Introduction

This document provides a detailed information on resources for debugging drivers pertaining to the Zynq UltraScale+™ MPSoC controller for the integrated block for PCI Express (PS-PCIe), DMA Subsystem for PCI Express (Bridge Mode) in Zynq UltraScale+ MPSoC (XDMA PL-PCIe) and AXI Bridge for PCI Express (AXI PCIe Gen2) in 7 Series devices.

To ease development of a PCIe system using Xilinx PCI Express IPs, Xilinx has created Wiki pages detailing the available reference designs, Device Tree and Drivers for Root Port configuration with PS-PCIe, XDMA PL-PCIe and AXI PCIe Gen2. This document provides links to relevant wiki pages in different sections.

Debugging designs related to PS-PCIe could be challenging especially because of the integrated nature of the IP, it doesn’t provide much visibility inside the IP in comparison to other base PCI Express IPs that Xilinx provides. In such scenarios, it would be necessary to probe different registers to troubleshoot a design. A list of such gotchas, registers, and debugging tips have been provided to help users to debug their designs.

There are three separate PCIe host drivers for the following IPs:

- Zynq UltraScale+ MPSoC controller for the integrated block for PCI Express (PS-PCIe)
- DMA Subsystem for PCI Express configured as Root Port in PL of Zynq UltraScale+ MPSoC (XDMA PL-PCIe)
- AXI Bridge for PCI Express (AXI PCIe Gen2) for Zynq-7000 devices.

The debugging approach for each IP should be considered differently. This document provides clear distinction between PS-PCIe, XDMA PL-PCIe and AXI PCIe Gen2.

PS-PCIe Controller Overview

UG1085 provides details on the integrated block for PCI Express v2.1 compliant AXI-PCIe Bridge and DMA modules (PS-PCIe) available in the Zynq UltraScale+ MPSoC. It supports Gen1/Gen2 rates at x1/x2/x4 link widths configured either as Endpoint or Root Port.

The AXI streaming and sideband signals between the AXI-PCIe Bridge and the integrated block for PCI Express are not directly accessible. Every PCIe transfer initiated in the AXI domain passes through the AXI-PCIe Bridge.
The PS-GTR transceivers in the processing system (PS) are used for serialization/deserialization (SerDes) purposes. The high-speed transceivers are used through the multiplexer switch and are shared with other blocks (such as DisplayPort, SATA, USB, and GEM) in the PS.

Figure 1 - PS PCI Express Controller Block Diagram

Figure 2 - Sharing of PS-GTR
PS-PCIe Controller Configuration

The Processing System Configuration Wizard (PCW) provides the PS-PCIe Controller with various configuration options. `psu_init.c` generated programs, clocks, GTs and PCIe attributes based on user-selection. The screenshots below shows different configuration options for the PS-PCIe Controller.

Figure 3 - Zynq UltraScale+ MPSoC PCW

Figure 4 - Enabling PCIe and Configuring Lane Width
PCI Express Drivers Overview

How to write PCIe Drivers

The link below provides details on Linux APIs for PCI device drivers.


Below are some of the topics discussed in the above link:
• Structure of PCI Drivers
• `pci_register_driver()` call
• How to find PCI devices manually
• Device Initialization Steps
• PCI device shutdown
• How to access the PCI config space

The PCI Express Advanced Error Reporting Driver Guide

This link below describes the basics of the PCI Express Advanced Error Reporting (AER) driver and provides information on how to use it, as well as how to enable the drivers of endpoint devices to conform with the PCI Express AER driver.


The MSI Driver Guide

This link below describes the basics of Message Signaled Interrupts (MSIs), the advantages of using MSI over traditional interrupt mechanisms, how to change your driver to use MSI or MSI-X, and some basic diagnostics to try if a device does not support MSIs.


Xilinx Linux Drivers

The link below gives details on the Xilinx drivers for Linux, such as testing, how to use the drivers, known issues, etc. The drivers included in the kernel tree are intended to run on the ARM (Zynq, Zynq UltraScale+ MPSoC) and MicroBlaze Linux.

All PCIe drivers for the following IPs are listed in the link:

• AXI PCIe Gen2 (Zynq)
• AXI PCIe Gen2 (MicroBlaze)
• ZynqMP Linux PS-PCIe Root Port
• ZynqMP Linux PL-PCIe Root Port

http://www.wiki.xilinx.com/Linux%20Drivers

Xilinx PCIe Drivers

The table below lists the drives that Xilinx provides for Xilinx PCI Express solutions. The commonly used drivers are `pci-xilinx-nwl.c` and `pcie-xdma-pl.c`. `pci-xilinx-nwl.c` is used in conjunction with Root Port configuration of PS-PCIe in Zynq UltraScale+ MPSoC devices. `pcie-xdma-pl.c` is used in conjunction with the Root Port DMA Subsystem for PCI Express (XDMA) in Bridge mode, typically implemented in Zynq UltraScale+ MPSoC devices.
### Component | Platform/IP Core | Link | In Mainline | Location
---|---|---|---|---
PCIe | axi_pcie (Zynq/MicroBlaze) | PCIe Root Port Driver | Yes | drivers/pci/host/pcie-xilinx.c
PS PCIe | Zynq UltraScale+ MPSoC | ZynqMP Linux PCIe Root Port Driver | Yes | drivers/pci/host/pci-xilinx-nwl.c
PL PCIe | Zynq UltraScale+ MPSoC | ZynqMP Linux PL PCIe Root Port | No | drivers/pci/host/pcie-xdma-pl.c

### PCIe Host Drivers

The following operations are carried by a host driver:

- Initialize bridge (Enabling ECAM, setting up required interrupt masks)
- Driver invokes relevant API in PCI subsystem
  - To Perform enumeration of entire PCIe hierarchy
  - Assign required resources for the Endpoint devices
- Route interrupt raised by Endpoint to respective end point driver handler

#### PCIe host controller driver for PS- PCIe Bridge


The ZynqMP PS-PCIe Root port driver is up streamed to main line. The Root Port driver supports following features:

- Supports MSI, Multi MSI, Legacy interrupts
- Supports non-prefetchable and prefetchable memory assignments

#### PCIe host controller driver for Xilinx XDMA PCIe Bridge Mode (PL-PCIe)


#### PCIe host controller driver for Xilinx AXI PCIe Gen2 Bridge


### PS-PCIe Driver Categories

The drivers for PS-PCIe can be classified into three categories.

#### Root Complex Drivers (RC) Drivers

This driver runs in Zynq UltraScale+ that has PS-PCIe configured as Root Port. The main functions of this driver are the system setup, enumeration, interrupt servicing etc.

