N710 P4080 PCIE on SDK1.0

P4080 PCIE Adapter Overview

The P4080 PCIE is a PCIE Adapter board with a Freescale QorIQ P4080 SOC/Processor. Even though the form factor is a PCIE Adapter, the HW design follows the Freescale's P4080 Development System, as much as possible. The P4080 PCIe Adapter is also referred to in some Freescale documentation as the N710 P4080PCIe adapter. The nomenclature for this document is P4080 PCIE Adapter.

QorIQ P4080 SOC combines eight Power Architecture e500mc cores with a Data Path Acceleration Architecture (DPAA) that includes acceleration blocks: Frame Manager (FMAN) - Packet parsing, classification and distribution, Queue Manager (QMAN) - Queue management for scheduling, packet sequencing and congestion management, Buffer Manager (BMAN) - Hardware Buffer management for buffer allocation and de-allocation, Cryptographic Security Acceleration (SEC 4.0) and Regular Expression Pattern Matching (PME 2.0).

In most cases, operating and programming the P4080 PCIE is the same as the Freescale DevSys. For additional reference, see the Freescale DevSys User guide, and the System Builder SDK 1.0 Documentation.

The P4080 PCIE Adapter comes with these user features

- One X8 PCIE Gen2 Interface
- One RJ45 Serial Console Port
- One 1G RGMII Management Port (10/100/1000 Ethernet Interface)
- Two 10G XAUI Ports (Dual 10GE SFP+ Ethernet Interfaces)
- 128MB NOR Flash memory (for boot)
- 4GB Memory
- USB 2.0
- JTAG Debug interface, (Supported by Freescale's CodeWarrior IDE)
- SD Media Card

P4080 PCIE Adapter Production Release Features

BSP

- RCW with Root Complex (to support stand-alone host mode) and End Point mode.
- U-boot with dual bank boot support.
- Support for standard u-boot commands.
- Linux USDPAA System Builder SDK 1.0 and DTS.
- 1G management port support.
- Linux SD card support.
- Linux USB support.
- Software Framework on P4080 to support exposure of multiple Hardware functions to Host.
- USDPAA Linux P4080 PCIE driver that enables control and data path between P4080 cores and x86 host - multi-ring support.
- USDPAA Linux P4080 PCIE driver also supports receiving commands, from the host, for multiple hardware functions and providing response to host.
- USDPAA Linux P4080 Ethernet driver that sends/receives network packets from two 10G ports on multi-core host - multi-ring and PCD Frame Queue support.
- USDPAA PCIE Ethernet Application that links in the PCIE Packet driver and Ethernet driver libraries and showcases the bi-directional control and data path between P4080 cores and multi-core host.
- Support of 8 Tx Frame Queues and 8 Tx confirmation Frame Queues for each 10G port on P4080.
- Support of 8 PCD Frame Queues for each 10G port on P4080.
- USDPAA APIs for support of Basic Direct and Basic Chaining DMA.
- Support for statistics of two 10G ports.

x86 software

- Software Framework on x86 Host to support exposure of multiple Hardware functions to x86 Host.
- **PCIE Packet driver**
  - Dynamically loaded and unloaded as a module. Also supports automatic load of modules on host boot-up.
  - PCIE initialization.
  - Initialization handshake with P4080 driver.
  - Command Request/Response mechanism to P4080.
  - API to upper-level (Ethernet) drivers – en-queue to Tx ring.
  - Callback for Rx ring of upper-level (Ethernet) drivers.
  - API to en-queue commands to the Command Ring.
  - Callback for handling command ring responses.
  - Read/Write to CCSR.
- **Ethernet driver**
  - Dynamically loaded and unloaded as a module. Also supports automatic load of modules on host boot-up.
  - Support of 8 Tx and Rx rings on x86 Host.
  - Support for both 10G ports.
  - Supports MSI for the device.
  - ifconfig statistics.
  - NAPI support.

    - Promiscuous mode.
    - Multicast filtering.
    - MTU change.
    - Support jumbo frames, 9600 KB max frame size.
    - MAC address change.
    - ifconfig up/down.
    - Tx checksum offload.
    - RX checksum verification and generation.
    - Limited Ethtool support – driver info, ring parameters, statistics, get/set Tx checksum.
- **PCIe Tool** - dump CCSR BAR and Read/Write into BAR memory based on offset

Pre-requisites and Assumption

- The release patches SHOULD be applied on the extracted source code of QorIQ-DPAA-SDK-20110609-systembuilder.iso file.
- The SDK installation and compilation is tested on Ubuntu-10.10. Make sure to use this version of OS for installation and compilation.
- Bash shell is needed for compilation of the source code. Install this before starting the compilation.

