TI81XX PCI Express Root Complex Driver User Guide

TI81XX PCI Express Root Complex Driver User Guide

Hemant Pedanekar

Introduction

This document is applicable to DM816x/AM389x and DM814x/AM387x family of devices referred hereafter as TI816X and TI814X respectively.

Descriptions common across both the device families use the term TI81XX/ti81xx.

TI81XX devices have PCI Express hardware module which can either be configured to act as a Root Complex or a PCIe Endpoint. This document caters to the Root Complex mode of operation and describes the Driver needed to configure and operate on TI81XX PCI Express device as Root Complex.

NOTE1: Support for TI814X Root Complex is added from 04.01.00.06 release, hence this document is not applicable for prior TI814X releases.

NOTE2: Various code snippets now use term `ti81xx` when referring to code common for TI81XX devices. For releases prior to 04.00.00.12 (TI816X) please consider the code snippets as having `ti816x` prefix, otherwise, refer the PDF of this user guide from respective release package.

Scope

This document covers following areas:

- Brief background
- Terminologies and conventions used
- Topology
- Features
- Linux PCI Subsystem
- Kernel Configuration to include RC Driver
- System Resources used on TI81XX Linux kernel
- Setup and configurations for using PCI Express Endpoints (with example)

Background

PCI Express (PCIe) is Third Generation I/O Interconnect, targeting low pin-count. It shares the concepts with earlier PCI and PCI-X and offers backwards compatibility for existing PCI software with following differences:

- PCIe is a point-to-point interconnect
- Serial link between devices
- Packet based communication
- Need PCIe switch to have connection between more than two PCIe devices

Terminology and Conventions

Following terminologies and conventions are used in this document:

PCIe Fabric

A topology comprised of various PCI Express nodes, also referred as devices. A device in the fabric can be either Root Complex, Endpoint, PCIe-PCI/PCI-X Bridge or a Switch.

Host

The entity comprising of one (or more) Central Processing Unit(s) (CPU) and resources, such as Memory (RAM) that can be shared across multiple PCIe nodes connected through a Root Complex.

Root Complex (RC)

Root of the PCIe topology. Used to connect Host to the PCIe I/O system. In this document, we consider Root Complex module to be integrated with Host so as to form a single entity and refer as "PCI Express Root Complex" device. Contains TYPE 1 configuration header. Has capability to generate TYPE 0 configuration transactions to configure devices and TYPE 1 configuration transactions for configuring bridges.

Endpoint

A PCI Express device/function with Type0 configuration header which either generates or terminates a PCIe transaction. This excludes PCIe-PCI bridges. Also referred as PCI Target

Switch

A PCI Express device with single upstream port and multiple downstream ports each of which can in turn have a PCIe Endpoint, Switch or PCIe-PCI/PCI-X bridge connected.

PCIe-PCI/PCI-X Bridge

Device with single upstream port interfacing a PCI Bus Segment to the PCIe topology.

Lane

A pair of Tx and Rx lines between two directly connected PCIe nodes.

Link

Interconnect between two point-to-point nodes. This can be a collection of multiple lanes. Each link can have 1, 2, 4, 8, 16 or 32 lanes, denoted as x1, x2 and so on.

Port

Represents a PCI Express link and consists of PCIe protocol layer implementation. Each PCIe node should have at least one PCI Express Port.

Root Port

A root complex port is called Root Port. A complete PCI Express hierarchy is spawned from a root port. An RC can have more than one root ports having distinct hierarchy domain each.

Downstream

Any element of the fabric which is relatively farther away from RC is treated as 'Downstream'. All PCIe Switch ports which are away from RC (providing connection to nodes far away from RC) are called Downstream Ports. Also RC

ports are Downstream Ports. A Downstream Flow is the communication moving away from the RC.

Upstream

Any element of the fabric which is relatively closer towards RC is treated as 'Upstream'. All PCIe Endpoint ports (including termination points for bridges) and Switch ports, which are closer to RC are called Upstream Ports on that device. A Upstream Flow is the communication moving towards RC.

