0	通道 ID 目标 [3:0]
7:0	8'h04	R0	版本

H2C 通道控制 (0x04)

表 4-5 H2C 通道控制

位索引	默认值	访问类型	描述
31:28	/	/	保留
27	1'b0	RW	在 AXI-Stream 模式下禁用 C2H 的置位写回信息时，启用默认写回。
26	1'b0	RW	pollmode_wb_enable 轮询模式写回使能位。 当该位置位后，如果描述符完成时“已完成 (Completed)”位已置位，那么 DMA 就会将已完成的描述符计数写回。
25	1'b0	RW	non_inc_mode 暂未支持。
24	/	RW	保留
23:19	5'h0	RW	ie_desc_error 全部设置为 1 值 (0x1F) 即可启用 Status.Desc_error 记录功能，并在检测到错误时停止引擎。
18:14	5'h0	RW	ie_write_error 全部设置为 1 值 (0x1F) 即可启用 Status.Write_error 记录功能，并在检测到错误时停止引擎。
13:9	5'h0	RW	ie_read_error 全部设置为 1 值 (0x1F) 即可启用 Status.Read_error 记录功能，并在检测到错误时停止引擎。



位索引	默认值	访问类型	描述
8:7	/	RW	保留
6	1'b0	RW	ie_idle_stopped 设为 1 即可启用 Status. Idle_stopped 记录功能
5	1'b0	RW	ie_invalid_length 设为 1 即可启用 Status. Invalid_length 记录功能
4	1'b0	RW	ie_magic_stopped 设为 1 即可启用 Status. Magic_stopped 记录功能
3	1'b0	RW	ie_align_mismatch 设为 1 即可启用 Status. Align_mismatch 记录功能
2	1'b0	RW	ie_descriptor_completed 设为 1 即可启用 Status. Descriptor_completed 记录功能
1	1'b0	RW	ie_descriptor_stopped 设为 1 即可启用 Status. Descriptor_stopped 记录功能
0	1'b0	RW	运行 设为 1 即可启动 SGDMA 引擎。复位为 0 即可停止传输，如果引擎繁忙，那么它会完成当前描述符。

注释：1. `ie_*` register 位为中断使能位。当这些位已置位并满足对应条件后，将在 H2C 通道状态 (0x40) 中记录状态。正确设置中断掩码（根据 H2C Channel Interrupt Enable Mask (0x90)）后，将生成中断。

H2C 通道控制 (0x08)

表 4-6 H2C 通道控制

位索引	默认值	访问类型	描述
31:28	/	/	保留
27:0	/	W1S	Control 位元描述与 H2C Channel Control (0x04) 中的位元描述相同

H2C 通道控制 (0x0C)



表 4-7 H2C 通道控制

位索引	默认值	访问类型	描述
27:0	/	W1C	Control 位元描述与 H2C Channel Control (0x04) 中的位元描述相同

H2C 通道状态 (0x40)

表 4-8 H2C 通道状态

位索引	默认值	访问类型	描述
31:24	/	/	保留
23:19	5'h0	RW1C	descr_error[4:0] 复位 (0) 条件为: 在控制寄存器的运行位进行置位时复位。4: 意外完成 3: 报头 EP 2: 奇偶校验错误 1: 完成方异常中止 0: 请求不受支持
18:14	5'h0	RW1C	write_error[4:0] 复位 (0) 条件为: 在控制寄存器的运行位进行置位时复位。位元位置: 4-2: 保留 1: 从接口错误 0: 解码器错误
13:9	5'h0	RW1C	read_error[4:0] 复位 (0) 条件为: 在控制寄存器的运行位进行置位时复位。 位元位置 4: 意外完成 3: 报头 EP 2: 奇偶校验错误 1: 完成方异常中止 0: 请求不受支持
8:7	/		保留
6	1'b0	RW1C	idle_stopped 复位 (0) 条件为: 在控制寄存器的运行位进行置位时复位。置位条件为: 在控制寄存器 ie_idle_stopped 位已置位的前提下, 将控制寄存器运行位复位之后, 引擎



位索引	默认值	访问类型	描述
			处于空闲状态时进行置位。
5	1'b0	RW1C	invalid_length 复位条件为：在控制寄存器运行位进行置位时复位。置位条件为：当描述符长度并非 AXI4-Stream 通道的数据宽度的倍数，并且控制寄存器的 ie_invalid_length 位已置位时进行置位。
4	1'b0	RW1C	magic_stopped 复位条件为：在控制寄存器运行位进行置位时复位。置位条件为：在控制寄存器 ie_magic_stopped 位已置位的前提下，当引擎遇到含无效 magic 的描述符并停止时进行置位。
3	1'b0	RW1C	align_mismatch 描述符上的源地址与目标地址彼此未正确对齐。
2	1'b0	RW1C	descriptor_completed 复位条件为：在控制寄存器运行位进行置位时复位。置位条件为：在控制寄存器 ie_descriptor_stopped 位已置位的前提下，当引擎已完成描述符并且 COMPLETE 位已置位时进行置位
1	1'b0	RW1C	descriptor_stopped 复位条件为：在控制寄存器运行位进行置位时复位。置位条件为：在控制寄存器 ie_descriptor_stopped 位已置位的前提下，当引擎已完成描述符并且 STOP 位已置位时进行置位。
0	1'b0	RW1C	busy 置位条件为：当 SGDMA 引擎处于繁忙状态时置位。当该引擎处于空闲状态为零(0)。

H2C 通道状态 (0x44)

表 4-9 H2C 通道状态

位索引	默认值	访问类型	描述
23:1	/	RC	状态



位索引	默认值	访问类型	描述
			读取时清除。位元描述与 H2C Channel Status (0x40) 中的位元描述相同。位 0 无法清除。

H2C 通道完成描述符计数 (0x48)

表 4-10 H2C 通道完成描述符计数

位索引	默认值	访问类型	描述
31:0	32'h0	R0	Compl_descriptor_count 表示完成列表中每个描述符后，引擎完成的描述符更新数量。在控制寄存器的上升沿复位为 0，该位元为运行位 (H2C Channel Control (0x04))。

H2C 轮询模式回写地址低 32 位 (0x88)

表 4-11 轮询模式回写地址低 32 位

位索引	默认值	访问类型	描述
31:0	0x0	RW	pollmode_lo_wb_addr[31:0] 轮询模式写回地址的下 32 位。

H2C 轮询模式回写地址高 32 位 (0x8C)

表 4-12 H2C 轮询模式回写地址高 32 位

位索引	默认值	访问类型	描述
31:0	0x0	RW	pollmode_hi_wb_addr[63:32] 轮询模式写回地址的上 32 位。

H2C 通道中断使能掩码 (0x90)

表 4-13 H2C 通道中断使能掩码

位索引	默认值	访问类型	描述
23:19	5'h0	RW	im_desc_error[4:0] 设为 1 则会在记录对应的状态寄存器 read_error 位时发生中断。
18:14	5'h0	RW	im_write_error[4:0] 设为 1 则会记录在对应的状态寄存器 write_error 位时发生中断。
13:9	5'h0	RW	im_read_error[4:0] 设为 1 则会在记录对应的状态寄存器



位索引	默认值	访问类型	描述
			<code>read_error</code> 位时发生中断。
8:7	/	/	保留
6	1'b0	RW	<code>im_idle_stopped</code> 设为 1 则会在记录状态寄存器 <code>idle_stopped</code> 位时发生中断。
5	1'b0	RW	<code>im_invalid_length</code> 设为 1 则会在记录状态寄存器 <code>magic_stopped</code> 位时发生中断。
4	1'b0	RW	<code>im_magic_stopped</code> 设为 1 则会在记录状态寄存器 <code>magic_stopped</code> 位时发生中断。
3	1'b0	RW	<code>im_align_mismatch</code> 设为 1 则会在记录状态寄存器 <code>descriptor_mismatch</code> 位时发生中断。
2	1'b0	RW	<code>im_descriptor_completed</code> 设为 1 则会在记录状态寄存器 <code>descriptor_completed</code> 位时发生中断。
1	1'b0	RW	<code>im_descriptor_stopped</code> 设为 1 则会在记录状态寄存器 <code>descriptor_stopped</code> 位时发生中断。

H2C 通道中断使能掩码 (0x94)

表 4-14 H2C 通道中断使能掩码

位索引	默认值	访问类型	描述
/	/	W1S	中断使能掩码

H2C 通道中断使能掩码 (0x98)

表 4-15 H2C 通道中断使能掩码

位索引	默认值	访问类型	描述
/	/	W1C	中断使能掩码

H2C 通道性能模拟控制 (0xC0)

表 4-16 H2C 通道性能模拟控制

位索引	默认值	访问类型	描述
2	1'b0	RW	运行 设为 1 以运行性能计数器。在控制寄存器 运行位已置位后，即可启动计数器。设为



位索引	默认值	访问类型	描述
			0 以停止性能计数器
1	1'b0	WO	清除 写入 1 即可清除性能计数器
0	1'b0	RW	自动 当含停止位的描述符完成后，即自动停止性能计数器。在控制寄存器运行位已置位后，即自动清除性能计数器。要启动计数器，则仍需向性能监控寄存器运行位写入 1。

H2C 通道性能周期计数 (0xC4)

表 4-17 H2C 通道性能周期计数

位索引	默认值	访问类型	描述
31:0	32'h0	R0	pmon_cyc_count[31:0] 运行时，按每个时钟递增。请参阅 Performance Monitor Control 寄存器 (0xC0) 的“清除”位和“自动”位以了解有关清除的信息。

H2C 通道性能周期计数 (0xC8)

表 4-18 H2C 通道性能周期计数

位索引	默认值	访问类型	描述
16	1'b0	R0	pmon_cyc_count_maxed 已达周期计数最大值
9:0	10'h0	R0	pmon_cyc_count[41:32] 运行时，按每个时钟递增。请参阅 Performance Monitor Control 寄存器 (0xC0) 的“清除”位和“自动”位以了解有关清除的信息。

H2C 通道性能数据量计数 (0xCC)

表 4-19 H2C 通道性能数据量计数

位索引	默认值	访问类型	描述
31:0	32'h0	R0	pmon_dat_count[31:0] 运行时，随每个有效读取数据节拍递增。 请参阅 Performance Monitor Control 寄存器 (0xC0) 的“清除”位和“自动”



位索引	默认值	访问类型	描述
			位以了解有关清除的信息。

H2C 通道性能数据量计数 (0xD0)

表 4-20 H2C 通道性能数据量计数

位索引	默认值	访问类型	描述
16	1'b0	R0	pmon_dat_count_maxed 已达数据计数最大值
15:10	/	/	保留
9:0	10'h0	R0	pmon_dat_count[41:32] 运行时，随每个有效读取数据节拍递增。 请参阅 Performance Monitor Control 寄存器 (0xC0) 的“清除”位和“自动” 位以了解有关清除的信息。

C2H 通道寄存器 (0x1), 本节描述了 C2H 通道寄存器空间。

C2H 通道标识符 (0x00)

表 4-21 C2H 通道标识符

位索引	默认值	访问类型	描述
31:20	12'h1fc	R0	子系统标识符
19:16	4'h0	R0	C2H 通道目标
15	1'b0	R0	数据流传输 1: AXI4-Stream 接口 0: AXI4 Memory Mapped 接口
14:12	0	R0	保留
11:8	可变	R0	通道 ID 目标 [3:0]
7:0	8'h04	R0	版本

C2H 通道控制 (0x04)

表 4-22 C2H 通道控制

位索引	默认值	访问类型	描述
31:28	/	/	保留
27	1'b0	RW	在 C2H AXI4-Stream 模式下禁用元数据 写回，如果此通道配置为使用 AXI Memory Mapped，则无任何影响。
26	0x0	RW	pollmode_wb_enable 轮询模式写回使能位。当该位置位后，如 果描述符完成时“已完成 (Completed)”



位索引	默认值	访问类型	描述
			位已置位，那么 DMA 就会将已完成的描述符计数写回。
25	1'b0	RW	non_inc_mode 暂未支持。
24	/	RW	保留
23:19	5'h0	RW	ie_desc_error 全部设置为 1 值 (0x1F) 即可启用 Status.Desc_error 记录功能，并在检测到错误时停止引擎。
18:14	5'h0	RW	ie_write_error 全部设置为 1 值 (0x1F) 即可启用 Status.Write_error 记录功能，并在检测到错误时停止引擎。
13:9	5'h0	RW	ie_read_error 全部设置为 1 值 (0x1F) 即可启用 Status.Read_error 记录功能，并在检测到错误时停止引擎。
8:7	/	RW	保留
6	1'b0	RW	ie_idle_stopped 设为 1 即可启用 Status.Idle_stopped 记录功能
5	1'b0	RW	ie_invalid_length 设为 1 即可启用 Status.Invalid_length 记录功能
4	1'b0	RW	ie_magic_stopped 设为 1 即可启用 Status.Magic_stopped 记录功能
3	1'b0	RW	ie_align_mismatch 设为 1 即可启用 Status.Align_mismatch 记录功能
2	1'b0	RW	ie_descriptor_completed 设为 1 即可启用 Status.Descriptor_completed 记录功能
1	1'b0	RW	ie_descriptor_stopped 设为 1 即可启用 Status.Descriptor_stopped 记录功能
0	1'b0	RW	运行



位索引	默认值	访问类型	描述
			设为 1 即可启动 SGDMA 引擎。复位为 0 即可停止传输，如果引擎繁忙，那么它会完成当前描述符。

注意：`ie_* register` 位为中断使能位。当这些位已置位并满足对应条件后，将在 C2H Channel Status (0x40) 中记录状态。正确设置中断掩码（根据 C2H Channel Interrupt Enable Mask (0x90)）后，将生成中断。

C2H 通道控制 (0x08)

表 4-23 C2H 通道控制

位索引	默认值	访问类型	描述
		W1S	Control 位元描述与 C2H Channel Control (0x04) 中的位元描述相同

C2H 通道控制 (0x0C)

表 4-24 C2H 通道控制

位索引	默认值	访问类型	描述
		W1C	Control 位元描述与 C2H Channel Control (0x04) 中的位元描述相同

C2H 通道状态 (0x40)