Installation of the SDK

Please follow the steps below to install System Builder on your host machine.

- Mount the ISO image on your machine:
  $ sudo mount -o loop QorIQ-DPAA-SDK-20110609-systembuilder.iso /mnt/
  NOTE: Install SDK as non-root user.
- Run the following command to extract the SDK source code from ISO file. This command will prompt for some imputs.
  $ /mnt/install
- Read the End User License Agreement and give "yes" as input.

This would prompt for the installation path for SDK. Give path to "/<path>/P4080-PCIE-SDK-1.0-05-03-2012" (Note that P4080-PCIE-SDK-1.0-05-03-2012 is the release directory which is created when P4080-PCIE-SDK-1.0-05-03-2012.tgz is
un-tarred)

NOTE: There is no uninstall script. To uninstall System Builder you need to remove the *<systembuilder_install_path>/QorIQ-DPAA-SDK-20110609-systembuilder* directory manually.

Applying the Patch

- Move to P4080-PCIE-SDK-1.0-05-03-2012 directory which is created as a result of untarring the tar file P4080-PCIE-SDK-1.0-05-03-2012.tgz.

$ *cd P4080-PCIE-SDK-1.0-05-03-2012*

- Run the script to apply the n710 patches to SDK with EP or RC as an argument to select Endpoint (EP) mode or Root Complex (RC) mode.
  $ sh script-patch.sh EP

Host Environment Setup

- Move to the SDK main directory.
  $ cd <system_builder_installation_path>
- Run the preparation script:
- *$ ./scripts/ubuntu-oe.sh*

Create a Platform Development Kit (PDK) project

- Move to the SDK main directory.
  $ cd <system_builder_installation_path>
- Create the platform with necessary compilation parameters.
  $ *./scripts/create-config.py --config-file=fsl-p4080n710/sample-create-config.ini -j 4 -t 2*
- After running above steps, a project folder named *build_p4080n710_release* will be created.

Cross-compilation environment setup

- Move to the SDK main directory.
  $ cd <system_builder_installation_path>
- Setup your shell environment by using the *bitbake.rc* file. The environment file must be loaded prior to building a project. Use the appropriate command for your shell to load a file into the environment.
- For example in bash shell:
  $ source build_p4080n710_release/bitbake.rc

Building the Images

- Use the following command to compile the SDK.
  $ bitbake devel-image

Building Host(x86) Device Drivers

- The P4080 card is connected to the x86 host machine.
- The x86 drivers are placed in P4080-PCIE-SDK-1.0-05-03-2012/x86 directory. Move to this directory.
  $ cd P4080-PCIE-SDK-1.0-05-03-2012/x86
  $ make

NOTE: Above step generates images for uboot bootloader, Linux Kernel, DTB files, FMAN micro code and root file system. The images are placed in *<systembuilder_install_path>/QorIQ-DPAA-SDK-20110609-systembuilder/build_p4080n710_release/deploy/glibc/images/* directory.

Setting up TFTP boot

- Install tftpd related packages
  $ sudo apt-get install xinetd tftpd tftp
- Create */etc/xinetd.d/tftp* file and put the following content.

```
service tftp {
protocol = udp
port = 69
socket_type = dgram
wait = yes
user = nobody
server = /usr/sbin/in.tftpd
server_args = /tftp
disable = no
}
```

- Create tftp directory for image storing

```
$ sudo mkdir /tftp
$ sudo chmod –R 777 /tftp
$ sudo chown –R nobody /tftp
```

- Start tftpd using xinetd
  $ sudo /etc/init.d/xinetd start

Images

Endpoint (EP) mode :

rcw_0x2_rev2_high_ep_1.3GHz_clock.bin
u-boot-p4080n710.bin
uImage-p4080n710.bin
devel-image-p4080n710.ext2.gz.u-boot
uImage-p4080n710-usdpaa.dtb

RootComplex (RC) mode :

rcw_0x2_rev2_high_rc_1.3GHz_clock.bin
u-boot-p4080n710.bin
uImage-p4080n710.bin
devel-image-p4080n710.ext2.gz.u-boot
uImage-p4080n710.dtb

Flashing the Images on P4080 Card

To Flash Endpoint mode images :

