[Notes]

RX Family

RSPI Module Using Firmware Integration Technology,

RX Driver Package

R20TS0667EJ0100
Rev.1.00
May. 16, 2021

Overview

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

1. Notes on Using High-Speed Mode

1. 1. Notes on Using High-Speed Mode

1.1 Applicable Product

(1) RSPI module using Firmware Integration Technology (RSPI FIT module)

The applicable revision numbers and document numbers are as follows.

Table 1.1 RSPI FIT Module Applicable Products

Revision of the RSPI FIT module	Document number
Rev.3.00	R01AN1827EJ0300
Rev.2.05	R01AN1827EJ0205
Rev.2.04	R01AN1827EJ0204
Rev.2.03	R01AN1827EJ0203
Rev.2.02	R01AN1827EJ0202
Rev.2.01	R01AN1827EJ0201
Rev.2.00	R01AN1827EJ0200

(2) RX Driver Package

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

Table 1.2 Products Which Include the RSPI FIT Module

Product name of the RX Driver Package	Revision of the RX Driver Package	Document number	Revision of the included RSPI FIT module
RX Family RX Driver Package Ver.1.29	Rev.1.29	R01AN5826EJ0129	Rev.3.00
RX Family RX Driver Package Ver.1.28	Rev.1.28	R01AN5761EJ0128	Rev.3.00
RX Family RX Driver Package Ver.1.27	Rev.1.27	R01AN5600EJ0127	Rev.3.00
RX Family RX Driver Package Ver.1.26	Rev.1.26	R01AN5401EJ0126	Rev.2.05
RX Family RX Driver Package Ver.1.25	Rev.1.25	R01AN371EJ0125	Rev.2.05
RX Family RX Driver Package Ver.1.24	Rev.1.24	R01AN5267EJ0124	Rev.2.04
RX Family RX Driver Package Ver.1.23	Rev.1.23	R01AN4976EJ0123	Rev.2.03

RX Family RX Driver Package Ver.1.22	Rev.1.22	R01AN4873EJ0122	Rev.2.03
RX Family RX Driver Package Ver.1.20	Rev.1.20	R01AN4794EJ0120	Rev.2.01
RX Family RX Driver Package Ver.1.19	Rev.1.19	R01AN4677EJ0119	Rev.2.00

1.2 Applicable MCUs

RX110, RX111, RX113, and RX130 groups

RX230, RX231, RX23E-A, RX23W, RX23T, RX24T, and RX24U groups

RX64M, RX651, RX65N, RX66T, and RX66N groups

RX71M, RX72T, RX72M, and RX72N groups

1.3 Description and condition

(1) Buffer overflow occurs

Buffer overflow occurs when the MCU is set to high-speed mode and the timing of reading data from a data register is delayed due to, for example, an interrupt from another peripheral.

(2) Incoming data might not be received correctly.

Some of the incoming data might be lost when the MCU is set to high-speed mode and the timing of a receive buffer full interrupt is delayed due to, for example, an interrupt from another peripheral because a transmit empty interrupt performs a dummy read on the SPDR register.

1.4 Workaround

If you use high-speed mode, add each of the following processing.

(1) Workaround for preventing buffer overflow

This workaround is not required for the RX110, RX111 and RX113 groups of MCUs because they do not have an RSPCK auto-stop function enable bit.

File: ¥src¥smc_gen¥r_rspi_rx¥src¥r_rspi_defaults.h

Before modification

```
  #define RSPI_SPCR2_DEF       (RSPI_SPCR2_SPPE_DEF | RSPI_SPCR2_SPOE_DEF |
RSPI_SPCR2_SPIIE_DEF | RSPI_SPCR2_PTE_DEF | ¥
                        RSPI_SPCR2_SCKASE_DEF)
```

