Using HSDP over PCIe with Alpha Data Boards with AMD Versal Adaptive SoC

Introduction

Alpha Data PCIe devices featuring AMD Versal Adaptive SoC support HSDP over PCIe. This feature allows access to the DPC over the PCIe interface. Additional configuration of the design is required to support this feature. This guide will demonstrate the configuration of a PCIe design with debug functionality.

HSDP Overview

The HSDP is a high-speed debug feature available in high-end AMD devices. It enables high-speed bidirectional debug and trace operations over a Gigabit Transceiver (GT) channel. In general, Versal devices provide three high-speed pathways to and from the Debug Packet Controller (DPC) through the GT channel. Access to this GT channel can be provisioned via an integrated Aurora 64B/66B block in the PS (hard Aurora interface), via CPM/PCIe, or through a soft Aurora IP instantiated in the Programmable Logic (PL).

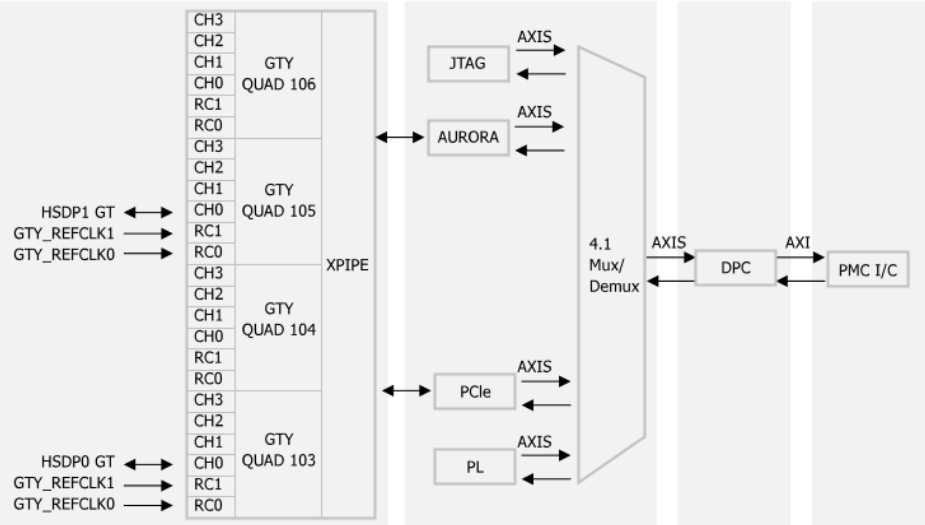

Figure 1 : HSDP pathways to the DPC

HSDP over PCIe

There are two methods that can be used to debug using HSDP over PCIe, **User Mode** and **Management Mode**. User mode is a simpler configuration suitable for debug of non-hardened debug cores. Management Mode provides access to hardened debug cores and has increased performance compared to User Mode, but requires additions to the design.

This guide is based on the **cpm_demo** example provided in the product SDK for Alpha Data high-performance reconfigurable computing cards that feature the AMD Versal Adaptive SoC device.

Pre-requisites

* Host PC with Vivado 2025.2
* **cpm_demo**
* Linux PC with Vivado 2025.2 Hardware Server (Host or Remote)
* Linux PC with Alpha Data high-performance reconfigurable computing card featuring the AMD Versal Adaptive SoC device. This application note is applicable but not limited to the following Alpha Data devices:
  - **ADM-PA100** https://alpha-data.com/product/adm-pa100/
  - **ADM-PA101** https://alpha-data.com/product/adm-pa101/
  - **ADM-PA120** https://alpha-data.com/product/adm-pa120/
  - **ADM-PB125** https://alpha-data.com/product/adm-pb125/
  - **ADM-VA210** https://alpha-data.com/product/adm-va210/
  - **ADM-VA601** https://alpha-data.com/product/adm-va601/
  - **ADM-VB630** https://alpha-data.com/product/adm-vb630/
  - **ADM-XA210** https://alpha-data.com/product/adm-xa210/
* **hsdp-pcie-driver** https://github.com/Xilinx/hsdp-pcie-driver

Configuring the CIPS PS PMC Module

The platform management controller subsystem must be configured to enable the HSDP over PCIe functionality.

1) Open the customization window for the **Control, Interfaces & Processing System (cips)** block.

2) Open the configuration for the **PS PMC** module.

3) Select the **Debug** option.

4) Select the **HSDP** tab.

5) In the **Pathway to/from Debug Packet Controller (DPC)** dropdown, select the **PCIE** option. This will enable Master/Slave ports for the PMC for debug functionality.

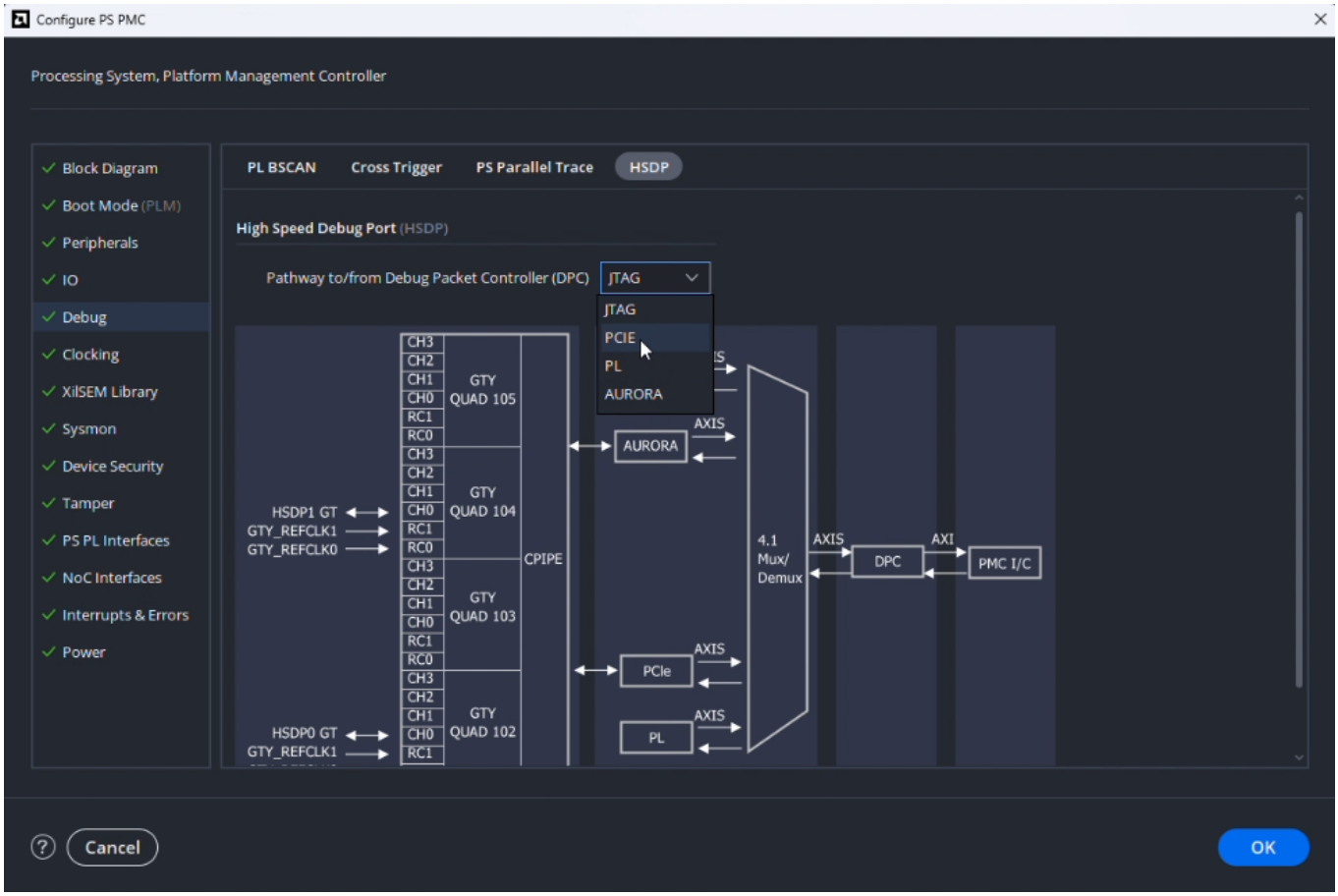

Figure 2 : Pathway to/from Debug Packet Controller (DPC) PCIE Option

Configuring the CIPS CPM Module

The CPM module provides the PCIe interface configuration for HSDP over PCIe. The configuration of the CPM module differs depending on whether User or Management mode debug functionality is required.

1) Open the customization window for the **Control, Interfaces & Processing System (cips)** block.

2) Open the configuration for the **CPM** module.

3) Select the **CPM5 PCIe Controller 0** option.

4) Check the **Enable Slave Bridge** Checkbox.

