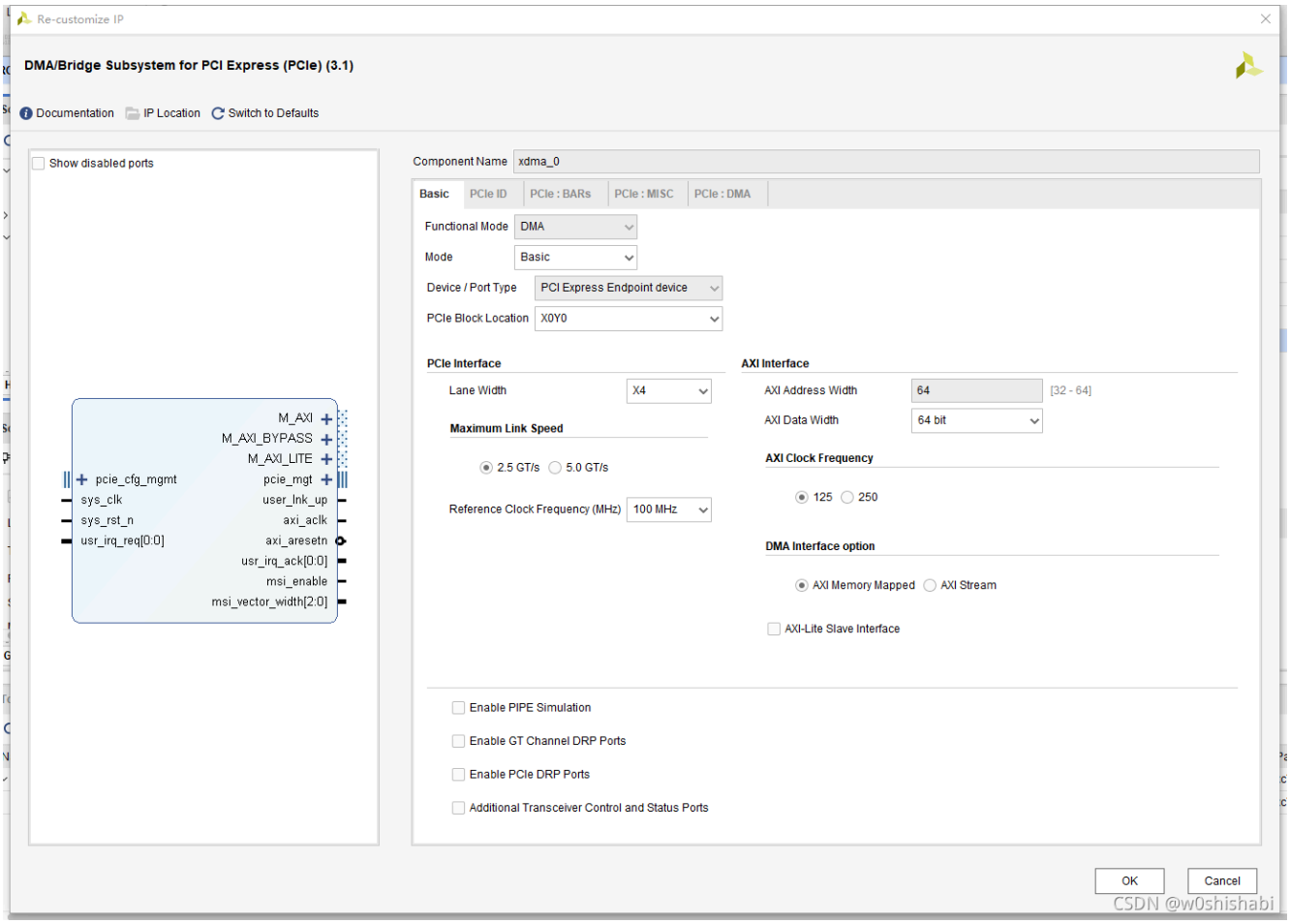
Xilinx XDMA 例程代码分析与仿真结果

PCIe学习笔记系列：

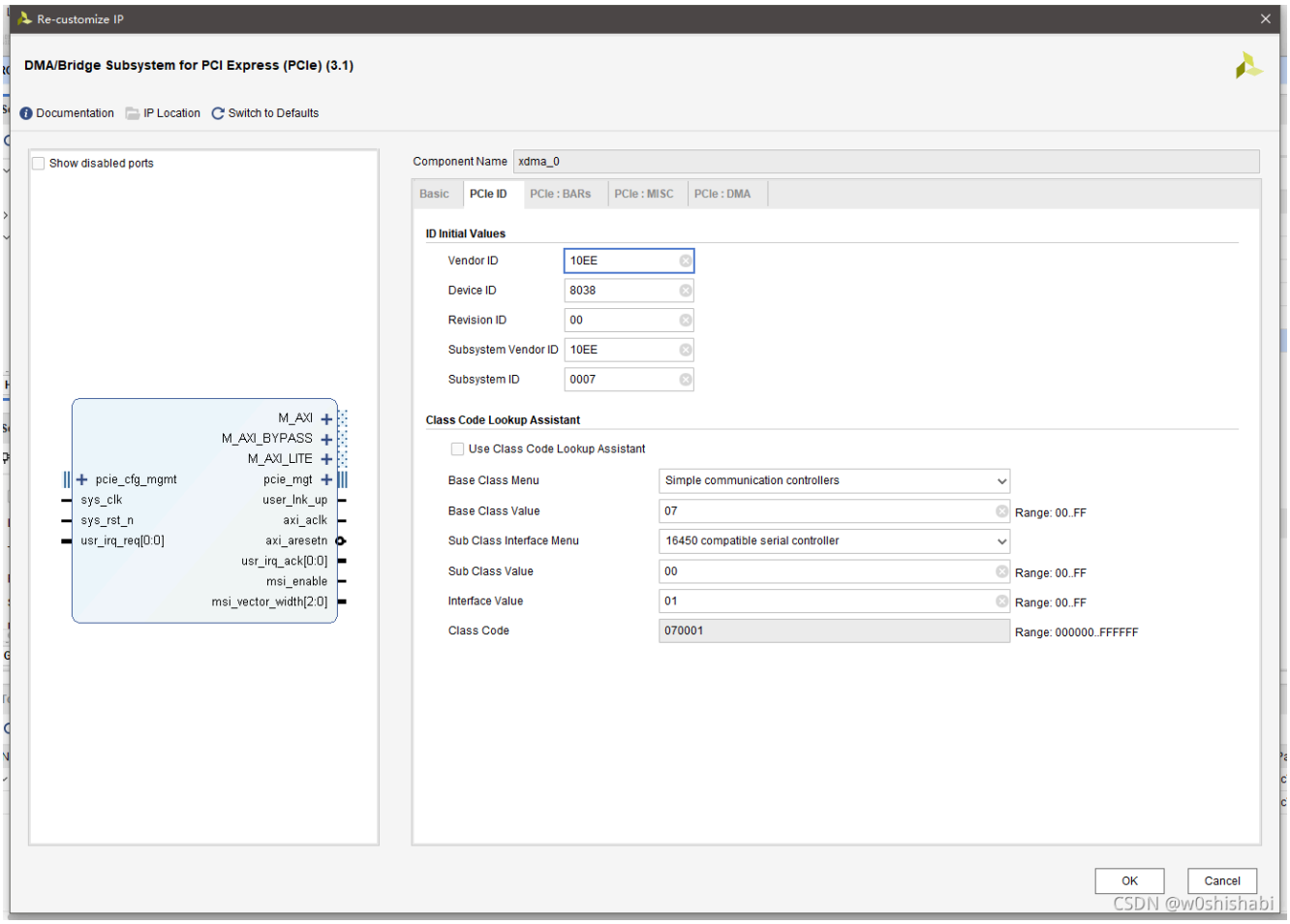
1. PCIe基础知识及Xilinx相关IP核介绍
   概念了解：简单学习PCIe的数据链路与拓扑结构，另外看看有什么相关的IP核。
2. 【PG054】7 Series Integrated Block for PCI Express IP核的学习
   基础学习：关于Pcie IP核的数据手册，学习PCIe相关的IP核的配置参数及其对应的含义。
3. Xilinx PCIe IP核示例工程代码分析与仿真
   基础学习：关于PCIe IP核的仿真，学习PCIe的配置流程以及应用过程。
4. Xilinx XDMA 例程代码分析与仿真结果
   应用学习：关于Xilinx PCIe DMA IP核的仿真，学习 PCIe DMA 的配置过程以及具体的数据传输流程。
5. XDMA linux平台调试过程记录
   应用学习：关于XDMA的实际调试过程，可在此基础上定制自己的需求。