#### Endpoint (EP) Drivers

This driver also runs in Zynq UltraScale+, similarly to RC drivers. It is used to communicate with the attached endpoint device for DMA control, Control and Status register reads, general reads and writes etc.
PSEP (Processing System Endpoint) Drivers

This driver runs in Zynq UltraScale+ that has PS-PCIe configured as an endpoint. It enables endpoint, configures registers, generate interrupt etc.

Figure 7 illustrates where each driver category resides.

Reference Designs

ZynqMP Linux PS-PCIe Root Port (ZCU102)

The link below provides an overview of Root Port driver for the controller for PCI Express which is available as part of the ZynqMP processing system. The illustration is based on the reference design for ZCU102. The link explains the endpoint driver configuration required for testing a Broadcom NIC endpoint with ZCU102 as the Rootport, the test procedure, and the mechanism to verify the test.

http://www.wiki.xilinx.com/ZynqMP+Linux+PCIe+Root+Port

Figure 8 - ZynqMP Linux PS-PCIe Root Port Design
ZynqMP Linux PL PCIe Root Port (ZCU106)

This link below gives an overview of the Root Port driver for Xilinx XDMA (Bridge mode) IP, when connected to a PCIe block in Zynq UltraScale+ MPSoC PL. The driver is available as part of the Xilinx Linux distribution as drivers/pci/host/pcie-xdma-pl.c.

The hardware setup uses the Xilinx ZCU106 hardware platform along with Root port FMC on a HPC FMC slot. The design uses an XDMA-bridge mode IP with PL-PCIe and targets GTs routed to HPC FMC.

http://www.wiki.xilinx.com/ZynqMP+Linux+PL+PCIe+Root+Port

AXI PCIe Root Port (ZC706)

http://www.wiki.xilinx.com/Linux+PCIe

The above link gives an overview of the AXI PCIe Root Complex driver for the Xilinx AXI PCIe Soft IP, which is available as part of the Zynq and MicroBlaze Linux distributions. The illustrated reference design is based on a ZC706 development board.

![ZC706 Root Port Design](image)

Figure 9 - ZC706 Root Port Design

ZCU102 BSP Post Mortem

The ZCU102 BSP (Board Support Package) is available for download from the Xilinx Download page. BSPs are reference designs for user to start working with and customize for their own projects. These are provided in the form of installable BSP files, and includes all necessary design and configuration files, pre-built and tested hardware and software images, ready to be downloaded on the board or to boot in the QEMU system emulation environment. See UG1156 for instructions on using the downloaded BSP.

The figure below shows the content of the BSP for the ZCU102 board.

- .petalinux
- components
- hardware
- pre-built
Note:

- Rename the extension of the downloaded file to *.zip and then unzip the file to see the content.

The pre-built->linux->images contains pre-built images. You just need to copy the image to your SD card to run it on the ZCU102 board.

The bit file is provided in: pre-built\linux\implementation directory.

The HDF file and device tree are available in components\plnx_workspace\device-tree\device-tree

The hardware Vivado project is available in: \xilinx-zcu102-v2018\xilinx-zcu102-2018.2\hardware\xilinx-zcu102-2018.2
Figure 10 shows the block diagram of the ZCU102 BSP hardware design.

The BSP reference design is built for Gen2x1 configuration. Users who need different configurations should re-generate the image by using the template flow as described in UG1144.

Figure 11 - ZCU102 BSP - PCIe Lane Width and Lane Speed Configuration
**Device Tree**

**Device Tree Blob**

http://www.wiki.xilinx.com/Build+Device+Tree+Blob

The link above provides a detailed explanation of the Device Tree. Some of the topics included are as follows:

- Device tree syntax example
- Device tree properties
- How to add a new driver to the DTG
- Device tree Generation
- Build Device Tree
- Creating a Device Tree Source (.dts/.dtsi) files (Vivado 2014.2 onwards)
- Generating an HDF file from the hardware project
- Generating Device Tree Source (.dts/.dtsi) files from SDK
- Compiling a Device Tree Blob (.dtb) file from the DTS

For further details on the Device Tree, see:

- http://www.wiki.xilinx.com/Build+Device+Tree+Compiler+%28dtc%29

**Xilinx PS PCIe Root Port Bridge DT description**

The following link provides a Device Tree description for PS-PCIe configured as Root Port in Bridge mode.

The following link provides a Device Tree description for XDMA configured as Root Port in Bridge mode:

Figure 14 - XDMA Root Port (Bridge mode) Device Tree description for MSI FIFO mode

```c
MSI DECODE mode:
  xdma_0: axi-pcie@00000000 {
    #address-cells = <3>;
    #interrupt-cells = <1>;
    #size-cells = <2>;
    compatible = "xilinx,xdma-host-3.0e";
    device_type = "pci";
    interrupt-map = <0 0 0 1 Apcie_intc_0 1>, <0 0 0 2 Apcie_intc_0 2>, <0 0 0 3 Apcie_intc_0 3>, <0 0 0 4 Apcie_intc_0 4>;
    interrupt-map-mask = <0 0 0 7>;
    interrupt-parent = <&gpio>;
    interrupt-names = "msic", "msi0", "msi1";
    interrupts = <0 80 40>, <0 90 40>, <0 91 40>;
    ranges = <0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x01000000>,
             <0x10000000 0x10000000 0x00000000 0x00000000 0x00000000 0x01000000>;
    reg = <0x00 0x40000000 0x10000000>;
    pcie_intc_0: interrupt-controller {
      #address-cells = <0>;
      #interrupt-cells = <1>;
      interrupt-controller ;
    };
  };
```

Figure 15 - XDMA Root Port (Bridge mode) Device Tree description for MSI Decode mode

Xilinx AXI PCIe Gen2 Root Port Bridge DT description

The following link provides a Device Tree description for AXI PCIe Gen2 configured as a Root Port in Bridge mode.


```c
Zynq:
  pci_express: axi-pcie@50000000 {
    #address-cells = <3>;
    #size-cells = <2>;
    #interrupt-cells = <1>;
    compatible = "xilinx,axi-pcie-host-1.00.e";
    reg = <0x50000000 0x10000000>;
    device_type = "pci";
    interrupts = <0x52 4>;
    interrupt-map = <0 0 0 7>;
    interrupt-map-mask = <0 0 0 7>,
    <0 0 0 3 Apcie_intc 1>,
    <0 0 0 2 Apcie_intc 2>,
    <0 0 0 3 Apcie_intc 3>,
    <0 0 0 4 Apcie_intc 4>;
    ranges = <0x00000000 0x00000000 0x00000000 0x10000000>;
    pcie_intc: interrupt-controller {
      interrupt-controller;
      #address-cells = <0>;
      #interrupt-cells = <1>;
    };
```
Figure 17 - AXI PCIe Gen2 (Bridge mode) Device Tree description for MicroBlaze

PS-PCIe Device Tree

The figure below shows the default Device Tree for a PS-PCIe Bridge.