After modification (Add the text in red)

```
#if (!defined(BSP_MCU_RX110) && !defined(BSP_MCU_RX111)
&& !defined(BSP_MCU_RX113))
  #define RSPI_SPCR2_SCKASE_ENABLE (0x10) /* 1: Enables the RSPCK auto-stop function
*/
#else
#define RSPI_SPCR2_SCKASE_ENABLE (0x00)
#endif


  #define RSPI_SPCR2_DEF       (RSPI_SPCR2_SPPE_DEF | RSPI_SPCR2_SPOE_DEF |
RSPI_SPCR2_SPIIE_DEF | RSPI_SPCR2_PTE_DEF | ¥
                        RSPI_SPCR2_SCKASE_DEF)
  #define RSPI_SPCR2_HIGH_SPEED   (RSPI_SPCR2_SPPE_DEF |
RSPI_SPCR2_SPOE_DEF | RSPI_SPCR2_SPIIE_DEF | RSPI_SPCR2_PTE_DEF | ¥
                        RSPI_SPCR2_SCKASE_ENABLE)
```

File: ¥src¥smc_gen¥r_rspi_rx¥src¥r_rspi_defaults.h

Before modification

```
static rspi_ctrl_reg_values_t g_ctrl_reg_values[] =
{
    RSPI_SPCR_DEF,          // Control Register (SPCR)
    RSPI_SSLP_DEF,          // Slave Select Polarity Register (SSLP)
    RSPI_SPPCR_DEF,         // Pin Control Register (SPPCR)
    RSPI_SPSCR_DEF,         // Sequence Control Register (SPSCR)
    RSPI_SPBR_DEF,          // Bit Rate Register (SPBR)
    RSPI_SPDCR_DEF,         // Data Control Register (SPDCR)
    RSPI_SPCKD_DEF,         // Clock Delay Register (SPCKD)
    RSPI_SSLND_DEF,         // Slave Select Negation Delay Register (SSLND)
    RSPI_SPND_DEF,          // Next-Access Delay Register (SPND)
    RSPI_SPCR2_DEF,         // Control Register 2 (SPCR2)
    RSPI_SPDCR2_DEF,        // Data Control Register 2 (SPDCR2)
```

After modification (Add the text in red)

```
static rspi_ctrl_reg_values_t g_ctrl_reg_values[] =
{
    RSPI_SPCR_DEF,          // Control Register (SPCR)
    RSPI_SSLP_DEF,          // Slave Select Polarity Register (SSLP)
    RSPI_SPPCR_DEF,         // Pin Control Register (SPPCR)
    RSPI_SPSCR_DEF,         // Sequence Control Register (SPSCR)
    RSPI_SPBR_DEF,          // Bit Rate Register (SPBR)
    RSPI_SPDCR_DEF,         // Data Control Register (SPDCR)
    RSPI_SPCKD_DEF,         // Clock Delay Register (SPCKD)
    RSPI_SSLND_DEF,         // Slave Select Negation Delay Register (SSLND)
    RSPI_SPND_DEF,          // Next-Access Delay Register (SPND)
#if RSPI_CFG_HIGH_SPEED_READ == 1
        RSPI_SPCR2_HIGH_SPEED, // Control Register 2 (SPCR2) High-speed mode
#else
        RSPI_SPCR2_DEF,     // Control Register 2 (SPCR2) Default-speed mode
#endif
    RSPI_SPDCR2_DEF,        // Data Control Register 2 (SPDCR2)
```

Before modification

```
#if    RSPI_NUM_CHANNELS > 1
      RSPI_SPCR_DEF, // Control Register (SPCR)
      RSPI_SSLP_DEF,          // Slave Select Polarity Register (SSLP)
      RSPI_SPPCR_DEF,         // Pin Control Register (SPPCR)
      RSPI_SPSCR_DEF,         // Sequence Control Register (SPSCR)
      RSPI_SPBR_DEF,          // Bit Rate Register (SPBR)
      RSPI_SPDCR_DEF,         // Data Control Register (SPDCR)
      RSPI_SPCKD_DEF,         // Clock Delay Register (SPCKD)
      RSPI_SSLND_DEF,         // Slave Select Negation Delay Register (SSLND)
      RSPI_SPND_DEF,          // Next-Access Delay Register (SPND)
      RSPI_SPCR2_DEF,         // Control Register 2 (SPCR2)
      RSPI_SPDCR2_DEF,        // Data Control Register 2 (SPDCR2)
#endif
```

