[Notes]
RX Family
Ethernet Module Firmware Integration Technology
RX Driver Package

R20TS0481EJ0100
Rev.1.00
Oct 01, 2019

Outline

When using the products in the title, note the following point.

1. When calling the “R_ETHER_Read_ZC2_BufRelease” function or “R_ETHER_Read” function
2. When calling the “R_ETHER_Write_ZC2_SetBuf” function or “R_ETHER_Write” function

1. When Calling the “R_ETHER_Read_ZC2_BufRelease” Function or “R_ETHER_Read” Function

1.1 Applicable Products

These functions are provided in the r_ether_rx_v*.**.zip (*.** is a revision number) file included in the products listed in (1) and (2) below.

(1) RX family Ethernet module using Firmware Integration Technology
(Ethernet FIT module)

The applicable revision numbers and document numbers are as follows.

Table 1.1 Ethernet FIT module applicable products

Ethernet FIT module revision number	Document number
Rev.1.00	R01AN2009EJ0100
Rev.1.01	R01AN2009EJ0101
Rev.1.02	R01AN2009EJ0102
Rev.1.10	R01AN2009EJ0110
Rev.1.11	R01AN2009EJ0111
Rev.1.12	R01AN2009EJ0112
Rev.1.13	R01AN2009EJ0113
Rev.1.14	R01AN2009EJ0114
Rev.1.15	R01AN2009EJ0115
Rev.1.16	R01AN2009EJ0116
Rev.1.17	R01AN2009EJ0117

(2) RX Driver Package

The Ethernet FIT module in (1) is also included in the RX Driver Package listed below.
The product names and revision numbers of the applicable RX Driver Package and the revision numbers of the Ethernet FIT module are as follows.

Table 1.2 Products including Ethernet FIT module

RX Driver Package product name	RX Driver Package revision	Document number	Revision of the included Ethernet FIT module
RX64M Group RX Driver Package User's Manual	Rev.1.01	R01AN2460EJ0101	Rev.1.00
RX64M, RX71M Group RX Driver Package Ver.1.02	Rev.1.04	R01AN2606EJ0104	Rev.1.02
RX Family Driver Package Ver.1.10	Rev.1.10	R01AN3345EJ0100	Rev.1.10
RX Family Driver Package Ver.1.11	Rev.1.11	R01AN3467EJ0111	Rev.1.11
RX Family Driver Package Ver.1.12	Rev.1.12	R01AN3651EJ0112	Rev.1.12
RX Family Driver Package Ver.1.13	Rev.1.13	R01AN3859EJ0113	Rev.1.13
RX Family Driver Package Ver.1.14	Rev.1.14	R01AN4191EJ0114	Rev.1.14
RX Family Driver Package Ver.1.15	Rev.1.15	R01AN4372EJ0115	Rev.1.15
RX Family Driver Package Ver.1.16	Rev.1.16	R01AN4471EJ0116	Rev.1.15
RX Family Driver Package Ver.1.18	Rev.1.18	R01AN4659EJ0118	Rev.1.15
RX Family Driver Package Ver.1.19	Rev.1.19	R01AN4677EJ0119	Rev.1.15
RX Family Driver Package Ver.1.20	Rev.1.20	R01AN4794EJ0120	Rev.1.16
RX Family Driver Package Ver.1.22	Rev.1.22	R01AN4873EJ0122	Rev.1.17

1.2 Applicable Devices

RX63N, RX65N, RX64M, RX71M and RX72M groups

1.3 Details and Conditions

If the number of receive descriptors is set to 1, Ethernet frames may not be received after the "R_ETHER_Read_ZC2_BufRelease" function(Note) or "R_ETHER_Read" function releases the buffer.

* The "R_ETHER_Read_ZC2_BufRelease" function is a lower function of the "R_ETHER_Read" function.