- 02 in the first dword in the first line indicates non-prefetchable and 43 in the second line indicates prefetchable memory.
  - 0x2 indicates 32-bit address to kernel so the PCI subsystem in Linux assigns lower 32-bit
  - 0x43 indicates 64-bit address to kernel
- 2nd and 3rd words are PCI Express address
- 3rd and 4th words are for host Root Port (AXI) address
- The last two words indicate the size of the memory range.
- In the above example, the first one is 256MB and the second is 8GB.
- Both of these are constant and are defined in UG1085.
The BARs in the PCW for Root Port configuration do not have anything to do with the ranges in the device tree.

The two memory ranges shown above basically mean that the hardware system can support only those ranges and that Linux will need to allocate BAR memory requested by EPs within those ranges.

**Relationship between IP Integrator Address Editor, Address Translation IP Configuration GUI and Device Tree (AXI PCIe Gen2 Root Port Design)**

The figures below illustrate the relationship between the IP Integrator Address Editor, Address Translation parameter in the IP configuration GUI, and the addresses and ranges in the Device Tree file.

![Diagram](image)

**Figure 18 - AXI PCIe Gen2 Root Port Block Design – IP Integrator Address Editor**
The Device tree is generated in the petalinux-build step. Figure 21 and Figure 22 show the list of generated device tree files, for a Zynq PetaLinux Project and a ZynqMP PetaLinux Project respectively, after petalinux-build executes successfully.

File locations are as follows:

Zynq: `<petalinux_project_directory>/components/plnx_workspace/device-tree/device-tree-generation`
ZynqMP: `<petalinux_project_directory>/components/plnx_workspace/device-tree/device-tree`
### Figure 21 - Device Tree files in Zynq PetaLinux Project

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>device-tree.mss</td>
<td>MSS File</td>
<td>10 KB</td>
</tr>
<tr>
<td>hardware_description.dhf</td>
<td>HDF File</td>
<td>875 KB</td>
</tr>
<tr>
<td>psw.dtsi</td>
<td>DTSI File</td>
<td>3 KB</td>
</tr>
<tr>
<td>pl.dtsi</td>
<td>DTSI File</td>
<td>2 KB</td>
</tr>
<tr>
<td>ps7_init.c</td>
<td>C File</td>
<td>833 KB</td>
</tr>
<tr>
<td>ps7_init.h</td>
<td>H File</td>
<td>1,588 KB</td>
</tr>
<tr>
<td>ps7_init.html</td>
<td>Firefox HTML Doc...</td>
<td>52 KB</td>
</tr>
<tr>
<td>ps7_init.tcl</td>
<td>TCL File</td>
<td>817 KB</td>
</tr>
<tr>
<td>ps7_init_gpl.c</td>
<td>C File</td>
<td>852 KB</td>
</tr>
<tr>
<td>ps7_init_gpl.h</td>
<td>H File</td>
<td>1,558 KB</td>
</tr>
<tr>
<td>skeleton.dtsi</td>
<td>DTSI File</td>
<td>1 KB</td>
</tr>
<tr>
<td>system-conf.dtsi</td>
<td>DTSI File</td>
<td>29 KB</td>
</tr>
<tr>
<td>system-top.dtsi</td>
<td>DTSI File</td>
<td>5 KB</td>
</tr>
<tr>
<td>zynq-7000.dtsi</td>
<td>DTSI File</td>
<td></td>
</tr>
</tbody>
</table>

### Figure 22 - Device Tree files in ZynqMP PetaLinux Project

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>device-tree.mss</td>
<td>MSS File</td>
<td>10 KB</td>
</tr>
<tr>
<td>hardware_description.dhf</td>
<td>HDF File</td>
<td>875 KB</td>
</tr>
<tr>
<td>psw.dtsi</td>
<td>DTSI File</td>
<td>3 KB</td>
</tr>
<tr>
<td>pl.dtsi</td>
<td>DTSI File</td>
<td>2 KB</td>
</tr>
<tr>
<td>ps7_init.c</td>
<td>C File</td>
<td>833 KB</td>
</tr>
<tr>
<td>ps7_init.h</td>
<td>H File</td>
<td>1,588 KB</td>
</tr>
<tr>
<td>ps7_init.html</td>
<td>Firefox HTML Doc...</td>
<td>52 KB</td>
</tr>
<tr>
<td>ps7_init.tcl</td>
<td>TCL File</td>
<td>817 KB</td>
</tr>
<tr>
<td>ps7_init_gpl.c</td>
<td>C File</td>
<td>852 KB</td>
</tr>
<tr>
<td>ps7_init_gpl.h</td>
<td>H File</td>
<td>1,558 KB</td>
</tr>
<tr>
<td>skeleton.dtsi</td>
<td>DTSI File</td>
<td>1 KB</td>
</tr>
<tr>
<td>system-conf.dtsi</td>
<td>DTSI File</td>
<td>29 KB</td>
</tr>
<tr>
<td>system-top.dtsi</td>
<td>DTSI File</td>
<td>5 KB</td>
</tr>
<tr>
<td>zynq-7000.dtsi</td>
<td>DTSI File</td>
<td></td>
</tr>
</tbody>
</table>

### Release Notes – PS-PCIe / PL-PCIe Driver Issues

[(*Xilinx Answer 70702*)](#) contains known Issues and information related to the drivers for PS PCIe in Zynq UltraScale+ MPSoC. This also includes information on the PL Root Port Solution (Driver and IP usage) in relation to Zynq UltraScale+ MPSoC.
Below are the list of answer records included in [Xilinx Answer 70702]. Please check the answer record link above for the latest update.

