Xilinx PCIe IP核示例工程代码分析与仿真

PCIe学习笔记系列：

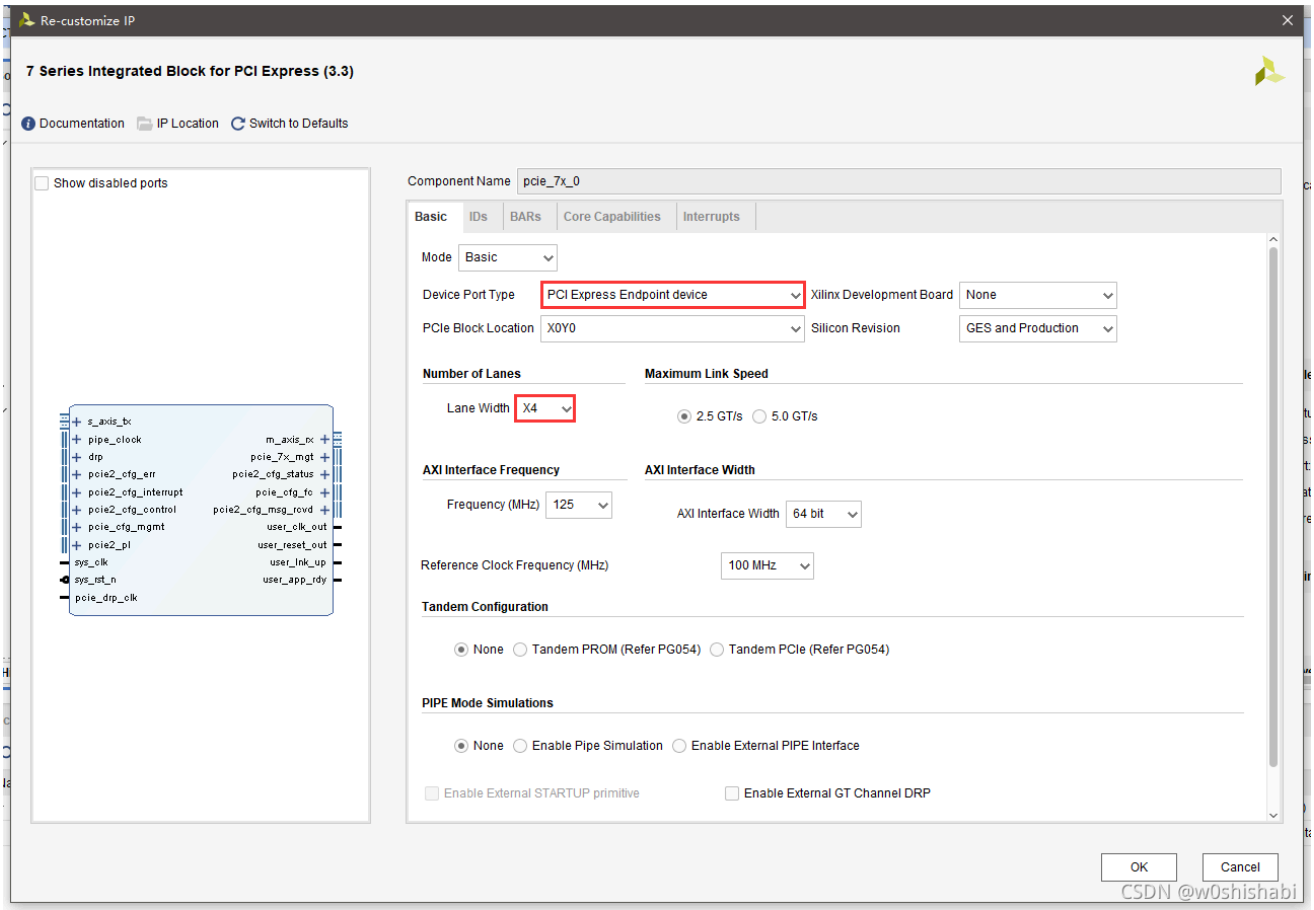
1. PCIe基础知识及Xilinx相关IP核介绍
   概念了解：简单学习PCIe的数据链路与拓扑结构，另外看看有什么相关的IP核。
2. 【PG054】7 Series Integrated Block for PCI Express IP核的学习
   基础学习：关于Pcie IP核的数据手册，学习PCIe相关的IP核的配置参数及其对应的含义。
3. Xilinx PCIe IP核示例工程代码分析与仿真
   基础学习：关于PCIe IP核的仿真，学习PCIe的配置流程以及应用过程。
4. Xilinx XDMA 例程代码分析与仿真结果
   应用学习：关于Xilinx PCIe DMA 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。

【PG054】7 Series Integrated Block for PCI Express IP核的学习中学习了7 Series Integrated Block for PCI Express IP核的一些基础知识，下面通过仿真进一步理解。