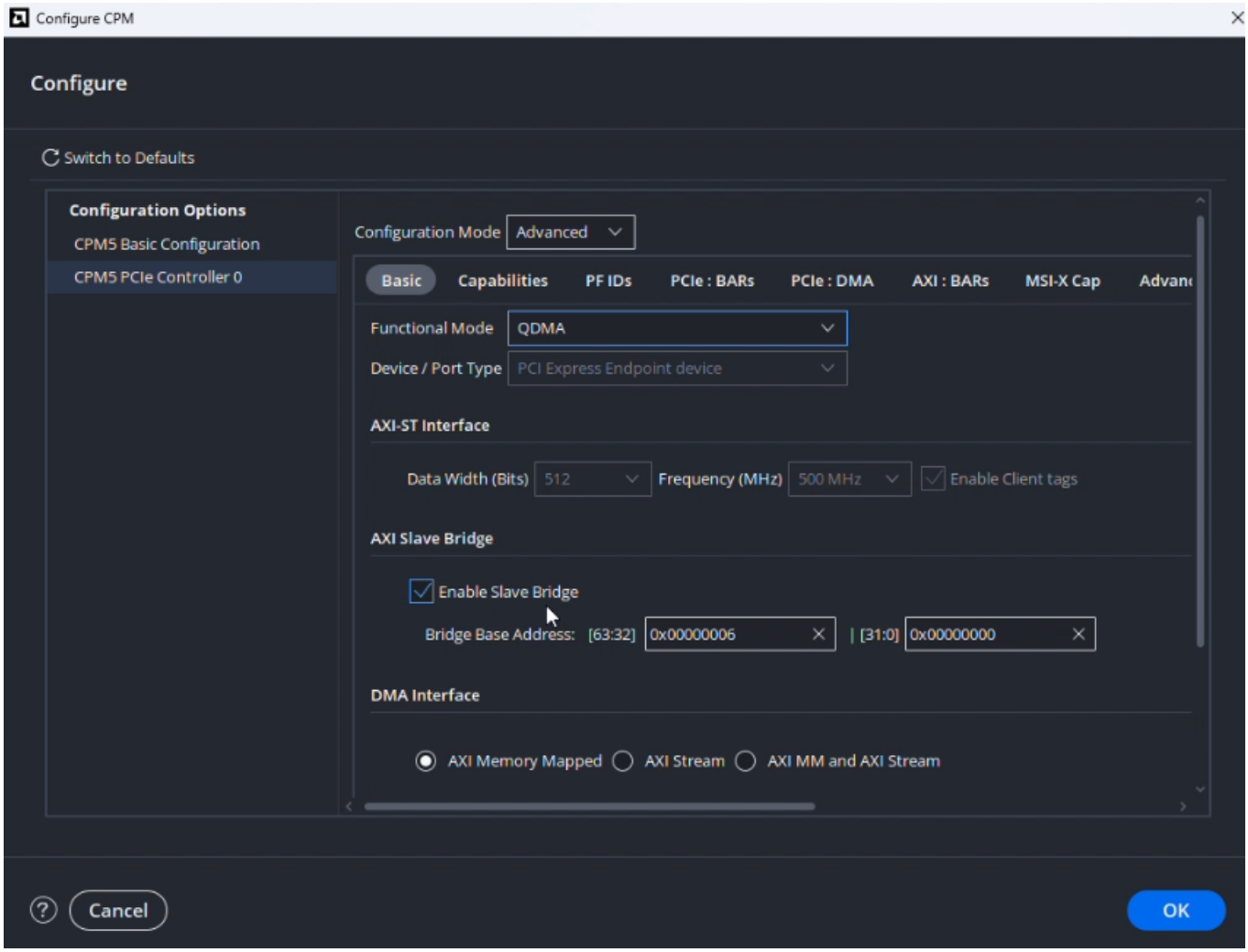

Figure 3 : Enable Slave Bridge Option

5) Select the **PCIe : BARs** tab.

6) To enable **User Mode**, configure BAR4, ensuring PCIe to AXI Translation address is configured to the Master Base Address of your AXI4 Debug Hub (See Section 8).

7) To enable **Management Mode**, configure BAR2, ensuring PCIe to AXI Translation address is configured to **0xFC000000**.

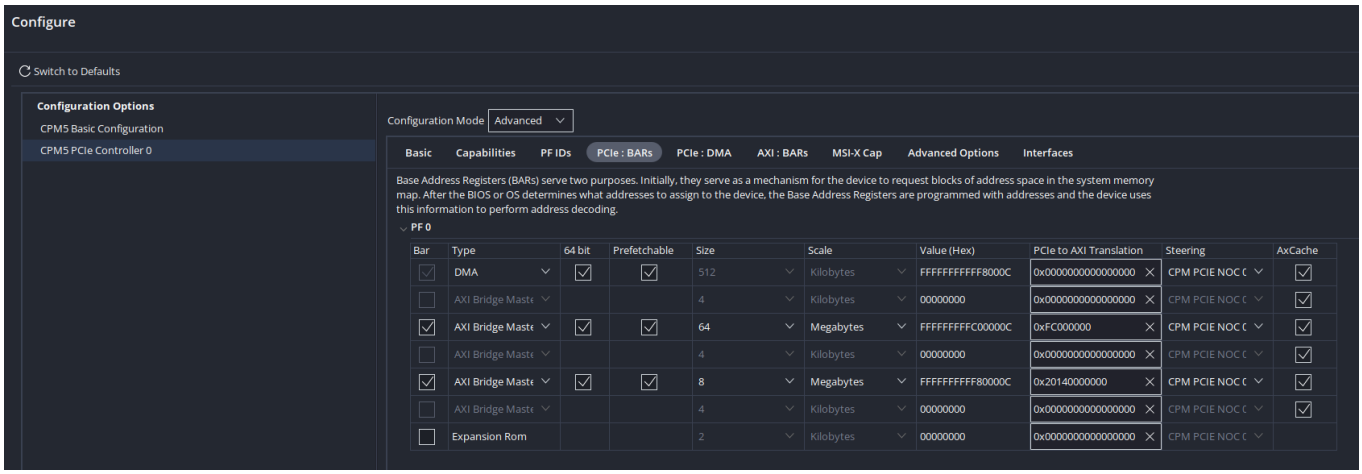

Figure 4 : PCIE BARs Configuration

8) To enable **Management Mode**, at least one PCIe BAR must be configured in the **AXI : BARs** tab.

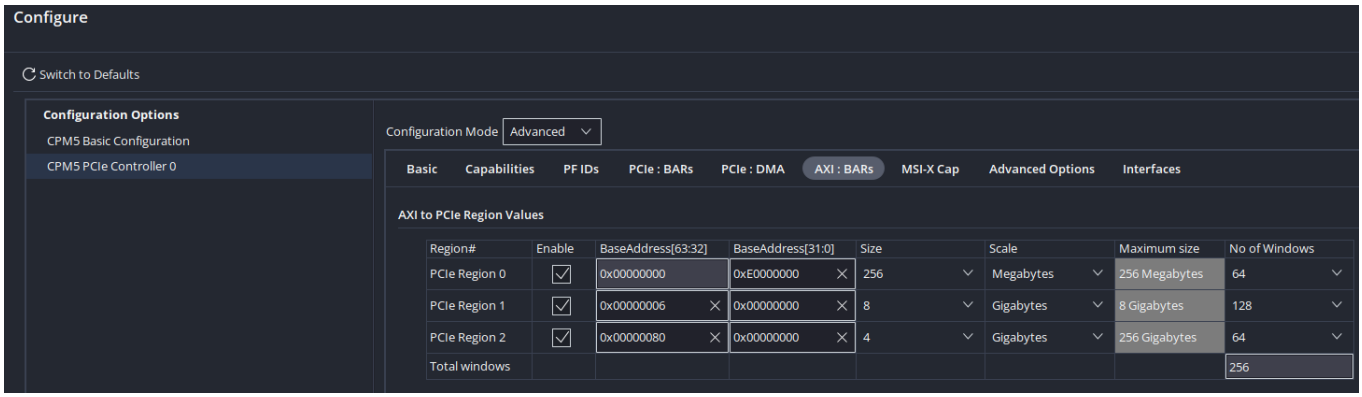

Figure 5 : AXI BARs Configuration

Configuring the AXI NoC

An AXI NoC must be configured to provide interconnect between the CIPS module and Debug hub(s). This section demonstrates the extension of the AXI NoC in the **cpm_demo** design with connections for the HSDP over PCIe configuration for PMC and CPM module.

1) Open the customization window for the **AXI NoC** block.

2) Select **General** tab.

3) In the **Number of AXI Clocks** dropdown, select **6**. This will add additional AXI clock pins for the newly generated interfaces for the PMC domain and AXI Slave interface added to the CIPS:

* **aclk3** clock for **NOC_PMC_AXI_0** slave port.
* **aclk4** clock for **PMC_NOC_AXI_0** master port.
* **aclk5** clock for **NOC_CPM_PCIE_0** slave port.