After modification (Add the text in red)

```
#if    RSPI_NUM_CHANNELS > 1
      RSPI_SPCR_DEF, // Control Register (SPCR)
      RSPI_SSLP_DEF,          // Slave Select Polarity Register (SSLP)
      RSPI_SPPCR_DEF,         // Pin Control Register (SPPCR)
      RSPI_SPSCR_DEF,         // Sequence Control Register (SPSCR)
      RSPI_SPBR_DEF,          // Bit Rate Register (SPBR)
      RSPI_SPDCR_DEF,         // Data Control Register (SPDCR)
      RSPI_SPCKD_DEF,         // Clock Delay Register (SPCKD)
      RSPI_SSLND_DEF,         // Slave Select Negation Delay Register (SSLND)
      RSPI_SPND_DEF,          // Next-Access Delay Register (SPND)
#if RSPI_CFG_HIGH_SPEED_READ == 1
          RSPI_SPCR2_HIGH_SPEED, // Control Register 2 (SPCR2) High-speed mode
#else
          RSPI_SPCR2_DEF,     // Control Register 2 (SPCR2) Default-speed mode
#endif
      RSPI_SPDCR2_DEF,        // Data Control Register 2 (SPDCR2)
#endif
```

Before modification

```
#if    RSPI_NUM_CHANNELS >2
      RSPI_SPCR_DEF, // Control Register (SPCR)
      RSPI_SSLP_DEF,          // Slave Select Polarity Register (SSLP)
      RSPI_SPPCR_DEF,         // Pin Control Register (SPPCR)
      RSPI_SPSCR_DEF,         // Sequence Control Register (SPSCR)
      RSPI_SPBR_DEF,          // Bit Rate Register (SPBR)
      RSPI_SPDCR_DEF,         // Data Control Register (SPDCR)
      RSPI_SPCKD_DEF,         // Clock Delay Register (SPCKD)
      RSPI_SSLND_DEF,         // Slave Select Negation Delay Register (SSLND)
      RSPI_SPND_DEF,          // Next-Access Delay Register (SPND)
      RSPI_SPCR2_DEF,         // Control Register 2 (SPCR2)
      RSPI_SPDCR2_DEF,        // Data Control Register 2 (SPDCR2)
#endif
```

After modification (Add the text in red)

```
#if    RSPI_NUM_CHANNELS >2
      RSPI_SPCR_DEF, // Control Register (SPCR)
      RSPI_SSLP_DEF,          // Slave Select Polarity Register (SSLP)
      RSPI_SPPCR_DEF,         // Pin Control Register (SPPCR)
      RSPI_SPSCR_DEF,         // Sequence Control Register (SPSCR)
      RSPI_SPBR_DEF,          // Bit Rate Register (SPBR)
      RSPI_SPDCR_DEF,         // Data Control Register (SPDCR)
      RSPI_SPCKD_DEF,         // Clock Delay Register (SPCKD)
      RSPI_SSLND_DEF,         // Slave Select Negation Delay Register (SSLND)
      RSPI_SPND_DEF,          // Next-Access Delay Register (SPND)
#if RSPI_CFG_HIGH_SPEED_READ == 1
            RSPI_SPCR2_HIGH_SPEED, // Control Register 2 (SPCR2) High-speed mode
#else
            RSPI_SPCR2_DEF,     // Control Register 2 (SPCR2) Default-speed mode
#endif
      RSPI_SPDCR2_DEF,        // Data Control Register 2 (SPDCR2)
#endif
```

(2) Workaround for receiving incoming data correctly:

File: ¥src¥smc_gen¥r_rspi_rx¥src¥r_rspi_defaults.h

Before modification

```
R_BSP_ATTRIB_STATIC_INTERRUPT void rspi_spti0_isr(void)
{
    if (RSPI_TRANS_MODE_SW == g_rspi_tcb[0].data_tran_mode)
    {
        g_rxdata[0] = RSPI0.SPDR.LONG; // Read rx-data register into temp buffer.

            /* If master mode then disable further spti interrupts on first transmit.
            If slave mode then we do two transmits to fill the double buffer,
            then disable spti interrupts.
            The receive interrupt will handle any remaining data. */
#if RSPI_CFG_HIGH_SPEED_READ == 0
            if ((RSPI0.SPCR.BIT.MSTR) || (g_rspi_tcb[0].tx_count > 0))
            {
```

