RTL8139(A/B) Programming guide: (V0.1)

1: Packet Transmission

(1). Architecture:

The transmit path of RTL8139(A/B) use 4 descriptors, each descriptor has a fixed IO address offset. The 4 descriptor is used in round-robin. As descriptor is written, PCI operation start and move packet in the memory which the descriptor specify to Transmit FIFO. Transmit FIFO is a 2k bytes buffer in the chip that hold the data prepared to move to line(cable). Data in Transmit FIFO start move to line when early transmit threshold is meet, Early transmit threshold is also specified in the descriptor.

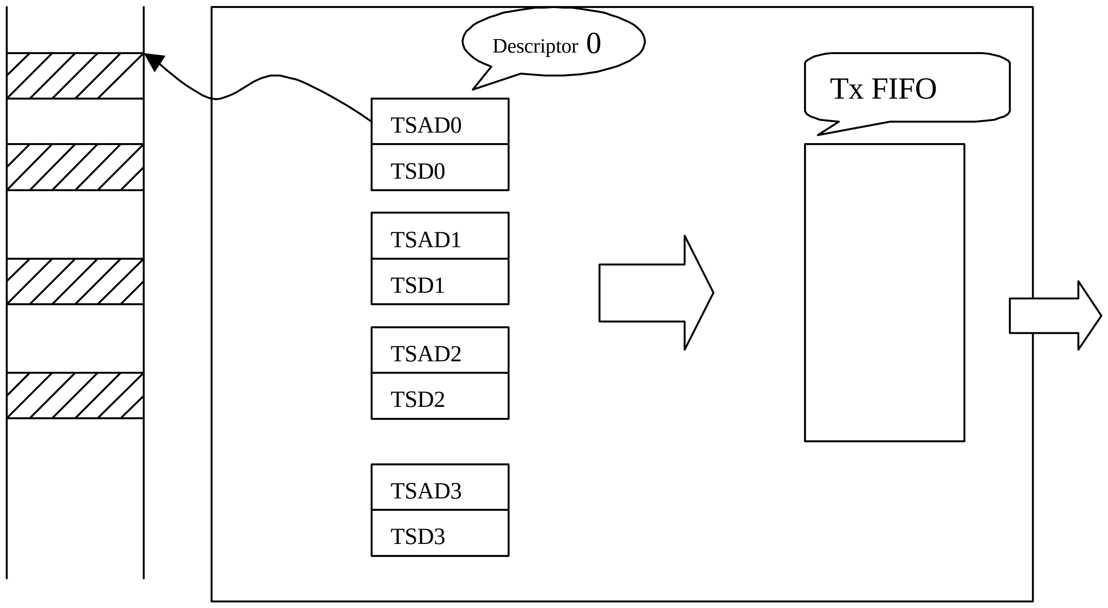
A transmit descriptor consist of 2 registers:

1: Transmit start address(TSAD0-3): the physical address of packet(Note: the packet must be in a continuous physical memory)

2: Transmit status(TSD0-3):

