The Spidernet Device Driver

Written by Linas Vepstas <linas@austin.ibm.com>

Version of 7 June 2007

Abstract

This document sketches the structure of portions of the spidernet device driver in the Linux kernel tree. The spidernet is a gigabit ethernet device built into the Toshiba southbridge commonly used in the SONY Playstation 3 and the IBM QS20 Cell blade.

The Structure of the RX Ring.

The receive (RX) ring is a circular linked list of RX descriptors, together with three pointers into the ring that are used to manage its contents.

The elements of the ring are called "descriptors" or "descrs"; they describe the received data. This includes a pointer to a buffer containing the received data, the buffer size, and various status bits.

There are three primary states that a descriptor can be in: "empty", "full" and "not-in-use". An "empty" or "ready" descriptor is ready to receive data from the hardware. A "full" descriptor has data in it, and is waiting to be emptied and processed by the OS. A "not-in-use" descriptor is neither empty or full; it is simply not ready. It may not even have a data buffer in it, or is otherwise unusable.

During normal operation, on device startup, the OS (specifically, the spidernet device driver) allocates a set of RX descriptors and RX buffers. These are all marked "empty", ready to receive data. This ring is handed off to the hardware, which sequentially fills in the buffers, and marks them "full". The OS follows up, taking the full buffers, processing them, and re-marking them empty.

This filling and emptying is managed by three pointers, the "head" and "tail" pointers, managed by the OS, and a hardware current descriptor pointer (GDACTDPA). The GDACTDPA points at the descr currently being filled. When this descr is filled, the hardware marks it full, and advances the GDACTDPA by one. Thus, when there is flowing RX traffic, every descr behind it should be marked "full", and everything in front of it should be "empty". If the hardware discovers that the current descr is not empty, it will signal an interrupt, and halt processing.

The tail pointer tails or trails the hardware pointer. When the hardware is ahead, the tail pointer will be pointing at a "full" descr. The OS will process this descr, and then mark it "not-in-use", and advance the tail pointer. Thus, when there is flowing RX traffic, all of the descrs in front of the tail pointer should be "full", and all of those behind it should be "not-in-use". When RX traffic is not flowing, then the tail pointer can catch up to the hardware pointer. The OS will then note that the current tail is "empty", and halt processing.

The head pointer (somewhat mis-named) follows after the tail pointer. When traffic is flowing, then the head pointer will be pointing at a "not-in-use" descr. The OS will perform various housekeeping duties on this descr. This includes allocating a new data buffer and dma-mapping it so as to make it visible to the hardware. The OS will then mark the descr as "empty", ready to receive data. Thus, when there is flowing RX traffic, everything in front of the head pointer should be "not-in-use", and everything behind it should be "empty". If no RX traffic is flowing, then the head pointer can catch up to the tail pointer, at which point the OS will notice that the head descr is "empty", and it will halt processing.

Thus, in an idle system, the GDACTDPA, tail and head pointers will all be pointing at the same descr, which should be "empty". All of the other descrs in the ring should be "empty" as well.

The show_rx_chain() routine will print out the locations of the GDACTDPA, tail and head pointers. It will also summarize the contents of the ring, starting at the tail pointer, and listing the status of the descrs that follow.

A typical example of the output, for a nearly idle system, might be:

```
net eth1: Total number of descrs=256
net eth1: Chain tail located at descr=20
net eth1: Chain head is at 20
net eth1: HW curr desc (GDACTDPA) is at 21
net eth1: Have 1 descrs with stat=x40800101
net eth1: HW next desc (GDACNEXTDA) is at 22
net eth1: Last 255 descrs with stat=xa0800000
```

In the above, the hardware has filled in one descr, number 20. Both head and tail are pointing at 20, because it has not yet been emptied. Meanwhile, hw is pointing at 21, which is free.

The "Have nnn decrs" refers to the descr starting at the tail: in this case, nnn=1 descr, starting at descr 20. The "Last nnn descrs" refers to all of the rest of the descrs, from the last status change. The "nnn" is a count of how many descrs have exactly the same status.

The status x4... corresponds to "full" and status xa... corresponds to "empty". The actual value printed is RXCOMST_A.