表 4-25 C2H 通道状态

位索引	默认值	访问类型	描述
31:24	/	/	保留
23:19	5'h0	RW1C	<code>descr_error[4:0]</code> 复位 (0) 条件为：在控制寄存器的运行位进行置位时复位。 4: 意外完成 3: 报头 EP 2: 奇偶校验错误 1: 完成方异常中止 0: 请求不受支持
18:14		RW1C	<code>write_error[4:0]</code> 复位 (0) 条件为：在控制寄存器的运行位进行置位时复位。 位元位置：



位索引	默认值	访问类型	描述
			4-2: 保留 1: 从接口错误 0: 解码器错误
13:9	5'h0	RW1C	<code>read_error[4:0]</code> 复位 (0) 条件为: 在控制寄存器的运行位进行置位时复位。 位元位置 4: 意外完成 3: 报头 EP 2: 奇偶校验错误 1: 完成方异常中止 0: 请求不受支持
8:7			保留
6	1'b0	RW1C	<code>idle_stopped</code> 复位 (0) 条件为: 在控制寄存器的运行位进行置位时复位。置位条件为: 在控制寄存器 <code>ie_idle_stopped</code> 位已置位的前提下, 将控制寄存器运行位复位之后, 引擎处于空闲状态时进行置位。
5	1'b0	RW1C	<code>invalid_length</code> 复位条件为: 在控制寄存器运行位进行置位时复位。置位条件为: 当描述符长度并非 AXI4-Stream 通道的数据宽度的倍数, 并且控制寄存器的 <code>ie_invalid_length</code> 位已置位时进行置位。
4	1'b0	RW1C	<code>magic_stopped</code> 复位条件为: 在控制寄存器运行位进行置位时复位。置位条件为: 在控制寄存器 <code>ie_magic_stopped</code> 位已置位的前提下, 当引擎遇到含无效 <code>magic</code> 的描述符并停止时进行置位。
3	1'b0	RW1C	<code>align_mismatch</code> 描述符上的源地址与目标地址彼此未正确对齐。
2	1'b0	RW1C	<code>descriptor_completed</code> 复位条件为: 在控制寄存器运行位进行置位时复位。置位条件为: 在控制寄存器



位索引	默认值	访问类型	描述
			<code>ie_descriptor_stopped</code> 位已置位的前提下，当引擎已完成描述符并且 <code>COMPLETE</code> 位已置位时进行置位
1	<code>1'b0</code>	RW1C	<code>descriptor_stopped</code> 复位条件为：在控制寄存器运行位进行置位时复位。置位条件为：在控制寄存器 <code>ie_descriptor_stopped</code> 位已置位的前提下，当引擎已完成描述符并且 <code>STOP</code> 位已置位时进行置位。
0	<code>1'b0</code>	RW1C	<code>Busy</code> 置位条件为：当 SGDMA 引擎处于繁忙状态时置位。当该引擎处于空闲状态为零（0）。

C2H 通道状态 (0x44)

表 4-26 C2H 通道状态

位索引	默认值	访问类型	描述
23:1		RC	状态 读取时清除。位元描述与 C2H Channel Status (0x40) 中的位元描述相同。位 0 无法清除。

C2H 完成描述符计数 (0x48)

表 4-27 C2H 完成描述符计数

位索引	默认值	访问类型	描述
31:0	<code>32'h0</code>	R0	<code>Compl_descriptor_count</code> 表示完成列表中每个描述符后，引擎完成的描述符更新数量。在控制寄存器的上升沿复位为 0，该位元为运行位 (C2H Channel Control (0x04))。

C2H 轮询模式回写地址低 32bit (0x88)

表 4-28 C2H 轮询模式回写地址低 32bit

位索引	默认值	访问类型	描述
31:0	<code>0x0</code>	RW	<code>pollmode_lo_wb_addr[31:0]</code> 轮询模式写回地址的下 32 位。

C2H 轮询模式回写地址高 32bit (0x8C)



表 4-29 C2H 轮询模式回写地址高 32bit

位索引	默认值	访问类型	描述
31:0	0x0	RW	pol mode_hi_wb_addr[63:32] 轮询模式写回地址的上 32 位。

C2H通道中断使能掩码 (0x90)

表 4-30 C2H 通道中断使能掩码

位索引	默认值	访问类型	描述
23:19	5'h0	RW	im_desc_error[4:0] 设为 1 则会在记录对应的状态寄存器 read_error 位时发生中断。
13:9	5'h0	RW	im_read_error[4:0] 设为 1 则会在记录对应的状态寄存器 read_error 位时发生中断。
8:7	/	/	保留
6	1'b0	RW	im_idle_stopped 设为 1 则会在记录状态寄存器 idle_stopped 位时发生中断。
5	1'b0	RW	im_invalid_length 设为 1 则会在记录状态寄存器 magic_stopped 位时发生中断。
4	1'b0	RW	im_magic_stopped 设为 1 则会在记录状态寄存器 magic_stopped 位时发生中断。
3	1'b0	RW	im_align_mismatch 设为 1 则会在记录状态寄存器 descriptor_mismatch 位时发生中断。
2	1'b0	RW	im_descriptor_completed 设为 1 则会在记录状态寄存器 descriptor_completed 位时发生中断。
1	1'b0	RW	im_descriptor_stopped 设为 1 则会在记录状态寄存器 descriptor_stopped 位时发生中断。
0	/	/	保留

C2H 通道中断使能掩码 (0x94)



表 4-31 C2H 通道中断使能掩码

位索引	默认值	访问类型	描述
		W1S	中断使能掩码 位元描述与 C2H Channel Interrupt Enable Mask (0x90) 中的位元描述相同

C2H 通道中断使能掩码 (0x98)

表 4-32 C2H 通道中断使能掩码

位索引	默认值	访问类型	描述
		W1C	中断使能掩码 位元描述与 C2H Channel Interrupt Enable Mask (0x90) 中的位元描述相同

C2H 通道性能模拟控制 (0xC0)

表 4-33 C2H 通道性能模拟控制

位索引	默认值	访问类型	描述
2	1'b0	RW	运行 设为 1 以装备性能计数器。在控制寄存器运行位已置位后，即可启动计数器。 设为 0 以停止性能计数器
1	1'b0	RW	清除 写入 1 即可清除性能计数器
0	1'b0	RW	自动 当含停止位的描述符完成后，即自动停止性能计数器。在控制寄存器运行位已置位后，即自动清除性能计数器。要启动计数器，则仍需向性能监控寄存器运行位写入 1。

C2H 通道性能周期计数 (0xC4)

表 4-34 C2H 通道性能周期计数

位索引	默认值	访问类型	描述
31:0	32'h0	R0	pmon_cyc_count[31:0] 运行时，按每个时钟递增。请参阅 Performance Monitor Control 寄存器 (0xC0) 的“清除”位和“自动”位以了



位索引	默认值	访问类型	描述
			解有关清除的信息。

C2H 通道性能周期计数 (0xC8)

表 4-35 C2H 通道性能周期计数

位索引	默认值	访问类型	描述
16	1'b0	R0	pmon_cyc_count_maxed 已达周期计数最大值
15:10	/	/	保留
9:0	10'h0	R0	pmon_cyc_count[41:32] 运行时, 按每个时钟递增。请参阅 Performance Monitor Control 寄存器 (0xC0) 的“清除”位和“自动”位以了 解有关清除的信息。

C2H 通道性能数据计数 (0xCC)

表 4-36 C2H 通道性能数据计数

位索引	默认值	访问类型	描述
31:0	32'h0	R0	pmon_dat_count[31:0] 运行时, 随每个有效读取数据节拍递增。 请参阅 Performance Monitor Control 寄存器 (0xC0) 的“清除”位和“自动” 位以了解有关清除的信息。

C2H 通道性能数据计数 (0xD0)

表 4-37 C2H 通道性能数据计数

位索引	默认值	访问类型	描述
16	1'b0	R0	pmon_dat_count_maxed 已达数据计数最大值
15:10	/	/	保留
9:0	10'h0	R0	pmon_dat_count[41:32] 运行时, 随每个有效读取数据节拍递增。 请参阅 Performance Monitor Control 寄存器 (0xC0) 的“清除”位和“自动” 位以了解有关清除的信息。

IRQ 块寄存器 (0x2), 本节描述了 IRQ 块寄存器。在 AXI Bridge 模式下选中“**MSI-X Capabilities**”时, 来自 BAR0 的 64KB 地址空间将保留以供 MSI-X 表格使用。默认情况下, 寄存器空间在 BAR0 内分配。仅当选中“**MSI-X Capabilities**”选项时, 该选项才可用。对于其它中断选项, 则不分配任何空



间。

IRQ 模块标识符 (0x00)

表 4-38 IRQ 模块标识符

位索引	默认值	访问类型	描述
31:20	12'h1fc	R0	子系统标识符
19:16	4'h2	R0	IRQ 标识符
15:8	8'h0	R0	保留
7:0	8'h04	R0	版本 8'h01: 2022.7 8'h02: 2023.2 8'h03: 2023.4 8'h04: 从 2023.5 到最新版本

IRQ 模块用户中断使能掩码 (0x04)

表 4-39 IRQ 模块用户中断使能掩码

位索引	默认值	访问类型	描述
[NUM_USR_INT-1:0]	'h0	RW	user_int_enmask 用户中断使能掩码 0: 当用户中断源断言有效时, 阻止生成中断。 1: 在用户中断源的上升沿生成中断。如果“使能掩码 (Enable Mask)”已置位, 并且源也已置位, 那么同样会生成用户中断。

IRQ 模块用户中断使能掩码 (0x08)

表 4-40 IRQ 模块用户中断使能掩码

位索引	默认值	访问类型	描述
		W1S	user_int_enmask 位元描述与 IRQ Block 用户 Interrupt Enable Mask (0x04) 中的位元描述相同。

IRQ 模块用户中断使能掩码 (0x0C)

表 4-41 IRQ 模块用户中断使能掩码

位索引	默认值	访问类型	描述
		W1C	user_int_enmask 位元描述与 IRQ Block 用户 Interrupt



位索引	默认值	访问类型	描述
			Enable Mask (0x04) 中的位元描述相同。

IRQ 模块通道中断使能掩码 (0x10)

表 4-42 IRQ 模块通道中断使能掩码

位索引	默认值	访问类型	描述
[NUM_CHNL-1:0]	h0	RW	<p>channel_int_enmask 引擎中断使能掩码。每个读取或写入引擎对应 1 个位。 0: 中断源断言有效时，阻止生成中断。 H2C 位元的位置始终从位 0 开始。C2H 位元的位置为高于最后一个 H2C 索引的索引，因此它取决于 NUM_H2C_CHNL 参数。 1: 在中断源的上升沿生成中断。如果 enmask 位已置位并且源已置位，那么同样会生成中断。</p>

IRQ 模块通道中断使能掩码 (0x14)

表 4-43 IRQ 模块通道中断使能掩码

位索引	默认值	访问类型	描述
		W1S	<p>channel_int_enmask 引擎中断使能掩码。每个读取或写入引擎对应 1 个位。 0: 中断源断言有效时，阻止生成中断。 H2C 位元的位置始终从位 0 开始。C2H 位元的位置为高于最后一个 H2C 索引的索引，因此它取决于 NUM_H2C_CHNL 参数。 1: 在中断源的上升沿生成中断。如果 enmask 位已置位并且源已置位，那么同样会生成中断。</p>

IRQ 模块通道中断使能掩码 (0x18)

表 4-44 IRQ 模块通道中断使能掩码

位索引	默认值	访问类型	描述
		W1C	<p>channel_int_enmask 引擎中断使能掩码。每个读取或写入引擎</p>



位索引	默认值	访问类型	描述
			<p>对应 1 个位。</p> <p>0：中断源断言有效时，阻止生成中断。</p> <p>H2C 位元的位置始终从位 0 开始。C2H 位元的位置为高于最后一个 H2C 索引的索引，因此它取决于 NUM_H2C_CHNL 参数。</p> <p>1：在中断源的上升沿生成中断。如果 enmask 位已置位并且源已置位，那么同样会生成中断。</p>

下图显示了 H2C 位元与 C2H 位元的打包方式。

表 4-45 打包 H2C 和 C2H

位	7	6	5	4	3	2	1	0
启用 4 个 H2C 和 4 个 C2H	C2H_3	C2H_2	C2H_1	C2H_0	H2C_3	H2C_2	H2C_1	H2C_0
启用 3 个 H2C 和 3 个 C2H			C2H_2	C2H_1	C2H_0	H2C_2	H2C_1	H2C_0
启用 1 个 H2C 和 1 个 C2H							C2H_0	H2C_0

IRQ 模块用户中断请求 (0x40)

表 4-46 IRQ 模块用户中断请求

位索引	默认值	访问类型	描述
[NUM_USR_INT-1:0]	'h0	R0	<p>user_int_req</p> <p>用户中断请求此寄存器反映仅当中断源与使能掩码寄存器同时断言有效时，此中断位才会断言有效。</p>

IRQ 模块通道中断请求 (0x44)



表 4-47 IRQ 模块通道中断请求

位索引	默认值	访问类型	描述
[NUM_CHNL-1:0]	'h0	R0	engine_int_req 引擎中断请求。每个读取或写入引擎对应 1 个位。此寄存器反映仅当中断源与使能掩码寄存器同时断言有效时，此中断位才会断言有效。H2C 位元的位置始终从位 0 开始。C2H 位元的位置为高于最后一个 H2C 索引的索引，因此它取决于 NUM_H2C_CHNL 参数。上图显示了 H2C 位元与 C2H 位元的打包方式。

IRQ 模块用户中断挂起 (0x48)

表 4-48 IRQ 模块用户中断挂起

位索引	默认值	访问类型	描述
[NUM_USR_INT-1:0]	'h0	R0	user_int_pend 用户中断暂挂。此寄存器表示存在暂挂事件。通过移除位于源组件上的事件原因条件即可清除暂挂事件。

IRQ 模块通道中断挂起 (0x4C)

表 4-49 IRQ 模块通道中断挂起

位索引	默认值	访问类型	描述
[NUM_CHNL-1:0]	'h0	R0	engine_int_pend 引擎中断暂挂。每个读取或写入引擎对应 1 个位。此寄存器表示存在暂挂事件。通过移除位于源组件上的事件原因条件即可清除暂挂事件。H2C 位元的位置始终从位 0 开始。C2H 位元的位置为高于最后一个 H2C 索引的索引，因此它取决于 NUM_H2C_CHNL 参数。上图显示了 H2C 位元与 C2H 位元的打包方式。