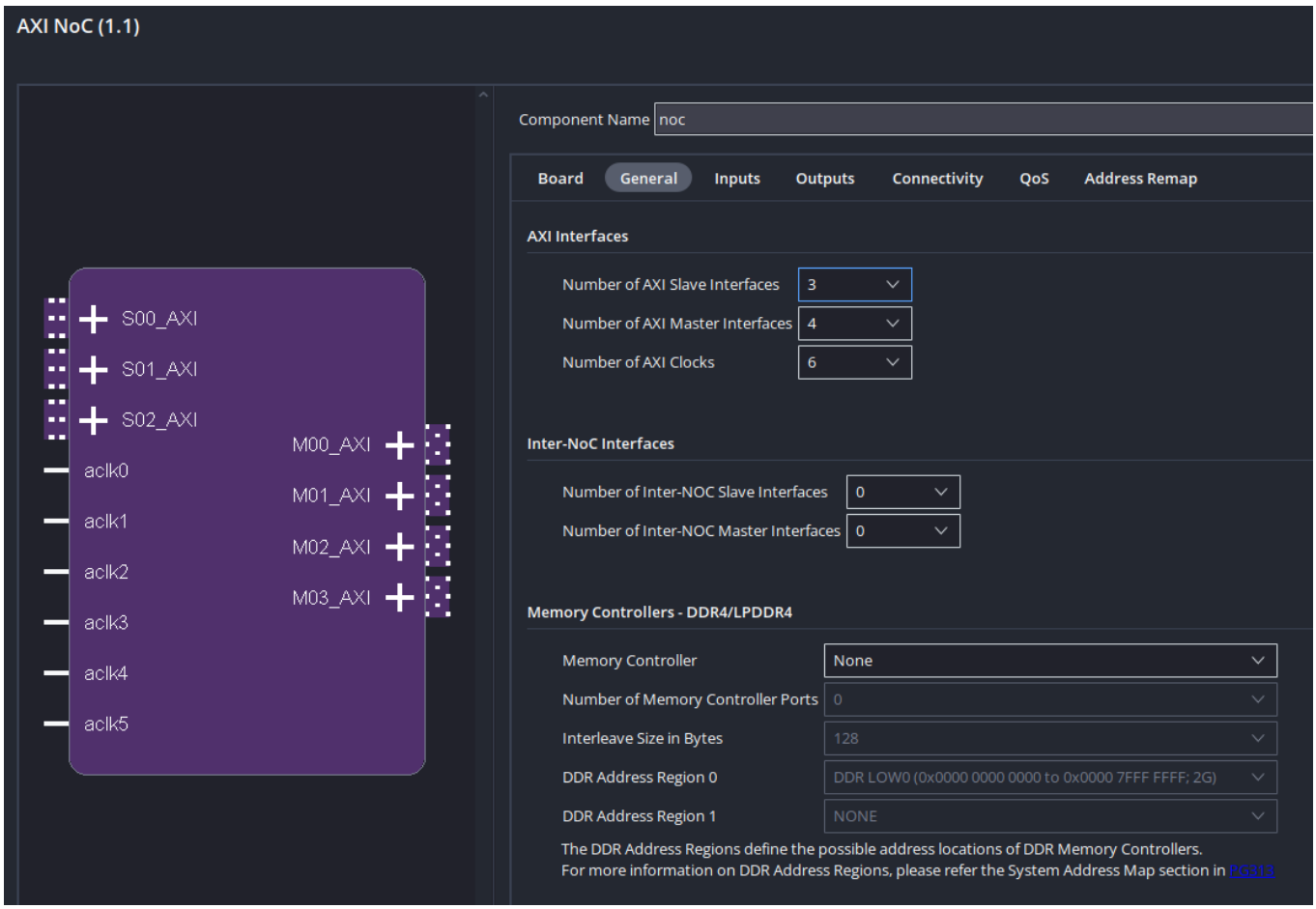

Figure 6 : General Configuration

4) Select **Inputs** tab.

5) For the **cpm_demo** design, 3 AXI Slave interfaces are configured:

* **S00_AXI** PS PCIe connection for **CPM_PCIE_NOC_0**. This is an interface that will be used to propagate data from the CPM to AXI NoC Master interfaces. In User Mode, this connection directly maps to an AXI4 Debug Hub. In Management Mode, this connection connects to the HSDP DMA Block in the PMC.
* **S01_AXI** PS PCIe connection for **CPM_PCIE_NOC_1**. This is an interface that will be used to propagate data from the CPM to AXI NoC Master interfaces. In User Mode, this connection directly maps to an AXI4 Debug Hub. In Management Mode, this connection connects to the HSDP DMA Block in the PMC.
* **S02_AXI** PS PMC connection for **PMC_NOC_AXI_0**. This is an interface used by the PMC to control the

DPC in Management Mode.

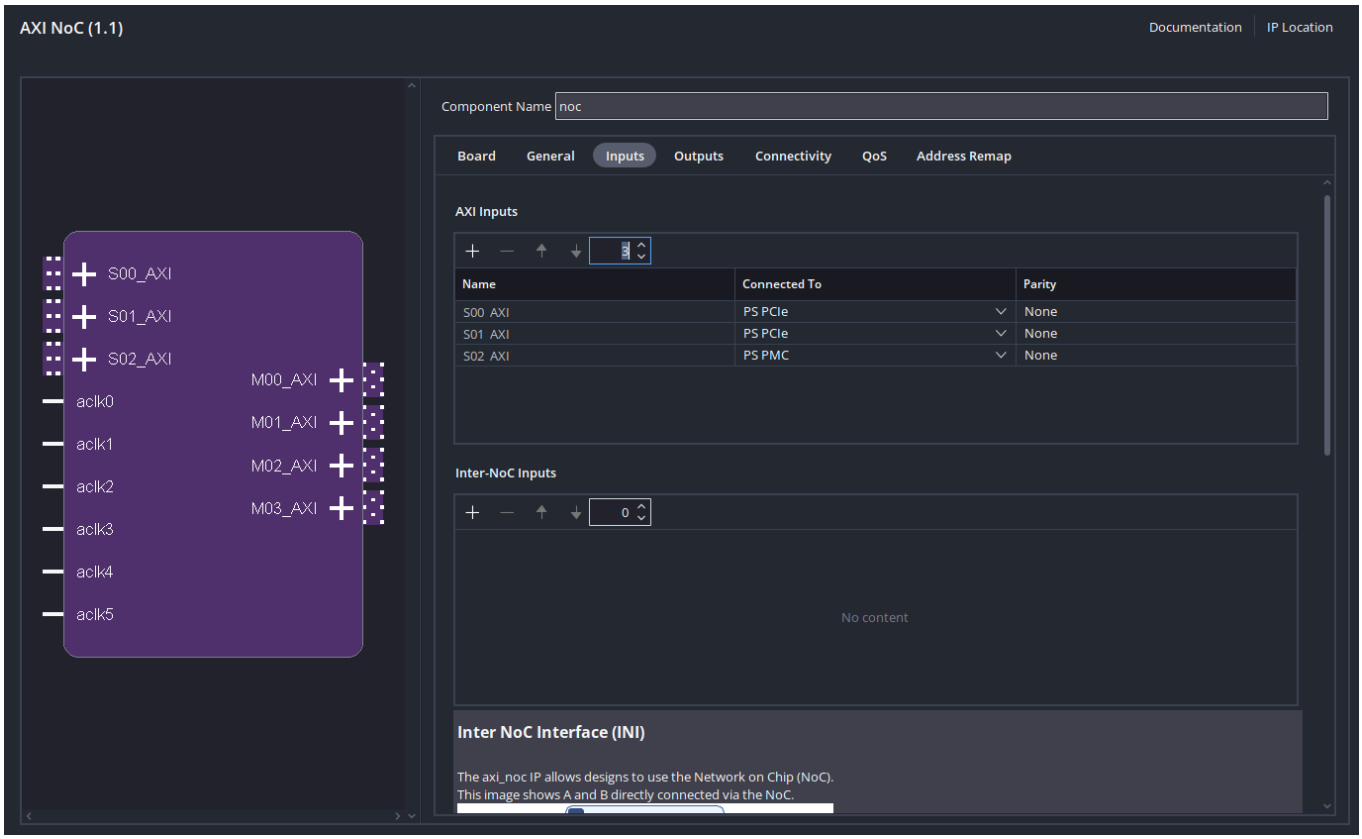

Figure 7 : Inputs Configuration

6) Select **Outputs** tab.

7) For the **cpm_demo** design, 4 AXI Master interfaces are configured:

* **M00_AXI** PS PCIe connection for **NOC_CPM_PCIE_0**. This is an interface that is used in management mode for DMA transfers to target and transfer data to the DPC.
* **M01_AXI** PL connection for **S_AXI** interface on AXI BRAM Controller **uram_ctl**. This connection retains compatibility with the functionality of **cpm_demo**.
* **M02_AXI** PL connection for **S_AXI** interface on example **dbg** module. This provides a direct connection to the CPM in User Mode, and a connection to the PMC in management mode.
* **M03_AXI** PS PMC connection for **NOC_PMC_AXI_0**. This provides a connection from the CPM to the PMC AXI Slave and HSDP DMA block in Management Mode.

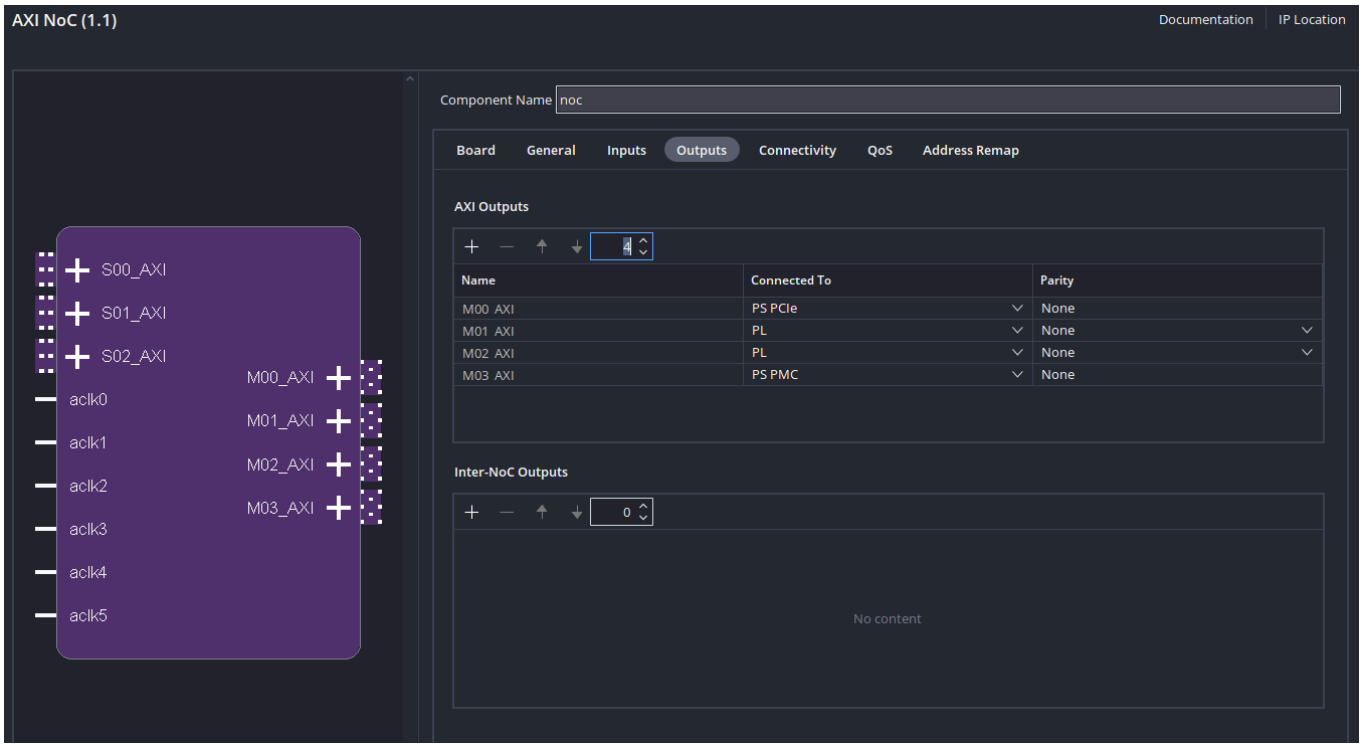

Figure 8 : Outputs Configuration

8) Select the **Connectivity** tab.