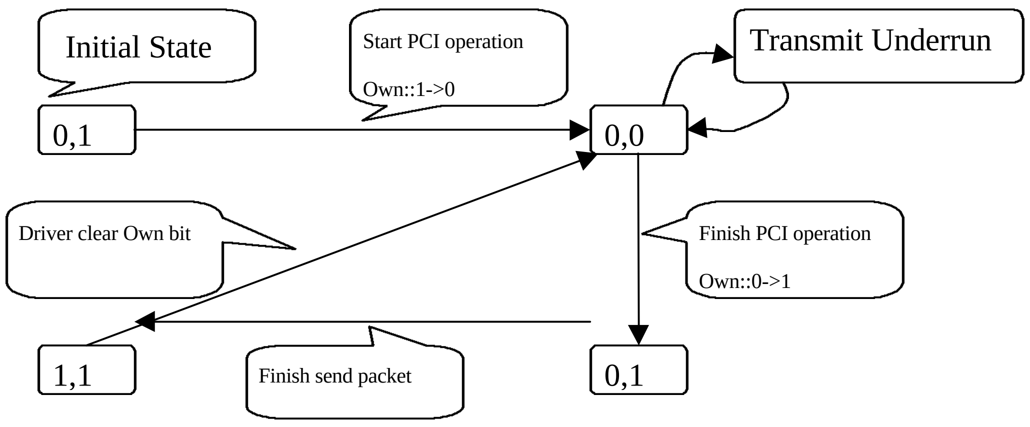
Bit	R/W	Symbol	Description
31	R	CRS	Carrier Sense Lost: Set to 1 when the carrier is lost during transmitting a packet.
30	R	TABT	Transmit Abort: Set to 1 if the transmission of a packet was aborted. This bit is read only, writing to this bit is not affected.
29	R	OWC	Out of Window Collision: Set to 1 if the RTL8139(A/B) encountered an "out of window" collision during the transmission of a packet.
28	R	CDH	CD Heart Beat: The same as RTL8029(AS). This bit is cleared in the 100Mbps mode.
27-24	R	NCC3-0	Number of Collision Count: Indicates that the number of collisions encountered during the transmission of a packet.
23-22	-	-	Reserved
21-16	R/W	ERTXTH5-0	Early Tx Threshold: Specifies the threshold level in the Tx FIFO to begin the transmission. When the byte count of the data in the Tx FIFO reaches this level, (or the FIFO contains at least one complete packet) the RTL8139(A/B) will transmit this packet. 000000 = 8 bytes These fields count from 000001 to 111111 in unit of 32 bytes. This threshold must be avoided from exceeding 2K byte.
15	R	TOK	Transmit OK: Set to 1 indicates that the transmission of a packet was completed successfully and no transmit underrun occurs.
14	R	TUN	Transmit FIFO Underrun: Set to 1 if the Tx FIFO was exhausted during the transmission of a packet. The RTL8139(A/B) can re-transfer data if the Tx FIFO underruns and can also transmit the packet to the wire successfully even though the Tx FIFO underruns. That is, when TSD<TUN>=1, TSD<TOK>=0 and ISR<TOK>=1 (or ISR<TER>=1).

13	R/W	OWN	OWN: The RTL8139(A/B) sets this bit to 1 when the Tx DMA operation of this descriptor was completed. The driver must set this bit to 0 when the Transmit Byte Count (bit0-12) is written. The default value is 1.
12-0	R/W	SIZE	Descriptor Size: The total size in bytes of the data in this descriptor. If the packet length is more than 1792 byte (0700h), the Tx queue will be invalid, i.e. the next descriptor will be written only after the OWN bit of that long packet's descriptor has been set.



The process of transmitting a packet:

1: copy the packet to a physically continuous buffer in memory.
2: Write the descriptor which is functioning
   (1). Fill in Start Address(physical address) of this buffer.
   (2). Fill in Transmit Status: the size of this packet, the early transmit threshold, Clear OWN bit in TSD (this starts the PCI operation).
3: As the number of data moved to FIFO meet early transmit threshold, the chip start to move data from FIFO to line..
4: When the whole packet is moved to FIFO, the OWN bit is set to 1.
5: When the whole packet is moved to line, the TOK(in TSD) is set to 1.
6: If TOK(IMR) is set to 1 and TOK(ISR) is set then a interrupt is triggered.
7: Interrupt service routine called, driver should clear TOK(ISR) State Diagram: (TOK,OWN)



(2). Register involved:

. TSAD0-3
. TSD0-3
. ISR(TOK,TER),IMR(TOK,TER)
. TCR: transmit configuration register
. TSAD: reflect the corresponding bits in the TSD0-3.