1 IP核的配置

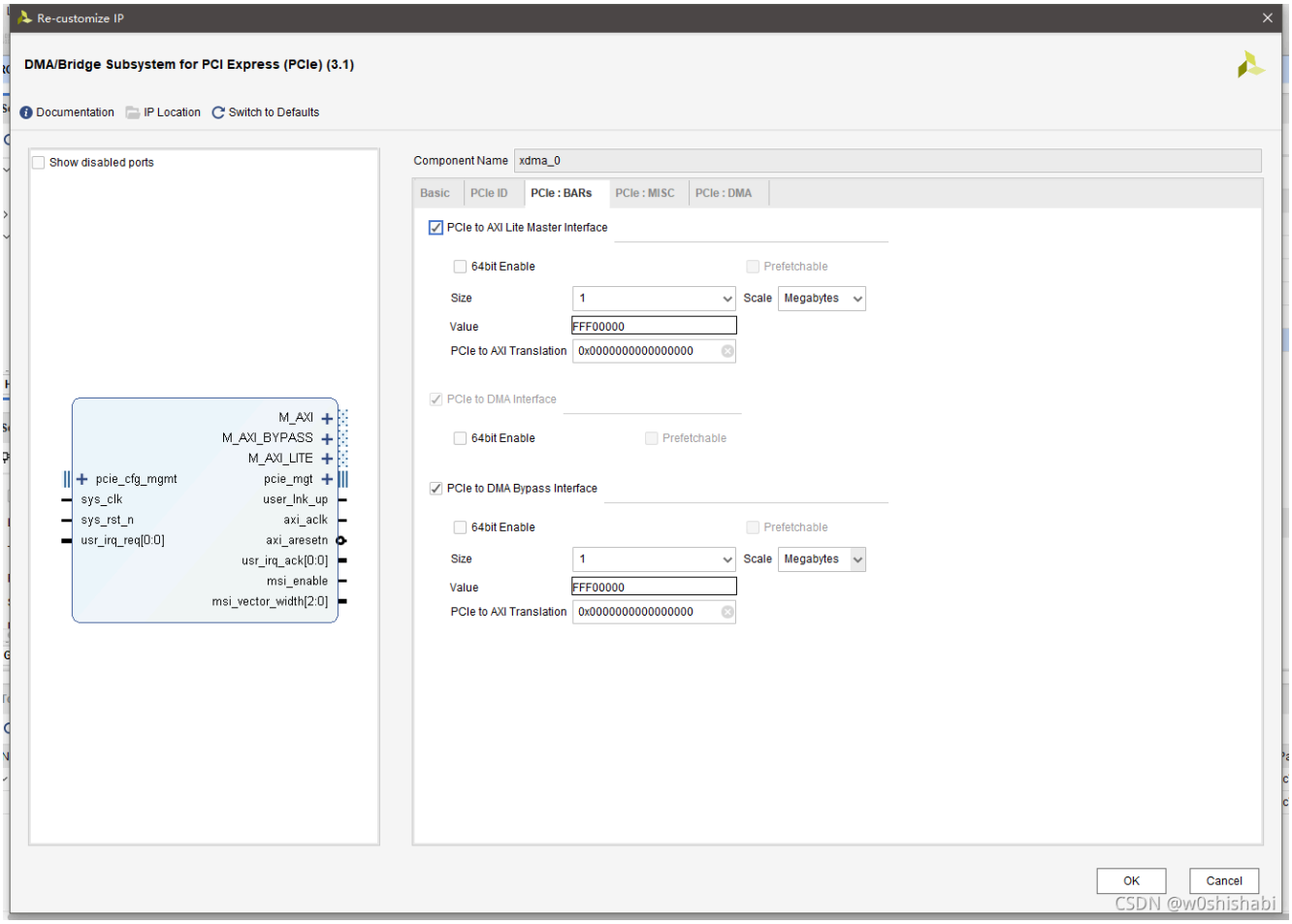
1.1 Basic



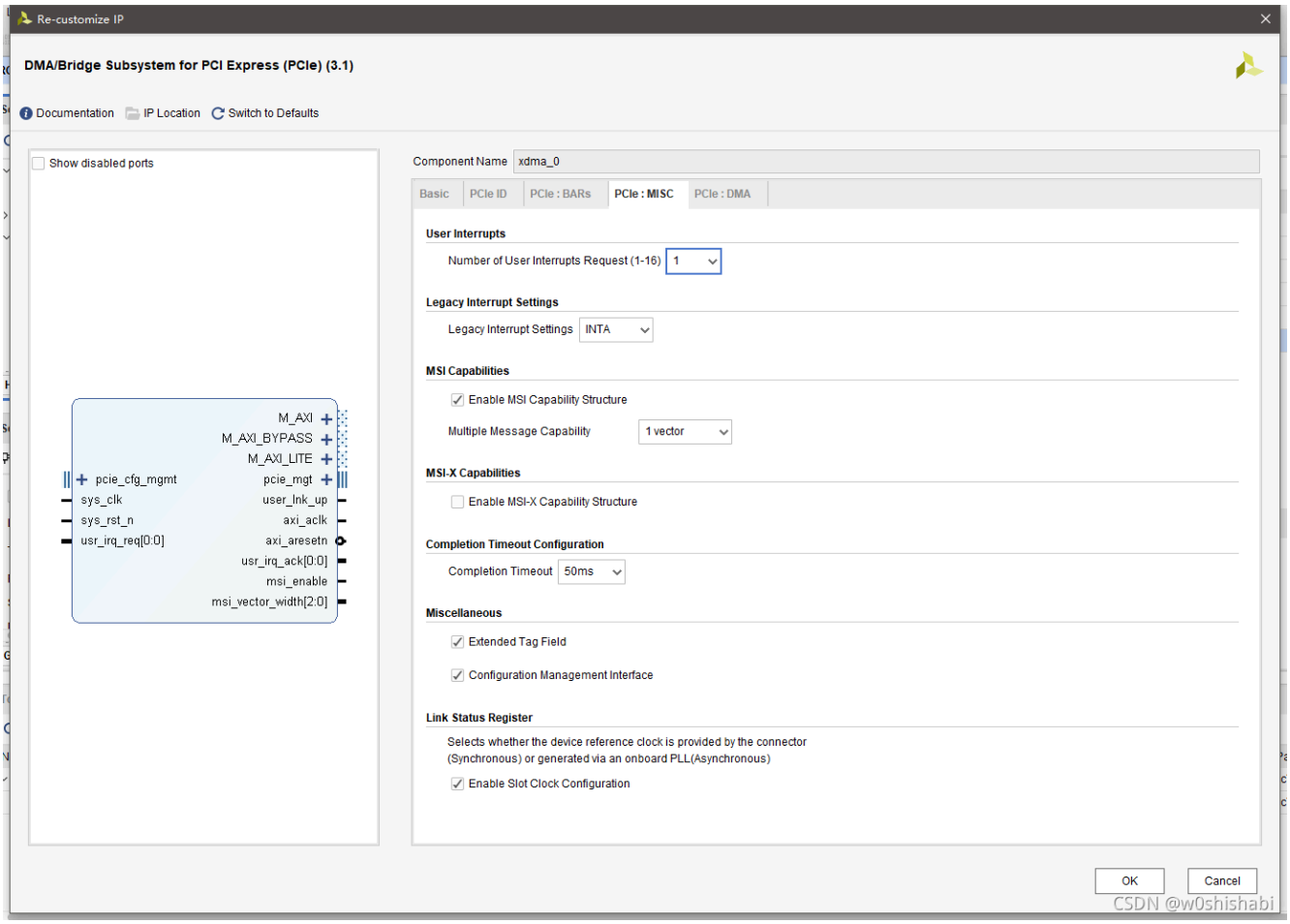
1.2 PCIe ID



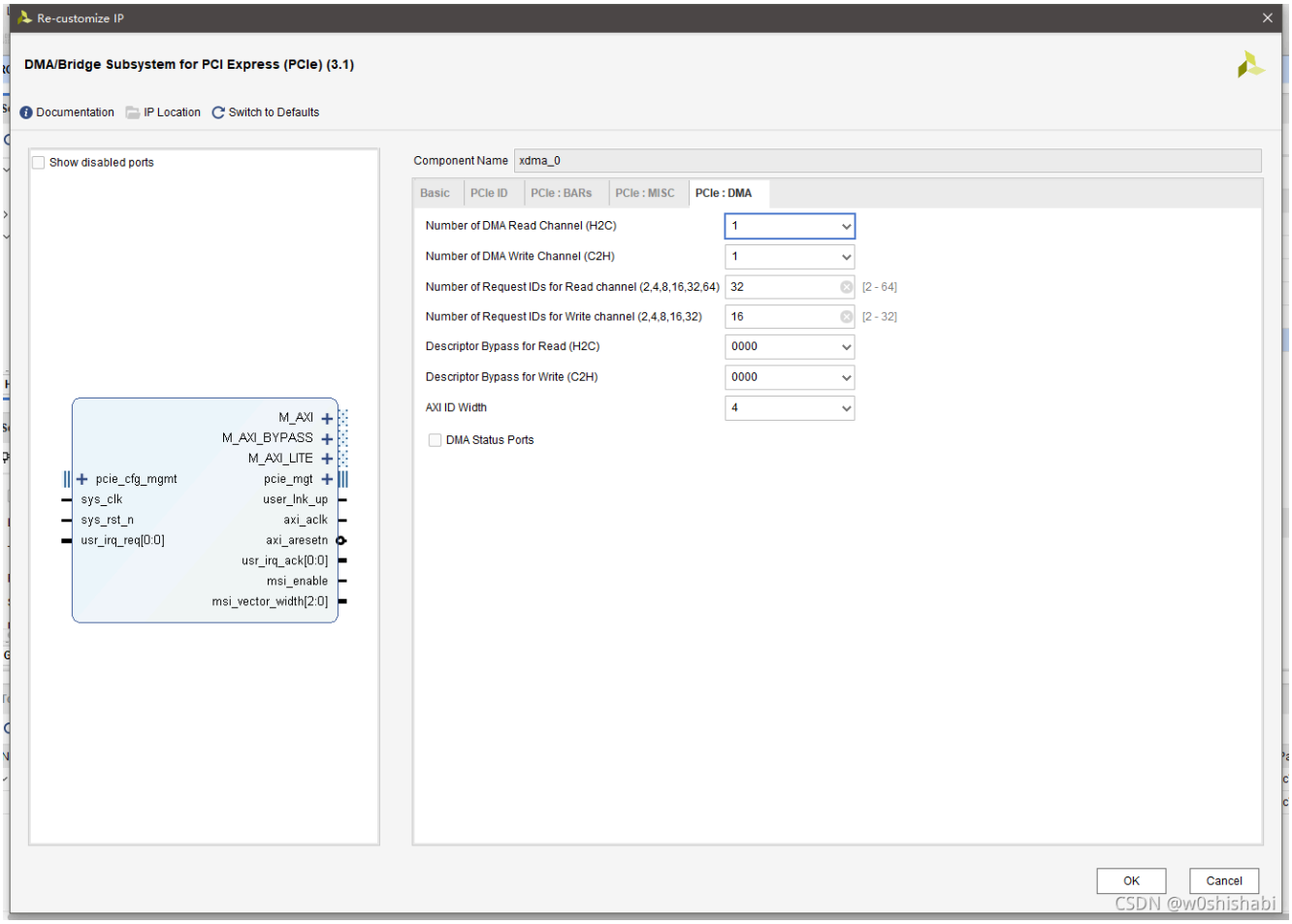
1.3 PCIe:BARs



1.4 PCIe:MISC


Re-customize IP
DMA/Bridge Subsystem for PCI Express (PCIe) (3.1)
Documentation
IP Location
Switch to Defaults
Show disabled ports
M_AXI
M_AXI_BYPASS
M_AXI_LITE
pcie_cfg_mgmt
pcie_mgt
sys_clk
user_lnk_up
sys_rst_n
axi_aclk
usr_irq_req[0:0]
axi_aresetn
usr_irq_ack[0:0]
msi_enable
msi_vector_width[2:0]
Component Name
xdma_0
Basic
PCIe ID
PCIe : BARs
PCIe : MISC
PCIe : DMA
User Interrupts
Number of User Interrupts Request (1-16)
1
Legacy Interrupt Settings
Legacy Interrupt Settings
INTA
MSI Capabilities
Enable MSI Capability Structure
Multiple Message Capability
1 vector
MSI-X Capabilities
Enable MSI-X Capability Structure
Completion Timeout Configuration
Completion Timeout
50ms
Miscellaneous
Extended Tag Field
Configuration Management Interface
Link Status Register
Selects whether the device reference clock is provided by the connector
(Synchronous) or generated via an onboard PLL(Asynchronous)
Enable Slot Clock Configuration
OK
Cancel
CSDN @w0shishabi

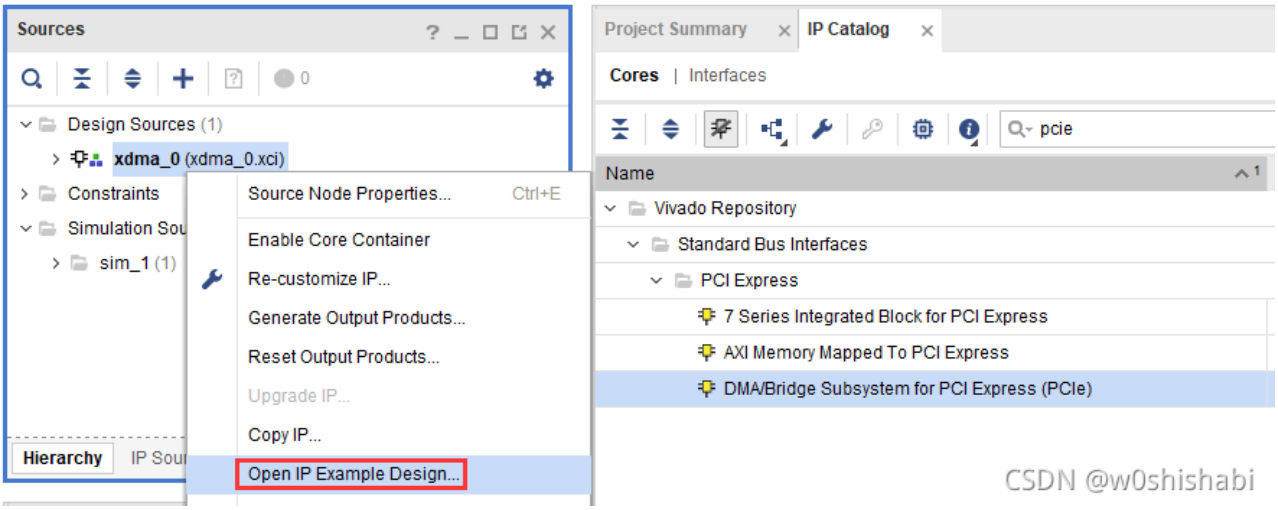
1.5 PCIe:DMA


Re-customize IP
DMA/Bridge Subsystem for PCI Express (PCIe) (3.1)
Documentation
IP Location
Switch to Defaults
Show disabled ports
M_AXI
M_AXI_BYPASS
M_AXI_LITE
pcie_cfg_mgmt
pcie_mgt
sys_clk
user_lnk_up
sys_rst_n
axi_aclk
usr_irq_req[0:0]
axi_aresetn
usr_irq_ack[0:0]
msi_enable
msi_vector_width[2:0]
Component Name
xdma_0
Basic
PCIe ID
PCIe : BARs
PCIe : MISC
PCIe : DMA
Number of DMA Read Channel (H2C)
1
Number of DMA Write Channel (C2H)
1
Number of Request IDs for Read channel (2,4,8,16,32,64)
32
[2 - 64]
Number of Request IDs for Write channel (2,4,8,16,32)
16
[2 - 32]
Descriptor Bypass for Read (H2C)
0000
Descriptor Bypass for Write (C2H)
0000
AXI ID Width
4
DMA Status Ports
OK
Cancel
CSDN @w0shishabi

2 生成IP核的例程

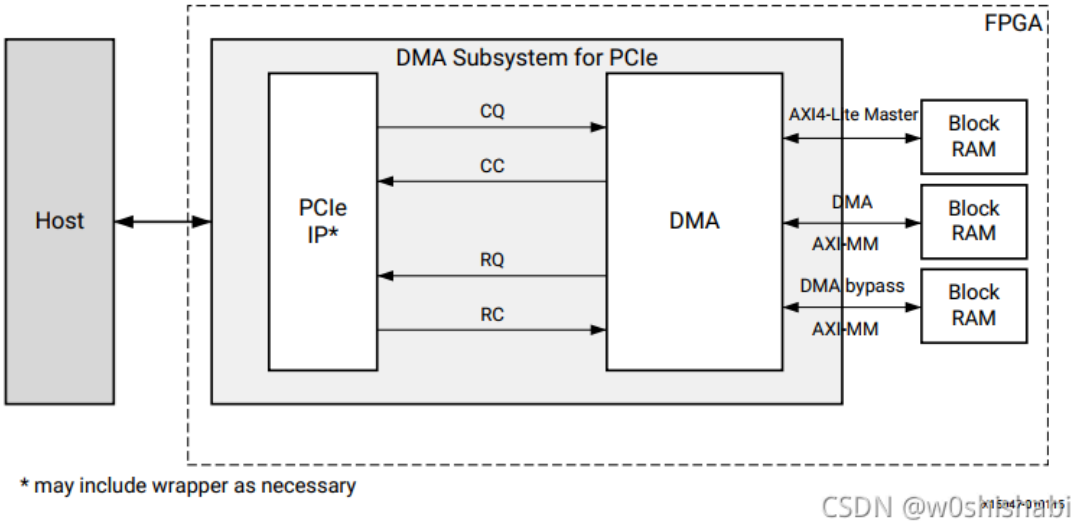
2.1 例程结构

配置好IP核后右击IP核选择Open Ip Exanple Design...打开例程



例程的结构：

Figure 22: **AXI-MM Example with PCIe to DMA Bypass Interface and PCIe to AXI-Lite Master Enabled**



当然，右边的接口根据自己是否使能会有变化。因为用于仿真，所以我们在BAR中同时勾选了AXI4-Lite master interface和DMA Bypass interface，所以仿真工程同时会有这三个接口。

2.2 例程的仿真

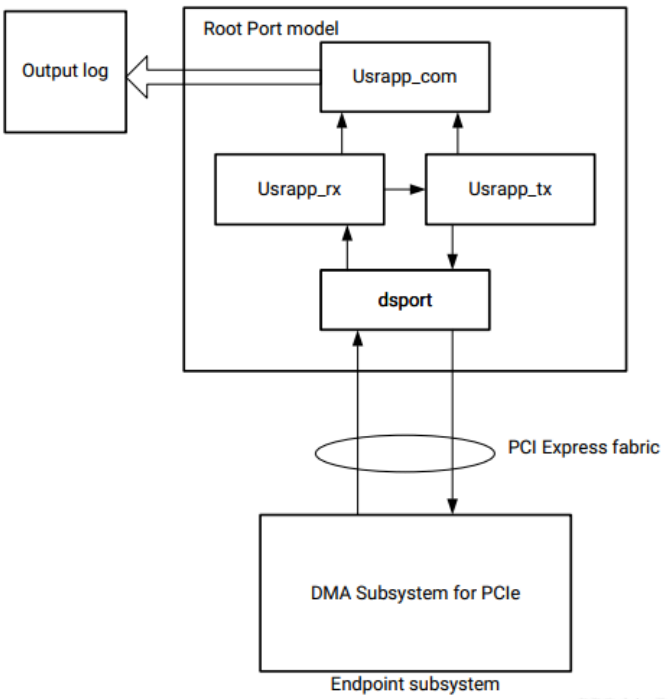
生成的例程的TestBench环境中有一个PCI Express Root Port 模型，可以与提供的PIO（Programmed Input/Output，PIO）设计或与个人设计一起使用。模型的目的是为下游提供源机制的TLP流激励用户的设计，以及在仿真环境中从用户设计接收上游PCI Express TLP流的机制。初始化核心配置空间、创建TLP事务、生成TLP日志以及提供创建和验证测试的接口的所有重要工作都已完成。不需要自己搭建testbench。