PCI Enumeration Mapping

Since PCI Express is point to point topology, to maintain compatibility with legacy PCI Bus - Device notion used for Software Enumeration, we introduce following concepts which allow identifying various nodes and their internals (e.g., PCIe Switches) in terms of PCI devices/functions:

- Host Bridge: A bridge, integrated into RC to have PCI compatible connection to Host. The PCI side of this bridge is Bus #0 always. This means, the device on this bus will be the host itself.
- Virtual PCI-PCI Bridge: Each PCI Express port which is part of RC or a Switch is treated as a virtual PCI-PCI bridge. This means each port has a primary and secondary PCI bus and the downstream is mapped into the remote configuration space.
  - Root port associated virtual bridge has Bus #0 on the primary side with secondary bus on the downstream.
  - Each PCIe Switch is viewed as collection of as many virtual PCI-PCI bridges as number of downstream ports, connected to a virtual PCI bus which is actually secondary bus of another PCI-PCI bridge forming the upstream port of the switch.
  - The upstream port of each EP can either be part of the secondary bus segment of virtual PCI-PCI bridge representing downstream port of a switch or of the root port.

Topology

This section shows a typical PCI Express fabric. To set up tone of rest of the document, we also consider an example scenario involving TI816X PCI Express Root Complex. Please note that the figures shown below rely on complete understanding of the terminologies and conventions described in earlier section.

Note: The only major difference between TI816X and TI814X PCIe module is TI816X has a x2 link while TI814X has x1 link. Since this doesn't lead to any differences in topology and software execution impact, all of the descriptions considering TI816X as example in rest of the document apply equally to TI814X as well, unless otherwise stated.
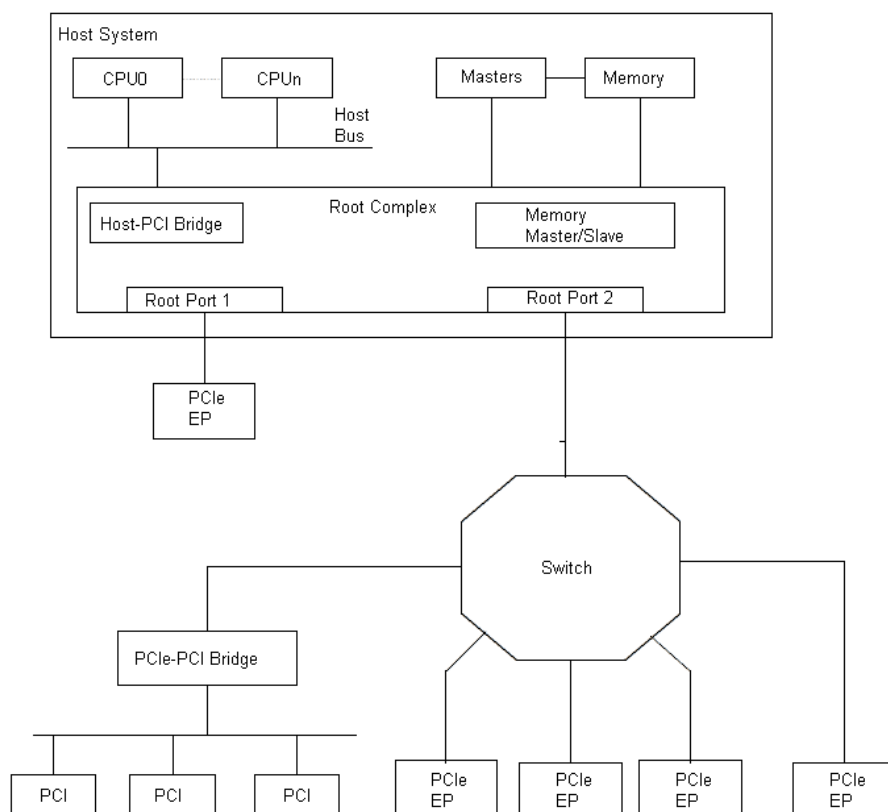
For the rest of this document, we will consider following PCIe System with TI816X as PCIe Root.
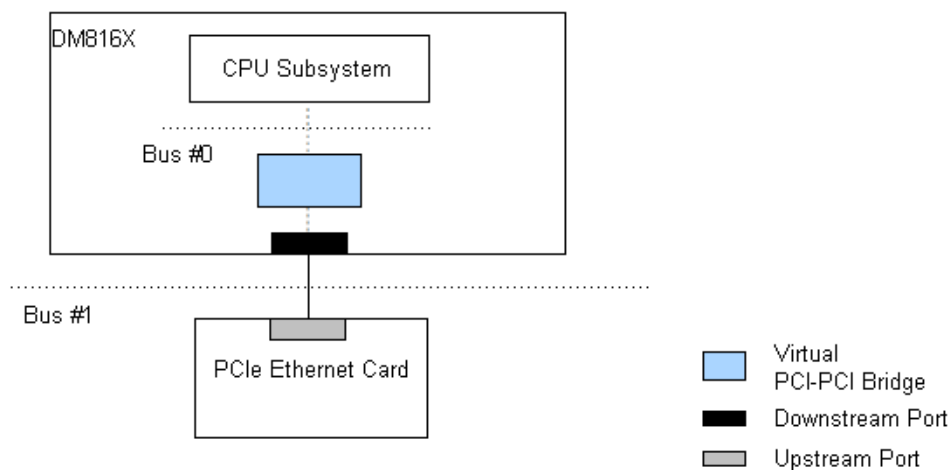
Features

Listed below are the various features supported by TI81XX as a PCI Express Root Complex.