Programming Bank 0

- Copy P4080 images into tftp directory on Host machine. Assume *"/tftp"* is the directory.
- Copy RCW using "*<systembuilder_install_path>/QorIQ-DPAA-SDK-20110609-systembuilder/build_p4080n710_release/deploy/glibc/images/rcw/rcw_0x2_rev2_high_ep_1.3GHz_clock.bin /tftp*"
- Copy U-boot using "*<systembuilder_install_path>/QorIQ-DPAA-SDK-20110609-*

systembuilder/build_p4080n710_release/deploy/glibc/images/u-boot-p4080n710.bin /tftp"

- Copy device tree using "*<systembuilder_install_path>/QorIQ-DPAA-SDK-20110609-systembuilder/build_p4080n710_release/deploy/glibc/images/* uImage-p4080n710-usdpaa.dtb */tftp*"
- Copy Linux image using "*<systembuilder_install_path>/QorIQ-DPAA-SDK-20110609-systembuilder/build_p4080n710_release/deploy/glibc/images/uImage-p4080n710.bin /tftp*"
- Copy root file system using "*<systembuilder_install_path>/QorIQ-DPAA-SDK-20110609-systembuilder/build_p4080n710_release/deploy/glibc/images/devel-image-p4080n710.ext2.gz.u-boot /tftp*"

- **Boot into Bank 4**(can be done using *"n710_reset altbank"* on U-boot prompt), interrupt the bootloader and execute the following commands
- Program RCW image into flash using following commands
  $ tftp 0x02000000 /tftp/rcw_0x2_rev2_high_ep_1.3GHz_clock.bin
  $ erase 0xec000000 +$filesize
  $ cp.b 0x02000000 0xec000000 $filesize
- Program U-boot image into flash using following commands
  $ tftp 0x01000000 /tftp/u-boot-p4080n710.bin
  $ erase 0xebf80000 +$filesize
  $ *cp.b 0x01000000 0xebf80000 $filesize*
- Program Linux image into flash using following commands
  $ tftp 0x1000000 /tftp/uImage-p4080n710.bin
  $ erase 0xec020000 +$filesize
  $ cp.b 0x1000000 0xec020000 $filesize
- Program root filesystem into flash using following commands
  $ tftp 0x1000000 /tftp/devel-image-p4080n710.ext2.gz.u-boot
  $ erase 0xed300000 +$filesize
  $ cp.b 0x1000000 0xed300000 $filesize
- **Boot into Bank 0,** interrupt the bootloader and execute the following commands
  $ tftp 0x01000000 /tftp/uImage-p4080n710-usdpaa.dtb
  $ erase 0xe8800000 +$filesize
  $ cp.b 0x01000000 0xe8800000 $filesize

Power off the Host and power it on.

Programming Bank 4

- Copy P4080 images into tftp directory on Host machine. Assume *"/tftp"* is the directory.
- Copy RCW using "*<systembuilder_install_path>/QorIQ-DPAA-SDK-20110609-systembuilder/build_p4080n710_release/deploy/glibc/images/rcw/rcw_0x2_rev2_high_ep_1.3GHz_clock.bin /tftp*"
- Copy U-boot using "*<systembuilder_install_path>/QorIQ-DPAA-SDK-20110609-systembuilder/build_p4080n710_release/deploy/glibc/images/u-boot-p4080n710.bin /tftp*"
- Copy device tree using "*<systembuilder_install_path>/QorIQ-DPAA-SDK-20110609-systembuilder/build_p4080n710_release/deploy/glibc/images/* uImage-p4080n710-usdpaa.dtb */tftp*"
- Copy Linux image using "*<systembuilder_install_path>/QorIQ-DPAA-SDK-20110609-systembuilder/build_p4080n710_release/deploy/glibc/images/uImage-p4080n710.bin /tftp*"

- Copy root file system using "*<systembuilder_install_path>/QorIQ-DPAA-SDK-20110609-systembuilder/build_p4080n710_release/deploy/glibc/images/devel-image-p4080n710.ext2.gz.u-boot /tftp*"