<table>
<thead>
<tr>
<th>Answer Record</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Xilinx Answer 69066)</td>
<td>Zynq UltraScale+ MPSoC Controller for PCI Express (Vivado 2017.1) - Root Port Error: hwirq 0x4 is too large for dummy</td>
</tr>
<tr>
<td>(Xilinx Answer 70703)</td>
<td>Zynq UltraScale+ MPSoC (Vivado 2017.4) - Issue fixes in driver for DMA/Bridge Subsystem for PCIe in AXI Bridge mode (PL PCIe) configured as Root Port</td>
</tr>
<tr>
<td>(Xilinx Answer 69587)</td>
<td>Zynq UltraScale+ MPSoC: Linux hangs when accessing PL peripheral by Yocto (2017.1 - 2017.4) - ILA / HW Manager usage with core in PetaLinux requires bootarg</td>
</tr>
<tr>
<td>(Xilinx Answer 71106)</td>
<td>Zynq UltraScale+ MPSoC - PL PCIe Root Port Bridge (Vivado 2018.1) - MSI Interrupt handling causes downstream devices to time out</td>
</tr>
<tr>
<td>(Xilinx Answer 70854)</td>
<td>Zynq UltraScale+ MPSoC - DMA/Bridge Subsystem for PCI Express - PL Bridge Root Port - IP Setup tips for use with PL PCIe Root Port driver</td>
</tr>
<tr>
<td>(Xilinx Answer 65443)</td>
<td>DMA Subsystem for PCI Express - Release Notes and Known Issues for Vivado 2015.3 and newer tool versions</td>
</tr>
<tr>
<td>(Xilinx Answer 70706)</td>
<td>DMA/Bridge Subsystem for PCI Express (Bridge Mode/Root Port - Vivado 2017.4) - AXI transactions fail when no Endpoint is connected</td>
</tr>
<tr>
<td>(Xilinx Answer 71094)</td>
<td>Zynq UltraScale+ MPSoC - DMA/Bridge Subsystem for PCIe (AXI Bridge mode/Root Port - Vivado 2018.1) - When 64-bit address is set in AXIBAR2PCIEBAR, endpoint PCIe BAR not enumerated in correct locations</td>
</tr>
<tr>
<td>(Xilinx Answer 71095)</td>
<td>DMA / Bridge Subsystem for PCI Express (Bridge Mode - Vivado 2017.4) - AXIBAR and AXIBARHIGHADDR are set incorrectly in an IP Integrator design resulting in DECERR during 64-bit S_AXI access</td>
</tr>
<tr>
<td>(Xilinx Answer 71105)</td>
<td>DMA Subsystem for PCI Express (Vivado 2018.1) - MSI Interrupt FIFO can overflow in Root Port configuration in Bridge Mode</td>
</tr>
</tbody>
</table>

### Debugging PL-PCIe Issues

When setting up a Zynq UltraScale+ MPSoC system for PetaLinux with XDMA PL-PCIe (Bridge Mode) in Root Port configuration, there are a number of settings and options that should be used to experience seamless interoperability of the System, IP, and the PetaLinux Driver (pcie-xdma-pl). [Xilinx Answer 70854] provides details on the following topics.

- AXI BAR Translation Registers for correct DTS enumeration
- Endpoint Devices with Non-Prefetch BARs
- AXI-Lite Control Bus on AXI Bridge
- Root Port PCI Express BAR Disable
Debugging PS-PCIe Driver Issues

General Checks

The PCI Express Controller Programming Model section in UG1085 summarizes programming of the PCI Express controller for Endpoint and Root Port mode operations. Review that section to make sure programming of the PS-GT Transceiver Interface, IOU for Reset Pin, PCI Express Controller and Bridge initialization has been done correctly.

Other initial checks are as follows:

- Try with the latest version of PetaLinux.
- Make sure that the correct version of the BSP is being used for a particular board and silicon revision if the design being run is based on BSP and not on template flow.
- For specific user design, it is recommended to configure and build a PetaLinux project from the HDF file generated from that design.
- Confirm that the correct driver and DTS are being used
- Test with an off the shelf card (for example, NIC as an endpoint)

Boot log

Below is a snippet of a Linux boot log. If an endpoint is connected and if the link is established successfully, the log file reports ‘Link is UP’ as shown in the log below. Confirm that the link is UP before proceeding with further investigation.

```
[ 3.014299] nwl-pcie fd0e0000.pcie: Link is UP
[ 3.015606] PCI host bridge /amba/pcie@fd0e0000 ranges:
[ 3.023739] No bus range found for /amba/pcie@fd0e0000, using [bus 00-ff]
[ 3.030671] MEM 0xe1000000-0xe0ffffff -> 0xe1000000
[ 3.035854] nwl-pcie fd0e0000.pcie: PCI host bridge to bus 0000:00
[ 3.041905] pci_bus 0000:00:00:00: root bus resource [bus 00-ff]
[ 3.047361] pci_bus 0000:00:00:00: root bus resource [mem 0xe1000000-0xe0ffffff]
[ 3.054320] pci 0000:00:00.0: of_irq_parse_pci() failed with rc=-19
[ 3.060905] pci 0000:00:00.0: BAR 8: assigned [mem 0xe1000000-0xe100ffffff]
[ 3.067499] pci 0000:01:00.0: BAR 0: assigned [mem 0xe1000000-0xe100007f]
```

If no Endpoint is connected, link would be shown as DOWN instead of UP in the above log. The 'lspci' will just show the root port as illustrated below.

```
root@Xilinx-ZCU102-2015_4:~# lspci
00:00.0 PCI bridge: Xilinx Corporation Device d024
```

BARs are not assigned

If the BARs are not assigned, check if the device is advertising a valid class code. If an incorrect class code is provided when configuring Root Port mode in PCW, BAR assignment fails during Linux enumeration. The Ispici log shown below depicts an invalid class code being reported and as a result BARs are not assigned.
The class code is configured in the PCW as shown below. Make sure to turn on the Advanced mode.

Figure 23 - PS-PCIe Class Code

The class code can be read out from the following register:

Accessing BARs

Unlike x86 platforms where the BIOS enables Memory space (MemEn) in the device configuration space, embedded platforms do not enable MemEn bits during enumeration. An Endpoint driver with relevant PCIe API's is required for this.

To check BAR access via devmem, you should set MemEn via the “setpci” command as shown below. setpci is a utility for querying and configuring PCI devices.

```
setpci -s 01:00.1 COMMAND=0x7
```

**Endpoint not detected**

If the endpoint is not detected during lspci, run the following tests:

- Hold the endpoint in reset until Linux boots
- After Linux boots, bring the endpoint out of reset and then insert the Root Port driver as a module.
  - e.g. `insmod /lib/modules/4.9.0/kernel/drivers/pci/host/pcie-xilinx-nwl.ko`

Also, the following test can be performed:

- Bring the endpoint out of reset, wait until Linux boots and then insert the Root Port driver.

**Enable Endpoint Drivers**

Endpoint functionality can be checked with endpoint drivers. The respective endpoint driver must be enabled in the kernel build.