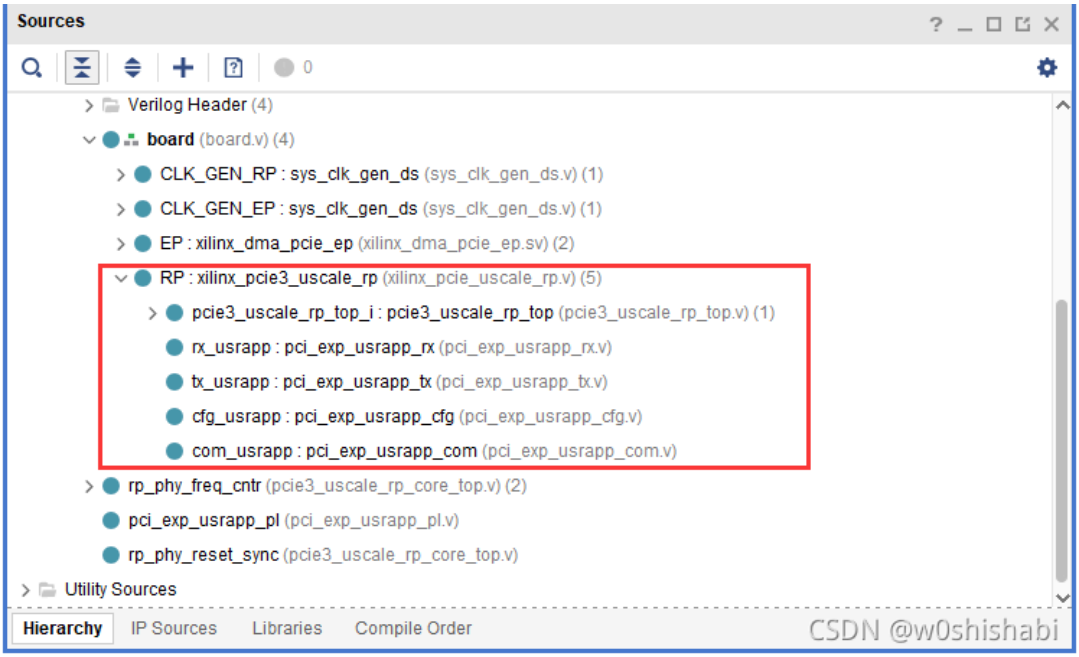
Root Port 模型包括：

- 测试编程接口(TPI)，激励端点设备PCI Express。
- 演示如何使用测试程序TPI的示例测试。
- 所有 Root Port模型组件的Verilog源代码，允许自定义test bench。

模型结构及与PCIe DMA Subsystem 的连接：

Figure 32: **Root Port Module with DMA Subsystem for PCIe**





大致的结构是usrapp_tx和 usrapp_rx通过dsport与DUT进行TLP包的收发。DUT中包含PCIe DMA Subsystem。

dsport则模拟数据链路层以及物理层在数据通信过程中的相关处理。

usrapp_com模块中则是一些共享的逻辑，比如TLP处理以及log文件的输出。

因为先安装的Vivado 2017.1（XDMA IP核版本为3.1），后安装的2018.3（对应版本为4.1），导致我的2018.3和Modelsim联合仿真环境有问题，所以使用2017.1版本进行仿真。

2.2.1 测试用例

DMA Subsystem for PCIe 可以配置成AXI4 内存映射（AXI-MM）或者AXI4-Stream（AXI_ST）接口。仿真测试用例通过读取配置寄存器来判断是配置的哪个接口。然后基于AXI的设置，执行对应的仿真。

测试用例名称	描述
Dma_test0	AXI-MM接口的仿真，读取Host的内存进行然后写入道block RAM中（H2C）。然后读取block RAM中的数据然后写入到Host内存（C2H）。这个测试用例最后会比较数据的正确性

测试用例名称	描述
Dma_stream0	AXI4-Stream接口仿真。从Host内存读取数据并发送到AXI4-Stream用户接口(H2C)，然后数据被环回到主机内存(C2H)。

H2C和C2H的仿真模型有64字节的传输大小限制。

2.2.2 仿真过程

官方手册描述的仿真步骤：

AXI4内存映射接口

首先，测试用例启动H2C引擎，H2C引擎从Host内存中读取数据然后写到用户端的Block RAM。然后，测试用例启动C2H引擎，从Block RAM 中读取数据然后写到Host 内存中。仿真的步骤：

1. 测试用例为H2C引擎设置一个描述符。
2. H2C描述符给出数据长度64字节、源地址(Host)和目的地址(Card)。
3. 将数据(增量数据)写入源地址空间。
4. 测试用例也为C2H引擎设置了一个描述符。
5. C2H描述符提供数据长度64字节、源地址(Card)和目的地址(Host)。
6. PIO写入H2C描述符启动寄存器。
7. PIO写入H2C控制寄存器以启动H2C传输。
8. DMA传输将数据从host源地址传输到block RAM目的地址。
9. 然后测试用例启动C2H传输。
10. PIO写入C2H描述符启动寄存器。
11. PIO写入C2H控制寄存器以启动C2H传输。
12. DMA传输将数据从block RAM源地址传输到host目的地址。
13. 测试用例比较数据的正确性。
14. 测试用例检查 H2C和C2H描述符完成计数(值为1)。

AXI-Stream 接口

对于 AXI-Stream，示例设计是一个环回设计。首先，测试用例启动C2H引擎。C2H引擎等待H2C引擎传输的数据。然后，测试用例启动H2C引擎。H2C引擎从host读取数据并发送到Card, Card被循环回给C2H引擎。然后，C2H引擎获取数据，并将数据写回host内存。下面是仿真步骤:

1. 测试用例为H2C引擎设置一个描述符。
2. H2C描述符提供数据长度64字节、源地址(Host)和目的地址(Card)。
3. 将数据(增量数据)写入源地址空间。
4. 测试用例也为C2H引擎设置了一个描述符。
5. C2H描述符给出数据长度64字节、源地址(Card)和目的地址(host)。
6. PIO写入C2H描述符启动寄存器。
7. PIO写入C2H控制寄存器，首先启动C2H传输。
8. C2H引擎启动，等待来自H2C端口的数据。
9. PIO写入H2C描述符启动寄存器。
10. PIO写入H2C控制寄存器以启动C2H传输。
11. H2C引擎从主机源地址获取数据到Card目的地址。
12. 数据环回到C2H引擎。
13. C2H引擎从Card获取数据，然后发送和写入host内存目标地址。
14. 测试用例检查H2C和C2H描述符完成计数(值为1)。

2.2.3 Descriptor Bypass 模式

当H2C和C2H的Channel 0 选择为描述符旁路选项时，可以进行描述符旁路模式的Vivado仿真。示例设计产生了一个描述符准备泵在描述符旁路模式接口。
当传输开始时，一个H2C和一个C2H描述符在描述符旁路接口中传输，然后按照上面章节的解释执行DMA传输。描述符仅为64字节的传输设置。

2.4 测试任务

名称	描述
TSK_INIT_DATA_H2C	这个任务为H2C引擎生成一个描述符，并在主机内存中初始化源数据。
TSK_INIT_DATA_C2H	这个任务为C2H引擎生成一个描述符。

名称	描述
TSK_XDMA_REG_READ	这个任务读取DMA子系统的PCIe寄存器。
TSK_XDMA_REG_WRITE	这个任务写DMA子系统的PCIe寄存器。
COMPARE_DATA_H2C	这个任务比较host内存中的源数据和写入到block RAM的目标数据。这个任务在AXI4 Memory Mapped仿真中使用。
COMPARE_DATA_C2H	该任务将hose内存中的原始数据与C2H引擎写入host的数据进行比较。这个任务在AXI4 Memory Mapped仿真中使用。
TSK_XDMA_FIND_BAR	该任务在不同启用的bar之间查找XDMA配置空间。(BAR0 BAR6)
TSK_WRITE_CFG_DW	对RP的配置空间进行写操作

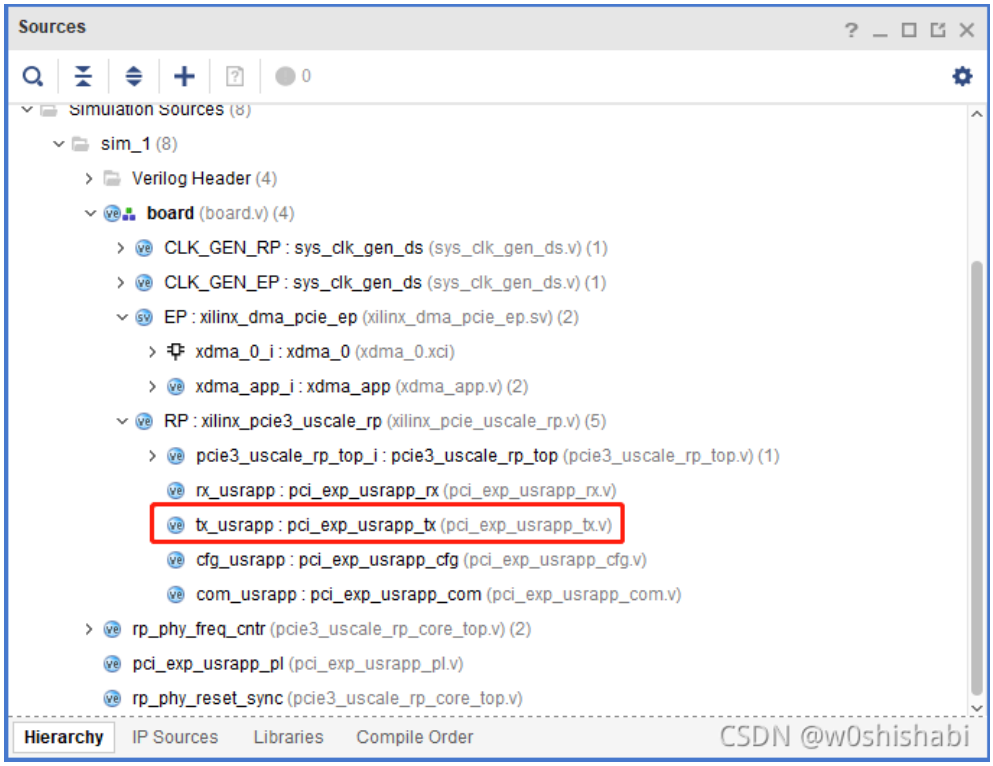
值得一提的是，我们在Xilinx PCIe IP核示例工程代码分析与仿真中已经了解过，TSK_TX_TYPE0/1_CONFIGURATION_WRITE是对EP的配置空间进行写操作。而TSK_WRITE_CFG_DW是对RP的配置空间进行写操作，同理，读操作TSK命名格式也是类似的。

3 仿真结果

3.1 tx_usrapp:pci_exp_usrapp_tx中的初始化

与Xilinx PCIe IP核示例工程代码分析与仿真关于*7 Series FPGAs Integrated Block for PCI Express* IP核的仿真分析过程类似。分析验证一下上述的仿真步骤。

首先查看测试用例的名称，文件层次为board/RP/tx_usrapp：



其中有一个initial块：

```
initial begin
  dmaTestDone         = 0;
  pfIndex             = 0;
  pfTestIteration     = 0;

  expect_status       = 0;
  expect_finish_check = 0;
  testError           = 1'b0;
  // Tx transaction interface signal initialization.
  pcie_tlp_data       = 0;
  pcie_tlp_rem        = 0;

  EP_BUS_DEV_FNS      = EP_BUS_DEV_FNS_INIT;
  // Payload data initialization.
  TSK_USR_DATA_SETUP_SEQ;
```

```
  board.RP.tx_usrapp.TSK_SIMULATION_TIMEOUT(10050);
  board.RP.tx_usrapp.TSK_SYSTEM_INITIALIZATION;
  board.RP.tx_usrapp.TSK_BAR_INIT;

  // Find which BAR is XDMA BAR and assign 'xdma_bar' variable
  board.RP.tx_usrapp.TSK_XDMA_FIND_BAR;

  if ($value$plusargs("TESTNAME=%s", testname))
      $display("Running test {%0s}......", testname);
  else begin
      // decide if AXI-MM or AXI-ST
      board.RP.tx_usrapp.TSK_XDMA_REG_READ(16'h00);
      if (P_READ_DATA[15] == 1'b1) begin
   testname = "dma_stream0";
          $display("*** Running XDMA AXI-Stream test {%0s}......", testname);
      end
      else begin
         testname = "dma_test0";
         $display("*** Running XDMA AXI-MM test {%0s}......", testname);
      end
  end

  //Test starts here
  if (testname == "dummy_test") begin
      $display("[%t] %m: Invalid TESTNAME: %0s", $realtime, testname);
      $finish(2);
  end
  `include "tests.vh"
  else begin
      $display("[%t] %m: Error: Unrecognized TESTNAME: %0s", $realtime, testname);
      $finish(2);
  end


end
```

line15: TSK_USR_DATA_SETUP_SEQ 负载数据初始化

将DATA_STORE中的数据初始化位0~4095。

line17: TSK_SIMULATION_TIMEOUT

设置board.RP.rx_usrapp.sim_timeout的值为10050

line18: TSK_SYSTEM_INITIALIZATION 对RP进行配置，等待系统初始化完成以及建立链路

- 等待复位完成
- 等待链路建立

在同一个源文件找到TSK_SYSTEM_INITIALIZATION 的定义进行分析：

1. 首先TSK_WRITE_CFG_DW(32’ h01, 32’ h00000007, 4’ h1)，对RP配置空间地址32’ h01写32’ h00000007，4’ h1是字节选通，含义是选通低字节。

首先回顾一下Xilinx type0(桥设备为type1，非桥设备为type0)PCIe设备的配置空间：

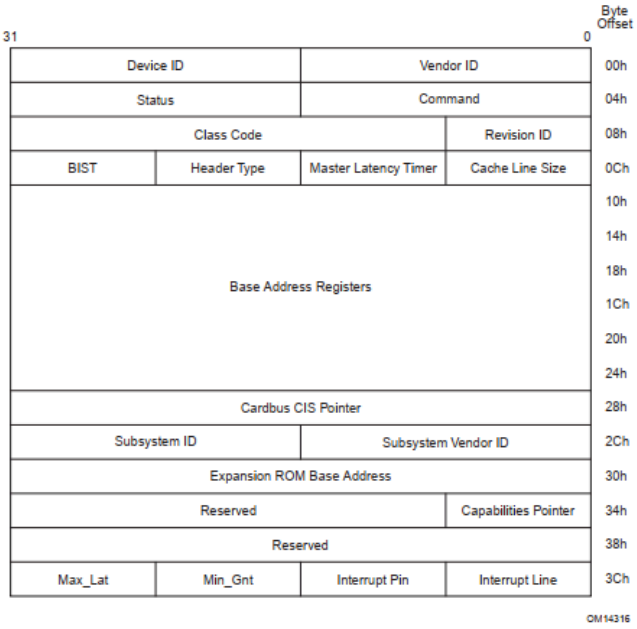


Figure 7-5: Type 0 Configuration Space Header

注意一下，数据位宽是32位，地址与EP写配置空间命令一样，指的是DW地址。

所以指令的作用是：

表 2-4 Command 寄存器

位	描 述
0 1	I/O Space 位，该位表示 PCI 设备是否响应 I/O 请求，为 1 时响应，为 0 时不响应。如果 PCI 设备支持 I/O 地址空间，系统软件会将该位置 1。复位值为 0
1 1	Memory Space 位，该位表示 PCI 设备是否响应存储器请求，为 1 时响应，为 0 时不响应。如果 PCI 设备支持存储器地址空间，系统软件会将该位置 1。复位值为 0
2 1	Bus Master 位。该位表示 PCI 设备是否可以作为主设备，为 1 时 PCI 设备可以作为主设备，为 0 时不能。复位值为 0
3 0	Special Cycle 位，该位表示 PCI 设备是否响应 Special 总线事务，为 1 时响应，为 0 时不响应。PCI 设备可以使用 Special 总线事务，将一些信息广播发送到多个目标设备，Specail 总线事务不能穿越 PCI 桥。如果一个 PCI 设备需要将 Special 总线事务发送到 PCI 桥之下的总线时，必须使用 Type 01h 配置周期。PCI 桥可以将 Type 01h 配置周期转换为 Special 周期。该位的复位值为 0
4	Memory Write and Invalidate 位，该位表示 PCI 设备是否支持 Memory Write and Invalidate 总线事务，为 1 时支持，为 0 时不支持。许多低端的 PCI 设备不支持这种总线事务。该位对 PCIe 设备无意义
5	VGA Palette Snoop 位。该位为 1 时支持 Palette Snoop 功能，为 0 时不支持
6	Parity Error Response 位，复位值为 0。该位为 1，而且 PCI 设备在传送过程中出现奇偶校验错误时，PCI 设备将 PERR#信号设置为 1；该位为 0 时，即便出现奇偶检验错误，PCI 设备也仅会将 Status 寄存器的“Detected Parity Error”位置 1
8	SERR# Enable 位，复位值为 0。该位为 1，而且 PCI 设备出现错误时，将使用 SERR#信号，将这个错误信息发送给 HOST 主桥，为 0 时，不能使用 SERR#信号
9	Fast Back-to-Back 位。该位为 1 时，PCI 设备使用 Fast Back-to-Back（快速背靠背）总线周期，这种周期是一种提高传送效率的方法。但并不是所有的 PCI 设备都支持 Fast Back-to-Back 传送周期。该位的复位值为 0
10	Interrupt Disable 位，复位值为 0。该位为 1 时，PCI 设备不能通过 INTx 信号向 HOST 主桥提交中断请求，为 0 时可以使用 INTx 信号提交中断请求。当 PCI 设备使用 MSI 中断方式提交中断请求时，该位将被置为 1

