PMC IPC-BDMA Detail Design

Table of Contents

[1 Introduction 3](#_Toc380570452)

[1.1 doc 3](#_Toc380570453)

[1.1.1 Plan 3](#_Toc380570454)

[1.1.2 Requirement 3](#_Toc380570455)

[1.1.3 Purposal 3](#_Toc380570456)

[1.1.4 Detail design 3](#_Toc380570457)

[1.1.5 Unit test 3](#_Toc380570458)

[1.2 Environment 3](#_Toc380570459)

[1.2.1 RC platform 3](#_Toc380570460)

[1.2.2 EP platform 3](#_Toc380570461)

[2 Definitions and flows 3](#_Toc380570462)

[2.1 Overall 3](#_Toc380570463)

[2.1.1 64-bit Arch compatiblity 3](#_Toc380570464)

[2.2 IPC-BDMA Flow overview 3](#_Toc380570465)

[2.2.1 Blocks 3](#_Toc380570466)

[2.2.2 NIC init/up flow 4](#_Toc380570467)

[2.2.3 RC initiate data transmit to EP 5](#_Toc380570468)

[2.2.4 EP initiate data transmit to RC 6](#_Toc380570469)

[2.3 Common FIFO descriptor definition and format 6](#_Toc380570470)

[2.3.1 FIFO descriptor format 6](#_Toc380570471)

[2.3.1.1 FIFO descriptor type 7](#_Toc380570472)

[2.3.1.2 FIFO descriptor session and action 7](#_Toc380570473)

[2.4 FIFO descriptors 8](#_Toc380570474)

[2.4.1 FIFO data descriptor 8](#_Toc380570475)

[2.4.1.1 Data descriptor format 8](#_Toc380570476)

[2.4.2 FIFO command descriptor 8](#_Toc380570477)

[2.4.2.1 Command descriptor format 8](#_Toc380570478)

[2.4.3 SGL data flow 8](#_Toc380570479)

[2.4.4 RC side SGL buffer 8](#_Toc380570480)

[2.4.5 EP side SGL buffer 8](#_Toc380570481)

[3 EP Side Feature 8](#_Toc380570482)

[3.1 Hardware FIFO 8](#_Toc380570483)

[3.2 Hardware PCIe 9](#_Toc380570484)

[3.3 Linux Kernel IPC-BDMA API/LIB 10](#_Toc380570485)

[3.3.1 components 10](#_Toc380570486)

[3.3.2 flow 10](#_Toc380570487)

[3.3.3 structure 10](#_Toc380570488)

[3.3.4 function 10](#_Toc380570489)

[3.4 Linux Kernel BDMA API 10](#_Toc380570490)

[3.5 Linux Kernel netdev driver 10](#_Toc380570491)

[3.6 Linux Application 10](#_Toc380570492)

[3.6.1 TCP bw test 10](#_Toc380570493)

[3.6.2 ethtool test 10](#_Toc380570494)

[4 RC side Feature 10](#_Toc380570495)

[4.1 Linux Kernel PCI driver 10](#_Toc380570496)

[4.2 Linux Kernel IPC-BDMA API/LIB 10](#_Toc380570497)

[4.2.1 components 10](#_Toc380570498)

[4.2.2 flow 10](#_Toc380570499)

[4.2.3 structure 10](#_Toc380570500)

[4.2.4 function 10](#_Toc380570501)

[4.3 Linux Kernel netdev driver 10](#_Toc380570502)

[4.4 Linux Application 10](#_Toc380570503)

[4.4.1 TCP bw test 10](#_Toc380570504)

[4.4.2 ethtool test 10](#_Toc380570505)

**Tables**

[**Table 1 FIFO descriptor format** 6](#_Toc380570506)

[**Table 2 FIFO descriptor fields** 6](#_Toc380570507)

[**Table 3 FIFO data descriptor request format** 8](#_Toc380570508)

**Figures**

[**Figure 1 IPC-BDMA block overview** 3](#_Toc380570509)

[**Figure 2 IPC-BDMA RC initiate data transmit to EP** 4](#_Toc380570510)

[**Figure 2 IPC-BDMA RC initiate data transmit to EP** 5](#_Toc380570511)

[**Figure 3 IPC-BDMA EP initiate data transmit to RC** 6](#_Toc380570512)

# Introduction

## Environment

### RC platform

Use x86 host PC as PCIe RC platform

### EP platform

Use PMCS-SOC(Storage Processor) as PCIe EP platform

# Definitions and flows

## Environment

### 64-bit Arch compatiblity

All structures and functions need to be 64-bit compatible.