- Examples

If the EDMAC sets the RR bit in the EDRRR register to 0 to stop the receive function in coding at (i) and later in the "R_ETHER_Read_ZC2_BufRelease" function, Ethernet frames cannot be received because the Ethernet FIT module cannot resume reception.

```
 int32_t R_ETHER_Read_ZC2_BufRelease (uint32_t channel)
 {
--------------------------- (omitted) ---------------------------
            papp_rx_desc[channel]->status &= (~status);
            papp_rx_desc[channel]->status |= RACT;
            papp_rx_desc[channel] = papp_rx_desc[channel]->next;
        }
        pether_ch = g_eth_control_ch[channel].pether_control;
        phy_access = g_eth_control_ch[channel].phy_access;
        pedmac_adr = pether_ch[phy_access].pedmac;

        if (0x00000000L == pedmac_adr->EDRRR.LONG)   (i)
        {
            /* Restart if stopped */
            pedmac_adr->EDRRR.LONG = 0x00000001L;
        }

      ret = ETHER_SUCCESS;
--------------------------- (omitted) ---------------------------
 } /* End of function R_ETHER_Read_ZC2_BufRelease() */
```

1.4 Workaround

Add the processing in red in the "R_ETHER_Read_ZC2" and "R_ETHER_Read_ZC2_BufRelease" functions.

Before modification: “R_ETHER_Read_ZC2”

```
 int32_t R_ETHER_Read_ZC2 (uint32_t channel, void **pbuf)
 {
    int32_t num_recvd;
    int32_t ret;
    int32_t complete_flag;
    int32_t ret2;
-------------------------- (omitted) --------------------------
        ret = ETHER_NO_DATA;
        complete_flag = ETHER_ERR_OTHER;
        while (ETHER_SUCCESS != complete_flag)
        {
            /* When receive data exists. */
            if (RACT != (papp_rx_desc[channel]->status & RACT))
            {
-------------------------- (omitted) --------------------------
} /* End of function R_ETHER_Read_ZC2() */
```

After modification: “R_ETHER_Read_ZC2”

```
  int32_t R_ETHER_Read_ZC2 (uint32_t channel, void **pbuf)
  {
      int32_t num_recvd;
      int32_t ret;
      int32_t complete_flag;
      int32_t ret2;
  #if (1 == ETHER_CFG_EMAC_RX_DESCRIPTORS)
      volatile struct st_edmac __evenaccess * pedmac_adr;
      const ether_control_t * pether_ch;
      uint32_t phy_access;
  #endif
--------------------------- (omitted) ---------------------------
          ret = ETHER_NO_DATA;
          complete_flag = ETHER_ERR_OTHER;

  #if (1 == ETHER_CFG_EMAC_RX_DESCRIPTORS)
          pether_ch = g_eth_control_ch[channel].pether_control;
          phy_access = g_eth_control_ch[channel].phy_access;
          pedmac_adr = pether_ch[phy_access].pedmac;
  #endif

          while (ETHER_SUCCESS != complete_flag)
          {
              /* When receive data exists. */
  #if (1 == ETHER_CFG_EMAC_RX_DESCRIPTORS)
              if (0x00000000L == pedmac_adr->EDRRR.LONG)
  #else
              if (RACT != (papp_rx_desc[channel]->status & RACT))
  #endif
            {
--------------------------- (omitted) ---------------------------
} /* End of function R_ETHER_Read_ZC2() */
```

Before modification: “R_ETHER_Read_ZC2_BufRelease”

```
  int32_t R_ETHER_Read_ZC2_BufRelease (uint32_t channel)
  {
--------------------------- (omitted) ---------------------------
     /* When the Link up processing is completed */
     else
     {
         /* When receive data exists */
         if (RACT != (papp_rx_desc[channel]->status & RACT))
         {
--------------------------- (omitted) ---------------------------
} /* End of function R_ETHER_Read_ZC2_BufRelease() */
```

After modification: “R_ETHER_Read_ZC2_BufRelease”

```
  int32_t R_ETHER_Read_ZC2_BufRelease (uint32_t channel)
  {
--------------------------- (omitted) ---------------------------
      /* When the Link up processing is completed */
      else
      {
  #if (1 == ETHER_CFG_EMAC_RX_DESCRIPTORS)
          pether_ch = g_eth_control_ch[channel].pether_control;
          phy_access = g_eth_control_ch[channel].phy_access;
          pedmac_adr = pether_ch[phy_access].pedmac;
  #endif
          /* When receive data exists */
  #if (1 == ETHER_CFG_EMAC_RX_DESCRIPTORS)
          if (0x00000000L == pedmac_adr->EDRRR.LONG)
  #else
          if (RACT != (papp_rx_desc[channel]->status & RACT))
  #endif
          {
--------------------------- (omitted) ---------------------------
} /* End of function R_ETHER_Read_ZC2_BufRelease() */
```

1.5 Schedule for Fixing the Problem

This problem will be fixed in a later version. (Scheduled to be released in February 2020.)