9) Create connections for each Slave/Master. No connection is needed between **S02_AXI** and **M03_AXI**. This example connectivity demonstrates all connections necessary to use User Mode and Management Mode debug functionality.

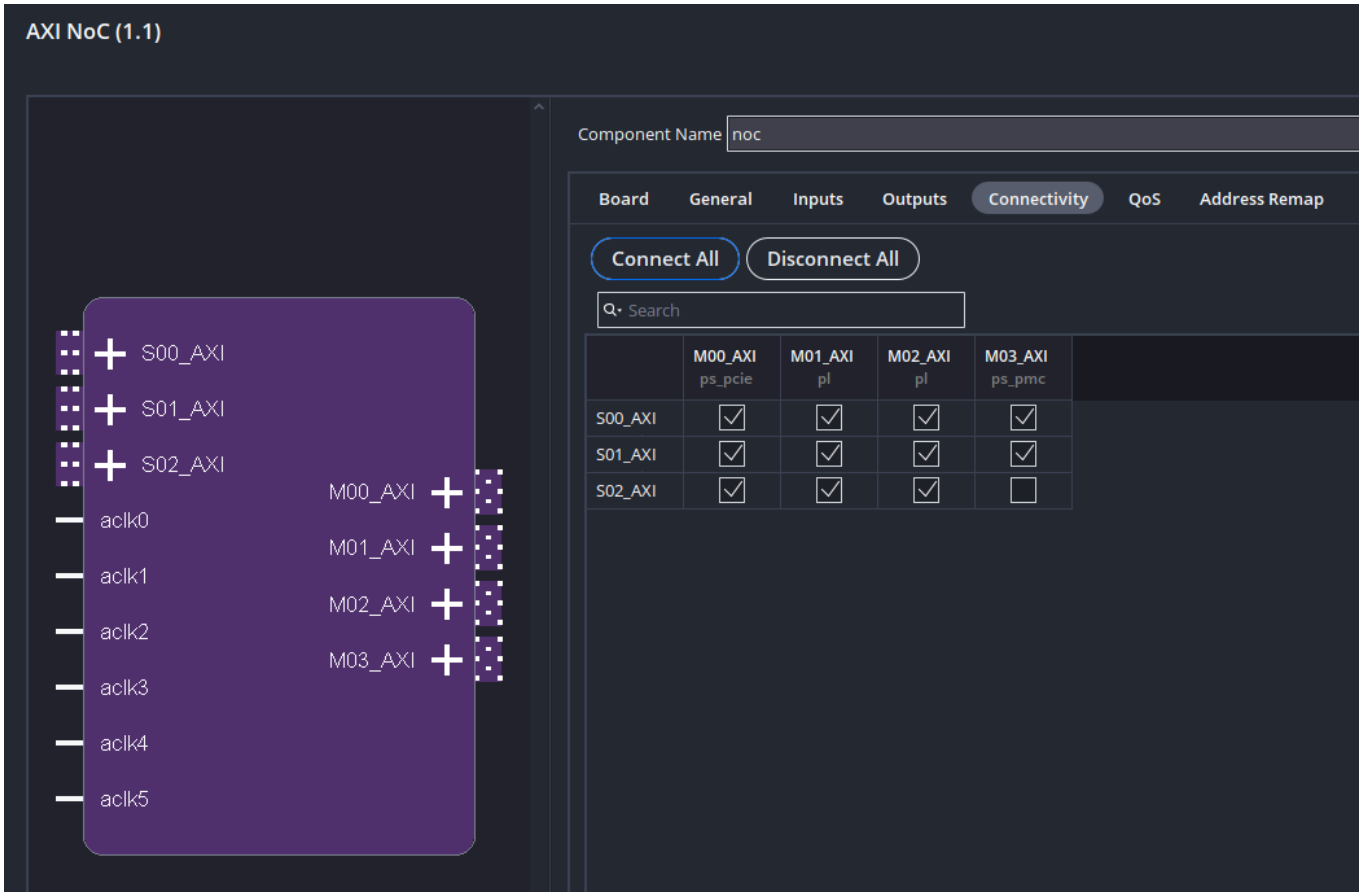

Figure 9 : Connectivity Configuration

Configuring a PL Debug Target

In addition to the changes to the **cpm_demo** example design needed to enable HSDP over PCIe, a Debug Target is required. In this example, an ILA and VIO core are attached to a debug hub. The VIO controls an external binary counter intended to simulate a debug scenario, however this implementation can also be adapted and applied to a variety of debug scenarios. Example Tcl code is provided that can be directly added to the **cpm_demo** or adapted to user code. The instructions below can be followed to create this module.

1) Create a module containing **ILA**, **AXI4 Debug Hub** and **VIO** blocks.

2) Open the customization window for the **AXI4 Debug Hub** block.

3) Set **Number of Debug Cores** to **2**.

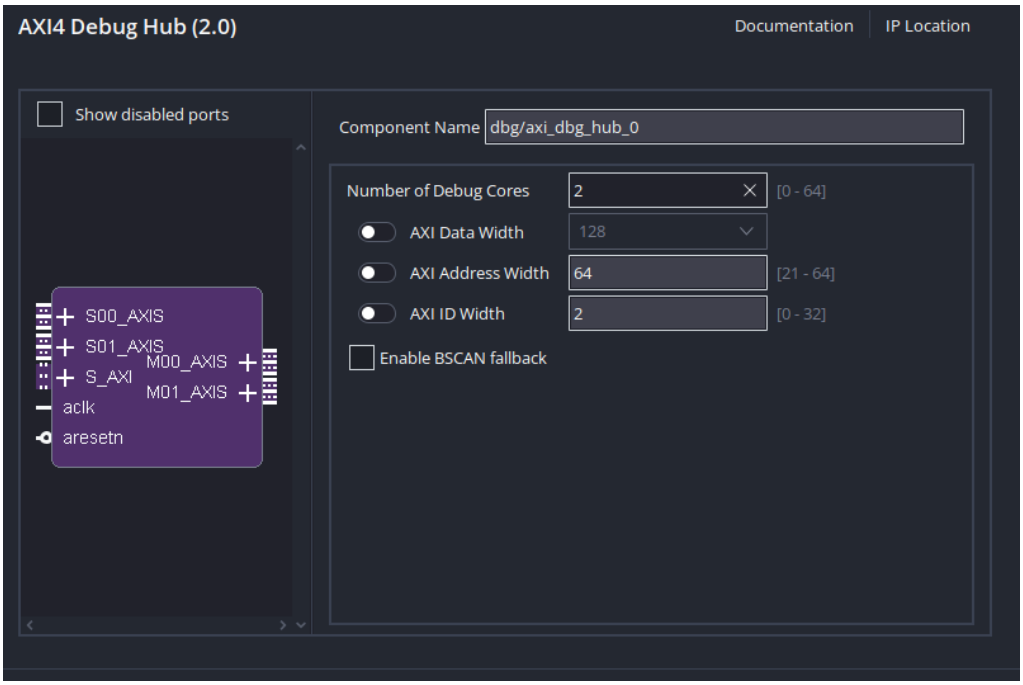

Figure 10 : Options

4) Open the customization window for the **ILA** block.

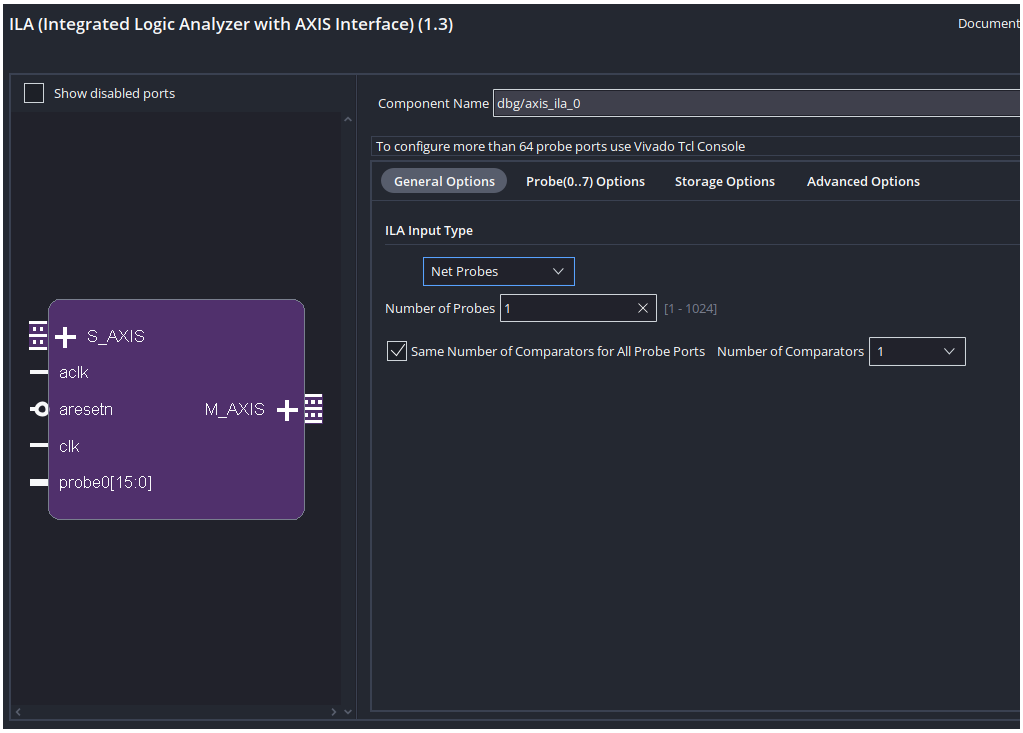

Figure 11 : General Options

5) Select **Probe(0..7) Options** tab.

6) Set **Probe Width** for **probe0** to 16.