IRQ 模块用户矢量号 (0x80)，如果启用 MSI，那么此寄存器会指定 MSI 或 MSI 的 MSI-X 矢量编号。在传统中断中，每个字段仅限 2 个 LSB 用于映射到 INTA、B、C 或 D。



表 4-50 IRQ 模块用户矢量号

位索引	默认值	访问类型	描述
28:24	5'h0	R0	矢量 3 矢量编号, 当用户 IRQ_usr_irq_req[3] 生成中断时使用。
20:16	5'h0	RW	矢量 2 矢量编号, 当用户 IRQ_usr_irq_req[2] 生成中断时使用。
12:8	5'h0	RW	矢量 1 矢量编号, 当用户 IRQ_usr_irq_req[1] 生成中断时使用。
4:0	5'h0	RW	矢量 0 矢量编号, 当用户 IRQ_usr_irq_req[0] 生成中断时使用。

IRQ 模块用户矢量号 (0x84)

表 4-51 IRQ 模块用户矢量号

位索引	默认值	访问类型	描述
28:24	5'h0	R0	矢量 11 矢量编号, 当用户 IRQ_usr_irq_req[11] 生成中断时使用。
20:16	5'h0	RW	矢量 10 矢量编号, 当用户 IRQ_usr_irq_req[10] 生成中断时使用。
12:8	5'h0	RW	矢量 9 矢量编号, 当用户 IRQ_usr_irq_req[9] 生成中断时使用。
4:0	5'h0	RW	矢量 8 矢量编号, 当用户 IRQ_usr_irq_req[8] 生成中断时使用。

IRQ 模块用户矢量号 (0x88)

表 4-52 IRQ 模块用户矢量号

位索引	默认值	访问类型	描述
28:24	5'h0	R0	矢量 15 矢量编号, 当用户 IRQ_usr_irq_req[15] 生成中断时使用。
20:16	5'h0	RW	矢量 14



位索引	默认值	访问类型	描述
			矢量编号, 当用户 IRQ_usr_irq_req[14] 生成中断时使用。
12:8	5'h0	RW	矢量 13 矢量编号, 当用户 IRQ_usr_irq_req[13] 生成中断时使用。
4:0	5'h0	RW	矢量 12 矢量编号, 当用户 IRQ_usr_irq_req[12] 生成中断时使用。

IRQ 模块通道中断矢量号 (0xA0), 如果启用 MSI, 那么此寄存器会指定 MSI 的 MSI 矢量编号。在传统中断中, 每个字段仅限 2 个 LSB 用于映射到 INTA、B、C 或 D。与其它 C2H/H2C 位打包澄清信息相似, 请参阅上图。第一个 C2H 矢量位于最后一个 H2C 矢量之后。例如, 如果 NUM_H2C_Channel = 1, 那么 H2C0 矢量位于 0xA0 (位 [4:0]), 而 C2H 通道 0 矢量则位于 0xA0 (位 [12:8])。如果 NUM_H2C_Channel = 4, 那么 H2C3 矢量位于 0xA0 28:24, 而 C2H 通道 0 矢量则位于 0xA4 (位 [4:0])。

表 4-53 IRQ 模块通道中断矢量号

位索引	默认值	访问类型	描述
28:24	5'h0	R0	矢量 3 矢量编号, 当通道 3 生成中断时使用。
20:16	5'h0	RW	矢量 2 矢量编号, 当通道 2 生成中断时使用。
12:8	5'h0	RW	矢量 1 矢量编号, 当通道 1 生成中断时使用。
4:0	5'h0	RW	矢量 0 矢量编号, 当用户 当通道 0 生成中断时 使用。

IRQ 模块通道中断矢量号 (0xA4)

表 4-54 IRQ 模块通道中断矢量号

位索引	默认值	访问类型	描述
28:24	5'h0	R0	矢量 7 矢量编号, 当通道 7 生成中断时使用。
20:16	5'h0	RW	矢量 6 矢量编号, 当通道 6 生成中断时使用。



位索引	默认值	访问类型	描述
12:8	5'h0	RW	矢量 5 矢量编号, 当通道 5 生成中断时使用。
4:0	5'h0	RW	矢量 4 矢量编号, 当用户 当通道 4 生成中断时 使用。

Configure 模块标识符 (0x00), 本节描述配置块寄存器。

表 4-55 Config 模块标识符

位索引	默认值	访问类型	描述
31:20	12'h1fc	R0	子系统标识符
19:16	4'h2	R0	IRQ 标识符
15:8	8'h0	R0	保留
7:0	8'h04	R0	版本 8'h01: 2022.7 8'h02: 2023.2 8'h03: 2023.4 8'h04: 从 2023.5 到最新版本

剩下的寄存器暂未支持, 此处跳过。

H2C SGDMA 寄存器 (0x4), 本节描述了 SGDMA 寄存器。

H2C SGDMA 标识符 (0x00)

表 4-56 H2C SGDMA 寄存器

位索引	默认值	访问类型	描述
31:20	12'h1fc	R0	子系统标识符
19:16	4'h2	R0	IRQ 标识符
15	1'b0	R0	数据流传输 1: AXI4-Stream 接口 0: AXI4 Memory Mapped 接口
14:2	8'h0	R0	保留
11:8	可变	R0	通道 ID 目标 [3:0]
7:0	8'h04	R0	版本 8'h01: 2022.7 8'h02: 2023.2 8'h03: 2023.4 8'h04: 从 2023.5 到最新版本

H2C SGDMA 描述符地址低位 (0x80)



表 4-57 H2C 描述符地址低位

位索引	默认值	访问类型	描述
31:0	32'h0	RW	dsc_adr[31:0] 描述符起始地址的低 32 位。 dsc_adr[63:0] 是在控制寄存器运行位已置位之后提取的首个描述符地址。

H2C SGDMA 描述符地址高位 (0x84)

表 4-58 H2C 描述符地址高位

位索引	默认值	访问类型	描述
31:0	32'h0	RW	dsc_adr[63:32] 描述符起始地址的低 32 位。 dsc_adr[63:0] 是在控制寄存器运行位已置位之后提取的首个描述符地址。

H2C SGDMA 临近描述符个数 (0x88)

表 4-59 H2C SGDMA 临近描述符个数

位索引	默认值	访问类型	描述
5:0	6'h0	RW	dsc_adj[5:0] 表示位于描述符起始地址之后的额外相邻描述符数量。

H2C SGDMA 描述符信用个数 (0x8C)

表 4-60 SGDMA 描述符信用个数

位索引	默认值	访问类型	描述
5:0	6'h0	RW	h2c_dsc_credit[9:0] 写入此寄存器将为通道添加描述符信用值。仅当在“描述符信用值模式”寄存器中通过通道的各个位启用此寄存器时，此寄存器才可供使用。在通道控制寄存器运行位的下降沿上会自动清除信用值，或者针对通道禁用“描述符信用值模式”时，也同样如此。通过读取该寄存器即可判定该通道当前剩余信用值。

C2H SGDMA 寄存器 (0x5)，本节描述了 SGDMA 寄存器。

C2H SGDMA 标识符 (0x00)



表 4-61 C2H SGDMA 寄存器

位索引	默认值	访问类型	描述
31:20	12'h1fc	R0	子系统标识符
19:16	4'h2	R0	IRQ 标识符
15	1'b0	R0	数据流传输 1: AXI4-Stream 接口 0: AXI4 Memory Mapped 接口
14:2	8'h0	R0	保留
11:8	可变	R0	通道 ID 目标 [3:0]
7:0	8'h04	R0	版本 8'h01: 2022.7 8'h02: 2023.2 8'h03: 2023.4 8'h04: 从 2017.1 到最新版本

C2H SGDMA 描述符地址低位 (0x80)

表 4-62 C2H 描述符地址低位

位索引	默认值	访问类型	描述
31:0	32'h0	RW	dsc_adr[31:0] 描述符起始地址的低 32 位。 dsc_adr[63:0] 是在控制寄存器运行位已置位之后提取的首个描述符地址。

C2H SGDMA 描述符地址高位 (0x84)

表 4-63 C2H 描述符地址高位

位索引	默认值	访问类型	描述
31:0	32'h0	RW	dsc_adr[63:32] 描述符起始地址的低 32 位。 dsc_adr[63:0] 是在控制寄存器运行位已置位之后提取的首个描述符地址。

C2H SGDMA 临近描述符个数 (0x88)

表 4-64 C2H SGDMA 临近描述符个数

位索引	默认值	访问类型	描述
5:0	6'h0	RW	dsc_adj[5:0] 表示位于描述符起始地址之后的额外相邻描述符数量。

C2H SGDMA 描述符信用个数 (0x8C)



表 4-65 SGDMA 描述符信用个数

位索引	默认值	访问类型	描述
5:0	6'h0	RW	c2h_dsc_credit[9:0] 写入此寄存器将为通道添加描述符信用值。仅当在“描述符信用值模式”寄存器中通过通道的各个位启用此寄存器时，此寄存器才可供使用。在通道控制寄存器运行位的下降沿上会自动清除信用值，或者针对通道禁用“描述符信用值模式”时，也同样如此。通过读取该寄存器即可判定该通道当前剩余信用值。

SGDMA 公用寄存器 (0x6), 本节介绍了 SGDMA 的公用寄存器。

SGDMA 标识符寄存器 (0x00)

表 4-66 SGDMA 标识符寄存器

位索引	默认值	访问类型	描述
31:20	12'h1fc	R0	子系统标识符
19:16	4'h2	R0	IRQ 标识符
15:8	8'h0	R0	保留
7:0	8'h04	R0	版本 8'h01: 2022.7 8'h02: 2023.2 8'h03: 2023.4 8'h04: 从 2023.5 到最新版本

0x10、0x14 和 0x18 对应的寄存器暂未支持，此处不做介绍。

SGDMA 描述符信用模式使能 (0x20)

表 4-67 描述符信用模式使能

位索引	默认值	访问类型	描述
3:0	0x0	RW	h2c_dsc_credit_enable [3:0] 每条 H2C 通道 1 位。设为 1 即可启用描述符信用值计算。对于每条通道，描述符提取引擎将把提取的描述符数量限制为通过写入该通道的“描述符信用值寄存器”赋予该描述符的信用值数量。
15:4	4'h2		保留
19:16	0x0	RW	c2h_dsc_credit_enable [3:0] 每条 C2H 通道 1 位。设为 1 即可启用



位索引	默认值	访问类型	描述
			描述符信用值计算。对于每条通道，描述符提取引擎将把提取的描述符数量限制为通过写入该通道的“描述符信用值寄存器”赋予该描述符的信用值数量。

SGDMA 描述符信用模式使能 (0x24)

表 4-68 描述符信用模式使能

位索引	默认值	访问类型	描述
5:0	6'h0	W1S	位元描述与 SGDMA Descriptor Credit Mode Enable (0x20) 中的位元描述相同。

表 4-69 描述符信用模式使能

位索引	默认值	访问类型	描述
5:0	6'h0	W1C	位元描述与 SGDMA Descriptor Credit Mode Enable (0x20) 中的位元描述相同。

MSI-X 矢量表和 PBA (0x8)，暂未支持，此处不做介绍。

4.6 描述符

DMA Subsystem for PCI Express 使用已链接的描述符列表来指定 DMA 传输的源、目标和长度。描述符列表是由驱动程序创建的，存储在主机存储器内。DMA 通道由含数个控制寄存器的驱动程序进行初始化，以开始提取描述符列表并执行 DMA 操作。

描述符用于描述 DMA Subsystem for PCIe 应执行的存储器传输。每个通道都有其自己的描述符列表。每个通道的描述符列表的起始地址均在硬件寄存器中由驱动程序进行初始化。启用通道后，描述符通道就会开始从初始地址提取描述符。随后，它会从已提取的最后一个描述符的 Nxt_adr[63:0] 字段中执行提取。描述符必须对齐到 32 个字节的边界处。相邻描述符的初始块的大小是使用 Dsc_Adj 寄存器来指定的。初始提取后，描述符通道使用最后一个提取的描述符的 Nxt_adj 字段来判定下一个描述符地址的描述符数量。相邻描述符块不得跨 4K 地址边界。描述符通道会在单一请求内提取尽可能多的描述符，可提取的数量受 MRRS、相邻描述符数量以及通道的描述符缓冲器内的可用空间所限。

注意：由于 MRRS 在大部分主机系统中为 512 个字节或 1024 个字节，因此在单一请求中不允许相邻描述符数量超过 32 个。但如果需要，设计在相邻描述符构成的单一块内允许的描述符最大数量为 64 个。

描述符列表中的每个描述符都必须准确描述紧随其后的描述符或描述符块。在相邻描述符构成的块中，Nxt_adj 值从第一个描述符开始递减直至倒数第二个描述符（值为 0）。同样，块中的每个描述符指向块中的下一个描述符，最后一个描述符除外，它可能指向新的块或者可能使列表终止。描述符列



表终止以停止 (Stop) 控制位来表示。观测到含 Stop 控制位的描述符后，针对该列表将不再发出描述符提取请求。Stop 控制位只能在块的最后一个描述符上置位。

使用 AXI4 Memory Mapped 接口时，不会对卡的 DMA 地址执行转换。如果主机不知道卡的地址掩码，则必须在用户逻辑中汇编描述符，并使用描述符旁路接口将其提交至 DMA。

表 4-70 描述符格式

偏移	字段			
0x0	Magic	Rsv[1:0]	Nxt_adj[5:0]	Control[7:0]
0x4	4'h0, Len[27:0]			
0x8	Src_adr[31:0]			
0xc	Src_adr[63:32]			
0x10	Dst_adr[31:0]			
0x14	Dst_adr[63:32]			
0x18	Nxt_adr[31:0]			
0x1c	Nxt_adr[63:32]			