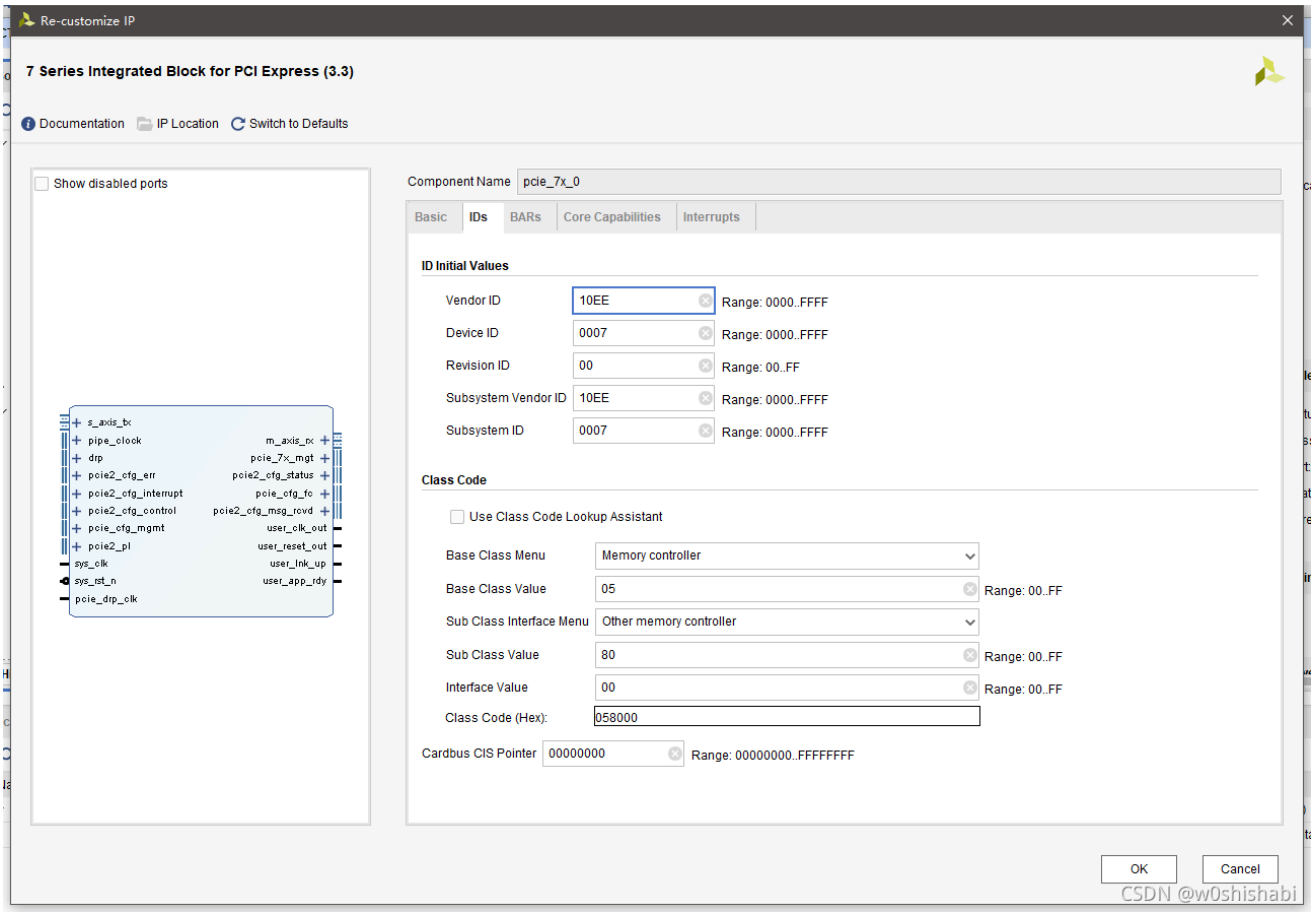
1 工程建立

只进行仿真设计，随便配置成Endpoint器件就行。

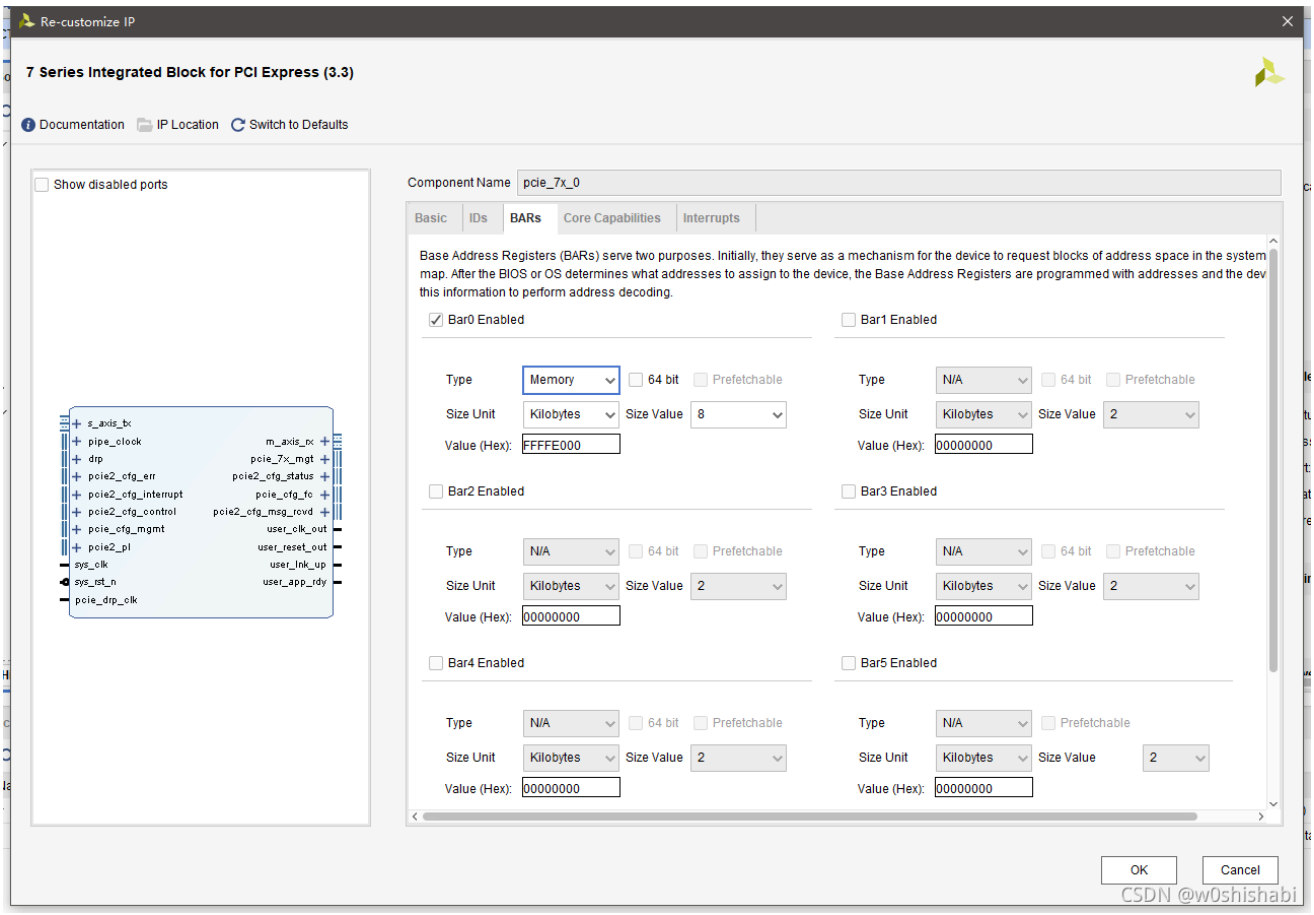
1 page1：Basic



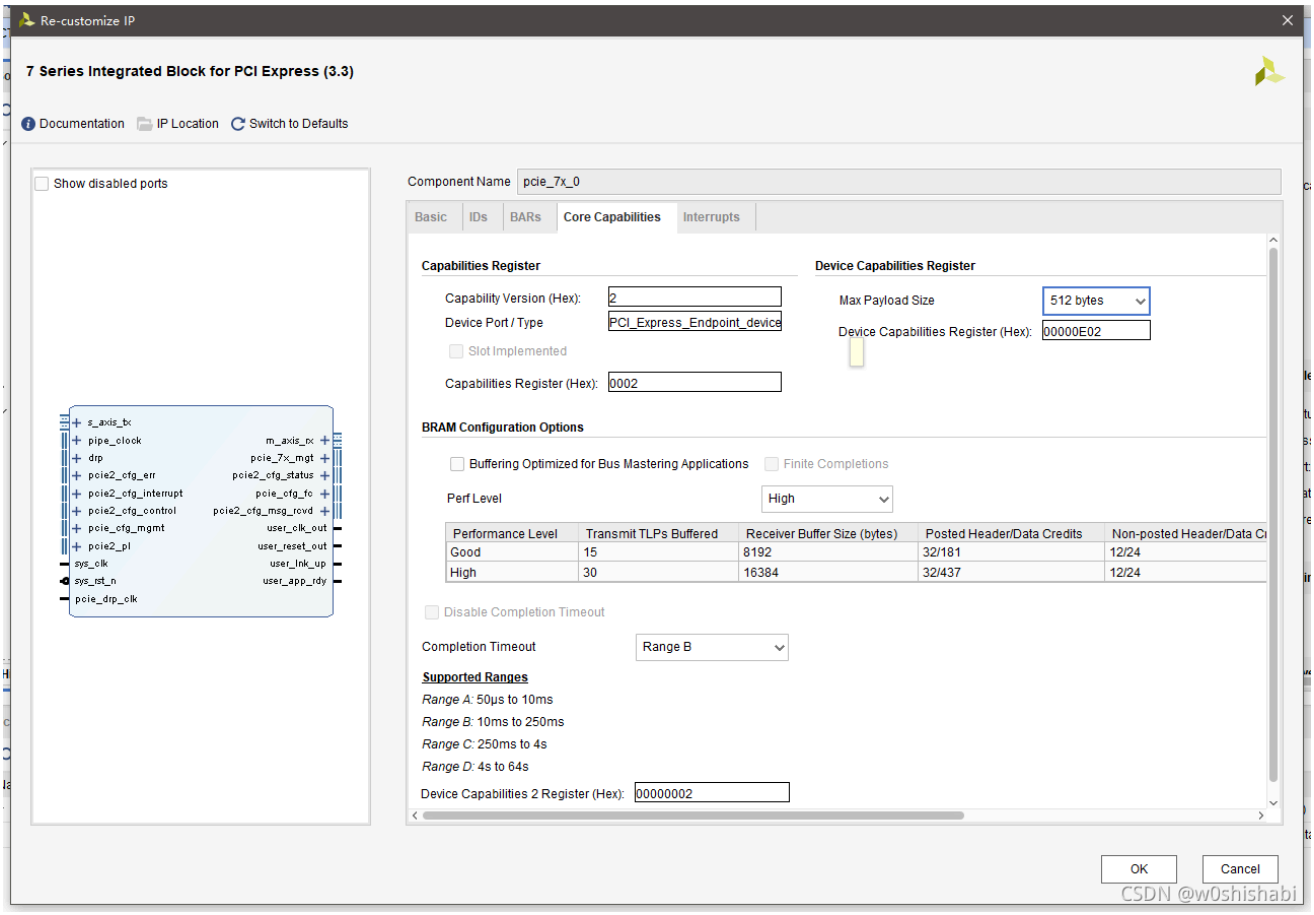
2 page2: IDs



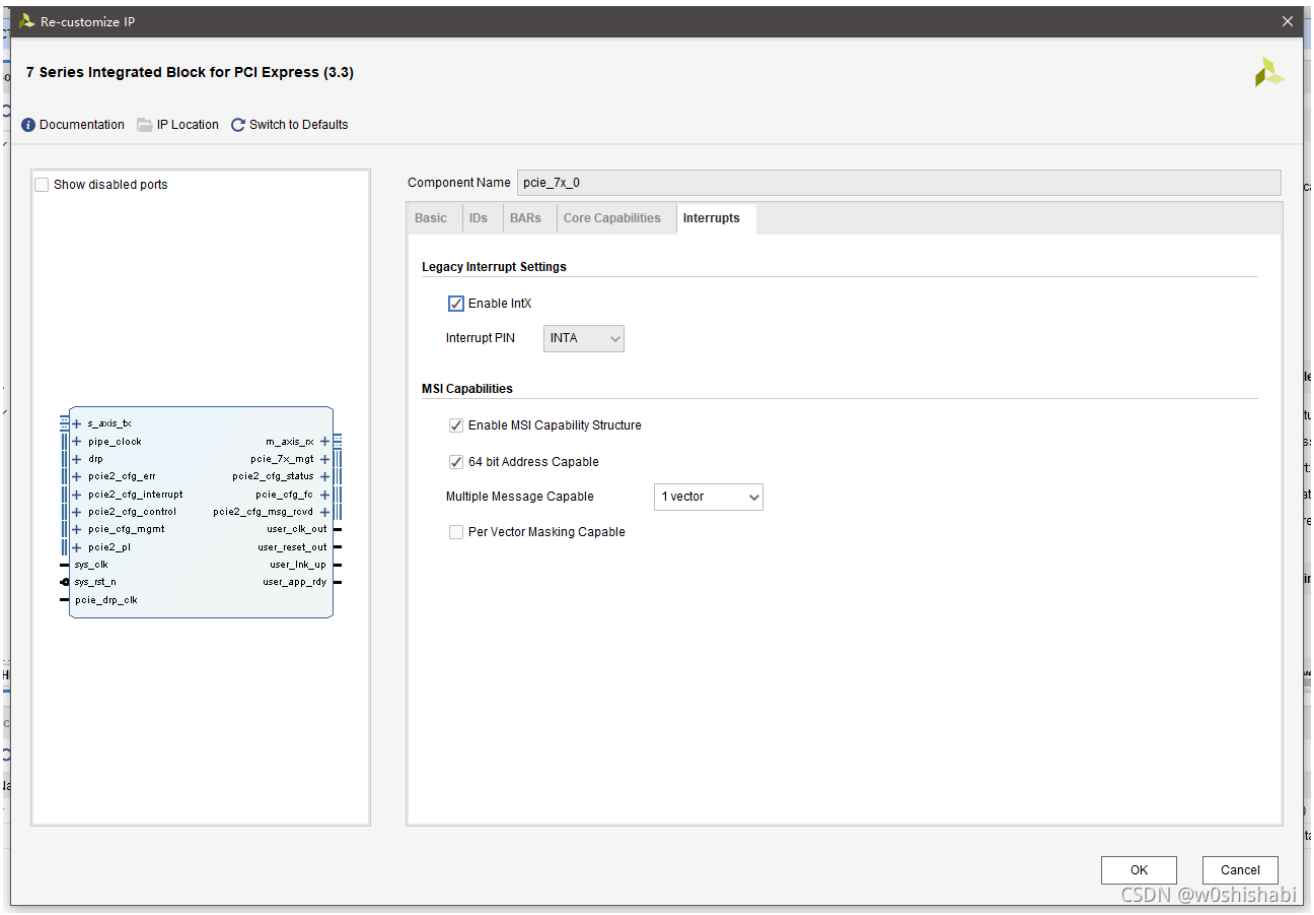
3 page3: BARs



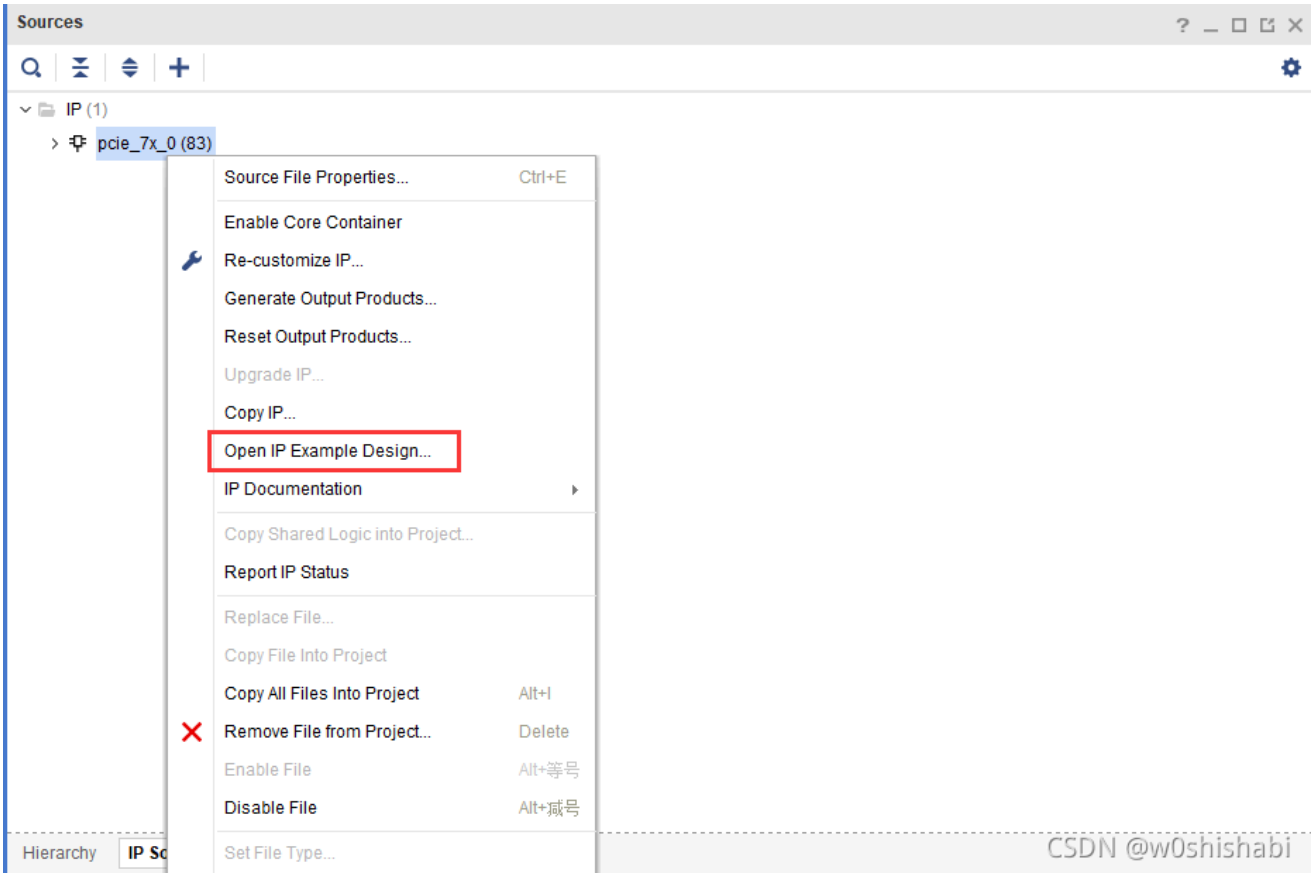
4 page4: Core Capabilities



5 page5: Interrupts



设置好IP核之后，右击IP核打开示例工程：



示例工程的文件结构：

```
xilinx_pcie_2_1_ep_7x
|
|--pcie_7x_0_support
|  |--pcie_7x_0_pipe_clock
|  |--pcie_7x_0 (Core Top level module Generated by Vivado in synth directory)
|     |--pcie_7x_v3_3_4_top (Static Top level file)
|        |--pcie_7x_v3_3_4_core_top
|           |
|           |--pcie_7x_0_pcie_top
|           |  |
|           |  |--pcie_7x_0_axi_basic_top
|           |  |  |
|           |  |  |--pcie_7x_0_axi_basic_rx
|           |  |  |  |
|           |  |  |  |--pcie_7x_0_axi_basic_rx_pipeline
|           |  |  |  |--pcie_7x_0_axi_basic_rx_null_gen
|           |  |  |
|           |  |  |--pcie_7x_0_axi_basic_tx
|           |  |     |
|           |  |     |--pcie_7x_0_axi_basic_tx_pipeline
|           |  |     |--pcie_7x_0_axi_basic_tx_thrtl_ctl
|           |  |
|           |  |--pcie_7x_0_pcie_7x
|           |  |  |
|           |  |  |--pcie_7x_0_pcie_bram_top_7x
|           |  |  |  |
|           |  |  |  |--pcie_7x_0_pcie_brams_7x (an instance each for Rx & Tx)
|           |  |  |     |