**Connecting with a Switch**

The Switch does not need any separate configuration/software, any PetaLinux revision should work. It is enumerated like any other PCIe device. There is a known issue with a multifunction device being used with all four legacy interrupts (See [Xilinx Answer 69066](#)).

The lspci log below shows a 6-port PLX switch with 2 endpoints connected.
Reading PS-PCle Register

devmem can be used to read and write to PS-PCle registers using the physical address. The command line is as follows:

```
devmem ADDRESS [w|b|h] -- read memory and print
devmem ADDRESS [w|b|h] VALUE -- write VALUE into ADDRESS
```

Note: w -> Word, b-> byte, h-> halfword

Examples:

```
root@xilinx-zcu102-2017_4:~# devmem 0xFD0E0000 0x00010007
root@xilinx-zcu102-2017_4:~# devmem 0xFD0E03E4 0x00000000
root@xilinx-zcu102-2017_4:~# devmem 0xFD480010 0x00000000
root@xilinx-zcu102-2017_4:~# devmem 0xFD480194 0x0000FFE2
root@xilinx-zcu102-2017_4:~# devmem 0xFD480304 0x00000000
root@xilinx-zcu102-2017_4:~# devmem 0xFD0E0000 0x00010007
```

Register Dump Comparison

When debugging a user design, it is helpful to compare register values with the example design for a development board such as the ZCU102. The command below can be used to read a range of registers:

```
xsc% mrd 0xFD480000 100
```
Figure 24 - Register Values Comparison between ZCU102 reference design and custom design

Registers to Read for Interrupt Issues

GICP3_IRQ_STATUS (LPD_SLCR) Register


<table>
<thead>
<tr>
<th>src24</th>
<th>24</th>
<th>wtc</th>
<th>0x0</th>
<th>FPD_APB_INT</th>
<th>OR'd of all APB interrupts from LPD</th>
</tr>
</thead>
<tbody>
<tr>
<td>src23</td>
<td>23</td>
<td>wtc</td>
<td>0x0</td>
<td>PCIe gen purpose interrupt</td>
<td></td>
</tr>
<tr>
<td>src22</td>
<td>22</td>
<td>wtc</td>
<td>0x0</td>
<td>PCIe misc (err etc) interrupts</td>
<td></td>
</tr>
<tr>
<td>src21</td>
<td>21</td>
<td>wtc</td>
<td>0x0</td>
<td>PCIe Bridge DMA interrupts</td>
<td></td>
</tr>
<tr>
<td>src20</td>
<td>20</td>
<td>wtc</td>
<td>0x0</td>
<td>PCIe legacy (INTA/BCD) interrupts</td>
<td></td>
</tr>
<tr>
<td>src19</td>
<td>19</td>
<td>wtc</td>
<td>0x0</td>
<td>PCIe_MSI[1]=PCIe interrupt for MSI vectors 63 to 32</td>
<td></td>
</tr>
<tr>
<td>src18</td>
<td>18</td>
<td>wtc</td>
<td>0x0</td>
<td>PCIe_MSI[0]=PCIe interrupt for MSI vectors 31 to 0</td>
<td></td>
</tr>
</tbody>
</table>

MSGF_LEG_STATUS (AXIPCIE_MAIN)

https://www.xilinx.com/html_docs/registers/ug1087/axipcie_main___msgf_leg_status.html#

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Type</th>
<th>Reset Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>msgf_leg_status_int</td>
<td>31:4</td>
<td>ro</td>
<td>0x0</td>
<td>msgf_leg_status_int is generated from received PCIe Assert_INTD and Deserpret_INTD messages. msgf_leg_status_int will clear only when all INTD interrupt sources have been serviced and their interrupt status registers cleared.</td>
</tr>
<tr>
<td>msgf_leg_status_intc</td>
<td>2</td>
<td>ro</td>
<td>0x0</td>
<td>msgf_leg_status_intc is generated from received PCIe Assert_INTC and Deserpret_INTC messages. msgf_leg_status_intc will clear only when all INTC interrupt sources have been serviced and their interrupt status registers cleared.</td>
</tr>
<tr>
<td>msgf_leg_status_intb</td>
<td>1</td>
<td>ro</td>
<td>0x0</td>
<td>msgf_leg_status_intb is generated from received PCIe Assert_INTB and Deserpret_INTB messages. msgf_leg_status_intb will clear only when all INTB interrupt sources have been serviced and their interrupt status registers cleared.</td>
</tr>
<tr>
<td>msgf_leg_status_inta</td>
<td>0</td>
<td>ro</td>
<td>0x0</td>
<td>msgf_leg_status_inta is generated from received PCIe Assert_INTA and Deserpret_INTA messages. msgf_leg_status_inta will clear only when all INTA interrupt sources have been serviced and their interrupt status registers cleared.</td>
</tr>
</tbody>
</table>
BRIDGE_CORE_CFG_PCIE_RX_MSG_FILTER