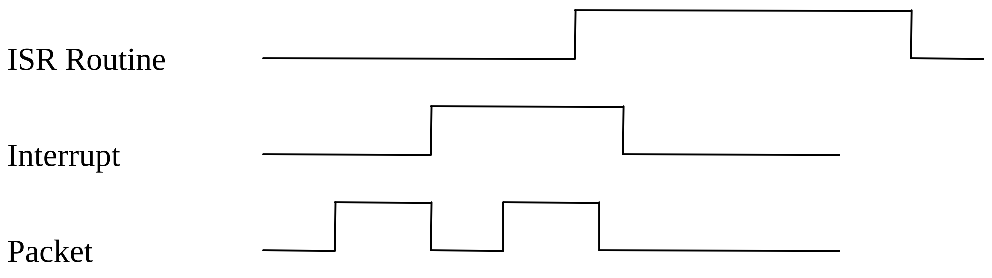
(3). Software Issue:

(1). Interrupt handling:

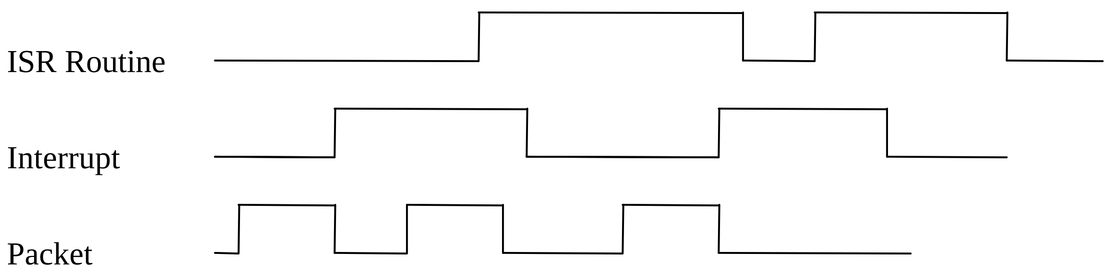
When driver is processing transmit interrupt, the following two cases should be taken care of:

Case 1: More than one interrupt between TOK and when ISR routine called.

=>Drivers have to check as many descriptor as possible.



Case 2: No packet TOK need to be handled, but ISR routine called.



(4). Configuration:

MXDMA: Max DMA burst size per TX DMA burst:
Suggest value: 1024 byte.

(5). Sample code:

```
unsigned char
NextDesc(
        unsigned char CurrentDescriptor
          )
{
//            (CurrentDescriptor == TX_SW_BUFFER_NUM-1) ? 0 : (1 + CurrentDescriptor);
          if(CurrentDescriptor == TX_SW_BUFFER_NUM-1)
          {
        return      0;
          }
          else
          {
        return ( 1 + CurrentDescriptor);
          }
}


unsigned char
CheckTSDStatus(
          unsigned char                    Desc
          )
{
          ULONG                 Offset = Desc << 2;
          ULONG                 tmpTSD;
```