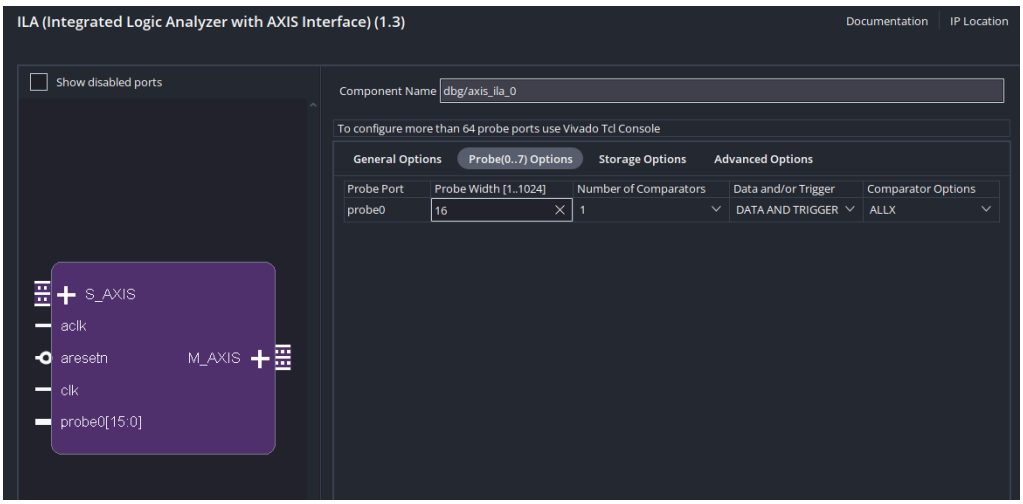

Figure 12 : Probe(0..7) Options

7) Select **Advanced Options** tab.

8) Check the **Enable AXI4-Stream Interfaces for Manual Connection to AXI4 Debug Hub** Checkbox.

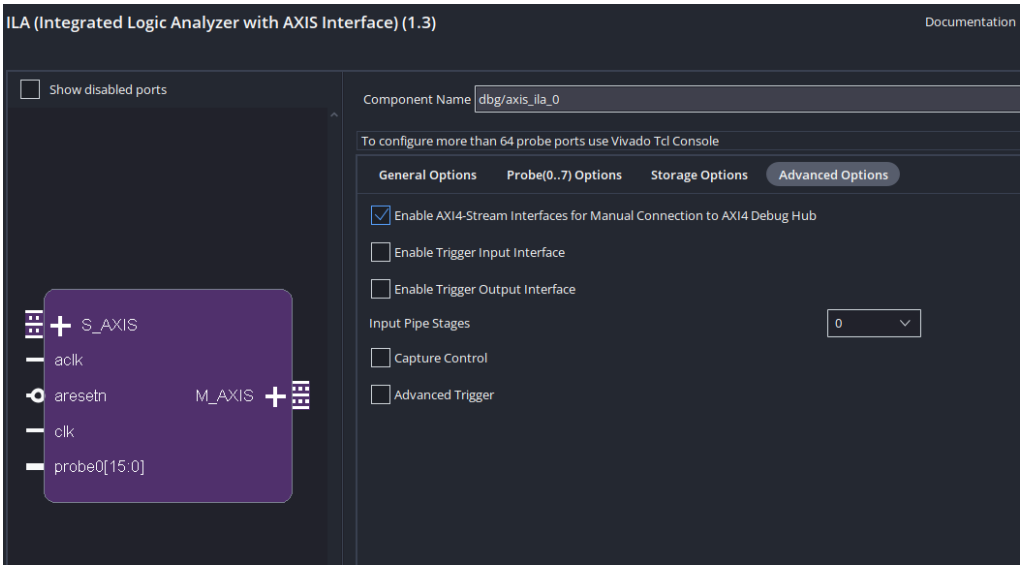

Figure 13 : Advanced Options

9) Open the customization window for the **VIO** block.

10) Check the **Enable AXI4-Stream Interfaces for Manual Connection to AXI4 Debug Hub** Checkbox.

11) Set **Input Probe Count** to 1. This input is used for binary counter input.

12) Set **Output Probe Count** to 4. These outputs are used for control of binary counter. is used for binary counter input.

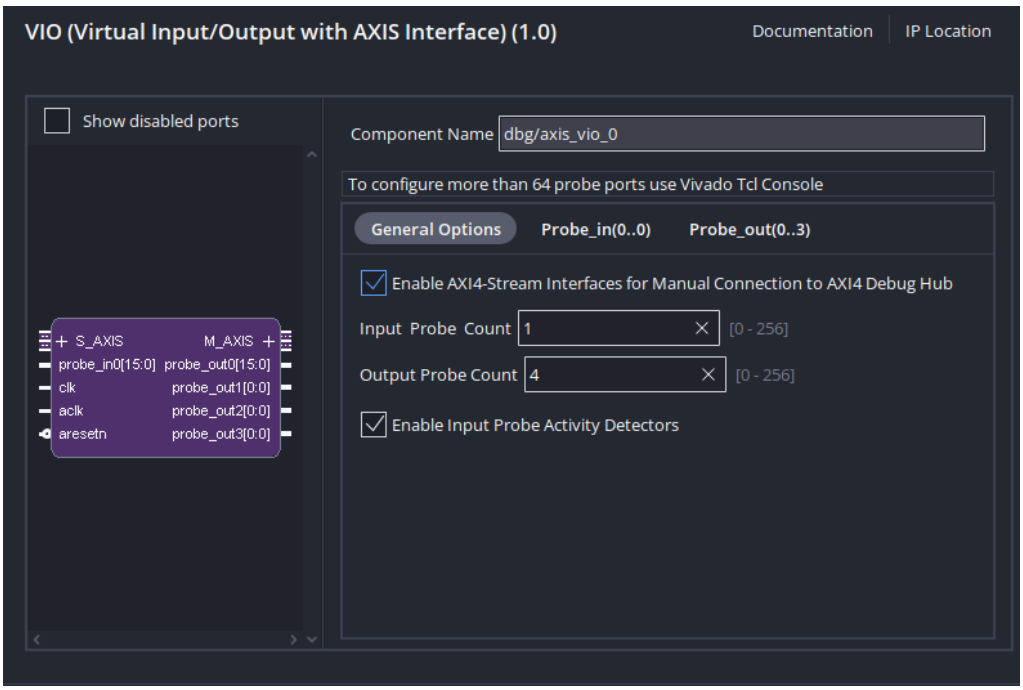

Figure 14 : General Options

13) Select **Probe_in(0..0) Options** tab.

14) Set **Probe Width** for **probe_in0** to 16.

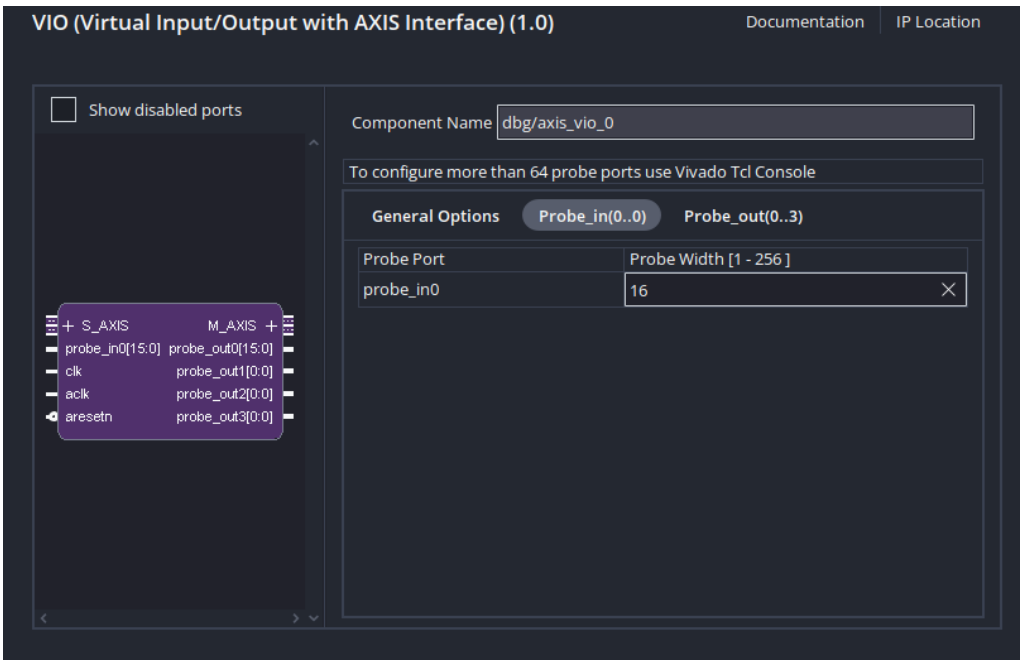

Figure 15 : Probe_in(0..0) Options

15) Select **Probe_out(0..3) Options** tab.

16) Set **Probe Width** for **probe_out0** to 16.