- Express Base Specification Revision 2.0 compliant
- TI816X: Gen1 and Gen2 operation with up to x2 link supporting 10 GT/s raw transfer rate in single direction
- TI814X: Gen1 and Gen2 operation with x1 link supporting 5 GT/s raw transfer rate in single direction
- Maximum outbound payload size of 128 bytes
- Maximum inbound payload size of 256 bytes
- Maximum remote read request size of 256 bytes
- Support for receiving MSI and Legacy Interrupts (INTx)
- Configurable BAR filtering, I/O filtering, configuration filtering and Completion lookup/timeout
- PCI Express Link Power Management states except L2 state

The PCIe Root Driver facilitates following:

- Fits into Linux PCI Bus framework to provide PCI compatible software enumeration support
- In addition, provides interface to Endpoint Drivers to access the respective devices detected downstream.
- The same interface can be used by the PCI Express Port Bus Driver framework in Linux to perform AER, ASP etc handling
- Interrupt handling facility for EP drivers either as MSI or Legacy Interrupts (INTx).
- Access to EP I/O BARs through generic I/O accessors in Linux.
- Seamless handling of PCIe errors

Note-1: Out of the above, I/O access, Port Bus Driver integration are currently untested/incomplete.

Note-2: Since TI81XX is a 32-bit host architecture, 64-bit PCIe addresses may not be directly supported and requires customization to fit into Linux framework, hence not supported currently.

Note-3: TI81XX PCIe hardware does not support hot plug and if an EP directly connected to the TI81XX RC goes down (e.g., powered down or disabled), the complete PCIe h/w initialization needs to be repeated and PCI enumeration re-triggered. This is NOT SUPPORTED by the RC driver and will require code modification to handle such cases.

Note-4: MSI support is available since 04.00.00.12 release onwards.

Taking care of PERSTn

Note: This section including sub-sections do not apply to TI814X.

On DM8168 EVMs the SW5 DIP switch has a switch for "PCIe RST". This corresponds to the in/out mode of the PERSTn line of the PCIe slot (acting as PWRGD) which in turn is tied to PCIe_PORz. The switch position 'OFF' (or '0', towards the PCIe slot and farther to R195) means the pin is set as INPUT while switch position 'ON' means the pin is in OUTPUT mode. The default position is OFF (INPUT).

Note: As per the issue SDOCM00077550 (refer Release Notes), setting this switch as ON (OUTPUT) causes failure to detect the EP connected in the EVM slot since the reset line remains asserted (low). So it is recommended to keep this switch in OFF (INPUT) state.

```
SW5 ---> PCIe RST = OFF ('0')
```

Note: If you intend to work with this switch set as OUTPUT (ON) for controlling reset of EP, then EVM modification is required to allow PCIe RST pin to be toggled using I2C I/O Expander (address 0x20, P14 port). The default of this pin will be reset de-asserted (high) on power-up. The modification is as follows:

```
Replace resistor R218 (near the PCIe slot towards the edge of the board) with a resistor with value lower than 2K (even a 0 Ohm resistor is ok).
```

Note that the above modification will allow the PWRGD line (connected to A11 of PCIe slot) be toggled using I2C writes to on board IOEXPANDER.

Following are the recommended operations to reset EP device connected to PCIe slot on EVM:

1. **Power Cycle EVM:** This will reset the EP device connected in PCIe slot.
2. **Pressing RESET switch (SW6) on RC EVM:** Note that this will not by default toggle the PERSTn line and hence EP will not be reset. Refer "Controlling PERSTn Using I2C Writes" section below.
3. **Doing "reboot" from Linux on RC:** Note that this will not by default toggle the PERSTn line and hence EP will not be reset. Refer "Controlling PERSTn Using I2C Writes" section below.

Controlling PERSTn Using I2C Writes

Note that this description assumes the above mentioned EVM modification is already done.