最主要的作用是配置为主设备。

2. TSK_READ_CFG_DW(12’h068/4)

读取字节地址为0x68的配置空间寄存器，字段为Device Control Register。

其他字节可以看Xilinx PCIe IP核示例工程代码分析与仿真的“2.2.2 PCIe配置空间”，这里就不重复贴图了。

读出的值cfg_rd_data为32’h0。

3. cfg_usrapp.TSK_WRITE_CFG_DW(DEV_CTRL_REG_ADDR/4,(board.RP.cfg_usrapp.cfg_rd_data | (DEV_CAP_MAX_PAYLOAD_SUPPORTED * 32)) , 4’h1)

其中board.RP.cfg_usrapp.cfg_rd_data | (DEV_CAP_MAX_PAYLOAD_SUPPORTED * 32)=32’h0 | 32’40 = 32’h40

作用：在原配置的基础上设置Max_Payload_Size的值。

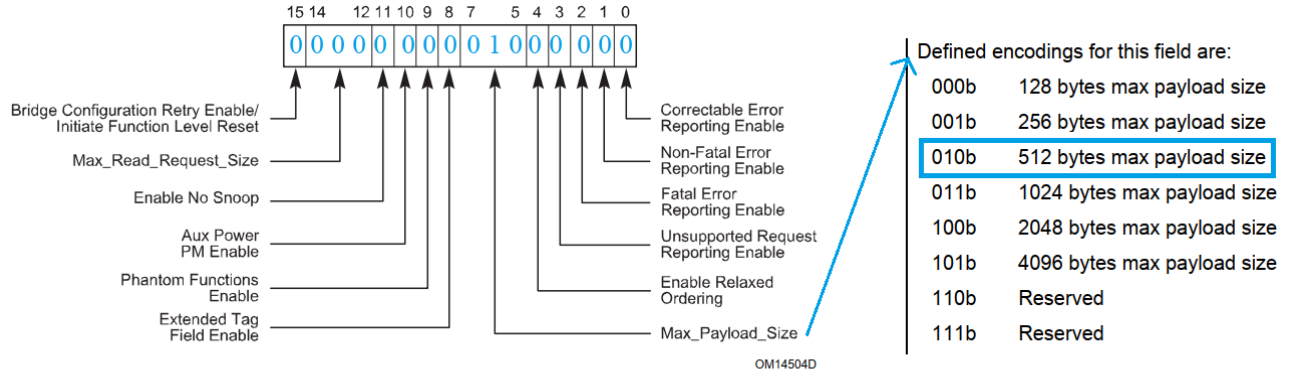
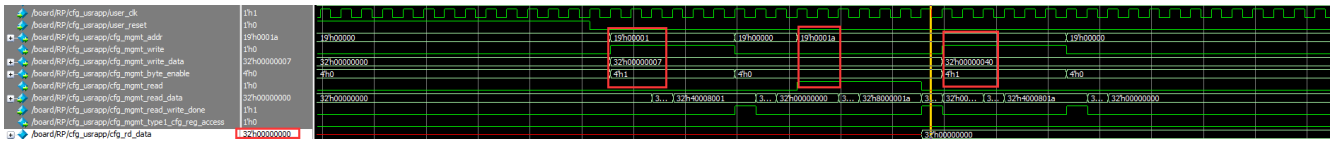


Figure 7-14: Device Control Register

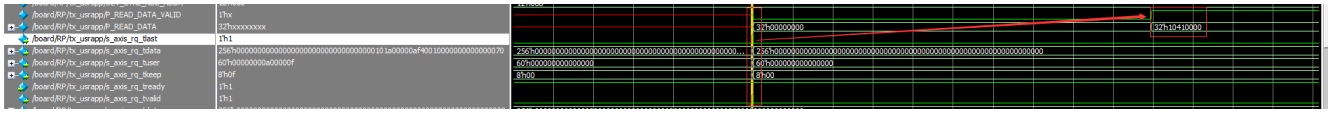
可见，设置Max_Payload_Size = 512Byte

以上3步的波形：

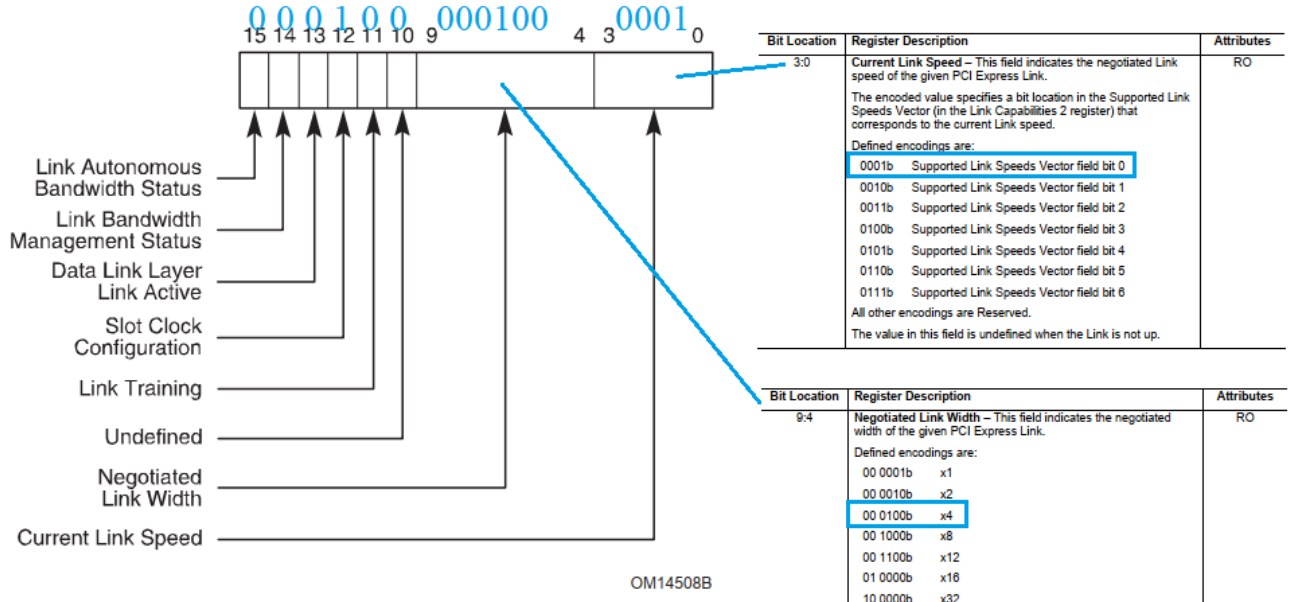


4. TSK_SYSTEM_CONFIGURATION_CHECK

这一步使用TSK_TX_TYPE0_CONFIGURATION_READ(DEFAULT_TAG, 12’ h070, 4’ hF);读取字节起始地址为12’ h070的DW，字段为Link Control和Link Status。读取到的数据为32’ h10410000。



此DW的bit[19：16]代表链路速度（我们设置的为2.5GT/s）。bit[23:20]代表链路宽度（我们设置为4）。



Bit Location	Register Description	Attributes
3:0	**Current Link Speed** – This field indicates the negotiated Link speed of the given PCI Express Link. The encoded value specifies a bit location in the Supported Link Speeds Vector (in the Link Capabilities 2 register) that corresponds to the current Link speed. Defined encodings are: 0001b Supported Link Speeds Vector field bit 0 0010b Supported Link Speeds Vector field bit 1 0011b Supported Link Speeds Vector field bit 2 0100b Supported Link Speeds Vector field bit 3 0101b Supported Link Speeds Vector field bit 4 0110b Supported Link Speeds Vector field bit 5 0111b Supported Link Speeds Vector field bit 6 All other encodings are Reserved. The value in this field is undefined when the Link is not up.	RO

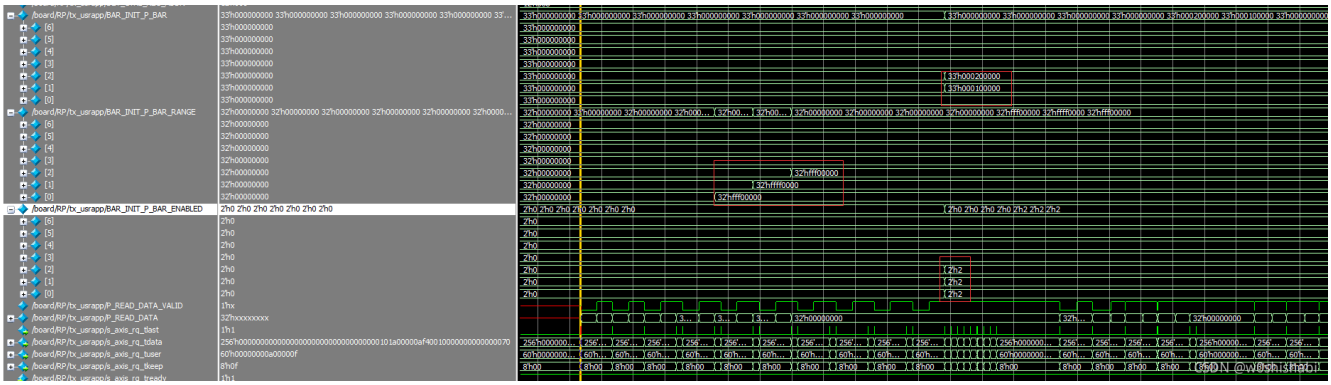
Bit Location	Register Description	Attributes
9:4	**Negotiated Link Width** – This field indicates the negotiated width of the given PCI Express Link. Defined encodings are: 00 0001b x1 00 0010b x2 00 0100b x4 00 1000b x8 00 1100b x12 01 0000b x16 10 0000b x32 All other encodings are Reserved. The value in this field is undefined when the Link is not up.	RO

同理完成供应商、CMPS ID的校验：

```
Transcript
# [          150699529] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [          151695504] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [          152699504] :    Check Max Link Speed = 2.5GT/s - PASSED
# [          152699504] :    Check Negotiated Link Width = 4 - PASSED
# [          152707504] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [          153703529] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [          154707529] :    Check Device/Vendor ID - PASSED
# [          154715529] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [          155711504] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [          156715504] :    Check CMPS ID - PASSED
# [          156715504] :    SYSTEM CHECK PASSED
# [          156715504] : Inspecting Core Configuration Space...
# [          156723504] : TSK PARSE FRAME on Transmit
```


line19: 任务TSK_BAR_INIT，其中包含4个子任务

- TSK_BAR_SCAN：PCI协议规定给基地址寄存器写全1就会返回BAR的范围。



寄存器 BAR_INIT_P_BAR_RANGE 存储BAR范围。

可以看到，完成BAR扫描后，BAR0（BAR_INIT_P_BAR_RANGE[0]），BAR1（BAR_INIT_P_BAR_RANGE[1]），BAR2（BAR_INIT_P_BAR_RANGE[2]）的值分别为32’ hfff00000、32’ hffff0000、32’ hfff00000，分别对应AXI-Lite master接口、DMA接口和Bypass接口连接的存储空间大小。这与我们设置的值是一致的。

但是我们可以看到，在XIlinx的应用中，设置的32’ hfff00000只是一个编号值，在初始化过程中，会通过任务FNC_CONVERT_RANGE_TO_SIZE_32对这个编号进行转换：

```
function [31:0] FNC_CONVERT_RANGE_TO_SIZE_32;
            input [31:0] bar_index;
            reg   [32:0] return_value;
    begin
            case (BAR_INIT_P_BAR_RANGE[bar_index] & 32'hFFFF_FFF0) // AND off control bits
              32'hFFFF_FFF0 : return_value = 33'h0000_0010;
              32'hFFFF_FFE0 : return_value = 33'h0000_0020;
              32'hFFFF_FFC0 : return_value = 33'h0000_0040;
              32'hFFFF_FF80 : return_value = 33'h0000_0080;
              32'hFFFF_FF00 : return_value = 33'h0000_0100;
              32'hFFFF_FE00 : return_value = 33'h0000_0200;
              32'hFFFF_FC00 : return_value = 33'h0000_0400;
              32'hFFFF_F800 : return_value = 33'h0000_0800;
              32'hFFFF_F000 : return_value = 33'h0000_1000;
              32'hFFFF_E000 : return_value = 33'h0000_2000;
              32'hFFFF_C000 : return_value = 33'h0000_4000;
              32'hFFFF_8000 : return_value = 33'h0000_8000;
              32'hFFFF_0000 : return_value = 33'h0001_0000;
              32'hFFFE_0000 : return_value = 33'h0002_0000;
              32'hFFFC_0000 : return_value = 33'h0004_0000;
              32'hFFF8_0000 : return_value = 33'h0008_0000;
              32'hFFF0_0000 : return_value = 33'h0010_0000;
              32'hFFE0_0000 : return_value = 33'h0020_0000;
              32'hFFC0_0000 : return_value = 33'h0040_0000;
              32'hFF80_0000 : return_value = 33'h0080_0000;
              32'hFF00_0000 : return_value = 33'h0100_0000;
              32'hFE00_0000 : return_value = 33'h0200_0000;
              32'hFC00_0000 : return_value = 33'h0400_0000;
              32'hF800_0000 : return_value = 33'h0800_0000;
              32'hF000_0000 : return_value = 33'h1000_0000;
              32'hE000_0000 : return_value = 33'h2000_0000;
              32'hC000_0000 : return_value = 33'h4000_0000;
              32'h8000_0000 : return_value = 33'h8000_0000;
              default :       return_value = 33'h0000_0000;
            endcase

            FNC_CONVERT_RANGE_TO_SIZE_32 = return_value;
    end
endfunction // FNC_CONVERT_RANGE_TO_SIZE_32
```