- **Boot into Bank 0**, interrupt the bootloader and execute the following commands
- Program RCW image into flash using following commands
  $ tftp 0x02000000 /tftp/rcw_0x2_rev2_high_ep_1.3GHz_clock.bin
  $ erase 0xec000000 +$filesize
  $ cp.b 0x02000000 0xec000000 $filesize
- Program U-boot image into flash using following commands
  $ tftp 0x01000000 /tftp/u-boot-p4080n710.bin
  $ erase 0xebf80000 +$filesize
  $ *cp.b 0x01000000 0xebf80000 $filesize*
- Program Linux image into flash using following commands
  $ tftp 0x1000000 /tftp/uImage-p4080n710.bin
  $ erase 0xec020000 +$filesize
  $ cp.b 0x1000000 0xec020000 $filesize
- Program root filesystem into flash using following commands
  $ tftp 0x1000000 /tftp/devel-image-p4080n710.ext2.gz.u-boot
  $ erase 0xed300000 +$filesize
  $ cp.b 0x1000000 0xed300000 $filesize
- **Boot into Bank 4,** interrupt the bootloader and execute the following commands
  $ tftp 0x01000000 /tftp/uImage-p4080n710-usdpaa.dtb
  $ erase 0xe8800000 +$filesize
  $ cp.b 0x01000000 0xe8800000 $filesize

Power off the Host and power it on.

Booting into bank4

- Use the following command at U-boot prompt to boot into flash bank4
  $ n710_reset altbank

Packet Process startup

- After Linux on P4080 boots up, a login prompt will be displayed. Use username "root" to login.
- To start the necessary initialization and start the PCIE Packet Process on P4080 Linux, run the startup script:
  $ /etc/rc.d/init.d/usdpaa_pcie start
- After this, load the Host device drivers on x86 host to complete the initialization handshake between P4080 application and Host Packet Driver.
  a. Load the Packet driver :
  $ insmod x86/drivers/pcie-core/src/pcie-core.ko
  b. Load the Ethernet driver:
  $ insmod x86/drivers/p4080-eth/src/p4080eth.ko
  c. Two 10G interfaces will be created as a result of above instructions. Check this using "ifconfig -a" command.
  d. Bring UP the 10G interface(s) and assign IP address:
  $ ifconfig <interface-name> 10.0.0.1

e. Ping any remote machine connected to network.

- After completion of initialization handshake, CLI client on the P4080 PCIE Adapter can be started by executing the following in Linux:
  $ /usr/bin/cli_client

NOTE: Only after completion of initialization handshake the CLI client can be started.

Jumbo Frame Support

To get the Jumbo Frame support, Ethernet Driver has to be loaded with a command line argument "jumbo_frames". Load the Ethernet driver using *"insmod x86/drivers/p4080-eth/src/p4080eth.ko jumbo_frames=1"*.

Flow classification

The default FMC configuration and policy files built into rootfs from "*QorIQ-DPAA-SDK-20110609-systembuilder/build_p4080n710_release/work/ppce500mc-linux/usdpaa-0.4.1-r1.0/usdpaa-0.4.1/apps/ppac*" directory. FMC configuration file us_config_serdes_0x2.xml sets the first 10G port to use the policy **hash_5tuple_policy1** and **hash_5tuple_policy2** for second 10G port. The policy file us_policy_hash_ipv4_src_dst.xml declares distributions **hash_5tuple_dist1** and **hash 5tuple_dist2** for the policy **hash_5tuple_policy1 and hash_5tuple_dist2** respectively to classify the TCP traffic.

The file has to be modified appropriately if UDP flow classification is required.

The FMC configuration file is configured as follows for setting up FMAN's 10G interface 1.

```
<cfgdata>
      <config>
         <engine name="fm0">
                <port type="10G" number="0" policy="hash_5tuple_policy1" />
         </engine>
         <engine name="fm1">
                <port type="10G" number="0" policy="hash_5tuple_policy2" />
         </engine>
      </config>
</cfgdata>
```

The FMC policy file is configured as follows for setting up the TCP distribution.

```
<policy name="hash_5tuple_policy1">
<dist_order>
         <distributionref name="hash_5tuple_dist1" />
         <distributionref name="default_5tuple_dist1" />
</dist_order>
</policy>

<distribution name="hash_5tuple_dist1">
   <queue count="8" base="0xe00"/>
```

```
    <key>
        <fieldref name="ipv4.src"/>
        <fieldref name="ipv4.dst"/>
        <fieldref name="tcp.sport"/>
        <fieldref name="tcp.dport"/>
    </key>
</distribution>
```