## IPC-BDMA Flow overview

### Overview

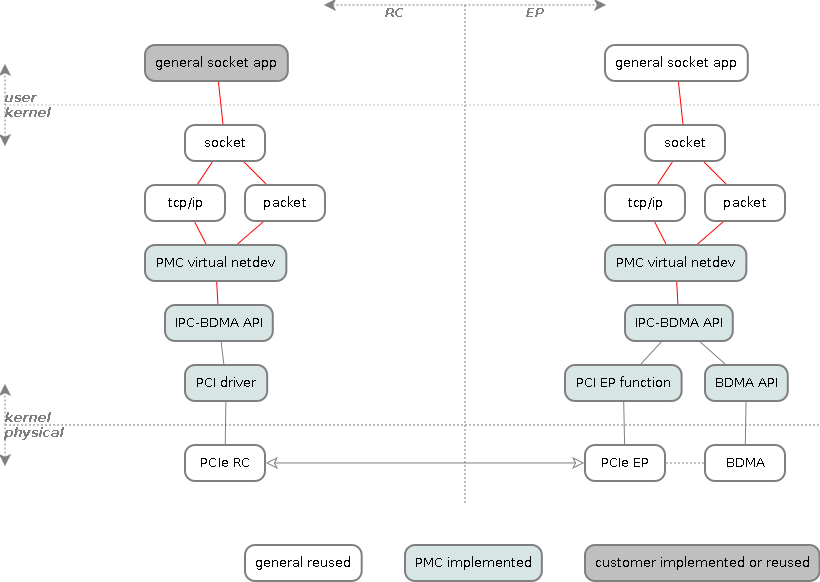
Data moving: The host and EP board use BDMA engine to move data between host memory and EP DDR memory, thru physical PCIe link.

BDMA control: The host and EP board contains IPC-BDMA functions to control the BDMA action.

Virtual network interface: The host and EP board exposed virtual Ethernet interface to service user application. The virtual Ethernet interface send/receive Ethernet packets thru "IPC-BDMA" API instead of real Ethernet physical link.

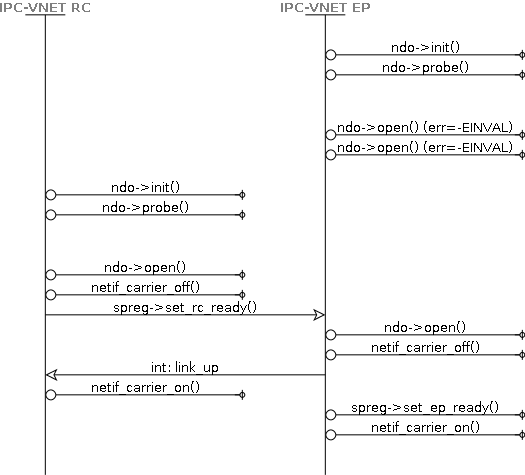
Application communication: The host application and EP board application communicate with each other thru general L2/L3/L7 Linux interface. For example, user can setup a SSH server on EP board and access by SSH client from host.