表 4-71 描述符字段

偏移	字段	位索引	子字段	描述
0x0	Magic	15:0		16'had4b。此代码用于验证驱动程序生成的描述符是否有效。
0x0		1:0		保留，置位为 0
0x0	Nxt_adj	5:0		位于下一个描述符地址字段所在处的描述符之后的额外相邻描述符的数量。相邻描述符块不得跨 4k 边界。
0x0	Control	5、6、7		保留
0x0		4	EOP	针对 Stream 接口的包结束。
0x0		2 和 3		保留
0x0		1	Completed	位于下一个描述符地址字段所在处的描述符之后的额外相邻描述符的数量。相邻描述符块不得跨 4k 边界。
0x0	Length	0	Stop	设为 1 即可为此描述符列表停止提取描述符。Stop 位只能在相邻描述符块的最后一个描述符上置位。
0x04		31:28		保留，置位为 0
0x04		27:0		数据长度（以字节为单位）。
0x00-0x8	Src_adr	63:0		H2C 和存储器掩码传输的源地址。C2H 传输的元数据写回地址。
0x14-0x10	Dst_adr	63:0		C2H 和存储器掩码传输的目标地址。不可



偏移	字段	位索引	子字段	描述
				用于 H2C 数据流传输。
0x1C–0x18	Nxt_adr	63:0		列表中下一个描述符的地址。

描述符旁路，根据通道，可通过 `para_def.vh` 中的宏定义来绕过描述符提取引擎。启用描述符旁路的通道可从其相应的 `c2h_dsc_byp` 或 `h2c_dsc_byp` 总线接受描述符。在通道接受描述符前，控制寄存器的“运行（Run）”位必须置位。绕过描述符时，不使用 `NextDescriptorAddress`、`NextAdjacentCount` 和 `Magic` 描述符字段。控制寄存器位中的 `ie_descriptor_stopped` 位不会阻止用户逻辑写入其它描述符。写入通道的所有描述符都会接受处理，当通道缓冲器已满时，将限制写入新描述符。

4.7 其它功能

DMA 引擎要高效的工作，除了常规的 DMA 读写和中断外，还会引入 Poll Mode 写回、AXI4-Stream 写回、链式描述符、环形描述符等功能，下面分别介绍这些功能特性。

Poll Mode(轮询模式)写回，每个引擎都能将已完成的描述符计数写回主机存储器。这样驱动程序即可轮询主机存储器以判定何时 DMA 完成，而无需等待中断。对于给定 DMA 引擎，当 DMA 完成描述符传输并且 `ie_descriptor_completed` 和 `Pollmode_wb_enable` 均已置位后，就会发生已完成的描述符计数写回操作。报告的已完成描述符计数是从 DMA 启动开始，已完成的描述符总数（而不仅是含“Completed”标记的描述符置位）。写回地址由 `Pollmode_hi_wb_addr` 和 `Pollmode_lo_wb_addr` 寄存器来定义。

表 4-72 已完成的描述符计数写回格式

偏移	字段		
0x0	Sts_err	7'h0	Compl_descriptor_count[23:0]

表 4-73 已完成的描述符计数写回字段

字段	描述
Sts_err	通道状态寄存器内任意错误状态位的按位 OR。
Compl_descriptor_count[23:0]	Complete Descriptor Count 寄存器的下 24 位。

DMA H2C 数据流传输写回，对于主机到卡传输，将从位于源地址的主机读取数据，但不使用描述符中的目标地址。数据包可跨多个描述符。包终止以 EOP 控制位来表示。具有 EOP 位的描述符会在数据最后一拍的 AXI4-Stream 用户接口上断言 `*_tlast` 有效。交付至 AXI4-Stream 接口的数据将按描述符逐个进行打包，`*_tkeep` 全部为 1，但如果描述符长度并非数据路径宽度的整数倍，则其数据传输的最后一个周期内，`*_tkeep` 并非全部为 1。DMA 不会跨多个描述符进行数据打包。

DMA C2H 数据流传输写回，对于卡到主机传输，数据接收来自 AXI4-Stream 接口并写入目标地址。数据包可跨多个描述符。C2H 通道启用时即可接受数据，并具有有效的描述符。接收到数据后，就会按顺序填充描述符。当描述符完成填充或者由于接口上出现包结束而导致描述符关闭后，C2H 通道会将信息写回含预定义的 WB Magic 值 16'h52b4 (表 10: C2H 数据流传输写回字段) 的主机上的写回地址，



并对 EOP 和长度 (Length) 进行相应的更新。对于 C2H AXI4-Stream 接口上的有效数据周期，与给定数据包关联的所有数据都必须连续。注释：C2H 通道写回信息不同于轮询模式更新。C2H 通道写回信息可提供特定描述符的驱动程序当前长度状态。这不同于 轮询模式 中所述的 Pollmode_*。对于任一数据包的所有传输（最后一次数据传输除外）而言，tkeep 位必须全部为 1。在执行数据包的最后一次传输时，当 tlast 断言有效时，可指定 tkeep 不全为 1，以指定该数据周期并非完整的数据路径宽度。断言有效的 tkeep 位需打包到 lsb 中以表示连续数据。C2H 数据流传输描述符的长度（目标缓冲器的大小）必须始终为 64 个字节的倍数。

表 4-74 数据流传输写回格式

偏移	字段		
0x0	WB Magic[15:0]	Reserved [14:0]	Status[0]
0x04	Length[31:0]		

表 4-75 C2H 数据流传输写回字段

字段	索引	子字段	描述
Status	0	EOP	包结束
Reserved	14:0		保留
WB Magic	15:0		16'h52b4。此代码用于验证 C2H 写回是否有效。
Length	31:0		数据长度（以字节为单位）。

DMA 链式描述符，由于链式描述符中的 Nxt_adj 只有 6bit，最多连续 64 个描述符，每个描述符对应的内存空间大多为 4K，所以最大内存 256K，而实际所需内存远大于 256K。此时，Nxt_adj 必须取值为 0x3F，并且需要连续的多个 0x3F，直到剩余描述符个数不足 0x3F，描述符中的 Nxt_adj 才开始逐渐递减，如 0x3F, ..., 0x3F, 0x3E, ..., 0x00, 0x00 结束。下面以 100K 芯片、C2H 通道、描述符个数为 72 为例列出它的描述符列表，如下表 4-76 所示，当描述符个数大于 64 个时，驱动配置的相邻描述符计数 Nxt_adj 为 0x3F，而当描述符个数小于 64 时，驱动配置的 Nxt_adj 与第一个描述符的 Nxt_adj 一致。

表 4-76 描述符个数为 72 时的链式描述符列表

描述符序号	描述符数据
第 1 个描述符	64'h00001000ad4b3f00
	64'h0000000000000000
	64'h000000001c001000
	64'h0000000018000020
第 2~7 个描述符
第 8 个描述符	64'h00001000ad4b3f00
	64'h0000000000000000



描述符序号	描述符数据
	64'h000000001c008000
	64'h0000000018000100
第 9 个描述符	64'h00001000ad4b3e00
	64'h0000000000000000
	64'h000000001c009000
	64'h0000000018000120
第 10~70 个描述符
第 71 个描述符	64'h00001000ad4b0000
	64'h0000000000000000
	64'h000000001c047000
	64'h00000000180008e0
第 72 个描述符	64'h00001000ad4b0003
	64'h0000000000000000
	64'h000000001c048000
	64'h0000000000000000

DMA 环形描述符，环形描述符常用在数据流一直都有，描述符需要循环更新的场景，如外接的数据源来自于相机，此时驱动常常申请一个连续的 4Kbyte 内存空间用来存描述符，然后通过 Credit 来调度 DMA 控制器核的工作。同样以 100K 芯片、C2H 通道为例，申请了 4Kbyte 的空间来存描述符，此时的描述符列表如下表 4-77 所示。表中环形描述符与链式描述符的主要不同点是：环形描述符中 stop 位一直为 0，当走完一圈后再次指向第一个描述符。

表 4-77 描述符个数为 128 时的环形描述符

描述符序号	描述符数据
第 1 个描述符	64'h00001000ad4b3f00
	64'h0000000000000000
	64'h000000001c001000
	64'h0000000018000020
第 2 到第 63 个描述符
第 64 个描述符	64'h00001000ad4b3f00
	64'h0000000000000000
	64'h000000001c040000
	64'h0000000018000500
第 65 个描述符	64'h00001000ad4b3f00
	64'h0000000000000000
	64'h000000001c041000
	64'h0000000018000520



描述符序号	描述符数据
第 66~126 个描述符
第 127 个描述符	64'h00001000ad4b0000
	64'h0000000000000000
	64'h000000001c07f000
	64'h00000000180009ec
第 128 个描述符	64'h00001000ad4b0002
	64'h0000000000000000
	64'h000000001c080000
	64'h0000000018000000

5 功能仿真验证

当 PCIe SGDMA 源代码设计完成且 TB 与自动化脚本准备好后，开始进入仿真阶段。下面依次从系统仿真框架、仿真流程、测试激励和仿真结果分析四个步骤详细介绍功能仿真的实现。

5.1 系统仿真框架

系统仿真框架如下图 5-1 所示，主要分成四部分：Auto simulation 模块（tcl 自动化仿真脚本），Stimulus&Response 模块（测试激励和 BFM），PCIeCore&SGDMA（PCIeCore 的 BFM 与 SGDMA），Auto Compare 模块（shell&perl 自动化测试脚本）。

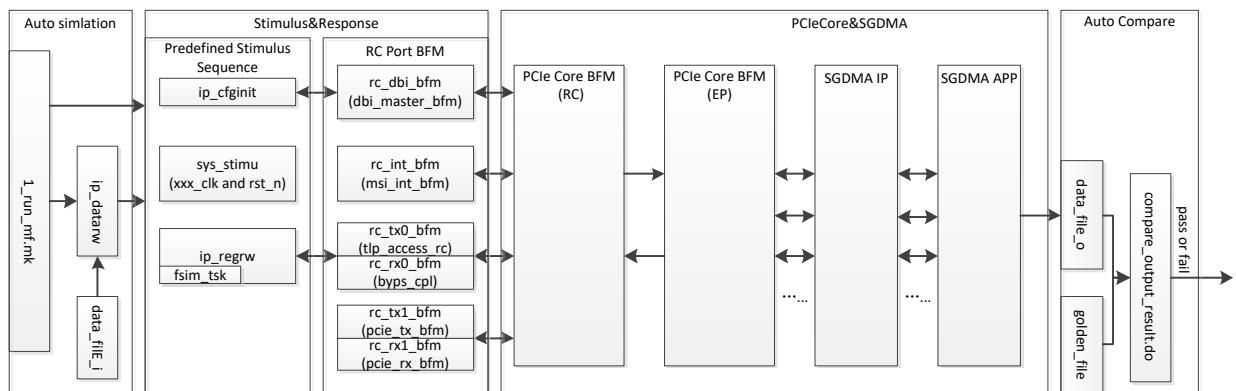


图 5-1 SGDMA 系统仿真框图

Auto Simulation 包括两组文件，1_run_mf.mk 里面包含 vcs 的测试自动化脚本 makefile 文件，这个脚本只会编译生成 fsdb 文件，看波形的话用 verdi，对应脚本 2_run_verdi.do。ip_datarw.v 从 data_file 中导出测试数据或者测试数据写入 data_file 便于比较。

Stimulus&Response 包括两大模块 Predefined Stimulus Sequence 和 RC Port BFM，ip_cfginit 为 PCIe 的初始化模块，如 BAR0 空间寄存器基地址配置和大小获取等；sys_stimu 为 sgdma_ip 的主要测试激励，如时钟和复位等；ip_regrw 为每个 CASE 的配置模块，如 CASE0 寄存器读写，就包括遍历 512Kbyte 的 User 寄存器；rc_db1_bfm 包括 RC 配置空间的读写，ep_db1_bfm 为 EP 配置空间的读写（需要接到 PCIe Core BFM EP）；rc_int_bfm 为 MSI 中断解析模块，后续版本会支持 MSI-X 中断和 Legacy 中断；rc_tx0_bfm&rc_rx0_bfm 为 RC Master 发出的读写请求和响应；rc_tx1_bfm 和 rc_rx1_bfm 为接收 EP Master 发出的读写和响应模块。

PCIe Core BFM 分为 RC 和 EP 两类，本设计的 SGDMA 是在 EP 端完成的，测试工程 sgdma_subsys_exam 包括 PCIe Core BFM (EP)、SGDMA_IP 和 SGDMA_APP 三部分，PCIe Core BFM (EP) 为 EP Core 的仿真模型，通过 IP Gen 生成；SGDMA_IP 为 SGDMA 控制器核的源码，SGDMA_APP 为用户应用程序，此处给的是简单的测试代码。

Auto Compare 为 sgdma_subsys_exam 工程吐出的文件与 Golden 文件的比较脚本，对应代码 compare_output_result.do。

5.2 仿真流程

当上述脚本、BFM、SGDMA 源代码等基础文件都准备好后，下一步准备测试数据、添加测试 CASE 的代码开始仿真，详细的仿真流程如下图 5-2 所示。除了流程图中的步骤外，在仿真之前最好跑下 nLint，nLint 能报告出代码当前的错误以及潜在的风险，如某一信号扇出过大，某模块端口位宽不匹配，wire 信号未定义等。

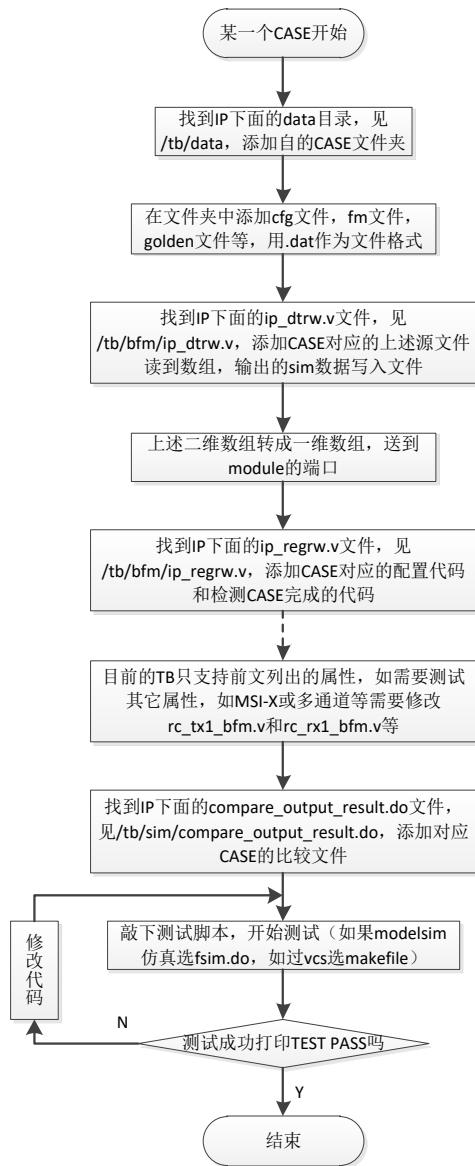


图 5-2 SGDMA IP 仿真流程图

注意：在 vcs 跑仿真时，可能由于代码进入死循环，或者卡住，导致无法跑完整个流程，此时由于 *.fsdb 尚未生成好，用 verdi 看不到波形。如果遇到此种情况可以进入 tb/qsim 目录，在命令行输入 vsim -do fsim.do，能打开 modelsim 并实时看到当前的波形。建议：在仿真第一个 CASE 的时候先选用此模式，待整个流程跑通后，再切换到 vcs+verdi 的模式，这两种模式只需要在脚本后面再加一个 CASE_NUM 即可开始仿真，看波形的时候，也是在 2_run_verdi.do 后面加 CASE_NUM。



