An Intelligent NIC Design

Xin Song

School of Electronic and Information Engineering Tianjin Vocational Institute Tianjin, 300410, China
email: lianix@163.com

Keywords: iNIC; PCIe; SR-IOV; TSO; Virtual Machine

Abstract. Intelligent network interface cards are widely deployed in databases, web servers, big data analytics, shared mass storage, virtualized network device, router, and firewall. This paper presents an intelligent NIC design based on Freescale T2080 processor. It is a PCI Express card and provides two SFP+ ports. The design supports SR-IOV and up to 16 virtual machines while offering the benefits of direct I/O throughput, maximizing the potential for using the full bandwidth. The customizable program run on the 8 cores is responsible for policing, classification, forwarding packets and supports TCP segment offload, UDP fragmentation offload, IP fragmentation offload, packet aggregation while improving the throughput and reducing server's CPU overhead. The iNIC can be widely used in various servers to reduce costs, increase network flexibility, scalability and reliability.

Introduction

As cloud computing continues to drive investment in networking, data center infrastructure with network virtualization applications is becoming the fastest growing markets. Cloud service providers are using virtualization technologies such as network function virtualization (NFV) and software-defined networking (SDN) to reduce costs, while increasing network flexibility, scalability and reliability [1]. It provides network connectivity from Virtual Machines (VMs) to the physical network and between Virtual Machines. The network interface card (NIC) with Single Root I/O Virtualization (SR-IOV) allows network traffic to bypass the software switch layer of the Hyper-V virtualization stack, and the virtual machine (VM) to separate access to its resources among PCIe hardware functions [2]. As a result, the I/O overhead in the software emulation layer is diminished and achieves network performance that is nearly the same performance as in nonvirtualized environments [3]. So, SR-IOV is very suitable for use in virtualization network.

With the development of cloud computing, servers provide more and more services, and data traffic is growing faster and faster. So it is important that a NIC can provide high throughput and offload and pre-process functionality which can reduce the server overhead and meet the performance requirement.

The paper presents an intelligent NIC (iNIC) design, which integrates standard NIC function and SR-IOV feature. All the packets will be processed by the customizable multiple threads application in user space. The application can provides various functionalities including TCP/IP checksum, TCP segment offload, UDP fragmentation offload, IP fragmentation offload, packet aggregation, filter, QoS. SR-IOV enables iNIC to supports up to 16 virtual machines. These VMs share the 20G bytes network bandwidth. The iNIC can be widely used in databases, web servers, big data analytics, shared mass storage, virtualized network device, router, and firewall.

Design of the hardware

The iNIC's hardware block diagram is shown in Figure 1. The main chip is Freescale T2080 processor combined 4 dual-threaded e6500 cores for a total of 8 threads with frequencies scaling to 1.8 GHz, integrated 1Gbps and 10Gpbs Ethernet, hardware acceleration and advanced system peripherals [4]. The two fully programmable DDR SDRAM controllers supports DDR3 memories up to 2x8G size.

The design selects a 128M nor flash connected to local bus. It is to store uboot, Linux kernel, root file system and iNIC application.