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Type</th>
<th>Reset Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cfg_defined_vem_msg_vem_id</td>
<td>31:16</td>
<td>rw</td>
<td>0x0</td>
<td>Vendor Defined Message Type 0/1 (Message Code 0x78 &amp; 0x79) Vendor ID used to perform Vendor ID-specific message filtering.</td>
</tr>
<tr>
<td>cfg_defined_vem_msg_vem_trn</td>
<td>13</td>
<td>rw</td>
<td>0x0</td>
<td>Selects how to apply Vendor ID-specific filtering.</td>
</tr>
<tr>
<td>cfg_defined_vem_msg_en</td>
<td>14</td>
<td>rw</td>
<td>0x0</td>
<td>When 1, received PCIe messages with msg_code[7:4] not matching 0x1, 0x2, 0x3, 0x5, or 0x7 (other message types) will be forwarded to the AXI Master Interface. When 0 such messages will be silently dropped and not forwarded.</td>
</tr>
<tr>
<td>cfg_enable MSC_msg_filter</td>
<td>13</td>
<td>rw</td>
<td>0x0</td>
<td>When 1, received PCIe messages with msg_code[7:4] not matching 0x1, 0x2, 0x3, 0x5, or 0x7 (other message types) will be forwarded to the AXI Master Interface. When 0 such messages will be silently dropped and not forwarded.</td>
</tr>
<tr>
<td>reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cfg_enable_vem_msg_fwd</td>
<td>12:8</td>
<td>ro</td>
<td>0x0</td>
<td>When 1, received PCIe messages with msg_code[7:4] not matching 0x1, 0x2, 0x3, 0x5, or 0x7 (other message types) will be forwarded to the AXI Master Interface. When 0 such messages will be silently dropped and not forwarded.</td>
</tr>
<tr>
<td>reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cfg_enable_vem_msg_fwd</td>
<td>7</td>
<td>rw</td>
<td>0x0</td>
<td>When 1, received PCIe messages with msg_code[7:4] not matching 0x1, 0x2, 0x3, 0x5, or 0x7 (other message types) will be forwarded to the AXI Master Interface. When 0 such messages will be silently dropped and not forwarded.</td>
</tr>
<tr>
<td>reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cfg_enable_st_msg_fwd</td>
<td>6</td>
<td>ro</td>
<td>0x0</td>
<td>When 1, received PCIe messages with msg_code[7:4] (such as Set Bit Power Limit) will be forwarded to the AXI Master Interface. When 0 such messages will be silently dropped and not forwarded.</td>
</tr>
<tr>
<td>reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cfg_enable_arr_msg_fwd</td>
<td>4</td>
<td>ro</td>
<td>0x0</td>
<td>When 1, received PCIe messages with msg_code[7:4] (such as ERR_IDR, ERR_MACRO, ERR_FATAL) will be forwarded to the AXI Master Interface. When 0 such messages will be silently dropped and not forwarded.</td>
</tr>
<tr>
<td>reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cfg_enable_scan_msg_fwd</td>
<td>3</td>
<td>rw</td>
<td>0x0</td>
<td>When 1, received PCIe messages with msg_code[7:4] (such as ASCEND_INT[0:7], DESCEND_INT[0:7]) will be forwarded to the AXI Master Interface. When 0 such messages will be silently dropped and not forwarded.</td>
</tr>
<tr>
<td>reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cfg_enable_pm_msg_fwd</td>
<td>2</td>
<td>rw</td>
<td>0x0</td>
<td>When 1, received PCIe messages with msg_code[7:4] (such as PMX_TURN_OFF, PMX_TURN_ON) will be forwarded to the AXI Master Interface. When 0 such messages will be silently dropped and not forwarded.</td>
</tr>
<tr>
<td>reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cfg_enable_vem_msg_fwd</td>
<td>1</td>
<td>rw</td>
<td>0x0</td>
<td>When 1, received PCIe messages with msg_code[7:4] (such as Set Bit Power Limit) will be forwarded to the AXI Master Interface. When 0 such messages will be silently dropped and not forwarded.</td>
</tr>
<tr>
<td>reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

MSI Issues

When the PS-PCIe is configured as an endpoint and MSI is not generated, confirm that the MSI structure exists by probing the following register.

https://www.xilinx.com/html_docs/registers/ug1087/pcie_attrib___attr_41.html#

<table>
<thead>
<tr>
<th>Register Name</th>
<th>ATTR_41</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative Address</td>
<td>0x000000A4</td>
</tr>
<tr>
<td>Absolute Address</td>
<td>0xFD4B00A4 (PCI_ATTR)</td>
</tr>
<tr>
<td>Width</td>
<td>32</td>
</tr>
<tr>
<td>Type</td>
<td>rw</td>
</tr>
<tr>
<td>Reset Value</td>
<td>0x00000160</td>
</tr>
<tr>
<td>Description</td>
<td>ATTR_41</td>
</tr>
</tbody>
</table>

This register should only be written to during reset of the PCIe block.

ATTR_41 (PCI_ATTR) Register Bit-Field Summary

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Type</th>
<th>Reset Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attr_msi_cap_per_vector_masking_capable</td>
<td>9</td>
<td>rw</td>
<td>0x0</td>
<td>MSI Per-Vector Masking Capable. The value is transferred to the MSI Control Register[8]. When set, adds Mask and Pending Dword to Cap structure; EP=0x0000; RP=0x0000</td>
</tr>
<tr>
<td>attr_msi_cap_off</td>
<td>8</td>
<td>ox</td>
<td>0x1</td>
<td>Indicates that the MSI structures exists. If this is FALSE, then the MSI structure cannot be accessed via either the link or the management port.; EP=0x0001; RP=0x0000</td>
</tr>
<tr>
<td>attr_msi_cap_nextbr</td>
<td>7:0</td>
<td>rw</td>
<td>0x60</td>
<td>MSI Capability's Next Capability Offset pointer to the next item in the capabilities list, or 0xh if this is the final capability.; EP=0x0060; RP=0x0000</td>
</tr>
</tbody>
</table>

PS-PCIe Device Status Register and PCIe Link Down

For the possible errors occurring in the system, check the following registers to find out the type of error. If the PCIe Link Down bit is set, check/confirm if the board layout is compliant to board layout guidelines w.r.t. GTR for MPSoC (See UG583).

cat /proc/iomem

The file provides a mapping of the system's memory. The address ranges define the memory registers used by different memory types.

```
root@xilinx-ecu102-2017:~# cat /proc/iomem
00000000-7fffffff : System RAM
00000000-00c2ffff : Kernel code
00000000-00daffff : Kernel data
e0000000-effffffff /amba/pceie/ffde0000
d9b00000-c0c00000 /amba/ahci8ffdc0000
d9e00000-00000000 : breg
ff020000-ff030000 : a9ou
ff0d0000-ff0f0000 : a9ed
ff180000-ff1d8000 : pcleg
ff18a0000-ff18d0000 : /amba/dpa0ff640000
ff2a0000-ff2a7000 : blend
ff400000-ff401000 : av_buf
ff4e0000-ff4e1000 : ann
ff400000-ff407000 : /amba/cma/ff640000
ff405000-ff40d000 : /amba/watchdog/ff4d0000
ff500000-ff50f0000 : /amba/cma/ff500000
ff600000-ff61f0000 : /amba/cma/ff615000
ff700000-ff720000 : /amba/cma/ff720000
ff750000-ff75d000 : /amba/cma/ff759000
ff760000-ff763000 : /amba/cma/ff760000
ff780000-ff787000 : /amba/cma/ff785000
ff7a0000-ff7a7000 : /amba/cma/ff7a0000
ff7c0000-ff7c7000 : /amba/cma/ff7c0000
ff7e0000-ff7f0000 : System RAM
```

cat /proc/interrupts

The /proc/interrupts file lists the number of interrupts per CPU per I/O device. The file lists the IRQ number, the number of times an interrupt is handled by each CPU core, the type of interrupt, and a list of drivers that are registered to receive that interrupt. In a running system, the corresponding counter for interrupts should increment.
The following command extracts PCI related messages in the log file:

```
grep -rEn -i 'pci' /var/log/messages
```

Rescan

To re enumerate devices in the system, run the following commands.