可以看到，编号32’ hfff00000对应的实际大小33’ h0010_0000、33’ h0001_0000、33’ h0010_0000（1MB,64KB,1MB）

- TSK_BUILD_PCIE_MAP：查询BAR的映射关系

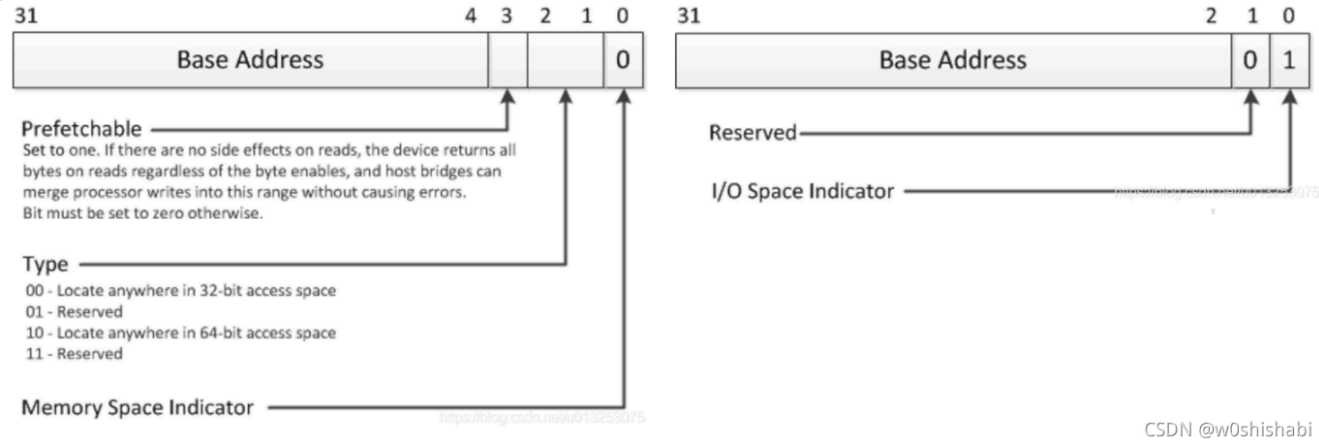
寄存器BAR_INIT_P_BAR_ENABLED 存储映射关系。

寄存器BAR_INIT_P_BAR 存储BAR对应的基地址。

寄存器BAR_INIT_P_BAR_ENABLED的值	**映射关系**
0	未使用BAR
1	io mapped
2	mem32 mapped
3	mem64 mapped

1. 关于映射类型

是根据最后一个字节进行判断的，具体没有找到资料，只是找到一张有点类似的图：



CSDN @w0shishabi

程序在判断映射关系时，也是按照上图的协议进行判断的。

根据范围推断出映射的类型，可以看到，完成BAR扫描后，BAR0（BAR_INIT_P_BAR_ENABLED[0]），BAR1（BAR_INIT_P_BAR_ENABLED[1]），BAR2（BAR_INIT_P_BAR_ENABLED[2]）的值都为2’ h2，说明AXI-Lite master接口、DMA接口和Bypass接口都是32位内存映射。这与我们设置的值是一致的。

2. 关于基地址的计算

根据初始地址以及各BAR尺寸计算，所以推测设置size为编号值的目的之一应该也是为了满足这个算法，可能x86架构都是这个规范，计算的算法为：

ii从0~6遍历：

```
    // We need to calculate where the next BAR should start based on the BAR's range
    BAR_INIT_TEMP = BAR_INIT_P_MEM32_START & {1'b1,(BAR_INIT_P_BAR_RANGE[ii] & 32'hffff_fff0)};

    if (BAR_INIT_TEMP < BAR_INIT_P_MEM32_START) begin

        // Current MEM32_START is NOT correct start for new base
        BAR_INIT_P_BAR[ii] =     BAR_INIT_TEMP + FNC_CONVERT_RANGE_TO_SIZE_32(ii);
        BAR_INIT_P_MEM32_START = BAR_INIT_P_BAR[ii] + FNC_CONVERT_RANGE_TO_SIZE_32(ii);

    end
    else begin

        // Initial BAR case and Current MEM32_START is correct start for new base
        BAR_INIT_P_BAR[ii] =     BAR_INIT_P_MEM32_START;
        BAR_INIT_P_MEM32_START = BAR_INIT_P_MEM32_START + FNC_CONVERT_RANGE_TO_SIZE_32(ii);

    end
```

初始值：

变量	初始值
BAR_INIT_P_MEM32_START	32’ h0
BAR_INIT_P_BAR_RANGE[0]	32’ hFFF00000
BAR_INIT_P_BAR_RANGE[1]	32’ hFFFF0000
BAR_INIT_P_BAR_RANGE[2]	32’ hFFF00000
BAR_INIT_P_BAR_RANGE[3]	32’ h0
BAR_INIT_P_BAR_RANGE[4]	32’ h0
BAR_INIT_P_BAR_RANGE[5]	32’ h0
BAR_INIT_P_BAR_RANGE[6]	32’ h0

计算过程：

step1	step2	step3	step4	
ii	BAR_INIT_P_MEM32_START	BAR_INIT_TEMP	BAR_INIT_TEMP < BAR_INIT_P_MEM32_START	BAR_INIT_P_BAR[ii]

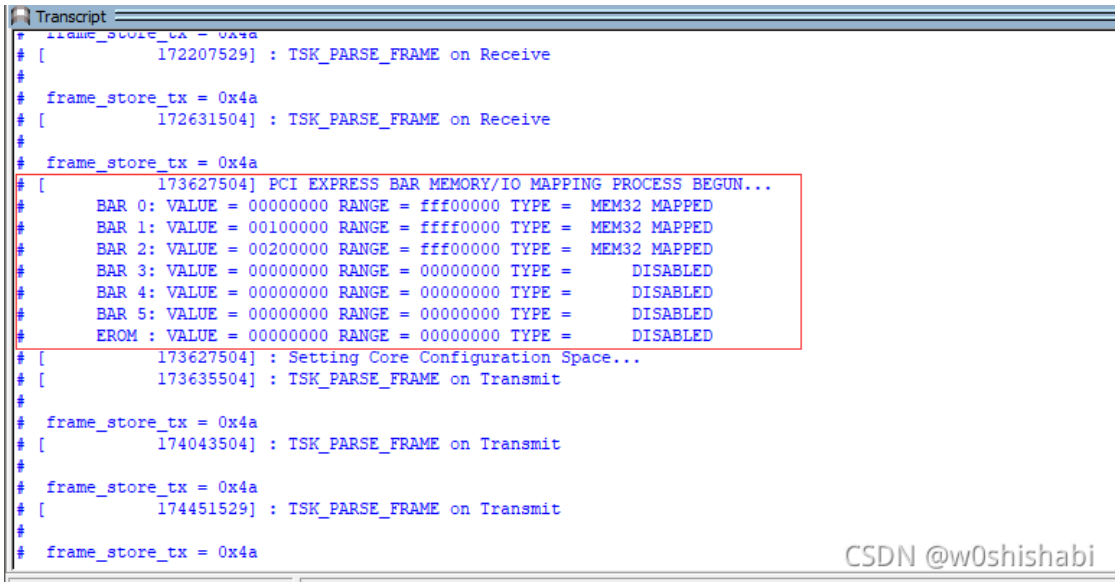
step1	step2	step3	step4	
0	32’ h0	32’ h0	FALSE	32’ h0
1	32’ h0010_0000	32’ h0010_0000	FALSE	32’ h0010_0000
2	32’ h0011_0000	32’ h0010_0000	TRUE	32’ h0020_0000
3	/	/	/	/
4	/	/	/	/
5	/	/	/	/
6	/	/	/	/

得到的基地址为：

BAR0对应的基地址：32’ h00000000
BAR1对应的基地址：32’ h00100000
BAR2对应的基地址：32’ h00200000

- TSK_DISPLAY_PCIE_MAP

将上面两个扫描的结果显示出来：

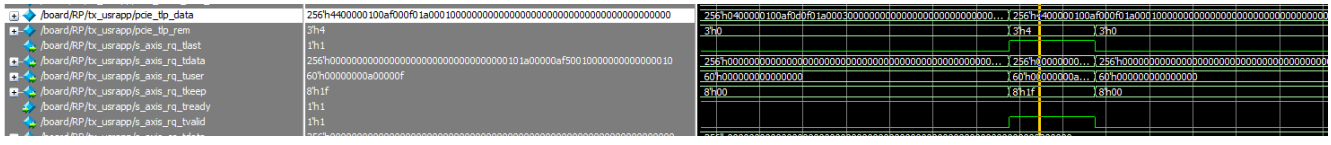


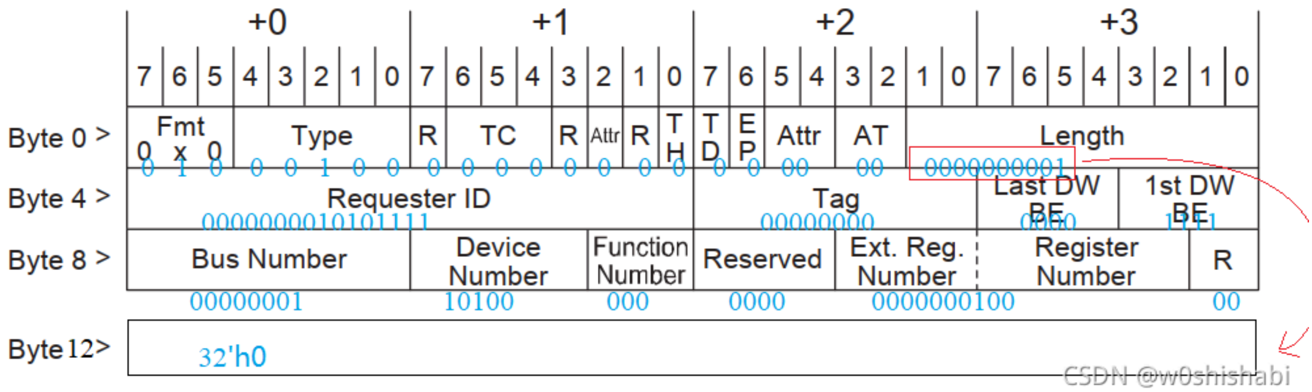
- TSK_BAR_PROGRAM

将计算出来的基地址写进BAR中。

虽然使用的是RQ总线，但是Xilinx还是把TLP包打包了一下方便分析。

以写BAR0为例，TLP为：





这是一个寄存器配置TLP包，作用为往地址为0x10(BAR0)写32’h0

line22:TSK_XDMA_FIND_BAR

找到哪个BAR是XDMA BAR并赋值’xdma_bar’变量

具体流程，分别读取映射类型为内存映射的BAR的偏移地址为0x0的数据，我们知道XIlinx对其DMA BAR偏移地址为0x0的寄存器的定义为：

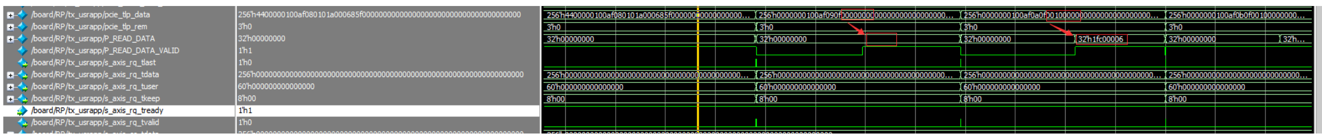
Table 2-39: **H2C Channel Identifier (0x00)**

Bit Index	Default Value	Access Type	Description
31:20	12’h1fc	RO	DMA Subsystem for PCIe identifier
19:16	4’h0	RO	H2C Channel Target
15	1’b0	RO	Stream 1: AXI4-Stream Interface 0: Memory Mapped AXI4 Interface
14:12	0	RO	Reserved
11:8	Varies	RO	Channel ID Target [3:0]
7:0	8'h04	RO	Version 8'h01: 2015.3 and 2015.4 8'h02: 2016.1 8'h03: 2016.2 8'h04: 2016.3 8'h05: 2016.4 8'h06: 2017.1, 2017.2 and 2017.3

判断的代码为：

```
    if (board.RP.tx_usrapp.P_READ_DATA[31:16] == 16'h1FC0) begin  //Mask [15:0] which will have revision number.
            xdma_bar = jj;
            xdma_bar_found = 1;
            $display (" XDMA BAR found : BAR %d is XDMA BAR\n", xdma_bar);
            end
    else begin
            $display (" XDMA BAR : BAR %d is NOT XDMA BAR\n", jj);
            end
    end
```