As mentioned earlier, the PERSTn line on the A11 pin of PCIe slot on the EVM can be toggled by controlling I2C0 writes to IOEXPANDER port 14. This corresponds to performing i2c writes to address 0x20 bit 12.

- Setting this bit to 1 makes PERSTn pin high (EP reset de-asserted).
- Setting this bit to 0 means PERSTn pin low (reset asserted) and EP will remain in reset.

As an example, following commands at U-Boot prompt will toggle reset to EP during boot:

```
TI8168_EVM# i2c mw 0x20 ff ef   <--- Set bit 12 to 0 to assert PERSTn (default is 1, de-asserted)
TI8168_EVM# i2c mw 0x20 ff ff   <--- Set bit 12 to 1 to de-assert PERSTn and get EP out of reset
```

The above sequence can be set as part of U-Boot's bootcmd as shown below to ensure it is executed on every reset and before booting the kernel (assuming kernel is flashed in NAND @0x280000 offset):

```
TI8168_EVM# setenv bootcmd 'i2c mw 0x20 ff ef; i2c mw 0x20 ff ff; nand read.i 81000000 280000 500000; bootm 81000000'
TI8168_EVM# saveenv
```

This sequence can also be added to Kernel board file to perform similar i2c writes at boot up.

Linux PCI Subsystem

In Linux, the PCI implementation can roughly be divided into following main components:

PCI BIOS

- Architecture specific Linux implementation to kick off PCI bus initialization. It interfaces with PCI Host Controller code as well as the PCI Core to perform bus enumeration and allocation of resources such as memory, interrupts.
- The successful completion of BIOS execution assures that all the PCI devices in system are assigned parts of available PCI resources and their respective drivers (referred as Slave Drivers) can take control of them using the facilities provided by PCI Core.
- Optionally skips resource allocation (if they were assigned before Linux was booted, for example PC scenario).

Host Controller (RC) Module

- Handles hardware (SoC + Board) specific initialization and configuration
- Invokes PCI BIOS.
- Should provide callback functions for BIOS as well as PCI Core, which will be called during PCI system initialization and accessing PCI bus for configuration cycles.
- Provide resources information for available memory/IO space, INTx interrupt lines, MSI.
- Should facilitate IO space access (as supported) through in _x_ () out _x_ ()
- May need to provide indirect memory access (if supported by h/w) through read _x_ () write _x_ ()

Core

- Creates and initializes the data structure tree for Bus, Devices as well as Bridges in the system. Handles Bus/Device numberings.
- Creates device entries and proc/sysfs information
- Provides services for BIOS and Slave Drivers
- Hot plug support (optional/as supported by h/w).

Target (EP) Driver Interface

- Query and initialize corresponding devices found during enumeration.
- Provide MSI interrupt handling framework

PCI Express Port Bus Support

- Provides Hot-Plug support (if supported)
- Advanced Error Reporting support
- Power Management Event support
- Virtual Channel support to run on PCI Express Ports (if supported)

RC Driver Source Files

The driver files are present at following path relative to extracted kernel source directory for TI81XX.

```
arch/arm/mach-omap2/pcie-ti81xx.c (RC driver source)
arch/arm/mach-omap2/pcie-ti81xx.h (RC driver private header file)
arch/arm/mach-omap2/include/mach/pci.h (Exported macros for PCI subsystem)
```

Kernel Configuration

The TI816X/TI814X EVM kernel configurations enable Root Complex support by default. To set default configuration, execute following command from TI816X kernel source directory on Linux Build Host (you may need to set CROSS_COMPILE with the tool chain/path as appropriate):

- For TI816X

```
$ make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm ti8168_evm_defconfig
```

- For TI814X

```
$ make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm ti8148_evm_defconfig
```

Further, you can view/modify the support by entering into interactive configuration as:

```
$ make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm menuconfig
```

From on screen menu, navigate by pressing ENTER or SPACE key on entries marked with '--->' and selecting appropriate configurations marked with '[*]' by pressing Y key as shown below. Note that a '*' inside the box '[ ]' indicates the corresponding option is enabled and pressing SPACE key again will disable corresponding option (empty box).

```
    General setup  --->
[*] Enable loadable module support  --->
-*- Enable the block layer  --->
    System Type  --->
    '''Bus support  --->'''
```

```
    ...
    ...
```

After entering inside "Bus Support":

```
'''[*] PCI support'''
'''[*] Message Signaled Interrupts (MSI and MSI-X)'''
[ ] PCI Debugging
< > PCI Stub driver
[ ] PCI IOV support
...
```

Note 1: PCIe bus support cannot be built as module.