2. When Calling the “R_ETHER_Write_ZC2_SetBuf” Function or “R_ETHER_Write” Function

2.1 Applicable Products

These functions are provided in the r_ether_rx_v*.**.zip (*.** is a revision number) file included in the products listed in (1) and (2) below.

(1) RX family Ethernet module using Firmware Integration Technology (Ethernet FIT module)

The applicable revision numbers and document numbers are as follows.

Table 2.1 Ethernet FIT module applicable products

Ethernet FIT module revision number	Document number
Rev.1.00	R01AN2009EJ0100
Rev.1.01	R01AN2009EJ0101
Rev.1.02	R01AN2009EJ0102
Rev.1.10	R01AN2009EJ0110
Rev.1.11	R01AN2009EJ0111
Rev.1.12	R01AN2009EJ0112
Rev.1.13	R01AN2009EJ0113
Rev.1.14	R01AN2009EJ0114
Rev.1.15	R01AN2009EJ0115
Rev.1.16	R01AN2009EJ0116
Rev.1.17	R01AN2009EJ0117

(2) RX Driver Package

The Ethernet FIT module in (1) is also included in the RX Driver Package listed below.
The product names and revision numbers of the applicable RX Driver Package and the revision numbers of the Ethernet FIT module are as follows.

Table 2.1 Products that includes the Ethernet FIT module

RX Driver Package product name	RX Driver Package Revision number	Document number	Revision number of the included Ethernet FIT module
RX64M Group RX Driver Package User's Manual	Rev.1.01	R01AN2460EJ0101	Rev.1.00
RX64M, RX71M Group RX Driver Package Ver.1.02	Rev.1.04	R01AN2606EJ0104	Rev.1.02
RX Family Driver Package Ver.1.10	Rev.1.10	R01AN3345EJ0100	Rev.1.10
RX Family Driver Package Ver.1.11	Rev.1.11	R01AN3467EJ0111	Rev.1.11
RX Family Driver Package Ver.1.12	Rev.1.12	R01AN3651EJ0112	Rev.1.12
RX Family Driver Package Ver.1.13	Rev.1.13	R01AN3859EJ0113	Rev.1.13
RX Family Driver Package Ver.1.14	Rev.1.14	R01AN4191EJ0114	Rev.1.14
RX Family Driver Package Ver.1.15	Rev.1.15	R01AN4372EJ0115	Rev.1.15
RX Family Driver Package Ver.1.16	Rev.1.16	R01AN4471EJ0116	Rev.1.15
RX Family Driver Package Ver.1.18	Rev.1.18	R01AN4659EJ0118	Rev.1.15
RX Family Driver Package Ver.1.19	Rev.1.19	R01AN4677EJ0119	Rev.1.15
RX Family Driver Package Ver.1.20	Rev.1.20	R01AN4794EJ0120	Rev.1.16
RX Family Driver Package Ver.1.22	Rev.1.22	R01AN4873EJ0122	Rev.1.17

2.2 Applicable Devices

RX63N, RX65N, RX64M, RX71M, and RX72M groups

2.3 Details and Conditions

If the number of transmit descriptors is set to 1, Ethernet frames may not be transmitted after the "R_ETHER_Write_ZC2_SetBuf" function(Note) or "R_ETHER_Write" function starts transmission.

* The "R_ETHER_Write_ZC2_SetBuf" function is a lower function of the "R_ETHER_Write" function.