5.3 测试激励介绍

SGDMA 的测试激励是根据 CASE_NUM 来产生的，每个 CASE 对应着不同的属性，在 ip_regrw.v 中实现，共分成三类测试模式，通过配置 User 寄存器偏移地址为 0 实现，当配置值 2'b11 为环回测试，其它模式不需要额外的配置。以 100K、C2H 引擎、AXI4-Stream 接口为例，除了公用的寄存器读写外，它还包括下列属性，详细的属性与对应激励表如下表 5-1 所示，H2C 引擎与 C2H 引擎的属性大体一致，此处不做介绍。

表 5-1 C2H 引擎支持的属性与它需要的激励

C2H 支持的属性	需要的测试激励
User 寄存器读写	User 寄存器地址和数据
PollMode 回写	FM 数据； PollMode 使能；最后一个描述符 completed_flag 和 stop 置位， MSI 中断使能。
AXI4-Stream 回写	FM 数据； AXI4-Stream 使能；最后一个描述符 completed_flag 和 stop 置位， MSI 中断使能。
Magic 错误	FM 数据；描述符分两半，一半的第一个描述符出现 Magic 错误，另一半正常；需要一个 MSI 中断结束。
C2H 描述符中的 Length 任意值 (64byte 对界)	FM 数据；描述符中的长度 64byte 的随机倍数；需要一个 MSI 中断结束。
目的地址任意值	FM 数据；目的地址低 3bit 覆盖 0~7；需要一个 MSI 中断结束。
Run 任意地方停止	FM 数据； RUN 随机地方停止但要覆盖 DMA 引擎工作前，工作时， DMA 完成工作后，同样需要一个 MSI 中断结束。
目的地址 32bit 和 64bit 混合	FM 数据；描述符中的目的地址随机的 32bit 和 64bit；同样需要一个 MSI 中断结束。
目的地址+长度跨 4K	FM 数据；产生随机的目的地址，刻意的长度，让两者之和跨 4K；同样需要一个 MSI 中断结束。
随机 Completed flag 置位	FM 数据；描述符随机产生 completed_flag 位，并使能 MSI 中断， stop 中断置为结束。
链式描述符	FM 数据；描述符如表 4-67 所示。
环形描述符	FM 数据；描述符如表 4-68 所示。
Credit 随机送出	FM 数据； Credit 模式使能， TB 中随机配置 Credit 个数到 EP；同样需要一个 MSI 中断结束。
用户中断测试	用户自定义寄存器，与 DMA 引擎中的中断测试一致； MSI 中断使能。

当单独的 C2H 和 H2C 通道测试完成后，下面开始进行回环仿真，回环仿真根据接口不同分成两类，它们的测试激励如下表 5-2 所示。

表 5-2 H2C&C2H 回环仿真与它需要的激励

H2C&C2H 回环仿真	需要的测试激励
AXI4-Stream 接口	FM 数据, H2C 返回的数据通过 C2H 写回 (H2C 的 AXI4-Stream 接口直接对上 C2H 的 AXI4-Stream 接口); 需要一个 MSI 中断提示测试完成。
AXI4-MM 接口	FM 数据, H2C 的数据通过 AXI4-MM 接口全部写入 DDR(或 BRAM), 然后从 DDR(或 BRAM) 读出; 需要一个 MSI 中断提示测试完成。

400K、90K、180K 芯片与 100K 芯片属性非常相似, 不同点主要体现在用户数据位宽不一致, 所以测试工程只提供了回环仿真的 CASE。500K 芯片的 PCIe 核与上述芯片差异较大, 目前仅支持 AXI4-MM 环回测试, AXI4-Stream 接口将在后续版本支持。

5.4 仿真波形分析

准备好上述仿真激励后, 同样以 100K 芯片为例, 选择常用的“User 寄存器读写”、“Poll Mode 回写”、“AXI4-Stream 回写”、“随机 Completed Flag 置位”、“两个回环仿真”作为典型 CASE 开始仿真, 其波形如下图 5-3 至 5-15 所示, 下文将依次分析各仿真波形。User 寄存器读写, 它的工作过程为 RC 通过 Client0 接口发送 Memory 写和读, 然后转到 DMA 控制器核 sgdma_subsys.v 的 AXI4-Lite Master 接口, 详细的时序如下图 5-3、5-4 和 5-5 所示, 图 5-3 中的一个 hv 脉冲对应一个寄存器读写, 图 5-4 为标准的 AXI4-Lite 协议, 图 5-5 的一个 hv 对应一个返回包。



图 5-3 User 寄存器读写对应 RC 侧 Client0 接口上的波形



图 5-4 User 寄存器读写映射到 C2H 引擎侧 AXI4-Lite 接口上的波形

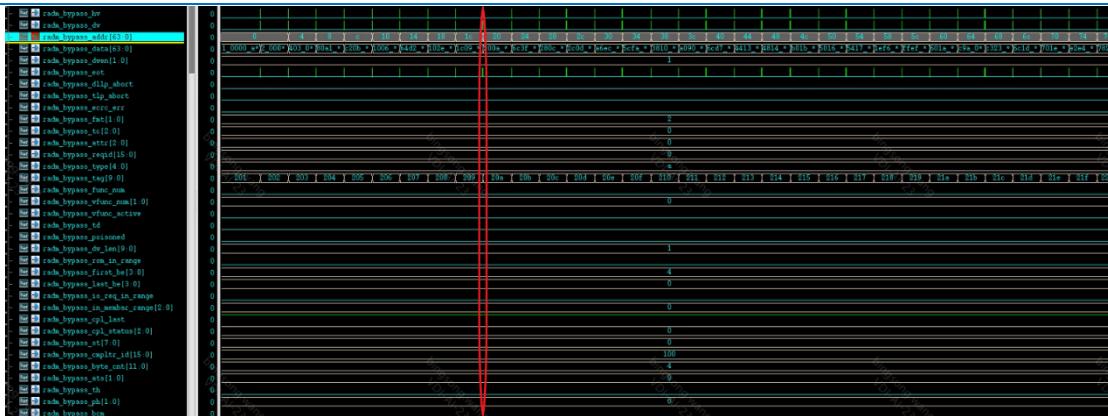


图 5-5 User 寄存器读返回到 RC 侧 Radm_bypass 接口上的波形

Poll Mode 回写，它的工作过程为 RC 通过 Client0 接口配置完 DMA 引擎寄存器后，sgdma_subsys.v 模块根据寄存器配置启动 DMA 引擎，当 Poll Mode 使能打开且带 Completed Flag 的描述符发送完成后，发起 Poll Mode 回写，回写格式见表 4-72，详细的时序波形如下图 5-6 和 5-7 所示，图 5-6 中展示了 Poll Mode 回写前后的波形，其中*_dv_o 较短的时序即为 Poll Mode 回写的时序，图 5-7 将放大这一块波形。EP 侧的回写发生后，随后 RC 的 Radm_trgt1 接口会收到回写数据，此接口的波形如下图 5-8 所示，*_header_o 中的[127:64]为 64'h1980_0000 回写的地址，包长度为 4byte，First DW BE 为 0xf，Last DW BE 为 0x0。



图 5-6 C2H 引擎 Poll Mode 回写前后的波形

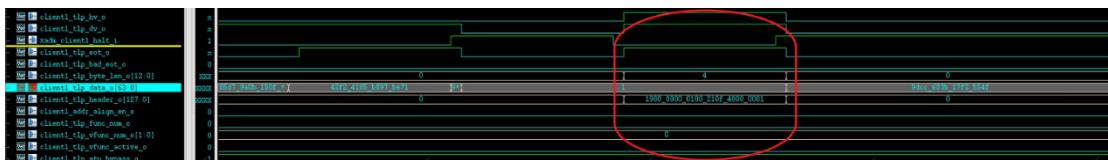


图 5-7 C2H 引擎 Poll Mode 回写波形

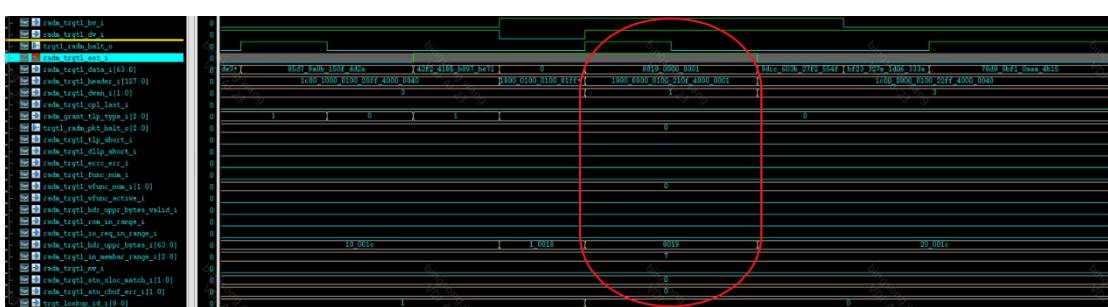


图 5-8 Poll Mode 回写到 RC 侧 trgt1 接口上的波形

AXI4-Stream 回写，它的工作过程同样为 RC 通过 Client0 接口配置 DMA 引擎寄存器，然后启动 DMA

引擎，RC 侧的配置时序与“User 寄存器读写”一致，此处不再介绍；然后判断 AXI4-Stream 回写使能打开且完成一个描述符的发送后，发起 AXI4-Stream 回写，回写格式见表 4-74，详细的时序波形见下图 5-9 和图 5-10。EP 侧发回写后，随后 RC 侧的 Radm_trgt1 接口同样会收到 AXI4-Stream 的回写数据，此接口的波形如下图 5-11 所示，*_header_i 中的 [127:64] 为 64'h1900_0100 回写的地址，包长度为 4byte，First 和 Last DW BE 为 0xf。



图 5-9 C2H 引擎 AXI4-Stream 回写前后的波形

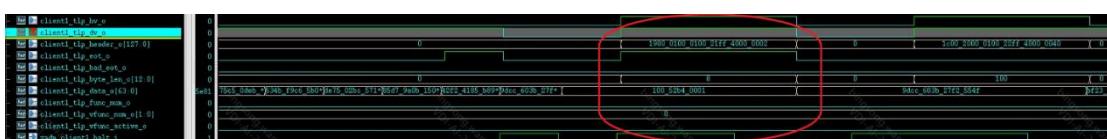


图 5-10 C2H 引擎 AXI4-Stream 回写波形

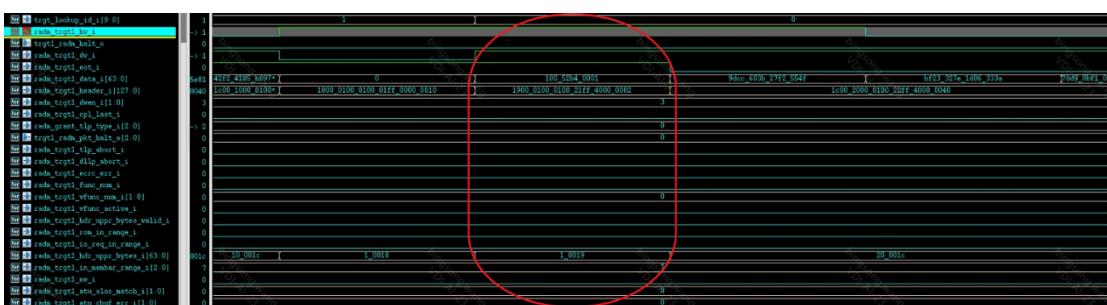


图 5-11 AXI4-Stream 回写到 RC 侧 trgt1 接口上的波形

随机 Completed Flag 置位，它的工作过程为 RC 通过 Client0 接口配置完 DMA 引擎并启动 C2H 通道工作后，当 MSI 使能打开且标注为 Completed Flag 的描述符发送完成后，通过 MSI 总线发起 DMA 中断，MSI 详细的时序波形见下图 5-12，图中存在*_req_o 与*_grant_i 的握手信号，握手成功一个中断发出。EP 侧的 MSI 中断发出后，同样是 RC 侧的 Radm_trgt1 接口接收此中断，波形如下图 5-13 所示，*_header_i 中的[127:64]为中断寄存器地址，长度为 1 个 DW，First DW BE 为 0xf。

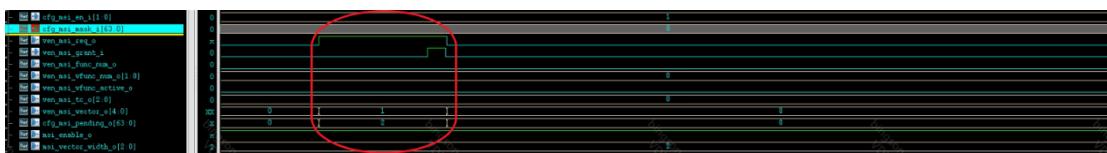


图 5-12 C2H 引擎产生的 MSI 中断波形

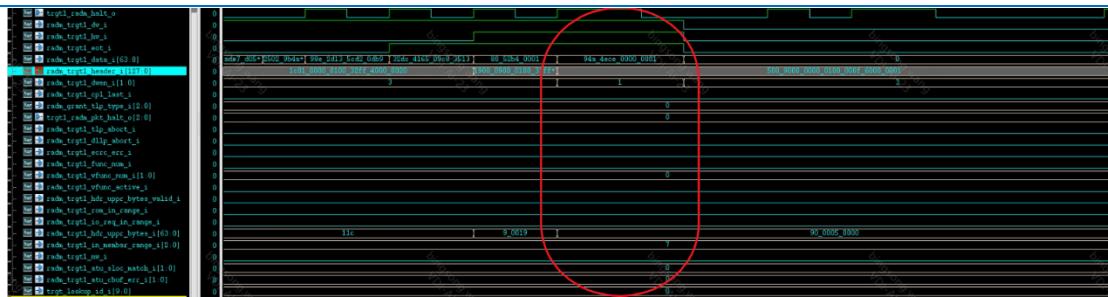


图 5-13 MSI 中断映射到 RC 侧 trgt1 接口上的波形

“两个回环仿真”，首先介绍 AXI4-Stream 回环仿真，它的工作过程为 RC 通过 Client0 接口配置完 DMA 引擎并启动 H2C 通道工作后，随后再用同样的方式启动 C2H 通道工作，H2C 返回的数据通过 AXI4-Stream 总线接口写入 C2H 的 FIFO，C2H 通道的引擎再把数据写回到 RC，这段时间 AXI4-Stream 总线的工作波形如下图 5-14 所示。

