Networking:

virtual network device called the E1000 to handle network communication.

the E1000 your driver will talk to is an emulation provided by qemu, connected to a LAN that is also emulated by qemu.

On this LAN, xv6 (the “guest”) has an IP address of 10.0.2.15. (xv6 emulator)

The only other (emulated) computer on the LAN has IP address 10.0.2.2. which is the real computer on which you’re running qemu (the “host”).

when xv6 uses the E1000 to send a packet to 10.0.2.2, it’s really delivered to the appropriate application on the (real) computer on which you’re running qemu (the “host”).

What is PCI Express?

PCI Express is the common motherboard interface for personal computers’ graphics cards, hard drives, SSDs, Wi-Fi and Ethernet hardware connections.

Direct memory mapping for E1000

in pci.c : PCI-Express initialization is done.

// we'll place the e1000 registers at this address.

// vm.c maps this range.

uint64 e1000\_regs = 0x40000000L;

// qemu -machine virt puts PCIe config space here.

// vm.c maps this range.

uint32 \*ecam = (uint32 \*) 0x30000000L;

in vm.c:

#ifdef LAB\_NET

// PCI-E ECAM (configuration space), for pci.c

kvmmap(kpgtbl, 0x30000000L, 0x30000000L, 0x10000000, PTE\_R | PTE\_W);

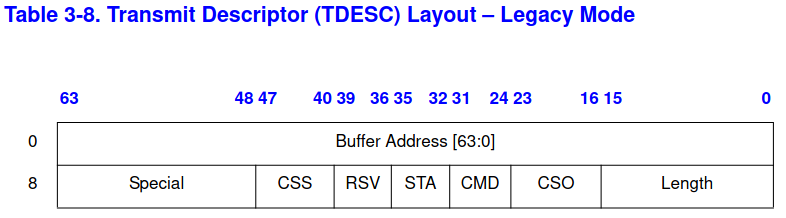
// pci.c maps the e1000's registers here.

kvmmap(kpgtbl, 0x40000000L, 0x40000000L, 0x20000, PTE\_R | PTE\_W);

#endif

3.3 packet transmission;

see in the manual



// [E1000 3.3.3]

struct tx\_desc

{

uint64 addr; //buffer

uint16 length; //the maximum length associated with any single legacy descriptor is 16288 bytes  
 uint8 cso; //check sum offset

uint8 cmd; //3.3.3.1 Transmit Descriptor Command Field Format

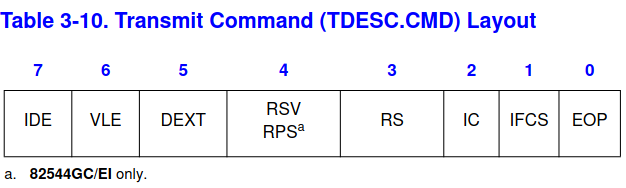
uint8 status; //3.3.3.2 Transmit Descriptor Status Field Format

uint8 css; //check sum start field

uint16 special;

};

3.3.3.1 Transmit Descriptor Command Field Format

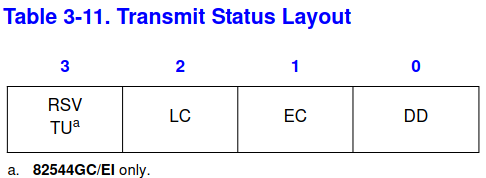


End Of Packet  
When set, indicates the last descriptor making up the packet. One or many  
descriptors can be used to form a packet.

Report Status  
When set, the Ethernet controller needs to report the status information. This ability may be used by software that does in-memory checks of the transmit descriptors to determine which ones are done and packets have been buffered in the transmit FIFO. Software does it by looking at the descriptor status byte and checking the Descriptor Done (DD) bit.

Notes:  
1. VLE (VLAN Packet Enable - bit 6), IFCS (insert FCS - bit 2), and IC (insert checksum - bit 2) are qualified by EOP. That is, hardware interprets these bits ONLY when EOP is set.  
2. Hardware only sets the DD bit for descriptors with RS set.  
3. Descriptors with the null address (0b) or zero length transfer no data. If they have the RS bit set then the DD field in the status word is written when hardware processes them.  
4. Although the transmit interrupt may be delayed, the descriptor write-back requested by setting  
the RS bit is performed without delay unless descriptor write-back bursting is enabled

3.3.3.2 Transmit Descriptor Status Field Format



Descriptor Done  
Indicates that the descriptor is finished and is written back either after the descriptor has been processed (with RS set) or for the 82544GC/EI, after the packet has been transmitted on the wire (with RPS set).

E1000 hardware definitions: registers and DMA ring format.

// from the Intel 82540EP/EM &c manual.

e1000\_dev.h

e1000.c

The e1000\_init() function we provide you in e1000.c configures the E1000 to read packets to be transmitted from RAM, and to write received packets to RAM. This technique is called DMA, for direct memory access, referring to the fact that the E1000 hardware directly writes and reads packets to/from RAM.

/\* Transmit Descriptor command definitions [E1000 3.3.3.1] \*/

#define E1000\_TXD\_CMD\_EOP 0x01 /\* bit 0, End of Packet masking \*/

#define E1000\_TXD\_CMD\_RS 0x08 /\*bit 3, Report Status masking \*/

/\* Transmit Descriptor status definitions [E1000 3.3.3.2] \*/

#define E1000\_TXD\_STAT\_DD 0x00000001 /\* bit 0 Descriptor Done \*/

Note: The DD bit reflects status of all descriptors up to and including the one with the RS bit set (or RPS for the 82544GC/EI).

🡪 the e1000\_init function configures the E1000 to read packets to be transmitted from RAM and to write it to RAM. The technique is called DMA. That means for direct memory access, referring to the fact that the hw directly writes and reads packet to/from RA.

🡪 because bursts of packets might arrive faster , E1000 has buffers. This is array of descriptors in ram. Each descriptor has an address in RAM where the E1000 can write a packet.

struct rx\_desc

{

uint64 addr; /\* Address of the descriptor's data buffer \*/

uint16 length; /\* Length of data DMAed into data buffer \*/

uint16 csum; /\* Packet checksum \*/

uint8 status; /\* Descriptor status \*/

uint8 errors; /\* Descriptor Errors \*/

uint16 special;

};

🡪when the net stack in net.c has to send a packet it calls e1000\_transmit() with mbuf that holds the packet to send.

🡪 tx code must place a pointer to the packet data in descriptor in tx ring struct.

**Rx**

🡪 when e1000 recieves each packet from ethernet, it firsts DMAs the packet to mbuf pointer to by Rx ring descriptor and then generates an interrupt.

🡪rx code scans the rx ring and deliver each new packet mbuf to network stack by calling net\_rx().

This array of descriptors are in circular queue.

The mbuf is the memory buffer in the DMA

rx\_ring – array of structures, our rx fifo mbuf is the structure here

net\_rx is called from e1000\_recv which is in net.c

net\_rx calls net\_rx\_ip which receives the IP packet

net\_rx\_ip calls net\_rx\_udp which validates the headers lengths and payload and parses the necessary fields.

Net\_rx\_udp calls sockrecvudp which is in sysnet.c which delivers UDP packets .. by calling mbufq\_pushtail

Mbufq\_pushtail pushes at the end of the mbufq queue

Whole process

Trap.c 🡪devintr() is called

In devintr() we check for e1000 IRQ and we called e1000\_intr

E1000\_intr will set e1000\_icr to 0xffffffff and call e1000\_recv

<https://xiayingp.gitbook.io/build_a_os/labs/lab-10-networking-part-1>