- `echo 1 > /sys/bus/pci/devices/0000:00:00.0/remove`
- `echo 1 > /sys/bus/pci/rescan`

© Copyright 2018 Xilinx
System Address Mapping and Address Translation

For egress translation, up to eight translation regions can be set up. Translation is done for AXI transaction destined for PCIe and not PCIe ECAM or any other internal bridge register access. For all egress translations, the AXI domain address must be limited to the following ranges:

- 256 MB region starting at 0xE000_0000
- 8 GB region starting at 0x6_0000_0000
- 256 GB region starting at 0x80_0000_0000

Only when AXI transactions target these ranges are they routed to the controller for PCIe for further translation by the bridge.

When a transaction fails to hit all translations, the transaction is forwarded without translation if the subtractive decode is enabled. This is controlled by:

- AXIPCIE_MAIN.I_ISUB_CONTROL register for ingress translations
  - In Root Port configuration, to allow all upstream transactions (memory read, write) to access the AXI interface without any translation, ingress subtractive decode must be enabled (AXIPCIE_MAIN.I_ISUB_CONTROL = 0x01).

I_ISUB_CONTROL (AXIPCIE_MAIN) Register

I_ISUB_CONTROL (AXIPCIE_MAIN) Register Description

<table>
<thead>
<tr>
<th>Register Name</th>
<th>I_ISUB_CONTROL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative Address</td>
<td>0x000003E8</td>
</tr>
<tr>
<td>Absolute Address</td>
<td>0xFD0E03E8 (AXIPCIE_MAIN)</td>
</tr>
<tr>
<td>Width</td>
<td>32</td>
</tr>
<tr>
<td>Type</td>
<td>mixed</td>
</tr>
<tr>
<td>Reset Value</td>
<td>0x00000000</td>
</tr>
<tr>
<td>Description</td>
<td>Ingress Subtractive Decode Translation - Control</td>
</tr>
</tbody>
</table>

I_ISUB_CONTROL (AXIPCIE_MAIN) Register Bit-Field Summary

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Type</th>
<th>Reset Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ingress_sub_enable</td>
<td>31:1</td>
<td>rw</td>
<td>0x0</td>
<td>Subtractive decode enable.</td>
</tr>
</tbody>
</table>
- AXIPCIE_MAIN.E_ESUB_CONTROL register for egress translations

**E_ESUB_CONTROL (AXIPCIE_MAIN) Register**

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Type</th>
<th>Reset Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td>31:1</td>
<td>ro</td>
<td>0x0</td>
<td></td>
</tr>
<tr>
<td>egress_sub_enable</td>
<td>0</td>
<td>rw</td>
<td>0x3</td>
<td>Egress Subtractive Decode Translation - Control</td>
</tr>
</tbody>
</table>

**PCIe Attributes**

To get a list of PCIe registers, search for PCIE_ATTRIB in (UG1087).
Negotiated Link Width


BAR0 and BAR1

https://www.xilinx.com/html_docs/registers/ug1087/pcie_attr__attr_8.html#
https://www.xilinx.com/html_docs/registers/ug1087/pcie_attr__attr_10.html#

cfg_trn_pending
PS High Address Access from PL Masters

The parameter `CONFIG.PSU__HIGH_ADDRESS__ENABLE` is used to expose PS high addresses to PL masters. When this parameter value is set to 1, the following addresses are exposed to PL masters in the IP Integrator address editor.

<table>
<thead>
<tr>
<th>Peripheral</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDR_HIGH</td>
<td>0x0800000000</td>
</tr>
<tr>
<td>PCIE_HIGH1</td>
<td>0x0600000000</td>
</tr>
<tr>
<td>PCIE_HIGH2</td>
<td>0x8000000000</td>
</tr>
</tbody>
</table>

The value of this parameter does not impact PS masters. PS masters can still access the above addresses even though this parameter value is set to 0.

When the parameter `CONFIG.PSU__HIGH_ADDRESS__ENABLE` is set to 0, the PL masters should not access this region. The work-around is to remove the higher address range from the device tree, as the PL masters should not access this region.

```plaintext
memory {
  device_type = "memory";
  reg = <0x0 0x0 0x0 0x7ff00000>, <0x00000008 0x00000000 0x0 0x80000000>;
}
```

The parameter can be set in the PCW as shown below:

![PS-PL Configuration](image)
Case Studies

Link Down: Endpoint is not detected (PS-PCIe)

- Ensure the Endpoint card has fit in properly on the PCIe slot of the Rootport.
- Check the link status by doing `devmem 0xfd480238` in the Linux prompt.
  - Value should be 0x3 if the link is up


- Read LTSSM state from 0xfd480228. 8 down to 3 gives the LTSSM coding. For LTSSM coding values, see (PG054).
- If the link is found to be up via reading PCIe Status register but the device is not seen in `lspci`:
  - Check if the endpoint card was programmed before Rootport booted
  - Do a rescan of the PCIe bus by using the following command on ZCU102 Linux prompt

```bash
$ echo 1 > /sys/bus/pci/devices/0000:01:00.0/remove
$ echo 1 > /sys/bus/pci/rescan
```

- If the rescan did not help:
  - Try with the latest FSBL.
  - Test on some other platform (x86) just to confirm the EP card has no issues.

- When EP is reset without resetting RP, the LTSSM on RP goes through either the Disabled or hot-reset states which are known to have some chances of link failures with errors.

© Copyright 2018 Xilinx
- Issue Rootport Reset (0xFD1A0100 bit 17 pcie_ctrl_reset only, this is like PERST)

```
root@xilinx-zcd102-2018.2:~ # dmesg 0xFD1A0100 32 0x2000D
root@xilinx-zcd102-2018.2:~ # dmesg 0xFD1A0100 0x00000000
root@xilinx-zcd102-2018.2:~ # dmesg 0xFD1A0100 32 0x00000000
root@xilinx-zcd102-2018.2:~ # dmesg 0xFD1A0100 0x00000000
```

- LSPCI does not show config space of EP (PS-PCIe)
  - If the link is up at boot time, but lspci shows all ff’s for EP config space, the GPIO driver might be modifying serdes configuration.
  - Disable CONFIG_GPIO_PCA953X in kernel.