寻找的结果：



```
#  frame_store_tx = 0x4a
# [           182947504] : Data read 00000000 from Address 0x0000
#  XDMA BAR : BAR          0 is NOT XDMA BAR
#
# [           182955504] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           184095529] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [           184995529] : Data read 1fc00006 from Address 0x0000
#  XDMA BAR found : BAR          1 is XDMA BAR
#
```

line24：关于这个系统函数$value$plusargs,这次也学习了一下verilog系统函数： v a l u e value valueplusargs、 t e s t test testplusargs。简单说就是在仿真器运行仿真命令时可以在后面添加一些参数，比如这里可以指定测试用例。

line29:
读取PCIe设备的PCIe to DMA (BAR1)空间，地址为0x00，上面有此寄存器的定义。

根据bit15的值判断是1: AXI4-Stream Interface或0: Memory Mapped AXI4 Interface并选择对应的测试用例是dma_stream0或dma_test0。

line45: 将测试用例的代码包含进来

3.2 sample_tests.vh中测试用例的仿真流程

sample_tests.vh文件中关于名为dma_test0的测试用例的部分为：

```
else if(testname =="dma_test0")
begin

    //------------- This test performs a 32 bit write to a 32 bit Memory space and performs a read back

|//-------------------------------------------------------------------------------------------
|// XDMA H2C Test Starts
|//-------------------------------------------------------------------------------------------

    $display(" *** XDMA H2C *** \n");

    $display(" **** read Address at BAR0  = %h\n", board.RP.tx_usrapp.BAR_INIT_P_BAR[0][31:0]);
    $display(" **** read Address at BAR1  = %h\n", board.RP.tx_usrapp.BAR_INIT_P_BAR[1][31:0]);

    //-------------- Load DATA in Buffer ---------------------------------------------------
      board.RP.tx_usrapp.TSK_INIT_DATA_H2C;

|//-------------- DMA Engine ID Read ----------------------------------------------------
      board.RP.tx_usrapp.TSK_XDMA_REG_READ(16'h00);

    //-------------- Descriptor start address x0100 ----------------------------------------
|   board.RP.tx_usrapp.TSK_XDMA_REG_WRITE(16'h4080, 32'h00000100, 4'hF);

    //-------------- Start DMA tranfer -----------------------------------------------------
      $display(" **** Start DMA H2C transfer ***\n");

    fork
    //-------------- Writing XDMA CFG Register to start DMA Transfer for H2C ---------------
      board.RP.tx_usrapp.TSK_XDMA_REG_WRITE(16'h0004, 32'hfffe7f, 4'hF);   // Enable H2C DMA

    //-------------- compare H2C data ------------------------------------------------------
      $display("------Compare H2C Data--------\n");
      board.RP.tx_usrapp.COMPARE_DATA_H2C({16'h0,board.RP.tx_usrapp.DMA_BYTE_CNT});    //input payload bytes
    join
    loop_timeout = 0;
    desc_count = 0;
    //For this Example Design there is only one Descriptor used, so Descriptor Count would be 1
      while (desc_count == 0 && loop_timeout <= 10) begin
          board.RP.tx_usrapp.TSK_XDMA_REG_READ(16'h0040);
          $display ("**** H2C status = %h\n", P_READ_DATA);
          board.RP.tx_usrapp.TSK_XDMA_REG_READ(16'h48);
          $display ("**** H2C Decsriptor Count = %h\n", P_READ_DATA);
          if (P_READ_DATA == 32'h1) begin
            desc_count = 1;
          end else begin
            #10;
            loop_timeout = loop_timeout + 1;
          end

      end
      if (desc_count != 1) begin
          $display ("---***ERROR*** H2C Descriptor count mismatch,Loop Timeout occured ---\n");
      end
      board.RP.tx_usrapp.TSK_XDMA_REG_READ(16'h40);
      $display ("H2C DMA_STATUS  = %h\n", P_READ_DATA); // bit2 : Descriptor completed; bit1: Descriptor end; bit0: DMA Stopped
      h2c_status = P_READ_DATA;
      if (h2c_status != 32'h6) begin
        $display ("---***ERROR*** H2C status mismatch ---\n");
      end
|   $display ("bit2 : Descriptor completed; bit1: Descriptor end; bit0: DMA Stopped\n");

    //-------------- XDMA H2C and C2H Transfer separated by 1000ns -------------------------
```

```
      #1000;

    //------------------------------------------------------------------------------------------------
    // XDMA C2H Test Starts
    //------------------------------------------------------------------------------------------------

      $display(" *** XDMA C2H *** \n");

      desc_count = 0;
      loop_timeout = 0;
    //-------------- Load DATA in Buffer ---------------------------------------------------
      board.RP.tx_usrapp.TSK_INIT_DATA_C2H;

    //-------------- Descriptor start address x0300 ----------------------------------------
      board.RP.tx_usrapp.TSK_XDMA_REG_WRITE(16'h5080, 32'h00000300, 4'hF);

    // Start DMA transfer
      $display(" **** Start DMA C2H transfer ***\n");

    fork
    //-------------- Writing XDMA CFG Register to start DMA Transfer for C2H ----------------
      board.RP.tx_usrapp.TSK_XDMA_REG_WRITE(16'h1004, 32'hfffe7f, 4'hF);   // Enable C2H DMA

    //compare C2H data
      $display("------Compare C2H Data--------\n");
      board.RP.tx_usrapp.COMPARE_DATA_C2H({16'h0,board.RP.tx_usrapp.DMA_BYTE_CNT});
    join

    //For this Example Design there is only one Descriptor used, so Descriptor Count would be 1

      while (desc_count == 0 && loop_timeout <= 10) begin
          board.RP.tx_usrapp.TSK_XDMA_REG_READ(16'h1040);
          $display ("**** C2H status = %h\n", P_READ_DATA);
          board.RP.tx_usrapp.TSK_XDMA_REG_READ(16'h1048);
          $display ("**** C2H Decsriptor Count = %h\n", P_READ_DATA);
          if (P_READ_DATA == 32'h1) begin
            desc_count = 1;
          end else begin
            #10;
            loop_timeout = loop_timeout + 1;
          end
      end
      if (desc_count != 1) begin
          $display ("---***ERROR*** C2H Descriptor count mismatch,Loop Timeout occured ---\n");
      end
      board.RP.tx_usrapp.TSK_XDMA_REG_READ(16'h1040);
      $display ("C2H DMA_STATUS  = %h\n", P_READ_DATA); // bit2 : Descriptor completed; bit1: Descriptor end; bit0: DMA Stopped
      c2h_status = P_READ_DATA;
      if (c2h_status != 32'h6) begin
        $display ("---***ERROR*** C2H status mismatch ---\n");
      end
      $display ("bit2 : Descriptor completed; bit1: Descriptor end; bit0: DMA Stopped\n");



      #100;

    #1000;

   $finish;
end
```

注释中写了，这个测试用例执行一个32位到内存空间的写操作，然后回读。

line12: 显示信息

```
# *** Running XDMA AXI-MM test {dma_test0}......
#  *** XDMA H2C ***
#
#  **** read Address at BAR0  = 00000000
#
#  **** read Address at BAR1  = 00100000
#
```

line16: TSK_INIT_DATA_H2C

作用：初始化描述符数据

描述符的格式为：

Table 2-3: **Descriptor Format**

Offset	Fields			
0x0	Magic[15:0]	Rsv[1:0]	Nxt_adj[5:0]	Control[7:0]
0x04	4'h0, Len[27:0]			
0x08	Src_adr[31:0]			
0x0C	Src_adr[63:32]			
0x10	Dst_adr[31:0]			
0x14	Dst_adr[63:32]			
0x18	Nxt_adr[31:0]			
0x1C	Nxt_adr[63:32]			

描述符初始化的值为：

```
#
#  **** TASK DATA H2C ***
#
#  **** Initilize Descriptor data ***
#
#  **** Descriptor data *** data = 13, addr=         256
#
#  **** Descriptor data *** data = 00, addr=         257
#
#  **** Descriptor data *** data = 4b, addr=         258
#
#  **** Descriptor data *** data = ad, addr=         259
#
#  **** Descriptor data *** data = 80, addr=         260
#
#  **** Descriptor data *** data = 00, addr=         261
#
#  **** Descriptor data *** data = 00, addr=         262
#
#  **** Descriptor data *** data = 00, addr=         263
#
#  **** Descriptor data *** data = 00, addr=         264
#
#  **** Descriptor data *** data = 04, addr=         265
#
#  **** Descriptor data *** data = 00, addr=         266
#
#  **** Descriptor data *** data = 00, addr=         267
#
#  **** Descriptor data *** data = 00, addr=         268
#
#  **** Descriptor data *** data = 00, addr=         269
#
#  **** Descriptor data *** data = 00, addr=         270
#
#  **** Descriptor data *** data = 00, addr=         271
#
#  **** Descriptor data *** data = 00, addr=         272
#
#  **** Descriptor data *** data = 00, addr=         273
#
#  **** Descriptor data *** data = 00, addr=         274
#
#  **** Descriptor data *** data = 00, addr=         275
#
#  **** Descriptor data *** data = 00, addr=         276
#
#  **** Descriptor data *** data = 00, addr=         277
#
#  **** Descriptor data *** data = 00, addr=         278
#
#  **** Descriptor data *** data = 00, addr=         279
#
#  **** Descriptor data *** data = 00, addr=         280
#
#  **** Descriptor data *** data = 00, addr=         281
#
#  **** Descriptor data *** data = 00, addr=         282
#
#  **** Descriptor data *** data = 00, addr=         283
#
#  **** Descriptor data *** data = 00, addr=         284
#
#  **** Descriptor data *** data = 00, addr=         285
#
#  **** Descriptor data *** data = 00, addr=         286
#
#  **** Descriptor data *** data = 00, addr=         287
#
```

Table 2-4: **Descriptor Fields**

Field	Bit Index	Sub Field	Description
Magic	15:0	16'had4b	16'had4b. Code to verify that the driver generated descriptor is valid.
Nxt_adj	5:0	6'h00	The number of additional adjacent descriptors after the descriptor located at the next descriptor address field. A block of adjacent descriptors cannot cross a 4k boundary.
Control	5, 6, 7		Reserved
	4	EOP 1'b1	End of packet for stream interface.
	2, 3		Reserved
	1	Completed 1'b1	Set to 1 to interrupt after the engine has completed this descriptor. This requires global IE_DESCRIPTOR_COMPLETED control flag set in the SGDMA control register.
	0	Stop 1'b1	Set to 1 to stop fetching descriptors for this descriptor list. The stop bit can only be set on the last descriptor of an adjacent block of descriptors.
Length	27:0	16'h0008	Length of the data in bytes.
Src_adr	63:0	64'h0000 0400	Source address for H2C and memory mapped transfers. Metadata writeback address for C2H transfers.
Dst_adr	63:0	64'h0000 0000	Destination address for C2H and memory mapped transfers. Not used for H2C stream.
Nxt_adr	63:0	64'h0000 0000	Address of the next descriptor in the list.

含义：单个描述符，源地址为64’ h0000_0400，目的地址为64’ h0000_0000，数据长度为16’ h0080(128Byte)

line19: TSK_XDMA_REG_READ
作用：读取DMA引擎的ID，也就是读取XDMA BAR 偏移地址为16’ h00的寄存器。

```
# [           187043504] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           188183529] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [           189075529] : Data read 1fc00006 from Address 0000
```

line22: TSK_XDMA_REG_WRITE(16’ h4080, 32’ h00000100, 4’ hF)

作用：设置描述符的起始地址为32’ h00000100

Table 2-35: **PCIe to DMA Address Format**

31:16	15:12	11:8	7:0
Reserved	Target 4'h4	Channel 4'h0	Byte Offset 8'h80

Table 2-114: **C2H SGDMA Descriptor Low Address (0x80)**

Bit Index	Default	Access Type	Description
31:0	32'h0	RW	dsc_adr[31:0] Lower bits of start descriptor address. Dsc_adr[63:0] is the first descriptor address that is fetched after the Control register Run bit is set (Table 2-59).

line29: TSK_XDMA_REG_WRITE(16’ h0004, 32’ hfffe7f, 4’ hF)

作用：使能H2C DMA，开始进行H2C DMA传输

31:16	15:12	11:8	7:0
Reserved	Target 4'h0	Channel 4'h0	Byte Offset 8'h04

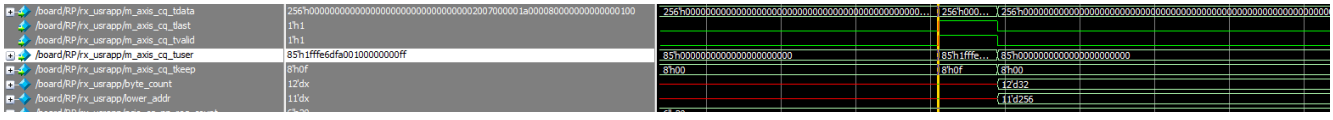
Table 2-40: **H2C Channel Control (0x04)**

Bit Index	Default	Access Type	Description
31:28			Reserved
27	1'b0	RW	When set write back information for C2H in AXI-Stream mode is disabled, default write back is enabled.
26	0x0	RW	pollmode_wb_enable Poll mode writeback enable. When this bit is set the DMA writes back the completed descriptor count when a descriptor with the Completed bit set, is completed.
25	1'b0	RW	non_inc_mode Non-incrementing address mode. Applies to m_axi_araddr interface only.
23:19	5'h0	RW	ie_desc_error Set to all 1s (0x1F) to enable logging of Status.Desc_error and to stop the engine if the error is detected.
18:14	5'h0	RW	ie_write_error Set to all 1s (0x1F) to enable logging of Status.Write_error and to stop the engine if the error is detected.
13:9	5'h0	RW	ie_read_error Set to all 1s (0x1F) to enable logging of Status.Read_error and to stop the engine if the error is detected.
8:7			Reserved
6	1'b0	RW	ie_idle_stopped Set to 1 to enable logging of Status.Idle_stopped
5	1'b0	RW	ie_invalid_length Set to 1 to enable logging of Status.Invalid_length
4	1'b0	RW	ie_magic_stopped Set to 1 to enable logging of Status.Magic_stopped
3	1'b0	RW	ie_align_mismatch Set to 1 to enable logging of Status.Align_mismatch
2	1'b0	RW	ie_descriptor_completed Set to 1 to enable logging of Status.Descriptor_completed
1	1'b0	RW	ie_descriptor_stopped Set to 1 to enable logging of Status.Descriptor_stopped
0	1'b0	RW	Run Set to 1 to start the SGDMA engine. Reset to 0 to stop transfer; if the engine is busy it completes the current descriptor.