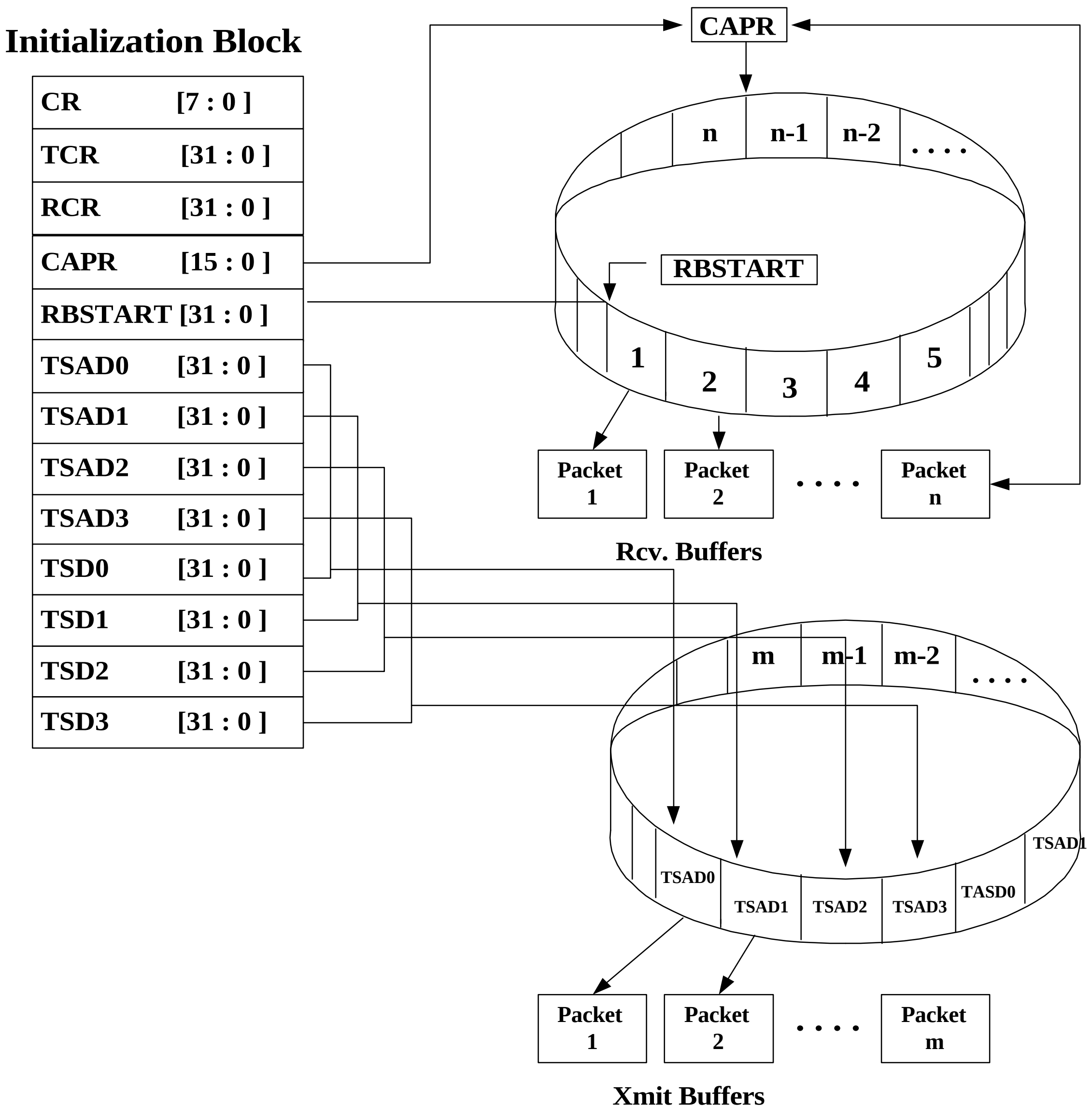
```
     tmpTSD=inpdw(IOBase + TSD0 + Offset);
     switch ( tmpTSD & (TSD_OWN | TSD_TOK) )
     {
    case (TSD_OWN | TSD_TOK):                return   TSDSTATUS_BOTH;
    case (TSD_TOK)          :          return    TSDSTATUS_TOK;
    case (TSD_OWN)          :          return    TSDSTATUS_OWN;
    case 0           :      return     TSDSTATUS_0;
     }
     return 0;
}

void
IssueCMD(unsigned char descriptor)
{
    unsigned long offset = descriptor << 2;
    outpdw(IOBase + TSAD0 + offset, TxDesc[TxHwSetupPtr].PhysicalAddress);
    outpdw(IOBase + TSD0 + offset , TxDesc[TxHwSetupPtr].PacketLength);
}

int
SendPacket(
    PPACKET pPacket
)
{
     disable();
     if( TxHwFreeDesc>0      )
     {
    TxDesc[TxHwSetupPtr].PacketLength=
        CopyFromPacketToBuffer( pPacket , TxDesc[TxHwSetupPtr].buffer);
    IssueCMD(TxHwSetupPtr);
    TxHwSetupPtr = NextDesc(TxHwSetupPtr);
    TxHwFreeDesc--;
    enable();
    return TRUE;//success
     }
     else
```

```
            {
        enable();
        return FALSE;//out of resource
            }
}

void
TxInterruptHandler()
{
            while( (CheckTSDStatus(TxHwFinishPtr) == TSDSTATUS_BOTH          ) &&
                  (TxHwFreeDesc < 4                                 )          )
            {
        //can Release this buffer now

        TxHwFinishPtr = NextDesc(TxHwFinishPtr);
        TxHwFreeDesc++;
            }
}
```

2. Packet Reception.

(1). Architecture:

The receive path of RTL8139(A/B) is designed as a ring buffer. This ring buffer is in a physical continuous memory. Data coming from line is first stored in a Receive FIFO in the chip, and then move to the receive buffer when the early receive threshold is meet. The register CBA keeps the current address of data moved to buffer. CAPR is the read pointer which keeps the address of data that driver had read. The status of receiving a packet is stored in front of the packet(packet header).