- Examples

If the EDMAC sets the TR bit in the EDTRR register to 0 to stop the transmit function in coding at (i) and later in the "R_ETHER_Write_ZC2_SetBuf" function, Ethernet frames cannot be transmitted because the Ethernet FIT module cannot resume transmission.

```
 ether_return_t R_ETHER_Write_ZC2_SetBuf (uint32_t channel, const
uint32_t len)
 {
--------------------------- (omitted) ---------------------------
    /* When the Link up processing is completed */
    else
    {
        /* The data of the buffer is made active.  */
        papp_tx_desc[channel]->bufsize = len;
        papp_tx_desc[channel]->status &= (~(TFP1 | TFP0));
        papp_tx_desc[channel]->status |= ((TFP1 | TFP0) | TACT);
        papp_tx_desc[channel] = papp_tx_desc[channel]->next;

        pether_ch = g_eth_control_ch[channel].pether_control;
        phy_access = g_eth_control_ch[channel].phy_access;
        pedmac_adr = pether_ch[phy_access].pedmac;

        if (0x00000000L == pedmac_adr->EDTRR.LONG)    (i)
        {
            /* Restart if stopped */
            pedmac_adr->EDTRR.LONG = 0x00000001L;
        }

        ret = ETHER_SUCCESS;
--------------------------- (omitted) ---------------------------
 } /* End of function R_ETHER_Write_ZC2_SetBuf() */
```

2.4 Workaround

Add the processing in red in the “R_ETHER_Write_ZC2_GetBuf” function.

Before modification:

```
  ether_return_t R_ETHER_Write_ZC2_GetBuf (uint32_t channel, void
**pbuf, uint16_t *pbuf_size)
  {
    ether_return_t ret;
--------------------------- (omitted) ---------------------------
     /* When the Link up processing is completed */
     else
     {
         /* All transmit buffers are full */
         if (TACT == (papp_tx_desc[channel]->status & TACT))
         {
             ret = ETHER_ERR_TACT;
         }
         else
         {
             /* Give application another buffer to work with */
             (*pbuf) = papp_tx_desc[channel]->buf_p;
             (*pbuf_size) = ETHER_CFG_BUFSIZE;
             ret = ETHER_SUCCESS;
         }
     }
--------------------------- (omitted) ---------------------------
} /* End of function R_ETHER_Write_ZC2_GetBuf() */
```

After modification:

```
  ether_return_t R_ETHER_Write_ZC2_GetBuf (uint32_t channel, void
**pbuf, uint16_t *pbuf_size)
  {
     ether_return_t ret;
  #if (1 == ETHER_CFG_EMAC_TX_DESCRIPTORS)
     volatile struct st_edmac R_BSP_EVENACCESS_SFR * pedmac_adr;
     const ether_control_t * pether_ch;
     uint32_t phy_access;
  #endif
--------------------------- (omitted) ---------------------------
     /* When the Link up processing is completed */
     else
     {
  #if (1 == ETHER_CFG_EMAC_TX_DESCRIPTORS)
        pether_ch = g_eth_control_ch[channel].pether_control;
        phy_access = g_eth_control_ch[channel].phy_access;
        pedmac_adr = pether_ch[phy_access].pedmac;
  #endif

        /* All transmit buffers are full */
  #if (1 == ETHER_CFG_EMAC_TX_DESCRIPTORS)
        if (0x00000000L != pedmac_adr->EDTRR.LONG)
  #else
        if (TACT == (papp_tx_desc[channel]->status & TACT))
  #endif
        {
           ret = ETHER_ERR_TACT;
        }
        else
        {
           /* Give application another buffer to work with */
           (*pbuf) = papp_tx_desc[channel]->buf_p;
           (*pbuf_size) = ETHER_CFG_BUFSIZE;
           ret = ETHER_SUCCESS;
        }
     }
--------------------------- (omitted) ---------------------------
} /* End of function R_ETHER_Write_ZC2_GetBuf() */
```

2.5 Schedule for Fixing the Problem

This problem will be fixed in a later version. (Scheduled to be released in February 2020.)

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Oct.01.19	-	First edition issued

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu, Koto-ku, Tokyo 135-0061 Japan

www.renesas.com

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:

www.renesas.com/contact/

Trademarks

TS Colophon 4.0