Note 2: You can disable MSI support by pressing 'N' for "Message Signaled Interrupts (MSI and MSI-X)" option above (default configuration has MSI support enabled).

In case you are facing any issues with PCI/e system initialization/configuration, enable debugging:

```
[*] PCI support
[*] Message Signaled Interrupts (MSI and MSI-X)
'''[*] PCI Debugging'''
< > PCI Stub driver
[ ] PCI IOV support
...
```

Note: Make sure to append "debug" to the bootargs to see the verbose messages during boot (you may also want to turn on low level debugging and append "earlyprintk" to kernel boot arguments).

System Resources

The RC Driver reserves following resources:

Outbound Memory

This memory window (address space) is used for memory transactions over PCIe initiated from the Root Complex.

- 0x20000000-0x2FFFFFFF: Assigned as 256MB Non-Prefetch memory
- Note that the address and size of the above window is fixed since it corresponds to PCIe slave port on the h/w

I/O Window

Since ARM architecture does not have I/O bus for I/O access, we reserve a memory window to perform PCIe I/O to EPs supporting I/O.

- 0x40000000-0x402FFFFF: Used as 3MB I/O window
- Note that the selection of the I/O space is done such that it does not conflict with the normal memory mapped space to allow software to distinguish between PCIe I/O and normal memory mapped access.

Inbound Memory

This memory is used to map TI81XX RAM so as to enable external masters (EP) to have access.

- Generally this should be the complete RAM available to kernel. Currently whole 2GB DDR space starting from 0x80000000 is provided for inbound memory access.

You can change this by updating ti81xx_pcie_resources data in arch/arm/mach-omap2/devices.c for "pcie-inbound0" resource:

```
        {
                /* Inbound memory window */
```

```
            .name           = "pcie-inbound0",
            .start          = PHYS_OFFSET,
            .end            = PHYS_OFFSET + SZ_2G - 1,
            .flags          = IORESOURCE_MEM,
      },
```

E.g., to enable access to 1GB memory, replace SZ_2G with SZ_1G above.

CAUTION: When changing inbound window/size, ensure that it covers valid RAM range as seen by the kernel else the EP devices may not be able to do DMA.

Interrupt Lines

TI816X uses interrupt line 48 for legacy interrupts and 49 for MSI interrupts. Interrupt multiplexing is handled by Root Complex Driver with Linux PCI Framework to allow multiple EPs sharing same interrupt lines.

Using PCIe Endpoint

This section describes setup and configuration for using PCIe Endpoint in the PCI system. We will consider an example with Broadcom NetXtreme BCM5751 based Gigabit Ethernet Controller card as a PCIe Endpoint.

Note that, though we consider a single EP, most of the description covered below is applicable to any system involving TI81XX with PCIe Switch and/or PCIe-PCI bridge devices having multiple PCIe Endpoints and/or PCI Target Devices.

IMPORTANT NOTE: Since TI81XX RC supports maximum remote read request size (MRRQS) as 256 bytes, ensure that the EP driver/device you are using doesn't set read request size more than this value. If it does, then modify the driver to set read request size to 256 bytes before building it. Ensuring Maximum Read Request size within 256 Byte limit is required even for any intermediate Switch/Bridge devices in the fabric.

- **TIP 1:** You can search for `pcie_set_readrq` calls inside driver file(s) and ensure that the second argument doesn't exceed 256. In case of setting for Switch/Bridge devices, there may not be any driver running on the RC for the same, so you will need to use PCI Config Write to Device Control Register (offset 0x78) of the respective devices to limit MRRQS. This can be done as part of a 'quirk' to call pcie_set_readrq() on PCI VENDOR and DEVICE ID match for that particular bridge/switch device added in file drivers/pci/quirks.c

- **TIP 2:** In case you don't have a driver for the EP device or do not desire to edit code, you can use the 'setpci' utility from the pciutils package (assuming it is installed on your filesystem) to change the MRRQS value on the respective EP device(s) as below.

If the PCI Express Capabilities structure on the EP device @01:00.0 is at offset 0x70, use the command as below to set the MRRQS field in Device Control register (bits 14:12 at offset 0x8 from PCIe Capabilities structure):

```
# setpci -v -s 01:00.0 78.W=0x1000:0x7000
0000:01:00.0 @78 3810->(1000:7000)->1810
```

In addition, ensure the following:

1. Since PCI I/O is not supported currently, you will need to ensure the use of memory mapped I/O for data transfer between RC and EP. Since PCI I/O is optional as per specification, most of the EP drivers would either use memory mapped I/O by default or support setting configuration to use memory mapped I/O instead of PCI I/O through use of driver/module parameter. Check respective EP driver manual for specific details about using memory mapped I/O.