In the device driver source code, a different set of names are used for these same concepts, so that:

```
"empty" == SPIDER_NET_DESCR_CARDOWNED == 0xa
```

```
"full"  == SPIDER_NET_DESCR_FRAME_END == 0x4
"not in use" == SPIDER_NET_DESCR_NOT_IN_USE == 0xf
```

The RX RAM full bug/feature

As long as the OS can empty out the RX buffers at a rate faster than the hardware can fill them, there is no problem. If, for some reason, the OS fails to empty the RX ring fast enough, the hardware GDACTDPA pointer will catch up to the head, notice the not-empty condition, ad stop. However, RX packets may still continue arriving on the wire. The spidernet chip can save some limited number of these in local RAM. When this local ram fills up, the spider chip will issue an interrupt indicating this (GHIINT0STS will show ERRINT, and the GRMFLLINT bit will be set in GHIINT1STS). When the RX ram full condition occurs, a certain bug/feature is triggered that has to be specially handled. This section describes the special handling for this condition.

When the OS finally has a chance to run, it will empty out the RX ring. In particular, it will clear the descriptor on which the hardware had stopped. However, once the hardware has decided that a certain descriptor is invalid, it will not restart at that descriptor; instead it will restart at the next descr. This potentially will lead to a deadlock condition, as the tail pointer will be pointing at this descr, which, from the OS point of view, is empty; the OS will be waiting for this descr to be filled. However, the hardware has skipped this descr, and is filling the next descrs. Since the OS doesn't see this, there is a potential deadlock, with the OS waiting for one descr to fill, while the hardware is waiting for a different set of descrs to become empty.

A call to show_rx_chain() at this point indicates the nature of the problem. A typical print when the network is hung shows the following:

```
net eth1: Spider RX RAM full, incoming packets might be discarded!
net eth1: Total number of descrs=256
net eth1: Chain tail located at descr=255
net eth1: Chain head is at 255
net eth1: HW curr desc (GDACTDPA) is at 0
net eth1: Have 1 descrs with stat=xa0800000
net eth1: HW next desc (GDACNEXTDA) is at 1
net eth1: Have 127 descrs with stat=x40800101
net eth1: Have 1 descrs with stat=x40800001
net eth1: Have 126 descrs with stat=x40800101
net eth1: Last 1 descrs with stat=xa0800000
```

Both the tail and head pointers are pointing at descr 255, which is marked xa... which is "empty". Thus, from the OS point of view, there is nothing to be done. In particular, there is the implicit assumption that everything in front of the "empty" descr must surely also be empty, as explained in the last section. The OS is waiting for descr 255 to become non-empty, which, in this case, will never happen.

The HW pointer is at descr 0. This descr is marked 0x4.. or "full". Since its already full, the hardware can do nothing more, and thus has halted processing. Notice that descrs 0 through 254 are all marked "full", while descr 254 and 255 are empty. (The "Last 1 descrs" is descr 254, since tail was at 255.) Thus, the system is deadlocked, and there can be no forward progress; the OS thinks there's nothing to do, and the hardware has nowhere to put incoming data.

This bug/feature is worked around with the spider_net_resync_head_ptr() routine. When the driver receives RX interrupts, but an examination of the RX chain seems to show it is empty, then it is probable that the hardware has skipped a descr or two (sometimes dozens under heavy network conditions). The spider_net_resync_head_ptr() subroutine will search the ring for the next full descr, and the driver will resume operations there. Since this will leave "holes" in the ring, there is also a spider_net_resync_tail_ptr() that will skip over such holes.

As of this writing, the spider_net_resync() strategy seems to work very well, even under heavy network loads.

The TX ring

The TX ring uses a low-watermark interrupt scheme to make sure that the TX queue is appropriately serviced for large packet sizes.

For packet sizes greater than about 1KBytes, the kernel can fill the TX ring quicker than the device can drain it. Once the ring is full, the netdev is stopped. When there is room in the ring, the netdev needs to be reawakened, so that more TX packets are placed in the ring. The hardware can empty the ring about four times per jiffy, so its not appropriate to wait for the poll routine to refill, since the poll routine runs only once per jiffy. The low-watermark mechanism marks a descr about 1/4th of the way from the bottom of the queue, so that an interrupt is generated when the descr is processed. This interrupt wakes up the netdev, which can then refill the queue. For large packets, this mechanism generates a relatively small number of interrupts, about 1K/sec. For smaller packets, this will drop to zero interrupts, as the hardware can empty the queue faster than the kernel can fill it.