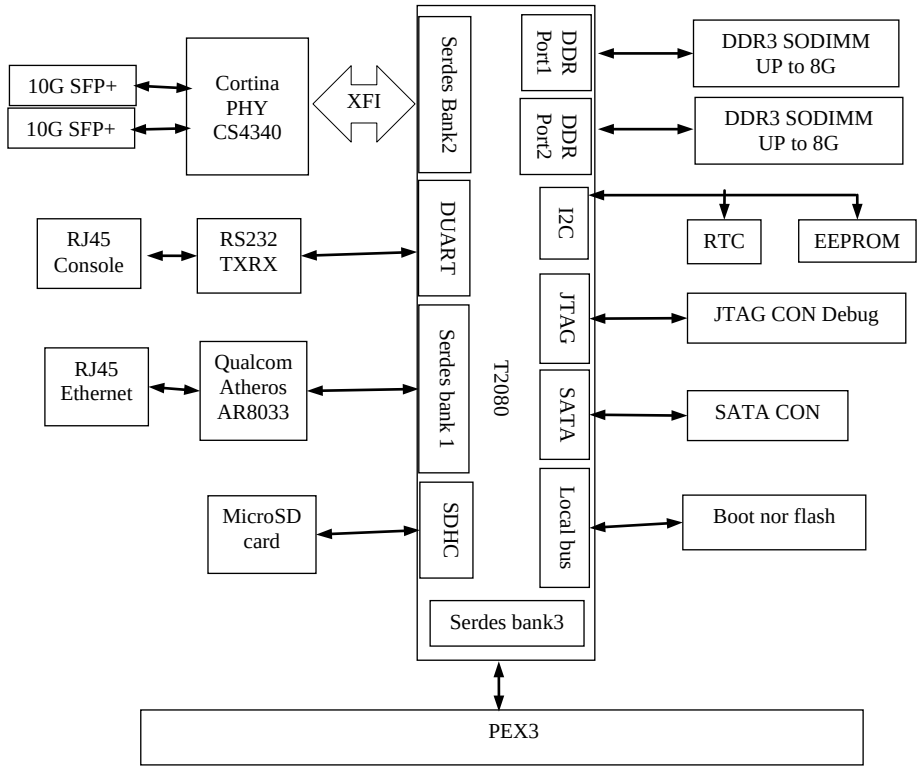

Fig.1. iNIC Block Diagram

The serialization/deserialization (Serdes) is for high speed serial interfaces. Serdes bank1 and bank2 is used for network protocol. Cortina Quad HY CS4340 is selected as PHY connected to Serdes bank2 and provides 2x10G SFP+ port. Serdes bank3 is used for PEX3.

T2080 contains 4 PCI Express controllers (PEX), which can be configured to operate as either a PCI Ex-press root complex (RC) or endpoint (EP) device. The third PCIe controller supports SR-IOV with up to 128 VFs. So the design selects PEX3 as the connection PCIe. PEX3 is assigned 8 lanes achieving 40Gbps bandwidth. Other PEX works in RC mode.

T2080 Data Path Acceleration Architecture (DPAA) provides the infrastructure to support simplified sharing of networking interfaces and accelerators by multiple CPU cores, which consists of Queue Manager (QMan), Buffer Manager's (BMan) and Frame Manager (FMan) [5]. QMan is to manage the queueing of data between multiple processor cores, network interfaces, and hardware accelerators in a multicore SoC. BMan is to manage buffer pools. FMan is process the packets at line rate. The DPAA significantly reduces software overheads associated with high-touch, packet forwarding operations.

Other units such as SATA controller, USB controller and PEX can extend iNIC functionality. For example, inserting hard disk to SATA allows iNIC to provide network attached storage server. Inserting a crypto chip to a PEX allows iNIC offload encryption and decryption operation.

Design of the Software

iNIC application is a multiple threads application in user space. Software Architecture is shown in Figure 2. iNIC software is divided to three layers hardware drivers, libs and apps. Hardware driver layer is located in kernel space including the driver of the main hardware such as DMA, PCIe, FMan, BMan, QMan. The libs layer is in user space, wraps the driver functionality and provides application programming interface. The top layer is APP including VF Ethernet, DPAA Ethernet and main application. The iNIC application is responsible for parsing packet header and processing/forwarding packets. It also manages QoS, filtering, offload functions. Each core will run

a separated a thread. There is no resource shared among the cores, so threads are completely parallel, and the lock-free design improves performance.

