[Notes]

RX Family TCP/IP for Embedded system M3S-T4-Tiny

R20TS0227EJ0100
Rev.1.00
Nov. 16, 2017

Outline

When using the RX family TCP/IP for embedded system M3S-T4-Tiny(Note), note the following points.

1. Ping Reply Packets
2. LAN Network Environment
3. Timeout Settings of the select() Function
4. Destination IP Address of UDP Packets Sent by Using the sendto() Function

Note: Refer to “Applicable MCUs” in each section for details.

1. Ping Reply Packets

1.1 Applicable Products

- RX Family Sample code that uses TCP/IP for Embedded system M3S-T4-Tiny Firmware Integration Technology V.1.00 to V.1.06
- RX Family TCP/IP for Embedded system M3S-T4-Tiny Firmware Integration Technology V.1.03 to V.2.06

1.2 Applicable MCUs

RX family

1.3 Details

M3S-T4-Tiny has an error in the Ping Reply packet send processing and sets an incorrect value in the IP Total length field of the IP header. Therefore, the terminal that sent the Ping Request might identify the Ping Reply as an invalid packet rather than a correct reply.

1.4 Conditions

This problem occurs if the value of the IP Total length of the IP header contained in the received Ping Request is 45 or smaller.

1.5 Workaround

There is currently no workaround for this problem.

1.6 Schedule for Fixing the Problem

This problem will be fixed in a later version.

2. LAN Network Environment

2.1 Applicable Products

- RX Family Sample code that uses TCP/IP for Embedded system M3S-T4-Tiny Firmware Integration Technology V.1.00 to V.1.06
- RX Family Interface conversion module for Ethernet Driver and Embedded system M3S-T4-Tiny Firmware Integration Technology V.1.00 to V.1.06

2.2 Applicable MCUs

RX family

2.3 Details

When you run a sample program in a LAN network environment where 10BASE half-duplex mode is used for communication, the sent packet returns to the RX MCU depending on the board. Therefore, the DHCP function does not operate correctly.

2.4 Conditions

This problem occurs when a connection is established in 10BASE half-duplex mode.

2.5 Workaround

If the send MAC address of the received packet is the same as the MAC address of the RX MCU, discard the received packet.

Add the text in red to the following functions.

■ r_t4_driver_rx¥src¥t4_driver.c: lan_read()

```
H lan_read(UB lan_port_no, B **buf)
{
    int32_t driver_ret;
    H return_code;
    UB *data;

    driver_ret = R_ETHER_Read_ZC2(lan_port_no, (void **)buf);
    if (driver_ret > 0)
    {
        data = (B *)*buf;
        if(0 == memcmp(&data[6],&_myethaddr[lan_port_no],6))
        {
            rcv_buff_release(lan_port_no);
            return_code = -1;
            return return_code;
        }
        t4_stat[lan_port_no].t4_rec_cnt++;
        t4_stat[lan_port_no].t4_rec_byte += (UW)driver_ret;
        return_code = (H)driver_ret;
    }
    else if (driver_ret == 0)
```

2.6 Schedule for Fixing the Problem

This problem will be fixed in a later version.

3. Timeout Settings of the select() Function

3.1 Applicable Products

- RX Family Sample code that uses TCP/IP for Embedded system M3S-T4-Tiny Firmware Integration Technology V.1.05
- RX Family Embedded TCP/IP M3S-T4-Tiny Socket API Module Firmware Integration Technology V1.31

3.2 Applicable MCUs

RX family

3.3 Details

Processing of the timeout time specified in the fifth argument of the select() function has an error. Note the following.

(1) A timeout occurs in a shorter time than the specified time.

(2) A maximum of 10 milliseconds is required to terminate the select() function.

3.4 Conditions

(1) For section 3.3 (1)

The problem always occurs.

The following is an example of the problem.

```
struct timeval select_timer;
/* 1.5 second */
select_timer.tv_sec = 1;
select_timer.tv_usec = 500000;
nready = select(1, NULL, NULL, NULL, &select_timer);
```

In the above, the value set for the tv_usec member is ignored, causing the operation to be the same as when 0 is set. As a result, a timeout takes less time than the specified time.

(2) For section 3.3 (2)

The problem occurs if both the tv_sec and tv_usec members are set to 0.

3.5 Workaround

Add the text in red to the following function.