DMA传输的过程：

1. 开启DMA后，因为是H2C，首先会收到一个来自EP的读请求：

```
# req_compl_wd = 1, cq_be = 15, lower_addr = 100, cq_data = 040
```



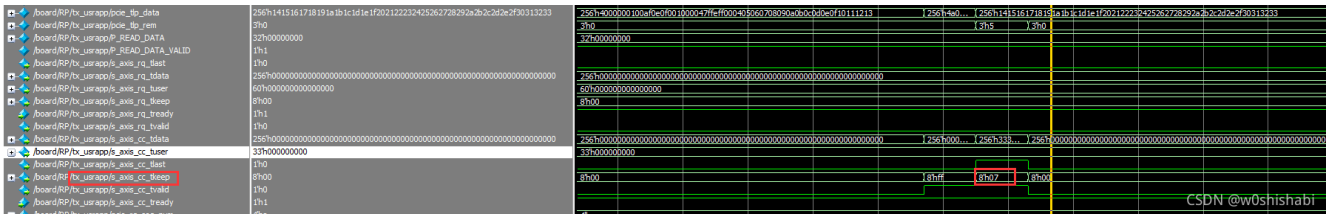
2. 然后RP用一个读完成包TSK_TX_COMPLETION_DATA发送数据来响应。

```
# ***** TSK_TX_COMPLETION_DATA ****** addr = 256., byte_count = 32(bytes), len = 8(DW), comp_status = 0(Success)
```

而我们知道刚才更新的描述符的地址为256，所以首先回的是描述符。

```
# [           190719529] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# req_compl_wd = 1, cq_be = 15, lower_addr = 100, cq_data = 040
#  ***** TSK_TX_COMPLETION_DATA ****** addr =  256., byte_count =  32, len =    8, comp_status = 0
#
# length =    3
#
# [           190731529] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           190731529] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
```

后面还输出一个length = 3 ,是因为最后一拍的数据长度为3DW，所以对应数据选通信号s_axis_cc_tkeep <= #(Tcq) 8'h07;



3. EP收到描述符后，再回一个获得DMA数据的读请求

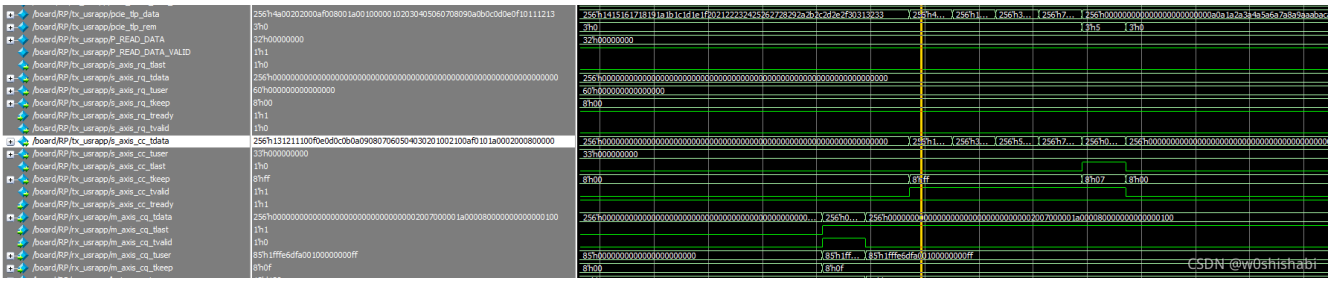
   # req_compl_wd = 1, cq_be = 15, lower_addr = 400, cq_data = 100

4. RP收到请求后，将对应内存空间的数据以完成包TSK_TX_COMPLETION_DATA发送出去。

   # ***** TSK_TX_COMPLETION_DATA ****** addr = 1024., byte_count = 128(bytes), len = 32(DW), comp_status = 0(Success)

```
# [           192039504] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# req_compl_wd = 1, cq_be = 15, lower_addr = 400, cq_data = 100
#  ***** TSK_TX_COMPLETION_DATA ****** addr = 1024., byte_count = 128, len =   32, comp_status = 0
#
# length =   27
#
# length =   19
#
# length =   11
#
# [           192059529] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# length =    3
#
```

CSDN @w0shishabi



我们可以看到，CC总线的数据位宽为256，总共分5拍发送出去，因为第一拍有包头，所以每拍发送的DW数分别为：5DW,8DW,8DW,8DW,3DW，总共发送32DW。

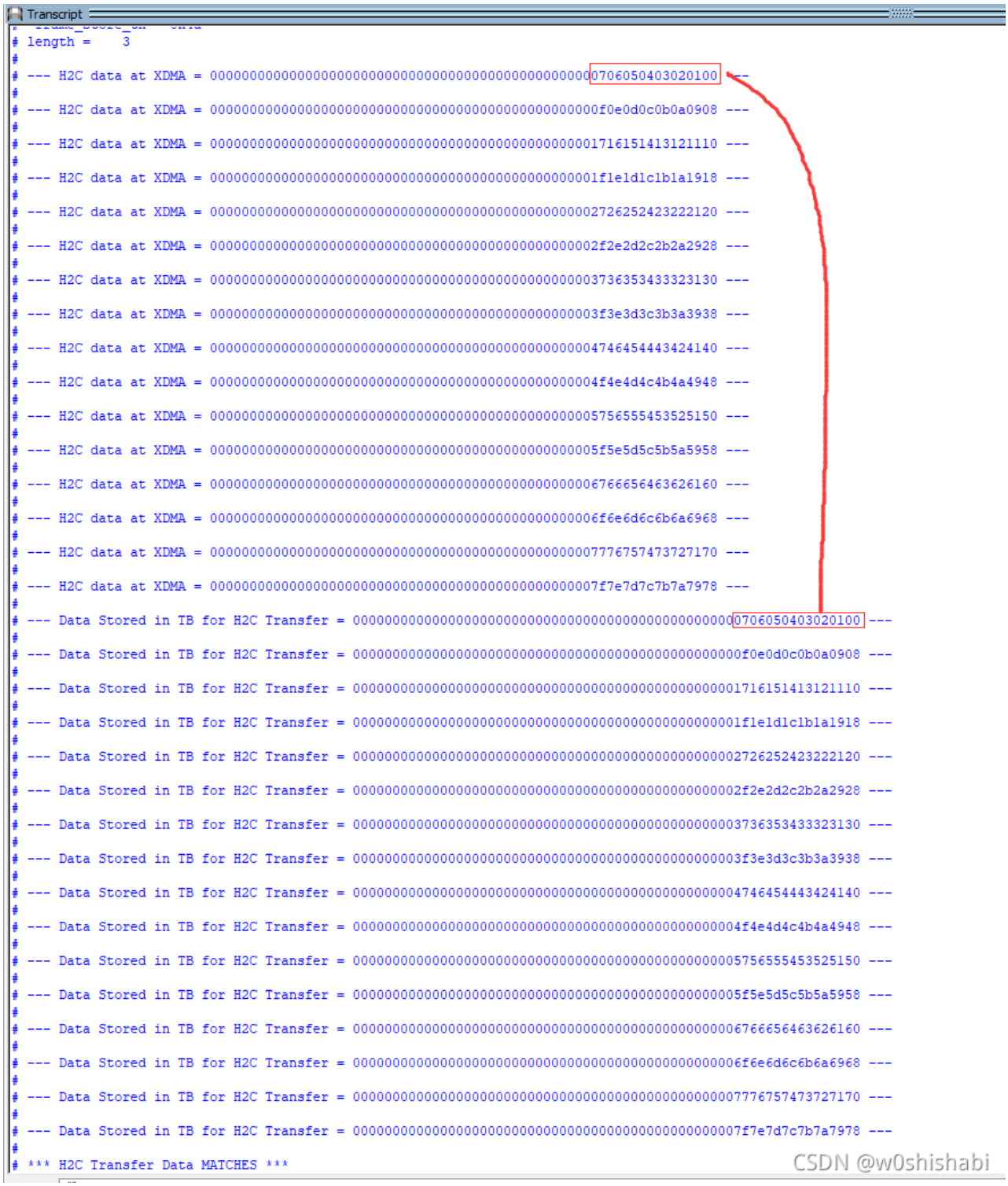
line33: COMPARE_DATA_H2C({16' h0,board.RP.tx_usrapp.DMA_BYTE_CNT});

作用：比较XDMA接收的数据和RP用户TB发送的数据。

首先根据payload_bytes = 128bytes 与数据位宽 （我们设置为64bit，8Bytes）计算得到需要分16beat来传送：

```
# ------Compare H2C Data--------
#
# Enters into compare read data task at 1.89484e+008ns
#
# payload bytes=00000080, data beat count =          16
#
```

然后分别获得XDMA接收的数据与testbench中给的数据并进行对比：



line38:

作用：分别读取DMA BAR偏移地址为16’ h0040、16’ h48获得H2C的状态以及描述符计数值：

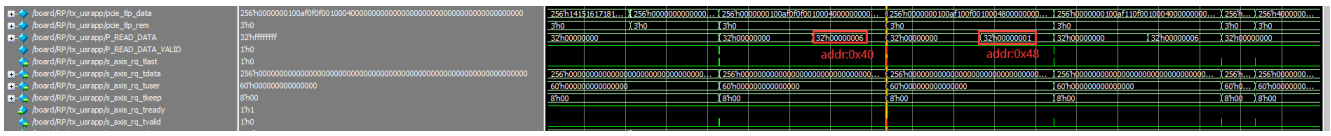


Table 2-43: **H2C Channel Status (0x40)**

Bit Index	Default	Access Type	Description
23:19	5'h0	RW1C	descr_error[4:0] 5'b0 Reset (0) on setting the Control register Run bit. 4: Unexpected completion 3: Header EP 2: Parity error 1: Completer abort 0: Unsupported request
18:14	5'h0	RW1C	write_error[4:0] 5'b0 Reset (0) on setting the Control register Run bit. Bit position: 4-2: Reserved 1: Slave error 0: Decode error
13:9	5'h0	RW1C	read_error[4:0] 5'b0 Reset (0) on setting the Control register Run bit. Bit position 4: Unexpected completion 3: Header EP 2: Parity error 1: Completer abort 0: Unsupported request
6	1'b0	RW1C	idle_stopped 1'b0 Reset (0) on setting the Control register Run bit. Set when the engine is idle after resetting the Control register Run bit if the Control register ie_idle_stopped bit is set.
5	1'b0	RW1C	invalid_length 1'b0 Reset on setting the Control register Run bit. Set when the descriptor length is not a multiple of the data width of an AXI4-Stream channel and the Control register ie_invalid_length bit is set.
4	1'b0	RW1C	magic_stopped 1'b0 Reset on setting the Control register Run bit. Set when the engine encounters a descriptor with invalid magic and stopped if the Control register ie_magic_stopped bit is set.
3	1'b0	RW1C	align_mismatch 1'b0 Source and destination address on descriptor are not properly aligned to each other.
2	1'b0	RW1C	descriptor_completed 1'b1 Reset on setting the Control register Run bit. Set after the engine has completed a descriptor with the COMPLETE bit set if the Control register ie_descriptor_stopped bit is set.
1	1'b0	RW1C	descriptor_stopped 1'b1 Reset on setting Control register Run bit. Set after the engine completed a descriptor with the STOP bit set if the Control register ie_descriptor_stopped bit is set.
0	1'b0	RO	Busy 1'b0 Set if the SGDMA engine is busy. Zero when it is idle.

Table 2-45: **H2C Channel Completed Descriptor Count (0x48)**

Bit Index	Default	Access Type	Description
31:0	32'h0	RO	compl_descriptor_count 32'h1 The number of competed descriptors update by the engine after completing each descriptor in the list. Reset to 0 on rising edge of Control register Run bit (Table 2-40).

```
#  frame_store_tx = 0x4a
# [          195179504] : Data read 00000006 from Address 0040
# **** H2C status = 00000006
#
# [          195187504] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [          196327529] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [          197219529] : Data read 00000001 from Address 0048
# **** H2C Decsriptor Count = 00000001
#
```

然后进行C2H测试，步骤差不多，先设置描述符，然后启动C2H DMA，比较在RP-userTB接收存储的数据 与 从RP-userTB发送的H2C传输数据。

```
#  frame_store_tx = 0x4a
# -- C2H data at RP = 1f1e1d1c1b1a191817161514131211100f0e0d0c0b0a09080706050403020100--
#
# -- C2H data at RP = 3f3e3d3c3b3a39383736353433323130 2f2e2d2c2b2a29282726252423222120--
#
# -- C2H data at RP = 5f5e5d5c5b5a59585756555453525150 4f4e4d4c4b4a49484746454443424140--
#
# -- C2H data at RP = 7f7e7d7c7b7a79787776757473727170 6f6e6d6c6b6a69686766656463626160--
#
# -- Data Stored in TB = 1f1e1d1c1b1a191817161514131211100f0e0d0c0b0a09080706050403020100--
#
# -- Data Stored in TB = 3f3e3d3c3b3a39383736353433323130 2f2e2d2c2b2a29282726252423222120--
#
# -- Data Stored in TB = 5f5e5d5c5b5a59585756555453525150 4f4e4d4c4b4a49484746454443424140--
#
# -- Data Stored in TB = 7f7e7d7c7b7a79787776757473727170 6f6e6d6c6b6a69686766656463626160--
#
# *** C2H Transfer Data MATCHES ***
#
# [          202765529] : XDMA C2H Test Completed Successfully
# [          202767529] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [          202771529] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [          203911504] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [          204803504] : Data read 00000006 from Address 1040
# **** C2H status = 00000006
#
# [          204811504] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [          205951529] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [          206843504] : Data read 00000001 from Address 1048
# **** C2H Decsriptor Count = 00000001
#
# [          206851504] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [          207991504] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [          208883529] : Data read 00000006 from Address 1040
# C2H DMA_STATUS  = 00000006
#
# bit2 : Descriptor completed; bit1: Descriptor end; bit0: DMA Stopped
#
# ** Note: $finish    : ../../../imports/sample_tests.vh(265)
#    Time: 208884629 ps  Iteration: 0  Instance: /board/RP/tx_usrapp
# 1
# Break in Module pci_exp_usrapp_tx at ../../../imports/sample_tests.vh line 265
```


4 总结

经过详细的仿真分析，知道了XDMA的配置与使用的详细过程：

- 首先host会读取PCIe EP配置空间，完成BAR初始化，这一步对应PC的上电扫描PCIe设备。
    1. 扫描PCIe BAR 的范围编号，确定映射类型与范围。
    2. 根据编号分配基地址。
- 设置好描述符。
- 打开DMA使能。
- 完成数据传输。