|           |  |  |     |--pcie_7x_0_pcie_bram_7x
|           |  |  |
|           |  |  |--PCIE_2_1 (Integrated Block Instance)
|           |  |
|           |  |--pcie_7x_0_pcie_pipe_pipeline
|           |     |
|           |     |--pcie_7x_0_pcie_pipe_misc
|           |     |--pcie_7x_0_pcie_pipe_lane (per lane)
|           |
|           |--pcie_7x_0_gt_top
|              |
|              |--pcie_7x_0_gt_rx_valid_filter
|              |
|              |--pcie_7x_0_pipe_wrapper
|              |  |
|              |  |--pcie_7x_0_pipe_reset
|              |  |--pcie_7x_0_qpll_reset
|              |  |--pcie_7x_0_pipe_user
|              |  |--pcie_7x_0_pipe_rate
```

```
|              |  |--pcie_7x_0_pipe_sync
|              |  |--pcie_7x_0_pipe_drp
|              |  |--pcie_7x_0_pipe_eq
|              |  |  |--pcie_7x_0_rxeq_scan
|              |  |  |
|              |  |--pcie_7x_0_gt_common
|              |  |  |--pcie_7x_0_qpll_drp
|              |  |  |--pcie_7x_0_qpll_wrapper
|              |  |
|              |  |--pcie_7x_0_gt_wrapper
|
|--pcie_app_7x (PIO design, in example_design directory)
   |
   |--PIO
      |
      |--PIO_EP
      |  |
      |  |--PIO_EP_MEM_ACCESS
      |  |  |
      |  |  |--EP_MEM
      |  |     |
      |  |     |--RAMB36
      |  |
      |  |--PIO_RX_ENGINE
      |  |--PIO_TX_ENGINE
      |
      |--PIO_TO_CTRL
```