**Figure 1 IPC-BDMA block overview**



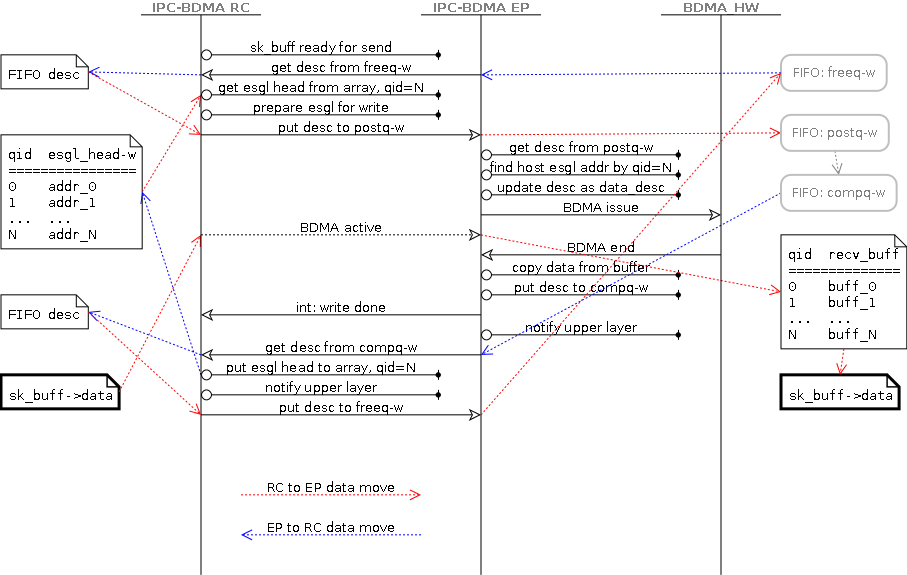
### NIC init/up flow

**Figure 2 Host and EP board virtual NIC initialize flow**



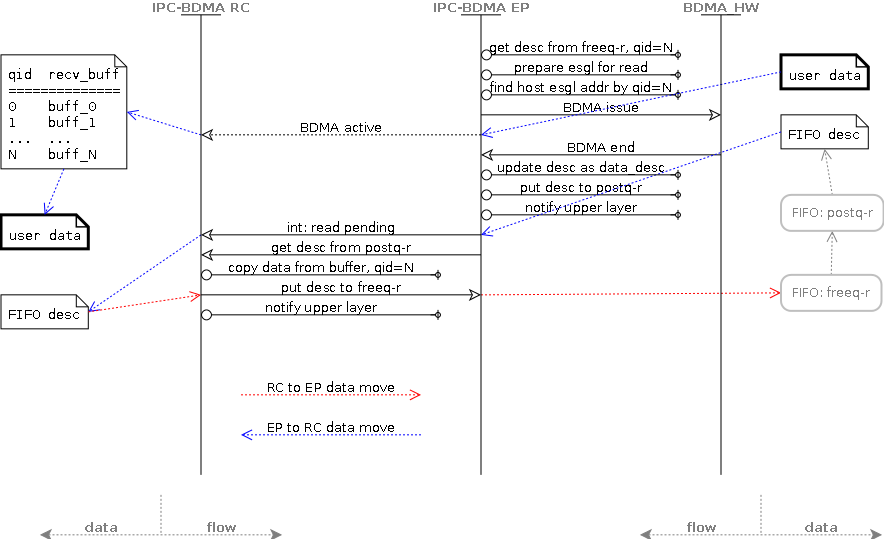
### RC initiate data transmit to EP

**Figure 3 IPC-BDMA RC initiate data transmit to EP**



### EP initiate data transmit to RC

**Figure 4 IPC-BDMA EP initiate data transmit to RC**



## Common FIFO descriptor definition and format

### FIFO descriptor format

The FIFO descriptor is the main message control block between RC and EP, which is accessed with writing to or reading from FIFO registers by RC and EP.

The FIFO descriptor is defined as a 32bit structure.

**Table 1 FIFO descriptor format**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Byte-3** | | | | | | | | **Byte-2** | | | | | | | | **Byte-1** | | | | | | | | **Byte-0** | | | | | | | |
| 1 | **0** | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | **0** | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | **0** | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | **0** |
| T | ID | | | | | | | | | | | CH | | OP | | Param | | | | | | | | | | | | | | | |

**Table 2 FIFO descriptor fields**

|  |  |  |  |
| --- | --- | --- | --- |
| Bit range | Symbol | Name | Description |
| [31] | T | descriptor Type | Type of the descriptor:   * T=0: Data descriptor * T=1: Command descriptor   Refer to <FIFO descriptor type>. |
| [30:20] | ID | ID | Read only field after initialize.  Sequence ID to identify the session.  Refer to <FIFO descriptor session and action>. |
| [19:18] | CH | Channel | Maximum supported IPC channels |
| [17:16] | OP | Operation | Operations according to descriptor type.  Refer to <FIFO data descriptor>, <FIFO command descriptor> for definition. |
| [15:0] | Param | Parameter | Parameter according to descriptor operations.  Refer to <FIFO data descriptor>, <FIFO command descriptor> for definition. |

### FIFO descriptor type

Data descriptor is used for transferring raw data from the upper IPC-BDMA layer without adding any IPC-BDMA header. For example: sending the sk\_buff->data from "PMC virtual netdev RC" to "PMC virtual netdev EP".

* The data descriptor can be queued into FIFO as many as possible. The number is limited by FIFO queue depth and total pre allocated data descriptor numbers.

Command descriptor is generated and handled within IPC-BDMA layer for RC and EP. For example: start or stop the IPC-DMA interface from RC to EP. The command descriptor is invisible to upper layer.

* The command descriptor can be queued into FIFO only one for a specific command. And different commands can be queued together into FIFO. For example: if a "write data request" is sent without response been received, the new "write data request" is not allowed to be issued.

### FIFO descriptor session and action

A complete single FIFO descriptor handling is defined as FIFO session.

A FIFO session contains two actions: Request action and Response action.

* Request action is used for starting a FIFO session from either RC or EP.
* Response action is used for reporting the completion to end a FIFO session.

Each request action from one (RC or EP) shall have a response action from another (EP or RC, correspondingly to request) within a predefined timeslot.

A session timeout for a request means response action cannot be delivered to request side within the predefined timeslot.

The descriptor related session is aborted and the descriptor is reclaimed after session timeout.

## FIFO descriptors

### FIFO data descriptor

Data descriptor is used for pure upper layer data transfer. For example, the sk\_buff->data is transferred by BDMA directly without adding any IPC-BDMA layer headers before and after the BDMA transfer.