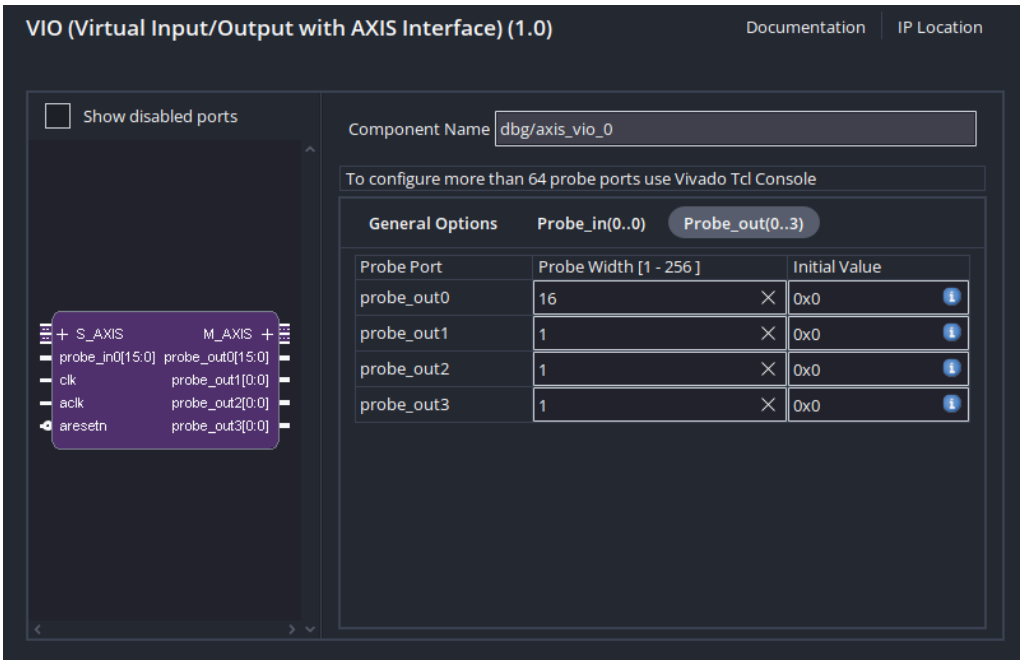

Figure 16 : Probe_out(0..3) Options

Tcl Example:

```
# Hierarchical cell: dbg
proc create_hier_cell_dbg { parentCell nameHier } {

  if { $parentCell eq "" || $nameHier eq "" } {
     catch {common::send_gid_msg -ssname BD::TCL -id 2092 -severity "ERROR" "creat
e_hier_cell_dbg() - Empty argument(s)!"}
     return
  }

  # Get object for parentCell
  set parentObj [get_bd_cells $parentCell]
  if { $parentObj == "" } {
     catch {common::send_gid_msg -ssname BD::TCL -id 2090 -severity "ERROR" "Unable
to find parent cell <$parentCell>!"}
     return
  }

  # Make sure parentObj is hier blk
  set parentType [get_property TYPE $parentObj]
  if { $parentType ne "hier" } {
     catch {common::send_gid_msg -ssname BD::TCL -id 2091 -severity "ERROR" "Parent
<$parentObj> has TYPE = <$parentType>. Expected to be <hier>."}
     return
  }

  # Save current instance; Restore later
  set oldCurInst [current_bd_instance .]

  # Set parent object as current
  current_bd_instance $parentObj

  # Create cell and set as current instance
  set hier_obj [create_bd_cell -type hier $nameHier]
  current_bd_instance $hier_obj
```

```
  # Create interface pins
  create_bd_intf_pin -mode Slave -vlnv xilinx.com:interface:aximm_rtl:1.0 S_AXI


  # Create pins
  create_bd_pin -dir I -type rst aresetn
  create_bd_pin -dir I -type clk aclk

  # Create instance: axi_dbg_hub_0, and set properties
  set axi_dbg_hub_0 [ create_bd_cell -type ip -vlnv xilinx.com:ip:axi_dbg_hub:2.0
axi_dbg_hub_0 ]
  set_property CONFIG.C_NUM_DEBUG_CORES {2} $axi_dbg_hub_0


  # Create instance: axis_ila_0, and set properties
 set axis_ila_0 [ create_bd_cell -type ip -vlnv xilinx.com:ip:axis_ila:1.3 axis_ila_0 ]
  set_property -dict [list \
    CONFIG.C_EN_AXIS_IF {1} \
    CONFIG.C_PROBE0_WIDTH {16} \
  ] $axis_ila_0


  # Create instance: axis_vio_0, and set properties
 set axis_vio_0 [ create_bd_cell -type ip -vlnv xilinx.com:ip:axis_vio:1.0 axis_vio_0 ]
  set_property -dict [list \
    CONFIG.C_EN_AXIS_IF {1} \
    CONFIG.C_NUM_PROBE_IN {1} \
    CONFIG.C_NUM_PROBE_OUT {4} \
    CONFIG.C_PROBE_IN0_WIDTH {16} \
    CONFIG.C_PROBE_OUT0_WIDTH {16} \
  ] $axis_vio_0


  # Create instance: c_counter_binary_0, and set properties
  set c_counter_binary_0 [ create_bd_cell -type ip -vlnv xilinx.com:ip:c_counter_b
inary:12.0 c_counter_binary_0 ]
  set_property -dict [list \
    CONFIG.CE {true} \
    CONFIG.Count_Mode {UPDOWN} \
    CONFIG.Load {true} \
  ] $c_counter_binary_0


  # Create interface connections
  connect_bd_intf_net -intf_net Conn1 [get_bd_intf_pins axi_dbg_hub_0/S_AXI] [get_
bd_intf_pins S_AXI]
  connect_bd_intf_net -intf_net axi_dbg_hub_0_M00_AXIS [get_bd_intf_pins axi_dbg_h
ub_0/M00_AXIS] [get_bd_intf_pins axis_ila_0/S_AXIS]
  connect_bd_intf_net -intf_net axi_dbg_hub_0_M01_AXIS [get_bd_intf_pins axis_vio_
0/S_AXIS] [get_bd_intf_pins axi_dbg_hub_0/M01_AXIS]
  connect_bd_intf_net -intf_net axis_ila_0_M_AXIS [get_bd_intf_pins axis_ila_0/M_A
XIS] [get_bd_intf_pins axi_dbg_hub_0/S00_AXIS]
  connect_bd_intf_net -intf_net axis_vio_0_M_AXIS [get_bd_intf_pins axi_dbg_hub_0/
S01_AXIS] [get_bd_intf_pins axis_vio_0/M_AXIS]

  # Create port connections
  connect_bd_net -net aclk_1  [get_bd_pins aclk] \
  [get_bd_pins axi_dbg_hub_0/aclk] \
  [get_bd_pins axis_ila_0/clk] \
```

```
  [get_bd_pins axis_ila_0/aclk] \
  [get_bd_pins axis_vio_0/clk] \
  [get_bd_pins axis_vio_0/aclk] \
  [get_bd_pins c_counter_binary_0/CLK]
  connect_bd_net -net aresetn_1  [get_bd_pins aresetn] \
  [get_bd_pins axi_dbg_hub_0/aresetn] \
  [get_bd_pins axis_ila_0/aresetn] \
  [get_bd_pins axis_vio_0/aresetn]
  connect_bd_net -net axis_vio_0_probe_out0  [get_bd_pins axis_vio_0/probe_out0] \
  [get_bd_pins c_counter_binary_0/L]
  connect_bd_net -net axis_vio_0_probe_out1  [get_bd_pins axis_vio_0/probe_out1] \
  [get_bd_pins c_counter_binary_0/CE]
  connect_bd_net -net axis_vio_0_probe_out2  [get_bd_pins axis_vio_0/probe_out2] \
  [get_bd_pins c_counter_binary_0/UP]
  connect_bd_net -net axis_vio_0_probe_out3  [get_bd_pins axis_vio_0/probe_out3] \
  [get_bd_pins c_counter_binary_0/LOAD]
  connect_bd_net -net probe0_1  [get_bd_pins c_counter_binary_0/Q] \
  [get_bd_pins axis_ila_0/probe0] \
  [get_bd_pins axis_vio_0/probe_in0]

  # Restore current instance
  current_bd_instance $oldCurInst
}
```

Configuring a Binary Counter

1) Add a **Binary Counter** block.

2) Open the customization window for the **Binary Counter** block.

3) Check the **Loadable** Checkbox.

4) In the **Count Mode** dropdown, select **UPDOWN**.

5) Select **Control** tab.

6) Check the **Clock Enable** Checkbox.

Connections Reference

See reference images for detail on Interface, Clocking and other connections for blocks.