2. RC driver prior to release 04.00.00.12 didn't have full MSI handling support and required a patch to enable the same. If you face any issues with data transfer when using older releases, check if the default interrupt mode used for EP is MSI based interrupts. If so, try changing the interrupt mode to Legacy IRQ based. Note that if there is

no driver/module parameter then you might need to modify the EP driver for this **or modify the TI816X kernel configuration to disable MSI support and rebuild the kernel.**

Ensuring PCIe System Initialization

Before proceeding to using PCIe Endpoint in the system, we must ensure that the PCIe RC is up and has detected and configured all PCIe/PCI targets present in the PCIe fabric. Steps mentioned below should be carried out before actually accessing the targets as described in subsequent subsections.

- Check that the desired PCIe card is interested in on-board PCIe Slot.
- Boot-up the TI816X PCIe RC. Refer TI81XX PSP User Guide for details about booting the kernel.
- Ensure that the PCI Enumeration is successful and the concerned EP is detected and configured. You can use either of the methods in subsequent points.
- Use 'lspci' command at TI816X Linux prompt to see all PCI devices in the system

```
TI8168_EVM# lspci -v
00:00.0 PCI bridge: Texas Instruments Device 8888 (rev 01) (prog-if 00 [Normal decode])
     Flags: bus master, fast devsel, latency 0
     Memory at <ignored> (32-bit, non-prefetchable)
     Memory at <ignored> (32-bit, prefetchable)
     Bus: primary=00, secondary=01, subordinate=01, sec-latency=0
     Memory behind bridge: 20000000-200fffff
     Capabilities: [40] Power Management version 3
     Capabilities: [50] MSI: Enable- Count=1/1 Maskable- 64bit+
     Capabilities: [70] Express Root Port (Slot-), MSI 00
     Capabilities: [100] Advanced Error Reporting
01:00.0 Ethernet controller: Broadcom Corporation NetXtreme BCM5751 Gigabit Ethernet PCI Express
     Subsystem: Broadcom Corporation NetXtreme BCM5751 Gigabit Ethernet PCI Express
     Flags: fast devsel, IRQ 48
     Memory at 20000000 (64-bit, non-prefetchable) [disabled] [size=64K]
     Capabilities: [48] Power Management version 2
     Capabilities: [50] Vital Product Data
     Capabilities: [58] MSI: Enable- Count=1/8 Maskable- 64bit+
     Capabilities: [d0] Express Endpoint, MSI 00
     Capabilities: [100] Advanced Error Reporting
     Capabilities: [13c] Virtual Channel
```

- The above output shows following devices are in the system

```
00:00.0 => TI816X PCIe RC Host
01:00.0 => Broadcom Corporation NetXtreme BCM5751 Gigabit Ethernet PCI Express Endpoint
```

- Notice that TI816X Host is itself detected as one of the PCI devices. This is expected.
- If 'lspci' utility is not available, then /sys/bus/pci/devices/ directory can be examined to get raw information about PCI devices found, such as vendor ID, device ID etc.

Selecting the EP Driver

- Once it is ensured that the expected device(s) are detected fine, enable the driver for the EP in kernel configuration
- If you don't readily know about which driver to be used, you can get the Vendor ID and device ID values for the device as follows:

```
TI8168_EVM# cat /sys/bus/pci/devices/0000\:01\:00.0/vendor
0x14e4
TI8168_EVM# cat /sys/bus/pci/devices/0000\:01\:00.0/device
0x1677
```

- The above values can be used to search in kernel 'drivers' directory. In our case, we know that it is a network device, so we use following to search the device ID in TI816X kernel source tree on Linux Build Host:

```
$ grep -nrs 1677 include/linux/pci_ids.h
2071:#define PCI_DEVICE_ID_TIGON3_5751   0x1677
```

- The above output indicates that the device with ID 0x1677 is identified with macro PCI_DEVICE_ID_TIGON3_5751
- Now use this macro to search inside 'drivers/net' directory of TI816X kernel source

```
$ grep -nrs PCI_DEVICE_ID_TIGON3_5751 drivers/net/*
drivers/net/tg3.c:203:  {PCI_DEVICE(PCI_VENDOR_ID_BROADCOM, PCI_DEVICE_ID_TIGON3_5751)},
...
```