2 工程仿真

2.1 代码分析

再回顾一下示例程序的结构：

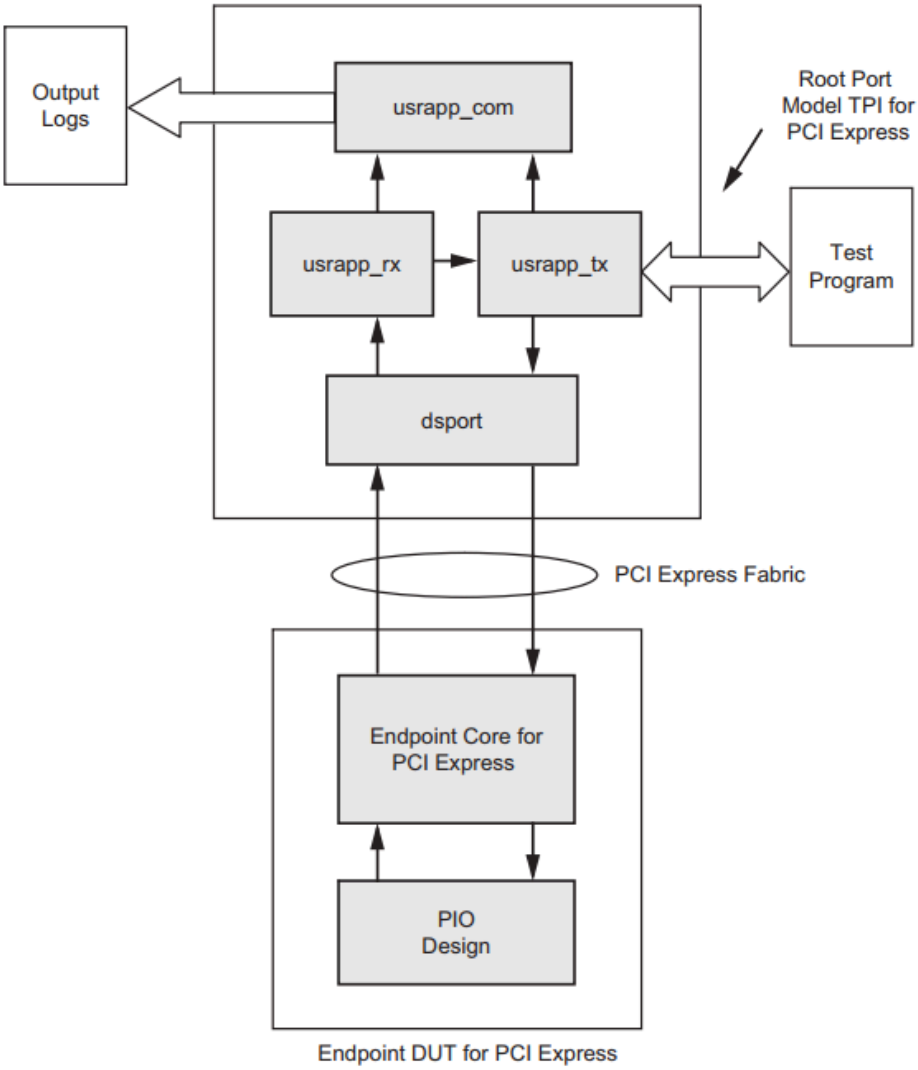
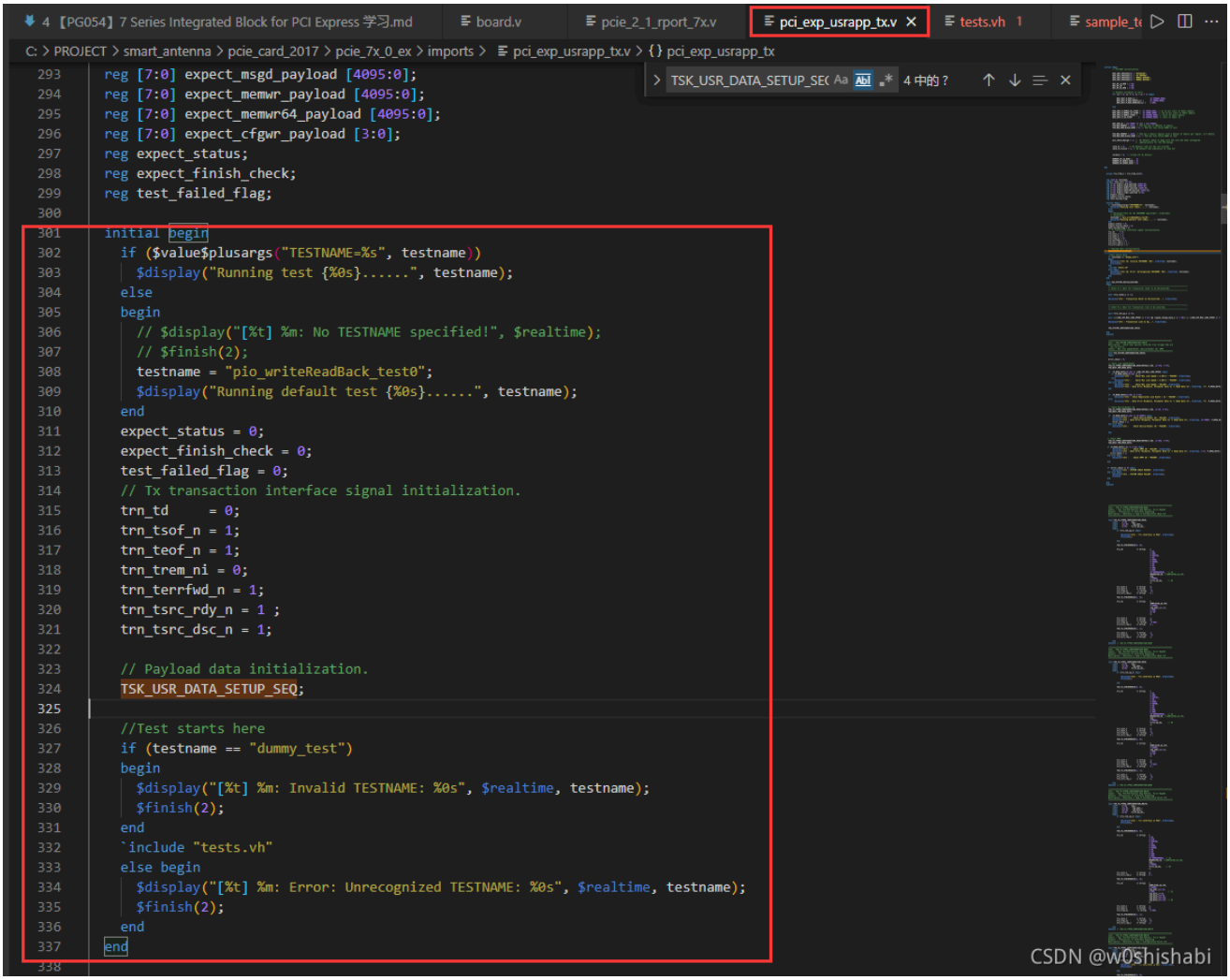
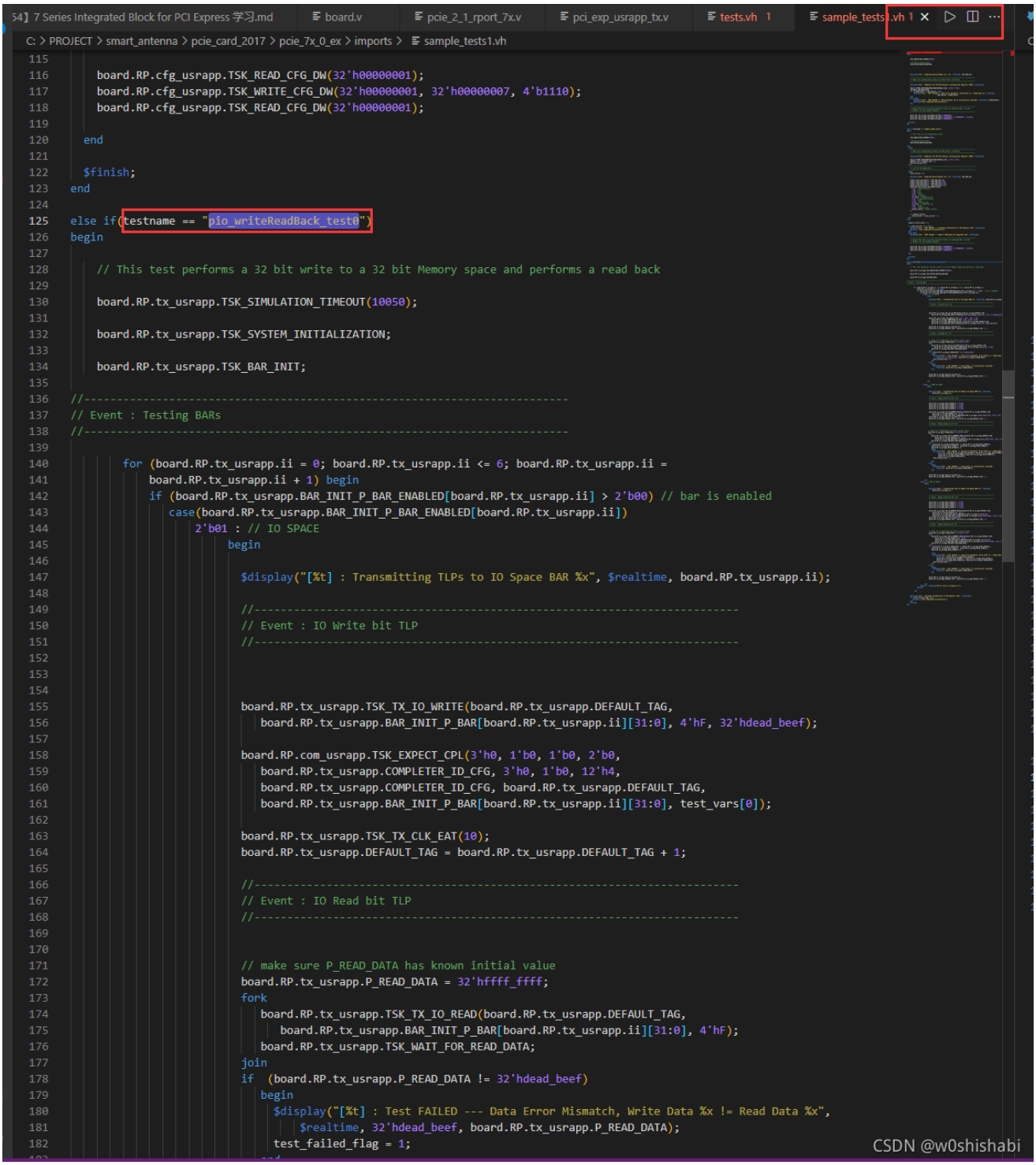


Figure 6-1: **Root Port Model and Top-Level Endpoint**

首先打开testbench中的文件`pci_exp_usrapp_tx.v`, 层次路径为：`board/RP/tx_usrapp`, 重点关注一个initial块：



- 308行：这里指定了测试任务的名称为pio_writeReadBack_test0。
- 324行：这里通过调用TSK_USR_DATA_SETUP_SEQ任务对DATA_STORE进行初始化。
- 332行：这里通过文件包含命令``include "tests.vh"添加了有关测试任务的具体内容。注意verilog的文件包含与c有区别，verilog是直接把tests.vh中的内容直接搬过来，打开tests.vh可以看到只有一句代码：\include “sample_tests1.vh”，sample_tests1.vh中的就是具体的每个测试任务的内容了：