### Data descriptor format

**Description**

Data descriptor request is used

**Table 3 FIFO data descriptor request format**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Byte-3** | | | | | | | | **Byte-2** | | | | | | | | **Byte-1** | | | | | | | | **Byte-0** | | | | | | | |
| 1 | **0** | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | **0** | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | **0** | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | **0** |
| 0 | ID | | | | | | | | | | | CH | | 0 | | Length/Status | | | | | | | | | | | | | | | |

### FIFO command descriptor

### Command descriptor format

Command descriptor request does not require BDMA data transfer.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Byte-3** | | | | | | | | **Byte-2** | | | | | | | | **Byte-1** | | | | | | | | **Byte-0** | | | | | | | |
| 1 | **0** | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | **0** | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | **0** | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | **0** |
| 1 | ID | | | | | | | | | | | CH | | OP | | Param/Status | | | | | | | | | | | | | | | |

## Doorbell registers

### Define

Doorbell registers are MSGU control registers for raising MSI/MSIx interrupt from EP board to host thru PCIe interface.

The interrupt is always triggered from EP board by writing predefined control bits in SET register.

The host is responsible to read the bits from the register after MSIx interrupt receives.

Currently four interrupts are defined

* Host write done
  + Notify host that the host's data write has been completed by BDMA
* Host read pending
  + Notify host that the host has data pending to be read
* Link up
  + Notify host that the virtual Ethernet link can goes to link up
* Link down
  + Notify host that the virtual Ethernet link can goes to link down

### Register detail

(Register detail TODO)

## Shared memory thru ScratchPad

### Define

The shared memory is used to setup global configuration by host or EP, so that the peer can understand each other and perform further actions.

Currently following information is defined:

* Host initializing status
  + EP check this during initialization process, refer to initialize flow for detail
* EP initializing status
  + Host check this during initialization process, refer to initialize flow for detail
* Host SGL receive buffer resources
  + Host set predefined SGL receive buffer resource information, so that EP can issue BDMA without requesting information for each data move

### Shared memory define

(TODO)

### ScratchPad define

(TODO)

# EP Side Feature

## Hardware FIFO

Enable 5 new FIFO in EP for RC/EP read/write purpose.

Enable also EP interrupt trigger for "EP interrupt enabled" FIFO.

|  |  |  |  |
| --- | --- | --- | --- |
| FIFO name | EP interrupt enabled | EP access mode | RC access mode |
| freeq-rc-read |  | r,w | - |
| postq-rc-read |  | w | r |
| freeq-rc-write |  | - | r,w |
| postq-rc-write | yes | r | w |
| compq-rc-write |  | w | r |

## Hardware PCIe

The EP needs to enable following PCIe features:

* Interrupt sources (INTx or MSI-X) to RC, which is issued by EP in following conditions:
  + RC write done
    - Used by EP to notify RC that the RC data write operation has finished
    - RC read from compq-rc-write FIFO to get the write response
  + RC read pending
    - Used by EP to notify RC that new data is available for RC read operation
    - RC read from postq-rc-read FIFO to get the read detail
  + RC link up
    - Used by EP to notify RC that the link is up
  + RC link down
    - Used by EP to notify RC that the link is down
* PCIe BAR0, which maps all necessary FIFO addresses for RC access (See FIFO define for all RC used FIFO)
* PCIe BAR1, which maps twelve 64bit scrachpad register for RC/EP internal inter communication such as init/configuration

## Linux Kernel BDMA API

The BDMA API accepts control either from host or EP, then move data between them according to the control information.

The control message and format is defined in "IPC-BDMA API".

## Linux Kernel platform driver

The platform driver is to server the IPC-BDMA API to control the PCIe interface, and send/receive data thru it.

## Linux Kernel IPC-BDMA API/LIB

The IPC-BDMA API behaves like a MAC layer for virtual network driver. Virtual network driver calls such APIs to control BDMA for transmitting/receiving data.

## Linux Kernel netdev driver

The kernel netdev driver implements a virtual Ethernet interface to EP, so that the normal customer network application service can use the interface without any modification.

It also integrate with platform driver and BDMA API for full Ethernet functionality.

## Linux Application

### TCP application

Use nuttcp as TCP application to send/receive network stream.

# RC side Feature

## Linux Kernel PCI driver

The PCIe driver is to server the IPC-BDMA API to send/receive data thru PCIe interface.

## Linux Kernel IPC-BDMA API/LIB

The IPC-BDMA API behaves like a MAC layer for virtual network driver. Virtual network driver calls such APIs to control BDMA for transmitting/receiving data.

## Linux Kernel netdev driver

The kernel netdev driver implements a virtual Ethernet interface to host, so that the normal customer network application can use the interface without any modification.

It also integrate with PCI driver for full Ethernet functionality.

## Linux Application

### TCP application

Use nuttcp as TCP application to send/receive network stream.