■ select() function of the source code "r_socket.c" of the Socket API module

```
  uint32_t polling = 0;

  (Omitted)
  if ( timeout == NULL )
  {
  }
  else if ((timeout->tv_usec >= 1000000) || (timeout->tv_usec < 0) ||
(timeout->tv_sec < 0))
  {
     timeout = NULL;
  }
  else if ((timeout->tv_sec == 0) && (timeout->tv_usec == 0))
  {
     polling = 1;
  }
  else
  {
  //       timeout->tv_usec /= 10000;
  }
  timer1 = tcpudp_get_time();

  (Omitted)

        if (tot_count > 0)
        {
           break;
        }
        if(polling == 1)
        {
           break;
        }
        if (timeout != NULL)
        {
```

3.6 Schedule for Fixing the Problem

This problem will be fixed in a later version.

4. Destination IP Address of UDP Packets Sent by Using the sendto() Function

4.1 Applicable Products

- RX Family Sample code that uses TCP/IP for Embedded system M3S-T4-Tiny Firmware Integration Technology V.1.05
- RX Family Embedded TCP/IP M3S-T4-Tiny Socket API Module Firmware Integration Technology V1.31

4.2 Applicable MCUs

RX family

4.3 Details

Processing of the destination IP address specified in the sendto() function that sends UDP packets has an error. Note the following.

(1) UDP packets might be sent to a wrong IP address.

4.4 Conditions

This problem occurs if UDP packets are received from another IP address at the same time when the socket executes the sendto() function.

4.5 Workaround

Separate IP address information for transmission control from IP address information for reception control.

Modify the text in red in the following.

(1) r_socket_rx_if.h

Before modification

```
typedef struct _tag_BSDSocket
{
    BSD_STATE   state;      /* BSD socket states */
    uint32_t    T4status;    /* T4 current status */
    int         event;      /* T4 event occuring in callback */
    ID          socket_type; /* SOCK_STREAM or SOCK_DGRAM */
    ID          backlog;    /* No Used */
    T_IPV4EP    dstaddr;    /* Partners Address */
    TMO         tmout;      /* Time out */
    (The rest is omitted)
} BSDSocket;
```

After modification

```
typedef struct _tag_BSDSocket
{
    BSD_STATE   state;      /* BSD socket states */
    uint32_t    T4status;    /* T4 current status */
    int         event;      /* T4 event occuring in callback */
    ID          socket_type; /* SOCK_STREAM or SOCK_DGRAM */
    ID          backlog;    /* No Used */
    T_IPV4EP    dstaddr;    /* Partners Address */
    T_IPV4EP    udpsndaddr; /* UDP Send Address */
    TMO         tmout;      /* Time out */
    (The rest is omitted)
} BSDSocket;
```

(2) r_socket_rx.c R_SOCKET_Open()

Before modification

```
        sockets[i].dstaddr.ipaddr = 0;
        sockets[i].dstaddr.portno = 0;
        sockets[i].tmout = 0;  /* default is TMO_POL */
        sockets[i].T4proc = 0;
        sockets[i].rcvLen = 0;
```

After modification

```
        sockets[i].dstaddr.ipaddr = 0;
        sockets[i].dstaddr.portno = 0;
        sockets[i].udpsndaddr.ipaddr = 0;
        sockets[i].udpsndaddr.portno = 0;
        sockets[i].tmout = 0;  /* default is TMO_POL */
        sockets[i].T4proc = 0;
        sockets[i].rcvLen = 0;
```

(3) r_socket_rx.c　connect()

Before modification

```
    else
    {
        /* UDP: remote port is used as a filter only.  no need to
call connect() */
        /* TODO any check for addr range, port range? */
        sockets[sock].dstaddr.portno = remote_port;
        sockets[sock].dstaddr.ipaddr = remote_ip;
        sockets[sock].state = BSD_CONNECTED;
    }
    return E_OK;
```

After modification

```
    else
    {
        /* UDP: remote port is used as a filter only.  no need to
call connect() */
        /* TODO any check for addr range, port range? */
        sockets[sock].dstaddr.portno = remote_port;
        sockets[sock].dstaddr.ipaddr = remote_ip;
        sockets[sock].udpsndaddr.portno = remote_port;
        sockets[sock].udpsndaddr.ipaddr = remote_ip;
        sockets[sock].state = BSD_CONNECTED;
    }
    return E_OK;
```

(4) r_socket_rx.c sendto()