其中，名为pio_writeReadBack_test0的测试任务的具体内容为：

```
else if(testname == "pio_writeReadBack_test0")
begin

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

    board.RP.tx_usrapp.TSK_SIMULATION_TIMEOUT(10050);

    board.RP.tx_usrapp.TSK_SYSTEM_INITIALIZATION;

    board.RP.tx_usrapp.TSK_BAR_INIT;

//------------------------------------------------------------------------
// Event : Testing BARs
//------------------------------------------------------------------------

        for (board.RP.tx_usrapp.ii = 0; board.RP.tx_usrapp.ii <= 6; board.RP.tx_usrapp.ii =
            board.RP.tx_usrapp.ii + 1) begin
            if (board.RP.tx_usrapp.BAR_INIT_P_BAR_ENABLED[board.RP.tx_usrapp.ii] > 2'b00) // bar is enabled
               case(board.RP.tx_usrapp.BAR_INIT_P_BAR_ENABLED[board.RP.tx_usrapp.ii])
                   2'b01 : // IO SPACE
                        begin
```

```
            $display("[%t] : Transmitting TLPs to IO Space BAR %x", $realtime, board.RP.tx_usrapp.ii);

            //------------------------------------------------------------------------
            // Event : IO Write bit TLP
            //------------------------------------------------------------------------



            board.RP.tx_usrapp.TSK_TX_IO_WRITE(board.RP.tx_usrapp.DEFAULT_TAG,
               board.RP.tx_usrapp.BAR_INIT_P_BAR[board.RP.tx_usrapp.ii][31:0], 4'hF, 32'hdead_beef);

            board.RP.com_usrapp.TSK_EXPECT_CPL(3'h0, 1'b0, 1'b0, 2'b0,
               board.RP.tx_usrapp.COMPLETER_ID_CFG, 3'h0, 1'b0, 12'h4,
               board.RP.tx_usrapp.COMPLETER_ID_CFG, board.RP.tx_usrapp.DEFAULT_TAG,
               board.RP.tx_usrapp.BAR_INIT_P_BAR[board.RP.tx_usrapp.ii][31:0], test_vars[0]);

            board.RP.tx_usrapp.TSK_TX_CLK_EAT(10);
            board.RP.tx_usrapp.DEFAULT_TAG = board.RP.tx_usrapp.DEFAULT_TAG + 1;

            //------------------------------------------------------------------------
            // Event : IO Read bit TLP
            //------------------------------------------------------------------------


            // make sure P_READ_DATA has known initial value
            board.RP.tx_usrapp.P_READ_DATA = 32'hffff_ffff;
            fork
               board.RP.tx_usrapp.TSK_TX_IO_READ(board.RP.tx_usrapp.DEFAULT_TAG,
                  board.RP.tx_usrapp.BAR_INIT_P_BAR[board.RP.tx_usrapp.ii][31:0], 4'hF);
               board.RP.tx_usrapp.TSK_WAIT_FOR_READ_DATA;
            join
            if  (board.RP.tx_usrapp.P_READ_DATA != 32'hdead_beef)
               begin
                 $display("[%t] : Test FAILED --- Data Error Mismatch, Write Data %x != Read Data %x",
                     $realtime, 32'hdead_beef, board.RP.tx_usrapp.P_READ_DATA);
                 test_failed_flag = 1;
               end
            else
               begin
                 $display("[%t] : Test PASSED --- Write Data: %x successfully received",
                     $realtime, board.RP.tx_usrapp.P_READ_DATA);
               end


            board.RP.tx_usrapp.TSK_TX_CLK_EAT(10);
            board.RP.tx_usrapp.DEFAULT_TAG = board.RP.tx_usrapp.DEFAULT_TAG + 1;


         end

      2'b10 : // MEM 32 SPACE
         begin


            $display("[%t] : Transmitting TLPs to Memory 32 Space BAR %x", $realtime,
                board.RP.tx_usrapp.ii);

            //------------------------------------------------------------------------
            // Event : Memory Write 32 bit TLP
            //------------------------------------------------------------------------

            board.RP.tx_usrapp.DATA_STORE[0] = 8'h04;
            board.RP.tx_usrapp.DATA_STORE[1] = 8'h03;
            board.RP.tx_usrapp.DATA_STORE[2] = 8'h02;
            board.RP.tx_usrapp.DATA_STORE[3] = 8'h01;

            board.RP.tx_usrapp.TSK_TX_MEMORY_WRITE_32(board.RP.tx_usrapp.DEFAULT_TAG,
                board.RP.tx_usrapp.DEFAULT_TC, 10'd1,
                board.RP.tx_usrapp.BAR_INIT_P_BAR[board.RP.tx_usrapp.ii][31:0]+8'h10, 4'h0, 4'hF, 1'b0);
            board.RP.tx_usrapp.TSK_TX_CLK_EAT(10);
            board.RP.tx_usrapp.DEFAULT_TAG = board.RP.tx_usrapp.DEFAULT_TAG + 1;

            //------------------------------------------------------------------------
            // Event : Memory Read 32 bit TLP
            //------------------------------------------------------------------------


           // make sure P_READ_DATA has known initial value
           board.RP.tx_usrapp.P_READ_DATA = 32'hffff_ffff;
            fork
               board.RP.tx_usrapp.TSK_TX_MEMORY_READ_32(board.RP.tx_usrapp.DEFAULT_TAG,
                   board.RP.tx_usrapp.DEFAULT_TC, 10'd1,
                   board.RP.tx_usrapp.BAR_INIT_P_BAR[board.RP.tx_usrapp.ii][31:0]+8'h10, 4'h0, 4'hF);
               board.RP.tx_usrapp.TSK_WAIT_FOR_READ_DATA;
```

```
            join
            if  (board.RP.tx_usrapp.P_READ_DATA != {board.RP.tx_usrapp.DATA_STORE[3],
               board.RP.tx_usrapp.DATA_STORE[2], board.RP.tx_usrapp.DATA_STORE[1],
               board.RP.tx_usrapp.DATA_STORE[0] })
               begin
                 $display("[%t] : Test FAILED --- Data Error Mismatch, Write Data %x != Read Data %x",
                      $realtime, {board.RP.tx_usrapp.DATA_STORE[3],board.RP.tx_usrapp.DATA_STORE[2],
                       board.RP.tx_usrapp.DATA_STORE[1],board.RP.tx_usrapp.DATA_STORE[0]},
                       board.RP.tx_usrapp.P_READ_DATA);
                 test_failed_flag = 1;

               end
            else
               begin
                 $display("[%t] : Test PASSED --- Write Data: %x successfully received",
                     $realtime, board.RP.tx_usrapp.P_READ_DATA);
               end


            board.RP.tx_usrapp.TSK_TX_CLK_EAT(10);
            board.RP.tx_usrapp.DEFAULT_TAG = board.RP.tx_usrapp.DEFAULT_TAG + 1;

      end
   2'b11 : // MEM 64 SPACE
      begin


            $display("[%t] : Transmitting TLPs to Memory 64 Space BAR %x", $realtime,
                board.RP.tx_usrapp.ii);


            //------------------------------------------------------------------------
            // Event : Memory Write 64 bit TLP
            //------------------------------------------------------------------------

            board.RP.tx_usrapp.DATA_STORE[0] = 8'h64;
            board.RP.tx_usrapp.DATA_STORE[1] = 8'h63;
            board.RP.tx_usrapp.DATA_STORE[2] = 8'h62;
            board.RP.tx_usrapp.DATA_STORE[3] = 8'h61;

            board.RP.tx_usrapp.TSK_TX_MEMORY_WRITE_64(board.RP.tx_usrapp.DEFAULT_TAG,
                board.RP.tx_usrapp.DEFAULT_TC, 10'd1,
                {board.RP.tx_usrapp.BAR_INIT_P_BAR[board.RP.tx_usrapp.ii+1][31:0],
                board.RP.tx_usrapp.BAR_INIT_P_BAR[board.RP.tx_usrapp.ii][31:0]+8'h20}, 4'h0, 4'hF, 1'b0);
            board.RP.tx_usrapp.TSK_TX_CLK_EAT(10);
            board.RP.tx_usrapp.DEFAULT_TAG = board.RP.tx_usrapp.DEFAULT_TAG + 1;

            //------------------------------------------------------------------------
            // Event : Memory Read 64 bit TLP
            //------------------------------------------------------------------------


            // make sure P_READ_DATA has known initial value
            board.RP.tx_usrapp.P_READ_DATA = 32'hffff_ffff;
            fork
               board.RP.tx_usrapp.TSK_TX_MEMORY_READ_64(board.RP.tx_usrapp.DEFAULT_TAG,
                   board.RP.tx_usrapp.DEFAULT_TC, 10'd1,
                   {board.RP.tx_usrapp.BAR_INIT_P_BAR[board.RP.tx_usrapp.ii+1][31:0],
                   board.RP.tx_usrapp.BAR_INIT_P_BAR[board.RP.tx_usrapp.ii][31:0]+8'h20}, 4'h0, 4'hF);
               board.RP.tx_usrapp.TSK_WAIT_FOR_READ_DATA;
            join
            if  (board.RP.tx_usrapp.P_READ_DATA != {board.RP.tx_usrapp.DATA_STORE[3],
               board.RP.tx_usrapp.DATA_STORE[2], board.RP.tx_usrapp.DATA_STORE[1],
               board.RP.tx_usrapp.DATA_STORE[0] })

               begin
                 $display("[%t] : Test FAILED --- Data Error Mismatch, Write Data %x != Read Data %x",
                     $realtime, {board.RP.tx_usrapp.DATA_STORE[3],
                     board.RP.tx_usrapp.DATA_STORE[2], board.RP.tx_usrapp.DATA_STORE[1],
                     board.RP.tx_usrapp.DATA_STORE[0]}, board.RP.tx_usrapp.P_READ_DATA);
                 test_failed_flag = 1;
               end
            else
               begin
                 $display("[%t] : Test PASSED --- Write Data: %x successfully received",
                     $realtime, board.RP.tx_usrapp.P_READ_DATA);
               end


            board.RP.tx_usrapp.TSK_TX_CLK_EAT(10);
            board.RP.tx_usrapp.DEFAULT_TAG = board.RP.tx_usrapp.DEFAULT_TAG + 1;


      end
```

```
                default : $display("Error case in usrapp_tx\n");
            endcase

        end


    $display("[%t] : Finished transmission of PCI-Express TLPs", $realtime);
    if (!test_failed_flag) begin
       $display ("Test Completed Successfully");
    end
    $finish;
end
```

可以看到任务功能为：这个测试执行对32位内存空间的32位写操作，并执行回读操作。

- 首先执行的任务是：`board.RP.tx_usrapp.TSK_SIMULATION_TIMEOUT(10050);`，这个任务的作用是设置仿真的rx_usrapp中的timeout的值。我们可以在`pci_exp_usrapp_rx.v`中看到，这个值的作用是当超过这个值链路如果还没有建立，就会报错并结束仿真。
- `TSK_SYSTEM_INITIALIZATION`任务：这个任务的作用是等待Transaction 复位的完成以及链路的建立。
- `TSK_BAR_INIT`: 基于PCI核的配置初始化PCI核。
- 上面的初始化完成后，就开始进行BAR的读写访问测试了。

2.2 仿真结果

这里使用Modelsim 进行仿真分析。

仿真完成后，可以在[工程名][工程名].sim\sim_1\behav文件夹中的`simulate.log`文件查看到仿真过程中打印的一些消息：

```
# Running default test {pio_writeReadBack_test0}......
# [                   0] : System Reset Asserted...
# [             4995000] : System Reset De-asserted...
# [            64025511] : Transaction Reset Is De-asserted...
# [            65025384] : Transaction Link Is Up...
# [            65057284] : TSK_PARSE_FRAME on Transmit
# [            66473286] : TSK_PARSE_FRAME on Receive
# [            69057326] :    Check Max Link Speed = 2.5GT/s - PASSED
# [            69057326] : Check Negotiated Link Width = 4x - PASSED
# [            69081328] : TSK_PARSE_FRAME on Transmit
# [            70329322] : TSK_PARSE_FRAME on Receive
# [            73081380] :    Check Device/Vendor ID - PASSED
# [            73105380] : TSK_PARSE_FRAME on Transmit
# [            74353380] : TSK_PARSE_FRAME on Receive
# [            77105265] :    Check CMPS ID - PASSED
# [            77105265] : SYSTEM CHECK PASSED
# [            77105265] : Inspecting Core Configuration Space...
# [            77129270] : TSK_PARSE_FRAME on Transmit
# [            77953284] : TSK_PARSE_FRAME on Transmit
# [            78361307] : TSK_PARSE_FRAME on Receive
# [            79201284] : TSK_PARSE_FRAME on Receive
# [            81977392] : TSK_PARSE_FRAME on Transmit
# [            82801289] : TSK_PARSE_FRAME on Transmit
# [            83209392] : TSK_PARSE_FRAME on Receive
# [            84049303] : TSK_PARSE_FRAME on Receive
# [            86825284] : TSK_PARSE_FRAME on Transmit
# [            87649322] : TSK_PARSE_FRAME on Transmit
# [            88057284] : TSK_PARSE_FRAME on Receive
# [            88897326] : TSK_PARSE_FRAME on Receive
# [            91673380] : TSK_PARSE_FRAME on Transmit
# [            92497270] : TSK_PARSE_FRAME on Transmit
# [            92905380] : TSK_PARSE_FRAME on Receive
# [            93745265] : TSK_PARSE_FRAME on Receive
# [            96521284] : TSK_PARSE_FRAME on Transmit
# [            97345392] : TSK_PARSE_FRAME on Transmit
# [            97753284] : TSK_PARSE_FRAME on Receive
# [            98593408] : TSK_PARSE_FRAME on Receive
# [           101369290] : TSK_PARSE_FRAME on Transmit
# [           102193284] : TSK_PARSE_FRAME on Transmit
# [           102601364] : TSK_PARSE_FRAME on Receive
# [           103441284] : TSK_PARSE_FRAME on Receive
# [           106217322] : TSK_PARSE_FRAME on Transmit
# [           107041380] : TSK_PARSE_FRAME on Transmit
# [           107449322] : TSK_PARSE_FRAME on Receive
# [           108289380] : TSK_PARSE_FRAME on Receive
```

```
# [           111041269] PCI EXPRESS BAR MEMORY/IO MAPPING PROCESS BEGUN...
#       BAR 0: VALUE = 00000000 RANGE = ffffe000 TYPE =  MEM32 MAPPED
#       BAR 1: VALUE = 00000000 RANGE = 00000000 TYPE =      DISABLED
#       BAR 2: VALUE = 00000000 RANGE = 00000000 TYPE =      DISABLED
#       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
# [           111041269] : Setting Core Configuration Space...
# [           111065277] : TSK_PARSE_FRAME on Transmit
# [           111889284] : TSK_PARSE_FRAME on Transmit
# [           112297271] : TSK_PARSE_FRAME on Receive
# [           112713380] : TSK_PARSE_FRAME on Transmit
# [           113129284] : TSK_PARSE_FRAME on Receive
# [           113537290] : TSK_PARSE_FRAME on Transmit
# [           113945411] : TSK_PARSE_FRAME on Receive
# [           114361284] : TSK_PARSE_FRAME on Transmit
# [           114769303] : TSK_PARSE_FRAME on Receive
# [           115185322] : TSK_PARSE_FRAME on Transmit
# [           115593284] : TSK_PARSE_FRAME on Receive
# [           116009380] : TSK_PARSE_FRAME on Transmit
# [           116417326] : TSK_PARSE_FRAME on Receive
# [           116833284] : TSK_PARSE_FRAME on Transmit
# [           117241380] : TSK_PARSE_FRAME on Receive
# [           117657284] : TSK_PARSE_FRAME on Transmit
# [           118065265] : TSK_PARSE_FRAME on Receive
# [           118889284] : TSK_PARSE_FRAME on Receive
# [           125657292] : Transmitting TLPs to Memory 32 Space BAR 0
# [           125681289] : TSK_PARSE_FRAME on Transmit
# [           125785277] : TSK_PARSE_FRAME on Transmit
# [           126993313] : TSK_PARSE_FRAME on Receive
# [           129761284] : Test PASSED --- Write Data: 01020304 successfully received
# [           129841285] : Finished transmission of PCI-Express TLPs
# Test Completed Successfully
# ** Note: $finish    : ../../../imports/sample_tests1.vh(325)
#    Time: 129841285 ps  Iteration: 6  Instance: /board/RP/tx_usrapp
# 1
# Break in Module pci_exp_usrapp_tx at ../../../imports/sample_tests1.vh line 325
```