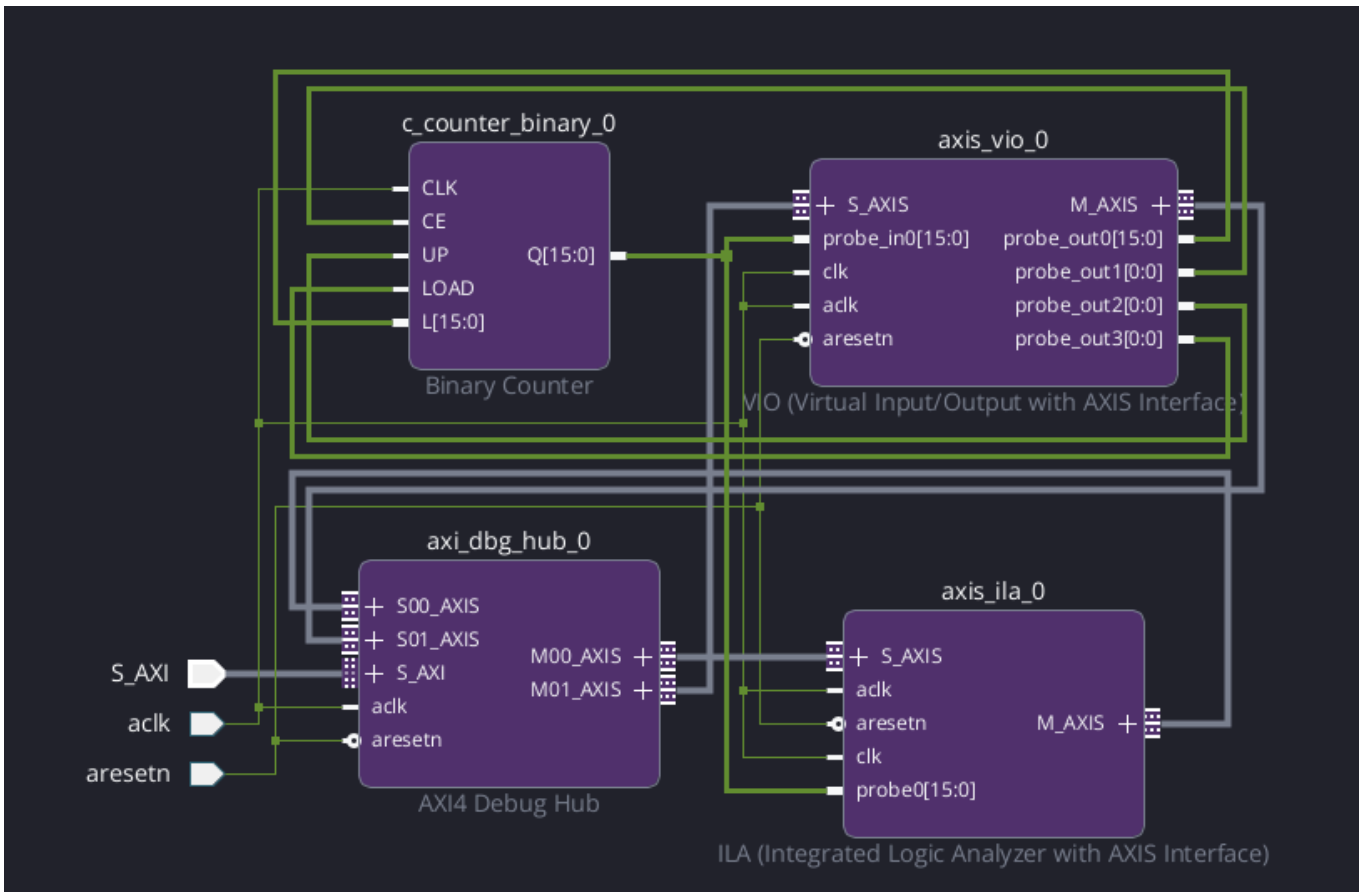

Figure 17 : Debug Block Diagram

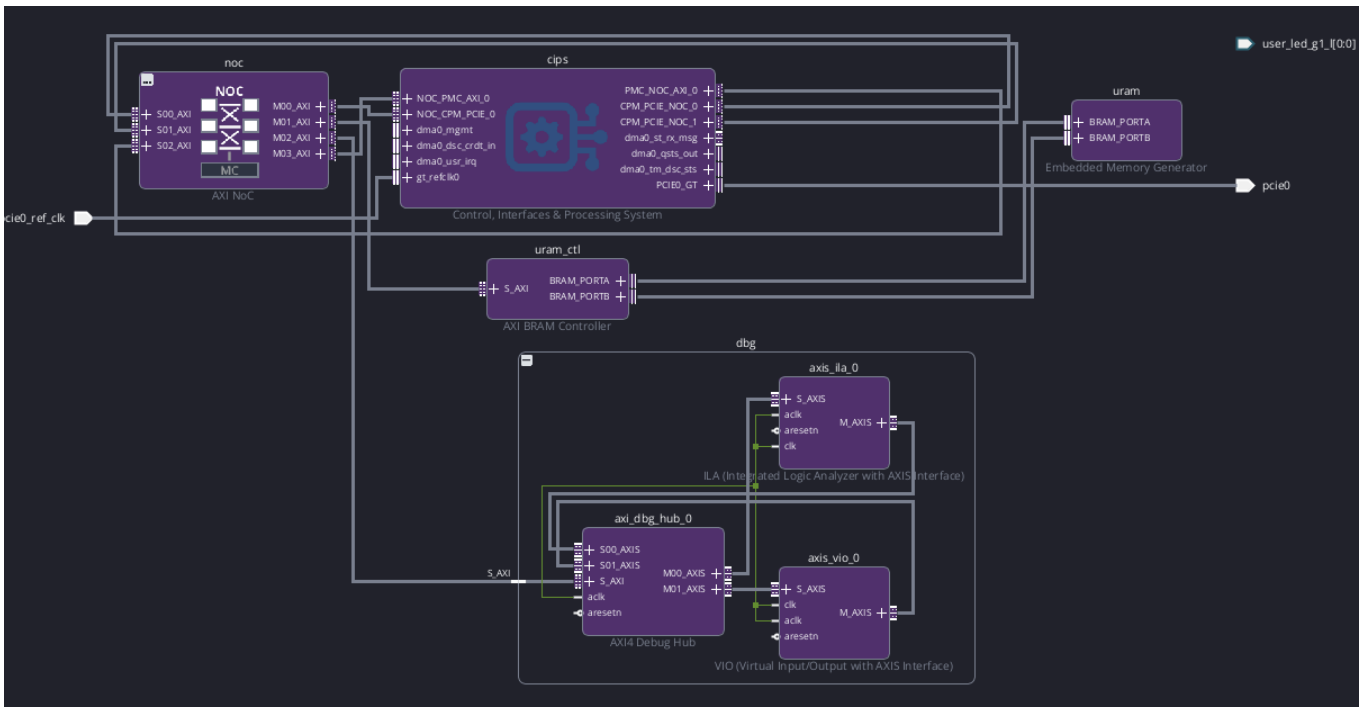

Figure 18 : Interfaces Diagram

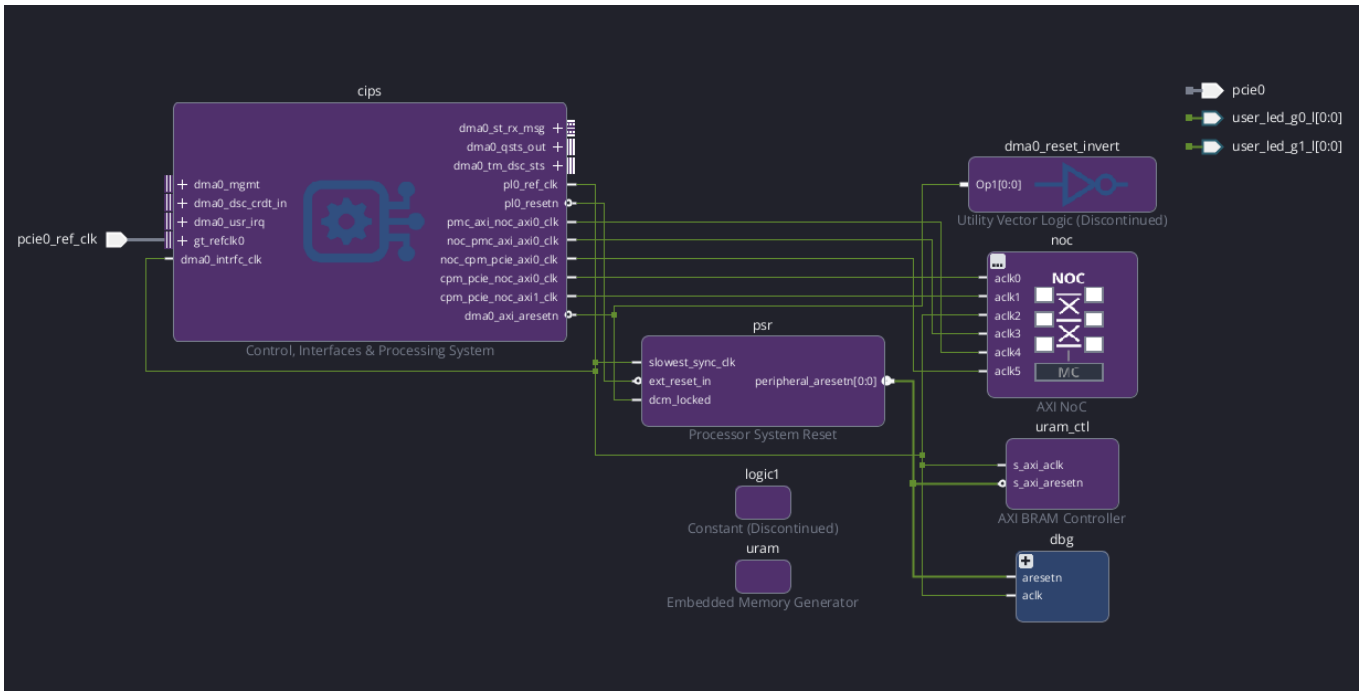

Figure 19 : Clocks Diagram

Address Editor Reference

See reference images for detail on addresses configured in Address Editor.

Diagram | NoC | Address Editor | Address Map

Assigned (17) | Unassigned (257) | Excluded (0) | Show All

Name	Interface	Slave Segment	Master Base Address	Range	Master High Address
Network 0 (/cips/CPM_PCIE_NOC_0, /cips/CPM_PCIE_NOC_1, /cips/PMC_NOC_AXI_0)					
/cips					
/cips/CPM_PCIE_NOC_0 (64 address bits : 0x00000000000 [16T])					
/cips/NOC_CPM_PCIE_0	NOC_CPM_PCIE_0	pspmc_0_psv_noc_pcie_0	0xE000_0000	256M	0xEFFF_FFFF
/cips/NOC_CPM_PCIE_0	NOC_CPM_PCIE_0	pspmc_0_psv_noc_pcie_1	0x6_0000_0000	8G	0x7_FFFF_FFFF
/cips/NOC_CPM_PCIE_0	NOC_CPM_PCIE_0	pspmc_0_psv_noc_pcie_2	0x80_0000_0000	256G	0xBF_FFFF_FFFF
/cips/NOC_PMC_AXI_0	NOC_PMC_AXI_0	pspmc_0_psv_cpm	0xFC00_0000	16M	0xFCFF_FFFF
/dbg/axi_dbg_hub_0/S_AXI_DBG_HUB	S_AXI	Mem0	0x201_4000_0000	2M	0x201_401F_FFFF
/uram_ctl/S_AXI	S_AXI	Mem0	0x201_0000_0000	2M	0x201_001F_FFFF
/cips/CPM_PCIE_NOC_1 (64 address bits : 0x00000000000 [16T])					
/cips/NOC_CPM_PCIE_0	NOC_CPM_PCIE_0	pspmc_0_psv_noc_pcie_0	0xE000_0000	256M	0xEFFF_FFFF
/cips/NOC_CPM_PCIE_0	NOC_CPM_PCIE_0	pspmc_0_psv_noc_pcie_1	0x6_0000_0000	8G	0x7_FFFF_FFFF
/cips/NOC_CPM_PCIE_0	NOC_CPM_PCIE_0	pspmc_0_psv_noc_pcie_2	0x80_0000_0000	256G	0xBF_FFFF_FFFF
/cips/NOC_PMC_AXI_0	NOC_PMC_AXI_0	pspmc_0_psv_cpm	0xFC00_0000	16M	0xFCFF_FFFF
/dbg/axi_dbg_hub_0/S_AXI_DBG_HUB	S_AXI	Mem0	0x201_4000_0000	2M	0x201_401F_FFFF
/uram_ctl/S_AXI	S_AXI	Mem0	0x201_0000_0000	2M	0x201_001F_FFFF
/cips/PMC_NOC_AXI_0 (64 address bits : 0x00000000000 [2G] ,0x000E0000000 [256M] ,0x00100000000 [128M] ,0x00600000000 [40G] ,0x04000000000 [1502G] ,0x200					
/cips/NOC_CPM_PCIE_0	NOC_CPM_PCIE_0	pspmc_0_psv_noc_pcie_0	0xE000_0000	256M	0xEFFF_FFFF
/cips/NOC_CPM_PCIE_0	NOC_CPM_PCIE_0	pspmc_0_psv_noc_pcie_1	0x6_0000_0000	8G	0x7_FFFF_FFFF
/cips/NOC_CPM_PCIE_0	NOC_CPM_PCIE_0	pspmc_0_psv_noc_pcie_2	0x80_0000_0000	256G	0xBF_FFFF_FFFF
/dbg/axi_dbg_hub_0/S_AXI_DBG_HUB	S_AXI	Mem0	0x201_4000_0000	2M	0x201_401F_FFFF
/uram_ctl/S_AXI	S_AXI	Mem0	0x201_0000_0000	2M	0x201_001F_FFFF