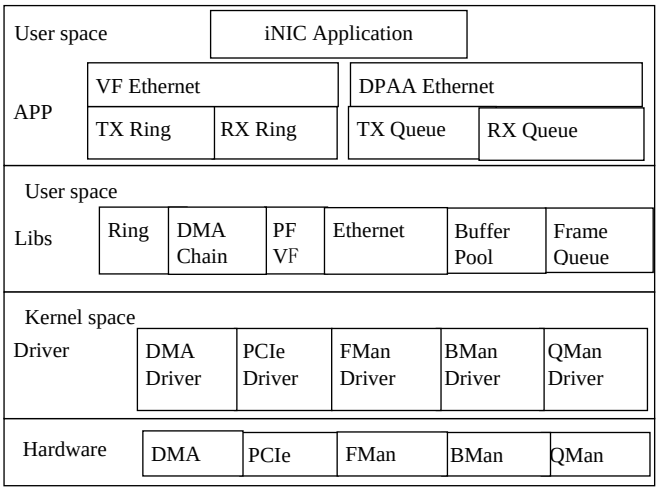

Fig.2. iNIC Software Layer

DMA Memory Design

To store the packets, a large continuous memory is needed. So a DMA memory driver and library is designed to manage the memory. The interfaces to allocate and free DMA memory and convert between the virtual address and physical address are provided for all the modules. A large memory region is reserved for the iNIC DMA memory when kernel initialize the memory and TLB1 will be created to do the map. Access the DMA memory will not encounter the page miss issue, which can improve the performance.

DMA Design

T2080 contains 3 DMA engines, each engine includes 8 DMA channels. The total 24 channels are evenly distributed on the 8 cores to transfer data between iNIC memory and server memory.

DMA channel works in chain mode. The Linux DMA drive based on UIO provides the standard UIO open/map interface for DMA library to access the DMA channel register. DMA library is designed to provide the interface to add/start/free the data transfer. To improve performance, DMA channel works in standard chain mode. The list of DMA descriptor (DD) will be created when the responding channel is assigned to a core. When the core needs to transfer data, the dma_add_start_dd() will be called. An idle DMA descriptor of the list will be updated to describe the new source address, destination address and size. DMA channel will dynamically load the DD and start the transfer. DMA chain library provides a polling mode interface to notify the core that the transfer has been done.

PCIe Design

PCIe driver in the kernel space is a UIO driver while providing the interface for access the controller register. PF/VF library is to enable VFs and PFs; configure the inbound windows, outbound windows and MSI-X trap window. Each function will be assigned different 8G outbound space for IOMMU support, so the VM's memory cannot be greater than 8G. Up to 8 MSI-X interrupts can be triggered. PCIe endpoint exposes the local resources through BARs to the host side [6]. The BAR setting is as follows:

BAR1: 8KB size to store MSI-X table and peddling table.

BAR2: 256kB size to store Ring Descriptor and 521 BD entries.

Ring Design

Ring is a circular linked list used by the DMA, VF Ethernet. So the design abstracts the common ring functions to develop a Ring library. Each ring has three index: head index, current index, tail index. DMA descriptor ring, Ethernet TX ring, RX ring are all the instance of ring. The entries between the head and tail index are idle and wait to use. The entries between the head and current index indicate the data has been processed by the hardware or server and is ready to free. The entries between the current and tail index mean that they are ready and wait to be processed by the hardware.

VF Ethernet Design

VF Ethernet is a virtual Ethernet card instance. When the VF is assigned to serve's VM, the corresponding Ethernet will be created dynamically. The Ethernet will poll the RX/TX rings and receive/transmit the data packets. Each ring contains 256 buffer descriptor (BD) entries. BD is a 32 bytes structure which describes the buffer address, length and command/status. The ring and BD contents are shared between the VM the iNIC. When transmitting the packets, VM Ethernet driver will update the BD and increase consumer index. The VF Ethernet TX function will check the ring index update and parse the corresponding BD, calculate the packet address and length, then call DMA to copy the data from server memory to the iNIC memory, finally push this packet to iNIC for further processing.