2.2.1 用到的任务说明

首先需要大概知道pio_writeReadBack_test0调用的一些基本子任务的功能，在【PG054】7 Series Integrated Block for PCI Express IP核的学习中2.6.1 Root Port模型TPI任务列表 章节介绍了一些，其他在PG054中可以查到。
其中，主要的几个任务功能大致说明：

任务	说明
TSK_TX_TYPE0_CONFIGURATION_READ	配置0读请求；TLP头大小为3个双字（double word）;不带数据
TSK_TX_TYPE1_CONFIGURATION_READ	配置1读请求；TLP头大小为3个双字（double word）;不带数据
TSK_TX_TYPE0_CONFIGURATION_WRITE	配置0写请求；TLP头大小为3个双字（double word）;带数据
TSK_TX_TYPE1_CONFIGURATION_WRITE	配置1写请求；TLP头大小为3个双字（double word）;带数据

```
TSK_TX_TYPE0_CONFIGURATION_READ(tag_,reg_addr_,first_dw_be_)
报头byte0-7: trn_td <= #(Tcq){
                                1'b0,
                                2'b00,
                                5'b00100,
                                1'b0,
                                3'b000,
                                4'b0000,
                                1'b0,
                                1'b0,
                                2'b00,
                                2'b00,
                                10'b0000000001,  // 32
                                REQUESTER_ID, //COMPLETER_ID_CFG,
                                tag_,
```

```
                                4'b0000,
                                first_dw_be_      // 64
                                };
报头byte8-15：trn_td <= #(Tcq){
                                COMPLETER_ID_CFG,
                                4'b0000,
                                reg_addr_[11:2],
                                2'b00,
                                32'b0
                                };
```

配置请求报文头格式为：

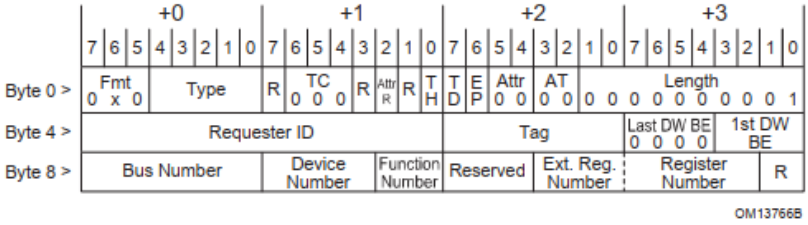
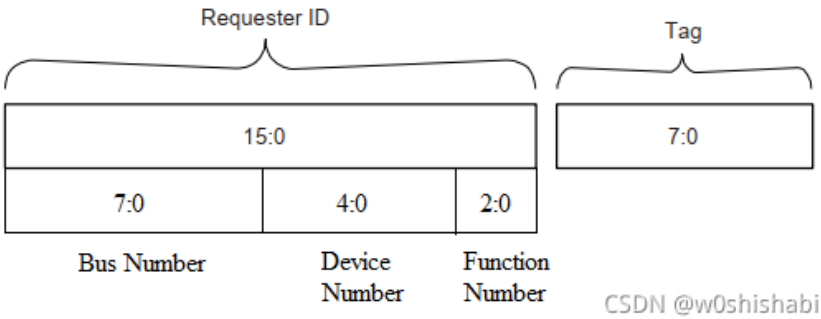


Figure 2-18: Request Header Format for Configuration Transactions

对照TSK_TX_TYPE0_CONFIGURATION_READ任务与对应的配置请求报文头格式，我们就知道具体请求的含义了。以这个任务为例说明，其他任务的分析方法是类似的。

- `REQUESTER_ID = 16'b0000_0001_1010_1111;,tag_`：



Non-Posted请求需要完成报文来应答，才能结束一次完成的数据传递。如果发送端没有收到完成报文，则这个tag不能再次使用，直到这个tag的请求结束。

- `first_dw_be_`： 首先我们可以看到TSK_TX_TYPE0_CONFIGURATION_READ任务给的报文中，`length`字段为10' b0000000001,表示需要从目标设备数据区域读取的数据长度，单位为DW。`1st DW BE`字段对应`first_dw_be_`，为payload数据的掩码，对应bit为1表示对应字节有效。

2.2.2 PCIe配置空间

PCIe配置空间包含三个主要的部分：

1. 传统的PCI V3.0 类型0/1 配置空间头: 0x00~0x3F
    - 类型0 PCI配置空间头（PCI Agent设备的配置空间）

Table 2-24: **Type 0 PCI Configuration Space Header**

31	16	15	0	
Device ID		Vendor ID		00h
Status		Command		04h
Class Code			Rev ID	08h
BIST	Header	Lat Timer	Cache Ln	0Ch
Base Address Register 0				10h
Base Address Register 1				14h
Base Address Register 2				18h
Base Address Register 3				1Ch
Base Address Register 4				20h
Base Address Register 5				24h
Cardbus CIS Pointer				28h
Subsystem ID		Subsystem Vendor ID		2Ch
Expansion ROM Base Address				30h
Reserved			CapPtr	34h
Reserved				38h
Max Lat	Min Gnt	Intr Pin	Intr Line	3Ch

- 类型1 PCI配置空间头（PCI 桥的配置空间）

Table 2-25: **Type 1 PCI Configuration Space Header**

31	16	15	0	
Device ID		Vendor ID		00h
Status		Command		04h
Class Code			Rev ID	08h
BIST	Header	Lat Timer	Cache Ln	0Ch
Base Address Register 0				10h
Base Address Register 1				14h
Second Lat Timer	Sub Bus Number	Second Bus Number	Primary Bus Number	18h
Secondary Status		I/O Limit	I/O Base	1Ch
Memory Limit		Memory Base		20h
Prefetchable Memory Limit		Prefetchable Memory Base		24h
Prefetchable Base Upper 32 Bits				28h
Prefetchable Limit Upper 32 Bits				2Ch
I/O Limit Upper 16 Bits		I/O Base Upper 16 Bits		30h
Reserved			CapPtr	34h
Expansion ROM Base Address				38h
Bridge Control		Intr Pin	Intr Line	3Ch

2. 传统的拓展配置空间： 0x40~0xFF

- PCIe Capability
- Power Management Capability
- Message Signaled Interrupt (MSI) Capability
- MSI-X Capability(optional)

3. PCIe拓展配置空间：0x100~0xFFF

- Device Serial Number Extended Capability Structure (optional)
- Virtual Channel Extended Capability Structure (optional)
- Vendor Specific Extended Capability Structure (optional)
- Advanced Error Reporting Extended Capability Structure (optional)

- Resizable BAR Extended Capability Structure (optional)

Table 2-23: **Common PCI Configuration Space Header**

	31 — 16	15 — 0		
	Device ID	Vendor ID		000h
	Status	Command		004h
	Class Code		Rev ID	008h
	BIST	Header	Lat Timer	Cache Ln	00Ch
	Header Type Specific (see Table 2-24 and Table 2-25)			010h–030h
			CapPtr	034h
				038h
		Intr Pin	Intr Line	03Ch
	PM Capability	NxtCap	PM Cap	040h
	Data / BSE	PMCSR		044h
Customizable[1]	MSI Control	NxtCap	MSI Cap	048h
	Message Address (Lower)			04Ch
	Message Address (Upper)			050h
	Reserved	Message Data		054h
	Mask Bits			058h
	Pending Bits			05Ch
	PE Capability	NxtCap	PE Cap	060h
	PCI Express Device Capabilities			064h
	Device Status	Device Control		068h
	PCI Express Link Capabilities			06Ch
	Link Status	Link Control		070h
Root Port Only[2]	Slot Capabilities			074h
	Slot Status	Slot Control		078h
	Root Capabilities	Root Control		07Ch
	Root Status			080h
	PCI Express Device Capabilities 2			084h
	Device Status 2	Device Control 2		088h
	PCI Express Link Capabilities 2			08Ch
	Link Status 2	Link Control 2		090h
	Unimplemented Configuration Space (Returns **0x00000000**)			094h–098h
Optional	MSIx Control	NxtCap	MSIx Cap	09Ch
	Table Offset		Table BIR	0A0h
	PBA Offset		PBA	0A4h