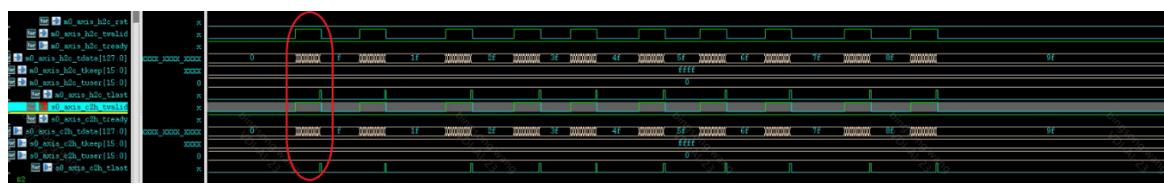


图 5-14 AXI4-Stream 回环仿真接口时序

接着 AXI4-MM 回环仿真，由于 100K 芯片上没有预留 DDR PHY，所以改成有 DDR PHY 的 400K 芯片上做仿真，它的工作过程为通过 Client0 接口配置完 DMA 引擎并启动 H2C 通道工作后，数据经 AXI4-MM 写接口写入 DDR；当数据全部写入 DDR 后，启动 C2H 引擎，再经 AXI4-MM 读接口从 DDR 中读出数据写入 FIFO，C2H 通道的引擎随后把 FIFO 中的数据读出同样写回到 RC，这段时间 AXI4-MM 工作波形如下图 5-15 所示。AXI-Interconnect 部分与 DDR 控制器部分已经是测试完好的 IP，用户无需关注，所以此处并没有贴出详细的时序图，详见附录中的参考文档。

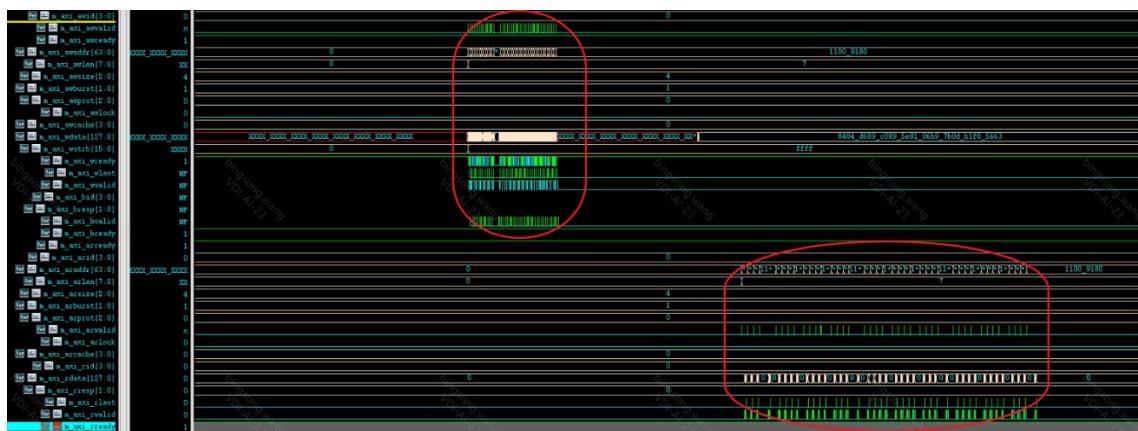


图 5-15 AXI4-MM 回环仿真接口时序

6 移植与上板验证

当前述章节的设计与仿真都做完后，下面进入移植与上板阶段。根目录下的 `impl` 文件夹内有 `ep_prj` 和 `rc_prj` 两个子目录，分别对应 `sgdma_subsys_exam` 工程和 `sgdma_rc_exam` 工程，本设计只介绍 `sgdma_subsys_exam` 工程。该工程完整的测试需要经过以下四个步骤：软硬件环境准备、SGDMA 工程移植、Linux 驱动移植、上板验证。

6.1 软硬件环境准备

以某一次性能测试的环境为例，上板之前需要准备好的硬件有：支持 Gen2 X2 的 X86 CPU，如本例中的 Intel (R) Core (TM) i7-9700 CPU；支持同样性能的 PCIe 板卡，如 PH1A100 开发板。软件需要准备的有：较新的 TD 软件（老的版本可能编译较慢），如本设计的 TD5.6.4；较新的 Linux 系统，如本设计的 Ubuntu20.4（其它系统由于库不一样，可能编译报错）；`sgdma_subsys_exam` 工程的代码、驱动代码以及 `CASE_NUM` 对应的测试代码。

6.2 SGDMA 工程移植

准备好上述软硬件工具后，接着开始 SGDMA 工程的移植，对应的目录为 `/impl/ep_prj/`。添加仿真的源文件和约束文件后，下面开始双击 `pcie_ep_core` 开始配置 EP Core 的属性，本设计中 PH1A100 系列芯片选择的配置如下图 6-1 到 6-5 所示。

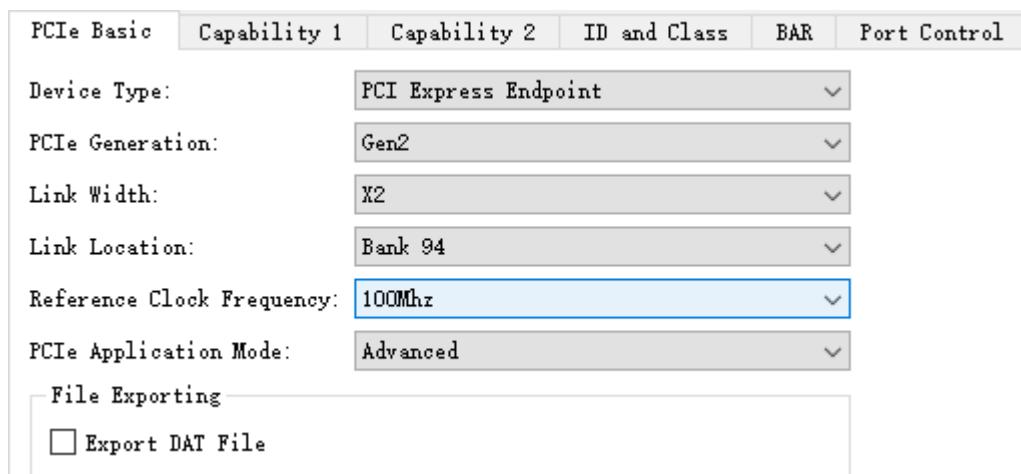


图 6-1 SGDMA PCIe Basic 设置

图 6-1 为当前设计的 PCIe 基本配置，用户可以根据需要改成 Gen3 或 X1，改完之后，PLL 的配置以及时序约束都要做响应的修改。图 6-2 只有在图 6-1 的 PCIe Application Mode 选成 Advanced 才能修改；本设计只用了一个 Physical Function 0；PF0 MPS 一般都设置成最大值，但实际只能到 256 或 128；选上 Extended Tag Field 后，本 PCIe Core 也只支持 64 个；没有用到 VF，不用勾选；其它保持默认值。图 6-3 只选中 MSI 中断，Vector 选择 4，实际测试发现，Ubuntu 系统申请不到多个连续的中断矢量，也就是 Vector 只有一个可用。其它的保持默认。图 6-4 中只关注 Vendor ID 和 Class ID，

PH1A 芯片都为 1EDB, ABCD, 其它默认。

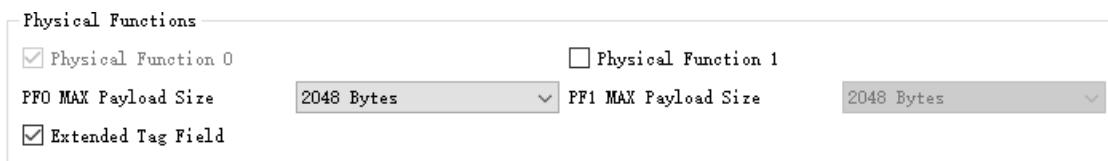


图 6-2 SGDMA PCIe Capability 1 设置

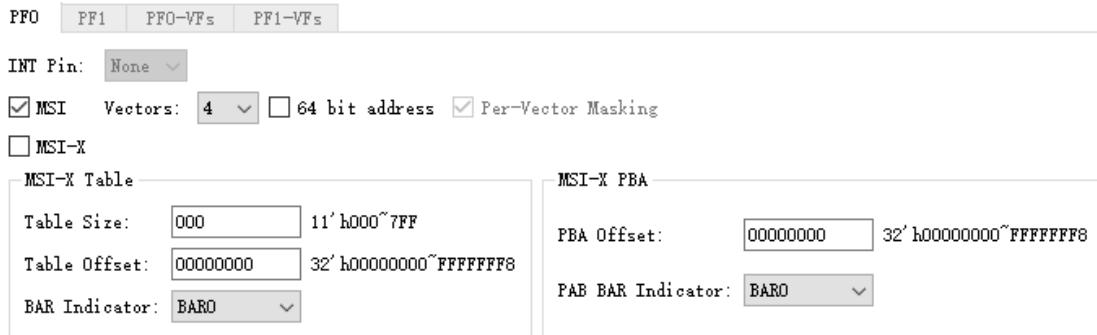


图 6-3 SGDMA PCIe Capability 2 设置

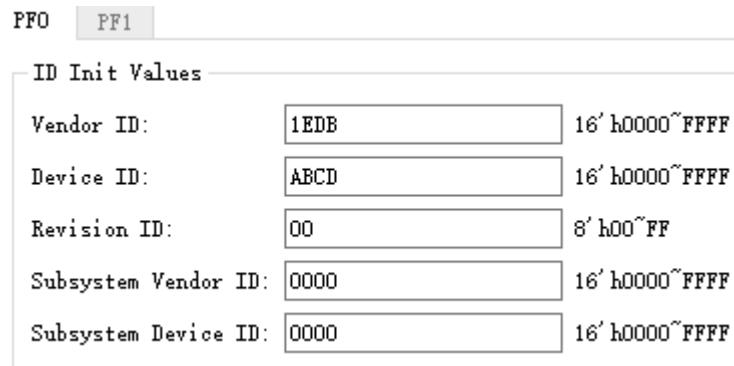


图 6-4 SGDMA PCIe ID and Class 设置

如下图 6-5 所示, 本设计只用到了一个 BAR0, 配置大小为 1MByte 空间, 选择接口为 ELBI, 主要用了读写 DMA 引擎寄存器和 User 寄存器。其它的 BAR 没用到, 保持默认。

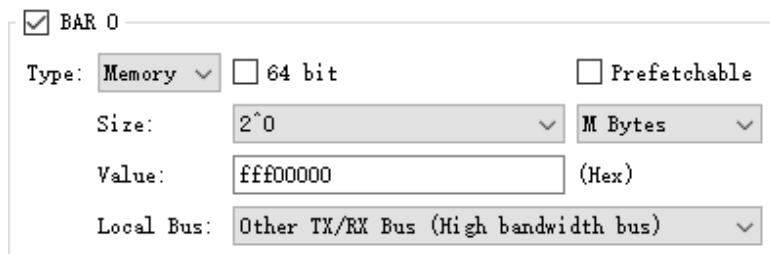


图 6-5 SGDMA PCIe BAR 设置

如下图 6-6 所示, TX port Control 窗口没有选中 TRGT1 接口, 是因为 BAR0 映射到了 ELBI 接口, 通过 ELBI 来读写寄存器, 每次只有 1DW, 无需 TRGT1 接口参与。Interrupt 窗口只是一个状态窗口, 只

要 Capability 2 窗口选中了某一中断，这里就会标注。Other Interfaces 窗口，ELBI 因为 BAR0 映射到了所以选中，除此之外还要选中 dbi 接口，用来读写配置空间寄存器。

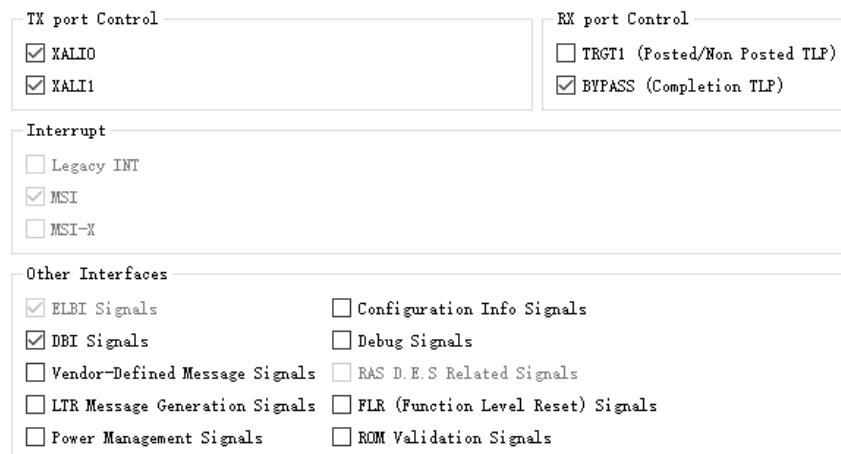


图 6-6 SGDMA PCIe Port Control 配置

配置完上述 PCIe 属性后，点击 OK 生成 IP，随后在 TD 中加入仿真好的源文件和其它 IP 文件，继续添加约束文件，开始编译，至此 SGDMA 工程移植好。PH1A 系列与此类似，不同点仅为 PCIe Generation 支持到 Gen3，Link Width 支持到 X4。