DPAA Ethernet Design

DPAA Ethernet includes the software modules of QMan, BMan and FMan. QMan driver is to manage frame queue. QMan library provides queue/dequeuer Frame Descriptor (FD) to/from frame queues. Each Ethernet is assigned 2 transmitting frame queues and 4 receiving frame queues. BMan program is used to manage the buffer pool. There are one pool to store all the receiving packets, one pool for sending packets. Each pools has 8K buffers. FMan is used to receive/transmit the packets.

The Frame Descriptor is the basic element. All the packets through the DPAA Ethernet must be descripted by FD. If walking through the VF Ethernet, the packets should be descripted by BD. The iNIC is responsible for the converting BD and FD according to destination.

DPAA provides Parse-Classify-Police-Distribute (PCD) to process the receiving packets. The DPAA Ethernet driver will configure the PCD to accelerate packets processing.

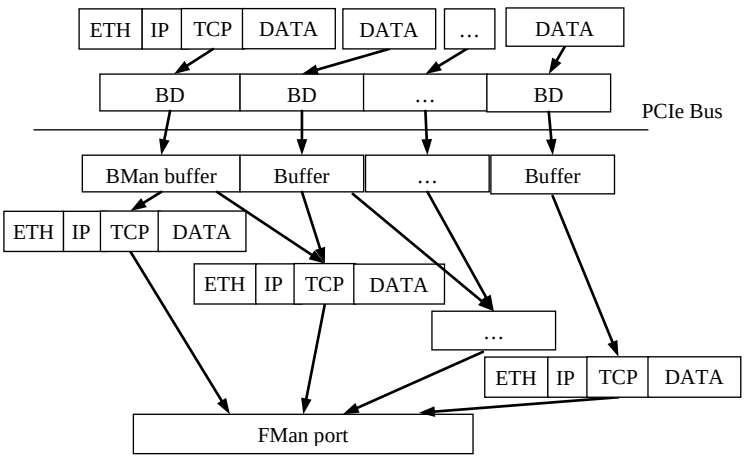

Fig.3. iNIC TCP segment offload Implementation

TCP Segment Offload Design

When a system needs to send large chunks of data out over a computer network, the chunks first need breaking down into smaller segments that can pass through all the network elements like routers and switches between the source and destination computers. This process is referred to as segmentation [7]. The Design can offload this work to reduce CPU overhead and increase the performance.

The implementation is shown in Figure 3. The host CPU can hand over the large packet; the driver will create multiple BD entries to describe this packet; the iNIC will break the data down into smaller segments, add the TCP, IP and data link layer protocol headers to each segment, and send the resulting frames over the network via FMan port. Because the offloading segment is performed by the software run in powerful processors, UDP and other transport layer protocols are supported. The design also supports IP fragmentation offload using the similar implementation.