Figure 20 : Address Editor

Using HDSP over PCIe

1) On the Linux machine with Vivado Hardware Server and the Alpha Data PCIe device, download the **hsdp-pcie-driver**.

2) Apply the following patch to the **src/hsdp_pcie_user_config.h** file. It will make the following modifications:

* Fix the **class_create()** interface for newer Linux kernel versions. Delete this change if you are using an older Kernel that uses the old interface.
* Changes the driver configuration to use CPM5.
* Changes the PCIe Vendor ID to detect the ID used in the **cpm_demo** example design.
* Changes the HSDP User Mode AXI address for the AXI4 Debug Hub to match the address configured in Address Editor. This demonstrates how you might modify the driver further depending on the Address of the AXI4 Debug Hub in your design.

```
diff --git a/src/hsdp_pcie_driver_base.c b/src/hsdp_pcie_driver_base.c
index c91dd5e..abd13b7 115
--- a/src/hsdp_pcie_driver_base.c
+++ b/src/hsdp_pcie_driver_base.c
@@ -383,7 +383,8 @@ static int __init xil_hsdp_init(void)
     printk(KERN_INFO LOG_PREFIX "xil_hsdp_init
");

     // Register the character device class for the actual files
-    xrt_class = class_create(THIS_MODULE, "xil_hsdp_class");
+    xrt_class = class_create("xil_hsdp_class");
+
     if (IS_ERR(xrt_class)) {
         xil_hsdp_cleanup();
         return PTR_ERR(xrt_class);
diff --git a/src/hsdp_pcie_user_config.h b/src/hsdp_pcie_user_config.h
index f939f51..b568fac 115
--- a/src/hsdp_pcie_user_config.h
+++ b/src/hsdp_pcie_user_config.h
@@ -73,8 +73,8 @@ struct hsdp_pcie_config {
 //   - AI Core = VC1502-VC1902
 //   - AI Edge = VE1752
 //   - Prime   = VM1302-VM1802
-#define CPM4
-//#define CPM5
+//#define CPM4
+#define CPM5

 /*
  *  Modify the macros and structure below with PCIe customizations for the driver,
if desired.
@@ -84,8 +84,8 @@ struct hsdp_pcie_config {

 #define PCIE_DRIVER_NAME "xilinx_hsdp_pcie_driver"

-#define PCIE_VENDOR_ID  0x10EE
-#define PCIE_DEVICE_ID  0xB03F
+#define PCIE_VENDOR_ID  0x4144
+#define PCIE_DEVICE_ID 0x982

 static const struct hsdp_pcie_config user_configs[] = {
     //////////////////////////////////////////////////////
@@ -172,7 +172,7 @@ static const struct hsdp_pcie_config user_configs[] = {
             .hub_infos = {
                 {
                     // CED : PCIe BAR 4 address translates to 0x20100000000
-                    .axi_address = 0x20100000000,
+                    .axi_address = 0x20140000000,
                     .bar_index   = 4,
                     .bar_offset  = 0,
                     .size        = 0x200000
```

3) Build and install the driver:

```
sudo make install
depmod
```

4) When probed, the driver will generate device files corresponding to HSDP for Management Mode and User Mode.

5) To use Management Mode with Vivado Hardware Server:

```
hw_server -e "set dpc-pcie /dev/hsdp_mgmt_<BB:DD.F>"
```

6) To use Management Mode with Vivado Hardware Server:

```
hw_server -e "set pcie-debug-hub /dev/hsdp_user_<BB:DD.F>_<name>"
```

7) In Vivado on the host PC, open the target. Note that in User Mode, only Debug Bridge components are presented. The connection in VIO for **CE** may be used to enable the Binary Counter. The ILA can then be triggered to verify debug functionality.

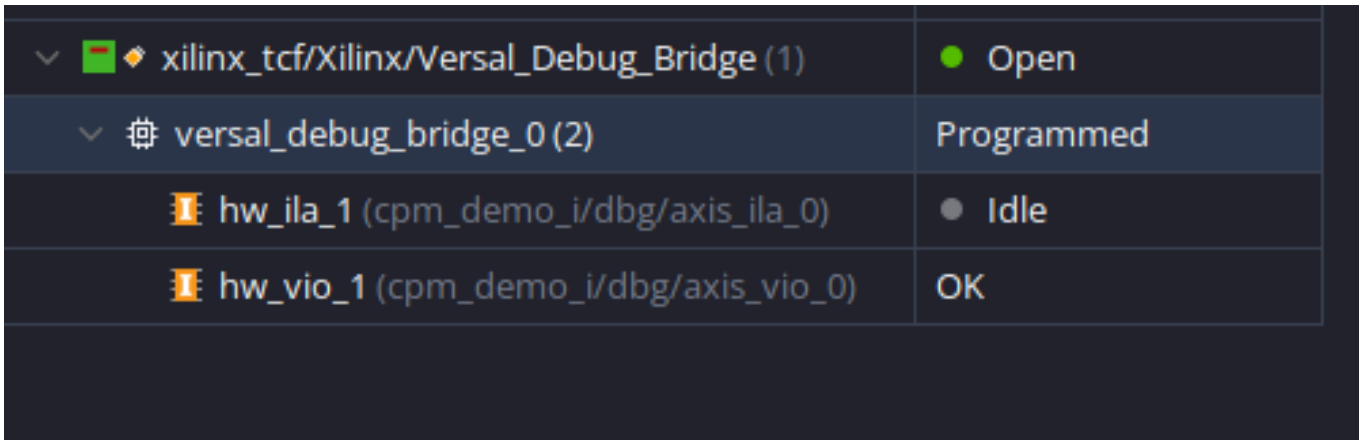

Figure 21 : Target in HSDP User Mode

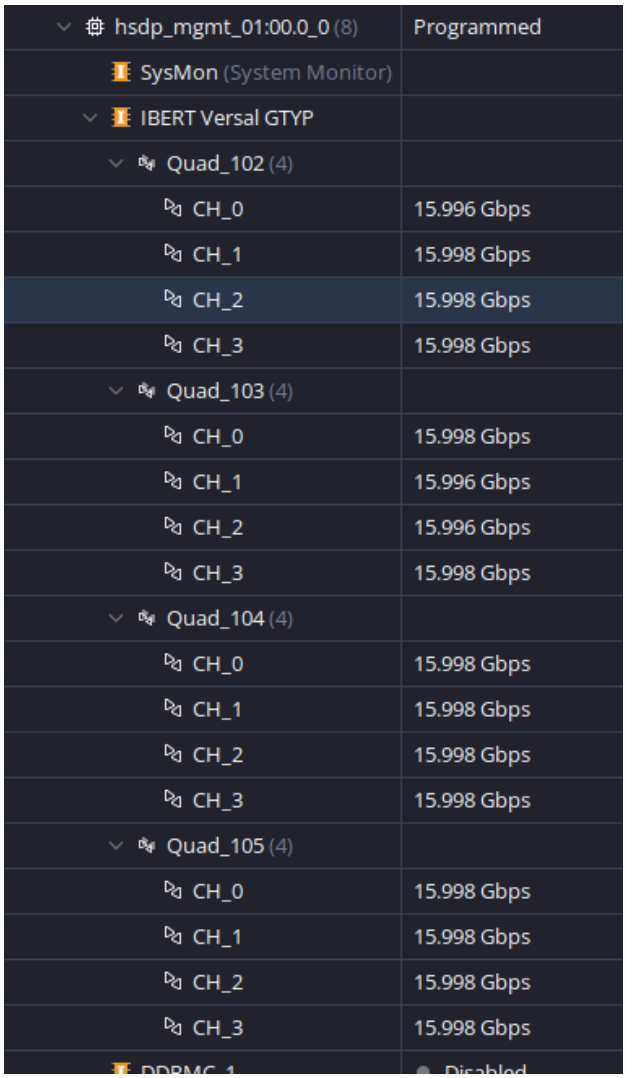

Figure 22 : Target in HSDP Management Mode

Revision History

Date	Document Revision	Nature of Change
02/02/2026	1.0	Initial release.

Address: 160 Dundee Street,
Edinburgh, EH11 1DQ, UK
Telephone: +44 131 558 2600
Email: sales@alpha-data.com
Website: http://www.alpha-data.com

Address: 10822 West Toller Drive, Suite 120,
Deer Creek Technology Center, Littleton, CO 80127
Telephone: (303) 954 8768
Email: sales@alpha-data.com
Website: http://www.alpha-data.com