Usage of 10G interfaces

When x86 Host boots up, the P4080 PCIE Adapter is shown as PCIe card in the output of *"lspci"* command. The verbose output of the P4080 PCIE Adapter as endpoint device is as follows.

```
01:00.0 Power PC: Freescale Semiconductor Inc P4080E (rev 20) (prog-if 01)
        Control: I/O+ Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr- Stepping- SERR+ FastB2B-
DisINTx+
        Status: Cap+ 66MHz- UDF- FastB2B- ParErr- DEVSEL=fast >TAbort- <TAbort- <MAbort- >SERR- <PERR-
INTx-
        Latency: 0, Cache Line Size: 64 bytes
        Interrupt: pin A routed to IRQ 29
        Region 0: Memory at f8000000 (32-bit, non-prefetchable) [size=16M]
        Region 1: Memory at f0000000 (32-bit, prefetchable) [size=64M]
        Capabilities: [44] Power Management version 3
                Flags: PMEClk- DSI- D1+ D2+ AuxCurrent=0mA PME(D0+,D1+,D2+,D3hot+,D3cold-)
                Status: D0 PME-Enable- DSel=0 DScale=0 PME-
        Capabilities: [4c] Express (v2) Endpoint, MSI 00
                DevCap: MaxPayload 256 bytes, PhantFunc 0, Latency L0s <64ns, L1 <1us
                        ExtTag- AttnBtn- AttnInd- PwrInd- RBE+ FLReset-
                DevCtl: Report errors: Correctable- Non-Fatal- Fatal+ Unsupported-
                        RlxdOrd+ ExtTag- PhantFunc- AuxPwr- NoSnoop+
                        MaxPayload 128 bytes, MaxReadReq 4096 bytes
                DevSta: CorrErr- UncorrErr- FatalErr- UnsuppReq- AuxPwr- TransPend-
                LnkCap: Port #0, Speed 5GT/s, Width x8, ASPM L0s, Latency L0 <2us, L1 unlimited
                        ClockPM- Suprise- LLActRep- BwNot-
                LnkCtl: ASPM Disabled; RCB 64 bytes Disabled- Retrain- CommClk-
                        ExtSynch- ClockPM- AutWidDis- BWInt- AutBWInt-
                LnkSta: Speed 2.5GT/s, Width x8, TrErr- Train- SlotClk- DLActive- BWMgmt- ABWMgmt-
        Capabilities: [88] Message Signalled Interrupts: Mask- 64bit+ Queue=0/4 Enable+
                Address: 00000000fee0300c  Data: 41c1
        Capabilities: [100] Advanced Error Reporting <?>
        Kernel driver in use: p4080-ep
```

In the above output, inbound ATMU is setup using BAR0 and BAR1. This is highlighted in green color. Region 0 refers to BAR0 and Region 1 refers to BAR1. Region 0 opens window to 16MB CCSR space of P4080. Region 1 opens window to 64MB memory of P4080 which includes initialization handshake memory, static and dynamic buffer pool memory. MSI capability is initialized and can be observed from the lines highlighted in grey color.

The x86 Host Packet Driver directly interfaces with the P4080 PCIE Adapter with the help of Linux Kernel PCIe subsystem. The Ethernet device driver uses the x86 Host packet driver APIs to interact with the adapter. To load the x86 Host drivers follow the following steps.

- Load the Packet driver using *"insmod x86/drivers/pcie-core/src/pcie-core.ko"*
- Load the Ethernet driver using *"insmod x86/drivers/p4080-eth/src/p4080eth.ko"*
- Two 10G interfaces will be created as a result of above instructions. Check this using *"ifconfig -a"* command.
- Bring UP the 10G interface(s) and assign IP address using "ifconfig <interface-name> 10.0.0.1".
- Ping any remote machine connected to network.

Packet Statistics Using CLI on P4080

The P4080 PCIE Adapter has a management Ethernet port that provides serial console access into

P4080 Linux. This can be used to setup mincom to some external host. After getting Linux command prompt in minicom, run *"/usr/bin/cli_client"* to start the CLI on P4080. This CLI offers the following commands.