- As mentioned at the beginning of this section, opening tg3.c shows this driver sets R-RREQ as 4096, so we need to edit all occurrences of `pcie_set_readrq` to pass 256 instead of 4096 and save the file.
- Now we need to find the kernel configuration option needed to enable the driver to which tg3.c file belongs and also the name of the module file
  - Checking inside drivers/net/Makefile indicates selecting CONFIG_TIGON3 builds tg3.c file and the module will be called tg3.ko
- Enter TI816X kernel configuration with following. Here, we assume that the default TI816X EVM configuration is set up as mentioned earlier and variables such as CROSS_COMPILE, ARCH are set appropriately.

```
$ make menuconfig
```

- From on screen menu, navigate by pressing ENTER or SPACE key on entries marked with '==>' and selecting appropriate configurations marked with '[*]' by pressing Y key and configurations with '<M>' by pressing M key as shown below. Note that a '*' inside the box '[ ]' indicates the corresponding option is enabled. Similarly, a '*' inside '< >' indicates that the corresponding driver is built into kernel while 'M' inside '< >' indicates the driver to be built as loadable kernel module

```
[*] Networking support  --->
==> Device Drivers  --->

==> [*] Network device support  --->
```

- After selecting above, entering inside "Network device support" shows...

```
--- Network device support
==> [*]   Ethernet (1000 Mbit)  --->

<M>   Broadcom Tigon3 support
```

- Use sequence of RIGHT ARROW key followed by ENTER key to exit the nested menu pages till you see "Do you wish to save your new kernel configuration?" dialog. Press ENTER to save the configuration and rebuild the kernel and modules as

```
$ make uImage
$ make modules
```

- Notice the output of modules build command

```
CC      drivers/net/tg3.mod.o
LD [M]  drivers/net/tg3.ko
IHEX    firmware/tigon/tg3.bin
IHEX    firmware/tigon/tg3_tso.bin
IHEX    firmware/tigon/tg3_tso5.bin
```

- From the above output, we see that kernel module tg3.ko is build. It also shows various firmware files generated for this driver. This means that this driver requires either (or all) of the firmware images to be (fully) functional.

Loading the EP Driver

- It may be possible to load the driver built with above step on already booted TI816X RC, but to avoid any configuration dependency issues, it is recommended to reboot the TI816X RC with newly built kernel.
- Once again, ensure that the PCIe devices are detected as expected. If not, try power cycling the system.
- Now, transfer the driver module with firmware images to the running TI816X system. You can use TFTP to get the files on TI816X EVM as follows. Note that in the example below, we assume TFTP server 172.24.190.43 is running with the concerned files copied into server's root directory.

```
TI8168_EVM # mkdir /home/pcie_rc
TI8168_EVM # cd /home/pcie_rc
TI8168_EVM # tftp -g -r tg3.ko 172.24.190.43
TI8168_EVM # tftp -g -r tg3.bin 172.24.190.43
TI8168_EVM # tftp -g -r tg3_tso.bin 172.24.190.43
```

- Note that above commands may vary depending upon the filesystem and tftp utility being used. Also, you can use different means of transferring the above file into TI816X filesystem as convenient. Assuming the files are transferred successfully and you have navigated to the directory containing the above files, use following commands to set up firmware files and load the EP driver:

```
TI8168_EVM # cp -f tg3.bin /lib/firmware
TI8168_EVM # cp -f tg3_tso.bin /lib/firmware
TI8168_EVM # insmod tg3.ko
```

- Something like following output should be seen:

```
tg3 0000:01:00.0: eth1: No interrupt was generated using MSI, switching to INTx mode
Please report this failure to the PCI maintainer and include system chipset information
tg3 0000:01:00.0: eth1: Link is up at 1000 Mbps, full duplex
tg3 0000:01:00.0: eth1: Flow control is off for TX and off for RX
```

- The driver is successfully loaded and the Ethernet interface associated with the PCIe Card is labelled as 'eth1' on the setup in above example.
- Notice 'lspci output now

```
TI8168_EVM # lspci  -kv -s 01:00
01:00.0 Ethernet controller: Broadcom Corporation NetXtreme BCM5751 Gigabit Ethernet PCI Express (rev 01)
    Subsystem: Broadcom Corporation NetXtreme BCM5751 Gigabit Ethernet PCI Express
    Flags: bus master, fast devsel, latency 0, IRQ 48
    Memory at 20000000 (64-bit, non-prefetchable) [size=64K]
    Capabilities: [48] Power Management version 2
    Capabilities: [50] Vital Product Data
    Capabilities: [58] MSI: Enable- Count=1/8 Maskable- 64bit+
    Capabilities: [d0] Express Endpoint, MSI 00
    Capabilities: [100] Advanced Error Reporting
    Capabilities: [13c] Virtual Channel
    Kernel driver in use: tg3
```