Packets Aggregation Design

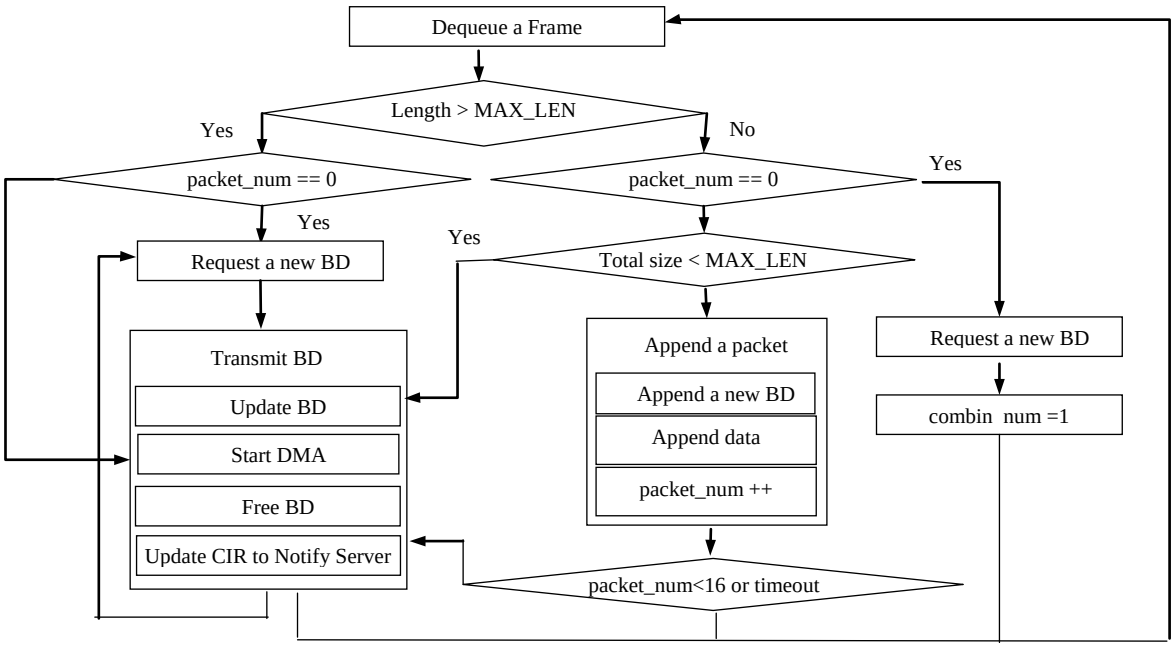

Fig.4. Packets Aggregation Implementation

The design implements the packets aggregation to improve the performance, which is shown in Figure 4. When receiving a packet, the code will check the size and aggregation packet number, if the condition is met, the packet will be appended to the previous packet tail and a new BD will be created to describe it. So multiple packets will be aggregated in the large buffer and will be transmit to the server at one time. But this will increase latency. The design provides a configuration interface to change maximum aggregation packet number and timeout to achieve the balance between the performance and latency.

iNIC Working Flow

The iNIC main thread run in the core 0 is responsible for initializing all the hardware including PCIe, DMA, DMA memory, QMan, FMan, BMan and creating a thread on every core to processing the packets. A command thread also runs in core 0 to process the kernel signal, PCIe command from server side. When VF is assigned to a VM, the corresponding virtual Ethernet will be created in the iNIC side, and the resource such as DMA channel, RX/TX ring, FMan port will be associated with

the Ethernet. When transmitting a packet, the packet will pass through the TX ring, core, DMA, QMan and finally FMan will send out it.

Conclusion

The design uses T2080's DPAA to parse, classify and distribute the packets; use configurable software run in 8 processors to further process the packets; uses PCIe to connect with server; uses SR-IOV to emulate up to 16 virtual Ethernet. It supports two SFP+ ports and intelligent packets processing and provides flexible and reliable Ethernet functionality. It is better suited to cloud network and can be will greatly reduce the data network deployment costs.

References

[1] Open Vswitch for NFV https://wiki.opnfv.org/open_vswitch_for_nfv.

[2] Y. Dong, J. Dai, et al. Towards high-quality I/O virtualization. Proceeding of the Israeli Experimental Systems Conference (SYSTOR), Haifa, Israel 2009.

[3] Y. Dong, et al. SR-IOV Networking in Xen: Architecture, Design and Implementation. 1st Workshop on I/O Virtualization, San Diego, CA, 2008.

[4] T2080 QorIQ Integrated Multicore Communications Processor Family Reference Manual, NXP Semiconductors, 2014, http://www.nxp.com/.

[5] QorIQ T2080 Data Path Acceleration Architecture (DPAA) Reference Manual, NXP Semiconductors, 2014, http://www.nxp.com/.

[6] PCI Special Interest Group, http://www.pcisig.com/home.

[7] Large segment offload, https://en.wikipedia.org/wiki/Large_segment_offload.