PH2A 系列芯片，由于 PCIe Core 有较大升级，配置界面改动较大，参考设计提供的详细配置如下图 6-7 至 6-11 所示。下文仅对图中 PH2A 系列与 PH1A100 系列属性不同点做详细介绍，其它不再介绍。

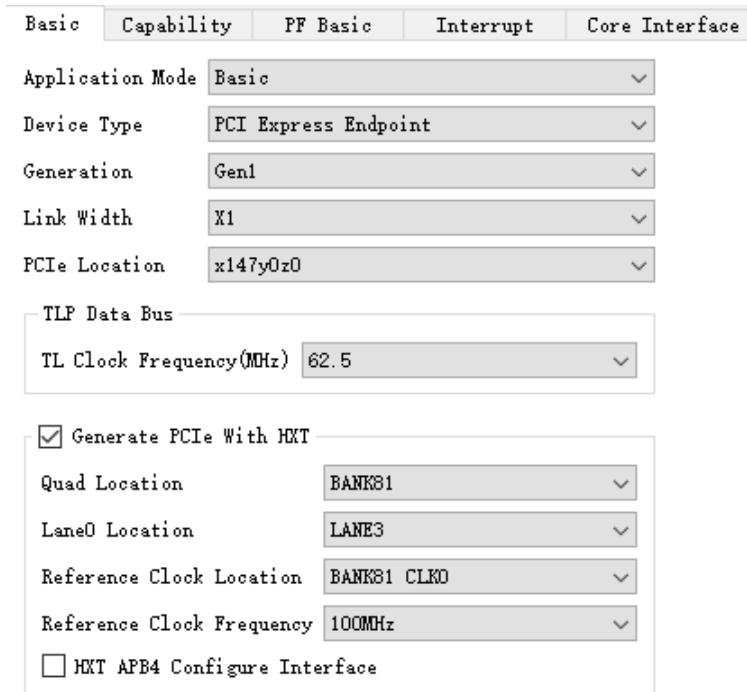


图 6-7 PH2A 系列芯片 SGDMA PCIe Basic 设置

如上图 6-7 所示，TL Clock Frequency (MHz)，为用户提供给 PCIe 核的时钟。当 Generation 和 Link

Width 改变时, 可选的频率也会改变; Quad Location 和 Lane0 Location 因为 PH2A 系列提供两个 PCIe Core 控制器, 可以选择不同位置的 Bank 和 Lane。下图 6-8 所示与 PH1A100 系列不同仅在于 Max Payload Size 扩大到 4096bytes。

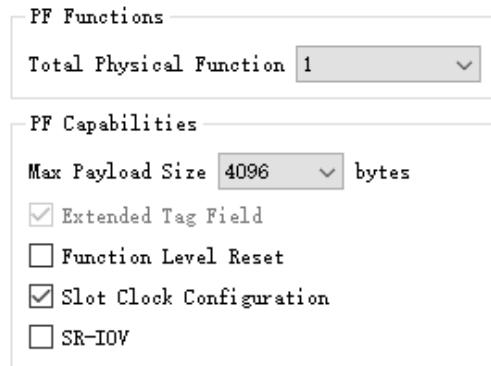


图 6-8 PH2A 系列芯片 SGDMA PCIe Capability 1 设置

Bar Setting						
Bar	Type	64 bit	Prefetchable	Size	Unit	Value (Hex)
<input checked="" type="checkbox"/> Bar0	Memory	<input type="checkbox"/>	<input type="checkbox"/>	2^{10}	MBytes	0xFFFF0000
<input type="checkbox"/> Bar1	Memory			2^7	Bytes	0x00000000
<input checked="" type="checkbox"/> Bar2	Memory	<input type="checkbox"/>	<input type="checkbox"/>	2^7	MBytes	0xF8000000
<input type="checkbox"/> Bar3	Memory			2^7	Bytes	0x00000000
<input type="checkbox"/> Bar4	Memory	<input type="checkbox"/>	<input type="checkbox"/>	2^7	Bytes	0x00000000

Expansion ROM
Size 2^0 Unit KBytes Value 0x00000000

图 6-9 PH2A 系列芯片 SGDMA PCIe BAR 设置

上图 6-9 中增加了 BAR2 的设置, 用于 PCIe to DMA Bypass 接口, 当前参考设计开了 128Mbyte 空间,BAR0 依旧用于 DMA 引擎与 User 寄存器的配置。接着 Interrupt 和 Core Interface 的设置, Interrupt 只用到了 MSI 一个 vector, 配置如下图 6-10 所示, Core Interface 的设置如下图 6-11 所示, 用到了 APB4 接口和 Configure Direct Access Interface。

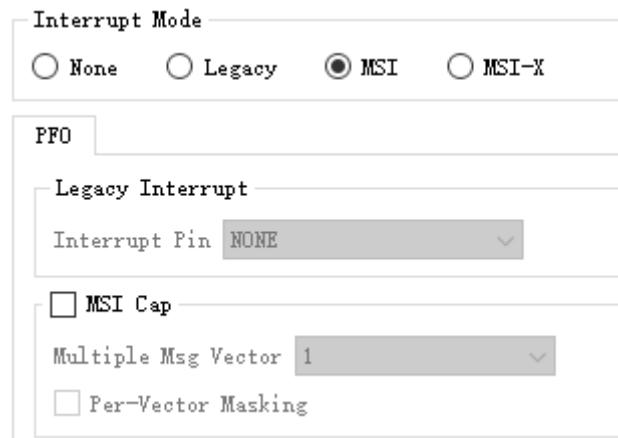


图 6-10 PH2A 系列芯片 SGDMA PCIe Interrupt 设置

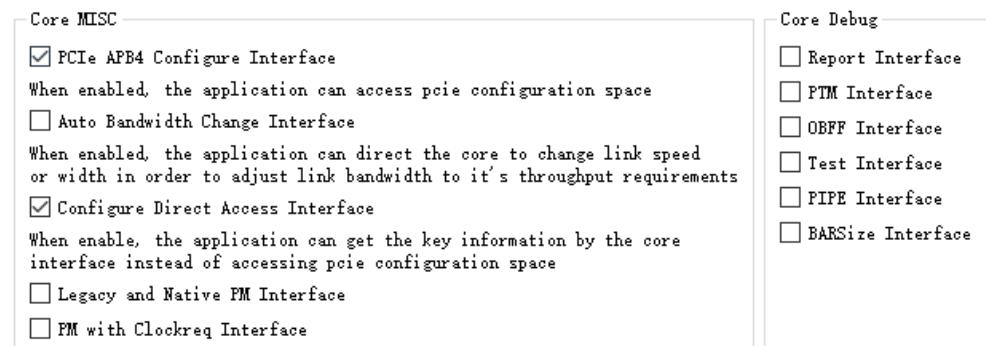


图 6-11 PH2A 系列芯片 SGDMA PCIe Core Interface 设置

上图 6-10 中 MSI 只用了一个矢量，不需要 MSI Cap；图 6-11 中用到了 APB4 去读写取配置空间寄存器，对应到用户的是 CFG_MGMT 总线；Configure Direct Access Interface 直接输出 PCIe 的一些信息，如 Generation、MPS、MRSS 等，不需要用户使用 APB4 总线去读，非常有用。

6.3 Linux 驱动移植

配套的 Linux 软件分为两部分：driver 和 app。Driver 负责 IP engine 的控制，App 负责数据的收发和处理。下图 6-12 所示为 c2h 方向轮询工作流程，可以参考进行二次开发。

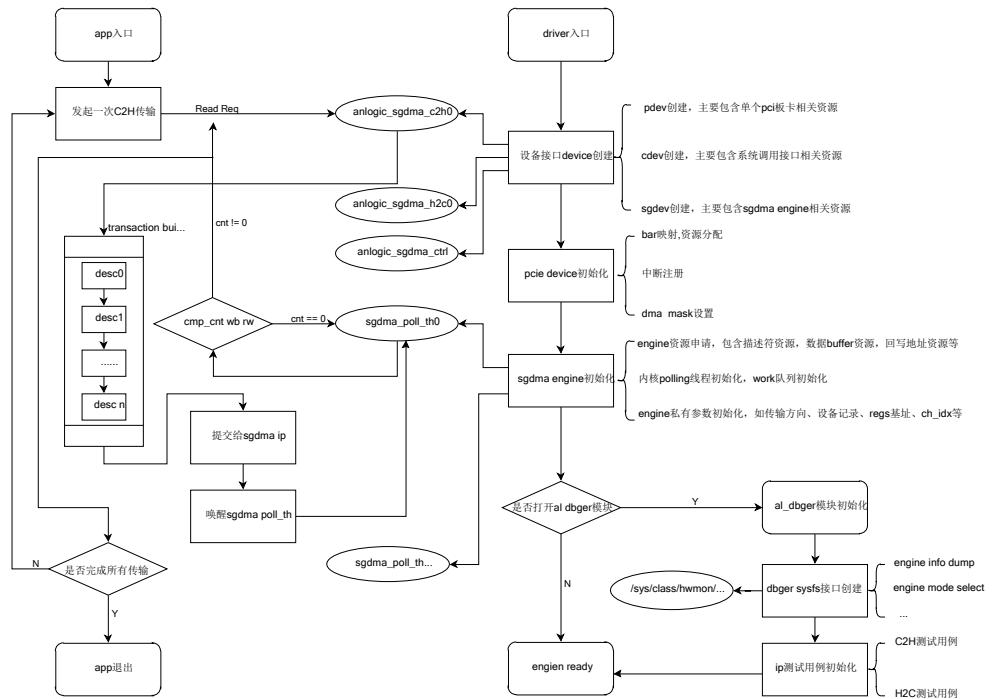


图 6-12 Linux 驱动 app 和 driver 的工作流程

6.4 Linux 上板验证

当 6.1 章提到的 ep_prj 工程编译完成，且没有 timing 问题，下载*.bit 到开发板，重新启动电脑进入 Ubuntu 系统，此时如果观察到 Demo 板有一个灯在闪烁（闪烁频率与 PCIe Gen Level 有关，如果



闪烁最慢为 Gen1，快些为协商后的 Gen2，代表 Link UP)。

编译完成后，若发现 PCIe Core 上的信号 Hold 不过，首先检查编译选项 fix_hold 是否打开，如果已经打开，请在工程目录找到 sgdma_subsys.v 模块，打开并找到 Hold 不过的信号，在路径上添加 BUF。以 xadm_client0_halt 信号 Hold 不过，参考代码如下。

```
PH1_LOGIC_BUF u_xadm_client0_halt (.i(xadm_client0_halt), .o(xadm_client0_halt_buf));
```

Link Up 正常后，打开终端，以管理员权限进入系统（命令 sudo -i），然后在终端上输入 lspci -xxx，找到自己设备的 Vendor ID 和 Class ID 看是否与配置一致，再看看协商后的 Link Width 与 Gen Level 是否一致，都一致代表设备枚举正常，如果没有一致，请参考“PCIe 培训相关 PPT”检查。

上述设备检测正常后，开始 cd 到 sw/sgdma_drv 目录，make 开始编译，由于 Linux 操作系统版本不一致，可能缺少某些库，导致编译失败，此时根据错误提示修正或者联系 FAE。编译成功后，insmod *.ko 安装，如果 lspci -xxx 没找到自己设备或 ID 不对，此处会安装不成功。然后，cd 到 sw/sgdma_app/src 目录，make 编译测试应用程序，将生成的 devmem2、dma_from_device、dma_to_device、performance、reg_rw 文件复制到 sw/sgdma_app/script 目录中。最后，cd 到 sw/sgdma_app/script 目录中，chmod +x *.sh 给目录中所有文件可执行权限，输入命令 ./complex_test.sh，窗口会列出本设计测试的 CASE，每个 CASE 带表的意义如下表 6-13 所示。

表 6-13 SGDMA 上板测试 CASE 说明

CASE 编号	测试内容	说明
1	Performance test for c2h	用于 C2H 方向测速。
2	Performance test for h2c	用于 H2C 方向测速。
3	One-way transmission for c2h	用于测试 C2H 方向数据传输。
4	One-way transmission for h2c	用于测试 H2C 方向数据传输。
5	Lookback test	用于数据环回测试。

上述测试默认为 PollMode 模式，中断模式需要把 anlogic_pci_lib.c 的 34 行 poll_mode 改成 0。下面以 PollMode 为例介绍各 CASE 项的几个关键参数，首先 target channel，从 0 开始，因为当前 IP 只有一个通道所有只能写 0；然后 transfer total size，单位为 byte，内部测试一般 128Mbyte，过小的 size 会导致性能过低；Enter a file name，可以不需要后缀名，以二进制文件存储；Enter a single descriptor size，单各描述符长度，一般为 4096byte。当选择 CASE5-Loopback 模式时，测试 OK 会打印 Test Pass，并校验数据；当选择 CASE1 和 CASE2 时，不会比较数据，且测试的是纯硬件性能；当选择 CASE3 或 CASE4 时，测试性能会比较低，因为添加了软件搬移数据，这里会涉及到客户软件性能的优化。

上述测试选项针对的是 AXI4-Stream 接口，如果为 AXI4-MM 接口，也可以用上述 CASE 选项，只不过默认起始读写地址为 0。

6.5 Windows 驱动介绍

当前的 windows driver，仅支持 PCIe sgdma ip，并且仅在 windows 10、windows11 平台上测试验证过。

6.5.1 文件目录

配套的 windows driver，有 driver 源码和 app 源码以及 target 测试文件。如下图所示：

名称	类型	压缩大小	密码保护	大小
anlogic pcie test app win	文件夹	测试app源码		
lib	文件夹			
target	文件夹	windows平台编译好的测试驱动和app		
.gitignore	GITIGNORE 文件			1 KB 否
AnlogicPcie.inf	安装信息			1 KB 否
AnlogicPcie.sln	SLN 文件			1 KB 否
AnlogicPcie.vcxproj	VCXPROJ 文件			2 KB 否
AnlogicPcie.vcxproj.filters	FILTERS 文件			1 KB 否
Device.c	C 文件			3 KB 否
Device.h	H 文件			1 KB 否
Driver.c	C 文件	驱动源码(包含lib目录的代码)		2 KB 否
Driver.h	H 文件			1 KB 否
pcie_common.h	H 文件			2 KB 否
PcieDeviceTab.c	C 文件			2 KB 否
PcieDeviceTab.h	H 文件			1 KB 否
Public.h	H 文件			1 KB 否
Queue.c	C 文件			3 KB 否
Queue.h	H 文件			1 KB 否
README.md	MD 文件			1 KB 否
Trace.h	H 文件			2 KB 否