• **?**	Displays command help
• **help**	Displays command help
• **tx-fm1-10g**	Displays statistics of Tx Frame Queues associated to 10G interface of FMAN-1
• **rx-default-fm1-10g**	Displays statistics of Rx Default Frame Queues associated to 10G interface of FMAN-1
• **rx-pcd-fm1-10g**	Displays statistics of Rx PCD Frame Queues associated to 10G interface of FMAN-1
• **msi-interrupts-fm1-10g**	Displays statistics of interrupts associated to 10G interface of FMAN-1
• **tx-fm2-10g**	Displays statistics of Tx Frame Queues associated to 10G interface of FMAN-2
• **rx-default-fm2-10g**	Displays statistics of Rx Default Frame Queues associated to 10G interface of FMAN-2
• **rx-pcd-fm2-10g**	Displays statistics of Rx PCD Frame Queues associated to 10G interface of FMAN-2
• **msi-interrupt-fm2-10g**	Displays statistics of interrupts associated to 10G interface of FMAN-2
• **stop-packet-process**	Stops the packet driver application gracefully
• **exit**	Exits from CLI prompt
• **q**	Exits from CLI prompt

Sample output for Tx Packet statistics

```
cli> get-tx-stats-1
+--------------------------------------------------------+
|CoreId     Tx FQId     Tx Packets        Tx Bytes |
+--------------------------------------------------------+
|Core-0     536         0                 0           |
|Core-1     537         5897              579521      |
|Core-2     538         0                 0           |
|Core-3     539         0                 0           |
|Core-4     540         0                 0           |
|Core-5     541         6                 468         |
|Core-6     542         2                 84          |
|Core-7     543         0                 0           |
+--------------------------------------------------------+
```

Sample output for Rx default Frame Queue statistics

```
cli> get-rx-default-stats-1
+------------------------------------------------------------------------------------------------------------+
|CoreId     Rx FQId     Rx Packets        Rx Bytes      Rx Drop Packets        Rx Drop Bytes |
+------------------------------------------------------------------------------------------------------------+
|Core-0         101           0            0                  0                      0
|
|Core-1         101           1           60                  0                      0
|
|Core-2         101         981        96138                  0                      0
```

```
|
|Core-3      101        980       96040          0                    0
|
|Core-4      101        980       96040          0                    0
|
|Core-5      101        980       96040          0                    0
|
|Core-6      101        979       95942          0                    0
|
|Core-7      101        980       96002          0                    0
|
+------------------------------------------------------------------------------------------------------------------+
```

Sample output for Rx PCD Frame Queue statistics

cli> get-rx-pcd-stats-1

```
+------------------------------------------------------------------------------------------------------------------+
|CoreId     Rx FQId     Rx Packets       Rx Bytes      Rx Drop Packets         Rx Drop Bytes |
+------------------------------------------------------------------------------------------------------------------+
|Core-1       3584          0            0               0                    0               |
|Core-2       3585          0            0               0                    0               |
|Core-3       3586          0            0               0                    0               |
|Core-4       3587          0            0               0                    0               |
|Core-5       3588          0            0               0                    0               |
|Core-6       3589          0            0               0                    0               |
|Core-7       3590        59760        3944228         0                    0                 |
|Core-1       3591          0            0               0                    0               |
+------------------------------------------------------------------------------------------------------------------+
```

Stopping the interface usage

- **Host Interface Down:** ***"ifconfig <interface-name> down"***. This disables the corresponding FMan interface on P4080.
- **Host Interface Up :** ***"ifconfig <interface-name> up"***. This enables the corresponding FMan interface on P4080.
- **Graceful Shutdown of P4080 Packet Driver Application:** These operations bring the system back to its initialized state.
  1. On P4080 execute ***"stop-packet-process***" from CLI.
  2. Unload x86 Host drivers: On host execute "***rmmod p4080eth pciecore***"
     NOTE: It is mandatory to execute step 2 as part of the graceful shutdown as described above. Stopping the P4080 Packet Driver Application and restarting it without unloading x86 host drivers is not supported and will result in unexpected behavior.
- **Restart P4080 Packet Driver Application:**
  On P4080 execute "/etc/rc.d/init.d/usdpaa_pcie start". X86 Host drivers can be loaded after this and the Ethernet interfaces on can be brought UP to send the traffic.
- **Host Drivers Unload/Reload:** When host drivers are unloaded "rmmod p4080eth pciecore", there is no need to explicitly stop the P4080 Packet driver application. The Host drivers can be re-loaded and the traffic can be sent on any of the two 10G interfaces after this.
- Stopping the P4080 PCIe Packet Driver Application using "/etc/rc.d/init.d/usdpaa_pcie stop" is not supported.

Packet Statistics using sysfs on x86 Host

The x86 Host Ethernet driver exposes sysfs files to show the packet statistics for an Ethernet interface. The statistics can be obtained using *"cat /sys/smart_nic/stats"*. The output of the statistics is as follows.