欢迎关注我的个人公众号：

5 附件

5.1 Modelsim波形文件

Modelsim Version：
Modelsim SE-64 10.5
Revision:2016.02
Date:Feb 13 2016

链接：https://pan.baidu.com/s/16LbV8CPwxHe_j304J_Dhjg
提取码：open

【20211103补充】Modelsim波形文件的保存与打开方法

5.2 Modelsim所有的输出信息

```
# [             4995000] : System Reset Is De-asserted...
# ** Warning: (vsim-12033) ../../../imports/board.v(163): Specified assertion type is not supported. Ignoring the assertion control task.
#    Time: 4995 ns  Iteration: 1  Instance: /board
# [           149451604] : Transaction Reset Is De-asserted...
# [           149455504] : Writing Cfg Addr [0x00000001]
# [           149491504] : Reading Cfg Addr [0x0000001a]
# [           149519504] : Writing Cfg Addr [0x0000001a]
# [           150691529] : Transaction Link Is Up...
# [           150699529] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           151695504] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [           152699504] :    Check Max Link Speed = 2.5GT/s - PASSED
# [           152699504] :    Check Negotiated Link Width = 4 - PASSED
# [           152707504] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           153703529] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [           154707529] :    Check Device/Vendor ID - PASSED
# [           154715529] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           155711504] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [           156715504] :    Check CMPS ID - PASSED
# [           156715504] :    SYSTEM CHECK PASSED
# [           156715504] : Inspecting Core Configuration Space...
# [           156723504] : TSK_PARSE_FRAME on Transmit
#
```

```
#  frame_store_tx = 0x4a
# [           157131529] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           157711529] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [           158135504] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [           159139504] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           159547529] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           160127529] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [           160543504] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [           161555504] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           161963529] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           162543529] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [           162967504] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [           163971504] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           164379529] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           164959529] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [           165375504] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [           166387504] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           166795529] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           167375529] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [           167799504] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [           168803504] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           169211529] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           169791529] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [           170207504] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [           171219504] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           171627529] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           172207529] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [           172631504] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [           173627504] PCI EXPRESS BAR MEMORY/IO MAPPING PROCESS BEGUN...
#       BAR 0: VALUE = 00000000 RANGE = fff00000 TYPE =  MEM32 MAPPED
#       BAR 1: VALUE = 00100000 RANGE = ffff0000 TYPE =  MEM32 MAPPED
#       BAR 2: VALUE = 00200000 RANGE = fff00000 TYPE =  MEM32 MAPPED
#       BAR 3: VALUE = 00000000 RANGE = 00000000 TYPE =      DISABLED
#       BAR 4: VALUE = 00000000 RANGE = 00000000 TYPE =      DISABLED
#       BAR 5: VALUE = 00000000 RANGE = 00000000 TYPE =      DISABLED
#       EROM : VALUE = 00000000 RANGE = 00000000 TYPE =      DISABLED
# [           173627504] : Setting Core Configuration Space...
# [           173635504] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           174043504] : TSK_PARSE_FRAME on Transmit
#
```

```
#  frame_store_tx = 0x4a
# [           174451529] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           174623529] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [           174859504] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           175031504] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [           175267529] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           175439529] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [           175675504] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           175847504] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [           176083529] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           176255529] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [           176491504] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           176663504] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [           176899529] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           177071529] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [           177479504] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [           177887529] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [           180907529] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           181967529] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [           182947504] : Data read 00000000 from Address 0x0000
#  XDMA BAR : BAR           0 is NOT XDMA BAR
#
# [           182955504] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           184095529] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [           184995529] : Data read 1fc00006 from Address 0x0000
#  XDMA BAR found : BAR           1 is XDMA BAR
#
# [           185003529] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           186143504] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [           187035504] : Data read 1fc00006 from Address 0000
# *** Running XDMA AXI-MM test {dma_test0}......
#  *** XDMA H2C ***
#
#  **** read Address at BAR0  = 00000000
#
#  **** read Address at BAR1  = 00100000
#
#  **** TASK DATA H2C ***
#
#  **** Initilize Descriptor data ***
#
#  **** Descriptor data *** data = 13, addr=         256
#
#  **** Descriptor data *** data = 00, addr=         257
#
#  **** Descriptor data *** data = 4b, addr=         258
#
#  **** Descriptor data *** data = ad, addr=         259
#
#  **** Descriptor data *** data = 80, addr=         260
#
#  **** Descriptor data *** data = 00, addr=         261
```

```
#
#  **** Descriptor data *** data = 00, addr=         262
#
#  **** Descriptor data *** data = 00, addr=         263
#
#  **** Descriptor data *** data = 00, addr=         264
#
#  **** Descriptor data *** data = 04, addr=         265
#
#  **** Descriptor data *** data = 00, addr=         266
#
#  **** Descriptor data *** data = 00, addr=         267
#
#  **** Descriptor data *** data = 00, addr=         268
#
#  **** Descriptor data *** data = 00, addr=         269
#
#  **** Descriptor data *** data = 00, addr=         270
#
#  **** Descriptor data *** data = 00, addr=         271
#
#  **** Descriptor data *** data = 00, addr=         272
#
#  **** Descriptor data *** data = 00, addr=         273
#
#  **** Descriptor data *** data = 00, addr=         274
#
#  **** Descriptor data *** data = 00, addr=         275
#
#  **** Descriptor data *** data = 00, addr=         276
#
#  **** Descriptor data *** data = 00, addr=         277
#
#  **** Descriptor data *** data = 00, addr=         278
#
#  **** Descriptor data *** data = 00, addr=         279
#
#  **** Descriptor data *** data = 00, addr=         280
#
#  **** Descriptor data *** data = 00, addr=         281
#
#  **** Descriptor data *** data = 00, addr=         282
#
#  **** Descriptor data *** data = 00, addr=         283
#
#  **** Descriptor data *** data = 00, addr=         284
#
#  **** Descriptor data *** data = 00, addr=         285
#
#  **** Descriptor data *** data = 00, addr=         286
#
#  **** Descriptor data *** data = 00, addr=         287
#
# [           187043504] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           188183529] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [           189075529] : Data read 1fc00006 from Address 0000
# [           189075529] : Sending Data write task at address 00004080 with data 00000100
# [           189083529] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           189483504] : Done register write!!
#  **** Start DMA H2C transfer ***
#
# [           189483504] : Sending Data write task at address 00000004 with data 00fffe7f
# ------Compare H2C Data--------
#
# Enters into compare read data task at 1.89484e+008ns
#
# payload bytes=00000080, data_beat_count =         16
#
# [           189491504] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           189891529] : Done register write!!
# [           190719529] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# req_compl_wd = 1, cq_be = 15, lower_addr = 100, cq_data = 040
#  ***** TSK_TX_COMPLETION_DATA ****** addr =  256., byte_count =  32, len =    8, comp_status = 0
#
# length =    3
#
# [           190731529] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           190731529] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           192039504] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# req_compl_wd = 1, cq_be = 15, lower_addr = 400, cq_data = 100
```

```
#  ***** TSK_TX_COMPLETION_DATA ****** addr = 1024., byte_count = 128, len =   32, comp_status = 0
#
# length =   27
#
# length =   19
#
# length =   11
#
# [           192059529] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# length =    3
#
# --- H2C data at XDMA = 00000000000000000000000000000000000000000000000000000706050403020100 ---
#
# --- H2C data at XDMA = 0000000000000000000000000000000000000000000000000000000f0e0d0c0b0a0908 ---
#
# --- H2C data at XDMA = 00000000000000000000000000000000000000000000000000001716151413121110 ---
#
# --- H2C data at XDMA = 0000000000000000000000000000000000000000000000000000000001f1e1d1c1b1a1918 ---
#
# --- H2C data at XDMA = 00000000000000000000000000000000000000000000000000002726252423222120 ---
#
# --- H2C data at XDMA = 0000000000000000000000000000000000000000000000000000000002f2e2d2c2b2a2928 ---
#
# --- H2C data at XDMA = 00000000000000000000000000000000000000000000000000003736353433323130 ---
#
# --- H2C data at XDMA = 0000000000000000000000000000000000000000000000000000000003f3e3d3c3b3a3938 ---
#
# --- H2C data at XDMA = 00000000000000000000000000000000000000000000000000004746454443424140 ---
#
# --- H2C data at XDMA = 0000000000000000000000000000000000000000000000000000000004f4e4d4c4b4a4948 ---
#
# --- H2C data at XDMA = 00000000000000000000000000000000000000000000000000005756555453525150 ---
#
# --- H2C data at XDMA = 0000000000000000000000000000000000000000000000000000000005f5e5d5c5b5a5958 ---
#
# --- H2C data at XDMA = 00000000000000000000000000000000000000000000000000006766656463626160 ---
#
# --- H2C data at XDMA = 0000000000000000000000000000000000000000000000000000000006f6e6d6c6b6a6968 ---
#
# --- H2C data at XDMA = 00000000000000000000000000000000000000000000000000007776757473727170 ---
#
# --- H2C data at XDMA = 0000000000000000000000000000000000000000000000000000000007f7e7d7c7b7a7978 ---
#
# --- Data Stored in TB for H2C Transfer = 00000000000000000000000000000000000000000000000000000706050403020100 ---
#
# --- Data Stored in TB for H2C Transfer = 0000000000000000000000000000000000000000000000000000000f0e0d0c0b0a0908 ---
#
# --- Data Stored in TB for H2C Transfer = 00000000000000000000000000000000000000000000000000001716151413121110 ---
#
# --- Data Stored in TB for H2C Transfer = 0000000000000000000000000000000000000000000000000000000001f1e1d1c1b1a1918 ---
#
# --- Data Stored in TB for H2C Transfer = 00000000000000000000000000000000000000000000000000002726252423222120 ---
#
# --- Data Stored in TB for H2C Transfer = 0000000000000000000000000000000000000000000000000000000002f2e2d2c2b2a2928 ---
#
# --- Data Stored in TB for H2C Transfer = 00000000000000000000000000000000000000000000000000003736353433323130 ---
#
# --- Data Stored in TB for H2C Transfer = 0000000000000000000000000000000000000000000000000000000003f3e3d3c3b3a3938 ---
#
# --- Data Stored in TB for H2C Transfer = 00000000000000000000000000000000000000000000000000004746454443424140 ---
#
# --- Data Stored in TB for H2C Transfer = 0000000000000000000000000000000000000000000000000000000004f4e4d4c4b4a4948 ---
#
# --- Data Stored in TB for H2C Transfer = 00000000000000000000000000000000000000000000000000005756555453525150 ---
#
# --- Data Stored in TB for H2C Transfer = 0000000000000000000000000000000000000000000000000000000005f5e5d5c5b5a5958 ---
#
# --- Data Stored in TB for H2C Transfer = 00000000000000000000000000000000000000000000000000006766656463626160 ---
#
# --- Data Stored in TB for H2C Transfer = 0000000000000000000000000000000000000000000000000000000006f6e6d6c6b6a6968 ---
#
# --- Data Stored in TB for H2C Transfer = 00000000000000000000000000000000000000000000000000007776757473727170 ---
#
# --- Data Stored in TB for H2C Transfer = 0000000000000000000000000000000000000000000000000000000007f7e7d7c7b7a7978 ---
#
# *** H2C Transfer Data MATCHES ***
#
# [           193141220] : XDMA H2C Test Completed Successfully
# [           193147529] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           194287504] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [           195179504] : Data read 00000006 from Address 0040
# **** H2C status = 00000006
#
# [           195187504] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           196327529] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [           197219529] : Data read 00000001 from Address 0048
```

```
# **** H2C Decsriptor Count = 00000001
#
# [           197227529] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           198367504] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [           199259529] : Data read 00000006 from Address 0040
# H2C DMA_STATUS  = 00000006
#
# bit2 : Descriptor completed; bit1: Descriptor end; bit0: DMA Stopped
#
#  *** XDMA C2H ***
#
#  **** TASK DATA C2H ***
#
#  **** Initilize Descriptor data ***
#
#  **** Descriptor data *** data = 13, addr=         768
#
#  **** Descriptor data *** data = 00, addr=         769
#
#  **** Descriptor data *** data = 4b, addr=         770
#
#  **** Descriptor data *** data = ad, addr=         771
#
#  **** Descriptor data *** data = 80, addr=         772
#
#  **** Descriptor data *** data = 00, addr=         773
#
#  **** Descriptor data *** data = 00, addr=         774
#
#  **** Descriptor data *** data = 00, addr=         775
#
#  **** Descriptor data *** data = 00, addr=         776
#
#  **** Descriptor data *** data = 00, addr=         777
#
#  **** Descriptor data *** data = 00, addr=         778
#
#  **** Descriptor data *** data = 00, addr=         779
#
#  **** Descriptor data *** data = 00, addr=         780
#
#  **** Descriptor data *** data = 00, addr=         781
#
#  **** Descriptor data *** data = 00, addr=         782
#
#  **** Descriptor data *** data = 00, addr=         783
#
#  **** Descriptor data *** data = 00, addr=         784
#
#  **** Descriptor data *** data = 08, addr=         785
#
#  **** Descriptor data *** data = 00, addr=         786
#
#  **** Descriptor data *** data = 00, addr=         787
#
#  **** Descriptor data *** data = 00, addr=         788
#
#  **** Descriptor data *** data = 00, addr=         789
#
#  **** Descriptor data *** data = 00, addr=         790
#
#  **** Descriptor data *** data = 00, addr=         791
#
#  **** Descriptor data *** data = 00, addr=         792
#
#  **** Descriptor data *** data = 00, addr=         793
#
#  **** Descriptor data *** data = 00, addr=         794
#
#  **** Descriptor data *** data = 00, addr=         795
#
#  **** Descriptor data *** data = 00, addr=         796
#
#  **** Descriptor data *** data = 00, addr=         797
#
#  **** Descriptor data *** data = 00, addr=         798
#
#  **** Descriptor data *** data = 00, addr=         799
#
# [           199260721] : Sending Data write task at address 00005080 with data 00000300
# [           199267529] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           199667504] : Done register write!!
#  **** Start DMA C2H transfer ***
#
# [           199667504] : Sending Data write task at address 00001004 with data 00fffe7f
# ------Compare C2H Data--------
#
# payload_bytes = 00000080, data_beat_count = 00000004
#
# [           199675504] : TSK_PARSE_FRAME on Transmit
```

```
#
#  frame_store_tx = 0x4a
# [           200075529] : Done register write!!
# [           200903529] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# req_compl_wd = 1, cq_be = 15, lower_addr = 300, cq_data = 0c0
#  ***** TSK_TX_COMPLETION_DATA ****** addr =  768., byte_count =  32, len =    8, comp_status = 0
#
# length =    3
#
# [           200915529] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           200915529] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# -- C2H data at RP = 1f1e1d1c1b1a191817161514131211100f0e0d0c0b0a09080706050403020100--
#
# -- C2H data at RP = 3f3e3d3c3b3a39383736353433323130 2f2e2d2c2b2a29282726252423222120--
#
# -- C2H data at RP = 5f5e5d5c5b5a595857565554535251504f4e4d4c4b4a494847464544434241 40--
#
# -- C2H data at RP = 7f7e7d7c7b7a797877767574737271706f6e6d6c6b6a696867666564636261 60--
#
# -- Data Stored in TB = 1f1e1d1c1b1a191817161514131211100f0e0d0c0b0a09080706050403020100--
#
# -- Data Stored in TB = 3f3e3d3c3b3a39383736353433323130 2f2e2d2c2b2a29282726252423222120--
#
# -- Data Stored in TB = 5f5e5d5c5b5a595857565554535251504f4e4d4c4b4a494847464544434241 40--
#
# -- Data Stored in TB = 7f7e7d7c7b7a797877767574737271706f6e6d6c6b6a696867666564636261 60--
#
# *** C2H Transfer Data MATCHES ***
#
# [           202765529] : XDMA C2H Test Completed Successfully
# [           202767529] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [           202771529] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           203911504] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [           204803504] : Data read 00000006 from Address 1040
# **** C2H status = 00000006
#
# [           204811504] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           205951529] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [           206843504] : Data read 00000001 from Address 1048
# **** C2H Decsriptor Count = 00000001
#
# [           206851504] : TSK_PARSE_FRAME on Transmit
#
#  frame_store_tx = 0x4a
# [           207991504] : TSK_PARSE_FRAME on Receive
#
#  frame_store_tx = 0x4a
# [           208883529] : Data read 00000006 from Address 1040
# C2H DMA_STATUS  = 00000006
#
# bit2 : Descriptor completed; bit1: Descriptor end; bit0: DMA Stopped
#
# ** Note: $finish    : ../../../imports/sample_tests.vh(265)
#    Time: 208884629 ps  Iteration: 0  Instance: /board/RP/tx_usrapp
```