Before modification

```
        if ( sockets[sock].state < BSD_CONNECTED )
        {
            sockets[sock].dstaddr.portno = addr->sin_port;
            sockets[sock].dstaddr.ipaddr = addr-
>sin_addr.S_un.S_addr;
        }

        memcpy( sockets[sock].snd_buf, buffer, length );
        sockets[sock].sndLen = 0;
        sockets[sock].sndSz = length;
        sockets[sock].pending_sndlen = length;
        sockets[sock].T4proc &= ~(T4_PROC_SND_END);
        sockets[sock].T4proc |= (T4_PROC_SND_START);
        ercd = udp_snd_dat( cepid, (T_IPV4EP*) &
sockets[sock].dstaddr, (VP)sockets[sock].snd_buf, length,
TMO_NBLK );
        if ( ercd == E_WBLK )
        {
            ercd = length;  /* Return length of data even though it
may not been sent*/

        }
```

After modification

```
        if ( sockets[sock].state < BSD_CONNECTED )
        {
            sockets[sock].udpsndaddr.portno = addr->sin_port;
            sockets[sock].udpsndaddr.ipaddr = addr-
>sin_addr.S_un.S_addr;
        }

        memcpy( sockets[sock].snd_buf, buffer, length );
        sockets[sock].sndLen = 0;
        sockets[sock].sndSz = length;
        sockets[sock].pending_sndlen = length;
        sockets[sock].T4proc &= ~(T4_PROC_SND_END);
        sockets[sock].T4proc |= (T4_PROC_SND_START);
        ercd = udp_snd_dat( cepid, (T_IPV4EP*) &
sockets[sock].udpsndaddr, (VP)sockets[sock].snd_buf, length,
TMO_NBLK );
        if ( ercd == E_WBLK )
        {
            ercd = length;  /* Return length of data even though it
may not been sent*/

        }
```

(5) r_socket_rx.c t4_udp_generic_callback()

Before modification

```
            case TFN_UDP_SND_DAT:
                sockets[i].T4proc &= ~((uint32_t)T4_PROC_SND_START);
                sockets[i].event = SOCKET_SND_DAT;
                sockets[i].sndLen += ercd;
                remain_size = sockets[i].sndSz - sockets[i].sndLen;
                if ( remain_size > 0 )
                {
                    udp_snd_dat(cepid,  &sockets[i].dstaddr,
(VP)(&sockets[i].snd_buf[sockets[i].sndLen]),
                                remain_size, TMO_NBLK);
                    sockets[i].T4proc |=
((uint32_t)T4_PROC_SND_START);
                }
```

After modification

```
            case TFN_UDP_SND_DAT:
                sockets[i].T4proc &= ~((uint32_t)T4_PROC_SND_START);
                sockets[i].event = SOCKET_SND_DAT;
                sockets[i].sndLen += ercd;
                remain_size = sockets[i].sndSz - sockets[i].sndLen;
                if ( remain_size > 0 )
                {
                    udp_snd_dat(cepid,  &sockets[i].udpsndaddr,
(VP)(&sockets[i].snd_buf[sockets[i].sndLen]),
                                remain_size, TMO_NBLK);
                    sockets[i].T4proc |=
((uint32_t)T4_PROC_SND_START);
                }
```

(6) r_socket_rx.c reset_socket ()

Before modification

```
    if ( sock != -1 )
    {
        sockets[sock].state = BSD_CLOSED;
        sockets[sock].T4status = T4_CLOSED;
        sockets[sock].event = -1;
        sockets[sock].socket_type = 0;
        sockets[sock].backlog = 0;
        sockets[sock].dstaddr.ipaddr = 0;
        sockets[sock].dstaddr.portno = 0;
        sockets[sock].tmout = 0;  //default is TMO_POL
```

After modification

```
    if ( sock != -1 )
    {
        sockets[sock].state = BSD_CLOSED;
        sockets[sock].T4status = T4_CLOSED;
        sockets[sock].event = -1;
        sockets[sock].socket_type = 0;
        sockets[sock].backlog = 0;
        sockets[sock].dstaddr.ipaddr = 0;
        sockets[sock].dstaddr.portno = 0;
        sockets[sock].udpsndaddr.ipaddr = 0;
        sockets[sock].udpsndaddr.portno = 0;
        sockets[sock].tmout = 0;  //default is TMO_POL
```

4.6 Schedule for Fixing the Problem

This problem will be fixed in a later version.

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Nov. 16, 2017	-	First edition issued

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

■Inquiry
https://www.renesas.com/contact/