Sample output for statistics on x86 Host Ethernet interface

$ cat /sys/smart_nic/stats

```
Interface eth1 stats:  Ring0    Ring1    Ring2    Ring3    Ring4    Ring5    Ring6   Ring7
Tx pkt count:           0       5897      0        0        0        6        2        0
Tx pkt Err count:        0        0        0        0        0        0        0        0
Tx byte count:           0       579521   0        0        0       468       84       0
Rx pkt count:            0        1       981      980      980      980      979      980
Rx pkt drop count:      0        0        0        0        0        0        0        0
Rx byte count:           0        46      54936    54880    54880   54880    54824   54870

Interface eth2 stats:  Ring0    Ring1    Ring2    Ring3    Ring4    Ring5    Ring6   Ring7
Tx pkt count:            0    0        0        0        0        0        0        0
Tx pkt Err count:        0    0        0        0        0        0        0        0
Tx byte count:           0    0         0        0        0        0        0        0
Rx pkt count:            0    0         0        0        0        0        0        0
Rx pkt drop count:      0    0         0        0        0        0        0        0
Rx byte count:           0    0         0        0        0        0        0        0
```

Using PCIE tool on x86 Host

PCIE tool can be used on x86 Host machine to read and write P4080 CCSR values and into any BAR memory region opened by P4080 PCIE endpoint device. To use the tool, run *"x86/utils/src/fsl_pcie_tool"*. Various options to this command are listed below

- -r - Read the register
- -w - Write into the register
- -o - Offset of the register in CCSR space
- -b - Number of BAR register that is to be used for read and write operations
- -D - Display various CCSR register blocks
- -d - Display all the register values in a specific CCSR register block.

Sample output showing CCSR registers of FMan 10G interface

Below command will print all the CCSR blocks with various IDs. These IDs can be used to print a specific block register values.