			BIR	
	Reserved Legacy Configuration Space (Returns **0x00000000**)			0A8h-0FFh
Optional[3]	Next Cap	Cap. Ver.	PCI Express Extended Capability - DSN	100h
	PCI Express Device Serial Number (1st)			104h
	PCI Express Device Serial Number (2nd)			108h
Optional[3]	Next Cap	Cap. Ver.	PCI Express Extended Capability - VC	10Ch
	Port VC Capability Register 1			110h
	Port VC Capability Register 2			114h
	Port VC Status		Port VC Control	118h
	VC Resource Capability Register 0			11Ch
	VC Resource Control Register 0			120h
	VC Resource Status Register 0			124h
Optional[3]	Next Cap	Cap. Ver.	PCI Express Extended Capability - VSEC	128h
	Vendor Specific Header			12Ch
	Vendor Specific - Loopback Command			130h
	Vendor Specific - Loopback Status			134h
	Vendor Specific - Error Count #1			138h
	Vendor Specific - Error Count #2			13Ch
Optional[3]	Next Cap	Cap. Ver.	PCI Express Extended Cap. ID (AER)	140h
	Uncorrectable Error Status Register			144h
	Uncorrectable Error Mask Register			148h
	Uncorrectable Error Severity Register			14Ch
	Correctable Error Status Register			150h
	Correctable Error Mask Register			154h
	Advanced Error Cap. & Control Register			158h
	Header Log Register 1			15Ch
	Header Log Register 2			160h
	Header Log Register 3			164h
	Header Log Register 4			168h
Optional, Root Port only[3]	Root Error Command Register			16Ch
	Root Error Status Register			170h
	Error Source ID Register			174h
Optional[3]	Next Cap	Cap. Ver.	PCI Express Extended Cap. ID (RBAR)	178h
	Resizable BAR Capability Register(0)			17Ch
	Reserved		Resizable BAR Control(0)	180h
	Resizable BAR Capability Register(1)			184h
	Reserved		Resizable BAR Control(1)	188h
	Resizable BAR Capability Register(2)			18Ch
	Reserved		Resizable BAR Control(2)	190h
	Resizable BAR Capability Register(3)			194h
	Reserved		Resizable BAR Control(3)	198h
	Resizable BAR Capability Register(4)			19Ch
	Reserved		Resizable BAR Control(4)	1A0h
	Resizable BAR Capability Register(5)			1A4h
	Reserved		Resizable BAR Control(5)	1A8h

Reserved Extended Configuration Space (Returns Completion with **0x00000000**)	1ACh-FFFh

Notes:

1. The MSI Capability Structure varies depending on the selections in the Vivado Integrated Design Environment (IDE).
2. Reserved for Endpoint configurations (returns **0x00000000**).
3. The layout of the PCI Express Extended Configuration Space (**100h-FFFh**) can change depending on which optional capabilities are enabled. This table represents the Extended Configuration space layout when all five optional extended capability structures are enabled. For more information, see Optional PCI Express Extended Capabilities.

2.2.2 仿真执行过程分析：

1. pci_exp_usrapp_tx的相关初始化

可以看到前两条消息为执行pci_exp_usrapp_tx的相关初始化过程发出的，表示执行的任务名称为{pio_writeReadBack_test0}。

2. pio_writeReadBack_test0.line6: 设置board.RP.rx_usrapp.sim_timeout的值为10050。

3. line8: TSK_SYSTEM_INITIALIZATION

- 等待事务层复位完成，完成后输出消息“Transaction Reset Is De-asserted...”
- 等待事务层链路建立，完成后输出消息“Transaction Link Is Up...”
- 检查IP核生成器中的相关设置，包括Max Link Speed/Width, Device/Vendor ID, CMPS

检查Link Speed/Width：

- TSK_TX_TYPE0_CONFIGURATION_READ(DEFAULT_TAG, 12’ h70, 4’ hF); //发送tag为0的，寄存器地址为12’ h70，Register Number为10’ h1C(由于都是32位的寄存器，所以寄存器号对应地址取高10位), first_dw_be_为4’ b1111的Type 0 PCI Express Config Read TLP。

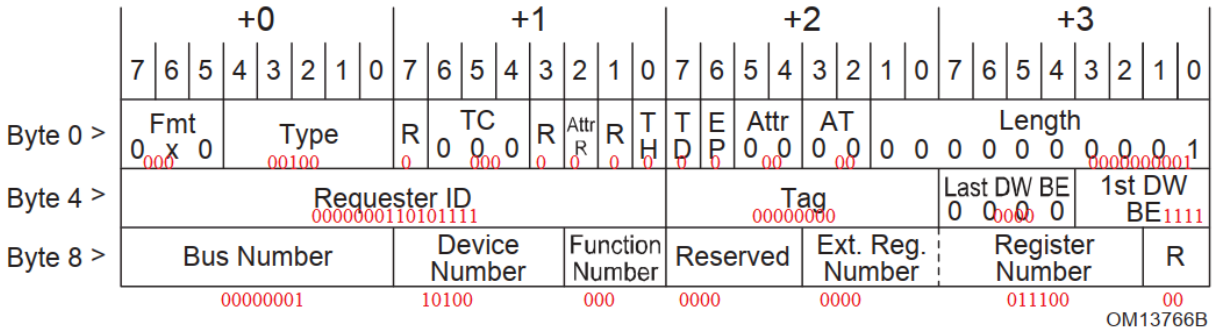


Figure 2-18: Request Header Format for Configuration Transactions

查找PCIe标准，PCIe配置空间地址12’ h70对应的寄存器名称为：Link Status(31:16) Link Control(15:0)

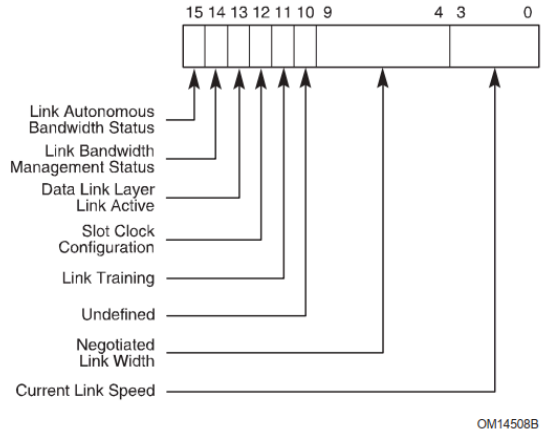


Figure 7-18: Link Status Register

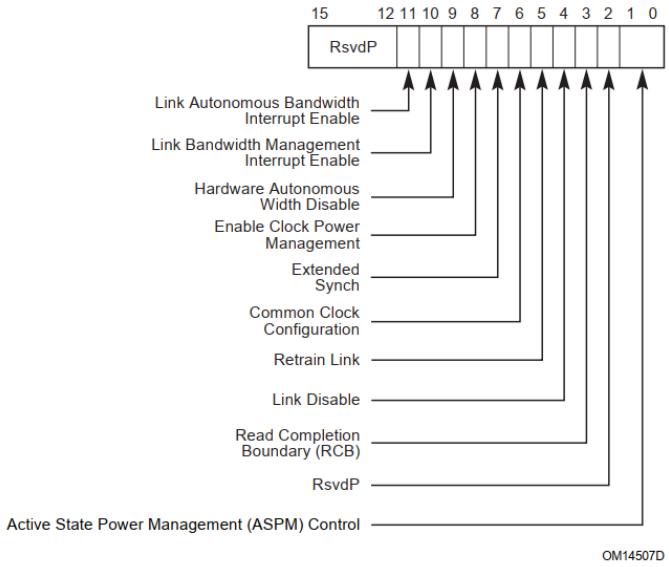


Figure 7-17: Link Control Register

- TSK_WAIT_FOR_READ_DATA：等待回包数据的到来。

关于接收数据这里不细说，大概的路径是：pci_exp_usrapp_rx收到数据 -> board.RP.com_usrapp.TSK_READ_DATA对数据进行处理 -> 收到的数据按字节存储在board.RP.com_usrapp.frame_store_rx[board.RP.com_usrapp.frame_store_rx_idx]中，其中指针frame_store_rx_idx随着收到的数据自增 -> board.RP.com_usrapp.TSK_PARSE_FRAME对这一帧数据进行解析。其中首先执行TSK_DECIPHER_FRAME对此帧进行解析，输出fmt, f_type, traffic_class, td, ep, attr, length等信息，然后执行TSK_3DW任务根据解析出来的信息进一步解析，然后执行board.RP.tx_usrapp.TSK_SET_READ_DATA将解析后的数据传递给tx_app, 在tx_usrapp中将收到的数据存在寄存器P_READ_DATA中。

- 最后对收到的数据进行比对，看是哪一种Link Speed/Width

可以看到P_READ_DATA=32’h10410000=32’b00010000_01000001_00000000_00000000。Link Status Register.Current Link Speed字段 = P_READ_DATA[19:16] = 4’b0001。所以输出消息“Check Max Link Speed = 2.5GT/s - PASSED”。

Link Status Register.Negotiated Link Width字段 = P_READ_DATA[23:20] = 4’b0100。所以输出消息“Check Negotiated Link Width = 4x - PASSED”。

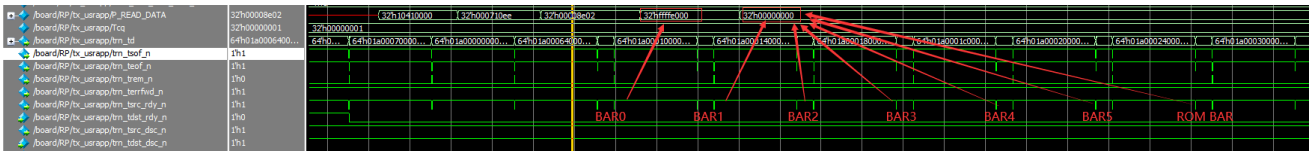
检查Check Device/Vendor ID 与 检查CMPS 的分析过程类似，这里不再赘述。