Initialization Block
CR [7 : 0]
TCR [31 : 0]
RCR [31 : 0]
CAPR [15 : 0]
RBSTART [31 : 0]
TSAD0 [31 : 0]
TSAD1 [31 : 0]
TSAD2 [31 : 0]
TSAD3 [31 : 0]
TSD0 [31 : 0]
TSD1 [31 : 0]
TSD2 [31 : 0]
TSD3 [31 : 0]
CAPR
n
n-1
n-2
. . . .
RBSTART
1
2
3
4
5
Packet 1
Packet 2
. . . .
Packet n
Rcv. Buffers
m
m-1
m-2
. . . .
TSAD0
TSAD1
TSAD2
TSAD3
TASD0
TSAD1
Packet 1
Packet 2
. . . .
Packet m
Xmit Buffers

The Packet Header:

Bit	R/W	Symbol	Description
15	R	MAR	Multicast Address Received: Set to 1 indicates that a multicast packet is received.
14	R	PAM	Physical Address Matched: Set to 1 indicates that the destination address of this packet matches the value written in ID registers.
13	R	BAR	Broadcast Address Received: Set to 1 indicates that a broadcast packet is received. BAR, MAR bit will not be set simultaneously.
12-6	-	-	Reserved
5	R	ISE	Invalid Symbol Error: (100BASE-TX only) An invalid symbol was encountered during the reception of this packet if this bit set to 1.
4	R	RUNT	Runt Packet Received: Set to 1 indicates that the received packet length is smaller than 64 bytes (i.e. media header + data + CRC < 64 bytes)
3	R	LONG	Long Packet: Set to 1 indicates that the size of the received packet exceeds 4k bytes.
2	R	CRC	CRC Error: When set, indicates that a CRC error occurred on the received packet.
1	R	FAE	Frame Alignment Error: When set, indicates that a frame alignment error occurred on this received packet.
0	R	ROK	Receive OK: When set, indicates that a good packet is received.

The process of packet receive:

1. Data received from line is stored in the receive FIFO.
2. When Early Receive Threshold is meet, data is moved from FIFO to Receive Buffer.
3. After the whole packet is moved from FIFO to Receive Buffer, the receive packet header(receive status and packet length) is written in front of the packet. CBA is updated to the end of the packet.
4. CMD(BufferEmpty) and ISR(TOK) set.
5. ISR routine called and then driver clear ISR(TOK) and update CAPR.

(2). Registers involved:

RBStart: Receive Buffer start address.
CR(BufferEmpty): Indicate if driver is empty.
CAPR: Buffer read pointer.
CBP: Buffer write pointer.
ISR/IMR(ROK, RER, RxOverflow, RxFIFOOverflow)
RCR: Receive configuration register.
(Packet Header.)

(3). Software Issue:

1. Receive Buffer Overflow handling:
   The Rx DMA(FIFO to buffer) is stopped, The CAPR must be updated first to dismiss

the ISR(RxBufferOverflow) event.. The correct actions to process RxBufOvw is:
(a). Update CAPR.
(b). write a '1' to ISR(ROK).
The Rx DMA resumes after step (b)

2. RxFIFOOvw handling:
When RxFIFOOvw occurs, all incoming packets are discarded. Clear ISR(RxFIFOOvw) doesn't dismiss RxFIFOOvw event. To dismiss RxFIFOOvw event, the ISR(RxBufOvw) must be written with a '1'.