$ x86/utils/src/fsl_pcie_tool -D
$ x86/utils/src/fsl_pcie_tool –d 134

```
+------------------------------------------------------------------------------------------------------------------------------------------------
------+
|Frame Manager 1 - Ethernet 10GEC 1 Registers
|
+------------------------------------------------------------------------------------------------------------------------------------------------
------+
|E10GEC1_EC10G_ID(0x4f0000): 0x0001032c          |E10GEC1_COMMAND_CONFIG(0x4f0008): 0x00020043
|
|E10GEC1_MAC_ADDR_0(0x4f000c): 0x00bd0c00          |E10GEC1_MAC_ADDR_1(0x4f0010): 0x0000d6b1
|
|E10GEC1_MAXFRM(0x4f0014): 0x00002580              |E10GEC1_PAUSE_QUANT(0x4f0018): 0x0000f000
|
|E10GEC1_HASHTABLE_CTRL(0x4f002c): Non-Readable  |E10GEC1_STATUS(0x4f0040): 0x00000000
|
|E10GEC1_TX_IPG_LENGTH(0x4f0044): 0x0000000c         |E10GEC1_MAC_ADDR_2(0x4f0048): 0x00000000
|
|E10GEC1_MAC_ADDR_3(0x4f004c): 0x00000000            |E10GEC1_IMASK(0x4f0060): 0x00010ffb
|
|E10GEC1_IEVENT(0x4f0064): 0x00000000                |E10GEC1_TFRM_U(0x4f0080): 0x00000000
|
|E10GEC1_TFRM_L(0x4f0084): 0x015cece7                |E10GEC1_RFRM_U(0x4f0088): 0x00000000
|
|E10GEC1_RFRM_L(0x4f008c): 0x0001008d                |E10GEC1_RFCS_U(0x4f0090): 0x00000000
|
|E10GEC1_RFCS_L(0x4f0094): 0x00000000                |E10GEC1_RALN_U(0x4f0098): 0x00000000
|
|E10GEC1_RALN_L(0x4f009c): 0x00000000                |E10GEC1_TXPF_U(0x4f00a0): 0x00000000
|
|E10GEC1_TXPF_L(0x4f00a4): 0x00000000                |E10GEC1_RXPF_U(0x4f00a8): 0x00000000
|
|E10GEC1_RXPF_L(0x4f00ac): 0x00000000                |E10GEC1_RLONG_U(0x4f00b0): 0x00000000
|
|E10GEC1_RLONG_L(0x4f00b4): 0x00000000               |E10GEC1_RFLR_U(0x4f00b8): 0x00000000
|
|E10GEC1_RFLR_L(0x4f00bc): 0x00000000                |E10GEC1_TVLAN_U(0x4f00c0): 0x00000000
|
|E10GEC1_TVLAN_L(0x4f00c4): 0x00000000               |E10GEC1_RVLAN_U(0x4f00c8): 0x00000000
|
|E10GEC1_RVLAN_L(0x4f00cc): 0x00000000               |E10GEC1_TOCT_U(0x4f00d0): 0x00000008
|
|E10GEC1_TOCT_L(0x4f00d4): 0x11df76ba                |E10GEC1_ROCT_U(0x4f00d8): 0x00000000
|
|E10GEC1_ROCT_L(0x4f00dc): 0x0049205e                |E10GEC1_RUCA_U(0x4f00e0): 0x00000000
|
|E10GEC1_RUCA_L(0x4f00e4): 0x0001008d                |E10GEC1_RMCA_U(0x4f00e8): 0x00000000
|
|E10GEC1_RMCA_L(0x4f00ec): 0x00000000                |E10GEC1_RBCA_U(0x4f00f0): 0x00000000
|
|E10GEC1_RBCA_L(0x4f00f4): 0x00000000                |E10GEC1_TERR_U(0x4f00f8): 0x00000000
|
|E10GEC1_TERR_L(0x4f00fc): 0x00000000                |E10GEC1_TUCA_U(0x4f0108): 0x00000000
|
|E10GEC1_TUCA_L(0x4f010c): 0x015cecce                |E10GEC1_TMCA_U(0x4f0110): 0x00000000
|
|E10GEC1_TMCA_L(0x4f0114): 0x00000018                |E10GEC1_TBCA_U(0x4f0118): 0x00000000
|
|E10GEC1_TBCA_L(0x4f011c): 0x00000001                |E10GEC1_RDRP_U(0x4f0120): 0x00000000
|
|E10GEC1_RDRP_L(0x4f0124): 0x00000000                |E10GEC1_REOCT_U(0x4f0128): 0x00000000
|
|E10GEC1_REOCT_L(0x4f012c): 0x00492bee               |E10GEC1_RPKT_U(0x4f0130): 0x00000000
|
|E10GEC1_RPKT_L(0x4f0134): 0x0001009a                |E10GEC1_TRUND_U(0x4f0138): 0x00000000
|
|E10GEC1_TRUND_L(0x4f013c): 0x00000000               |E10GEC1_R64_U(0x4f0140): 0x00000000
|
|E10GEC1_R64_L(0x4f0144): 0x00000004                 |E10GEC1_R127_U(0x4f0148): 0x00000000
|
|E10GEC1_R127_L(0x4f014c): 0x00010069                |E10GEC1_R255_U(0x4f0150): 0x00000000
|
|E10GEC1_R255_L(0x4f0154): 0x00000003                |E10GEC1_R511_U(0x4f0158): 0x00000000
|
|E10GEC1_R511_L(0x4f015c): 0x0000002a                |E10GEC1_R1023_U(0x4f0160): 0x00000000
|
|E10GEC1_R1023_L(0x4f0164): 0x00000000               |E10GEC1_R1518_U(0x4f0168): 0x00000000
```

```
|
|E10GEC1_R1518_L(0x4f016c): 0x00000000                    |E10GEC1_R1519X_U(0x4f0170): 0x00000000
|
|E10GEC1_R1519X_L(0x4f0174): 0x00000000                   |E10GEC1_TROVR_U(0x4f0178): 0x00000000
|
|E10GEC1_TROVR_L(0x4f017c): 0x00000000                     |E10GEC1_TRJBR_U(0x4f0180): 0x00000000
|
|E10GEC1_TRJBR_L(0x4f0184): 0x00000000                     |E10GEC1_TRFRG_U(0x4f0188): 0x00000000
|
|E10GEC1_TRFRG_L(0x4f018c): 0x00000000                     |E10GEC1_RERR_U(0x4f0190): 0x00000000
|
|E10GEC1_RERR_L(0x4f0194): 0x00000000                   |E10GEC1_MDIO_CFG_STAT(0x4f1030): 0x00007400
|
|E10GEC1_MDIO_CTRL(0x4f1034): Non-Readable          |E10GEC1_MDIO_DATA(0x4f1038): 0x00000000
|
|E10GEC1_MDIO_ADDR(0x4f103c): Non-Readable          |
|
+-----------------------------------------------------------------------------------------------------------------
------------------------------+
```