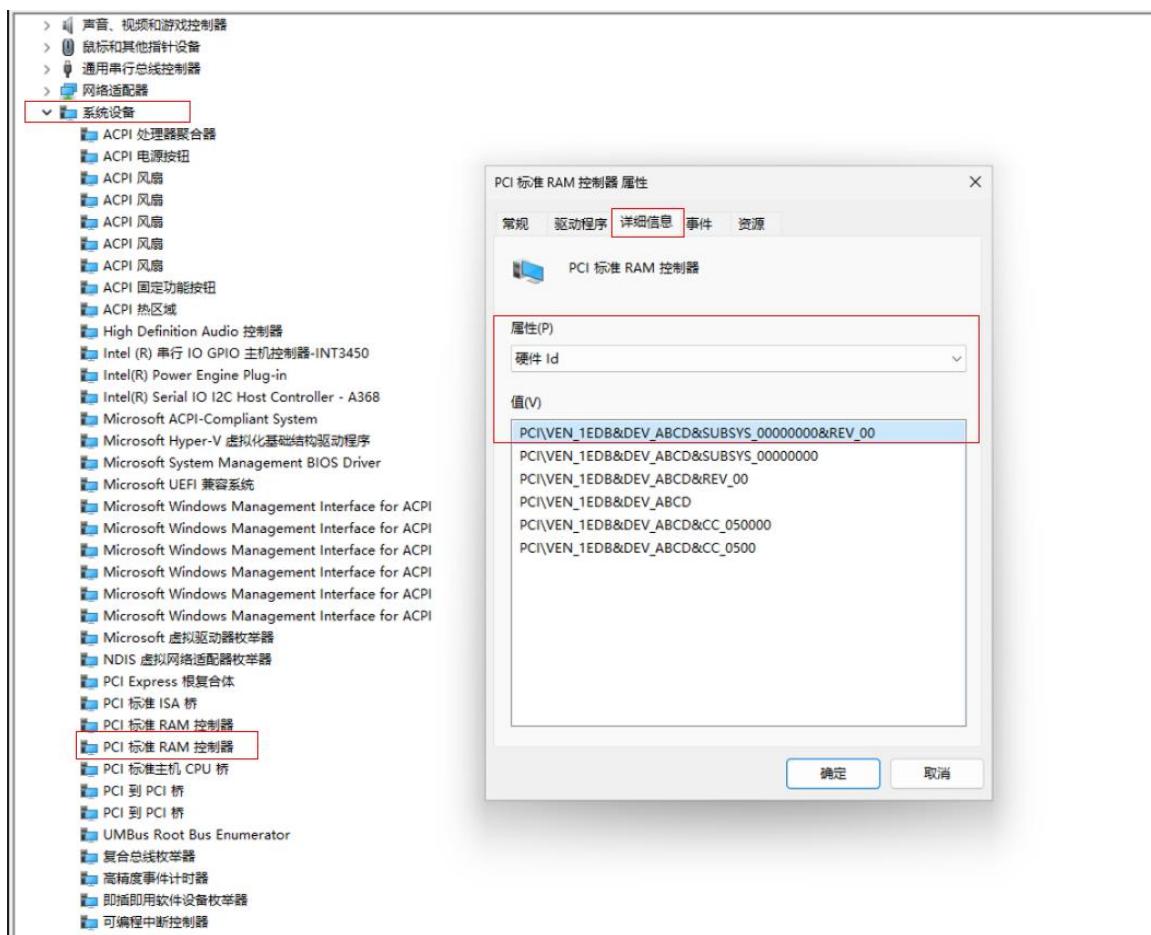
6.5.2 文件编译

其中 windows driver 文件采用 visual studio WDK 工具包开发，测试 app 也通过 visual studio 编译。关于 visual studio 和 WDK 的开发环境安装，请参照微软官方介绍。

6.5.3 上板验证

关机 host 系统，单独供电 demo 板，将 demo 板烧写好 sgdma 位流后，然后开机进入 windows 系统。当前的 fpga 板卡的 ID 会被默认识别为 PCI 标准 RAM 控制器如下图，如果硬件 ID 不是 1EDB:ABCD，可

能烧入的位流不是 SGDMA IP。



6.5.4 驱动安装

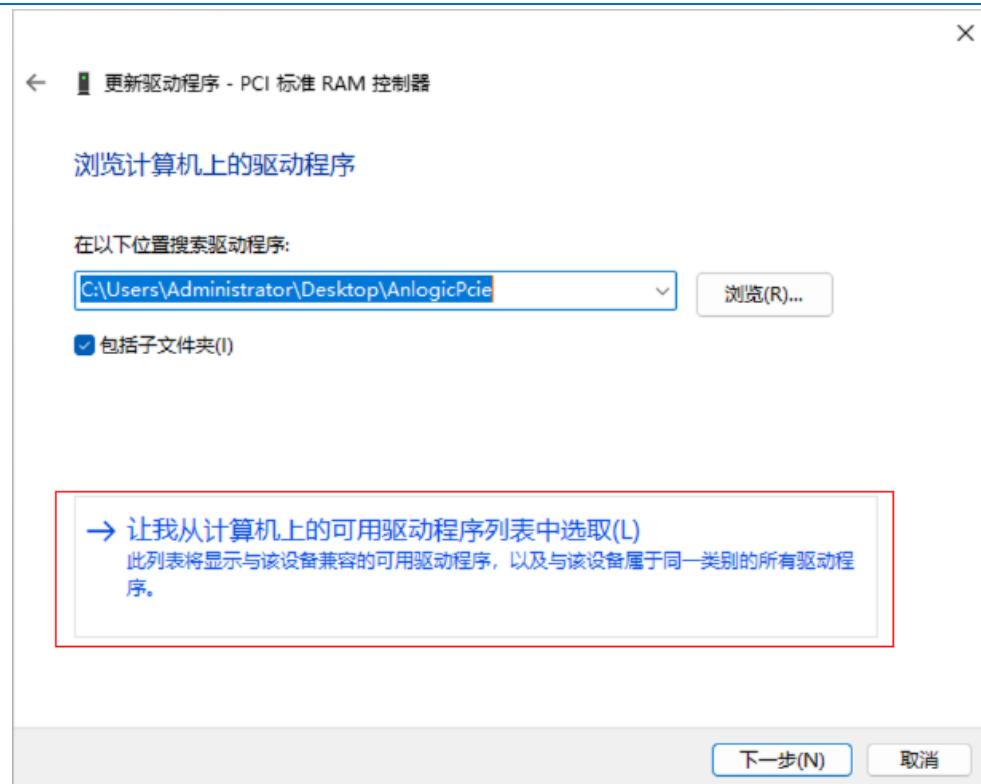
由于 windows 系统需要对驱动进行签名认证，因此对于编译出的驱动默认使用的是 wdk 测试签名，安装的时候需要将系统禁用驱动签名启动。一般步骤如下：

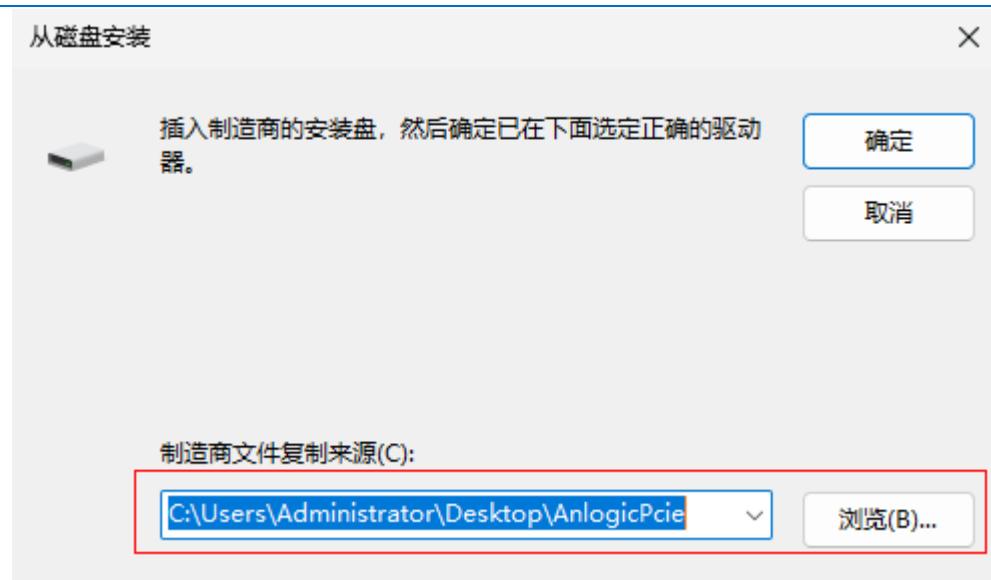
- 首先按下 Win 键+I 打开设置。
- 点击“更新与安全”，然后点击“恢复”。
- 在“高级启动”部分，点击“立即重启”。
- 计算机将进入高级启动选项。在这里，选择“疑难解答”。
- 接着选择“高级选项”。
- 在高级选项中，选择“启动设置”。
- 在启动设置中，点击“重新启动”。

- 计算机将重新启动并显示启动设置。 在这里，找到“禁用驱动程序签名强制执行”的选项，按其对应的数字键（一般是 7）。

当系统以禁用驱动签名方式启动后，在设备管理器中按照以下步骤安装驱动。











6.5.5 APP 测试

测试 app 帮助命令如下：

```
E:\>anlogic_pcnie_test_app_win.exe

Anlogic Driver Tester v1.0 (c) 2023, Anlogic Bsp

Usage: Anlogic Driver Tester [options] <filename>

Options:

-h, --help
    Display this help.

-d, --device <argument>
    Device name, One of: control | user | hc2_* | c2h_*.

-a, --address <argument>
    The target offset address of the read/write operation.

-s, --size <argument>
    Length of data to read/write (default: 4 bytes or whole file if '-f' flag is used).

-f, --file <argument>
    Use contents of file as input or write output into file.

-b, --binary
    Open file as binary.

-r, --read
    Read data from device.

-w, --write
    Write data to device, (if not use -f) Space separated bytes (big endian) in decimal or hex,
    Example: -w 1 2 3 4
    or: -w 0x11 0x22 0x33 0x44.

-v, --view
    More verbose output.
```

读 BAR0:

```
E:\>anlogic_pcnie_test_app_win.exe -d user -a 0 -s 0x100 -r -v
Devices found: 1
Device base path: Device node: reading from device...
Allocating host-side buffer of size 256, aligned to 4096 bytes
256 bytes received in 39us
software calc speed 0.051282s Gbit/s
0x0000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..... .
0x0010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..... .
0x0020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..... .
0x0030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..... .
0x0040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..... .
0x0050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..... .
0x0060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..... .
0x0070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..... .
0x0080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..... .
0x0090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..... .
0x00A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..... .
0x00B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..... .
0x00C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..... .
0x00D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..... .
0x00E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..... .
0x00F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```



写 BAR0 并再次读出确认:

```
E:\>anlogic_pcnie_test_app_win.exe -d user -a 0 -s 0x100 -w 0x11 0x22 0x33 0x44  
4 bytes written in 6us  
software calc speed 0.000000s Gbit/s  
  
E:\>anlogic_pcnie_test_app_win.exe -d user -a 0 -s 0x100 -r -v  
Devices found: 1  
Device base path: Device node: reading from device...  
Allocating host-side buffer of size 256, aligned to 4096 bytes  
256 bytes received in 37us  
software calc speed 0.053763s Gbit/s  
0x0000: 11 22 33 44 00 00 00 00 00 00 00 00 00 00 00 00 . "3D.....  
0x0010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..  
0x0020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..  
0x0030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..  
0x0040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..  
0x0050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..  
0x0060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..  
0x0070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..  
0x0080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..  
0x0090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..  
0x00A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..  
0x00B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..  
0x00C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..  
0x00D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..  
0x00E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..  
0x00F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
```

发起 C2H 传输:

```
E:\>anlogic_pcnie_test_app_win.exe -d c2h_0 -s 0x10000000  
16777216 bytes received in 5747us  
software calc speed 23.352646s Gbit/s
```

发起 H2C 传输:

```
E:\>anlogic_pcnie_test_app_win.exe -d h2c_0 -s 0x10000000  
16777216 bytes written in 7254us  
software calc speed 18.500696s Gbit/s
```

7 参考文档

1. AMBA® AXI Protocol Version:2.0 Specification;
2. AMBA® 4 AXI4-Stream Protocol Version:1.0 Specification;
3. 《UG713_安路科技 PH1A100 系列 FPGA PCI Express 用户手册》；
4. 《UG913_安路科技 PH1A 系列 FPGA PCIE 用户手册》；



5. 《APUG051_DDR3 Memory Controller_Reference_Design》；
6. 《UG1112_安路科技 PH2A 系列 FPGA PCIE 用户手册_v0.1. docx》；



版本信息

日期	版本	修订记录
2023/2/6	1. 0	首次发布中文版
2023/3/1	2. 0	增加支持 AXI4-MM 接口。增加支持 Windows 下 Modelsim 仿真，增加支持 180K 和 90K 芯片。
2023/12/22	3. 0	增加支持 PH2A 系列芯片，支持 SGDMA 中的 AXI4-MM 到 BRAM 的读写。修改文件目录说明、上板验证部分，增加易用性。新增 windows 驱动，优化提升传输效率。

版权所有© 2024 上海安路信息科技股份有限公司

未经本公司书面许可，任何单位和个人都不得擅自摘抄、复制、翻译本文档内容的部分或全部，并不得以任何形式传播。

免责声明

本文档并未授予任何知识产权的许可，并未以明示或暗示，或以禁止发言或其他方式授予任何知识产权许可；本文档仅为向用户提供使用器件的参考，协助用户正确地使用安路科技产品之用，其著作权归安路科技所有；本文档所展示的任何产品信息均不构成安路科技对所涉产品或服务作出任何明示或默示的声明或保证。

安路科技将不定期地对本文档进行更新、修订。用户如需获取最新版本的文档，可通过安路科技的官方网站（网址为：<https://www.anlogic.com>）自行查询下载，也可联系安路科技的销售人员咨询获取。