3. Rx FIFO early threshold:
No early(FIFO->Buffer DMA start when the whole packet is in FIFO). If a incoming packet is larger than the size of FIFO(2K), RxFIFOOvw will Be set, but Rx DMA will never start, then the receive path is disabled.
=>Never set Rx FIFO early threshold to NoEarly.

4. suggested handling:

```
if (RxFIFOOvw | RxBufOvw | ROK)
{
        clear ISR(RxFIFOOvw | RxBufOvw | ROK)
}
if (ROK)
{
        while(BufEmpty=0)
        {
                read one packet then update CAPR
        }
}
```

(4). Configuration:

1. MXDMA: Max DMA burst size per Rx DMA burst:
   Suggest value:1024.
2. WRAP: when enabled, RTL8139(A/B) will move the rest of the packet data Immediately after the buffer, this will make the last packet in the Buffer continuous. But the Receive buffer have to leave 1.5k more space for this packet.

(5). Sample Code:

```
BOOLEAN
PacketOK(
        PPACKETHEADER pPktHdr
)
{
          BOOLEAN BadPacket = pPktHdr->RUNT ||
                        pPktHdr->LONG ||
                        pPktHdr->CRC      ||
                        pPktHdr->FAE;
          if( ( !BadPacket )         &&
        ( pPktHdr->ROK )          )
          {
        if ( (pPktHdr->PacketLength > RX_MAX_PACKET_LENGTH ) ||
                    (pPktHdr->PacketLength < RX_MIN_PACKET_LENGTH )              )
        {
                return(FALSE);
        }
        PacketReceivedGood++;
        ByteReceived += pPktHdr->PacketLength;
        return TRUE ;
          }
          else
          {
        return FALSE;
          }
}

BOOLEAN
RxInterruptHandler(
          )
{
          unsigned char       TmpCMD;
          unsigned int          PktLength;
          unsigned char       *pIncomePacket, *RxReadPtr;
          PPACKETHEADER       pPacketHeader;
```

```
   while (TRUE)
   {
 TmpCMD = inportb(IOBase + CR);
 if (TmpCMD & CR_BUFE)
 {
      break;
 }

 do
 {
      RxReadPtr        = RxBuffer + RxReadPtrOffset;
      pPacketHeader = (PPACKETHEADER)      RxReadPtr;
      pIncomePacket = RxReadPtr + 4;
      PktLength        = pPacketHeader->PacketLength;        //this length include CRC
      if ( PacketOK( pPacketHeader ) )
      {
    if ( (RxReadPtrOffset + PktLength) > RX_BUFFER_SIZE )
    {                  //wrap around to end of RxBuffer
          memcpy( RxBuffer + RX_BUFFER_SIZE , RxBuffer,
                (RxReadPtrOffset + PktLength - RX_BUFFER_SIZE)      );
    }
    //copy the packet out here
    CopyPacket(pIncomePacket,PktLength - 4);//don't copy 4 bytes CRC

    //update Read Pointer
    RxReadPtrOffset = (RxReadPtrOffset + PktLength + 4 + 3) & RX_READ_POINTER_MASK;
          //4:for header length(PktLength include 4 bytes CRC)
          //3:for dword alignment
    outport( IOBase + CAPR, RxReadPtrOffset - 0x10);  //-4:avoid overflow
      }
      else
      {
//          ResetRx();
    break;
      }
      TmpCMD = inportb(IOBase + CR);
```

```
        } while (!(TmpCMD & CR_BUFE));
        }
          return (TRUE);                                //Done
}
```

3: Initialization

(1). Initialization procedure

1. Enable Transmit/Receive(RE/TE in CommandRegister)
2. configure TCR/RCR.
3. Enable IMR.

(2). Transmit reset and Receive reset can be done individually.

Appendix

(A). Compiling and tracing sample code

. This sample code is developed under Borland C 3.0, and the debugging process is under Softice for DOS. All testing is under DOS(win98).

. To enable source code debugging under Softice, the compiling/linking process need to generate a '.map. file. Softice provide a 'msym' program to translate '.map' file to '.sym' file. After the '.sym' file is generated. Load the demo program by 'Ldr demo'.