4. line10: TSK_BAR_INIT

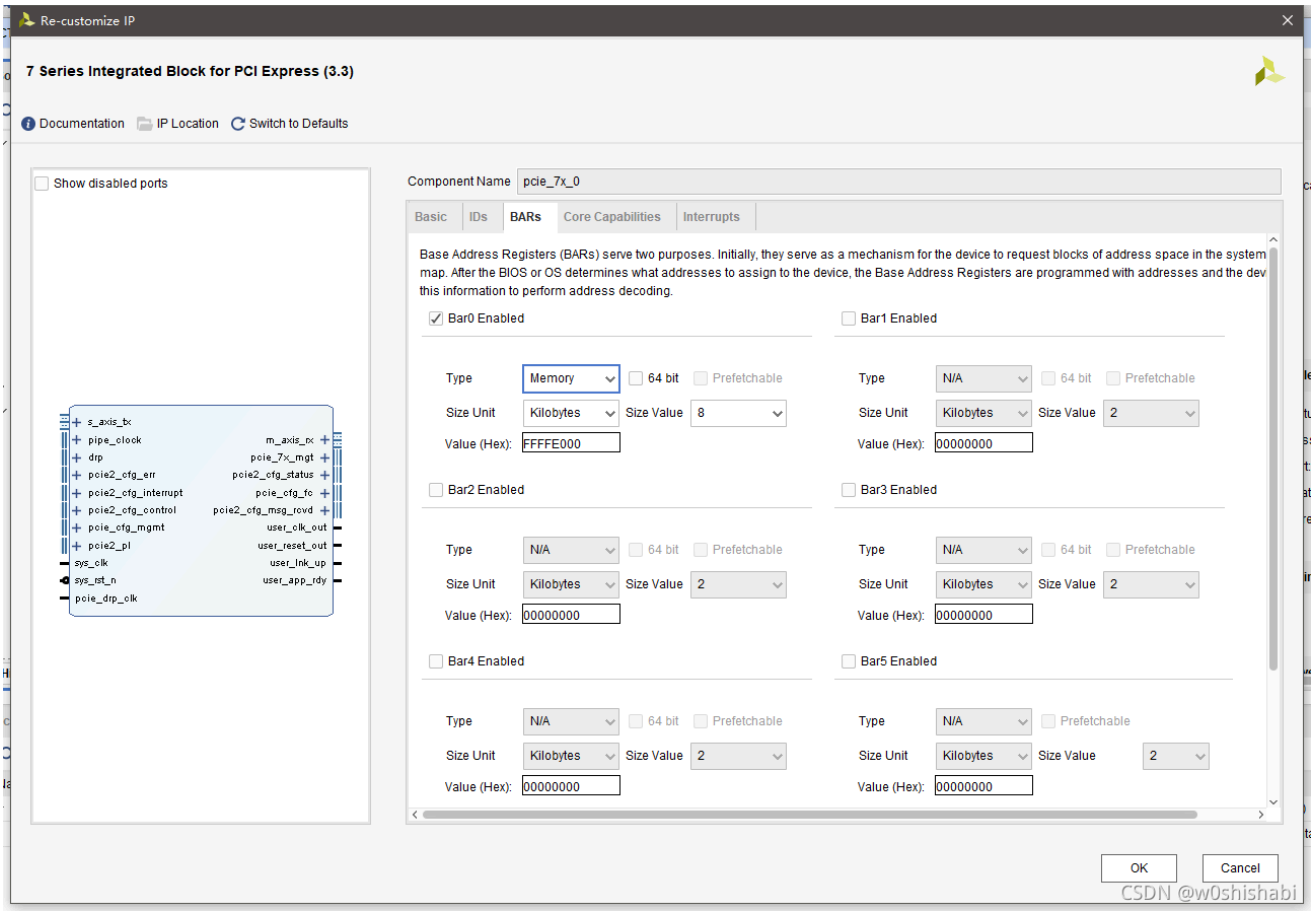
- TSK_BAR_SCAN：扫描PCIe核的配置寄存器。
  通过PCIe fabric接口将PCI_MASK写入bar的空间来查找范围。

PCI总线规范规定了获取BAR空间的标准实现方法。其步骤是首先向BAR寄存器全写1，之后再读取BAR寄存器的内容，即可获得BAR空间的大小。PCIe兼容PCI。

所以我们可以看到这个任务分别向12’h10（BAR0）、12’h14（BAR1）、12’h18（BAR2）、12’h1C（BAR3）、12’h20（BAR4）、12’h24（BAR5）和12’h30（ROM BAR）写了P_ADDRESS_MASK（32’hffff_ffff），再读取BAR的内容即范围并保存到寄存器BAR_INIT_P_BAR_RANGE：



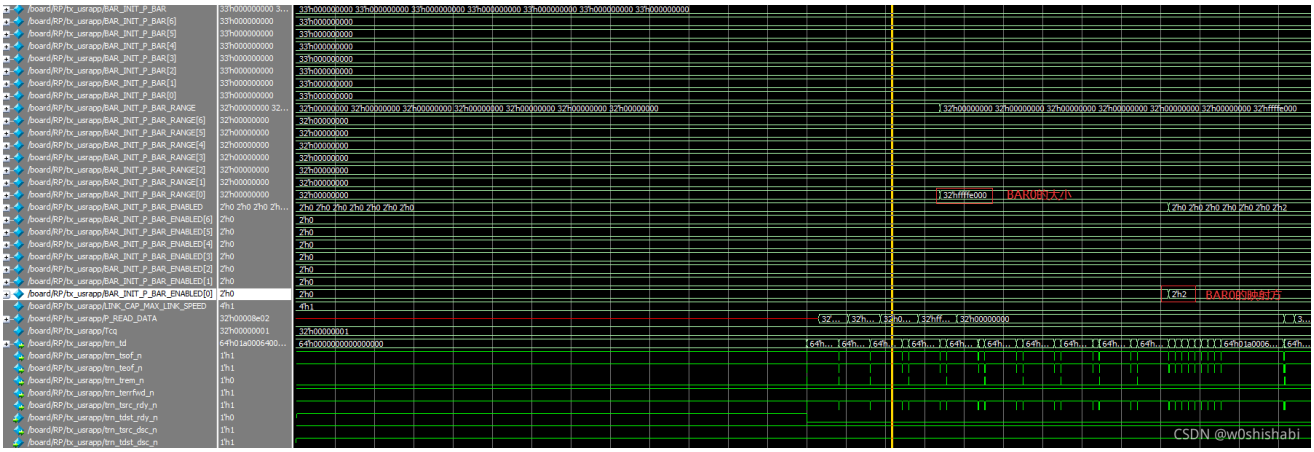
当然，与我们设置的参数一致：



- TSK_BUILD_PCIE_MAP: 从配置空间查看查看范围值，建立相应的mem/io映射。

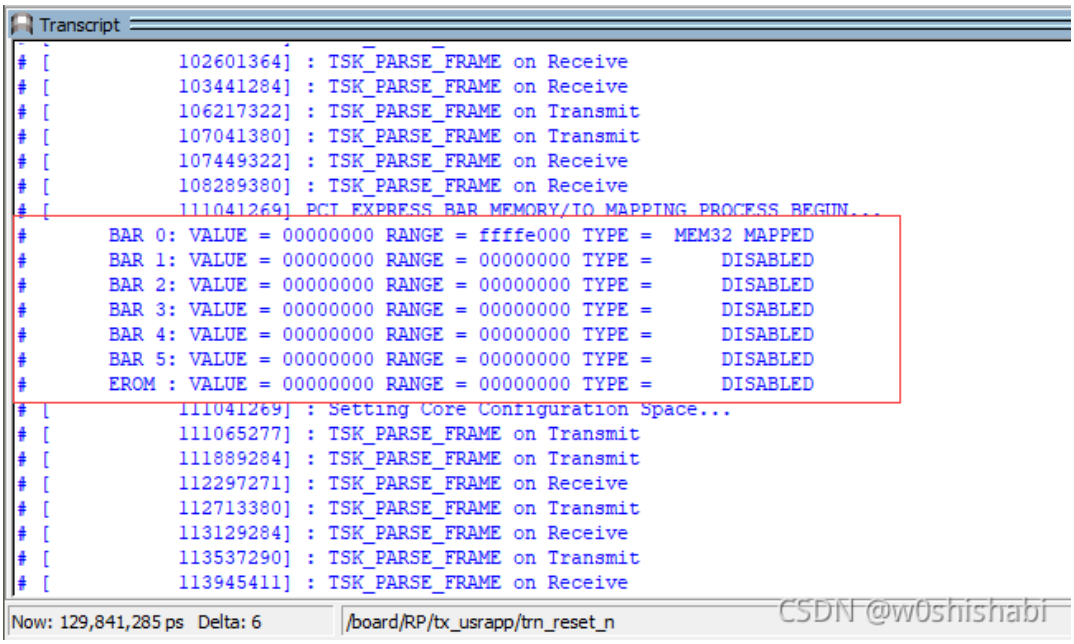
可以看到这里根据读取的一些信息对寄存器BAR_INIT_P_BAR_ENABLED进行赋值，各BAR对应此寄存器的含义为：

值	含义
0	disable
1	bar is io mapped
2	bar is mem32 mapped
3	bar is mem64 mapped



- TSK_DISPLAY_PCIE_MAP: 基于从PCI_E设备读取的范围值，显示出来。

消息line47~53:



- TSK_BAR_PROGRAM: 配置的PCIe核的配置寄存器。

5. line16~194: 进行BAR映射空间的读写测试

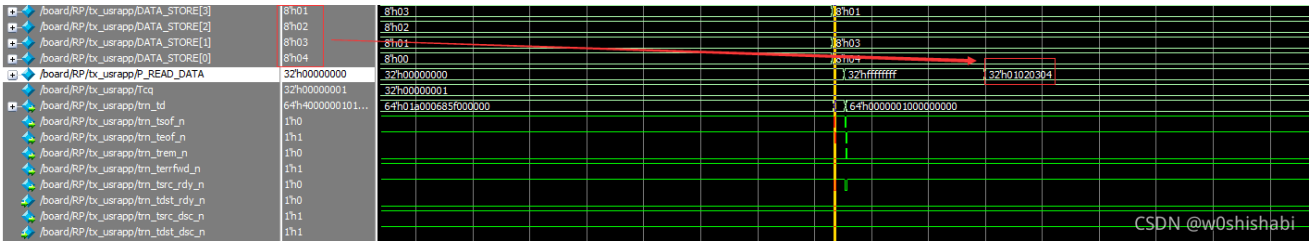
首先判断BAR是否使能，然后判断BAR的映射方式。

- TSK_TX_MEMORY_WRITE_32:

往board.RP.tx_usrapp.BAR_INIT_P_BAR[board.RP.tx_usrapp.ii][31:0]+8’h10，也就是8’h10中写{DATA_STORE[0],DATA_STORE[1],DATA_STORE[2],DATA_STORE[3]}={8’h04,8’h03,8’h02,8’h01}。

- TSK_TX_MEMORY_READ_32:

读取这个地址的数据并进行比对，如果一直输出消息“Test PASSED — Write Data: 01020304 successfully received”。



整个仿真的过程就是这样啦。

//每次摆开阵势想要好好看看，总被打断，耗了这么久终于分析完了，通过这个仿真，可以很好的理解PCIe的相关知识。包括事务类型，BAR的配置与分析等等，还是很不错的过程。