After modification (Add the text in red)

```
R_BSP_ATTRIB_STATIC_INTERRUPT void rspi_spti0_isr(void)
{
    if (RSPI_TRANS_MODE_SW == g_rspi_tcb[0].data_tran_mode)
    {
        if (0 == g_rspi_tcb[0].tx_count)
        {
            g_rxdata[0] = RSPI0.SPDR.LONG; // Read rx-data register into temp buffer.
        }
            /* If master mode then disable further spti interrupts on first transmit.
            If slave mode then we do two transmits to fill the double buffer,
            then disable spti interrupts.
            The receive interrupt will handle any remaining data. */
#if RSPI_CFG_HIGH_SPEED_READ == 0
            if ((RSPI0.SPCR.BIT.MSTR) || (g_rspi_tcb[0].tx_count > 0))
            {
```

Before modification

```
R_BSP_ATTRIB_STATIC_INTERRUPT void rspi_spti1_isr(void)
{
    if (RSPI_TRANS_MODE_SW == g_rspi_tcb[1].data_tran_mode)
    {
        g_rxdata[1] = RSPI1.SPDR.LONG; // Read rx-data register into temp buffer.

            /* If master mode then disable further spti interrupts on first transmit.
            If slave mode then we do two transmits to fill the double buffer,
            then disable spti interrupts.
            The receive interrupt will handle any remaining data. */
#if RSPI_CFG_HIGH_SPEED_READ == 0
            if ((RSPI1.SPCR.BIT.MSTR) || (g_rspi_tcb[1].tx_count > 0))
            {
```

After modification (Add the text in red)

```
R_BSP_ATTRIB_STATIC_INTERRUPT void rspi_spti1_isr(void)
{
    if (RSPI_TRANS_MODE_SW == g_rspi_tcb[1].data_tran_mode)
    {
        if(0 == g_rspi_tcb[1].tx_count)
        {
            g_rxdata[1] = RSPI1.SPDR.LONG; // Read rx-data register into temp buffer.
        }
            /* If master mode then disable further spti interrupts on first transmit.
            If slave mode then we do two transmits to fill the double buffer,
            then disable spti interrupts.
            The receive interrupt will handle any remaining data. */
#if RSPI_CFG_HIGH_SPEED_READ == 0
            if ((RSPI1.SPCR.BIT.MSTR) || (g_rspi_tcb[1].tx_count > 0))
            {
```

Before modification

```
R_BSP_ATTRIB_STATIC_INTERRUPT void rspi_spti2_isr(void)
{
    if (RSPI_TRANS_MODE_SW == g_rspi_tcb[2].data_tran_mode)
    {
        g_rxdata[2] = RSPI2.SPDR.LONG; // Read rx-data register into temp buffer.
            /* If master mode then disable further spti interrupts on first transmit.
            If slave mode then we do two transmits to fill the double buffer,
            then disable spti interrupts.
            The receive interrupt will handle any remaining data. */
#if RSPI_CFG_HIGH_SPEED_READ == 0
            if ((RSPI2.SPCR.BIT.MSTR) || (g_rspi_tcb[2].tx_count > 0))
            {
```

After modification (Add the text in red)

```
R_BSP_ATTRIB_STATIC_INTERRUPT void rspi_spti2_isr(void)
{
    if (RSPI_TRANS_MODE_SW == g_rspi_tcb[2].data_tran_mode)
    {
        g_rxdata[2] = RSPI2.SPDR.LONG; // Read rx-data register into temp buffer.
        if (0 == g_rspi_tcb[2].tx_count)
        {
            g_rxdata[2] = RSPI2.SPDR.LONG; // Read rx-data register into temp buffer.
        }
            /* If master mode then disable further spti interrupts on first transmit.
            If slave mode then we do two transmits to fill the double buffer,
            then disable spti interrupts.
            The receive interrupt will handle any remaining data. */
#if RSPI_CFG_HIGH_SPEED_READ == 0
            if ((RSPI2.SPCR.BIT.MSTR) || (g_rspi_tcb[2].tx_count > 0))
            {
```

1.5 Schedule for Fixing the Problem

This problem will be fixed in a later version.

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	May.16.21	-	First edition issued

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Trademarks

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/.

TS Colophon 4.2