PS-PCIe (Configured as Root Port) does not recognize incoming INTX interrupts

If the PS-PCIe, configured as Rootport, does not recognize incoming INTX interrupts when the protocol link analyzer shows interrupt on the PCIe link, check the following registers:

**BRIDGE_CORE_CFG_PCIE_RX_MSG_FILTER**


```
| Field Name                  | Bits | Type | Reset Value | Description
|-----------------------------|------|------|-------------|--------------|
| pcie_cfg_reset              | 19   | rw   | 0x1         | PCIe Configuration reset.
| pcie_bridge_reset           | 18   | rw   | 0x1         | PCIe Bridge reset.
| pcie_ctrl_reset             | 17   | rw   | 0x1         | PCIe Controller reset.
| dp_reset                    | 16   | rw   | 0x1         | DisplayPort Controller and DMA reset.
| swdt_reset                  | 15   | rw   | 0x1         | FPD_SWDT reset.
| s_axi_hpc_3_fpdp_reset      | 12   | rw   | 0x1         | S_AXI_HP3_FPD reset.
| s_axi_hpc_2_fpdp_reset      | 11   | rw   | 0x1         | S_AXI_HP2_FPD reset.
| s_axi_hp_1_fpdp_reset       | 10   | rw   | 0x1         | S_AXI_HPI_FPD reset.
| s_axi_hp_0_fpdp_reset       | 9    | rw   | 0x1         | S_AXI_HPI_FPD reset.
| s_axi_hpc_1_fpdp_reset      | 8    | rw   | 0x1         | S_AXI_HPC1_FPD reset.
| s_axi_hpc_0_fpdp_reset      | 7    | rw   | 0x1         | S_AXI_HP0_FPD reset.
| fpd_dma_reset               | 6    | rw   | 0x1         | FPD_DMA reset. (gdmc)
| gpu_ppl_reset               | 5    | rw   | 0x1         | GPU PPI reset.
| gpu_ppo_reset               | 4    | rw   | 0x1         | GPU PPO reset.
| gpuประเภ�헤지           | 3    | rw   | 0x1         | GPU and PPO reset.
| gt_reset                    | 2    | rw   | 0x1         | PS GTR reset.
| sata_reset                  | 1    | rw   | 0x1         | SATA Controller reset.
```

The above bit must be set to ‘1’ for INTx to be forwarded to the AXI Master Interface.

**nwl-pcie fd0e0000.pcie: Unsupported request Detected (PS-PCIe)**

If the following error is reported when trying to initiate DMA from a third party GPU, make sure to enable the bus mastering bit in the root port as well as endpoint (EP).

```
[3130.483591] nwl-pcie fd0e0000.pcie: Unsupported request Detected
```

© Copyright 2018 Xilinx
[3130.490863] nwl-pcie fd0e0000.pcie: Non-Fatal Error Detected

The Bus mastering bit can be set by using the following commands:

    setpci -s 00:00.0 COMMAND=0x4
    setpci -s 01:00.0 COMMAND=0x7
    setpci -s 01:00.1 COMMAND=0x7

Note: confirm the bus structure before executing the above command.

In the kernel EP driver flow, when EP driver invokes "pci_enable_device_mem" API, the bus mastering bit will be set. If there is no driver, it has to be done manually.

On an x86 machine this bit is set by the BIOS, on ARM system this needs to be enabled using the above API.

Miscellaneous Points

- ZynqMP PL-PCIe as Root is supported through the XDMA-bridge based root driver along with the 2017.3 IP release. AXI-PCIe Root Bridge IP has been deprecated from 2017.3 onwards.
- Bar assignment happens from Linux. There is no control from the Driver. The driver only controls ranges property via DTS.
- When using PS-PCIe on Zynq UltraScale+ in Endpoint mode, FSBL should be able to program the PS/PS-PCIe and GTR within the 100ms requirement. FSBL is enough to set up the block for Endpoint mode operation. However, this does not include PL-bitstream programming as including that would make this >100ms.
- PERST# MIO is driven from psu_init.c
- MPSoC Boot Flow:
  - MPSoC power on, pmu-fw
  - FSBL
    - As part of FSBL, GTR and PS-PCIe attributes are programmed
      - GTR PLL locking is waited upon, ensuring that the clock is stable
    - The Root port is held under reset throughout
    - Once programming is done, PERST# pulse is asserted and RP brought out of reset so as to start link training
      - Actual PCIe hierarchy discovery happens only when Linux boots and RP driver is inserted (and there is ATF, u-boot before that)
    
      Note: For more information on MPSoC Boot Flow, refer to (UG1137).

- In both Endpoint and Root Port modes, the integrated block for PCIe advertises infinite completions; finite completions are not supported.

Filing an SR or posting in Forums

After having gone through this document, if the issue is still unresolved, submit an SR to Xilinx Technical Support or post the issue in Xilinx Community Forum for PCI Express. Attach the following along with the detail description of the issue.
• “lspci –vvv –xxx” dump
• cat /proc/interrupts dump
• Boot log
• Relevant PCIe Attributes Status
• Design/Hardware details
• Vivado Tool version
• What works and what does not?
• Things that have been checked so far and the corresponding results
• Has the Endpoint in use been tested working along with its driver on any other root port? If yes, what is the platform/OS/Linux etc.?
• Is it an out of the box BSP/RP design or is it a custom design?

Appendix

PetaLinux

PetaLinux Design Flow Overview

<table>
<thead>
<tr>
<th>Design Flow Step</th>
<th>Tool / Workflow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Platform Creation</td>
<td>Vivado</td>
</tr>
<tr>
<td>Create PetaLinux Project</td>
<td>petalinux-create -t project</td>
</tr>
<tr>
<td>Initialize PetaLinux Project</td>
<td>petalinux-config -i get-hw-description</td>
</tr>
<tr>
<td>Configure System-Level Options</td>
<td>petalinux-config</td>
</tr>
<tr>
<td>Create User Components</td>
<td>petalinux-create -t COMPONENT</td>
</tr>
<tr>
<td>Configure the Linux Kernel</td>
<td>petalinux-config -c kernel</td>
</tr>
<tr>
<td>Configure the Root Filesystem</td>
<td>petalinux-config -c rootsfs</td>
</tr>
<tr>
<td>Build the System</td>
<td>petalinux-build</td>
</tr>
<tr>
<td>Deploy the System</td>
<td>petalinux-package</td>
</tr>
<tr>
<td>Test the System</td>
<td>petalinux-boot</td>
</tr>
</tbody>
</table>

PetaLinux Project Structure

A PetaLinux project has the following basic directory structure.
References

- (UG1085)  
- (UG1087)  
- PetaLinux Documentation  
  o  [https://www.xilinx.com/support/documentation-navigation/development-tools/software-development/petalinux-tools.html?resultsTablePreSelect=documenttype:SeeAll#documentation](https://www.xilinx.com/support/documentation-navigation/development-tools/software-development/petalinux-tools.html?resultsTablePreSelect=documenttype:SeeAll#documentation)

Revision History

07/30/2018 - Initial release