- From this point, the PCIe EP can be used as any other (on-chip) Ethernet Interface.
- Note that it is recommended to have this interface in different domain than TI816X EMAC interfaces. Other option could be to set up preference to this interface by setting 'metric' or bring TI816X EMAC interface(s) down so that Linux network stack will use PCIe Ethernet interface for network communications.
- In some cases it may so happen that the interface name will be different than the one printed in during 'insmod' (e.g., due to udev rules) hence it is advised to do `cat /proc/net/dev` to check interface names.
- **Note:** As mentioned in the *Features* section earlier, any EP driver operation resulting into EP device directly connected to TI816X RC being powered down or disabled would cause the link to go down and complete PCIe hardware initialization needs to be repeated (requires RC driver modification) or needs system reboot. Some possible examples include `ifconfig down` or `rmmod` on the EP driver.

Troubleshooting

This section lists various possible issues and how to troubleshoot them during initial setup using TI81XX RC device with various combination of PCIe Endpoint devices.

In all the cases, as a first step, it is recommended to enable PCIe debugging. Refer *Kernel Configuration* section described earlier to enable debugging.

Endpoint device not detected

This issue is generally detected by one (or more) of the following observations:

- Once the RC kernel is booted, 'lspci' command doesn't show any device.
- The driver for the endpoint you are using reports failure to load indicating error such as "no device found to operate on".
- You see following message for all the device from 0000:00:00 to 0000:1f:00 during boot time (need to have PCIe debugging enabled).

```
No link/device
```

In such case, follow the troubleshooting options (preferably in order) described in sub-sections below:

Troubleshoot 100MHz reference clock

- Ensure that the 100MHz reference clock (refclk) supplied to RC and EP meets PCIe specification constraints.
- If the refclks are separate for RC and EP, make sure that they are both fixed @100MHz and do not use SSC
- If SSC has to be used, design the system to use common clock. For example, use single 100MHz refclk to supply to RC as well as route it to the PCIe slot as output to be provided to the connected EP. Same can be carried forward to the scenario where PCIe switch and multiple EPs are involved.

Reset/Power on sequence

- Explore changing power up sequence across RC & EP(s). For example:
  - Power up EP before RC.
  - **Note** that is may not be straightforward in case when using refclk sourced from RC board. That is, you will still need to ensure that the refclk is provided to the note which is powered up first
  - Also, it is possible that the h/w is configured such that a reset is applied to downstream when RC is powered up. In such case, try to isolate the downstream from this reset.
- Depending upon the observations in the above step, you may need to time the reset sequence for final system.

DISCLAIMER: Various trademarks used in this document are copyrights of respective owners

References

[1] http://creativecommons.org/licenses/by-sa/3.0/

Article Sources and Contributors

TI81XX PCI Express Root Complex Driver User Guide *Source*: http://processors.wiki.ti.com/index.php?oldid=95836 *Contributors*: Hemantp

Image Sources, Licenses and Contributors

Image:TIBanner.png *Source*: http://processors.wiki.ti.com/index.php?title=File:TIBanner.png *License*: unknown *Contributors*: Nsnehaprabha

Image:PCIe_Topology.png *Source*: http://processors.wiki.ti.com/index.php?title=File:PCIe_Topology.png *License*: unknown *Contributors*: Hemantp

Image:PCIe_Topology_DM81XX_RC.png *Source*: http://processors.wiki.ti.com/index.php?title=File:PCIe_Topology_DM81XX_RC.png *License*: unknown *Contributors*: Hemantp

License

b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

a. Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

b. Each time You Distribute or Publicly Perform an Adaptation, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.

c. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

d. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

e. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

f. The rights granted under, and the subject matter referenced, in this License were drafted utilizing the terminology of the Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979), the Rome Convention of 1961, the WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996 and the Universal Copyright Convention (as revised on July 24, 1971). These rights and subject matter take effect in the relevant jurisdiction in which the License terms are sought to be enforced according to the corresponding provisions of the implementation of those treaty provisions in the applicable national law. If the standard suite of rights granted under applicable copyright law includes additional rights not granted under this License, such additional rights are deemed to be included in the License; this License is not intended to restrict the license of any rights under applicable law.